Estimated and Observed Performance of Heuristic Algorithms for Task Scheduling on Heterogeneous Processors*

J.P.W. Kitajima
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702
30123-970 Belo Horizonte - MG
{kita@dcc.ufmg.br}

S.C.S. Porto
Computação Aplicada e Automação
Universidade Federal Fluminense
Rua Passo da Pátria 156
24210-240 Niteói - RJ
{stella@caa.uff.br}

Abstract - Parallel applications with regular and well-known behavior, where task execution time estimates are fairly reliable, are suited to static task scheduling (in opposition to dynamic scheduling, performed during the execution of the application). This is the case of many parallel scientific applications. Task scheduling on a heterogeneous environment, where processors present different processing speeds, is even more complex than on a homogeneous one. Static task scheduling is, thus, performed based on estimated data about the parallel application and the system architecture, and has long taken a heuristic approach. Therefore, a realistic performance evaluation of a task scheduling algorithm can only be fully accomplished if practical results are also considered. In this sense, the present work analyzes the quality of greedy and tabu search task scheduling algorithms comparing estimated deterministic results with the actual observed makespan of several parallel synthetic applications executing on real parallel machines following the static schedule previously determined.

Resumo - Aplicações paralelas com um comportamento regular e bem conhecido, onde as estimativas dos tempos de execução de tarefas são bastante razoáveis, adequam-se bem ao escalonamento estático (em oposição ao escalonamento dinâmico, realizado durante a execução da aplicação). Este é o caso de várias aplicações científicas paralelas. O escalonamento de tarefas em um ambiente heterogêneo, onde os processadores apresentam capacidades diferentes de processamento, é ainda mais complexo do que em um ambiente homogêneo. Escalonamento estático de tarefas baseia-se em dados estimados sobre a aplicação paralela e a arquitetura do sistema e tem recebido freqüentemente um enfoque heurístico. Por isso, uma avaliação de desempenho realística de um algoritmo de escalonamento de tarefas poderá apenas ser completamente realizado se resultados práticos também forem considerados. Neste sentido, o presente trabalho analisa a qualidade de dois algoritmos de escalonamento, um heurístico de construção e outro baseado na metaheurística de busca tabu. Esta análise compara resultados determinísticos com resultados observados do tempo de execução de diversas aplicações paralelas sintéticas em máquinas paralelas reais de acordo com o escalonamento previamente obtido.

*This work has been accomplished thanks to the financial support from CNPq agency. The computational experiments were performed using the IBM SP machine available at the LMC/IMAG laboratory, Grenoble, France.

1 Introduction

Task scheduling is one of the most challenging problems in parallel and distributed computing, and is known to be NP-hard [5]. Scheduling algorithms have long taken a heuristic approach. An important argument in favor of the use of heuristics is that optimization is usually performed over a model of a real world problem. There is no guarantee that the best solution for this model is also the best solution for the associated real world problem [15]. Obviously, truly exact models are not possible, but heuristics are generally more flexible and capable of better manipulating objective functions or complex constraints than exact algorithms. This is the case of methods such as simulated annealing, tabu search and genetic algorithms, for example, where objective functions do not rely on simplifying linearity premises. Hence, heuristics allow more precise and elaborate models than those used by exact algorithms [15].

Parallel applications with regular and well-known behavior, where task execution time estimates are fairly reliable, are suited to static task scheduling (in opposition to dynamic scheduling, performed during the execution of the application). This is the case of a great majority of scientific applications. For these applications, the static scheduling algorithm is executed once, before the execution of the parallel program, which is then actually run several times according to the previously obtained schedule. Consequently, even if the scheduling algorithm is a costly procedure, this cost will be amortized throughout the numerous executions of the parallel application, i.e. the obtained schedule is re-applied repeatedly.

Processor heterogeneity, here represented by processors with different processing speeds, has already demonstrated the potentiality in reducing the performance degradation resulting from the execution of the inherent serial fractions of the parallel application on a homogeneous processor set [9]. Moreover, we can definitely envision the design of parallel and distributed systems with heterogeneous structures as a trend in contemporary computer architecture. In this sense, modeling parallel machines as heterogeneous systems means taking into account more realistic issues in computer design. Task scheduling on this heterogeneous environment is even more complex than on a homogeneous one [10]. Greedy heuristic task scheduling algorithms, considering heterogeneous processors and parallel applications represented by task precedence graphs, were first presented in [10]. In [12, 13], Porto and Ribeiro proposed and analyzed sequential and parallel implementations of a tabu search based algorithm. Tabu search [6] is a local search adaptive procedure for solving combinatorial optimization problems, which guides a hill-descending heuristic to continue exploration without becoming confounded by the absence of improving moves, and without falling back into local optimum from which it previously emerged. This tabu search task scheduling algorithm was thoroughly studied in recent work [14], where the solution quality was compared to the results obtained by the best greedy algorithm for this same problem [10]. In most of the studied cases, the tabu search approach achieved higher solution quality, i.e. reduced makespan for the parallel application. In all these mentioned work, scheduling algorithms were evaluated based on estimated results, obtained through deterministic calculation or simulation using different sets of problem instances.

Barr *et al.* [1] comment that, since an algorithm is an abstraction, it is evaluated indirectly by experimenting with a specific implementation. There is a wide range of options when selecting the problems, implementing the algorithm, choosing a computing environment, selecting performance measures setting algorithm options, and reporting results. The choice made for each factor can have a substantial effect on the results and significance of the experiment. To ensure that the reported information is meaningful, the experimental design to be documented and used must consider as many factors as possible, effectively measuring their impacts on the results.

Static task scheduling is, thus, performed based on estimated data about the parallel application and the system architecture. Therefore, realistic performance evaluation of a task scheduling algorithm can only be fully accomplished if practical results are also considered. In this sense, the present work analyzes the quality of greedy and tabu search task scheduling algorithms comparing estimated deterministic results with the actual observed makespan of several parallel synthetic applications executing on real parallel machines following the static schedule previously determined. The following section presents the schedule system model. In Section 3, both the greedy and tabu search algorithms are described. In Section 4, we report the overall experimentation, including: (i) a description of the testing platform and problem instances considered during the testing phase; (ii) the most significant numerical results, and (iii) the comparative solution quality analysis according to different parameters. Section 5 presents some brief concluding remarks.

2 The Scheduling Model

A parallel application Π with a set of n tasks $T = \{t_1, \cdots, t_n\}$ and a heterogeneous multiprocessor system composed by a set of m interconnected processors $P = \{p_1, \cdots, p_m\}$ can be represented by a task precedence graph $G(\Pi)$ and an $n \times m$ matrix μ, where $\mu_{kj} = \mu(t_k, p_j)$ is the estimated execution time of a task $t_k \in T$ at processor $p_j \in P$. Each processor can run one task at a time, all tasks can be executed by any processor, and processors are said to be uniform in the sense that $\frac{\mu_{kj}}{\mu_{ki}} = \frac{\mu_{lj}}{\mu_{li}}, \forall t_k, t_l \in T, \forall p_i, p_j \in P$. This implies that processors may be ranked according to their processing speeds. In a framework with one single faster (heterogeneous) processor, the heterogeneity may be expressed by a unique parameter called *processor power ratio*, PPR, which is the ratio between the processing speed of the fastest processor and that of the remaining ones (those in the subset of homogeneous processors). Thus, an instance of our scheduling problem is characterized by the workload and parallel system models.

Given a solution s for the scheduling problem, a processor assignment function is designed as the mapping $\mathcal{A}_s : T \rightarrow P$. A task t_k is said to be assigned to processor $p_j \in P$ in solution s if $\mathcal{A}_s(t_k) = p_j$. The task scheduling problem can then be formulated as the search for an optimal assignment of the set of tasks onto that of the processors, in terms of the *makespan* $c(s)$ of the parallel application (cost of the solution s), i.e. the completion time of the last task being executed. At the end of the scheduling process, each processor ends up with an ordered list of tasks that will run on it as soon as they become executable.

3 Heuristic Task Scheduling Algorithms

We consider two algorithms in this work, namely: a greedy algorithm called DES+ MFT and a parallel tabu search algorithm, here referred as TSpar. Although both of them are heuristic, they present different fundamental characteristics. The former is a construction algorithm, which iteratively assigns tasks to processors based on heuristic criteria, taking into account the static information of the system model. On the other hand, the TSpar is a synchronous parallel implementation of a tabu search metaheuristic algorithm, which guides an aggressive local search procedure over the task scheduling solution space.

3.1 The DES+MFT Greedy Algorithm

DES+MFT stands for *Deterministic Execution Simulation with Minimum Finish Time* [10]. This algorithm iteratively schedules tasks in a partial order according to the simulated execution of the parallel application (DES), based on the estimated task execution times, while scheduling decisions are made according to the minimum finishing time (MFT) for each "schedulable" task. Figure 1 describes the DES+MFT in a procedural scheme. In this scheme, the *clock* variable measures the evolution of the execution. At the end of this procedure, $c(s)$ = *clock* is the cost of the obtained solution, i.e., the makespan of the parallel application when submitted to the DES+MFT processor assignment. At each iteration, certain tasks are scheduled to processors, building an ordered list of tasks associated to each processor. This is the actual execution order if tasks were to be executed in an ideal system with estimated execution times. During this deterministic execution simulation, each task $t_k \in T$ assumes one of the following states at each time instant: *non-executable*, *executable*, *executing*, *executed*. At the same time, each processor $p_j \in P$ alternates between two different states: *free* and *busy*. A processor p_j is said to be *busy* if it has a task in the *executing* state allocated to it.

Every task starts in the *non-executable* state. It will change to the *executable* state if it has no predecessors or if all of its predecessors have achieved the *executed* state. Every *executable* task is considered as a candidate for scheduling at each time instant.

A task will be effectively scheduled (changing to the *executing* state), when the processor (chosen from the entire processor set), which determines the minimum finishing time for that task, becomes *free*. Otherwise the task will not be assigned to any processor until a further iteration. This is the so called "look-ahead" feature of this greedy algorithm. To determine the minimum finishing time of each task-processor pair, in the moment where scheduling decisions are eventually made, the algorithm maintains a record of the closest instant in future simulation time in which each processor will be available. This means that there may be iterations where *schedulable* tasks are not selected, because the processor presumed by the algorithm as the best one is not available (i.e. it is *busy*) at that moment. After a task is scheduled, the algorithm simulates the elapsed execution time, using the variable *clock*, which is updated accordingly. It should be noticed that DES+MFT, like most greedy algorithms, does not come back to re-evaluate the scheduling decisions taken in previous iterations. This means that besides the "look-ahead" feature, it is not capable of making changes in scheduling decisions made in previous iterations, which were based on snapshots of the simulated execution. Consequently, these scheduling decisions depend on how strongly tasks are tied through precedence relations, because they determine the order in which tasks may possibly be scheduled. Differently, the TSpar algorithm, departing from the initial solution obtained by the DES+MFT algorithm, evaluates many other possible assignments, which eventually improve the makespan of the parallel application, as we can see in the following section.

3.2 The Parallel Tabu Search Algorithm

To describe the TSpar algorithm, we first consider a general combinatorial optimization problem (P) formulated as to

$$\begin{array}{ll} \text{minimize} & c(s) \\ \text{subject to} & s \in S, \end{array}$$

where S is a discrete set of feasible solutions. Local search approaches for solving problem (P) are based on search procedures in the solution space S starting from an initial solution $s_0 \in S$. At each iteration, a heuristic is used to obtain a new solution s' in the neighborhood $N(s)$ of the current solution s, through slight changes in s. A move is an atomic change which transforms the current solution, s, into one of its neighbors, say $\bar{s}$. Thus, *movevalue* $= c(\bar{s}) - c(s)$ is the difference between the value of the cost function after the move, $c(\bar{s})$, and the value of the cost function before the move, $c(s)$. Every feasible solution $\bar{s} \in N(s)$ is evaluated according to the cost function $c(.)$, which is eventually optimized. The current solution moves smoothly towards better neighbor solutions, enhancing the best obtained solution s^*.

Tabu search [6, 7] may be described as a higher level heuristic for solving minimization problems, designed to guide other hill-descending heuristics in order to escape from local optima. Thus, tabu search is an adaptive search technique that aims to intelligently exploring the solution space in search of good, hopefully optimal, solutions. The learning capability determines that tabu search supplies richer knowledge about the instance of the problem to be solved than that generated in other iterative algorithms. In the case of the task scheduling problem considered in this paper, the cost of a solution is given by its makespan, i.e., the overall execution time of the parallel application. The neighborhood $N(s)$ of the current solution s is the set of all solutions differing from it by only a single assignment. If $\bar{s} \in N(s)$, then there is only one task $t_i \in T$ for which $\mathcal{A}_s(t_i) \neq \mathcal{A}_{\bar{s}}(t_i)$. Each move may be characterized by a simple representation given by $(\mathcal{A}_s(t_i), t_i, p_l)$, as far as the position task t_i will occupy in the task list of processor p_l is uniquely defined. If the best move takes the current solution s to a best neighbor solution s' degenerating its cost function, i.e. $c(s') \geq c(s)$, then the reverse move must be prohibited during a certain number of iterations (tabu tenure) in order to avoid cycling. However, there are situations in which a recently prohibited move, if applied after some iterations, will provide a better solution than the best one found by the algorithm so far, despite its prohibited status. In these cases, an *aspiration criterion* is used to override this prohibition,

```
DES+MFT algorithm
begin
    $clock \leftarrow 0$
    $state(p_j) \leftarrow$ free $\forall p_j \in P$
    $start(t_k), finish(t_k) \leftarrow 0 \; \forall t_k \in T$
    while ($\exists t_k \in T \mid state(t_k) \neq$ executed) do
    begin
        for (each $t_k \in T \mid state(t_k) =$ executable and $p_j \in P$) do
            obtain the pair $(t_l, p_i)$ with the minimum finish time
            if ($state(p_i) =$ free) then
            begin
                $state(t_l) \leftarrow$ executing
                $\mathcal{A}_s(t_l) = p_i$
                $state(p_i) \leftarrow$ busy
                $start(t_l) \leftarrow clock$
                $finish(t_l) \leftarrow start(t_l) + \mu(t_l, p_i)$
            end
        Let $i$ be such that $finish(t_i) = \min_{t_k \in T \mid state(t_k)=\text{executing}}\{finish(t_k)\}$
        $clock \leftarrow finish(t_i)$
        for (each $t_k \in T \mid state(t_k) =$ executing and $finish(t_k) = clock$) do
        begin
            $state(t_k) \leftarrow$ executed
            $state(\mathcal{A}_s(t_k)) \leftarrow$ free
        end
    end
    $c(s) \leftarrow clock$
end
```

Figure 1: DES+MFT algorithm description.

enabling the move to be executed. In [12] and [13] the reader will find more detailed description of the tabu search algorithm.

Several implementations have been proposed in recent literature for the parallelization of tabu search [2, 3, 4, 17]. The parallelization schemes proposed in [13] for the tabu search scheduling algorithm have a synchronization point at the end of each iteration of the search. The search for the best neighbor during each iteration is performed in parallel and different sets of neighbor solutions are analyzed by each task. This typically characterizes a strict domain decomposition parallelization scheme.

Two basic parallel programming models were used: Master-Slave (MS) and Single-Program-Multiple-Data (SPMD). In both models, parallel strategies determine a rigid synchronized cooperation scheme, and differences arise essentially in the way information is exchanged between processes at the end of each tabu search iteration. Four different strategies were implemented, derived from these two models. In the SPMD model, communication follows a ring structure. In the master-slave model, slaves communicate strictly with the master, and the number and size of the domain partitions determine two different approaches: single and multiple partitioning (MS-SP and MS-MP, respectively).

In both master-slave strategies, neighborhood partitions are distributed to all the slaves. The master is responsible for selecting the best move at the end of each iteration among all partial results. The best move is then sent to all the slaves in order to start the next iteration from a new current solution. In the MS-MP approach, to attend load balancing requirements, the distribution of the domain partitions is done on a work-demand-basis. The neighborhood is divided in equal size partitions, but the number of partitions is significantly greater than the number of slaves. When a slave ends its partial search over a certain partition, the master may eventually send to it a new partition, if there are any still left to be searched. More work will be then given to the slaves which are less loaded (and are thus working faster), and the master dedicates itself exclusively to the partition distribution.

The promising results obtained through parallelization led to the possibility of more effectively evaluating solution quality of the proposed tabu search task scheduling algorithm using its parallel implementation. Considering both sequential and parallel implementation, solution quality was analyzed according to different parameters and strategies, which needed to fully specify the tabu search algorithm with a certain variety of application model parameters (such as task graph structures, number of tasks, serial fraction and task service demands) and system configurations (such as number of processors and architecture heterogeneity measured by the processor power ratio). It was shown that the tabu search algorithm obtained better results, i.e. shorter completion times for parallel applications, improving up to 40% the makespan obtained by the DES+MFT algorithm, which in fact is the most appropriate greedy algorithm previously published in the literature [12, 14]. We have used the MS-MP parallel version to carry out the experimentation reported here, because it has demonstrated the best speedup results in most of the studied cases [13].

4 Experimental Results

In this section, we depict some experimental results obtained from the execution of synthetic parallel programs scheduled with both the greedy and tabu search algorithms. We first present some results derived from the estimated improvement analysis of tabu search schedules over those generated by the DES+MFT, which is the initial solution for the tabu search algorithm. The performance criterium is the makespan (solution cost) estimated by both algorithms. In the following, we describe *ANDES* [8], a framework for performance evaluation using parallel program models and synthetic programs. Finally, using this framework, we compare execution times of synthetic parallel programs scheduled by DES+MFT and TSpar algorithms.

4.1 Estimated Performance Analysis

DES+MFT and TSpar scheduling algorithms were implemented using ANSI C and PVM (*Parallel Virtual Machine*) [16]. The schedule quality is estimated based on the computed makespan. In other words, the makespan represents the schedule cost, $c(.)$, which is to be minimized.

One of the main goals is to achieve makespan reduction when changing from the schedule produced by DES+MFT to the one produced by TSpar. Thus, solution quality is measured by relative cost reduction, $\mathcal{R}$, computed as

$$\mathcal{R} = \frac{c(s_0) - c(s^*)}{c(s_0)}$$

where s_0 is the initial solution obtained by the greedy algorithm DES+MFT and s^* is the best solution found by the TSpar algorithm.

In [12], relative cost reductions of up to 30% were obtained considering applications modeled by diamond-shaped precedence graphs. In [14], new results were presented considering other structures for the parallel applications. Part of the *ANDES* benchmark was then used: other types of diamond-shaped graphs (`Diamond3` and `Diamond4`), iterative graphs (`FFT` and `PDE2`), divide-and-conquer strategies (`Div-conq`), typical matrix computation structures (`Gauss`), and master-slave models (`MS3`). We can summarize the following results of these above experiments:

- A parallel application is said to be *serialized* by a certain processor assignment algorithm when all of its tasks are scheduled to one unique processor. When the serial fraction (F_s) and/or the processor power ratio (PPR) are very high, the best solution is usually obtained through the serialization of the application over the heterogeneous processor, which has greater processing capacity. This seems to be clear if we imagine two extreme cases: $F_s = 1$ or $PPR \longrightarrow \infty$. In the first case, we face a totally serial application, which must be executed on the heterogeneous processor (F_s corresponds to the serial fraction defined as the fraction of the total parallel execution time when just one task is executing even if infinite processors were available). In the latter case, the heterogeneous processor is able to execute any task in infinitesimal time, consequently serialization determines again the best performance.

  In certain circumstances, serialization will be performed by the DES+MFT algorithm, when there is still available parallelism to be explored in the parallel application. In these cases, the tabu search algorithm will start from a serialized initial schedule, and more easily will be capable of finding different assignments which greatly reduce the overall makespan of the application, augmenting the relative cost reduction.

  For very low and very high PPR values low or null makespan improvements are obtained. A low PPR value means low heterogeneity degree, and, in this case, the greedy algorithm improvements are sufficient (it is suitable for homogeneous configurations). On the other end of the heterogeneity range, very high PPR values mean that *serialization* on the very fast processor is the best solution. In these cases both the DES+MFT and TSpar algorithms are able of serializing the application, so makespan improvements are not observed;

- Between the two extremes of the PPR value range, we find a mountain-like peak of improvements, culminating with a PPR that gives the best relative performance achieved by the TSpar algorithm. This point is referred as the PPR_{peak} point. The PPR_{peak} point is highly dependent on the shape of the input task graph. Groups of similar task graphs have a similar behavior. For example, diamond-shaped graphs present a low PPR_{peak} (around 5). On the other hand, iterative graphs produce a more smooth improvement curve, with higher PPR_{peak} (around 20 or 30), depending on the size of the task graph;

- Not only the structure of the task graph is critical in the relative quality improvement analysis. The number of processors available for scheduling assignments influence the results. The relationship between solution quality improvements and the number of processors is variable depending on the structure of the task graph. On one hand, the greater the number of processors we have, the less heterogeneous the system becomes and thus lower relative cost reduction is achieved. However, a greater number of processors also represents more available parallelism and therefore a greater number of different scheduling possibilities arise.

Figure 2 presents some estimated relative cost reductions computed between DES+MFT and TSpar algorithms. In [14], Porto *et al.* measured improvements for discrete values of PPR (2, 5, 10, 20, ..., depending on the input). Figure 3 presents a more detailed experiment, with a fine variation of PPR values and number of processors, considering the `Diamond3` benchmark with 66 tasks.

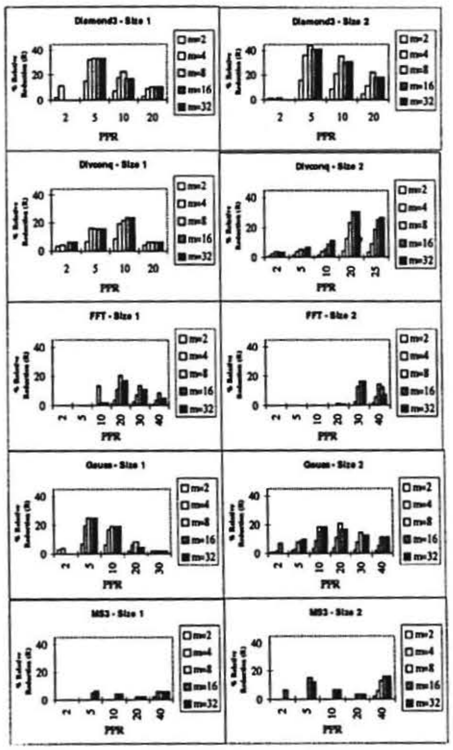

Figure 2: Relative cost reduction $\mathcal{R}$ versus PPR for two different sizes of `Diamond3`, `Divconq`, `FFT`, `Gauss`, and `MS3` graphs (m corresponds to the number of processors to which the tasks are scheduled).

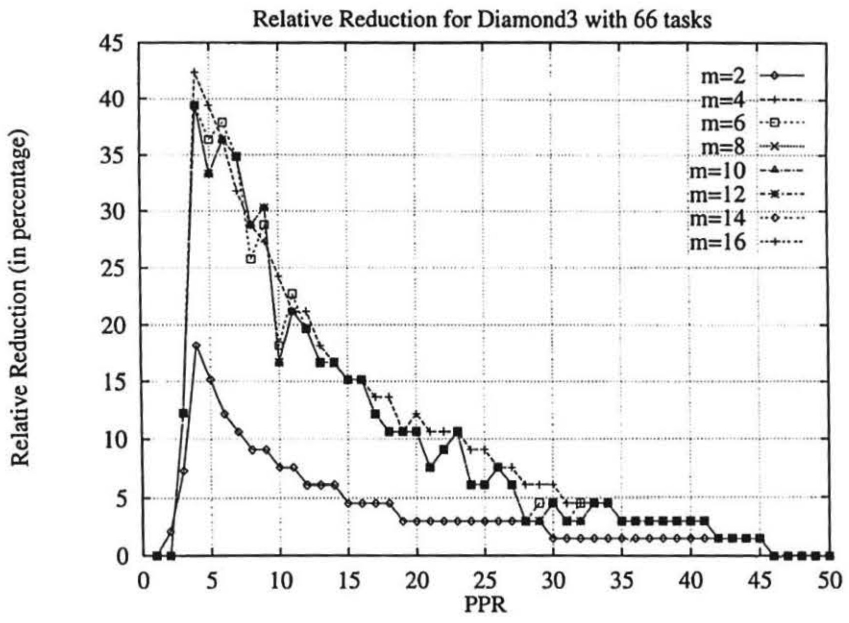

Figure 3: Detailed relative cost reduction $\mathcal{R}$ versus PPR for `Diamond3` graph (m corresponds to the number of processors to which the tasks are scheduled).

4.2 The Experimental Framework

The *ANDES* Environment – *ANDES* [14] is a PVM-based parallel tool that supports performance evaluation of parallel programs at the prediction level. *ANDES* considers the existing complex overheads of parallel computers. This is achieved through the use of *synthetic parallel executions* directly on the parallel machine. In a synthetic parallel execution, the resources of the parallel computer are used in a controlled way, but no code is generated. All the steps from the interpretation of the parallel program graph-based and of the parallel machine models to the synthetic execution on the target parallel machine are automatically managed by *ANDES*. *ANDES* finally computes performance metrics from the execution of that workload implemented according to mapping and/or scheduling strategies. Synthetic execution is the chosen performance technique due to the easy control of parameters as well as the real environment in use. The idea is to conjugate the best of model-based approaches with the best of realistic parallel executions. *ANDES* has been used to refine analytical and simulation analysis. With the current high availability of parallel systems, the results of *ANDES* have been proved to be precise and useful.

The Parallel System – *ANDES* along with the synthetic parallel programs were executed on an IBM SP multicomputer composed of 32 RS6000 RISC microprocessors with 64 megabytes of RAM. The processors are interconnected by a high-speed switch (bidirectional with nominal speed of 80 megabytes per second).

The Benchmarks – In order to compare estimated and observed improvements of the overall execution times of real parallel synthetic programs, we have used the following benchmark (part of the *ANDES* package): (i) `Diamond3` with 66 tasks; (ii) `FFT` with 194 tasks; (iii)`Gauss` with 192 tasks; and (iv) `Divconq` with 46 tasks.

This benchmark picks representative task graphs from the ones studied in [14]. Small and larger task graphs are used. The TSpar was executed using 4 processors of the IBM SP. The estimated quality of both TSpar and DES+MFT algorithms is computed using a conventional C procedure for computing the makespan of the task graphs, detailed in Figure 4 (very similar to the DES+MFT description). The final value of *clock* is the actual makespan. Each graph of the benchmark is scheduled to 2, 4, 8, and 16 processors.

```
makespan computation algorithm
begin
   Let $s = (\mathcal{A}_s(t_1), \ldots, \mathcal{A}_s(t_n))$ be a feasible solution for the scheduling problem, i.e.,
      for every $k = 1, \ldots, n$, $\mathcal{A}_s(t_k) = p_j$ for some $p_j \in P$
   $clock \leftarrow 0$
   $state(p_j) \leftarrow$ free $\forall p_j \in P$
   $start(t_k), finish(t_k) \leftarrow 0 \; \forall t_k \in T$
   while ($\exists t_k \in T \mid state(t_k) \neq$ executed) do
   begin
      for (each $t_k \in T \mid state(t_k) =$ executable) do
         if ($state(\mathcal{A}_s(t_k)) =$ free) then
         begin
            $state(t_k) \leftarrow$ executing
            $state(\mathcal{A}_s(t_k)) \leftarrow$ busy
            $start(t_k) \leftarrow clock$
            $finish(t_k) \leftarrow start(t_k) + \mu(t_k, \mathcal{A}_s(t_k))$
         end
      Let $i$ be such that $finish(t_i) = \min_{t_k \in T \mid state(t_k)=\text{executing}}\{finish(t_k)\}$
      $clock \leftarrow finish(t_i)$
      for (each $t_k \in T \mid state(t_k) =$ executing and $finish(t_k) = clock$) do
      begin
         $state(t_k) \leftarrow$ executed
         $state(\mathcal{A}_s(t_k)) \leftarrow$ free
      end
   end
   $c(s) \leftarrow clock$
end
```

Figure 4: Computation of the makespan of a given schedule.

The generated schedules are read by *ANDES* which generates the synthetic load to be interpreted by *ANDES-Synth*, the synthetic execution kernel. Synthetic loads are then executed according to the given schedules.

In order to simulate heterogeneity, the size of synthetic loops corresponding to tasks allocated to the faster processor are reduced by a factor corresponding to the PPR itself. Thus, a PPR of 2 means that loops to be executed on the heterogeneous processor are reduced by half. The scheduling algorithms consider communications with zero overhead. This corresponds in *ANDES* to communications of a single byte (in the IBM SP machine, such message transmitted through the switch determines a latency of around 47.03 microseconds [11]).

Preliminary experiments were performed on an idle machine. The standard deviation was always under 1% for 10 consecutive executions. Considering this low degree of variability, we have performed measures using a sample of size 5.

4.3 Results and Analysis

Figures 5, 6, 7, and 8 present, in the same graphic, estimated and measured relative cost reductions. The chosen PPR value range includes, for all graphics, the higher relative cost reductions achieved by TSpar. Differences between estimated and observed improvements are under 5% for all experiments.

Our results demonstrated by the similarity between estimated and observed relative cost reductions that the makespan computation used in both scheduling algorithms is in fact reliable. This is a completely deterministic computation. On the other hand, the observed execution times are definitely non-deterministic due to overheads from the operating system and the communication subsystem. However, the execution times presented very low variability. Therefore, these overheads do not influence significantly the experimental execution times, i.e. the makespan algorithm shows itself to be very useful to the static scheduling decisions based on estimated data. Although intuitive, this conclusion is not obvious and experiments were necessary to validate it.

Taking into account a precise makespan computation, one important consequence is that tabu search improvements are real and significant. This was foreseen from previous works, based on the estimated relative cost reductions between DES+MFT and TSpar algorithms. In this paper, we demonstrate that these improvements also occur in more realistic execution environments.

Another interesting result is that the PPR_{peak} is not always the same. As a matter of fact, there is a range of PPR values where the best relative cost reductions vary. This irregular behavior occurs due to the irregular search through the solution space performed by the tabu search algorithm, which depends on different heuristic parameters such as tabu list size, number of iterations without improvements, and aspiration criteria. Metaheuristics, such as tabu search, frequently depend on a fine tuning stage, where parameters are tested and calibrated. After this step, they remain unchanged, and in some test cases they are not always set to achieve the best results.

Finally, *ANDES* has been proven to be a useful tool in the validation of scheduling algorithms. The direct combination of both scheduling algorithms and the synthetic execution runtime system provided an environment where response time measurements could be quickly obtained.

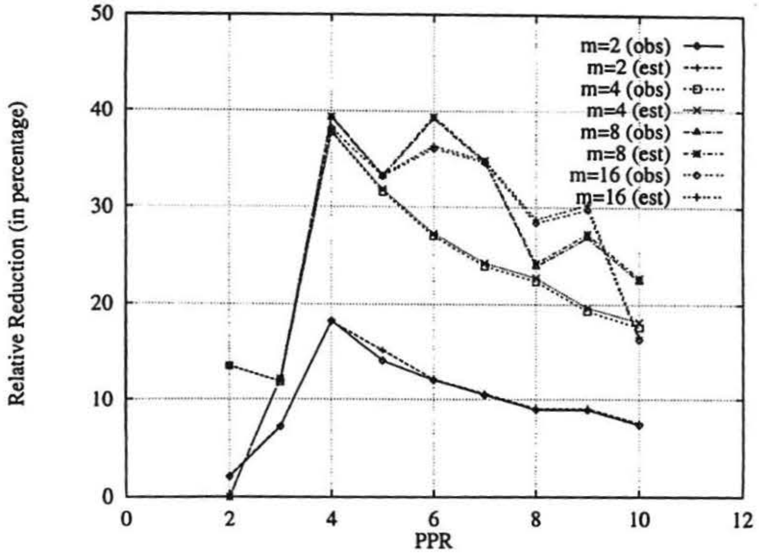

Figure 5: Estimated (est) and observed (obs) relative cost reduction $\mathcal{R}$ versus PPR for Diamond3 graph (m corresponds to the number of processors on which the tasks are scheduled).

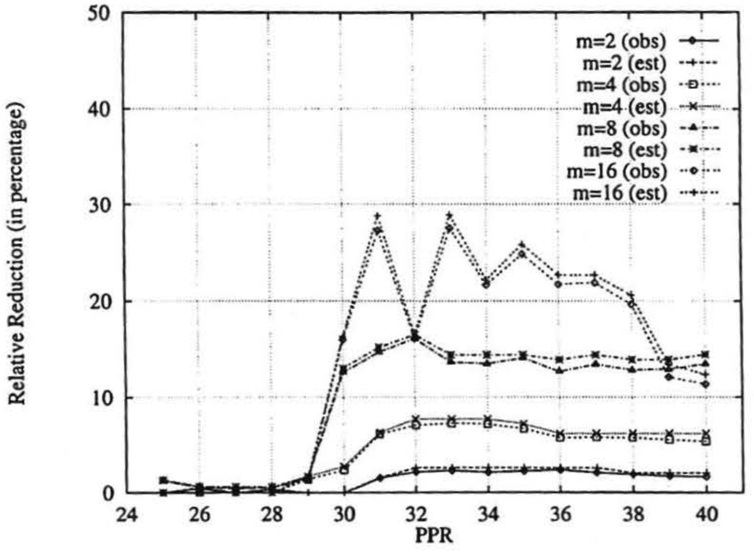

Figure 6: Estimated (est) and observed (obs) relative cost reduction $\mathcal{R}$ versus PPR for FFT graph (m corresponds to the number of processors on which the tasks are scheduled).

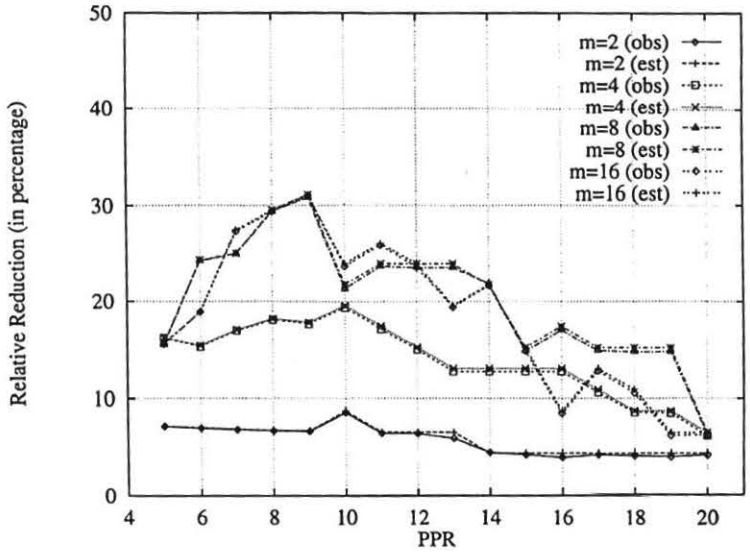

Figure 7: Estimated (est) and observed (obs) relative cost reduction $\mathcal{R}$ versus PPR for `Divconq` graph (m corresponds to the number of processors on which the tasks are scheduled).

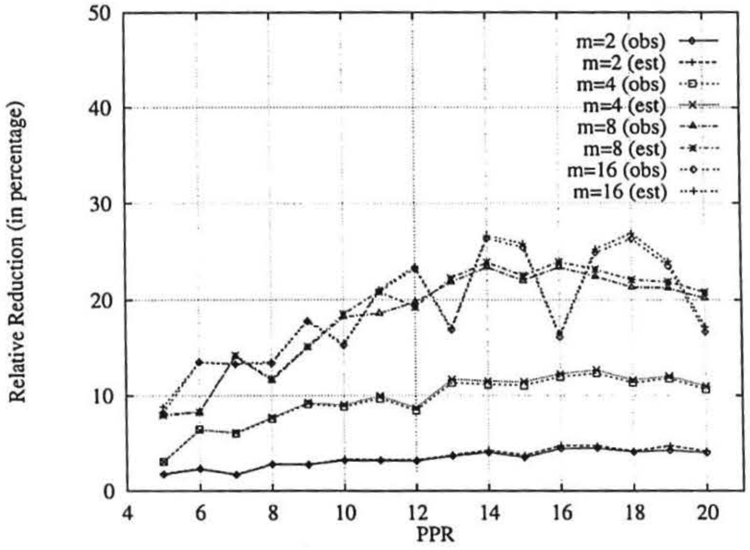

Figure 8: Estimated (est) and observed (obs) relative cost reduction $\mathcal{R}$ versus PPR for `Gauss` graph (m corresponds to the number of processors on which the tasks are scheduled).

5 Final Remarks

This paper presents an experimental validation of makespan improvements of two scheduling algorithms: a greedy construction algorithm and a tabu search based algorithm. Synthetic parallel executions were performed using the scheduled graph costs. These synthetic executions were performed on a real parallel machine (IBM SP). The estimated and observed response times improvements are very similar, representing the low impact of system overheads on makespan improvement estimation. This guarantees a reliable cost function for static scheduling algorithms and confirms the actual better results of the tabu search metaheuristic applied to scheduling problems.

References

[1] R.S. BARR, B.L. GOLDEN, J.P. KELLY, M.G.C. RESENDE, and W.R. STEWART,"Designing and Reporting on Computational Experiments with Heuristic Methods", *Journal of Heuristics* 1 (1995), 9–32.

[2] T.G. CRAINIC, M. TOULOUSE, and M. GENDREAU, "Towards a Taxonomy of Parallel Tabu Search Algorithms", Research Report CRT-933, Centre de Recherche sur les Transports, Université de Montréal, 1993.

[3] C.-N. FIECHTER "A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems", *Discrete Applied Mathematics* 51 (1994), 243–267.

[4] B. GARCIA and M. TOULOUSE, "A Parallel Tabu Search for the Vehicle Routing Problem with Time Windows", *Computers and Operations Research* 21 (1994), 1025–1033.

[5] M.R. GAREY and D.S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[6] F. GLOVER and M. LAGUNA, "Tabu Search", Chapter 3 in *Modern Heuristic Techniques for Combinatorial Problems* (C.R. Reeves, ed.), 70–150, Blackwell Scientific Publications, Oxford, 1992.

[7] F. GLOVER, E. TAILLARD, and D. DE WERRA, "A User's Guide to Tabu Search", *Annals of Operations Research* 41 (1993), 3–28.

[8] J.P. KITAJIMA, B. PLATEAU, P.BOUVRY, and D. TRYSTRAM, "A method and a tool for performance evaluation. A case study: Evaluating mapping strategies", *Proceedings of the 1994 Cray Users Group Meeting*, Tours, 1994.

[9] D.A. MENASCÉ and V. ALMEIDA, "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures", *Proceedings of the Supercomputing'90 Conference*, New York, 1990.

[10] D.A. MENASCÉ and S.C.S. PORTO, "Processor Assignment in Heterogeneous Parallel Architectures", *Proceedings of the IEEE International Parallel Processing Symposium*, 186–191, Beverly Hills, 1992.

[11] J. MIGUEL, A. ARRUABARRENA, R. BEIVIDE, and J. A. GREGORIO, "Assessing the performance of the new IBM SP2 communication subsystem", *IEEE Parallel & Distributed Technology* 4(1996), 12–22.

[12] S.C.S. PORTO and C.C. RIBEIRO, "A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints", *International Journal of High-Speed Computing* 7 (1995), 45–71.

[13] S.C.S. PORTO and C.C. RIBEIRO, "Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constraints", *Journal of Heuristics* 1 (1996), 207–233.

[14] S.C.S. PORTO, J.P.W. KITAJIMA, and C.C. RIBEIRO, "Performance Evaluation of a Parallel Tabu Search Scheduling Algorithm", *Solving Combinatorial Problems in Parallel* (joint workshop with the *International Parallel Processing Symposium'97*), April 1-5 1997, Geneva.

[15] C.R. REEVES, Chapter 1 in *Modern Heuristic Techniques for Combinatorial Problems* (C.R. Reeves, ed.), Blackwell Scientific Publications, Oxford, 1992.

[16] V. S. SUNDERAM, G. A. GEIST, J. DONGARRA, and R. MANCHEK, "The PVM concurrent computing system: evolution, experiences, and trends", *Parallel Computing* 20(1994), 531–546.

[17] E. TAILLARD, "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem", *ORSA Journal on Computing* 6 (1994), 108–117.