Scheduling Divisible Workloads Using the Adaptive Time Factoring Algorithm[⋆]

Tiago Ferreto and César De Rose

Catholic University of Rio Grande do Sul (PUCRS),
Faculty of Informatics, Porto Alegre, Brazil
{ferreto, derose}@inf.pucrs.br

Abstract. In the past years a vast amount of work has been done in order to improve the basic scheduling algorithms for master/slave computations. One of the main results from this is that the workload of the tasks may be adapted during the execution, using either a fixed increment or decrement (*e.g.* based on an arithmetical or geometrical ratio) or a more sophisticated function to adapt the workload. Currently, the most efficient solutions are all based on some kind of evaluation of the slaves' capacities done exclusively by the master. We propose in this paper the Adaptive Time Factoring scheduling algorithm, which uses a different approach distributing the scheduling between slaves and master. The master computes, using the Factoring algorithm, a time slice to be used by each slave for processing, and the slave predicts the correct workload size it should receive in order to accomplish this time slice. The prediction is based on a performance model located on each slave which is refined during the execution of the application in order to provide better predictions. We evaluated the proposed algorithm using a synthetic testbed and compared the obtained results with other scheduling algorithms.

1 Introduction

Load balancing has been an ongoing issue for decades. Algorithms based on list-scheduling which manage a list of ready to execute tasks that are sent to slave processors are mainly used because of their suitability to dynamically evolving computations, and also because they cope with heterogeneous resources, since when one processor has finished his work it simply gets more work from the list. This is a simply way to automatic compensate for the differences in the performance of the slaves.

A vast amount of work has been done in order to improve the basic algorithms for master/slave computations. One of the main features concerning load balancing that resulted from this is that the workload of the tasks may be adapted during the execution, using either a fixed increment or decrement (*e.g.* based on an arithmetical or geometrical ratio) or a more sophisticated function to adapt the workload. We present a briefly review of some of these techniques in Section 2.

Yet the solutions presented are all based on some evaluation by the master of the slaves' capacities and of the tasks workload. This implies a significant overhead since the master has to maintain some kind of information about its slaves. We present in

[⋆] This research was done in cooperation with HP-Brazil.

this paper the Adaptive Time Factoring scheduling algorithm, which uses a different approach distributing the scheduling between slaves and master. The master computes, using the Factoring algorithm, a time slice to be used by each slave for processing, and the slave predicts the correct workload size it should receive in order to accomplish this time slice. The prediction is based on a performance model located on each slave which is refined during the execution of the application in order to provide better predictions.

In this paper we review in Section 2 some scheduling algorithms used for master/slave applications with a brief state of the art for each one. Section 3 presents our algorithm and the way each slave can evaluate its capacities. In order to validate our algorithm we devised a synthetic small testbed and Section 4 shows the measurement results that we have obtained using the algorithm proposed in comparison to other algorithms. At last we draw some conclusions about our contribution.

2 Related Work

We present below some classic self-scheduling algorithms proposed in the literature. Self-scheduling [1] represents a large class of dynamic centralized loop scheduling methods. These methods divide the total workload based on a specific distribution, providing a natural load balancing to the application during its execution. We present also some adaptive algorithms that add extensions to the classic self-scheduling algorithms in order to support heterogeneity and adaptability. They consider the load variation in the system environment and adjust the size of the chunks delivered to each processor dynamically. This class of algorithms presents a good performance on dynamic and heterogeneous environments based on its ability to adapt itself to the changes in the environment during the execution of an application.

The Pure Self-scheduling or Workqueue scheduling algorithm divides equally the workload in several chunks. A processor obtains a new chunk whenever it becomes idle. Due to the scheduling overhead and communication latency incurred in each scheduling operation, the overall finishing time may be greater than optimal.

The Fixed-size Chunking scheduling algorithm [2] proposes that each processor receives chunks with size K each time it becomes idle. Although it is hard to determine the best K value in realistic applications due to the high number of dependable variables, the authors give an approximation for an acceptable fixed chunk-size K (using *Pth* order statistics to model the last P chunks).

The Guided Self-scheduling algorithm [3], schedules large chunks initially, implying reduced communication/scheduling overheads in the beginning, but at the last steps too many small chunks are assigned generating more overhead [1]. Each time a processor requests for more work, the algorithm assigns to it a chunk of size equal to the size of the remaining workload divided by the total number of processors being used for the computation.

Factoring [4] was specifically designed to handle iterations with execution-time variance. With factoring, iterations are scheduled in batches of P equal-sized chunks. The total size of the chunk per batch is a fixed ratio (α) of the remaining workload, *i.e.* *Remaining_Workload /* $\alpha *$ *Number_Of_Processors*.