

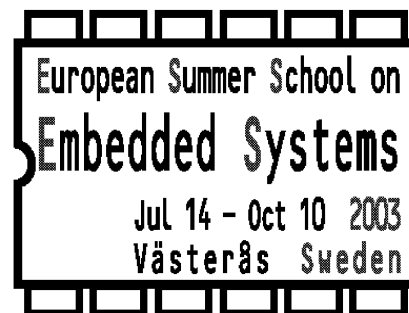
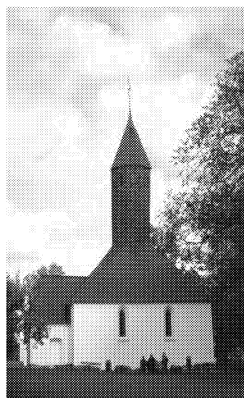
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes Part IV

Low Power Systems: Dynamic Voltage Scheduling



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, July 21-25, 2003

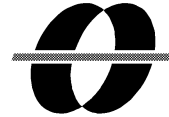
ISSN 1404-3041

ISRN MDH-MRTC-104/2003-1-SE

MRTC

MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se



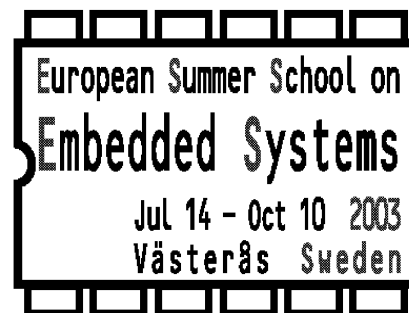
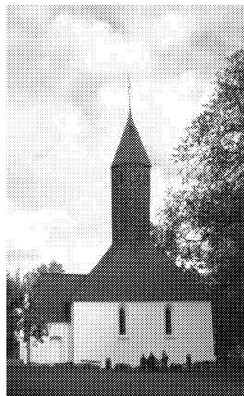
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes Part V

Low Power Systems: Dynamic Voltage Scheduling



Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, July 21-25, 2003

ISSN 1404-3041

ISRN MDH-MRTC-104/2003-1-SE

MRTC

MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se

Dynamic Voltage Scaling for Hard Real-Time Systems

Jihong Kim

School of Computer Science & Engineering
Seoul National University, Korea

Dynamic Voltage Scaling for Hard Real-Time Systems

Jihong Kim

**School of Computer Science & Engineering
Seoul National University**

ESSES 2003

Västerås, Sweden

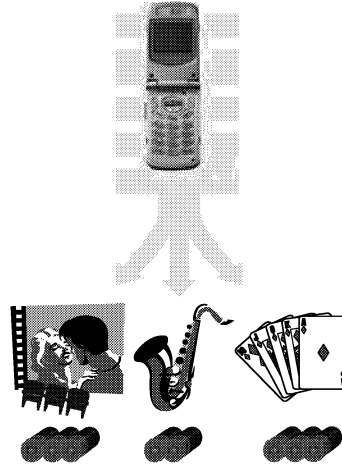
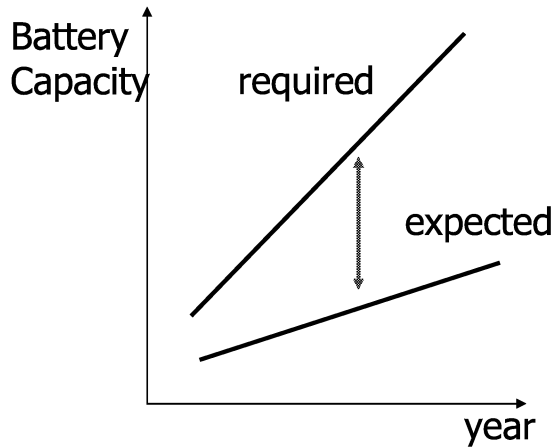
July 22, 2003

Course Organization

- **Low-power systems 101**
 - **Low-power binary encoding**
 - **Power-aware compiler techniques**
- **Dynamic voltage scaling techniques**
 - **OS-level DVS: Inter-Task DVS**
 - **Compiler-level DVS: Intra-Task DVS**
 - **Application-level DVS**
 - **Low-power convolution**

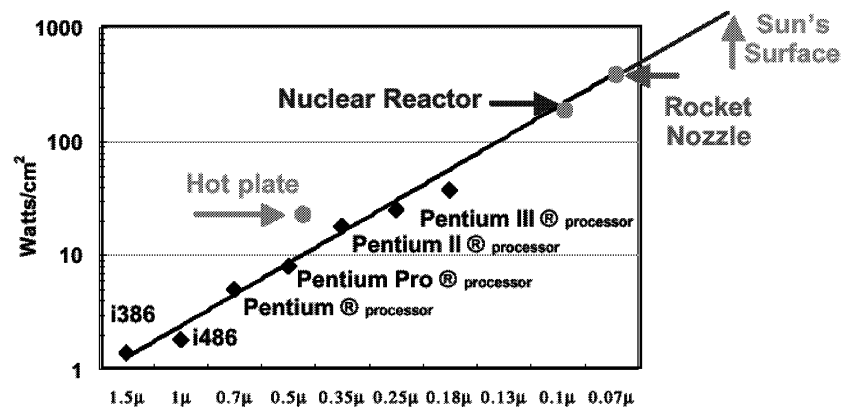
Why Low Power?

- Limited Battery Capacity



Why Low Power?

- Heat Dissipation

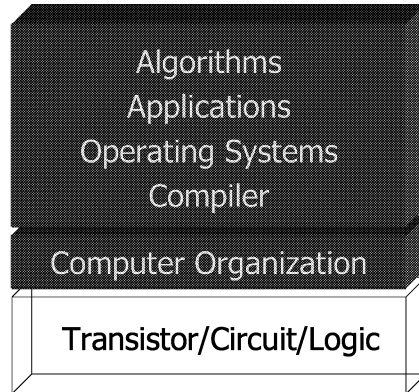


Power density getting worse

From F. Pollack

Low Power S/W Research

- Goal: ***Power-Aware Computing***



Why S/W Techniques for Low Power?

- **S/W techniques require no H/W modifications**
- **Many low-power H/W techniques require concurrent engineering between H/W and S/W.**
 - **Examples: DPM, DVS, ...**
 - **Efficiency of S/W Techniques are Critical for overall high energy efficiency**



Power Consumption in CMOS

- **Dynamic Power Consumption**
 - **Charging and discharging capacitors**
- **Short circuit currents**
 - **Short circuit path between supply rails during switching**
- **Leakage current**
 - **Leaking diodes and transistors**

Dynamic Power Consumption

$$P_{\text{dynamic}} = K \times C_{\text{out}} \times V_{\text{dd}}^2 \times f$$

K: activity factor

C_{out} : total chip capacitance

V_{dd}^2 : supply voltage

f: clock frequency

Reduce

- 1) switching activity
- 2) supply voltage

Roadmap

- ✓ Low-power systems 101
- Low-power binary encoding
- **Power-aware compiler techniques**
- **Dynamic voltage scaling techniques**
 - **intraDVS**
 - **interDVS**
 - **Low-power convolution**

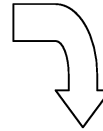
Low Power Binary Encoding

- **Switching activity reduction**
 - **Switching activity can account for over 90% of power dissipation of CMOS circuit.**
[Chandrakasan *et al*, '92]
- **Goal of Low power binary encoding**
 - **Modify the binary encoding/representation so that the switching activity is reduced.**
 - **Target Areas:**
 - **Op-code field**
 - **Register field**
 - **Bus**

Register Relabeling

- **Goal**
 - **Assign register numbers to minimize the switching activities in register field**

ADD	r1 (0001)	r2 (0010)	r3 (0011)
SUB	r14 (1110)	r13 (1101)	r12 (1100)
MUL	r3 (0011)	r12 (1100)	r3 (0011)



```

r1 -> r12
r2 -> r2
r3 -> r0
r12 -> r1
r13 -> r3
r14 -> r8
    
```

Switching Activity

Above : 20bit

Right : 6bit

ADD	r12 (1100)	r2 (0010)	r0 (0000)
SUB	r8 (1000)	r3 (0011)	r1 (0001)
MUL	r0 (0000)	r1 (0001)	r0 (0000)

ESSES 2003
2003/7/22 (Jihong Kim)

11

Register Relabeling

- **General Approach**
 - **Collect the trace of register field usage information**
 - **Construct the Register Field Transition Graph (RFTG)**
 - **Nodes: registers**
 - **Edges: transitions**
 - **Edge weights: relative frequency of corresponding edges**
 - **Find new register number assignment that minimize the total bit changes.**

ESSES 2003
2003/7/22 (Jihong Kim)

12

Problem Formulation

- A register field transition graph (RFTG)
 - $G = (V, E, w) : V = V_{reg} \cup V_{imm}$
- A relabeling function
 - find $f: V_{reg} \rightarrow V_{reg}$, to minimize the following cost metric

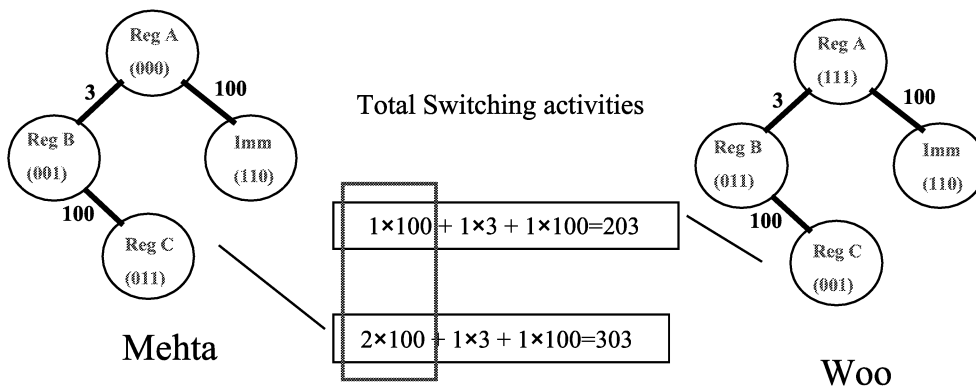
$$P(G, f) =$$

$$\sum_{\substack{e=(v_1, v_2) \\ v_1, v_2 \in V_{reg}}} w(e) \times h(f(v_1), f(v_2)) + \sum_{\substack{e=(v_1, v_2) \\ v_1 \in V_{reg} \\ v_2 \in V_{imm}}} w(e) \times h(f(v_1), v_2) + \sum_{\substack{e=(v_1, v_2) \\ v_1 \in V_{imm} \\ v_2 \in V_{reg}}} w(e) \times h(v_1, f(v_2))$$

where, $w(e)$ is the weight of edge e

Register Relabeling

- Alternatives:
 - Mehta's method: Immediate field not considered
 - Woo's method: Immediate field considered



Register Relabeling Heuristics

- Relabeling is a NP-hard problem
 - $\binom{N}{2}$ possible choices for each pair of exchanging candidates
- Slack-based heuristic [Woo, '01]
 - Define slack value for each node (encoding)

$$slack(v_i, v_j) = [h(v_i, v_j) - 1] \cdot w(e)$$
 - Exchange the encoding between most promising candidates until no more reduction is obtained from exchanging encoding
- Greedy method [Woo, '01]
 - Exchange randomly but undo the exchange if no gain is obtained from it

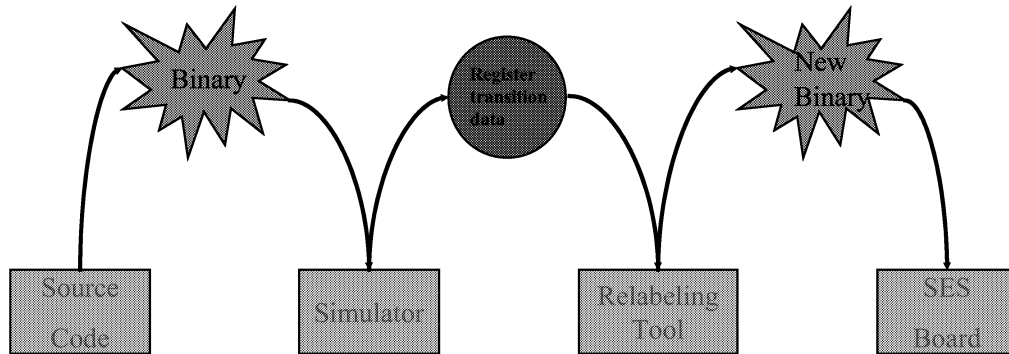
Experiment (Switching Activity)

- Simulation environment
 - SimpleScalar simulator is used
 - Benchmark
 - SPEC95 int and SPEC95 fp
 - UTDSP benchmark
 - MPEG2 decoder with video only streams
- Result
 - % of switching activity

Program	No relabeling	Mehta relabelling	New relabelling
SPEC95 geometric mean	1.0	0.96	0.91
applu	1.0	0.96	0.90
compress	1.0	0.96	0.89
gcc	1.0	0.98	0.93
UTDSP geometric mean	1.0	0.93	0.91
adpcm	1.0	0.93	0.92
histogram	1.0	0.91	0.87
turbo3d	1.0	0.96	0.93
MPEG2 decoder	1.0	0.91	0.86
Total average	1.0	0.94	0.89

Experiment (Energy Reduction)

- **Environment**
 - Target architecture : ARM7TDMI
 - Measurement Tool : SES (SNU Energy Scanner) board



ESSES 2003
2003/7/22 (Jihong Kim)

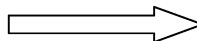
17

Experiment (Energy Reduction)

- **Effect of register relabeling at Instruction level**

D/I	Inst/Data	Energy
1	e0222098	1011.946950
1	e1d380f8	1064.029688
1	e1dc00f8	906.189522
1	e0222098	1077.069527
1	e1d300fa	1062.568274
1	e1dc80fa	944.328240
1	e0222098	1010.444222
1	e1d300fc	1065.396874
1	e1dc30fc	899.009519
1	e0222098	1083.144593
1	e1d300fe	1062.473957
1	e1dc30fe	950.174787
1	e0222098	1088.944918
1	e1d381f0	1052.254592
1	e1dc01f0	928.342904
1	e0222098	1075.622301
1	e2833012	1068.433202
1	e1520001	945.908777

Relabeling



D/I	Inst/Data	Energy
1	e0288092	1100.673882
1	e1d920f8	997.394383
1	e1dc00f8	941.629293
1	e0288092	1055.144933
1	e1d900fa	982.343107
1	e1dc20fa	932.772563
1	e0288092	1080.059544
1	e1d900fc	1010.521764
1	e1dc20fc	922.440499
1	e0288092	994.389910
1	e1d900fe	1002.936386
1	e1dc20fe	902.043706
1	e0288092	1016.250972
1	e1d921f0	1006.228806
1	e1dc01f0	903.564458
1	e0288092	992.725140
1	e2899012	977.829571
1	e1580004	1155.485459

ESSES 2003
2003/7/22 (Jihong Kim)

Total energy : 480.80mJ

Total energy : 456.53mJ

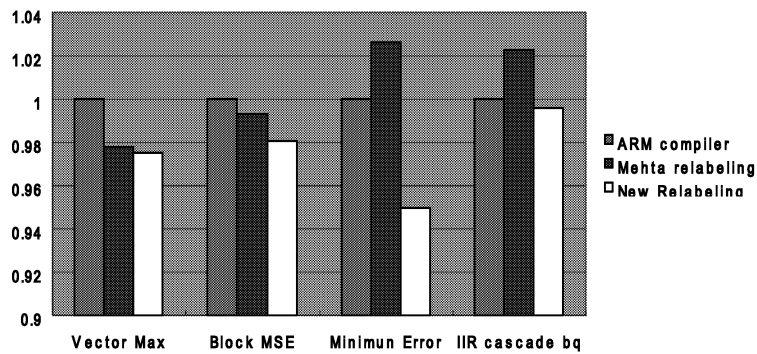
18

Experiment (Energy Reduction)

- **Result**

- **Benchmark : TI C6000 Benchmark**
 - Up to 5% energy reduction

Programs	ARM compiler	Mehta relabeling	New Relabeling
Vector Max	1	0.978	0.975
Block MSE	1	0.993	0.981
Minimun Error	1	1.026	0.950
IIR cascade bq	1	1.023	0.998



Related Work

- **Register Relabeling**
 - **Kandemir [Kandemir et al, '00]**
 - **Similar to Mehta**
 - **Give more time efficient heuristics**
- **Low power opcode encoding [Kim et al, '99]**

Conclusion

- **Register relabeling with immediate values.**
- **Energy reduction without H/W modification**
- **Energy reduction with simple modification of binary codes**
- **Energy reduction up to 5% in CPU**

References

- **Chandrakasan, T. Shyng, and R. W. Brodersen, "Low power CMOS Digital Design", IEEE Journal of Solid State Circuits, 1992**
- **S. Kim, and J. Kim, "Opcode Encoding for Low Power Instruction Fetch", IEE Electronic Letters, 1999**
- **H. Mehta, R. M. Owens, M. Irwin, R. Chen, and D. Ghosh, "Techniques for Low Energy Software", Proc of ISLPED, 1997**
- **Ching-Long Su, Chi-Ying Tsui, and Alvin M. Despain, "Low Power Architecture Design and Compilation Techniques for High-Performance Processors", Proc of COMPCON, 1994**
- **M. Kandemir, N. Vijaykrishnan, M. J. Irwin, W. Ye, and I. Demirkiran, "Register relabeling : A post-compilation technique for energy reduction", Proc of the Workshop on Compilers and Operating Systems for Low Power, 2000**
- **S. Woo, J. Yoon, and J. Kim, "Low-Power Instruction Encoding Techniques", Proc of SoC Design Conference, 2001**

Roadmap

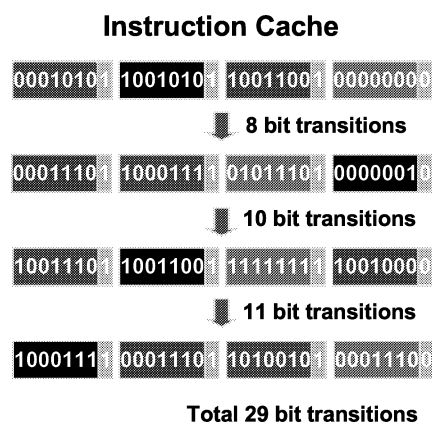
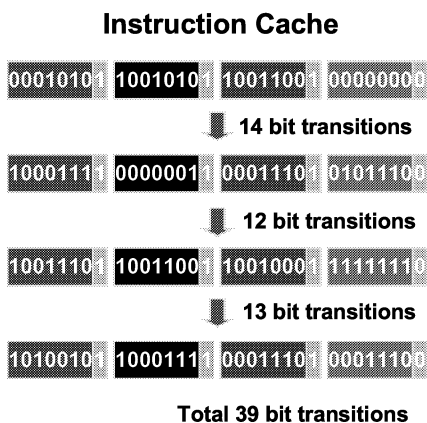
- ✓ Low-power systems 101
- ✓ Low-power binary encoding
- Power-aware compiler techniques
- **Dynamic voltage scaling techniques**
 - intraDVS
 - interDVS
 - **Low-power convolution**

Power-aware compiler techniques (for VLIW processors)

- **Many mobile devices are designed using VLIW processors for high performance, which usually consume more power than single-issue processors.**
- Operation rearrangement in VLIW instruction fetches
 - **A post-past optimization technique**
 - **Reduce switching activities by rearranging operations in each VLIW instruction.**
- **Battery-aware balanced modulo scheduling**
 - **Effective battery utilization depends on current fluctuation**
 - **Less fluctuation leads to longer battery lifetime**
 - **Reduce power fluctuation**

Operation Rearrangement in VLIW Instruction Fetches

Basic idea



(a) Before operation rearrangement

(b) After operation rearrangement

The total # of bit changes are reduced by 25%

VLIW instruction encoding: uncompressed

```

IADD /*IntU*/
|| FADD /*FpU*/
|| LOAD /*MemU*/
|| STORE /*MEMU*/

ISUB /*IntU*/
|| IMUL /*IntU*/

IADD /*IntU*/
|| BEG /*BrU*/
    
```

Alternative encoding

<i>IntU</i>	<i>IntU</i>	<i>FpU</i>	<i>FpU</i>	<i>MemU</i>	<i>MemU</i>	<i>CmpU</i>	<i>BrU</i>
-------------	-------------	------------	------------	-------------	-------------	-------------	------------

IADD	NOP	FADD	NOP	LOAD	STORE	NOP	NOP
ISUB	IMUL	NOP	NOP	NOP	NOP	NOP	NOP
IADD	NOP	NOP	NOP	NOP	NOP	NOP	BEG

NOP	IADD	NOP	FADD	STORE	LOAD	NOP	NOP
IMUL	ISUB	NOP	NOP	NOP	NOP	NOP	NOP
IADD	NOP	NOP	NOP	NOP	NOP	BEG	NOP

VLIW instruction encoding: compressed

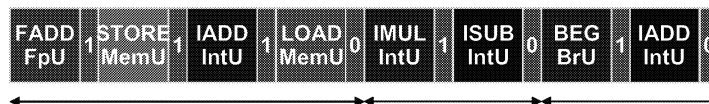
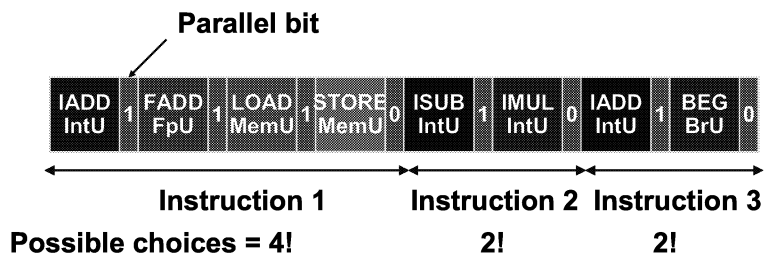
```

IADD /*IntU*/
|| FADD /*FpU*/
|| LOAD /*MemU*/
|| STORE /*MEMU*/

ISUB /*IntU*/
|| IMUL /*IntU*/

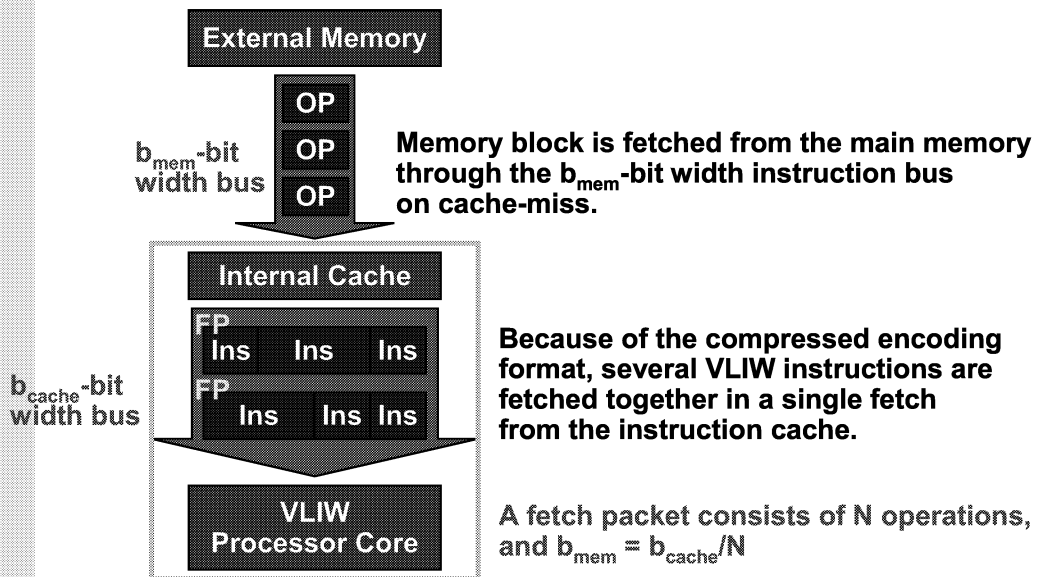
IADD /*IntU*/
|| BEG /*BrU*/
    
```

Alternative encoding



Which encoding is the best for low-power consumption?

Machine model



Problem formulation

Problem

how to reorder given VLIW instructions to reduce the number of bit transitions between successive instruction fetches.

Solutions

Local Operation Rearrangement (LOR) :
each basic block is independently considered.
Global Operation Rearrangement (GOR) :
all the basic blocks are simultaneously considered.

LOR problem

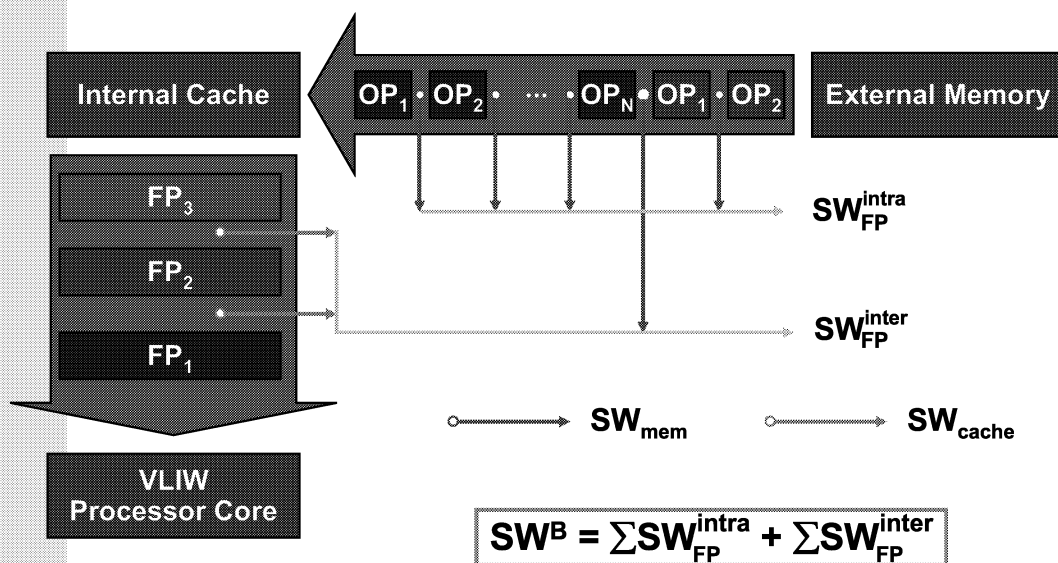
$$SW^B = SW_{cache}^B + \alpha \cdot SW_{mem}^B$$

α is the load capacitance ratio of the external instruction bus to the internal instruction bus.

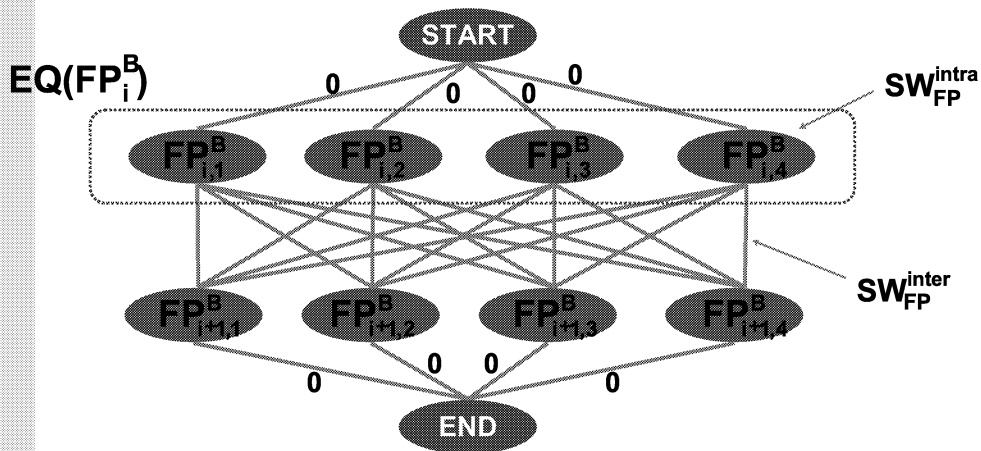
SW_{cache}^B is the number of bit changes at the internal instruction bus.

SW_{mem}^B is the number of bit changes at the external instruction bus.

LOR problem



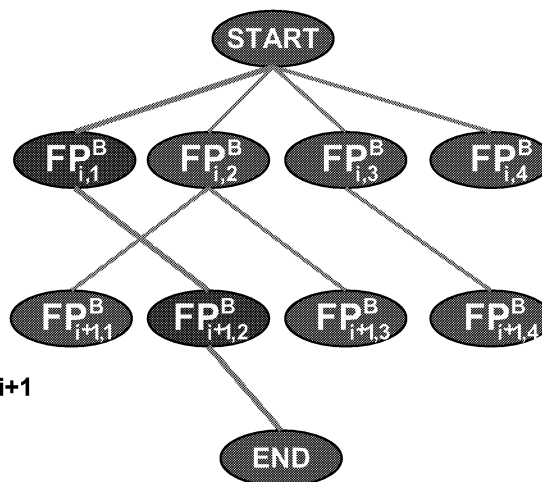
Solution for LOR



$EQ(FP_i^B)$: The set of equivalent fetch packets of FP_i^B .

Solution for LOR

- We find the shortest path from **START** to **END**, which is the solution of operation rearrangement to minimize the SW^B
- A node v_{i+1} in graph finds the node v_i through which the shortest path from **START** to the node v_{i+1} should pass.



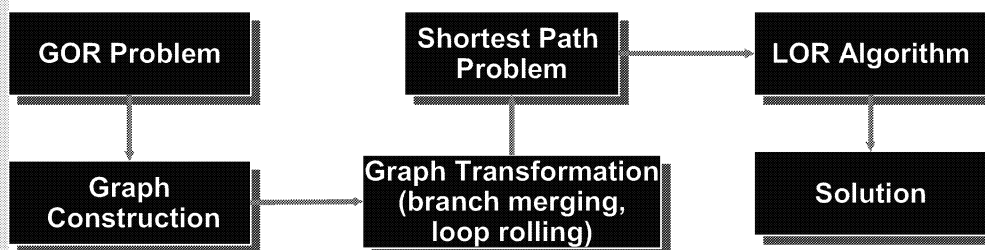
GOR problem

- All the basic blocks in a program are simultaneously considered
 - how many times each basic block is executed.
 - how often each basic block experiences cache misses.
 - how basic blocks are related each other.

$$SW^S = \sum \sum SW_{BB}^{inter}(bb_i, bb_j) + \sum SW_{BB}^{intra}(bb_i)$$

- SW_{BB}^{inter} and SW_{BB}^{intra} is represented by SW_{FP}^{inter} , SW_{FP}^{intra} , weight of each basic block, and cache miss rate.

Solution for GOR



This method may require an excessive amount of memory and cycles.



We need a heuristic solution.

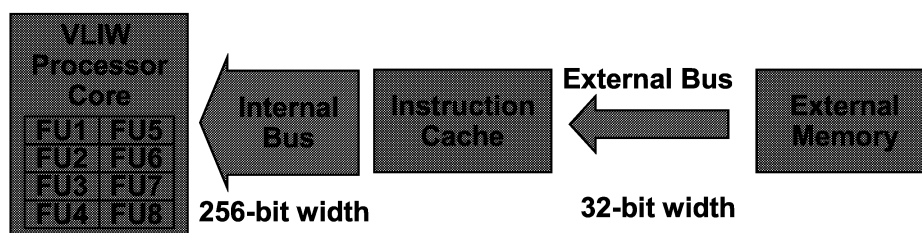
Heuristic for GOR

- All the basic blocks are not equally treated.
 - Basic blocks with larger effects on the total switching activity are more thoroughly reordered than ones with smaller effects.
- Not all the equivalent basic blocks in $EQ(bb_i)$ are tried to find an optimal solution.
 - Only N_{cand} equivalent basic blocks are created and included in graph.

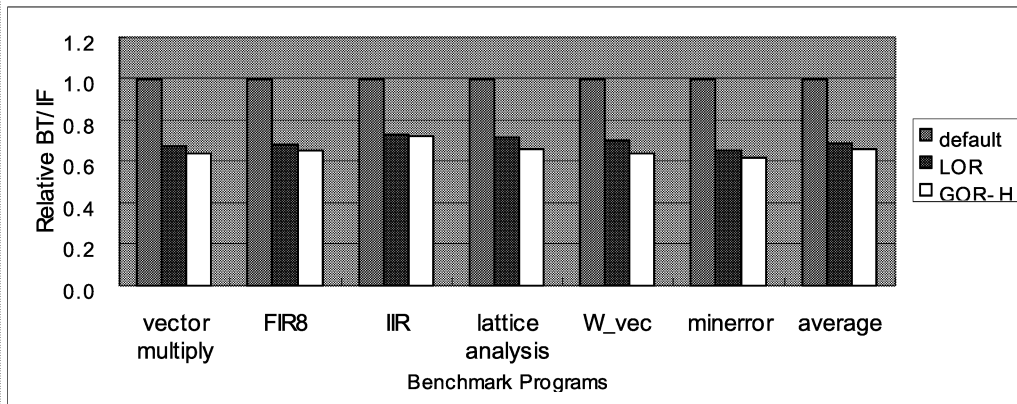
Experiment

TMS320C6201

- Fixed-point DSP
- VLIW processor that can specify eight 32-bit operations in a single 256-bit instruction.
- Use a compressed encoding



Experimental results



For our benchmark programs, the bit transitions was reduced by 34% on an average.

Conclusion

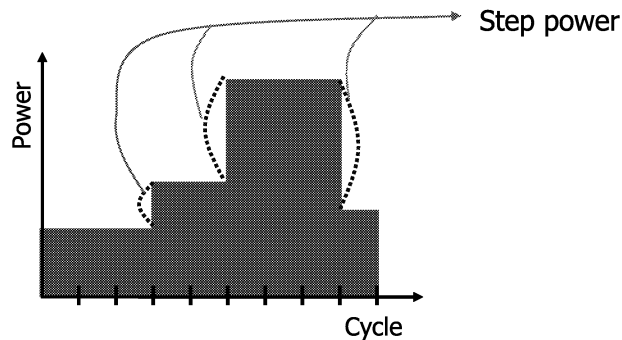
- **Described a post-pass optimal operation rearrangement method for low-power VLIW instruction fetch.**
 - **The switching activity was reduced by 34% on an average.**
- **Future works**
 - **The phase-ordering problem between the operation rearrangement and other compiler optimization steps.**
 - **Operation rearrangement problem in super-scalar processors.**

Battery-Aware Modulo Scheduling for VLIW Processors

Power Fluctuation

- In VLIW processors, power fluctuation significantly depends on the parallel schedule generated by compilers
- Closely related to battery-lifetime
 - As current fluctuation becomes larger, battery lifetime becomes shorter
- Battery-aware balanced modulo scheduling
 - Traditional power-unaware modulo scheduling algorithm is modified so that the power fluctuation is reduced
 - No performance loss nor additional energy consumption

Power Fluctuation



- **step power**
 - differences in the instantaneous power between consecutive cycles
 - Inductive noise $L \cdot di/dt$ (voltage glitch induced at power/ground buses) \Rightarrow timing & logic errors

Step-power Aware Compilation

- Programs spend most of the execution time in loops
 - Optimizing compilers (for VLIWs) perform software pipelining to shorten the execution time of loops
- The traditional power-unaware software pipelining can be modified so that the power fluctuation is reduced
- Quite effective in reducing the power fluctuation
 - The compiler can fully control the usage of all the FUs in a VLIW processor

VLIW machine model & power model

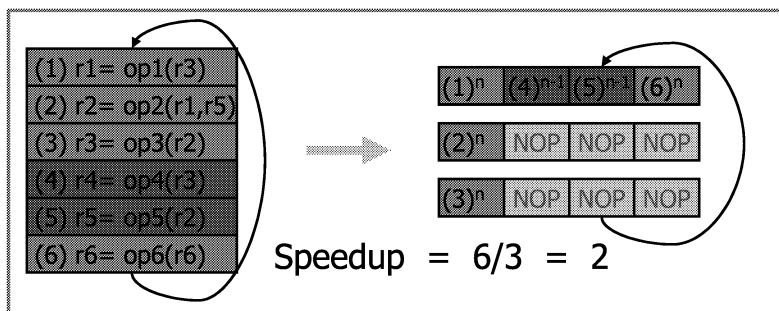
- MIPS-like integer pipeline, UltraSPARC-like FPU pipeline
- 8-issue VLIW model
 - 1 integer ALU , 2 load/store unit , 1 integer MPY/DIV
 - 2 FP ALU , 2 FP MPY/DIV
- 16-issue VLIW model : # of each FU is doubled
- Use instruction-level power model
 - ignore inter-instruction effect
- The proposed algorithm can be easily extended to work with more accurate power model
 - does not depend on a particular power model

ESSES 2003
2003/7/22 (Jihong Kim)

45

Software pipelining

- Aggressive fine-grained loop scheduling technique
 - For VLIW processors (e.g., Intel IA-64, TI C6x, ...)
- Essentially, equivalent to retiming technique used in VLSI synthesis
- Overlaps the execution of multiple iterations in a pipelined fashion



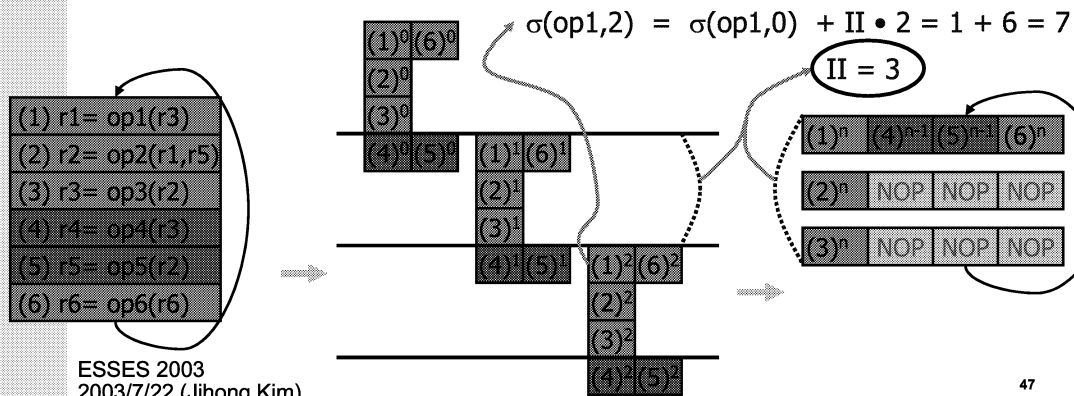
- **Modulo scheduling** is one of the scheduling algorithms for implementing software pipelining

ESSES 2003
2003/7/22 (Jihong Kim)

46

Traditional modulo scheduling formulation

- II : the length of an iteration of parallelized loop body
- $\sigma(op,i)$: execution cycle when the instance of operation op in iteration i is begin to execute
- Periodicity constraint : $\sigma(op,i) = \sigma(op,0) + II \cdot i$
- Goal : find the minimum II and a corresponding schedule $\sigma(op,0)$ for each v subject to dependence constraint and resource constraint

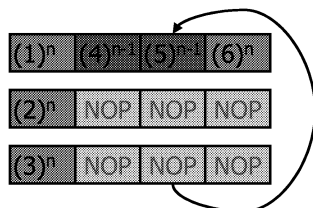


Power-aware modulo scheduling

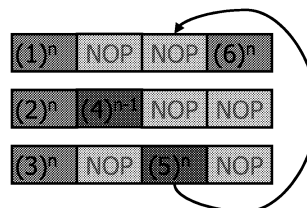
- Our goal

Given the II (found by traditional MS algorithm), find the schedule such that the power consumption distribution is as **flat** as possible

- No performance loss; no additional energy consumption

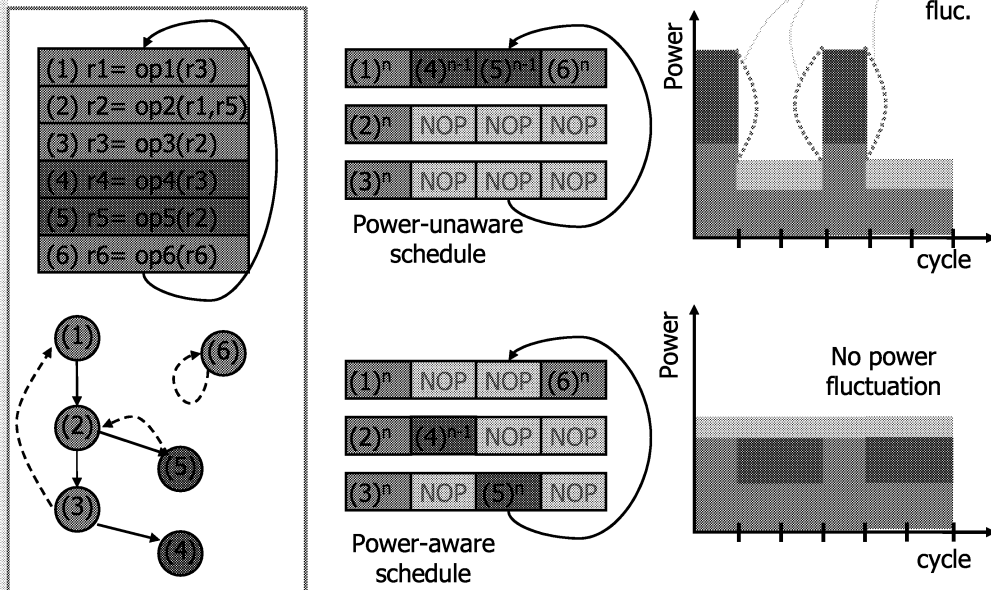


Traditional power-unaware schedule



power-aware schedule

Cycle-by-cycle power dissipation



ESSES 2003
2003/7/22 (Jihong Kim)

49

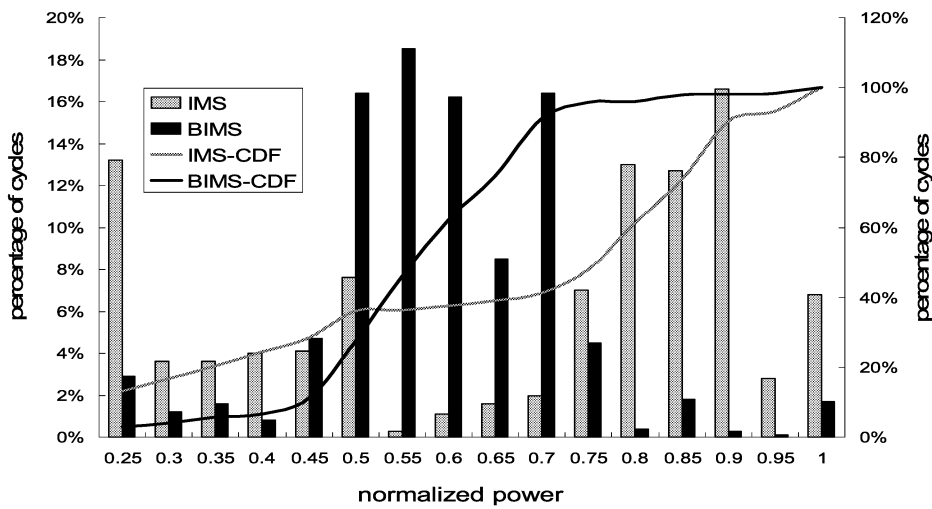
Experiment setting

- **Base algorithm**
 - **Iterative Modulo Scheduling (IMS)** [Rau, MICRO'94]
 - **Outperforms most of other MS algorithms**
- **Our power-aware algorithm : Balanced IMS (BIMS)**
- **Battery lifetime model** [Pedram, DAC'99]
- **SPEC95 FP benchmark programs**
- **SPARC-based VLIW testbed** [Moon, MICRO'97]
 - **8 & 16-issue VLIW**

ESSES 2003
2003/7/22 (Jihong Kim)

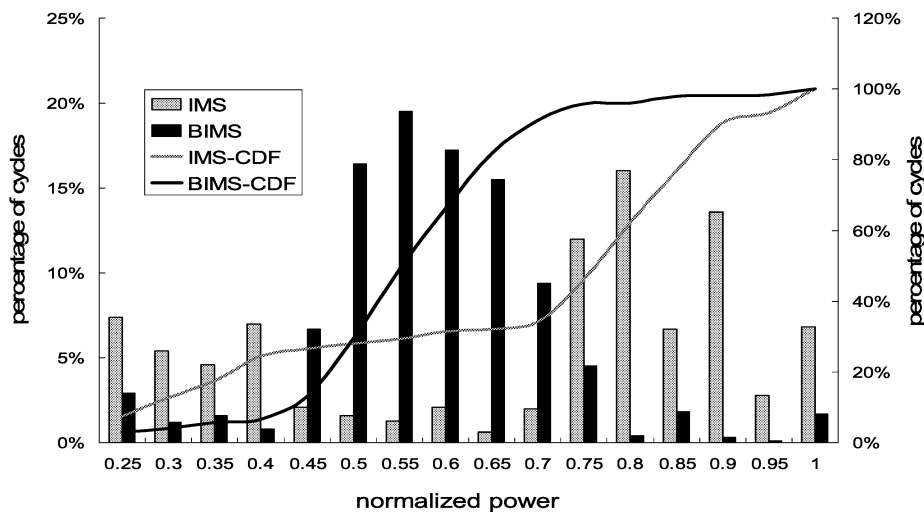
50

Power distribution for 8-issue VLIW



Battery lifetime: 28% increased

Power distribution for 16-issue VLIW



Battery lifetime: 31% increased

Conclusion

- **Quite effective in reducing the power fluctuation**
 - **The compiler can fully control the usage of all the FU in a VLIW processor**
- **Battery lifetime increases significantly**
 - **29% for 8-issue VLIW**
 - **31% for 16-issue VLIW**

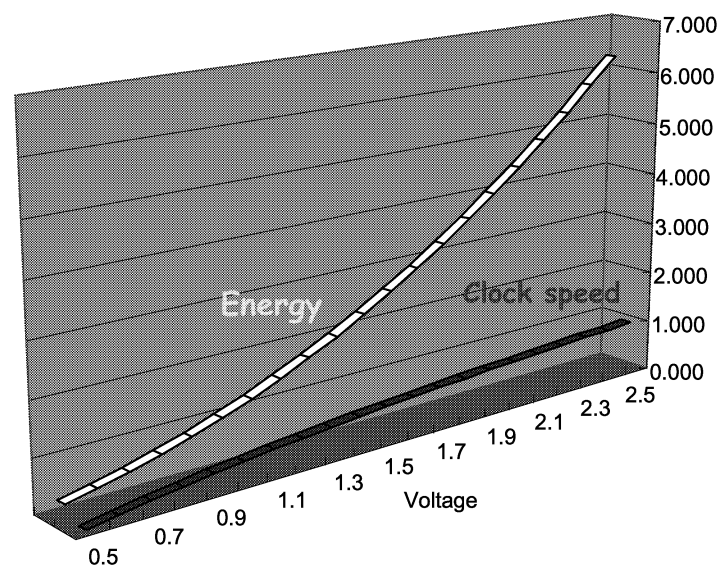
References

- **D. Shin and J. Kim, "Operation Rearrangement for Low Power VLIW Instruction Fetch", Proc of DATE, 2001**
- **H.-S. Yun and J. Kim, "Power-Aware Modulo Scheduling for High-Performance VLIW Processors", Proc of ISLPED, 2001**

Roadmap

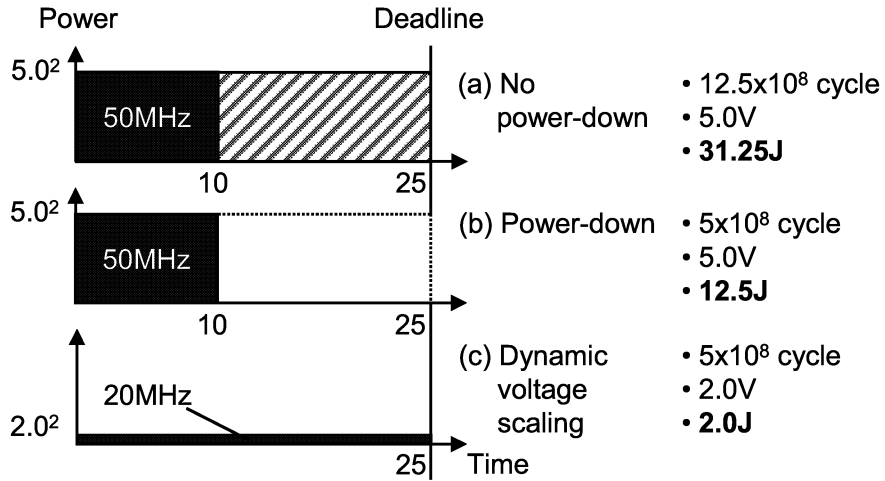
- ✓ Low-power systems 101
- ✓ Low-power binary encoding
- ✓ Power-aware compiler techniques
- Dynamic voltage scaling techniques
 - intraDVS
 - interDVS
 - **Low-power convolution**

Voltage, Frequency & Energy



DVS 101

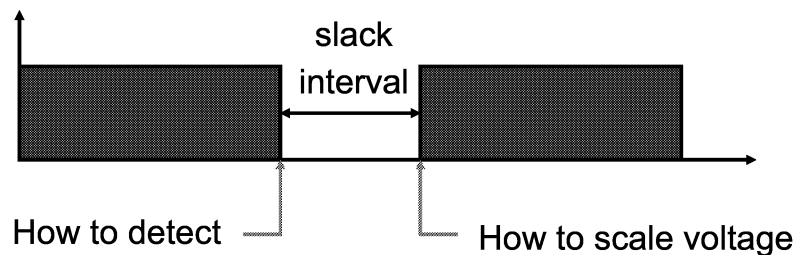
$$E \propto N_{\text{cycle}} \cdot V_{DD}^2$$



→ **Slow and Steady wins the race!**

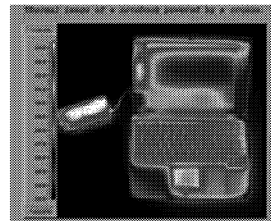
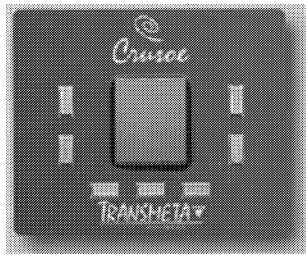
Key Issues for successful DVS

- Efficient Detection of Slack/Idle Intervals
- Efficient Voltage Scaling Policy for Slack Intervals



Commercial DVS Processors

- Transmeta Crusoe
- AMD K2+ (PowerNow Technology)
- Intel SpeedStep
- XScale



ESSES 2003
2003/7/22 (Jihong Kim)

59

Voltage Scaling Processors

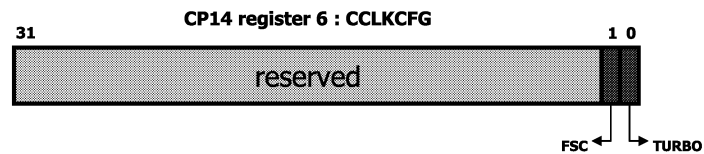
	Commercial			Academic	
Processors	Transmeta Crusoe (LongRun)	AMD Mobile K6 (PowerNow)	Intel PXA250	UC Berkely (ARM8)	Uvicom LART(StrongARM)
Scaling Level	200~700MHz 1.1~1.65V	192~588MHz 0.9~2.0V	100~400MHz 0.85~1.3V	5~80MHz 1.2~3.8V	59~251MHz 0.79~1.65V
Scaling Time	1.1 ↔ 1.65V < 300μs	0.9 ↔ 2.0V 200μs	Each step 500μs	1.2 ↔ 3.8V 520μs	59↔251MHz : 140μs 0.79→1.65V : 40μs 0.79←1.65V : 5.5ms
Scaling Power	??	??	??	130μJ	??

ESSES 2003
2003/7/22 (Jihong Kim)

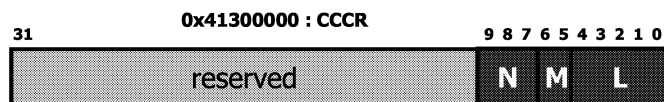
60

DVS in XScale

- Use Two Registers in PXA250 Xscale Core
 - CCCR (Core Clock Configuration Register):
 - Specify memory clock & core clock
 - CCLKCFG (Core Clock Configuration) Register
 - Set FCS (Frequency Change Sequence) bit to change the clock speed



Change if FCS bit = 1



L	M	N							
		2		3		4		6	
1	1	99.5	.85V			199.1	1.0V	298.6	1.1V
2	1	118.0	1.0V			235.9	1.1V	353.9	1.3V
3	1	132.7				265.4		398.9	
4	1	147.5				294.9			
5	1	165.9				331.8	1.3V		
1	2	199.1		298.6	1.1V	398.1	1.3V		
2	2	235.9	1.1V						
3	2	265.4							
4	2	294.9							
5	2	331.9		1.3V					

Ex) DVS on PAX250

```
1 #include <machine/pmu.h>
2 #include <machine/cp14.h>
3
4 int thread_args[3] = {0, 1, 2};
5
6 void
7 change_clock_speed(int speed)
8 {
9     int settings[20] = { 0, 0x121, 0x122, 0x123, 0x124, 0x125, 0x1a2,
10                        0x141, 0x1a4, 0x142, 0x1a5, 0x143, 0x144, 0x145 };
11     int cccr_val = 0x121, clkcfg_val = 2;
12
13     cccr_val = settings[speed];
14     switch (speed) {
15         case 6 : clkcfg_val = 3; break;
16         case 8 : clkcfg_val = 3; break;
17         case 10 : clkcfg_val = 3; break;
18         default : clkcfg_val = 2; break;
19     }
20     memcpy(0x40000000+0x1300000, &cccr_val, 4);
21     CP14_WRTIE_CCLKCFG(clkcfg_val);
22 }
23
24 int
25 get_os_time()
26 {
27     int ostime;
28
29     memcpy(&ostime, 0x40000000+0xa00010, 4);
30     return(ostime);
31 }
```

```
void Main(void)
{
    int i, bb, cc, j;

    for ( k = 1 ; k < 13 ; k++ ) {
        change_clock_speed(k);
        bb = get_os_time();
        for ( i = 0 ; i < 10000 ; i++ ) j = 10;
        cc = get_os_time();
        printf("%d : %d \n", k, cc - bb);
    }
}
```

DVS Outline

- DVS for Non-Real Time Systems
- **DVS for Real-Time Systems**
 - Intra-task DVS design techniques
 - Inter-task DVS design techniques
 - DVS-aware algorithm development

Non Real-Time Jobs

- **Non Real-Time Jobs**
 - **No timing constraints**
 - **Non-periodic execution**
 - **Unknown WCET**

It is hard to predict the future workload!!

DVS for Non-Real-Time Jobs

- **Usually based on *Interval Scheduler***
 - **PAST, FLAT**
 - **LONG_SHORT, AGED_AVERAGE**
 - **CYCLE, PATTERN, PEAK**

Major Issues

How can we predict the future ?

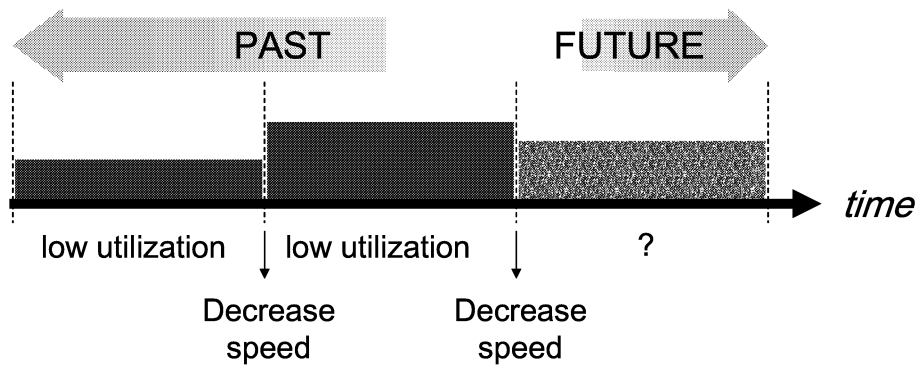
- Based on long term history:
Hard to adapt quickly for the changed workload
- Based on short term history:
Too many clock/voltage changes

PAST

- **Looking a fixed window into the past**
- **Assume the next window will be like the previous one**
- **If the past window was**
 - **mostly busy \Rightarrow increase speed**
 - **mostly idle \Rightarrow decrease speed**

Example - PAST

$$\text{Utilization} = \frac{\text{busy time}}{\text{window size}}$$

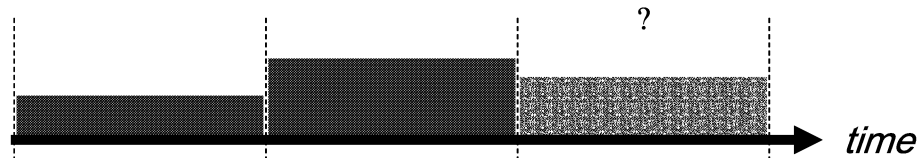


FLAT

- Try to smooth speed to a global average
- Make the utilization of next window to be $\langle \text{const} \rangle$
 - Set speed fast enough to complete the predicted new work being pushed into the coming window

Example - FLAT

$\langle \text{Const} \rangle = 0.7$



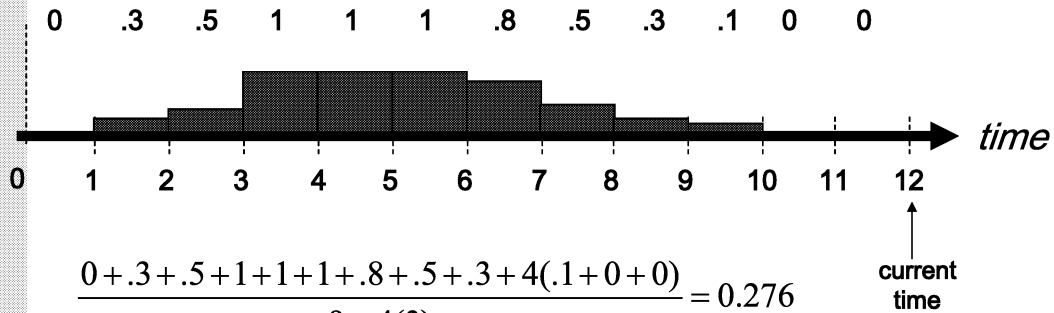
Increase/Decrease the speed
the next utilization to be 0.7

LONG-SHORT

- **Look up the last 12 windows**
 - **Short-term past : 3 most recent windows**
 - **Long-term past : the remaining windows**
- **Prediction**
 - **the utilization of next window will be a weighted average of these 12 windows' utilizations**

Example – LONG-SHORT

utilization = # cycles of busy interval / window size



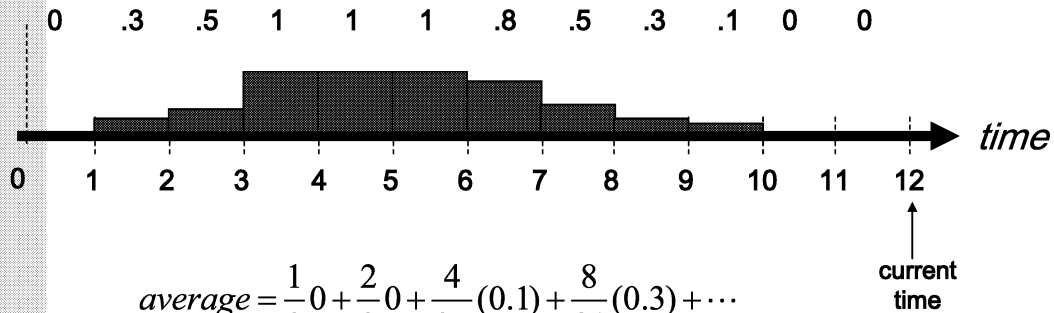
$$f_{clk} = 0.276 \times f_{max}$$

AGED-AVERAGE

- Employs an exponential-smoothing method
- Prediction
 - The utilization of next window will be a weighted average of all previous windows' utilization
 - geometrically reduces the weight

Example – AGED_AVERAGE

utilization = # cycles of busy interval / window size

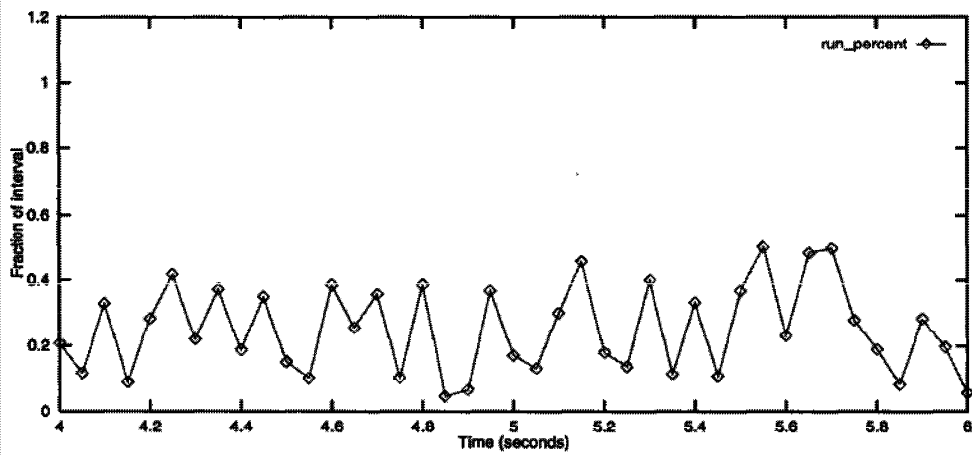


$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \dots$$

$$f_{clk} = average \times f_{max}$$

CYCLE

- Prediction
 - Examine the last 16 windows
 - Does there exist a cyclic of length X?
 - If so, predict by extending this cycle
 - Otherwise, use the FLAT algorithm

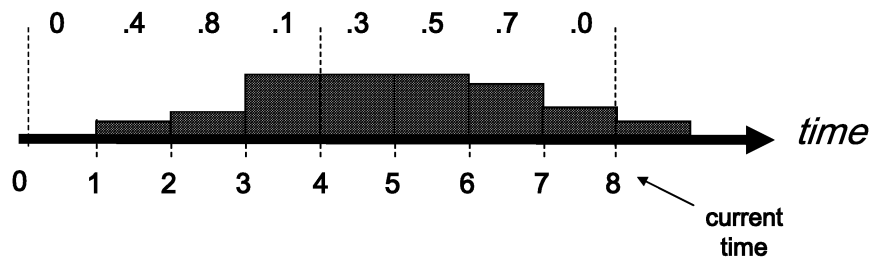


ESSES 2003
2003/7/22 (Jihong Kim)

77

Example - CYCLE

utilization = # cycles of busy interval / window size



$$\text{error measur: } \frac{|0 - .3| + |.4 - .5| + |.8 - .7| + |.1 - 0|}{4} = 0.15$$

Predict : The next utilization will be .3

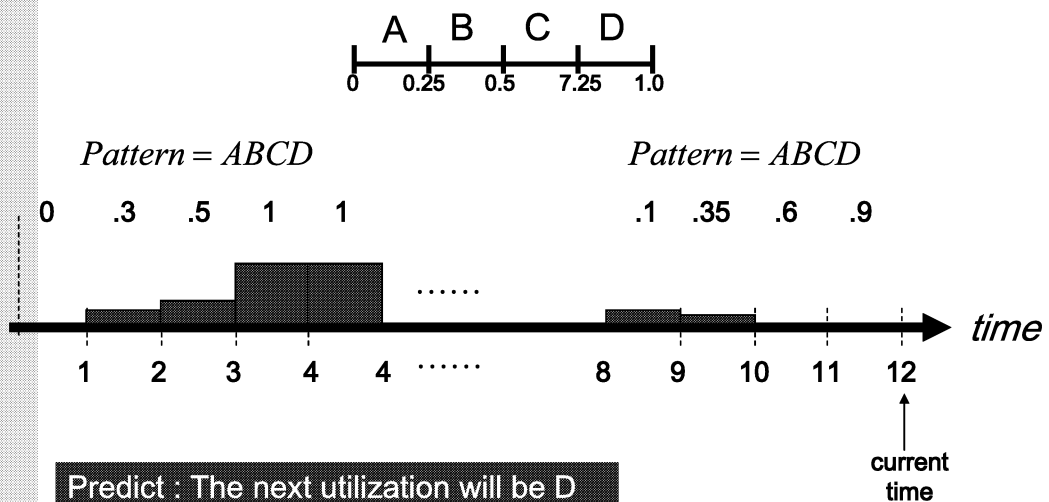
ESSES 2003
2003/7/22 (Jihong Kim)

78

PATTERN

- Generalized version of CYCLE
- Prediction
 - Convert the n-most recent windows' utilization into a pattern in alphabet {A, B, C, D}.
 - Find a same pattern in the past
- Set the speed based on the prejudication

Example - PATTERN



DVS Outline

- **DVS for Non-Real Time Systems**
- **DVS for Real-Time Systems**
 - **Intra-task DVS design techniques**
 - **Inter-task DVS design techniques**
 - **DVS-aware algorithm development**

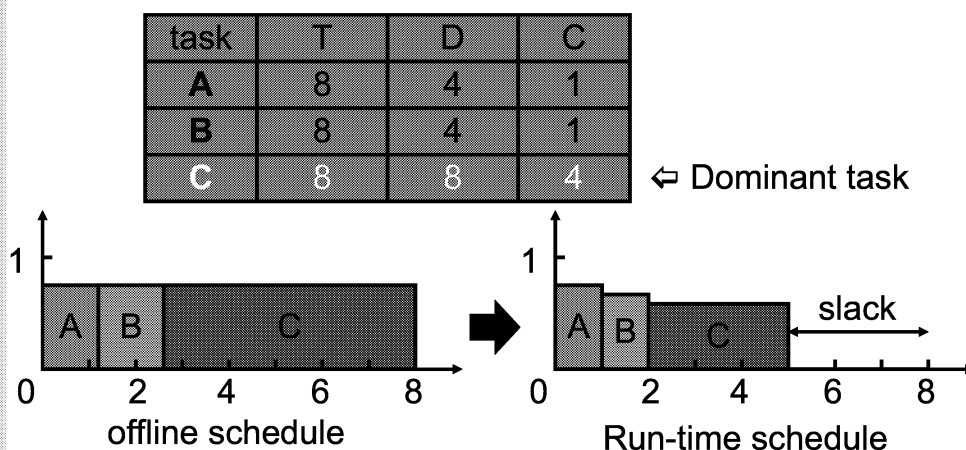
Two Types of DVS Algorithms

- **Inter-task DVS algorithms**
 - **Determine the supply voltage and clock speed on *task-by-task* basis**
- **Intra-task DVS algorithms**
 - **Determine the supply voltage and clock speed within a single task boundary**

Related Work : Inter-task DVS

- **Inter-Task Voltage Scheduling for Hard Real-Time Systems [Yao95, Hong98, Okuma99, Shin99, Lee99].**
 - **Problem** : Given a set of tasks, how to assign the proper speed to each task dynamically while guaranteeing all their deadlines.
 - **Task-by-task Speed Assignment**
 - The slack time due to a task used by following tasks, not by the current one.
 - **Practical Limitations**
 - Requires OS modifications
 - Cannot be applied to a single-task environment
 - Can be ineffective in a multi-task environment

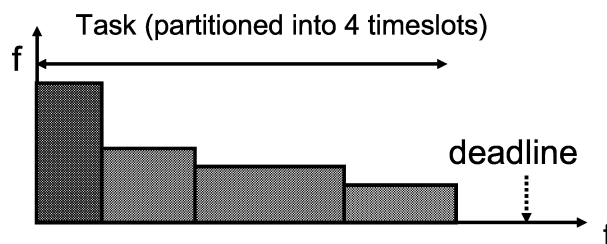
An Example of Ineffective Inter



- **A dominant task (C)**
 - Exploits small slack times from other tasks.
 - Cannot use its own.

Related Work : Intra-task DVS

- **Run-time voltage hopping [Lee00]**
 - Each task is **partitioned** into **N timeslots**.
 - Frequency and voltage determined for each timeslot.
 - Voltage scheduling **embedded** in application programs.
 - Can be applied to conventional **non-DVS OS**.
 - No systematic guideline
 - Manual selection of scaling points
 - Too much burden for average programmers



ESSES 2003
2003/7/22 (Jihong Kim)

85

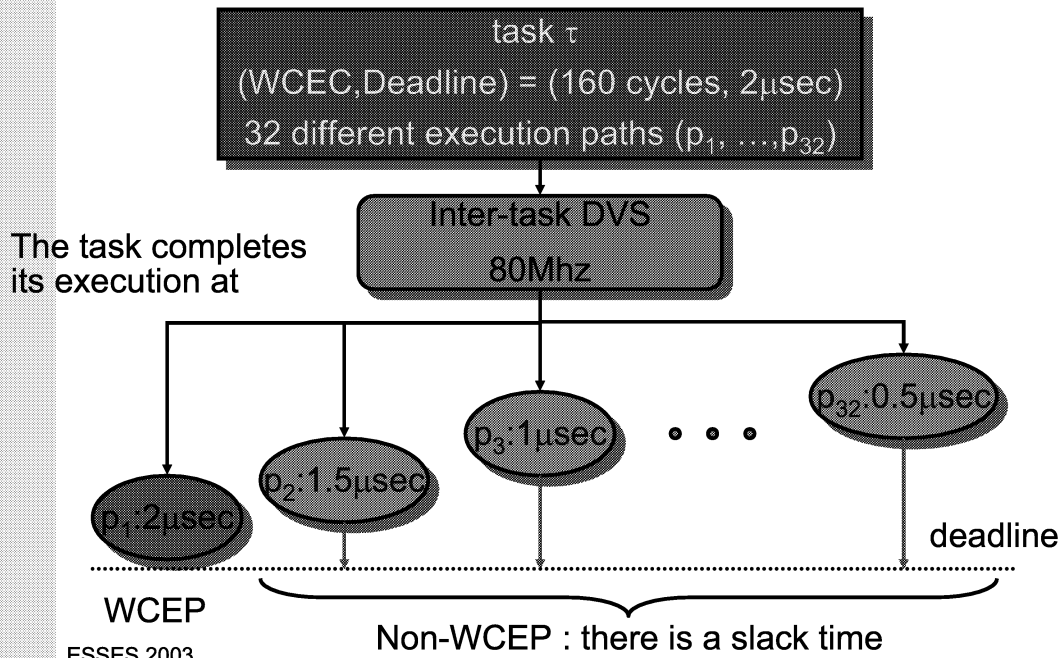
Overview

- **A novel intra-task voltage scheduling framework based on a static timing analysis of RT programs.**
 - The clock speed is adjusted in a task by embedded codes.
 - Fully exploits all slack times coming from execution time variations within a single task
 - No OS modification
 - Applicable to a single-task environment.
 - Provides a systematic methodology for developing DVS-aware programs
 - ⇒ Automatic Voltage Scaler Tool

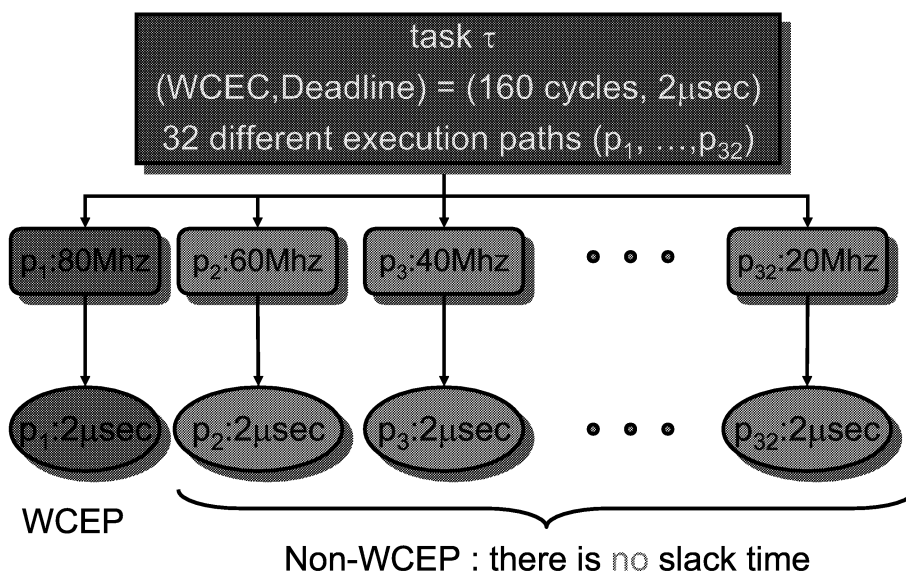
ESSES 2003
2003/7/22 (Jihong Kim)

86

Basic Idea : Inter-task DVS

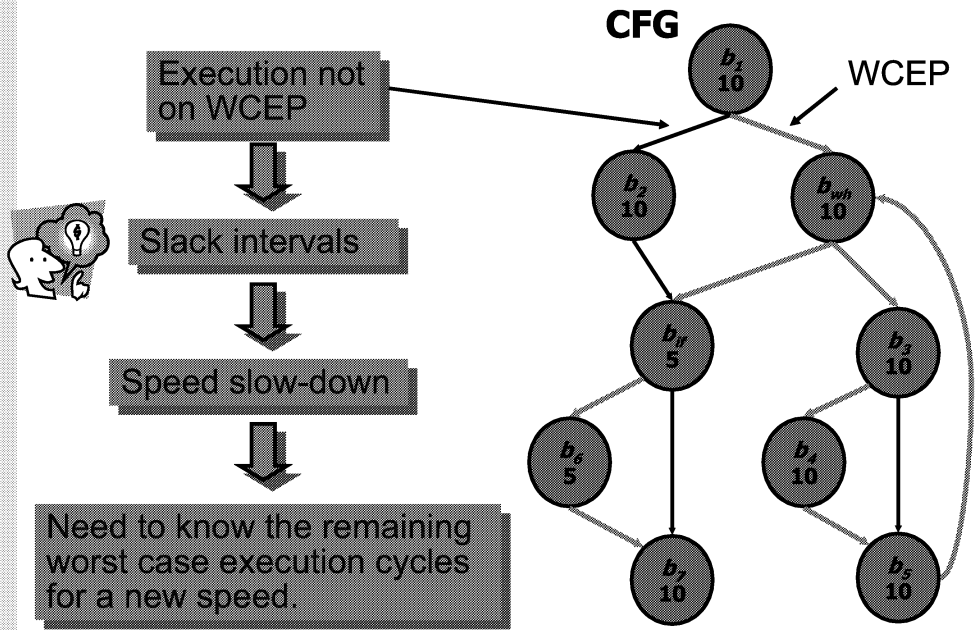


Basic Idea : Optimal DVS

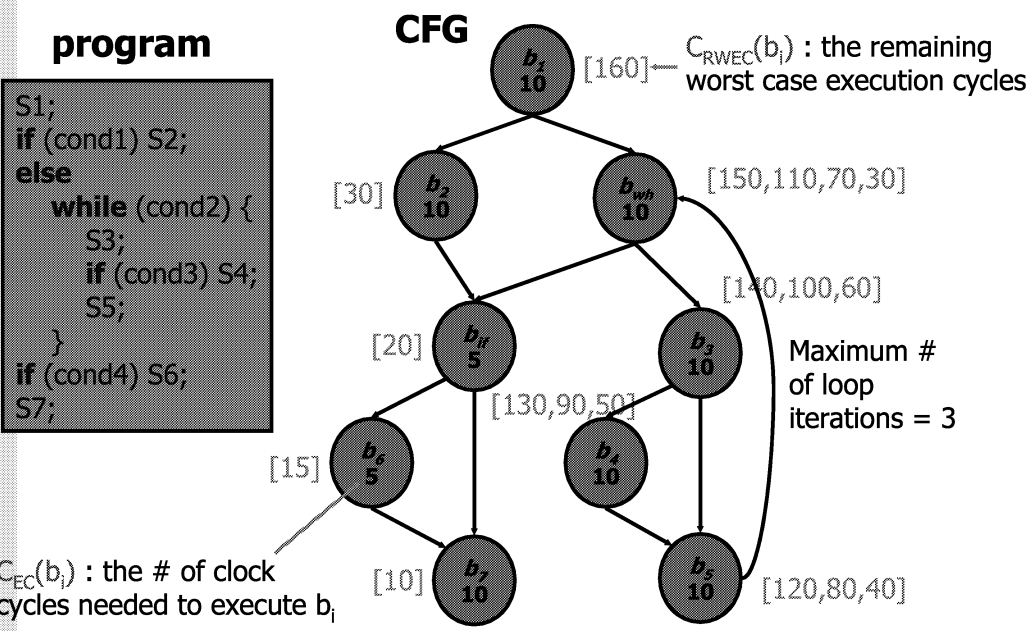


But we cannot know the execution path in advance !!

Basic Idea : Intra-task DVS

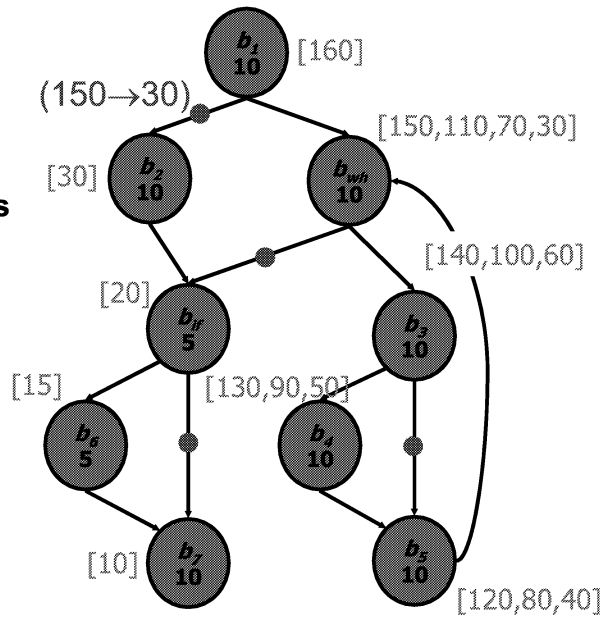


Remaining Execution Cycle

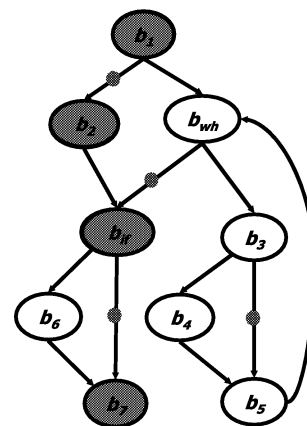
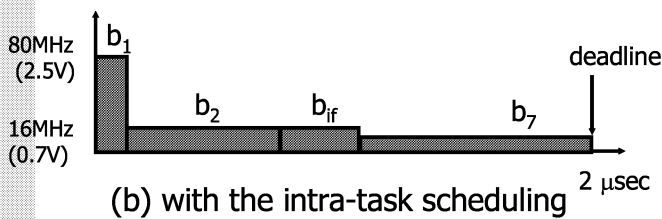
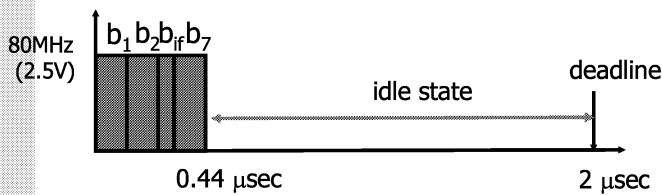


Speed Assignment Algorithm

- **Branching edges are candidate for speed changes :**
 - **Branches :** The execution control follows the shorter path at if-then-else node. => B-type VSEs
 - **Loops :** The execution control exits a loop after it iterates by the smaller number of times than the maximum iteration number. => L-type VSEs



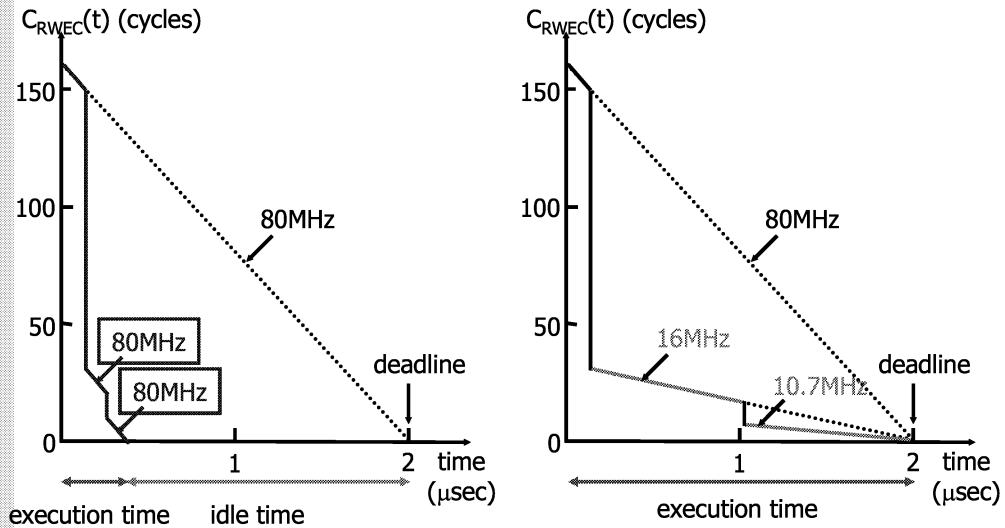
The Effect of Intra-Task



$$\frac{\text{Energy Consumption of (b)}}{\text{Energy Consumption of (a)}} = 0.34$$

The Change of C_{RWEC}

For execution path : (b_1, b_2, b_{if}, b_7)



(a) No intra-task scheduling

(b) Intra-task scheduling

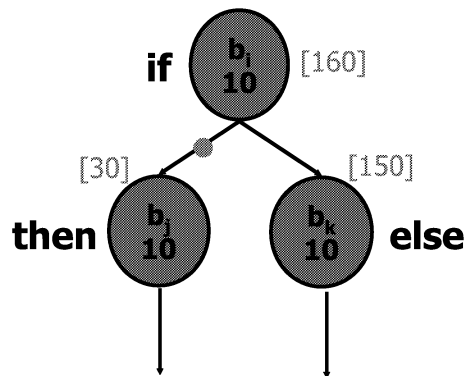
ESSES 2003
2003/7/22 (Jihong Kim)

93

B-type VSE

- C_{RWEC}
 - 150 \rightarrow 30
- Speed
 - $S(b_j) \leftarrow S(b_i) \times 1/5$

↑
Speed update ratio



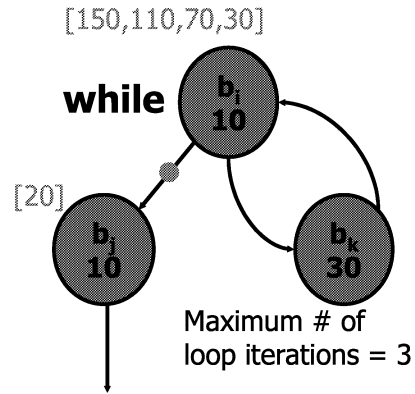
ESSES 2003
2003/7/22 (Jihong Kim)

94

L-type VSE

- When the actual loop iterations measured at run time is 2
- C_{RWEC}
 - $60 \rightarrow 20$
- Speed
 - $S(b_j) \leftarrow S(b_i) \times 1/3$

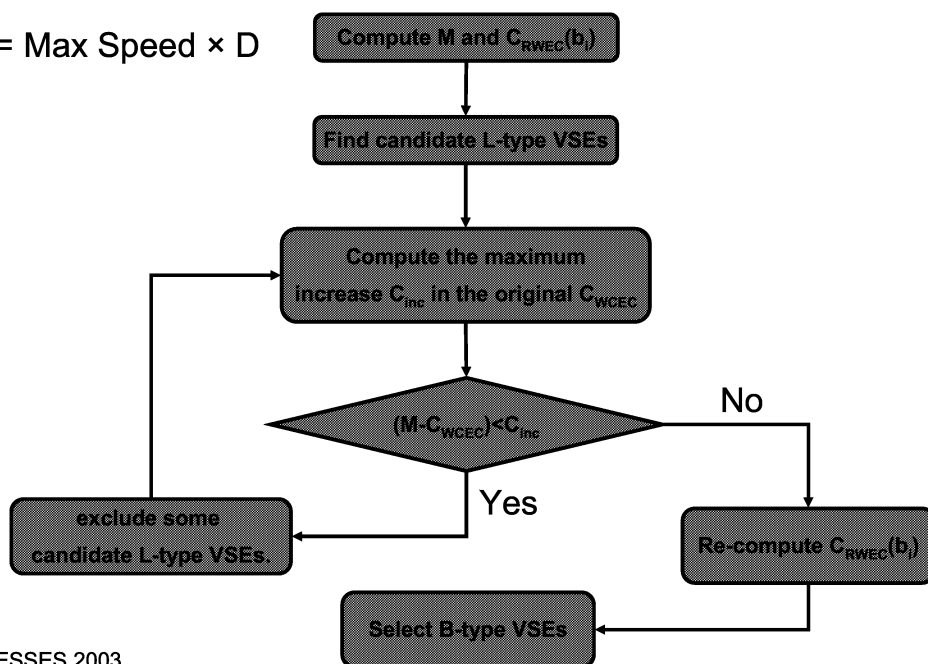
↑
Speed update ratio



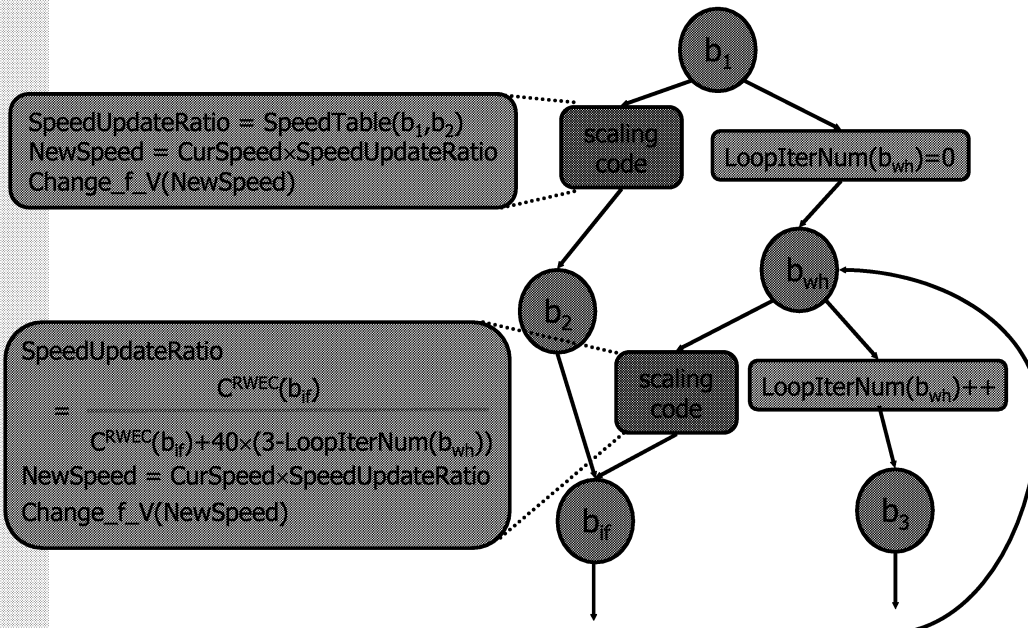
⇒ The transition overhead is considered to determine a new speed.

VSE Selection

$$M = \text{Max Speed} \times D$$



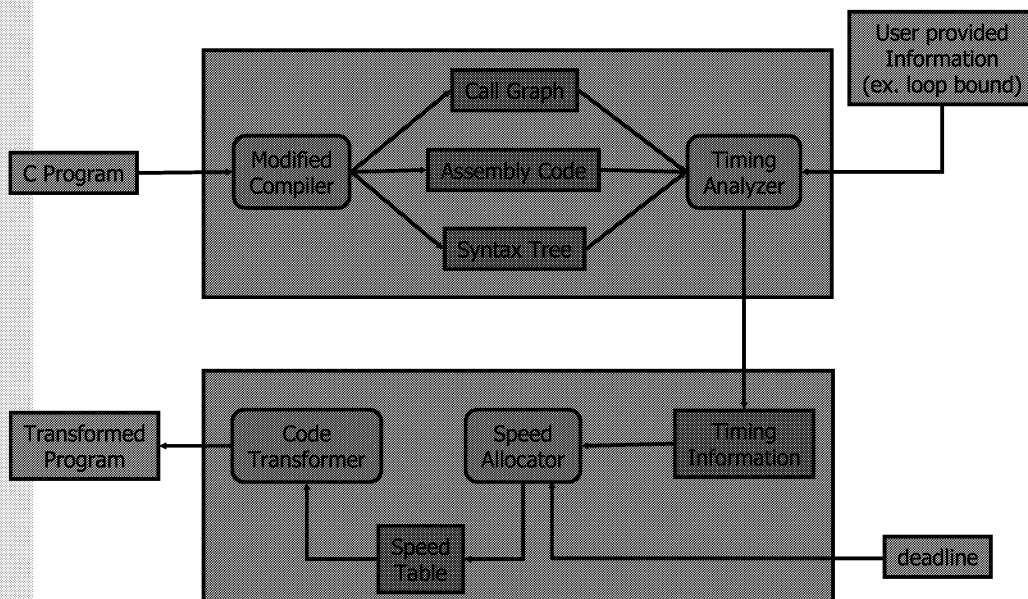
Code Generation for VSEs



ESSES 2003
2003/7/22 (Jihong Kim)

97

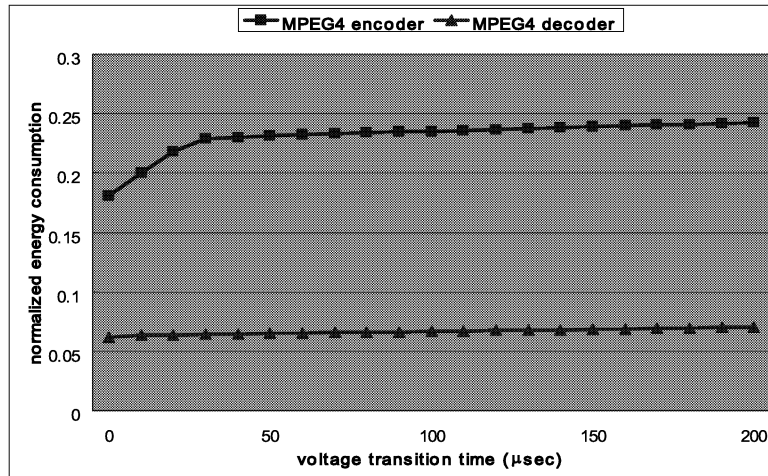
Automatic Voltage Scaler



ESSES 2003
2003/7/22 (Jihong Kim)

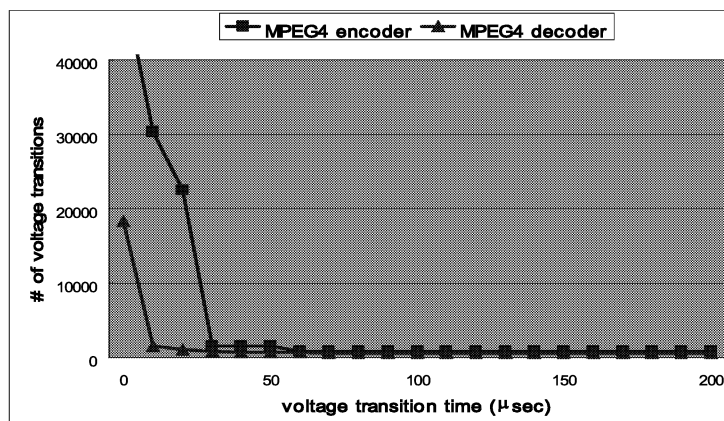
98

Simulation Results



- ◆ Less than 25% and 7% of the original program
- ◆ There is a large difference between WCET and ACET of the MPEG-4 decoder

Simulation Results

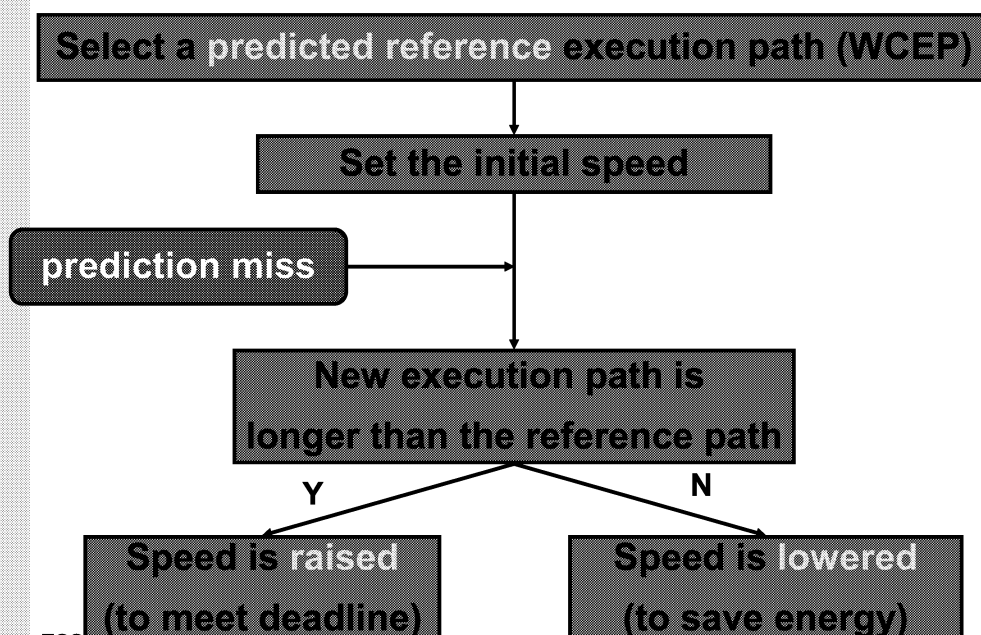


- How many times voltage scaling code were executed
- When $C_{VTO} < 30\mu\text{sec}$ in MPEG-4 encoder, the number of voltage transitions decreases sharply, and energy consumption does not increase rapidly.
- How many copies of voltage scaling code ?
 - 20 VSEs are inserted when $C_{VTO} > 50 \mu\text{sec}$.

A Profile-Based Intra-Task

- IntraVS algorithm based on *average-case* execution information
- **Average-case execution paths (ACEPs)** are the most frequently executed paths
- **More effective than the original intraVS algorithm**
- **The timing constraints of a hard real-time program is still satisfied, even if the ACEPs are used for voltage scaling decisions.**

General IntraVS Algorithm

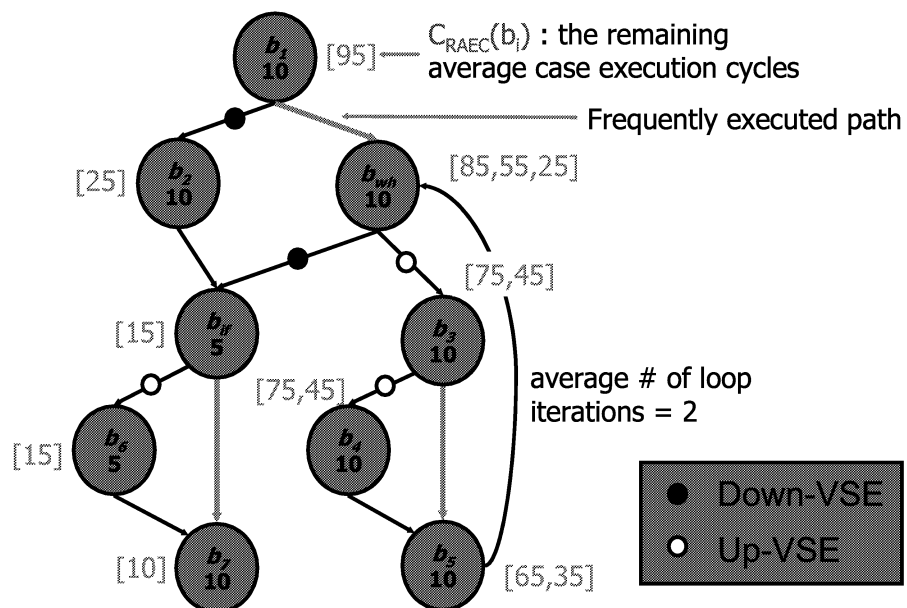


RAEP-based IntraVS

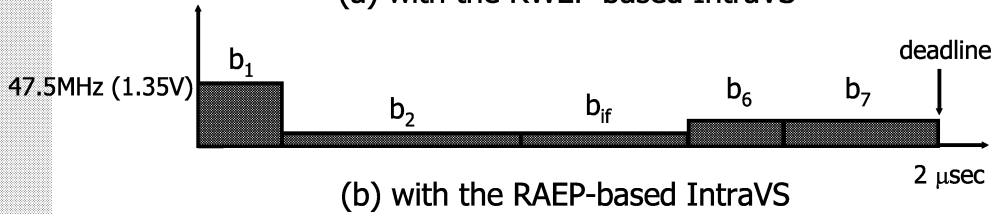
- **Motivations**

- **To make the common case more energy-efficient**
- **If we use one of hot paths as a reference path for intraVS, the speed change graph for the hot paths will be a near flat curve with little changes in clock speed.**
- **Even for the paths that are not the hot paths, they are more energy-efficient because they can start with a lower clock speed that RWEV-based IntraVS.**
- **RAEP is the best representative of the hot paths.**

RAEP-based IntraVS

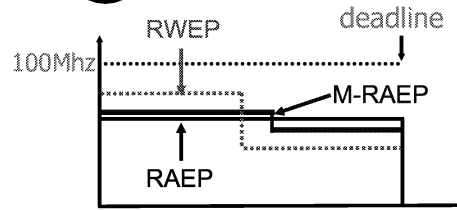
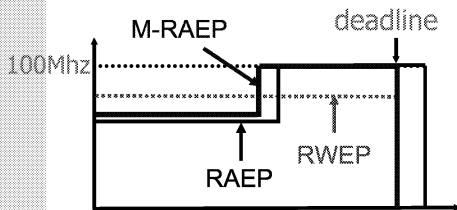
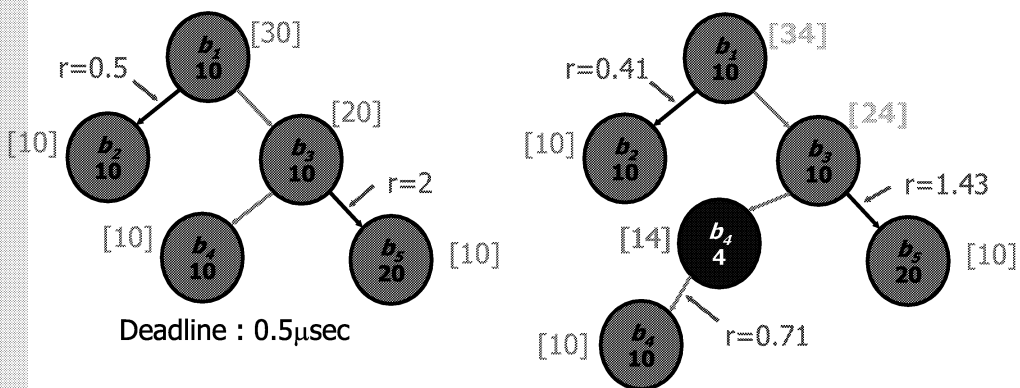


RAEP-based IntraVS

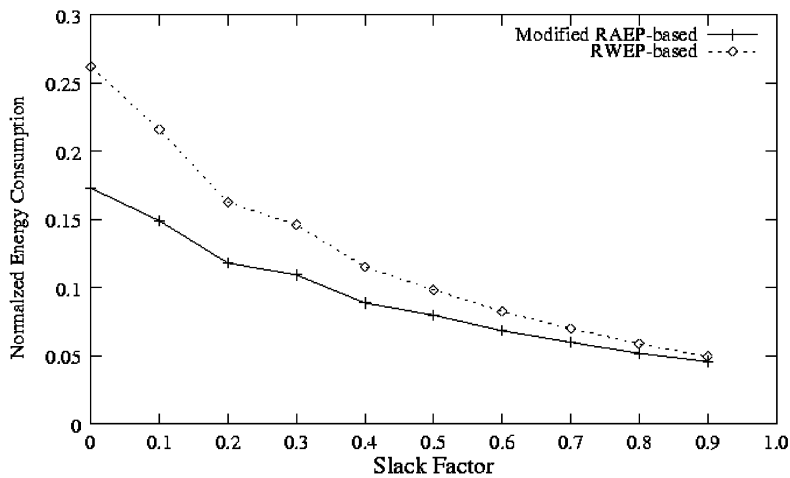


- There are Up-VSEs as well as Down-VSEs at the RAEP-based IntraVS.
- The RAEP-based IntraVS achieves 55% more energy reduction.
- But, the deadline can be missed.

Reference Path Modification



Experimental Results

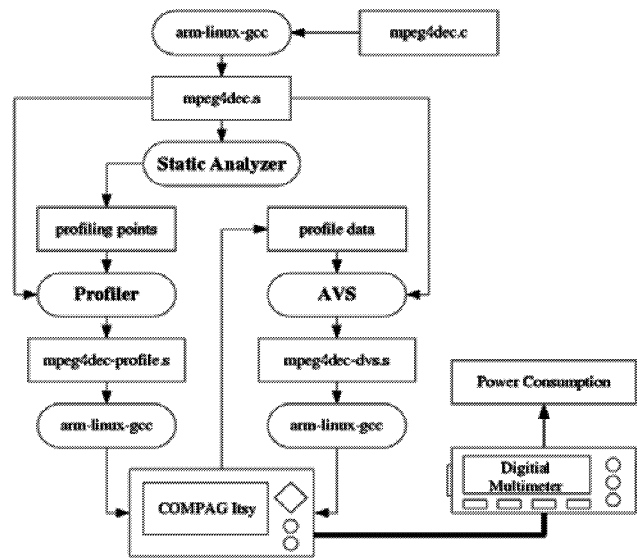


$$\text{Slack Factor} = (\text{deadline} - \text{WCET}) / \text{deadline}$$

Conclusion

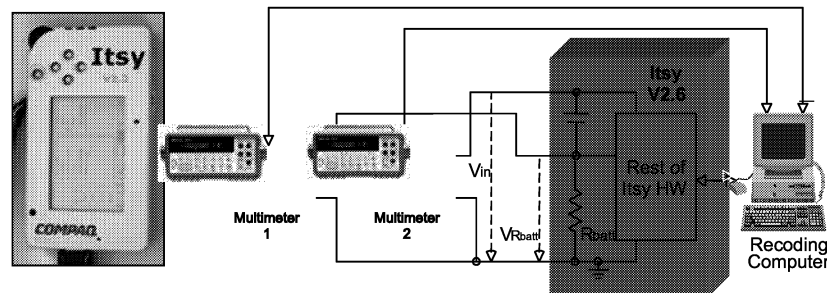
- Presented a novel intra-task DVS algorithm using static timing analysis on RWECS
- Provides a framework for automatic DVS-aware low-power program generation
- The RAEP-based IntraVS algorithm exploits the fact that the average-case execution paths are more likely to be followed at run time than the WCEP.
- Demonstrated the effectiveness of the approach using MPEG-4 encoder/decoder programs

Experiments on Itsy



Experimental Environment

- Itsy Pocket Computer V2.6
 - CPU : Intel StrongARM SA1110
 - Frequency scaling: 11 levels (59.0 MHz ~ 206.4 MHz)
 - Voltage scaling: 30 levels (1.00 V ~ 2.00 V)
 - Default setting: 1.55 V/206.4 MHz
- Linux operating system (ver. 2.0.30)



Experimental Results

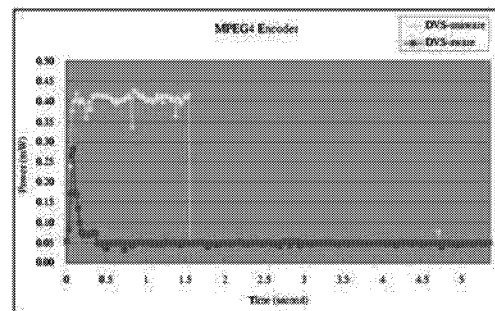
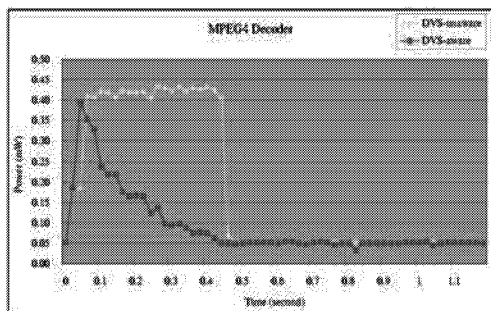
DVS EXPERIMENTS ON ITSY.

Factors	MPEG-4 Decoder		MPEG-4 encoder	
	DVS-aware	normal	DVS-aware	normal
Energy (mJ)	0.11	0.22(0.18)	0.28	0.81(0.62)
Normalized Energy	0.51(0.62)	1	0.35(0.46)	1
Execution Time (sec)	1.18	0.46	5.34	1.54
WCET (sec)	3.2		21.0	
Selected VSEs	B-VSE	2	1	
	L-VSE	1	2	
Management Code	Function	2	3	
	Loop	5	8	

MANAGEMENT CODE OVERHEAD.

Management Code	Code number in Fig. 7	Number of assembly instructions
Loop Enter	code 1 and 2	30
Loop Header	code 3	16
Loop Exit	code 4	16
Function Enter	code 5	14
Function Return	code 6	11
B-type VSE	code B	≈ 200
L-type VSE	code L	≈ 200

Experimental Results



Roadmap

- ✓ Low-power systems 101
- ✓ Low-power binary encoding
- ✓ Power-aware compiler techniques
- Dynamic voltage scaling techniques
 - ✓ intraDVS
 - interDVS
 - **Low-power convolution**

Introduction

- **Inter-task DVS algorithms**
 - **Determine the supply voltage and clock speed on *task-by-task* basis**
- **Inter-task DVS**
 - **Is similar to that of imprecise computation in conventional real-time systems**
 - **Imprecise computation**
 - Use the slack time to increase the values of results
 - While guaranteeing the feasible schedule of tasks
 - **Dynamic voltage scaling**
 - Use the slack time to lower the voltage/clock speed
 - While guaranteeing the feasible schedule of tasks

Preliminaries

- **Computing model**
 - **Non-real-time**
 - tasks have no timing constraints
 - **Real-Time**
 - Timing constraints
 - Periodic and(or) aperiodic tasks
 - Scheduling policy : EDF, RM, and etc.
- **Different DVS algorithms are necessary depending on different computing models.**

Inter-Task DVS

- **“Run-Calculate-Assign-Run” strategy for the supply voltage determination**
 - **Running the current task**
 - **Calculating the maximum allowable execution time for the next task**
 - WCET plus slack time
 - **Assigning the supply voltage for the next task**
 - **Running the next task**

Generic Inter-DVS Algorithms

- **Consist of two parts**
 - **Slack estimation**
 - Identify as much slack times as possible
 - **Slack times**
 - **Static slack times**
 - Extra times available for the next task that can be identified statically
 - **Dynamic slack times**
 - Ones caused from run-time variations of the task executions
 - **Slack distribution**
 - Adjust the speed so that the resultant speed schedule is as flat as possible

Static and Dynamic Approaches

- **Off-line (Static) voltage scheduling approaches**
 - The execution times are assumed to be known a priori
 - There are several optimal solutions for EDF, RM, and etc.
- **On-line (Dynamic) voltage scheduling approaches**
 - The execution times are assumed to be not known
 - There cannot be an optimal solution

Slack Estimation Methods

	Voltage Scaling Methods	Scaling Decision
IntraDVS	(1) Path-based method	Off-line
	(2) Stochastic method	
InterDVS	(3) Maximum constant speed	On-line
	(4) Stretching to NTA	
	(5) Priority-based slack-stealing	
	(6) Utilization updating	

Maximum Constant Speed

- The lowest possible clock speed that guarantees the feasible schedule of a task set
 - EDF scheduling
 - If the worst case processor utilization U of a given task set is lower than 1.0 under the maximum speed f_{\max} , the task set can be scheduled with a new maximum speed

$$U = \sum_{i=1}^n \frac{c_i}{p_i} \quad f_{MCS} = U \cdot f_{\max}$$

- Rate Monotonic scheduling

$$L_i(t) = \frac{\sum_{j=1}^i \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \quad f_{MCS} = f_{\max} \times \max_i \{L_i(t) \mid 1 < i \leq n, 0 < t < d_i\}$$

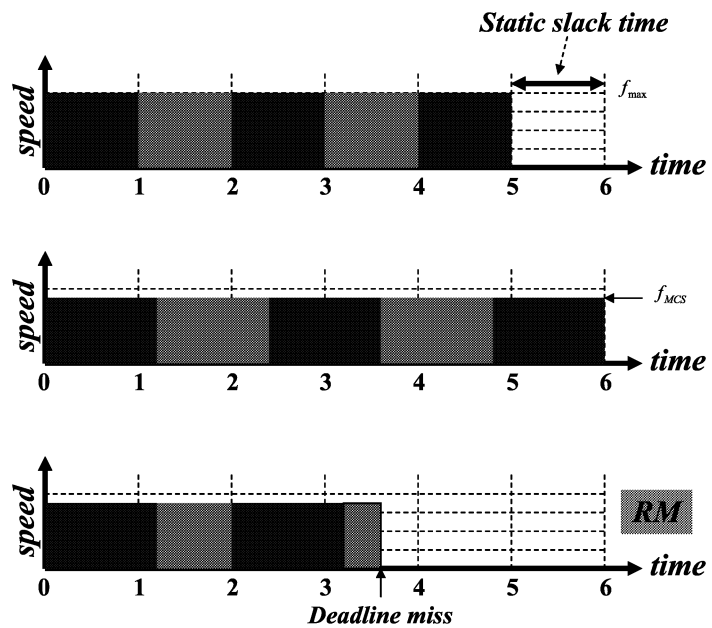
Maximum constant speed

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1

$$U = \frac{1}{2} + \frac{1}{3} = 0.833$$

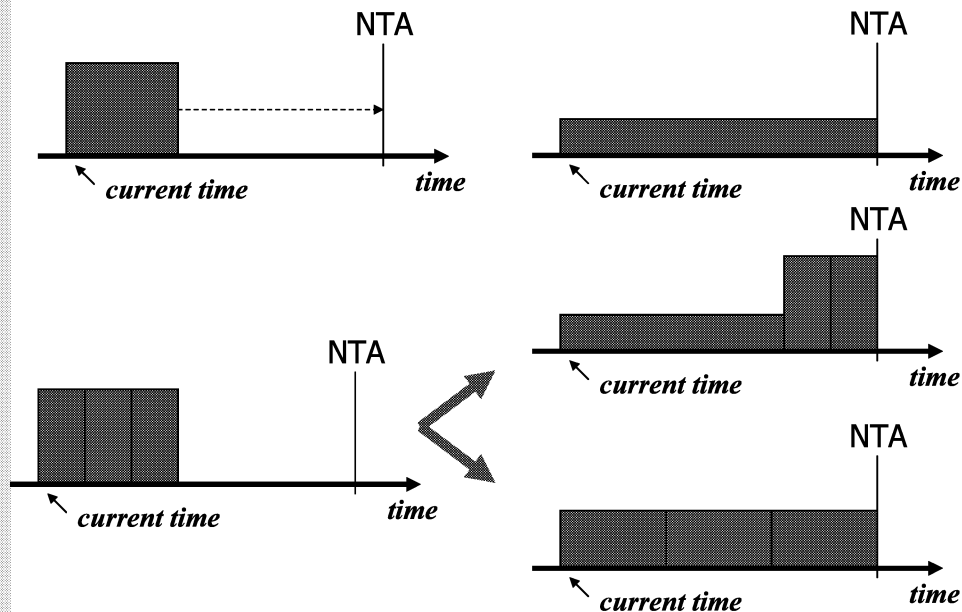
$$f_{MCS} = 0.833 \cdot f_{max}$$



Stretching to NTA

- Even though a given task set is scheduled with the maximum constant speed, since the actual execution times of tasks are usually much less than their WCETs, the tasks usually have dynamic slack times
- For the task τ which is scheduled at time t
 - If the next task is later than $t + WCET(\tau)$
 - We can slow down the execution of τ so that its execution completes exactly at this next task arrival time (NTA)

Example



ESSES 2003
2003/7/22 (Jihong Kim)

123

Priority-Based Slack Stealing

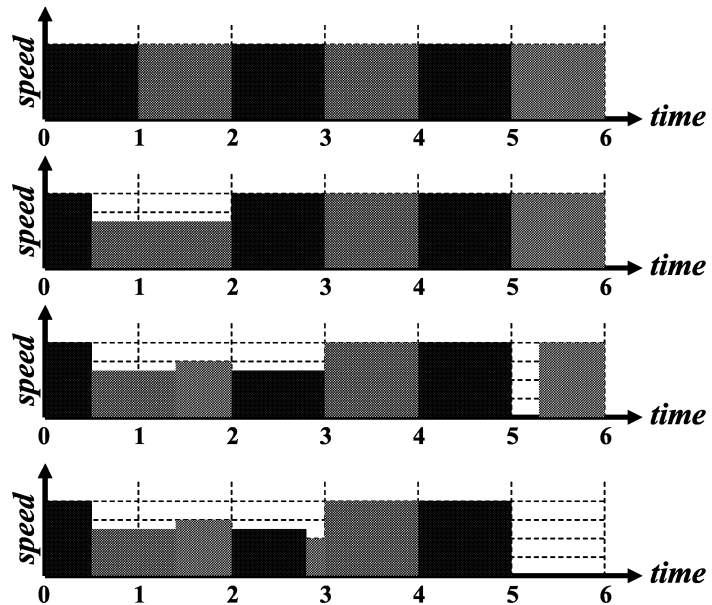
- Exploits basic properties of priority-driven scheduling such as EDF and RM
 - When a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task
- Advantage
 - Most task instances in a hyper-period may have chances to utilize dynamic slack times
 - Because most task executions complete earlier than WCETs
 - Therefore, many task instances can be scheduled with lowered voltages and speeds

ESSES 2003
2003/7/22 (Jihong Kim)

124

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1
Task 3	6	1



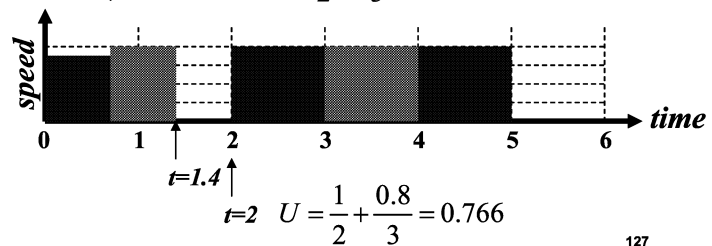
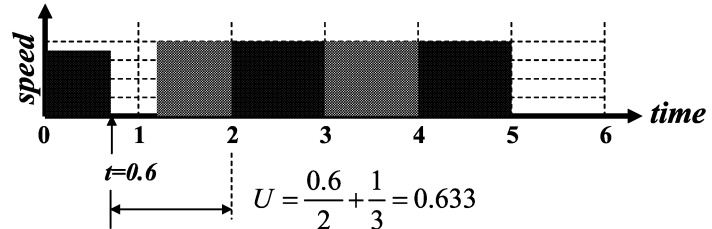
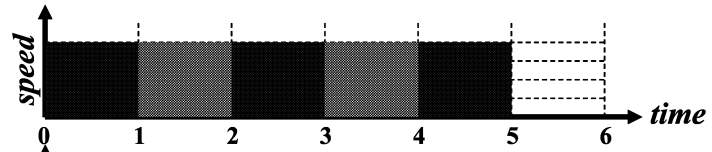
Utilization Updating

- The actual processor utilization during run time is usually lower than the worst case processor utilization
- This method is to estimate the required processor performance at the current scheduling point
 - By recalculating the expected worst case processor utilization
 - Using the actual execution times of completed task instances

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1

$$U = \frac{1}{2} + \frac{1}{3} = 0.833$$



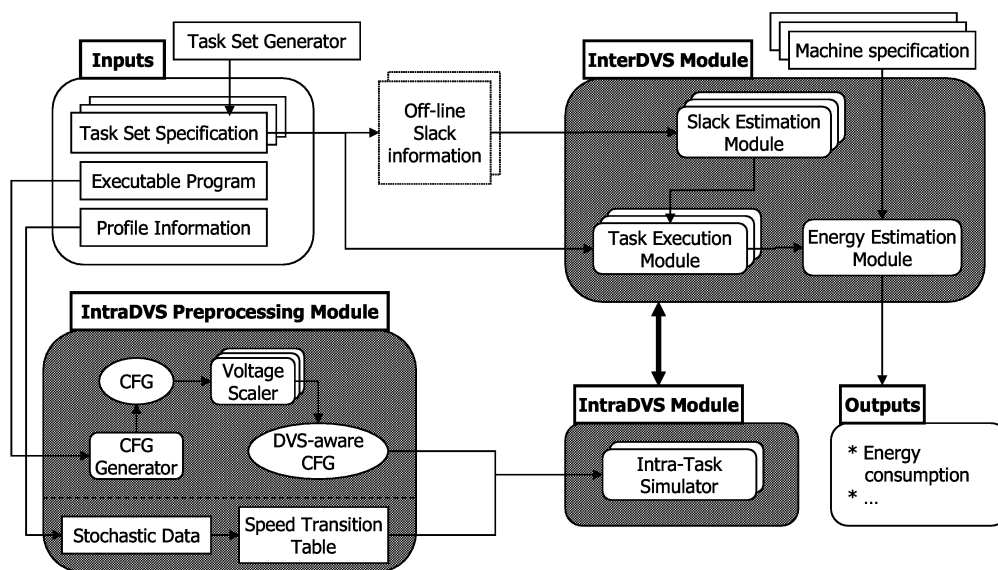
Slack Distribution Method

- Greedy approach
 - All the slack times are given to the next activated task
 - Most inter-task DVS algorithms have adopted it
 - Clearly, this approach is not an optimal solution, but is widely used because of its simplicity

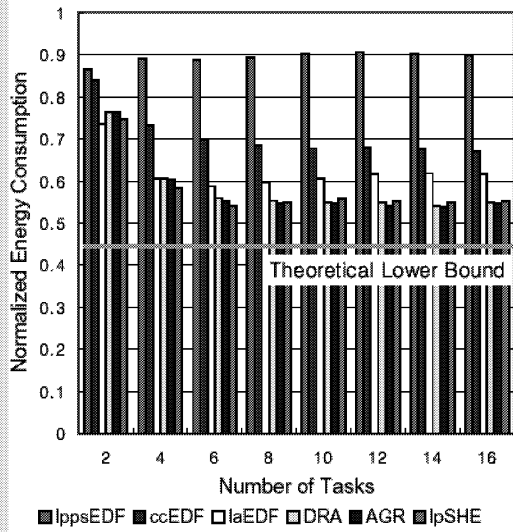
Existing Inter-Task DVS Algorithms

Category	Scheduling Policy	DVS Policy	Used Method
Inter-task DVS	EDF	lppsEDF	(3)+(4)
		ccEDF	(6)
		laEDF	(6)*
		DRA	(3)+(4)+(5)
		AGR	(4)*+(5)
		lpSEH	(3)+(4)+(5)*
	RM	lppsRM	(3)+(4)
	ccRM	(4)*	
Intra-task DVS	Path-based method	intraShin	(1)
	Stochastic method	intraGruian	(2)

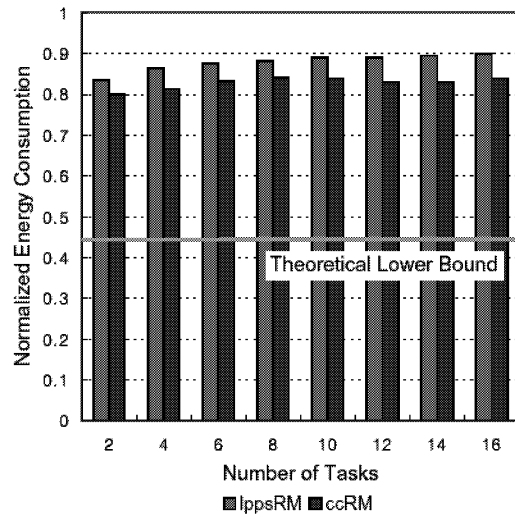
SimDVS: A Unified DVS Evaluation Environment



Experimental Results

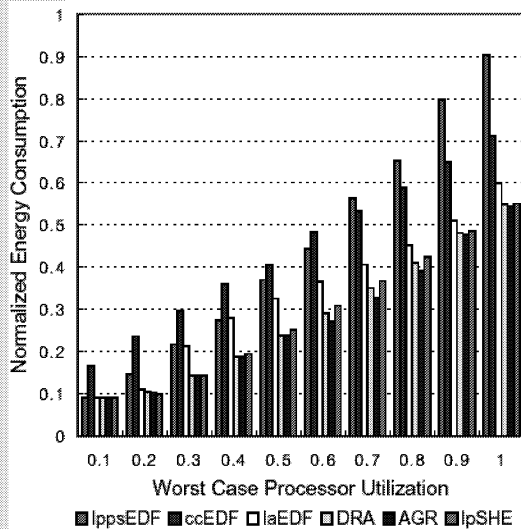


(a) DVS for EDF

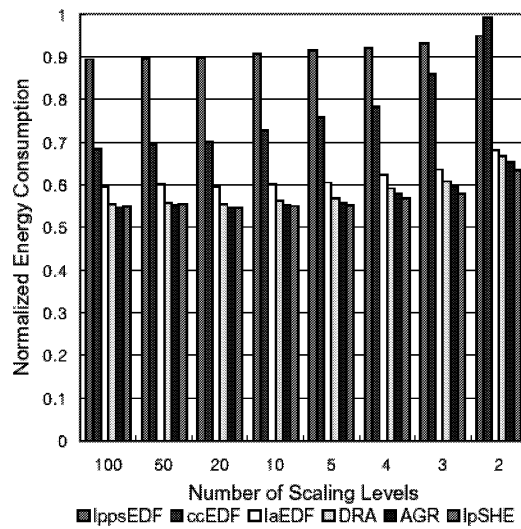


(b) DVS for RM

Experimental Results

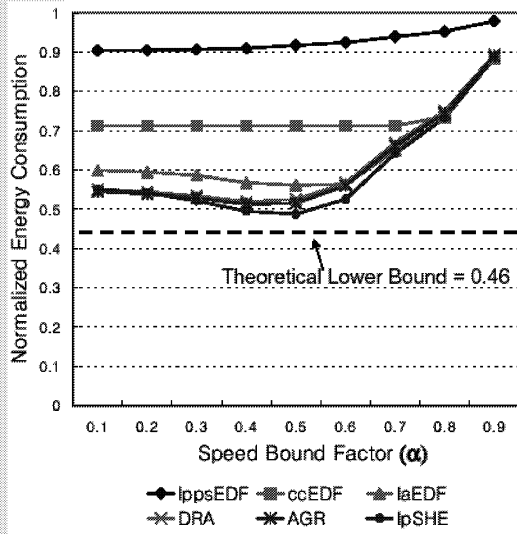


(a) WCPU

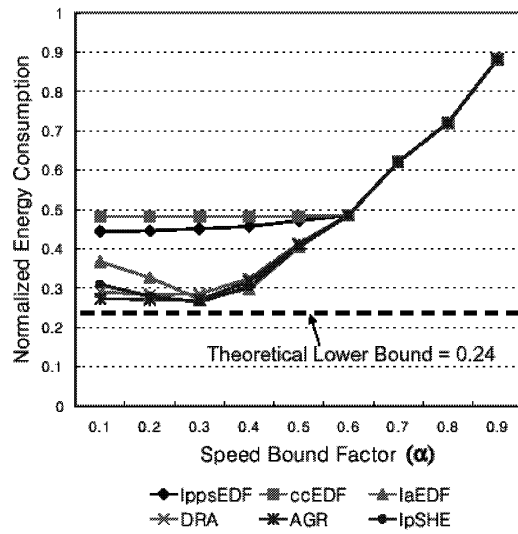


(b) Machine Specification

Experimental Results

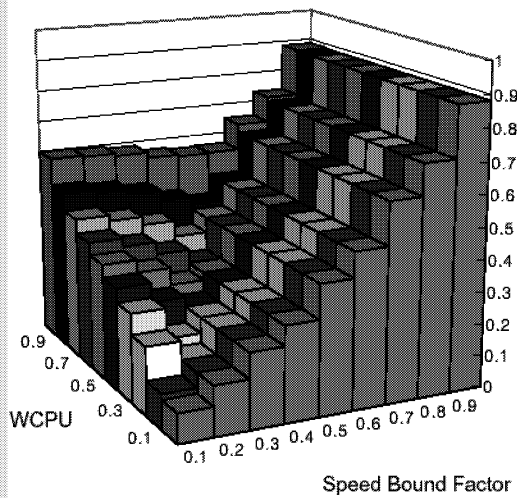


(a) WCPU=1.0 & ACPU=0.55

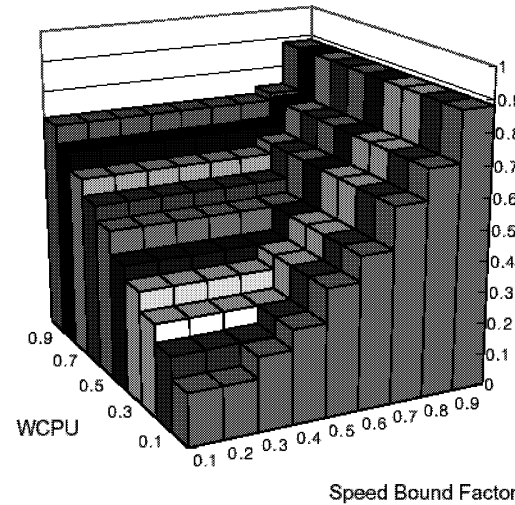


(b) WCPU=0.6 & ACPU=0.33

Experimental Results

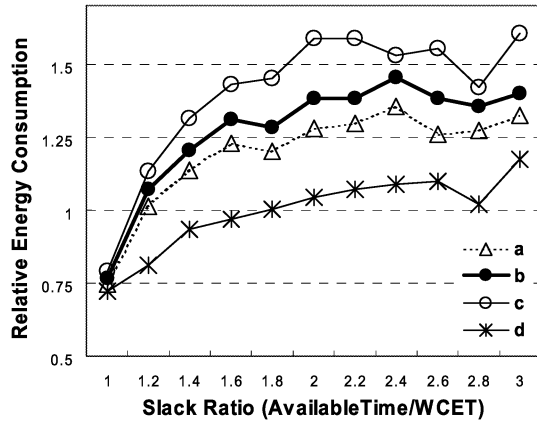


(a) laEDF

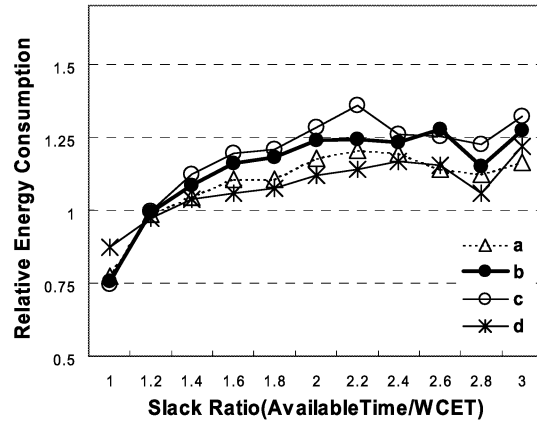


(b) ccEDF

Experimental Results (IntraDVS)



MPEG4 Decoder



MPEG4 Encoder

Performance Evaluation DVS Algorithms for Hard Real-Time Systems Using DEW

DEW - DVS Evaluation Workbench

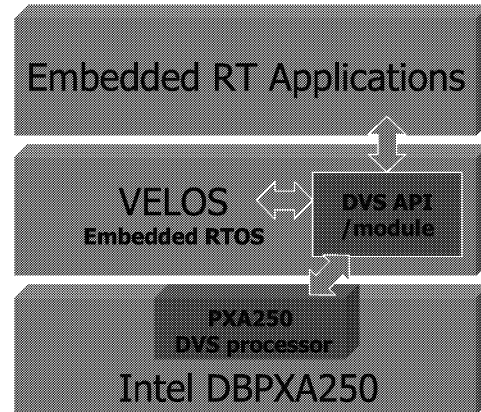
- **XScale-based DVS evaluation environment**

- **Pros**

- Allows to monitor real system behaviors under DVS

- **Cons**

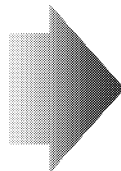
- **Slower than software simulation**
 - Because DEW runs actual applications
- **Less flexible for experimental studies**
 - Because DEW represents a single machine specification



Applications

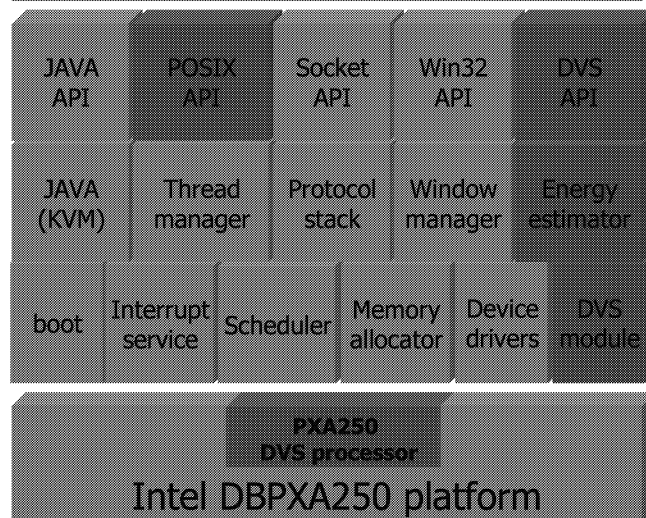


VELOS
Library
Kernel

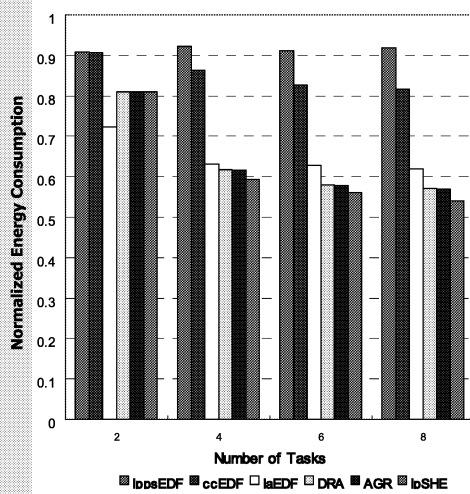


Libraries

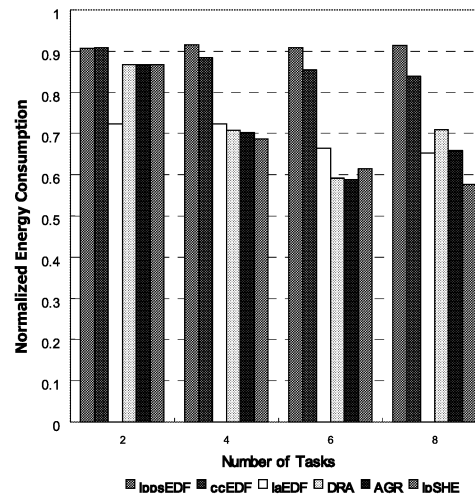
Embedded RT Applications



Evaluation Results Using SimDVS and DEW



SimDVS



DEW

ESSES 2003
2003/7/22 (Jihong Kim)

139

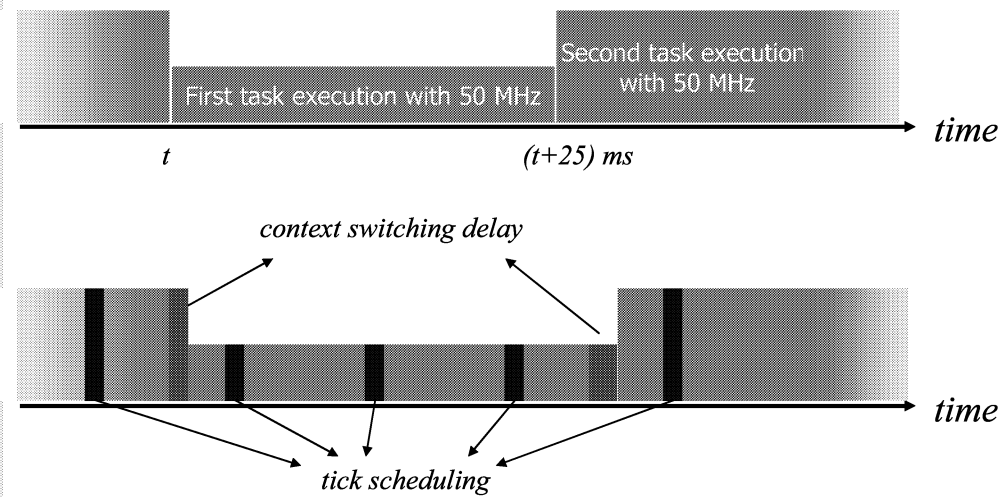
Sources of Differences

- **Impacts of**
 - **System overhead**
 - **Basic** : context switching overhead and tick scheduler overhead
 - **DVS** : slack computation and clock/voltage scaling
 - **System timing resolution**
 - **Simulator** : continuous time model
 - **Real system** : discrete time model
 - **Memory behavior**
 - **Changes in cache and memory access behavior**
 - **Data/Instruction fetch latency**

ESSES 2003
2003/7/22 (Jihong Kim)

140

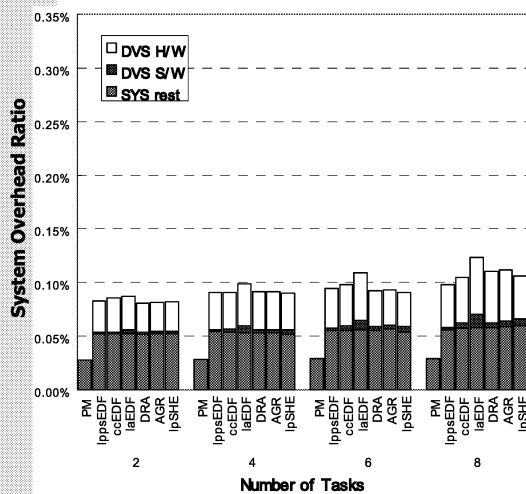
Example of System Overheads



ESSES 2003
2003/7/22 (Jihong Kim)

141

System Overhead



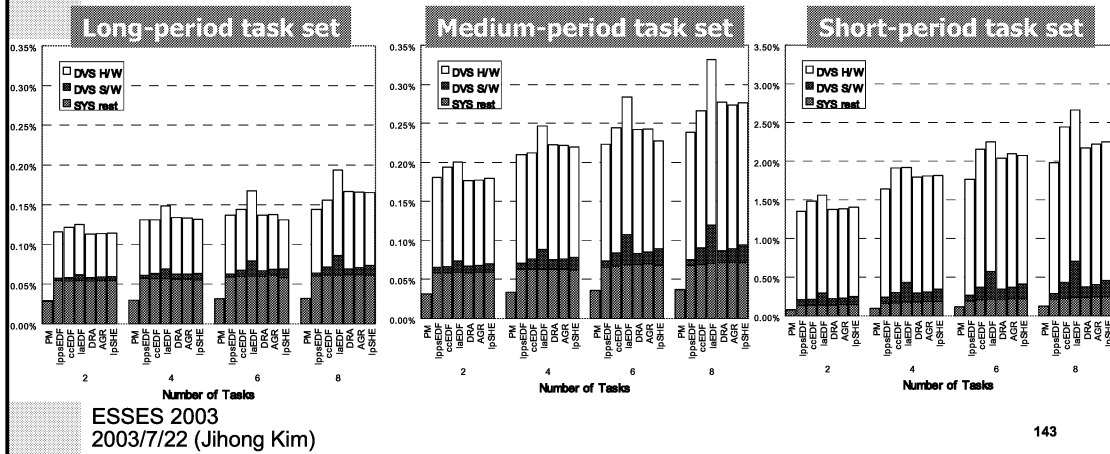
- DVS H/W
The ratio of time delay caused by the clock/voltage scaling hardware
- DVS S/W
The ratio of time delay caused by the slack computation in a DVS algorithm
- SYS rest
The ratio of the rest of the system overhead such as context switching and timer service

ESSES 2003
2003/7/22 (Jihong Kim)

142

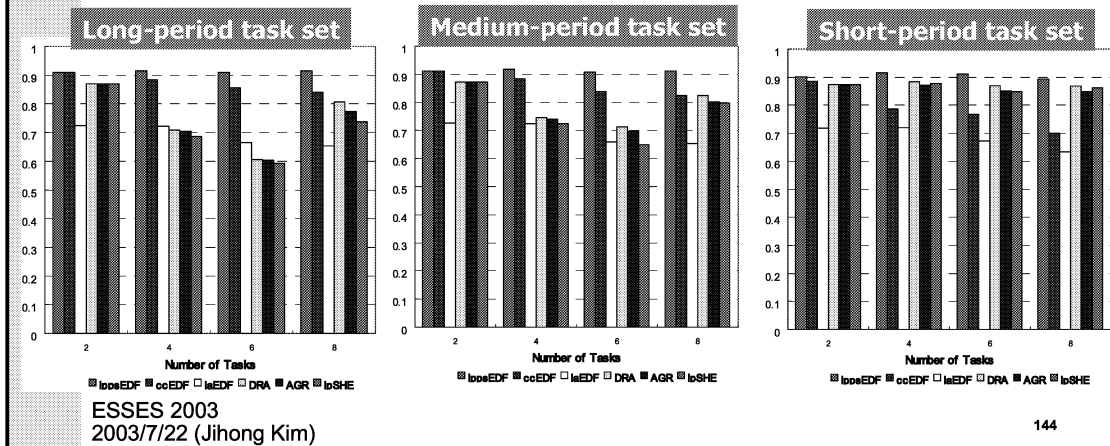
System Overhead Variations

- The system overhead increases very quickly as the task execution frequency increases
 - In particular, DVS parts increase quickly

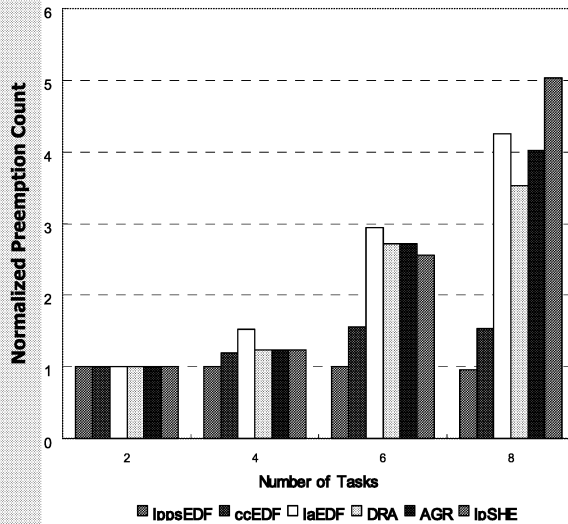


Energy Efficiency Variations

- In DRA, AGR, and IpSHE, the increased system overhead (due to the increased execution frequency) significantly affect the energy efficiency

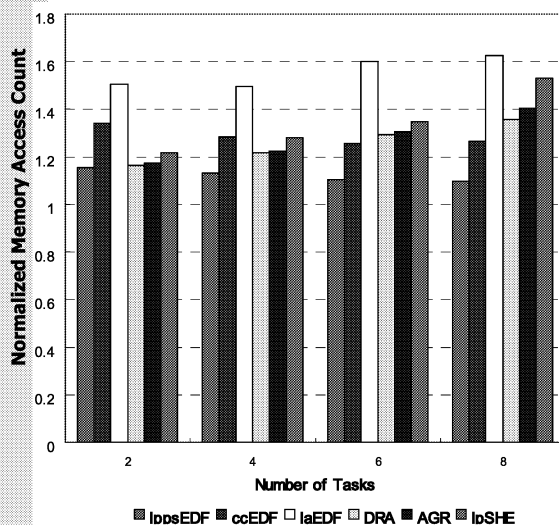


Changes in Memory System



- Under a DVS-enabled RTOS, Task's execution time increases due to the lowered clock speed
 - Desirable for reducing energy consumption
 - But, it can introduce negative side effects as well
 - An increase in the number of task preemptions which increases the number of memory accesses
- In aggressive algorithms, the number of preemptions increases more rapidly than the others

Changes in Memory System



- PXA250
 - Performance Monitoring Unit
 - 32-way set-associative cache of Inst/Data cache
- Each application
 - 16-KB program code
- The increases in memory accesses can be attributed to two sources
 - The increase in the number of preemptions
 - The increase in memory accesses from the algorithm itself

References

- **Transmeta Corporation. Crusoe Processor.** <http://www.transmeta.com>, June 2000.
- **AMD Corporation. PowerNow! Technology.** <http://www.amd.com>, December 2000.
- **Intel Corporation. Intel XScale Technology.** <http://developer.intel.com/design/intelxscale/>, November 2001.
- **I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor.** In Proceedings of the IEEE Real-Time Systems Symposium, pages 178-187, December 1998.
- **Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors.** In Proceedings of the International Conference on Computer-Aided Design, pages 365-368, November 2000.
- **H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems.** In Proceedings of IEEE Real-Time Systems Symposium, December 2001.

References

- **P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems.** In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), October 2001.
- **D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications.** IEEE Design and Test of Computers, 18(2):20-30, March 2001.
- **F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors.** In Proceedings of the International Symposium on Low Power Electronics and Design, pages 46-51, August 2001.
- **W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis.** To appear in Proceedings of Design, Automation and Test in Europe (DATE'02), March 2002.

References

- **D. Grunwald, P. Levis, and K. I. Farkas. Policies for Dynamic Clock Scheduling. In Proceedings of the 4th Symposium on Operating Systems Design and Implementation, pages 73-86, October 2000.**
- **S. Lee and T. Sakurai. Run-time Voltage Hopping for Low-power Real-Time Systems. In Proceedings of the 37th Design Automation Conference, pages 806-809, June 2000.**
- **T. Burd and R. Brodersen. Design Issues for Dynamic voltage scaling . In Proceedings of the International Symposium on Low Power Electronics and Design, pages 9-14, July 2000.**
- **D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. Technical Report 1342, University of Wisconsin-Madison, CS Department, June 1997.**
- **F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In Proceedings of the IEEE Foundations of Computer Science, pages 374-382, 1995.**

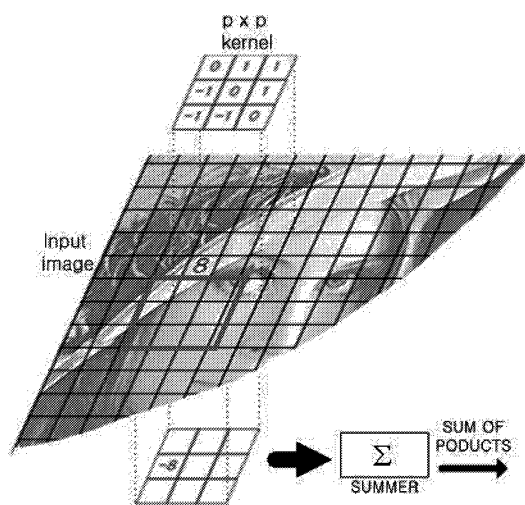
DVS-Aware Algorithm Development

Image Convolution



- One of the fundamental operations of image processing.
- DVS Unfriendly!!

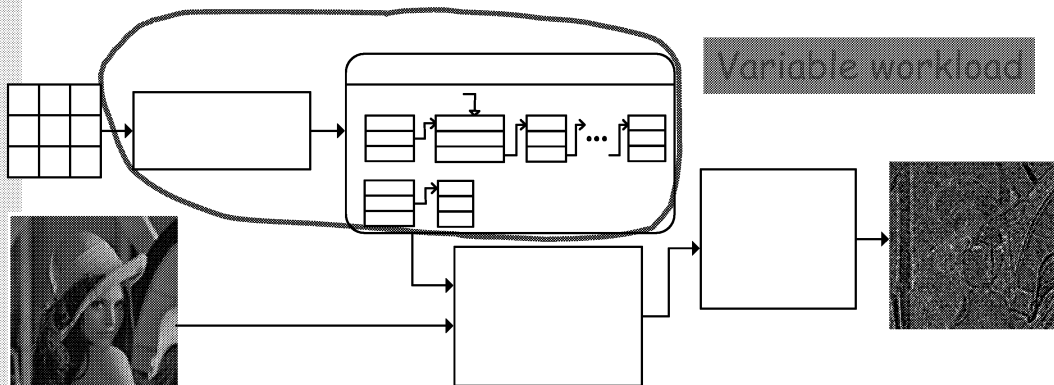
Direct Implementation:



- p^2 multiplications
 p^2 additions for each convolved element.

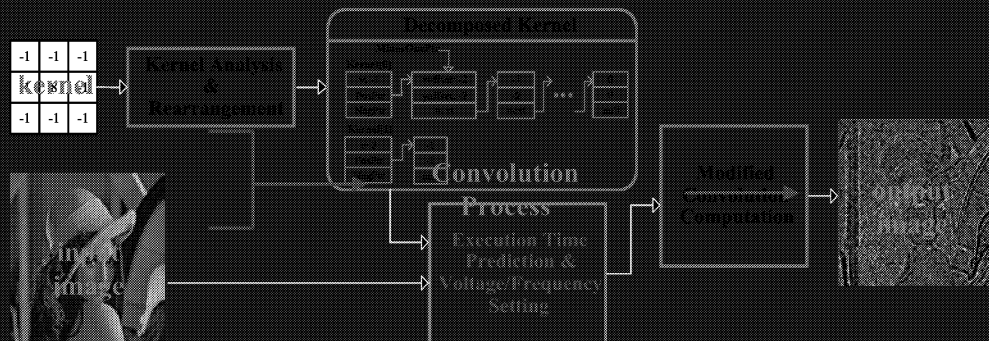
No Workload Variations!

Low-Power Implementation



Variable Workload Algorithm
Based on Kernel Characteristics

Low-Power Implementation



Overall processing step

Kernel Analysis and Rearrangement

- **Property 1.** For most kernels, the number of distinct kernel elements is small.
- **Property 2.** 0, 1 and -1 are used frequently.
- **Property 3.** Many kernel elements have the same absolute values.

0	1	1
-1	0	1
-1	-1	0

-1	-2	-1
0	0	0
1	2	1

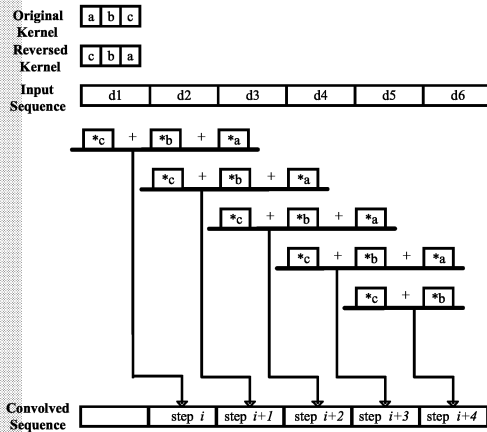
Modified Convolution Algorithm: SDM

- For 1 or -1, no multiplication.
- For 0, no addition & no multiplication.
- For the same absolute values, a single multiplication.

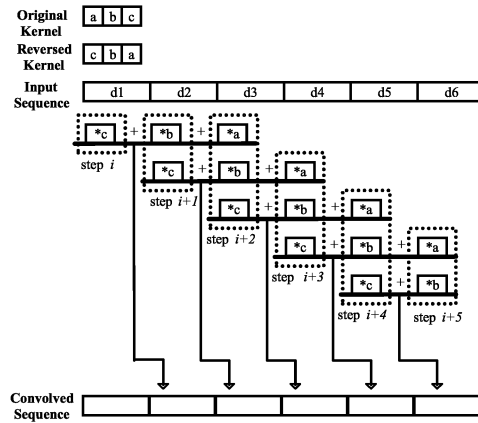
0	1	1
-1	0	1
-1	-1	0

	Direct	SDM
Number of Operations/pixel	9 additions & 9 multiplications	6 additions & 0 multiplications

Direct vs. SDMK



Direct implementation



SDMK implementation

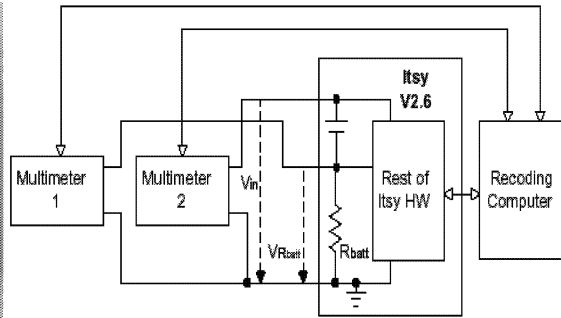
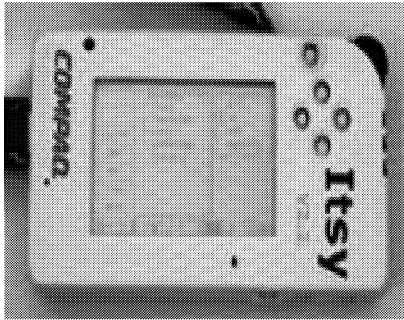
Exec Time Prediction & Speed Setting

- **By a static method**
 - Based on the number of required arithmetic operations

- **By a dynamic-method**
 - Based on actual measurements of execution times
 - In the direct algorithm, by pre-constructed speed table

Experimental Environments

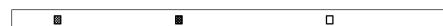
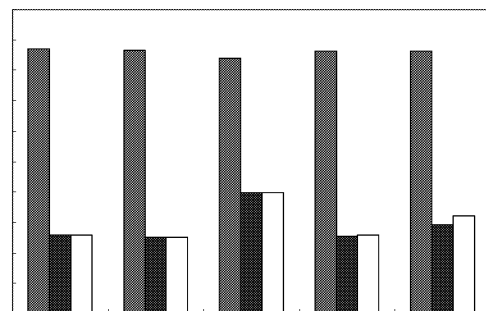
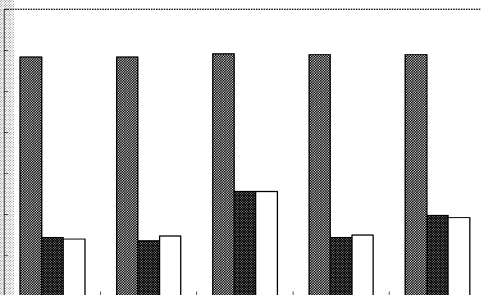
- Itsy Pocket Computer V2.6
 - CPU : Intel StrongARM 1110
 - Frequency scaling: 11 levels (59.0 MHz ~ 226.4 MHz)
 - Voltage scaling: 30 levels (1.00 V ~ 2.00 V)
- **Linux operating system (ver. 2.0.30)**



ESSES 2003
2003/7/22 (Jihong Kim)

159

Results (Energy Dissipation)

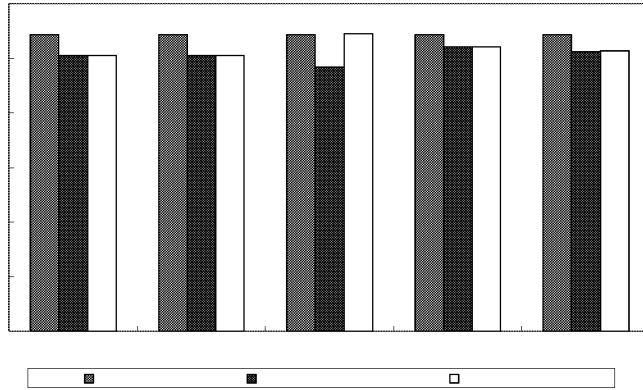


- **Average 67.6% energy saving in the core processor, and 62.8% in the whole Itsy system.**

ESSES 2003
2003/7/22 (Jihong Kim)

160

Results (Execution Time)



- **There is no performance degradation over the direct approach.**

Conclusions

- **Presented a low-power implementation of image convolution algorithm for variable voltage processors.**
- **The energy efficiency of the proposed implementation comes from:**
 - **Smaller N_{cycle}**
 - **Lower V_{dd}**
 - **Fewer memory references**
 - **i.e., less energy consumed in non-CPU components**

$$E \propto N_{\text{cycle}} \cdot V_{\text{DD}}^2$$

Execution Time (us)

Energy-Optimal Off-Line Voltage Scheduling

Off-Line Volt. Sched. Problem

- Voltage schedule (speed schedule) : $S(t)$
 - the processor speed as a function of time
 - The energy consumption under $S(t)$ is given by
$$E(S) = \int_{\text{interval}} P(S(t)) dt$$
 - P is a convex function from speed to power
 - Given N jobs J_1, J_2, \dots, J_N where
 - r_i : the release time of J_i
 - d_i : the deadline of J_i
 - c_i : the workload (# of execution cycles) of J_i
 - assumed to be known a priori
 - p_i : the priority of J_i
- compute a feasible voltage schedule $S(t)$ that minimizes $E(S)$
- $S(t)$ is feasible iff $S(t)$ gives J_i its workload c_i between r_i and d_i for all J_1, J_2, \dots, J_N

Existing Works for the Problem

- Note that the system model covers
 - Fixed-priority (RM, DM) periodic/aperiodic task set
 - EDF periodic/aperiodic task set
 - $p_i < p_j$ iff $d_i < d_j$
- For EDF job sets (a special case), the problem can be solved in poly. time by Yao's algo.[FOCS'95]
 - solution space = convex , obj. func. = convex
- For general job sets, the problem becomes much difficult
 - main source of difficulty : feasibility condition
 - Quan & Hu [TCAD'03]: exhaustive optimal algo.
 - Yun & Kim [TECS'03]: NP-hardness & FPTAS

References

- Optimal algorithm for EDF job sets
 - F. Yao, A. Demers, S. Shenker, "A Scheduling Model for Reduced CPU Energy", In Proc. Foundations of Computer Sciences (FOCS'95), 1995
- Heuristic for FP job sets
 - G. Quan and X. Hu, "Energy Efficient Scheduling for Real-Time Systems On Variable Voltage Processor", In Proc. Design Automation Conference (DAC'01), 2001.
- Exhaustive optimal algorithm for FP job sets
 - G. Quan and X. Hu, "Minimum Energy Fixed Priority Scheduling for Variable Voltage Processors", IEEE Transactions on Computer Aided Design and Systems, 2003.
- NP-hardness proof & FPTAS for FP job sets
 - H.-S. Yun and J. Kim, "On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems", ACM Transactions on Embedded Computing Systems, 2003.

A Profile-Based Energy-Efficient Intra-Task Voltage Scheduling Algorithm for Hard Real-Time Applications*

Dongkun Shin
School of Computer Science and Engineering
Seoul National University
sdk@davinci.snu.ac.kr

Jihong Kim
School of Computer Science and Engineering
Seoul National University
jihong@davinci.snu.ac.kr

ABSTRACT

Intra-task voltage scheduling (IntraVS), which adjusts the supply voltage within an individual task boundary, is an effective technique for developing low-power applications. In this paper, we propose a novel intra-task voltage scheduling algorithm for hard real-time applications based on average-case execution information. Unlike the original IntraVS algorithm where voltage scaling decisions are based on the worst-case execution cycles, the proposed algorithm improves the energy efficiency by controlling the execution speed based on average-case execution cycles while still meeting the real-time constraints. The experimental results using an MPEG-4 decoder program show that the proposed algorithm reduces the energy consumption by up to 34% over the original IntraVS algorithm.

1. INTRODUCTION

Since energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage V_{DD} , lowering the supply voltage V_{DD} is the most effective way of reducing energy consumption. However, lowering the supply voltage also decreases the clock speed, since the CMOS circuit delay T_D is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^\alpha$ [6], where V_T is a threshold voltage, and α is a velocity saturation index. This trade-off introduced various dynamic voltage scaling (DVS) techniques. DVS techniques change the clock speed and its corresponding supply voltage dynamically to the lowest possible level while meeting the task's performance constraint.

1.1 Dynamic Voltage Scaling

For hard real-time systems, there exist two DVS approaches depending on the scaling granularity. *Inter-task voltage scheduling* (InterVS) [9, 2, 8, 5] determines the supply voltage on task-by-task basis, while *intra-task voltage scheduling* (IntraVS) [4, 7] adjusts the supply voltage within an individual task boundary. Both approaches can guarantee the required performance constraints of real-time systems.

*This work is supported by the Ministry of Information & Communication of Korea (Support Project of University foundation research<'00> supervised by IITA).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.
Copyright 2001 ACM 1-58113-371-5/01/0008 ...\$5.00.

Intra-task voltage scheduling [4, 7] has been proposed as a solution to overcome the limitations of inter-task voltage scheduling. IntraVS algorithms exploit all the slack time from run-time variations of different execution paths; there is no slack time when the scheduled program completes its execution, thus significantly improving energy efficiency. Furthermore, since IntraVS does not involve OS in adjusting the clock speed, it can be used with an existing OS without any modifications on a variable voltage processor.

We propose an energy-efficient IntraVS algorithm for hard real-time applications based on *average-case* execution information. Unlike the original IntraVS algorithm [7] where voltage scaling decisions are based on *worst-case execution cycles*, the proposed algorithm controls the execution speed based on the *average-case* execution paths (ACEPs), which are the most frequently executed paths. Since the proposed algorithm is optimized for the energy reduction in the ACEP(s), which are the most likely path(s) that will be executed at run time, the proposed algorithm is more effective than the original intraVS algorithm [7] in reducing the energy consumption. The novel aspect of the proposed algorithm is that the timing constraints of a hard real-time program is still satisfied, even if the ACEP(s) are used for voltage scaling decisions.

2. ORIGINAL INTRA-TASK VOLTAGE SCHEDULING ALGORITHM

For a hard real-time task, the goal of an intra-task voltage scheduling algorithm is to assign a proper clock speed to each basic block so that energy consumption is minimized while satisfying timing requirements. In this section, we briefly describe the original intra-task voltage scheduling algorithm [7] as a short introduction to intra-task voltage scheduling.

Throughout this paper, we assume the following about the target variable voltage processor: The processor provides a special instruction, `change_f_v(fCLK)`, which changes the current clock frequency to f_{CLK} and adjusts the supply voltage to the corresponding voltage V_{DD} . f_{CLK} and V_{DD} can be set continuously within the operational range of the processor. When the processor changes the clock/voltage, there is a clock/voltage transition overhead. During clock/voltage transition, the processor stops running.

Consider a hard real-time program P with the deadline of $2 \mu\text{sec}$ shown in Fig. 1(a). The CFG G_P of the program P is shown in Fig. 1(b). In G_P , each node represents a basic block of P and each edge indicates the control dependency between basic blocks. The number within each node indicates $C_{EC}(b_i)$ which is the number of execution cycles of the corresponding basic block. The back edge from b_5 to b_{wh} models the **while** loop of the program P .

Using a WCET analysis tool, we can find the path $p_{worst} = (b_1, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_{if}, b_6, b_7)$ as the worst case

execution path (WCEP) for the example program P , assuming that the maximum number of **while** loop iterations is set to 3 by user. The predicted execution cycles of p_{worst} is, therefore, 160 cycles, which is the worst case execution cycles (WCEC) of program P . If a target processor operates at the maximal 80-MHz clock frequency, the program P completes its execution in $2 \mu\text{sec}$, resulting in no slack time.

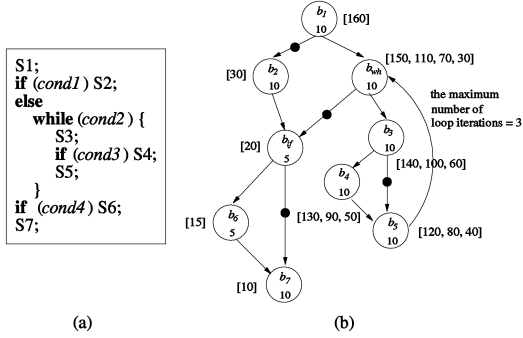


Figure 1: An example program; (a) an example real-time program P and (b) its CFG G_P .

The key observation behind the IntraVS approach is that there are large execution time variations among different execution paths. For short execution paths, if we were able to identify them in the early phase of its execution, we can lower the clock speed substantially, thus saving a significant amount of energy consumption.

For the speed adjustment, intra-task voltage scheduling technique uses an adaptive approach with the help of a static program analysis technique on worst case execution times. Assume that $CRWEC(b_i)$ represents the remaining worst case execution cycles (RWEC) among all the execution paths that start from b_i . Using a modified WCET analysis tool, for each basic block b_i , we can compute $CRWEC(b_i)$ in *compile time*. In Fig. 1(b), the symbol $[]$ contains the $CRWEC(b_i)$ values of each basic block. For the basic blocks related to the **while** loop (i.e., b_{wh} , b_3 , b_4 , b_5), the corresponding nodes are associated with multiple $CRWEC(b_i)$ values, reflecting the maximum three iterations of the **while** loop.

With the $CRWEC(b_i)$ values computed, we can statically identify an edge (b_i, b_j) (of a CFG G) where $[CRWEC(b_i) - C_{EC}(b_i)] \neq CRWEC(b_j)$. For example, in Fig. 1(b), we can identify four such edges, i.e., (b_1, b_2) , (b_{wh}, b_{if}) , (b_{if}, b_7) and (b_3, b_5) , which are marked by the symbol \bullet . These marked edges form a set of candidate Voltage Scaling Edges (VSEs). If an edge (b_i, b_j) is selected as a VSE, it means that the clock speed will change when the thread of execution control branches to b_j from b_i . For example, the clock speed will be lowered when the basic block b_2 is executed after b_1 because the remaining work is reduced by 1/5 (i.e., the ratio of $CRWEC(b_2)$ to $[CRWEC(b_1) - C_{EC}(b_1)]$).

At the selected VSEs, the new clock speed is determined by how much the remaining work is reduced. For example, when the thread of execution control meets a VSE (b_i, b_j) , the clock speed can be lowered because the remaining work is reduced by $[R - CRWEC(b_j)]$ where $R = CRWEC(b_i) - C_{EC}(b_i)$. After b_i is executed at the clock speed S , the clock speed can be changed to reflect the reduction in the remaining work. The new clock speed for b_j is set to $S \times \frac{CRWEC(b_j)}{R}$. We call $\frac{CRWEC(b_j)}{R}$ as the speed update ratio (SUR) for the edge $b_i \rightarrow b_j$, denoted by $SUR(b_i \rightarrow b_j)$.

By the original IntraVS algorithm, the clock speed is changed from 80 MHz to 16 MHz ($= 80 \text{ MHz} \times \frac{30}{160-10}$) at the edge (b_1, b_2) . Assuming that no energy is consumed in an idle state and $E \propto C_L \cdot$

$N_{cycle} \cdot V_{DD}^2$, when the execution follows the path $p_1 = (b_1, b_2, b_{if}, b_6, b_7)$, the original IntraVS algorithm reduces the energy consumption by 69%.

Since there exists the transition overhead during speed changes, not all the candidate VSEs are selected as VSEs. A candidate VSE is selected as a VSE when the number of saved cycles at the candidate VSE is larger than a given threshold value. The threshold value is determined by a VSE selection policy, which is a function of the transition time overhead, the transition power overhead, and the code size increase (by the added scaling code).

3. PROFILE-BASED INTRA-TASK VOLTAGE SCHEDULING

3.1 Motivation

Before the profile-based IntraVS algorithm is presented, we first generalize the original IntraVS algorithm described in Section 2. In order to adjust the clock speed at VSEs, IntraVS first selects a (predicted) reference execution path such as the WCEP. Once the reference execution path is decided, IntraVS sets the initial operating voltage and its corresponding clock frequency assuming that the task execution will follow the predicted reference execution path.

When the actual execution deviates from the (predicted) reference execution path (say, by a branch instruction), the clock speed can be adjusted depending on the difference between the number of remaining execution cycles of the reference execution path and the number of remaining execution cycles of the newly deviated execution path. If the new execution path takes significantly longer to complete its execution than the reference execution path, the clock speed should be *raised* to meet the deadline constraint. On the other hand, if the new execution path can finish its execution earlier than the reference execution path, the clock speed can be *lowered* to save the energy consumption. Once the actual execution takes a different path from the reference path, a new reference path is constructed starting from the deviated basic block.

Using a static program-analysis technique, IntraVS identifies the appropriate program locations where the clock speed should be raised or lowered relative to the current clock speed. For the clock speed adjustment at run time, IntraVS algorithm inserts voltage scaling code to the selected program positions. The candidate positions for inserting voltage scaling code are the branching edges of the CFG, which correspond to the branch or loop statements.

We call the original IntraVS as the remaining worst-case execution path (RWEP)-based IntraVS, because the remaining worst-case execution path (RWEP) is used as the reference path. In the RWEP-based IntraVS, the clock speed is monotonically decreasing at all the VSEs. Depending on how the reference path is selected, however, the clock speed may be increased as well at some VSEs. Therefore, we divide VSEs into Up-VSEs and Down-VSEs. The clock speed is increased at an Up-VSE while the clock speed is decreased at a Down-VSE.

Although the RWEP-based IntraVS reduces the energy consumption significantly while guaranteeing the deadline, this is a pessimistic approach because it always predicts that the longest path will be executed. A more optimistic approach is to use the average case execution path (ACEP) as a reference path. The ACEP is defined to be an execution path that is most likely to be executed. The ACEP can be decided by the execution profile information.

The main motive of using the ACEP instead of the WCEP is to make the common case more energy-efficient. For a typical program, about 80 percent of the program's execution occurs in only 20 percent of its code, which is called the hot paths [1]. For an In-

traVS algorithm to be energy-efficient, it should be energy-efficient when the hot paths are executed. If we use one of hot paths as a reference path for intra-task voltage scheduling, the speed change graph for the hot paths will be a near flat curve with little changes in the clock speed, which gives the best energy efficiency under a given amount of work [3]. Even for the paths that are not the hot paths, if we take one of hot paths as a reference paths, they are more energy-efficient because they can start with a lower clock speed than when the WCEP is used as a reference path.

In the profile-based IntraVS, we take the ACEP, which is the best representative of the hot paths, as the reference path. We call such an IntraVS algorithm as the remaining average-case execution path (RAEP)-based IntraVS because the remaining average-case execution path (RAEP) is used as the reference path.

Figure 2 shows an RAEP-based CFG G_P^{RAEP} with $C_{RAEC}(b_i)$ values that represent the remaining average-case execution cycles among all the paths that start from b_i . The bold edges in G_P^{RAEP} means that it has a higher probability to be followed at run time between two branching edges. In Fig. 2, the initial reference path is $(b_1, b_{wh}, b_3, b_5, b_{wh}, b_3, b_5, b_{wh}, b_{if}, b_7)$. With the reference path, $C_{RAEC}(b_i)$ is computed. For example, $C_{RAEC}(b_{if}) = C_{EC}(b_{if}) + C_{RAEC}(b_7)$. At the RAEP-based IntraVS, there are Up-VSEs (marked by \circ in Fig. 2) as well as Down-VSEs (marked by \bullet in Fig. 2). Figure 3 shows how the speed and voltage change by the RAEP-based scheduling. The speed is changed from 14 MHz to 21 MHz at the edge (b_{if}, b_6) because this is an Up-VSE with the SUR value of $1.5 (= \frac{15}{10})$. Compared to the energy consumption of the RWEP-based IntraVS algorithm, the RAEP-based IntraVS algorithm achieves 55% more energy reduction.

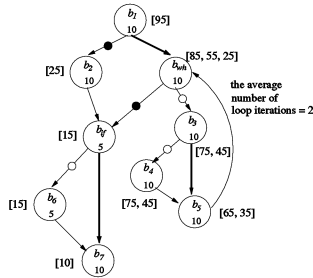


Figure 2: A RAEP-based CFG G_P^{RAEP} .

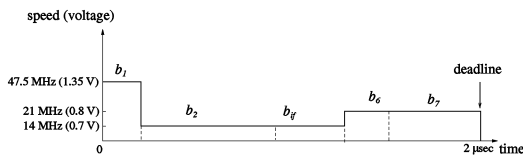


Figure 3: Speed and voltage changes by the RAEP-based IntraVS.

Though the RAEP-based scheduling is more effective in reducing the energy consumption than the RWEP-based scheduling, the pure RAEP-based approach cannot meet the timing requirements of hard real-time applications. This is because it does not satisfy the timing constraint for all the execution paths. For example, consider the case when the WCEP and ACEP take significantly different number of execution cycles. When the execution takes the WCEP at the middle of program execution, it is possible that the program cannot meet its deadline even if the remaining path executes with the maximum clock speed. The next section describes a novel ap-

proach that is still based on the RAEC but can guarantee the timing constraint for all the execution paths.

3.2 Reference Path Modification

To overcome the deadline miss problem of the pure RAEP-based IntraVS algorithm, we modify the reference path whenever the deadline miss situations are identified. Assume that the reference path is $p_{ref} = (b_1, \dots, b_i, b_{i+1}, \dots, b_N)$, b_i is a branching node whose children basic blocks are b_{i+1} and b_{miss} , and the current clock speed at b_i is S . If the clock speed at b_{miss} , given by $S \times SUR(b_i \rightarrow b_{miss})$, is larger than the maximal clock speed ($MaxS$) of the processor, it indicates that if the current execution branches to b_{miss} , the deadline will be missed. This is because the remaining time T_R to the deadline is $T_R = \frac{C_{RAEC}(b_{i+1})}{S}$ and $MaxS \times T_R < C_{RAEC}(b_{miss})$. There are $M = \lceil C_{RAEC}(b_{miss}) - MaxS \times T_R \rceil$ cycles that miss the deadline. In order to avoid the deadline miss, we increment $C_{RAEC}(b_k)$ by M for all $k \leq i$. That is, we modify the reference path by adding a new virtual basic block b_v between b_i and b_{i+1} . $C_{EC}(b_v)$ is set to M . The virtual basic block is used only to prevent the deadline miss during the speed assignment and not executed at run time.

Figure 4 illustrates how the reference path modification works. Given an original G_P^{RAEP} , the ACEP, (b_1, b_3, b_4) , is used as the reference path. (The bold edges indicate higher probability edges to be selected at run time.) With the 100-MHz maximal clock frequency, the path (b_1, b_3, b_5) misses the 0.5- μ sec deadline, because the speed at (b_3, b_5) should be raised to 120 MHz (i.e., $60 \text{ MHz} \times 2$). Because $\frac{10}{3}$ cycles¹ are missed from the deadline, we add a virtual block b_v between b_3 and b_4 , as shown in Fig. 4(b). $C_{EC}(b_v)$ is set to 4 ($= \lceil \frac{10}{3} \rceil$).

With the added b_v , $C_{RAEC}(b_1)$ and $C_{RAEC}(b_3)$ are modified to 34 and 24, respectively, and the speed update ratios are recalculated. For example, the SUR at (b_3, b_5) is modified to $1.43 (= \frac{20}{14})$ from 2.

Figures 4(c) and 4(d) compare the speed changes for the RWEP-based IntraVS, the RAEP-based IntraVS and the modified RAEP-based IntraVS for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5) , respectively. The modified RAEP-based scheduling is more energy-efficient than the RWEP-based scheduling for the hot paths (Fig. 4(c)), which affect most on the overall energy efficiency. It also satisfies the deadline requirement (Fig. 4(d)), unlike the pure RAEP-based scheduling algorithm.

4. EXPERIMENTAL RESULTS

We have extended the existing voltage scaling tool, the Automatic Voltage Scaler (AVS) [7], to evaluate the energy efficiency of the proposed IntraVS over the original IntraVS. AVS takes as inputs an original *DVS-unaware* program P and its timing requirements, and produces a low-energy *DVS-aware* program P_{DVS} that satisfies the same timing requirements of P . The converted program P_{DVS} contains voltage scaling code that handles all the idiosyncrasy of scaling speed/voltage on a variable voltage processor. The extended AVS can convert a program using either the RWEP-based IntraVS or the RAEP-based IntraVS.

To evaluate the power reduction effect of the proposed extensions to the original IntraVS algorithm, we have experimented with an MPEG-4 video decoder using an energy simulator [7]. We assume that both DVS-aware and DVS-unaware systems enter into a power-down mode when the system is idle. The energy consumption of a power-down mode is assumed to be 0. The supply voltage for a given clock frequency is obtained from $f_{CLK} = 1/T_D \propto (V_{DD} - V_T)^\alpha / V_{DD}$ [6] where V_{DD} , V_T , and α are assumed to be 2.5V, 0.5V, and 1.3, respectively. For the RAEP-based IntraVS, the

¹20 cycles - 100 MHz \times $\frac{10 \text{ cycles}}{60 \text{ MHz}} = \frac{10}{3}$ cycles

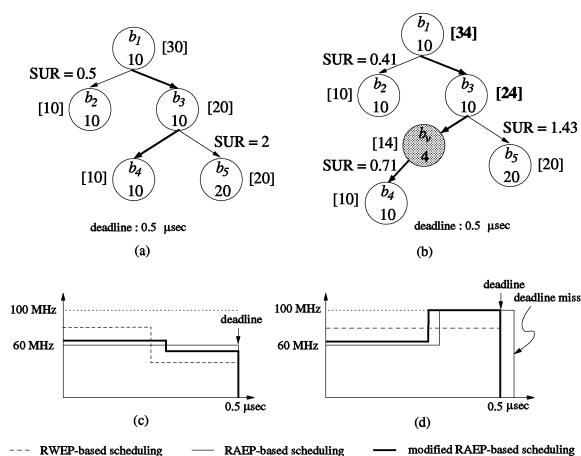


Figure 4: Modified RAEP-based IntraVS: (a) an original G_P^{RAEP} , (b) a modified G_P^{RAEP} , and (c)-(d) the speed change graphs of three IntraVS algorithms for the paths (b_1, b_3, b_4) and (b_1, b_3, b_5) , respectively.

probability of branch edges and the average number of loop iterations in a CFG of the MPEG-4 video decoder are estimated using the profiled information. A probability of 0.5 is assigned to the branch edges for which we cannot collect the execution profiles with sample test bitstreams. For the experiments, the slew rate of the clock/voltage transition is assumed to be $1.0V/200\mu\text{sec}$, which is typical for state-of-the-art DC-DC converters.

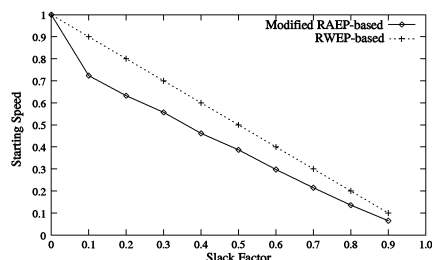


Figure 5: Normalized starting speed changes of the RWEP-based IntraVS and the RAEP-based IntraVS (varying the slack factor).

Figure 5 shows how the normalized starting speeds change over various slack factor values. The slack factor, defined by $\frac{\text{deadline} - \text{WCET}}{\text{deadline}}$, represents the fraction of time that a processor becomes idle after WCET. The execution times of modified ACEPs (by the procedure described in Section 3.2) for the MPEG-4 decoder is up to 35% smaller than the WCET. This means that the processor can start initially 35% more slowly than the speed required by the RWEP-based IntraVS algorithm.

Figure 6 compares the energy consumption of two IntraVS scheduling algorithms, varying the slack factor. (All the results were normalized over the energy consumption of the original program running on a DVS-unaware system.) For the MPEG-4 decoder, the modified RAEP-based IntraVS algorithm reduces the energy consumption up to 34% over the RWEP-based IntraVS algorithm.

Note that there is a large gap between energy consumption of RWEP-based and RAEP-based IntraVS algorithms, even when the slack factor is 0 (i.e. deadline = WCET). This is because, although the starting speed is set to the same speed as in the RWEP-based IntraVS, there are many execution paths that still can take advan-

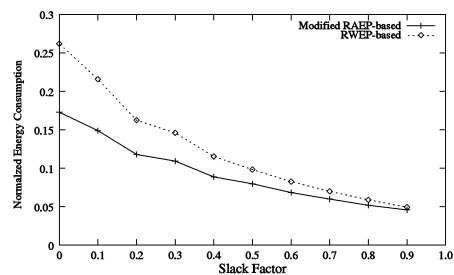


Figure 6: Normalized energy consumption of the RWEP-based IntraVS and RAEP-based IntraVS (varying the slack factor).

tage of the RAEP-based speed settings. That is, in order to meet the timing constraint, virtual blocks are added so that the initial speed is set to the same speed as in the RWEP-based IntraVS algorithm. However, the (partial) paths following the virtual blocks can take advantage of the ACEP-based speed settings. As the slack factor increases, the energy consumption gap decreases because supply voltages of both IntraVS algorithms get lower. Since the energy consumption is proportional to V_{DD}^2 , the lower voltage values result in a smaller difference in the energy consumption.

5. CONCLUSION

We have presented a novel IntraVS algorithm based on the RAEP information. The proposed algorithm exploits the fact that the average-case execution paths are more likely to be followed at run time than the WCET, and optimize the energy consumption for such hot paths. The main contribution of the proposed algorithm is that it enhances the original IntraVS algorithm by exploiting the probability of each execution path, while guaranteeing the worst-case timing constraints. The experimental results using an MPEG-4 video decoder show that the RAEP-based IntraVS improves the energy efficiency up to 34% over the RWEP-based IntraVS.

6. REFERENCES

- [1] T. Ball and J. R. Larus. Using paths to measure, explain, and enhance program behavior. *IEEE Computer*, 33(7):57–65, 2000.
- [2] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processor. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 178–187, 1998.
- [3] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of International Symposium On Low Power Electronics and Design*, pages 197–202, 1998.
- [4] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of the 37th Design Automation Conference*, pages 806–809, 2000.
- [5] Y. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, pages 272–279, 1999.
- [6] T. Sakurai and A. Newton. Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas. *IEEE Journal of Solid State Circuits*, 25(2):584–594, 1990.
- [7] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers*, 18(2):20–30, 2001.
- [8] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of the 36th Design Automation Conference*, pages 134–139, 1999.
- [9] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995.

An Opcode Encoding Method for Low-Power Instruction Fetch

Sunghwan Kim

Department of Computer Science
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-5378
Fax: +82-2-871-4912
e-mail: shift@davinci.snu.ac.kr

Jihong Kim

Department of Computer Science
Seoul National University
Seoul, Korea 151-742
Tel: +82-2-880-8792
Fax: +82-2-871-4912
e-mail: jihong@davinci.snu.ac.kr

Abstract— In designing today's mobile embedded systems such as cellular phones and PDAs, power consumption is an important design constraint. In a CMOS circuit, switching activity accounts for over 90% of total power dissipation. In this paper, we describe a method of encoding opcodes for low-power instruction fetch by reducing the switching activity from the instruction fetch logic. To reduce the switching activity from the instruction-fetch logic, our method encodes opcodes so that more frequently consecutive instruction pairs have a smaller Hamming distance between their opcodes. Our experiment shows that the switching activity reduction of 36.4% to 66.7% is achievable over a naive encoding method.

I. INTRODUCTION

Power consumption has become a dominant design constraint for mobile embedded systems such as cellular phones and PDAs. In digital CMOS circuits (that use well-designed logic gates), switching activity accounts for over 90% of total power consumption [1]. Therefore, many techniques have been proposed and developed to reduce the amount of switching activity in multiple levels of design abstraction [2].

One popular approach widely used in reducing the switching activity is to encode digital values so that the number of bit changes by the values are reduced. For example, bus-invert coding tries to minimize the power dissipated in system bus by dynamically inverting the bus lines if the inversion reduces the number of bits switched on system bus [3]. Gray code addressing takes advantage of temporal redundancy in the instruction access patterns during program execution by using Gray-coded addresses as instruction addresses [4]. Register relabeling modifies the register number assignments so that more frequently consecutive register pairs have a smaller Hamming distance, reducing the switching activity in the register fields during instruction fetches and decodes [5].

In this paper, we describe a method of encoding opcodes for low-power instruction fetches by reducing the

switching activity from the instruction fetch logic. Many redundant bit changes between consecutive instructions can be removed by encoding opcodes so that more frequently consecutive instruction pairs have a smaller Hamming distance between their opcodes. In principle, our method is similar to Gray code addressing [4] and register relabeling [5], in that digital values are statically encoded to minimize the number of bit changes by the values. However, we believe that this is the first attempt applying the low-power encoding scheme for the opcode encoding. For benchmark programs we tested, we were able to reduce the switching activity by 36.4% to 66.7% over a naive encoding method. We explain the opcode encoding method in Section II and report the experimental results in Section III.

II. LOW-POWER OPCODE ENCODING

A. Basic Idea

When a new instruction is fetched from the instruction cache into the instruction register, many bit positions of the current instruction cache bus and instruction register are switched to the opposite state. The switching activity during the instruction fetch phase is directly proportional to the number of bits switched in the instruction cache bus and instruction register between the successively fetched instructions. In order to reduce the switching activity from the instruction fetch logic, therefore, it is necessary to encode each field of an instruction so that more frequently consecutive field values have a smaller Hamming distance between their values. Our method of encoding opcodes is based on the observation that the distribution of instruction transitions is not uniform, but highly skewed. By assigning opcodes with a smaller Hamming distance to more frequently consecutive instruction pairs, the switching activity from the opcode field can be decreased.

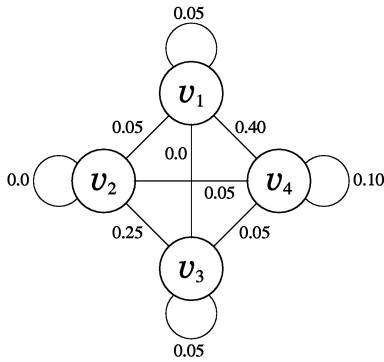


Fig.1. An Example Instruction Transition Graph G

B. Instruction Transition Graph

Once an instruction set architecture of a microprocessor is determined and its compiler is available, using the benchmark programs of the target application domains of the microprocessor, we can measure the instruction transition frequencies for all the pairs of the instructions. The instruction transition frequencies can be represented by an *instruction transition graph* (ITG) $G = (V, E, w)$ where V is a set of instructions, E is a set of the undirected edges between all the elements in V , and w is a probability density function that maps each edge $e = (v_1, v_2)$ in E to a real number between 0 and 1. $w(e)$ indicates the relative frequency of the instruction transitions between v_1 and v_2 . Fig. 1 shows an example instruction transition graph G with four instructions. Each edge represents the relative transition frequency between two instructions connected by the edge. For example, the transition frequency between instructions v_2 and v_3 is 0.25.

An instruction transition graph of a benchmark program can be constructed by counting the number of transitions between all the pairs of the instructions. Once an instruction transition graph of an individual benchmark program is obtained, a global instruction transition graph can be built by merging the individual instruction transition graphs using an appropriate weighting scheme (e.g., equal-time weighting).

C. Optimal Opcode Encoding

Given a global instruction transition graph $G = (V, E, w)$, our goal is to find an opcode assignment that minimizes the number of bit changes on the opcode field. If an instruction set architecture has M instructions, we need at least $\lceil \log_2 M \rceil$ -bit codes to uniquely identify the M instructions. An opcode assignment can be represented by an opcode assignment function $f : V \rightarrow S$ where S is a set of binary codes of length $\lceil \log_2 |V| \rceil$. The quality of the opcode assignment function is determined by our power metric $asa(G, f)$, the average switching ac-

Instruction	Opcode	Instruction	Opcode
v_1	00	v_1	00
v_2	01	v_2	01
v_3	10	v_3	11
v_4	11	v_4	10

a. assignment function f_1
 $asa(G, f_1) = 1.45$

b. assignment function f_2
 $asa(G, f_2) = 0.85$

Fig.2. Two Opcode Assignment Functions for the Instruction Transition Graph G shown in Fig. 1

tivity per instruction in G under f , which is defined as follows:

$$asa(G, f) = \sum_{e=(v_1, v_2) \in E} w(e) \times h(f(v_1), f(v_2))$$

where h is a function that returns the Hamming distance between two binary codes. As an example, consider the instruction transition graph G shown in Fig. 1. For two opcode assignment functions f_1 and f_2 shown in Figs. 2a and 2b, we can see that $asa(G, f_1)$ is 1.45 and $asa(G, f_2)$ is 0.85. That is, the switching activity in the opcode field is reduced by 41.4% by changing the opcode assignment function from f_1 to f_2 .

Finding an optimal opcode assignment function from an instruction transition graph is an NP-hard problem [6]. For an approximate solution, we have used three heuristics, *2-opt* [7], *simulated annealing* [8], and *slack-based heuristic* [6]. The 2-opt heuristic repeatedly swaps the opcodes of randomly selected two instructions if the swap results in the switching activity reduction. When a locally optimal solution is found, the 2-opt heuristic restarts another local search from a random solution. Simulated annealing is similar to the 2-opt heuristic, but it may swap the opcodes even if the swap increases the switching activity. The slack-based heuristic first sorts all the instruction pairs in the decreasing order based on their contributions to the total switching activity, then changes the opcodes starting from the first instruction pair in the sorted list to reduce switching activity.

D. Decoding Restriction

In practice, for a simpler decoding logic implementation, we often do not have the complete freedom in assigning opcodes to the instructions. For example, the instructions of similar types may have the same bit pattern for some of their opcodes. In order to reflect this restriction during the optimization process, the opcode field is divided into two subfields, the decode-restricted subfield and decode-free subfield. The decode-restricted subfield represents the portion of the opcode whose encoding is limited for a simpler decoder implementation. The decode-free subfield, on the other hand, can be assigned to any code possible.

TABLE I
TOP 10 INSTRUCTION PAIRS WITH THE HIGHEST TRANSITION
FREQUENCIES

SPEC CPU95 Benchmark		UTDSP Benchmark	
Inst. Pair	Frequency	Inst. Pair	Frequency
sll, addu	6.6%	addiu, sw	4.8%
mul.d, l.d	3.9%	addiu, lw	4.4%
lw, lw	3.5%	lw, lw	3.2%
sw, sw	3.3%	sw, sw	3.1%
addu, l.d	2.9%	addiu, addiu	2.9%
add.d, l.d	2.8%	addu, sw	2.8%
addu, lw	2.8%	lw, bne	2.2%
addiu, addiu	2.4%	addiu, addu	2.2%
addiu, sw	2.3%	sw, lw	2.2%
addiu, lw	2.1%	addiu, beq	2.1%
total	32.6%	total	29.9%

To find an optimal opcode encoding under the decoding restriction, we use a two-phase optimization method. In the first phase, the encoding of decode-restricted subfield is determined. Instructions that have the same value for the decode-restricted subfield form an instruction group. To minimize the switching activity from the decode-restricted subfield, information on the transition frequencies between the instruction groups is necessary. The transition frequencies between the instruction groups can be extracted from the global instruction transition graph and represented by a *group transition graph (GTG)*. A group transition graph can be constructed in a similar fashion as an instruction transition graph except that a vertex represents an instruction group, not an instruction. Given a group transition graph, we can find the encoding for the decode-restricted subfield with the same manner as the optimal opcode encoding from an instruction transition graph is found. In the second phase, with the decode-restricted subfield fixed, the encoding of decode-free subfield is decided using an appropriate heuristic.

III. EXPERIMENTAL RESULTS

In order to compare the switching activity reduction over a naive opcode encoding method, we have performed experiments using the SimpleScalar tool set [9]. The SimpleScalar architecture is a derivative of MIPS architecture and has 119 instructions with the 7-bit opcode field. Two benchmark suites, SPEC CPU95 benchmark [10] and UTDSP benchmark [11], were used for the experiments to evaluate the applicability of the proposed method in the different application domains. Two ITGs, G_{SPEC} and G_{UTDSP} , were built from the instruction transition information collected from the modified SimpleScalar simulator. As expected, the probability density functions w_{SPEC} and w_{UTDSP} of G_{SPEC} and G_{UTDSP} , respectively, were highly skewed: for example, about 1% of the total instruction pairs accounted for about 90% of the

TABLE II
SWITCHING ACTIVITY REDUCTION RESULTS FROM SPEC CPU95
BENCHMARK

	2-opt Heuristic	Simulated Annealing	Slack-based Heuristic
applu	62.1%	62.1%	59.9%
compress	44.3%	45.8%	46.2%
fpppp	50.5%	51.2%	51.0%
gcc	39.6%	38.9%	39.3%
m88ksim	45.2%	47.1%	49.2%
perl	40.3%	38.9%	40.7%
tomcatv	45.9%	44.6%	44.1%
wave	53.9%	51.3%	47.1%
average	49.1%	49.5%	48.0%

TABLE III
SWITCHING ACTIVITY REDUCTION RESULTS FROM UTDSP
BENCHMARK

	2-opt Heuristic	Simulated Annealing	Slack-based Heuristic
V32	41.7%	38.5%	40.3%
adpcm	43.0%	46.6%	44.5%
edge_detect	37.5%	36.4%	37.0%
histogram	43.9%	40.9%	41.8%
jpeg	45.0%	48.9%	45.9%
lpc	37.4%	40.8%	38.7%
spectral	66.7%	66.7%	66.7%
trellis	46.3%	50.8%	50.6%
average	45.2%	46.3%	45.7%

total instruction transitions. Table I lists the top 10 instruction pairs that have the highest transition frequencies. The top 10 instruction pairs (out of the total 7140 pairs) account for 32.6% and 29.9% of the total instruction transitions for SPEC CPU95 and UTDSP, respectively. From the ITGs, the optimized opcode assignment functions f_{SPEC} and f_{UTDSP} are obtained using the three heuristics described in Section II.C. Tables II and III summarize the switching activity reduction results for the selected benchmark programs over the original opcode assignment used for SimpleScalar. As shown in Tables II and III, three heuristics performed equally well. With the 2-opt heuristic, on an average, switching activities were reduced by 49.1% for SPEC CPU95 and 45.2% for UTDSP. In order to consider the decoding restriction, we have performed the experiments with the upper three bits (out of the 7-bit opcode field) set as the decode-restricted subfield for a simpler decoding. Tables IV and V show the switching activity reduction results under the decoding restriction using the 2-opt heuristic. Although the improvements are smaller than ones reported in Tables II and III, on an average, the switching activity reduction of 38.2% and 36.6% is achieved for SPEC CPU95 and UTDSP, respectively.

TABLE IV
SWITCHING ACTIVITY REDUCTION RESULTS FROM SPEC CPU95
BENCHMARK UNDER THE DECODING RESTRICTION

	No Restriction	Decoding Restriction
applu	62.1%	57.8%
compress	44.3%	32.0%
fpppp	50.5%	39.2%
gcc	39.6%	26.0%
m88ksim	45.2%	30.0%
perl	40.3%	30.7%
tomcatv	45.9%	27.1%
wave	53.9%	48.0%
average	49.1%	38.2%

IV. CONCLUSION

A method of encoding opcodes for low-power instruction fetch is presented. It is based on the observation that the distribution of the instruction transitions is highly skewed. Our method exploits this observation in the opcode encoding so that more frequently consecutive instruction pairs are encoded to have a smaller Hamming distance between their opcodes. Experimental results show that we can reduce the switching activity from the instruction fetch logic by 36.4% to 66.7% over a naive opcode encoding method. Considering the decoding restriction for a simpler decoding, the encoding method is still effective, reducing the switching activity by 26.0% to 57.8% over a naive encoding method.

ACKNOWLEDGEMENT

This research was supported in part by S.N.U. Posco Research Fund (98-09-2093).

REFERENCES

- [1] A. Chandrakasan, T. Shung, and R. W. Brodersen, "Low power CMOS digital design," *Journal of Solid State Circuits*, Vol. 27, No. 4, pp. 473-484, 1992.
- [2] S. Devadas and S. Malik, "A Survey of optimization techniques targeting low power VLSI circuits," *Proceedings of the 32nd Design Automation Conference*, pp. 242-247, 1995.

TABLE V
SWITCHING ACTIVITY REDUCTION RESULTS FROM UTDSP
BENCHMARK UNDER THE DECODING RESTRICTION

	No Restriction	Decoding Restriction
V32	41.7%	32.7%
adpcm	43.0%	34.6%
edge_detect	37.5%	33.0%
histogram	43.9%	33.9%
jpeg	45.0%	37.5%
lpc	37.4%	27.8%
spectral	66.7%	50.0%
trellis	46.3%	42.0%
average	45.2%	36.6%

- [3] M. R. Stan and W. P. Bursleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on VLSI Systems*, Vol. 3, No. 1, pp. 49-58, 1995.
- [4] L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," *Proceedings of COMPCON'94*, pp. 489-498, 1994.
- [5] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh, "Techniques for low energy software," *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, pp. 72-75, 1997.
- [6] V. Veeramachaneni, A. Tyagi, and S. Rajgopal, "Re-encoding for low power state assignment of FSMs," *Proceedings of the International Symposium on Low Power Design*, pp. 173-178, 1995.
- [7] E. Aart and J. K. Lenstra, *Local search in combinatorial optimization*, John Wiley & Sons Ltd., Baffins Lane, Chichester, 1997.
- [8] L. Davis and M. Steenstrup, *Handbook of Genetic Algorithms*, L. Davis Ed., PHG Van Nostrand Reinhold, 1991.
- [9] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," <http://www.cs.wisc.edu/~mscalar/simplescalar.html>, 1997.
- [10] SPEC, "SPEC CPU95 benchmarks," <http://www.spec.org/org/cpu95>, 1995.
- [11] M. Stoodley and C. Lee, "UTDSP benchmark suite," <http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html>, 1997.

An Operation Rearrangement Technique for Power Optimization in VLIW Instruction Fetch

Abstract

As mobile applications are required to handle more computing-intensive tasks, many mobile devices are designed using VLIW processors for high performance. In VLIW machines where a single instruction contains multiple operations, the power consumption during instruction fetches varies significantly depending on how the operations are arranged within the instruction. In this paper, we describe a post-pass operation rearrangement method that reduces the power consumption from the instruction-fetch datapath. The proposed method modifies operation placement orders within VLIW instructions so that the switching activity between successive instruction fetches is minimized. Our experiment shows that the switching activity can be reduced by 34% on average for benchmark programs.

1 Introduction

As mobile applications are required to handle more computing-intensive tasks (such as video decoding), many mobile devices are designed using VLIW processors for high performance. For example, the Crusoe processors [6] from Transmeta (which were developed for mobile Internet computing market) are based on 64 bits or 128 bits VLIW CPU cores. Fujitsu Microelectronics' FR300 [4] (whose main application area is in wireless cellular phones) also has a VLIW architecture. In addition, there are many VLIW digital signal processors such as Texas Instruments' TMS320C6x series that can be used for wireless devices [3, 5].

While VLIW CPU-based mobile devices generally provide enough computing power to handle many computing intensive applications, they usually consume a large amount of power. For example, TMS320C620x processors consume between 1.2W and 2.3W at 1.8V while high-end embedded microprocessors such as StrongArm 110 consume between 100mW and 1W at 3V [15, 9]. Therefore, in designing VLIW CPU-based mobile devices, low power consumption becomes an important design constraint.

In digital CMOS circuits (that use well-designed logic gates), dynamic power consumption accounts for over 90% of total power consumption [1]. Since the dynamic power consumption is proportionate to the switching activity \times capacitance term, to reduce the overall dynamic power consumption, the switching activity should be reduced from the components having large capacitances. For example, system-level off-chip busses are such components. The power consumption from off-chip driving can reach up to 70% of the total chip power, where bus transitions are the most dominant factor due to the large capacitances of the bus lines [7].

Many techniques have been proposed and developed to reduce the frequency of bit transitions in a system-level off-chip bus [12, 11, 13, 16]. For example, various bus encoding techniques [12, 11] convert bus data representations, exploiting the characteristics of bus access patterns, so that the power consumption from the off-chip bus can be reduced. On the other hand, low-power instruction scheduling techniques [13, 16] modify instruction placement orders to reduce the bit changes from the successive instruction fetches.

In this paper, we propose a post-pass optimization technique that can significantly reduce switching activity from both the internal and external instruction busses of VLIW processors. The proposed method takes advantage of a VLIW machine's instruction encoding characteristic: VLIW CPUs can place the same operation in multiple operation slots within the VLIW instruction.¹ We reduce switching activity by modifying operation placement orders within VLIW instructions so that the switching activity in the instruction-fetch datapath is minimized. The proposed technique also takes into account of the inter-block switching activity, which was ignored in the existing low-power instruction scheduling techniques, in scheduling instructions for low-power fetches.

The main contribution of this paper is two-fold. First, as far as we know, our work is the *first* attempt to reduce the power consumption from the VLIW instruction-

¹We distinguish between an operation and an instruction in a VLIW CPU. A VLIW *instruction* is assumed to consist of several *operations*.

fetch datapath² by instruction scheduling. Second, the proposed technique, unlike the existing low-power instruction scheduling such as [13, 16], tries to reduce the overall power consumption from both the on-chip and off-chip instruction busses. In reordering instructions for low-power instruction fetch, existing low-power instruction scheduling techniques do not consider simultaneously both the on-chip bus consumption and the off-chip bus consumption. For example, the instruction scheduling technique [16] by Tomiyama *et al.* does not take account of the switching activity at the on-chip instruction bus. For their target processors, since off-chip bus accesses are the dominating power consumer, the authors did not consider to reduce the power consumption from the on-chip bus, which contributes much less to the overall power consumption than the off-chip bus.

On the other hand, in VLIW processors, an instruction scheduling technique must weigh the switching activity from the on-chip instruction bus as well as the off-chip instruction bus. Since the width of the on-chip instruction bus is generally much larger than that of the off-chip instruction bus in VLIW architectures, if an instruction schedule were produced considering only the bit changes from the off-chip bus, it might be a bad (thus more bit-changing) schedule for the on-chip instruction bus. Furthermore, since the on-chip instruction bus has the large wire capacitance as well as the large output load capacitance [17] and it is used every cycle in a high speed, the impact on the total power consumption of the switching activity at the on-chip instruction bus cannot be ignored.

The organization of the rest of the paper is as follows. Before presenting the proposed operation rearrangement technique, we review prior work on low-power techniques for instruction fetch in Section 2. In Section 3, we describe a target VLIW machine model and define several terms. An operation rearrangement technique is explained in Section 4. Experimental results are presented in Section 5 followed by conclusions in Section 6.

2 Related Work

The research to reduce the bit transitions of bus can be classified by three approaches.

The first approach is the bus encoding. Bus-invert coding [12] reduces a significant number of bit changes from bus lines by dynamically inverting the bus lines when the number of switched bus lines is more than half the number of bitlines. Shin *et al.* advanced the bus-invert coding by selecting a sub-group of bus lines involved in bus encoding to avoid unnecessary inversion of bus lines not in the sub-group [11].

²This includes an external memory, an off-chip bus, an instruction cache and an on-chip bus.

The second approach is the instruction scheduling. Su *et al.* proposed an instruction scheduling technique, called cold scheduling, to reduce the amount of switching activity in the control path [13]. Used in conjunction with a traditional list scheduling algorithm, cold scheduling schedules instructions in the ready list based on the power cost of an instruction. The power cost of an instruction is determined by the number of bit changes when the instruction in question is scheduled following the last instruction. Tomiyama *et al.* proposed an instruction scheduling technique which reduces transitions on an instruction bus between an on-chip cache and a main memory when instruction cache misses occur [16]. This scheduling technique schedules instructions in each basic block in a way that binary representations of consecutive two machine instructions are less different while maintaining the control/data dependencies of the original program.

The third approach is the instruction encoding. Register relabeling [8] assigns register numbers of instructions so that more frequently consecutive register numbers have a smaller Hamming distance, thus reducing the switching activity of the instruction bus and decode logic. The instruction scheduling and instruction encoding techniques also reduce the switching activity of the instruction fetch and decoding logic.

Most of existing low-power instruction scheduling techniques (including the techniques described above), however, assume that processors can issue at most one instruction at each cycle. Therefore, these techniques cannot be directly applied to multiple-issue machines such as a VLIW CPU. In a VLIW CPU, since multiple operations are packed into a single instruction, two levels of scheduling decisions should be made to reduce power consumption. In the first level, we have to decide that which operations are packed into which instructions. Once the first level scheduling decision is made, in the second level, we have to decide which orders the selected operations are placed in specific instructions. The technique proposed in this paper solves the second-level low-power scheduling problem for a VLIW CPU assuming that the decision for the first-level scheduling problem was already made.

3 VLIW Machine Model and Definitions

3.1 Target VLIW Machine Model

VLIW architectures use long instruction words to execute multiple operations simultaneously. In specifying multiple operations within a single VLIW instruction, two encoding methods are typically used: uncompressed encoding and compressed encoding [2]. In a VLIW machine with an uncompressed encoding, each operation slot of a VLIW instruction corresponds to a particular functional unit. The

operation specified in a particular operation slot, therefore, is executed only in the corresponding functional unit. If a functional unit is not scheduled to execute an operation at the given cycle, NOP should be specified in the corresponding operation slot. Under this encoding method, the number of candidate operation slots for an operation is limited to the number of corresponding functional units that can execute the operation.

On the other hand, in a VLIW machine with a compressed encoding, the position of operation slots within a VLIW instruction does not directly correspond to a particular functional unit. The assignment of a particular functional unit to an operation is generally decided by the functional unit subfield of the operation encoding. The functional unit subfield specifies which functional unit should be assigned to the operation. In addition, in order to increase memory utilization, NOP operations are not explicitly encoded in the VLIW instruction. In this type of VLIW machines, an operation can be placed in any operation slot within the same VLIW instruction.

Figures 1 shows compressed encoding method using a sample VLIW program sequence S . In the program sequence S , three VLIW instructions are shown where “||” specifies parallel operations that are executed simultaneously. For a compressed VLIW instruction encoding shown in Figures 1.(b) and 1.(c), there are many chances for operation rearrangements because there is no direct correspondence between the position of an operation slot and a corresponding functional unit. For example, for the first VLIW instruction of S , $4!$ different operation rearrangements are all possible.³ Although the proposed operation rearrangement technique is equally effective for a VLIW machine with an uncompressed encoding, we assume that a target VLIW CPU was encoded using a compressed encoding method.

Throughout this paper, we consider a target system with an architectural organization shown in Figure 2. The VLIW processor with a compressed encoding has an on-chip instruction cache. The VLIW instructions are fetched through the b_{cache} -bit width instruction bus. If the instruction is not found in the on-chip instruction cache, the corresponding memory block is fetched from the main memory through the b_{mem} -bit width instruction bus. Because of the compressed encoding format, several VLIW instructions can be fetched together in a single fetch from the instruction cache. We call these instructions a *fetch packet* as a group. For a description purpose, we make the following assumptions on the target system:

³In Figures 1.(b) and 1.(c), parallel operations within the same VLIW instruction is specified using tail bits (shown in the shaded boxes). If a tail bit of an operation O is 1, the operation O is executed in parallel with the next operation. Otherwise, the next operation is executed after the current instruction is executed.

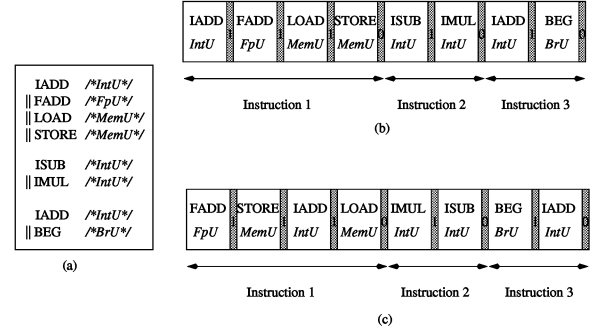


Figure 1. Compressed VLIW instruction encoding; (a) a sample instruction sequence S , (b) one compressed encoding of S and (c) an alternative encoding of S .

- In a single b_{cache} -bit fetch packet, exactly N operations are included. (That is, the width of a single operation slot is exactly b_{cache}/N .)
- No instruction crosses the fetch packet boundary.
- b_{mem} is equal to the operation width. (That is, $b_{mem} = b_{cache}/N$.)
- When the external instruction bus is not used, each line in the external bus is assumed to hold a logic 1 value to prevent from the high impedance condition.

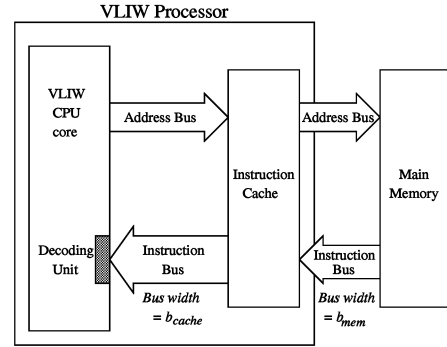


Figure 2. Target system architecture.

3.2 Definitions

In explaining the operation rearrangement technique, we use the following definitions:

Definition 1 A permutation $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ is said to be an operation rearrangement function.

Definition 2 Two VLIW instructions $I_1 = (OP_1^1, OP_2^1, \dots, OP_n^1)$ and $I_2 = (OP_1^2, OP_2^2, \dots, OP_n^2)$ are said to

be equivalent under operation rearrangement if there exists an operation rearrangement function σ such that $OP_{\sigma(i)}^1 = OP_i^2$ for all $1 \leq i \leq n$.

Definition 3 Two fetch packets $FP_1 = (I_1^1, I_2^1, \dots, I_n^1)$ and $FP_2 = (I_1^2, I_2^2, \dots, I_n^2)$ are said to be equivalent under operation rearrangement if there exist operation rearrangement functions $(\sigma_1, \sigma_2, \dots, \sigma_n)$ such that I_i^1 is equivalent to I_i^2 under σ_i for all $1 \leq i \leq n$. $EQ(FP_i)$ is used to represent the set of equivalent fetch packets for a given FP_i .

Definition 4 Two basic blocks $bb_1 = (FP_1^1, FP_2^1, \dots, FP_n^1)$ and $bb_2 = (FP_1^2, FP_2^2, \dots, FP_n^2)$ are said to be equivalent under operation rearrangement if FP_i^1 is equivalent to FP_i^2 under operation rearrangement for all $1 \leq i \leq n$. $EQ(bb)$ is used to represent the set of equivalent basic blocks for a given basic block bb .

Definition 5 Two programs $S_1 = (bb_1^1, bb_2^1, \dots, bb_n^1)$ and $S_2 = (bb_1^2, bb_2^2, \dots, bb_n^2)$ are said to be equivalent under operation rearrangement if bb_i^1 is equivalent to bb_i^2 under operation rearrangement for all $1 \leq i \leq n$. $EQ(S)$ is used to represent the set of equivalent programs for a given program S .

In the rest of paper, we use “equivalent” to mean “equivalent under operation rearrangement” where no confusion arises.

4 Operation Rearrangement Problem

In this section, we introduce the operation rearrangement problem and present its solution, formulating the problem into a shortest path problem.

4.1 Basic Idea

In order to reduce the switching activity during the instruction fetch phase in a target system, we reduce the number of bit transitions between successive instruction fetches, because switching activity is directly proportional to the number of bit changes. Since, in a VLIW machine with a compressed encoding, an operation can be placed in any operation slot within the instruction boundary, the number of bit transitions between successive instruction fetches can be reduced by reordering given VLIW instructions to equivalent instructions that have less switching activity. Consider an example shown in Figure 3. There are four fetch packets each of which is 32-bit wide (that is, $b_{cache} = 32$). In the example, each fetch packet consists of a single VLIW instruction which in turn consists of four operations. Figure 3.(b) shows the instruction sequence after an operation placement order was modified to reduce the bit transitions in the instruction bus. When the four instructions are executed sequentially only once, the rearranged instruction sequence shown in Figure 3.(b) reduces the total number of

bit changes by about 25% from 39 to 29, while maintaining the same semantics of the original sequence.

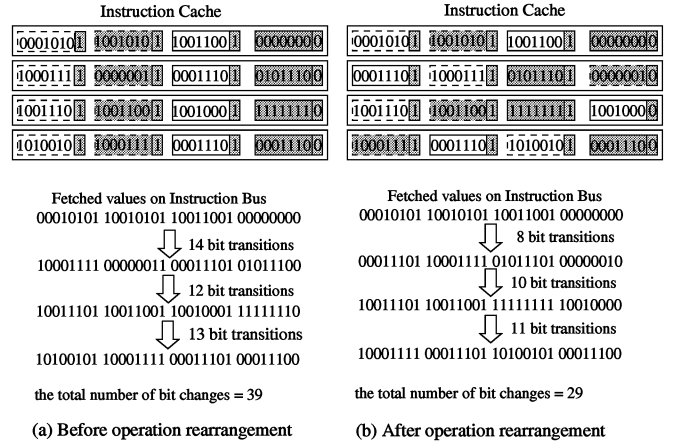


Figure 3. An operation rearrangement example.

4.2 Problem Formulation and Solution

We first consider the operation rearrangement problem for a single basic block where each basic block is assumed to be independent. We call this problem the single basic block (SBB) problem. A complete operation rearrangement problem is solved by extending the solution for the single basic block problem.

4.2.1 Single Basic Block (SBB) Problem

For a given execution of a program P , the total number of bit changes SW^B from a basic block B during the instruction fetch phase is given by the sum of two terms, SW_{cache}^B and SW_{mem}^B . SW_{cache}^B represents the number of bit changes at the internal instruction bus and SW_{mem}^B indicates the number of bit changes at the external instruction bus. Using the notations explained in Table 1, SW_{cache}^B and SW_{mem}^B are computed as follows. (In the explanations below, we use the basic block B_{eg} shown in Figure 4 as an example. The basic block B_{eg} consists of three cache memory blocks, MB_1 , MB_2 and MB_3 , and each cache memory block consists of three fetch packets. Each fetch packet consists of four operations.)

SW_{cache}^B is the sum of all the bit changes incurred during successive fetches of fetch packets from the instruction cache and calculated as follows:

$$SW_{cache}^B = w(B) \sum_{i=1}^{N_{fp}(B)-1} d_{fp}(FP_i^B, FP_{i+1}^B) \quad (1)$$

Symbol	Meaning
$w(B)$	The number of times that a basic block B is executed.
$N_{fp}(B)$	The number of fetch packets in a basic block B .
N_{op}	The number of operations in a fetch packet. (This is a fixed value regardless of B .)
$\mathbf{1}$	The bit vector where every bit is 1 and whose length is b_{mem} .
FP_i^B	The i -th fetch packet of a basic block B .
$OP_n^{FP_i^B}$	The n -th operation of FP_i^B . (Within a fetch packet FP_i^B , the first operation is $OP_1^{FP_i^B}$ and the last one is $OP_{N_{op}}^{FP_i^B}$.)
$d_{fp}(FP_i^B, FP_j^B)$	The Hamming distance between the fetch packets FP_i^B and FP_j^B .
$d_{op}(OP_n^{FP_i^B}, OP_m^{FP_j^B})$	The Hamming distance between the operations $OP_n^{FP_i^B}$ and $OP_m^{FP_j^B}$.
$MB(FP_i^B)$	The memory block that contains FP_i^B .
N_{miss}^{MB}	The number of cache misses of the memory block MB .

Table 1. Notations used in Section 4.2.1

where $w(B)$ is the number of times that a basic block B is executed. The upper portion of Figure 4 shows how Equation (1) is calculated for the example basic block B_{eg} .

SW_{mem}^B is equal to the sum of all the bit changes between adjacent operation fetches from the main memory because we assumed that $b_{mem} = b_{cache}/N_{op}$ in Section 3.1. For the description purpose, if we assume that basic blocks are aligned by the cache memory block size, and their sizes are the multiple of cache memory block size, SW_{mem}^B can be computed by adding the bit changes of all the memory blocks that consist of the basic block B . For such a memory block MB , if we assume that the memory block has K fetch packets, the number of bit changes SW_{mem}^{MB} from the memory block MB at the external instruction bus is given

$$SW_{mem}^{MB} = \sum_{i=1}^K N_{miss}^{MB} \cdot intra(i) + \sum_{i=1}^{K-1} N_{miss}^{MB} \cdot inter(i) \quad (2)$$

where N_{miss}^{MB} is the number of cache misses of the memory block MB , and

$$intra(i) = \begin{cases} d_{op}(\mathbf{1}, OP_1^{FP_i^{MB}}) + S_{op}(i) & \text{if } (i\%K) = 1 \\ d_{op}(OP_{N_{op}}^{FP_i^{MB}}, \mathbf{1}) + S_{op}(i) & \text{if } (i\%K) = 0 \\ S_{op}(i) & \text{otherwise} \end{cases} \quad (3)$$

$$\text{(where } S_{op}(i) = \sum_{n=1}^{N_{op}-1} d_{op}(OP_n^{FP_i^{MB}}, OP_{n+1}^{FP_i^{MB}})$$

$$inter(i) = \begin{cases} 0 & \text{if } (i\%K) = 0 \\ d_{op}(OP_{N_{op}}^{FP_i^{MB}}, OP_1^{FP_{i+1}^{MB}}) & \text{otherwise} \end{cases} \quad (4)$$

In Equations (3) and (4), FP_i^{MB} represents the i -th fetch packet of the memory block MB . In Equation (3), the number of bit transitions between the $\mathbf{1}$ vector and the first operation of the memory block and the number of bit transitions between the last operation of the memory block and $\mathbf{1}$ vector are included in the calculation. This is because we assumed that in Section 3.1, each bus line of the external instruction bus holds a logic 1 value when the bus is not used. The $intra(i)$ and $inter(i)$ terms above can be easily understood with an example. For example, for the first memory block MB_1 of the Figure 4 that consists of three fetch packets, the $intra(i)$ and $inter(i)$ terms are as follows:

$$intra(1) = d_{op}(\mathbf{1}, A) + d_{op}(A, B) + d_{op}(B, C) + d_{op}(C, D)$$

$$intra(2) = d_{op}(E, F) + d_{op}(F, G) + d_{op}(G, H)$$

$$intra(3) = d_{op}(I, J) + d_{op}(J, K) + d_{op}(K, L) + d_{op}(L, \mathbf{1})$$

$$inter(1) = d_{op}(D, E), \quad inter(2) = d_{op}(H, I), \quad inter(3) = 0$$

Since SW_{mem}^B can be computed by summing SW_{mem}^{MB} over

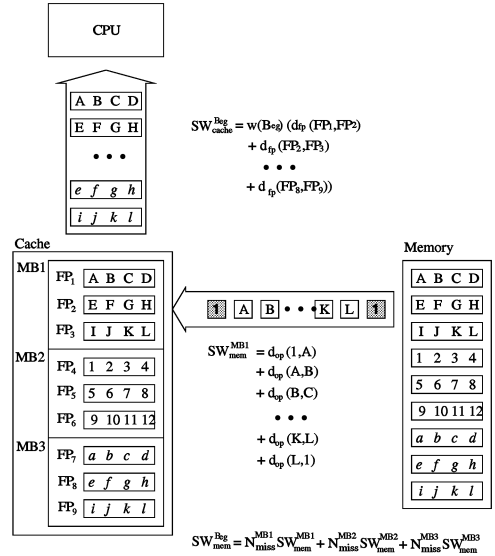


Figure 4. An example calculation of bit transitions at the instruction busses during the execution of a basic block B_{eg} .

all the memory blocks of B , SW_{mem}^B is calculated as follows:

$$SW_{mem}^B = \sum_{i=1}^{N_{fp}(B)} N_{miss}^{MB(FP_i^B)} \cdot intra(i) + \sum_{i=1}^{N_{fp}(B)-1} N_{miss}^{MB(FP_i^B)} \cdot inter(i) \quad (5)$$

Assuming the load capacitance ratio of the external instruction bus to the internal instruction bus is α , SW^B , in the

number of bit transitions at the internal bus, is computed as follows using the Equations (1) and (2):

$$\begin{aligned}
SW^B &= SW_{cache}^B + \alpha \cdot SW_{mem}^B \\
&= \sum_{i=1}^{N_{fp}(B)-1} SW_{FP}^{inter}(FP_i^B, FP_{i+1}^B) \\
&\quad + \sum_{i=1}^{N_{fp}(B)} SW_{FP}^{intra}(FP_i^B)
\end{aligned} \tag{6}$$

where

$$\begin{aligned}
SW_{FP}^{inter}(FP_i^B, FP_{i+1}^B) &= w(B) \cdot d_{fp}(FP_i^B, FP_{i+1}^B) + \alpha \cdot N_{miss}^{MB}(FP_i^B) \cdot inter(i) \\
SW_{FP}^{intra}(FP_i^B) &= \alpha \cdot N_{miss}^{MB}(FP_i^B) \cdot intra(i)
\end{aligned} \tag{8}$$

Given a basic block B , the SBB problem is to find an equivalent basic block B' such that $SW^{B'} \leq SW^{B''}$ for all $B'' \in EQ(B)$. If operations are rearranged, $d_{fp}(FP_i^B, FP_{i+1}^B)$, $d_{op}(OP_n^{FP_j^B}, OP_{n+1}^{FP_j^B})$ and $d_{op}(OP_{N_{op}}^{FP_j^B}, OP_1^{FP_{j+1}^B})$ in Equations (1), (3) and (4) are changed.

4.2.2 Solution for the SBB Problem

We compute an optimal solution for the SBB problem by converting the problem to the shortest path problem between two special nodes, **START** and **END**. Using the notations described in Table 2, given a basic block B , we construct a weighted directed graph $G_B = \{V, E, W_{node}, W_{edge}\}$, where

$$\begin{aligned}
V &= \{\text{START}, \text{END}\} \cup \bigcup_{i=1}^{N_{fp}(B)} EQ(FP_i^B) \\
&= \{\text{START}, \text{END}\} \cup \bigcup_{i=1}^{N_{fp}(B)} \{FP_{i,1}^B, \dots, FP_{i, N_{eq}(FP_i^B)}^B\}, \\
E &= \{(v, w) \mid v = \text{START}, w \in EQ(FP_1^B)\} \cup \\
&\quad \{(v, w) \mid w = \text{END}, v \in EQ(FP_{N_{fp}(B)}^B)\} \cup \\
&\quad \{(v, w) \mid v \in EQ(FP_i^B), w \in EQ(FP_{i+1}^B) \\
&\quad \text{for } 1 \leq i < N_{fp}(B)\},
\end{aligned}$$

$$\begin{aligned}
W_{node}(v) &= \begin{cases} SW_{FP}^{intra}(v) & \text{if } v \in V - \{\text{START}, \text{END}\} \\ 0 & \text{otherwise} \end{cases}, \text{ and} \\
W_{edge}(v, w) &= \begin{cases} SW_{FP}^{inter}(v, w) & \text{if } v, w \in V - \{\text{START}, \text{END}\} \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Figure 5 shows an example graph G_B constructed by transforming the operation rearrangement problem to the shortest path problem. For each fetch packet FP_i^B ,

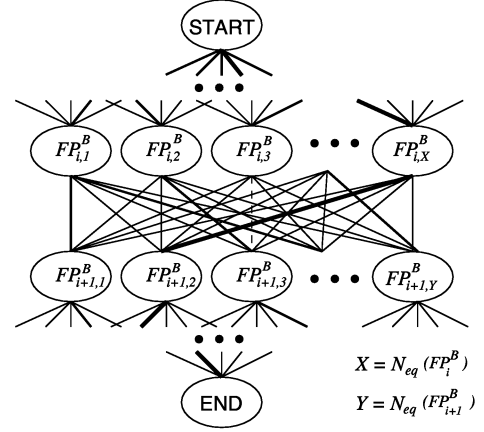


Figure 5. A shortest path problem formulation of the operation rearrangement problem (with node and edge weights omitted).

$N_{eq}(FP_i^B)$ vertices are created in G_B , and for successive fetch packets, FP_i^B and FP_{i+1}^B , every pair of $(FP_{i,k}^B, FP_{i+1,k'}^B)$ is connected by an edge. We call the $N_{eq}(FP_i^B)$ vertices created from the fetch packet FP_i^B to be in the level i . In the graph G_B , the distance of a path $P = (\text{START}, v_1, \dots, v_k, \text{END})$ is given by $\sum_{i=1}^k W_{node}(v_i) + \sum_{i=1}^{k-1} W_{edge}(v_i, v_{i+1})$. The distance of path P is equal to SW^B when each fetch packet FP_i^B is reordered to v_i for $1 \leq i \leq k$.

An optimal solution of the shortest path problem described above can be found by using a modified shortest path algorithm shown in Figure 6. The modified shortest path algorithm is based on the following theorem whose proof is trivial.

Theorem 1 Let a path $P(FP_{i,j}^B) = (\text{START}, v_1, \dots, v_{i-1}, FP_{i,j}^B)$ be the shortest path from **START** to $FP_{i,j}^B \in EQ(FP_i^B)$ and the distance of the path $P(FP_{i,j}^B)$ be $d_{P(FP_{i,j}^B)}$. Then the minimum distance of the path $P(FP_{i+1,k}^B) = (\text{START}, v_1, \dots, v_i, FP_{i+1,k}^B)$, $d_{P(FP_{i+1,k}^B)}$, is given by

$$\min_{1 \leq j \leq N_{eq}(FP_i^B)} [d_{P(FP_{i,j}^B)} + W_{edge}(FP_{i,j}^B, FP_{i+1,k}^B) + W_{node}(FP_{i+1,k}^B)]. \tag{9}$$

In Figure 6, SW_{min} is a variable to store the minimum distance of a path from **START** to $FP_{i+1,k}^B$ (in Line 15) and SW_{cur} is a variable to store the minimum distance of a path from **START** to $FP_{i+1,k}^B$ that passes through $FP_{i,j}^B$. The shortest path is constructed by visiting $MinPath$ in reverse order. The complexity of the modified shortest path algorithm is given by $O(N_{fp}(B) \cdot (N_{eq}^{FP^B})^2)$ where $\frac{N_{eq}^{FP^B}}{N_{fp}(B)} = \frac{1}{N_{fp}(B)} \sum_{i=1}^{N_{fp}(B)} N_{eq}(FP_i^B)$. $\frac{N_{eq}^{FP^B}}{N_{op}!}$ is bounded by $N_{op}!$.

Symbol	Meaning
$N_{ins}(FP_i^B)$	The number of instructions in FP_i^B .
$I_j^{FP_i^B}$	The j -th instruction of FP_i^B ($1 \leq j \leq N_{ins}(FP_i^B)$).
$N_{op}(I_j^{FP_i^B})$	The number of operations in $I_j^{FP_i^B}$.
$N_{eq}(I_j^{FP_i^B})$	The number of instructions that are equivalent to $I_j^{FP_i^B}$ ($N_{eq}(I_j^{FP_i^B}) = (N_{op}(I_j^{FP_i^B}))!$).
$N_{eq}(FP_i^B)$	The number of fetch packets that are equivalent to FP_i^B ($N_{eq}(FP_i^B) = \prod_{j=1}^{N_{ins}(FP_i^B)} N_{eq}(I_j^{FP_i^B})$).
$FP_{i,n}^B$	The n -th fetch packet in $EQ(FP_i^B)$ ($1 \leq n \leq N_{eq}(FP_i^B)$).

Table 2. Notations used in Section 4.2.2

```

1: for  $i \leftarrow 0$  to  $N_{fp}(B)$  {
2:   /* for each vertex in the level  $i + 1$  */
3:   for  $k \leftarrow 1$  to  $N_{eq}(FP_{i+1}^B)$  {
4:      $SW_{min} := \infty$ ;
5:     /* for each vertex in the level  $i$  */
6:     for  $j \leftarrow 1$  to  $N_{eq}(FP_i^B)$  {
7:        $SW_{cur} := d_{P(FP_{i,j}^B)} + W_{edge}(FP_{i,j}^B, FP_{i+1,k}^B)$ 
8:         +  $W_{node}(FP_{i+1,k}^B)$ ;
9:       /* find the minimum value */
10:      if ( $SW_{min} > SW_{cur}$ ) {
11:         $SW_{min} := SW_{cur}$ ;
12:         $MinNode := j$ ;
13:      }
14:    }
15:     $d_{P(FP_{i+1,k}^B)} := SW_{min}$ ;
16:    /* store  $MinNode$  for the final path construction */
17:     $MinPath[FP_{i+1,k}^B] := FP_{i,MinNode}^B$ ;
18:  }
19: }

```

Figure 6. A modified shortest path algorithm.

4.2.3 Operation Rearrangement Problem for Whole Program

The operation rearrangement solution for the SBB problem described above does not take account of inter-block switching activity. Therefore, simply solving the SBB problem for each basic block does not minimize the number of bit changes for a whole program. In order to find a global (thus better) solution for the complete program, we need additional information on the dynamic behavior of program execution as well as ones required for the SBB problem. For example, we should know how branches are resolved in run time to compute the relative adjacency frequency between two basic blocks.

The solution of the operation rearrangement problem for a whole program can be solved in a similar fashion on the SBB problem by transforming the problem to the shortest path problem. The main difference is that in the whole program, because of branches and loops, nodes in a constructed graph for a shortest path problem formulation may

span multiple paths from a given node. We use two techniques, branch merging and loop rolling [10] to convert the graph with no branches and loops. Once the graph is converted to have no branches and loops, the shortest path algorithm for the SBB problem can be used to find a global solution [10].

5 Experiments

In order to evaluate how well the proposed operation rearrangement technique works on application programs, we have performed experiments using a VLIW digital signal processor, TMS320C6201 [14], from Texas Instruments. The TMS320C6201 is a fixed-point DSP that can specify eight 32-bit operations in a single 256-bit instruction. The TMS320C6201 uses a compressed encoding with $b_{cache} = 256$. As benchmark programs, various DSP programs were used. The proposed global solution was implemented as a separate post-pass tool, which takes as an input an executable file produced by the TI's TMS320C6x optimizing C compiler and produces as an output the rearranged low-power version of the same program.

We have measured the number of bit transitions during the instruction fetch phase for each benchmark program using a switching activity counter. Given an executable file with appropriate input data, a switching activity counter program computes the number of bit transitions from both the internal and external busses during the program execution using instruction address traces.

Table 3 summaries the experimental results with selected DSP benchmark programs. For each benchmark program, the average number of bit transitions per instruction fetch (BT/IF) is computed. For α , we have used 100 [12]. We have compared BT/IF's between TI compiler generated programs (the default column in Table 3), and rearranged programs by the proposed operation rearrangement technique (the ORT column in Table 3).

As shown in Table 3, our operation rearrangement technique reduces the number of bit transitions during the instruction fetch phase on an average by 34.3% compared with the programs generated by the TI compiler.

Benchmark Program	Bit transitions/IF		
	default	ORT	Reduction
vector multiply	68.6	43.7	36.3%
FIR8	86.8	56.7	34.6%
FIRcx	79.5	60.5	24.0%
IIR	71.7	51.7	28.0%
lattice analysis	88.4	58.2	34.2%
W_vec	89.5	57.1	36.3%
dotp_sqr	79.2	44.3	44.1%
minerror	50.6	31.3	38.1%
biquad	78.1	52.3	33.0%
Average	76.9	50.6	34.3%

Table 3. Experimental results

6 Conclusions

In this paper we have described and evaluated an operation rearrangement method for power optimization in instruction fetches of VLIW machines. The proposed method, which works as a post-pass tool for compiled programs, reorganizes the operation placement orders within VLIW instructions such that the resulting program has the minimum number of bit transitions during instruction fetches. The experimental results show that the proposed rearrangement technique can reduce the switching activity significantly from the complete instruction-fetch datapath of VLIW machines. For our benchmark programs, the switching activity was reduced by 34% on an average.

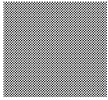
The work described in this paper can be extended in several directions. One of important future tasks is to quantify the *real* energy gains, not the simulated ones, from using the proposed technique. We are currently building a cycle-accurate measurement-based power profiling tool for an embedded microprocessor. If this tool works as expected, we plan to extend it for VLIW processors in the future.

Although we focused on VLIW processors in this paper, a similar operation rearrangement technique can be effective for low-power instruction fetches in superscalar processors. The preliminary result using a four-way superscalar processor suggests that the *total* processor energy can be reduced by about 7%. We are currently implementing a modified operation rearrangement algorithm for a superscalar processor.

In this paper, we considered the problem of modifying operation orders for *pre-compiled* VLIW programs. However, optimization decisions made during the compilation process can affect the outcome of operation rearrangement. For example, depending on how instructions are scheduled, the number of bit changes during the instruction fetch phase can vary significantly. We plan to investigate the phase-ordering problem between the operation rearrangement and other optimization steps as a next research topic.

References

- [1] A. Chandrakasan, T. Shung, and R. W. Broderson. Low power CMOS digital design. *IEEE Journal of Solid State Circuits*, 27(4):473–484, 1992.
- [2] T. Conte, S. Banerjia, S. Larin, K. N. Menezes, and S. W. Sathaye. Instruction fetch mechanisms for VLIW architectures with compressed encodings. In *Proc. of the 29th IEEE/ACM Int. Symp. on Microarchitecture*, pages 201–211, 1996.
- [3] P. Faraboschi, G. Desoli, and J. A. Fisher. The latest word in digital and media processing. *IEEE Signal Processing Magazine*, 15(2):59–85, 1998.
- [4] Fujitsu Microelectronics, Inc. *Fujitsu's new high-performance VLIW processor cores*. <http://www.fujitsumicro.com/>.
- [5] R. Henning and C. Chakrabarti. High-level design synthesis of a low power, VLIW processor for the IS-54 VSELP speech encoder. In *Proc. of Int. Conf. on Computer Design*, pages 571–576, 1997.
- [6] A. Klaiber. *The technology behind the Crusoe processor*. Transmeta Corporation White Paper, 2000.
- [7] D. Liu and C. Svensson. Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid State Circuits*, 29(6):663–670, 1994.
- [8] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh. Techniques for low energy software. In *Proc. of Int. Symp. on Low Power Electronics and Design*, pages 72–75, 1997.
- [9] J.-M. Puiatti, J. Llosa, C. Piguat, and E. Sanchez. Low-power VLIW processors: A high-level evaluation. In *Proc. of Int. Workshop - Power and Timing Modeling, Optimization and Simulation*, pages 399–408, 1998.
- [10] D. Shin and J. Kim. A global operation rearrangement technique for low-power instruction fetch. Technical Report SNU-CSE-AE-99-001, Computer Architecture and Embedded Systems Laboratory, Seoul National University, 1999.
- [11] Y. Shin, S. Chae, and K. Choi. Partial bus-invert coding for power optimization of system level bus. In *Proc. of Int. Symp. on Low Power Electronics and Design*, pages 127–129, 1998.
- [12] M. R. Stan and W. P. Burtleson. Bus-invert coding for low power I/O. *IEEE Trans. on VLSI Systems*, 3:49–58, Mar. 1995.
- [13] C. L. Su, C. Y. Tsui, and A. Despain. Low power architectural design and compilation techniques for high-performance processor. In *Proc. of COMPCON94*, pages 489–498, 1994.
- [14] Texas Instruments. *TMS320C62xx CPU and Instruction Set*, 1997.
- [15] Texas Instruments. *TMS320C6000 Power Consumption Summary*, 1999.
- [16] H. Tomiyama, T. Ishihara, A. Inoue, and H. Yasuura. Instruction scheduling for power reduction in processor-based system design. In *Proc. of the 1998 Design Automation and Test in Europe*, pages 855–860, 1998.
- [17] Y. Zhang, R. Y. Chen, W. Ye, and M. J. Irwin. System level interconnect power modeling. In *Proc. of the 11th international ASIC Conference*, pages 289–293, 1998.



Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications

Dongkun Shin

Jihong Kim

Seoul National University

Seongsoo Lee

Ewha Woman's University

A novel intra-task voltage-scheduling algorithm controls the supply voltage within an individual task boundary. By fully exploiting slack time, it achieves a high-energy reduction ratio. Using this algorithm, a software tool automatically converts an application into a low energy version.

■ **IN MODERN VLSI SYSTEM DESIGN**, power consumption is one of the most important design constraints. For battery-powered portable systems such as digital cellular phones, personal digital assistants, and mobile videophones, low power consumption is a primary design goal because the battery operation time is one of the most critical performance measures. Energy consumption E of CMOS circuits, which is dominated by total dynamic power consumption in most VLSI systems, is given by $E \propto C_L \times N_{\text{cycle}} \times V_{\text{DD}}^2$.¹ C_L is the load capacitance, N_{cycle} is the number of executed cycles, and V_{DD} is the supply voltage. Because energy consumption E has a quadratic dependency on supply voltage V_{DD} , lowering V_{DD} is the most effective way of reducing energy consumption. However, lowering the supply voltage also decreases the clock speed, because CMOS circuit delay T_D is given by $T_D \propto V_{\text{DD}} / (V_{\text{DD}} - V_T)^\alpha$,² where V_T is threshold voltage, and α is a velocity saturation index.

When a given task's required performance is lower than a VLSI system's maximum performance, the clock speed and its corresponding supply voltage can be dynamically controlled to the lowest possible level while meeting the task's deadline constraint. This is the key idea behind the dynamic voltage-scaling (DVS) technique.

For example, consider a task with a deadline of 25 ms, running on a 50-MHz processor with a 5.0-V supply voltage. If executing the task requires 5×10^5 cycles, the processor executes it in 10 ms and idles for the remaining 15 ms. However, if the clock speed and supply voltage are lowered to 20 MHz and 2.0 V, the processor finishes the given task just at the task's deadline (25 ms), resulting in an 84% energy reduction.

Recently, several research groups have investigated the DVS problem for hard real-time systems.³⁻⁶ Most research focused on real-time systems with multiple tasks; in this case, the key question is how to assign the proper speed to each task dynamically while guaranteeing all task deadlines. These techniques exploit the *run-calculate-assign-run* strategy for the supply voltage determination. The steps in that strategy include

1. running the current task,
2. calculating the maximum allowable execution time for the next task,
3. assigning the supply voltage for the next task, and

Table 1. Typical videophone application.

Characteristic	MPEG-4 video encoding	MPEG-4 video decoding	VSELP speech encoding	VSELP speech decoding
Period or deadline (s)	66.667	66.667	40.000	40.000
Worst-case execution time (s)	50.386	9.826	1.844	1.383
Average execution time (s)	13.099	1.460	0.907	0.680
Normalized energy consumption				
Inter-task voltage scheduling ⁹			0.826	
Offline optimal voltage scheduling			0.106	

4. running the next task.

These techniques determine the supply voltage on a task-by-task basis, a strategy we call *inter-task* voltage scheduling.

While generally effective in reducing energy consumption of multitask real-time systems, inter-task voltage scheduling has several practical limitations. For example, because a task scheduler in an operating system determines a task's supply voltage, using inter-task voltage scheduling requires OS modifications. Furthermore, these techniques cannot be applied to a single-task environment, because the supply voltage is determined as a constant value for a given task. Because the single-task model is the basis for many small, embedded mobile applications, variable-voltage processors may be difficult to widely use in practice.

Even in a multi-task environment, inter-task voltage scheduling may not be effective in energy reduction if the execution time of one task dominates total execution time. For example, consider a typical videophone application with the four tasks shown in Table 1. In this application, the MPEG-4 video encoding task dominates execution time but has the lowest priority. For these reasons, inter-task voltage scheduling cannot take advantage of the slack time caused by the MPEG-4 video-encoding task; these algorithms are thus ineffective in reducing energy consumption.

For example, using the inter-task-scheduling algorithm of Shin and Choi⁵ results in only a 17% energy reduction. In contrast, an offline (theoretical) optimal voltage-scheduling algorithm achieves about a 90% energy reduction.

We propose intra-task voltage scheduling—

which adjusts the supply voltage within an individual task's boundary—as a solution to overcome the limitations of inter-task voltage scheduling. For example, a recent work by Lee et al.⁷ demonstrates that dynamic voltage scaling within a single task boundary can significantly reduce energy consumption. Because intra-task voltage scheduling does not involve the OS in adjusting the clock speed, it has an advantage in that existing OSs can be used without modification on a variable-voltage processor.

However, at the current state of the art, it is fully a programmer's responsibility to apply intra-task voltage scheduling to applications. For example, there are no systematic guidelines for selecting the best program locations for inserting voltage-scaling code. Average programmers are generally not familiar with low-energy software issues and timing analysis techniques. In practice, therefore, it is difficult to use intra-task voltage scheduling for real-time applications without the support of a systematic programming methodology.

We propose a novel intra-task voltage-scheduling algorithm that can automate the development of DVS-aware, hard real-time programs on variable-voltage processors. It is based on static execution-time analysis techniques commonly used in developing hard real-time programs. Using these techniques, the proposed algorithm selects locations for inserting voltage-scaling code to reduce the overall energy consumption.

The proposed scheduling algorithm exploits *all* the slack time from runtime variations of different execution paths; there is no slack time when the scheduled program completes its execution, thus significantly improving energy efficiency. The novel aspect of our algorithm is

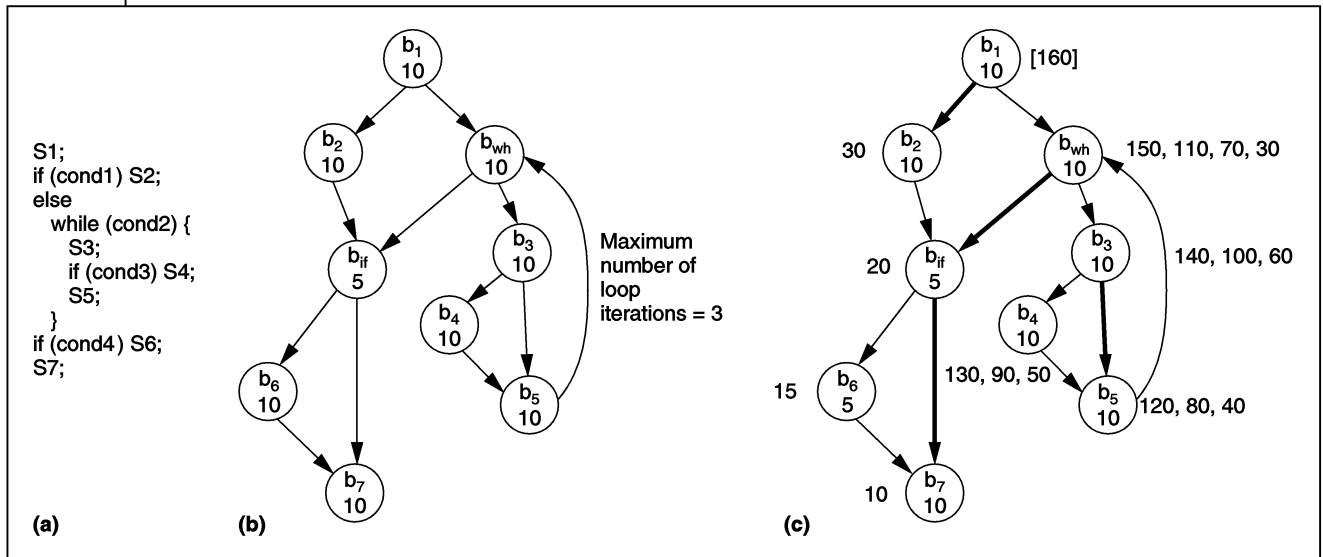


Figure 1. Example program P (a), a real-time program with a 2- μ s deadline, has this CFG representation G_P (b) and an augmented CFG G_P^A with $C_{RWEC}(b_i)$ values (c).

that voltage-scaling decisions are made in compile time, although the voltage-scaling code itself can require some runtime information in determining an appropriate clock speed.

Furthermore, the proposed algorithm provides a systematic methodology for developing an automatic program conversion tool to convert DVS-unaware programs into DVS-aware ones. This means the original program's developers need no knowledge of DVS, making the proposed algorithm very practical.

Based on the proposed algorithm, we have developed a software tool called Automatic Voltage Scaler (AVS). It supports a fully automatic conversion of a DVS-unaware program P into a DVS-aware, low-energy program P_{DVS} that satisfies the same timing requirement as P.

Basic idea

Consider hard real-time program P, as shown in Figure 1a, with a 2 μ s deadline. The control flow graph (CFG) G_P for program P is shown in Figure 1b. In G_P , each node represents a basic block of P, and each edge indicates the control dependency between basic blocks. The number within each node indicates the number of execution cycles for the basic block. The back edge from b_5 to b_{wh} models the while loop of program P.

In developing hard real-time systems where

tasks have strict timing constraints (such as deadlines), the tasks' worst-case execution times (WCETs) are estimated in advance (before runtime) to guarantee that required timing constraints are met. Such WCETs can be predicted by existing WCET analysis tools, which produce safe and accurate WCET prediction results.^{8,9}

Using a WCET analysis tool, we can find path $p_{\text{worst}} = (b_1, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_{if}, b_6, b_7)$ as the worst-case execution path (WCEP) for the example program P, assuming that the user sets the maximum number of while loop iterations to three. The predicted number of execution cycles in p_{worst} is 160 cycles, which is the number of worst-case execution cycles (WCEC) of program P.

If a target processor operates at the 80-MHz maximal clock frequency, program P completes its execution in 2 μ s, resulting in no slack time. We used execution cycles instead of execution times because as we adjust the clock speed on a variable-voltage processor the execution time changes but the number of execution cycles remains constant.

Intra-task voltage scheduling is based on a simple observation: There are large execution time variations among different execution paths. In particular, this strategy exploits the fact that the average-case execution paths

(ACEPs) complete execution much earlier than the WCEP(s).⁵

The example program shown in Figure 1b has 32 different execution paths. While the WCEP p_{worst} takes 160 cycles, eight of 32 possible execution paths take less than 80 cycles. If we can identify such short execution paths in the early phase of execution, we can substantially lower the clock speed and significantly decrease energy consumption.

Consider the path $p_1 = (b_1, b_2, b_{if}, b_6, b_7)$

of Figure 1b; its execution takes 40 cycles. In the ideal case—when we can perfectly predict in advance that the actual execution path will be p_1 —we can start the execution with a clock speed of 20 MHz without violating the 2- μ s deadline. Although this will significantly improve energy efficiency, we cannot start with the 20-MHz clock speed from b_1 , because we do not generally know in advance which execution path the next program execution will take.

In intra-task voltage scheduling, we take the second best approach, with the help of a static program-analysis technique on worst-case execution times. Assume that $C_{\text{RWEC}}(b_i)$ represents the remaining worst-case execution cycles (RWEC) among all the execution paths that start from b_i . Using a modified WCET analysis tool, for each basic block b_i , we compute $C_{\text{RWEC}}(b_i)$ at compile time. For example, Figure 1c shows an augmented CFG G_p^A with $C_{\text{RWEC}}(b_i)$ values. A modified WCET tool statically constructs the graph G_p^A .

For the basic blocks (such as b_{wh} , b_3 , b_4 , b_5) related to the while loop, the corresponding nodes are associated with multiple $C_{\text{RWEC}}(b_i)$ values, reflecting the while loop's maximum three iterations. Once G_p^A is constructed, we can statically identify branching edges (of CFG G) that

drop the remaining worst-case execution cycles faster than the current execution rate.

For example, in Figure 1c, we can identify four such edges, (b_1, b_2) , (b_{wh}, b_{if}) , (b_{if}, b_7) , and (b_3, b_5) . In Figure 1c, these edges are bold face. When the execution control thread branches to the next basic block through one of these edges, say (b_1, b_2) , the clock speed can be lowered because the remaining work is reduced by the difference between $C_{\text{RWEC}}(b_{\text{wh}})$ and $C_{\text{RWEC}}(b_2)$. By reducing clock speed so that the $C_{\text{RWEC}}(b_2)$ cycles can be completed exactly at the deadline, we ensure that the proposed technique always meets the required timing constraint. Because voltage-scaling decisions are made at compile time—not runtime—there exists no runtime overhead directly related to the selection of voltage-scaling edges. In addition, the compile-time analysis procedure does not require special programmer intervention other than that typically required in developing normal hard real-time programs (such as setting the maximum number of loop iterations).

Figure 2 compares how the speed and voltage changes, with and without the use of intra-task voltage scheduling. Assuming that no energy is consumed in an idle state and $E \propto C_L \times N_{\text{cycle}} \times V_{\text{DD}}^2$ when the execution follows path

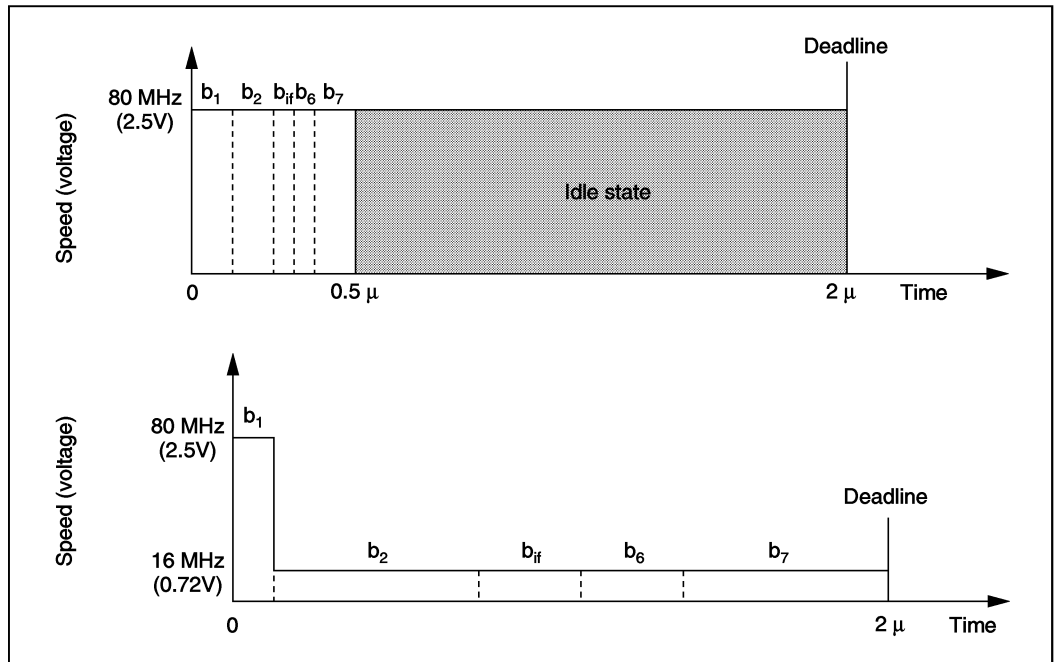


Figure 2. Speed and voltage changes without (a) and with (b) intra-task scheduling.

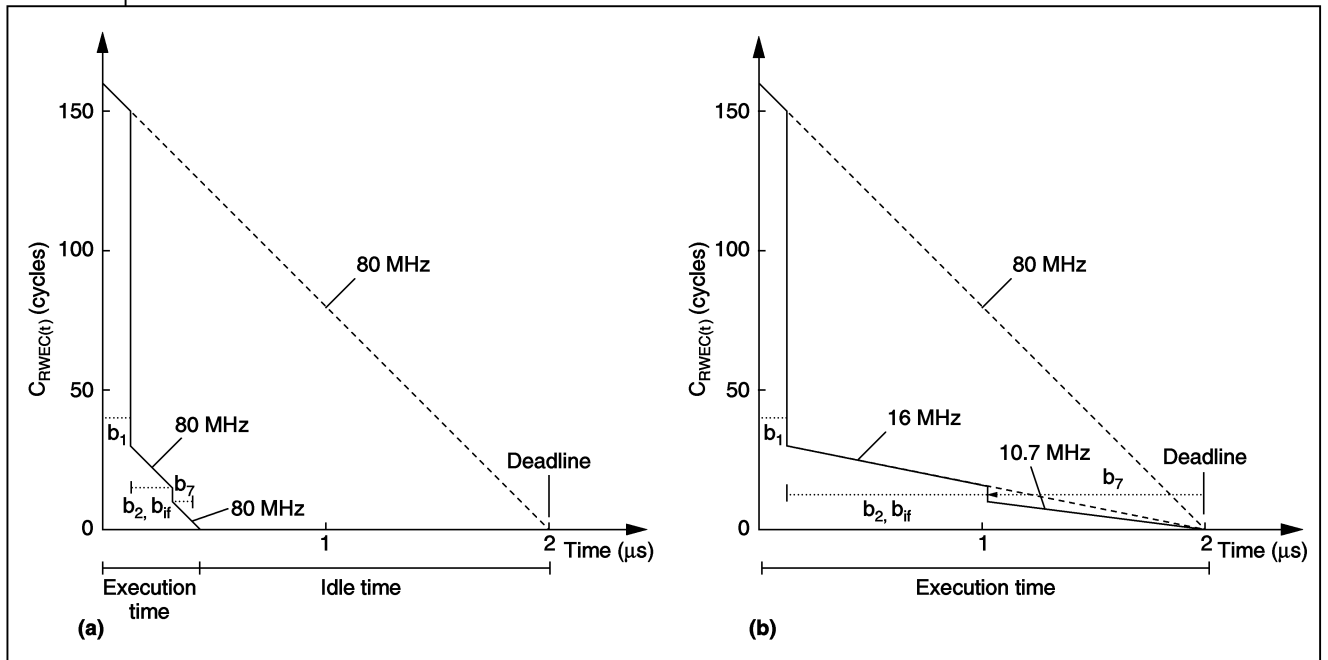


Figure 3. $C_{RWEC}(t)$ changes over different speed-scaling algorithms: no intra-task scheduling (a) and RWEC-based intra-task scheduling (b).

$p_i = (b_1, b_2, b_{if}, b_6, b_7)$, the energy consumption ratio of Figure 2b to Figure 2a is 0.31. Using intra-task voltage scheduling reduces the energy consumption by 69%.

Intra-task voltage-scheduling algorithm

The intra-task voltage-scheduling algorithm assigns each basic block a proper speed at which to execute. For a hard real-time task, this algorithm's goal is to assign the speed to each basic block to minimize energy consumption while satisfying timing requirements. Throughout this article, we assume the following about the target variable-voltage processor:

- The processor provides special instruction change_f_V(f_{CLK}), which can dynamically control the processor's clock frequency f_{CLK} and its corresponding voltage V_{DD} .
- f_{CLK} and V_{DD} can be set continuously within the processor's operational range. When the processor changes clock and voltage, there is a clock/voltage transition overhead period of C_{VTO} cycles.
- During clock/voltage transition, the processor stops running and enters power-down mode.

Although some processors, such as Transmeta's Crusoe,¹⁰ can run during voltage transition, for simplicity we assume that the processor stops. The techniques described in this article can support both processor types with a slight modification of clock/voltage transition overhead modeling.

Remaining WCET-based speed assignment

If actual execution path p_{act} of task τ were known in advance, the optimal execution speed could be easily computed. For each basic block b_i in p_{act} , $S(b_i) = C_{EC}(p_{act})/D$, where $S(b_i)$ represents the processor clock speed in frequency, $C_{EC}(p_{act})$ denotes the number of clock cycles needed to execute p_{act} , and D denotes the deadline of task τ .

However, because the exact execution path is generally unknown until program execution completes, we adjust $S(b_i)$ based on the remaining worst-case execution cycles $C_{RWEC}(b_i)$. Using a modified version of the static WCET prediction algorithm, such as the one developed by S.-S. Lim and colleagues,⁹ we can estimate $C_{RWEC}(b_i)$ for each basic block b_i . $S(b_i)$ is set to clock speed S at which the remaining $C_{RWEC}(b_i)$ cycles can be

completed exactly at the deadline. The quantities $C_{RWEC}(b_i)$ are computed at compile time without incurring any runtime performance penalty.

At entry basic block b_1 , $C_{RWEC}(b_1)$ is set to WCEC, and the starting speed is set to WCEC/D. If $C_{RWEC}(t)$ denotes the remaining worst-case execution cycles at time t , as execution proceeds, $C_{RWEC}(t)$ decreases linearly at the same rate as the clock speed when execution follows worst-case execution path p_{worst} .

However, if execution deviates from basic block b_i in worst-case execution path p_{worst} to a basic block b_j not in p_{worst} , $C_{RWEC}(t)$ drops after the execution of b_i is completed. It drops by the difference between $C_{RWEC}(b_i) - C_{EC}(b_i)$ and $C_{RWEC}(b_j)$, where $C_{EC}(b_i)$ denotes the number of clock cycles needed to execute b_i .

Figure 3 shows how $C_{RWEC}(t)$ dynamically changes during execution of path $p = (b_1, b_2, b_{if}, b_7)$ from example program P. In Figure 3a, which illustrates an execution path that uses no speed scheduling, $C_{RWEC}(t)$ drops at two points: $C_{EC}(b_1)/80$ MHz and $[C_{EC}(b_1) + C_{EC}(b_2) + C_{EC}(b_{if})]/80$ MHz. Because the execution path of Figure 3a uses no speed scheduling, $C_{RWEC}(t)$ decreases at the rate of 80 MHz, resulting in a slack time interval of 1.5625 μ s.

Figure 3b shows the effect of speed scheduling for the same execution path. Because $C_{RWEC}(t)$ drops right after executing b_1 , the speed changes from 80 to 16 MHz, the minimum speed at which the processor can complete the remaining program execution before the deadline. When $C_{RWEC}(t)$ drops right after b_{if} , the speed also changes for the same reason.

Because the proposed RWEC-based intra-task scheduling makes all execution paths complete execution exactly at the deadline, the RWEC-based technique provides two benefits. It

- eliminates slack time, thus increasing energy efficiency; and
- guarantees that the scheduled program always meets the timing constraint.

We call the points in Figure 3 at which $C_{RWEC}(t)$ vertically drops voltage-scaling edges (VSEs), because the speed and voltage can be scaled at these points. The number of cycles reduced at VSEs is C_{saved} .

B-type voltage-scaling edges

We classify VSEs into two categories: B and L. B-type VSEs correspond to the CFG edge between two basic blocks that are part of conditional statements such as the **if** statement. For the **if** statement, WCET is predicted to be the larger of two execution times, one for the **then** path and the other for the **else** path.

Assume that the **if**-statement condition is evaluated in b_{cond} , the **then** path starts at b_{cond} , and the **else** path starts at b_{else} . If the **if** statement is true and the **then** path is shorter than the **else** path, $C_{RWEC}(t)$ is decreased by $C_{RWEC}(b_{\text{else}}) - C_{RWEC}(b_{\text{then}})$. In this case, before the b_{then} block is executed, the speed can be decreased by a ratio of $C_{RWEC}(b_{\text{then}}) / C_{RWEC}(b_{\text{else}})$. We call this ratio a speed update ratio and represent it by $r(b_{\text{cond}} \rightarrow b_{\text{then}})$.

Because the same basic block can be executed at several different clock speeds, rather than associate each VSE with an absolute speed, we associate it with a speed update ratio. For example, in Figure 1, if we were to assign a fixed speed at the VSE between b_{if} and b_7 , 53.3 MHz ($80 \text{ MHz} \times 10/15$) should be assigned so that p_{worst} can complete before the 2- μ s deadline. However, if the executed path up to b_{if} is short—say (b_1, b_2, b_{if}) —the execution will end far earlier than the deadline (resulting in a long slack time interval) if path (b_{if}, b_7) uses a fixed 53.3-MHz clock frequency. By assigning a speed update ratio $r(b_{if} \rightarrow b_7) = 2/3$ to the VSE, we avoid this problem.

In adjusting speed/voltage at VSEs, several instructions—other than `change_f_V(fCLK)`—are required. We denote the number of cycles needed to execute these instructions at a B-type VSE as $C_{VSO,B}$. The total number of overhead cycles $C_{\text{overhead},B}$ for a B-type VSE, therefore, is given by $C_{VTO} + C_{VSO,B}$. The speed update ratio $r(b_i \rightarrow b_j)$ for B-type VSE (b_i, b_j) is

$$r(b_i \rightarrow b_j) = \frac{C_{RWEC}(b_j)}{C_{RWEC}[\text{succ}_{\text{worst}}(b_i)] - C_{\text{overhead},B}} \quad (1)$$

In this equation, $\text{succ}_{\text{worst}}(b_i)$ is basic block b_k , an immediate successor of b_i with the largest $C_{RWEC}(b_k)$ among all b_i 's successors.

If $C_{RWEC}(b_j) \geq C_{RWEC}[\text{succ}_{\text{worst}}(b_i)] - C_{\text{overhead},B}$,

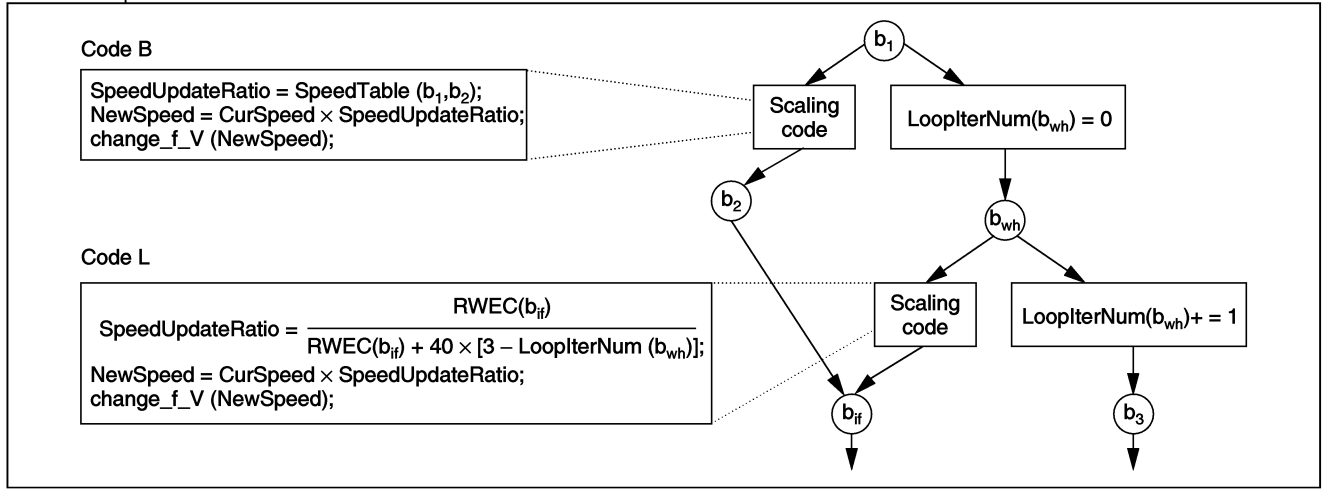


Figure 4. Given the control flow diagram on the right, our technique would insert B- and L-type voltage-scaling code as shown.

that is $r(b_i \rightarrow b_j) \geq 1$, edge (b_i, b_j) is not selected as a VSE. For a VSE between b_i and b_j , a speed update ratio $r(b_i \rightarrow b_j)$ is multiplied with the current speed before b_j starts its execution. That is, $S(b_j)$ is set to the current speed $\times r(b_i \rightarrow b_j)$.

As an example, consider how the speed changes at B-type VSEs as the execution proceeds following the path (b_1, b_2, b_{if}, b_7) of Figure 1, assuming $C_{\text{overhead},B}$ is 0. Just before basic block b_2 executes, RWEC decreases from 150 to 30 cycles, so clock speed changes from 80 to 16 MHz ($80 \text{ MHz} \times 30/150$). The clock speed for basic block b_7 also changes from 16 to 10.7 MHz ($16 \text{ MHz} \times 10/15$) because RWEC changes from 15 to 10 cycles. Figure 4 shows the code generated for a B-type VSE (b_1, b_2) in Figure 1.

L-type voltage-scaling edges

Although our technique predicts WCEC assuming that a loop will execute the user-provided maximum number of iterations, a loop is generally iterated fewer times than the maximum loop bound. In this case, slack time exists and clock speed can be scaled down—a technique we call L-type scaling. L-type VSEs correspond to the loop exit edges in a CFG. In L-type scaling, saved cycles C_{saved} for loop l equal

$$C_{\text{saved}}(l) = C_{\text{WCEC}}(l) \times [N_{\text{worst}}(l) - N_{\text{exec}}(l)] \quad (2)$$

$C_{\text{WCEC}}(l)$ is the worst-case number of execution cycles to execute loop l once, $N_{\text{worst}}(l)$ is the

user-provided maximum number of loops for loop l , and $N_{\text{exec}}(l)$ is the number of actual loop iterations measured at runtime. For L-type scaling, consider the edge (b_{wh}, b_{if}) in Figure 1, which is an example L-type VSE. When we denote the total number of overhead cycles at an L-type VSE as $C_{\text{overhead},L}$, $S(b_{if})$ is updated as

$$S(b_{if}) = S(b_{wh}) \times \frac{C_{\text{RWEC}}(b_{if})}{C_{\text{RWEC}}(b_{if}) + C_{\text{saved}}(l) - C_{\text{overhead},L}} \quad (3)$$

Assuming $S(b_{wh}) = 80 \text{ MHz}$, $N_{\text{exec}}(l) = 1$, and $C_{\text{overhead},L} = 0$, then $S(b_{if})$ decreases to 16 MHz before executing b_{if} .

Unlike for a B-type VSE, calculating the speed update ratio of an L-type VSE requires runtime information such as $N_{\text{exec}}(l)$. The speed update ratio may be larger than 1, depending on the value of $N_{\text{exec}}(l)$ and $C_{\text{overhead},L}$. To avoid this problem, we select a loop exit edge of loop l as an L-type VSE if $C_{\text{WCEC}}(l) > C_{\text{overhead},L}$. Doing so means that if $N_{\text{exec}}(l) < N_{\text{worst}}(l)$, the speed update ratio is always smaller than 1. When $N_{\text{exec}}(l) = N_{\text{worst}}(l)$, the speed is not changed.

Despite the increased code complexity for L-type VSEs, the overall reduction in energy consumption is still significant. This is because slack time arising from the execution of loops is generally far larger than that from conditional statements. For an L-type VSE (b_{wh}, b_{if}) in

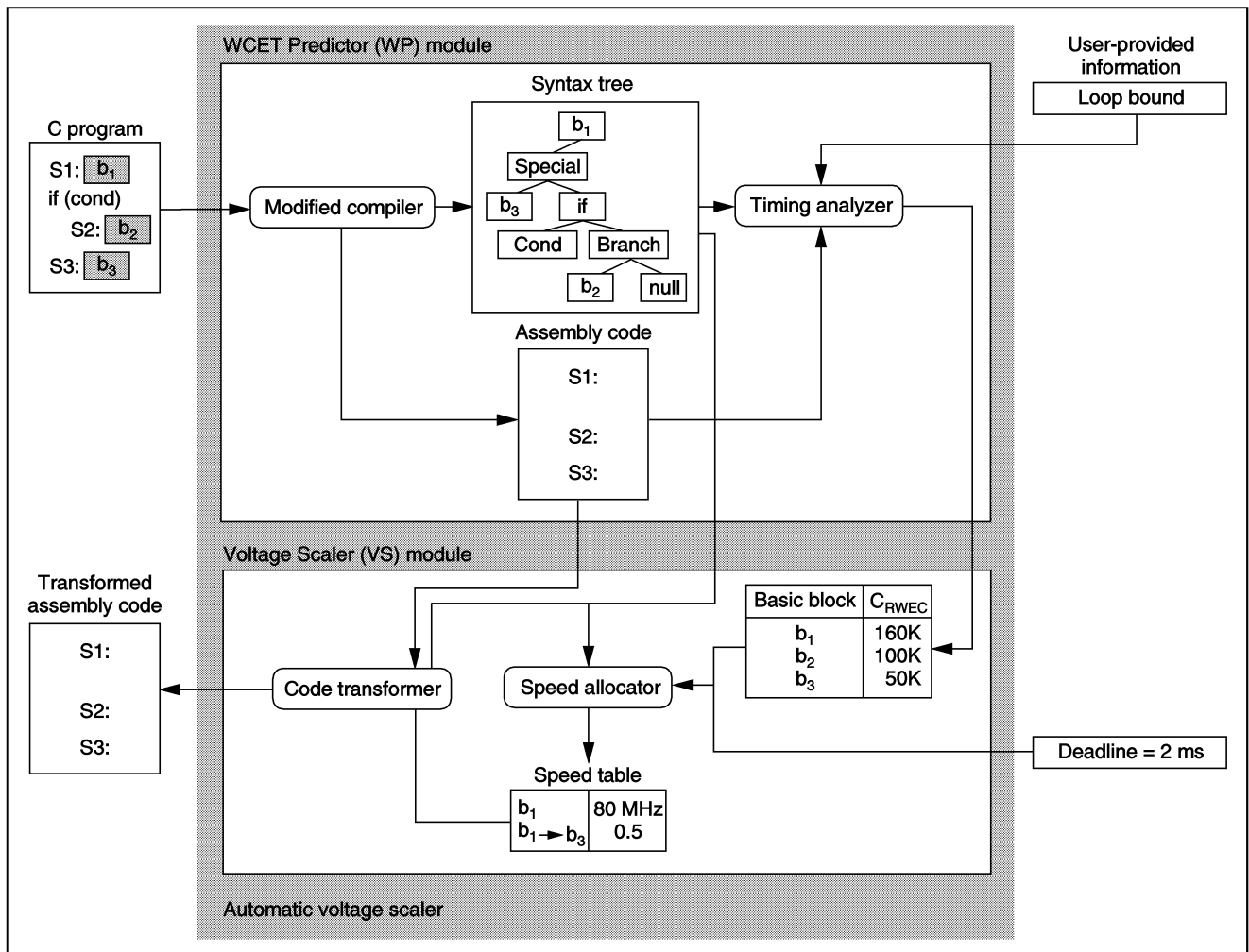


Figure 5. Automatic Voltage Scaler tool's overall structure.

Figure 1, Figure 4 shows the code sequence generated.

Automatic Voltage Scaler

We developed a software tool, the Automatic Voltage Scaler, to automate the development of hard real-time programs on a variable-voltage processor. This tool uses the intra-task scheduling algorithm. AVS takes as an input DVS-unaware program P and its timing requirements. It produces DVS-aware low-energy program P_{DVS} , which satisfies the same timing requirements as P . Converted program P_{DVS} contains voltage-scaling code that handles all the idiosyncrasies of scaling speed/voltage on a variable-voltage processor.

Using AVS, DVS-unaware hard real-time programs can be converted to DVS-aware low-ener-

gy programs in a way completely transparent to software developers. In the current version of AVS, we used the MIPS R3000 instruction set architecture as the target processor.

The WCET Prediction module estimates the $C_{RWEC}(b_i)$ values of all the basic blocks in an input program. To estimate $C_{RWEC}(b_i)$ of given basic block b_i , AVS uses a modified version of a timing tool developed by S.-S. Lim and his colleagues.⁹ Their original timing tool estimates the WCET of an entire program by traversing the program's syntax tree bottom-up and applying the timing formulas of the extended timing schema (ETS). Because AVS uses RWEC from each basic block, we modified the original timing tool accordingly. As shown in Figure 5, the WP module, like the original timing tool,⁹ takes as an input a high-level language program and

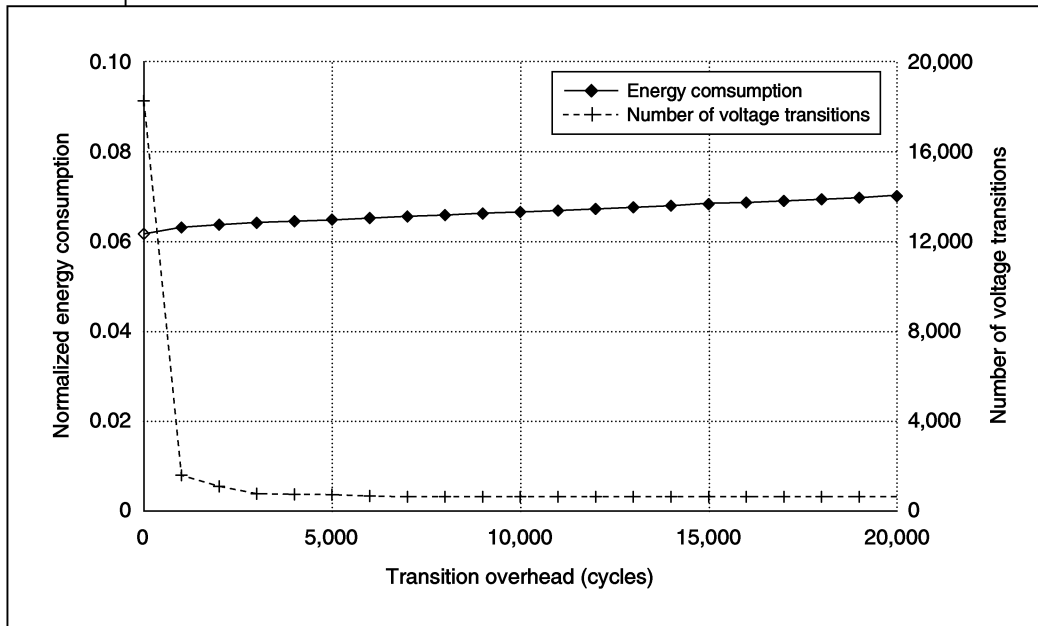


Figure 6. Energy consumption of the AVS-converted MPEG-4 decoder program normalized with respect to a conventional DVS-unaware system.

the user-provided information (such as the loop bound) to estimate $C_{\text{RVEC}}(b_i)$ values.

The Voltage Scaler module identifies VSEs based on $C_{\text{RVEC}}(b_i)$ values within the program syntax tree, assigns proper speeds to these edges, and generates a converted program. The Speed Allocator module in Figure 5 selects VSEs using the VSE selection algorithm and allocates appropriate speed update ratio r to each VSE. For example, in Figure 5, the speed update ratio of 0.5 is assigned to edge (b_1, b_3) .

Experimental results

To evaluate the power reduction performance of AVS, we have experimented with an MPEG-4 video decoder. Because we don't have the proper hardware platform (one with a variable-voltage processor), we developed an energy simulator for the experiment. The energy simulator takes an assembly program and its execution trace as inputs and calculates the total energy consumption of the program's execution.

In this simulation, we assume that both DVS-aware and DVS-unaware systems enter into a power-down mode when the system is idle. We assume the energy consumption of power-down mode is 5% of the normal mode running at maximum clock frequency.¹ Supply voltage for a

given clock frequency comes from $f_{\text{CLK}} = 1/T_D$, which is proportional to $(V_{\text{DD}} - V_T)^\alpha / V_{\text{DD}}^2$ where V_{DD} , V_T , and α are assumed to be 2.5V, 0.5V, and 1.3. Clock/voltage transition overhead C_{VTO} is assumed to be 0 to about 20,000 cycles, corresponding to 0 to about 200 μs of transition time with a 100-MHz clock frequency.

Figure 6 shows the energy consumption of the AVS-converted MPEG-4 decoder program. (In converting the MPEG-4 decoder program, AVS took

less than 100 ms.) Results were normalized over the energy consumption of the original program running on a DVS-unaware system. In Figure 6, the number of voltage transitions represents how many times voltage-scaling code was executed during the program execution. The AVS-converted program consumes less than 7% of the original program's energy consumption.

When $C_{\text{VTO}} < 1,000$ cycles, the number of voltage transitions decreases sharply, but energy consumption does not increase rapidly because the discarded VSEs have little effect on energy reduction. When $C_{\text{VTO}} > 5,000$ cycles, the number of voltage transitions remains nearly constant. The increase in energy consumption is due to the increased overhead cycles.

The number of VSEs—which represents how many copies of voltage-scaling code AVS inserted into the converted program—indicates the increase in code size caused by inserting voltage-scaling code via an inline expansion. For the AVS-converted MPEG-4 decoder, about 10 VSEs are sufficient when $C_{\text{VTO}} > 5,000$ cycles, meaning that insertion of voltage-scaling code hardly increases total code size. This is because only a few voltage-scaling edges are responsible for a large portion of the total power reduction.

BY USING the RWEC information for each basic block, the proposed technique makes it easier to apply intra-task voltage scheduling to DVS-unaware programs. First, it automatically selects appropriate program locations for performing voltage scaling to decrease overall energy consumption. Second, the proposed technique transparently inserts voltage-scaling code to the selected program locations. By automating these two steps, our algorithm makes it possible for programmers to develop DVS-aware programs on a variable-voltage processor without any knowledge of DVS.

Our work can be extended in several directions. We have based the speed assignment on RWEC but most program executions do not take the WCEP at runtime. We are currently devising the improved algorithm where the speed assignment is based on the ACEP.

We believe that both inter-task voltage scheduling and intra-task voltage scheduling have relative advantages and disadvantages over each other. It will be an interesting research topic to compare the two scheduling approaches quantitatively. ■

Acknowledgment

This work was supported in part by the Ministry of Information & Communication of Korea (Support Project of University foundation research < '00 > supervised by IITA). We thank Sung-Soo Lim for providing us with his WCET tool and explaining the tool's internals.

References

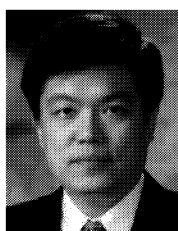
1. T. Burd and R. Broderson, "Processor Design for Portable Systems," *J. VLSI Signal Processing*, vol. 13, no. 2, 1996, pp. 203-222.
2. T. Sakurai and A. Newton, "Alpha-Power Law MOS-FET Model and Its Application to CMOS Inverter Delay and Other Formulas," *IEEE J. Solid State Circuits*, vol. 25, no. 2, Feb. 1990, pp. 584-594.
3. I. Hong et al., "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor," *Proc. 19th IEEE Real-Time Systems Symp. (RTSS 99)*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 178-187.
4. Y. Lee and C.M. Krishna, "Voltage-Clock Scaling for Low Energy Consumption in Real-Time Embedded Systems," *Proc. 6th Int'l Conf. Real-*

Time Computing Systems and Applications (RTCSA 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 272-279.

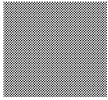
5. Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. 36th Design Automation Conf.*, IEEE Press, Piscataway, N.J., 1999.
6. F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. 36th Ann. Symp. Foundations of Computer Science (FOCS 96)*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 374-382.
7. S. Lee and T. Sakurai, "Runtime Voltage Hopping for Low-Power Real-Time Systems," *Proc. 37th Design Automation Conf.*, IEEE Press, Piscataway, N.J., 2000, pp. 806-809.
8. C.A. Healy, D.B. Whalley, and M.G. Harmon, "Integrating the Timing Analysis of Pipelining and Instruction Caching," *Proc. 16th IEEE Real-Time Systems Symp. (RTSS 95)*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 288-297.
9. S.-S. Lim et al., "An Accurate Worst-Case Timing Analysis for RISC Processors," *IEEE Trans. Software Eng.*, vol. 21, no. 7, July 1999, pp. 593-604.
10. M. Fleischmann, "Crusoe Power Management: Reducing the Operating Power with LongRun," *Proc. Hot Chips 12 Symp.*, Palo Alto, Calif., 2000.



Dongkun Shin is a PhD student at the School of Computer Science and Engineering, Seoul National University. His research interests include low-power systems, computer architecture, and embedded and real-time systems. Shin has a BS in computer science and statistics and an MS in computer science, both from Seoul National University, Korea. He is a member of the ACM.



Jihong Kim is an assistant professor in the School of Computer Science and Engineering, Seoul National University, Korea. His research interests include computer architecture, embedded systems, Java computing, and multimedia and real-time



systems. Kim has a BS in computer science and statistics from Seoul National University, and an MS and PhD in computer science and engineering from the University of Washington. He is a member of the IEEE and ACM.



Seongsoo Lee is a research professor in the Department of Information Electronics, Ewha Woman's University, Korea. His research interests include low-power VLSI systems, dynamic voltage scaling, and VLSI implementation of MPEG-2 and MPEG-4. Lee has a PhD degree in electrical engineering from Seoul National University, Korea. He is a member of the IEEE Circuits and Systems Society.

**Learn
Something
New**

Built-In Self-Test for SOCs
An online tutorial from the
IEEE Computer Society

[http://computer.org/
DT-tutorials/BIST](http://computer.org/DT-tutorials/BIST)

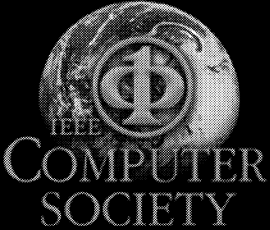
The first step in the
D&T Community Project

**IEEE
Design & Test
of Computers**

you@computer.org
FREE!

All IEEE Computer Society members can obtain a free, portable email ***alias@computer.org***. Select your own user name and initiate your account. The address you choose is yours for as long as you are a member. If you change jobs or Internet service providers, just update your information with us, and the society automatically forwards all your mail.

Sign up today at
http://computer.org



A LOW-POWER IMAGE CONVOLUTION ALGORITHM FOR VARIABLE VOLTAGE PROCESSORS*

Hyugin Kwon

Jihong Kim

School of Computer Science & Engineering
Seoul National University
Seoul, Korea
sonmaps@davinci.snu.ac.kr

School of Computer Science & Engineering
Seoul National University
Seoul, Korea
jihong@davinci.snu.ac.kr

ABSTRACT

We describe a low-power image convolution algorithm for variable voltage processors. The algorithm takes advantages of common properties of popular kernels. Unlike a direct algorithm of convolution operation where the dynamic voltage scaling (DVS) feature of variable voltage processors cannot be used, our algorithm modifies the sequence of computing convolution sums so that DVS can be effectively utilized. Our implementation on Itsy, a DVS research platform from Compaq, shows the energy saving of up to 71% over that of the direct algorithm without any performance degradation.

1. INTRODUCTION

For battery-powered portable imaging systems such as digital cameras and video recorders, low power consumption is a primary design goal because the battery operation time is one of the most important performance measures. Since the energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage V_{DD} , lowering the supply voltage is an effective way of reducing the energy consumption of portable imaging systems. However, lowering the supply voltage also decreases the maximum achievable clock speed; in the CMOS circuit, the delay T_D is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^\alpha$ where V_T is the threshold voltage and α is a velocity saturation index [1].

When a given application's required performance is lower than the system's maximum performance, the clock speed and its corresponding supply voltage can be dynamically controlled to the lowest possible level while meeting the application's deadline constraint. This is the key idea behind the dynamic voltage scaling (DVS) technique [2]. Several recent microprocessors (e.g., Crusoe [3], XScale [4], and AMD PowerNOW! processors [5]) support dynamic voltage scaling in the software level. (We call these processors *variable-voltage processors*.)

Since the key idea of DVS is to reduce the supply voltage when the required performance of a given application is lower than the maximum performance of a system, accurately predicting workload variation is an important requirement in utilizing the DVS feature of variable-voltage processors. For example, if a target application does not exhibit any workload variation, it is impossible to take advantage of the DVS feature for reducing the energy consumption.

A convolution operation, which is widely used in image processing applications, is such a *constant-workload algorithm*, making it very difficult to implement a convolution operation on variable voltage processors in a power-efficient fashion. (With a fast expansion of mobile imaging market, it is expected that many future mobile imaging products will be based on variable-voltage processors for an improved energy efficiency.) In this paper, we describe a low-power image convolution algorithm suitable for variable-voltage processors.

We consider $p \times p$ square kernels. It is convenient to assume that p is an odd number and to denote $q = (p-1)/2$. Let A be an $n \times m$ matrix input image and K be a $p \times p$ matrix kernel, where $n, m > p$. Then for all i, j satisfying $q < i \leq n - q$ and $q < j \leq m - q$, let $A_{i,j}$ be the $p \times p$ square submatrix of A centered in $A[i, j]$. We say that an output $n \times m$ matrix B is a discrete convolution of A with the kernel K :

$$B[i, j] = \sum_{1 \leq k, l \leq p} A_{i,j}[k, l] K[p-k+1, p-l+1]. \quad (1)$$

The boundary elements can be treated as a special case or ignored. The direct algorithm of computing convolution sums would require p^2 multiplications and p^2 additions for each convolved element.

In order to effectively utilize the dynamic voltage scaling feature of variable voltage processors, we modify the sequence of computing convolution sums so that there are large fluctuations on the execution times depending on kernels used. With the modified convolution algorithm, we propose two DVS heuristics for adjusting the supply voltage and clock frequency *under the constraint that the performance of a low-power algorithm is as good as that of the direct algorithm*. Our implementation on Itsy [6], a DVS research platform from Compaq¹, shows the energy saving of up to 71% over that of the direct algorithm without any performance degradation.

The rest of the paper is organized as follows. Section 2 presents a low-power convolution algorithm based on a modified sequence of computing convolution sums. In Section 3, the experimental results on performance/energy measurements are described. Section 4 concludes with a summary.

*This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation.

¹We appreciate a kindly support from Compaq Western Research Laboratory for providing us with two Itsy systems.

2. LOW-POWER CONVOLUTION ALGORITHM

In our low-power convolution algorithm, the key step is to modify the sequence of computing convolution sums so that the workload variation is easily detected in the early stage of computing convolution sums. Figure 1 shows the overall processing steps of the low-power convolution algorithm. First, the kernel elements are analyzed and rearranged, grouping the kernel elements of the same absolute value together and arranging trivial multiplication cases separately. Once a decomposed version of the original kernel is constructed, the sequence of computing convolution sums is modified so that all the kernel elements could be multiplied by the same data element at each step. Based on the characteristics of the decomposed kernel, we compute an appropriate clock frequency and corresponding supply voltage so that the execution time of the low-power convolution algorithm does not exceed that of the direct convolution algorithm.

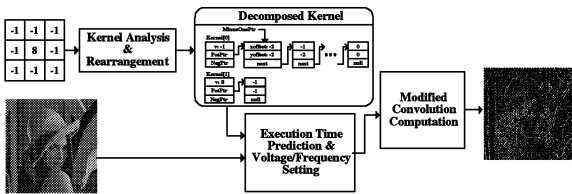


Fig. 1. Overall processing steps of the low-power convolution algorithm.

2.1. Kernel Analysis and Rearrangement

The key observations leading to our low-power convolution algorithm can be summarized by the following three properties from an analysis of commonly used kernels [7]:

Property 1 For most kernels, the number of distinct kernel elements is small.

Property 2 0, 1, and -1 are used frequently.

Property 3 Many kernel elements have the same absolute values.

These properties are useful in eliminating redundant multiplications so that the execution time of convolution operation can vary depending on kernels used. Property 2 is used in reducing the number of multiplications by skipping multiplications between input pixels and kernel elements which have a value of 0, 1, or -1. Properties 1 and 3 are useful as well in reducing the number of multiplications if the sequence of computing convolution sums is appropriately modified as described in the next section.

2.2. Single-Data Multiple-Kernel Convolution Algorithm

Based on the observations summarized in Section 2.1, the single-data multiple-kernel (SDMK) convolution algorithm computes convolution sums differently from the direct implementation in two ways [7]. First, each step computes partial sums for the multiple locations. Second, in each step, all the kernel elements are multiplied by the same input data (i.e., a single data).

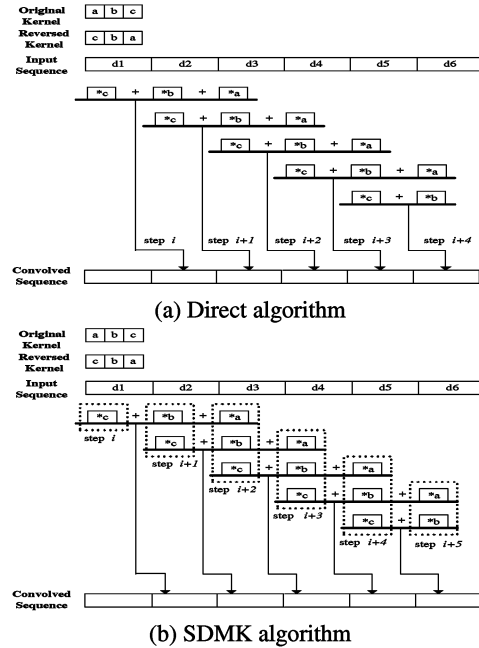


Fig. 2. A comparison of the direct algorithm and SDM algorithm.

Figure 2 illustrates the key differences in computing convolution sums between the direct algorithm and SDM algorithm. Unlike the direct algorithm shown in Figure 2.(a), for each step, the SDM works with a single pixel. For example, in the step (i+2), d3 is multiplied to all the kernel elements and the computed result is accumulated to three partial sums, respectively. If all three kernel elements had the same absolute values, a single multiplication is enough, saving two multiplications from the direct algorithm.

In the SDM algorithm, the number of multiplications per convolved element is reduced to $N_{abs-distinct}$, the number of kernel elements having distinct absolute values excluding 0, 1 and -1 from the total number of kernel elements, N_{total} . The number of additions per convolved element is also decreased by the number of zero elements, N_{zero} , in kernel elements. For example, in the example kernel shown in Figure 1, a single multiplication per convolved element is sufficient.

2.3. Execution Time Prediction and Speed Setting

Since the execution time of the SDM algorithm varies depending on kernels used, the proposed low-power convolution algorithm predicts the expected workload before computing convolution sums. If the estimated workload is less than one required by the direct algorithm, the supply voltage/clock frequency is lowered so that the resulting execution consumes less energy. We lower the supply voltage to the extent that the execution time of the low-power algorithm is less than or equal to that of the direct algorithm.

We use two heuristics in predicting the execution time of the SDM algorithm: one based on a kernel analysis and the other based on the dynamic measurement of the execution time for a small portion of actual execution.

The static prediction method, $SDMK_{static}$, is based on the number of required arithmetic operations obtained in the kernel analysis step (see Section 2.1.). Given an $n \times m$ input image and a $p \times p$ kernel, let C_{direct} and C_{sdmk} be the number of arithmetic operations required for the direct implementation and the $SDMK$ implementation, respectively. Then, C_{direct} and C_{sdmk} are given as follows:

$$C_{direct} = (n \times m) \times p^2 \times (N_{mul} + N_{add}) \quad (2)$$

$$C_{sdmk} = (n \times m) \times N_{abs-distinct} \times N_{mul} + (n \times m) \times (p^2 - N_{zero}) \times N_{add} \quad (3)$$

where N_{mul} and N_{add} are the execution latencies (in cycles) of multiplication and addition operations, respectively.

Once C_{sdmk} is computed, we calculate the new clock frequency f_{sdmk} as follows:

$$f_{sdmk} = \left(\frac{C_{sdmk}}{C_{direct}} \right) \cdot f_{max} \quad (4)$$

where f_{max} is the maximum clock frequency of a target system. The corresponding supply voltage V_{new} can be determined by the voltage-frequency formula:

$$f = \frac{(V_{new} - V_T)^\alpha}{V_{new}} \quad (5)$$

where V_{new} is a supply voltage.

The dynamic prediction method, $SDMK_{dynamic}$, uses actual measurements instead of the number of arithmetic operations in estimating the required workload. Convolution operations are performed for the beginning $n \times S$ convolved outputs and the execution time T_S for $n \times S$ outputs is measured during run time. An execution time estimate T_{sdmk} for an $n \times m$ image is given by:

$$T_{sdmk} = (T_S - T_{kernel}) \cdot \left\lceil \frac{n \times m}{n \times S} \right\rceil \quad (6)$$

where T_{kernel} is the execution time taken by the kernel analysis and rearrangement step. In all our experiments, S was set to 3. Once T_{sdmk} is computed, we can determine the execution speed from a pre-constructed speed table. The speed table specifies how to adjust the clock frequency using the ratio of T_{sdmk} to T_{direct} (where T_{direct} is the execution time when the direct algorithm is used.)

3. EXPERIMENTAL RESULTS

3.1. Experimental platform: Itsy Pocket Computer

We use the Itsy pocket computer v2.6 from Compaq [6] as our experimental platform. Figure 3 shows the experimental setup with Itsy. Itsy v2.6 is equipped with a StrongARM SA1100 processor as a main processor. The SA1100 processor uses the phase-locked loop (PLL), allowing to change the CPU core frequency to one of 11 levels between 59.0 MHz and 226.4 MHz. Furthermore, Itsy v2.6 has a programmable core voltage regulator; supply voltage can scale to one of 30 levels between 1.00 V and 2.00 V.

Itsy runs the Linux operating system (ver. 2.0.30) with a kernel support for dynamic voltage scaling. Applications can access the DVS function by the *iocli* system call to the `"/dev/clkspeed"` device file.

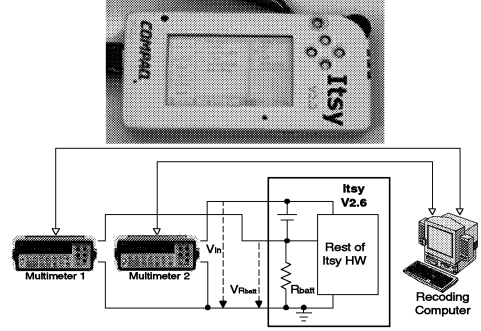


Fig. 3. Experimental setup with Itsy.

3.2. Results

We have implemented three convolution algorithms, the direct algorithm, the $SDMK_{static}$ -based low-power algorithm, and the $SDMK_{dynamic}$ -based low-power algorithm on Itsy using the C programming language. As shown in Figure 3, we have measured the voltage drops in the current-sense resistors embedded in the Itsy system. Using Itsy v2.6, we can measure the power consumed in the processor core only or can measure the power consumption of the whole system. The energy consumption is computed by multiplying the execution time by the average power consumption measured.

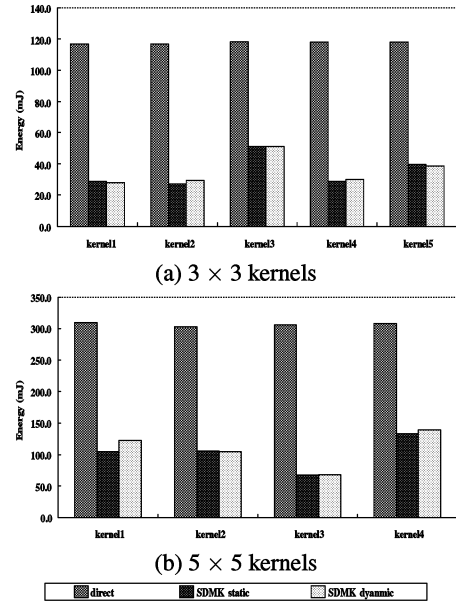


Fig. 4. Energy consumption in the StrongARM processor core.

Figures 4 and 5 show the experimental results of three algorithms. Since the performance and energy consumption of three algorithms do not depend on pixel values, a single 256×256 image was used as an input image for all the experiments. For the direct algorithm, we used the clock frequency of 206.4 MHz and the supply voltage of 1.55 V. Figure 4 compares the energy consumed

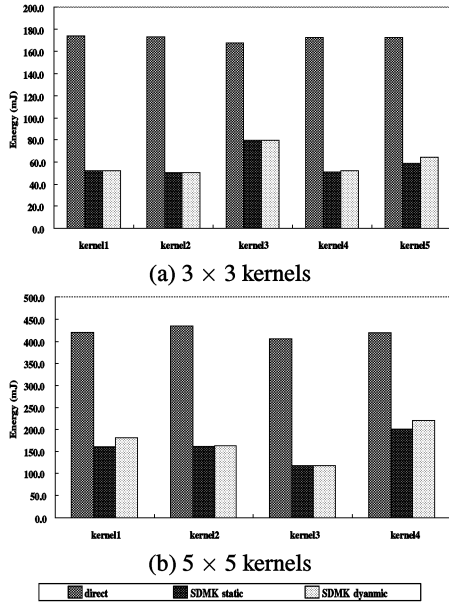


Fig. 5. Energy consumption in the whole Itsy system.

only in the processor core while Figure 5 compares the energy consumption of the whole Itsy system. As shown in Figure 4, the core power consumption of the proposed low-power algorithms is reduced by up to 76.7% over the direct algorithm. Even for the whole Itsy system, the best reduction ratio of 71.1% is achieved. On average, the low-power algorithms reduce about 67.6% and 62.8% of energy consumption for the core processor only and the whole Itsy system, respectively.

In order to understand the high energy efficiency of the proposed low-power convolution algorithms, it is useful to remind that the energy consumption E of a program P is proportional to the product of N_{cycle} and V_{DD}^2 where N_{cycle} is the number of cycles executed for P .² In the proposed algorithms, both N_{cycle} and V_{DD} are reduced, resulting in high energy savings in the processor core as shown in Figure 4. Furthermore, as discussed in [8], lowering supply voltage in Itsy also decreases the energy consumption of non-CPU parts (e.g., LCD) as well (although, in theory, DVS should not affect these parts.). This additional savings contributed a higher-than-expected energy saving ratio in the whole Itsy system as shown in Figure 5.

Figure 6 compares the execution time variations by the proposed algorithms. For most kernels tested, the execution times of the proposed algorithms are less than that of the direct algorithm without violating the timing constraint. As shown in Figure 6, $SDMK_{dynamic}$ always takes less times than the direct algorithm.

4. CONCLUSION

We have described two low-power convolution algorithms suitable for variable voltage processors. Unlike the direct algorithm, the proposed algorithms intelligently identify and predict the workload variations by the modified convolution algorithm. From the

²Note that lowering the clock frequency does not change N_{cycle} . It increases the clock cycle time.

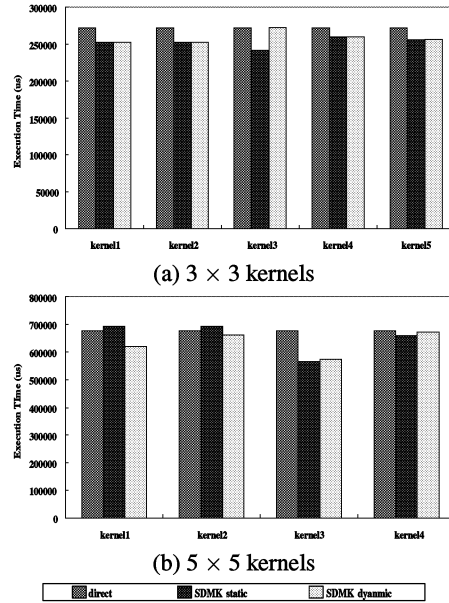


Fig. 6. Execution time comparisons of three approaches.

actual measurements on the Itsy system, we have demonstrated that the proposed algorithms achieve an energy reduction of up to 71% over that of the direct algorithm without any performance degradation.

5. REFERENCES

- [1] T. Sakurai and A. Newton, "Alpha-Power Law MOSEFT Model and Its Applications to CMOS Inverter Delay and Other Formulas," *IEEE Journal of Solid State Circuits*, vol. 25, no. 2, pp. 584–594, 1990.
- [2] T. Burd and R. Broderson, "Processor Design for Portable Systems," *Journal of VLSI Signal Processing*, vol. 13, no. 2, pp. 203–222, August 1996.
- [3] L. Geppert and T. Perry, "Transmeta's Magic Show," *IEEE Spectrum*, vol. 37, pp. 22–32, May 2000.
- [4] Intel Inc, "Intel XScale Technology," <http://www.intel.com/design/intelxscale>.
- [5] AMD Inc, "AMD PowerNow!™ Technology Platform Design Guide for Embedded Processors," <http://www.amd.com/epd/processors>.
- [6] R. Hamburg, D. Wallach, M. Viredaz, L. Brakmo, C. Waldspurger, J. Bartlett, T. Mann, and K. Farkas, "Itsy: Stretching the Bounds of Mobile Computing," *IEEE Computer*, vol. 34, no. 4, pp. 28–36, April 2001.
- [7] J. Kim and Y. Kim, "Efficient 2-D Convolution Algorithm with the Single-Data Multiple Kernel Approach," *Graphical Models and Image Processing*, vol. 57, no. 2, pp. 175–182, March 1995.
- [8] M. Viredaz and D. Wallach, "Power Evaluation of a Handheld Computer: A Case Study," Tech. Rep. 2001/1, Comapp Western Research Laboratory, May 2001.

On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems

HAN-SAEM YUN and JIHONG KIM
Seoul National University

We address the problem of energy-optimal voltage scheduling for fixed-priority hard real-time systems, on which we present a complete treatment both theoretically and practically. Although most practical real-time systems are based on fixed-priority scheduling, there have been few research results known on the energy-optimal fixed-priority scheduling problem. First, we prove that the problem is NP-hard. Then, we present a fully polynomial time approximation scheme (FPTAS) for the problem. For any $\epsilon > 0$, the proposed approximation scheme computes a voltage schedule whose energy consumption is at most $(1 + \epsilon)$ times that of the optimal voltage schedule. Furthermore, the running time of the proposed approximation scheme is bounded by a polynomial function of the number of input jobs and $1/\epsilon$. Given the NP-hardness of the problem, the proposed approximation scheme is practically the best solution because it can compute a near-optimal voltage schedule (that is provably arbitrarily close to the optimal schedule) in polynomial time. Experimental results show that the approximation scheme finds more efficient (almost optimal) voltage schedules faster than the best existing heuristic.

Categories and Subject Descriptors: C.3 [Special-Purpose and Application-Based Systems]: Real-Time and Embedded Systems; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*

General Terms: Algorithms

Additional Key Words and Phrases: Fixed-priority scheduling, real-time systems, approximation algorithms, fully polynomial time approximation scheme, variable voltage processor, dynamic voltage scaling

1. INTRODUCTION

Energy consumption is one of the most important design constraints in designing battery-operated embedded systems such as personal digital assistants, digital cellular phones, and mobile videophones. For such systems, the energy consumption is a critical design factor because the battery operation time is a primary performance measure. The dynamic energy consumption E , which dominates the total energy consumption of CMOS circuits, is given by $E \propto C_L \cdot N_{\text{cycle}} \cdot V_{\text{DD}}^2$, where C_L is the load capacitance, N_{cycle} is the number of executed cycles, and V_{DD} is the supply voltage. Because the dynamic energy consumption E is quadratically dependent on the supply voltage V_{DD} , lowering V_{DD} is an effective technique in reducing the energy consumption. However, lowering the supply voltage

This work was supported by grant No. R01-2001-00360 from the Korea Science and Engineering Foundation. Authors' address: H.-S. Yun and J. Kim, School of Computer Science and Engineering, Seoul National University, Shilim-dong, Kwanak-ku, Seoul, 151-742, Korea; email: {hsyun, jihong}@davinci.snu.ac.kr.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20 ACM 0000-0000/20/0000-0001 \$5.00

also decreases the clock speed, because the circuit delay T_D of CMOS circuits is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^\alpha$ [Sakurai and Newton 1990], where V_T is the threshold voltage and α is a technology-dependent constant.

When a given job does not require the maximum performance of a VLSI system, the clock speed (and its corresponding supply voltage) can be dynamically adjusted to the lowest possible level that still satisfies the job's required performance. This is the key principle of the dynamic voltage scaling (DVS) technique. With a recent explosive growth of the portable embedded system market, several commercial variable-voltage processors were developed (e.g., Intel's *Xscale*, AMD's *K6-2+* and Transmeta's *Crusoe* processors). Targeting these processors, various DVS algorithms [Yao et al. 1995; Hong et al. 1998; Pillai and Shin 2001; Aydin et al. 2001; Kim et al. 2002; Shin and Choi 1999; Gruian 2001; Shin et al. 2000; Quan and Hu 2001; 2002] have been proposed, especially for embedded hard real-time systems.

For hard real-time systems, the goal of voltage scheduling algorithms is to find an *energy-efficient* voltage schedule with all the stringent timing constraints satisfied. A voltage schedule is a function that associates each time unit with a voltage level (i.e., a clock frequency).¹ In this paper, we consider *fixed-priority* real-time jobs running on variable voltage processors.

1.1 Previous Work

Previous investigations on the voltage scheduling problem have focused mainly on real-time jobs running under dynamic-priority scheduling algorithms such as the EDF (earliest-deadline-first) algorithm [Hong et al. 1998; Pillai and Shin 2001; Aydin et al. 2001; Kim et al. 2002]. For example, the problem of energy-optimal EDF scheduling has been well understood. For EDF job sets, the algorithm by Yao *et al.* [Yao et al. 1995] computes the energy-optimal voltage schedules in polynomial time. Although the EDF scheduling policy makes the voltage scheduling problem easier to solve, fixed-priority scheduling algorithms such as the RM (rate monotonic) algorithm are more commonly used in practical real-time systems due to their low overhead and predictability [Liu 2000].

Although there exist several voltage scheduling techniques proposed for fixed-priority real-time tasks (e.g., on-line scheduling algorithms [Shin and Choi 1999; Gruian 2001; Pillai and Shin 2001] and off-line scheduling algorithms [Shin et al. 2000; Gruian 2001; Quan and Hu 2001; 2002]), there have been few research results on the *optimal* voltage scheduling problem for fixed-priority hard real-time systems; neither a polynomial-time optimal voltage scheduling algorithm nor the computational complexity of the problem is known.

Up to now, the only significant research result on the optimality issue of fixed-priority voltage scheduling is the one presented by Quan *et al.* [Quan and Hu 2002], where energy-optimal voltage schedules for fixed-priority jobs are found by an *exhaustive* algorithm. However, Quan *et al.* did not justify their exhaustive approach. If they had presented the computational complexity of the voltage scheduling problem, their result would have been much more significant. Since the worst-case complexity of Quan's algorithm is of higher order than $O(N!)$ where N is the number of jobs, the algorithm is practically unusable for most real-time applications.

Quan *et al.* also proposed a polynomial-time voltage scheduling algorithm for fixed-

¹Throughout the remainder of the paper, we use the term voltage scheduling instead of DVS.

priority hard real-time systems [Quan and Hu 2001], which is the best known polynomial-time heuristic for the problem. Although efficient, being a heuristic, this algorithm cannot guarantee the quality of the voltage schedule computed.

1.2 Contributions

In this paper, we give a complete treatment on the optimal voltage scheduling problem for fixed-priority hard real-time systems. As with the work of Quan *et al.* [Quan and Hu 2002; 2001], we assume that the timing parameters of each job is known a priori. Our problem is identical to the one solved by Yao *et al.* [Yao *et al.* 1995] except that the priority assignment is changed from the dynamic EDF assignment to the fixed assignment. As illustrated by Quan *et al.* [Quan and Hu 2001], the voltage scheduling problem for fixed-priority tasks is more difficult to solve because the preemption relationship among the tasks is much more complex to analyze.

First, we prove that the optimal voltage scheduling problem is NP-hard, which implies that no optimal polynomial-time algorithm is likely to exist. Second, we present a *fully polynomial time approximation scheme* for the problem. A fully polynomial time approximation scheme (FPTAS) is an approximation algorithm that takes any $\epsilon (> 0)$ as an additional input and returns a solution whose cost is at most a factor of $(1 + \epsilon)$ away from the cost of the optimal solution with the running time bounded by a polynomial both in the size of the input instance and in $1/\epsilon$ [Woeginger 1999]. Given the NP-hardness of the problem, the proposed approximation scheme is practically the best solution. The proposed approximation scheme computes a near-optimal voltage schedule in polynomial time. By changing ϵ , the approximation scheme can find a voltage schedule that is provably arbitrarily close to the optimal solution.

The rest of the paper is organized as follows. In Section 2, we formulate the problem and characterize feasible voltage schedules. We describe important properties of an energy-optimal voltage schedule in Section 3, which provide a basis of later proofs. In Section 4, we present the intractability result of the problem including its NP-hardness. The FPTAS for the problem is presented in Section 5. Experimental results are given in Section 6 and we conclude with a summary and directions for future work in Section 7.

2. PROBLEM FORMULATION

We consider a set $\mathcal{J} = \{J_1, J_2, \dots, J_{|\mathcal{J}|}\}$ of priority-ordered jobs with J_1 being the job with the highest priority. A job $J \in \mathcal{J}$ is associated with the following timing parameters, which are assumed to be known off-line:

- r_J : the release time of J .
- d_J : the deadline of J .
- c_J : the number of execution cycles required for J .

We use p_J to denote the priority of the job J . We assume that J has a higher priority than J' if $p_J < p_{J'}$. In the rest of the paper, we use i instead of J_i as a subscript of timing parameters when no confusion arises. (e.g., r_i, d_i and c_i stand for r_{J_i}, d_{J_i} and c_{J_i} .) Note that our job model can be directly applicable to a periodic real-time system by considering all the task instances within a hyperperiod of periodic tasks.

Since there is a one-to-one correspondence between the processor speed and the supply voltage, we use $\mathcal{S}(t)$, the processor speed, to denote the voltage schedule in the rest of the

paper. Given a voltage schedule, the job executed at time t can be uniquely determined and is denoted by $job(\mathcal{J}, \mathcal{S}, t)$. A voltage schedule $\mathcal{S}(t)$ is said to be *feasible* if $\mathcal{S}(t)$ gives each job the required number of cycles between its release time and deadline. (An exact characterization of a feasible voltage schedule is given in Section 2.1.)

As with other related work [Yao et al. 1995; Quan and Hu 2001; 2002], we assume that the processor speed can be varied continuously with a negligible overhead both in time and power. Furthermore, we model that the power P , energy consumed per unit time, is a convex function of the processor speed; given a voltage schedule $\mathcal{S}(t)$, the power can be written as a function of time by $P(\mathcal{S}(t))$. For simplicity, we assume that all the jobs have the same switching activity and that P is dependent only on the processor speed.²

The goal of the voltage scheduling problem is, therefore, to find a feasible schedule $\mathcal{S}(t)$ that minimizes

$$E(\mathcal{S}) = \int_{t_s}^{t_f} P(\mathcal{S}(t)) dt \quad (1)$$

where t_s and t_f are the lower and upper limits of release times and deadlines of the jobs in \mathcal{J} , respectively. For the rest of this paper, the energy-optimal voltage schedule of a job set \mathcal{J} is denoted by $\mathcal{S}_{opt}^{\mathcal{J}}$.

2.1 Feasibility Analysis

In this section, we derive a necessary and sufficient condition for a voltage schedule to be feasible, which will provide a basis for the proofs in Section 3. We first introduce some useful notations and definitions.

$W(\mathcal{S}, [t_1, t_2])$ is used to denote the number of cycles executed under a voltage schedule $\mathcal{S}(t)$ from t_1 to t_2 , i.e., $W(\mathcal{S}, [t_1, t_2]) = \int_{t_1}^{t_2} \mathcal{S}(t) dt$. Among $W(\mathcal{S}, [t_1, t_2])$ cycles, $W_i(\mathcal{S}, [t_1, t_2])$ denotes the number of cycles between t_1 and t_2 used for executing a set of jobs J_1, J_2, \dots, J_i whose priorities are higher than or equal to p_{J_i} . $R_{\mathcal{J}}$ and $D_{\mathcal{J}}$ represent the sets of release times and deadlines of the jobs in \mathcal{J} , respectively, i.e., $R_{\mathcal{J}} = \{r_J | J \in \mathcal{J}\}$ and $D_{\mathcal{J}} = \{d_J | J \in \mathcal{J}\}$. $T_{\mathcal{J}}$ denotes the union of $R_{\mathcal{J}}$ and $D_{\mathcal{J}}$, i.e., $T_{\mathcal{J}} = R_{\mathcal{J}} \cup D_{\mathcal{J}}$. Given a job set $\mathcal{J}' \subseteq \mathcal{J}$, $C(\mathcal{J}')$ represents the total workload of jobs in \mathcal{J}' , i.e., $C(\mathcal{J}') = \sum_{J \in \mathcal{J}'} c_J$. Furthermore, $\mathbf{I}_{\mathcal{J}'}$ represents the minimum interval that includes the execution intervals of jobs in \mathcal{J}' , i.e., $\mathbf{I}_{\mathcal{J}'} = [\min R_{\mathcal{J}'}, \max D_{\mathcal{J}'}]$. $\mathcal{T}^{\mathcal{J}}$ represents the cartesian product of $[r_{J_i}, d_{J_i}]$'s, for $1 \leq i \leq |\mathcal{J}|$, i.e., $\mathcal{T}^{\mathcal{J}} = [r_{J_1}, d_{J_1}] \times [r_{J_2}, d_{J_2}] \times \dots \times [r_{J_{|\mathcal{J}|}}, d_{J_{|\mathcal{J}|}}]$. Given voltage schedules $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$ such that

$$\mathcal{S}_i(t) = 0 \text{ for all } t \notin [\alpha_i, \beta_i] \text{ for all } 1 \leq i \leq n \text{ and } \beta_i \leq \alpha_{i+1} \text{ for all } 1 \leq i < n ,$$

the concatenation of $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$ is

$$\oplus_{i=1}^n \mathcal{S}_i = \mathcal{S}_1 \oplus \mathcal{S}_2 \oplus \dots \oplus \mathcal{S}_n \stackrel{def}{=} \sum_{i=1}^n \mathcal{S}_i(t) .$$

Since jobs should be released before they can be processed, we assume that a voltage schedule \mathcal{S} always satisfies the constraint that for any $t > 0$, $W(\mathcal{S}, [0, t]) \leq C(\{J | r_J < t\})$.

The condition for a voltage schedule $\mathcal{S}(t)$ to be feasible can be expressed as follows:

²See [Yun and Kim 2002] for a more general heterogeneous case where the switching activity of each job is different.

Condition I (Feasibility Condition).

There exists a $|\mathcal{J}|$ -tuple $(f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$ such that

$$\forall 1 \leq i \leq |\mathcal{J}| \quad \forall r \in \{t \mid t \in R_{\mathcal{J}} \wedge t < f_{J_i}\} \\ W(\mathcal{S}, [r, f_{J_i}]) \geq C(\{J \mid p_J \leq p_{J_i} \wedge r_J \in [r, f_{J_i}]\}) . \quad (2)$$

For a $|\mathcal{J}|$ -tuple $(f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, f_{J_i} can be considered as a modified deadline of J_i , which is equal to or precedes the original deadline d_{J_i} . (The meaning of the $|\mathcal{J}|$ -tuple is further clarified in Section 3.) If $\mathcal{S}(t)$ satisfies Condition I for a given $|\mathcal{J}|$ -tuple $(f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, J_i completes its execution by f_{J_i} for all $1 \leq i \leq |\mathcal{J}|$. Such $|\mathcal{J}|$ -tuples are said to be *valid* with respect to $\langle \mathcal{J}, \mathcal{S}(t) \rangle$. Theorem 2.1 gives a proof for the feasibility condition.

THEOREM 2.1. *Condition I is a necessary and sufficient condition for $\mathcal{S}(t)$ to be feasible.*

PROOF. For the necessary part, suppose that $\mathcal{S}(t)$ is feasible, i.e., J_i completes its execution at $f_{J_i} \in (r_{J_i}, d_{J_i}]$ for all $1 \leq i \leq |\mathcal{J}|$. Then, for any $r \in R_{\mathcal{J}}$ such that $r < f_{J_i}$, all the higher priority jobs whose release times are within $[r, f_{J_i})$ complete their executions by f_{J_i} . So, the total amount of work that should be done within $[r, f_{J_i}]$ must be greater than or equal to the sum of workload of the jobs. Thus, we have for all $1 \leq i \leq |\mathcal{J}|$:

$$W(\mathcal{S}, [r, f_{J_i}]) \geq C(\{J \mid p_J \leq p_{J_i} \wedge r_J \in [r, f_{J_i}]\}) .$$

For the sufficient part, assume that Condition I is satisfied for a $|\mathcal{J}|$ -tuple $(f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}})$. By induction on i , we prove that J_i is given its required execution cycles c_{J_i} within $[r_{J_i}, f_{J_i}]$ for all $1 \leq i \leq |\mathcal{J}|$. The base case holds trivially.

For the induction step, assume that the proposition holds for all $k = 1, 2, \dots, i-1$. Let $r < r_{J_i}$ be the earliest time point in $R_{\mathcal{J}}$ such that no lower priority jobs (i.e., J_k 's for $k > i$) are executed within $[r, r_{J_i}]$, i.e., $W(\mathcal{S}, [r, r_{J_i}]) = W_{i-1}(\mathcal{S}, [r, r_{J_i}])$. If such r does not exist, r is set to r_{J_i} . Then, a higher priority job J' (i.e., J_l 's for $l < i$) released before r (i.e., $r_{J'} < r$) must complete its execution before r ; otherwise, since any lower priority jobs cannot be executed within $[r_{J'}, r]$, we have

$$W(\mathcal{S}, [r_{J'}, r_{J_i}]) = W(\mathcal{S}, [r_{J'}, r]) + W(\mathcal{S}, [r, r_{J_i}]) \\ = W_{i-1}(\mathcal{S}, [r_{J'}, r]) + W_{i-1}(\mathcal{S}, [r, r_{J_i}]) = W_{i-1}(\mathcal{S}, [r_{J'}, r_{J_i}]) ,$$

which contradicts the definition of r . Since only higher priority jobs (i.e., J_l 's for $l < i$) are executed within $[r, r_{J_i}]$, the amount of remaining workload of the higher priority jobs (which are released within $[r, r_{J_i})$) at time r_{J_i} is $C(\{J_k \mid 1 \leq k < i \wedge r_{J_k} \in [r, r_{J_i}]\}) - W(\mathcal{S}, [r, r_{J_i}])$. So, we have

$$W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}]) \leq C(\{J_k \mid 1 \leq k < i \wedge r_{J_k} \in [r, r_{J_i}]\}) - W(\mathcal{S}, [r, r_{J_i}]) \\ + C(\{J_k \mid 1 \leq k < i \wedge r_{J_k} \in [r_{J_i}, f_{J_i}]\}) \\ = C(\{J_k \mid 1 \leq k < i \wedge r_{J_k} \in [r, f_{J_i}]\}) - W(\mathcal{S}, [r, r_{J_i}]) . \quad (3)$$

To complete the induction, we only need to show that $W(\mathcal{S}, [r_{J_i}, f_{J_i}]) - W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}])$ is not smaller than c_{J_i} . (Note that J_i preempts any lower priority jobs.) From (3) and the

assumption that Condition I is satisfied, we have

$$\begin{aligned}
& W(\mathcal{S}, [r_{J_i}, f_{J_i}]) - W_{i-1}(\mathcal{S}, [r_{J_i}, f_{J_i}]) \\
& \geq W(\mathcal{S}, [r, f_{J_i}]) - C(\{J_k | 1 \leq k < i \wedge r_{J_k} \in [r, f_{J_i}]\}) \quad (\text{From (3).}) \\
& \geq C(\{J_k | 1 \leq k \leq i \wedge r_{J_k} \in [r, f_{J_i}]\}) - C(\{J_k | 1 \leq k < i \wedge r_{J_k} \in [r, f_{J_i}]\}) \quad (\text{From (2).}) \\
& = C(\{J_i\}) = c_{J_i} .
\end{aligned}$$

□

A job set \mathcal{J} is said to be an EDF job set if for any $J, J' \in \mathcal{J}$ (where $p_J < p_{J'}$), $d_J \leq d_{J'}$ or $d_{J'} \leq r_J$. When the priority assignment follows the EDF policy, we can prove that Condition I is simplified as follows:

Condition II (EDF Feasibility Condition).

For any $r \in R_{\mathcal{J}}$ and $d \in D_{\mathcal{J}}$ (where $r < d$),
 $W(\mathcal{S}, [r, d]) \geq C(\{J | [r_J, d_J] \subseteq [r, d]\})$.

LEMMA 2.2. *Given an EDF job set \mathcal{J} , a voltage schedule $\mathcal{S}(t)$ of \mathcal{J} is feasible if and only if Condition II is satisfied.*

PROOF. Consider a new job set $\mathcal{J}' = \{J'_1, J'_2, \dots, J'_{|\mathcal{J}|}\}$ where $r_{J'_i} = W(\mathcal{S}, [0, r_{J_i}])$, $d_{J'_i} = W(\mathcal{S}, [0, d_{J_i}])$, $c_{J'_i} = c_{J_i}$ and $p_{J'_i} = p_{J_i}$ for all $1 \leq i \leq |\mathcal{J}'|$. Because $W(\mathcal{S}, [0, t])$ is a monotonically increasing function of t , \mathcal{J}' is also an EDF job set (i.e., for any $J'_i, J'_k \in \mathcal{J}'$ where $i < k$, $d_{J'_i} \leq d_{J'_k}$ or $d_{J'_k} \leq r_{J'_i}$). Let $\mathcal{S}'(t) = 1$ ($\forall t > 0$) be the voltage schedule of \mathcal{J}' . Then, we can easily verify that the index of the job $job(\mathcal{J}, \mathcal{S}, t)$ is the same as that of $job(\mathcal{J}', \mathcal{S}', W(\mathcal{S}, [0, t]))$. Therefore, $J_i \in \mathcal{J}$ finishes its execution by its deadline d_{J_i} under $\mathcal{S}(t)$ if and only if its corresponding job $J'_i \in \mathcal{J}'$ finishes its execution by $d_{J'_i}$ ($= W(\mathcal{S}, [0, d_{J_i}])$) under \mathcal{S}' .

It is well known that all the jobs in an EDF job set meet their deadlines under a constant speed if and only if the utilization ratio for any time interval is less than or equal to 1 [Liu 2000]. That is, \mathcal{S}' is a feasible voltage schedule of \mathcal{J}' if and only if the following is satisfied:

$$\begin{aligned}
& \text{For any } r' \in R_{\mathcal{J}'} \text{ and } d' \in D_{\mathcal{J}'} \text{ (where } r' < d') \text{ ,} \\
& C(\{J | J \in \mathcal{J}' \wedge [r_J, d_J] \subseteq [r', d']\}) \leq d' - r' . \quad (4)
\end{aligned}$$

Since (4) is equivalent to Condition II, Condition II is a necessary and sufficient condition for $\mathcal{S}(t)$ to be a feasible voltage schedule of \mathcal{J} . □

As shown in Conditions I and II, the complexity of fixed-priority voltage scheduling mainly comes from the inherent exhaustiveness in finding a valid $|\mathcal{J}'|$ -tuple. In the EDF scheduling algorithm, it is sufficient for a single $|\mathcal{J}'|$ -tuple of the original deadlines to be checked if it satisfies Condition II.

3. SOME PROPERTIES OF OPTIMAL SCHEDULES

In this section, we explain several properties for a feasible voltage schedule to be an energy-optimal schedule. These properties provide a key insight in devising a fast approximation

algorithm described in Section 5. The first property, which was proven by Quan *et al.* [Quan and Hu 2001] is that an energy-optimal voltage schedule should be a piecewise-constant function.

The existing optimal voltage scheduling algorithm by Quan *et al.* is based on an observation that if a given job set satisfies the requirement of an EDF job set, the optimal voltage schedule can be easily computed by Yao’s “peak-power greedy” algorithm [Yao et al. 1995]. Simply applying Yao’s algorithm to a fixed-priority job set may cause some jobs to miss their deadlines. However, if the deadlines of the jobs are appropriately modified before scheduling, Yao’s algorithm can yield a feasible optimal schedule as shown in [Quan and Hu 2002]. The efficiency of an optimal voltage scheduling algorithm is, therefore, dependent on how efficiently the job set is modified to be an EDF job set. To give a better insight into our approach for solving the voltage scheduling problem, we derive an equivalent result to Quan *et al.* [Quan and Hu 2002] using Conditions I and II.

3.1 Properties on $|\mathcal{J}|$ -Tuples

Given a $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$, $\mathcal{J}^{\mathbf{f}}$ represents the job set $\{J'_1, J'_2, \dots, J'_{|\mathcal{J}|}\}$ where $p_{J'_i} = p_{J_i}$, $c_{J'_i} = c_{J_i}$, $r_{J'_i} = r_{J_i}$ and $d_{J'_i} = f_{J_i}$ for all $1 \leq i \leq |\mathcal{J}|$. We say that a $|\mathcal{J}|$ -tuple \mathbf{f} is *EDF-ordered* if $\mathcal{J}^{\mathbf{f}}$ follows the EDF priority. Furthermore, $\mathcal{J}^{\mathbf{f}}$ is said to be *EDF-equivalent* to \mathcal{J} . We first establish a link between Conditions I and II.

LEMMA 3.1. *If Condition I is satisfied for a job set \mathcal{J} by a voltage schedule \mathcal{S} and an EDF-ordered $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}})$, Condition II is satisfied for a job set $\mathcal{J}^{\mathbf{f}}$ by \mathcal{S} .*

PROOF. For any $r \in R_{\mathcal{J}^{\mathbf{f}}}$ and $d \in D_{\mathcal{J}^{\mathbf{f}}}$ ($r < d$), we have

$$\begin{aligned} r &\in \{t \mid t \in R_{\mathcal{J}} (= R_{\mathcal{J}^{\mathbf{f}}}) \wedge t < d\} \quad \text{and} \\ d &= f_{J_i} \text{ for } \exists f_{J_i} \in D_{\mathcal{J}^{\mathbf{f}}} (= \{f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}\}) . \end{aligned}$$

Furthermore, since \mathbf{f} is EDF-ordered, we have

$$\begin{aligned} \forall J'_k \in \mathcal{J}^{\mathbf{f}} \text{ s.t. } r_{J'_k} (= r_{J_k}) &\in [r, d (= f_{J_i})], \\ d_{J'_k} = f_{J_k} \leq f_{J_i} = d &\quad \text{if } p_{J'_k} \leq p_{J'_i} (= p_{J_i}) \\ d_{J'_k} = f_{J_k} > f_{J_i} = d &\quad \text{otherwise.} \end{aligned}$$

Thus, we have for all $J'_k \in \mathcal{J}^{\mathbf{f}}$:

$$p_{J'_k} \leq p_{J'_i} \wedge r_{J'_k} \in [r, d] \iff [r_{J'_k}, d_{J'_k}] \subseteq [r, d]. \quad (5)$$

Finally, by substituting d for f_{J_i} in (2), we have

$$\begin{aligned} W(\mathcal{S}, [r, d]) &\geq C(\{J \in \mathcal{J} \mid p_J \leq p_{J_i} \wedge r_J \in [r, d]\}) \\ &= C(\{J'_k \in \mathcal{J}^{\mathbf{f}} \mid p_{J'_k} \leq p_{J'_i} (= p_{J_i}) \wedge r_{J'_k} (= r_{J_k}) \in [r, d]\}) \\ &= C(\{J' \in \mathcal{J}^{\mathbf{f}} \mid [r_{J'}, d_{J'}] \subseteq [r, d]\}) . \quad (\text{From (5).}) \end{aligned}$$

□

LEMMA 3.2. *If Condition II is satisfied for a job set $\mathcal{J}^{\mathbf{f}}$ by a voltage schedule \mathcal{S} where $\mathbf{f} = (f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}})$ is an EDF-ordered $|\mathcal{J}|$ -tuple, Condition I is satisfied for a job set \mathcal{J} by \mathcal{S} .*

PROOF. Let $r \in \{t \mid t \in R_J \wedge t < f_{J_i}\}$. Then, we have

$$r \in R_{J^f} (= R_J), f_{J_i} \in D_{J^f} (= \{f_{J_1}, f_{J_2}, \dots, f_{J_{|J|}}\}) \text{ and } r < f_{J_i}$$

and substituting f_{J_i} for d in Condition II gives

$$W(S, [r, f_{J_i}]) \geq C(\{J' \in \mathcal{J}^f \mid [r_{J'}, d_{J'}] \subseteq [r, f_{J_i}]\}).$$

Since \mathbf{f} is EDF-ordered, we have for all $J'_k \in \mathcal{J}^f$ (Refer to the proof of Lemma 3.1.):

$$p_{J'_k} \leq p_{J'_i} \wedge r_{J'_k} \in [r, f_{J_i}] \iff [r_{J'_k}, d_{J'_k}] \subseteq [r, f_{J_i}]. \quad (6)$$

Therefore, we have

$$\begin{aligned} W(S, [r, f_{J_i}]) &\geq C(\{J' \in \mathcal{J}^f \mid [r_{J'}, d_{J'}] \subseteq [r, f_{J_i}]\}) \\ &= C(\{J'_k \in \mathcal{J}^f \mid p_{J'_k} \leq p_{J'_i} (= p_{J_i}) \wedge r_{J'_k} (= r_{J_k}) \in [r, f_{J_i}]\}) \\ &= C(\{J \in \mathcal{J} \mid p_J \leq p_{J_i} \wedge r_J \in [r, f_{J_i}]\}). \end{aligned}$$

□

From Lemmas 3.1 and 3.2, we can derive the following useful theorem which states how a feasible voltage schedule of a job set can be obtained from its EDF-equivalent job sets.

THEOREM 3.3. *Given a job set \mathcal{J} , let \mathcal{F}_J be the set of all feasible voltage schedules for \mathcal{J} . Then, $\mathcal{F}_J = \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{J^f}$ where \mathcal{T}_{EDF} is the set of all EDF-ordered $|\mathcal{J}|$ -tuples for \mathcal{J} .*

PROOF. To show that $\mathcal{S} \in \mathcal{F}_J \Rightarrow \mathcal{S} \in \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{J^f}$, assume that J_i completes its execution at $f_{J_i} (\leq d_{J_i})$ for all $1 \leq i \leq |\mathcal{J}|$ under $\mathcal{S} \in \mathcal{F}_J$. Let $\mathbf{f} = (f_{J_1}, f_{J_2}, \dots, f_{J_{|J|}})$. Then, \mathcal{J}^f is an EDF job set. If not, we have for some $J'_k, J'_i \in \mathcal{J}^f$ (where $p_{J'_k} < p_{J'_i}$)

$$r_{J'_k} < d_{J'_i} (= f_{J_i}) < d_{J'_k} (= f_{J_k}),$$

which contradicts a fact that once a higher priority job (i.e., J_k) is released during the execution of a lower priority job (i.e., J_i), the higher priority job completes earlier than the lower priority job (i.e., $f_{J_k} < f_{J_i}$). Furthermore, from Lemma 3.1, $\mathcal{S}(t)$ is a feasible schedule for the EDF job set \mathcal{J}^f . Thus, we have $\mathcal{S} \in \cup_{\mathbf{f} \in \mathcal{T}_{\text{EDF}}} \mathcal{F}_{J^f}$.

Conversely, given an EDF-ordered $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_{J_1}, f_{J_2}, \dots, f_{J_{|J|}})$, let $\mathcal{S} \in \mathcal{F}_{J^f}$ be a feasible schedule for the EDF-equivalent job set \mathcal{J}^f . Then, from Lemma 3.2, \mathcal{S} satisfies Condition I for \mathcal{J} . Thus, we have $\mathcal{S} \in \mathcal{F}_J$. □

COROLLARY 3.4. *Given a job set \mathcal{J} , $E(\mathcal{S}_{\text{opt}}^J) \leq E(\mathcal{S}_{\text{opt}}^{J^f})$ for any EDF-equivalent job set \mathcal{J}^f . Furthermore, there exists an EDF-equivalent job set \mathcal{J}^f such that $\mathcal{S}_{\text{opt}}^J \equiv \mathcal{S}_{\text{opt}}^{J^f}$.*

From Theorem 3.3, there is a one-to-one correspondence between feasible schedules of a fixed-priority job set \mathcal{J} and feasible schedules of \mathcal{J} 's EDF-equivalent job sets. Since the energy-optimal schedule $\mathcal{S}_{\text{opt}}^{J^f}$ for an EDF-equivalent job set \mathcal{J}^f can be directly computed (in polynomial time) by Yao's algorithm [Yao et al. 1995], the problem of finding an energy-optimal (feasible) voltage schedule of \mathcal{J} is reduced to the problem of finding an EDF-equivalent job set \mathcal{J}^f (or to selecting an EDF-ordered $|\mathcal{J}|$ -tuple \mathbf{f}) that minimizes $E(\mathcal{S}_{\text{opt}}^{J^f})$.

Figure 1 shows an example of EDF-equivalent job sets and EDF-ordered $|\mathcal{J}|$ -tuples. Figure 1.(a) shows the original job set $\mathcal{J} = \{J_1, J_2\}$. In this example, J_2 has a lower priority but earlier deadline than J_1 , so \mathcal{J} is not an EDF job set. (So Yao's algorithm cannot be

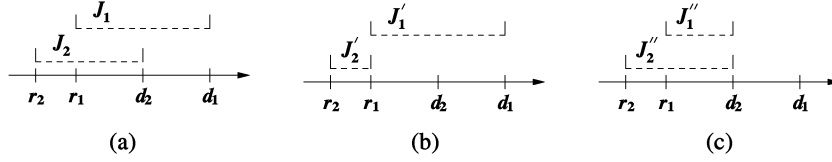


Fig. 1. An example of EDF-equivalent job sets.

directly applied to \mathcal{J} .) In Figures 1.(b) and 1.(c), two job sets are shown, which are EDF-equivalent to \mathcal{J} . The job sets $\{J'_1, J'_2\}$ and $\{J''_1, J''_2\}$ are obtained by choosing (r_{J_1}, d_{J_1}) and (d_{J_2}, d_{J_2}) as EDF-ordered $|\mathcal{J}|$ -tuples, respectively. Both job sets follow the EDF priority assignment³ and the optimal voltage schedule for each job set can be computed by Yao's algorithm. (As will be explained below, the energy-optimal voltage schedule of \mathcal{J} is equal to $\mathcal{S}_{\text{opt}}^{\{J'_1, J'_2\}}$ or $\mathcal{S}_{\text{opt}}^{\{J''_1, J''_2\}}$ depending on the workload of J_1 and J_2 .)

Now we are to restrict the search space of EDF-ordered $|\mathcal{J}|$ -tuples (equivalently, EDF-equivalent job sets). First, an EDF-ordered $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_{|\mathcal{J}|})$ does not need to be considered if for another EDF-ordered $|\mathcal{J}|$ -tuple $\mathbf{f}' = (f'_1, f'_2, \dots, f'_{|\mathcal{J}|})$ ($\mathbf{f}' \neq \mathbf{f}$), $f_i \leq f'_i$ for all $1 \leq i \leq |\mathcal{J}|$. This is because, for any voltage schedule $\mathcal{S}(t)$ which is feasible under \mathbf{f} , $\mathcal{S}(t)$ is also feasible under \mathbf{f}' . We define that an EDF-ordered $|\mathcal{J}|$ -tuple \mathbf{f} (or $\mathcal{J}^{\mathbf{f}}$) is *essential* if such \mathbf{f}' does not exist. (The term 'essential' is equivalent to the term 'NAP' in [Quan and Hu 2002].) Quan's optimal algorithm [Quan and Hu 2002] finds an optimal voltage schedule by *exhaustively* enumerating all the essential (or NAP) job sets and then applying Yao's algorithm for each essential job set. Our fast algorithm avoids the exhaustiveness by carefully enumerating the essential job sets.

3.2 $|\mathcal{J}|$ -Permutations

It is easy to check if a $|\mathcal{J}|$ -tuple is EDF-ordered (or essential). On the contrary, it is not obvious how such $|\mathcal{J}|$ -tuples can be enumerated. In this section, we describe how to construct EDF-ordered $|\mathcal{J}|$ -tuples efficiently using a permutation-based analysis.

Given a $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_{|\mathcal{J}|})$, let $\sigma_{\mathbf{f}} : \{1, 2, \dots, |\mathcal{J}|\} \Rightarrow \{1, 2, \dots, |\mathcal{J}|\}$ be a permutation that maps a new tuple index when the tuple elements are sorted in a non-decreasing order, i.e., $f_{\sigma_{\mathbf{f}}^{-1}(1)} \leq f_{\sigma_{\mathbf{f}}^{-1}(2)} \leq \dots \leq f_{\sigma_{\mathbf{f}}^{-1}(|\mathcal{J}|)}$. Ties are broken by the priority, i.e., if $f_i = f_j$ where $i < j$, $\sigma_{\mathbf{f}}(i) < \sigma_{\mathbf{f}}(j)$. (From now on, we call such σ a $|\mathcal{J}|$ -permutation.) For example, let $\mathbf{f} = (f_1, f_2, f_3, f_4) = (4, 10, 2, 10)$. Then, since $f_3 \leq f_1 \leq f_2 = f_4$, we have $\sigma(3) = 1$, $\sigma(1) = 2$, and (from the tie-breaking rule) $(\sigma(2), \sigma(4)) = (3, 4)$. (Equivalently, we have $(\sigma^{-1}(1), \sigma^{-1}(2), \sigma^{-1}(3), \sigma^{-1}(4)) = (3, 1, 2, 4)$.) Note that $\sigma^{-1}(i)$ denotes the index of the i -th smallest element in \mathbf{f} , i.e., $f_{\sigma^{-1}(i)}$ is the i -th smallest element in \mathbf{f} .

The following lemma states that there cannot exist more than one essential $|\mathcal{J}|$ -tuples whose $|\mathcal{J}|$ -permutations are the same, that is, each essential $|\mathcal{J}|$ -tuple can be uniquely addressed by its corresponding $|\mathcal{J}|$ -permutation (and, obviously, vice versa).

LEMMA 3.5. *For any two essential $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_{|\mathcal{J}|})$ and $\mathbf{f}' = (f'_1, f'_2, \dots, f'_{|\mathcal{J}|})$ ($\mathbf{f} \neq \mathbf{f}'$), $\sigma_{\mathbf{f}} \neq \sigma_{\mathbf{f}'}$.*

³In Figure 1.(c), J''_1 need not have an earlier deadline than J''_2 for the job set to be an EDF job set; $d_{J''_1} = d_{J''_2}$ is sufficient for the job set to be optimally scheduled by Yao's algorithm [Yao et al. 1995].

PROOF. Suppose $\sigma_{\mathbf{f}} \equiv \sigma_{\mathbf{f}'}$ and let i ($1 \leq i \leq |\mathcal{J}|$) be the largest integer such that $f_{\sigma_{\mathbf{f}}^{-1}(i)} \neq f'_{\sigma_{\mathbf{f}'}^{-1}(i)}$, i.e.,

$$f_{\sigma_{\mathbf{f}}^{-1}(k)} = f'_{\sigma_{\mathbf{f}'}^{-1}(k)} \quad (= f'_{\sigma_{\mathbf{f}}^{-1}(k)}) \quad \text{for all } i < k < |\mathcal{J}|. \quad (7)$$

Without loss of generality, we can assume $f_{\sigma_{\mathbf{f}}^{-1}(i)} < f'_{\sigma_{\mathbf{f}'}^{-1}(i)}$. Let us consider a new $|\mathcal{J}|$ -tuple $\mathbf{f}'' = (f''_1, f''_2, \dots, f''_{|\mathcal{J}|})$ where

$$f''_k = \begin{cases} f'_k & k = \sigma_{\mathbf{f}}^{-1}(i), \\ f_k & \text{otherwise.} \end{cases}$$

From the definition of \mathbf{f}'' , it can be easily seen that $\sigma_{\mathbf{f}''} \equiv \sigma_{\mathbf{f}} \equiv \sigma_{\mathbf{f}'}$. (We omit the subscripts in the rest of the proof.) We are now to prove that \mathbf{f}'' is EDF-ordered, i.e., for any $1 \leq j < k \leq |\mathcal{J}|$,

$$f''_j \leq f''_k \text{ or } f''_k \leq r_{J_j}. \quad (8)$$

Since \mathbf{f} is EDF-ordered, (8) holds for all $1 \leq j < k \leq |\mathcal{J}|$ except for $j = \sigma^{-1}(i)$ or $k = \sigma^{-1}(i)$. So, it remains to show that (8) holds for all $1 \leq j < \sigma^{-1}(i) \leq |\mathcal{J}|$ and $1 \leq \sigma^{-1}(i) < k \leq |\mathcal{J}|$.

Case (a): $1 \leq j < \sigma^{-1}(i) \leq |\mathcal{J}|$ (when J_j has a higher priority than $J_{\sigma^{-1}(i)}$.)

If $f''_j \leq f''_{\sigma^{-1}(i)}$, (8) trivially holds. So, we only consider j such that $f''_j > f''_{\sigma^{-1}(i)}$, i.e., $f_j (= f_{\sigma^{-1}(\sigma(j))}) > f'_{\sigma^{-1}(i)} (> f_{\sigma^{-1}(i)})$. From the definition of σ , we have $\sigma(j) > i$. Thus, by substituting $\sigma(j)$ for k in Eq. (7), we have $f_j (= f''_j) = f'_j$. From the assumption, \mathbf{f} is EDF-ordered, but we have $f'_j = f_j > f'_{\sigma^{-1}(i)}$. So, it must be the case that $f'_{\sigma^{-1}(i)} \leq r_{J_j}$. Therefore, we have

$$f''_{\sigma^{-1}(i)} = f'_{\sigma^{-1}(i)} \leq r_{J_j}.$$

Case (b): $1 \leq \sigma^{-1}(i) < k \leq |\mathcal{J}|$ (when J_k has a lower priority than $J_{\sigma^{-1}(i)}$.)

First, we can exclude the case when $f_k = f_{\sigma^{-1}(i)}$. Otherwise, we have $\sigma(k) > \sigma(\sigma^{-1}(i)) = i$. (Recall the tie-breaking rule.) But, by the definition of σ , $f'_{\sigma^{-1}(\sigma(k))} (= f'_k) \geq f'_{\sigma^{-1}(i)}$ and we finally have

$$f'_k \geq f'_{\sigma^{-1}(i)} > f_{\sigma^{-1}(i)} = f_k,$$

which contradicts Eq. (7).

Second, consider k such that $f_k < f_{\sigma^{-1}(i)}$. \mathbf{f} is EDF-ordered, but we have $f_{\sigma^{-1}(i)} > f_k$. So, it must be the case that $f_k \leq r_{J_{\sigma^{-1}(i)}}$. Therefore, we have

$$f''_k = f_k \leq r_{J_{\sigma^{-1}(i)}}.$$

Finally, for k such that $f_k > f_{\sigma^{-1}(i)}$, we have

$$f''_{\sigma^{-1}(i)} = f'_{\sigma^{-1}(i)} \leq f'_k = f_k = f''_k.$$

Thus, \mathbf{f}'' is EDF-ordered. However, since we have

$$f_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i)} = f''_{\sigma^{-1}(i)} \quad \text{and} \quad f_k = f''_k \quad \text{for all } 1 \leq k \neq \sigma^{-1}(i) \leq |\mathcal{J}|,$$

\mathbf{f} is not essential, a contradiction. Therefore, $\sigma_{\mathbf{f}} \neq \sigma_{\mathbf{f}'}$. \square

```

1:   $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$ 
2:  for ( $i := |\mathcal{J}| - 1$  to 1)
3:      let  $\mathcal{J}^H$  be  $\{J_{\sigma^{-1}(k)} \mid i < k \leq |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$ 
4:      if ( $r_{J_{\sigma^{-1}(i)}} \geq \min(\{r_J \mid J \in \mathcal{J}^H\} \cup \{f_{\sigma^{-1}(i+1)}\})$ ) return FALSE
5:      else  $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J \mid J \in \mathcal{J}^H\})$ 
6:      end if
7:  end for
    
```

Fig. 2. The algorithm to build a $|\mathcal{J}|$ -tuple from a $|\mathcal{J}|$ -permutation.

The proof of Lemma 3.5 also implies how to build a unique essential job set for a σ .

LEMMA 3.6. *Given a $|\mathcal{J}|$ -permutation σ , the algorithm in Figure 2 finds a unique essential $|\mathcal{J}|$ -tuple for σ if such a $|\mathcal{J}|$ -tuple exists. Otherwise, it returns FALSE.*

PROOF. First, suppose that the essential $|\mathcal{J}|$ -tuple for σ exists and denote it by $\mathbf{f}' = (f'_1, f'_2, \dots, f'_{|\mathcal{J}|})$. (Note that $f'_{\sigma^{-1}(1)} \leq f'_{\sigma^{-1}(2)} \leq \dots \leq f'_{\sigma^{-1}(|\mathcal{J}|)}$.) We are to prove that $f'_{\sigma^{-1}(i)} = f_{\sigma^{-1}(i)}$ and the algorithm does not abort in line 4 for all $i = |\mathcal{J}|, |\mathcal{J}| - 1, \dots, 1$ by induction on i . The base case holds trivially, i.e., $f'_{\sigma^{-1}(|\mathcal{J}|)} = d_{J_{\sigma^{-1}(|\mathcal{J}|)}} = f_{\sigma^{-1}(|\mathcal{J}|)}$. For the induction step, assume that the proposition holds for all $k = |\mathcal{J}|, |\mathcal{J}| - 1, \dots, i + 1$. Let $\mathcal{J}^H = \{J_{\sigma^{-1}(k)} \mid i < k \leq |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$ (as in line 3 of the algorithm). Note that any job in \mathcal{J}^H has the higher priority than $J_{\sigma^{-1}(i)}$ and that $f'_{\sigma^{-1}(i)} \leq d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} \leq f'_{\sigma^{-1}(i+1)}$.

Case (a): $\mathcal{J}^H = \emptyset$.

Suppose that $f'_{\sigma^{-1}(i)} < d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i+1)}$, that is,

$$f'_{\sigma^{-1}(1)} \leq \dots \leq f'_{\sigma^{-1}(i)} < \min\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \leq f'_{\sigma^{-1}(i+1)} \leq \dots \leq f'_{\sigma^{-1}(|\mathcal{J}|)}.$$

Let $\mathbf{f}'' = (f'_1, \dots, f'_{\sigma^{-1}(i)-1}, \min\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\}, f'_{\sigma^{-1}(i+1)}, \dots, f'_{|\mathcal{J}|})$. Then, \mathbf{f}'' is EDF-ordered, and \mathbf{f}' is not essential, a contradiction. Therefore, we have

$$f'_{\sigma^{-1}(i)} = \min\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} = \min\{d_{J_{\sigma^{-1}(i)}}, f_{\sigma^{-1}(i+1)}\} = f_{\sigma^{-1}(i)}.$$

Case (b): $\mathcal{J}^H \neq \emptyset$.

For all $J_{\sigma^{-1}(k)} \in \mathcal{J}^H$, we have $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(k)}$ from the definition of σ (Recall the tie-breaking rule.), and $f'_{\sigma^{-1}(i)} \leq r_{J_{\sigma^{-1}(k)}}$ since \mathbf{f}' is EDF-ordered. Suppose that $f'_{\sigma^{-1}(i)} < \min\{r_J \mid J \in \mathcal{J}^H\}$, $f'_{\sigma^{-1}(i)} < d_{J_{\sigma^{-1}(i)}}$ and $f'_{\sigma^{-1}(i)} < f'_{\sigma^{-1}(i+1)}$, that is,

$$f'_{\sigma^{-1}(1)} \leq \dots \leq f'_{\sigma^{-1}(i)} < \min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J \mid J \in \mathcal{J}^H\}) \leq f'_{\sigma^{-1}(i+1)} \leq \dots \leq f'_{\sigma^{-1}(|\mathcal{J}|)}.$$

Let $\mathbf{f}'' = (f'_1, \dots, f'_{\sigma^{-1}(i)-1}, \min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J \mid J \in \mathcal{J}^H\}), f'_{\sigma^{-1}(i+1)}, \dots, f'_{|\mathcal{J}|})$. Then, it can be easily shown that \mathbf{f}'' is EDF-ordered. Thus, \mathbf{f}' is not essential, a contradiction. Therefore, we have

$$\begin{aligned} f'_{\sigma^{-1}(i)} &= \min(\{d_{J_{\sigma^{-1}(i)}}, f'_{\sigma^{-1}(i+1)}\} \cup \{r_J \mid J \in \mathcal{J}^H\}) \\ &= \min(\{d_{J_{\sigma^{-1}(i)}}, f_{\sigma^{-1}(i+1)}\} \cup \{r_J \mid J \in \mathcal{J}^H\}) = f_{\sigma^{-1}(i)}. \end{aligned}$$

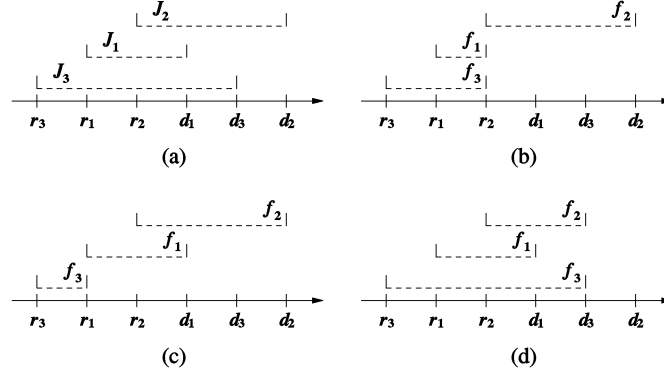


Fig. 3. An example of $|\mathcal{J}|$ -permutations. (a) A job set and its EDF-equivalent job sets for which $(\sigma^{-1}(3), \sigma^{-1}(2), \sigma^{-1}(1)) =$ (b) $(2, 3, 1)$, (c) $(2, 1, 3)$, and (d) $(3, 2, 1)$, respectively. $((\sigma^{-1}(3), \sigma^{-1}(2), \sigma^{-1}(1)) = (1, 2, 3), (1, 3, 2)$ and $(3, 1, 2)$ are not valid \mathcal{J} -permutations.)

Furthermore, we have for both cases

$$r_{J_{\sigma^{-1}(i)}} < f'_{\sigma^{-1}(i)} \leq \min(\{r_J | J \in \mathcal{J}^H\} \cup \{f'_{\sigma^{-1}(i+1)}\}) = \min(\{r_J | J \in \mathcal{J}^H\} \cup \{f_{\sigma^{-1}(i+1)}\}),$$

and the algorithm does not abort in line 4 at iteration i , which completes the induction.

If the algorithm does not abort, the $|\mathcal{J}|$ -tuple built by the algorithm is always a correct EDF-ordered $|\mathcal{J}|$ -tuple, implying the existence of such $|\mathcal{J}|$ -tuple for σ . Therefore, if such $|\mathcal{J}|$ -tuple does not exist, the algorithm eventually returns FALSE. \square

If a $|\mathcal{J}|$ -permutation σ has the corresponding EDF-ordered $|\mathcal{J}|$ -tuple \mathbf{f} , it is said to be *valid*. Furthermore, if \mathbf{f} is essential, σ is said to be *essential*. From the above argument, we can establish one-to-one correspondences between EDF-ordered $|\mathcal{J}|$ -tuples and valid $|\mathcal{J}|$ -permutations, and between essential $|\mathcal{J}|$ -tuples and essential $|\mathcal{J}|$ -permutations. Figure 3.(a) shows a job set with three jobs and Figures 3.(b), 3.(c) and 3.(d) show its EDF-equivalent job sets with their $|\mathcal{J}|$ -permutations. Among $3! (= 6)$ possible $|\mathcal{J}|$ -permutations, only three permutations are valid (and essential).

Based on the algorithm in Figure 2, we describe another way to enumerate $|\mathcal{J}|$ -tuples. In the following, r_{J_i} and d_{J_i} are interpreted as symbolic values, not as real numbers. Then, $R_{\mathcal{J}} \cup D_{\mathcal{J}}$ has $2 \cdot |\mathcal{J}|$ distinct symbolic values. Furthermore, the algorithm in Figure 2 is assumed to assign symbolic values to elements of a $|\mathcal{J}|$ -tuple with the following tie-breaking rule in line 5:

$$(a) r_{J_i} = r_{J_j} (i < j) : r_{J_i} < r_{J_j} \quad (b) d_{J_i} = d_{J_j} (i < j) : r_{J_i} < r_{J_j} \quad (c) r_{J_i} = d_{J_j} : r_{J_i} < d_{J_j}.$$

Given a $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_{|\mathcal{J}|})$, let $\zeta_{\mathbf{f}} : R_{\mathcal{J}} \cup D_{\mathcal{J}} \Rightarrow \{0, 1\}$ be a bit-vector of length $2 \cdot |\mathcal{J}|$ such that

$$\zeta_{\mathbf{f}}(t) = \begin{cases} 1 & t = f_k \text{ for some } 1 \leq k \leq |\mathcal{J}|, \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm in Figure 4 constructs a $|\mathcal{J}|$ -tuple from an arbitrary bit-vector $\zeta : R_{\mathcal{J}} \cup D_{\mathcal{J}} \Rightarrow \{0, 1\}$. The correctness of the algorithm can be proved in a similar manner as the algorithm in Figure 2.

```

1:   $J' := \{\}, D := \{\}$ 
2:  foreach ( $d_{J_i} \in D_{J_i}$  s.t.  $\zeta(d_{J_i}) = 1$ )
3:       $f_i := d_{J_i}$ ,  $J' := J' \cup \{J_i\}$ ,  $D := D \cup \{d_{J_i}\}$ 
4:  end foreach /* return FALSE here if  $J'$  does not follow the EDF priority. */
5:  foreach ( $r_{J_i} \in R_{J_i}$  s.t.  $\zeta(r_{J_i}) = 1$  in a decreasing order)
6:       $f_i := \max\{d \in D \mid J' \cup \{J_i\} \text{ follows the EDF priority where } p_J = p_{J_i}, r_J = r_{J_i}, d_J = d\}$ 
7:      /* return FALSE here if such  $f_i$  does not exist. */
8:       $J' := J' \cup \{J_i\}$ ,  $D := D \cup \{r_{J_i}\}$ 
9:  end foreach
10: foreach ( $J_i$  s.t.  $f_i$  is not determined (in any order))
11:      $f_i := \max\{d \in D \mid J' \cup \{J_i\} \text{ follows the EDF priority where } p_J = p_{J_i}, r_J = r_{J_i}, d_J = d\}$ 
12:     /* return FALSE here if such  $f_i$  does not exist. */
13:      $J' := J' \cup \{J_i\}$ 
14: end foreach

```

Fig. 4. The algorithm to build a $|J|$ -tuple from a bit-vector.

3.3 An Alternative Formulation

The problem formulation given in Section 2 is based on the voltage schedule $\mathcal{S}(t)$. In this section, we describe an alternative formulation, based on the following intuitive property, which states that each job runs at the same constant speed if the voltage schedule is an optimal one.

LEMMA 3.7. *For an energy-optimal voltage schedule $\mathcal{S}(t)$, $\mathcal{S}(t_1) = \mathcal{S}(t_2)$ for any t_1 and t_2 such that $\text{job}(J, \mathcal{S}, t_1) = \text{job}(J, \mathcal{S}, t_2)$.*

PROOF. Given an optimal schedule $\mathcal{S}(t)$, suppose that $\mathcal{S}(t_1) \neq \mathcal{S}(t_2)$ for some t_1 and t_2 such that $\text{job}(J, \mathcal{S}, t_1) = \text{job}(J, \mathcal{S}, t_2)$. Given that $\mathcal{S}(t)$ is optimal, there exist t'_1, t'_2, S_1, S_2 and Δt such that $\mathcal{S}(t) = S_1$ for $t'_1 \leq t \leq t'_1 + \Delta t$, $\mathcal{S}(t) = S_2$ for $t'_2 \leq t \leq t'_2 + \Delta t$, and $S_1 \neq S_2$. Let $\mathcal{S}(t)'$ be defined by

$$\mathcal{S}(t)' = \begin{cases} \frac{S_1 + S_2}{2} & t'_1 \leq t \leq t'_1 + \Delta t, t'_2 \leq t \leq t'_2 + \Delta t, \\ \mathcal{S}(t) & \text{otherwise.} \end{cases}$$

Then, it is obvious that $\mathcal{S}(t)'$ is feasible and $E(\mathcal{S}') < E(\mathcal{S})$, a contradiction. \square

From Lemma 3.7, it can be shown that the voltage scheduling problem is equivalent to determining the allowed execution time a_i allocated to each J_i . Given a feasible voltage schedule \mathcal{S} , the corresponding tuple of the allowed execution times $(a_1, a_2, \dots, a_{|J|})$, called a *time-allocation tuple*, can be uniquely determined. Conversely, given a time-allocation tuple $\mathbf{A} = (a_1, a_2, \dots, a_{|J|})$, the corresponding voltage schedule $\mathcal{S}_{\mathbf{A}}$ can be uniquely constructed by assigning the constant execution speed c_i/a_i to J_i . \mathbf{A} is said to be *feasible* if the corresponding voltage schedule $\mathcal{S}_{\mathbf{A}}$ is feasible.

Let us now consider the exact condition for a time-allocation tuple $\mathbf{A} = (a_1, a_2, \dots, a_{|J|})$ to be feasible by rewriting Condition I in Section 2 in terms of \mathbf{A} .

Condition III (Feasibility Condition for Time-Allocation Tuples).

There exists a $|\mathcal{J}|$ -tuple $(f_{J_1}, f_{J_2}, \dots, f_{J_{|\mathcal{J}|}}) \in \mathcal{T}^{\mathcal{J}}$ such that

$$\forall 1 \leq i \leq |\mathcal{J}| \quad \forall r \in \{t \mid t \in R_{\mathcal{J}} \wedge t < f_{J_i}\} \\ \sum_{J_k / p_{J_k} \leq p_{J_i} \wedge r_{J_k} \in [r, f_{J_i})} a_k \leq f_{J_i} - r . \quad (9)$$

LEMMA 3.8. *Condition III is a necessary and sufficient condition for \mathbf{A} to be feasible.*

PROOF. Given a job set $\mathcal{J} = \{J_1, J_2, \dots, J_{|\mathcal{J}|}\}$ and a time-allocation tuple $\mathbf{A} = (a_1, a_2, \dots, a_{|\mathcal{J}|})$ for \mathcal{J} , consider a new job set $\mathcal{J}' = \{J'_1, J'_2, \dots, J'_{|\mathcal{J}|}\}$ where $c_{J'_i} = a_i$, $r_{J'_i} = r_{J_i}$, $d_{J'_i} = d_{J_i}$, and $p_{J'_i} = p_{J_i}$ for all $1 \leq i \leq |\mathcal{J}|$, i.e., \mathcal{J}' is identical to \mathcal{J} except for the workload.

Let $S'(t) = 1$ ($\forall t > 0$) be the voltage schedule of \mathcal{J}' . Then, it is obvious that the response time of J_i under $S_{\mathbf{A}}$ is the same as that of J'_i under S' . Thus, \mathbf{A} is feasible if and only if S' is a feasible voltage schedule for \mathcal{J}' . After replacing S and c_{J_i} in Condition I by S' and a_i , respectively, we have Condition III. \square

By applying the same argument to Condition II, we have the following condition for EDF job sets.

Condition IV (EDF Feasibility Condition for Time-Allocation Tuples).

For any $r \in R_{\mathcal{J}}$ and $d \in D_{\mathcal{J}}$ (where $r < d$),

$$\sum_{J / [r_J, d_J] \subseteq [r, d]} a_i \leq d - r .$$

Now, the voltage scheduling problem can be reformulated as follows:

Find a time-allocation tuple $\mathbf{A} = (a_1, a_2, \dots, a_{|\mathcal{J}|})$ such that $E(S_{\mathbf{A}})$ is minimized subject to Condition III (or Condition IV for an EDF job set).

The energy consumption of the voltage schedule $S_{\mathbf{A}}$ can be directly computed:

$$E(S_{\mathbf{A}}) = \sum_{i=1}^{|\mathcal{J}|} a_i \cdot P(c_i/a_i) . \quad (10)$$

The set of feasible time-allocation tuples represents the solution space for the voltage scheduling problem stated in terms of time-allocation tuples. For an EDF job set, the solution space is specified by a conjunction of linear inequalities which can be directly obtained from Condition IV. However, this is not the case for a fixed-priority job set; the existential quantifier in Condition III is not always removable. Consequently, the solution space for an EDF job set is a convex set while the solution space for an arbitrary fixed-priority job set may not be a convex set.

Before we present an intractability result for the voltage scheduling problem in the next section, we illustrate the inherent complexity of fixed-priority voltage scheduling based on the results explained in this section. Figures 5.(a) and 5.(b) show the solution spaces for an example EDF job set and an example fixed-priority job set, respectively. As a fixed-priority job set, we use the job set $\{J_1, J_2\}$ of Figure 1. As an EDF job set, we use the same job set $\{J_1, J_2\}$ in Figure 1 with the same timing parameters, but the priority assignment is

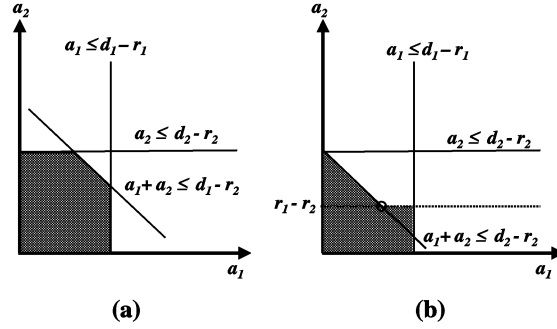


Fig. 5. Solution spaces for (a) an EDF job set and (b) a fixed-priority job set.

changed such that it follows the EDF priority assignment, i.e., $p_{J_2} < p_{J_1}$. For the EDF job set, we have the following constraint:

$$a_1 \leq d_{J_1} - r_{J_1} \wedge a_2 \leq d_{J_2} - r_{J_2} \wedge a_1 + a_2 \leq d_{J_1} - r_{J_2}$$

Similarly, we have the following constraint for the fixed-priority job set:

$$a_1 \leq d_{J_1} - r_{J_1} \wedge a_2 \leq r_{J_1} - r_{J_2} \quad (\text{Figure 1.(b)}) \vee$$

$$a_1 \leq d_{J_2} - r_{J_1} \wedge a_1 + a_2 \leq d_{J_2} - r_{J_2} \quad (\text{Figure 1.(c)})$$

In Figures 5.(a) and 5.(b), the solution spaces for the EDF job set and the fixed-priority job set are depicted as a convex region and a concave region, respectively. (Each point in the shaded regions represents a feasible schedule.) In general, the solution space of any EDF job set with N jobs are represented by a convex set in \mathbf{R}^N , whereas the solution space of a fixed-priority job set is represented by a concave set. Note that for EDF job sets, the objective function, the total energy consumption, can be efficiently minimized by an optimization technique for a convex set (as in Yao's algorithm). However, optimization problems defined on a concave set are generally intractable.

4. INTRACTABILITY RESULT

In this section, we present some observations related to the complexity issue of the optimal fixed-priority scheduling problem. We first show that the decision version of the problem is NP-hard.

THEOREM 4.1. *Given a job set \mathcal{J} and a positive number K , the problem of deciding if there is a feasible voltage schedule $S(t)$ for \mathcal{J} such that $E(S) \leq K$ is NP-hard.*

PROOF. Without loss of generality, we assume that the energy consumption (per CPU cycle) is quadratically dependent on the processor speed. That is, the instantaneous power consumption (per time) is cubically dependent on the processor speed, i.e., $P(t) = S(t)^3$. (The reduction can be easily modified for other power functions.) We prove the theorem by reduction from the subset-sum problem, which is NP-complete [Garey and Johnson 1979]:

SUBSET-SUM

INSTANCE: A finite set U , a size $s : U \Rightarrow \mathbf{Z}^+$, and a positive integer B .

Question: Is there a subset $\mathbf{U}' \subseteq \mathbf{U}$ such that $\sum_{u \in \mathbf{U}'} s(u) = B$?

Given an instance $\langle \mathbf{U} (= \{u_1, \dots, u_{|\mathbf{U}|}\}), s, B \rangle$ of the subset-sum problem, we construct a job set \mathcal{J} and a positive number K such that there is a voltage schedule $S(t)$ of \mathcal{J} with $E(S(t)) \leq K$ if and only if $\exists \mathbf{U}' \subseteq \mathbf{U}, \sum_{u \in \mathbf{U}'} s(u) = B$. The corresponding job set \mathcal{J} consists of $2 \cdot |\mathbf{U}| + 1$ jobs as follows:

$$\begin{aligned} \mathcal{J} &= \{J_1, J_2, \dots, J_{2 \cdot |\mathbf{U}| + 1}\} \quad \text{where} \\ p_{J_i} &= i \quad \text{for all } 1 \leq i \leq 2 \cdot |\mathbf{U}| + 1, \\ r_{J_{2 \cdot i + 1}} &= s(u_{i+1}) + \sum_{j=1}^i 3 \cdot s(u_j), \quad r_{J_{2 \cdot i + 2}} = \sum_{j=1}^i 3 \cdot s(u_j), \\ d_{J_{2 \cdot i + 1}} &= \sum_{j=1}^{i+1} 3 \cdot s(u_j), \quad d_{J_{2 \cdot i + 2}} = 2 \cdot s(u_{i+1}) + \sum_{j=1}^i 3 \cdot s(u_j), \\ c_{J_{2 \cdot i + 1}} &= 8 \cdot \gamma \cdot s(u_{i+1}), \quad c_{J_{2 \cdot i + 2}} = 8 \cdot s(u_{i+1}) \quad \text{for all } 0 \leq i \leq |\mathbf{U}| - 1, \text{ and} \\ r_{J_{2 \cdot |\mathbf{U}| + 1}} &= 0, \quad d_{J_{2 \cdot |\mathbf{U}| + 1}} = B + \sum_{j=1}^{|\mathbf{U}|} 3 \cdot s(u_j), \quad c_{J_{2 \cdot |\mathbf{U}| + 1}} = \sqrt[3]{4} \cdot B. \end{aligned}$$

where γ is the unique positive solution of the following quadratic equation:

$$\gamma^2 + \gamma = 1 + \frac{4}{3 \cdot 8^3} \quad (\implies \frac{1}{2} < \gamma < 1).$$

Furthermore, K is set to be

$$K = (8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot \sum_{i=1}^{|\mathbf{U}|} s(u_i) + 2 \cdot B.$$

From the construction of \mathcal{J} , we have

$$\begin{aligned} r_{J_{2 \cdot i + 2}} &< r_{J_{2 \cdot i + 1}} (= r_{J_{2 \cdot i + 2}} + s(u_{i+1})) < d_{J_{2 \cdot i + 2}} (= r_{J_{2 \cdot i + 1}} + s(u_{i+1})) < d_{J_{2 \cdot i + 1}} (= d_{J_{2 \cdot i + 2}} + s(u_{i+1})), \\ [r_{J_{2 \cdot i + 2}}, d_{J_{2 \cdot i + 1}}] &\subset [r_{J_{2 \cdot i + 1}}, d_{J_{2 \cdot i + 1}}] \quad \text{for all } 0 \leq i \leq |\mathbf{U}| - 1 \quad \text{and} \\ [r_{J_{2 \cdot i + 2}}, d_{J_{2 \cdot i + 1}}] &\cap [r_{J_{2 \cdot i + 2}}, d_{J_{2 \cdot i + 1}}] = \emptyset \quad \text{for all } 0 \leq i \neq i' \leq |\mathbf{U}| - 1. \end{aligned}$$

Let $\kappa: \{0, 1\}^{|\mathbf{U}|} \Rightarrow \mathcal{T}^{\mathcal{J}}$ be a function defined by

$$\begin{aligned} \kappa((b_1, b_2, \dots, b_{|\mathbf{U}|})) &= (f_1, f_2, \dots, f_{|\mathcal{J}|}) \quad \text{where} \\ f_{2 \cdot i + 1} &= d_{J_{2 \cdot i + 1}}, \quad f_{2 \cdot i + 2} = r_{J_{2 \cdot i + 1}} \quad \text{if } b_{i+1} = 0, \\ f_{2 \cdot i + 1} &= f_{2 \cdot i + 2} = d_{J_{2 \cdot i + 2}} \quad \text{if } b_{i+1} = 1 \quad \text{for all } 0 \leq i \leq |\mathbf{U}| - 1, \text{ and} \\ f_{2 \cdot |\mathbf{U}| + 1} &= d_{J_{2 \cdot |\mathbf{U}| + 1}}. \end{aligned}$$

Then, the set of essential job sets of \mathcal{J} is given by:

$$\{\mathcal{J}^{\mathbf{f}} \mid \mathbf{f} = \kappa(\mathbf{b}), \mathbf{b} \in \{0, 1\}^{|\mathbf{U}|}\}$$

To compute the energy consumption of an essential job set by Yao's algorithm [Yao et al.

1995], we first compare the intensity of each interval. Let

$$I_1 = \frac{c_{J_{2:i+2}}}{r_{J_{2:i+1}} - r_{J_{2:i+2}}}, I_2 = \frac{c_{J_{2:i+1}}}{d_{J_{2:i+1}} - r_{J_{2:i+1}}},$$

$$I_3 = \frac{c_{J_{2:i+1}}}{d_{J_{2:i+2}} - r_{J_{2:i+1}}}, I_4 = \frac{c_{J_{2:i+1}} + c_{J_{2:i+2}}}{d_{J_{2:i+2}} - r_{J_{2:i+1}}} \text{ and } I_5(\delta) = \frac{c_{J_{2:|U|+1}}}{B + \delta}.$$

Then, we have

$$I_1 = \frac{8 \cdot s(u_{i+1})}{s(u_{i+1})} = 8 > I_2 = \frac{8 \cdot \gamma \cdot s(u_{i+1})}{2 \cdot s(u_{i+1})} = 4 \cdot \gamma > 2 > \sqrt[3]{4} > I_5 = \frac{\sqrt[3]{4} \cdot B}{B + \delta} \text{ and}$$

$$I_4 = \frac{8 \cdot (1 + \gamma) \cdot s(u_{i+1})}{2 \cdot s(u_{i+1})} = 4 + 4 \cdot \gamma > I_3 = \frac{8 \cdot \gamma \cdot s(u_{i+1})}{s(u_{i+1})} = 8 \cdot \gamma > I_5.$$

So, the energy consumption of $\mathcal{S}_{\text{opt}}^{\mathbf{f}}$ for $\mathbf{f} = \kappa((b_1, b_2, \dots, b_{|U|}))$ can be computed as follows:

$$E(\mathcal{S}_{\text{opt}}^{\mathbf{f}}) = \sum_{i=1}^{|U|} E_i + E_L \quad \text{where}$$

$$E_i = \begin{cases} (8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot s(u_i) & (= \frac{(8 \cdot s(u_i))^3}{s(u_i)^2} + \frac{(8 \cdot \gamma \cdot s(u_i))^3}{(2 \cdot s(u_i))^2}) & b_i = 0, \\ \frac{(1+\gamma)^3}{4} \cdot 8^3 \cdot s(u_i) & (= \frac{(8 \cdot (1+\gamma) \cdot s(u_i))^3}{(2 \cdot s(u_i))^2}) & b_i = 1 \end{cases} \text{ and}$$

$$E_L = \frac{4 \cdot B^3}{(B + \sum_{i=1}^{|U|} b_i \cdot s(u_i))^2} \left(= \frac{c_{J_{2:|U|+1}}^3}{(B + \sum_{i=1}^{|U|} b_i \cdot (d_{J_{2:i-1}} - d_{J_{2:i}}))^2} \right).$$

Since we have

$$\begin{aligned} \frac{(1+\gamma)^3}{4} \cdot 8^3 \cdot s(u_i) &= \frac{1 + 3 \cdot \gamma + 3 \cdot \gamma^2 + \gamma^3}{4} \cdot 8^3 \cdot s(u_i) \\ &= \frac{1 + 3 \cdot (1 + 4/(3 \cdot 8^3)) + \gamma^3}{4} \cdot 8^3 \cdot s(u_i) = (8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot s(u_i) + s(u_i), \end{aligned}$$

we can rewrite $E(\mathcal{S}_{\text{opt}}^{\mathbf{f}})$ as follows:

$$E(\mathcal{S}_{\text{opt}}^{\mathbf{f}}) = (8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot \sum_{i=1}^{|U|} s(u_i) + x + \frac{4 \cdot B^3}{(B + x)^2} \quad \text{where } x = \sum_{i=1}^{|U|} b_i \cdot s(u_i).$$

It can be easily shown that $E(\mathcal{S}_{\text{opt}}^{\mathbf{f}})$ has the minimum $(8^3 + \frac{\gamma^3}{4} \cdot 8^3) \cdot \sum_{i=1}^{|U|} s(u_i) + 2 \cdot B (= K)$ at $x = B$. That is, $E(\mathcal{S}_{\text{opt}}^{\mathbf{f}}) \leq K$ if and only if

$$\begin{aligned} \exists (b_1, b_2, \dots, b_{|U|}) \in \{0, 1\}^{|U|}, \quad \sum_{i=1}^{|U|} b_i \cdot s(u_i) &= B, \text{ which is equivalent to} \\ \exists U' \in \mathbf{U}, \quad \sum_{u \in U'} s(u) &= B. \end{aligned}$$

It is obvious that the transformation can be done in polynomial time. Therefore, the problem is NP-hard. \square

From the NP-hardness proof, the problem seems unlikely to have polynomial time algorithms that compute optimal solutions. The NP-hardness of the problem strongly depends

on the fact that extremely large input numbers are allowed, as with some other NP-hard problems (e.g., the subset-sum problem and the knapsack problem [Garey and Johnson 1979]). The NP-hardness in the ordinary (but not strong) sense does not rule out possibility of existence of a pseudo-polynomial time algorithm or an FPTAS. Since our problem is an optimization problem that handles real numbers, we focus our attention on the FPTAS in the next section.

5. A FAST APPROXIMATION SCHEME

In this section, we present a fully polynomial time approximation scheme (FPTAS) for the problem. We first consider a dynamic programming formulation that always finds the optimal solution, but may run in exponential time. Then, the dynamic programming formulation is transformed into an FPTAS by using a standard technique, the *rounding-the-input-data* technique [Woeginger 1999]. The technique brings the running time of the dynamic program down to polynomial by rounding the input data so that sufficiently close input data are treated by a representative data [Sahni 1976]. The relative error of an approximation scheme depends on how we define the closeness; the smaller the threshold value for the closeness is, the smaller the relative error is. For a smaller error bound, however, the computation time becomes longer.

5.1 Algorithm for Optimal Solutions

We first present an exponential-time optimal algorithm based on the properties of optimal voltage schedules described in Section 3. The exponential-time algorithm essentially enumerates all the essential job sets. However, unlike Quan’s exhaustive algorithm [Quan and Hu 2002], it enumerates the essential job sets intelligently without actually enumerating all of them. Furthermore, it is based on dynamic programming formulation so that it can be easily transformed into an FPTAS by the standard technique.

In formulating the problem by dynamic programming, we first identify appropriate ‘overlapping’ (or reusable) subproblems to which dynamic programming can be applied iteratively. We note that the ‘optimal substructure’ of our problem is naturally reflected by *blocking tuples*, which are just sequences of time points in \mathcal{T}_j in strictly increasing order. (We formally define the blocking tuples later in this section.) That is, the optimal solution of the original problem can be built by just merging the optimal schedules of the sub-intervals defined by a blocking tuple. Figure 6 shows an example job set and its corresponding EDF-equivalent job set whose time interval is partitioned by a blocking tuple $(r_N, r_{N-3}, d_{N-1}, \dots, r_2, d_2)$, which is depicted by a set of the dashed thick lines in Figure 6.(b). Note that jobs in each sub-interval follow the EDF-priority assignment.

The original problem is partitioned into subproblems by partitioning the overall time interval into sub-intervals such that jobs in each sub-interval follow the EDF priority assignment. If a job is released within a sub-interval with its deadline outside the sub-interval, the deadline can be modified to the end of the sub-interval. Each partitioned interval can be optimally scheduled in polynomial time by Yao’s algorithm [Yao et al. 1995]. The challenge is how to find the set of sub-intervals whose optimal sub-schedules build an energy-optimal voltage schedule.

5.1.1 Basic Idea: The First Example. We now explain the basic idea of the optimal algorithm by describing the optimal algorithm on a simple but illustrative job set $\mathcal{J} = \{J_1, J_2, \dots, J_N\}$ in Figure 6.(a) where $r_{i+1} < r_i < d_{i+1} < d_i$ for $1 \leq i < N$. (Note that if

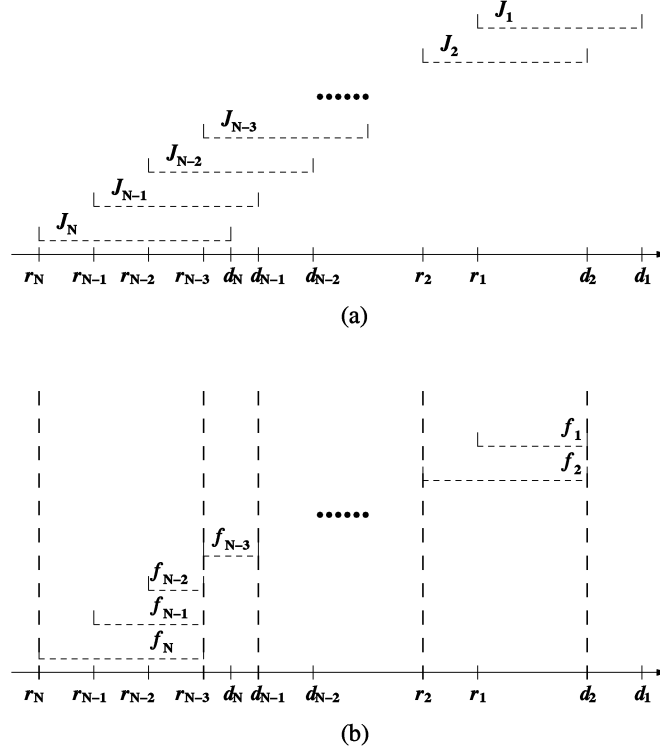


Fig. 6. An example illustrating the optimal algorithm. (a) An original job set and (b) an essential job set defined by a $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_{N-3}, f_{N-2}, f_{N-1}, f_N) = (d_2, d_2, \dots, d_{N-1}, r_{N-3}, r_{N-3}, r_{N-3})$. Jobs in each sub-interval between the thick dashed lines follows the EDF priority assignment and can be optimally scheduled by Yao's algorithm.

the priorities of jobs are reversed, the job set follows the EDF priority.) For this job set, an essential job set \mathcal{J}^e (such as one in Figure 6.(b)) is partitioned into $\mathcal{J}_1^e, \mathcal{J}_2^e, \dots, \mathcal{J}_k^e$ such that each \mathcal{J}_i^e ($1 \leq i \leq k$) follows the EDF priority assignment and the union I_i of execution intervals of jobs in \mathcal{J}_i^e (i.e., $I_i = \cup_{J \in \mathcal{J}_i^e} [r_J, d_J]$) does not overlap with I_j ($= \cup_{J \in \mathcal{J}_j^e} [r_J, d_J]$) for all $1 \leq i \neq j \leq k$. To be more concrete,

$$\text{for all } 1 \leq i < j \leq k, \forall J \in \mathcal{J}_i^e, J' \in \mathcal{J}_j^e, d_J \leq r_{J'}.$$

Therefore, the optimal voltage schedule $\mathcal{S}_{\text{opt}}^{\mathcal{J}^e}$ of \mathcal{J}^e is equal to the concatenation of the optimal voltage schedules of \mathcal{J}_i^e 's, i.e.,

$$\mathcal{S}_{\text{opt}}^{\mathcal{J}^e}(t) \equiv \oplus_{i=1}^k \mathcal{S}_{\text{opt}}^{\mathcal{J}_i^e}(t)$$

Note that $\mathcal{S}_{\text{opt}}^{\mathcal{J}_i^e}$ can be directly computed by Yao's algorithm [Yao et al. 1995] since \mathcal{J}_i^e follows the EDF priority assignment. Therefore, the energy-optimal fixed-priority voltage scheduling problem is further reduced to the problem of finding a partition that gives the energy-optimal voltage schedule for the whole time interval.

In defining a partition, we use a blocking tuple. For example, assume that f_N is selected

as r_{N-3} as in Figure 6.(b). Then, both f_{N-1} and f_{N-2} should be selected as r_{N-3} , so that the job set becomes EDF-equivalent and, furthermore, essential. As shown in Figure 6.(b), these three jobs are separated from the other jobs by a thick vertical line at time r_{N-3} . These jobs constitutes the first partitioned job set \mathcal{J}_1^e . The remaining job sets $\mathcal{J}_2^e, \dots, \mathcal{J}_k^e$ can be constructed by applying the same argument. In this way, any essential job set can be partitioned and represented by a blocking tuple.

Let $\mathbf{b} = (b_1, b_2, \dots, b_l)$ ($b_1 < b_2 < \dots < b_l$, $b_j \in T_j$) be a blocking tuple where

$$\forall 1 \leq j < l, \exists J_i \text{ s.t. } b_j = r_i \wedge b_{j+1} \leq d_i$$

Then, the corresponding EDF-ordered $|\mathcal{J}|$ -tuple $\mathbf{f} = (f_1, f_2, \dots, f_N)$ is given by

$$f_k = b_j \text{ s.t. } r_k \in [b_{j-1}, b_j) \text{ for all } 1 \leq k \leq N.$$

We call such $[b_{j-1}, b_j]$ an *atomic interval*. For example, the intervals $[r_N, r_{N-3}]$ and $[r_N, d_N]$ in Figure 6.(a) are atomic, but the interval $[r_N, d_{N-1}]$ is not atomic. (Later, we formally define the term atomic interval in arbitrary job sets other than this example.) Let t_h be the h -th earliest time point in T_j and let $\mathcal{S}_{h,g}$ represent the energy-optimal voltage schedule defined within $[t_h, t_g]$ for the job set $\mathcal{J}_{h,g}$ defined by

$$\begin{aligned} \mathcal{J}_{h,g} &= \{J'_i \mid r_{J'_i} \in [t_h, t_g)\} \text{ where} \\ r_{J'_i} &= r_{J_i}, c_{J'_i} = c_{J_i}, p_{J'_i} = p_{J_i} \text{ and } d_{J'_i} = \min\{d_{J_i}, t_g\}. \end{aligned}$$

Then, we have

$$\begin{aligned} E(\mathcal{S}_{\text{opt}}^J) &= E(\mathcal{S}_{1,|T_j|}) = \\ &\min\left\{\sum_{j=1}^{k-1} E(\mathcal{S}_{h_j, h_{j+1}}) \mid 1 = h_1 < h_2 < \dots < h_k = |\mathcal{J}| \text{ and} \right. \\ &\quad \left. [t_{h_j}, t_{h_{j+1}}] \text{ is atomic for all } j = 1, \dots, k-1\right\}. \end{aligned}$$

Given an atomic interval $[t_{h_j}, t_{h_{j+1}}]$, $\mathcal{S}_{h_j, h_{j+1}}$ can be directly computed by Yao's algorithm. In this way, the optimal voltage scheduling problem is reduced to a variant of the subset-sum problem. That is, for such job sets as in Figure 6, our problem can be formulated as follows:

Select a tuple (h_1, h_2, \dots, h_k) ($1 = h_1 < \dots < h_k = |\mathcal{J}|$) of integers such that the sum

$$q_{h_1, h_2} + q_{h_2, h_3} + \dots + q_{h_{k-1}, h_k}$$

is minimized subject to $[t_{h_i}, t_{h_{i+1}}]$ is atomic for all $1 \leq i < k$ where $q_{h_j, h_{j+1}}$ denotes $E(\mathcal{S}_{h_j, h_{j+1}})$ (which can be directly computed by Yao's algorithm).

5.1.2 Basic Idea: The Second Example. The example job set in Figure 6 is illustrative in showing how our problem can be formulated by dynamic programming. However, the easily partitionable structure comes from the fact the job set follows the 'reverse' EDF priority. For example, in Figure 6, since f_N is set to be r_{N-3} , which is within the execution intervals of J_{N-1} and J_{N-2} , f_{N-1} and f_{N-2} cannot be larger than f_N (or r_{N-3}) so that the modified job set should be EDF-equivalent. Furthermore, f_{N-1} and f_{N-2} are set to be the maximum possible value, f_N , for the modified job set to be essential.

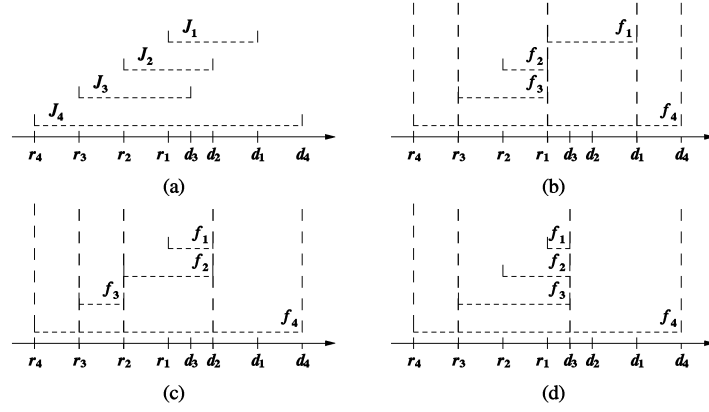
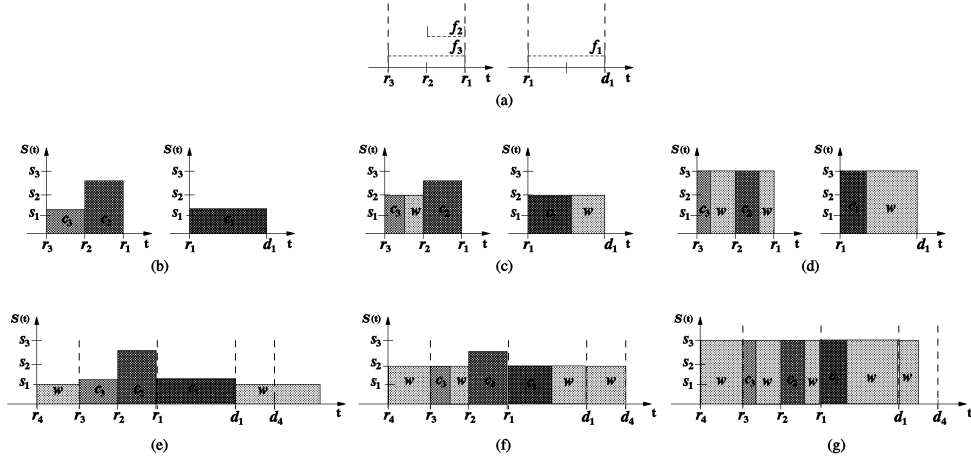


Fig. 7. An example of background workload.


 Fig. 8. An example illustrating the algorithm on a job set with background workload. (a) Atomic intervals (obtained from the job set in Figure 7.(b)). The optimal schedules for two atomic intervals where the speeds of background workload of J_4 are (b) s_1 , (c) s_2 and (d) s_3 , respectively. The voltage schedules for overall time intervals where the speeds of J_4 are (e) s_1 , (f) s_2 and (g) s_3 , respectively.

If the priority pattern is not the same as the example job set in Figure 6, the partitioning becomes difficult. For example, the essential job sets in Figures 3.(c) and 3.(d) cannot be obtained by the partitioning procedure just explained. In Figure 7.(a), J_4 has the lowest priority and the latest deadline, which makes f_4 to be d_4 for all essential job sets (Figures 7.(a), 7.(b) and 7.(c)). Therefore, any atomic interval (e.g., $[r_3, r_1]$, $[r_1, d_1]$ or $[r_3, d_3]$) contains partial workload of J_4 , which we call a *background* workload. In the following, we first explain how to extend the dynamic programming formulation to handle the *background* workload. Then, we describe how to explore essential job sets of a given arbitrary job set (as in Figure 3) by dynamic programming.

From Lemma 3.7, the job J_4 in Figure 7 runs at the same speed if the voltage schedule

```

1:  $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$ 
2:  $\mathbf{b}_\sigma := (d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$ 
3: for ( $i := |\mathcal{J}| - 1$  to 1)
4:   let  $\mathcal{J}^H$  be  $\{J_{\sigma^{-1}(k)} \mid i < k \leq |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$ 
5:   if  $(r_{J_{\sigma^{-1}(i)}} \geq \min(\{r_J \mid J \in \mathcal{J}^H\} \cup \{f_{\sigma^{-1}(i+1)}\}))$  return FALSE
6:   else  $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J \mid J \in \mathcal{J}^H\})$ 
7:   end if
8:   

|                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>if</b> $(f_{\sigma^{-1}(i)} \leq \min\{r_{J_{\sigma^{-1}(k)}} \mid i < k \leq  \mathcal{J} \})$ <b>append</b> $f_{\sigma^{-1}(i)}$ <b>onto the head of</b> $\mathbf{b}_\sigma$ |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|


9:   end if
10: end for
11: append  $\min R_J$  onto the head of  $\mathbf{b}_\sigma$ 

```

Fig. 9. The algorithm to build a strongly-blocking tuple from a $|\mathcal{J}|$ -permutation.

is an optimal one. For the time being, let us assume that the constant speed is among $S_C = \{s_1, s_2, s_3\}$. (For now, S_C is set to be the set of all the possible constant speeds in the optimal voltage schedule. In section 5.2, we explain how the set S_C is selected such that the size of S_C is bounded by a polynomial function.) For each $s_i \in S_C$, we first compute the amount of background workload of J_4 for each atomic interval, and then find the minimum-energy essential job set (among those in Figures 7.(b), 7.(c) and 7.(d)) by using the similar procedure to the previous case in Figure 6. However, unlike the previous case, we discard any job set for which the sum of background workloads executed in overall time interval is less than the total workload of J_4 .

Figure 8.(a) shows the atomic intervals $[r_3, r_1]$ and $[r_1, d_1]$, which are obtained from the essential job set in Figure 7.(b). Figures 8.(b), 8.(c) and 8.(d) show the optimal voltage schedules for the atomic intervals where J_4 runs at the speed s_1 , s_2 and s_3 , respectively. The workloads of jobs J_1 , J_2 and J_3 are denoted by c_1 , c_2 and c_3 , respectively, and the background workloads are denoted by w . The amount of the background workload (and the resultant optimal voltage schedule) for each atomic interval and speed can be easily computed by a slightly modified version of Yao's algorithm [Yao et al. 1995]. That is, when the critical interval is selected, if the speed to be assigned (by the intensity of the critical interval) is less than or equal to the speed of the background workload, we assign the speed of the background workload to all the unscheduled time intervals (including the critical interval). Then, the amount of background workload can be directly computed as in Figure 8.(b), 8.(c) and 8.(d).

Once the background workload and the optimal voltage schedule is computed for each atomic interval, we apply the same procedure as in the job set in Figure 6 to find the minimum-energy essential job set and the energy-optimal voltage schedule. In exploring the solution space, we should discard any infeasible schedules. Figure 8.(e) shows an infeasible schedule where J_4 runs at s_1 and cannot complete its execution until its deadline. The voltage schedule in Figure 8.(g) is feasible, but not an optimal one. Thus, only the schedule in Figure 8.(f) is not removed in the pruning procedure and is compared with another schedules obtained from the essential job sets in Figures 8.(c) and 8.(d).

5.1.3 Putting It Altogether. We now describe the optimal algorithm for arbitrary job sets based on the observations from the example job sets. First, we formally define the terms *strongly-atomic interval* and *strongly-blocking tuple*. Given a valid $|\mathcal{J}|$ -permutation

```

/*  $T_j = \{t_1, t_2, \dots, t_N\}$  */
1: foreach (strongly-atomic interval  $[t_i, t_j]$ )
2:      $g_{i,j} := E(\mathcal{S}_{\text{opt}}^{J_{[t_i, t_j]}})$ 
3: end foreach
4:  $\mathbf{V} := \{v_1, v_2, \dots, v_N\}$ 
5:  $\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j] \text{ is strongly-atomic}\}$ 
6: foreach  $((v_i, v_j) \in \mathbf{E})$   $w((v_i, v_j)) := g_{i,j}$  end foreach /* weight of edges */
7: Find the shortest path from  $v_1$  to  $v_N$  in  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ . /* Note that  $\mathbf{G}$  is acyclic. */
/* The shortest path =  $\langle v_{q_1}, v_{q_2}, \dots, v_{q_k} \rangle$  ( $v_{q_1} = v_1, v_{q_k} = v_N$ ) */
8: return  $\bigoplus_{j=1}^{k-1} \mathcal{S}_{\text{opt}}^{J_{[v_{q_j}, v_{q_{j+1}]}}$ 

```

Fig. 10. An exponential-time optimal algorithm based on strongly-atomic intervals.

σ , the algorithm in Figure 9 builds the corresponding strongly-blocking tuple $\mathbf{b}_\sigma = (b_1, b_2, \dots, b_k)$ where $b_1 < b_2 < \dots < b_k$ and $b_i \in T_j$ for all $1 \leq i \leq k$. The algorithm is identical to the algorithm in Figure 2 except for lines 2, 8, 9 and 11. In line 8, $f_{\sigma^{-1}(j)}$ is selected as an element of a strongly-blocking tuple if it partitions the execution interval.

Definition 5.1. Given a valid $|\mathcal{J}|$ -permutation σ , the tuple \mathbf{b}_σ built by the algorithm in Figure 9 is called a *strongly-blocking tuple*. An interval $[t, t']$ is *strongly-atomic* if there is a strongly-blocking tuple $\mathbf{b} = (b_1, b_2, \dots, b_k)$ such that $[t, t'] = [b_i, b_{i+1}]$ for some $1 \leq i < k$. Furthermore, the job set $\mathcal{J}_{[t, t']}$ defined by

$$\mathcal{J}_{[t, t']} = \{J' \mid J \in \mathcal{J}, r_J \in [t, t']\} \text{ where}$$

$$r_{J'} = r_J, c_{J'} = c_J, p_{J'} = p_J \text{ and } d_{J'} = \min\{d_J, t'\}.$$

is said to be *induced* by an interval $[t, t']$.⁴

For the job set in Figure 3 not only $[r_3, r_2], [r_2, d_2]$ (Figure 3.(b)) and $[r_3, r_1]$ (Figure 3.(c)) but also $[r_1, d_2]$ (Figure 3.(c)) and $[r_3, d_3]$ (Figure 3.(d)) are strongly-atomic. Note that the intervals $[r_1, d_2]$ and $[r_3, d_3]$ are not covered by the previous definition in Section 5.1.1. Furthermore, (r_3, r_2, d_2) (Figure 3.(b)), (r_3, r_1, d_2) (Figure 3.(c)) and (r_3, d_3) (Figure 3.(d)) are strongly-blocking tuples.

Figure 10 shows an optimal algorithm which is based on strongly-atomic intervals. The correctness of the algorithm is proved in Appendix A.1. The algorithm may work efficiently for some job sets (e.g., the job set in Figure 6). But, the running time may not be bounded by a polynomial function; For the job set in Figure 7, there are only one strongly-atomic interval $[r_4, d_4]$ and the algorithm cannot but enumerate all the essential job sets. Furthermore, the algorithm does not have a structure suitable to be transformed into an FPTAS. So, we consider another optimal algorithm based on *weakly-atomic* intervals, *weakly-bounding* tuples, and the background workload. First, we formally define the terms based on the algorithm in Figure 11, which is identical to the algorithm in Figure 9 except for the boxed code segment (line 8).

Definition 5.2. Given a valid $|\mathcal{J}|$ -permutation σ , the tuple \mathbf{b}_σ^w built by the algorithm in Figure 11 is called a *weakly-blocking tuple*. An interval $[t, t']$ is *weakly-atomic* if there is a

⁴ $[t, t']$ is not required to be strongly-atomic.

```

1:  $f_{\sigma^{-1}(|\mathcal{J}|)} := d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$ 
2:  $\mathbf{b}_{\sigma}^w := (d_{J_{\sigma^{-1}(|\mathcal{J}|)}}$ 
3: for ( $i := |\mathcal{J}| - 1$  to 1)
4:   let  $\mathcal{J}^H$  be  $\{J_{\sigma^{-1}(k)} \mid i < k \leq |\mathcal{J}| \wedge \sigma^{-1}(k) < \sigma^{-1}(i)\}$ 
5:   if ( $r_{J_{\sigma^{-1}(i)}} \geq \min(\{r_J \mid J \in \mathcal{J}^H\} \cup \{f_{\sigma^{-1}(i+1)}\})$ ) return FALSE
6:   else  $f_{\sigma^{-1}(i)} := \min(\{f_{\sigma^{-1}(i+1)}, d_{J_{\sigma^{-1}(i)}}\} \cup \{r_J \mid J \in \mathcal{J}^H\})$ 
7:   end if
8:   if ( $f_{\sigma^{-1}(i)} \leq \min\{r_J \mid r \in \mathcal{J}^H\}$ ) append  $f_{\sigma^{-1}(i)}$  onto the head of  $\mathbf{b}_{\sigma}^w$ 
9:   end if
10: end for
11: append  $\min R_{\mathcal{J}}$  onto the head of  $\mathbf{b}_{\sigma}^w$ 

```

Fig. 11. The algorithm to build a weakly-blocking tuple from a $|\mathcal{J}|$ -permutation.

weakly-blocking tuple $\mathbf{b}^w = (b_1, b_2, \dots, b_k)$ such that $[t, t'] = [b_i, b_{i+1}]$ for some $1 \leq i < k$. Furthermore, the job set $\mathcal{J}_{[t, t']^w}$ defined by

$$\mathcal{J}_{[t, t']^w} = \{J' \mid J \in \mathcal{J}, r_J \in [t, t'] \wedge (\exists J_h \in \mathcal{J}, p_{J_h} < p_J \wedge r_{J_h} = t' \wedge d_J \in [r_{J_h}, d_{J_h}])\} \text{ where } r_{J'} = r_J, c_{J'} = c_J, p_{J'} = p_J \text{ and } d_{J'} = \min\{d_J, t'\}.$$

is said to be *weakly-induced* by an interval $[t, t']$.

Furthermore, given a weakly-blocking tuple $\mathbf{b}^w = (b_1, b_2, \dots, b_k)$ and the corresponding EDF-equivalent job set \mathcal{J}' , any job in $\mathcal{J}' - \bigcup_{j=1}^{k-1} \mathcal{J}_{[b_j, b_{j+1}]^w}$ is called a *background job* with respect to the weakly-blocking tuple \mathbf{b}^w . The workload of background jobs are called *background workload*.

Based on weakly-atomic interval, we construct another optimal voltage scheduling algorithm. Figure 12 shows the optimal algorithm which is based on the dynamic programming formulated by weakly-atomic intervals. The algorithm identifies weakly-atomic intervals and computes the optimal schedule for the weakly-atomic interval. (Note that jobs in a weakly-atomic interval follow the EDF priority assignment.) In computing the optimal schedule for a weakly-atomic interval, we consider the background workload, that is, the algorithm computes the optimal schedule for each candidate background speed in S_C . Given a job set \mathcal{J} , the algorithm first computes the set S_C of candidates for the speed of background workload. For the optimal algorithm, the set S_C is set to be

$$S_C = \left\{ \frac{C(\mathcal{J}')}{\sum_{i=0}^{k-1} (t_{p_{2i+2}} - t_{p_{2i+1}})} \mid \mathcal{J}' \subset \mathcal{J}, t_1 < t_2 < \dots < t_{p_{2k}}, t_j \in T_{\mathcal{J}} \right\}.$$

It is obvious that the speed of the background workload in an optimal voltage schedule is included in S_C . (In the FPTAS which will be presented in Section 5.2, the set S_C is selected such that the size of S_C is bounded by a polynomial function.) Given the optimal schedules of weakly-atomic intervals, the algorithm searches the minimum sum of the energy values of the weakly-atomic intervals. The correctness of the algorithm is proved in Appendix A.2. The worst-case running time of the algorithm is not bounded by a polynomial function, but it can be easily transformed into an FPTAS.

5.2 Approximation Algorithm

First, we prove a miscellaneous property which is useful in bounding the error of our approximation algorithm.

```

procedure OPTIMAL_VOLTAGE_SCHEDULE ( $\mathcal{J}$ )
    /*  $T_j = \{t_1, t_2, \dots, t_N\}$ ,  $S_C := \{s_1, s_2, \dots, s_n\}$  */
    1: foreach ( $s \in S_C$ )
    2:    $\mathbf{V} := \{v_1, v_2, \dots, v_N\}$ 
    3:    $\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j] \text{ is weakly-atomic}\}$ 
    4:   foreach ( $(v_i, v_j) \in \mathbf{E}$ )
    5:      $w((v_i, v_j)) := W(\max\{\mathcal{S}_{\text{opt}}^{\mathcal{J}[t_i, t_j]}(t), s\}, [t_i, t_j]) - W(\mathcal{S}_{\text{opt}}^{\mathcal{J}[t_i, t_j]}(t), [t_i, t_j])$  /* weight of edges */
    6:   end foreach
    7:   Find longest paths between all pairs of vertices in  $\mathbf{V}$ . /* Note that  $G$  is acyclic. */
    8:   foreach ( $1 \leq i < j \leq N$  s.t.  $[t_i, t_j]$  is a concatenation of weakly-atomic intervals)
    9:     /* The longest path from  $v_i$  to  $v_j = \langle v_{q_1}, v_{q_2}, \dots, v_{q_l} \rangle$ 
    10:     $c :=$  the weight of the longest path from  $v_i$  to  $v_j$ .
    11:     $E_{i,j}[c] := E(\oplus_{h=1}^{l-1} \max\{\mathcal{S}_{\text{opt}}^{\mathcal{J}[t_{q_j}, t_{q_{j+1}}]}(t), s\}, [t_i, t_j])$ 
    12:  end foreach
    13:  for ( $i := 1$  to  $N-1$ )
    14:    for ( $j := 1$  to  $N-i$ )
    15:       $E_{j,j+i} := \infty^+$ 
    16:      for ( $k := j+1$  to  $j+i$ )
    17:         $c_{j,j+i,k} := C(\{J \in \mathcal{J}^B \mid r_J \in [t_j, t_k] \wedge d_J \in [t_k, t_{j+i}]\})$ 
    18:         $E_{j,j+i,k} := E_{j,k}[c_{j,j+i,k}] + E_{k,j+i}$ 
    19:        if ( $E_{j,j+i} > E_{j,j+i,k}$  and  $\mathcal{S}_{\text{opt}}^{\mathcal{J}[t_j, t_k]^w [c_{j,j+i,k}]}$  is feasible for  $\mathcal{J}_{[t_i, t_j]^w} \cup \{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_i, t_j]\}$ )
    20:           $E_{j,j+i} := E_{j,j+i,k}$ ,  $h := k$ 
    21:        end if
    22:      end for
    23:       $\mathbf{b}_{j,j+i} := \{t_h\} \cup \mathbf{b}_{j,h} \cup \mathbf{b}_{h,j+i}$ 
    24:    end for
    25:  end for
    /*  $E_{1,N} = E(\mathcal{S}_{\text{opt}}^{\mathcal{J}})$  and  $\mathcal{S}_{\text{opt}}^{\mathcal{J}} \equiv \mathcal{S}_{\text{opt}}^{\cup_{h=1}^{j-1} \mathcal{J}[b_h, b_{h+1}]^w \cup \mathcal{J}^B}$  where  $\mathbf{b}_{1,N} = (b_1, b_2, \dots, b_i)$  */
    26:   $\mathcal{J}_{\text{opt}} := \cup_{h=1}^{j-1} \mathcal{J}[b_h, b_{h+1}]^w \cup \mathcal{J}^B$  where  $\mathbf{b}_{1,N} = (b_1, b_2, \dots, b_i)$ 
    27:  return  $\mathcal{S}_{\text{opt}}^{\mathcal{J}_{\text{opt}}}$  /*  $\mathcal{J}_{\text{opt}}$  is an EDF job set. So,  $\mathcal{S}_{\text{opt}}^{\mathcal{J}_{\text{opt}}}$  can be directly computed by Yao's algorithm */
end procedure
    
```

Fig. 12. An exponential-time optimal algorithm based on weakly-atomic intervals.

LEMMA 5.3. Given a function $P : \mathbf{R}^+ \Rightarrow \mathbf{R}^+$ and a constant $0 < \varepsilon < 1$, if

$$0 < x_1 < x_2 < \left(1 + \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) \mid x > 0\}}\right) \cdot x_1 \quad \text{where} \quad \eta(x) = \frac{P'(x)}{P(x)} \cdot x,$$

then $P(x_2) < (1 + \varepsilon) \cdot P(x_1)$.

PROOF. From the condition, we have

$$\log x_2 - \log x_1 < \log \left(1 + \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) \mid x > 0\}}\right) < \frac{\varepsilon \cdot \log 2}{\max\{\eta(x) \mid x > 0\}}. \quad (11)$$

Let $y_1 = \log x_1$ and $y_2 = \log x_2$. Then we have

$$\log P(x_2) - \log P(x_1) = \log P(e^{y_2}) - \log P(e^{y_1}) \leq (y_2 - y_1) \cdot \max\left\{\frac{d(\log P(e^y))}{dy}\right\}.$$

From (11) and

$$\frac{d(\log P(e^y))}{dy} = \frac{P'(e^y)}{P(e^y)} \cdot e^y = \eta(e^y),$$

we have

$$\log P(x_2) - \log P(x_1) < \frac{\varepsilon \cdot \log 2}{\max\{\eta(x)|x > 0\}} \cdot \max\{\eta(x) | x > 0\} = \varepsilon \cdot \log 2 .$$

It follows that

$$P(x_2) < e^{\varepsilon \cdot \log 2} \cdot P(x_1) < e^{\log(1+\varepsilon)} \cdot P(x_1) = (1+\varepsilon) \cdot P(x_1) .$$

□

For a power function $P(s) = \alpha \cdot s^n$, we have $\eta(s) = n$. In the following, we use ρ_P to denote $\log 2 / \max\{\eta(x)|x > 0\}$. From Lemma 5.3, we can construct an FPTAS as in Figure 13. The FPTAS is slightly different from the algorithm in Figure 12. To bring the running time down to polynomial, we use S'_C instead of S_C :

$$S'_C = \{ \min\{S_C\} \cdot (1 + \varepsilon \cdot \rho_P)^k \mid k = 0, 1, \dots, l \text{ where} \\ \min\{S_C\} \cdot (1 + \varepsilon \cdot \rho_P)^{l-1} < \max\{S_C\} \leq \min\{S_C\} \cdot (1 + \varepsilon \cdot \rho_P)^l \} .$$

THEOREM 5.4. *APPROX_VOLTAGE_SCHEDULE is a fully polynomial time approximation scheme for the voltage scheduling problem.*

PROOF. Let s_1 and s_2 be elements of S'_C such that $s_2 = s_1 \cdot (1 + \varepsilon \cdot \rho_P)$. Given a weakly-atomic interval $[t_i, t_j]$, we have for $t_i \leq t \leq t_j$:

$$\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_2\} \leq (1 + \varepsilon \cdot \rho_P) \cdot \max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_1\} .$$

Thus, from Lemma 5.3, we have for $t_i \leq t \leq t_j$

$$P(\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_2\}) \leq (1 + \varepsilon) \cdot P(\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_1\}) \quad , \text{ which implies} \\ E(\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_2\}, [t_i, t_j]) \leq (1 + \varepsilon) \cdot E(\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s_1\}, [t_i, t_j]) .$$

Let us compare $E_{j,k}[c']$ in line 21 of APPROX_VOLTAGE_SCHEDULE and $E_{j,k}[c_{j,j+i,k}]$ in line 18 of OPTIMAL_VOLTAGE_SCHEDULE. Let s' and s be the corresponding elements in S'_C and S_C , respectively. Then, from the definition of S'_C , we have $s' < (1 + \varepsilon \cdot \rho_P) \cdot s$, which implies $E_{j,k}[c'] < (1 + \varepsilon) \cdot E_{j,k}[c_{j,j+i,k}]$. Therefore, $E_{1,N} < (1 + \varepsilon) \cdot E(S_{\text{opt}}^{\mathcal{J}})$. Finally, since we have

$$|S'_C| = 1 + \lceil \log_{1+\varepsilon \cdot \rho_P}(\max\{S_C\} / \min\{S_C\}) \rceil \quad (12)$$

$$< 2 + \frac{\log(\max\{S_C\} / \min\{S_C\})}{\varepsilon \cdot \log(1 + \rho_P)} , \quad (13)$$

the running time is bounded a polynomial function of $|\mathcal{J}|$ and $1/\varepsilon$. □

6. EXPERIMENTAL RESULTS

In order to evaluate how the proposed FPTAS performs, we have performed several experiments using the FPTAS described in Figure 13. For a comparison, we also implemented Quan's heuristic [Quan and Hu 2001], which is currently the best polynomial-time voltage

```

procedure APPROX_VOLTAGE_SCHEDULE ( $\mathcal{J}, \varepsilon$ )
    /*  $T_j = \{t_1, t_2, \dots, t_N\}$  */
    /*  $S'_C = \{\min\{S_C\} \cdot (1 + \delta)^k \mid k = 0, 1, \dots, \lceil \log_{1+\delta}(\max\{S_C\} / \min\{S_C\}) \rceil\}$  where  $\delta = \varepsilon \cdot \rho_P$  */
    1: Initialize  $C_{i,j} := \{\}$  for  $1 \leq i < j \leq N$ .
    2: foreach ( $s \in S'_C$ )
    3:      $\mathbf{V} := \{v_1, v_2, \dots, v_N\}$ 
    4:      $\mathbf{E} := \{(v_i, v_j) \mid [t_i, t_j] \text{ is weakly-atomic}\}$ 
    5:     foreach ( $(v_i, v_j) \in \mathbf{E}$ )
    6:          $w((v_i, v_j)) := W(\max\{S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), s\}, [t_i, t_j]) - W(S_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}}(t), [t_i, t_j])$  /* weight of edges */
    7:     end foreach
    8:     Find longest paths between all pairs of vertices in  $\mathbf{V}$ . /* Note that  $G$  is acyclic. */
    9:     foreach ( $1 \leq i < j \leq N$  s.t.  $[t_i, t_j]$  is a concatenation of weakly-atomic intervals)
    10:        /* The longest path from  $v_i$  to  $v_j = \langle v_{q_1}, v_{q_2}, \dots, v_{q_l} \rangle$ 
    11:         $c :=$  the weight of the longest path from  $v_i$  to  $v_j$ .
    12:         $E_{i,j}[c] := E(\oplus_{h=1}^{l-1} \max\{S_{\text{opt}}^{\mathcal{J}_{[q_j, q_{j+1}]}}(t), s\}, [t_i, t_j])$ 
    13:         $C_{i,j} := C_{i,j} \cup \{c\}$ 
    14:    end foreach
    15: end foreach
    16: for ( $i := 1$  to  $N - 1$ )
    17:     for ( $j := 1$  to  $N - i$ )
    18:          $E_{j,j+i} := \infty^+$ 
    19:         for ( $k := j + 1$  to  $j + i$ )
    20:              $c_{j,j+i,k} := C(\{J \in \mathcal{J}^B \mid r_J \in [t_j, t_k] \wedge d_J \in [t_k, t_{j+i}]\})$ 
    21:              $c' := \min\{c \in C_{j,k} \mid c \geq c_{j,j+i,k}\}$ 
    22:              $E_{j,j+i,k} := E_{j,k}[c'] + E_{k,j+i}$ 
    23:             if ( $E_{j,j+i} > E_{j,j+i,k}$  and
    24:                  $S_{\text{opt}}^{\mathcal{J}_{[t_j, t_k]}^w [c_{j,j+i,k}]}$  is feasible for  $\mathcal{J}_{[t_i, t_j]}^w \cup \{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_i, t_j]\}$ )
    25:                  $E_{j,j+i} := E_{j,j+i,k}$ ,  $h := k$ 
    26:             end if
    27:         end for
    28:          $\mathbf{b}_{j,j+i} := \{t_h\} \cup \mathbf{b}_{j,h} \cup \mathbf{b}_{h,j+i}$ 
    29:     end for
    30: end for
    31:     /*  $E_{1,N} < (1 + \varepsilon) \cdot E(S_{\text{opt}}^{\mathcal{J}})$  */
    32:      $\mathcal{J}_\varepsilon := \cup_{h=1}^{l-1} \mathcal{J}_{[b_h, b_{h+1}]}^w \cup \mathcal{J}^B$  where  $\mathbf{b}_{1,N} = (b_1, b_2, \dots, b_l)$ 
    33:     return  $S_{\text{opt}}^{\mathcal{J}_\varepsilon}$  /*  $\mathcal{J}_\varepsilon$  is an EDF job set. So,  $S_{\text{opt}}^{\mathcal{J}_\varepsilon}$  can be directly computed by Yao's algorithm */
end procedure
    
```

Fig. 13. The fully polynomial time approximation scheme.

scheduling algorithm for fixed-priority real-time tasks. We compared the energy efficiency and computation time between two algorithms.⁵

In our experiments, we assumed that the energy consumption is quadratically dependent on the supply voltage. For a given supply voltage V , the corresponding clock frequency f is proportional to $(V_{\text{DD}} - V_{\text{TH}})^\alpha / V_{\text{DD}}$, where V_{TH} and α are assumed to be 0.5V and 1.3 [Sakurai and Newton 1990].

We constructed test job sets from periodic task sets of three real-world applications:

⁵We have implemented the exhaustive optimal algorithm by Quan *et al.* [Quan and Hu 2002] as well for experiments. This algorithm, however, takes an excessive amount of time. For example, it took more than a day when $N = 25$. Therefore, we cannot include the experimental results for this algorithm.

MPEG4 Videophone [Shin et al. 2001], CNC [Kim et al. 1996] and Avionics [Locke et al. 1991]. Table I summarizes the experimental results for these job sets. In each experiment, the execution time of each job (i.e., task instance) was randomly drawn from a Gaussian distribution⁶ within the range of $[\text{WCET}/10, \text{WCET}]$ of each task. Results were normalized over the energy consumption of each application scheduled by the proposed FPTAS with $\epsilon = 0.1\%$. As shown in Table I, the FPTAS outperforms Quan’s algorithm spending reasonable CPU times. In the experiments, actual errors were always less than given ϵ ’s. (We omit CPU times for MPEG4 Videophone because they are less than 0.1 seconds.)

Applications		Normalized Energy			CPU Time(s)	
		MPEG4	CNC	Avionics	CNC	Avionics
# jobs		22	289	1372	289	1372
FPTAS	$\epsilon = 0.1\%$	1	1	1	44.71	4506.63
	$\epsilon = 0.5\%$	1.003	1.004	1.003	11.67	1021.48
	$\epsilon = 1.0\%$	1.006	1.008	1.007	6.12	631.15
	$\epsilon = 1.5\%$	1.012	1.013	1.011	5.16	512.32
	$\epsilon = 2.0\%$	1.017	1.018	1.018	3.81	313.15
Quan [Quan and Hu 2001]		1.041	1.062	1.059	4.76	580.32

Table I. Experimental results for three real-world real-time applications.

We also performed experiments using synthesized job sets with the varying number of jobs from 50 to 1600. We conjectured that one of the key parameters affecting the performance of Quan’s algorithm is the degree of interferences among jobs. Since the degree of interferences is mainly dependent on the lengths of the execution intervals of the jobs, we generated three classes of job sets as follows: For the first class of job sets (**Class 1**), the release time and the length of the execution interval of a job are selected under the uniform distribution within $[0, 1000]$ and $[50, 100]$, respectively. The workload of each job was randomly selected from a uniform distribution within $[0.2, 1.0]$. (Note that it is sufficient to consider only the relative values of workloads since the maximum processor speed can be always appropriately adjusted.) For the second class of jobs (**Class 2**) and the third class of jobs (**Class 3**), we used $[100, 300]$ and $[300, 500]$ (instead of $[50, 100]$) for the length of the execution interval, respectively. Note that **Class 1**, **Class 2** and **Class 3** correspond to job sets with low, medium and high degrees of the interferences among the jobs. Tables II, III and IV show the experimental results for **Class 1**, **Class 2** and **Class 3**. As shown in tables, in general, the higher the degree of interferences becomes, the larger the improvement of our algorithm over Quan’s algorithm becomes.

7. CONCLUSIONS

We investigated the problem of energy-optimal voltage scheduling for fixed-priority real-time systems implemented on a variable voltage processor. First, we proved the NP-hardness of the problem. Our complexity analysis provided an important new insight into the problem.

Knowing the NP-hardness of the problem, as the best practical solution, we described a fully polynomial time approximation scheme for the problem. That is, for any $\epsilon > 0$, the proposed approximation scheme computes a voltage schedule whose energy consumption

⁶With the mean $m = \frac{\text{WCET}/10 + \text{WCET}}{2}$ and the standard deviation $\sigma = \frac{\text{WCET} - \text{WCET}/10}{6}$.

		Normalized Energy					
Job sets		\mathcal{J}_1	\mathcal{J}_2	\mathcal{J}_3	\mathcal{J}_4	\mathcal{J}_5	\mathcal{J}_6
# jobs		50	100	200	400	800	1600
FPTAS	$\epsilon = 0.1\%$	1	1	1	1	1	1
	$\epsilon = 0.5\%$	1.003	1.003	1.004	1.004	1.003	1.003
	$\epsilon = 1.0\%$	1.008	1.007	1.009	1.009	1.008	1.009
	$\epsilon = 1.5\%$	1.013	1.012	1.012	1.014	1.014	1.014
	$\epsilon = 2.0\%$	1.016	1.016	1.019	1.018	1.019	1.019
Quan [Quan and Hu 2001]		1.044	1.047	1.051	1.054	1.052	1.071

Table II. Experimental results for synthesized jobs (Class 1).

		Normalized Energy					
Job sets		\mathcal{J}_1	\mathcal{J}_2	\mathcal{J}_3	\mathcal{J}_4	\mathcal{J}_5	\mathcal{J}_6
# jobs		50	100	200	400	800	1600
FPTAS	$\epsilon = 0.1\%$	1	1	1	1	1	1
	$\epsilon = 0.5\%$	1.004	1.004	1.003	1.004	1.003	1.004
	$\epsilon = 1.0\%$	1.009	1.007	1.007	1.008	1.009	1.009
	$\epsilon = 1.5\%$	1.013	1.012	1.014	1.014	1.013	1.014
	$\epsilon = 2.0\%$	1.018	1.016	1.018	1.018	1.019	1.019
Quan [Quan and Hu 2001]		1.055	1.062	1.070	1.079	1.103	1.127

Table III. Experimental results for synthesized jobs (Class 2).

		Normalized Energy					
Job sets		\mathcal{J}_1	\mathcal{J}_2	\mathcal{J}_3	\mathcal{J}_4	\mathcal{J}_5	\mathcal{J}_6
# jobs		50	100	200	400	800	1600
FPTAS	$\epsilon = 0.1\%$	1	1	1	1	1	1
	$\epsilon = 0.5\%$	1.004	1.004	1.004	1.003	1.004	1.004
	$\epsilon = 1.0\%$	1.009	1.007	1.007	1.009	1.008	1.009
	$\epsilon = 1.5\%$	1.014	1.013	1.014	1.013	1.014	1.014
	$\epsilon = 2.0\%$	1.018	1.017	1.019	1.018	1.019	1.019
Quan [Quan and Hu 2001]		1.094	1.114	1.121	1.134	1.142	1.137

Table IV. Experimental results for synthesized jobs (Class 3).

is bounded by $(1 + \epsilon)$ times of that of the optimal voltage schedule. Furthermore, the running time of the proposed approximation scheme is bounded as well by a polynomial function of the number of input jobs and $1/\epsilon$. Experimental results show that the proposed approximation scheme runs sufficiently fast even for a small error bound (i.e., 0.5%).

While the proposed approximation scheme is efficient for general fixed-priority job sets, the proposed scheme can be further extended in several directions. For example, we are interested in devising more efficient algorithms for more specialized job sets such as job sets from periodic task sets. In addition, we plan to modify the proposed approximation scheme to work under a more realistic processor model with a limited number of voltage levels and voltage transition overheads.

A. APPENDIX: PROOFS

A.1 Proof of the Correctness of the Algorithm in Figure 10

We first prove some properties on strongly-blocking tuples and strongly-atomic intervals. Note that for an interval $[t, t']$, $\mathbf{I}_{\mathcal{J}_{[t, t']}} \subseteq [t, t']$ since $t \leq r_J < d_J \leq t'$ for all $J \in \mathcal{J}_{[t, t']}$. Therefore, for a strongly-blocking tuple $\mathbf{b} = (b_1, b_2, \dots, b_k)$, $\mathbf{I}_{\mathcal{J}_{[b_1, b_2]}}$, $\mathbf{I}_{\mathcal{J}_{[b_2, b_3]}}$, \dots , $\mathbf{I}_{\mathcal{J}_{[b_{k-1}, b_k]}}$ are disjoint. Now, we prove that a job set can be partitioned by strongly-blocking tuples as with the job set in Figure 6 so that the formulation described in Section 5.1.1 can be extended to cover arbitrary job sets.

LEMMA A.1. *Given a job set \mathcal{J} and an essential $|\mathcal{J}|$ -tuple \mathbf{f} , $\mathcal{J}^{\mathbf{f}} \equiv \cup_{j=1}^{k-1} \mathcal{J}_j$ where $\mathbf{b}_{\sigma_{\mathbf{f}}} = (b_1, b_2, \dots, b_k)$ and \mathcal{J}_j is an EDF-equivalent job set of $\mathcal{J}_{[b_j, b_{j+1}]}$ for all $1 \leq j < k$.*

PROOF. Let $\mathcal{J}^{\mathbf{f}} = \{J'_1, J'_2, \dots, J'_{|\mathcal{J}|}\}$ and let $\mathcal{J}_j = \{J'_l \in \mathcal{J}^{\mathbf{f}} \mid r_{J'_l} (= r_{J_l}) \in [b_j, b_{j+1}]\}$. Then, $\{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{k-1}\}$ forms a partition of $\mathcal{J}^{\mathbf{f}}$, i.e.,

$$\mathcal{J}^{\mathbf{f}} \equiv \cup_{j=1}^{k-1} \mathcal{J}_j \quad \text{and} \quad \mathcal{J}_j \cap \mathcal{J}_{j'} = \emptyset \quad \text{for all } 1 \leq j \neq j' < k.$$

Thus, it suffices to show that \mathcal{J}_j is an EDF-equivalent job set of $\mathcal{J}_{[b_j, b_{j+1}]}$ for all $1 \leq j < k$. Let $i_j = \max\{i \mid f_{\sigma^{-1}(i)} = b_j\}$ for all $1 \leq j \leq k$, and suppose that $d_{J'_l} > b_{j+1}$ for a job $J'_l \in \mathcal{J}_j$. Then, we have $\sigma(l) > i_{j+1}$ since

$$f_{\sigma^{-1}(\sigma(l))} = f_l = d_{J'_l} > b_{j+1} = f_{\sigma^{-1}(i_{j+1})}.$$

From line 8 of the algorithm in Figure 9, we have

$$b_{j+1} = f_{\sigma^{-1}(i_{j+1})} \leq \min\{r_{J_{\sigma^{-1}(k)}} \mid i_{j+1} < k \leq |\mathcal{J}|\} \leq r_{J_{\sigma^{-1}(k)}} \Big|_{k=\sigma(l)}^{(>i_{j+1})} = r_{J_l},$$

which contradicts $r_{J'_l} (= r_{J_l}) \in [b_j, b_{j+1})$. Therefore, $d_{J'_l} \in [b_j, b_{j+1}]$ for all $J'_l \in \mathcal{J}_j$. Furthermore, \mathcal{J}_j follows the EDF priority since it is a subset of the EDF job set $\mathcal{J}^{\mathbf{f}}$.

It remains to show that $|\mathcal{J}_j| = |\mathcal{J}_{[b_j, b_{j+1}]}|$ and there is a bijective function $\alpha: \mathcal{J}_{[b_j, b_{j+1}]} \Rightarrow \mathcal{J}_j$ such that

$$\forall J' \in \mathcal{J}_{[b_j, b_{j+1}]}, \quad p_{J'} = p_{\alpha(J')}, \quad c_{J'} = c_{\alpha(J')} \quad \text{and} \quad r_{J'} = r_{\alpha(J')}. \quad (14)$$

For the former, we have

$$|\mathcal{J}_j| = |\{J' \in \mathcal{J}^{\mathbf{f}} \mid r_{J'} \in [b_j, b_{j+1}]\}| = |\{J \in \mathcal{J} \mid r_J \in [b_j, b_{j+1}]\}| = |\mathcal{J}_{[b_j, b_{j+1}]}|.$$

For the latter, we define α such that $\alpha(J') = J''$ iff $p_{J'} = p_{J''}$. Then, it is clear that α is a bijective function and (14) holds. \square

LEMMA A.2. *Let $S(t) = \oplus_{j=1}^{h-1} S_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}}$ for $\min R_{\mathcal{J}} = t_1 < t_2 < \dots < t_h = \max D_{\mathcal{J}}$ ($t_j \in T_{\mathcal{J}}$). Then, S is a feasible voltage schedule of \mathcal{J} . Furthermore,*

$$E(S) = \sum_{j=1}^{h-1} E(S_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}}) \geq E(S_{\text{opt}}^{\mathcal{J}}).$$

PROOF. Let $u_{[t_0, t'_0]}(t)$ be defined by

$$u_{[t_0, t'_0]}(t) = \begin{cases} 1 & t_0 \leq t \leq t'_0, \\ 0 & \text{otherwise,} \end{cases}$$

Since $\mathbf{I}_{[t_j, t_{j+1}]} \subseteq [t_j, t_{j+1}]$, \mathcal{S} is feasible if $\mathcal{S}(t) \cdot u_{[t_j, t_{j+1}]}(t)$ is a feasible schedule of $\mathcal{J}_{[t_j, t_{j+1}]}$ for all $1 \leq j < h$. By definition, $\mathcal{S}(t) \cdot u_{[t_j, t_{j+1}]}(t) = \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}}$ is a feasible schedule of $\mathcal{J}_{[t_j, t_{j+1}]}$ for all $1 \leq j < h$. $E(\mathcal{S}) = \sum_{j=1}^{h-1} E(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}})$ holds trivially from $\mathbf{I}_{[t_j, t_{j+1}]} \subseteq [t_j, t_{j+1}]$. Finally, since \mathcal{S} is feasible, $E(\mathcal{S}) \geq E(\mathcal{S}_{\text{opt}}^{\mathcal{J}})$. \square

The following lemma implies how an energy-optimal voltage scheduling problem can be partitioned into subproblems.

LEMMA A.3. *Let*

$$E_1 = \min \left\{ \sum_{j=1}^{k-1} E(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[b_j, b_{j+1}]}}) \mid (b_1, b_2, \dots, b_k) \text{ is a strongly-blocking tuple.} \right\},$$

$$E_2 = \min \left\{ \sum_{j=1}^{h-1} E(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}}) \mid \min R_{\mathcal{J}} = t_1 < t_2 < \dots < t_h = \max D_{\mathcal{J}}, t_j \in T_{\mathcal{J}} \right\} \text{ and}$$

$$E_3 = \min \left\{ \sum_{j=1}^{h-1} E(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_j, t_{j+1}]}}) \mid [t_j, t_{j+1}] \text{ is a sub-interval}$$

of a strongly-atomic interval for all $1 \leq j < h$.

Then, $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}}) = E_1 = E_2 = E_3$.

PROOF. Let

$$\mathbf{S}_1 = \left\{ \bigoplus_{j=1}^{k-1} \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[b_j, b_{j+1}]}} \mid (b_1, b_2, \dots, b_k) \text{ is a strongly-blocking tuple.} \right\}$$

and define \mathbf{S}_2 and \mathbf{S}_3 similarly. Then, from Lemma A.2, $E_i = \min\{E(\mathcal{S}) \mid \mathcal{S} \in \mathbf{S}_i\}$ for $i = 1, 2, 3$. By definition, $\mathbf{S}_1 \subseteq \mathbf{S}_3 \subseteq \mathbf{S}_2$ and consequently $E_2 \leq E_3 \leq E_1$. Furthermore, $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}}) \leq E_2$ from Lemma A.2. From Theorem 3.3 and Lemma A.1, $\mathcal{S}_{\text{opt}}^{\mathcal{J}} \in \mathbf{S}_1$. Thus, we have $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}}) \geq E_1$, which implies $E(\mathcal{S}_{\text{opt}}^{\mathcal{J}}) = E_1 = E_2 = E_3$. \square

From Lemma A.3, it is obvious that the algorithm in Figure 10 always computes an optimal voltage schedule.

A.2 Proof of the Correctness of the Algorithm in Figure 12

We start with some lemmas to prove the correctness of the algorithm. First, note that for an interval $[t, t']$, $\mathbf{I}_{\mathcal{J}_{[t, t']^w}} \subseteq \mathbf{I}_{\mathcal{J}_{[t, t']}} \subseteq [t, t']$ since $\mathcal{J}_{[t, t']^w} \subseteq \mathcal{J}_{[t, t']}$. Therefore, for a weakly-blocking tuple $\mathbf{b}^w = (b_1, b_2, \dots, b_k)$, $\mathbf{I}_{\mathcal{J}_{[b_1, b_2]^w}}, \mathbf{I}_{\mathcal{J}_{[b_2, b_3]^w}}, \dots, \mathbf{I}_{\mathcal{J}_{[b_{k-1}, b_k]^w}}$ are disjoint.

LEMMA A.4. *Given a weakly-blocking tuple \mathbf{b}^w , let $\mathcal{J}_{\mathbf{b}^w}^{\mathcal{B}}$ represent the set of background jobs with respect to \mathbf{b}^w . Then, $\mathcal{J}_{\mathbf{b}_1^w}^{\mathcal{B}} \equiv \mathcal{J}_{\mathbf{b}_2^w}^{\mathcal{B}}$ for any weakly-blocking tuples \mathbf{b}_1^w and \mathbf{b}_2^w .*

PROOF. Let $\mathbf{b}_1^w = (b_1, b_2, \dots, b_k)$ and $\mathbf{b}_2^w = (b'_1, b'_2, \dots, b'_k)$. Assume that $J \in \mathcal{J}_{\mathbf{b}_1^w}^{\mathcal{B}}$ and $r_J \in [b_j, b_{j+1}]$. From the definition of a background job, we have

$$\exists k > j + 1, d_J \geq b_{j+1} \text{ and } p_J > \max\{p_{J'} \mid J' \in \bigcup_{l=j+1}^{k-1} \mathcal{J}_{[b_l, b_{l+1}]^w}\}. \quad (15)$$

Suppose that $J \notin \mathcal{J}_{\mathbf{b}_2^w}^{\mathcal{B}}$. From (15), we have

$$[b_{j+1}, b_{j+2}] \subseteq (b'_{j'}, b'_{j'+1}] \text{ for } r_J \in [b'_{j'}, b'_{j'+1}] ,$$

a contradiction. So, $\mathcal{J}_{\mathbf{b}_1^w}^B \subseteq \mathcal{J}_{\mathbf{b}_2^w}^B$. Similarly, we have $\mathcal{J}_{\mathbf{b}_2^w}^B \subseteq \mathcal{J}_{\mathbf{b}_1^w}^B$. \square

Lemma A.4 states that we can specify background jobs irrespective of weakly-blocking tuples. For the rest of this paper, we use \mathcal{J}^B to represent the set of background jobs.

LEMMA A.5. *Given a job set \mathcal{J} and an essential $|\mathcal{J}|$ -tuple \mathbf{f} , let $\mathbf{b}_{\sigma_f}^w = (b_1, b_2, \dots, b_k)$. Then, for any weakly-atomic interval $[b_j, b_{j+1}]$ ($1 \leq j < k$) and a background job J , we have the following assuming jobs are executed under $\mathcal{S}_{\text{opt}}^J$.*

- (a) $d_J \in [b_j, b_{j+1}]$: J completes its execution by b_j .
- (b) $r_J \in [b_j, b_{j+1}]$: J completes its execution by b_{j+1} .
- (c) $[b_j, b_{j+1}] \subseteq [r_J, d_J]$ executes its partial workload at constant speed.

Furthermore, for any interval $[t, t'] \subseteq [b_j, b_{j+1}]$, $\mathcal{J}_{[t, t']^w}$ is an EDF job set.

PROOF. Case (a) and Case (b) are obvious from the construction of the weakly-blocking tuple $\mathbf{b}_{\sigma_f}^w$. Case (c) follows from Lemma 3.7. Finally, suppose that $\mathcal{J}_{[t, t']^w}$ is not an EDF job set. Then, we have

$$\exists J, J' \in \mathcal{J}_{[t, t']^w} \text{ s.t. } p_J > p_{J'}, \quad d_J \in (r_{J'}, d_{J'}),$$

and the algorithm in Figure 11 selects $r_{J'}$ ($\in (b_j, b_{j+1})$) as an element of $\mathbf{b}_{\sigma_f}^w$, a contradiction. \square

From Lemma A.5, we characterize the optimal schedule in terms of weakly-atomic intervals, weakly-blocking tuples and background workload.

LEMMA A.6. *Given a job set \mathcal{J} and an essential $|\mathcal{J}|$ -tuple \mathbf{f} ,*

$$\mathcal{S}_{\text{opt}}^J \equiv \bigoplus_{j=1}^{k-1} \mathcal{S}_{\text{opt}}^{J_j} \quad (16)$$

where $\mathbf{b}_{\sigma_f}^w = (b_1, b_2, \dots, b_k)$ and $\mathcal{J}_j = \mathcal{J}_{[b_j, b_{j+1}]} \cup \{J_j^b\}$ such that

$r_{J_j^b} = b_j$, $d_{J_j^b} = b_{j+1}$, $p_{J_j^b} = \max\{p_J | J \in \mathcal{J}_{[b_j, b_{j+1}]}\} + 1$ and $c_{J_j^b} = c_j^b$ for some $c_j^b \geq 0$.

PROOF. From Lemma A.5, we have

$$\{\text{job}(\mathcal{J}, \mathcal{S}_{\text{opt}}^J(t), t) \mid t \in [b_j, b_{j+1}]\} \equiv \mathcal{J}_{[b_j, b_{j+1}]^w} \cup \mathcal{J}' \cup \mathcal{J}'' \quad \text{where}$$

$$\mathcal{J}' = \{J' \in \mathcal{J}^B \mid r_{J'} \in [b_j, b_{j+1}]\} \quad \text{and}$$

$$\mathcal{J}'' = \{J' \in \mathcal{J}^B \mid [b_j, b_{j+1}] \subseteq [r_{J'}, d_{J'}]\}.$$

From Case (b) of Lemma A.5, $\mathcal{J}_{[b_j, b_{j+1}]^w} \cup \mathcal{J}' \equiv \mathcal{J}_{[b_j, b_{j+1}]}$, and from Case (c), $\mathcal{J}'' = \{J_b\}$. So, we have

$$\mathcal{S}_{\text{opt}}^J(t) \cdot u_{[b_j, b_{j+1}]}(t) \equiv \mathcal{S}_{\text{opt}}^{J_j} \quad \text{for all } 1 \leq j < k,$$

which is equivalent to (16). \square

From Lemma A.6, the voltage scheduling problem is reduced to the problem of finding a weakly-blocking tuple $\mathbf{b}^w = (b_1, b_2, \dots, b_k)$ and the amount of background workload $c_{[b_j, b_{j+1}]}^B$ for each weakly-atomic interval $[b_j, b_{j+1}]$. To find the *background speed* $s_{[b_j, b_{j+1}]}^B$ instead of the amount of background workload makes it possible to exploit Lemma 3.7.

LEMMA A.7. Given a weakly-atomic interval $[t_1, t_2]$, let $\mathcal{J}' = \mathcal{J}_{[t_1, t_2]} \cup \{\mathcal{J}^b\}$ where

$$r_{\mathcal{J}^b} = t_1, \quad d_{\mathcal{J}^b} = t_2, \quad p_{\mathcal{J}^b} = \max\{p_J | J \in \mathcal{J}_{[t_1, t_2]}\} + 1 \text{ and } c_{\mathcal{J}^b} = c_{[t_1, t_2]}^B (> 0),$$

and let $s_{[t_1, t_2]}^B$ be the constant speed of \mathcal{J}^b under $\mathcal{S}_{\text{opt}}^{\mathcal{J}'}$. Then,

$$\mathcal{S}_{\text{opt}}^{\mathcal{J}'}(t) = \begin{cases} \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_1, t_2]}}(t) & t \text{ s.t. } \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_1, t_2]}}(t) > s_{[t_1, t_2]}^B, \\ s_{[t_1, t_2]}^B & t \text{ s.t. } \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_1, t_2]}}(t) \leq s_{[t_1, t_2]}^B. \end{cases}$$

Furthermore, $s_{[t_1, t_2]}^B$ strictly increases as $c_{[t_1, t_2]}^B$ increases, and vice versa.

PROOF. From Lemmas A.5 and A.6, both $\mathcal{J}_{[t_1, t_2]}$ and \mathcal{J}' follow the EDF priority and their optimal voltage schedules are obtained by Yao's algorithm [Yao et al. 1995]. For an interval $[t'_1, t'_2] \subset [t_1, t_2]$ such that $\mathcal{S}_{\text{opt}}^{\mathcal{J}'_{[t'_1, t'_2]}}(t) > s_{[t'_1, t'_2]}^B$, Yao's algorithm selects the same speed for $\mathcal{S}_{\text{opt}}^{\mathcal{J}'}(t)$. For the other intervals, $\mathcal{S}_{\text{opt}}^{\mathcal{J}'}(t) = s_{\mathcal{J}^b}$ since $[t_1, t_2] \subseteq [r_{\mathcal{J}^b}, d_{\mathcal{J}^b}]$.

Because $W(\mathcal{S}_{\text{opt}}^{\mathcal{J}'}, [t_1, t_2])$ strictly increases as $s_{[t_1, t_2]}^B$ increases and $c_{[t_1, t_2]}^B = W(\mathcal{S}_{\text{opt}}^{\mathcal{J}'}, [t_1, t_2]) - W(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_1, t_2]}}, [t_1, t_2])$, $c_{[t_1, t_2]}^B$ increases as $s_{[t_1, t_2]}^B$ increases. Hence, it follows that $s_{[t_1, t_2]}^B$ increases as $c_{[t_1, t_2]}^B$ increases (and vice versa). \square

Definition A.8. Given a job set \mathcal{J} and background workload c , the job set \mathcal{J} with background workload c is defined as

$$\mathcal{J}[c] \stackrel{\text{def}}{=} \mathcal{J} \cup \{\mathcal{J}^b\} \text{ where } r_{\mathcal{J}^b} = R_{\mathcal{J}}, \quad d_{\mathcal{J}^b} = D_{\mathcal{J}}, \quad p_{\mathcal{J}^b} = \max\{p_J | J \in \mathcal{J}\} + 1 \text{ and } c_{\mathcal{J}^b} = c.$$

Furthermore, given a job set $\mathcal{J}[c]$, the constant speed of background workload under $\mathcal{S}_{\text{opt}}^{\mathcal{J}[c]}$ is called a *background speed* of $\mathcal{J}[c]$ and is denoted by $BS(\mathcal{J}, c)$.

The following lemma is an extension of Lemma A.7 for arbitrary intervals.

LEMMA A.9. Given a job set $\mathcal{J}[c]$

$$\mathcal{S}_{\text{opt}}^{\mathcal{J}[c]} \equiv \bigoplus_{j=1}^{k-1} \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[b_j, b_{j+1}]}}^{[c_j]} \text{ for } b_1, \dots, b_k \in \mathcal{T}_{\mathcal{J}}, \quad b_1 < \dots < b_k \text{ such that}$$

$$c = \sum_{j=1}^{k-1} c_j \text{ and } BS(\mathcal{J}_{[b_j, b_{j+1}]}, c_j) = BS(\mathcal{J}_{[b_{j'}, b_{j'+1}]}, c_{j'}) \text{ for all } 1 \leq j \neq j' < k.$$

PROOF. Directly from Lemmas A.6 and 3.7. \square

Along with Lemma A.9, the following lemma implies how the problem can be reduced to a dynamic programming formulation.

LEMMA A.10. Given $t_i, t_j, t_m \in \mathcal{T}_{\mathcal{J}}$ where $t_i < t_m \leq t_j$, let

$$\mathcal{J}_{[t_i, t_j]}^B = \mathcal{J}_{[t_i, t_j]}^w \cup \{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_i, t_j]\} \text{ and}$$

$$c_{[t_i, t_m]}^B = C(\{J \in \mathcal{J}^B \mid r_J \in [t_i, t_m) \wedge d_J \in [t_m, t_j]\}),$$

and let $\mathcal{S}_{\text{opt}}^{[t_i, t_j]}$ represent $\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}^B}$. Then,

$$\mathcal{S}_{\text{opt}}^{[t_i, t_j]} \in \{\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w} \oplus \mathcal{S}_{\text{opt}}^{[t_m, t_j]} \mid \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w} \text{ is feasible for } \mathcal{J}_{[t_i, t_m]}^B\}.$$

PROOF. If all the jobs in $\{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_i, t_j]\}$ runs at the same speed under $\mathcal{S}_{\text{opt}}^{[t_i, t_j]}$, $\mathcal{S}_{\text{opt}}^{[t_i, t_j]} \equiv \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_j]}^w [c_{[t_i, t_j]}^B]}$. Otherwise, there must exist $t_m \in \mathcal{T}_{\{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_i, t_j]\}} (\subseteq \mathcal{T}_J)$ such that all the jobs in $\{J \in \mathcal{J}^B \mid r_J \in [t_i, t_m) \wedge d_J \in [t_m, t_j]\}$ finish their executions by t_m with the same constant speed and all the jobs in $\{J \in \mathcal{J}^B \mid [r_J, d_J] \subseteq [t_m, t_i]\}$ are not executed before t_m under $\mathcal{S}_{\text{opt}}^{[t_i, t_j]}$. Therefore, we have $\mathcal{S}_{\text{opt}}^{[t_i, t_j]} \equiv \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w [c_{[t_i, t_m]}^B]} \oplus \mathcal{S}_{\text{opt}}^{[t_m, t_j]}$ where $\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w [c_{[t_i, t_m]}^B]}$ is feasible for $\mathcal{J}_{[t_i, t_m]}^B$. \square

COROLLARY A.11. Let $E_{\text{opt}}^{[t_i, t_j]}$ denote $E(\mathcal{S}_{\text{opt}}^{[t_i, t_j]})$ where $\mathcal{S}_{\text{opt}}^{[t_i, t_j]}$ is defined as in Lemma A.10. Then,

$$E_{\text{opt}}^{[t_i, t_j]} = \min (\{ E(\mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w [c_{[t_i, t_m]}^B]}) + E_{\text{opt}}^{[t_m, t_j]} \mid t_m \in \mathcal{T}_J, t_i < t_m < t_k, \mathcal{S}_{\text{opt}}^{\mathcal{J}_{[t_i, t_m]}^w [c_{[t_i, t_m]}^B]} \text{ is feasible for } \mathcal{J}_{[t_i, t_m]}^B \}) .$$

The correctness of the algorithm in Figure 12 directly follows from Lemma A.10 and Corollary A.11.

ACKNOWLEDGMENTS

We would like to thank Prof. Gang Quan and anonymous referees for their helpful comments and suggestions. This work was supported by grant No. R01-2001-00360 from the Korea Science and Engineering Foundation. The RIACT at Seoul National University provides research facilities for the study.

REFERENCES

- AYDIN, H., MELHEM, R., MOSSE, D., AND ALVAREZ, P. M. 2001. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. of Real-Time Systems Symposium*.
- GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability*. W.H. Freeman and Company.
- GRUIAN, F. 2001. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *Proc. of International Symposium on Low Power Electronics and Design*. 46–51.
- HONG, I., QU, G., POTKONJAK, M., AND SRIVASTAVA, M. B. 1998. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proc. of Real-Time Systems Symposium*. 178–187.
- KIM, N., RYU, M., HONG, S., SAKSENA, M., CHOI, C., AND SHIN, H. 1996. Visual Assessment of a Real-Time System Design: A Case Study on a CNC Controller. In *Proc. of Real-Time Systems Symposium*. 300–310.
- KIM, W., KIM, J., AND MIN, S. L. 2002. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proc. of Design, Automation and Test in Europe*.
- LIU, W.-S. 2000. *Real-Time Systems*. Prentice Hall.
- LOCKE, C., VOGEL, D., AND MESLER, T. 1991. Building a Predictable Avionics Platform in Ada: A Case Study. In *Proc. of Real-Time Systems Symposium*.
- PILLAI, P. AND SHIN, K. G. 2001. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of ACM Symposium on Operating Systems Principles*.
- QUAN, G. AND HU, X. 2001. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proc. of Design Automation Conference*. 828–833.
- QUAN, G. AND HU, X. 2002. An Optimal Voltage Schedule for Real-Time Systems on a Variable Voltage Processor. In *Proc. of Design, Automation and Test in Europe*.
- SAHNI, S. 1976. Algorithms for Scheduling Independent Tasks. *Journal of the ACM* 23, 116–127.
- SAKURAI, T. AND NEWTON, A. 1990. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas. *IEEE Journal of Solid State Circuits* 25, 2, 584–594.

- SHIN, D., KIM, J., AND LEE, S. 2001. Low-Energy Intra-Task Voltage Scheduling Using Static Timing Analysis. In *Proc. of the 38th Design Automation Conference*.
- SHIN, Y. AND CHOI, K. 1999. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of Design Automatioin Conference*. 134–139.
- SHIN, Y., CHOI, K., AND SAKURAI, T. 2000. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proc. of International Conference on Computer-Aided Design*. 365–368.
- WÖEGINGER, G. J. 1999. When Does a Dynamic Programming Formulation Guarantee the Existence of an FPTAS? In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*. 820–829.
- YAO, F., DEMERS, A., AND SHENKER, S. 1995. A Scheduling Model for Reduced CPU Energy. In *Proc. of IEEE Annual Foundations of Computer Science*. 374–382.
- YUN, H.-S. AND KIM, J. 2002. On Energy-Optimal Off-Line Scheduling for Fixed-Priority Real-Time Systems on a Variable Voltage Processor. Tech. rep., School of CSE, Seoul National Univ.

Received February 2002; revised August 2002; accepted September 2002

Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems

Woonseok Kim* Dongkun Shin† Han-Saem Yun† Jihong Kim† Sang Lyul Min*

School of Computer Science and Engineering
Seoul National University ENG4190, Seoul, Korea, 151-742
wskim@archi.snu.ac.kr, {sdk, hsyun, jihong}@davinci.snu.ac.kr, symin@dandelion.snu.ac.kr

Abstract

Dynamic voltage scaling (DVS) is an effective low-power design technique for embedded real-time systems. In recent years, many DVS algorithms have been proposed for reducing the energy consumption of embedded hard real-time systems. However, the proposed DVS algorithms were not quantitatively evaluated under a unified framework, making it a difficult task to select an appropriate DVS algorithm for a given application/system. In this paper, we compare several key DVS algorithms recently proposed for hard real-time periodic task sets, analyze their energy efficiency, and discuss the performance differences quantitatively. Our evaluation results give quantitative answers to several important DVS questions.

1 Introduction

Dynamic voltage scaling (DVS), which adjusts the supply voltage and correspondingly the clock frequency dynamically, is an effective low-power design technique for embedded real-time systems. Since the energy consumption E of CMOS circuits has a quadratic dependency on the supply voltage V_{dd} , lowering the supply voltage V_{dd} is one of the most effective ways of reducing the energy consumption.

With a recent explosive growth in the portable and mobile embedded device market, where a low-power consumption is an important design requirement, several commercial variable-voltage microprocessors [19, 1, 8] were developed. Targeting these microprocessors, many DVS algorithms have been proposed or developed, especially for hard

real-time systems [7, 9, 18, 2, 14, 16, 5, 10]. Since lowering the supply voltage also decreases the maximum achievable clock speed [15], various DVS algorithms for hard real-time systems have the goal of reducing supply voltage dynamically to the lowest possible level while satisfying the tasks' timing constraints.

Although each DVS algorithm is shown to be quite effective in reducing the energy/power consumption of a target system under its own experimental scenarios, these recent DVS algorithms have not been quantitatively evaluated under a unified framework, making it a difficult task for low-power embedded system developers to select an appropriate DVS algorithm for a given application/system. A quantitative analysis of the energy-efficiency is particularly important because most of these DVS algorithms are based on both static and dynamic slack analysis techniques whose performance is difficult to predict analytically. In addition, their energy efficiency fluctuate significantly depending on the workload variations, task set characterizations, and execution paths taken, further requiring a quantitative comparison study.

In this paper, we quantitatively evaluate the energy efficiency of several recent DVS algorithms proposed for hard real-time systems using a unified DVS simulation environment called *SimDVS* [17]. We focus on a preemptive hard real-time systems in which periodic real-time tasks are scheduled with the Earliest-Deadline-First (EDF) algorithm or the Rate-Monotonic (RM) algorithm, the two most widely used real-time system models [12]. Our study is different from the previous performance comparisons such as [13, 6]. [13] and [6] focus on *aperiodic* tasks in hard real-time systems and non real-time systems, respectively, while our study focuses on periodic tasks in hard real-time systems.

For the target hard real-time systems, two categories of algorithms are used: inter-task DVS (InterDVS) and intra-task DVS (IntraDVS). InterDVS algorithms determine the supply voltage on task-by-task basis, while IntraDVS algorithms

*This work was supported in part by the Ministry of Education under the BK21 program, and by the Ministry of Science and Technology under the National Research Laboratory program.

†This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation.

adjust the supply voltage within an individual task boundary. For a comparative study, we use eight InterDVS algorithms [18, 2, 14, 10] and two IntraDVS algorithms [16, 5] that were recently proposed.

We also evaluate the energy efficiency of HybridDVS algorithms. (If a DVS algorithm uses both the IntraDVS and InterDVS approaches, we call the algorithm a hybrid DVS algorithm (HybridDVS).)

Since many factors affect the energy efficiency of DVS algorithms, our comparative study cannot answer all the DVS performance questions. In this paper, we limit our evaluation goals to the following questions which represent some of the most important unanswered questions:

- **InterDVS:** What is the best InterDVS algorithm under given conditions? How close is the algorithm’s energy efficiency to the theoretical lower bound? What restrictions of variable-voltage processors, if any, limit the achievable energy efficiency of InterDVS algorithms?
- **IntraDVS:** Which IntraDVS algorithm performs better under what condition?
- **HybridDVS:** Can we achieve better energy efficiency if we combine an InterDVS algorithm and an IntraDVS algorithm?

Our comparative study shows that the existing EDF InterDVS algorithms such as [2, 14, 10], are very effective; their energy consumption is only 9~12% worse than the theoretical lower bound. Moreover, this gap can be further reduced by using a more intelligent slack distribution method. With a better slack distribution heuristic, we strongly believe that the energy efficiency of the current state-of-art EDF InterDVS algorithms is very close to that of the theoretical optimal algorithm. However, in the RM InterDVS algorithms, there still remains a room for improvement. Also, the energy efficiency of each algorithm can vary from 10% to 32% according to the number of voltage levels supported by the target variable-voltage processor.

For the IntraDVS algorithms, our results indicate that the path-based IntraDVS [16] achieves better performance than the stochastic IntraDVS [5] when the slack time is limited. On the other hand, when there is a large amount of slack time, the stochastic IntraDVS algorithm works better.

For the HybridDVS algorithms, our experiments show that the energy efficiency of a HybridDVS is better than the one that can be achieved by using an IntraDVS algorithm or an InterDVS algorithm alone.

The rest of the paper is organized as follows; before the selected DVS algorithms are evaluated, we first classify existing DVS techniques in Section 2. In Section 3, we summarize the selected DVS algorithms using the classification framework of Section 2. Simulation environments are described in Section 4. We present the performance evaluation

Table 1. Classification of DVS techniques.

	Voltage Scaling Methods	Scaling Decision
IntraDVS	(1) Path-based method	Off-Line
	(2) Stochastic method	
	(3) Maximum constant speed	
InterDVS	(4) Stretching to NTA	On-Line
	(5) Priority-based slack-stealing	
	(6) Utilization updating	

results in Section 5, and Section 6 concludes with a summary.

2 Classification of DVS algorithms

In this section, we classify the existing DVS techniques and briefly describe the key characteristics of each technique. (See Table 1 for summary.)

For hard real-time systems, there are two kinds of voltage scheduling approaches depending on the voltage scaling granularity: intra-task DVS (IntraDVS) and inter-task DVS (InterDVS). The intra-task DVS algorithms [16, 5] adjust the voltage within an individual task boundary, while the inter-task DVS algorithms determine the voltage on a task-by-task basis at each scheduling point. The main difference between them is whether the slack times are used for the current task or for the tasks that follow. InterDVS algorithms distribute the slack times from the current task for the following tasks, while IntraDVS algorithms use the slack times from the current task for the current task itself.

2.1 Intra-task DVS algorithm design factors

In scheduling hard real-time tasks, in order to guarantee the timing constraint of each task, the execution times of tasks are usually assumed to be the worst case execution times (WCETs). However, since a task has many possible execution paths, there are large execution time variations among them. So, when the execution path taken at run time is not the worst case execution path (WCEP), the task may complete its execution before its WCET, resulting in a slack time. In that case, IntraDVS exploits such slack times and adjusts the processor speed. IntraDVS algorithms can be classified into two types depending on how to estimate slack times and how to adjust speeds.

2.1.1 Path-based method

In the path-based IntraDVS, the voltage and clock speed are determined based on a predicted reference execution path, such as WCEP. For example, when the actual execution deviates from the predicted reference execution path (say, by a branch instruction), the clock speed is adjusted. If the new path takes significantly longer to complete its execution than

the reference path, the clock speed is *raised* to meet the deadline constraint. On the other hand, if the new path can finish its execution earlier than the reference path, the clock speed is *lowered* to reduce the energy consumption.

In the path-based IntraDVS, program locations for possible speed scaling are identified using static program analysis [16] or execution time profiling [11].

2.1.2 Stochastic method

The stochastic method is based on the idea that it is better to start the execution at a low speed and accelerate the execution later when needed than to start with a high speed and reduce the speed later when slack time is found. By starting at a low speed, if the task finishes earlier than its WCET, it does not need to execute at a high speed. Theoretically, if the probability density function of execution times of a task is known *a priori*, the optimal speed schedule can be computed [5]. Under the stochastic method, the clock speed is *raised* at specific time instances, regardless of the execution paths taken. Unlike the path-based IntraDVS that can utilize all the slack times of the task in scaling speed, the stochastic IntraDVS may not utilize all the potential slack times.

2.2 Inter-task DVS algorithm design factors

InterDVS algorithms exploit the “run-calculate-assign-run” strategy to determine the supply voltage, which can be summarized as follows: (1) run a current task, (2) when the task is completed, calculate the maximum allowable execution time for the next task, (3) assign the supply voltage for the next task, and (4) run the next task. Most InterDVS algorithms differ during step (2) in computing the maximum allowed time for the next task τ which is the sum of WCET of τ and the slack time available for τ .

A generic InterDVS algorithm consists of two parts: slack estimation and slack distribution. The goal of the slack estimation part is to identify as much slack times as possible while the goal of the slack distribution part is to distribute the resulting slack times so that the resulting speed schedule is as uniform as possible. Slack times generally come from two sources; *static slack times* are the extra times available for the next task that can be identified statically, while *dynamic slack times* are caused from run-time variations of the task executions.

2.2.1 Slack estimation methods

(1) Static slack estimation

Maximum constant speed One of the most commonly used static slack estimation methods is to compute the *maximum constant speed*, which is defined as the lowest possible clock speed that guarantees the feasible schedule of a task set [18]. For example, in EDF scheduling, if the worst case

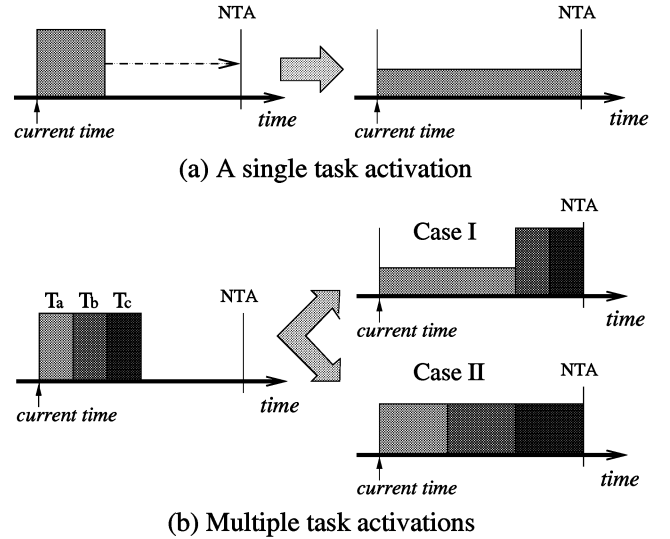


Figure 1. Examples of Stretching-to-NTA.

processor utilization (WCPU) U of a given task set is lower than 1.0 under the maximum speed f_{max} , the task set can be scheduled with a new maximum speed $f'_{max} = U \cdot f_{max}$. Although more complicated, the maximum constant speed can be statically calculated as well for RM scheduling [18, 5].

(2) Dynamic slack estimation

Three widely-used techniques of estimating dynamic slack times are briefly described below.

Stretching to NTA Even though a given task set is scheduled with the maximum constant speed, since the actual execution times of tasks are usually much less than their WCETs, the tasks usually have dynamic slack times. One simple method to estimate the *dynamic slack time* is to use the arrival time of the next task [18]. (The arrival time of the next task is denoted by NTA.) Assume that the current task τ is scheduled at time t . If NTA of τ is later than $(t + WCET(\tau))$, task τ can be executed at a lower speed so that its execution completes exactly at the NTA.

Figure 1 shows examples of the *Stretching-to-NTA* method. When a single task τ is activated as shown in Figure 1(a), the execution of τ can be stretched to NTA. When multiple tasks are activated, there can be several alternatives in stretching options. For example, the dynamic slack time may be given to a single task or distributed equally to all activated tasks. Cases I and II of Figure 1(b) illustrate these two options, respectively.

Priority-based slack stealing This method exploits the basic properties of priority-driven scheduling such as RM and EDF. The basic idea is that when a higher-priority task completes its execution earlier than its WCET, the following

Table 2. Target DVS algorithms.

Category	Scheduling Policy	DVS Policy	Used Methods [†]
InterDVS	EDF	lppsEDF [18]	(3)+(4)
		ccEDF [14]	(6)
		laEDF [14]	(6)*
		DRA [2]	(3)+(4)+(5)
		AGR [2]	(4)*+(5)
	lpSHE [10]	(3)+(4)+(5)*	
	RM	lppsRM [18]	(3)+(4)
ccRM [14]		(3)+(4)*	
IntraDVS	Path-based Method	intraShin [16]	(1)
	Stochastic Method	intraGruian [5]	(2)

[†] Numbers indicate corresponding techniques in Table 1.

(n)* indicates an improved version of n.

lower-priority tasks can use the slack time from the completed higher-priority task. It is also possible for a higher-priority task to utilize the slack times from completed lower-priority tasks. However, the latter type of slack stealing is computationally expensive to implement precisely. Therefore, the existing algorithms are based on heuristics [2, 10].

Utilization updating The actual processor utilization during run time is usually lower than the worst case processor utilization. The utilization updating technique estimates the required processor performance at the current scheduling point by recalculating the expected worst case processor utilization using the actual execution times of completed task instances [14]. When the processor utilization is updated, the clock speed can be adjusted accordingly. The main merit of this method is its simple implementation, since only the processor utilization of completed task instances have to be updated at each scheduling point.

2.2.2 Slack distribution methods

In distributing slack times, most InterDVS algorithms have adopted a greedy approach, where all the slack times are given to the next activated task. This approach is not an optimal solution, but the greedy approach is widely used because of its simplicity.

3 Target DVS algorithms

Table 2 summarizes the DVS algorithms selected for the comparative study. Here, eight InterDVS algorithms are chosen, two [18, 14] of which are based on the RM scheduling policy, while the other six algorithms [18, 14, 2, 10] are based on the EDF scheduling policy. For IntraDVS algorithms, two algorithms are selected, one from path-based IntraDVS algorithms [16], and the other from stochastic methods [5].

In these selected DVS algorithms, one or sometimes more than one slack estimation methods explained in the previous section were used. In lppsEDF and lppsRM which were proposed by Shin *et. al.* in [18], slack time of a task is es-

timated using the maximum constant speed and Stretching-to-NTA methods.

The cCRM algorithm proposed by Pillai *et. al.* [14] is similar to lppsRM in the sense that it uses both the maximum constant speed and the Stretching-to-NTA methods. However, while lppsRM can adjust the voltage and clock speed only when a single task is active (Figure 1(a)), cCRM extends the stretching to NTA method to the case where multiple tasks are active (Case-II in Figure 1(b)).

Pillai *et. al.* also proposed two other DVS algorithms [14], ccEDF and laEDF, for EDF scheduling policy. These algorithms estimate slack time of a task using the utilization updating method. While ccEDF adjusts the voltage and clock speed based on run-time variation in processor utilization alone, laEDF takes a more aggressive approach by estimating the amount of work required to be completed before NTA.

DRA and AGR, which were proposed by Aydin *et. al.* in [2], are two representative DVS algorithms that are based on the priority-based slack stealing method. The DRA algorithm estimates the slack time of a task using the priority-based slack stealing method along with the maximum constant speed and the Stretching-to-NTA methods. Aydin *et. al.* also extended the DRA algorithm and proposed another DVS algorithm called AGR for more aggressive slack estimation and voltage/clock scaling. In AGR, in addition to the priority-based slack stealing, more slack times are identified by computing the amount of work required to be completed before NTA (Case-I in Figure 1(b)).

lpSHE is another DVS algorithm which is based on the priority-based slack stealing method [10]. Unlike DRA and AGR, lpSHE extends the priority-based slack stealing method by adding a procedure that estimates the slack time from lower-priority tasks that were completed earlier than expected. DRA, AGR, and lpSHE algorithms are somewhat similar to one another in the sense that all of them use the maximum constant speed in the off-line phase and the Stretching-to-NTA method in the on-line phase in addition to the priority-based slack stealing method.

For IntraDVS algorithms, Shin's intra-task DVS algorithm [16] (intraShin) and Gruian's algorithm [5] (intraGruian) are used as representative algorithms of the path-based method and the stochastic method, respectively. (The details of these algorithms were described in Section 2.)

4 Simulation environment

In this section, we describe SimDVS [17], a unified DVS simulation environment, used for the quantitative analysis. In order to support a wide variety of DVS algorithms and simulation scenarios, SimDVS was designed to achieve the following goals: 1) support both IntraDVS and InterDVS

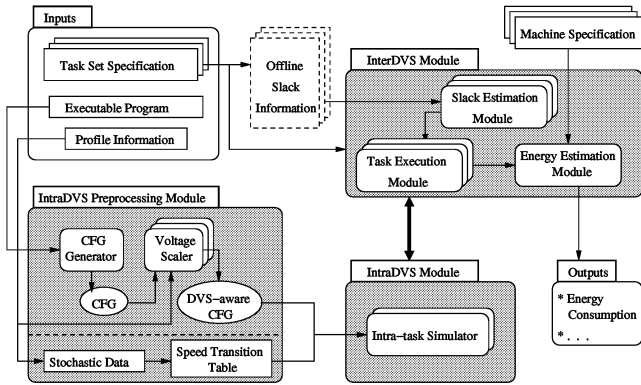


Figure 2. Overview of the SimDVS simulation environment.

algorithms, 2) integrate different DVS algorithms easily, 3) support different task workloads, variations in execution paths taken, and different task set configurations easily, and 4) support different variable-voltage processors easily.

Figure 2 shows an overview of SimDVS, which consists of three main modules: 1) the InterDVS module, 2) the IntraDVS module, and 3) the IntraDVS pre-processing module. SimDVS takes as an input a task set specification for an InterDVS algorithm and a DVS-aware control flow graph (CFG) for an IntraDVS algorithm. The DVS-aware CFG is built from the input binary program. As output, SimDVS reports the energy consumption of the input task set (or the input CFG).

The InterDVS module is responsible for the overall operation of SimDVS. It simulates a given task set under the selected scheduling policy using a given slack estimation heuristic. The IntraDVS module simulates IntraDVS algorithms using the Intra-task simulator. The input to the IntraDVS module is pre-processed by the tools available in the IntraDVS pre-processing module. For faster simulations of IntraDVS algorithms, the CFG of the input program is simulated rather than the instructions in the program. For a comparative study, SimDVS supports all DVS algorithms described in Section 3.

4.1 Submodules of InterDVS module

The InterDVS module, responsible for scheduling tasks, plays the role of a real-time scheduler in a hard real-time system. It takes as an input the specification of a periodic task set. The task set specification describes the properties of simulated periodic tasks, such as the period and WCET of each task and the workload variation factors (e.g., the worst case utilization and execution time distribution). To simulate

a given InterDVS scheduling algorithm, it has two modules, one for slack estimation and the other for slack distribution. Slack estimation is done by the *slack estimation module* that computes the total available time of the scheduled task, and the slack distribution is done by the *task execution module* that determines the operating speed of the scheduled task and simulates the execution of the task instance. To simulate a new InterDVS algorithm, these two modules for the new algorithm need to be added.

Slack estimation module This module is highly dependent on the simulated target InterDVS algorithm. Therefore, the exact implementation of this module depends on the DVS algorithm. Currently, all the InterDVS algorithms described in Table 2 are supported. In addition, an optimal slack stealing method under EDF scheduling is also supported to evaluate the effectiveness of the slack estimation parts of various InterDVS algorithms.

Some DVS algorithms (e.g., [5]) may require off-line pre-processing steps for a more efficient on-line slack estimation. In this case, the slack estimation module takes such an off-line information as an additional input.

Task execution module This module has two roles. First, it determines the voltage and clock speed based on the available execution time t_a for the current task. Using the supported voltage levels by the target machine (specified in the *machine specification* file), it sets the voltage and clock speed so that the activated task finishes its execution within t_a time units even in the case where its execution takes WCEP. Second, it simulates the execution of the task. It generates the effective workload of each task based on the input workload variation factor, calculates the elapsed time and the unused time from the assigned available time interval, and reports this timing/speed information to the *energy estimation module*. If an intra-task scheduling is used, this module calls the *Intra-task simulator* of the *IntraDVS module* to simulate intra-task voltage scaling.

Energy estimation module This module takes the timing and speed information from the task execution module, and computes the energy consumption of the current task execution using the current machine configuration. By default, the energy consumption is estimated based on the equations described in [3]. The current version of SimDVS supports the specifications of XScale [8], AMD's K6-2+ [1], and Crusoe [19] processors.

4.2 Submodules of IntraDVS & its pre-processing modules

The IntraDVS module that contains the intra-task simulator has two roles; it simulates the execution behavior of real applications, and performs intra-task DVS. To reflect the execution behavior of real applications, the CFG generator in the *IntraDVS pre-processing module* produces CFGs

from SimpleScalar 2.0 [4] binary program. Each node of a CFG is annotated with extra information (e.g., the number of instructions in a basic block) necessary for proper simulation runs. In order to support the simulation of path-based IntraDVS algorithms and stochastic IntraDVS algorithms, voltage scaling locations within a task should be determined during the off-line phase. The following two submodules in the IntraDVS pre-processing module are responsible for this.

Voltage scaler This module takes the CFG of the target application and extracts the timing information from the CFG. It analyzes the given CFG and computes the predicted remaining execution times from each basic block. Then, it inserts the voltage scaling information at selected scaling points. Finally, Voltage scaler generates the *DVS-aware CFG*, which includes voltage scaling information, and passes it to the Intra-task simulator for the path-based IntraDVS.

Speed transition table To simulate stochastic IntraDVS algorithms, the stochastic data (such as the cumulative distribution function of task execution times) should be collected from profiling. Based on the stochastic data, the speed transition table, which describes when the execution speed is changed to what level, is constructed. Then, the speed transition table is passed to the Intra-task simulator for the stochastic Intra-DVS.

5 Experimental results

The DVS algorithms described in Section 3 are evaluated by implementing them in SimDVS and performing experiments with various key parameters that may affect the energy efficiency of the DVS algorithms. Three classes of DVS algorithms were evaluated: InterDVS algorithms, IntraDVS algorithms, and HybridDVS algorithms.

For the experiments, the energy consumption model based on the ARM8 microprocessor core is used. The clock speed can be varied in the range of [8, 100] MHz with a step size of 1 MHz and the supply voltage can be varied in the range of [1.1, 3.3] V. We assume that the system enters a power-down mode whenever the system becomes idle and that no energy is consumed in the power-down mode. We also assume that the voltage scaling overhead is negligible both in the time and the energy consumed.

5.1 Performance evaluation of InterDVS algorithms

The energy efficiency of InterDVS algorithms depends significantly on the accuracy of slack estimation and the appropriateness of slack distribution. To evaluate the effectiveness of the slack estimation method used in each InterDVS algorithm, extensive experiments while varying the number of tasks and WCPUs of task sets are performed. Then,

the energy efficiency of the algorithms are measured while changing the number of available voltage levels, in order to evaluate their adaptability to different machine specifications. Finally, to evaluate the effect of slack distribution methods, experiments were performed while restricting the amount of slack time that a task can utilize.

5.1.1 Number of tasks

To evaluate the impact of the number of tasks on the energy efficiency of DVS algorithms, experiments with various numbers of tasks were performed. For each task set with n tasks (where $n = 2, 4, 6, \dots, 16$), 100 task sets were randomly generated. The period and the WCET of each task were randomly generated using uniform distribution with the ranges of [10, 100] ms and [1, *period*] ms, respectively. To eliminate the effect of static slack times, we chose the task sets which have high worst case processor utilization; WCPUs are equal to 1.0 for EDF InterDVS algorithms and 0.9 for RM InterDVS algorithms. The execution time of each task instance was randomly drawn from a Gaussian distribution¹ with the range of [$\frac{1}{10}$ WCET, WCET] of each task, and the resulting average case processor utilization (ACPU) was set to 0.55.

Figure 3 shows the impact of the number of tasks on the energy consumption. In the figure, the y axis indicates the normalized energy consumption value over the energy consumption of an application running on a DVS-unaware system with a power-down mode only. As the number of tasks increases, the energy efficiency of `lppsEDF`, `lppsRM`, and `cCRM` that only use the *Stretching-to-NTA* technique do not significantly improve, while that of the other more aggressive InterDVS algorithms improves significantly. This can be explained by the fact that, in the *Stretching-to-NTA* method, the slack time that can be exploited is limited to the time between the completion of a task instance and the arrival time of the next task instance, which is largely independent of the number of tasks in the system. On the other hand, for the other InterDVS algorithms, since the slack times can be taken from any completed task instance, as the number of task increases, each task has more slack sources and can be scheduled with a lowered clock speed.

Since the energy efficiency of each InterDVS algorithm is not affected by the number of tasks when there are more than eight tasks, the rest of experiments were performed using task sets with 8 tasks.

5.1.2 Worst case processor utilization of task set

When the WCPU of a given task set is less than 1.0, the tasks have inherent static slack times. Figure 4(a) shows the re-

¹With the mean $m = \frac{\text{WCET}/10 + \text{WCET}}{2}$ and the standard deviation $\sigma = \frac{0.9 \cdot \text{WCET}}{6}$.

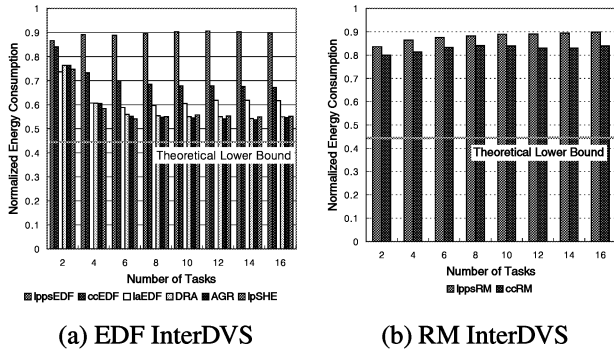


Figure 3. Impact of the number of tasks.

sults for varying WCPUs of 8-task task sets. The results indicate that, except for $lppsEDF$, the energy consumption of InterDVS algorithms increases as a linear function of WCPU of a task set. For $lppsEDF$, the energy consumption increases faster than a linear function of WCPU of a task set. This indirectly indicates that the dynamic slack estimation method of $lppsEDF$ is not very effective.

One interesting observation from Figure 4(a) is that $lppsEDF$ shows better energy efficiency than $ccEDF$ when WCPU is less than 0.7. This is because, in $ccEDF$, the clock speed is determined using the *actual* processor utilization² at the scheduling point. Since the actual processor utilization increases when a low-speed task instance completes its execution, the next task instance needs to be executed in a higher speed. Such voltage fluctuation occurs more often as the WCPU decreases. Thus, as the WCPU decreases, the energy efficiency of $ccEDF$ becomes worse than that of $lppsEDF$.

Because of the space limitation, the results for $lppsRM$ and $ccRM$ are not included but they are very similar to that of $lppsEDF$.

5.1.3 Machine specification

Variable-voltage processors provide a finite number of voltage levels, from two to as many as 100 levels. To evaluate the impact of the number of scaling levels on the energy efficiency of the InterDVS algorithms, several different machine specifications were tested. In the experiments, when there are k scaling levels, the voltage and the clock speed can be varied with a step size of $\frac{92}{k}$ MHz within the range of [8,100] MHz.

Figure 4(b) shows the effect of the number of scaling levels on the energy efficiency of the InterDVS algorithms.

²The actual processor utilization is computed by summing the individual task processor utilization, i.e., $U_a = \sum \frac{c_i}{p_i}$ where p_i is the period of task τ_i and c_i is assumed to be WCET if τ_i is not completed, otherwise the actual execution time of τ_i .

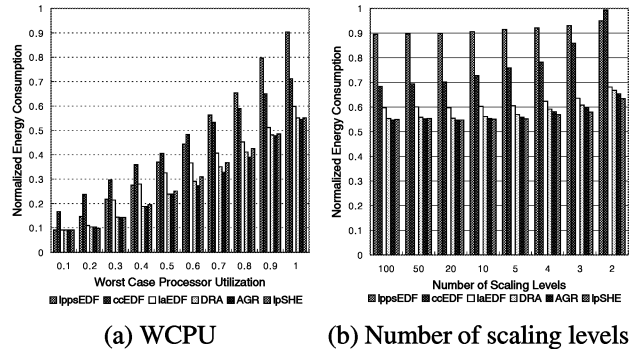


Figure 4. Impact of WCPU and the number of scaling levels.

As shown, the energy consumption increases as the number of scaling levels decreases. For more aggressive algorithms (e.g., DRA, AGR, $laEDF$, and $lpSHE$), the impact of the number of scaling levels is relatively marginal (roughly 8%) compared to that of less aggressive algorithms (e.g., $lppsEDF$ and $ccEDF$).

5.1.4 Speed bound

In the previous experiments, we assumed the greedy method in the slack distribution. That is, all the slack time identified is given to the current task instance. While the greedy policy is simple, it is not the best one. For example, in aggressive InterDVS algorithms such as $laEDF$, AGR and $lpSHE$, slack times may be distributed unevenly among task instances. When the current task instance exhausts its assigned slack time by the greedy distribution policy, task instances that follow may not benefit from slack times at all. In order to understand the effect of different slack distribution policies, we experimented by varying the amount of usable slack times. In the experiments, we specified the lower bound on the clock speed regardless of available slack times.

Figure 5 shows the experimental results for various minimum speeds. In each experiment, it is assumed that the clock speed can be varied within the range of $[\alpha \cdot f_{max}, f_{max}]$ with a step size of 1 MHz where $f_{max} = 100$ MHz and α is the speed bound factor. As α becomes larger, the task instances is scheduled with lowered clock speed less aggressively because the clock scaling is restricted by $\alpha \cdot f_{max}$. When $\alpha \cdot f_{max}$ is close to the lowest possible clock speed of the target machine, it is similar to when the greedy slack distribution is used. The experiments were performed varying α from 0.1 to 0.9. In Figure 5, the x -axis indicates the speed bound factor α . The energy efficiency of InterDVS algorithms (except for $lppsEDF$ and $ccEDF$) is generally higher when α values are between 0.3 and 0.5. For exam-

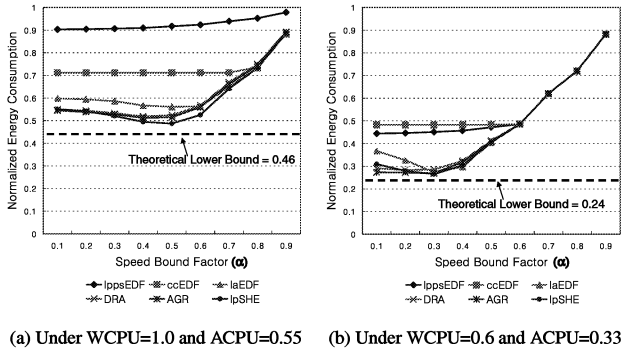


Figure 5. Impact of speed bound.

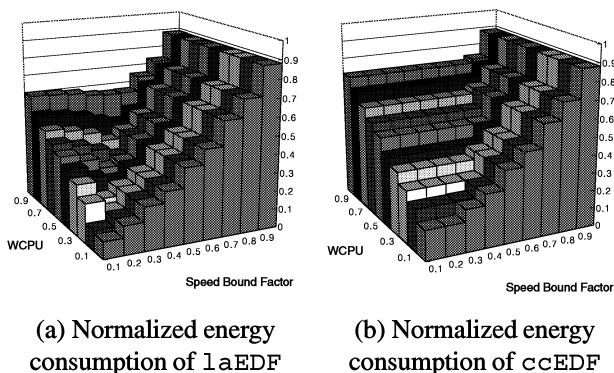


Figure 6. Impact of speed bound.

ple, when the speed bound factor is 0.5 in Figure 5(a), an improvement of 6~11% was achieved over when the greedy policy is used.

In Figure 5, it is shown that the energy efficiency of AGR and lpSHE is very close to the theoretical lower bound³ when the speed bound factor is near 0.5. In fact, one interesting observation is that for the *aggressive* InterDVS algorithms, the energy efficiency is highest when the speed bound factor was set to ACPU. This trend can be noted in Figure 5(a) and 5(b).

To show the relationship between the speed bound and ACPU, extensive experiments were performed for various task sets while varying ACPU and scaling bound. Figure 6 shows the results. (Due to the lack of space, only the results for laEDF (an example of aggressive InterDVSs) and ccEDF (an example of non-aggressive InterDVSs) are shown. (The results for AGR and lpSHE are very similar to that of laEDF.) The results confirm that when the selected speed bound factor is close to ACPU ($= 0.55 \times \text{WCPU}$), the

³The theoretical lower bound is computed with the complete execution trace information using Yao's algorithm [20].

best energy efficiency is achieved for laEDF. For ccEDF, however, this trend does not hold as we can notice in Figure 6(b).

Similar study with the RM InterDVS algorithms show that the performance gap between the energy efficiency of the RM InterDVS algorithms and that of the theoretical lower bound was roughly 35~40%. This result indicates that there is a substantial room for improvement in developing more energy-efficient RM InterDVS algorithms.

5.2 Performance evaluation of Intra-Task DVS algorithms

We have evaluated the energy efficiency of intraShin and intraGruian using an MPEG4 video decoder and an MPEG4 video encoder that were previously used in [16]. Both applications were pre-processed for speed/voltage changes as described using the tools in the IntraDVS pre-processing module described in Section 4.2.

For intraGruian, the execution times of both the MPEG4 decoder and encoder were assumed to follow a normal distribution $N_o = N(m_1, (\frac{m_2}{6})^2)$ where $m_1 = \frac{1}{2} \times \text{WCET}$ and $m_2 = \frac{9}{10} \times \text{WCET}$.

For intraShin, we first collected a large number of execution paths; in SimDVS, each execution path can be represented by a pair of parameters [17]. For each execution path, we estimated the energy consumption of the execution path using the IntraDVS simulator. The overall average energy consumption is computed by taking the weighted average of estimated energy consumptions using the execution path distributions used for intraGruian.

Since the energy efficiency of intraGruian largely depends on the slack ratio⁴ given in the on-line phase and the accuracy of the execution time distribution used in the off-line profiling, we performed experiments varying these two factors. Figure 7 shows the relative energy consumption ratio of intraGruian over intraShin. If the ratio is larger (smaller) than 1, intraGruian performs better (worse) than intraShin. In Figure 7, the N_o line represents the case when the actual execution times follow the assumed N_o distribution. The N_a , N_b and N_c lines indicate the cases where the actual execution times follow different normal distributions from the assumed N_o , where $N_a = N(m_1, (\frac{m_2}{5})^2)$, $N_b = N(m_1, (\frac{m_2}{7})^2)$ and $N_c = N(1.5 \cdot m_1, (\frac{m_2}{7})^2)$.

When the slack ratio is less than 1.2, intraShin outperforms intraGruian because intraShin spends more time in the lower speed region than intraGruian. When the slack ratio is increased, intraGruian spends more time in the lower speed region than intraShin. Figure 7 also shows that intraShin works better than in-

⁴The slack ratio is defined as the ratio of WCET to the assigned execution time.

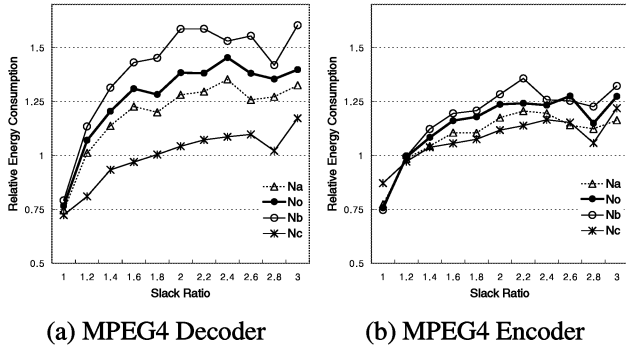


Figure 7. Energy consumption ratio of intraShin and intraGruian.

traGruian when the distribution of actual execution times is significantly different from the assumed distribution, as shown in the N_c line.

5.3 Performance evaluation of hybrid methods

In this section, the question of whether HybridDVS algorithms will perform better than pure IntraDVS algorithms or pure InterDVS algorithms is investigated. Although both intraShin and intraGruian can be used for a comparative study, we use intraShin as a base IntraDVS algorithm. This is because intraShin is less likely to generate dynamic slack times, thus making the distinctions among the different HybridDVS methods clearer.

HybridDVS algorithms select either the *intra mode* or the *inter mode* when slack times are produced during the execution of the current task instance. In the inter mode, the slack time is used not for the current task instance but for the following task instances. In the intra mode, all the slack times are used for the current task instance, allowing it to execute at a lower speed. Table 3 summarizes four heuristics [17] for HybridDVS algorithms considered in this section. The heuristics differ in how close they are to the pure IntraDVS approach or pure InterDVS approach.

We have experimented four heuristics in Table 3 with six EDF InterDVS algorithms and two RM InterDVS algorithms in Table 2. H1 and H3 are close to the pure InterDVS approach and H2 is close to the pure IntraDVS approach. The performance of HybridDVS algorithms depends on the dynamic slack estimation methods adopted by each InterDVS algorithm. In 1aEDF, DRA, AGR, and lpSHE where slack times are identified more aggressively, it is a good idea that some (or all) slack times produced by the current task instance are passed to the following tasks. However, in lppsEDF/RM and ccEDF/RM where slack times are less aggressively identified, it is better for the current task in-

Table 3. Four heuristics for HybridDVS algorithms.

Heuristic	Description
H1	uses the inter mode as a default but uses the intra mode if no activated task instance exists.
H2	uses the intra mode first, but changes into the inter mode when the current task instance has used a predefined amount of slack time.
H3	uses the inter mode first, but changes into the intra mode when the unused slack time is more than a predefined amount of slack time.
H4	alternates the intra mode and the inter mode keeping the balance of slack consumption in each mode.

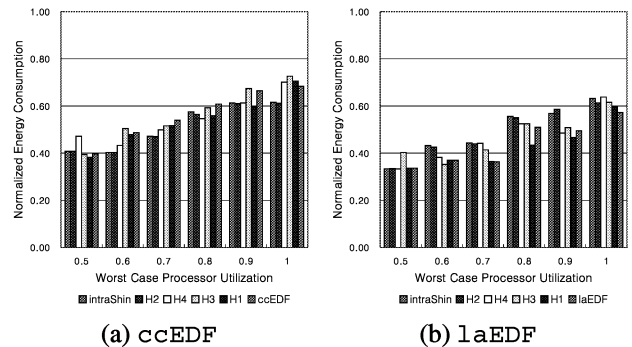


Figure 8. Energy efficiency of HybridDVS algorithms.

stance to utilize most of the slack time generated. Therefore, if a HybridDVS is based on 1aEDF, DRA, AGR, or lpSHE, H1 and H3 are better choices. On the other hand, for lppsEDF/RM and ccEDF/RM, H2 and H4 are better choices.

Figure 8 shows the energy efficiency of the HybridDVS methods. The graphs show the energy consumption for various WCPUs. As explained before, if a HybridDVS algorithm is based on a non-aggressive InterDVS algorithm, the heuristic H2 gives good results as shown in Figure 8(a). For an aggressive InterDVS algorithm, H1 and H3 give good results as shown in Figure 8(b). Though the performance of HybridDVS algorithms is also dependent on the properties of the task set tested and the execution time variations, in these experiments, HybridDVS algorithms are shown to reduce the energy consumption by 5~20% over that of the pure DVS algorithms.

6 Conclusions

We have compared the energy efficiency of recent DVS algorithms for hard real-time periodic tasks. The evaluated DVS algorithms include eight InterDVS algorithms and two

IntraDVS algorithms. We also performed experiments with four versions of HybridDVS algorithms. For a fair and efficient comparative study, we have also developed SimDVS, a unified DVS simulation environment.

Our comparative study shows that the existing EDF InterDVS algorithms such as AGR, 1aEDF and 1pSHE are close to optimal; for our test task sets, their power consumption is only 9~12% worse than the theoretical lower bound. We demonstrated that the performance gap from the theoretical lower bound can be further reduced with a more intelligent slack distribution policy. However, in the RM InterDVS algorithms, our study indicates that there is still a significant performance gap from the theoretical lower bound. Therefore, our findings strongly suggest that more research should be directed toward developing better RM InterDVS algorithms.

From the evaluation of IntraDVS algorithms, we demonstrated that two representative IntraDVS algorithms perform quite differently depending on available slack times. Our study indicates that the performance of a HybridDVS algorithm can be better than a pure IntraDVS algorithm or a pure InterDVS algorithm. However, the differences in energy efficiency depend on the characteristics of both the IntraDVS and the InterDVS components used in the HybridDVS algorithm. One of interesting future research topics will be to devise an intelligent guideline on selecting the best HybridDVS algorithm for a given task set.

References

- [1] AMD Corporation. PowerNow! Technology. <http://www.amd.com>, December 2000.
- [2] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, December 2001.
- [3] T. Burd and R. Brodersen. Design Issues for Dynamic Voltage Scaling. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 9–14, July 2000.
- [4] D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. Technical Report 1342, University of Wisconsin-Madison, CS Department, June 1997.
- [5] F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 46–51, August 2001.
- [6] D. Grunwald, P. Levis, and K. I. Farkas. Policies for Dynamic Clock Scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pages 73–86, October 2000.
- [7] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 178–187, December 1998.
- [8] Intel Corporation. Intel XScale Technology. <http://developer.intel.com/design/intelxscale/>, November 2001.
- [9] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.
- [10] W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. In *Proceedings of Design, Automation and Test in Europe (DATE'02)*, pages 788–794, March 2002.
- [11] S. Lee and T. Sakurai. Run-time Voltage Hopping for Low-power Real-Time Systems. In *Proceedings of the 37th Design Automation Conference*, pages 806–809, June 2000.
- [12] W.-S. Liu. *Real-Time Systems*. Prentice Hall, Englewood Cliffs, NJ, June 2000.
- [13] T. Pering and R. Brodersen. Energy Efficient Voltage Scheduling for Real-Time Operating Systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, Work in Progress Session*, June 1998.
- [14] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 89–102, October 2001.
- [15] T. Sakurai and A. Newton. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas. *IEEE Journal of Solid State Circuits*, 25(2):584–594, 1990.
- [16] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design and Test of Computers*, 18(2):20–30, March 2001.
- [17] D. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min. SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceedings of Workshop on Power-Aware Computer Systems (PACS 2002)*, February 2002.
- [18] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 365–368, November 2000.
- [19] Transmeta Corporation. Crusoe Processor. <http://www.transmeta.com>, June 2000.
- [20] F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proceedings of the IEEE Foundations of Computer Science*, pages 374–382, October 1995.

Task Level Speed Scheduling on Dynamic Voltage Supply Processors

Flavius Gruian

Embedded Systems Design Laboratory
Lund Institute of Technology, Sweden

Task Level Speed Scheduling on Dynamic Voltage Supply Processors

Flavius Gruian



Embedded Systems Design
Laboratory



Lund Institute of Technology
Sweden



Outline

- A Taxonomy of Techniques
- Models & Assumptions
- Intrusive DVS (Compiler assisted)
- Non-Intrusive DVS (Stochastic)
- Comparison
- Improvements
- Soft RT Considerations
- Conclusions

2(27)



A Taxonomy

- depending on the scheduling decisions
 - offline (static) vs. runtime (dynamic)
 - OS level vs. user (task) level
- depending on the task characteristics
 - fixed vs. variable execution pattern
 - hard vs. soft deadlines
- depending on the level of intrusion
 - control flow sensitive vs. instance history sensitive



Models & Assumptions

Task:

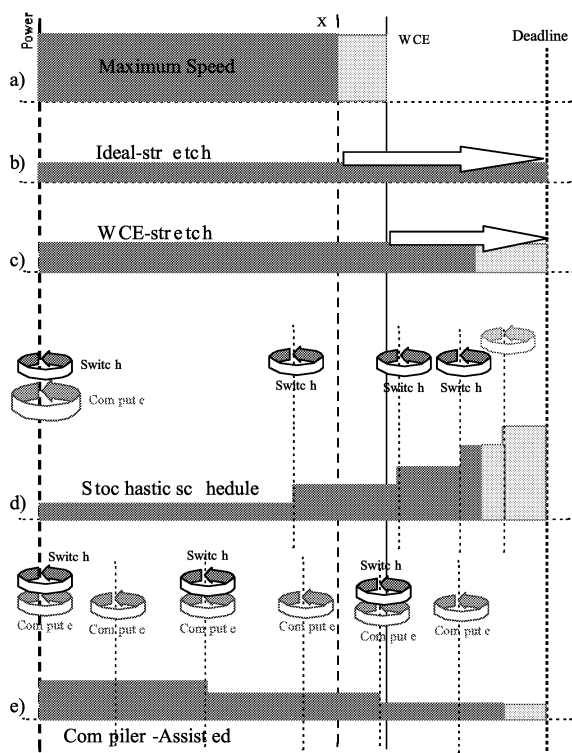
- consists of atomic computational steps:
clock cycles (C)
- variable execution patterns can be described using
probability distribution functions ($\eta(x)$)
- allowed to execute for a given time interval (A)

Speed switches (usual but not limiting assumptions):

- time & energy overheads are negligible
- can occur with unlimited frequency



Task Level Scheduling

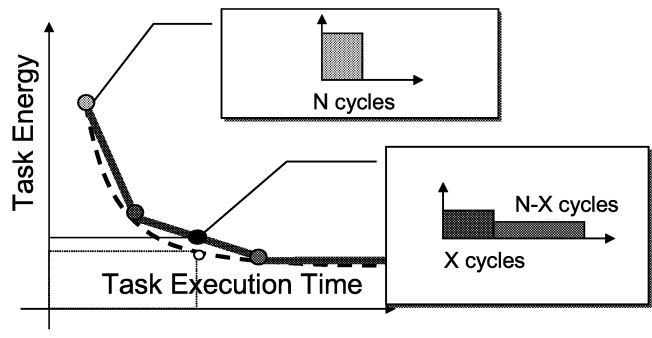
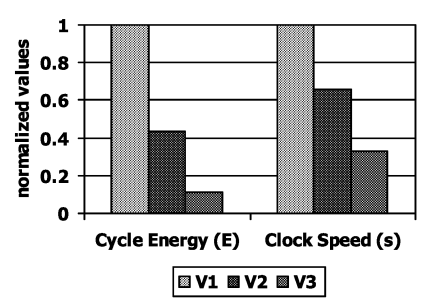


Schedule types:

- a) no voltage scaling
 - b) known instance run time
 - c) unknown runtime
 - d) accelerated execution
 - e) decelerated execution
- a,b: single* speed, static
d,e: multi-speed, dynamic*

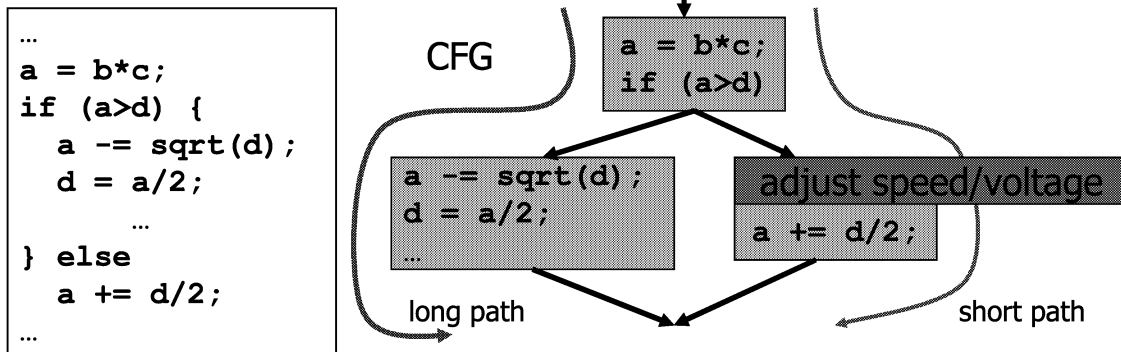
The Dual Speed Model

- running at any virtual speed on processors with limited number of speeds
- [Ishihara98] any virtual speed: optimal to use only the two speeds bounding the virtual speed



Intrusive DVS at a Glance

- Adapts the speed to the instance processor demand



- after each decision that may reduce the worst case path, new code for adjusting speed and voltage is inserted

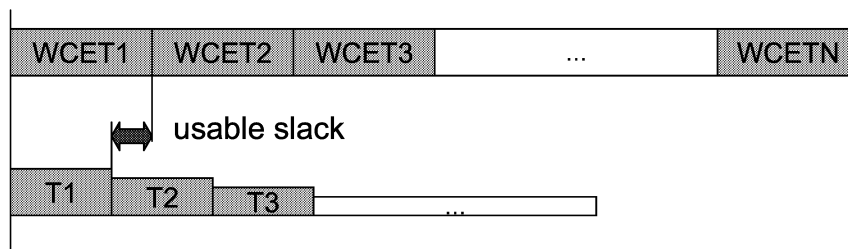
Intrusive DVS Methods

- require information about task internals
 - block-wise worst case execution time
 - control flow graph
- inserts code snippets at specific points
 - re-computing the optimal speeds
 - switching speeds
 - inform the OS about workload variations
- usually compiler assisted
- usually user-level speed switching (not OS)

Intrusive DVS Examples (I)

[Lee&Sakurai00, Mosse00, Shin&Sakurai01]

- slice the application in known WCET but arbitrary parts (loop boundaries, calls)
 - how to select these points?
- insert re-scheduling code between each



Intrusive DVS Examples (II)

[Hsu&Kremer01]

- slow-down whenever processor speed has little impact on performance (memory busy)
- top-down, single region selection
- 10% energy reduction at 2% performance degradation

[AbouGhazaleh03]

- insert hints rather than rescheduling points and let the OS select the speed at periodic intervals
- work at groups of basic blocks level



Non-Intrusive DVS Methods

- do not change the tasks
- require only information about the execution pattern: average cycles, probability distribution,...
- decide the full schedule (speed sequence) before the instance starts executing
- usually OS level
- usually do not require detailed and expensive offline task analysis



Non-Intrusive DVS Examples

[Lorch01]: the PACE approach

[Gruian01b]: stochastic scheduling

- both use the cumulative probability distribution function of execution cycles
- minimize the energy over a large number of instances
- accelerated execution
- PACE chooses the switching points and speeds
- Stochastic scheduling uses the existing real speeds to build the schedule



Deriving a Stochastic Schedule

- $\eta(x)$ probability of a task finishing at cycle x
- $cdf(x)$ probability of a task finishing before x

1. expected energy: $\bar{E} = \sum_{x=1}^{WCE} (1 - cdf(x))e_x$

2. energy for the x th clock:

$$e_x = e_{ref} \left(\frac{f_x}{f_{ref}} \right)^\beta \approx \frac{1}{k_x^\beta}$$

3. constraint on execution time:

$$\Rightarrow \text{E Lower Bound for: } \sum_{x=1}^{WCE} k_x \leq A$$

($\beta = 2$)

$$k_y = A \frac{\sqrt[3]{1 - cdf(y)}}{\sum_{x=1}^{WCE} \sqrt[3]{1 - cdf(x)}}$$



Deriving a Stochastic Schedule (cont'd)

- k_y ideal (virtual) clock lengths have to be mapped to real (available) clock lengths CK_i
- ☺ distribute the work of each virtual clock:

$$k_y = w_{iy} CK_i + w_{(i+1)y} CK_{i+1}$$

$$w_{iy} + w_{(i+1)y} = 1$$

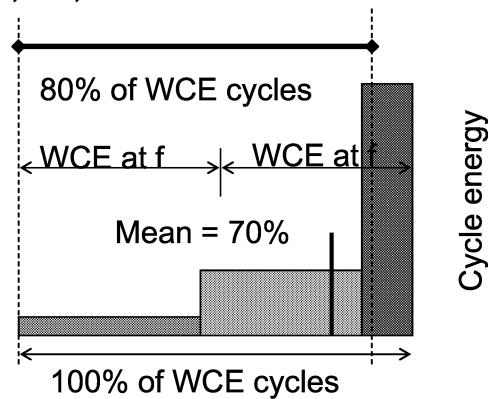
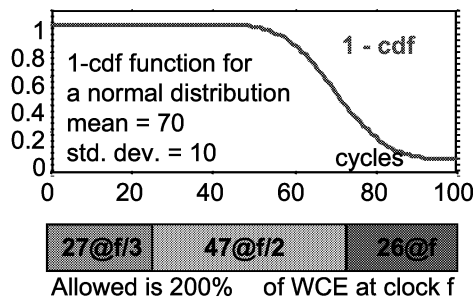
- real workloads: $k_y \in (CK_i, CK_{i+1}]$

$$w_j = \sum w_{jy} \quad j \in 1 \dots VL$$



Stochastic Schedule Example

Obtaining a stochastic schedule for a processor with clock speeds f , $f/2$, $f/3$



More on Stochastic Scheduling

- algorithmic complexity $O(\text{WCE} \log \text{VL})$
 - use groups of clock cycles to reduce complexity
- can be employed at runtime for every instance, if the allowed time changes
- requires maximum VL-1 speed switches
- easy to adapt for soft deadlines
- oblivious to task internal structure



Stochastic vs. WCE-Stretch

- Energy consumption:

$$E_{stoch} = K \frac{1}{A^2} \left(\sum_{x=1}^{WCE} \sqrt[3]{1 - cdf(x)} \right)^3$$

$$E_{WCE-stretch} = e_{WCE-stretch} \bar{X} = K \frac{1}{A^2} WCE^2 \bar{X}$$
- ratio:

$$\varepsilon = \frac{E_{stoch}}{E_{WCE-stretch}} = \frac{\left(\sum_{x=1}^{WCE} \sqrt[3]{1 - cdf(x)} \right)^3}{WCE^2 \bar{X}}$$
- dependent on the distribution shape
- independent on the allowed execution time

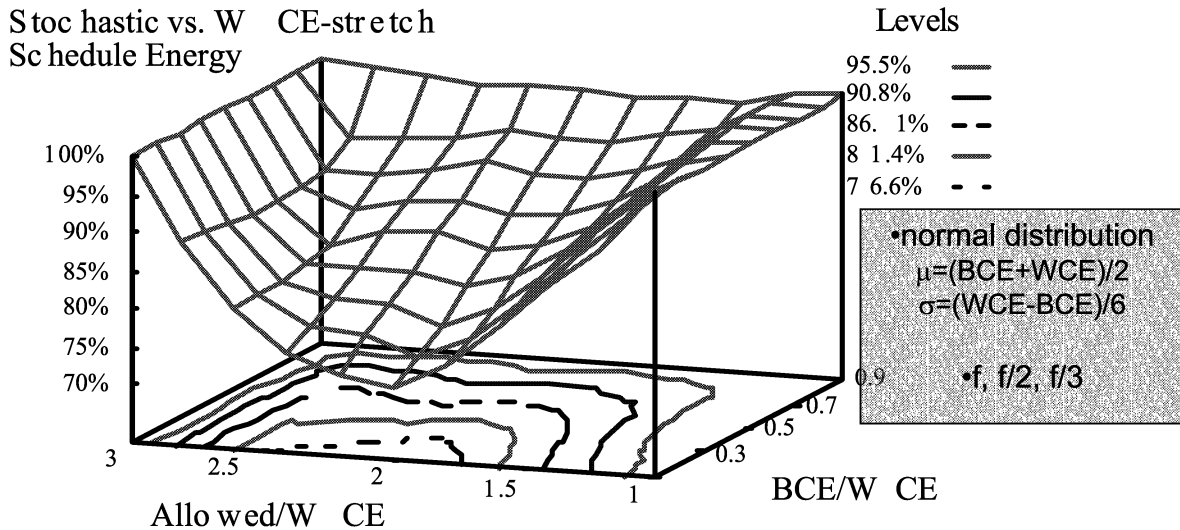


Stochastic vs. WCE-Stretch: Experiments

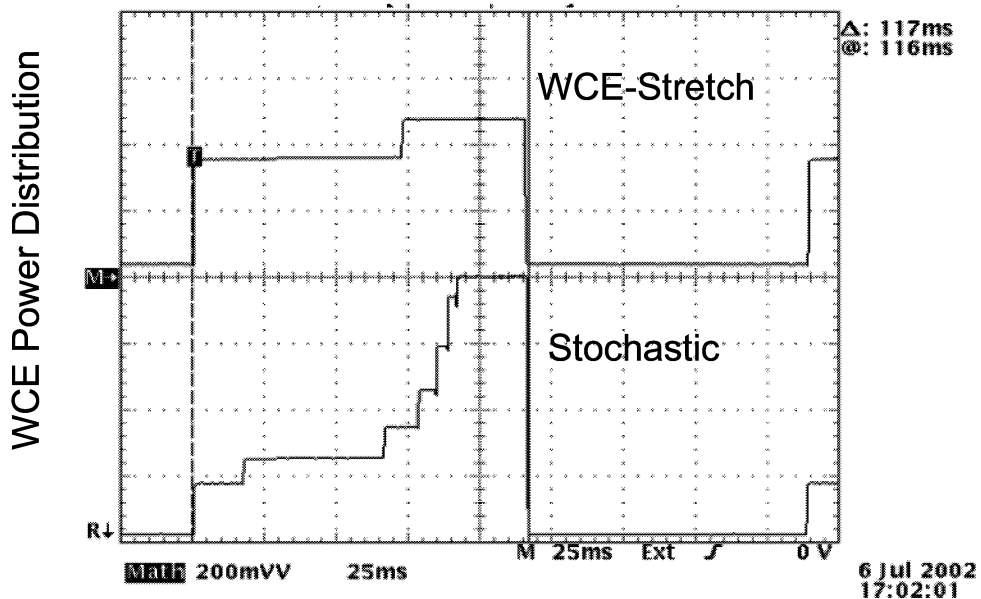
Distribution	ε
Uniform (0, WCE)	0.8419
Normal ($\mu = WCE/2$, $\sigma = WCE/6$)	0.6461
Normal ($\mu = 2WCE/3$, $\sigma = WCE/9$)	0.7417
Normal ($\mu = WCE/3$, $\sigma = WCE/9$)	0.3048
Exponential ($\lambda = 10/WCE$)	0.2434



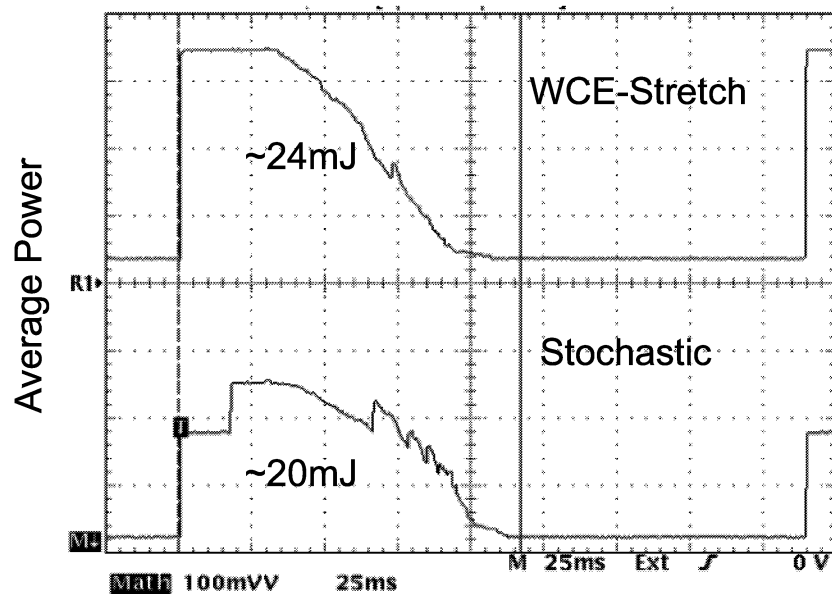
Stochastic vs. WCE-Stretch: More Experiments



Example: a WCE-Stretch and a Stochastic Schedule on i80200 (I)



Example: a WCE-Stretch and a Stochastic Schedule on i80200 (II)



July 21-25

ESSES 2003 Task Level DVS

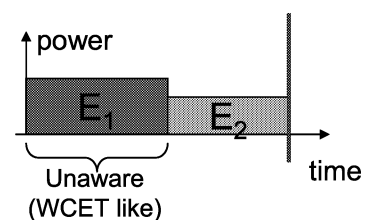
Intrusive vs. WCE-Stretch

- Energy:
 - Intrusive energy is very much dependent on the internal task structure and rescheduling points

Example of Energy Efficiency Analysis:

- normal distribution, $\mu = WCE/2$, $\sigma = WCE/6$
- unawareness factor $\vartheta = X_1/X$
- $\beta = 2$, ideal processor

$$E_{Intr}(X) = K \left(\frac{WCE}{A} \right)^2 X \left[\vartheta + \frac{(1 - \vartheta)^3}{(WCE/X - \vartheta)^2} \right]$$

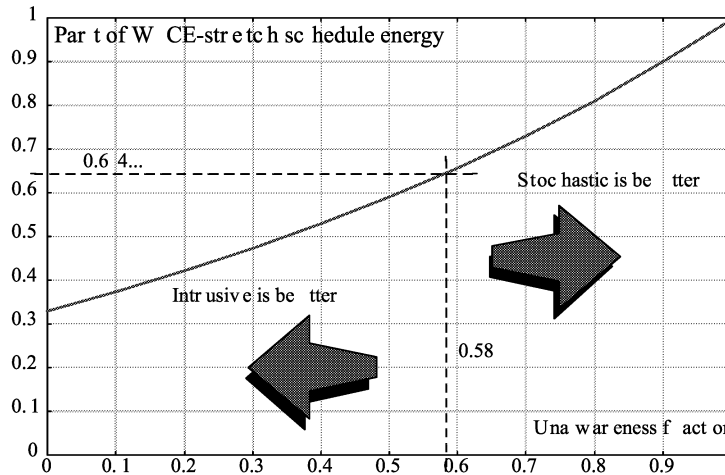


July 21-25

ESSES 2003 Task Level DVS

Analysis Example (cont'd)

$$\mathcal{E}' = \frac{E_{Intr}}{E_{WCE-stretch}} = \mathcal{G} + \frac{(1-\mathcal{G})^3}{\bar{X}} \int_0^{WCE} \frac{x}{(WCE/x - \mathcal{G})^2} \eta(x) dx$$



- numerical integration for \mathcal{G} in $0 \dots 1$

- independent of WCE and A

Intrusive vs. Stochastic

Other aspects

Feature	Intrusive	Stochastic
1. Task-transparent?	No	Yes
2. OS-transparent?	Yes/Partially	Possibly
3. Needs compiler/timing tool support?	Yes	No/little
4. Can use execution history?	Possibly	Yes
5. Run-time interference:	Moderate/high	Low/moderate
6. Sensitive to internal task structure?	Yes	No
7. Sensitive to deadline variations?	Slightly	Very

Improvements

- intrusive:
 - online (runtime) profiling
 - use hints instead of speed switches
 - use hint location cache (at jmps, calls)
- mixed approaches:
 - start using average case speed
 - increase the speed if over the average case
 - decrease the speed if under the average case



Soft RT Considerations

- deadline miss percentage = easier to control in stochastic scheduling
- deadline overshoot = easier to control in intrusive scheduling
- discarding specific cases (execution paths) = easier in intrusive scheduling



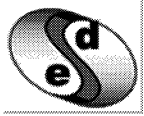
Conclusions

- many readily available techniques
- no absolutely best approach
- application specific trade-off:
method complexity vs. energy efficiency
- analyze the whole system: task level benefits may even out at system level



Hard Real-Time Scheduling for Task Groups on Dynamic Voltage Supply Processors

Flavius Gruian



Embedded Systems Design
Laboratory



Lund Institute of Technology
Sweden



2(55)

Outline

- Motivation
- A Taxonomy
- Models & Assumptions
- Offline Scheduling
(Proportional Stretch, LEnes, MRS,...)
- Runtime Scheduling (RMS with Slack Management)
- Advanced Methods (Uncertainty Based Ordering)
- Conclusions



Why Do We Need Task Group Level DVS ?

- classic scheduling:
 - cannot decide both task timing and duration
- multiple-voltage scheduling (in HLS):
 - cannot handle varying processor characteristics
- only task-level DVS + classic scheduling:
 - sub-optimal slack management (processor level)
 - sub-optimal resource utilization in heterogeneous systems



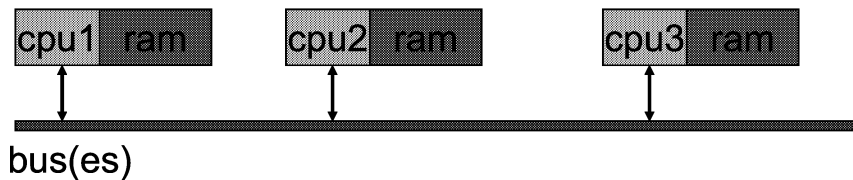
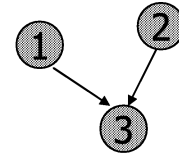
A Taxonomy

- depending on the hardware architecture
 - uni-processor vs. multi-processor
 - homogeneous vs. heterogeneous
- depending on the task group characteristics
 - independent (sets) vs. dependent (graphs)
 - fixed vs. variable execution
 - hard, firm, soft deadlines
 - periodic vs. a-periodic
- depending on the scheduling decisions
 - offline (static) vs. runtime (dynamic)



Working Models (I)

- task graphs (DA community)
 - describe communicating tasks (start:in, end:out)
 - usually single rate, single deadline
 - mapped on multi-processor architectures
 - critical path delay is important
 - scheduled using static cyclic executive approach



Working Models (II)

- task sets (RT community)
 - emphasize on timing, tasks have different
 - periods, deadlines, execution delays, arrival times,...
 - on uni-processor systems
 - individual response times are important
 - offline analysis and runtime scheduling
- hybrids: multi-rate graphs



Working Assumptions

- hard real-time deadlines
- tasks require a rather large number of clock cycles (coarse granularity)
- the power demand varies little inside a task and over tasks on a certain processor
- switching overhead is negligible
- intra-processor communication takes zero time (using shared memory)
- inter-processor communication is handled in parallel by controllers (the processor can still run tasks)



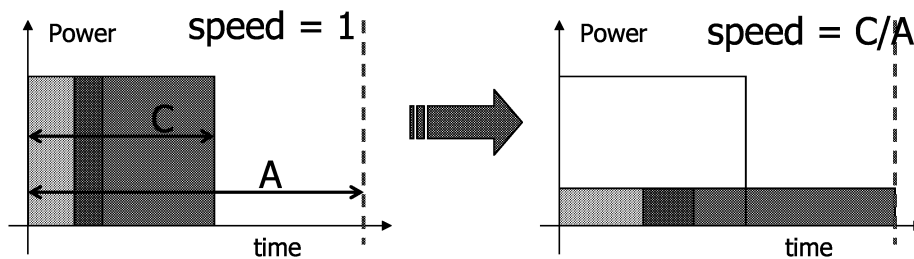
Offline (Static) Scheduling

- start times and speeds for all tasks decided before the system becomes operational
- speed switches still occur at runtime
- + simplest to implement (static cyclic executive)
- + smallest runtime overhead
- cannot use the dynamic slack resulted for tasks with variable execution time
- same problems as in classic static scheduling



The Simplest Case

- offline, uni-processor, unique period and deadline
- the ideal schedule is the one with constant speed exactly finishing at the deadline



Proportional Stretch (I)

- the easy way out: Proportional Stretch
 1. find the tightest schedule (L) using a classic algorithm assuming maximum processor speed
 2. for a deadline A use the same constant speed for all tasks, equal to L/A

Given the clock energy-speed dependency: $E(s) = E(1) \cdot s^\beta$

The energy is reduced by: $E/E_{\max} = (L/A)^\beta$

☞ Simple bound for homogeneous multi-processor:

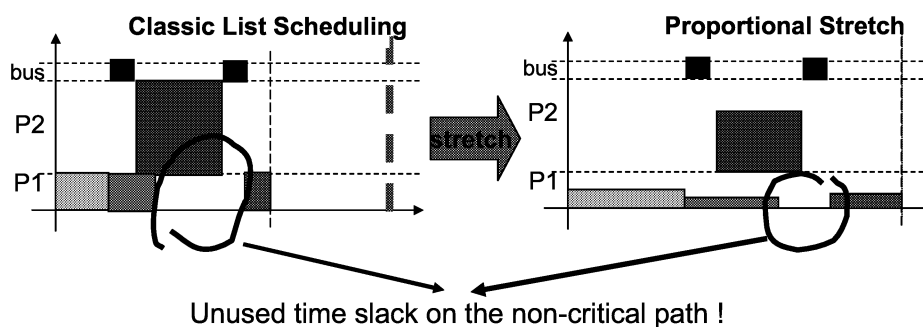
$$\frac{E}{E_{\max}} \geq \left(\frac{\sum C}{N_{proc} \cdot A} \right)^\beta$$

processor utilization



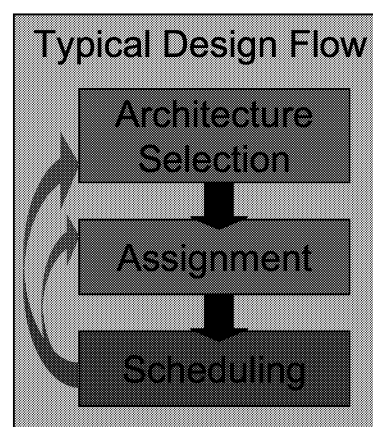
Proportional Stretch (II)

- ☞ optimal for independent, single rate tasks
- ☞ optimal for homogeneous architectures with 100% utilization
- ☞ sub-optimal for dependent tasks on multi-processors
- ☞ sub-optimal for heterogeneous architectures



Scheduling Task Graphs

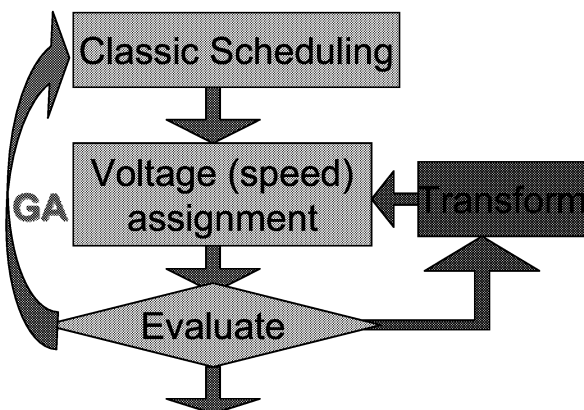
- static (offline) hard RT method
- works on graphs already assigned to processors
- even classic scheduling is a complex (NP) problem on multi-processor systems
- DVS even more so



Heuristics based on classic scheduling algorithms:
address both timing and speed (voltage)



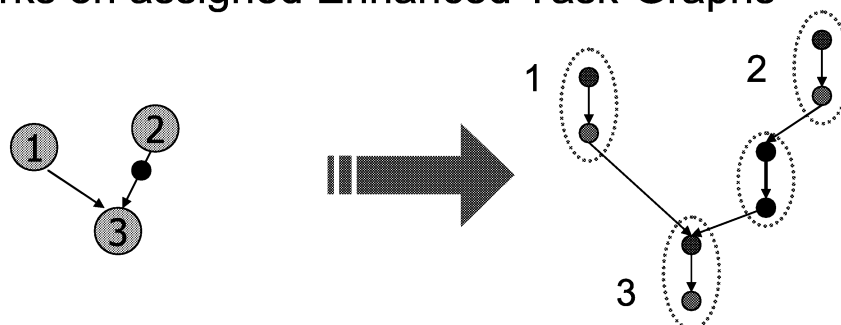
Multi-step Approaches



- Voltage assigned according to local slack (delay between task end and next start)
- [Luo00, Liu01]: Transform: shifting start times
- [Schmitz01,02]: list scheduling inside a Genetic Algorithm, handles different task power profiles

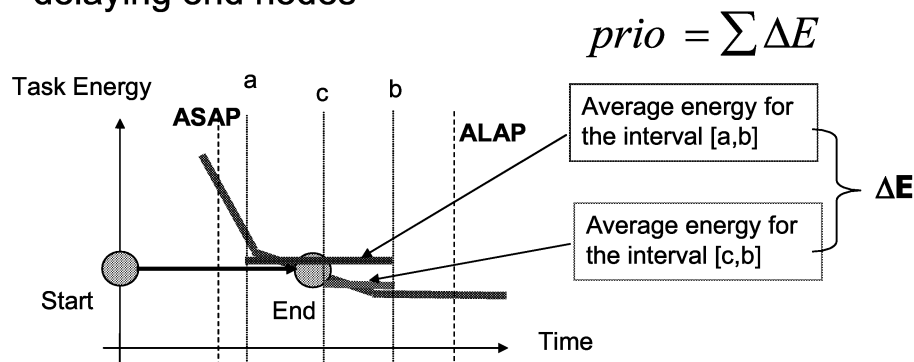
Single-step Approaches: LEneS

- Low Energy Scheduling [Gruian01]
- list-scheduling with dynamic priority recalculation
- works on assigned Enhanced Task-Graphs



The LEneS Approach: Basics

- start and end nodes are scheduled separately with list-scheduling, allowing task stretching
- priorities reflect the energy change resulted from delaying end nodes



The LEneS Algorithm: Brief Description

- All nodes start from [ASAP, ALAP] without resource constraints
- For every scheduling step decide
 - the list of schedulable nodes
 - the energy variation of scheduling a node now vs. δt later
 - schedule the right nodes
- Finish when:
 - all steps considered + all nodes scheduled: SUCCESS
 - unscheduled nodes, null interval nodes: FAIL
- Problem: no feedback on schedule length -- cannot find tight schedules for some cases



The LEneS Algorithm: Improved Version

- Solution: include “path-length” in priority

$$g(\text{node, time}) = f(\text{node, time}) + \alpha_{\text{node}} \frac{|f(\text{node, time})|}{\text{Deadline} - \text{time} - \text{Criticalpath}(\text{node})}$$

- α controls the emphasis on timing vs. energy
- use the α set to differently scale different paths
- Procedure:
 - start from all $\alpha = 0$
 - try to schedule
 - if scheduling fails, increase all α on the critical path and retry



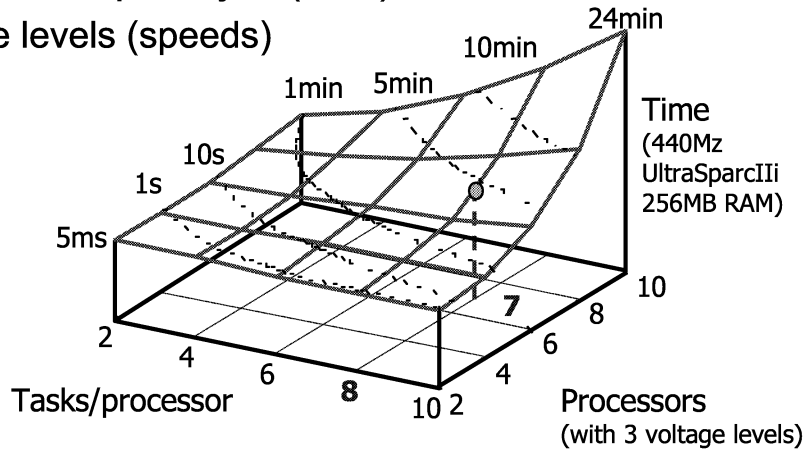
LEneS Algorithm Evaluation

- Execution time, algorithmic complexity?
- Energy reduction?
 - without performance degradation?
single voltage (max) schedule using classic LS
 - with relaxed deadlines?
vs. classic LS
vs. LS with Proportional Stretch

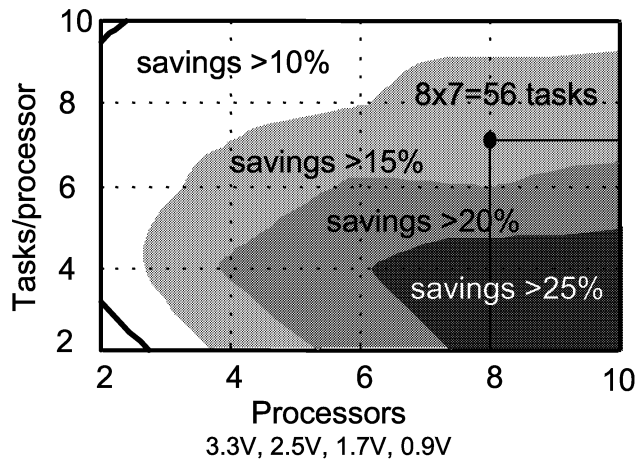


LEneS Performance: Scheduling Time

- Algorithmic Complexity: $O(VN^3)$
 - V: voltage levels (speeds)
 - N: tasks



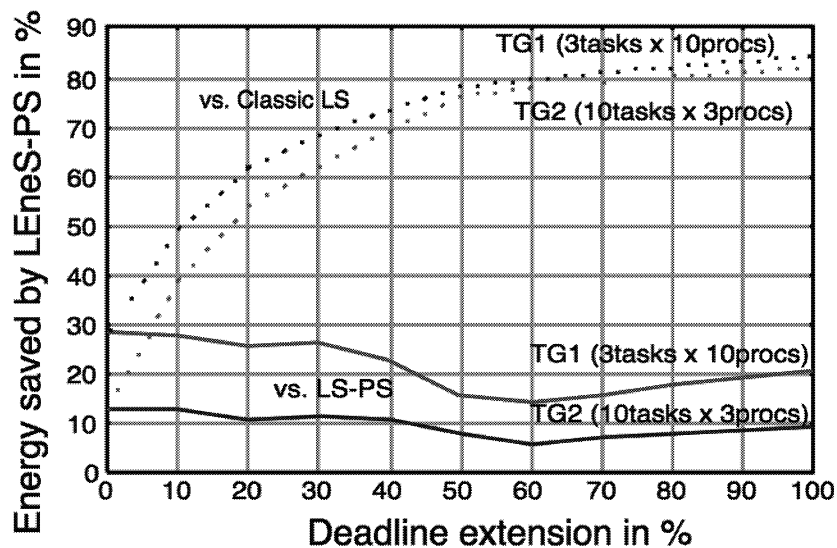
LEneS Performance: Energy Savings, Tightest Deadline



- gain increases with parallelism
- 4x smaller gains for processors with only two voltage levels (3.3V, 0.9V)



LEneS Performance: Energy Savings, Relaxed Deadlines

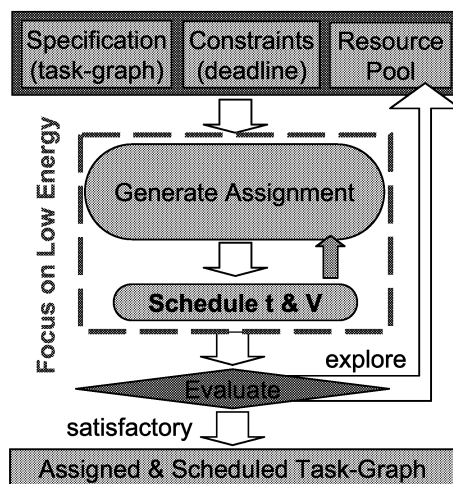


July 21-25

ESSES 2003 Task Group DVS



Combining Scheduling and Assignment



- minimize energy under time and resource constraints
- The scheduling step:
 - INPUT: assigned task-graph and a deadline
 - OUTPUT: scheduled TG

July 21-25

ESSES 2003 Task Group DVS



Simulated Annealing for Assignment

Why SA?

- classic heuristic, easy to implement
- highly tunable

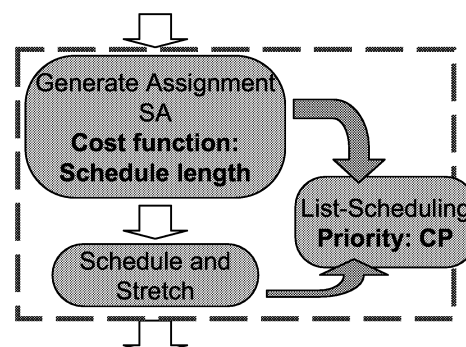
Implementation issues:

- neighborhood:
 - $\text{random}(\text{task}).\text{processor} = \text{random}(\text{processor})$
- cost:
 - quality measure of the final step solutions
 - fast to compute (ms)



The "Speed-up and Stretch" Design Flow

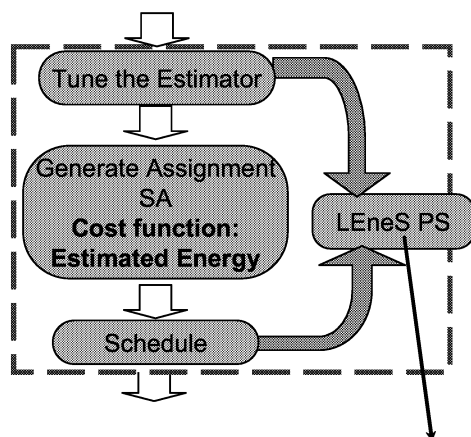
- ad hoc method
- minimize the schedule length in the assignment step
- list-scheduling with critical-path priority as a core scheduling technique
- voltage scheduling is only performed as the last step using Proportional Stretch



- ☞ conventional design algorithms(no special DVS method)
- ☞ no information about the energy until the very end!



The "Eye-on-Energy" Design Flow



- uses estimated energy as feedback
- not much more time consuming than S&S
- indirect feedback - estimates
- special DVS scheduling method

Unfortunately LEneS is too slow to be used inside the SA loop:
Solution: use a fast estimator instead!



EonE: The Energy Estimator

$$E_{\text{est}} = a \left(\frac{E_{\text{max}}}{Pr} \right) \left(\frac{T_{\text{min}}}{T_{\text{req}}} \right)^2 + b \delta_E + c$$

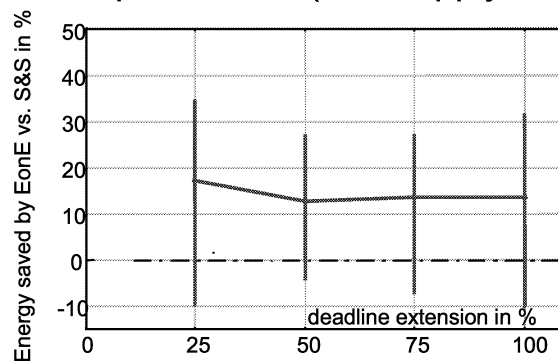
Processor Energy Load is associated with $\frac{E_{\text{max}}}{Pr}$.
Deadline Load is associated with $\frac{T_{\text{min}}}{T_{\text{req}}}$.
Processor Energy Mean Deviation is associated with δ_E .

- a, b, and c:
 - task-graph dependent
 - tuned by regression
- fast evaluation (slowest is T_{min} - Classic LS)
- under 10% avg. deviation from E final values



Experiments: EonE vs. S&S

- hundred random graphs (20 nodes)
- max four processors (four supply voltages)

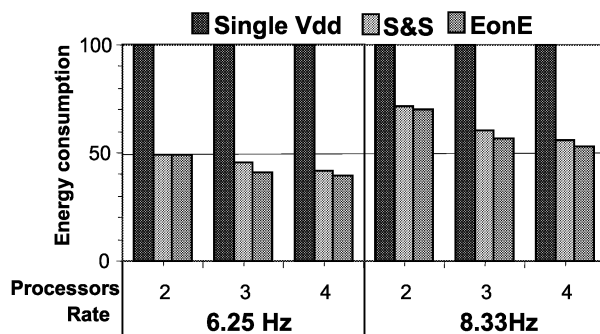


EonE to in average 15% lower energy than S&S



A Real-Life Application: Optical Flow Detection

- 32 tasks on DSPs with four Vdd
- 12.5Hz, 78x120 pixels



- >50% energy at half rate (6.25Hz)
- higher parallelism allows lower energy consumption



Scheduling Task Sets

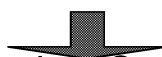
- both offline and runtime decisions (laying a complete schedule offline has exponential complexity for certain multi-rate sets)
 - offline (slow): schedulability analysis + assign minimal required speeds
 - runtime (fast): accommodate dynamic slack, tasks with varying execution time
- build on classic real-time scheduling methods
 - uni-processor architectures
 - fixed (RM) or dynamic (EDF) priorities
- extends these with assigning speeds to tasks
- task sets notation: $\{\tau_i = \langle C_i, T_i, D_i \rangle\}_{i=1..N}$



Typical Offline Decisions

Purposes:

- increase/balance processor utilization
- better use of the variations in task execution



- find the Maximum Required Speed (MRS) for each task, while keeping the schedulability
- priority refinement/energy efficient ordering
- insertion of possible preemption points/task splitting



Maximum Required Speed for EDF

1. Deadlines equal to periods ($D_i = T_i$)
 - feasible schedule if $U \leq 1$ (necessary & sufficient)
 - optimal if all execute at a constant speed: $s = U$
2. Deadlines shorter than periods ($D_i \leq T_i$)
 - feasibility analysis becomes intractable [Stankovic98]
 - assigning optimal speeds is even harder
 - sufficient condition \rightarrow speed upper bound:

$$s_{UB} = \sum_{i=1}^N \frac{C_i}{\min\{D_i, T_i\}} \quad \text{Obs. may be } > 1!$$



Maximum Required Speed for EDF

3. Deadlines shorter than period, unique common period ($D_i \leq T_i, T_i = T_j$)
 - feasibility analysis - polynomial time
 - [Yao95]: method for finding optimal speeds for tasks with different arrival times (asynchronous) (extended for fixed priorities in [Quan01])
 - similar, but simpler method for tasks having the same arrival time (synchronous) the sufficient and necessary condition:

$$\sup_{0 \leq t_1 < t_2 \leq D_N} \{u_{[t_1, t_2]}\} \leq 1$$



MRS for synchronous tasks with $D_i \leq T_i$ and $T_i = T_j = T$

Obs: loading factors change only at D_i

- examine intervals $[t_0, D_i)$, compute h_{0i}
- the highest h_{0k} defines the required speed for tasks up to k so that all deadlines are met
- move t_0 to D_k , and repeat the procedure for tasks starting with $k+1$

Example:

$\langle C, D \rangle, T=20$ $\{ \langle 1, 4 \rangle, \langle 3, 8 \rangle, \langle 2, 9 \rangle, \langle 1, 14 \rangle, \langle 3, 20 \rangle \}$

$$E_{\text{MRS}}/E_{\text{max}} \approx 0.32$$



Maximum Required Speed for RMS

Different periods, deadlines

- 1) trivial method using the Liu & Layland

sufficient condition for feasibility: $s = \frac{U}{N(2^N - 1)}$

- 2) Task specific speeds: Scheduling Points Analysis

- speeds are assigned starting with the highest priority task
- each task i has to be executed before one of:

- task i exactly meets its deadline if there exists a $S_{ij} \in S_i$

$$\sum_{1 \leq r \leq q} C_r / s_r \lfloor S_{ij} / T_r \rfloor + 1 / s_{ij} \sum_{q < p \leq i} C_p \lfloor S_{ij} / T_p \rfloor \leq S_{ij}$$



Example: MRS for EDF & RM

- 5 tasks $\langle C, T \rangle$:
 $\{ \langle 1, 5 \rangle, \langle 5, 11 \rangle, \langle 1, 45 \rangle, \langle 1, 130 \rangle, \langle 1, 370 \rangle \}$
 $U = 0.687$
 $H = \text{LCM}(5, 11, 45, 130, 370) = 476190$ ☀
- $s = 1 \rightarrow E_{\max}$
- $s_{\text{EDF}} = U \rightarrow E_{\text{EDF}} = 47.20\% E_{\max}$
- $s_{\text{RM}} = \text{scheduling points analysis} \rightarrow E_{\text{RM}} = 48.15\% E_{\max}$



MRS extensions

- accounting for different task power demand
 - add weights for each task
- accounting for switching overhead
 - adjust C to include 1...2 speed switches
- accounting for blocking in tasks with synchronization [Jejurikar02a,b]



Runtime Scheduling

- uses the dynamic slack
(analysis of a full hyperperiod may be impractical at times)
- more efficient for tasks with variable execution
- should not affect the response times
- should be fast enough (low complexity)
- should work in conjunction with the offline and task-level techniques



Runtime Approaches

- EDF
 - use the offline methods whenever a new task arrives in the system or finishes executing
 - use faster heuristics, such as AVR [Yao95]
- RM: trickier to keep the timing
 - lonely running instances are stretched until the next task arrival [Shin99]
 - if an instance generates slack, run as slow as possible for that time if computation awaits [Lee99]

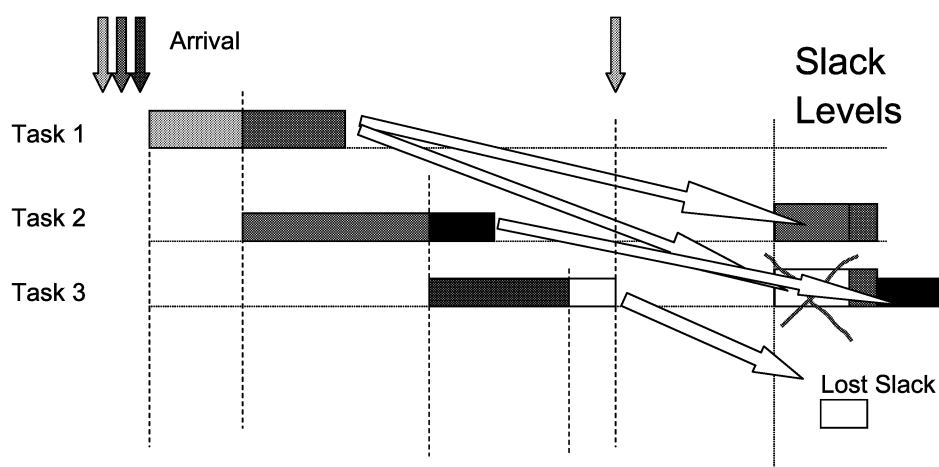


RMS with Slack Levels (I)

- Remaining time slack from early finishing instances used to slow down future tasks
- Must keep Worst Case Response Time:
 - Use slack (priority) levels
 - A task instance:
 - Will use higher level slack
 - Will produce lower level slack
 - Lowest level slack is lost



RMS with Slack Levels (II)



Slack Management Strategy (I)

S_i : the slack available at level i

- instance k of task τ_i starts executing:

$$A_i^k = C_i + \Delta C_i^k \quad S_j' = \begin{cases} 0, j \leq i \\ S_j - \Delta C_i^k, j > i \end{cases}$$

- instance k of task τ_i finishes executing:

$$\Delta A_i^k = A_i^k - X_i^k \quad S_j'' = \begin{cases} S_j, j \leq i \\ S_i + \Delta A_i^k, j > i \end{cases}$$

- idle processor times are subtracted from all slacks



Slack Management Strategy (II)

How to distribute the available slack (ΔC)
to waiting instances?

- Greedy: the next instance gets everything

$$\Delta C_i^k = S_i$$

- Mean proportional:
distribute the slack to all waiting instances according
to their mean execution time

- ...
$$\Delta C_i^k = S_i \frac{\bar{X}_i}{\sum_{\tau_j \in \text{ReadyQ}} \bar{X}_j}$$



Some Experiments (I)

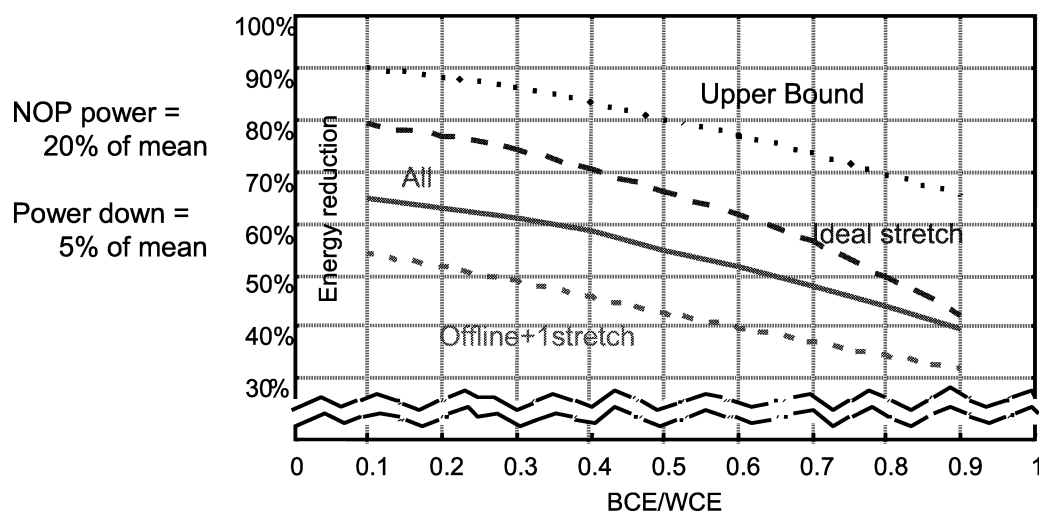
Examined cases:

- 100%
 - Tasks run at max speed, NOP during idle.
- Upper Bound
 - Post-execution analysis, all tasks uniformly stretched to max U
- Offline +1stretch
 - MRS for RM, stretch lone running tasks to the next arrival
- All
 - + Use the described slack management strategy & task-level DVS
- Ideal
 - + The exact execution time of an instance is known at arrival time

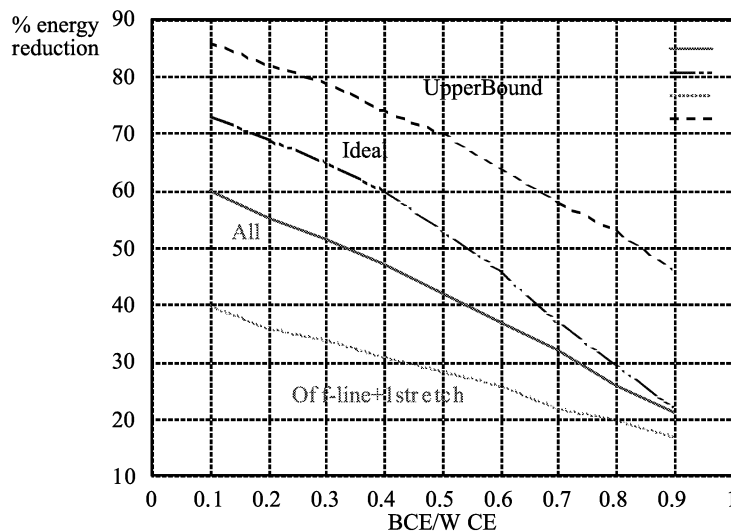


Some Experiments (II)

randomly generated sets of 100 tasks



Some Experiments (III)



Generic
Avionic
Platform

16 tasks



Advanced Methods

- More important changes to classic real-time scheduling strategies
 - Tamper with priorities/task ordering
 - Change the task amount/characteristics
- Designed for more realistic task models
 - variable execution pattern
- Use more task-level information
 - average execution time
 - execution time probability distribution



Uncertainty-Based Ordering (I)

- Problem:

- tasks: period=deadline, variable execution time
- off-line (static) ordering but
- run-time speed selection

- speed for the k^{th} task

$$s(\tau_1, \dots, \tau_{k-1}) = \frac{\sum_{i=1}^N WCE_i}{Af_{ref} - \sum_{j=1}^{k-1} X_j}$$

- energy for a period
(clock ene. $e(s)=Ks^\beta$)

$$E(X_{1,\dots,N}) = X_1 K_1 s_0^\beta + \sum_{i=2}^N X_i K_i s(\tau_1, \dots, \tau_{i-1})$$

- average energy

$$\overline{E(X_\pi)} = \int_{x_1} \dots \int_{x_N} E(x_\pi) \eta(x_1) \dots \eta(x_N) dx_1 \dots dx_N$$



Uncertainty-Based Ordering (II)

Example: 3 tasks, uniform distribution

$$(BCE, WCE) = \{\tau_1:(12,20), \tau_2:(10,30), \tau_3:(24,40)\}$$

$$A = 100, K=1, f_{ref}=1, \beta=2$$

Execution Type	$\overline{E[X]}$	$\overline{E[\bar{X}]}$	$\overline{E[X]}\%Ideal$
<1, 3, 2>	42.094	41.839	134%
<2, 3, 1>	37.482	36.978	119%
Ideal: always mean	31.443 (speed 0.68)		100%
Offline WCE	55.080 (speed 0.9)		175%



Uncertainty-Based Ordering (III)

- Main ideas:
 - achieve a low speed ASAP by ordering tasks wisely
 - approximate $\overline{E[X]}$ by $E[\bar{X}]$

- Priority:

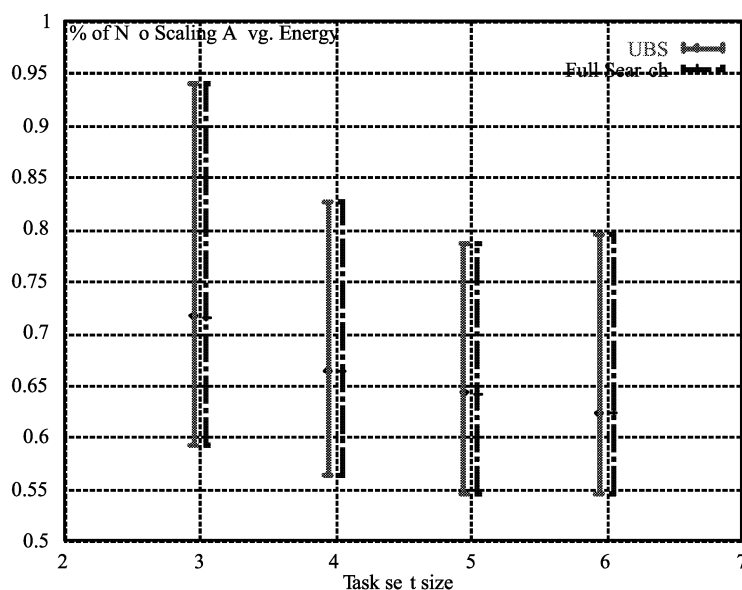
$$P_{ubs}(\tau_k) = \frac{K_k \bar{X}_k}{S^\beta - S_k^\beta}$$

- Observations:

- prioritize short tasks
- prioritize tasks with large variation in execution
- prioritize power efficient tasks
- algorithmic complexity $O(N^2)$



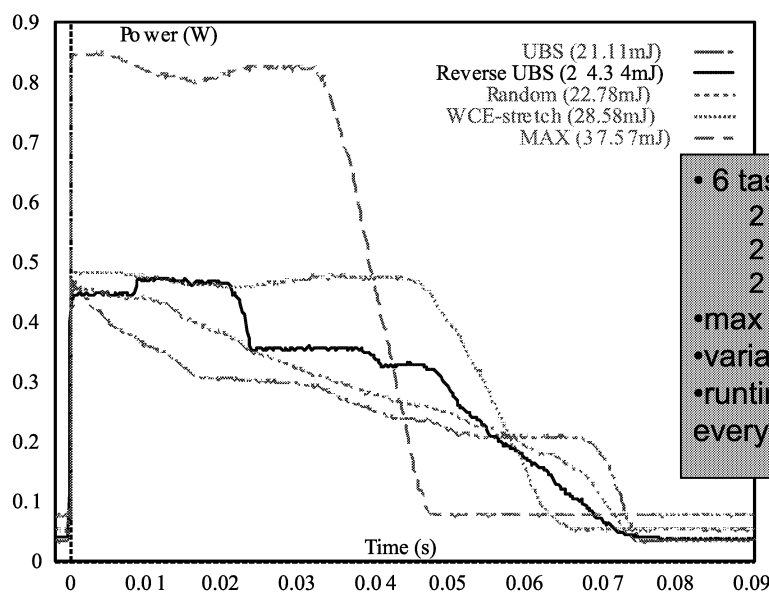
UBO vs. Full Search



- 300 sets of each size
- used the “real” E formula
- under 2% difference



UBO example on i80200



- 6 tasks
 - 2 LZ (K=770mW)
 - 2 QS (K=840mW)
 - 2 FOR (K=800mW)
- max speed time 49ms
- variation 17ms
- runtime rescheduling after every five H

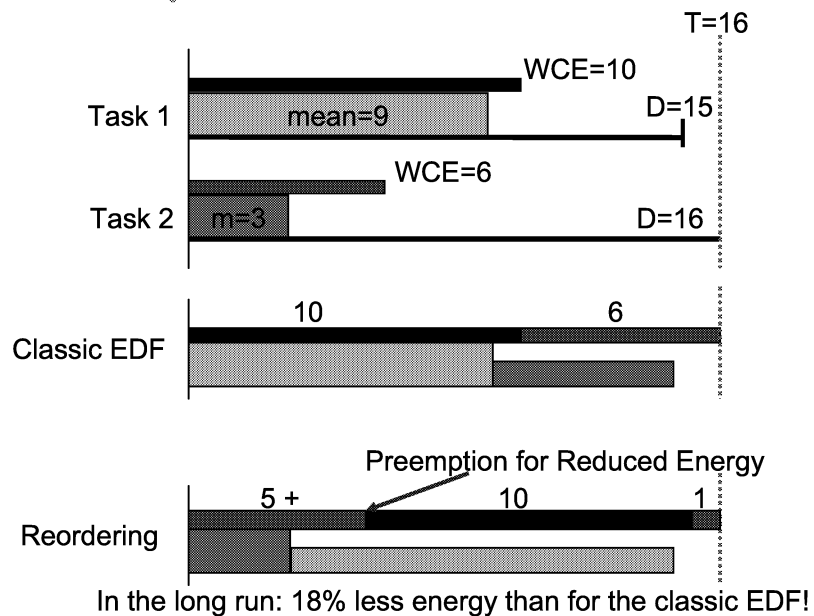


UBO extension to EDF

- Use preemption to extract regions
- Push forward uncertain regions
- Algorithm:
 1. Start from the latest deadline
 2. Between two consecutive deadlines order the regions according to the already given priorities
 3. Preempt the task which does not fit entirely
 4. Proceed with the next consecutive deadlines



An Example of UBO EDF



Some Soft RT Considerents

Techniques:

- increase the dynamic slack by skipping instances
- modify execution time probability distribution to account for zero-length instances

Yet:

- avoid discarding tasks in the middle of execution (don't waste energy)
- keep the QoS or the rate of completion



Conclusions

- wide range of task group DVS methods
- can be combined with task level methods
- can be successfully employed on already existent DVS processors (5-6 speeds)
- use more information about the tasks to obtain better methods
- even hard real-time applications can use DVS



Dynamic Voltage Scheduling

F. Gruian

July 10, 2003

References

1 Introduction

1.1 Recommended

- [BPSB00] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid State Circuits*, 35(11), November 2000.
- [PBB00] T. A. Pering, T. D. Burd, and R. W. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *Proc. of the 2000 ISLPED*, pages 96–101, 2000.

1.2 Useful

- [ADI] ADI Engineering. *80200EVB Reference Platform*. <http://www.adiengineering.com/product80200EVB.html>.
- [AMD00] AMD. *AMD PowerNow™ Technology Dynamically Manages Power and Performance, Rev. A*, November 2000. Informational White Paper No. 24404.
- [Fle01] M. Fleischmann. LongRun power management - dynamic power management for crusoe processors. Technical report, Transmeta Corporation, January 17, 2001.
- [Int01] Intel. *Intel® 80200 Processor based on Intel® XScale™ Microarchitecture Datasheet*, September 2001. Order Number: 273414-003.
- [RP96] J. M. Rabaey and M. Pedram, editors. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

2 Task Level DVS

2.1 Recommended

- [ACM⁺03] N. AbouGhazaleh, B. Childers, D. Mosse, R. Melhem, and M. Craven. Energy management for real-time embedded applications with compiler support. In *ACM SIGPLAN Languages, Compilers, and Tools for Embedded Systems (LCTES'03)*, 2003.
- [Gru01a] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In *Proc. of the 2001 ISLPED*, pages 46–51, August 6–7 2001.
- [Gru01b] F. Gruian. On energy reduction in hard real-time systems containing tasks with stochastic execution times. In *IEEE Workshop on Power Management for Real-Time and Embedded Systems*, pages 11–16, May 29 2001.
- [IY98] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of the 1998 ISLPED*, pages 197–202, 1998.
- [LS01] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proc. of ACM SIGMETRICS 2001*, pages 50–61, 2001.
- [SKL01] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design & Test of Computers*, 18(2), March-April 2001.
- [SKS01] Y. Shin, H. Kawaguchi, and T. Sakurai. Cooperative voltage scaling (CVS) between OS and applications for low-power real-time systems. In *Proc. of the 2001 ICICC*, pages 553–556, 2001.

2.2 Useful

- [HK01] C.-H. Hsu and U. Kremer. Compiler-directed dynamic voltage scaling based on program regions. Technical Report DCS-TR461, Rutgers University, November 2001.
- [LS00] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of the 2000 DAC*, pages 806–809, 2000.
- [MAAM02] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mosse. *Power Aware Computing*, chapter Power Management Points in Power-Aware Real-Time Systems. Plenum/Kluwer Publishers, 2002.

- [MACM00] D. Mossè, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compilers and Operating Systems for Low-Power*, October 2000.

3 Task Group Level DVS

- [GK01] F. Gruian and K. Kuchcinski. LEneS: Task-scheduling for low-energy systems using variable voltage processors. In *Proceedings of the 2001 Asia South Pacific – Design Automation Conference*, pages 449–455, January 30 – February 2 2001.
- [GK03] F. Gruian and K. Kuchcinski. Uncertainty-based scheduling: Energy-efficient ordering of tasks with variable execution time. In *International Symposium on Low Power Electronics and Design (ISLPED'03)*, August 2003.
- [Gru00] F. Gruian. System-level design methods for low-energy architectures containing variable voltage processors. In B. Falsafi and T.N. Vijaykumar, editors, *Lecture Notes in Computer Science*, number 2008, pages 1–12. Springer, 2000. First International Workshop on Power-Aware Computer Systems.
- [LCBK01] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Proceedings of the 38th Design Automation Conference*, pages 840–845, June 2001.
- [LJ00] J. Luo and N. K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time systems. In *Proceedings of the 2000 IEEE/ACM ICCAD*, pages 357–364, 2000.
- [LK99] Y.-H. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Proc. of the 6th IC RTCSA*, pages 272–279, 1999.
- [SAHE02] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition*, pages 514–521, 2002.
- [SC99] Y. Shin and W. Choi. Power conscious fixed priority scheduling for real-time systems. In *Proc. of the 36th DAC*, pages 134–139, 1999.
- [YDS95] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of CS*, pages 374–382, 1995.

3.1 Useful

- [JG02a] R. Jejurikar and R. Gupta. Energy aware edf scheduling with task synchronization for embedded real time systems. Technical Report 02-24, CECS, UC Irvine, August 2002.
- [JG02b] R. Jejurikar and R. Gupta. Energy aware task scheduling with task synchronization for embedded real time systems. In *Proc. of the 2002 CASES*, pages 164–169, 2002.
- [Gru02] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. Doctoral dissertation, Lund University, December 2002. ISBN 91-628-5494-1, ISSN 1404-1219.
- [QH01] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the 2001 Design Automation Conference*, pages 828–833, 2001.
- [LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the 1989 Real Time Systems Symposium*, pages 166–171, 1989.
- [SSRB98] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

Sponsored by:



Low Power Circuits/Systems Design and Dynamic Voltage Scaling

Kiyoung Choi

School of Electrical Engineering and Computer Science
Seoul National University, Korea

Low Power Circuits/Systems Design and Dynamic Voltage Scaling

July 21, 2003

ESSES

Kiyoung Choi
School of EECS
Seoul National University

Introduction

Introduction

- **Low power design**
 - Increasing demand on performance and integrity of VLSI circuits
 - Popularity of portable devices
- **Low power design at higher levels of abstraction**
 - Faster design space exploration
 - Wider view
 - Higher power reduction
 - Less cost increase

- Opportunities for power reduction at every level of abstraction

System	50-90%	algorithms, HW-SW tradeoffs, supply voltage scaling, bus encoding
Architecture	40-70%	scheduling, resource binding, operand swapping
Register-Transfer	30-50%	clock gating, operand isolation, pre-computation, dynamic operand interchange, FSM encoding
Gate / Logic	20-30%	technology mapping, don't care optimization, de-glitching
Transistor	10-20%	transistor sizing
Physical	5-10%	interconnect capacitance reduction, clock-tree synthesis

- Power dissipation in CMOS circuits
 - Dynamic power dissipation (dominant)
 - Short-circuit power dissipation
 - Leakage power dissipation
- Dynamic power dissipation

$$\begin{aligned}
 P_{dynamic} &= C_{eff} V_{dd}^2 f_{clk} \\
 &= \alpha C_{phy} V_{dd}^2 f_{clk}
 \end{aligned}$$

C_{eff} : effective (switched) capacitance

f_{clk} : clock frequency

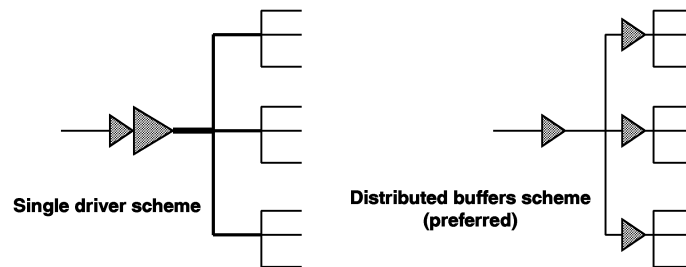
α : switching activity

V_{dd} : supply voltage

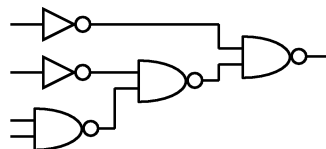
C_{phy} : physical capacitance

Physical/Transistor/Gate-Level Design

- **Interconnect capacitance reduction**
 - Signals having high switching activity are assigned short wires
- **Clock-tree synthesis**
 - Clock is a major source of dynamic power dissipation
 - Clock of 200MHz DEC Alpha chip drives 3250pF load, 3.3V supply voltage => 7W (30% of the total power)
 - Clock skews must be controlled within tolerable values

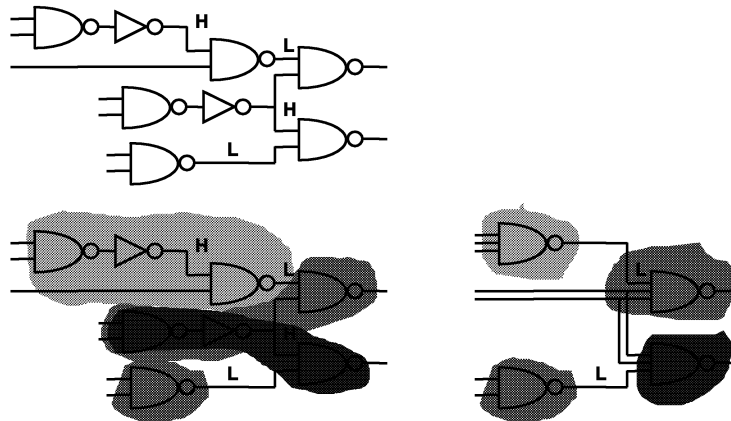


- **Transistor sizing**
 - Compute the slack at each gate
 - Sizes of the transistors in the gate are reduced until the slack becomes zero
 - Reduced size => reduced capacitance => reduced power
 - Critical path is not affected
 - Path balancing => reduced glitch => reduced power



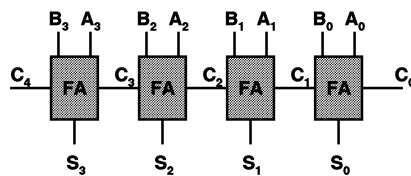
- **Technology mapping**

- V. Tiwari, P. Ashar, and S. Malik, “Technology mapping for low power,” *Proc. of Design Automation Conference*, pp. 74-79, June 1993
- Hide nodes with high switching activity inside the gates where they drive smaller load capacitances



- **De-glitching**

- Glitch consumes 10% - 40% of the dynamic power in typical combinational logic circuits



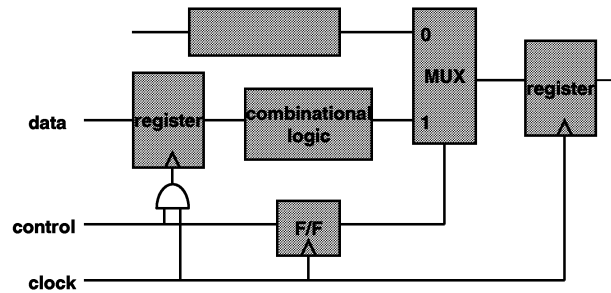
- **Path balancing**

- Add unit-delay buffers selectively such that the delays of all paths can be made equal

RTL Design

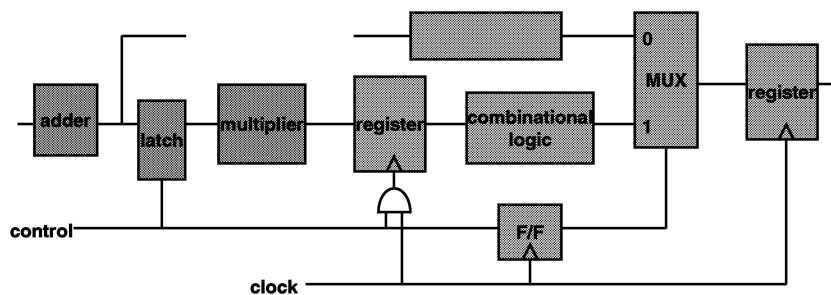
- **Clock gating**

- Disable clocks to idle part of the circuit
- Saves clock power and power consumed by registered value change

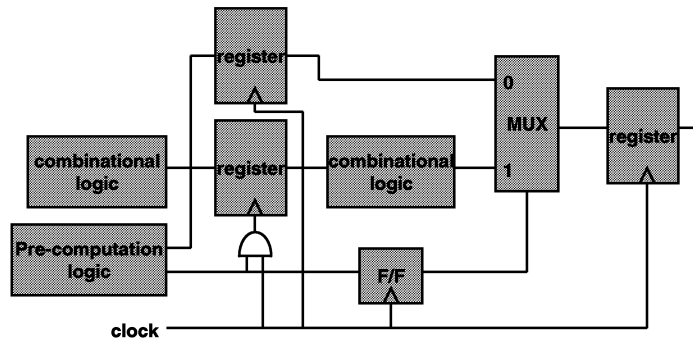


- **Operand isolation**

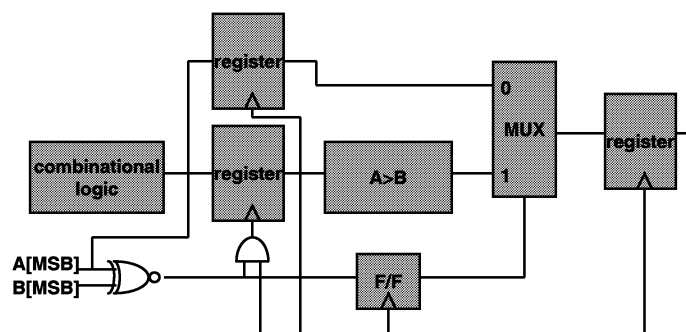
- Exploit output don't cares of large circuit blocks in unused clock cycles
- Insert latches before the circuit blocks to reduce circuit activity



- **Pre-computation**
 - Pre-compute the results of subsequent pipeline stages



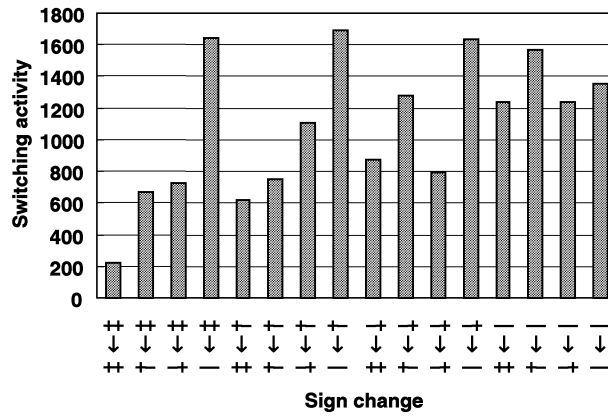
- **Comparator example**



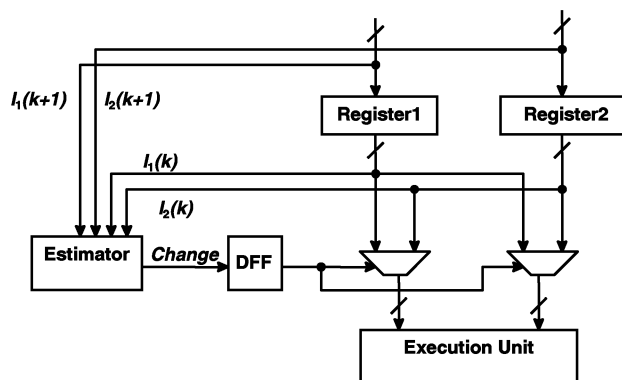
- **Dynamic operand interchange**

- T. Ahn and K. Choi, “Dynamic operand interchange for low power,” *Electronics Letters*, pp. 2118-2120, Dec. 1997

- **Switching activity of 16-bit array multiplier**



- **Architecture**



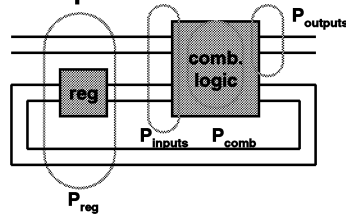
• **FSM encoding**

- C.-Y. Tsui, M. Pedram, C.-A. Chen, and A.M. Despain, "Low power state assignment targeting two- and multi-level logic implementations," *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 82-87, Nov. 1994
- Low power state encoding of FSM
- Reduce switching activity on state bit lines

- Cost function:
$$\sum_{S_i, S_j \in S} p_{ij} H(S_i, S_j)$$

where p_{ij} is the transition probability from state S_i to state S_j and $H(S_i$ to state $S_j)$ is the Hamming distance between the encodings of the two states

- Also reduce power consumed in the combinational logic



Architecture-Level Design

• **Supply voltage reduction**

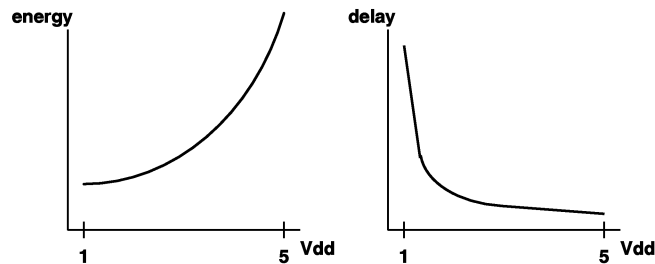
- Quadratic effect of voltage scaling on power

$$P_{dynamic} = C_{eff} V_{dd}^2 f_{clk} \quad E_{dynamic/cycle} = C_{eff} V_{dd}^2$$

5V --> 3.3V => 60% power reduction

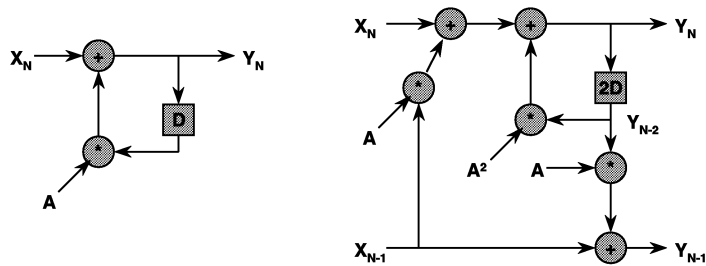
- Supply voltage reduction => increased latency

$$T_g = K_d \frac{V}{(V - V_{th})^\alpha}$$

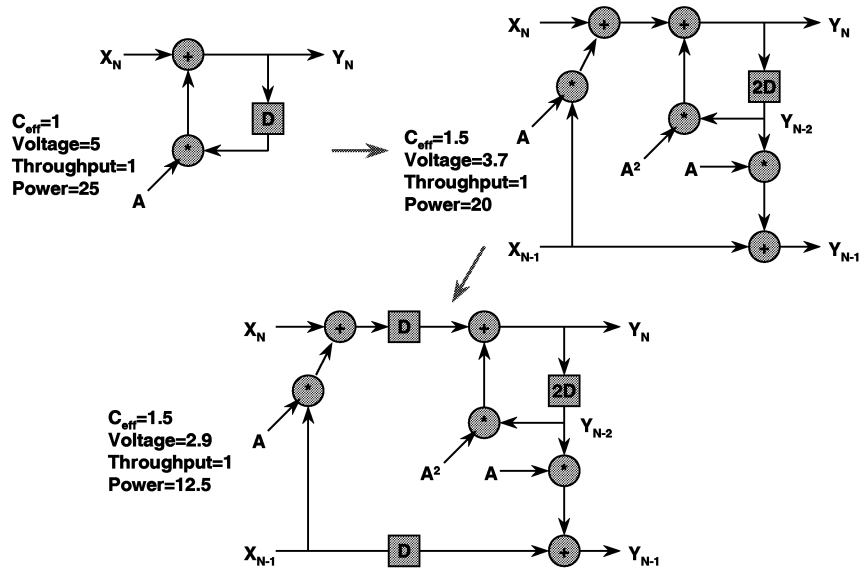


Architecture-Level Design

- Use of optimizing transformation for meeting throughput constraint even with the voltage reduction
- Concurrency increasing transformation (increased hardware cost) => critical path reduction
- Loop unrolling, pipelining, retiming, algebraic transformation, module selection
 - A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R.W. Brodersen, "Optimizing power using transformation," *IEEE Tr. on CAD/ICAS*, pp. 12-31, Jan. 1995
- $Y_N = AY_{N-1} + X_N \quad \rightarrow \quad Y_N = A^2Y_{N-2} + AX_{N-1} + X_N$
 $Y_{N-1} = AY_{N-2} + X_{N-1} \quad Y_{N-1} = AY_{N-2} + X_{N-1}$

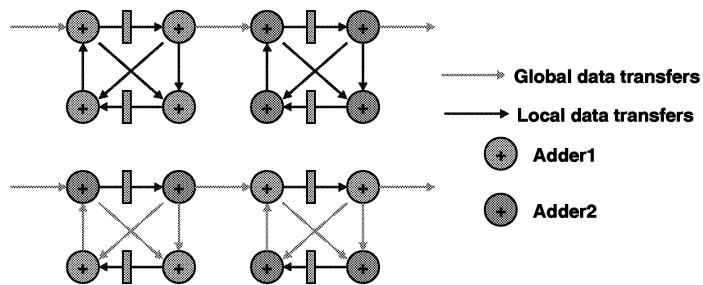


Architecture-Level Design



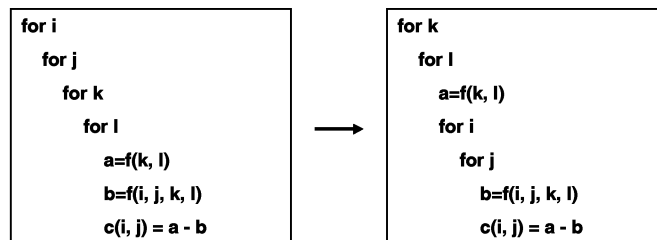
• **Reduction of effective capacitance**

- R. Mehra, L.M. Guerra, and J.M. Rabaey, “Low power architectural synthesis and the impact of exploiting locality,” *Journal of VLSI Signal Processing*, 1996
- Buses consume 5-40% of the total power
- Reducing access to global resource thru clustering



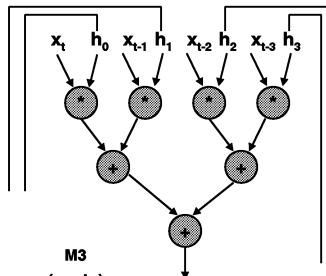
• **Switching activity reduction**

- Increasing data correlation thru operand sharing
 - Operations sharing an operand also share resource
 - Actively increase the chance of operand sharing thru loop interchange, operand reordering, loop unrolling, loop folding
- Loop interchange



Architecture-Level Design

- Operand reordering
 - 4th order LMS adaptive filter

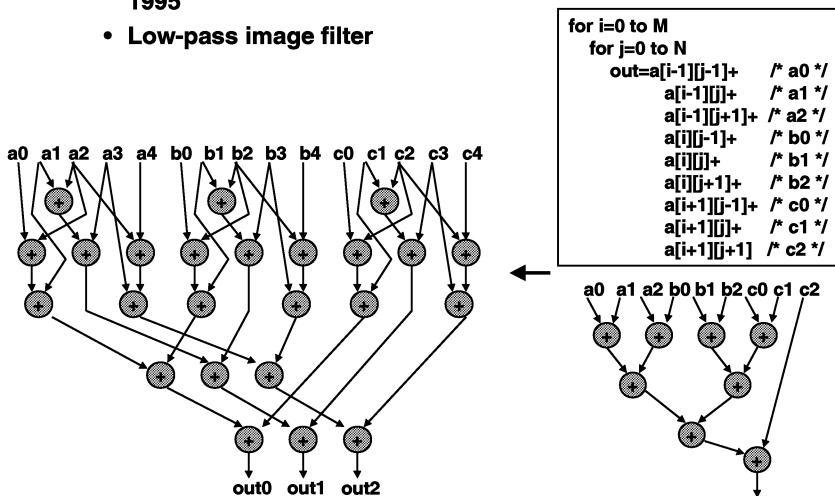


iter	Reordering A			
	M0	M1	M2	M3
i	(x_t, h_0)	(x_{t-1}, h_1)	(x_{t-2}, h_2)	(x_{t-3}, h_3)
i+1	(x_{t+1}, h_0)	(x_t, h_1)	(x_{t-1}, h_2)	(x_{t-2}, h_3)
i+2	(x_{t+2}, h_0)	(x_{t+1}, h_1)	(x_t, h_2)	(x_{t-1}, h_3)
i+3	(x_{t+3}, h_0)	(x_{t+2}, h_1)	(x_{t+1}, h_2)	(x_t, h_3)

iter	Reordering B			
	M0	M1	M2	M3
i	(x_t, h_0)	(x_{t-1}, h_1)	(x_{t-2}, h_2)	(x_{t-3}, h_3)
i+1	(x_t, h_1)	(x_{t-1}, h_2)	(x_{t-2}, h_3)	(x_{t+1}, h_0)
i+2	(x_t, h_2)	(x_{t-1}, h_3)	(x_{t+2}, h_0)	(x_{t+1}, h_1)
i+3	(x_t, h_3)	(x_{t+3}, h_0)	(x_{t+2}, h_1)	(x_{t+1}, h_2)

Architecture-Level Design

- Loop unrolling
 - E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," *Proc. of Int'l Symp. on Low Power Design*, pp. 99-104, Nov. 1995
 - Low-pass image filter

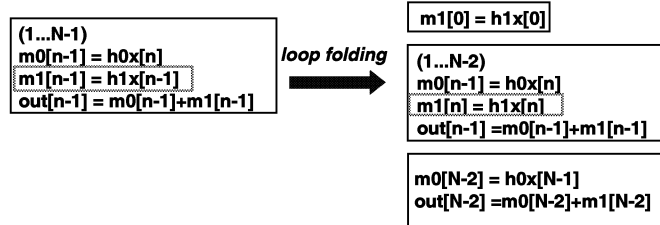


– Loop folding

- D. Kim and K. Choi, “Power-conscious high level synthesis using loop folding,” *Proc. of Design Automation Conference*, pp. 441-445, June 1997
- Fold two consecutive iterations in such a way that $h(i) * x[n-i]$ for $y[n]$ and $h(i+1) * x[(n+1)-(i+1)]$ for $y[n+1]$ are computed consecutively in one shared multiplier

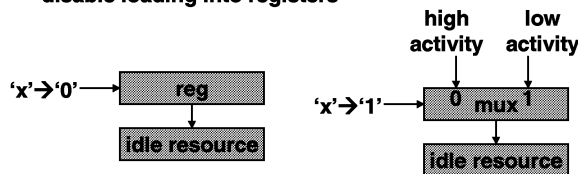
$$y[n] = \sum_i h_i x[n-i] \quad \Rightarrow \quad \text{Significant effects on DSP applications such as filters}$$

$$\begin{aligned} y[n] &= \square + h_i \times x[n-i] + \square \\ y[n+1] &= \square + h_{(i+1)} \times x[(n+1)-(i+1)] + \square \\ &\square \qquad \qquad \qquad \square \end{aligned}$$



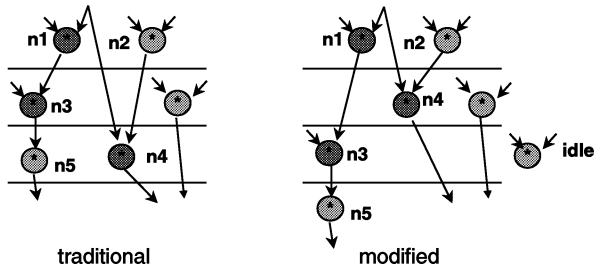
– Binding

- A. Raghunathan and N. K. Jha, “Behavioral synthesis for low power,” *Proc. of Int’l Conf. on Computer Design*, pp. 318-322, Oct. 1994
- Binding based on edge weighted compatibility graph
 - weight = $(1-W_t)W_c$
 - where W_t is transition activity (higher $1-W_t \rightarrow$ lower activity)
 - W_c is capacitance weight (higher $W_c \rightarrow$ lower capacitance)
- Functional unit and register sharing
- Controller optimization to reduce power consumed during idle time of functional units
 - use don’t cares
 - select the mux port with least transition activity
 - disable loading into registers



– Scheduling and binding

- E. Musoll and J. Cortadella, “Scheduling and resource binding for low power,” *Proc. of Int’l Symp. on System Synthesis*, pp. 104-109, Apr. 1995
- Resource sharing by sibling operations
- Operations sharing the same operand are scheduled in control steps as close as possible (higher priority is given for list scheduling)



- After functional unit binding, bind registers such that useless power is reduced (no change of inputs to idle functional unit)
- A few sibling operations available in normal circuits

– Scheduling and binding

- A. Raghunathan and N. K. Jha, “An iterative improvement algorithm for low power data path synthesis,” *Proc. of Int’l Conf. on Computer-Aided Design*, pp. 597-602, Nov. 1995
- Thorough power minimization including voltage scaling, clock selection, and module selection as well as scheduling and binding
- Iterative improvement
- Pruning for efficiency of the algorithm
 - supply voltage pruning:
 - prune V_{dd} if the lower bound of power at V_{dd} is greater than the best solution seen
 - clock period pruning:
 - Consider only T_{clk} such that $T_{clk} \times i = T_s$ for some integer i , and sampling period T_s
 - $T_{clk1} < T_{clk2}$ and $\lceil \text{delay}_t / T_{clk1} \rceil = \lceil \text{delay}_t / T_{clk2} \rceil$ for all functional unit template $t \Rightarrow$ prune T_{clk2}

```

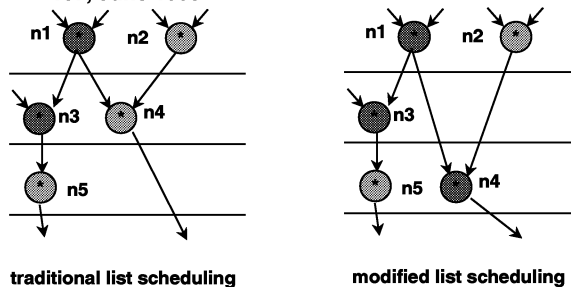
SCALP (CDFG G, Sample Period Ts, Library L) {
  Vmin=estimate_min_volt(G, Ts, L);
  Vmax=5V;
  best_dp=null;
  cur_dp=null;
  for(Vdd=Vmin; Vdd≤Vmax; Vdd=Vdd+ΔV) {
    if(Vdd_prune(G,cur_dp,Vdd)) continue;
    for(csteps=max_csteps; csteps≥min_csteps;
      csteps=csteps-1){
      if(clk_prune(G, L, csteps)) continue;
      cur_dp=initial_solution(G, L, Vdd, csteps);
      iterative_improvement(G, L, cur_dp);
      if(power_est(cur_dp) < power_est(best_dp))
        best_dp=cur_dp;
    }
  }
}

iterative_improvement(G, L, DP) {
  do {
    for(i=1; i ≤ max_moves; i=i+1) {
      gaini = generate_moves(G, L, DP);
      append gaini to gain_list;
    }
    find subsequence, gain1 ... gaink in
      gain_list so that G=Σgaini is maximized;
    if(G>0) {
      accept moves 1...k;
    }
  }
  until(G<0);
}

```

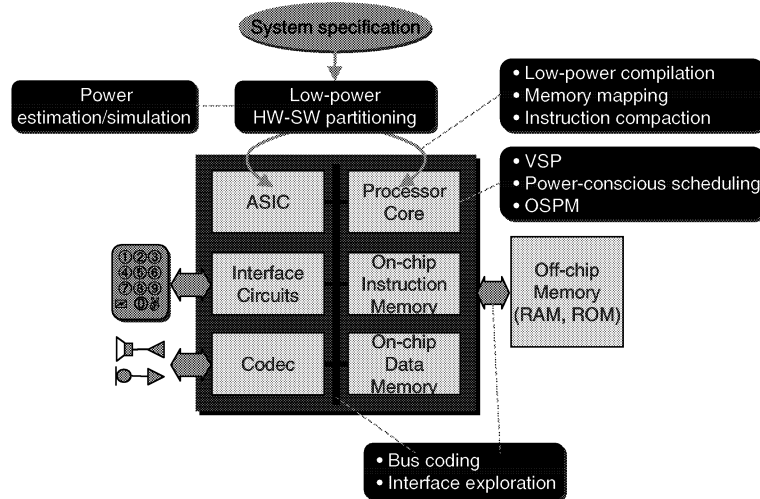
– Scheduling and binding

- D. Shin and K. Choi, “Low power high level synthesis by increasing data correlation,” *Proc. of Int’l Symp. on Low Power Electronics and Design*, pp. 62-67, Aug. 1997
- Simultaneous scheduling and binding in such a way that input data correlation between consecutive inputs increase
- (Modified) list scheduling is used for efficiency
- DBT (Dual Bit Type) method for estimating switched capacitance in execution units
 - P.E. Landman and J.M. Rabaey, “Architectural power analysis: the dual bit type method,” *IEEE Tr. on VLSI Systems*, pp. 173-187, June 1995



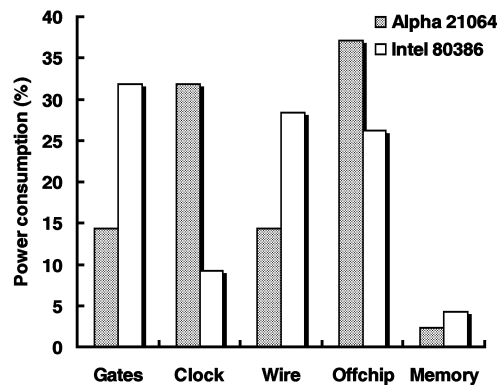
System-Level Design

• System-level power optimization

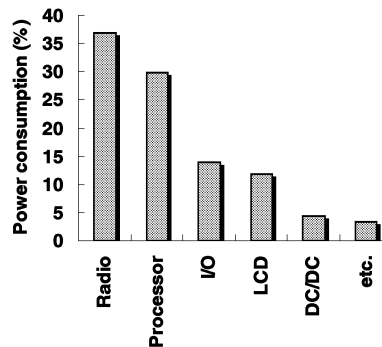


• Power consumption in processors

- Buses consume significant power
 - Capacitive load at I/O of a chip is three orders of magnitude larger than that of internal nodes
- Example
 - D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE JSSC*, pp. 663-670, June 1994



- **Power consumption in portable embedded systems**
 - Power consumption in processors becomes more significant as increasing amount of functionality is realized through software
 - Example
 - T. Truman, T. Pering, R. Doering, and R. Brodersen, “The InfoPad multimedia terminal: a portable device for wireless information access,” *IEEE Transactions on Computers*, pp. 1073-1087, October 1998



- **Low power design issues**
 - L. Benini and G. De Micheli, “System-level power optimization: techniques and tools,” *Proc. of Int’l Symp. on Low Power Electronics and Design*, pp. 288-293, Aug. 1999
 - Memory optimization
 - Memory hierarchy, cache size, memory size (related with software transformation), data transfer and placement
 - E.g. large cache size → low cache miss → high speed and low power, but large capacitance
 - Hardware-software partitioning
 - Power consumption in hardware, software, and interface
 - Instruction-level power optimization
 - Dedicated low-power instruction set, instruction transformation,
 - Variable-voltage
 - Dynamically variable voltage supply
 - Effective
 - Dynamic power management
 - Low-power sleep state
 - Predictive, stochastic
 - Standard (OnNow, ACPI)
 - Interface power minimization
 - Bus encoding

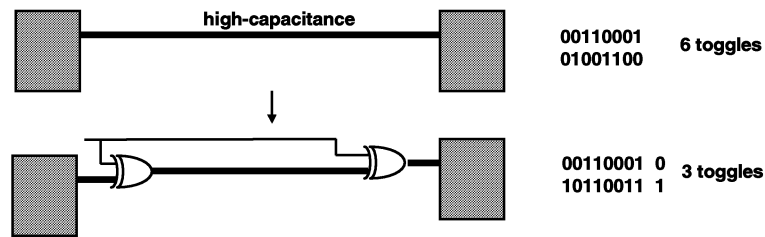
Bus Encoding

- Reduce number of transitions on high-capacitance, multi-bit buses by encoding the signals

- Example

- Bus-invert coding

- M.R. Stan, W.P. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Trans. on VLSI Systems*, Vol. 3, No. 1, pp. 49-58, Mar. 1995



Narrow Bus Encoding

- Y. Shin and K. Choi, “Narrow bus encoding for low power systems,” *Proc. of Asia South Pacific Design Automation Conf.*, pp. 217-220, Jan. 2000

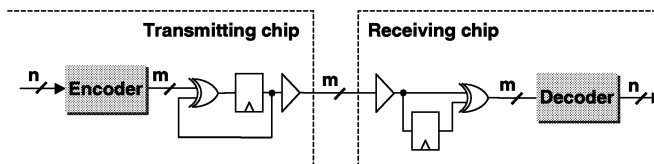
- Previous work

- Bus-invert coding

- Appropriate for uncorrelated patterns
 - Redundant encoding

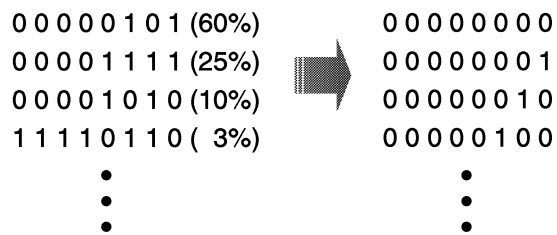
- Transition signaling

- Logic 1 → transition, logic 0 → no transition (or vice versa)
 - Efficient when the signal probability is biased
 - Reduce number of 1's → reduce transitions



– Probability-based mapping

- S. Ramprasad, N.R. Shanbhag, and N. Hajj, “A coding framework for low-power address and data busses,” *IEEE Tr. on VLSI Systems*, Vol. 7, No. 2, pp. 212-221, June 1999
- Appropriate for coding PCM signals
- Very high complexity of coding logic
- Irredundant encoding



• Motivation

– Core-based design

- A lot of components integrated into a chip
- QUALCOMM MSM3000: ARM7TDMI, DSP, CDMA processor, DFM processor, several peripheral interfaces



Excessive number of pins

Narrow bus

ARM7TDMI (32-bit microprocessor core)

→ support different data transfer size
(Byte, Halfword, and Word)

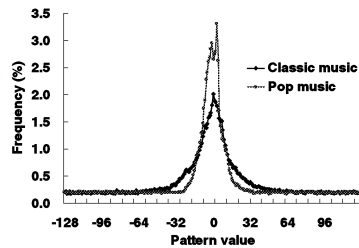
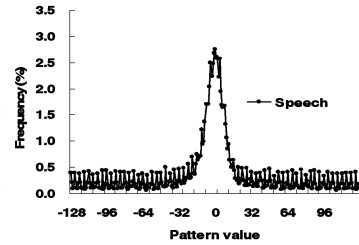


How to encode ?

Narrow Bus Encoding

- Mapping for narrow bus
 - Low-power: reduced transition
 - Irredundant: one-to-one mapping
 - Low-overhead: simple logic

0111	0111	1111	1000
0110	0110	1110	1001
0101	0101	1101	1010
0100	0100	1100	1011
0011	0011	1011	1100
0010	0010	1010	1101
0001	0001	1001	1110
0000	0000	1000	1111
Half Identity		Half Reverse	



Narrow Bus Encoding

- How to obtain hibr mapping
 - If MSB=1, invert lower (n-1) bits
 - else, identity

0111	0111	1111	1000
0110	0110	1110	1001
0101	0101	1101	1010
0100	0100	1100	1011
0011	0011	1011	1100
0010	0010	1010	1101
0001	0001	1001	1110
0000	0000	1000	1111

Proposed coding method
hibr mapping + transition signaling

Narrow Bus Encoding

00000101 → 00000101
10101111
00001010
10010100
11111010
01100000
00000001
00000100
00000011
01111100
37 transitions

Narrow Bus Encoding

00000101 → 00000101
10101111 → 11010101
00001010
10010100
11111010
01100000
00000001
00000100
00000011
01111100
37 transitions

Narrow Bus Encoding

00000101		00000101
10101111		11010101
00001010	→	01011111
10010100		
11111010		
01100000		
00000001		
00000100		
00000011		
01111100		

37 transitions

Narrow Bus Encoding

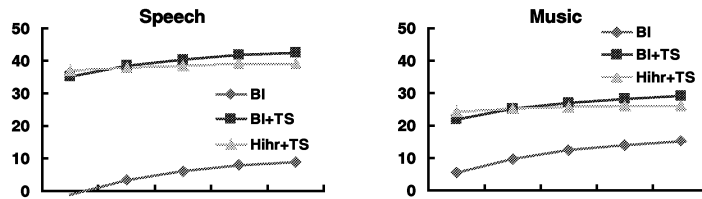
00000101		00000101
10101111		11010101
00001010		01011111
10010100		10110100
11111010	→	10110001
01100000		01010001
00000001		01010000
00000100		01010100
00000011		01010111
01111100		00101011

37 transitions *26 transitions*

- **Experimental Results**
 - **Data sets**
 - 2 sets from speech samples: 16 bit
 - 2 sets from music signals: 16-bit
 - 2 sets from FFT processor: 20-bit
 - **Width of narrow bus**
 - Half of original width
 - **Number of transitions**
 - Hihr + TS Savings = 35.7%
 - pbm Savings = 39.5%
 - BI + TS Savings = 42.8%
 - BI Savings = 17.9%

- **Coding logic**
 - Synthesized and mapped onto TSMC 0.35 μ m, 3.3 V gate library
 - Assume 100 MHz clock
- **Encoding logic**
 - Hihr + TS Area= 4659 μ m², Delay=0.38ns, Power= 411 μ W
 - BI + TS Area=18626 μ m², Delay=3.87ns, Power=2409 μ W
 - BI Area=19076 μ m², Delay=3.29ns, Power=2309 μ W
- **Decoding logic**
 - Hihr + TS Area= 9968 μ m², Delay=0.38ns, Power=1618 μ W
 - BI + TS Area=11392 μ m², Delay=0.38ns, Power=2102 μ W
 - BI Area= 2662 μ m², Delay=0.15ns, Power= 120 μ W

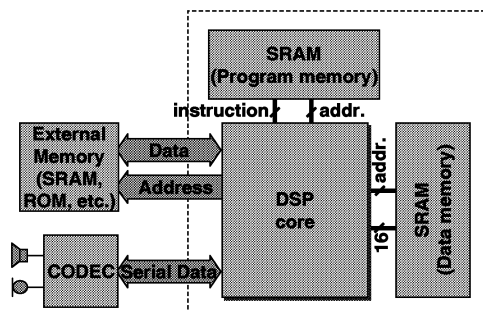
– Overall power savings during off-chip driving



• Case study

– Digital hearing aid (DHA) system

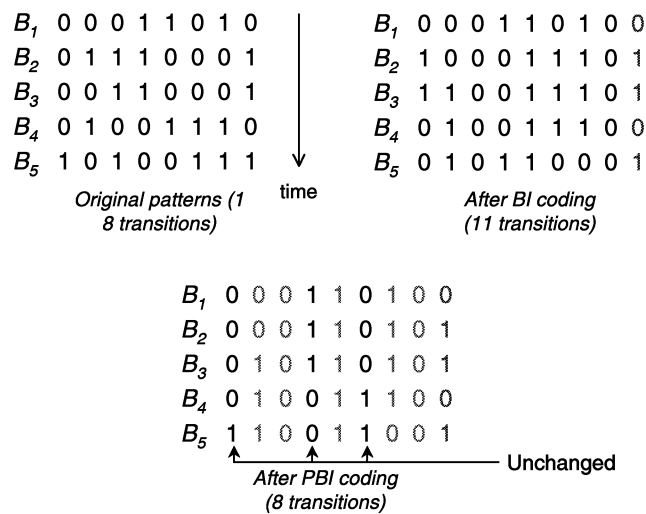
- DSP core supports special instructions for 8-bit wide narrow bus communication
- Savings = 37.5% (10 pF total off-chip capacitance)
~ 40.8% (30 pF total off-chip capacitance)



Partial Bus-Invert Coding

- Y. Shin, S. Chae, and K. Choi, “Partial bus-invert coding for power optimization of system level bus,” *Proc. of Int’l Symp. on Low Power Electronics and Design*, pp. 127-129, Aug. 1998
- **Address patterns**
 - Instruction address: mostly sequential
 - Data address: less sequential than instruction address and less random than data
- **Partial Bus-Invert (PBI) coding**
 - Apply Bus-Invert (BI) coding to a sub-set of bus lines
 - Reduced overhead of coding logic
 - Efficient for data address busses of special-purpose applications

• **Motivation**



- **PBI coding**
 - **Partition a bus into two sub-buses S, R**
 - **S: involved in BI coding**
 - **R: unchanged at all times**
 - **Process of PBI coding**
 - **Compute Hamming distance between S_{i-1}' and S_i**
 - **If it is larger than $|S|/2$, set invert=1 and invert the lines in S_i without inverting the lines in R_i**
 - **Otherwise, set invert=0 and let entire bus lines unchanged**

S_i : value of **S** at time i

S_i' : encoded version of S_i

- **Sub-bus selection algorithm**
 - **Problem**
 - **Given a set of patterns**
 - **Partition a bus into a configuration (S, R) in a way that transitions are minimized**
 - **Optimum set S_{opt}**
 - **Infeasible to find because there are 2^n possible configurations**
 - **Observation**
 - **BI coding is effective when most bus lines make transitions together**
 - **Heuristic**
 - **Exploit transition probability and transition correlation**
 - **Select a seed line with highest transition probability**
 - **Then perform clustering bus lines**

Partial Bus-Invert Coding

B_1 0 0 0 1 1 0 1 0 0	B_1 0 0 0 1 1 0 1 0 0	B_1 0 0 0 1 1 0 1 0 0
B_2 0 1 1 1 0 0 0 1 0	B_2 0 0 0 1 0 0 0 1 1	B_2 0 0 0 1 1 0 0 1 1
B_3 0 0 1 1 0 0 0 1 0	B_3 0 1 0 1 0 0 0 1 1	B_3 0 1 0 1 1 0 0 1 1
B_4 0 1 0 0 1 1 1 0 0	B_4 0 1 0 0 1 1 1 0 0	B_4 0 1 0 0 1 1 1 0 0
B_5 1 0 1 0 0 1 1 1 0	B_5 1 1 0 0 0 1 1 1 1	B_5 1 1 0 0 1 1 1 1 1
Config. 1 (18 transitions)	Config. 2 (15 transitions)	Config. 3 (12 transitions)
B_1 0 0 0 1 1 0 1 0 0	B_1 0 0 0 1 1 0 1 0 0	B_1 0 0 0 1 1 0 1 0 0
B_2 0 0 0 1 1 0 0 0 1	B_2 0 0 0 1 1 0 1 0 1	B_2 0 0 0 0 1 0 1 0 1
B_3 0 1 0 1 1 0 0 0 1	B_3 0 1 0 1 1 0 1 0 1	B_3 0 1 0 0 1 0 1 0 1
B_4 0 1 0 0 1 1 1 0 0	B_4 0 1 0 0 1 1 1 0 0	B_4 0 1 0 0 1 1 1 0 0
B_5 1 1 0 0 1 1 1 0 1	B_5 1 1 0 0 1 1 0 0 1	B_5 1 1 0 1 1 1 0 0 1
Config. 4 (9 transitions)	Config. 5 (8 transitions)	Config. 6 (9 transitions)
B_1 0 0 0 1 1 0 1 0 0	B_1 0 0 0 1 1 0 1 0 0	
B_2 0 0 0 0 1 1 1 0 1	B_2 1 0 0 0 1 1 1 0 1	
B_3 0 1 0 0 1 1 1 0 1	B_3 1 1 0 0 1 1 1 0 1	
B_4 0 1 0 0 1 1 1 0 0	B_4 0 1 0 0 1 1 1 0 0	
B_5 1 0 1 0 0 1 1 1 0	B_5 0 1 0 1 1 0 0 0 1	
Config. 7 (10 transitions)	Config. 8 (11 transitions)	

Partial Bus-Invert Coding

- **Overhead of encoding/decoding circuits**
 - XOR gates + majority voter
 - Models with C_{int} (average load capacitance of internal nodes) and T_{int} (total number of transitions in the encoding/decoding circuits)
- **Estimation of T_{int}**
 - Approximate to transitions in the majority voter
 - $T_{int} = \kappa N(m+1) a_p L$
 - κ : gate equivalents of a FA
 - $N(x)$: approximate number of FAs in the majority voter and equals to $x-2$
 - a_p : average transition probability of m bus lines
 - L : the number of patterns

$$\text{Total effective bus transitions } T_{eff} = T_{bus} + C_{int}/C_{bus} T_{int}$$

• **Experimental Results**

- **Data sets**
 - **Set 1: Compress, Laplace, Linear, Lowpass, SOR, Wavelt**
 - **Set 2: Parser, FFT (patterns from MPEG-2 audio and AC3-Decoder)**
- **Total bus transitions**
 - **PBI encoding**
 - **Set 1: savings = 62.6% (62.5% lines encoded)**
 - **Set 2: savings = 48.5% (36.2% lines encoded)**
 - **BI encoding**
 - **Set 1: savings = 38.1%**
 - **Set 2: savings = 1.4%**
- **Total effective bus transitions (including encoder/decoder overhead)**
 - **PBI encoding**
 - **Set 1: savings = 59.1% (56.3% lines encoded)**
 - **Set 2: savings = 45.5% (36.2% lines encoded)**

Dynamic Voltage Scaling

Dynamic power dissipation

$$P_{dynamic} = C_{eff} V_{dd}^2 f_{clk}$$

Energy per cycle

$$E_{per_cycle} = C_{eff} V_{dd}^2$$

Energy consumed by a task that takes n cycles

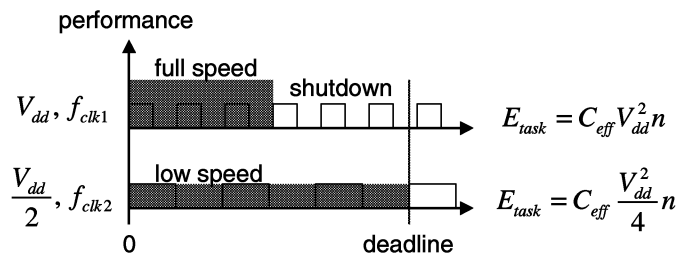
$$E_{task} = C_{eff} V_{dd}^2 n$$

→ not a function of time but a function of # cycles (switchings)

Gate delay by α power model

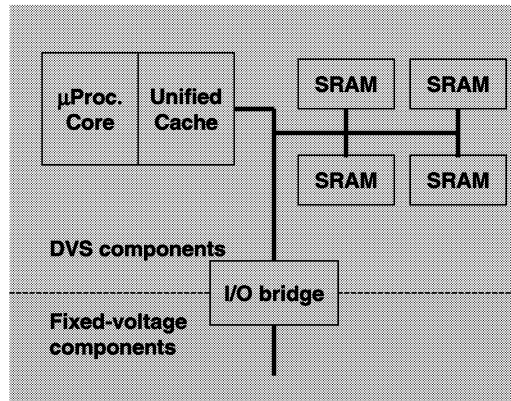
$$T_g = K_d \frac{V}{(V - V_{th})^\alpha}$$

$$f_{clk} = K_f \frac{(V - V_{th})^\alpha}{V}$$

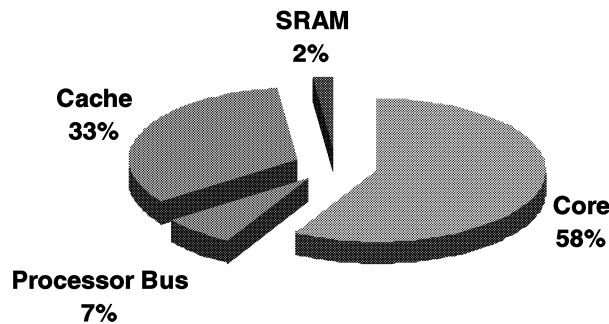


• **DVS on a Microprocessor System**

- T. Pering, T., and R. Brodersen, "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System," in *Power Driven Microarchitecture Workshop* in conjunction with ISCA98, June 1998
- System block diagram (ARM8 architecture)



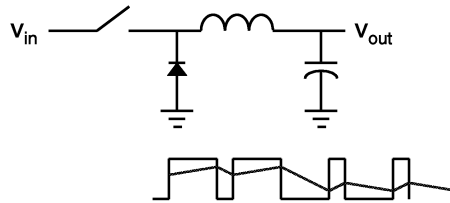
- **System energy breakdown**



Benchmark	Miss Rate	Idle Time	Bus Activity
AUDIO	0.23%	67%	0.35%
MPEG	1.7%	22%	14%
UI	0.62%	95%	0.52%

– DC-DC Converter

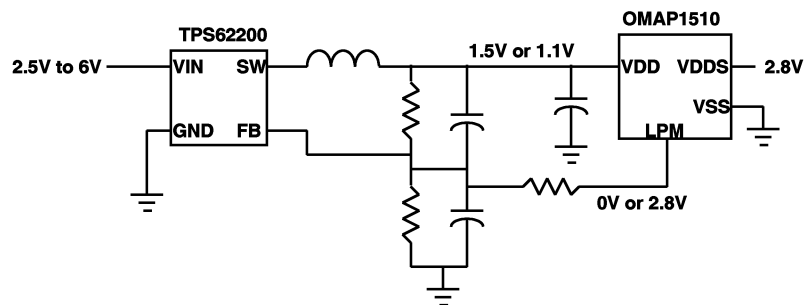
- Buck converter



- 0.6 μ m, 3-metal, 1-poly CMOS \rightarrow die size:1.6mmx3.4mm
- 1.0V-4.0V dynamically adjustable output
- Efficiency: 88% at 1.1V, 5mA, 5MHz
95% at 3.3V, 200mA, 100MHz
- 1.1V \rightarrow 3.3V transition time=20ms
- 1.1V \rightarrow 3.3V \rightarrow 1.1V transition energy=20mJ

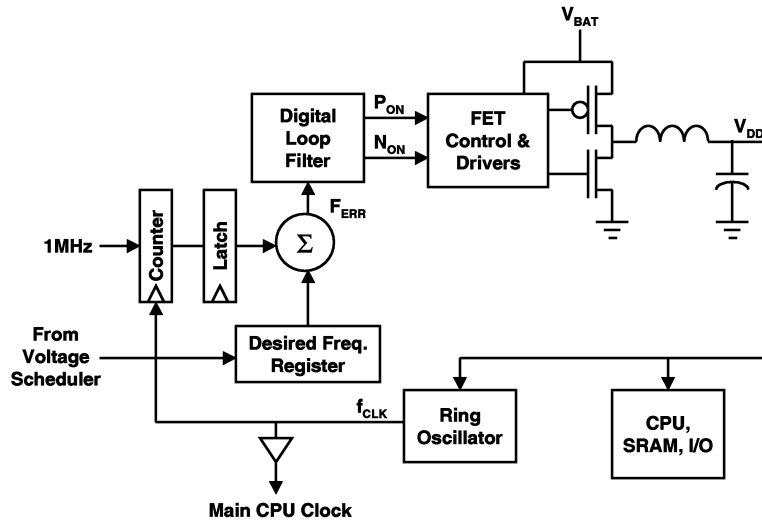
– Commercial DC-DC Converters

- Intersil's ISL6211
 - For LongRun power management of Transmeta's Crusoe processor
 - 0.6V-1.0V in 25mV steps and 1.0V-1.75V in 50mV steps
 - With 5V input and 1.3V output, ~90% efficiency depending on the output current
- TI's TPS62200
 - With 3.6V input and 1.5V, 1.1V output, >90% efficiency



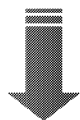
– Frequency to voltage feedback loop

- T. Burd, T. Pering, A. Stratakos, and R Brodersen, “A dynamic voltage scaled microprocessor system,” IEEE International Solid-State Circuits Conference, 2000



Real-Time Scheduling on a VSP

- Y. Shin and K. Choi, “Power conscious fixed priority scheduling for hard real-time systems,” *Proc. of Design Automation Conf.*, pp. 134-139, June 1999
- Two methods for power reduction in processors
 - Power-down mode
 - VSP (Variable Speed Processor)



How to exploit these features ?
→ Scheduling

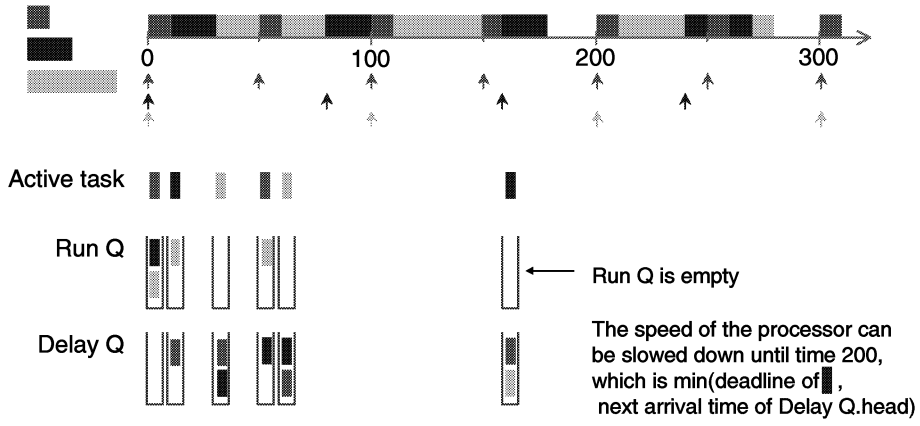
– Proposed method:

- Combine the two methods to obtain power saving for real-time systems
- Exploit execution time variation and idle interval

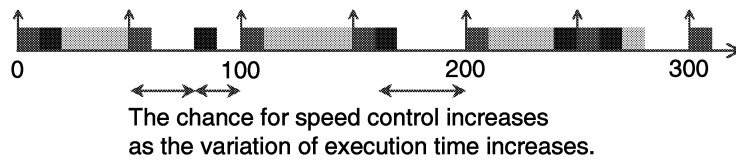
- **Related work**
 - **Power-down mode**
 - **Conventional method: power-down after predefined idle period**
 - **Prediction-based methods**
 - M. Srivastava, A. Chandrakasan, and R. Brodersen, “Predictive system shutdown and other architectural techniques for energy efficient programmable computation,” *IEEE Trans. on VLSI Systems*, Vol. 4, No. 1, pp. 42-55, Mar. 1996
 - C.-H. Hwang and A. Wu, “A predictive system shutdown method for energy saving of event-driven computation,” *Proc. of Int’l Conf. on Computer-Aided Design*, pp. 28-82, Nov. 1997
 - **Dynamic voltage scaling on a VSP**
 - **Analysis of dynamic voltage scaling schemes**
 - T. Pering, T. Burd, and R. Brodersen, “The simulation and evaluation of dynamic voltage scaling algorithms,” *Proc. of Int’l Symp. on Low Power Electronics and Design*, pp. 76-81, Aug. 1998
 - **Adaptive scaling of the supply voltage in self-timed circuits**
 - L.S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, “Low-power operation using self-timed circuits and adaptive scaling of the supply voltage,” *IEEE Tr. on VLSI Systems*, Vol. 2, No. 4, pp. 391-397, Dec. 1994
 - **Scheduling of real-time tasks**
 - T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” *Proc. of Int’l Symp. on Low Power Electronics and Design*, pp. 197-202, Aug. 1998

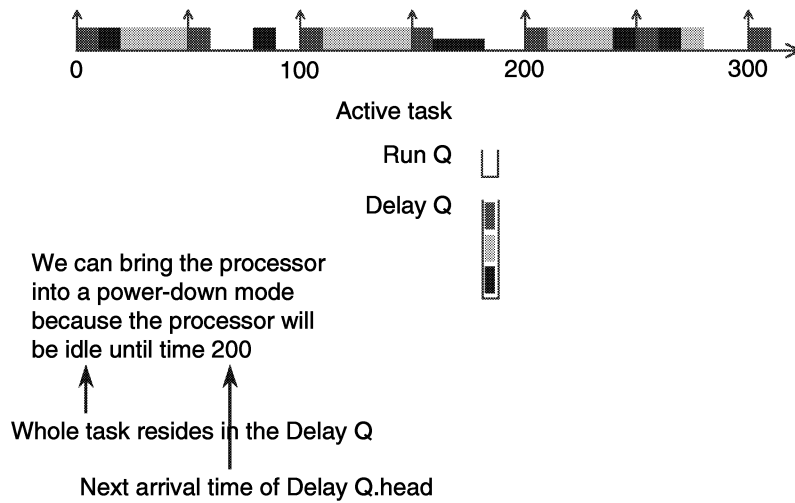
- **Priority-based preemptive scheduling**
 - Simple to implement
 - Many analytical methods for schedulability analysis
 - Fixed (static) priority (RMS, DMS) → LPFPS (Low Power Fixed Priority Scheduling)
 - Dynamic priority → LPEDF
- **Implementation of priority-based preemptive scheduling**
 - Active task, Run Q, Delay Q

Real-Time Scheduling on a VSP



Real-Time Scheduling on a VSP



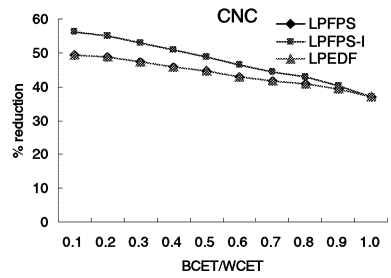
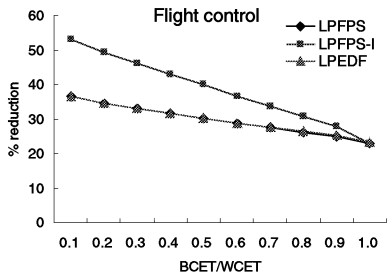
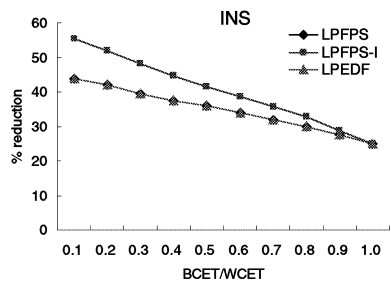
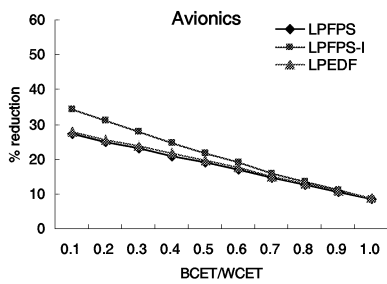
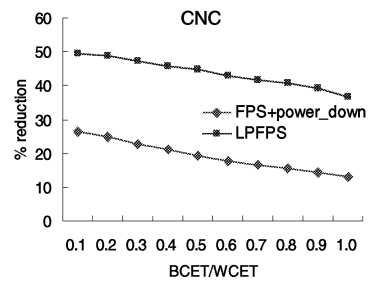
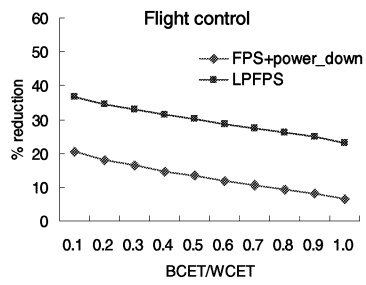
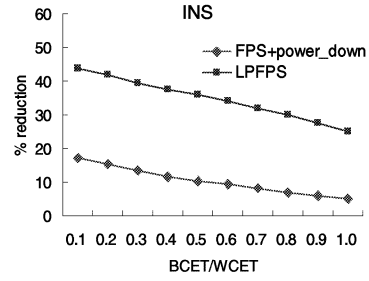
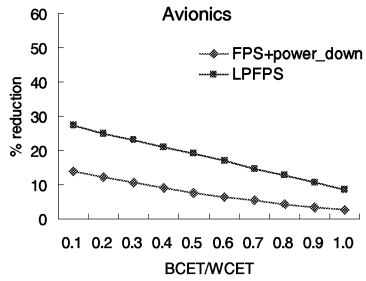


- **Summary of LPFPS**
 - Works when the Run Q is empty
 - If there is an Active task: speed control
 - Else: power-down
- **Extension of Basic LPFPS (LPFPS-n)**
 - Works when $|\text{Run Q}| = 0, 1, \dots, n$ ($n \leq \# \text{ tasks}$)
 - If there is an Active task: speed control
 - Else: power-down

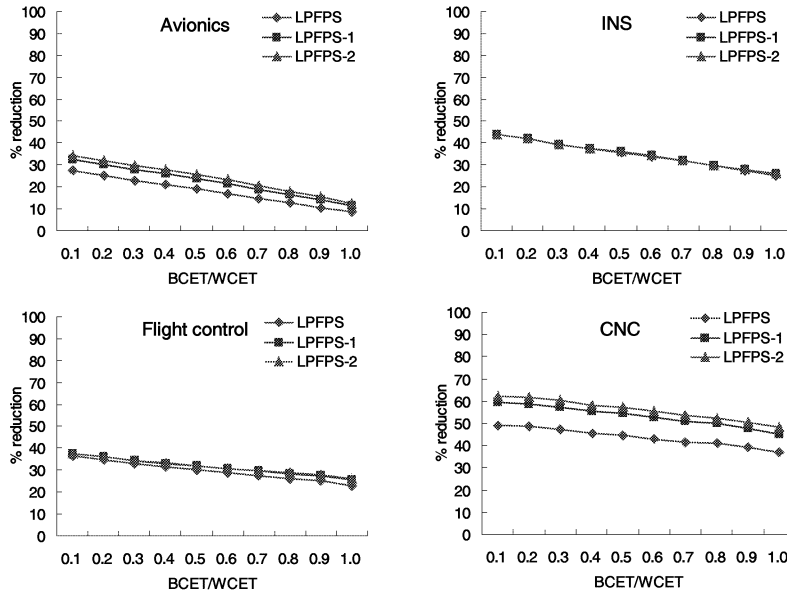
- **Experimental results**
 - **Compare average power consumption of VSP**
 - FPS: NOPs are assumed for idle time
 - LPFPS: VSP + power-down
 - LPFPS-I: VSP + power-down + knowledge of future execution time → upper bound of power saving
 - LPEDF: VSP + power-down
 - **Applications**
 - Avionics, INS, Flight control, CNC
 - Timing parameters: period, deadline, WCET

- **VSP**
 - NOP: 20% power consumption compared to typical instructions
 - Power-down mode: 5% power consumption of fully active mode with 10 cycles delay
 - Frequency: 100 MHz to 8 MHz with 1 MHz step
 - Voltage: 3.3 V to 1.1 V
- **Experimental procedure**
 - Control BCET: $0.1 \cdot \text{WCET} \sim 1.0 \cdot \text{WCET}$
 - Execution time: random variable following Normal distribution with $m = (\text{BCET} + \text{WCET})/2$, $\sigma = (\text{WCET} - \text{BCET})/6$
 - Run 3 times for each method and take average

• Experimental results

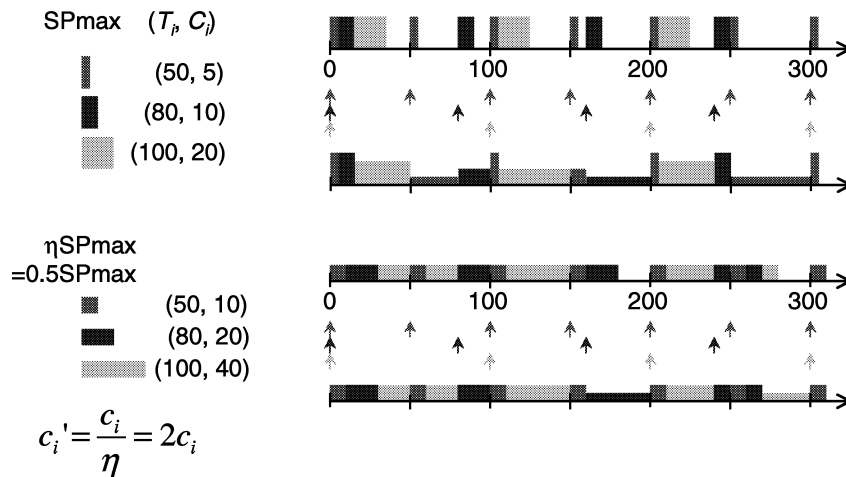


Real-Time Scheduling on a VSP



Real-Time Scheduling on a VSP

- **Computation of Maximum Speed**
 - Y. Shin, K. Choi, and T. Sakurai “Power optimization of real-time embedded systems on variable speed processors,” *Proc. ICCAD, 2000*



– **How to compute η**

- **Schedulability analysis is needed only at scheduling points**

- J. Lehoczky, L. Sha, and Y. Ding, “The rate monotonic scheduling algorithm: exact characterization and average case behavior,” *Proc. IEEE Real-Time Systems Symposium*, Dec. 1989.

$$S_i = \left\{ kT_j \mid j = 1, \dots, i; k = 1, \dots, \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\}$$

$$= \{S_{i,1}, S_{i,2}, S_{i,j}, \dots\}$$

- Task τ_i is schedulable if the following inequality is satisfied

$$\exists t \in S_i \text{ such that } \sum_{j=1}^i C_j \left\lfloor \frac{t}{T_j} \right\rfloor \leq t$$

- **At reduced speed,**

$$S_{i,j} \in S_i \text{ such that } \sum_{k=1}^i \frac{C_k}{\eta_i} \left\lfloor \frac{S_{i,j}}{T_k} \right\rfloor \leq S_{i,j}$$

$$\eta_i \geq \frac{\sum_{k=1}^i C_k \left\lfloor \frac{S_{i,j}}{T_k} \right\rfloor}{S_{i,j}}$$

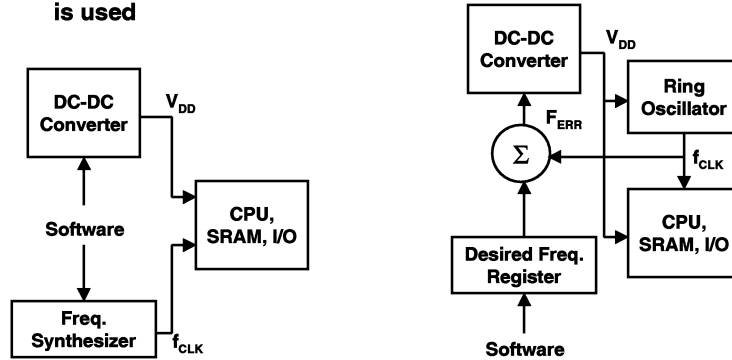
$$\eta_i = \min_j \frac{\sum_{k=1}^i C_k \left\lfloor \frac{S_{i,j}}{T_k} \right\rfloor}{S_{i,j}}$$

$$\eta = \max_i \eta_i$$

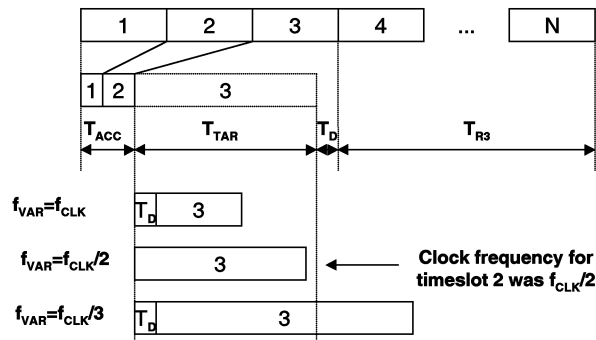
Intra-Task Dynamic Voltage Scaling

- S. Lee, T. Sakurai, "Run-time voltage hopping for low-power real-time systems," *Proc. DAC, 2000*

- DVS system architecture
 - Generate f_{CLK} , $f_{CLK}/2$, $f_{CLK}/3$, ... to avoid interface problem
 - Voltage-frequency Lookup table is used

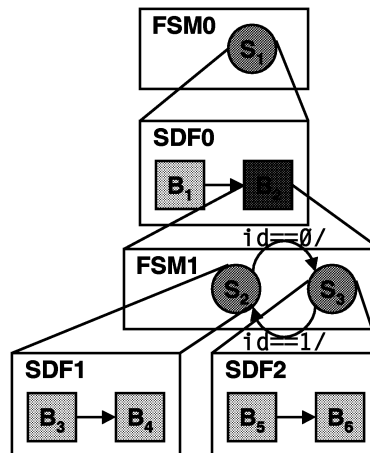


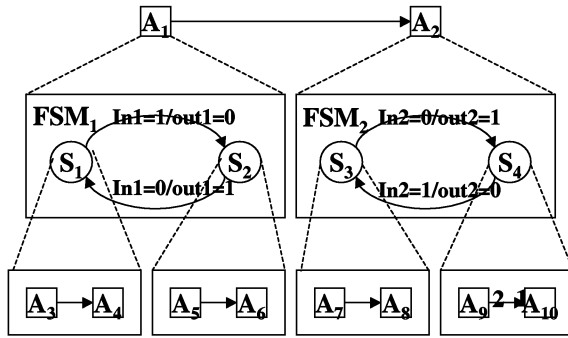
- Run-time voltage hopping
 - A task is divided into N timeslots.
 - For each timeslot, calculate $T_{TAR} = T_{WC} - T_{RI} - T_{ACC} - T_D$
 - Select best f_{VAR}
 - Obtain V_{VAR} from the lookup table



- **S. Lee, S. Yoo, and K. Choi, “An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model,” *Proc. ISLPED*, 2002**
 - To cope with productivity issues with ever growing complexity of SoC design,
 - Adopt design methodologies based on formal models of computation to enable shorter design cycle by formal analysis.
 - E.g. CFSM in Cadence VCC, hierarchical FSM with dataflow in Synopsys Cocentric System Studio, etc.
 - To reduce power consumption
 - Use dynamic voltage scaling (DVS) technique
 - Lower voltage/frequency when slack occurs
 - The effectiveness of DVS depends on the accuracy of slack estimation.

- **Hierarchical FSM**
 - {Input, output, states, transitions}
 - Hierarchical nesting of FSM, SDF
 - Different from State-charts





(a) Example HFSM/SDF model

```

1  if(A1.state==S1) {
2    A3();
3    A4(); st1();
4  }
5  else {
6    A5();
7    A6(); st2();
8  }
9  if(A2.state==S3) {
10   A7();
11   A8(); st3();
12 }
13 else {
14   A9();
15   A10(); A10(); st4();
16 }

```

Prepare Next Transition

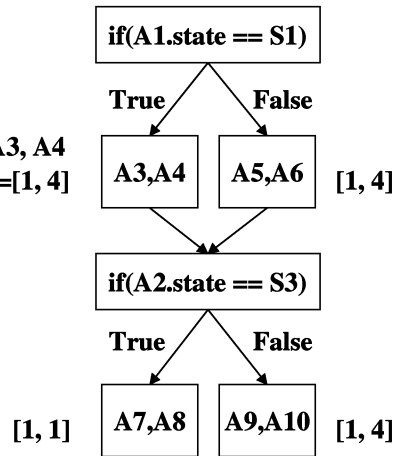
(b) Generated Code Example

```

1  if(A1.state==S1) {
2    A3();
3    A4(); st1();
4  }
5  else {
6    A5();
7    A6(); st2();
8  }
9  if(A2.state==S3) {
10   A7();
11   A8(); st3();
12 }
13 else {
14   A9();
15   A10(); A10(); st4();
16 }

```

WCET of A3, A4 = [1, 4]

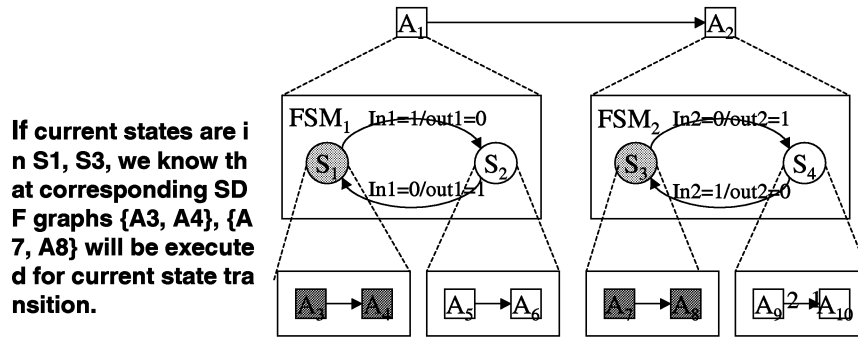


(a) Generated Code Example

(b) Generated Code Structure

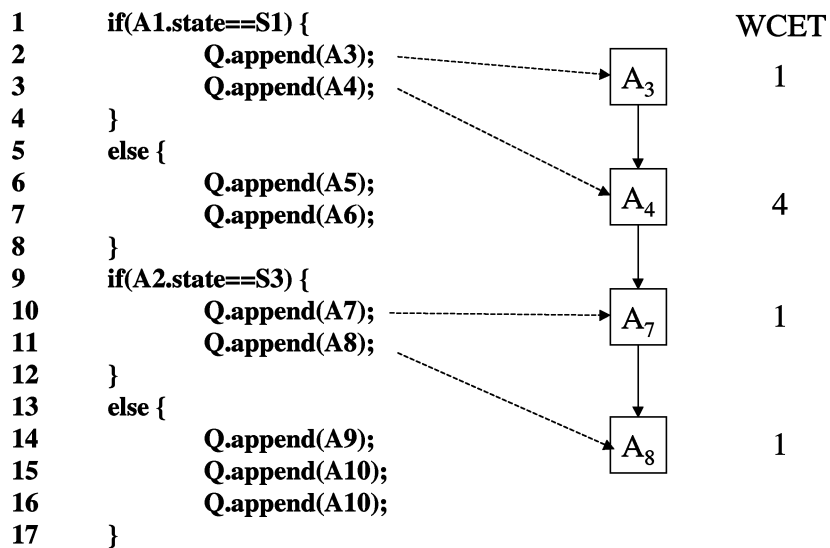
– HFSM/SDF property

- Exact execution paths are identifiable from the HFSM/SDF operational rules, by exploring current states.

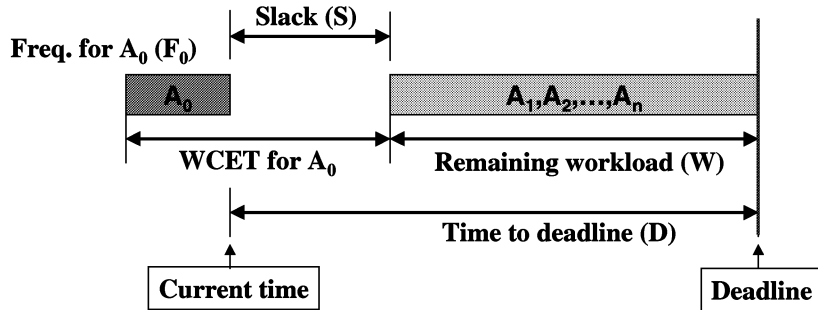


– Runtime Path Identification

- Code Example



Intra-Task Dynamic Voltage Scaling

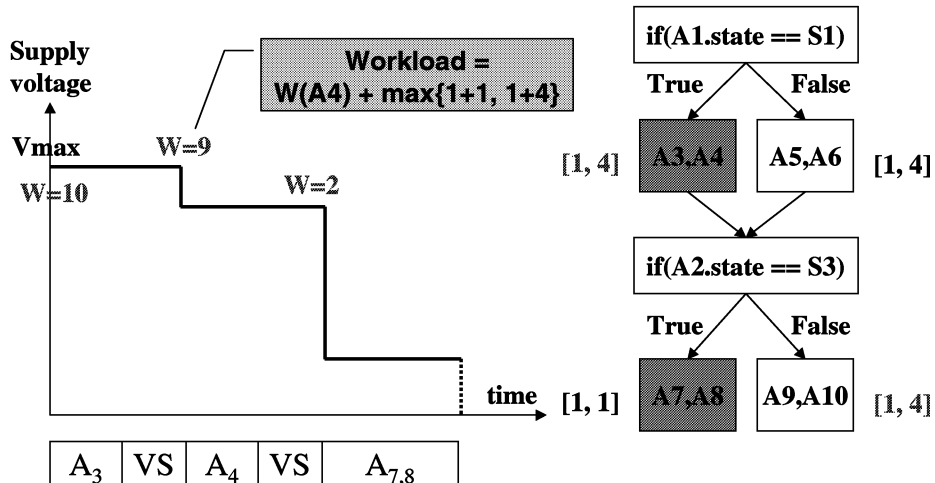


Speed ratio : $R = \frac{W}{D}$

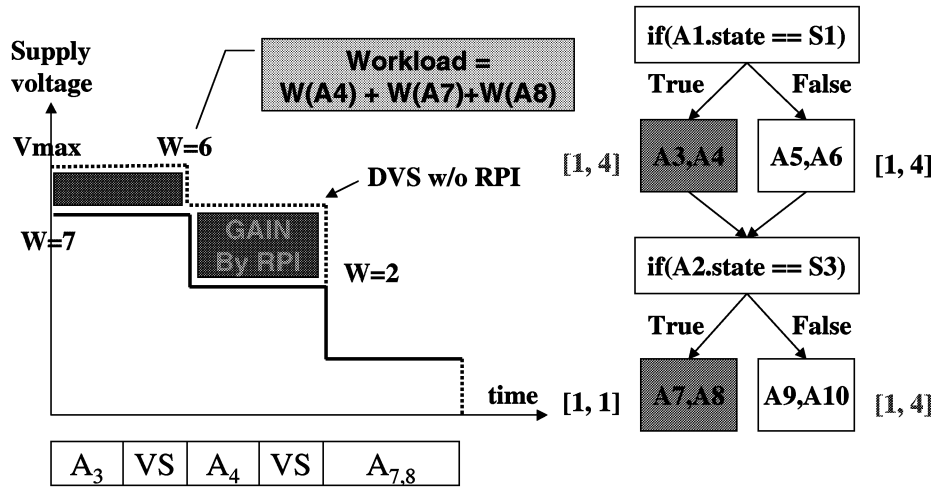
Freq. for A_1 : $F_1 = R \times F_0$

The effectiveness of DVS largely depends on the accuracy of estimated remaining workload (W)!!!

Intra-Task Dynamic Voltage Scaling



Intra-Task Dynamic Voltage Scaling



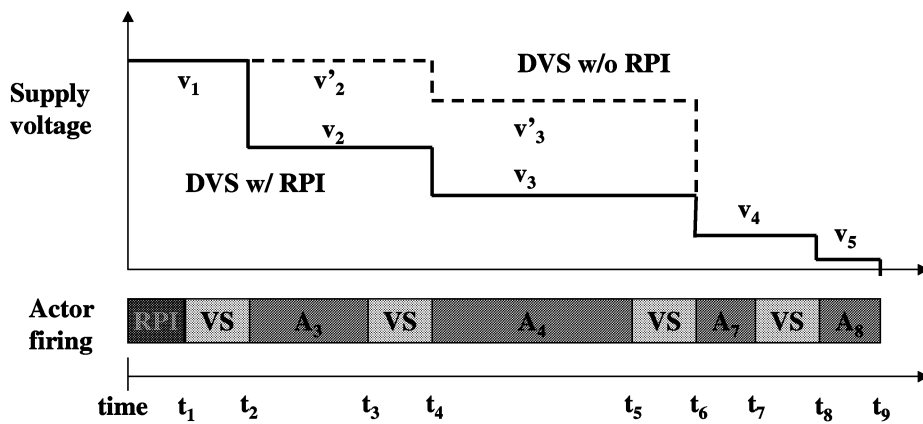
Intra-Task Dynamic Voltage Scaling

– Dynamic Voltage Scaling with RPI

- Example

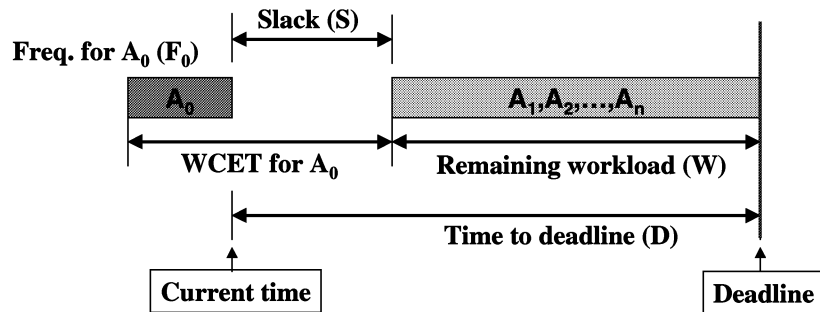


(a) Execution path identified during runtime



(b) Intra-task DVS based on exact execution paths

Intra-Task Dynamic Voltage Scaling



$$\text{Speed ratio : } R = \frac{W}{D - k \cdot T_{RPI} - n \cdot T_{VS}}$$

k : 1 if this is the first invocation of voltage scaler, 0 otherwise.
 n : # of remaining actors

$$\text{Freq. for } A_1 : F_1 = R \times F_0$$

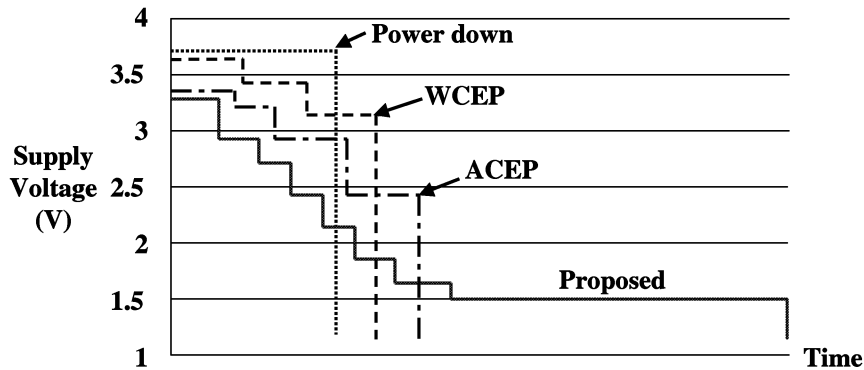
Intra-Task Dynamic Voltage Scaling

– HFSM-SDF model of MPEG4

- 9 hierarchical FSMs, 31 states, 44 state transitions
- 89 SDF actors (10 hierarchical, 79 leaf actors)
- Target architecture
 - Energy simulator based on commercial ISS (ARMulator) augmented with ARM8 variable voltage processor energy model
 - Include energy model of underlying (cacheless) memory subsystems

• Comparison of energy consumption

Power down	WCEP	ACEP	Proposed
214 mJ	174 mJ	158 mJ	138 mJ



References

References

- V. Tiwari, P. Ashar, and S. Malik, "Technology mapping for low power," *Proc. of Design Automation Conference*, pp. 74-79, June 1993
- T. Ahn and K. Choi, "Dynamic operand interchange for low power," *Electronics Letters*, pp. 2118-2120, Dec. 1997
- C.-Y. Tsui, M. Pedram, C.-A. Chen, and A.M. Despain, "Low power state assignment targeting two- and multi-level logic implementations," *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 82-87, Nov. 1994
- A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R.W. Brodersen, "Optimizing power using transformation," *IEEE Tr. on CAD/ICAS*, pp. 12-31, Jan. 1995
- R. Mehra, L.M. Guerra, and J.M. Rabaey, "Low power architectural synthesis and the impact of exploiting locality," *Journal of VLSI Signal Processing*, 1996
- E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," *Proc. of Int'l Symp. on Low Power Design*, pp. 99-104, Nov. 1995
- D. Kim and K. Choi, "Power-conscious high level synthesis using loop folding," *Proc. of Design Automation Conference*, pp. 441-445, June 1997
- A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," *Proc. of Int'l Conf. on Computer Design*, pp. 318-322, Oct. 1994
- E. Musoll and J. Cortadella, "Scheduling and resource binding for low power," *Proc. of Int'l Symp. on System Synthesis*, pp. 104-109, Apr. 1995
- A. Raghunathan and N. K. Jha, "An iterative improvement algorithm for low power data path synthesis," *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 597-602, Nov. 1995
- D. Shin and K. Choi, "Low power high level synthesis by increasing data correlation," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 62-67, Aug. 1997

References

- D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE JSSC*, pp. 663-670, June 1994
- T. Truman, T. Pering, R. Doering, and R. Brodersen, "The InfoPad multimedia terminal: a portable device for wireless information access," *IEEE Transactions on Computers*, pp. 1073-1087, October 1998
- L. Benini and G. De Micheli, "System-level power optimization: techniques and tools," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 288-293, Aug. 1999
- M.R. Stan, W.P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. on VLSI Systems*, Vol. 3, No. 1, pp. 49-58, Mar. 1995
- Y. Shin and K. Choi, "Narrow bus encoding for low power systems," *Proc. of Asia South Pacific Design Automation Conf.*, pp. 217-220, Jan. 2000
- S. Ramprasad, N.R. Shanbhag, and N. Hajj, "A coding framework for low-power address and data busses," *IEEE Tr. on VLSI Systems*, Vol. 7, No. 2, pp. 212-221, June 1999
- Y. Shin, S. Chae, and K. Choi, "Partial bus-invert coding for power optimization of system level bus," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 127-129, Aug. 1998
- T. Pering, T., and R. Brodersen, "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System," in *Power Driven Microarchitecture Workshop in conjunction with ISCA98*, June 1998
- T. Burd, T. Pering, A. Stratakos, and R Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE International Solid-State Circuits Conference*, 2000
- Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proc. of Design Automation Conf.*, pp. 134-139, June 1999
- M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. on VLSI Systems*, Vol. 4, No. 1, pp. 42-55, Mar. 1996

References

- C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 28-32, Nov. 1997
- T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 76-81, Aug. 1998
- L.S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *IEEE Tr. on VLSI Systems*, Vol. 2, No. 4, pp. 391-397, Dec. 1994
- T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. of Int'l Symp. on Low Power Electronics and Design*, pp. 197-202, Aug. 1998
- R. Ernst and W. Ye, "Embedded program timing analysis based on path clustering and architecture classification," *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 598-604, Nov. 1997
- Y. Shin, K. Choi, and T. Sakurai "Power optimization of real-time embedded systems on variable speed processors," *Proc. ICCAD*, 2000
- J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," *Proc. IEEE Real-Time Systems Symposium*, Dec. 1989.
- S. Lee, T. Sakurai, "Run-time voltage hopping for low-power real-time systems," *Proc. DAC*, 2000
- S. Lee, S. Yoo, and K. Choi, "An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model," *Proc. ISLPED*, 2002

Sponsored by:

