A Performance-Based Parallel Loop Self-Scheduling on Grid Computing Environments

Wen-Chung Shih[1], Chao-Tung Yang[*2], and Shian-Shyong Tseng[1,3]

[1]Department of Computer and Information Science, National Chiao Tung University
Hsinchu 300, Taiwan, R.O.C.
{gis90805, sstseng}@cis.nctu.edu.tw

[2]High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University
Taichung 407, Taiwan, R.O.C.
ctyang@thu.edu.tw

[3]Department of Information Science and Applications
Asia University
Taichung 413, Taiwan, R.O.C.
sstseng@asia.edu.tw

Abstract. Efficient loop scheduling on parallel and distributed systems depends mostly on load balancing, especially on heterogeneous PC-based cluster and grid computing environments. In this paper, a general approach, named Performance-Based Parallel Loop Self-Scheduling (PPLSS), was given to partition workload according to performance of grid nodes. This approach was applied to three types of application programs, which were executed on a testbed grid. Experimental results showed that our approach could execute efficiently for most scheduling parameters when estimation of node performance was accurate.

Keywords. Parallel loops, Loop scheduling, Self-scheduling, Grid computing, Globus, MPI

1 Introduction

A promising approach to parallel computing is grid computing, which utilizes heterogeneous computers through the Internet to compute [2, 5, 6]. Traditional schemes for parallel loop scheduling include static scheduling and dynamic scheduling [8]. While the former might incur load imbalancing on heterogeneous environments, the latter has not been investigated thoroughly on grid environments.

Self-scheduling is a major class of dynamic loop scheduling schemes. Well-known self-scheduling schemes include Pure Self-Scheduling (PSS), Chunk Self-Scheduling

* Corresponding author.

(CSS), Guided Self-Scheduling (GSS) [9], Factoring Self-Scheduling (FSS) [7], and Trapezoid Self-Scheduling (TSS) [10]. These schemes partition work load according to a simple formula, not considering performance of processors.

In [11], a method (α self-scheduling) is proposed to improve well-known self-scheduling schemes. Although this scheme partition work load according to CPU clock speed of processors, CPU could not completely represent performance of processors. In [12], an approach is proposed to adjust α scheduling parameter, but performance is still estimated only by CPU speed. In [4], a class of self-scheduling schemes is extended to heterogeneous distributed systems.

In this paper, we address the performance estimation issue in parallel loop scheduling, and propose a general approach called Performance-Based Parallel Loop Self-Scheduling (PPLSS). This approach estimates the performance ratio of each node to partition loop iterations. For verification, this approach is applied to three types of application programs.

We organize the rest of this paper as follows. Section 2 describes the background about parallel loop self-scheduling schemes. Next, our approach is presented in section 3. In section 4, our system configuration is specified and experimental results on three application programs are also reported. Finally, the conclusion is given in the last section.

2 Background

In this section, related work on self-scheduling schemes is described. First, we review several well-known self-scheduling schemes. Next, two recently proposed schemes are introduced.

2.1 Well-Known Self-scheduling Schemes

Traditional self-scheduling schemes operate in common. At each step, the master assigns some amount of loop iterations to an idle slave. These schemes differ in the way how the master computes the amount to next idle slave. The well-known schemes include PSS, CSS, GSS, FSS and TSS. Table 1 shows the different chunk sizes for a problem with the number of iteration N=1536 and the number of processor p=4.

Table 1. Sample partition size

Scheme	Sample partition size
PSS	1, 1, 1, 1, 1, 1, 1, 1, 1, …
CSS(125)	125, 125, 125, 125, 125, 125, 125, 125, 125, …
FSS	192, 192, 192, 192, 96, 96, 96, 96, 48, …
GSS	384, 288, 216, 162, 122, 91, 69, 51, 39, …
TSS	192, 180, 168, 156, 144, 132, 120, 108, 96, …

2.2 Schemes for Cluster and Grid Environments

In [11], the authors revise known loop self-scheduling schemes for extremely heterogeneous PC-cluster environments. The algorithm is divided into two phases. In phase one, α% of workload is partitioned according to CPU clock of processors. Then, the rest of workload is scheduled according to some well-known self-scheduling in the second phase.

In [3, 12], a new scheme for heterogeneous grid computing environments is proposed. This scheme is still a two-phased approach. However, it can adjust the α scheduling parameter according to the relative heterogeneity of the environment.

3 Performance-based Parallel Loop Self-Scheduling (PPLSS)

In this section, the concept of performance estimation is presented first. After that, the algorithm of our approach is described.

3.1 Performance Estimation

We propose to estimate performance of each grid node, and assign work load to each node accordingly. In this paper, our performance function (PF) for node j is defined as

$$PF_j = w \times \frac{1/T_j}{\sum_{\forall node_i \in S} 1/T_i} \quad (1)$$

where

- S is the set of all grid nodes.
- T_i is the execution time (sec.) of node i for some application program, such as matrix multiplication.
- w is the weight of this term.

The performance ratio (PR) is defined to be the ratio of all performance functions. For instance, assume the PF of three nodes are 1/2, 1/3 and 1/4. Then, the PR is 1/2 : 1/3 : 1/4; i.e., the PR of the three nodes is 6 : 4 : 3. In other words, if there are 13 loop iterations, 6 iterations will be assigned to the first node, 4 iterations will be assigned to the second node, and 3 iterations will be assigned to the last one.

3.2 Algorithm

The algorithm of our approach is modified from [11], and master program and slave program are listed as follows.

Module MASTER

```
Gather performance ratio of all slave nodes
r = 0;
for (i = 1; i < number_of_slaves; i++) {
```

```
    partition α% of loop iterations according to the
  performance ratio;
    send data to slave nodes;
    r++;
  }
  Partition (100-α)% of loop iterations into the task
  queue using some known self-scheduling scheme
  Probe for returned results
  Do {
       Distinguish source and receive returned data
       If the task queue is not empty then
             Send another data to the idle slave
             r -- ;
           else
             send TAG = 0 to the idle slave
  } while (r > 0)
  END MASTER
```

Module SLAVE

```
  Probe if some data in
  While (TAG > 0) {
         Receive initial solution and size of subtask
  work and compute to fine solution
       Send the result to the master
       Probe if some data in
  }
  END SLAVE
```

4 Experimental Results

In this section, our grid configuration is presented. Then, experimental results for matrix multiplication, Mandelbrot and circuit satisfiability are shown respectively.

4.1 Grid Environments

The testbed grid includes three clusters which are located in three universities respectively. Cluster 1, located in Providence University, has five nodes. One of the nodes is designated as the master node. Cluster 2, located in Hsiuping Institute of Technology, has four nodes. Cluster 3, located in Tunghai University, also has four nodes. We use the following middleware to build the grid:

- Globus Toolkit 3.0.2
- Mpich library 1.2.6

For readability of experimental results, the naming of our implementation is listed in Table 2.

Table 2. Description of our implementation for all programs

AP	Name	Description
Matrix Multiplication, Mandelbrot, and Circuit Satisfiability	G(F, T)SS	Dynamic scheduling G(F, T)SS
	NG(F, T)SS	Fixed α scheduling + G(F, T)SS
	PG(F, T)SS	Our scheduling + G(F, T)SS

4.2 Application 1: Matrix Multiplication

The matrix multiplication is a fundamental operation in many applications. In this subsection, we investigate how scheduling parameters influence performance. In the experiment as shown in Fig. 1(a), we find NGSS get best performance when α= 50. Therefore, this value is adopted for the next experiment. Fig. 1(b) illustrates the result for α= 50. Although both NGSS and our PGSS seem to perform well the same, PGSS is not restricted by the selection of α value. In other words, PGSS is more robust.

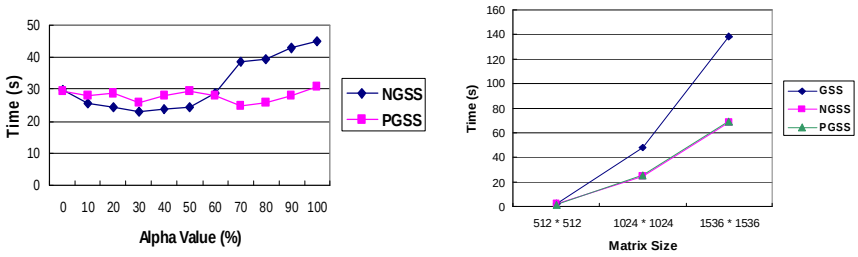

Fig. 1. (a)Execution time for different alpha values (b) Execution Time of Matrix Multiplication with GSS

Fig. 2(a) illustrates the result for α= 30. Although FSS, NFSS and our PFSS seem to perform well the same, PFSS is not restricted by the selection of α value. In other words, PFSS is more robust. Fig. 2(b) illustrates the result for α= 30. Although TSS, NTSS and our PTSS seem to perform well the same, PTSS is not restricted by the selection of α value. In other words, PTSS is more robust.

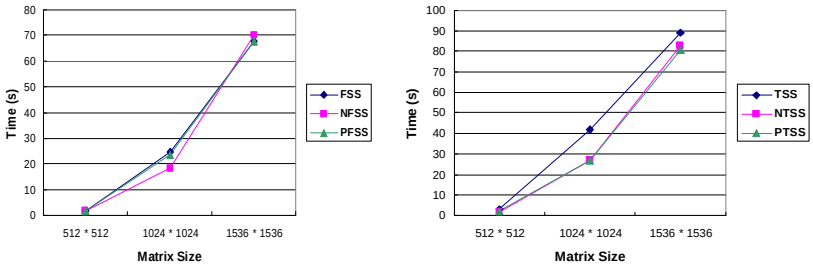

Fig. 2. (a) Execution Time of Matrix Multiplication with FSS (b) Execution Time of Matrix Multiplication with TSS

For application of Matrix Multiplication, experimental results show that our performance-based approach is efficient and robust.

4.3 Application 2: Mandelbrot

The Mandelbrot set is a problem involving the same computation on different data points which have different convergence rates [1]. In this subsection, we investigate how scheduling parameters influence performance. In the experiment as shown in Fig. 3(a), we find NGSS get best performance when α= 50. Therefore, this value is adopted for the next experiment. Fig. 3(b) illustrates the result for α= 50. Although both NGSS and our PGSS seem to perform well the same, PGSS is not restricted by the selection of α value. In other words, PGSS is more robust.

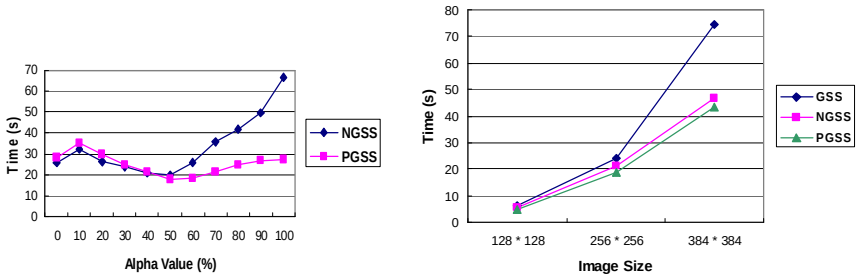

Fig. 3. (a)Execution time for different alpha values (b) Execution Time of Mandelbrot with GSS

Fig. 4(a) illustrates the result for α= 50. Although FSS, NFSS and our PFSS seem to perform well the same, PFSS is not restricted by the selection of α value. In other words, PFSS is more robust. Fig. 4(b) illustrates the result for α= 50. Although TSS, NTSS and our PTSS seem to perform well the same, PTSS is not restricted by the selection of α value. In other words, PTSS is more robust.

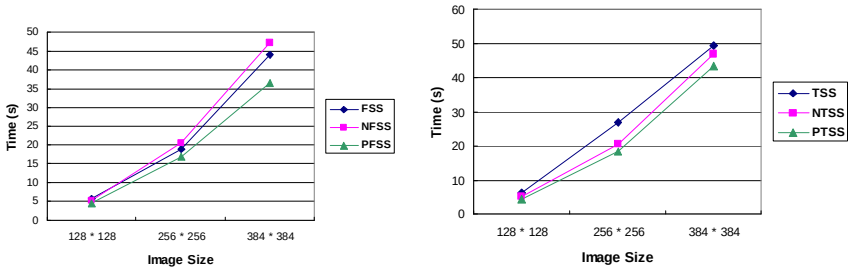

Fig. 4. (a) Execution Time of Mandelbrot with FSS (b) Execution Time of Mandelbrot with TSS

For application of the Mandelbrot set, experimental results show that our performance-based approach is efficient and robust.

4.4 Application 3: Circuit Satisfiability

The circuit satisfiability problem is one involving a combinational circuit composed of AND, OR, and NOT gates. In this subsection, we investigate how scheduling parameters influence performance. In the experiment as shown in Fig. 5(a), we find NGSS get best performance when α= 50. Therefore, this value is adopted for the next experiment. Fig. 5(b) illustrates the result for α= 50. Although both NGSS and our PGSS seem to perform well the same, PGSS is not restricted by the selection of α value. In other words, PGSS is more robust.

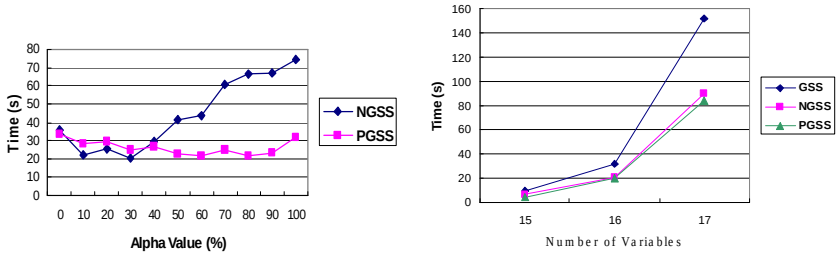

Fig. 5. (a)Execution time for different alpha values (b) Execution Time of Circuit Satisfiability with GSS

Fig. 6(a) illustrates the result for α= 50. Although FSS, NFSS and our PFSS seem to perform well the same, PFSS is not restricted by the selection of α values. In other words, PFSS is more robust. Fig. 6(b) illustrates the result for α= 50. Although TSS, NTSS and our PTSS seem to perform well the same, PTSS is not restricted by the selection of α value. In other words, PTSS is more robust.

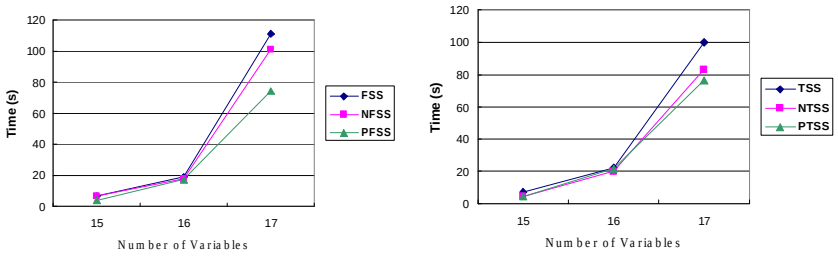

Fig. 6. (a) Execution Time of Circuit Satisfiability with FSS (b) Execution Time of Circuit Satisfiability with TSS

For application of the Circuit Satisfiability problem, experimental results show that our performance-based approach is efficient and robust.

5 Conclusions and Future Work

We have proposed a performance-based parallel loop self-scheduling (PPLSS) approach, which partitions work load according to performance ratio of grid nodes. It has been compared with previous algorithms by experiments on three types of application programs. In each case, our approach can obtain performance improvement on previous schemes. Besides, our approach is less sensitive to α values than previous schemes; in other words, it is more robust. In our future work, we will implement more types of application programs to verify our approach. Furthermore, we hope to find better ways of modeling the performance function, incorporating network information.

References

1. Introduction To The Mandelbrot Set, http://www.ddewey.net/mandelbrot/
2. What Is Grid Computing, http://www-1.ibm.com/grid/about_grid/what_is.shtml/
3. Kuan-Wei Cheng, Chao-Tung Yang, Chuan-Lin Lai, and Shun-Chyi Chang, "A Parallel Loop Self-Scheduling on Grid Computing Environments," *Proceedings of the 2004 IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 409-414, KH, China, May 2004.
4. A. T. Chronopoulos, R. Andonie, M. Benche and D.Grosu, "A Class of Loop Self-Scheduling for Heterogeneous Clusters," *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, pp. 282-291, 2001.
5. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, pp. 181-194, August 2001.
6. I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, 55(2):42-47, 2002.
7. S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: a method scheme for scheduling parallel loops," *Communications of the ACM*, vol. 35, 1992, pp. 90-101.
8. H. Li, S. Tandri, M. Stumm and K. C. Sevcik, "Locality and Loop Scheduling on NUMA Multiprocessors," *Proceedings of the 1993 International Conference on Parallel Processing*, vol. II, pp. 140-147, 1993.
9. C. D. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. on Computers*, vol. 36, no. 12, pp. 1425-1439, 1987.
10. T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, 1993, pp. 87-98.
11. Chao-Tung Yang and Shun-Chyi Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *Journal of Information Science and Engineering*, vol. 20, no. 2, pp. 263-273, March 2004.
12. Chao-Tung Yang, Kuan-Wei Cheng, and Kuan-Ching Li, "An Efficient Parallel Loop Self-Scheduling on Grid Environments," *NPC'2004 IFIP International Conference on Network and Parallel Computing, Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, Hai Jin, Guangrong Gao, Zhiwei Xu (Eds.), vol. 3222, pp. 92-100, Oct. 2004.