A Flexible Tool for Evaluating Scheduling, Synchronization and Partitioning Algorithms on Multiprocessors *

Farhang Nemati, Thomas Nolte
Mälardalen Real-Time Research Centre, Västerås, Sweden
{farhang.nemati,thomas.nolte}@mdh.se

Abstract

Multi-core platforms seem to be the way towards increasing performance of processors. Single-chip multiprocessors (multi-cores) are today the dominating technology for desktop computing. As the multi-cores are becoming the defacto processors, the need for new scheduling and resource sharing protocols has arisen. There are two major types of scheduling under multiprocessor/multi-core platforms. Global scheduling, under which migration of tasks among processors is allowed, and partitioned scheduling under which tasks are allocated onto processors and task migration is not allowed. The partitioned scheduling protocols suffer from the problem of partitioning tasks among processors/cores, which is a bin-packing problem. Heuristic algorithms have been developed for partitioning a task set on multiprocessor platforms. However, taking such technology to an industrial setting, it needs to be evaluated such that appropriate scheduling, synchronization and partitioning algorithms are selected.

In this paper we present our work on a tool for investigation and evaluation of different approaches to scheduling, synchronization and partitioning on multi-core platforms. Our tool allows for comparison of different approaches with respect to a number of parameters such as number of schedulable systems and number of processors required for scheduling. The output of the tool includes a set of information and graphs to facilitate evaluation and comparison of different approaches.

1 Introduction

The multiprocessor architectures are getting an increasing interest as the multi-cores offer higher performance and are becoming defacto processors in practice. This arises the need for new methods to take advantage of the multi-core platforms. A multi-core processor is a combination of two or more independent processors (cores) on a single chip, also called single-chip multiprocessors. The different cores are connected to a single shared memory via a shared bus. The cores typically have independent L1 caches and share an on-chip L2 cache. However, previously well-known and verified scheduling and synchronization protocols with an assumption of uniprocessors can not work properly on multi-cores, especially with the presence of shared resources. The industry has already begun to migrate towards multi-cores, although the existing scheduling and synchronization protocols are not yet mature enough to take advantage of the performance offered by multi-cores.

There have been several scheduling and synchronization protocols developed in the domain of multiprocessors. Mainly, two approaches for scheduling real-time systems on multiprocessors exist; global and partitioned scheduling [2, 1, 7, 9]. Under global scheduling, e.g., Global Earliest Deadline First (G-EDF), tasks are scheduled by a single scheduler and each task can be executed on any processor. A single global queue is used for storing jobs and a job can be preempted on a processor and resumed on another processor, i.e., migration of tasks among processors is permitted. Under a partitioned scheduling, tasks are statically assigned to processors and tasks within each processor are scheduled by a uniprocessor scheduling protocol, e.g., Rate Monotonic (RM) and EDF. Each processor is associated with a separate ready queue for scheduling task jobs.

Partitioned scheduling protocols have been used more often and are supported (with fixed priority scheduling) widely by commercial real-time operating systems [14], because of their simplicity, efficiency and predictability. Besides, the well studied uniprocessor scheduling and synchronization methods can be reused for multiprocessors with less changes (or no changes). However, partitioning (allocating tasks to processors) is known to be a bin-packing problem which is a NP-hard problem in the strong sense; hence finding an optimal solution in polynomial time is not realistic in the general case. Thus, to take advantage of performance offered by multi-cores, scheduling pro-

*This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via Mälardalen Real-Time Research Centre (MRTC) at Mälardalen University.

tocols should be coordinated with appropriate partitioning algorithms. Heuristic approaches and sufficient feasibility tests for bin-packing algorithms have been developed to find a near-optimal partitioning [2, 7]. However, the scheduling protocols and existing partitioning algorithms for multiprocessors (multi-cores) mostly assume independent tasks while in real applications, tasks often share resources.

We have proposed a *blocking-aware* partitioning algorithm [17, 18]. The assumption include presence of mutually exclusive shared resources. The heuristic partitions a system (task set) on an identical shared memory single-chip multiprocessor (multi-core) platform. In the context of this paper the blocking-aware algorithm refers to an algorithm that attempts to decrease blocking overheads by assigning tasks to appropriate processors (partitions). This consequently increases the schedulability of the system and may reduce the number of processors. In contrast, a *blocking-agnostic* algorithm refers to a bin-packing algorithm that does not consider blocking parameters and does not attempt to decrease the blocking overhead, although blocking times are included in the schedulability test. Our blocking-aware algorithm identifies task constraints, e.g., dependencies between tasks, timing attributes, and resource sharing, and extends the best-fit decreasing (BFD) bin-packing algorithm with blocking time parameters. A similar heuristic has been proposed in [14].

As the scheduling and synchronization protocols together with partitioning algorithms are being developed, the industry needs to evaluate the different methods to choose appropriate methods and apply them in their applications. This arises the need for development of tools to facilitate investigation and evaluation of different approaches and compare them to each other according to different parameters. Hence, in this paper we present a tool which we have developed for evaluation of different scheduling, synchronization protocols coordinated with different partitioning algorithms. The output of the tool includes a set of information and graphs to facilitate evaluation and comparison of different approaches. We have implemented our blocking-aware partitioning algorithm together with the algorithm proposed in [14] and added them to the tool. The tool is modular making it possible to easily add any new scheduling, synchronization and partitioning algorithm. However, in this paper the focus of the tool has been directed to partitioned scheduling and synchronization approaches as well as partitioning heuristics while extending the tool to global scheduling methods remains as a future work.

The rest of the paper is as follows: we present the task and platform model in Section 2. We briefly explain our partitioning algorithms together with the existing algorithm in Section 3. In Section 4 we present the tool. In Section 5 we present some examples of the outputs of the tool in which we have compared different partitioning algorithms.

1.1 Related Work

A study of bin-packing algorithms for designing distributed real-time systems is presented in [8]. The method partitions software into modules to be allocated on hardware nodes. In their approach they use two graphs; a graph which models software modules and a graph that represents the hardware architecture. The authors extend the bin-packing algorithm with heuristics to minimize the number of bins (processors) needed and the bandwidth required for the communication between nodes. However, their partitioning method assumes independent tasks.

Liu et al. [15] present a heuristic algorithm for allocating tasks in multi-core-based massively parallel systems. Their algorithm has two rounds; in the first round processes (groups of threads - partitions in this paper) are assigned to processing nodes, and the second round allocates tasks in a process to the cores of a processor. However, the algorithm does not consider synchronization between tasks.

Baruah and Fisher have presented a bin-packing partitioning algorithm (first-fit decreasing (FFD) algorithm) in [3] for a set of sporadic tasks on multiprocessors. The tasks are indexed in non-decreasing order based on their relative deadlines and the algorithm assigns the tasks to the processors in first-fit order. The algorithm assigns each task τ_i to the first processor, P_k for which both of following conditions, under the Earliest Deadline First (EDF) scheduling, hold:

$$D_i - \sum_{\tau_j \in P_k} DBF^*(\tau_j, D_i) \geq C_i$$

and

$$1 - \sum_{\tau_j \in P_k} u_j \geq u_i$$

where C_i and D_i specify worst-case execution time (WCET) and deadline of task τ_i respectively, $u_i = \frac{C_i}{T_i}$, and

$$DBF^*(\tau_i, t) = \begin{cases} 0 & \text{if } t < D_i; \\ C_i + u_i(t - D_i) & \text{otherwise.} \end{cases}$$

The algorithm, however, assumes independent tasks.

In the work presented by Lakshmanan et al. in [14] they investigate and analyze two alternatives of execution control policies (suspend-based and spin-based remote blocking) under multiprocessor Multiprocessor Priority Ceiling Protocol (MPCP) [19]. They have developed a blocking-aware task allocation algorithm (an extension to BFD) and evaluated it under both execution control policies.

In their partitioning algorithm, the tasks that directly or indirectly share resources are put into what they call bundles (in this paper we call them macrotasks) and each bundle is tried to be allocated onto a processor. The bundles that can not fit into any existing processors are ordered by their cost, which is the blocking overhead that they introduce into the system. Then the bundle with minimum cost

is broken and the algorithm is run from the beginning. However, their algorithm does not consider blocking parameters when it allocates the current task to a processor, but only its size (utilization). Furthermore, no relationship (e.g., as a cost based on blocking parameters) among individual tasks within a bundle is considered which could help to allocate tasks from a broken bundle to appropriate processors to decrease the blocking times. However, according to our experimental results performed by the our tool, their heuristic performs slightly better than blocking-agnostic algorithm, and our algorithm performs significantly better than both.

In the context of multiprocessor synchronization, the first protocol was MPCP presented by Rajkumar in [19], which extends PCP [20] to multiprocessors hence allowing for synchronization of tasks sharing mutually exclusive resources using partitioned fixed priority scheduling (FPS) protocols. Our partitioning algorithm attempts to decrease blocking times under MPCP and consequently decrease worst case response times which in turn may reduce the number of needed processors. Gai et al. [13, 12] present MSRP (Multiprocessor SRP), which is a P-EDF (Partitioned EDF) based synchronization protocol for multiprocessors. The shared resources are classified as either (i) local resources that are shared among tasks assigned to the same processor, or (ii) global resources that are shared by tasks assigned to different processors. In MSRP, tasks synchronize local resources using SRP [1], and access to global resources is guaranteed a bounded blocking time. Lopez et al. [16] present an implementation of SRP under P-EDF. Devi et al. [10] present a synchronization technique under G-EDF. The work is restricted to synchronization of non-nested accesses to short and simple objects, e.g., stacks, linked lists, and queues. In addition, the main focus of the method is on soft real-time systems.

Block et al. [4] present FMLP (Flexible Multiprocessor Locking Protocol), which is the first synchronization protocol for multiprocessors that can be applied to both partitioned and global scheduling algorithms, i.e., P-EDF and G-EDF. An implementation of FMLP has been described in [5]. However, although in a longer version of [4] (available at http://www.cs.unc.edu/˜anderson/papers/rtcsa07along.pdf), the blocking times have been calculated, but to our knowledge there is no schedulability test for FMLP.

Recently, a synchronization protocol under fixed priority scheduling, has been proposed by Easwaran and Andersson in [11], but they focus on a global scheduling approach.

2 Task and Platform Model

The tool is capable of performing evaluations by both fixed priority and dynamic scheduling scheduling protocols. The tasks can also share resources. Thus the task model is assumed as a task set that consists of n sporadic tasks, $\tau_i(T_i, C_i, \rho_i, \{c_{i,p,q}\})$ and $\tau_i(T_i, C_i, \{c_{i,p,q}\})$ for dynamic and fixed priority scheduling protocols respectively, where T_i is the minimum inter-arrival time between two successive jobs of task τ_i with worst-case execution time C_i and ρ_i (in fixed priority scheduling task model) as its priority. The tasks share a set of resources, $R = \{R_q\}$ which are protected using semaphores. The set of critical sections, in which task τ_i requests resources in R is denoted by $\{c_{i,p,q}\}$, where $c_{i,p,q}$ indicates the maximum execution time of the p^{th} critical section of task τ_i in which the task locks resource $R_q \in R$. Critical sections of tasks should be sequential or properly nested. The deadline of each job is equal to T_i. A job of task τ_i, is specified by J_i. The utilization factor of task τ_i is denoted by u_i where $u_i = C_i/T_i$.

The tool also assumes that the multiprocessor (multicore) platform is composed of identical, unit-capacity processors (cores) with shared memory. The task set is partitioned into partitions $\{P_1, \ldots, P_m\}$, and each partition is allocated onto one processor (core), thus m represent the minimum number of required processors.

3 Included Partitioning Algorithms

In this section we briefly present the partitioning algorithms we have developed and added to the tool. Please observe that the tool is flexible and any new partitioning algorithm can be added to the tool easily.

We have implemented and added three partitioning algorithms: (i) a blocking-aware algorithm which we proposed in [18], (ii) a similar blocking-aware algorithm proposed in [14] and (iii) a blocking-agnostic algorithm. We have explained these algorithms in details in [18].

Our blocking-aware algorithm is an extension to the BFD algorithm. In a blocking-agnostic BFD algorithm, bins (processors) are ordered in non-increasing order of their utilization and tasks are ordered in non-increasing order of their size (utilization). The algorithm attempts to allocate the task from the top of the ordered task set onto the first processor that fits it (i.e., the first processor on which the task can be allocated while all processors are schedulable), beginning from the top of the ordered processor list. If none of the processors can fit the task, a new processor is added to the processor list. At each step the schedulability of all processors should be tested, because allocating a task to a processor can increase the remote blocking time of tasks previously allocated to other processors and may make the other processors unschedulable. This means, it is possible that some of the previous processors become unschedulable even if a task is allocated to a new processor, which makes the algorithm fail.

Our algorithm attempts to decrease the blocking times of tasks by partitioning a task set on processors based on heuristics. This generally increases the schedulability of a task set which may reduce the number of partitions (proces-

sors). The algorithm attempts to allocate the tasks that directly or indirectly share resources onto the same processor. Tasks that directly or indirectly share resources are called *macrotasks*, e.g., if tasks τ_i and τ_j share resource R_p and tasks τ_j and τ_k share resource R_q, all three tasks belong to the same macrotask.

The algorithm performs partitioning of a task set in two rounds and the result will be the output of the round with better partitioning results. Each round allocates tasks to the processors in a different strategy. When a bin-packing algorithm allocates an object (task) to a bin (processor), it usually allocates the object in a bin that fits it better, and it does not consider the unallocated objects that will be allocated after the current object. The rational behind the two rounds is that the heuristic tries to consider both past and future by looking at tasks allocated in the past and those that are not allocated yet. In the first round the algorithm considers the tasks that are not allocated to any processor yet; and tries to take as many as possible of the best related tasks (based on remote blocking parameters) with the current task. On the other hand, in the second round it considers the already allocated tasks and tries to allocate the current task onto the processor that contains best related tasks to the current task. In our tool, for more precise schedulability analysis, it always performs response time analysis [6] to check schedulability test of a task set.

We have also implemented and added the partitioning algorithm proposed in [14] which is similar to our blocking-aware algorithm. Their algorithm, similarly to our algorithm, attempts to group tasks in macrotasks and allocate each macrotask on a processor. The macrotasks that can not fit onto processors are ordered in the order of the cost of breaking them. The cost of breaking a macrotask is defined based on the estimated cost (blocking overhead) introduced into the tasks by transforming a local resource into a global resource (i.e., the tasks sharing the resource are allocated to different processors). The macrotask with minimum breaking cost is picked and is broken in two pieces such that the size of one piece is as close as the largest utilization available among processors. If the fitting is still not possible a new processor is added and the whole algorithm is repeated again. However, their algorithm does not consider blocking parameters when it allocates a task from a broken macrotask to a processor, but only its utilization, i.e. the tasks are ordered in order of their utilization only. On the other hand, our algorithm assigns a weight which besides the utilization includes the blocking terms as well. Besides, in our heuristic, we have defined an attraction function, which attracts the most attracted tasks to the picked task from its broken macrotask, and attempts to allocate them on the same processor.

4 The Tool

In this section we present our evaluation and partitioning tool.

4.1 The Structure

The tool has been developed in an object-oriented manner and every concept has been treated as an object, e.g., tasks, critical sections, resources, processors, etc.

We aimed to make the tool flexible to be able to easily add any partitioning, scheduling and synchronization (lock-based) algorithm. Thus, we have separated the development in three major parts (packages) as shown in Figure 1; *Scheduling Analysis* package, *Partitioning Algorithms* package, and *Task Generation* package.

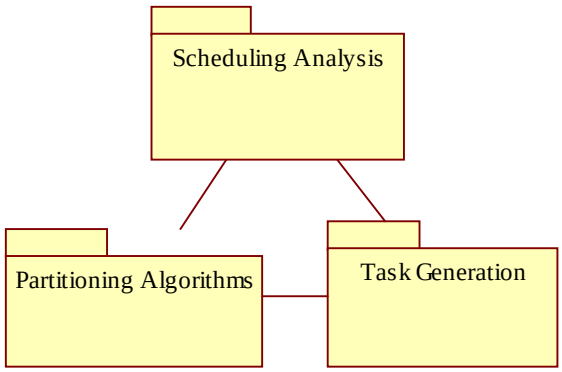


Figure 1. The three major packages

When a partitioning algorithm attempts to assign a task to a processor it should test the schedulability of the all processors, hence it uses the classes in the scheduling analysis to perform the test. As the schedulability analysis is different depending on different scheduling or synchronization protocols (e.g., different blocking time terms), several classes are provided in the scheduling analysis package to facilitate the schedulability test in an object-oriented manner. This makes the package reusable and extendable as the new scheduling and synchronization protocols are added.

4.1.1 The Scheduling Analysis package

This package contains classes associated with scheduling protocols, e.g., RM, as well as synchronization protocols, e.g., MPCP. Besides classes used for schedulability test for each scheduling protocol, the package contains classes to facilitate calculation of blocking times of tasks to be used in the schedulability analysis. Under a multiprocessor synchronization protocol, any task, τ_i, may face mainly two types of blocking times; (i) the local blocking times by

interference from the lower priority tasks assigned to the same processor as τ_i's processor, (ii) the remote blocking times introduced by the tasks (with any priority) assigned to different processors than of τ_i's processor.

To easily and in a modular manner calculate the local and remote blocking times of each task, in this package a task class includes a local processor class and a set of remote processors, i.e., the local processor is the processor that the task is assigned to and the rest of processors are contained in the remote processors set. Under partitioned scheduling approaches the total blocking time of a task is the summation of the local and the global blocking terms. Depending on the used synchronization protocol, the local and remote blocking terms of the task on each processor may be different, e.g., under MPCP the total blocking time (B_i) of task τ_i, consists of five blocking terms, of which one is local and four different remote blocking terms [19]. Figure 2 shows the local processor and the remote processors associated with the task class. This structure facilitates calculating each term of the blocking time of a task from each processor, i.e., the local blocking terms introduced from the local tasks (tasks allocated on the same processor as τ_i's processor) are calculated using local processor class and the remote blocking terms from remote tasks (tasks allocated on a different processor than of τ_i's processor) are calculated by the remote processors. Each scheduling and synchronization protocol uses these classes differently as they may have different blocking time terms.

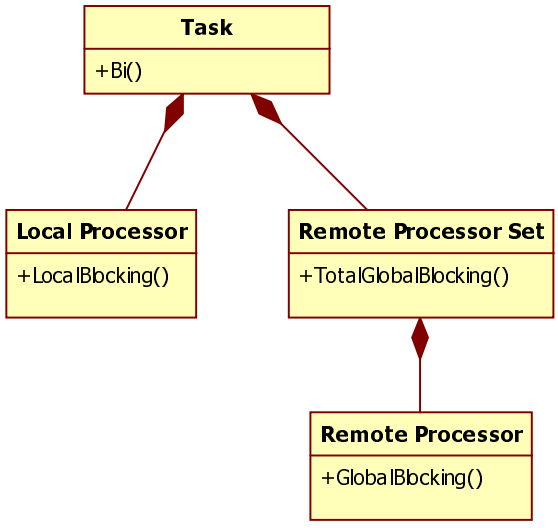


Figure 2. The local and remote blocking times calculation

For more precise schedulability test in scheduling protocols, in this package response times analysis is performed by calculating the worst case response time of each task. Thus, another output of the package is the worst case response times of the tasks within a task set.

4.1.2 The Partitioning Algorithms package

This package is used to partition a task set to be allocated onto a multiprocessor platform. Any partitioning algorithm can easily be plugged into the package. The only requirement of the a new algorithm is that it has to have the task set, the target scheduling and synchronization protocols as inputs. The output of the algorithm will be a set processors each of which contains the allocated tasks. Each allocated task contains its calculated worst case response time.

The purpose of a partitioning may be different. The goal can be to plug in a partitioning heuristic to reduce the required number of processors. On the other hand, the goal of a partitioning algorithm may be to distribute the tasks fairly onto processors to balance the utilization of processors, thus, the output of each algorithm include the utilization of each processor as well.

We have developed a blocking-aware partitioning heuristic (Section 3). We have implemented our algorithm together with a similar blocking-aware algorithm and plugged into the partitioning algorithms package. The objective of those algorithms is to reduce the blocking times of tasks by co-allocating the tasks sharing the same resources as far as possible. Furthermore, we have implemented a BFD bin-packing algorithm (blocking-agnostic algorithm) and inserted this algorithm into the package.

Any partitioning algorithm needs to test schedulability of each processor each time it allocates a task or a group of tasks on a processor. The algorithms, within this package use the schedulability analysis provided in the scheduling analysis package. This separates the partitioning algorithms from the schedulability analysis making it easy to develop any new partitioning algorithms and scheduling protocols independently and insert them into the tool.

4.1.3 The Task Generation package

This package is used for task set generation in two different ways. The tool can be used for two different purposes; (i) the schedulability analysis and partitioning of a task set defined by a user, or (ii) evaluation and comparison of different scheduling, synchronization and partitioning algorithms according to a number of randomly generated task sets. This package provides two different ways of task set generation. One way is to provide the user to enter the tasks, critical sections, resources, and relationships between tasks and resources. In this case the tool partitions the task set using the selected partitioning algorithm and the selected schedulability test. The second way is to generate a number of task sets according to several given parameters (Figure 3). In this case the tool uses the generated task sets to

perform evaluation and comparison of different scheduling, synchronization and partitioning algorithms.

As shown in Figure 3, for the random task set generation two groups of parameters are provided. The first group includes the desired number of task sets, total workload, the number of tasks per processor and maximum execution time of each task (maximum WCET). The minimum WCET is limited by the maximum number and length of critical sections per each task, e.g., with maximum number of critical section set to 5 and maximum length of any critical section set to 6 the minimum execution time of any task will be 30. The second group of parameters for the task sets are resource sharing parameters, i.e., the number of resources shared among tasks of each task set, minimum number, maximum number of critical sections per each task, minimum length and maximum length of each critical section. The random task generation provides the possibility of generating task sets by combination of the parameters which can be used to evaluate algorithms.

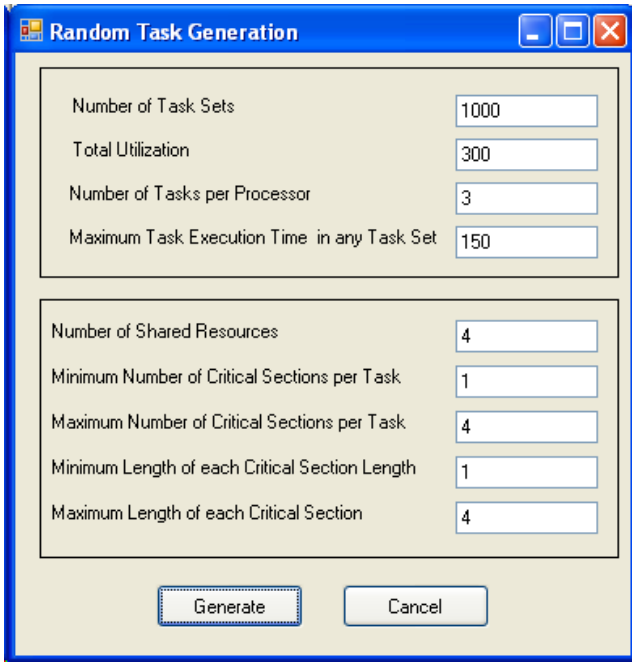


Figure 3. The randon task generation

The random task generation process performed by this package is as follows.

The total workload presents the number of fully utilized processors (e.g., Total Utilization $= 300$ means 3 fully utilized processors). The utilization of each of the processors (utilization $= 100\%$) is randomly divided among the given number of tasks per processor, e.g., for 3 tasks per processor a possible utilization assignment of the tasks of a processor can be 25%, 45% and 30% respectively. Usually for generating task sets, utilization and periods are randomly assigned to tasks and worst case execution times of tasks are calculated based on them. However, in our random task set generation package, the worst case execution times (WCET) of tasks are randomly assigned and the period of each task is calculated based on its utilization and WCET. The reason is that we had to restrict that the WCET of a task not to be less than the maximum length of its critical sections restricted by the maximum number of critical sections per each tasks and the maximum length of each critical section.

The number of critical sections for each task is randomly chosen from the range between the defined minimum and maximum number of critical sections. The length of each critical section is chosen the same way. For each critical section the accessed resource is randomly chosen from the defined resources, e.g, given the number of resources $= 2$ (R_1 and R_2), the resource accessed by any critical section is randomly chosen from $\{R_1, R_2\}$.

5 Example: An Evaluation and Comparison of Partitioning Algorithms

In this section we present an experimental example which we have performed by our tool. In this example we have compared the outputs of the two blocking-aware heuristics as well as the blocking-agnostic bin-packing algorithm. To ease referring to our blocking-aware algorithm and the similar algorithm proposed in [14] in this example we refer them as BPA and SPA respectively. The scheduling and synchronization protocols under which the algorithms were evaluated were the RM and the MPCP respectively.

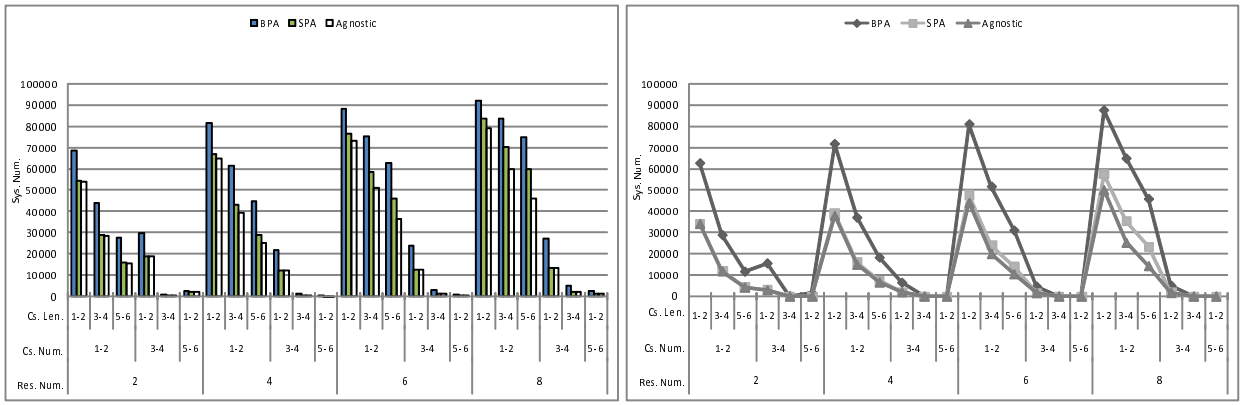
For a number of systems (task sets), we have compared the performance of the algorithms in two aspects; (i) Given a number of systems, what is the total number of systems that each of the algorithms can schedule, (ii) what is the processor reduction aspect of the two algorithms.

5.1 Task Set Generation

Using the random task set generation, we generated systems (task sets) for different workloads. Since we have limited the maximum number of critical sections to 6 and the maximum length of any critical section to 6 time units, hence the execution time of each task should be greater than 36 time units. The maximum execution time of any task was defined as 150 time units.

We ran the evaluation tool with respect to different configurations with the input parameters as follows.

- Workload (3, 4, 6, or 8 fully utilized processors).
- The number of tasks per processor (3, 6 or 9 tasks per processor).
- The number of resources (2, 4 or 6).



(a) Workload: 3 processors, 3 tasks per processor

(b) Workload: 3 processors, 6 tasks per processor

Figure 4. Output the tool for performance of the algorithms with respect to task sets each algorithm schedules.

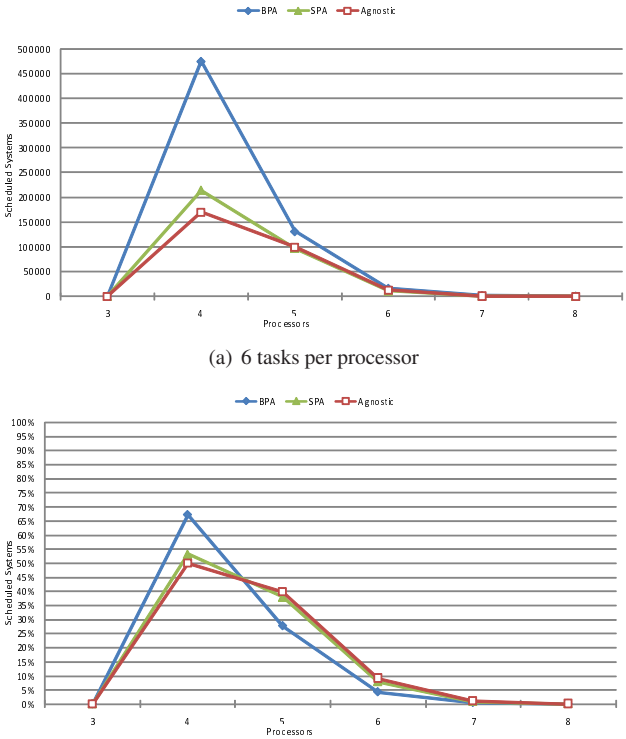
- The range of the number of critical sections per task (1 to 2, 3 to 4 or 5 to 6 critical sections per task).
- The range of length of critical sections (1 to 2, 3 to 4, or 5 to 6).

For each configuration, we chose the number of systems to be 100.000, and combining the parameters of configurations (432 different configurations), the total number of systems generated for the experiment sums up to 43.200.000.

With the generated systems we were able to evaluate the partitioning algorithms with respect to different factors, i.e., various workloads (number of fully utilized processors), number of tasks per processor, number of shared resources, number of critical sections per task, and length of critical sections.

Figure 4 shows the examples of the output regarding the number of schedulable systems by each algorithm, i.e., the first aspect of comparison of the partitioning algorithms. The vertical axis shows the total number of systems that the algorithms could schedule successfully. The horizontal axis shows three factors in three different lines; the bottom line shows the number of shared resources within systems (Res. Num.), the second line shows the number of critical sections per task (Cs. Num.), and the top line represents the length of critical sections within each task (Cs. Len.), e.g., Res. Num.=4, Cs. Num.=1-2, and Cs. Len.=1-2 represents the systems that share 4 resources, the number of critical sections per each task are between 1 and 2, and the length of these critical sections are between 1 and 2 time units.

The second aspect for comparison of performance of the algorithms is the processor reduction aspect. The examples of the output of the tool for illustrating this aspect are shown in Figure 5. For each algorithm, the total number of schedulable systems are ordered in order of the number of required processors. The schedulable systems of an algorithm with each number of processors can also be illustrated as percentage of the total scheduled systems.



(a) 6 tasks per processor

(b) 9 tasks per processor (as percentage of the total scheduled systems)

Figure 5. Task sets each algorithm schedules, ordered by required number of processors. Workload: 3 processors.

6 Conclusion

In this paper we have presented our work on a tool that we have developed for evaluation of different scheduling and synchronization protocols coordinated with different partitioning algorithms. The output of the tool includes a set of information and graphs to facilitate evaluation and comparison of different approaches.

Moreover, we briefly presented our blocking-aware partitioning algorithm proposed in [18] together with a similar algorithm proposed in [14]. We have implemented the two approaches together with an usual bin-packing (blocking-agnostic) algorithm and added all three approaches to the tool. The tool has the possibility to evaluate and compare different multiprocessor scheduling, synchronization and partitioning algorithms. The tool has been developed in an object-oriented manner making the tool flexible. Any new scheduling, synchronization or partitioning algorithm can be developed and added to the tool easily. We have presented a few examples of the illustrated outputs of the tool for evaluation and comparison of the partitioning algorithms included in the tool.

The focus of the tool is currently multiprocessor partitioned scheduling protocols and extending the tool to global scheduling and synchronization protocols remains as a future work. Another plan for future work is to extend the tool to simulate the execution of a task set on a multi-core platform and visualize the simulated timing behavior of the task set. In the domain of multiprocessor scheduling and synchronization we will also work on investigating global and hierarchical scheduling protocols and appropriate synchronization protocols.

References

[1] T. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3(1):67–99, 1991.

[2] T. Baker. A comparison of global and partitioned EDF schedulability test for multiprocessors. Technical report, 2005.

[3] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *proceedings of 26th IEEE Real-Time Systems Symposium (RTSS'05)*, pages 321–329, 2005.

[4] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *proceedings of 13th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*, pages 47–56, 2007.

[5] B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson. Synchronization on multiprocessors: To block or not to block, to suspend or spin? In *proceedings of 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, pages 342–353, 2008.

[6] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In *Principles of Real-Time Systems*, pages 225–248. Prentice Hall, 1994.

[7] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[8] D. de Niz and R. Rajkumar. Partitioning bin-packing algorithms for distributed real-time systems. *Journal of Embedded Systems*, 2(3-4):196–208, 2006.

[9] U. Devi. Soft real-time scheduling on multiprocessors. In *PhD thesis, available at www.cs.unc.edu/˜anderson/diss/devidiss.pdf*, 2006.

[10] U. Devi, H. Leontyev, and J. Anderson. Efficient synchronization under global EDF scheduling on multiprocessors. In *proceedings of 18th IEEE Euromicro Conference on Real-time Systems (ECRTS'06)*, pages 75–84, 2006.

[11] A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *proceedings of 30th IEEE Real-Time Systems Symposium (RTSS'09)*, pages 377–386, 2009.

[12] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the janus multiple processor on a chip platform. In *proceedings of 9th IEEE Real-Time And Embedded Technology Application Symposium (RTAS'03)*, pages 189–198, 2003.

[13] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *proceedings of 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, pages 73–83, 2001.

[14] K. Lakshmanan, D. de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *proceedings of 30th IEEE Real-Time Systems Symposium (RTSS'09)*, pages 469–478, 2009.

[15] Y. Liu, X. Zhang, H. Li, and D. Qian. Allocating tasks in multi-core processor based parallel systems. In *proceedings of Network and Parallel Computing Workshops, in conjunction with IFIP'07*, pages 748–753, 2007.

[16] J. M. López, J. L. Díaz, and D. F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Journal of Real-Time Systems*, 28(1):39–68, 2004.

[17] F. Nemati, T. Nolte, and M. Behnam. Blocking-aware partitioning for multiprocessors. Technical report, Mälardalen Real-Time research Centre (MRTC), Mälardalen University, March 2010. Available at http://www.mrtc.mdh.se/publications/2137.pdf.

[18] F. Nemati, T. Nolte, and M. Behnam. Partitioning real-time systems on multiprocessors with shared resources. In *In submition*, 2010.

[19] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[20] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *Journal of IEEE Transactions on Computers*, 39(9):1175–1185, 1990.