A Lower Bound for Dynamic Scheduling of Data Parallel Programs

Fabricio Alves Barbosa da Silva[2], Luis Miguel Campos[1], Isaac D. Scherson[1,2]

[1] Information and Comp. Science, University of California, Irvine, CA 92697 U.S.A.
{isaac,lcampos}@ics.uci.edu***
[2] Université Pierre et Marie Curie, Laboratoire ASIM, LIP6, Paris, France.†
fabricio.silva@asim.lip6.fr‡

Abstract. Instruction Balanced Time Slicing (IBTS) allows multiple parallel jobs to be scheduled in a manner akin to the well-known gang scheduling scheme in parallel computers. IBTS however allows for time slices to change dynamically and, under dynamically changing workload conditions is a good non-clairvoyant scheduling technique when the parallel computer is time sliced one job at a time. IBTS-parallel is proposed here as a dynamic scheduling paradigm which improves on IBTS by allowing also dynamically changing space sharing of the computer's processors. IBTS-parallel is also non-clairvoyant and it is characterized under the competitive ratio metric. A lower bound on its performance is also derived.

1 Introduction

A solution to the dynamic parallel job scheduling problem is proposed together with its complexity analysis. The problem is one of defining how to share, in an optimal manner, a parallel machine among several parallel jobs. A job is defined as a set of data parallel threads. One important characteristic that makes the problem dynamic, as defined in [3], is the possibility of arrival of new jobs at arbitrary times. In the static case, the set of jobs to be executed is already defined when scheduling decisions are made, and arbitrary job arrivals are not permitted.

In this paper we propose a new scheduling algorithm dubbed *IBTS-Parallel*, which is derived from the IBTS algorithm. IBTS stands for instruction balanced time slicing, and was originally defined in [6]. IBTS is a non-clairvoyant [3] scheduling algorithm designed to optimize the competitive ratio (CR) metric, also defined in [6][4].

In addition to the theoretical analysis of IBTS-Parallel, we present an experimental analysis performed using a general purpose event driven simulator developed by our research group.

*** Supported in part by the Irvine Research Unit in Advanced Computing and NASA under grant #NAG5-3692.
† Professor Alain Greiner is gratefully acknowledged.
‡ Supported by Capes, Brazilian Government, grant number 1897/95-11.

2 Previous Work

Various classifications for the parallel job scheduling problem have been suggested in the literature. The classification used here is based on the way in which computing resources are shared : temporal sharing, also known as time slicing or preemption; and space slicing, also called partitioning. These two classifications are orthogonal, and may lead to a taxonomy based on the possible options. Table 1 shows the scheduling policies adopted by commercial and research systems, and was borrowed from [2]. It is worth noting that the lack of consensus on which scheduling policy is best among those showed in table 1 is total. The problem is that the assumptions leading to and justifying the different schemes are usually quite different, which makes difficult the comparison between different solutions.

			time slicing			
			yes			no
			independent PEs		gang scheduling	
			global queue	local queue		
space slicing	yes	flexible	Mach	Paragon/service Meiko/timeshare KSR/interactive transputers Tera/streams Chrysalis	Medusa Butterfly@LLNL Cray T3E Meiko/gang Paragon/gang SGI/gang Tera/PB MAXI/gang	IBM SP2, Victor Meiko/batch Paragon/slice KSR/bath 2-level/bottom TRAC, MICROS Amoeba
		structured		NX/2 on iPSC/2 nCUBE	CM-5 Cedar DHC on SP2 DQT on RWC-1	Cray T3D CM-2 PASM hypercubes
	no		IRIX on SGI NYU Ultra Dynix 2-level/top Hydra/C.mmp	Star OS Psyche Elxsi AP1000	MasPar MP2 Alliant FX/8 Chagori on K2	Illiac IV MPP GF11 Warp

Fig. 1. Scheduling policies followed in current commercial and research systems

We address the scheduling problem by using the same assumptions (i.e. model and metrics) described by Subramaniam and Scherson in [6][4]. In [6], Subramaniam and Scherson studied a scheduling policy named Instruction Balanced Time Slicing (IBTS), which performs well under the competitive ratio metrics [4]. Each job is allocated on all P processing elements (PE) of the machine for a finite quantum (time-slice). All time-quanta devoted to the various jobs are equal as measured in machine instructions (that is, all jobs are preempted after equal number of machine instructions).

Note that time quanta may vary in absolute value while still being equal when measured in number of instructions. A good analysis of gang scheduling, which is similar to IBTS but imposes an invariant constant time slice can be found in [5].

In [4] the programming model used is the data parallel (V-RAM) model[1]. The scheduling problem was defined as a optimization problem, and the *compet-*

itive ratio was the metric used as the objective function. The competitive ratio (CR) was defined in function of the non clairvoyance of the scheduler.

The competitive ratio (CR) is based on the *happiness* [4] concept. The happiness metric attempts to capture the satisfaction of a job as a function of the scheduling decisions made by a scheduler. The CR is then the ratio between the happiness achieved by a knowledgeable *malicious adversary*, and the that achieved by a partially ignorant scheduler. IBTS is hence a non-clairvoyant algorithm, and the malicious adversary always has more information that the scheduler itself. It is that hidden information that is used by the adversary to keep the happiness as low as possible. By minimizing the CR the scheduler approaches the result that would be obtained by an adversary with global knowledge.

$$CR = \overset{max}{y'} \frac{Happiness(x, y')}{Happiness(x, y)} \tag{1}$$

In equation 1, x represents the input to the algorithm, y is the result obtained by the scheduler and y' is the result obtained by the adversary.

Also:

$$Happiness(x, y) = \frac{\int_0^T min_j \wp^j(\tau) d\tau}{T} \tag{2}$$

Where $\wp^j$ is the power delivered to job j, and T is the time at which the last job completes. In analogy with physics terminology, the power delivered to a job is defined as:

$$\wp = \frac{W}{\Gamma} \tag{3}$$

Γ is the running time of a job as a function of the number of statements and the mean time completion of each statement (the mean time completion is computed according to the type of the statement: local statements, remote access statements, etc, and is machine architecture dependent). W is the *work*, or the processor-time product in the ideal world, i.e. it is the product of the ideal number of processors for execution of the job and the running time assuming all statements as local statements.

$\wp$ is valid for the single job case. For multiple jobs we have:

$$W^j = \int_0^{T_j} \wp^j(\tau) d\tau \tag{4}$$

Another related definition is the inefficiency of running the job on a machine:

$$\eta = \frac{P'\Gamma}{W}, \eta > 1 \tag{5}$$

Γ is the running time of a job, W is the work and P' is the number of processors actually allocated to the job.

The most important result contained in [4][6] is that, of all non-clairvoyant schedulers, instruction-balanced time-slicing has the least CR, with the corresponding proof. As a consequence of the proof it was verified that the least

possible CR is equal to η_{max}, which is the maximum inefficiency among all jobs that are running in a machine at a given moment.

3 IBTS-parallel

In IBTS, each job is allocated to the whole machine and executes a predefined number of instructions before being preempted. However, not all jobs necessarily use all processors of the machine at all times. In order to further optimize the space sharing in parallel scheduling policies should allocate simultaneously multiple jobs. We use the fact that IBTS is the non-clairvoyant algorithm with the least CR to propose a modified version of IBTS that also permits a better spatial allocation of the machine.

Let us start by considering a machine with N processors in a MIMD architectural model as described in [4]. In our modified version of IBTS the machine is shared by more than one job at any given time. Jobs are preempted after all threads, running in parallel, execute a fixed number I of instructions. The resulting scheduling algorithm is dubbed IBTS-parallel. The use of many different spatial scheduling strategies are possible with IBTS-Parallel (first-fit, best-fit, etc.

IBTS-Parallel has at least the same performance than IBTS under the CR metric, as stated in the lemma below.

Lemma 1. *The CR of IBTS-parallel is* $\leq \eta_{max}$

Proof. It follows from [4][6] by verifying that the two properties of IBTS are valid for IBTS-parallel :

1. At any time, an equal amount of power is supplied to all running jobs
2. The job that finishes last incurs the maximum amount of work

Using the lemma above, we can state the main result of this section.

Theorem 1. $\eta_{max}^{IBTS\_par} \leq \eta_{max}^{IBTS}$

In other words, IBTS represents the worst case of IBTS-parallel

Proof. As we are running multiple jobs in parallel, the execution time of each job will necessarily be smaller than if we run one job after another, as is the case in IBTS since one job allocates all processing elements of a machine. From the definition of inefficiency :

$$\eta = \frac{P'\Gamma}{W} \tag{6}$$

P' and W do not change from IBTS to IBTS-parallel. The time Γ is smaller or equal for all jobs under IBTS-parallel as compared to pure IBTS. So, all jobs have smaller or equal inefficiencies under IBTS-parallel as compared to pure IBTS, which makes the maximum inefficiency in IBTS-parallel smaller than or equal to the corresponding quantity in IBTS.

4 Simulation and Verification

To verify the results above, we used a general purpose event driven simulator, developed by our research group for studying a variety of related problems (e.g., dynamic scheduling, load balancing, etc). The simulator accepts two different formats for describing jobs. The first is a fully qualified DAG. The second is a set of parameters used to describe the job characteristics such as computation/communication ratio. When the second form is used the actual communication type, timing and pattern are left unspecified and it is up to the simulator to convert this user specification into a DAG, using probabilistic distributions, provided by the user, for each of the parameters. Other parameters include the spawning factor for each thread, a thread life span, synchronization pattern, degree of parallelism (maximum number of thread that can be executed at any given time), depth of critical path, etc. Even-though probabilistic distributions are used to generate the DAG, the DAG itself behaves in a completely deterministic way.

Once the input is in the form of a DAG, and the module responsible for implementing a particular scheduling heuristics is plugged into the simulator, several experiments can be performed using the same input by changing some of the parameters of the simulation such as the number of processing elements available or the topology of the network, among others. The outputs can be recorded in a variety of formats for later visualization.

For this study we grouped parallel jobs in classes where each class represents a particular degree of parallelism (maximum number of threads that can be executed at any given time). We divided the workload into ten different classes with each class containing 50 different jobs. The arrival time of a job is described by a Poisson random variable with an average rate of two job arrivals per time slice. The actual job selection is done in a round robin fashion by picking one job per class. This way we guarantee the interleaving of heavily parallel jobs with shorter ones.

We distinguish the classes of computation and of communication instructions in the various threads that compose a job. A communication forces the thread to be suspended until the communication is concluded. If the communication is concluded during the currently assigned time-slice the thread resumes execution. All threads are preempted only after executing 100 instructions (duration of a time slice), with the caveat that any thread that executed one or more communication instruction will be preempted at the end of the time-slice regardless of how many instructions it was able to execute in the current time-slice. We used a factor of 0.001 communications per computation instructions.

The practical implementation of IBTS-parallel was based on a greedy algorithm applied at the beginning of each workload change.During the workload change interval, called cycle, the workload is obviously assumed constant. Thus, the eligible threads of queued jobs are allocated to processors using the first fit strategy for each time slice. Clearly, after all eligible threads are scheduled on a processor for some time slice (slot), the temporal sequence is repeated peri-

odically until a workload change again occurs. We considered in simulations a machine with 1024 processors.

Preliminary results are shown in table 1. They have been normalized to show the total execution time of IBTS-parallel to be 100%.

Table 1. Preliminary experimental results

IBTS		IBTS-parallel	
Total Running Time (%)	Total Idle Time (%)	Total Running Time (%)	Total Idle Time (%)
123.6	41.9	100	28.2

It is important to dissect the value obtained for *idle time*. This value is the result of following three factors:

1. Communications
2. Absence of ready threads
3. Lack of job virtualization

The first is a natural consequence of threads communicating among themselves.. The second reflects the fact that jobs arrive and finish at random times and at any given instance there might not be any job ready to be scheduled. The last is a result of not allowing individual threads of a same job to be scheduled at different time-slices. This factor is, we believe, the chief reason behind the high idle time measured and should be greatly reduced by extending IBTS-parallel with job virtualization [4] techniques.

References

1. Blelloch,G. E.: Vector Models for Data-parallel Computing *MIT Press*,Cambridge, MA (1990)
2. Feitelson, D.: Job Scheduling in Multiprogrammed Parallel Systems *IBM Research Report RC 19970*, Second Revision (1997)
3. Motwani, R., Phillips, S., Torng, E., Non-clairvoyant scheduling, *Theoretical Computer Science*, (1994) 130(1):17- 47
4. Scherson, I. D., Subramaniam R., Reis, V. L. M., Campos, L. M.. Scheduling Computationally Intensive Data Parallel Programs. *Ecole Placement Dynamique et Re'-partition de charge* (1996) 39–61
5. Squillante, M. S., Wang, F., Papaefthymiou, M., An Analysis of Gang Scheduling for Multiprogrammed Parallel Computing Environments, *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, (1996) 89-98
6. Subramaniam, R.: A Framework for Parallel Job Scheduling *PhD Thesis*, University of California at Irvine (1995)