Proceedings of the 5th WSEAS Int. Conf. on DISTANCE LEARNING AND WEB ENGINEERING, Corfu, Greece, August 23-25, 2005 (pp215-223)

# Data parallel scheduling of Operations in Linear Algebra on heterogeneous clusters

C. Morais
Instituto Politécnico de Bragança
Bragança, Portugal
Instituto de Engenharia Biomédica
Porto, Portugal

J. Barbosa
Faculdade de Engenharia da Universidade do Porto
Instituto de Engenharia Biomédica
Porto, Portugal

P. Tadeu
Instituto Politécnico de Bragança
Bragança, Portugal

***ABSTRACT***
The aim of data and task parallel scheduling for dense linear algebra kernels is to minimize the processing time of an application composed by several linear algebra kernels. The scheduling strategy presented here combines the task parallelism used when scheduling independent tasks and the data parallelism used for linear algebra kernels. This problem has been studied for scheduling independent tasks on homogeneous machines. Here it is proposed a methodology for heterogeneous clusters and it is shown that significant improvements can be achieved with this strategy.

***KEYWORDS***
static scheduling, parallel tasks, heterogeneous clusters.

## 1 Introduction

The aim of the work herein presented is to improve the performance of heterogeneous clusters in the processing of applications composed by linear algebra kernels. The optimization of the processing time of a given parallel application is achieved by obtaining an optimal scheduling for the tasks that form the algorithm. Static scheduling methods use Directed Acyclic Graphs (DAGs) to describe parallel algorithms, where the tasks to process and the precedence among them are represented by an acyclic graph. The scheduling consists on the distribution of the DAG nodes among the machine nodes, so that the *makespan* is minimum, this is the total length of the schedule. Finding a scheduling that minimizes the processing time of the parallel algorithm is a NP-complete problem [12]. Therefore, since the optimal scheduling is not feasible to obtain, many researchers have presented heuristic algorithms [1, 15, 21, 25]. These algorithms exploit the task parallel paradigm, this is one task to one processor.

The parallel implementation of dense linear algebra kernels in a distributed memory computer (cluster) consists in rewriting state of the art sequential algorithms in order to extract their intrinsic parallelism [10], by using the data parallel model. These kernels are highly constrained, having a predefined pattern of computation and communication, limiting the advantages of a DAG representation and the scheduling methods. Another approach more efficient for linear algebra kernels is to apply data parallelism by determining beforehand which data element goes to each processor. Several studies present solu-

tions for homogeneous as well as heterogeneous clusters [3, 6, 8].

The approach presented in this paper mixes both solutions referred to above, this is, task parallel and data parallel scheduling. The DAG represents an algorithm to be scheduled as a task parallel paradigm where each task can be a dense linear algebra kernel. Each task can be itself scheduled as a data parallel paradigm. This approach is also called as scheduling malleable tasks, this is scheduling tasks that can be executed on any number of processors with its execution time being a function of the number of processors alloted to it [18, 26, 14, 16].

The remaining of the paper is organized as follows: section 2 revises related work in scheduling DAGs, section 3 presents the methodology used in this paper, section 4 presents results and section 5 conclusions.

# 2 Scheduling strategies

The mixed scheduling strategy, combination of task and data parallel scheduling, appears to be a promising way to achieve better performance with dense linear algebra tasks. In order to obtain a methodology for the mixed scheduling it is required to explore the many heuristics developed for scheduling DAGs and the scheduling strategies used for the data parallelism paradigm on heterogeneous clusters.

## 2.1 Task scheduling

Task scheduling methods represent a parallel program by a DAG [20] such as the one of Figure 1. The nodes represent tasks to compute and the edges establish the precedence among tasks. In these graphs there is no cycles so that the path is always directed.

In the DAG of Figure 1 there are $4$ paths of processing, starting from the tasks of the first level to the last one. The longest path, corresponding to the most computation complexity, determines the total processing time of the algorithm and is called the *critical path*.

To obtain a scheduling for the parallel algorithm it is required to define the computational cost $C_j$ of each DAG task $T_j$ ($j = \{1, ..., m\}$, where $m$ is the number of tasks), and their communication requirements, for each processor of the machine $P_i$ ($i = \{1, ..., P\}$, where $P$ is the number of processors). The aim of the scheduling method is to minimize the total computational time which is a NP-complete problem [12]. Therefore, many researchers have presented heuristic algorithms to find a sub-optimal solution.

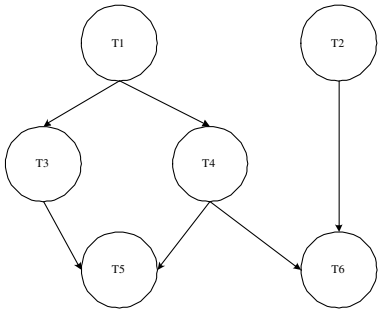


Figure 1: Direct Acyclic Graph

In the general case the cluster is built by machines of distinct processing capacities, i.e. heterogeneous clusters, where the homogeneous cluster is a particular case. The heuristics referred to below were proposed for homogeneous systems.

Task scheduling heuristics are based on the *critical path* (CP) analysis, and try to obtain a scheduling so that tasks on the CP determine the shortest possible execution delay for the whole program [21]. Another approach that take in consideration other DAG characteristics, called List Scheduling, creates a list of schedulable DAG nodes according to some priority assigned to each node. Then, until all nodes are scheduled, selects the node with highest priority and assign it to the processor that is most appropriated to process it. In [21] two list scheduling algorithms are presented: the Heavy Node First (HNF), that is based on a local analysis of each DAG level, and the Weighted Length (WL), that achieves better results and considers a global view of the DAG taking into account the relationship among nodes at different levels. In [23] several contention aware list scheduling heuristics are compared.

For heterogeneous systems there is also a considerable collection of heuristics that have been proposed. Some of them are referred next. In [17] a heuristic called Best Imaginary Level (BIL) is presented, which consists in defining the priority of each task in each DAG level. The

tasks are them sorted by decreasing priority and executed by this order. In [24] two heuristics are presented, the Heterogeneous Earliest Finish Time (HEFT) and the Critical Path on a Processor (CPOP) which are very similar. Both determine the tasks computational priority at each DAG level but use different criteria to select the processor to each task. In [22] the heuristic Generalized Dynamic Level (GDL) is presented. Boudet in [7] compares the heuristics referred to above.

In [9] static scheduling heuristics for independent tasks on heterogeneous clusters are described and compared. They have no precedence restrictions. From the heuristics that were compared the most successful ones are Max-Min, Min-Min and Genetic Algorithm.

In our study, the first algorithm to be defined implements only task parallelism. As a first approach, the HNF algorithm is used to process the DAG and the max-min heuristic is used to schedule on heterogeneous clusters. Although HNF is not the best algorithm it gives better results then the Critical Path analysis and is the simplest to implement. The algorithm is described as follows:

**Algorithm1.** First, use HNF algorithm to obtain on each DAG level the list of independent tasks ordered by priority; in this case, the priority is given by the computational complexity. Second, for each set of independent tasks use max-min heuristic to obtain a schedule for the heterogeneous cluster.

## 2.2 Scheduling with data parallelism

The schedule in a data parallel implementation of a linear algebra kernel consists in determining the data elements to assign to each processor in the cluster. In a homogeneous cluster the block cyclic distribution is used [10, 11] and guarantees that the computational complexity is uniformly distributed among the processors. Therefore, the load balancing can be achieved, minimizing the processing time of the linear algebra kernel.

On heterogeneous clusters the parallel implementation of a linear algebra algorithm presents additional difficulties when compared to other classes of algorithms. Due to the fact that the computational load for data matrix columns increases along the matrix [3], a fine tuned load assignment and distribution is required. In [3, 6, 8] two static scheduling algorithms are proposed in order to minimize the processing time. These algorithms estimate the processing time of the linear algebra kernels based on the following computational model.

### 2.2.1 Computational model

The computational platform is a distributed memory machine composed by $P$ processors of possibly different processing capacities (heterogeneous cluster), connected by a switched Ethernet. The computational model that supports the estimation of the processing time, for each task, is based on the processing capacity $S_i$ of processor $i$ ($i \in [0, P-1]$) measured in $Mflop/s$, the network latency $T_L$ and bandwidth $\omega$ measure in $Mbit/s$. The TCP/IP protocol divide the messages into packets, being required to consider $k$ latency times for a message divided into $k$ packets. The total computation time is obtained by summing the time spent communicating $T_C$ and the time spent in parallel operations $T_P$. The time required to transmit a message of $b$ elements is $T_C = kT_L + b\omega^{-1}$. The time required to compute the pure parallel part of the code, without any sequential part or synchronization time, on $P$ processors is $T_P = f(n)/\sum_{i=1}^{P} S_i$. The numerator $f(n)$ is the cost function of the algorithm measured in floating point operations. As an example, for a matrix-matrix multiplication of $(n, n)$ matrices, $f(n) = 2n^3$.

The second algorithm to be defined implements only data parallelism. It is assumed that the application is represented by a DAG and it is regular, this is there exists dependencies among tasks and the execution time can be estimated at compile time or before starting processing.

**Algorithm2.** First, order the processors by decreasing processing capacity $S_i$. For all tasks, use the computational model to estimate the processing time when using from 1 to $P$ processors. Choose the number of processors that give the minimum processing time for the task (the fastest ones). Second, use the HNF algorithm to determine the order by which the tasks are executed.

## 2.3 Data and task parallelism

The idea of combining task and data parallelism is based on the fact that individual tasks of an application may not use the computational power available on a distributed

memory multicomputer, because with data parallelism the performance improvement does not increase significantly when more processors are used. Better results are achieved if the number of processors used on each data parallel task are controlled and if the tasks are concurrently executed. In [19] this methodology is studied as a compiler optimization. In [2] it is compared four systems that explore the use of task and data parallelism.

As referred to above this approach is also called as scheduling malleable tasks. The existing work is for homogeneous computers and is based on a two phase solution. First, selection of an allotment for each task and second, solve the resulting non-malleable scheduling problem, which is similar to a 2-d strip packing problem. A non-malleable task is such that it requires a specific number of processors for a specific units of time. In [26] it is presented an efficient low cost scheduling of independent malleable tasks on homogeneous clusters, divided in the two phases referred to above. In the second phase it is used a simple list algorithm. It is assumed that the time for executing a task is a non-increasing function of the processors used. In [18] it is proposed two polynomial-time approximation algorithms for independent non-malleable and malleable tasks with linear speedup and individual deadlines. The aim is to maximize the total work performed by the tasks which complete their executions before deadlines. In [14] it is proposed an asymptotic fully polynomial time approximation scheme for scheduling a set of $n$ independent malleable tasks on an arbitrary number $m$ of identical processors. In [5] it is proposed a polynomial algorithm for scheduling preemptive parallel tasks for the case of processor availability only in restricted intervals of time. In [16] it is presented an approximation algorithm for scheduling dependent malleable tasks restricted to the structure of a tree. The problem is formulated as a constrained allotment problem, and solved using a dynamic programming algorithm for the case of trees. In this case the two phases are merged into one.

The problem studied in this paper differs from the work referred to above mainly in two aspects. First, the processing time does not decrease always as the number of processors increases and second, the target cluster is heterogeneous. Section 3 presents this scheduling algorithm.

# 3 Scheduling parallel tasks

On the heterogeneous cluster, contrary to the homogeneous case, assigning one processor to a task may influence the processing time of another that will execute simultaneously, because the remaining processors may have less capacity. If the computational model is used to compute the set of processors for each task as in Algorithm2, it would produce unrealistic scheduling because it is selecting the fastest ones for all tasks. The approach proposed here (Algorithm3) estimates the processing capacity that minimizes the execution of each task, called best processing capacity ($BPC$), independently of the processors required to achieve that capacity. It is important to note that less or more capacity than $BPC$ may result in an increase of the task processing time.

**Algorithm3.** First, for all tasks, estimate the ($BPC$). Second, use the HNF algorithm to determine the order by which the tasks are executed.

The best processing capacity ($BPC_j$) of task $j$ indicates the processing capacity to use in order to minimize its processing time on the heterogeneous cluster. The aim is not to specify the processors to use but the capacity required by each task. Since the processors to be used are not known a priori, the computation is based on an average processor capacity ($\overline{S}_M$), so that a *near optimal* estimation is obtained. The following equations define $\overline{S}_M$.

$$S_T = \sum_{i=1}^{P} S_i \tag{1}$$

$$\overline{S}_M = \frac{S_T}{P} \tag{2}$$

On equation 1 and 2, $S_T$ is the total processing capacity and $P$ the number of processors available on the heterogeneous cluster. The machine is now considered homogeneous with $P$ processors of $\overline{S}_M$ capacity. For each task $j$, the estimated processing time is computed, when using from 1 to $P$ processors, based on the computational complexity $f(n)$ of the task. For each machine added, the processing capacity increases by $\overline{S}_M$ $Mflops$ and the number of machines used by 1. This influences the estimation of the processing time by reducing the parallel execution time and by increasing the communications required to process the task. If the minimum is

found for $p$ processors, then the best processing capacity is $p*\overline{S}_M Mflops$. This estimation is not so accurate as the one made by Algorithm2 but it gives an estimation closed to the capacity required for the task so that it is useless to use more capacity. If the standard deviation of processor capacity is high it may be required a second step, this is, a reevaluation of $\overline{S}_M$ with a number of processors around $p$, the last solution.

The scheduling problem is now reduced to the scheduling of independent tasks, that require a specific processing capacity, and restricted to the maximum processing capacity available on the machine ($S_T$). This is a 2-d packing problem [13].

The packing heuristic used is similar to the BLF (Bottom Line Fill) with a particular modification; it is allowed to overload the machine if only 10% of the task is above $S_T$. The practical effect is an increase on the task processing time. The BLF algorithm with this modification is: a) select the heaviest task and put it at the bottom line; b) if more than 10% of the task is over $S_T$, then select the next heaviest task; c) go back to a) until all tasks are placed.

Note that if one single task requires more than $S_T Mflops$, it would be scheduled to execute alone in the machine. In this case Algorithm3 is reduced to Algorithm2.

Figure 2 shows an example of independent tasks to be scheduled. The block height represents the $BPC$ computed for the task ($Mflops$) and the block length represents the processing time of the task when $BPC$ is used.

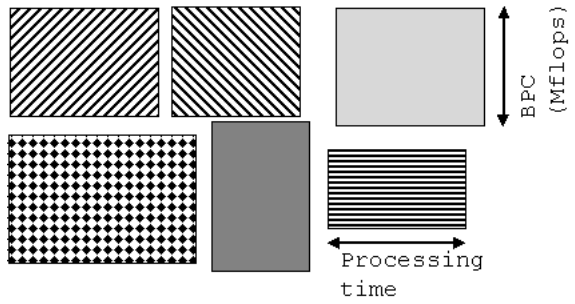


Figure 2: Example of tasks to schedule

For each task, the processors are selected until the sum of their capacities is approximated to the $BPC$ capacity required. Figure 3 illustrates the assignment of the scheduling algorithm for the 6 independent tasks of Figure 2. Only two tasks are executed in the first stage, and one of them overloads the machine. Since the overload is below a threshold (considered here to be 10%) the option was to use less capacity than $BPC$ and therefore extend the processing time. After that the machine is never used at full load because the starting time of tasks were postponed.

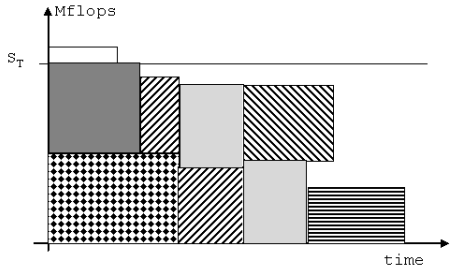


Figure 3: Scheduling example

It is important to note that the groups of processors used may change between DAG levels (sets of independent tasks). The scheduler keeps a record of the instant each processor will be free due to computations from former levels, so that it can see the machine as a whole and select the capacity that will minimize the processing time for the actual DAG level.

# 4 Results

This section presents results of the comparison of the scheduling methods described above for a heterogeneous machine composed by 40 processors, connected by a $100Mbit$ Ethernet. Figure 4 shows the processors capacities. The results where obtained by simulating in Matlab the cluster behavior and by estimating the processing time of the dense linear algebra kernels as exposed in [4], which uses the computational model present above. The compared methods were:

- Task scheduling, described by Algorithm1;
- Process one task at a time on the cluster in a data parallel implementation, described by Algorithm2;
- Combination of task and data parallelism, described by Algorithm3.

The results presented are referred to the DAG of Figure 5 that is a common one on the computer vision field. The input for each path are matrices that can be of different sizes. The DAG nodes are linear algebra kernels, namely LU factorization ($T_1, T_4$), tridiagonal reduction ($T_2, T_5$) and QR reduction ($T_3, T_6$). The last task computes the correlation between the results of the two independent paths.

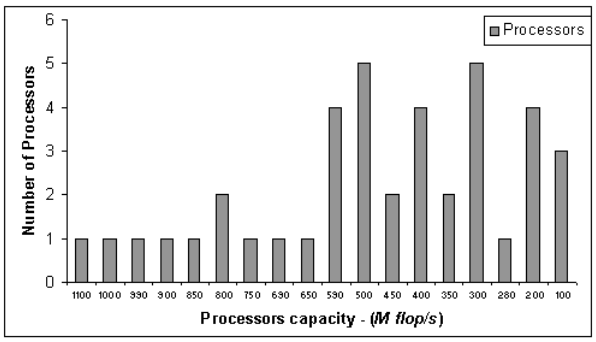


Figure 4: Processing capacity of the heterogeneous computer nodes

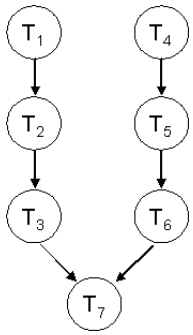


Figure 5: DAG for the matching algorithm

The test cases were matrices of different dimensions, from few hundreds to thousands of elements, and different combinations in the same problem in order to observe the scheduler behavior.

Figure 6 shows results for matrices of small size. For those matrix dimensions the processing times are small but it can be seen the significant difference between Algorithm1 and the other two. This is, the exclusively task parallel scheduling is the worst, as expected, since each task

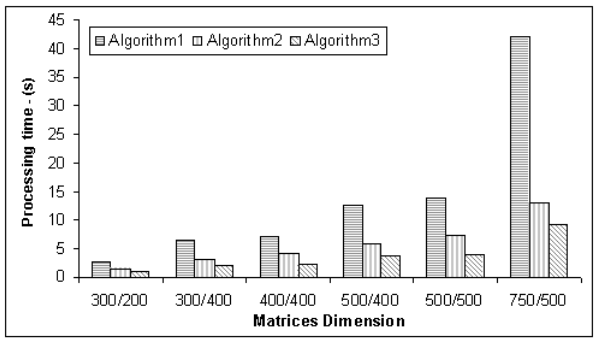


Figure 6: Comparison of algorithms 1, 2 and 3

is computed by only one processor. Increasing the matrix dimension this difference is amplified, therefore to keep the graphs readable it was decided to not include results for Algorithm1 for higher dimension problems. Figure 7 shows results comparing Algorithm2 and Algorithm3 when the input matrices are greater than $2500^2$ elements. It can be seen that Algorithm3 achieves always better results than Algorithm2.

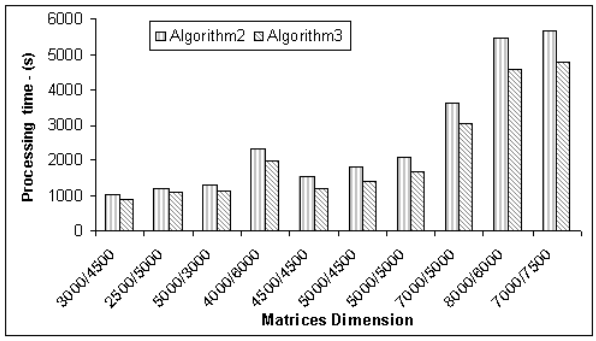


Figure 7: Comparison of algorithms 2 and 3

To measure the improvement achieved by Algorithm3 an $Improvement$ index is defined according to equation 3. Figure 8 shows the improvement in percentage. A detailed analysis of the results leads to the conclusion that the improvement is bigger when the two input matrices are of similar size.

$$Improvement = \frac{T_{Algorithm2} - T_{Algorithm3}}{T_{Algorithm2}} \qquad (3)$$

When there is a significant difference between the two matrices, the biggest determines the total processing time and the improvement is mainly due to the smaller matrix. The processing time of the smaller matrix is less significant to the total processing time and therefore the improvement is also less significant.

When they are of similar size, the improvement is of the same scale as the matrix that imposes the total processing time, and therefore the improvement is more significant.

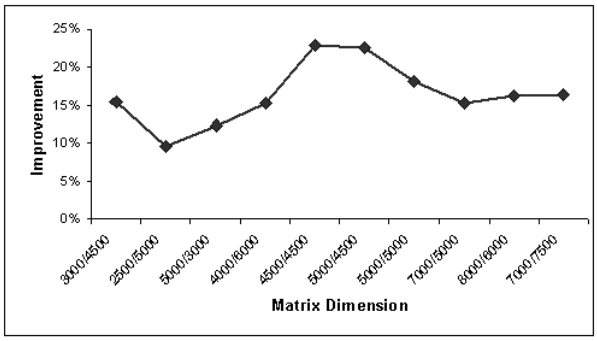


Figure 8: Improvement achieved with $Algorithm3$

From Figure 8 it can be seen that the improvement achieved for the DAG of Figure 5 is always above 9.5% and goes as high as 22.8%. This shows that the scheduling obtained with Algorithm3 achieves better performances on heterogeneous clusters. The amount of improvement achieved depends on the machine size and on how well the independent tasks can maximize the machine capacity as shown before in Figure 3.

In order to test the methodology for DAGs with other configurations, some tests were carried out for the DAGs of Figure 9. Results for DAG1, presented in Figure 10, were taken for a machine with 45 processors and $\overline{S}_M = 805Mflop$. The cases tested refer to tasks whose matrices sizes were [1000,2000], [1000,1000], [1000,1500], [3000,4000], [3000,5000] for case 1, 2, 3, 4 and 5 respectively. The improvements achieved were 33.8%, 51.3%, 41.8%, 11.0% and 1.7% respectively. Again, the improvements were significant for all cases except case 5. For the latter the machine capacity is too low in order to allow task parallelism, resulting in a schedule equivalent to Algorithm2.

Results for DAG2, presented in Figure 11, were taken for a machine with 50 processors and $\overline{S}_M = 700Mflop$. The cases tested refer to tasks whose matrices sizes were [3000,4000], [3000,3000], [2500,5000] and [4000,5000] for case 1, 2, 3, 4 and 5 respectively. The improvements achieved were 19.9%, 21.4%, 16.8%, 16.2% and 10.3% respectively. Again, the improvement was positive in all cases.

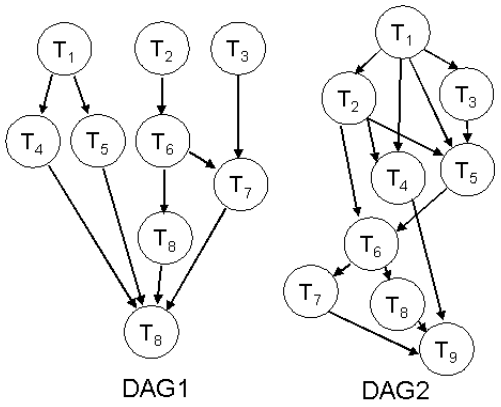


Figure 9: DAGs to test the scheduling methodology

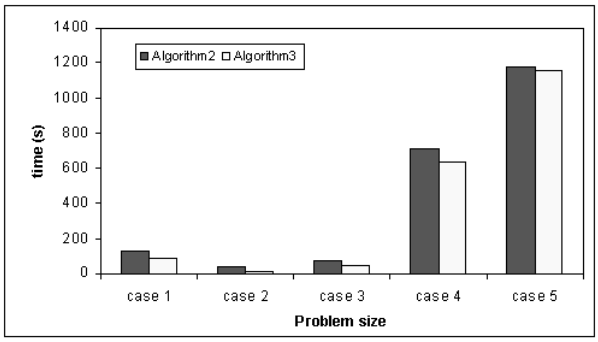


Figure 10: Results for DAG1

# 5 Conclusions

In this paper, two static scheduling strategies were compared, namely data parallelism and a mixed task and data parallelism (malleable tasks), proposed here for heteroge-

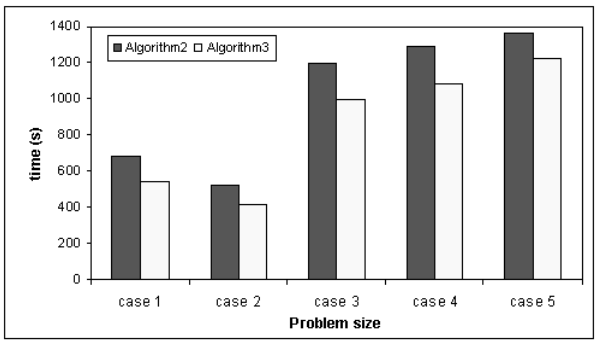


Figure 11: Results for DAG2

neous clusters. The DAGs considered are composed by dense linear algebra kernels.

From the simulation results obtained for a heterogeneous cluster it can be concluded that the mixed strategy that combines task and data parallelism achieves better performance in the execution of the DAGs considered, mainly due to a better resource utilization. The amount of capacity, in $Mflops$, that minimizes the processing time of each task is computed. Then, the algorithm allocates capacity for all tasks, instead of processors, restricted by the maximum capacity available. Depending on the requirements of each task and the capacity available, the resulting schedule can achieve high improvements if it can execute more than one task simultaneously, using data parallelism. In the worst case, the result is the execution of only one task at a time, using data parallelism.

The mixed strategy proposed does not require a static subdivision of processors. The groups of processors into which the machine in divided in each DAG level can change from one level to the other, thus allowing a better use of the machine and consequently achieving improvements in processing time.

# References

[1] A. K. Amoura, E. Bampis, and J.-C. König. Scheduling algorithms for parallel gaussian elimination with communication costs. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):679–686, July 1998.

[2] H. Bal and M. Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, 6(3):74–84, 1998.

[3] J. Barbosa, J. Tavares, and A. Padilha. A group block distribution strategy for a heterogeneous machine. In M. H. Hamza, editor, *Applied Informatics*, pages 378–383. IASTED, ACTA Press, February 2002.

[4] J. Barbosa, J. Tavares, and A. J. Padilha. Linear algebra algorithms in a heterogeneous cluster of personal computers. In *Proceedings of 9th Heterogeneous Computing Workshop*, pages 147–159. IEEE CS Press, May 2000.

[5] J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and P. Maczka. Scheduling multiprocessor tasks on parallel processors with limited availability. *European journal of Operational Research*, (149):377–389, 2003.

[6] V. Boudet, F. Rastello, and Y. Robert. A proposal for a heterogeneous cluster ScaLAPACK(dense linear solvers). In *Proceedings of PDPTA*. CSREA Press, 1999.

[7] V. Boudet and Y. Robert. Scheduling heuristics for heterogeneous processors. In Arabnia HR, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques end Applications (PDPTA 2001)*, pages 2109–2115, June 2001.

[8] Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, and Frédéric Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2):197–213, 1999.

[9] T. Braun, H. Siegel, N. Beck, L. Bõlõni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Parallel and Distributed Computing*, 61:810–837, 2001.

[10] J. Choi, J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. The design and implementation of the ScaLAPACK LU, QR, and

CHOLESKY factorization routines. *Scientific Programming*, 5:173–184, 1996.

[11] J. Dongarra and D. Walker. The design of linear algebra libraries for high performance computers. Technical Report LAPACK Working Note 58, University of Tennessee, Knoxville, June 1993.

[12] M. Garey and D. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[13] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128:34–57, 2001.

[14] K. Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39:59–81, 2004.

[15] Y.-K. Kwok. Parallel program execution on a heterogeneous pc cluster using task duplication. In *Proceedings of 9th Heterogeneous Computing Workshop*, pages 364–374. IEEE CS Press, May 2000.

[16] R. Lepère, G. Mounié, and D. Trystram. An approximation algorithm for scheduling trees of malleable tasks. *European journal of Operational Research*, (142):242–249, 2002.

[17] Hyunok Oh and Soonhoi Ha. A static scheduling heuristic for heterogeneous processors. In *Proceedings of Europar'96*, volume 1123. Springer Verlag, August 1996.

[18] Oh-Heum and K-Y Chwa. Scheduling parallel tasks with individual deadlines. *Theoretical Computer Science*, 215:209–223, 1999.

[19] S. Ramaswamy, S. Sapatnekar, and P. Banerjee. A framework for exploiting task and data parallelism on distributed memory multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1098–1116, November 1997.

[20] V. Sarkar. *Partitionning and scheduling Parallel Programs for Execution on Multiprocessors*. MIT Press, Cambridge MA, 1989.

[21] B. Shirazi, M. Wang, and G. Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributing Computing*, 10:222–232, 1990.

[22] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions os Parallel and Distributed Systems*, 4(2):175–187, 1993.

[23] O. Sinnen and L. Sousa. Comparison of contention aware list scheduling heuristics for cluster computing. In *Workshop on Scheduling and Resource Management for Cluster Computing (ICPP)*, pages 382–387. IEEE CS Press, September 2001.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *8th Heterogeneous Computing Workshop*. IEEE CS Press, 1999.

[25] H. Topcuoglu, S. Hariri, and M.-Y Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[26] Denis Trystram. Scheduling parallel applications using malleable tasks on clusters. In *15th International Conference on Parallel and Distributed Processing Symposium*, 2001.