Asymptotically optimal algorithm for Laplace task graphs on heterogeneous platforms

O. Beaumont, P. Ramet and J. Roman
LaBRI UMR CNRS 5800, ENSEIRB, University Bordeaux 1
INRIA Futurs ScAlApplix Project *
{obeaumon|ramet|roman}@labri.fr

May 16, 2003

Abstract

In this paper, we focus on the scheduling of Laplace task graph on a general platform where both communication links and processing units are heterogeneous. In this context, it is known that deriving optimal algorithm, in the sense of makespan minimization, is NP-Complete, and several inapproximation results have been proved. Nevertheless, we provide an asymtotically optimal algorithm in this general context. Moreover, we expect that this methodolgy can be extended to more general task graphs, especially for nested loops where the inner-most loop is parallel.

* http://www.labri.fr/scalapplix

1 Introduction

Parallel distributed memory machines improve performances and memory capacity but their use induces an overhead due to communications. To obtain programs that perform and scale well, this overhead must be hidden. Several solutions exist. The choice of a good data distribution can be used to decrease the number and the size of communications, and to obtain large grain computations. Moreover, depending of the dependences within the code, asynchronous communications can be used to overlap communications with computations. In the context of heterogeneous computing, an additional difficulty comes from load balancing.

Pipelined algorithms have been widely studied in homogeneous architectures (see [10] and references therein). We focus on the problem of finding good data distributions in order to minimize the execution time when the size of the task graph is large, on heterogeneous platforms such as clusters of SMP nodes or heterogeneous networks of workstations.

Scheduling general task graphs is known to be NP-Complete in the context of heterogeneous resources (see for instance [7] and [9]). In this paper we consider more specifically the scheduling of Laplace task graph on a platform where both communication links and processing units are heterogeneous. The general framework of Laplace task graph is depicted in Figure 1, where the nodes represent the updates and the edges represent the dependencies. Each element is updated using its left and upper neighbors. This graph can model for instance the main loop of ADI (Alternative Direction Implicit) algorithm [8].

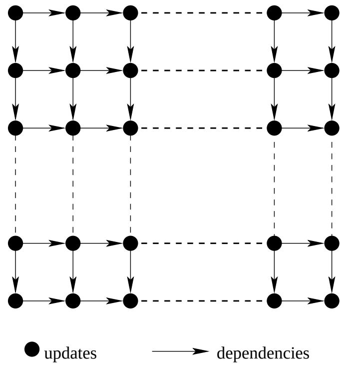

Figure 1: Laplace task graph.

The remainder of the paper is organized as follows. In Section 2, we present some related work, dealing with close problems (pipeline communications on heterogeneous networks) or using close techniques in order to prove asymptotical optimality. In Section 3, the model of the platform is presented, and the asymptotically optimal algorithm is described and proved. At last, we give some conclusions and future work in Section 4.

2 Related Work

2.1 Pipelined communications on heterogeneous networks

In [10], the authors study the minimization of the latency of a message through a network of store-and-forward stages. The platform is characterized by the number n of pipeline stages, and stage i is characterized by g_i, the fixed per-fragment overhead (its latency) and G_i, the time transfer per-byte (the inverse of the bandwidth). It is worth of interest to note that processing can be included in this model, by adding stages where the $g_i = 0$ and G_i represents the time necessary to process one element.

The main contribution of [10] is to provide a polynomial algorithm when the size and the number of fragments is fixed, i.e. re-fragmentation and reassembling is not possible between the stages. In this context, let us denote by B the size of the message, k the number of fragments and x_i the size of the ith fragment. The following linear program (in rational numbers) provides the optimal solution for the size of the fragments.

Minimize $\tau_{k,n}$,
subject to

$$\begin{cases} \forall\, 1 \le j \le k, & \tau_{1,j} \ge \sum_{h=1}^{j} (g_h + x_0 G_h) & (\text{ first fragment }) \\ \forall\, 1 \le i \le k,\ \forall\, 2 \le j \le n & \tau_{i,j} \ge \tau_{i,j-1} + (g_j + x_i G_j) & (\text{ at least } g_j + x_i G_j \text{ between stage } j-1 \text{ and } j) \\ \forall\, 2 \le i \le k,\ \forall\, 2 \le j \le n & \tau_{i,j} \ge \tau_{i-1,j} + (g_j + x_i G_j) & (\text{ at least } g_j + x_i G_i \text{ between fragment } i-1 \text{ and } i) \end{cases},$$

where $\tau_{i,j}$ denotes the exit time of the ith fragment from the jth stage, and therefore $\tau_{k,n}$ denotes the completion time of the algorithm.

When the size and the number of fragments is not fixed, i.e. re-fragmentation and reassembling is possible between the stages, the solution is only known under strong assumptions and with limited pipeline sizes. In [10], the solution for two-stages pipelines is given. For three-stages pipelines, closed form solutions are derived for very special cases only. In particular, a strict ordering of the pipeline stages is required (a stage is said to be slower than another stage if and only if both its latency is larger and its bandwidth smaller). In this context, solutions are known for "fast-slow-fast" and "fast-slow-slower" pipelines.

2.2 Asymptotically optimal packet routing

The packet routing problem is the following: let $G = (V, E)$ be a non-oriented graph modeling the target architectural platform, and consider a set of same-size packets to be routed through the network. Each packet is characterized by a source node (where it initially resides) and a destination node (where it must be located in the end). For each pair of nodes (v_k, v_l) in G, let n_{kl} be the number of packets to be routed from v_k to v_l. Bertsimas and Gamarnik [4] introduce a scheduling algorithm which is asymptotically optimal when $n = \sum n_{kl} \to +\infty$.

The algorithm consists in three main phases. First, a linear program is solved, where, so to speak, temporal constraints have been removed: it is never written than a packet must have reached a node before leaving it. This solution of this linear program provides a lower bound T_{OPT} for the completion time of the packet routing problem. Then, a periodic algorithm is derived from the solution of the linear program. It roughly consists in $\sqrt{T_{\text{OPT}}}$ periods of duration $\sqrt{T_{\text{OPT}}}$, and in each period, the packets are routed using the rounded behavior given by the linear program. At last, the (few) remaining packets are routed sequentially.

The makespan of this schedule is $T_{\text{OPT}} + O(\sqrt{T_{\text{OPT}}})$, hence the asymptotic optimality.

2.3 Divisible load scheduling

The divisible load model has been widely studied in the last several years, and popularized by the landmark book written by Bharadwaj, Ghose, Mani and Robertazzi [5]. A divisible job is a job that can be arbitrarily split in a linear fashion among any number of processors. This corresponds to a perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of processors. The applications of the divisible load model encompass a large spectrum of scientific problems, including among others Kalman filtering, image processing, video and multimedia broadcasting, database searching, and the processing of large distributed files (see [5] for more examples).

On the practical side, the divisible load model provides a simple yet realistic framework to study the mapping on independent tasks on heterogeneous platforms. The granularity of the tasks can be arbitrarily chosen by the user, thereby providing a lot of flexibility in the implementation tradeoffs. On the theoretical side, the success of the divisible load model is mostly due to its analytical tractability. Optimal algorithms and closed-form formulas exist for the simplest instances of the divisible load problem. This is in sharp contrast with the theory of task graph scheduling, which abounds in NP-Completeness theorems [6] and in inapproximability results [7].

In [3], the target computing platform is a heterogeneous master/worker platform, with p worker processors labeled $P_1, P_2, \ldots, P_p$. The master P_0 sends out chunks to workers over a network. The master processor can distribute the chunks to the workers in a single round, (also called installment in [5]), so that there will be a single communication between the master and each worker.

For large workloads, the single round approach is not efficient, because of the idle time incurred by the last processors to receive their chunks. In order to minimize the makespan, i.e. the total execution time, it is proposed in [3] to send the chunks to the workers in multiple rounds: the communications will be shorter (less latency) and pipelined, and the workers will be able to compute the current chunk while receiving data for the next one. As in the case of packet routing, an asymptotically optimal algorithm is derived, using $\sqrt{T_{\text{OPT}}}$ periods of duration $\sqrt{T_{\text{OPT}}}$, whose makespan is of order $T_{\text{OPT}} + O(\sqrt{T_{\text{OPT}}})$, hence the asymptotic optimality.

3 Asymptotically optimal algorithm for Laplace task graphs

3.1 Modeling of the platform

In this section, we consider the scheduling of Laplace task graphs on heterogeneous platforms. The platforms we consider are very general, since both communication links and processing units are heterogeneous. The only assumption we make is the possibility of overlapping communications with computations, i.e. the existence of a "front end". More precisely, we model the platform as a graph $G = (V, E, w_i, c_{i,j})$. The nodes V_i ($|V| = m$) represent the processors, labeled with w_i, their processing power, i.e. the necessary time to process one update. The edges $(V_i, V_j) \in E$ represent the communication links, labeled with $c_{i,j}$, i.e. the necessary time to transfer one element from V_i to V_j.

In order to derive an asymptotically optimal algorithm, we need a cyclic ordering of the nodes of the platform. Since it is known to be NP-Complete to find an edge disjoint Hamiltonian path in G [6], and that such a path may well not exist, we rely on a very simple heuristic. This heuristic may lead to a bad ordering, inducing a lot of communications. Nevertheless, as we will prove it in Section 3.5, it is always possible to overlap communications with computations, so that this bad ordering has little impact on the overall performance of the algorithm.

The heuristic we rely on is the following. Let $T = (V, E')$ be a spanning tree of G, rooted in V_0. We build the path using depth first order of the tree T. This path is denoted by $P = (V_0, V_{i_1}, \ldots, V_{i_K}, V_0)$. For sake of simplicity, we consider a reordering of the indexes of the nodes, so that the nodes first appear in the path P in lexicographic order, namely, if $f(i)$ denotes the smallest index k such that $f(i) = i_k$, then $\forall i,\ f(i) < f(i+1)$.

In the algorithm we propose in Section 3.5, each node V_i sends a message to node $V_{i+1[m]}$ inside each period. Thus, we need to evaluate the cost of such an exchange on the network topology we have defined. Since communications are handled along the edges of the tree according to depth first order, each edge of the tree is used exactly twice (one for forward and one for backward communications). Moreover, we make the assumption of one-port model for the nodes, so that one processor cannot be involved in more than one communication at the same time.

Thus, for node V_i, the overall communication time is given by $2 \sum_{V_j \in \mathcal{V}(V_i)} c_{i,j}$, where $\mathcal{V}(V_i)$ denotes the neighborhood of V_i in the tree. Thus, the overall communication time for the whole exchange procedure can be bounded by $2\ m\ c_{\max}$, where $c_{\max} = \max c_{i,j}$. This bound is very pessimistic in general (though it may be reached in the case of homogeneous communications and a star shaped network), but as already mentioned, is enough in order to prove the asymptotic optimality of the algorithm we propose in Section 3.5.

3.2 Lower bound for processing time

Let us consider a $n \times n$ Laplace task graph. The algorithm consist in updating all n^2 elements. To establish a lower bound for the complexity, we neglect for a while both communication costs and precedence constraints in order to concentrate on load balancing issues. Let us denote by n_i the number of elements updated in the optimal solution by node V_i, and let us denote by T_{OPT} the optimal time under those simplifying assumptions. The time required for V_i to update all n_i elements is given by $w_i n_i$, where w_i denotes the time to update one element.

Therefore, the following linear program provides a lower bound for overall processing time:

Minimize T_{OPT},
subject to

$$\begin{cases} \forall i, & n_i w_i \leq T_{\text{OPT}} \quad \text{(lower bound for processing time on } V_i) \\ & \sum n_i = n^2 \quad \text{(all the elements are processed).} \end{cases}$$

The solution of this linear program is not trivial in integer numbers, but solving it in rational number also provides a (lower) lower bound for processing time and, in this context, the solution is given by

$$\forall i, \quad n_i = \frac{n^2}{w_i \sum \frac{1}{w_j}}$$

so that

$$T_{\text{OPT}}(n) = \frac{n^2}{\sum \frac{1}{w_j}}$$

is a lower bound for the processing time of the $n \times n$ Laplace task graph.

3.3 Data distribution

Let us consider again a $n \times n$ Laplace task graph. We allocate rows to processors in a periodic way. The Laplace task graph is decomposed into roughly $\sqrt{n}$ periods of roughly $\sqrt{n}$ rows, and in each period, each processor receives approximately the amount of rows that corresponds to perfect load balancing.

More precisely, let us denote by

$$a_i = \lfloor \frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}} \rfloor$$

the number of rows allocated to V_i inside one period, so that the size B of the period is given by $B = \sum a_i$. Inside one period, the rows are allocated to the processors accordingly to their ordering in the path defined in Section 3.1, i.e. a_0 rows to processor V_0, and then a_1 to V_1 an so on, so that a period is as depicted in Figure 2.

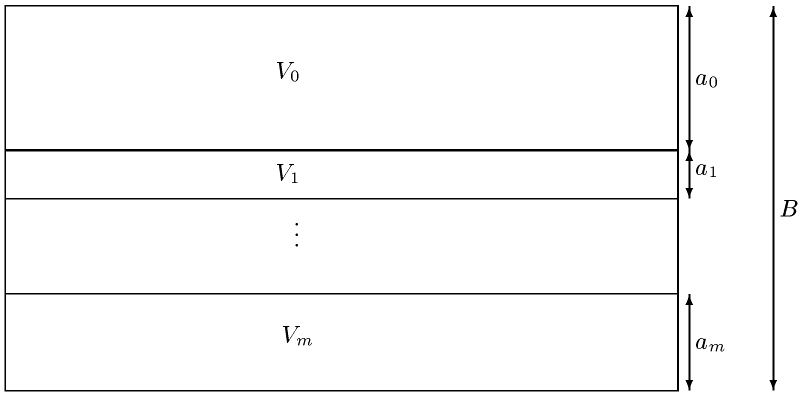

Figure 2: Data distribution inside a period of size B

Let us denote by $n = n^{(\text{per})}B + n^{(\text{rem})}$ the euclidean division of n by B, where $1 \leq n^{(\text{rem})} < B$. The data distribution of the whole matrix is depicted in Figure 3, and consists of $n^{(\text{per})} + 1$ periods of size B. The last $B - n^{(\text{rem})}$ rows and columns are filled with zeros. Clearly, the processing time of the filled matrix is larger than the processing time of the original matrix, so that asymptotical optimal results obtained on the filled matrix can be extended to the original matrix.

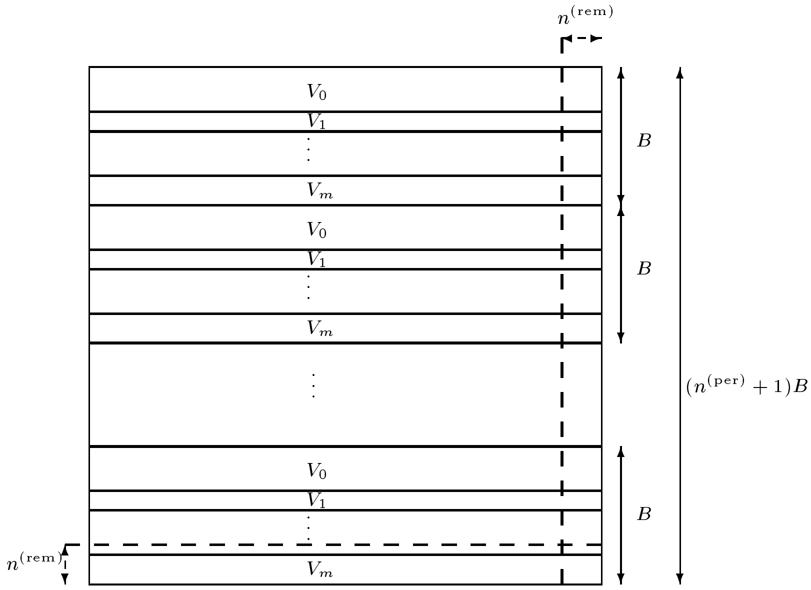

Figure 3: Data distribution of the filled matrix onto the processors

3.4 Sketch of the algorithm

The algorithm we propose consists in updating the elements along a moving diagonal. At step k, the elements (i, j) where $i + j < k$ have already been updated. Step k consists in updating the elements on the diagonal $i + j = k$, as depicted in Figure 4.

In Figure 4, the boxed elements of the k^{th} diagonal are those for which a communication is required from above processor before the update. For all other diagonal elements, both left and upper neighbors are located on the same processor, so that the update can take place without any communication.

This very simple algorithm is in fact asymptotically optimal, as proved in Section 3.5. The proof is based on the following two remarks. First, if n is large, the bands allocated to each processor are large (of order $\sqrt{n}$), and since only one communication is required for the computation of the whole band, then communications can be overlapped with computations. Moreover, since load balancing into each period of size B is almost perfect and (provided k is sufficiently large) the number of full periods is large with respect to the last (partial) period, then the processors receive an amount of work which is roughly proportional to their processing capabilities.

3.5 Proof of the asymptotic optimality

3.5.1 Overlapping condition

As already mentioned, we expect that, for sufficiently large values of n and k, communications can be overlapped with computations. Let us consider precisely the k^{th} step of the algorithm, and let $k = pB + q$ denote the euclidean division of k by B. Thus, p denotes the number of full periods, and $q < B$ denotes the number of remaining rows. Let us evaluate the overall volume of communication involved in step k. Inside each period, each processor sends one element to its below neighbor processor. In Section 3.1, we have seen that the cost of this pipelined exchange takes less than $2\ m\ c_{\max}$ time units, so that the overall volume of communication involved at step k is bounded by

$$2\ (p + 1)\ m\ c_{\max}.$$

Let us now evaluate the amount of computation that does not require any communication. Inside each period, processor V_i can process $a_i - 1$ updates without communication (see Figure 4). Thus, at step k, processor V_i can process at least $p(a_i - 1)$ elements in $p(a_i - 1)w_i$ time without the need of any communication.

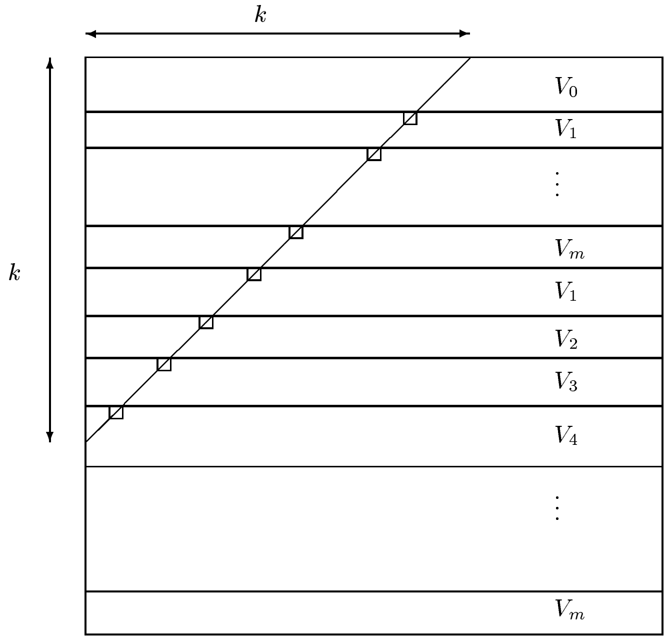

Figure 4: Step k of the algorithm

In order to overlap communications with computations, the processing time that does not require communications must be smaller than the overall communication time, on any processor. This condition can be written as

$$\forall i, \quad p(a_i - 1)w_i \geq 2(p+1)m\, c_{\max}\ .$$

Since $\frac{k}{B} - 1 < p \leq \frac{k}{B}$, and $a_i \geq \frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}}$, above condition holds true as soon as

$$(\frac{k}{B} - 1)(\frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}})w_i \geq 2(\frac{k}{B} + 1)\, c_{\max} \Longleftrightarrow \frac{k}{B} \geq \frac{\sqrt{n} + 2\, c_{\max} \sum \frac{1}{w_j}}{\sqrt{n} - 2\, c_{\max} \sum \frac{1}{w_j}}\ .$$

Therefore, as soon as $n \geq (2\, c_{\max} \sum \frac{1}{w_j})^2$, the above condition becomes $k \geq B$. Therefore, if n is sufficiently large, communication can be overlapped with computation at any step, except for the B first steps. Thus, we need to consider separately the first B steps, where communications cannot a priori be overlapped, and the next pB steps.

3.5.2 Case where $k \leq B$

Let us first consider the case where $k \leq B$. Since B is relatively small (of order $\sqrt{n}$), we can rely of a very rough bound. We bound the necessary time to process step k by the time necessary to process all the updates on the slowest processor, if we had to pay one communication (on the slowest link) between each pair of rows. In this case, the overall processing time for step k would be $(w_{\max} + c_{\max})k$, where $w_{\max} = \max w_i$. Therefore, if we denote by $T(k)$ the overall processing time of step k, then $T(k) \leq (w_{\max} + c_{\max})k$.

3.5.3 Case where $k > B$

Let us now consider the general case where $k > B$. In this case, the time to process all the rows distributed on V_i is bounded by $(p+1)(a_i - 1)w_i + (p+1)w_i$, i.e. the time to process all the updates that do not require any communication plus the time to process the updates of remaining tasks, once the communications have been done (what is possible because of the overlapping condition). Thus, if we denote by $T(V_i, k)$ the time to process all the tasks distributed on V_i, then

$$\begin{aligned} T(V_i, k) &\leq (p+1)a_i w_i \\ &\leq (p+1)\left(\frac{\sqrt{n}}{w_i \sum \frac{1}{w_j}} + 1\right) w_i \\ &\leq \left(\frac{k}{B} + 2\right)\left(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}\right) . \end{aligned}$$

The last bound holds true for any processor V_i, so that $T(k) \leq (\frac{k}{B} + 2)(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max})$.

3.5.4 Asymptotic optimality

We have evaluated the processing time $T(k)$ of any step of the algorithm. By a trivial symmetry argument, the overall processing time $T_{\text{ALG}}(n)$ for a $n \times n$ Laplace task graph $T(k)$ is bounded by

$$2 \sum_{1}^{(n^{(\text{per})}+1)B} T(k) .$$

Therefore

$$\begin{aligned} T_{\text{ALG}}(n) &\leq 2\left(\sum_1^B (w_{\max} + c_{\max})k + \sum_{B+1}^{(n^{(\text{per})}+1)B} \left(\frac{k}{B} + 2\right)\left(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}\right)\right) \\ &\leq (w_{\max} + c_{\max})B(B+1) + (n^{(\text{per})} + 1)((n^{(\text{per})} + 1)B + 1)\left(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}\right) + \\ &\qquad 2(n^{(\text{per})} + 1)\left(\frac{\sqrt{n}}{\sum \frac{1}{w_j}} + w_{\max}\right) . \end{aligned}$$

Moreover, since $\sqrt{n} - m \leq B \leq \sqrt{n}$, then, as soon as $\sqrt{n} \geq m$, then

$$T_{\text{ALG}}(n) \leq \frac{n^2}{\sum \frac{1}{w_j}} + 2n\sqrt{n}\left(\frac{1}{\sum \frac{1}{w_j}} + 2m + w_{\max}\right) + O(n) .$$

Therefore

$$\frac{T_{\text{ALG}}(n)}{T_{\text{OPT}}(n)} \leq 1 + \frac{1}{\sqrt{n}}\left(2 + (2m + w_{\max}) \sum \frac{1}{w_j}\right) + O\left(\frac{1}{n}\right) .$$

Thus, when n is large, $\frac{T_{\text{ALG}}(n)}{T_{\text{OPT}}(n)}$ becomes arbitrarily close to 1, what achieves the proof of the asymptotical optimality of the algorithm we propose.

4 Conclusion

In this paper, we have shown that it is possible to derive an asymptotically optimal algorithm for Laplace task graph, when the size of the task graph becomes arbitrarily large. The algorithm is based on an heterogeneous block data distribution, that ensures a good load balancing all along the processing. Since the size of the task graph is large, it is possible to make use of large grain distribution and thus, communications can be fully overlapped with computations.

This study may be a good testbed in a more general context. Indeed, Laplace task graph is an example of DO − DO-PAR loops, if the matrix is updated along its diagonals. This context is much more general, and the transformation of a nested loop into a DO − DO-PAR loop may be obtained, for instance, with the algorithm of Allen and Kennedy [1].

The case when the outermost loop is parallel has already been addressed in [2], since it corresponds to the distribution of independent tasks, when data is supposed to be initially located on a single processor and master slave tasking is used. Our study may lead to comparable results for inner DO-PAR loops.

Finally, the algorithm we propose shows that asymptotical optimality is much easier to obtain than classical makespan minimization, especially in the context of heterogeneous computing (see also [3] and [4]). Moreover, considering very large task graphs makes sense in the context of heterogeneous (grid) computing, since the startup time on such a platform may be large, thus restricting its use to very large regular applications.

References

[1] J. R. Allen and K. Kennedy. Automatic loop interchange. In S. Notices, editor, *Proceedings of the SIGPLAN Symposium on Compiler Construction*, volume 19-6, pages 233–246, Montreal, Canada, June 1984.

[2] C. Banino, O. Beaumont, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS 2367, pages 423–432. Springer Verlag, 2002.

[3] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Technical Report 2002-36, LIP, ENS Lyon, France, Oct. 2002.

[4] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[5] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Computer Society Press, 1996.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1991.

[7] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS, 1997.

[8] D. K. Lowenthal. Accurately selecting block size at run time in pipelined parallel programs. *International Journal of Parallel Programming*, 28(3):245–274, 2000.

[9] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems.* IEEE Computer Science Press, 1995.

[10] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Anderson, and D. E. Culler. Modeling and optimizing communication pipelines. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.* ACM, 1998.