Versatile Task Scheduling of Binary Trees for Realistic Machines

Cristina Boeres* and Vinod E. F. Rebello

Departamento de Computação, Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
e-mail: boeres@dcc.uff.br

Abstract. In general, scheduling models only consider message latency as the sole dominant communication parameter. However, in many parallel systems, latency is negligible when compared to the CPU penalties associated with sending and receiving *communication events*. Our work considers a model in which this *CPU penalty* can also be a significant communication parameter. This paper focusses on analysing the effect of such a model on the scheduling of Full Binary Trees. We briefly describe a versatile, multi-stage scheduling approach that can be customised to classes of parallel systems according to their communication performance characteristics and which produces better makespans when compared with traditional techniques.

1 Introduction

It is well known that the scheduling problem is NP-complete in its general form and that a number of architectural and application-related characteristics have influenced the design of many scheduling heuristics [7, 9, 11]. With the development of distributed memory machines, more realistic models of parallel computation have been proposed where communication characteristics are represented. More recently however, the interface between a processor and the communication network has been identified as a potential *bottleneck* [5, 8]. Just as machines continue to evolve and their characteristics change, so must the models used to represent them. The scheduling model considered in this work defines not only the network delay or latency as a significant parameter, which has been given much attention previously, but also the overhead incurred when sending and receiving messages. This is not part of the delay but rather a cost in the form of a *CPU penalty* incurred on processors when communicating. Many scheduling models (e.g. [7]) fall into the class of the linear communication model, adding the overhead to the delay cost either for simplicity or when considering dedicated processors are used to perform communication. However, since the overhead is a blocking factor, the linear model may not reflect reality. The CLAUD model [4] is adopted as a more realistic model because it defines important characteristics

* This work was carried out while the author was with the Department of Computer Science, University of Edinburgh, Scotland, UK, and was being supported by a grant from CAPES, Ministry of Education, Brazil, under grant number 1553/91-8.

of applications and target architectures such as: the existence of *communication events* as tasks; the overheads associated with the *sending* and *receiving* of a message (denoted by λ_s and λ_r, respectively); and the delay τ which is incurred when a processor transmits a data item to an adjacent processor.

In this work, an input application is represented by a directed acyclic graph (*DAG*) $G = (V, E)$. The replication of tasks onto distinct processors is allowed, since task replication can be used to minimise communication costs in architectures with high communication costs [10]. The time that the processor is *blocked* characterises a *communication event* and can also be regarded as a "task". The existence of these special tasks depends on the scheduling of the computation tasks of the application. In parallel architectures with high overheads, the number of communication events can be reduced by bundling many small messages into fewer, larger ones through the clustering of tasks [1]. In this context, a cluster c of tasks is defined as a set of tasks of G, listed in accordance to their precedence relationship and with no communication occuring while the tasks in c are being executed. Messages to be sent by tasks in c to tasks in an immediate successor cluster c' are bundled and sent together. The dependencies between clusters represent both the dependency inherited from the original *DAG* G and the communication between the distinct processors to which the clusters have been allocated. This dependency relation and the set of all clusters defined are represented by a *DAG* called a *superdag*. The vertices in the superdag, the *supertasks*, correspond to clusters of tasks and the *superedges* are the communications between supertasks.

This paper gives a brief description of a *versatile* task scheduling approach followed by a discussion on the merits of two alternative partitioning strategies for Full Binary Trees together with experimental results.

2 A General Overview of the New Approach

The *multi-stage approach* (MSA) [1] performs a series of transformations on the input *DAG* until its schedule on the target machine is finalised. MSA aims to produce good schedules for general applications on a variety of target architectures, particularly those which have high communication overheads. The following sections briefly describe this new approach, but a more detailed description of each of the two stages of MSA can be found in [1].

I) Constructing a superdag – The first stage of MSA is comprised of two heuristics and produces a superdag with characteristics that are dependent on the target architecture and the input *DAG*. The first heuristic, the *replication* algorithm, adopts the *replication* principle (also used by Papadimitriou and Yannakakis [11]) to cluster tasks so that the number of communication events can be reduced. A cluster c is created by allocating v_i to a *virtual* processor p_i and replicating γ selected ancestors of v_i on to p_i, where γ is the *clustering factor*. The γ ancestors are chosen, to minimise the number of communication events, based on their value $e_l(v_i)$ which is the *latest* time at which task v_i can start

without increasing the makespan of the resulting schedule. One advantage of using e_l-values is an increase in the number of candidate tasks that can be clustered with v_i (there tends to be larger groups of tasks with the same e_l-value than with the same earliest start time as defined in [11]). The result of the *replication* algorithm is a schedule containing sets of clearly defined parallel clusters, each set being executed during a *computational interval* or *band*. Between any two bands there exists a *communication interval* where only communication event tasks are scheduled.

Not all of the clusters produced by the *replication* algorithm may be needed in order to schedule the *DAG* G on the target machine. In order to select reasonably good candidates, a set of *rules* has been implemented in the second heuristic (the *superdag* algorithm [1]). This algorithm aims to construct a superdag with a granularity that matches the granularity of the target architecture. For each supertask, the immediate predecessors chosen are those which send the largest messages and have the smallest out degrees. These are necessary characteristics when the overhead is high. On the other hand, when the delay is the dominant communication parameter, the rules are parameterised in such a way that the immediate predecessors chosen are those which send the smallest messages.

II) Remapping onto physical processors – The superdag produced by the first stage will never contain more supertasks than the number of tasks in the original *DAG*. The number of supertasks may, of course, be greater than the number of available physical processors m, since the first stage of MSA does not take this parameter into account but rather allocates each supertask to a distinct virtual processor. Thus, the second stage of MSA transforms the given superdag so it can be executed efficiently on a fully connected network of m processors. This is achieved by exploiting the degree of parallelism within the superdag, i.e. the *parallel* supertasks scheduled in the same *computational* interval.

Given the schedule S and the superdag $G_S = (V_S, E_S)$, the supertasks are clustered by merging parallel supertasks and remapping them on to physical processors, if the number of supertasks in that band is greater than m. The merging is achieved in such a way that the number of communication events is minimised, even though an increase in the length of messages may be necessary.

3 Scheduling Full Binary Trees

The Full Binary Tree $B_n = (V_B, E_B)$ is a *DAG* with n tasks, in which the direction of the edges in E_B are towards the unique sink in V_B. In order to analyse the impact of the communication parameters on the scheduling of B_n, the *layer* and *subtree* partitions are each considered under the following models: the *delay* model (as described in [11]), where the communication cost is solely characterised by the communication delay τ; and the CLAUD model, where the overheads λ_s and λ_r are also defined as communication parameters together with τ (for brevity, whenever the overheads for sending and receiving are referred to as λ, it means that $\lambda_s = \lambda_r = \lambda$).

3.1 The *layer* partition

Based on the asymptotically optimal schedule of Papadimitriou and Yannakakis [11], Jung *et al.* [10] devised a scheduling strategy for B_n under the *delay* model by dividing it into layers. In order to incorporate the overhead λ, which is a blocking factor, the *layer* partition for B_n is formed by abstracting away knowledge of the communication parameters and making use of the *clustering factor* γ. Depending on the target model γ may assume different values, i.e. it may be assigned to λ, τ or even some other characteristic of the input *DAG*.

The tree B_n is partitioned into $h = \log_{\gamma+2}(n+1)$ layers, producing subtrees (supertasks) each comprised of $(\gamma+1)$ tasks. The in-degree of each supertask (apart from the sources) is equal to $\gamma+2$ and therefore, the resulting superdag is a $(\gamma+2)$-*ary tree*. Note also that each superedge is associated with unit weight since only the respective sink task in each subtree sends data to an immediate successor supertask.

The layer ***partition and the*** delay ***model*** – Since all communications between each layer take place in parallel, the cost (which is τ) is constant irrespective of the number of immediate predecessors allocated to distinct processors. Nevertheless, if more than one supertask from each layer is allocated to the same processor, then the computation time is increased. Therefore, the minimum makespan $h(\tau+1)+(h-1)\tau$ is achieved when there are at least $\frac{n+1}{\gamma+2}$ processors.

The layer ***partition and the*** CLAUD ***model*** – The supertasks at each layer are allocated to m processors with each supertask being allocated to the same processor as one of its immediate predecessors. Note that the upper bound on the number of processors is the number of supertasks in layer $h-1$, i.e. $(\gamma+2)^{h-1}$. The makespan of this schedule, proved in [1], is expressed by
$(\gamma+1)\cdot\sum_{i=k}^{h-1}\frac{(\gamma+2)^i}{m} + k(\gamma+1) + (k-1)[(\gamma+2)\lambda+\tau] + \frac{m}{(\gamma+2)^{k-1}}\lambda + \frac{(\gamma+2)^k}{m}\tau$
where $k = \lceil \log_{\gamma+2} m \rceil$ is the layer in which the number of supertasks equals the number of processors.

3.2 The *subtree* partition

There exist scheduling algorithms [7, 9, 12] which tend to allocate the critical path of a *DAG* to a single processor. When such algorithms are applied to B_n, the allocation may be "inefficient" if the target machine has high overheads and particularly when they are much greater than the execution costs of the tasks. The *layer* partition provides subtrees as clusters of tasks, producing a coarser *DAG*. However, depending on the value of λ, the number of communication events which are incurred can be high since a $(\gamma+2)$-ary tree is produced.

A partition for B_n is proposed in which the number of communication events is not dependent on the overhead parameter. The partition divides B_n *vertically* with each subtree being allocated to one of the m processors. For the sake of clarity, let $m = 2^k$, for some k. The $\frac{n+1}{2}$ sources of B_n are divided into m subsets so that $\frac{n+1}{2m}$ tasks are sources of m subtrees and with each subtree being

allocated to a distinct processor. Each one of the m subtrees has its respective sink at level $k = \log m$. The expression for the makespan of a schedule produced by this partitioning strategy holds regardless of whether the communication cost is comprised of latency and/or overheads [1]. The makespan of the *subtree* partition, also proved in [1], is $\frac{n+1}{m} - 1 + (\log m - 1)(2\lambda + \tau + 1)$.

4 Experimental evaluations

MSA can produce a schedule for the Full Binary Tree which is equivalent to the *subtree* partition if the *clustering factor* γ has the value zero. For values other than zero, MSA produces the *layer* partition [1]. In this section, an analysis of the partitions discussed in this paper is carried out taking into account different target models.

Overheads are the only communication costs – When executing the *subtree* partition on m processors, the number of communication events that occur when two processors communicate is equal to two, regardless of the number of tasks, the value of λ and the number of processors m. On the other hand, in the *layer* partition with $\gamma = \lambda$, the binary tree is "transformed" into an $(\lambda + 2)$-ary tree corresponding to the fact that $(\lambda + 1)$ receiving events are carried out. Therefore, the higher the value of λ, the smaller the number of processors that can usefully be utilised to achieve the minimum makespan.

Figure 1 compares the makespans of the two partitioning strategies for B_{2047}. When $\lambda = 2$, the layer partition produces results very close to the *subtree* partition because of the low in-degree of each supertask. On the other hand, for larger overheads (e.g. when $\lambda = 30$), the schedules provided by the *layer* partition become worse when compared with those of the *subtree* partition.

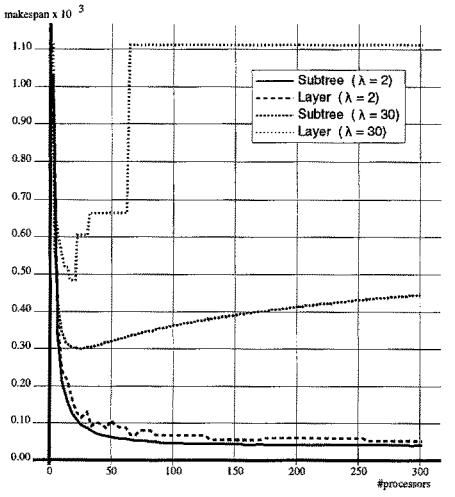

Fig. 1. The makespans of the *subtree* ($\gamma = 0$) and *layer* ($\gamma = \lambda$) partitions for B_{2047} with $\tau = 0$, when $\lambda = 2$ and $\lambda = 30$.

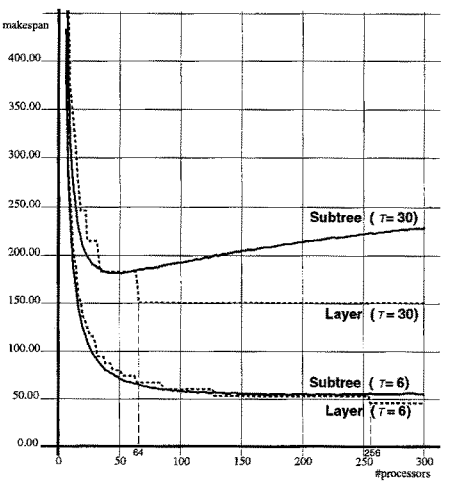

Fig. 2. The makespans of the *subtree* ($\gamma = 0$) and *layer* ($\gamma = \tau$) partitions for B_{2047} with $\lambda \simeq 0$, when $\tau = 6$ and $\tau = 30$.

The delay ***model*** – In order to compare the *subtree* and the *layer* partitions under the *delay* model, suppose that the number of available physical processors is at least $\frac{n+1}{\tau+2}$. Thus, the higher τ is, the smaller n (the number of tasks of B_n) must be so that the *subtree* partition produces a better schedule. The results confirm that the *layer* partition does provide schedules with smaller makespans than the *subtree* partition, particularly when $m \geq \frac{n+1}{\tau+2}$ (Figure 2 is an example for B_{2047}).

The CLAUD ***model*** – Both communication parameters are now considered significant and two cases are given attention in the experiments. Firstly, when $\lambda > \tau$ (the overhead is dominant), the clustering of tasks as in the *layer* partition is not wise since the in degrees can be large. Regardless of the number of processors, the best approach is the *subtree* partition, as seen in Figure 3. In the second case, when $\lambda \leq \tau/2$ (the factor of two allows a full subtree to be clustered into a supertask), the delay is dominant and although for $\gamma = \tau$ the respective superdag may have a high in degree, the values for λ are not high enough to hinder the benefits of partitioning B_n into layers. The best results were achieved when $m \geq \frac{n+1}{\tau+2}$ and $\gamma = \tau$ for the *layer* partition (Figure 4).

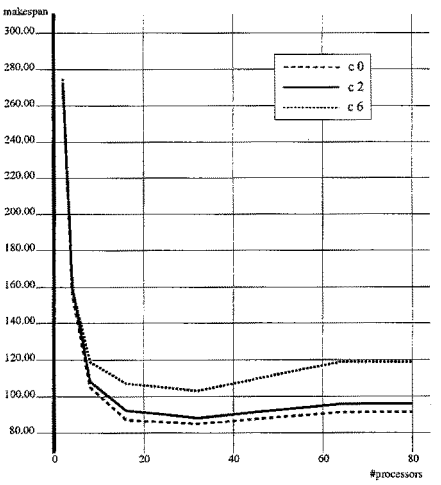

Fig. 3. Makespans of the *subtree* ($\gamma = 0$ corresponds to $c0$) and *layer* ($\gamma = 2, 6$ correspond to $c2$ and $c6$, respectively) partitions for B_{511}, $\tau = 1$ and $\lambda = 6$.

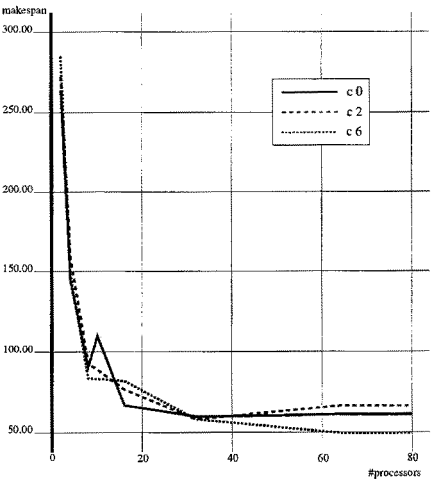

Fig. 4. Makespans of the *subtree* ($\gamma = 0$ corresponds to $c0$) and *layer* ($\gamma = 2, 6$ corresponds to $c2$ and $c6$, respectively) partitions for B_{511}, $\tau = 6$ and $\lambda = 1$.

Comparing MSA with other heuristics – In evaluating MSA and the *layer* and *subtree* partitions, two algorithms classified as list scheduling heuristics were implemented: Earliest Task First (ETF) of Hwang *et al.* [9] and the Mapping Heuristic (MH) of Rewini *et al.* [6]. The sending and receiving overheads were incorporated in the cost function that guides ETF. MH incorporates the cost of sending a message into its cost function, therefore representing the linear

communication model. The schedules produced by MSA, ETF and MH were executed on a *simulated machine*, i.e. a tool capable of simulating the execution of given schedules on parallel machines with different characteristics.

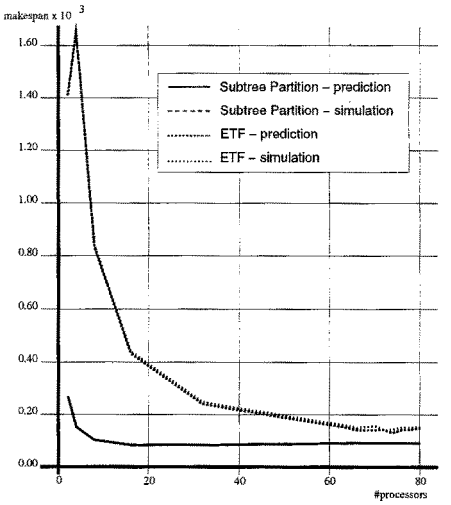

Fig. 5. Makespans of MSA (the *subtree* partition using $\gamma = 0$) and ETF for B_{511}, $\tau = 1$ and $\lambda = 6$.

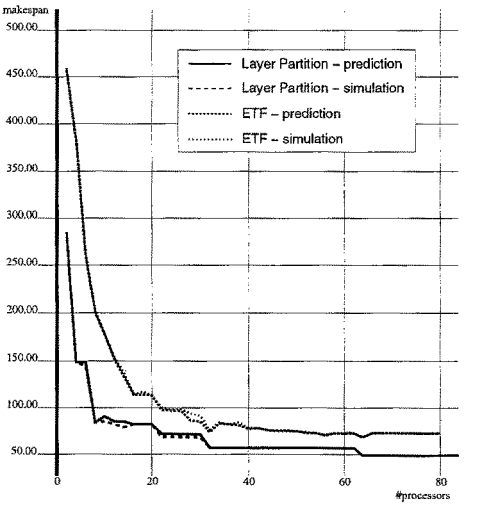

Fig. 6. Makespans of MSA (the *layer* partition using $\gamma = \tau$) and ETF for B_{511}, $\tau = 6$ and $\lambda = 1$.

The best results produced by the *subtree* and *layer* partitions for B_{511}, are compared with ETF (see Figures 5 and 6). When $\lambda > \tau$, it is better to set the parameter γ to zero in the first stage of MSA (for the *subtree* partition). One of the problems with ETF is that due to local decisions, when the number of available processors is smaller than the number of source tasks, the source tasks which do not belong to the same subtree may be allocated to the same processor, i.e. ETF does not attempt to minimise the number of communication events. The second case examines the situation in which $\lambda < \tau$ and where MSA is executed with $\gamma = \tau$. In Figure 6, ETF's results are close to the *layer* partition of MSA, but only because the overheads are much smaller than the delay τ. The schedules generated by MSA and ETF for B_{511} were executed on the simulated machine considering a fully connected network. Both predictions are very accurate, but MSA gives better results than ETF for any number of processors.

To compare MSA with MH, experiments are carried out with the significant communication parameters being λ_s and τ. For B_n, the *sending event* tasks will be executed in parallel irrespective of the partitioning applied. Therefore, the best partitioning technique is the one that minimises the number of *sending event* tasks along the critical path of the corresponding superdag. In this case, the in degree of the $(\gamma+2)$-ary tree produced by the *layer* partition does not hinder the performance of the resulting schedule. First, experiments are performed with the delay τ as the only significant parameter (Figure 7). The best value of γ is that which produces supertasks containing Full Binary Trees (verified experimentally [1]). Figure 8 shows the results for the CLAUD model where λ_s and τ are

significant. The times to completion of the simulation of MSA schedules are as predicted and therefore have been omitted from the respective figures. In all of the cases considering the *delay* model and CLAUD model, MSA produces better results then MH. Also, in the case of MH, Figures 7 and 8 together show that as the value of λ_s increases, so does the difference between the simulated execution time and the predicted one.

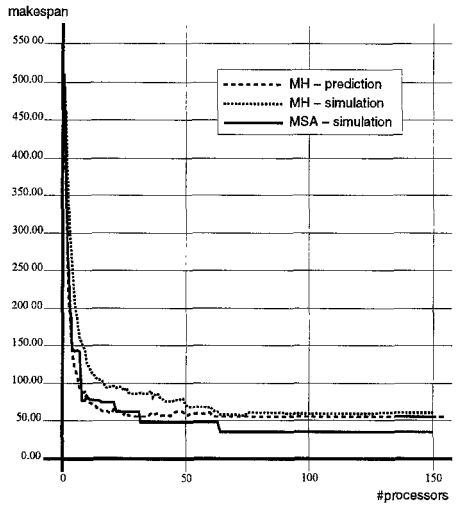

Fig. 7. Makespans of MSA with $\gamma = 6$ and MH, for B_{511} with $\tau = 6$ and $\lambda_s = \lambda_r = 0$.

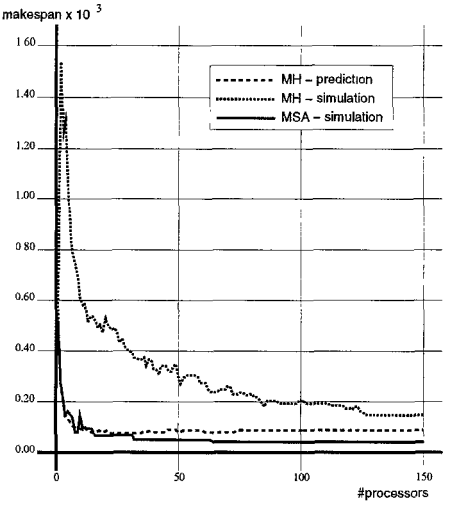

Fig. 8. Makespans of MSA with $\gamma = 6$ and MH, for B_{511} with $\tau = 1$, $\lambda_s = 10$ and $\lambda_r = 0$.

5 Concluding Remarks

Jung *et al.* [10] showed that the Full Binary Tree can be asymptotically optimally scheduled without replication, given that the target model has a single communication parameter – the delay τ. The *layer* partition provides a mechanism of transforming the input *DAG* onto a coarser one. However, when the *blocking* parameters λ_s and λ_r are considered, the partitioning of the Full Binary Tree into layers produces supertasks with in degree ≥ 2. In architectures with high overheads, the parallel execution of such superdags leads to a high communication cost due to the sending and receiving events. The *subtree* partitioning is proposed to overcome this problem and is provided by MSA when γ, the *clustering factor*, is zero.

Note that ETF and MH do not provide schedules that are completely unreasonable. In fact, for the *delay* model, both ETF and MH can provide better results than MSA when there is a small number of available physical processors and τ is very low. However, this is not true (under the *delay* model) for larger τ and, particularly, for the CLAUD model. In the latter case, by adding the startup overhead to the delay, ETF and MH specify that the startup cost can also be overlapped with task computation. This may be unrealistic and consequently,

the execution times of the schedules may be much higher than their respective predictions.

This paper has evaluated the impact of the delay and the overhead communication parameters on the schedules of a certain class of regular *DAGs*, known as Full Binary Trees. Two types of partitions were analysed and the results showed that the partition which produced the best schedule depends on the characteristics of the target model. Our *multi-stage approach* displayed its versatility by tuning the input parameters that guide the clustering of tasks and bundling of messages so that a suitable schedule for the given underlying model is produced. An analysis of other regular structures, such as the diamond DAG and irregular graphs, and their partitioning strategies is described in [1].

References

1. C. Boeres. *Versatile Communication Cost Modelling for Multicomputer Task Scheduling Heuristics.* PhD thesis, Department of Computer Science, University of Edinburgh, Oct 1996.
2. C. Boeres, G. Chochia, and P. Thanisch. On the scope of applicability of the ETF algorithm. In *Proceedings of the 2nd International Workshop on Parallel Algorithms for Irregularly Structured Problems (IRREGULAR'95)*, Lecture Notes in Computer Science (LNCS 980), pages 159–164, Lyon, France, Sept 1995. Springer.
3. R.P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21:201–206, 1974.
4. G. Chochia, C. Boeres, M. Norman, and P. Thanisch. Analysis of multicomputer schedules in a cost and latency model of communication. In *Proceedings of the 3rd Workshop on Abstract Machine Models for Parallel and Distributed Computing*, Leeds, UK., April 1996. IOS press.
5. D. Culler *et al.* LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993.
6. H. El-Rewini and T.G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *J. Parallel Dist. Comput.*, 9:138–153, 1990.
7. A. Gerasoulis, S. Venugopol, and T. Yang. Clustering task graphs for message passing architectures. In *The International Conference on Supercomputing*, pages 447–456, Amsterdam, The Netherlands, June 1990.
8. F.W. Howell. Reverse profiling. In Innes Jelly, Ian Gorton, and Peter Croll, editors, *Software Engineering for Parallel and Distributed Systems*, pages 244–255. Chapman and Hall on behalf of IFIP, March 1996.
9. J-J. Hwang, Y-C. Chow, F.D. Anger, and C-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257, 1989.
10. H. Jung, L. Kirousis, and P. Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of DAGs with communication delays. In *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pages 254–264, 1989.
11. C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328, 1990.
12. V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors.* Pitman, London, 1989.