An Efficient Parallel Loop Self-Scheduling on Grid Environments

Chao-Tung Yang[1], Kuan-Wei Cheng[1], and Kuan-Ching Li[2]

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

[2]Parallel and Distributed Processing Center
Dept. of Computer Science and Information Management
Providence University
Shalu, Taichung, 433 Taiwan, R.O.C.
kuancli@pu.edu.tw

Abstract. The approaches to deal with scheduling and load balancing on PC-based cluster systems are famous and well-known. Self-scheduling schemes, which are suitable for parallel loops with independent iterations on cluster computer system, they have been designed in the past. In this paper, we propose a new scheme that can adjust the scheduling parameter dynamically on an extremely heterogeneous PC-based cluster and grid computing environments in order to improve system performance. A grid computing environment consists of multiple PC-based clusters is constructed using Globus Toolkit and SUN Grid Engine middleware. The experimental results show that our scheduling can result in higher performance than other similar schemes.

Keywords. Parallel loops, Self-scheduling, PC-based clusters, Grid Computing

1. Introduction

Parallel computers are increasingly widespread, and nowadays, many of these parallel computers are no longer shared-memory multiprocessors, but follow the distributed memory model due to scalability factor. These systems consist of homogeneous workstations, where all these workstations have processors, memory and cache memory with exactly identical specifications. Nowadays, more and more systems are composed of homogeneous and clustered together with a number of heterogeneous workstations, where they may have similar or different architectures, speed, and operating systems. For this reason, first of all we have to do is to distinguish whether the target system is homogeneous or heterogeneous. Therefore, we define a frame of relativity to decide the cluster system to two typical cases comparatively, say relatively homogeneous and relatively heterogeneous.

After the system architecture is clear, the next starting point is the task analysis. As we know, the major source of program parallelization is loop. If the loop iterations can be distributed to different processors as evenly as possible, the parallelism within loop iterations can be exploited. Loops can be roughly divided into four kinds, as shown in Figure 1: uniform workload, increasing workload, decreasing workload, and random workload loops. They are the most common ones in programs, and should cover most case. In a relatively homogeneous case, workload can be partitioned proportionally by computing power respectively to each working computer, but in relatively heterogeneous case, this method will not work. The self-scheduling scheme works well not only in moderate heterogeneous cluster environments but also in extremely heterogeneous environment where the performance difference between the fastest computer and the slowest computer is large.

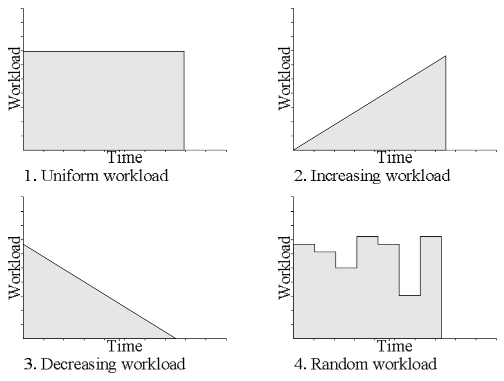


Figure 1. Four kinds of loop style

In this paper, we revise known loop self-scheduling schemes to fit both homogeneous and heterogeneous PC clusters environment. The HINT Performance Analyzer [2] is given for a help to distinguish whether the target system is relatively homogeneous or relatively heterogeneous. Afterwards we partition loop iteration styles by four different ways according to the cluster system typical cases for achieving good performance in any possible executive environment. In this paper, we propose a new scheme that can adjust the scheduling parameter dynamically on an extremely heterogeneous PC-based cluster and grid computing environments in order to improve system performance. A grid computing environment consists of multiple PC-based clusters is constructed using Globus Toolkit and SUN Grid Engine middleware. The experimental results show that our scheduling can result in higher performance than other similar schemes.

2. Background

2.1. Self-Scheduling

Self-scheduling is a large class of adaptive/dynamic centralized loop scheduling schemes. In a common self-scheduling scheme, p denotes the number of processors, N denotes the total iteration and f() is a function to produce the chunk-size at each step. At the *i-th* scheduling step, the master computes the chunk-size C_i and the remaining number of tasks R_i,

$$R_0=N, \qquad C_i=f(i,p), \qquad Ri=R_{i-1}-C_i$$

where *f()* possibly has more parameters than just *i* and *p*, such as R_{i-1}. The master assigns C_i tasks to an idle slave and the load imbalancing will depend on the execution time gap between t_j, for j=1, ⋯, p [7].

2.2. The α Self-Scheduling Scheme

In the previous scheduling paper [1], α% partition of workload was according to their performance weighted by CPU clock in the first phase and the rest (100-α)% of workload according to known self-scheduling in the second phase. The experimental results were conducted on a PC cluster with six nodes and the fastest computer is 7.5 times faster than the slowest ones in CPU-clock cycle. Many various α values are applied to the matrix multiplication and a best performance is obtained with α=75. Thus, our approach is suitable in all applications with regular parallel loops. Through α Self-Scheduling Scheme, we get three new improved self-scheduling schemes; From FSS, GSS, TSS, so called **NFSS**, **NGSS**, and **NTSS** [1], where N means “new” here.

3. Methodology

The adjustment of scheduling parameters dynamically and fit multiform system architectures to accomplish our system has been implemented. Later, we combined Grid computing technology, the HINT Performance Analyzer, our α self-scheduling scheme, and the dynamic adjustment of scheduling parameters into a whole new approach.

3.1. System Definition

System definition is the first step in our approach. The HINT Performance Analyzer [2] is given for helping us to distinguish whether the target system is relatively homogeneous or relatively heterogeneous. We gather CPU performance capabilities, amounts of memory, cache sizes, and basic system performance by HINT. An updatable library, called System Information Array (**SIA**), is build to record the collection of the information. Define the two Cluster System Typical Cases as follows:

Gather CPU Information, P_1, $P_2 \ldots P_n$,

Assume P_1 is the node that has the worst performance (working ability) of all.

Say, $P_n = r^n P_1$

Partition α% of workload according to their performance weighted by CPU clock and the rest (100-α)% of workload according to known self-scheduling scheme.

(1) Define *Heterogeneous Ratio* (**HR**), HR= $\frac{p_1}{p_n} \approx \frac{MinQUIPS}{MaxQUIPS} \approx \frac{1}{r^n} < \alpha'/100$, where α’ is the temporary value of α.

(2) Case 1: If α’ < HR, then we say the target system is relatively heterogeneous case.
Case 2: If α’ > HR, then we say the target system is relatively homogeneous case.

(3) If the target system is relatively heterogeneous system, we start the α self-scheduling scheme with $\alpha = \alpha'$ %
If the target system is relatively homogeneous, then we run the HINT benchmark to build (and update) the SIA, and start the α self-scheduling scheme with α =100 %

There is still a point for attention: not always update the SIA before each time of job submission, only when the system has one or more new nodes added, SIA-update will be needed and α will be properly adjusted.

3.2. Loop Styles Analysis

For the programs with regular loops, intuitively, we may want to partition problem size according to their CPU clock in heterogeneous environment. However, the CPU clock is not the only factor which affects computer performance. Many other factors also have dramatic influences in this aspect, such as the amount of memory available, the cost of memory accesses, and the communication medium between processors, etc [5]. Using this intuitive approach, the result will be degraded if the performance prediction is inaccurate. A computer with largest inaccurate prediction will be the last one to finish the assigned job.

Loops can be roughly divided into four kinds, as shown in figure 1: uniform workload, increasing workload, decreasing workload, and random workload loops. They are the most common ones in programs, and should cover most cases. These four kinds can be classified two types: regular and irregular. The first kind is regular and the last three ones are irregular. Different loops may need to be handled in different ways in order to get the best performance. Since workload is predictable in regular loops, it is not necessary to process load balancing at beginning.

We propose to partition problem size in two stages. At first stage, partition α% of total workload according to their performance weighted by CPU clock. In the way, the communication between master and slaves can be reduced efficiently. At second stage, partition following (100-α) % of total workload according to known self-scheduling scheme. In the way, load balancing can be archived. This approach can be suitable for all regular loops. An appropriate α value will lead to good performance.

Furthermore, dynamic load balancing approach should not be aware of the run-time behavior of the applications before execution. But in *GSS* and *TSS*, to achieve good performance, computer performance of each computer in the cluster has to be in order in extreme heterogeneous environment, which is not very applicable. With our schemes, this trouble will not exist. In this paper, the terminology “*FSS*-80” stand for “α=80, and remainder iterations use *FSS* to partition” and so on.

Example 1

Suppose that there is a cluster consisting of five slaves. Each of computing nodes has CPU clock of 200MHz, 200MHz, 233MHz, 533MHz, and 1.5GHz, respectively. Table 1 shows the different chunk sizes for a problem with the number of iteration *I*=2048 in this cluster. The number of scheduling steps is parenthesized.

Table 1. Sample partition size of Example 1

GSS	410, 328, 262, 210, 168, 134, 108, 86, 69, 55, 44, 35, 28, 23, 18, 14, 12, 9, 7, 6, 5, 4, 3, 2, 2, 2, 1, 1, 1, 1 (N=30)
GSS-80	923, 328, 144, 123, 121, 82, 66, 53, 42, 34, 27, 21, 17, 14, 11, 9, 7, 6, 4, 4, 3, 2, 2, 1, 1, 1, 1, 1 (N=28)
FSS	205, 205, 205, 205, 205, 103, 103, 103, 103, 103, 51, 51, 51, 51, 51, 26, 26, 26, 26, 26, 13, 13, 13, 13, 13, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1 (N=43)
FSS-80	923, 328, 144, 123, 121, 41, 41, 41, 41, 41, 21, 21, 21, 21, 21, 10, 10, 10, 10, 10, 5, 5, 5, 5, 5, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1 (N=39)
TSS	204, 194, 184, 174, 164, 154, 144, 134, 124, 114, 104, 94, 84, 74, 64, 38 (N=16)
TSS-80	923, 328, 144, 123, 121, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 1 (N=23)

To model our approach, we use following terminology:

- *T* is the total workload of all iterations in a loop.

- W is the α% of total workload.
- b is the fewest workload in an increasing/decreasing workload loop. It can be the workload of the first iteration (in an increasing workload loop) or the workload of the last iteration (in a decreasing workload loop).
- h is the different of workload between consequence iterations. h is a positive integer.
- x is the iteration number on which the α % accumulating workload is reached. x is positive real.

3.3. System Modeling

In our new parallel loop self-scheduling scheme, the HINT Performance Analyzer help us to decide the cluster system for two typical cases comparatively, and the next we must have proper reaction and appropriate self scheduling scheme processed on which system architecture and loop style are changeable. Parallel loop style analysis is essential since parallel loops can be roughly divided into four kinds, as shown in Figure 1: uniform workload, increasing workload, decreasing workload, and random workload loops. They should be the most common ones in programs, and should cover most cases. Moreover, we implement the adjustment of scheduling parameters dynamically to fit multiform system architectures, and message passing interface (MPI) directives parallelizing code segment to be executed by multiple CPUs which is so called cluster. In the loop parallelism region, our self-scheduling scheme must be hand inserted into source code in the region where the largest possible loops that may be parallelized. An example of how our new self-scheduling scheme works is shown in Figure 2.

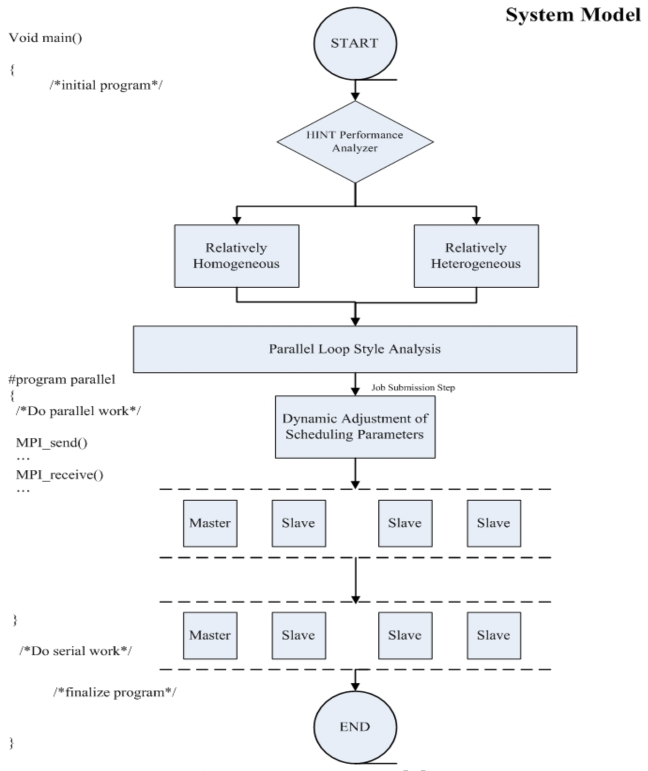


Figure 2. System model.

4. Experimental Results

4.1. Hardware and Software Configuration

Our Grid architecture is implemented on top of Globus Toolkit, name grid-cluster. It is built three PC clusters to form a computational grid environment (Figure 3).

- Alpha site: Four PCs, each PC has two AMD Athlon MP2000 processors, 512MB DDRAM and Intel PRO100VE NIC.
- Beta site: Four PCs, each PC has one Intel Celeron 1.7GHz processor, 256MB DDRAM, and 3Com 3c9051 NIC.
- Gamma site: Four PCs, each PC has two Intel P3 866 MHz processors, 256MB SDRAM and 3Com 3c9051 NIC.

SGE QMaster daemon is run on the master node of each PC cluster, and SGE execute daemon is run to manage and monitor incoming job and Globus Toolkit v2.4. Each slave node is running SGE execute daemon to execute income job only. The operating system is RedHat Linux release 9. Parallel application we use MPICH-G2 v1.2.5 for message passing.

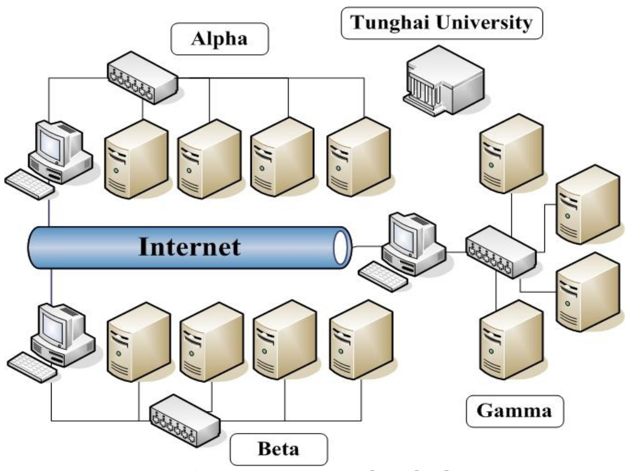


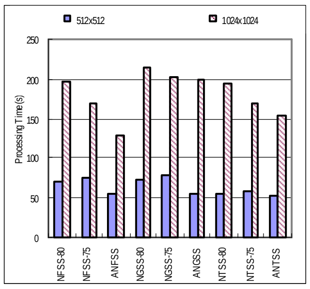
Figure 3. THU Grid testbed

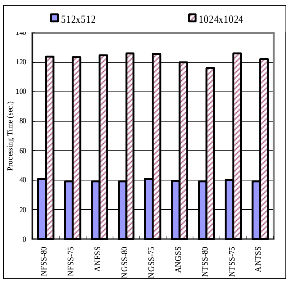
4.2. Experimental Results

4.2.1. Regular Workload

The experiment consists of three different scenarios: (1) Differences performance presentation of scheduling schemes in uniform workload. (2) Different grid environment and (3) Matrix multiplication with different matrix sizes. At first step, we run a MPI program on different grid system to evaluate the system performance. Second step, we connect these grid systems together to form a grid environment (In our testbed is grid Alpha, Beta and Gamma) Then, running the same MPI program to evaluate the system performance. Third step, through the different system topologies, we connect the system characteristics together for a performance analysis. Finally, we run the same MPI program to evaluate the system performance of different system architectures. Our new scheme can guarantee whether what kind of parallel loop scheduling situation happen, they can be properly well-arranged in our approach and achieved better performance than other scheme developed before, all of the performance analysis are presented in Figures 4, 5, and 6.

Figures 4, 5, and 6 note that our approach connects these grid systems together to form a grid environment (In our testbed is grid Alpha, Beta and Gamma) Then, running the same MPI program to evaluate the system performance and implements FSS, GSS, and TSS group approach. In previous methods, NFSS, NTSS, and NGSS get worse performance than new scheme with dynamic parameterization and systematic adjustment automatically.





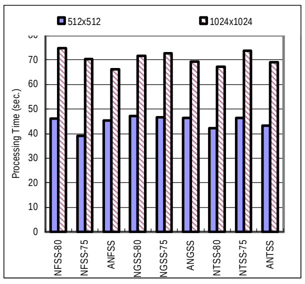


Figure 4 **Figure 5** **Figure 6**

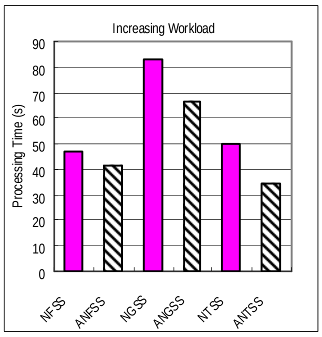
Figure 4. A chart of execution time of different sizes of matrix multiplication by grid $\alpha+\beta+\gamma$.

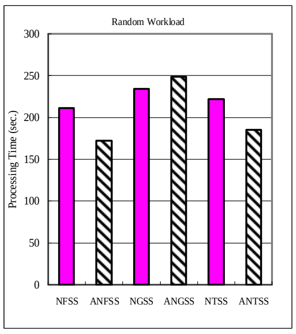
Figure 5. A chart of execution time of different sizes of matrix multiplication by grid β.

Figure 6. A chart of execution time of different sizes of matrix multiplication by grid $\beta+\gamma$.

4.2.2. Irregular Workload

The experiment consists of three scenarios: Differences performance presentation of scheduling schemes in (1) Increasing workload. (2) Decreasing workload and (3) Random workload. Fig 7, 8, 9, note that execution time of simulated increasing, random, and decreasing workload loop by various self-scheduling approaches grid $\alpha+\beta+\gamma$.





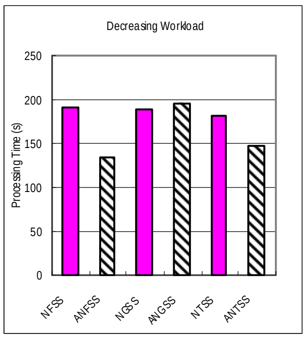


Figure 7 **Figure 8** **Figure 9**

Figure 7. A chart of execution time of simulated increasing workload loop by various self-scheduling approaches grid $\alpha+\beta+\gamma$.

Figure 8. A chart of execution time of simulated random workload loop by various self-scheduling approaches grid $\alpha+\beta+\gamma$.

Figure 9. A chart of execution time of simulated decreasing workload loop by various self-scheduling approaches grid $\alpha+\beta+\gamma$.

5. Conclusion and Future Work

In this paper, we can find that Grid Computing technology certainly can bring more computing performance than the traditional PC Cluster or SMP system. Moreover, we try to draw up and integrate a nice and complete system implemented on parallel loop self-scheduling. The system can guarantee whether what kind of parallel loop scheduling situation happen, they can be properly well-arranged in our system and achieved better performance than other scheme developed before. We revise known loop self-scheduling schemes to fit both homogeneous and heterogeneous PC clusters and Grid environment when loop style is regular or irregular. After enough feedback information has been investigated, collected, and analyzed, the performance will well-improved in each time of feedback information collection and job submission. Now we combine Grid Computing technology, the HINT Performance Analyzer, our α self-scheduling scheme, and the dynamic adjustment of scheduling parameters into a whole new approach successfully. The goal of achieving good performance on parallel loop self-scheduling by our approach is definitely practicable. The appropriate method to investigate the performance trend after the new computing nodes added and the proper way to adjust the value of α are our future work.

References

1. Chao-Tung Yang and Shun-Chyi Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *Lecture Notes in Computer Science*, vol. 2600, Springer-Verlag, pp. 1079-1088, P.M.A. Sloot, D. Abramson, A.V. Bogdanov, J.J. Dongarra, A.Y. Zomaya, Y.E. Gorbachev (Eds.), June 2003.
2. T. H. Tzen and L.M. Ni, "Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers," *IEEE Trans. on Parallel and Distributed Systems*, Vol 4, No 1, Jan. 1993, pp 87 - 98.
3. Christopher A. Bohn, Gary B. Lamont, "Load Balancing for Heterogeneous Clusters of PCs," *Future Generation Computer Systems*, 18 (2002) 389–400.
4. E. Post, H. A. Goosen, "Evaluating the Parallel Performance of a Heterogeneous System," *Proceedings of HPCAsia2001.*
5. 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, 1993, pp. 140-147.
6. 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
7. P. Tang and P. C. Yew, "Processor self-scheduling for multiple-nested parallel loops," *Proceedings of the 1986 International Conference on Parallel Processing* , 1986, pp. 528-535.
8. Yun-Woei Fann, Chao-Tung Yang, Shian-Shyong Tseng, and Chang-Jiun Tsai, "An intelligent parallel loop scheduling for multiprocessor systems," *Journal of Info. Science and Engineering - Special Issue on Parallel and Distributed Computing,* vol. 16, no. 2, pp. 169-200, March 2000.
9. S. F. Hummel, E. Schonberg, L. E. Flynn, "Factoring, a Scheme for Scheduling Parallel Loops," *Communications of the ACM*, Vol 35, No 8, Aug. 1992.
10. C. D. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. on Computers*, Vol 36, Dec. 1987, pp 1425 - 1439.
11. I. Foster, C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure,* Morgan Kaufmann; 1st edition (January 1999)
12. *A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems.* I. Foster, N. Karonis. Proc. 1998 SC Conference, November, 1998.