Hierarchical Two-Dimensional Guided Loop Self-Scheduling for Distributed Systems

Satish Penmatsa[1] **and Akash Laddha**[2]
[1]Department of Computer Science, Framingham State University, Framingham, MA, USA
[2]Staples Inc., Framingham, MA, USA

Abstract – *Efficient scheduling of large loops inside computation intensive applications can significantly improve their performance on distributed computing systems. With nested loop constructs, multiple levels of partition may need to be considered for further reducing the application execution time. With increasing number of processors for executing the distributed applications, multiple levels of master-worker processor architecture may need to be considered to improve the scalability of the loop scheduling schemes. In this paper, we implement a hierarchical distributed two-dimensional guided self-scheduling (HDGSS-2D) scheme whose objective is to minimize the total execution time of an application by a load balanced allocation of the loop iterations of nested loops among available processors. HDGSS-2D is implemented and its performance evaluated using the Stampede high performance computing cluster at the Texas Advanced Computing Center of the University of Texas at Austin.*

Keywords: Loop scheduling, hierarchical architecture, distributed systems.

1 Introduction

Distributed computing systems such as the cloud computing systems are emerging as viable platforms for executing computation-intensive applications [1]. Parallel loop constructs in these applications when scheduled for concurrent execution on the distributed computing resources (processors) can significantly improve the performance of these applications [2]. In static loop scheduling, a scheduler processor usually pre-computes the set of loop iterations (chunks) to be executed by the available processors and allocates them. In dynamic or self-scheduling, idle processors in the system request the scheduler for new chunks to be executed by them during run time [5].

The computing resources in distributed systems may be heterogeneous with varying processor speeds and memory capacities. Distributed loop scheduling schemes takes this heterogeneity into account when making allocation decisions. Also, loops can be nested with multiple levels. Scheduling schemes may consider partitioning of only the outermost loop of a nested loop structure when computing the loop chunks (one-dimensional loop scheduling) or compute the chunks by considering multiple levels of the nested loop construct (multi-dimensional loop scheduling) [8].

The different ways of computing the loop chunks has given rise to different loop scheduling schemes. Examples include: *Chunk self-scheduling* (the chunk size is determined by the user), *Pure self-scheduling* (chunk size is set to 1), *Fixed-size self-scheduling* (total loop iterations are divided by the number of processors to determine the fixed chunk size), *Trapezoid self-scheduling* (chunk sizes are decreased using a decrement that is computed based on a fixed first and last chunk, and the total number of chunks), *Factoring self-scheduling* (multiple rounds of scheduling with the same chunk size in each scheduling step), and *Guided self-scheduling* (has a non-linear chunk size function with large initial chunks to reduce communication/scheduling overheads). Please see [3, 4, 5, 6, 7, and 9] and references there-in for further details.

Two-dimensional versions of the Trapezoid, Factoring, and Guided self-scheduling schemes have been presented in [8, 13] and references there-in. Experimental results showed that the two-dimensional scheduling schemes provide a more load balanced allocation of tasks to the processors compared to the one-dimensional schemes.

With a single processor acting as the scheduler (master processor) for allocating chunks to the available (worker) processors and gathering results, issues related to scalability, communication/synchronization overheads, and fault tolerance may occur. A hierarchical structure with a lower level consisting of worker processors, several superior levels of master processors, and a super-master at the top level (of the hierarchy) would be an option for a solution to the above mentioned issues.

Hierarchical versions of distributed one-dimensional Trapezoid, Factoring, and Guided loop self-scheduling schemes for large-scale clusters and cloud systems have been presented and analyzed in [1, 2]. Results showed that the hierarchical schemes exhibit good scalability.

In this paper, we implement a hierarchical *two-dimensional* distributed Guided self-scheduling scheme (HDGSS-2D) with two-levels of master processors and compare its performance with DGSS-2D [13] (with one master processor). We consider parallel loops without dependencies among loop iterations (DOALL loops). The schemes are implemented and their performance compared using the *Stampede* high performance computing cluster [10] at the *Texas Advanced Computing Center* of the *University of Texas at Austin*.

2 Two-Dimensional Distributed Guided Self-Scheduling

In this section, we review the two-dimensional distributed Guided self-scheduling scheme (DGSS-2D) presented in [13]. Guided Self – Scheduling (GSS) [6, 9] is a dynamic scheme with a non-linear chunk-size function. It assigns large chunks (set of iterations) initially, which implies reduced communication/scheduling overheads in the first scheduling steps. A modified version GSS(*l*) with minimum assigned chunk-size *l* attempts to improve on the weaknesses of GSS.

One-dimensional distributed Guided self-scheduling scheme (DGSS-1D) partitions only the outermost loop of a nested loop construct. Two-dimensional distributed Guided self-scheduling scheme (DGSS-2D) partitions both the outer loop and the inner loop of a two-level nested loop construct. The above schemes were implemented using *Master-Worker* architecture [5, 8].

In the following, the DGSS-2D algorithm is presented. The methodology for computing the two-dimensional chunks is similar to the one described in [8]. The two-dimensional chunks will be allocated to the worker processors (PEs) by the master PE based on the worker *available powers* [5]. A worker with higher available power will be allocated more chunks than compared to a worker with lower available power.

ALGORITHM: Two-Dimensional Distributed Guided Self-Scheduling Scheme (DGSS-2D)

MASTER

1. (a) Receive processor speeds (P_j) from the worker PEs ($j = 1,\ldots,p$).

   (b) Compute processor *Available (Virtual) Powers*, V_j using worker PE workloads.

   (c) Send V_j to the worker PEs.

2. (a) While there are unassigned iterations, if a request comes, put it in a queue.

   (b) Compute the rectangular chunks and *istart1*, *istart2* [8] using DGSS-1D.

   (c) Pick a request from queue with virtual powers V_j and assign next V_j rectangular chunk along same or adjacent wavefront diagonals.

WORKER

1. (a) Send processor speed (P_j) to the Master PE.

   (b) Receive Virtual Power (V_j) from Master PE

2. Send a request for work (chunks of loop iterations).

3. (a) Wait for a response from Master.

   (b) If more tasks arrive, compute the new task, and go to Step 2. Else, *Terminate*.

3 Hierarchical Two-Dimensional Distributed Guided Self-Scheduling

In hierarchical self-scheduling, instead of making one master process (processor) responsible for all the workload (chunks) distribution, multiple master processes are introduced [1, 2]. For example, a three-level hierarchical structure (with two-levels of master processors) would have all the worker processors at the bottom, a set of master processors to oversee the set of workers (by allocating chunks and collecting results) at the next higher level, and a super-master to oversee the masters at the top level.

Figure 1 presents the hierarchical structure of HDGSS-2D with two levels of master processes. The worker processes (W) are at the bottom. The super-master uses DGSS-2D to allocate chunks (loop iterations) to the master processes. The master processes in turn allocate chunks to the worker processes (using DGSS-2D). The master processes do not perform any computation but assign tasks to the workers from the pool of tasks they obtain from the super-master.

Initially, the super-master computes the available (virtual) powers of the masters based on their worker processing speeds. When a request from a master comes to the super-master, if there are any unassigned iterations (rectangular (2D) chunks) (computed based on DGSS-2D), the next rectangular chunk will be computed and sent to the master based on its virtual power. Similarly, when a request from a worker comes to the master, if there are any unassigned iterations (from those obtained from the super-master), the next rectangular chunk will be computed and sent to the worker based on its virtual power. Any results that are computed by the workers will be sent to the masters who in turn send them to the super-master (aggregation of results).

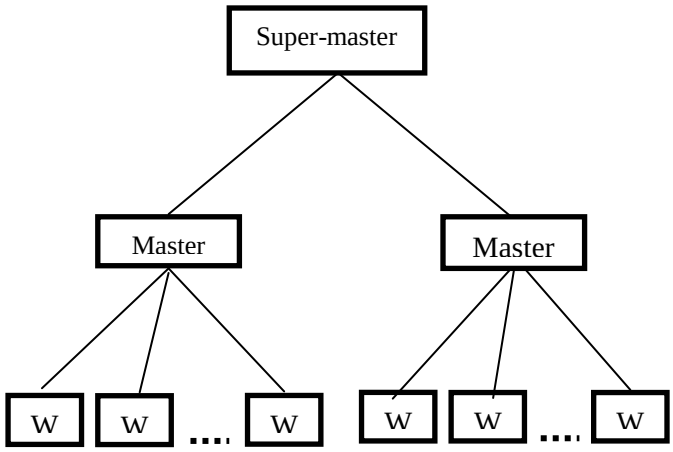

Figure 1: Hierarchical DGSS-2D (Two levels of masters)

4 Implementation and Results

The non-hierarchical and hierarchical two-dimensional Guided loop self-scheduling schemes (DGSS-2D and HDGSS-2D respectively) were implemented using the Message Passing Interface [11] on the Stampede [10] high performance computing cluster at the Texas Advanced Computing Center of the University of Texas at Austin. The test problem used for the experiments is the Mandelbrot Computation [12]. The Mandelbrot Computation is a doubly-nested loop without any dependencies. The schemes are implemented with the number of worker processors ranging from 16 to 64 and the Mandelbrot computation sizes ranging from 16000 x 16000 to 64000 x 64000.

To create a heterogeneous environment, we put an artificial load (one continuously running matrix multiplication process) in the background on half of the worker processors (similar to [13]). The workers with one load in the background are assumed to have virtual power of 1 and the workers without any load are assumed to have virtual power of 2. Thus, we have half fast and half slow worker processors.

In the following, we present the experimental results for various problem sizes and number of worker processors. All timings are in seconds.

Figure 2 presents the total execution time for computing the test problem using DGSS-2D and HDGSS-2D for a problem size of 16000 x 16000 with number of worker processors ranging from 16 to 64. It can be observed that the hierarchical scheduling scheme performs better than the non-hierarchical one in terms of the total execution time. For example, when the number of worker processors is 16, the total execution time of HDGSS-2D is about 10% less than that of DGSS-2D and when the number of worker processors is 32, the total execution time of HDGSS-2D is about 20% less than that of DGSS-2D. The performance improvement is around 25% when the number of worker processors is increased to 64.

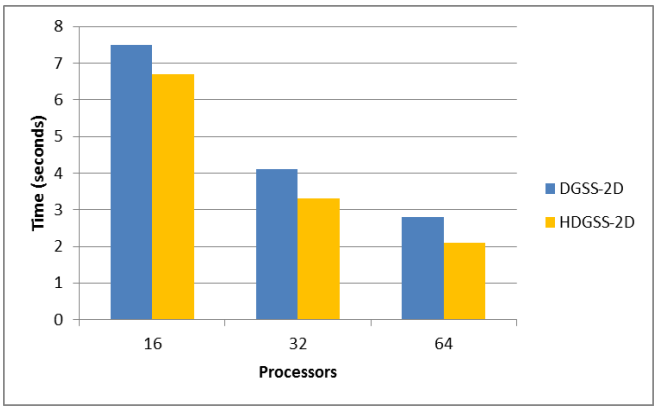

Figure 2: Problem Size – 16000 x 16000

Figure 3 presents the total execution time when using DGSS-2D and HDGSS-2D for a problem size of 24000 x 24000 with varying number of processors. It can be observed that there is a considerable reduction in the total execution time with HDGSS-2D compared to that of DGSS-2D. For example, when the number of worker processors is 64, the total execution time of HDGSS-2D is about 25% less than that of DGSS-2D. HDGSS-2D super-master distributes the work to the master processes that helps decentralize the chunk distribution and reduces the queuing/communication time between the processors to better load balance the application.

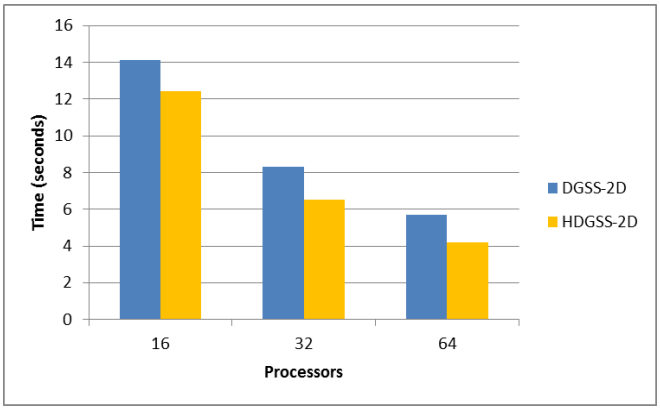

Figure 3: Problem Size – 24000 x 24000

In Figure's 4 and 5, we present the total execution time for computing the test problem when using the guided scheduling schemes for problem sizes of 32000 x 32000 and 64000 x 64000 respectively. The performance improvement with HDGSS-2D can again be observed. For example, for a problem size of 32000 x 32000 and when the number of worker processors is 32, the total execution time of HDGSS-2D is about 21% less than that of DGSS-2D, and for a problem size of 64000 x 64000 and when the number of worker processors is 32, the total execution time of HDGSS-2D is about 25% less than that of DGSS-2D. For a problem size of 32000 x 32000 and when the number of worker processors is 64, the total execution time of HDGSS-2D is

about 26% less than that of DGSS-2D, and for a problem size of 64000 x 64000 and when the number of worker processors is 64, the total execution time of HDGSS-2D is about 29% less than that of DGSS-2D.

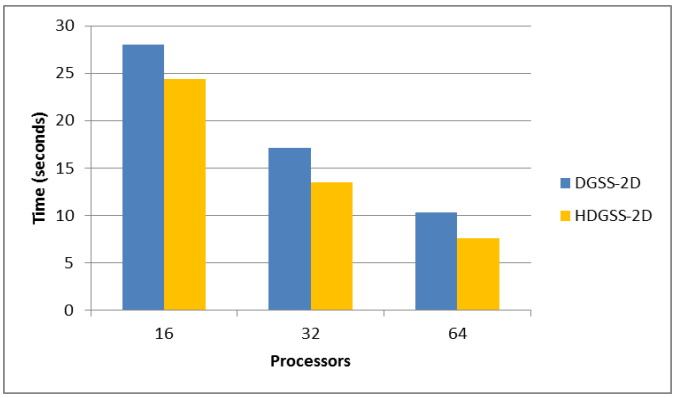

Figure 4: Problem Size – 32000 x 32000

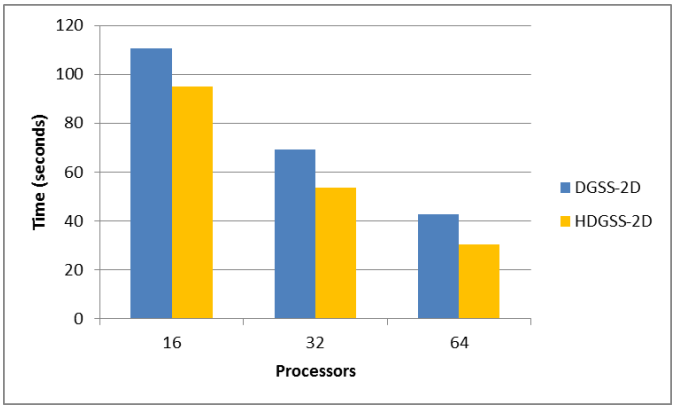

Figure 5: Problem Size – 64000 x 64000

5 Conclusions

In this work, we implemented a hierarchical distributed two-dimensional Guided loop self-scheduling scheme (HDGSS-2D) and compared its performance with the non-hierarchical DGSS-2D. Based on the experimental results, it was observed that HDGSS-2D performs better then DGSS-2D and the total execution time of the test problem is reduced by about 29% for larger problem sizes and increasing number of worker processors. The hierarchical structure decentralizes the workload distribution and reduces the queuing/communication time between the worker and master processes. In future work, we plan to further test the scalability of the hierarchical scheme with increasing number of worker processors and more number of master processor levels.

6 References

[1] Y. Han and A. T. Chronopoulos, "A hierarchical distributed loop self-scheduling scheme for cloud systems", 12th IEEE Intl. Symp. on Network Computing and Applications, pp. 7 – 10, 2013.

[2] Y. Han and A. T. Chronopoulos, "Scalable loop self-scheduling schemes implemented on large-scale clusters", IEEE International Symposium on Parallel & Distributed Proc., Workshops and Phd Forum, pp. 1735 – 1742, 2013.

[3] I. Banicescu, V. Velusamy, and J. Devaprasad, "On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring", Cluster Computing, vol. 6, pp. 215–226, 2003.

[4] A. Kejariwal, A. Nicolau, and C. Polychronopoulos, "History-aware self-scheduling", Intl. Conference on Parallel Processing, Columbus OH, Aug 2006, pp. 185–192.

[5] A. T. Chronopoulos, S. Penmatsa, J. Xu, and S. Ali, "Distributed loop-scheduling schemes for heterogeneous computer systems", Concurrency and Computation: Practice and Experience, vol. 18, no. 7, pp. 771–785, 2006.

[6] C. D. Polychronopoulos and D. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers", IEEE Trans. on Computers, 1987; 36:1425–1439.

[7] J. Herrera, E. Huedo, R. S. Montero, and I. M. Llorente, "Loosely-coupled loop scheduling in computational grids", in Proc. of the 20th IEEE Intl. Parallel and Distributed Processing Symp., Greece, pp. 25-29 April 2006.

[8] A. T. Chronopoulos, L. M. Ni, and S. Penmatsa, "Multidimensional dynamic loop scheduling algorithms", in IEEE International Conference on Cluster Computing, Austin, TX, 17-20 Sept. 2007, pp. 241 – 248.

[9] T. L. Freeman, D. J. Hancock, J. M. Bull, and R. W. Ford, "Feedback guided scheduling of nested loops", Proc. of the 5th International Applied Parallel Computing (PARA) Workshop, Bergen, Norway, 2000 (Lecture Notes in Computer Science, vol. 1947), Springer: Berlin, 2001; 149–159.

[10] https://www.tacc.utexas.edu/systems/stampede

[11] P. Pachecho, "Parallel Programming with MPI", Morgan Kauffman, 1997.

[12] M. F. Bransley, R. L. Devaney, B. B. Mandelbrot, H. O. Peitgen, D. Saupe, R. F. Voss, Y. Fisher, and M. McGuire, "The science of fractal images", NY: Springer-Verlag, 1988.

[13] S. Penmatsa and A. Laddha, "Distributed two-dimensional guided loop self-scheduling for heterogeneous computer systems", 21st International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), July 27-30, Las Vegas, NV, USA, 2015.