Performance Evaluation of a Parallel Tabu Search Task Scheduling Algorithm

Stella C.S. Porto

Instituto de Computação
Universidade Federal Fluminense
Niterói, Brazil
stella@caa.uff.br

João Paulo F.W. Kitajima

Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
kitajima@dcc.ufmg.br

Celso C. Ribeiro

Computer Science Department
Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
celso@inf.puc-rio.br

Abstract

This paper presents the solution quality analysis of a parallel tabu search algorithm for the task scheduling problem on heterogeneous processors under precedence constraints. We evaluate the achieved makespan reduction of different parallel applications, relatively to the results obtained by the best greedy algorithm in the literature, as a function of parameters such as problem size, system heterogeneity, and number of processors. Our results show that the parallel tabu search algorithm leads to much better solutions than the greedy algorithm in many cases where the latter is not capable of profiting from the inherent application parallelism and system heterogeneity.

Keywords: Parallel algorithms, task scheduling, greedy algorithm, tabu search, solution quality.

1 Introduction

In parallel processing, the parallel portion of the application can be accelerated according to the number of processors allocated to it. In a homogeneous architecture, where all processors are identical, the sequential portion of the application will have to be executed in one of the processors, considerably degrading the execution time of the application. A more powerful processor, if correctly scheduled, may be responsible for executing the serial portion of the parallel application, leading to a higher performance. Menascé and Almeida [15] have proposed analytical models to improve the cost-effectiveness of a multiprocessor, introducing a heterogeneous architecture, where one processor with greater processing capacity is responsible for the execution of the serial fraction of computation, thus reducing the completion time of the parallel application.

Two main distinguishing features of parallel versus sequential programming are *program partitioning* and *task scheduling* [5]. Both techniques are essential to high-performance computing on both homogeneous and heterogeneous systems. The partitioning problem deals with how to detect parallelism and determine the best trade-off between parallelism and overhead, which means finding the best grain size that maximizes parallelism while reducing overhead. After program partitioning, tasks must be optimally scheduled on the processors such that the overall makespan of the parallel application is minimized. In general, the scheduling problem deals with choosing the order in which a certain number of tasks may be performed and their assignment to processors in a parallel/distributed environment.

According to El-Rewini [5], scheduling techniques can be classified based on the availability of program task information as *deterministic* and *non-deterministic*. In the first case, all the information about tasks to be scheduled and the relation between them are entirely known prior to execution time. In non-deterministic scheduling, some information may not be known before program execution. The distinction between *static* or *dynamic* scheduling methods indicates the time at which the scheduling decisions are made. With static scheduling, information regarding the task graph representing the program must be estimated prior to execution, whereas in dynamic scheduling, the parallel processor system schedules tasks on the fly. The disadvantage in the latter is the overhead incurred to determine the schedule while the program is running, so that processor assignment should be done very fast by a simple algorithm, eventually deteriorating the quality of the solution thus obtained. On the other hand, in the case of static scheduling, although possibly less accurate information is available, more sophisticated algorithms may be used since the compiler will be in charge of the assignment. The compilation time will certainly be longer, but the cost of task management should be smaller, since each processor will be ready in advance. Moreover, increasing compilation times is entirely justified for large scientific computation programs, where the execution times are much more relevant. These applications present a known behavior and regularity, thus reliable estimates of the task execution times are predicted. Scheduling will be performed once and the program itself is separately executed. Consequently, even if the scheduling effort is costly, its cost will be amortized over the many times the same schedule will be re-applied. In a heterogeneous parallel environment, where processors have different speeds, task scheduling turns out to be an even more difficult problem, since a distinct processor assignment decision for a certain task eventually determines a different execution time. This work deals with the static task scheduling problem on heterogeneous processors under precedence constraints, using a tabu search approach.

Task scheduling (or processor assignment) is one of the most challenging problems in parallel and distributed computing and is known to be NP-hard [8, 9] in its general form, as well as in several restricted cases. There are few known polynomially solvable scheduling problems, even when severe constraints are placed on the parallel application and target machine models. The intractability of the scheduling problem has led to a large number of heuristics, each of which may work under different circumstances. These heuristics do not guarantee an optimal solution to the problem, but instead they try to find near optimal solutions. Straightforward heuristics tend to have well-defined termination rules, as with greedy and local-search methods. More complex algorithms, such as the metaheuristics tabu search, simulated annealing, genetic algorithms, and GRASP, may not have standard termination rules and typically search for improved solutions until an arbitrary stopping point is reached.

The task scheduling problem in a heterogeneous multiprocessor environment with applications represented by task precedence graphs was first considered by Porto and Menascé in [16, 17]. Fast greedy algorithms for processor assignment of parallel applications on heterogeneous multiprocessor architectures were proposed and compared. Porto and Ribeiro [18] then applied the tabu search metaheuristic to the same problem. Tabu search [10, 11, 12, 13] 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 optima from which it previously emerged. Solution quality was evaluated according to application and machine model parameters. The results obtained by tabu search considerably improved by approximately 25% the makespan (i.e., the completion time) of the parallel applications, with respect to schedules generated by the best known greedy algorithm. Recently, Porto and Ribeiro [19] have presented different parallel implementations for this tabu search algorithm, using synchronous message-passing strategies based on domain decomposition. The performance of these parallel programs was assessed as a function of problem size and the number of parallel processes, achieving efficiencies close to one in most of the cases and showing the suitability of this kind of parallelization approach for this class of algorithms.

Due to the large computational times observed for tabu search algorithm, the experimentation performed in [18] to validate and analyze solution quality was based in a rather strict testing framework. Specifically, a single parallel application structure was used to report numerical results. 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 the 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. In this sense, the present work analyzes the quality of the solutions obtained by a parallel tabu search algorithm through a broader set of test problems. We describe the behavior of the algorithm as a function of distinct and important problem parameters, and demonstrate the sharp effect of the task graph structure in task scheduling procedures. The following section presents the schedule system model. Section 3 briefly describes the parallel tabu search algorithm. In Section 4, we report the overall experimentation, including: (i) a description of the parameters and performance measures considered during the testing phase, (ii) issues concerning the basic greedy algorithm, (iii) the most significant numerical results, and (iv) the evaluation of solution quality relatively to these different parameters. Section 5 presents some brief concluding remarks.

2 The Scheduling System Model

A scheduling system consists of a parallel application, a target machine, a processor assignment (final schedule) and a certain performance criterion [5]. In what follows, we show how each of these four components is used in the present work.

A parallel application Π with a set of n tasks $T = \{t_1, \ldots, t_n\}$ and a heterogeneous multiprocessor system composed by a set of m interconnected processors $P = \{p_1, \ldots, p_m\}$ 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 execution time of task $t_k \in T$ at processor $p_j \in P$. Each processor can run one task at a time, all tasks can be processed by any processor, and processors are said to be uniform in the sense that $\frac{\mu(t_k,p_j)}{\mu(t_k,p_i)} = \frac{\mu(t_l,p_j)}{\mu(t_l,p_i)}$ $\forall t_k, t_\ell \in T, \forall p_i, p_j \in P$. This implies that processors may be ranked according to their processing speeds. In a framework with a single 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 system models.

Given a solution s for the scheduling problem, a processor assignment function is defined as a mapping $\mathcal{A}_s : T \rightarrow P$. A task $t_k \in T$ 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, 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 The Parallel Tabu Search Algorithm

To describe the tabu search 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 [10, 11, 12, 13] 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 optimum. Thus, tabu search is an adaptive search technique that aims to intelligently explore 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, enabling the move to be executed. The tabu search task scheduling heuristic strongly depends on two parameters, namely the tabu tenure (size of the tabu list) and the maximum number of iterations without improvement. Its behavior also depends on the strategy implemented for the type and restrictiveness of the tabu list, as well as on the candidate list and aspiration criteria strategies.

The reader is referred to [18] for a detailed description of the tabu search algorithm for the considered task scheduling problem. Different parameters and strategies needed to fully specify this tabu search algorithm were studied side-by-side with a certain variety of application model parameters (number of tasks, serial fraction, and task service demands) and system configurations (number of processors and architecture heterogeneity measured by the processor power ratio). The structure of the Mean Value Analysis (MVA) [20] solution package for product form queueing networks was used for solution quality evaluation. It was shown that the tabu search algorithm obtained better results, i.e. shorter completion times for parallel applications, improving from 20 to 30% the makespan obtained by the most appropriate algorithm previously published in the literature.

However, this solution quality evaluation analysis lacked diversity in the class of reported test problem instances, mainly due to the restriction on the structure of the task graph of the parallel application, in the form of a *wavefront* precedence graph. Porto and Ribeiro clearly stated in [18] that a straightforward and mandatory extension of their work would consist in the analysis of the tabu search algorithm behavior considering a broader range of input data, in particular different parallel programs described by distinct precedence graph structures, accompanied by the variation of the other problem parameters, such as the processor power ratio and the number of processors. The promising results obtained through parallelization gave way to the possibility of effectively evaluating solution quality of the proposed tabu search task scheduling algorithm using its parallel implementation.

Several implementations have been proposed in the recent literature for the parallelization of tabu search [2, 3, 4, 6, 7, 21, 22] The parallelization schemes proposed in [19] for the tabu search scheduling algorithm are synchronized 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 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 into 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. We have used this parallel version to carry out the experimentation reported here, because it has demonstrated the best speedup results in most of the studied cases [19].

4 Computational Results

4.1 Experimentation Framework

Numerical results for solution quality evaluation were obtained considering a deterministic model, in which precedence relations between tasks and the execution time needed by each task are fixed and known beforehand. Although deterministic models are sometimes said to be unrealistic, since they ignore concerns such as deviations in task execution times due to interrupts and contention for shared memory, they make possible the static assignment of tasks to processors.

In an experimental study of heuristics, one of the most important decisions is the definition (or characterization) of algorithm performance. The literature contains a variety of criteria for evaluating a heuristic method and the choice of performance measure is a critical one. Based on the most common questions among investigators, Barr *et al.* [1] suggest that performance measures have tended to cluster in three areas: solution quality, computational effort, and robustness. Particularly in our case, experimentation has its goal in analyzing the relative solution quality improvements achieved by the tabu search scheduling algorithm. Numerical experimentation used the following hardware and software resources: (i) an IBM SP multicomputer composed of 32 RS6000 RISC microprocessors interconnected by a high-speed switch (bidirectional with nominal speed of 80 megabytes per second); (ii) the parallel implementation of the tabu search task scheduling algorithm, with the MS-MP strategy [19]; (iii) a benchmark of directed acyclic task graphs, previously used in the *ANDES* performance evaluation tool [14], modeling typical program structures such as diamonds, simple iterations, divide-and-conquer, and master-slave computations.

The eight representative task graphs, which compose the *ANDES* benchmark, are presented in Table 1 with their respective number of tasks, n; maximum parallelism (i.e., the largest number of tasks that can be executed in parallel), $maxpar$, and serial fraction (the fraction of total parallel execution time when just one task is executing even if infinite processors were available), F_s. Two different sizes (referenced by *size 1* and *size 2*) for each graph are taken into account, i.e. two different values for the number of tasks n for each graph structure. Computation costs are identical, i.e., tasks have unit time service demands. The graphs are to be scheduled on 2, 4, 8, 16, and 32 processors (with a single heterogeneous processor), with the processor power ratio equal to 2, 5, 10, and 20 (meaning that only one processor is heterogeneous among a subset of homogeneous ones). To achieve consistent conclusions on algorithm behavior, we have also used higher PPR values (30 and 40) during some of the experiments. Figure 1 shows the structures of these task graphs.

Graph	*size 1*			*size 2*		
	n	F_s	$maxpar$	n	F_s	$maxpar$
`Diamond3`	66	33.3	12	218	22.2	27
`Diamond4`	66	22.2	6	218	14.3	12
`Divconq`	46	22.2	16	190	15.4	64
`FFT`	83	28.5	16	194	25.0	32
`Gauss`	47	27.3	16	192	14.3	19
`MS3`	52	66.7	24	294	33.3	24
`PDE2`	50	40.0	16	194	40.0	64
`Warshall`	66	20.0	8	198	12.5	14

Table 1: Task graph benchmark characterization

4.2 Issues Concerning the Basic Greedy Algorithm

The evaluation of solution quality is based on the relative cost reduction, 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 [16] and s^* is the best solution found by the tabu search algorithm. The DES+MFT algorithm was selected because it has previously shown the best results published in recent literature.

This relative performance metric implies that the behavior of this greedy algorithm directly affects the evaluation of solution quality. In this sense, it is necessary to clearly understand the internal mechanisms which determine scheduling decisions in the DES+MFT algorithm. We notice that DES+MFT is a scheduler, which effectively considers the dynamic behavior of the parallel application. It takes into account the estimated data of the parallel application and the target machine, detailed in Section 2, to iteratively assign tasks to processors. On the other hand, the tabu search algorithm, due to its local search nature, searches for different assignments based exclusively on makespan improvements and solution search history. At each iteration, the tabu search algorithm evaluates all possible changes of the current schedule, not limited by any of the schedule system model characteristics, such as tasks precedence relations. Contrarily, the greedy algorithm has its scheduling decisions partially determined, and consequently restricted, by the task graph structure.

Before describing the DES+MFT algorithm, we primarily provide a few definitions. 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. In the latter case, the heterogeneous processor is able to execute any task in infinitesimal time, consequently serialization determines again the best performance.

DES+MFT stands for *Deterministic Execution Simulation with Minimum Finish Time*. 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. Scheduling decisions are made according to the minimum finishing time (MFT) for each schedulable task. 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 each iteration, 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 tabu search 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.

According to the above description, 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, as it will be shown in the next section.

4.3 Solution Quality

We present the numerical results obtained in the computational experiments with the tabu search algorithm, investigating its relative behavior with respect to the DES+MFT greedy algorithm, according to the variation of problem factors characterizing either the application or the heterogeneous processor set: (i) number of tasks in the precedence graph, reflecting the problem size; (ii) task graph structure; (iii) processor power ratio, quantifying the system heterogeneity; and (iv) number of processors, which can measure the level of system contention. The evaluation of solution quality is based on the relative cost reduction, $\mathcal{R}$, defined in Section 4.2. In Figures 2 and 3, bar graphs are used to illustrate solution quality, plotting $\mathcal{R}$ as a function of PPR and n, respectively, with different configurations for the number m of processors and task graph structures described in Section 4.1. Similar results have been observed for different types of task graph structures, thus: (i) `Diamond4` results are summarized by `Diamond3` curves, (ii) `FFT` curves summarizes also `PDE2` results, and (iii) `Warshall` results are summarized through `Gauss` graph curves.

Considering the serialization effect, described in Section 4.2, it seems reasonable to presume that for very high values of the processor power ratio and serial fraction, the best solution is, in fact, to overload the heterogeneous processor through the serialization. In this case, this occurs for both DES+MFT and tabu search algorithms, determining a low relative cost reduction. On the other hand, for very low values of PPR, which determine an almost completely homogeneous processor set, the greedy algorithm exploits well the potential application parallelism, thus the tabu search algorithm has a very narrow range of possible choices to improve the task allocation achieved during the construction of the initial solution. In these cases, relative cost reduction is also low. Thus, in certain ranges of PPR values in Figure 2, depending on the task structure, the tabu search algorithm demonstrates a better ability in distributing tasks across the processors, benefiting from the existing parallelism and system heterogeneity, and attaining significant relative makespan reduction with respect to the DES+MFT algorithm.

In most of the cases in Figure 2, the "mountain" shaped curves show a higher peak, corresponding to a PPR value called here PPR_{peak}, where the greedy algorithm has serialized the application, but the tabu search algorithm has found another assignment, which profits from the potential parallelism inherent in the parallel application. It was observed that PPR_{peak} is the first PPR value (among those considered in the experiments) where serialization is performed by the greedy algorithm. For PPR values higher than PPR_{peak}, serialization slowly becomes a more attractive solution for the scheduling problem and thus the improvements achieved by the tabu search algorithm relatively to the schedule obtained by DES+MFT algorithm are increasingly lower. Comparing the results between different graph structures of approximately the same size, the decreasing order of the relative cost reduction values would correspond to `Diamond3`, `Gauss` and `Divconq`, `FFT` and `MS3`.

`Diamond` structured task graphs, also referenced as *wavefront* (because parallelism grows from one to *maxpar* and then decreases back to one), show the highest peak, due to its strongly tied and intricate precedence constraints and also due to its uniformly varying parallelism according to each *layer* of tasks. Specifically for the `Diamond3` graph structure, we have measured the relative cost reduction $\mathcal{R}$ for all PPR values ranging from 2 to 50 (Figure 4). The relative cost reduction variation is not uniform, but it precisely identifies the PPR_{peak}.

`Divconq` applications are also wavefront, but have more independence between groups of tasks than `Diamond` graphs. `Gauss` graphs have layers of tasks with a parallelism level starting from a maximum value, which steadily decreases at each layer. In this sense, DES+MFT, at least in the beginning of the scheduling process, has greater possibilities of exploring the available parallelism, which in turn produces lower relative cost reduction values.

FFT is a typical iterative graph, such as PDE2, but with less precedence constraints than the latter. In these cases, there are many configurations of PPR and m where the relative cost reduction is null, reflecting that the tabu search algorithm has practically no choice for changes which effectively reduce the makespan of the parallel application. It is interesting to notice that, in many cases, the tabu search algorithm may in fact find many other possible assignments, but which produce the same overall effect in the execution of the application. This is also the case of the MS3 applications. This graph structure represents a typical master-slave application, with a very strong synchronization point followed by a pool of independent tasks (with no precedence constraints between them). This characteristic determines that the DES+MFT algorithm serializes the application only for extremely high PPR values, when this decision in fact produces the best final schedule. The tabu search algorithm is not capable of significantly improving the solution obtained by the greedy algorithm. So, the relative cost reduction for this kind of graph is extremely low along the entire range of processor power ratio values when compared to other graph structures.

For some graphs, such as Gauss, Divconq and FFT, the change in graph size determines a higher PPR_{peak} value. This can be explained if we notice that the larger problem size in these cases determines more available parallelism throughout the execution of the application. So, greater heterogeneity in the target machine (higher PPR value) is necessary for the DES+MFT algorithm to start serializing the application. This is not observed for applications with Diamond structure graphs. The latter tend to have a lower PPR_{peak}, even for different sizes of this same graph. For Diamond3, the PPR_{peak} is always around 5. We have also done some experiments with a larger Diamond3 graph (with 514 tasks) and the observed PPR_{peak} is again around 5 (Figure 5). This interesting behavior is explained if one accurately observes the structure of diamond-like graphs. These graphs starts with a parallelism level equal to one, which regularly increases to $maxpar$ and then regularly decreases again to one. In addition, precedence constraints are very intricate between tasks inside the diamond (unlike Divconq, also a diamond-shaped graph, but with a low number of precedence relations between the internal tasks). These intricate task precedence relations force the greedy algorithm to serialize the tasks for low PPR values, independently from the graph size. The presence of a common successor for every pair of internal tasks (i.e., inside the diamond) in Diamond graphs is the key factor for serialization as a domino effect. Divconq tasks have no common successors (except during the second half of the graph, when the division phase has finished) and its behavior is completely different from the typical diamond graphs.

In Figure 3, three different structures (`Diamond`, `FFT` and `Gauss`) were experimented with a fixed processor power ratio equal to five. As before, `FFT` structure presents the lowest relative cost reduction when compared to the other two task graphs, due to its iterative feature, which maintains the same parallelism level during the whole execution. The behavior of the relative cost reduction in `Diamond` graphs is the opposite of that observed for the `Gauss` task graph. In the first case, increasing problem size maintains the very intricate precedence relations between tasks, which determine low quality solutions obtained by the greedy algorithm. Thus, the tabu search algorithm can easily find better assignments than those found by DES+MFT, whose scheduling decisions are not re-evaluated after iteratively assigning processor to tasks. With a `Gauss` structured application, a larger number of tasks (with a PPR value which is not very high) determines that the greater available parallelism in the beginning of the application is almost fully utilized by the greedy algorithm. The tabu search algorithm again may find different assignments which are, however, almost as good as the one defined by the DES+MFT algorithm. For the above graphs and considering a PPR value equal to five, the experiments with `FFT` and `Gauss` graphs are to the left of their PPR_{peak}, while experiments with the `Diamond4` graph is exactly over or to the right of its PPR_{peak}.

The effect of increasing the number of processors is the reduction in system resource contention. A very low resource contention demands very accurate resource management from the scheduling algorithm, so that the heterogeneous processor does not become underutilized, reducing the benefit provided by heterogeneity. There is always a twofold commitment: profiting from system and application parallelism, which means spreading tasks over all processors; and profiting from system heterogeneity, which means to concentrate tasks on the heterogeneous processor. Thus, for very high resource contention (i.e., a small number of processors), the initial solution and the final solution found by tabu search achieve similar performance. In this case, the available parallelism is small and the number of possibilities for rescheduling is still incipient. With very low resource contention (i.e., a large number of processors), the makespan relative reduction is reduced and stabilizes thereof. Neither algorithm is able to make use of extra processors, because the application has attained its maximum parallelism, beyond which there is no benefit in providing more processors. Surely the makespan relative reductions are always subject to the PPR of the parallel machine.

5 Conclusions

We have presented a thorough evaluation of the quality of the solutions generated by a parallel tabu search algorithm for task scheduling on heterogeneous processors under precedence constraints. Solution quality was measured by the makespan reduction on a broad set of parallel applications, relatively to the results of the best greedy algorithm for the same problem. This relative cost reduction achieved maximum values of 40% in many of the studied cases. The numerical experiments, performed with an efficient parallel implementation, show that the tabu search algorithm is more robust towards problem characterization factors, achieving much superior results when compared to the greedy algorithm. In many cases, the latter is not capable of adequately profiting from inherent application parallelism and processor heterogeneity. Precedence relations between parallel tasks proved to have a significant influence in the scheduling decisions of the greedy algorithm, re-enforcing the superiority and robustness of the tabu search approach.

Due to the serialization effect, the tabu search algorithm demonstrates a better ability in distributing tasks across the processors for certain ranges of PPR values, benefiting from the existing parallelism and system heterogeneity, and attaining significant relative makespan reductions with respect to the DES+MFT algorithm. Comparing the results between different graph structures of approximately the same size, the best results were obtained for `Diamond3` task graphs, which presented the largest relative cost reductions, followed by `Gauss` and `Divconq`, `FFT` and `MS3`. Considering the variation of problem size n (number of tasks), three different structures (`Diamond`, `FFT` and `Gauss`) were experimented with a fixed processor power ratio equal to five. Here again, `Diamond` graphs showed the best results and `FFT` presented the lowest relative cost reduction when compared to the other graph structures.

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, "Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements", *OR Spektrum* 17 (1995), 113–123.

[3] T.G. Crainic, M. Toulouse, and M. Gendreau, "Towards a Taxonomy of Parallel Tabu Search Algorithms", *INFORMS Journal on Computing* 9 (1997), 61–72.

[4] T.G. Crainic and M. Toulouse, "Parallel Metaheuristics", Research report, *Centre de Recherche sur les Trnasports*, *Universitè de Montréal*, 1997.

[5] H. El-Rewini, "Partitioning and Scheduling", Chapter 9 in *Parallel and Distributed Handbook* (A.Y.H. Zomaya, ed.), 239 – 273, McGraw-Hill, 1996.

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

[7] 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.

[8] M.R. Garey and D.S. Johnson, "Strong NP-Completeness Results: Motivation, Examples and Implications", *Journal of the ACM* 25 (1978), 499–508.

[9] 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.

[10] F. Glover, "Tabu Search - Part I", *ORSA Journal on Computing* 1 (1989), 190–206.

[11] F. Glover, "Tabu Search - Part II", *ORSA Journal on Computing* 2 (1990), 4–32.

[12] 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.

[13] F. GLOVER and M. LAGUNA, *Tabu Search*, Kluwer, 1997, Boston.

[14] 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.

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

[16] 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.

[17] S.C.S. PORTO and D.A. MENASCÉ, "Processor Assignment in Heterogeneous Message Passing Parallel Architectures", *Proceedings of the Hawaii International Conference on System Science*, Kauai, 1993.

[18] 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.

[19] 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.

[20] M. REISER and S.S. LAVENBERG, "Mean Value Analysis of Closed Multichain Queueing Networks", *Journal of the ACM* 27 (1980), 313–322.

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

[22] M. TOULOUSE, T.G. CRAINIC, and M. GENDREAU, "Communication Issues in Designing Cooperative Multi-thread Parallel Searches", in *Meta-Heuristics: Theory and Applications* (I.H. Osman and J.P. Kelly, eds.), 50–522, Kluwer, 1996.

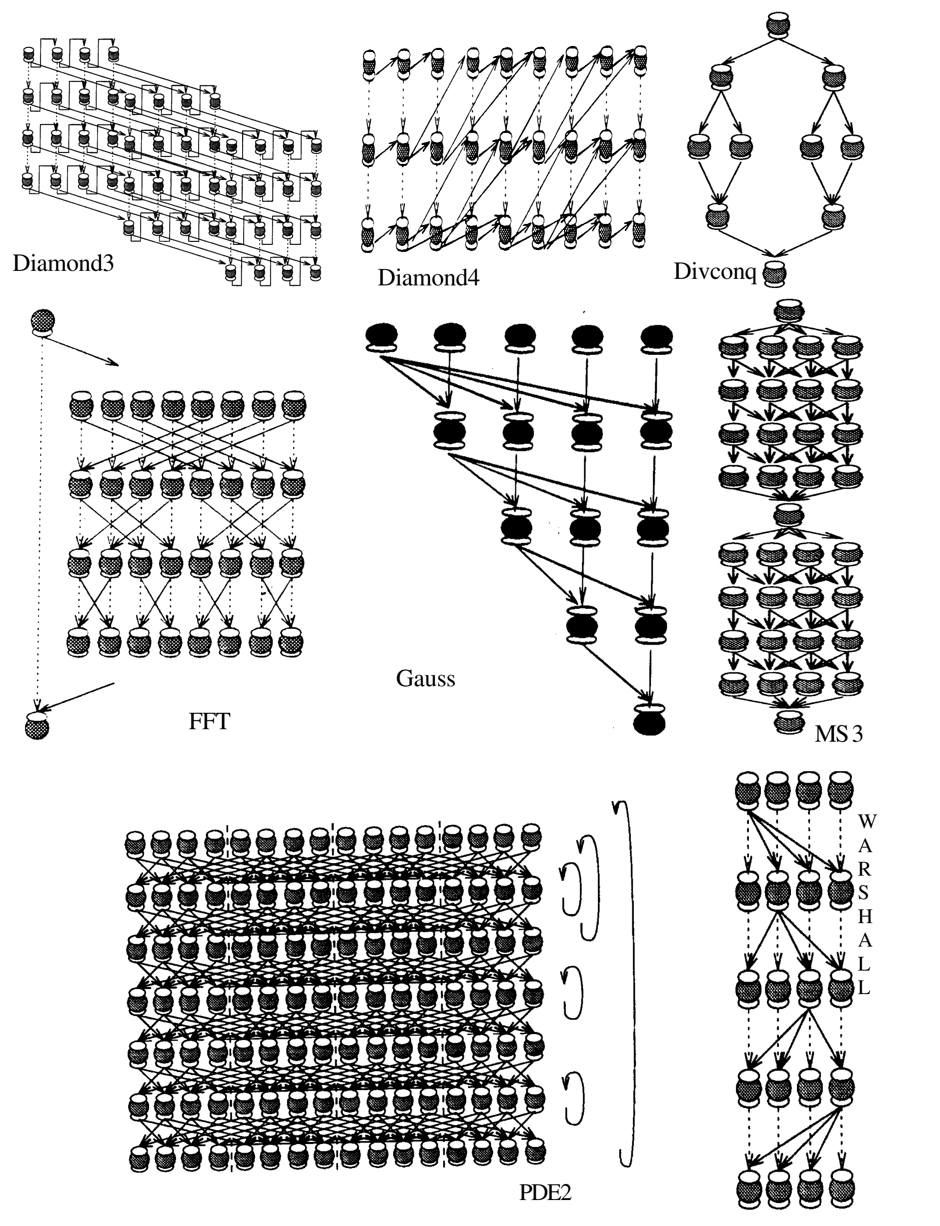

Figure 1: Task graph structures

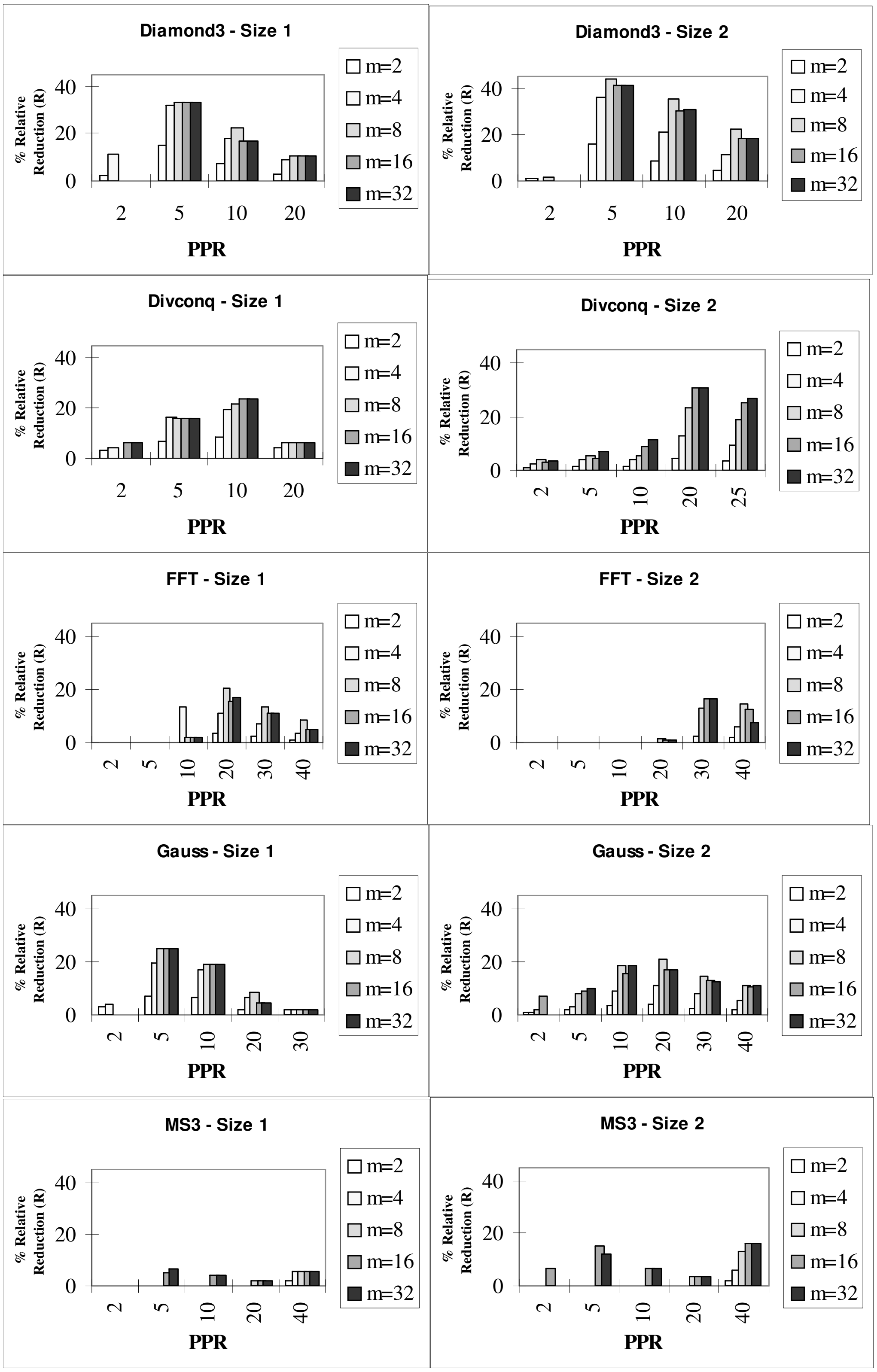

Figure 2: Relative cost reduction $\mathcal{R}$ versus PPR

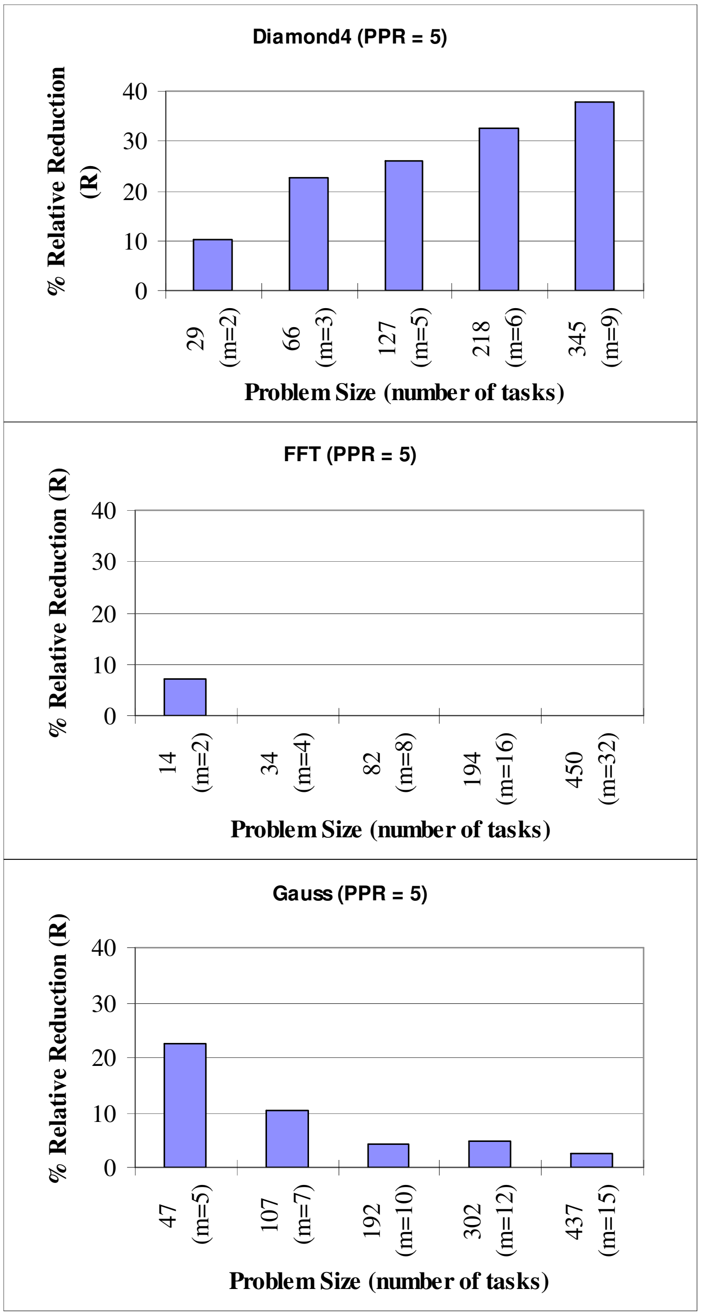

Figure 3: Relative cost reduction $\mathcal{R}$ versus problem size n

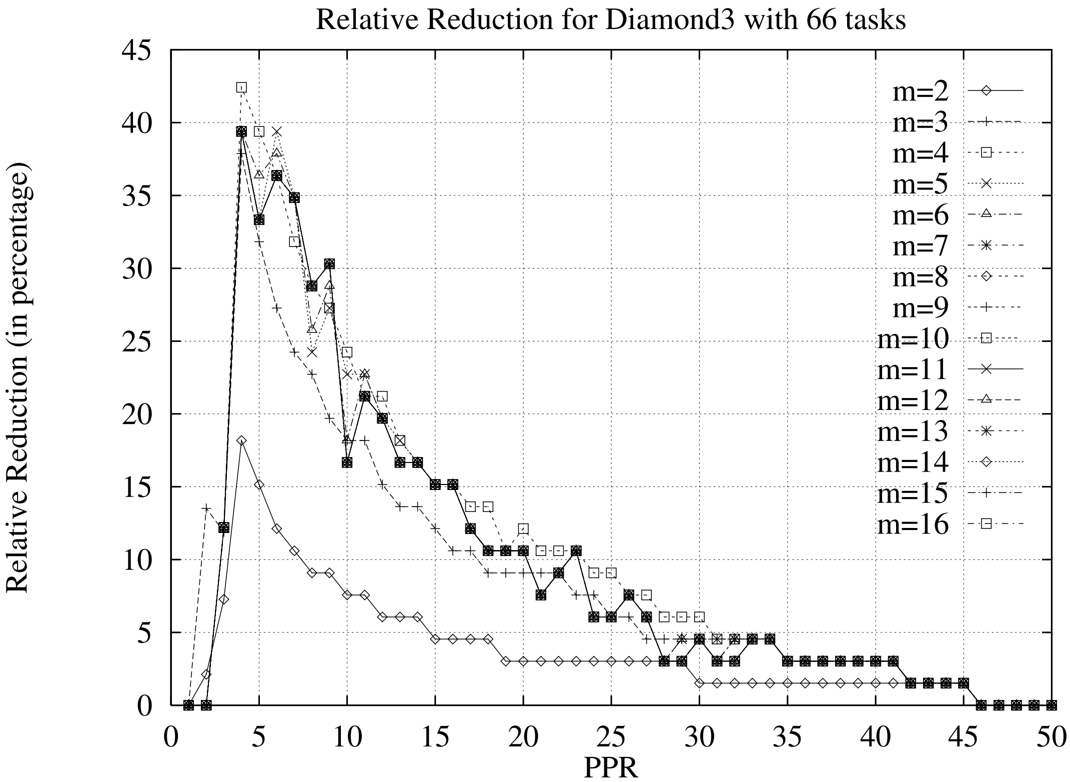

Figure 4: Detailed relative cost reduction $\mathcal{R}$ versus PPR for `Diamond3` graph (m corresponds to the number of processors to which the tasks are scheduled).

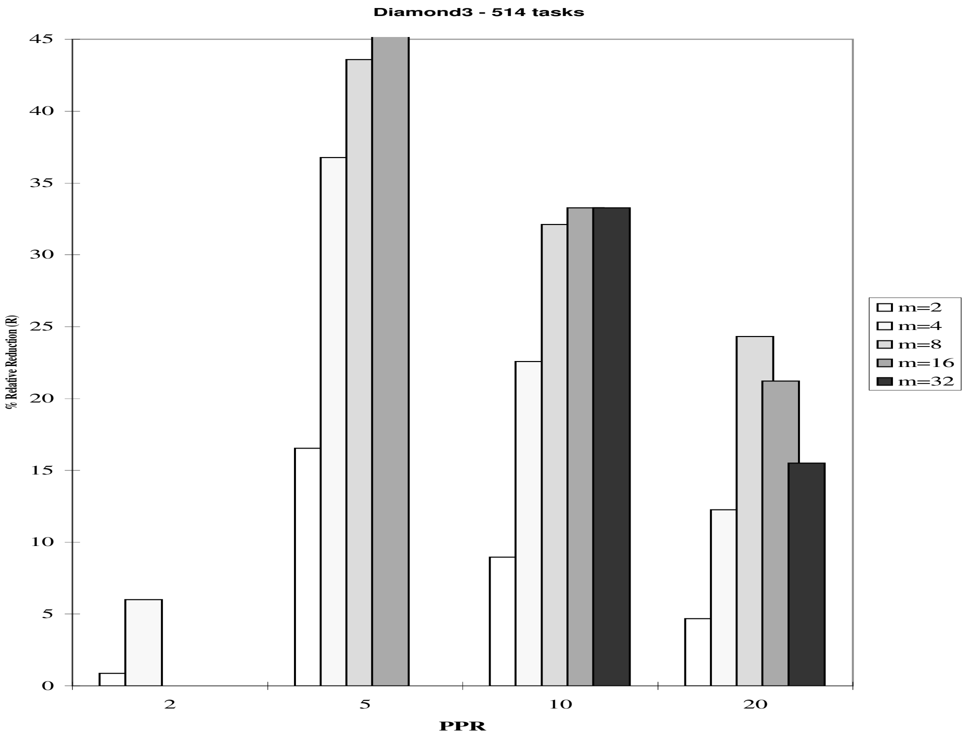

Figure 5: Relative cost reduction $\mathcal{R}$ versus PPR for a `Diamond3` graph with 514 tasks