A Cloud Based Super-Optimization Method to Parallelize the Sequential Code's Nested Loops

Amin Majd
Department of Information Technology
Åbo Akademi University
Turku, Finland
amin.majd@abo.fi

Mohammad Loni
School of Innovation, Design and Engineering
Mälardalen University, Sweden
mohammad.loni@mdh.se

Golnaz Sahebi
Department of Information Technology
University of Turku
Turku, Finland
Golnaz.Sahebi@utu.fi

Masoud Daneshtalab
School of Innovation, Design and Engineering
Mälardalen University, Sweden
masoud.daneshtalab@mdh.se

Elena Troubitsyna
Theoretical Computer Science Department
KTH Royal Institute of Technology, Stockholm, Sweden
Elena.troubitsyna@abo.fi

***Abstract*— Advances in hardware architecture regarding multi-core processors make parallel computing ubiquitous. To achieve the maximum utilization of multi-core processors, parallel programming techniques are required. However, there are several challenges standing in front of parallel programming. These problems are mainly divided into three major groups. First, although recent advancements in parallel programming languages (e.g. MPI, OpenCL, etc.) assist developers, still parallel programming is not desirable for most programmers. The second one belongs to the massive volume of old software and applications, which have been written in serial mode. However, converting millions of line of serial codes to parallel codes is highly time-consuming and requiring huge verification effort. Third, the production of software and applications in parallel mode is very expensive since it needs knowledge and expertise.**

Super-optimization provided by super compilers is the process of automatically determine the dependent and independent instructions to find any data dependency and loop-free sequence of instructions. Super compiler then runs these instructions on different processors in the parallel mode, if it is possible. Super-optimization is a feasible solution for helping the programmer to get relaxed from parallel programming workload. Since the most complexity of the sequential codes is in the nested loops, we try to parallelize the nested loops by using the idea of super-optimization. One of the underlying stages in the super-optimization is scheduling tiled space for iterating nested loops. Since the problem is NP-Hard, using the traditional optimization methods are not feasible.

In this paper, we propose a cloud-based super-optimization method as Software-as-a-Service (SaaS) to reduce the cost of parallel programming. In addition, it increases the utilization of the processing capacity of the multi-core processor. As the result, an intermediate programmer can use the whole processing capacity of his/her system without knowing anything about writing parallel codes or super compiler functions by sending the serial code to a cloud server and receiving the parallel version of the code from the cloud server. In this paper, an evolutionary algorithm is leveraged to solve the scheduling problem of tiles. Our proposed super-optimization method will serve as software and provided as a hybrid (public and private) deployment model.

Keywords*— *Parallel Programming; Cloud Computing; Super Compiler; Speedup; Diophantine Algorithm

I. INTRODUCTION

Multi-core processors provide a rich computing capacity and from which a wide range of application domains can benefit. For maximizing the utilization of multi-core environments, parallel programming is required since the serial codes only take one core for the processing, while the other cores are idle. On the other hand, parallel programming is time consuming and requires deep knowledge of hardware, compiler, performance profilers and code parallel patterns. Many high-level parallel programming libraries have been proposed to overcome the parallel programming challenges, however, they are expressive and inefficient [15] [1].

Leveraging super-optimization technique provided by super compilers is an alternative solution. Super compilers are dynamic code optimizers that run the independent parts of the code on available cores in parallel. However, using super-compiler has programming overhead since it needs expertise to work with super-compiler functions. The problem will be more highlighted when different programming languages have different super compilers.

Cloud computing provide new processing and delivery model for IT service. Cloud-based services are device-independent, on-demand, and cost-efficient. Plus, Cloud improves the performance of the QoS parameters such as reliability, scalability, resource utilization, system throughput, response time, system stability and power consumption. Software-as-a-Service (SaaS) is a software delivery scheme in which the software is delivered via web. SaaS helps organizations avoid capital expenditure and let them focus on their core business instead of support services such as IT infrastructure management, software maintenance, etc.

Nested loops are more complex than the other normal instructions which they are highly prevalent in the most application programs. Therefore, they can be parallelized to reduce execution times. We will parallelize them as a SIMD method [4] and [5]. Some prior works [2] [17] [20] tried to provide compiler services on the cloud. Although cloud-based compilers are effective, they just help users by reducing the

price of service and/or faster compilation for huge codes size and cannot help users to run applications in parallel mode.

Converting nested loops into parallel mode is done via the following steps: (1) Creating a timetable to save loop data dependencies [13]. (2) Tiling the iteration space for achieving better parallelization performance. A tile is a set of loop iterations running on the same processor. (3) Generating parallel code corresponding to the shape and size of produced tiles automatically for the iteration space of step 2. (4) In the last step, we need to schedule the tiling space of the former step.

In this paper, we propose a super-optimization technique provided as SaaS to tackle the challenge of super compilers and benefit from cost-efficient SaaS solutions. In the proposed code optimization paradigm, the programmer sends the sequential codes to the cloud, then receive the parallel version of the code. This paradigm avoids programmers from deep understanding of code parallelization leading to decrease development time and increase the utilization of the processing capacity of the system. The users who can benefit from the proposed SaaS super-optimization are divided into four categories including: (1) programmer who can write program only by serial approaches. (2) Owner or programmer of any program or software who cannot rewrite them by parallel techniques. (3) Users or programmers who cannot work with super compilers. (4) Users or programmers who cannot buy super compilers. Since the scheduling of tiling space is an NP-Hard problem, we leveraged a customized multi-population genetic algorithm (MPGA) approach to efficiently explore the design space in a reasonable time.

Contribution. In this paper we propose a super-optimization method provided as SaaS that parallelize the nested loops of the program to diminish the complexity of writing parallel code for programmers. This code compilation paradigm simultaneously provides improved compile time, run time, space storage, and the price. In addition, we used MPGA to provide a fast and near-optimal scheduling.

Paper Organization. The paper is organized as following: Section II represents the background needed for our work. Section III reviews the related research work. Details of the proposed method are presented in Section IV. Section V presents the experimental results. Conclusions are presented in Section VI.

II. BACKGOUNDS

A. Cloud Computing

Cloud computing is a model for enabling omnipresent network access to a shared pool of unlimited computing resources. This enable the end users to access the cloud computing resource anytime from any platform, such as mobile cellphone to high-performance the desktop computers [2]. Cloud advantages are including reducing setup costs of applications, easy recovery, and scalable environment [3].

The systems architecture of the cloud computing includes multiple components communicating with each other over the application programming interfaces. Such as the philosophy of UNIX, it has multiple programs that each does one thing well and work together over general interfaces. Cloud platforms mainly provide three services categorizing as: 1) ***SaaS*** (Software-as-a-Service), 2) ***PaaS*** (Platform-as-a-Service), and 3) ***IaaS*** (Infrastructure-as-a-Service) [2]. SaaS provides a software for users that is installed on the cloud servers and users utilize it over the internet. Software can be accessed via the network for different clients. In this processing paradigm, clients usually do not require to software installation, instead it is enough to have a browser or other client device and network connectivity. Due to the benefits of SaaS, we provide the super compiler application as a SaaS, uploaded in a local server.

Four available deployment models of cloud are listed as follows: *private cloud*, *public cloud*, *community cloud* and *hybrid cloud*. Any subscriber can access to the *public cloud* with an internet connection. A specific group or organization can access to the *private cloud*, and a permission is issued to access for resource of the cloud. *Community cloud* is collaborative cloud infrastructure that is shared between several organizations with common profits. A hybrid cloud is fundamentally a combination of at least two clouds. In this paper, we utilize the hybrid cloud to enable everyone to access to the cloud.

B. Super-optimization

A super compiler is a program optimizer that transforms the code of a program with the objective of reduction the running time. Automated parallel, which is a kind of super compilers, is an application that gets serial code, and converts it to parallel code in order to run on several processors in the shared memory multi-processor machines. The goal of automatic parallelization is to release the programmers from writing parallel code that cause to waste a lot of time and energy. The main attention of the programmers is to focus on the nested loops in the programs, because the loops get the most section of a program's runtime. In the engineering applications, the loop parallelism is the most important task toward code parallelization.

```
1.  for (p = 6; p < q; p++ )
2.          y[p]  =  y[p-1] + I ;
```

p=6	p=7	p=8
y[6] = x[5] + I	y[7] = y[6] + I	y[8] = y[7] + I

Fig. 1. Dependent loope

```
1.  for (p= 1; p < q; p++)
2.          y[p] = y[p] + I;
```

p=6	p=7	p=8
y[6] = y[6] + I	y[7] = y[7] + I	y[8] = y[8] + I

Fig. 2. Independent loop

In *numerical programs*, the first step in detecting loop level parallelism is testing data dependency which is needed to detect parallelism in programs. The data dependency exists between two adjacent data references if both references access the same memory location, and at least one of them is a write access (Fig. 2). Diophantine equation is a polynomial equation usually with two or more unknown variables, such that only the integer values are considered as the acceptable solutions. Diophantine equations are utilized to checks if there is any data dependence in the codes or not. If there is any

dependency, the next phases will not run. If it does not have the data dependency, the next phases will run. If a loop does not have any data dependence between any iterations (Fig. 1), it can be safely executed in the parallel mode. We can fragment the parts of the serial code and give them to the different processors to compile faster. In *the non-numerical programs*, the other control structures are also important. Here are some examples to realize it better:

```
1.  For ( i=1; i< 1000; i++)
2.      For (j=1; j< I ; j++)
3.          {
4.              For (k=j ; k< I ; k++)
5.                  d=X[i+2k+5, 4k-j]
6.                  X [i-j,i+j]=...
7.          }
```

Fig. 3. Complex nested loop

In this example, there is a probable data dependence between S1(i_1, j_1, k_1) and $S_2(i_2, j_2)$. The equations that must be solved are:

$$i_1 + 2k_1 + 5 = i_2 - j_2$$
$$4k_1 - j_1 = i_2 + j_2 \quad (1)$$

The values of (i_2, j_2, k_2) rise from the solution of the above equations on the basis of the different values of (i_1, j_1, k_1). It is obvious that only the values between loop bounds are acceptable. The admissible values are illustrated in Table I. How to compute iterations that are data depended is a major problem here. Equations resulting from subscripts equality can be solved to resolve this problem, such that the values for computed indices to sink and source dependence are identical. It is essential that the resultant indices be within the corresponding loop bounds. Fig. 3 illustrates the nested loop problem [6], [7], [8], [9], [10], [11], [12], [13].

TABLE I. DEPENDENCY SOURCE AND VECTOR AND SINK

Dependence source (i_1 , j_1 , k_1)	Dependence sink (i_2, j_2)	Distance vector
(16, 1, 12)	(46, 1)	(30, 0)
(16, 1, 13)	(49, 2)	(33, 1)
(16, 1, 14)	(52, 3)	(36, 2)
(16, 1, 15)	(54, 4)	(39, 3)
(16, 1, 16)	(58, 5)	(42, 4)
(16, 3, 16)	(48, 1)	(32, −2)
…	…	…

C. Multi-Population Evolutionary Optimization

Evolutionary Computing is a set of methods proposed to solve the allocation and scheduling problem. Genetic Algorithm (GA) is a popular EC method which can better locate a near-optimal solution than other similar approaches in most cases [22]-[25]. In the procedure of GA, individuals with better solutions have higher possibilities to appear in next generations. The GA-based algorithms generally consist of the following components: Chromosome, genetic population, fitness function, genetic operators including mutation and cross-over. Although GA is a powerful solution, defining a proper fitness function is always challenging and requiring expertise especially when the size of design space is huge. Plus, GA is relatively slow and maybe trapped in local optima [26].

MPGA is a static scheduling strategy, where the execution times of tasks and the data transfer times between tasks are known. MPGA is the parallel version of GA that provides better convergence rate and more speedup compared to single population GA [26]. In addition, MPGA highly reduces the probability of falling into local optima trap.

III. RELATED WORK

Abdulla S. et al. proposed cloud-based compiler [2] with the private cloud implementation. The software compiles the program and returns the output to the user. The software has been provided for the end users by using a SaaS cloud. Their software contains a text editor and a terminal with an option to the users to select the compiled program language. Compared to [2], we proposed a cloud-based super compiler providing more functionality and flexibility.

Ansari [17] proposed an online compiler trying to decrease the portability problems and storage space by using cloud computing. The programmers benefit different cloud-based compilers in order to picking up the fastest and/or the most convenient compiler. Also, the overhead of installing the compiler on each computer is avoided. Although these advantages make application ideal for online conducting compilation/examinations, they do not offer super compilation service.

Patel [20] proposed an online Java compiler. The main goal of this compiler is to provide Java developers with online compiler that write a java program and compile/debug it. Therefore, the client machine doesn't need java development kit (JDK). The aim of this work is to help programmers to reduce the problems of code portability. Like similar previous works, this tool does not support super compilation.

Zefreh et al. [18] proposed a tiling and scheduling nested loops techniques with dependencies by awareness of computational capacity of the processing nodes. In addition, they developed a theoretical model to estimate the parallel execution time of tilled nested loops. Since the tile sizes play a key role to improve the performance of nested loops, they proposed a tiling genetic algorithm using the proposed model to find the near-optimal tile size.

Zefreh et al. [19] addressed the nested loops parallelization on partially connected heterogeneous distributed systems. In addition, they proposed a topology and power-aware tile mapping approach to parallelize nested loops. Besides considering the node's computational power, the exploitation of the network topology has been considered during assigning tiles to processing nodes. Their proposed approach minimizes the execution time by better the load balancing and minimizing the data transferring cost.

Inter-nest data locality is the data locality between a pair of loop nest. Stencil processing pattern is a class of loop nests with

a considerable inter-nest data locality. Seyfari et al. [21] proposed EALB, an automatic approach to optimize inter-nest data locality for the stencils. EALB partitions two "compute" and "copy" loop nests within the time loop nest of the stencils into blocks to be executed interleaved. They used an evolutionary method to determine the optimum block size by using the cache miss rate and the cache eviction rate.

The main step of loop parallelization is finding iterations of the loop that do not exist any data dependence between them [6], [7], [8], [9], [10], [11], [12] and [13].

IV. PROPOSED WORK

The main goal of this work is to present a cloud base super compiler as SaaS to solve the following problems: 1) create an easier and more popular method to generate parallel codes from serial codes by an automatic converting method. 2) Prevent to rewrite a serial code to a parallel code. 3) Helping programmers who cannot work with super compilers to create their parallel codes. 4) Providing a super-optimization method as a cloud-based SaaS to offer an inexpensive solution for the programmers who cannot buy super compilers.

A. System Architecture

In this paper, a cloud service has been simulated by connecting eight PCs together with a switch. A virtual machine (VMWare) has been utilized to connect them such as a cloud center. Thus, the client PC sends its serial code, which contains the nested loop, to the cloud center, next cloud center receives it, and analyzes it to find any data dependency. The Cloud center utilizes the Diophantine equations to find dependent and independent data, and afterward selects independent data to create a parallel code to run them. Fig. 4 illustrates the proposed system architecture.

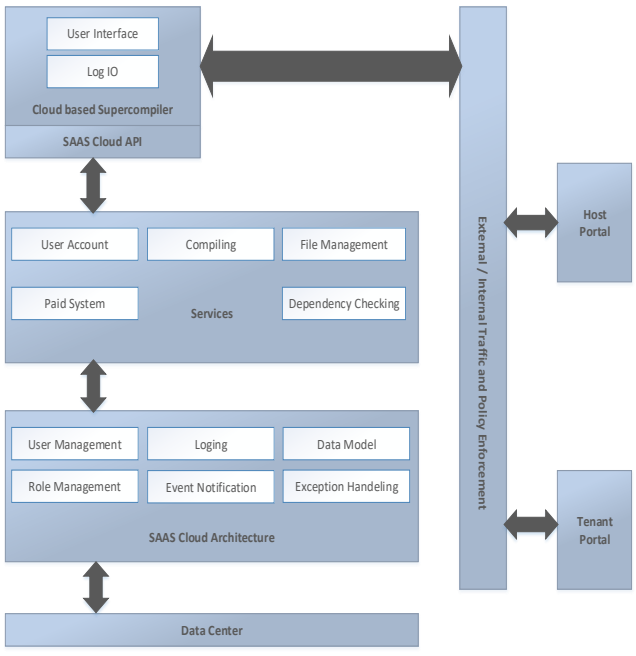

Fig. 4. System architecture

B. General Overview of the Proposed Strategy

The proposed processing paradigm of the cloud-based super compiler is illustrated in Fig. 5. In the first phase, the cloud center receives a serial code from clients (users) that obtains some tasks. In this figure, the tasks with the same colors have data dependency. There is not any data dependency between two tasks with different colors. In the second phase, the cloud base super-optimization finds any data dependency and divides the tasks into the sub groups that they do not have any data dependency. We used Diophantine equations to do this task. Next, we schedule the independent task to the available cores by using MPGA (see Section IV.C). In the final phase, our application generates a parallel code and returns the parallel code to the users. In this phase, we can create a parallel code only with having index of each sets. For example, if we have a set of tasks such as A=$\{T_{17}, T_{18}, T_{19}, T_{20}\}$, then we will have an index set of members of set A, named B= $\{17, 18, 19, 20\}$. We can define a certain processor to run this task one by one. Fig. 6 shows this compilation procedure, where core #1 has been selected to run all the tasks in set A.

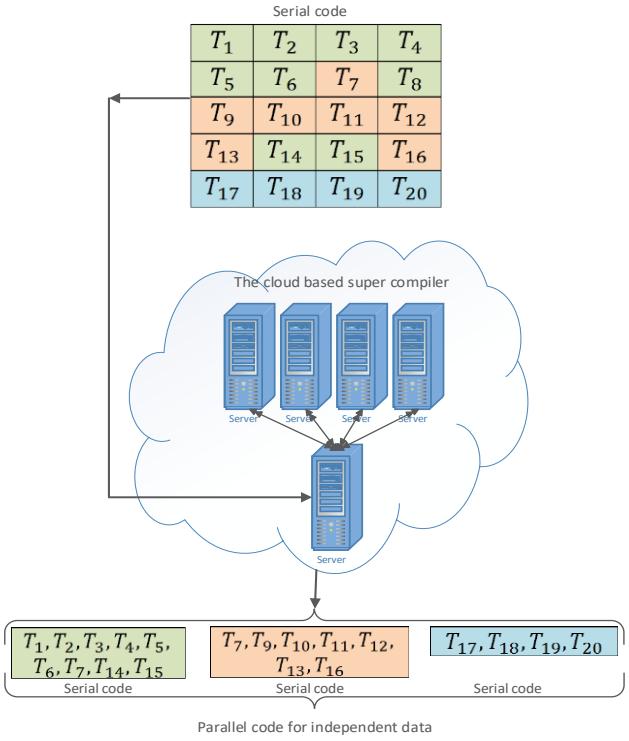

Fig. 5. The cloud-based supercompiler

```
1.  int size;
2.  MPI_Comm_size (MPI_COMM_WORLD, & size);
3.  int rank;
4.  MPI_Comm_rank (MPI_COMM_WORLD, & rank);
5.  If (rank==1)
6.        For (i=all numbers in B)
7.              RUN ($T_i$);
```

Fig. 6. A sample code for parallelizing all tasks in set A.

C. Task Scheduling by Using MPGA

GA is an iterative population-based exploration solution mimicking the process of natural selection and evolution where the characteristics of the process can be utilized in solving optimization problems. All GA-based methods have an initial population where selection, crossover, mutation operators are applied to initial population for producing improved population. The operations will be repeated until satisfying user criteria (reaching suitable results) or stopping after a predefined number of iterations.

Although GA methods can improve the quality of results, using it have some difficulties. First, GA may not converge towards the optimal solutions or even to near-optimal solutions in the case of very huge exploration space. One possible solution is to increase the initial population size but leading to increase the execution time of evolutionary algorithms. Parallelizing these algorithms can remarkably diminish their execution time and improve the quality of results. In the parallelized GA, multiple processors work together where each one runs a simple GA and has an independent population. Fig. 7 represents the flowchart of MPGA. Moreover, the following subsections explain the basic components of MPGA.

1) Encoding

The Tile-to-processor assignment matrix is used for encoding the chromosomes, as proposed by [27]. This matrix shows the processor assignment to the available tiles for the nested loops.

2) Generating Initial Population

The initial population includes random solutions in the design space, where each solution represented by chromosome is a scheduling for all the jobs. The size of initial population depends on the size of design space (see Table II).

3) Fitness Evaluation

Objective function (fitness function) is a metric for comparing different scheduling that satisfy problem constraints. Inspired by [27], Equation (1) represents the fitness functions for evaluating the individual for task scheduling. The goal of *Fitness (s)* is to reduce the execution time of all tiles. Therefore, *makespan (s)* represents the total time duration for executing all current scheduled tiles which is described in Equation (2).

$$Fitness\ (s) = \frac{1}{makespan\ (s)} \tag{1}$$

$$makespan\ (s) = max\ (CompilationTime\ (J)), \quad J \in \text{Tiled IterationSpace} \tag{2}$$

4) Selection Operator

In this paper, we used the Roulette wheel selection and the Tournament selection methods for selecting parent chromosome.

5) Crossover Operator

Inspired by [27], the K-point crossover method or the uniform crossover method is leveraged for crossing the selected pairs with each other.

6) Mutation Operator

The mutation operator is applied through using either the bit-flipping or gene-swap method, inspired by [27].

7) Migration Operator

After a predefined number of iterations, all the populations share their best chromosomes among each other. Step 7 above is specific to the parallel procedure. Sharing the best individuals aids the MPGA to get avoid of local optima.

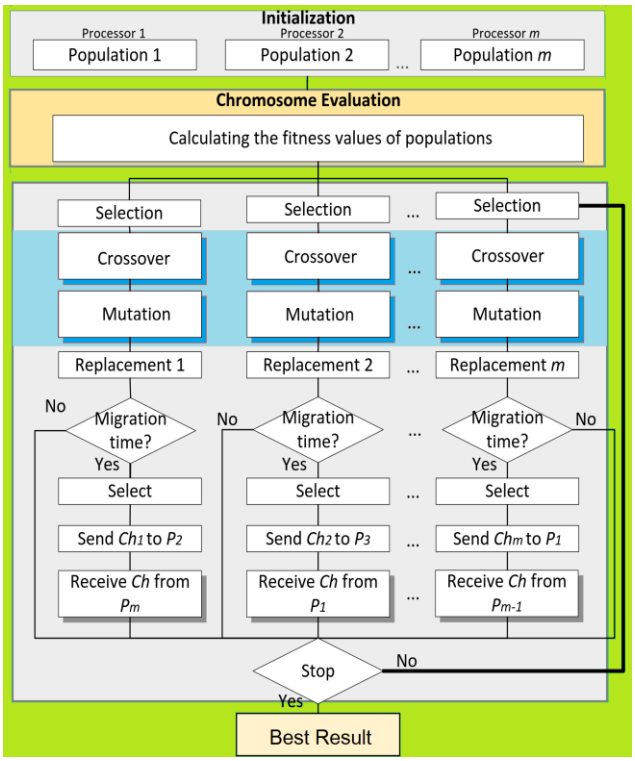

Fig. 7. The flowchart of MPGA

The proposed super-optimization method receives serial code that is written with C++ and transforms this code to parallel code, which is written with C++, and some parallel instruction of MPI library. Fig. 8 illustrates a simple example from a simple loop that has written by Borland C. Fig. 9 illustrates the parallel code of it, with MPI that generates from our proposed method. Fig. 10 illustrates the more complex nested loops in C that we want to parallelize it, and Fig. 11 indicates its parallel code that generate from our cloud-based super compiler.

V. Experimental Results

In this section, we evaluate the quality, the efficiency, and the accuracy of the proposed work. The evaluations are based on the serial of two synthetic benchmarks described in Fig. 8 and Fig. 10. The parallel code is generated by the super-optimization method from the serial code. We achieved the results based on the cluster network with four multi-core processors. The specification of each processor is *Intel Core i7 Processor 2.7 GHz, Windows 7 Home Premium (64-bit),* and *16 GB Memory*. The specification of MPGA parameters is shown in Table II.

To run parallel codes, we use the MPICH2, the MPI codes for the implementation. In the first benchmark, the number of elements in the array is equal to 1000, and in the second benchmark is equal to 10000. The results of the speedup and the efficiency have been presented in Table III. Table III also compares the results of the initial code (serial code) with the final code (parallel code). Also, compare results of the initial code (serial code) with the final code (parallel code). Fig. 12, and Fig. 13 illustrate the achieved the speedup and the efficiency of the proposed SaaS-based super compiler. This indicates that our parallel code has an impressive efficiency, and the suggested method is outstanding. We can claim that our proposed method obtains considerable performance hat is a very suitable choice for users to utilize it on the cloud. Fig. 13 indicates that our method's efficiency will be better by growing up the size of problem (by increasing the loop iterations).

TABLE II. MPGA PARAMETERS

Parameter	Value
Migration Rate	10
Crossover	0.8
Mutation Rate	0.2
Population	250
Maximum # Iterations	10000

```
1.  for (i = 0; i < n; i++)
2.          A[i] = 2 * A[i];
```

Fig. 8. A serial code of an independent loope

```
1.  MPI_Init ( NULL , NULL ) ;
2.  int size ;
3.  MPI_Comm_size ( MPI_COMM_WORLD , &size ) ;
4.  int rank ;
5.  MPI_Comm_rank ( MPI_COMM_WORLD , & rank )
;
6.  int p;
7.  p = n / size;
8.  if ( rank != ( size - 1 ) )
9.      for ( i = rank * p ; i < ( rank * p ) + p ; i++ )
10.         A[i] = 2 * A[i];
11. else
12.     for (i = rank * p ; i < n ; i++ )
13.         A[i] = 2 * A[i];
14. MPI_Finalize ();
```

Fig. 9. The derived parallel code from the serial code on Fig. 8

```
1.  for (i = 0; i < n; i++)
2.      for (j = 0; j < m; j++)
3.          A[i][j] = A[i][j] + 1;
```

Fig. 10. The serial code of the indepence nested loop

```
1.  MPI_Init (NULL, NULL);
2.  int size;
3.  MPI_Comm_size (MPI_COMM_WORLD, & size);
4.  int rank;
5.  MPI_Comm_rank (MPI_COMM_WORLD, & rank);
6.  int p;
7.  p = n / size;
8.  if ( rank != ( size -1 ) )
9.      for (i = rank * p; i< (rank * p) + p; i++)
10.         for (j = 0; j < m; j++)
11.             A[i][j] = A[i] + 1;
12. else
13.     for (i = rank * p; i< n; i++)
14.         for (j = 0; j <m; j++)
15.             A[i][j] = A[i] + 1;
16. MPI_Finalize ();
```

Fig. 11. Parallel code derived from serial code on Fig. 10.

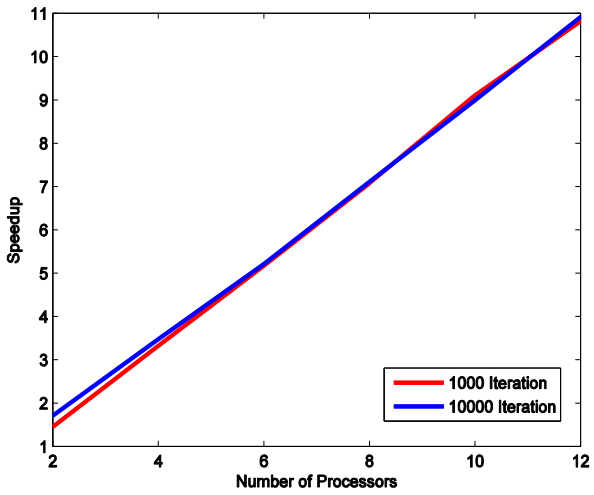

Fig. 12. Speedup Diagram.

TABLE III. SPEEDUP AND EFFICENCY VALUES

Number of elements	Number of processors	Serial time	Parallel Time	Speed up	Efficiency
10000	2	1.26	0.82	1.52	0.76
100000	2	8.45	5.21	1.62	0.81

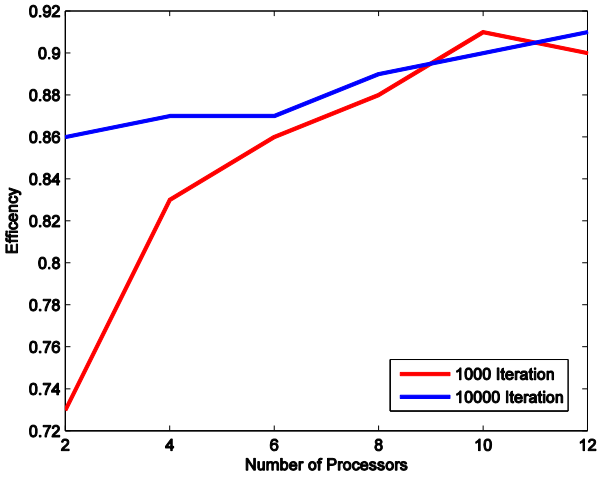

Fig. 13. Efficiency Diagram.

VI. Conclusion

Multi-core platforms have been developed, however, we cannot utilize the whole provided processing capacity since implementing and verification of common sequential codes are highly time consuming and expensive. There have existed several reasons that cause to produce this problem, like difficulty of writing parallel programs, the inability of beginner programmers, the high cost of rewriting the serial programs to the parallel mode, the complexity of using the super compilers, and the high price of the super compilers. To tackle these challenges, we present an economical service provided as a cloud-base SaaS. The proposed service is a customized super compiler that automatically detect the independent parts of the code and schedule/run them on different processing cores. Since the most complexity of the programs is in their nested loops, if we parallelize nested loops, we can achieve considerable performance. The output is a generated parallel code by MPI commands. The evaluating results shows the effectiveness of the solution of the speedup and code efficiency.

References

[1] T. Rauber and G. Riinger, "Parallel Programming for Multicore and Cluster Systems," Springer- Verlag Berlin Heidelberg , 2013.

[2] S. Abdulla , S. Iyer and S. kutty "Cloud Based Compiler" International Journal of Students Research in Technology & Management. Vol. 1, No. 3, pp. 308-322, 2013.

[3] K. K. Lavania, Y. Sharma, C. Bakliwal, "A Review on Cloud Computing Model" International Journal on Recent and Innovation Trends in Computing and Communication. Vol.1, pp. 161-163, 2013.

[4] B. Parhami, "Introduction to Parallel Processing Algorithms and Architectures" Kluwer Academic Publishers, 2002.

[5] S. G. Akl, "The Design and Analysis of Parallel Algorithms" A Division of Simon & Schuster, Englewood Cliffs. New Jersey , 1989.

[6] F. Baiard, D. Guerri, P. Mori, and L. Ricci, "Evaluation of a Virtual Sshared Memory Machine by the Compilation of Data Parallel Loops" 8th Euromicro Workshop on Parallel and Distributed Processing, pp. 309-316, IEEE, 2000.

[7] D.K. Chen and P.Ch. Yew. "On Eeffective Execution of Non-Uniform Doacross Loops" IEEE Transaction on Parallel and Distributed System, Vol. 7, pp. 463-476, IEEE, 1996.

[8] C. Eisenbeis and J.C. Sogno, "A General Algorithm for Data Dependence Analysis" In International Conference on Supercomputing—Washington, pp. 1–28, July 19–23, 1992.

[9] D.E. Maydan, J.L. Hennessy, and M.S. Lam, "Efficient and Exact Data Dependence Analysis" Conference on Programming Language Design and Implementation, Toronto, Ontario, Canada, pp. 1–10, June 26–28, 1991

[10] Ch.T. Wu, Ch.T. Yang, and Sh.Sh. Tseng, "PPD: A Practical Parallel Loop Detector for Parallelizing Compilers" International Conference on Parallel and Distributed System, pp. 280–281, Tokyo, 1996.

[11] Ch.T. Yang, Sh.Sh. Tseng, M.H. Hsieh, Sh.H. Kao, and M.F. Jiang. "Run-Time Parallelization for Partially Parallel Loops" International Conference on Parallel and Distributed Systems, pp. 308–309, Seoul, 1997.

[12] H. Zima and B. Chapman, "Super Compilers for Parallel and Vector Computers" Addison-Wesley, 1991.

[13] S.Parsa and Sh. Lotfi, "A New Approach to Parallelization of Serial Nested Loops Using Genetic Algorithms," The Journal of Supercomputing, Springer,36, 83–94, 2006.

[14] R. V. Ahmadabadi and A. Majd, "Cloud Based Super Compiler, " 4^{th} International conference on information technology management, comunication and computer, 233-239, Tehran, Iran, 2014.

[15] Robison, Arch, Michael Voss, and Alexey Kukanov. "Optimization via reflection on work stealing in TBB." In 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1-8. IEEE, 2008.

[16] Mitchell, Neil. "Rethinking supercompilation." In ICFP, vol. 10, pp. 309-320. 2010.

[17] Ansari, Aamir Nizam, Siddharth Patil, Arundhati Navada, Aditya Peshave, and Venkatesh Borole. "Online C/C++ compiler using cloud computing." In 2011 International Conference on Multimedia Technology, pp. 3591-3594. IEEE, 2011.

[18] Zefreh, Ebrahim Zarei, Shahriar Lotfi, Leyli Mohammad Khanli, and Jaber Karimpour. "Tiling and Scheduling of Three-level Perfectly Nested Loops with Dependencies on Heterogeneous Systems." Scalable Computing: Practice and Experience 17, no. 4 (2016): 331-350.

[19] Zefreh, Ebrahim Zarei, Shahriar Lotfi, Leyli Mohammad Khanli, and Jaber Karimpour. "Topology and computational-power aware tile mapping of perfectly nested loops with dependencies on distributed systems." Journal of Parallel and Distributed Computing (2019).

[20] Patel, Mayank. "Online java compiler using cloud computing." International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN (2013): 2278-3075.

[21] Seyfari, Yousef, Shahriar Lotfi, and Jaber Karimpour. "Optimizing inter-nest data locality in imperfect stencils based on loop blocking." The Journal of Supercomputing 74, no. 10 (2018): 5432-5460.

[22] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):11320.

[23] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. Proceedings of IEEE conference on electronics, information and systems; 2004.

[24] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 2004;15(9):82434.

[25] Majd, Amin, et al. "NOMeS: Near-optimal meta-heuristic scheduling for MPSoCs." Computer Architecture and Digital Systems (CADS), 2017 19th International Symposium on. IEEE, 2017.

[26] Majd, Amin, Golnaz Sahebi, Masoud Daneshtalab, Juha Plosila, Shahriar Lotfi, and Hannu Tenhunen. "Parallel imperialist competitive algorithms." Concurrency and Computation: Practice and Experience 30, no. 7 (2018): e4393.

[27] Hajieskandar, A., Shahriar Lotfi, and Simin Ghahramanian. "Two Level Nested Loops Tiled Iteration Space Scheduling By Changing Wave-Front Angles Approach." International Journal of Advanced Research in Computer and Communication Engineering 1, no. 3 (2012): 126-133.