Bounds on Gang Service Scheduling

Fabricio Alves Barbosa da Silva[1]
Isaac D. Scherson[1,2*]

fabricio.silva@lip6.fr, isaac@uci.edu

[1]Laboratoire ASIM, LIP6, Université Pierre et Marie Curie, Paris, France.

[2]Dept. of Information and Comp. Science, University of California, Irvine
CA 92697, U.S.A

Abstract

We define the bounds of two gang service algorithms, namely time balanced and instruction balanced, using interactivity and throughput based metrics. The workload is composed of V-RAM jobs, and the architectural model chosen is LogP. We also analyze the effect of workload and space sharing strategies on the bounds found.

The following results are demonstrated: when interactivity is considered, instruction balanced gang service is the best possible non-clairvoyant algorithm. When the objective is to increase throughput Instruction balanced and time balanced gang service are equivalent algorithms. In this case the preemption strategy is not a determinant factor in performance

1 Introduction

The time sharing of a parallel machine, according to the definition of the parallel job scheduling problem, has both a spatial and a temporal dimension: The scheduler is not only responsible for deciding when a thread will run, but also where it will run, i.e. in which processing element. That gives another dimension to scheduling in multiprocessors as compared with uniprocessors [7].

In this paper we find performance bounds for parallel job scheduling algorithms, instruction balanced gang service (IBGS) and time balanced gang service (TBGS). In IBGS, each thread is preempted after executing I instructions. TBGS preempts a thread after a fixed period of time. We view parallel job scheduling as an optimization problem, and the define formally the input space, output space and metrics associated. We find bounds for the performance of these two algorithms under different workload conditions (light and heavy) and using the metric better suited for each case. Moreover, an analysis of effect of workload and space sharing strategies in those bounds is made.

This paper is divided as follows: in section 2 a detailed problem formulation if made, with definitions of gang service, the characteristics of the workloads, and the formalization of the optimization problem. In section 3 bounds are found for IBGS and TBGS under light and heavy workloads. The effects of workload on bounds are described in section 4 and the influences of space sharing strategies are in section 5.Section 6 contains our concluding remarks.

1.1 Terminology

In this paper we define a *Clairvoyant scheduling algorithm* as one that may use knowledge of the jobs' execution times to assign time intervals to jobs in a set of processors. A *Non Clairvoyant Scheduling Algorithm* assigns time interval to jobs in a set of processors without knowing the execution times of jobs that have not yet terminated. A scheduling problem is said to be *static* if all release times are 0, i.e., all the jobs are available for execution at the start of the schedule. A *dynamic* scheduling problem allows arbitrary (nonnegative) release jobs. These definitions are similar to those stated in [13]. A Parallel job is composed of threads, and we define *Preemptive scheduling algorithms* as those where the execution of any thread can de suspended at any time and resumed later from the point of preemption.

2 Problem Statement

Our objective in this paper is to define bounds for gang service algorithms, which is formally stated in subsection 2.1. Bounds are found for instruction balanced and time balanced algorithms under light and heavy workloads.

In this paper we make the distinction between heavy workloads and light workloads. Tasks that compose heavy workload have the following characteristics: they execute for a long time, have large memory requirements and contain a large amount of parallelism. Heavy workload jobs are usually submitted via a remote entry batch system, and have a non interactive nature.

Tasks that belong to light workloads have the inverse characteristics: they are short lived or use resources in burst (interactive), have small memory requirements, are less parallel than heavy workload jobs, and overall, do not consume a large amount of resources. These jobs are expected to have good interactive response. A similar differentiation between light and heavy workloads was already used on previous work [1, 5].

Two preemptions strategies will be considered in this paper: Time balanced and instruction balanced. In time balanced, each thread run for a predefined time before being preempted. This strategy is used by most sequential and parallel systems that support preemption. In instruction Balanced, each thread executes a fixed number of instructions before being preempted. Multithreaded processors are examples of such systems. In Multithreaded processors provides hardware primitives for scheduling in synchronization[12]. In such processors, multiple contexts are stored at the same time inside the processor or in a cached memory and the switching between these multiple contexts are done in hardware. The unit execution for a Multithreaded processor is the micro-thread, which is a control flow routine composed by a predefined number of instructions. Each thread of control belonging to a job is then broken up into a number of sequentially ordered micro-threads. A micro-thread can be as small as 1 instruction, as in the Tera MTA System[6], and may vary until tens of instructions. Multithreaded processors with fixed micro-thread size are examples of instruction balanced scheduling.

In this section we will state formally the scheduling problem in parallel machines using gang service. First we make a formal definition of gang service. Then we present the architecture and job model used in this paper. Finally the scheduling problem is formally stated as an optimization problem, through the definition of the input space, output space and metrics that are considered for the heavy load and light load.

2.1 Definition of Gang Service

A Gang service algorithm is defined as any algorithm belonging to the class of gang scheduling algorithms, using either instruction balanced or time balanced preemption. That is, Instruction Balanced Gang Service is the class of all algorithms where the jobs in a workload are gang schedule and preemption is made in function of the number of instructions executed, and Time balanced Gang Service is the class of all algorithms where the jobs in the workload are gang schedule and preemption occurs at fixed intervals of time. Gang service is a paradigm where all threads of a job in the service stage are grouped into a gang and concurrently scheduled in distinct processors. Reasons to consider gang service are responsiveness [8], efficient sharing of resources[10] and ease of programming. In gang service the threads of a job are supplied with an environment that is very similar to a dedicated machine [10]. It is useful to any model of computation and any programming style. The use of time slicing allows performance to degrade gradually as load increases. Applications with fine-grain interactions benefit of large performance improvements over uncoordinated scheduling[9].

In parallel job scheduling in general, and in gang service in particular, as the number of processors is grater than one, the time utilization as well as the spatial utilization can be better visualized with the help of a bidimensional diagram dubbed *trace diagram*. One dimension represents processors while the other dimension represents time or number of executed instructions. Through the trace diagram it is possible to visualize the time utilization of the set of processors given a scheduling algorithm. A similar representation has already been used, for instance, in [14]. One such diagram is illustrated in figure 1

Gang service algorithms are preemptive algorithms. We will be particularly interested in gang service algorithms which are *periodic and preemptive*. Related to periodic preemptive algorithms are the concepts of cycle, slice, period and slot. A *Workload change* occurs at the arrival of a new job, the termination of an existing one, or through the variation of the number of eligible threads of a job to be scheduled. The time between workload changes is defined as a *cycle*. Between workload changes, we may define a period that is a function of the workload and the spatial allocation. The period in the minimum interval of time where all jobs are scheduled at least once. A cycle/period is composed of *slices*; a slice includes all processors of the machine. One generalization of gang scheduling that

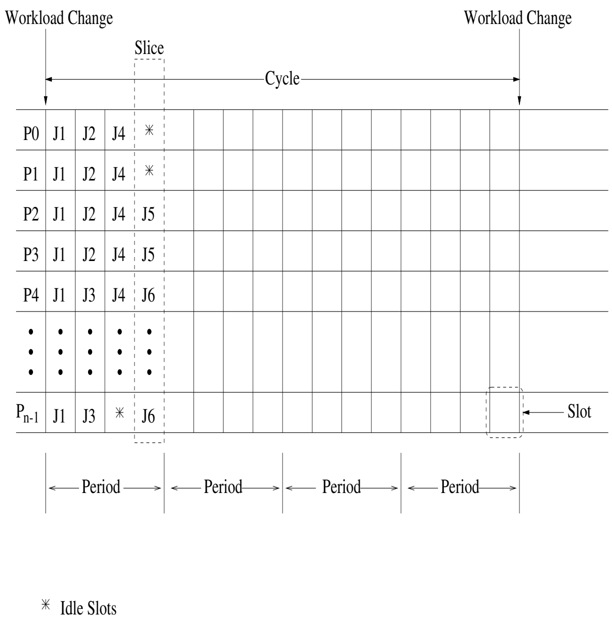

Figure 1: Definition of slice, slot, period and cycle

we will consider here is that we may have several jobs simultaneously scheduled in a slice. A *slot* is the processors' view of a slice. A Slice is composed of N slots, for a machine with N processors. If a processor has no assigned thread during its slot in a slice, then we have an idle slot. The number of idle slots in a period divided by the total number of slots in the period defines the *Idling Ratio*.

' We will consider that workload changes are detected between periods. By that we mean that, for instance, if a job arrives in the middle of a period, corresponding action of allocating the job is only taken by the end of the period.

2.2 Job and Architecture Model

For the purposes of analysis here, we consider data parallel jobs running on a LogP architectural model.

The Data Parallel Programming Model The Data Parallel Programming model is also known as the *Vector Random Access Machine (V-RAM)*. It is a theoretical model for *parallel* programming [2], a simple generalization of the RAM model in which numbers are replaced by vectors or arrays of basic data elements.

It is worth noting that in the V-RAM model, as in the RAM model, only one statement is executed at a time. In this sense *the program is strictly sequential*. All the parallelism is packed inside the individual statements, which operate on whole vectors at a time.

The data parallel programming model is the most popular and well-accepted model for parallel programming. One fact that reinforces this statement is that Sabot [16] mentions that in a survey of 120 parallel algorithms from three ACM Symposia on Theory of Computing (STOC), *all of them* were found to naturally fit the data-parallel model. Perhaps the reason for this popularity is that a data-parallel program has a single, sequential thread of control; programmers can think about it easily because they don't have to worry about conflicting threads, lost messages, race conditions, deadlocks etc.

The LogP Model LogP[4] defines a distributed memory machine where different nodes work asynchronously suggesting that current and short-term architectures will most likely consist of few hundreds of off-the-shelf processing elements, which is indeed what we observe today.

The LogP model defines a set of parameters, namely:

- L: an upper bound on the *latency*, or delay, incurred in communicating a message containing a word (or a few of them) from its source module to its target module.
- o: the *overhead*, defined as the time a processor is engaged in the transmission or reception of each message.
- g: the *gap*, defined as the minimum time interval between consecutive message transmission or consecutive message receptions at a processor.
- P: The number of processor/memory modules (processing elements - PEs).

It is assumed that the network has finite capacity, such at most $\lceil L/g \rceil$ massages can be in transit from any processor to any processor at any time. The LogP model is asynchronous,i.e. processors work asynchronously and the latency experienced by any message is unpredictable, but is bounded above by L in the absence of stalls. In function of this unpredictability, messages may not arrive in the same order that they were sent.

We use LogP as archituture model due to its wide acceptance in the research community. Other models could be also used in the sections that follow, including machine models, but since many machine models can be represented by a LogP model with appropriate parameters, the use of LogP allows us to make a more general formulation of the scheduling problem, suitable to a larger number of parallel machines.

2.3 Scheduling as an Optimization Problem

We would like to view scheduling as an optimization problem. To do so, we must clearly define what the input space, output space, and objective functions (metric) are, since these are the main elements that compose an optimization problem.

2.3.1 Input Space

The input to the scheduler consists of an MIMD machine and a data-parallel *workload* (i.e., a set of data-parallel jobs). The *input space* has two main components, those that characterize the machine and those that characterize the workload.

- The machine is characterized by a set of parameters that are a function of the architectural model chosen. In the LogP model these parameters are described in section 2.2.
- As we will deal with non-clairvoyant scheduling algorithms the workload is characterized by (a) the number of jobs J; (b) the two parameters $\langle P^j, M^j \rangle$, for each job j ($1 \leq j \leq J$), where, $P^j \times M^j$ is the amount of vector memory required by job j. The number of processors P^j denotes the degree of parallelism of the job and the ideal number of processors to be allocated to job j.

2.3.2 Output Space

The *output space* of the scheduler consists of a *spatial schedule* and a *temporal schedule*:

Spatial Schedule: For each job j, the scheduler has to decide which subset $\mathbf{P}^j$ of PEs to run it on. (The various $\mathbf{P}^j$s need not be disjoint.)

Temporal Schedule: For each job j, the scheduler has to decide when to run each job and how much priority the job has on each of the allocated PEs. The priority of job j is specified by a function $\pi^j : P^j \rightarrow \Re$, the quantity $\pi^j(p)$ being the priority job j holds on each PE p. Each PE time-slices between the various jobs it is responsible for in proportion to their priorities.

2.3.3 Metric

Due to the different characteristics and requirements for jobs belonging to light and heavy workloads, we will define a convenient metric for each case. First we will define the Happiness metric for light (interactive) workloads, and then the throughput metric for heavy workloads

Happiness Metric For the definition of happiness, we need to introduce the concepts of time, work, power and inefficiency. To define these concepts we use the representation of a machine in which communication between nodes occurs through remote write and remote read instructions in a LogP model. This representation can model a shared memory machine or a machine with remote DMA support. However it should be stressed that the following results are valid for any architecture parameterized into the LogP model (with the remark that some of the following formulae should be changed accordingly).

Thus, in the LogP model, a computation instruction executes in one unit of time by definition, a remote read instruction takes 4o+2L time units and a remote write corresponds to 2o+L time units. Two special cases of these remote instructions occur when a remote communication is made just after another one of the same type. In this case, the remote write costs $2o+L+g$ and the remote read $4o+2L+g$. Using these quantities, we define τ, the running time of a job, as:

$$\tau = f_l + f_r(4o + 2L) + f_w(2o + L) \\ +f_w^c(2o + L + g) + f_r^c(4o + 2L + g) \quad (1)$$

Where $f_l, f_r, f_w, f_r^c, f_w^c$ represent the number of times local computations, remote reads, remote writes, consecutive remote reads and consecutive remote writes will be encountered during a hypothetical execution of a V-RAM program.

In equation 1, we have supposed that the VRAM vector memory made available by the machine was equal or greater than the one required by the job, which is by definition equal to $P \times M$, where P is the optimal number of processors to run the job, and M is the corresponding number of lines of vector memory. In most cases that assumption is not true. In the case of a job j, if a partition of P^j processors are available to run that j, the required amount of lines vector memory will be multiplied by $\rho = \lceil P/P^j \rceil$. This technique, know as virtualization[17], will also have an impact in the total execution time of the job. It is worth noting that the variation in the number of processors will also have an impact in the *communication locality* ϵ of the program ($0 < \epsilon < 1$), by transforming remote access statements in local statements. We should then review the definition of the execution time of the time necessary to execute the VRAM job to take into account the possible variation in the number of available processors to execute job j

$$\tau = f_l\rho + f_r\rho(\epsilon + ((1 - \epsilon)(4o + 2L))) +$$

$$f_w\rho(\epsilon+((1-\epsilon)(2o+L)))+$$
$$f_w^c\rho(\epsilon+((1-\epsilon)(2o+L+g)))+$$
$$f_r^c\rho(\epsilon+((1-\epsilon)(4o+2L+g))) \qquad (2)$$

Where ρ and ϵ and dependent on the number of allocated processors P^j. Given the ideal number of processors for job j, the time τ decreases as the number of processors increases while $P^j \leq P$. If $P^j > P$ the time τ may remain stable or even increase due to, for instance, contention on the system or greater synchronization costs.

We now introduce the notion of work. Imagine an utopian machine in which there are infinite processors, and all networks are as fast as local accesses (i.e. all statements take one unit of time. In such a world our job will use P processors and take $f_l+f_r+f_w+f_r^c+f_w^c$ time. We define work to be the processor-time product in the ideal world:

$$W = P(f_l+f_r+f_w+f_w^c+f_r^c) \qquad (3)$$

Needless to say, the processor time product in the real world generally has a higher value:

$$P^j\tau \geq W \qquad (4)$$

Where P^j is the number of processors actually allocated to job j. Let us define inefficiency of running the job in the machine to be the discrepancy between the two sides of the inequality:

$$\eta = \frac{P^j\tau}{W}, \eta > 1 \qquad (5)$$

In analogy with physics terminology, let us define the power delivered to a job to be the work done on the job per unit time, that is:

$$\wp = \frac{W}{\tau} = \frac{P(f_l+f_r+f_w+f_w^c+f_r^c)}{\tau} \qquad (6)$$

In equation 6, Power was defined in the context of a single job. We have to modify that to handle multiple jobs. The main difference with multiple jobs is that the power supplied to a given job varies over time, increasing whenever one job terminates. Rather than the power supplied to a job j, it is more meaningful to talk about the power supplied to job j at time t.

Let us get the easy case out of way first: if job j has already terminated by time t, then the power supplied to job j at time t is undefined.

Next, we consider the hard case, when job j is still[1] running. Let us focus our attention on any one of the PEs on which job j is running and call it PE p. Let us say that currently p is also running jobs $j_1, j_2, ..., j_k$. Note that this may not be the full list of all jobs originally allocated on p, because some jobs may have terminated.What fraction of time is p currently devoting to job j ? It is :

$$\phi^j(p,t) = \frac{\pi^j(p)}{\pi^j(p)+\pi^{j_1}(p)+\pi^{j_2}(p)+\pi^{j_3}(p)+...+\pi^{j_k}(p)} \qquad (7)$$

In equation 7, $\pi^j(p)$ specifies the priority of j at PE p.

In the general case, considering any scheduling algorithm, different PEs may be devoting different fractions of time to job j. However, due to barrier synchronization constraints, the job only makes progress proportional to the least fraction of time devoted to it. If PE A devotes very little time to the job j then it will not help for another PE B to devote more time to it, because B will eventually find j stuck at a barrier synchronization point, unable to proceed to the next statement. Therefore, the effective fraction of time currently being devoted to job j is

$$\min_{p\in P^j} \phi^j(p,t) \qquad (8)$$

We call this quantity $\phi^j(t)$.

We are finally ready to write a formula for the power supplied to job j at time t.

$$\wp(t) = \frac{W}{\tau} = \frac{P(f_l+f_r+f_w+f_w^c+f_r^c)\phi^j(t)}{\tau} \qquad (9)$$

The intuitive significance of power is that it is the rate at which work is done on the job. Thus, whereas in the case of a single job we had $W=\wp\tau$, we now have:

$$W^j = \int_0^{t^j} \wp^j(t)dt \qquad (10)$$

We now introduce a different metric that explicitly recognizes the requirements of responsiveness in parallel job scheduling. The motivating principle behind the metric is: *No job should be ignored for a long time.* Here "long time" is defined as anything greater than the interactive response time required by that job.

In the following discussion, we will be treating time as a continuous variable. This is just for clarity of presentation. More accurately, we would divide time into intervals of length δ, where δ is the interactive response time required by the jobs. All integrals would then change to sums, graph to histograms, and so on.

[1]For convenience, in this section, we will use words such as "still" and "currently" to refer to time

It is reasonable to defines the *happiness* of job j at time t as the amount of power being supplied to it at that time, i.e., $\wp^j(t)$. Equally reasonably, we may identify the happiness of the user community at time t with the happiness of the most frustrated user, i.e., $min_j \wp^j(t)$.

The final metric is the average happiness of the user community over time:

$$Happiness(T) = \frac{\int_0^T min_j \wp^j(t)dt}{T} \tag{11}$$

Where T represents the time interval of the measure.

There is a problem with the previous formulation though. Most likely than not, schedulers do not have access to information such as number and type of statements in order to make an optimized scheduling decision. We accept this lack of information by considering only *non-clairvoyant* schedulers. Non-clairvoyant schedulers decide when and where to run a job based only on a limited amount of information available at the beginning of execution or at arrival time.

To evaluate the effectiveness of scheduling algorithms, we use the *competitive ratio*(CR) [17] because it is a formal way of evaluating algorithms that are limited in some way (e.g., limited information, computational power, number of preemptions) which is indeed our case. The measure was first introduced in the study of a memory management problem[11, 18].

The CR is then the ratio between the happiness that can be obtained by a *optimal algorithm* and the scheduler itself. It is worth noting that the scheduling algorithm is a non-clairvoyant one, and an optimal scheduler will always have more information that the non clairvoyant one. By minimizing the CR the scheduler approaches the result that would be obtained by an algorithm with all the available information about the set of jobs.

$$CR(x) = \frac{Happiness_{OPT}(x)}{Happiness_A(x)} \tag{12}$$

In equation 12, x represents the input to the algorithm, $Happiness_A$ is the happiness obtained by the scheduler and $Happiness_{OPT}$ is the happiness obtained by the optimal scheduler.

Throughput Metric In heavy workloads, throughput, not interactivity, is the main objective in order to minimize the running time of those jobs. For the definition of a throughput based metric, we also use in this section the concepts of time, work, power and inefficiency. By throughput we mean the maximization of the total work done in a given period of time. Formally, we have:

$$Th(T) = \frac{W^{tot}}{T} \tag{13}$$

where:

$$W^{tot} = \sum_j W^j \tag{14}$$

As with the happiness metric, we use the competitive ration to evaluate the effectiveness of scheduling algorithm in the heavy workload case:

$$CR(x) = \frac{Th_{OPT}(x)}{Th_A(x)} \tag{15}$$

In equation 15, x represents the input to the algorithm, Th_A is the throughput obtained by the scheduler and Th_{OPT} is the throughput obtained by the optimal scheduler.

3 Bounds

In this section, bounds are found for the light and heavy workloads. First we begin by finding the bounds related with light workloads, using either instruction balanced or time balanced as strategies of preemption.

3.1 Light Workload

In the light workload case, the main objective is interactivity, as stated before in the definition of the happiness metric. In this section we find the bounds of instruction balanced and time balanced strategies in order to compare them under happiness.

As we emphasize interactivity, we concentrate ourselves in time sharing algorithms, since with non time sharing algorithms the minimum power in devoted to the jobs of a workload W is zero unless all remaining jobs are scheduled in the machine.

First we will define the upper an lower bounds for any non clairvoyant scheduling algorithm, as stated in the theorem below:

Theorem 1 *The CR of any time-shared non-clairvoyant scheduling algorithm is bounded by* $1 \leq CR \leq \eta_{max} \frac{W^{max}}{W^L}$

Proof 1 *The proof will use the following notation:*
W^j *= work requirement of job j*
$W^{max} = max_j W^j$
$W^{tot} = \sum_j W^j$
η^j *= Inefficiency of job j, if it were running alone in all PE's of the machine.*
$\eta^{max} = max_j \eta^j$

$\wp^j(t)$ = *Power supplied to job j at time t*
P' = *Number of PEs in the machine*

The lower bound represents the trivial case where the machine is dedicated, having only one job with local computation instructions only. In this case the Happiness reduces to the power delivered to that job, which is constant over time, given the job requirement of P^j processors and $P^j \leq P'$.

Conversely, a clairvoyant adversary will always have more information than a non-clairvoyant scheduler to make a scheduling decision. So the CR is always grater than or equal to one.

For the upper bound, note that, under a non-clairvoyant algorithm, job L finishes last and its completion time is T_L.

$$Happiness = \frac{\int_0^{T_L} min_j \wp^j(t)dt}{T_L} = \frac{W^L}{T_L} \tag{16}$$

Given W^{tot}, the total amount of work for all jobs, and T_L, the total time to finish all jobs, and since all jobs run with inefficiency of $\leq \eta^{max}$, it follows, from the definition of inefficiency:

$$T_L \leq \frac{\eta^{max} W^{tot}}{P'} \tag{17}$$

Substituting back, we see that the Happiness for a non-clairvoyant satisfies

$$Happiness \geq \frac{P' W^L}{\eta^{max} W^{tot}} \tag{18}$$

Now let us consider any arbitrary scheduler, perhaps even a clairvoyant one. Suppose that under the given scheduler, job K finishes last, and that its finishing time is T_K. Then:

$$\begin{aligned} Happiness = \frac{\int_0^{T_K} min_j \wp^j(t)dt}{T_K} &\leq \frac{\int_0^{T_K} \wp^K(t)dt}{T_K} \\ &= \frac{W^K}{T_K} \leq \frac{W^{max}}{T_K} \end{aligned} \tag{19}$$

Moreover, from the definition of inefficiency (equation 5).

$$T_K \geq \frac{W^{tot}}{P'} \tag{20}$$

Where the equal sign represents the extreme case where all jobs of a workload have only local computation instructions.

Substituting back, we see that the happiness of any arbitrary scheduler (clairvoyant or not) must satisfy:

$$Happiness \leq \frac{P' W^{max}}{W^{tot}} \tag{21}$$

From equations 18 and 21, it follows that the CR of IBGS is $\leq \eta_{max} \frac{W^{max}}{W^L}$.

3.1.1 Instruction Balanced Gang Service

Instruction Balanced Gang Service(IBGS) is a very simple non-clairvoyant scheduling algorithm. Instruction balanced means that the time quanta devoted to the various jobs are equal as measured in machine instructions (that is, all jobs are preempted after an equal number of machine instructions).

Given a workload W and a time quanta of I instructions, we define a cycle in a IBGS scheduling when all jobs of W that have not finished yet have executed I instructions.

Theorem 2 *The CR of IBGS is bounded by $1 \leq CR \leq \eta_{max}$*

Proof 2 *The proof for the upper bound makes use of the following properties of the IBGS: For a workload W and a time quanta of I instructions:*

1. *At every cycle, an equal amount of power is supplied to all running jobs, if no jobs finish at that cycle.*
2. *The job that executes the greater number of cycles incurs the maximum amount of work*

For the upper bound, note that, under a non-clairvoyant algorithm, job L finishes last and its completion time is T_L. Then L incurs the maximum amount of work.

$$Happiness = \frac{\int_0^{T_L} min_j \wp^j(t)dt}{T_L} = \frac{W^L}{T_L} = \frac{W^{max}}{T_L} \tag{22}$$

The above equation follows from the fact that, at every cycle, all jobs receives the same amount of power P. So $min_j \wp^j(t) = P$ at each cycle.

Substituting back, we see that the Happiness for IBGS satisfies

$$Happiness \geq \frac{P' W^{max}}{\eta^{max} W^{tot}} \tag{23}$$

From equations 23 and 21, it follows that the CR of IBGS is $\leq \eta_{max}$.

3.1.2 Time Balanced Gang Service

The bounds of time balanced gang service under happiness are determined in the following theorem:

Theorem 3 *The CR of Gang Scheduling with fixed time slice is bounded by* $1 \leq CR \leq \eta_{max} \frac{W^{max}}{W^L}$

Where W^L *is the work of the job that finishes last.*

Proof 3 *The lower bound is obtained as the lower bound of IBGS. For the upper bound, we observe that the two properties of IBGS are no longer valid for gang scheduling with fixed time slice. Consider that, under gang with fixed time slice, job L finishes last and its completion time is* T_L *:*

$$Happiness = \frac{\int_0^{T_L} min_j \wp^j(t)dt}{T_L} = \frac{W^L}{T_L} \tag{24}$$

Equation 17 is also valid for fixed time slice gang. We then see that the happiness for fixed time slice gang satisfies:

$$Happiness \geq \frac{P' W^L}{\eta^{max} W^{tot}} \tag{25}$$

From equations 21 and 25 it follows that the CR of gang scheduling with a fixed time slice is $\leq \eta_{max} \frac{W^{max}}{W^L}$.

Observe that, for the same workload and the same preemption strategy, η_{max} is constant. We can conclude that IBGS is always better than or equal to any other non-clairvoyant algorithm, since min $\frac{W^{max}}{W^L} = 1$

3.2 Heavy workload

Since jobs belonging to heavy workloads have large memory requirements, processor scheduling in these workloads must also take into account the memory scheduling associated. The memory scheduler determines which subset of ready tasks to load into memory. The processor scheduler determines from the set of in-memory tasks, which tasks schedule on each processor at a given moment on time.

This problem is operating system dependent, and two main scenarios are possible: full swapping system with no demand paging virtual memory and system that support virtual memory. In the first case, the entire address space of a job must be in main memory in order to the task to be scheduled, and in the second case only a subset of the address space of each task eligible to be schedule must be in main memory.

In full swapping system, a memory scheduler is mandatory[5]: In function of the decisions previously taken by the memory schedule, the processor schedule decide which job to schedule. Demand paging virtual memory given more flexibility to the processor schedule, since more tasks can be eligible to be schedule in a given moment of time. Although the vast majority of parallel systems do not support system-wide virtual memory, work on this area [15] proves not only that such a schema is not only feasible but it is highly desirable. In this work we consider that the parallel machine has support to efficient demand paging virtual memory, that is, the processor scheduler is not limited by decisions taken previously by the memory scheduler.

3.2.1 Instruction Balanced Gang Service

The bounds for instruction balanced gang service under heavy workload are defined in the following theorem.

Theorem 4 *The CR of IBGS is bounded by* $1 \leq CR \leq \eta_{max}$

Proof 4 *As in the proof for IBGS in a light workload, this proof will make the use of the following notation:*

W^j *= work requirement of job j*

$W^{max} = max_j W^j$

$W^{tot} = \sum_j W^j$

η^j *= Inefficiency of job j, if it were running alone in all PE's of the machine.*

$\eta^{max} = max_j \eta^j$

$\wp^j(t)$ *= Power supplied to job j at time t*

P' *= Number of PEs in the machine*

For the lower bound, given a workload W, we minimize the time for the execution of the workload when the Work is evenly distributed among all processors. Consider the case of only one job in the machine which the ideal number of processors is equal to P'. Given the definition of inefficiency, we have:

$$Th_{IBGS} = \frac{W^{tot}}{T} = \frac{P'}{\eta} = \frac{P'}{\eta_{max}} = Th_{OPT} \tag{26}$$

Where T is the execution time of the job. From equation 26 we conclude that the lower bound for the CR under throughput is equal to one.

We now consider the upper bound. For any algorithm, even an optimal one, we have:

$$T_L \geq \frac{W^{tot}}{P'} \tag{27}$$

We conclude that, for the optimal algorithm

$$Th_{OPT}(T) \leq P' \tag{28}$$

With IBGS we have:

$$Th_{IBGS}(T) = \frac{W^{tot}}{T_L} \quad (29)$$

Where T_L is the time to finish all jobs under IBGS.

Given W^{tot}, the total amount of work for all jobs, and T_L, the total time to finish all jobs, and since all jobs run with inefficiency of $\leq \eta^{max}$, it follows, from the definition of inefficiency:

$$T_L \leq \frac{\eta^{max} W^{tot}}{P'} \quad (30)$$

The Throughput of the IBGS in is then equal to:

$$Th_{IBGS}(T) \geq \frac{P'}{\eta_{max}} \quad (31)$$

From equations 31 and 28 we conclude that the CR of IBGS under throughput is $\leq \eta_{max}$

3.2.2 Time Balanced Gang Service

Theorem 5 states the bounds for TBGS under heavy workload.

Theorem 5 *The CR of TBGS is bounded by $1 \leq CR \leq \eta_{max}$*

Proof 5 *In the lower bound, as we considered that there is only one job at the machine, there no preemption and the behavior is the same using either IBGS or TBGS. For the upper bound observer that equations 31 and 28 are also valid for TBGS; equation 28 represents the optimal throughput, while equation 31 is a consequence of the definition of inefficiency.*

4 Effect of Workload on Bounds and on CR

The maximum inefficiency η_{max} is function of the workload. Since all upper bounds found are directly proportional to η_{max}, the workload do have an effect in the definition of the bounds. Furthermore, the workload is also determinant on where inside the bounds the CR is for a particular space sharing strategy. This can be illustrated by a simple example. Suppose and workload composed by only one job, which requires P' processors (the number of processors of the machine). As it is a VRAM job, the work is evenly distributed among all processors. The maximum inefficiency if a function of the number of communication instructions of the job, and can be well greater than one. However, the CR in this case is equal to one, since the work is evenly distributed among all processors, which minimizes the execution time of the job.

By other side the lower bound is not function of the workload, since the CR cannot be smaller than one, being equal to in some special cases as demonstrated.

5 Effect of Spatial allocation On Bounds

In section 3 we have seen that the upper bounds for IBTS and TBGS under light and heavy workload are proportional to η_{max}. The objective of this section is to investigate the effect of the spatial allocation on the bounds defined on previous subsections.

For the analysis of the effect of spatial allocation on bounds we take into account the ideal number of processors P_{id}^j for a job j. We define as the ideal number of processors for a job j the minimum number of processors that minimizes τ. This information is know by the user in job submission time since, as we have seen, the ideal number of processors P_{id}^j of job j is part of the input space. The fact that we have jobs where $P_{id}^j < P^j$ implies that the spatial allocation strategy that a impact in the maximum inefficiency of the workload and then in the upper bound. This is formally stated in the theorem below.

Theorem 6 *The maximum inefficiency of a workload W scheduled in a system with no space sharing is always greater than or equal to than the same workload scheduled in a system with support to space sharing, using the same preemption strategy.*

Proof 6 *In the proof we make use of the following notation:*

η^j = *Inefficiency of job j.*
W_j = *Work on job j.*
P' = *Number of PEs in the machine*
P_{id}^j = *Ideal number of PEs for job j.*

For each job j in the workload W in a machine with no space sharing we have:

$$\eta^j = \frac{P'\tau_j'}{W_j} \quad (32)$$

Where τ_j' is the time the job takes to execute in P' processors. Let's now consider a machine with support for space sharing that allocates a smaller number of processors for a job j when $P_{id}^j < P'$, always trying to share the machine with two or more processors. In that case

$$\eta^j = \frac{P^j\tau_j}{W_j} \quad (33)$$

Which are necessarily smaller than or equal to the corresponding inefficiency defined in equation 32 since $P^j_{id} \leq P^j \leq P'$ and $\tau_j \leq \tau'_j$. So is the maximum inefficiency η^{SH}_{max} for a space sharing system.

In the best case $P^j_{id} = P^j$ for every job j. However, this is a variation of the classical, one dimensional bin-packing problem. In bin-packing a given list of items $L = I_1, I_2, I_3, ...$ is to be partitioned into a minimum number of subsets such that the items on each subset sum to no more than C. In the standard physical interpretation, we view the items of a subset as having been packed in a bin of capacity C. This problem is NP-Hard[3], so research has concentrated on algorithms that are merely close to optimal.

The lower bounds do not change with space sharing, since they represent special cases which are not dependent on the space sharing strategy.

6 Conclusion

We shown that IBGS is the best possible algorithm for light workloads under the models considered in this paper, due to the particular characteristics of the algorithm. For interactivity aimed systems, such as parallel servers for multimedia applications IBGS provides better interactive response than other non clairvoyant strategies.

By other side, IBGS is equivalent to TBGS when the objective is to increase throughput. In that case the preemption strategy is not determinant; spatial sharing and workload characteristics define which performance is achieved. other characteristics, such as I/O accesses and support for virtual memory, which are not directly considered by the models we use, should also be taken into consideration. What we can conclude is that preemption strategy is not a major factor for throughput aimed systems.

References

[1] I. Ashok and Z. Zahorjan. Scheduling a Mixed Interactive and Batch Workload on a Parallel, Shared Memory Computer. In *Proceedings of Supercomputing' 92*, pages 616–625, 1992.

[2] Guy E. Blelloch. *Vector Models for Data-Parallel Computing.* The MIT Press, Cambridge,Massachusetts, 1990.

[3] E.G. Coffman, D.S. Johnson, P.W. Shor, and R.R. Weber. Markov Chains, Computer Proofs, and Average Case Analysis of Best Fit Bin Packing. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 412–421, 1993.

[4] D. Culler et al. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of 4th ACM SIGPLAN Symposium on Principles an Practice of Parallel Programming*, pages 1–12, 1993.

[5] G. Averson et al. Scheduling on the Tera MTA. *Job Scheduling Strategies for Parallel Processing*, LNCS 949, 1995.

[6] R. Alverson et al. The Tera Computer System. In *Proceedings of the 1990 International Conference on Supercomputing*, 1990.

[7] D. Feitelson. Job Scheduling in Multiprogrammed Parallel Systems. Technical report, IBM T. J. Watson Research Center, 1997. RC 19970 - Second Revision.

[8] D. Feitelson and M. A.Jette. Improved Utilization and Responsiveness with Gang Scheduling. *Job Scheduling Strategies for Parallel Processing*, LNCS 1291:238–261, 1997.

[9] D. Feitelson and L. Rudolph. Gang Scheduling Performance Benefits for Fine-Grain Synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.

[10] M. A. Jette. Performance Characteristics of Gang Scheduling In Multiprogrammed Environments. In *Proceedings of SC'97*, 1997.

[11] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive Algorithms for On Line problems. In *Proceedings of the Twentieth Annual Symposium on the theory of Computing*, pages 322–333, 1988.

[12] S. W. Moore. *Multithreaded Processor Design.* Kluwer Academic Publishers, 1996.

[13] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[14] J.K. Ousterhout. Scheduling Techniques for Concurrent Systems. In *Proceedings of the 3rd International Conference on Distributed Comp. Systems*, pages 22–30, 1982.

[15] Veronica Reis. Designing virtual memory systems for parallel and distributed computing. In *PhD Thesis.* University of California, Irvine, December 1996.

[16] G. Sabot. *The Paralation Model : Architecture Independent Parallel Programming.* The MIT Press, Cambridge, Massachussets, 1988.

[17] Isaac D. Scherson, Raghu Subramanian, Veronica L. M. Reis, and Luis Miguel Campos. Scheduling computationally intensive data parallel programs. In Bertil Folliot, editor, *Placement Dynamique et Répartition de Charge: application aux systèmes parallèles et répartis*, pages 39–61. Centre National De La Recherche Scientifique, July 1996.

[18] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202 – 208, 1985.