Towards an Effective Task Clustering Heuristic for LogP Machines

Cristina Boeres, Aline Nascimento* and Vinod E. F. Rebello

Instituto de Computação, Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
e-mail: {boeres,aline,vefr}@pgcc.uff.br

Abstract. This paper describes a task scheduling algorithm, based on a *LogP*-type model, for allocating arbitrary task graphs to fully connected networks of processors. This problem is known to be NP-complete even under the delay model (a special case under the *LogP* model). The strategy exploits the replication and clustering of tasks to minimise the ill effects of communication overhead on the makespan. The quality of the schedules produced by this *LogP*-based algorithm, initially under delay model conditions, is compared with that of other good delay model-based approaches.

1 Introduction

The effective scheduling of a parallel program onto the processors of a parallel machine is crucial for achieving high performance. One common program representation is a *directed acyclic graph* (*DAG*) whose nodes denote *tasks* and whose edges denote data dependencies (and hence communication, if the tasks are mapped to distinct processors). These program communication costs can be reduced by clustering or grouping tasks together to share processors. Until recently, the standard communication model in the scheduling community has been the *delay model*, where the sole architectural parameter for the communication system is the *latency* or *delay*, i.e. the transit time for each message [15].

Clearly, the majority of scheduling algorithms perform some sort of *clustering* (i.e. grouping two or more tasks onto the same processor). However in this paper, the term *cluster algorithms* refer specifically to the class of algorithms which initially consider each task as a cluster (allocated to a unique processor) and then merge clusters (tasks) if the makespan can be reduced.

DSC [17] is a clustering algorithm that finds optimal 1-linear schedules (without replicating tasks) for *DAG*s with granularity γ which are within a factor of $1 + \frac{1}{\gamma}$ of the true optimal schedule. Under this delay model heuristic, an optimal schedule can be found in polynomial time for fork, join, and coarse-grained (inverted) tree *DAG*s. Note that Jung *et al.* concluded that the best schedule produced by an algorithm that ignores replication may be worse than

* The author is supported by a postgraduate scholarship from CAPES, Ministério de Educação e Cultura (MEC), Brazil.

the true optimal schedule by a multiplicative factor which is a function of the communication delay [10].

If recomputation is allowed, there are cases where coarse-grained *DAG*s can be scheduled (truly) optimally in polynomial time [13, 14]. Palis *et al.* [14] proposed a clustering algorithm PLW which produces a schedule with a makespan at most $(1+\varepsilon)$ times the makespan of the true optimal, where the granularity of the DAG is at least $\frac{1-\varepsilon}{\varepsilon}$. Therefore, for any granularity of graph, PLW produces a schedule whose makespan is at most twice the optimal.

Unfortunately, research has shown that the delay model is not that realistic since it assumes certain properties (e.g. the ability to *multicast*, i.e. to send a number of different messages to different destinations simultaneously; and that communication can completely overlap with computation of tasks) that may not hold in practice – the CPU generally has to manage or at least initiate each communication [6]. The dominant cost of communication in today's architectures is that of crossing the network boundary. This is a cost which cannot be modelled as a latency and thus implies that new classes of scheduling heuristics are required to generate efficient schedules for realistic abstractions of today's parallel computers [4].

These architectural characteristics have motivated new (and now widely accepted) parallel programming models such as $LogP$ [6] and BSP [16]. The *LogP model*, for example, is an MIMD message-passing model with four architectural parameters: the latency, L; the overhead, o, the CPU penalty incurred by a communication action; the gap, g, a lower bound on the interval between two successive communications; and the number of processors, P. Although the *BSP model* takes a less prescriptive view of communication actions, it has been shown that the two models are similar particularly terms of communication [4, 16].

In comparison with the delay model, identifying scheduling strategies that specify such $LogP$-type characteristics in their communication models is difficult. Only recently have a few scheduling algorithms appeared in the literature. Zimmerman *et al.* extended the work of [11, 17] to propose a clustering algorithm which produces optimal k-linear schedules for tree-like graphs when $o \geq g$ [18]. Boeres *et al.* [2, 3, 4] proposed a task replication strategy for scheduling arbitrary UET-UDC (unit execution task/unit data communication) *DAG*s on machines with a bounded number of processors under both the $LogP$ and BSP models. A novel feature being the use of task replication and tight clustering to support the bundling of messages as a technique to reduce communication costs.

The purpose of this work is to continue to study the problem of scheduling arbitrary task graphs under $LogP$-type models, by building on the results of existing delay model-based task replication clustering algorithms. In the following section, we present a replication-based task clustering algorithm for scheduling general or arbitrary *DAG*s on $LogP$-type machines (although an unbounded number of processors is assumed). The specific architectural parameters considered have been adopted from the *Cost and Latency Augmented DAG* (CLAUD) model [5] (developed independently of $LogP$). In the CLAUD model, an application is represented by a DAG G and the set of m identical processors by $\mathcal{P}$.

The overheads associated with the *sending* and *receiving* of a message are denoted by λ_s and λ_r, respectively, and the communication latency or delay by τ. The CLAUD model very naturally models $LogP$, given the following parameter relationships: $L = \tau$, $o = \frac{\lambda_s + \lambda_r}{2}$, $g = \lambda_s$ and $P = m$.

The delay model is a special case in under a $LogP$-type model where the overheads are zero. Therefore, any proposed $LogP$-type scheduling algorithm still has to produce good schedules under delay model conditions. This is the focus of this paper and Section 3 compares the makespans produced by the new algorithm under delay model conditions against two other well known delay model clustering algorithms – *DSC* and *PLW*. Section 4 draws some conclusions and outlines future work regarding scheduling for $LogP$-type machines. (Throughout this paper, the term *delay model conditions* is used to refer to situation where the overheads are zero and the term *LogP conditions* used when the overheads are nonzero.)

2 A New Heuristic

A parallel application is represented by a DAG $G = (V, E, \varepsilon, \omega)$, where V is the set of tasks, E is the precedence relation among them, $\varepsilon(v_i)$ is the execution cost of task $v_i \in V$ and $\omega(v_i, v_j)$ is the weight associated to the edge $(v_i, v_j) \in E$ representing the amount of data units transmitted from task v_i to v_j.

For the duration of each communication overhead the processor is effectively *blocked* unable to execute other tasks in V or even to send or receive other messages. Consequently, any scheduling algorithm must, in some sense, view these sending and receiving overheads also as "tasks" to be executed by processors.

The new clustering heuristic (BNR) attempts to minimise the makespan by trying to construct a cluster for each task v_i so that v_i starts executing at the earliest possible time. Cluster v_i will contain the *owner task* v_i and, determined by a cost function, copies of some of its ancestors (i.e. BNR employs task replication). Like most clustering algorithms, BNR consists of two phases: the first determines which tasks should be included and their order within a cluster; the second, identifies the clusters required to implement the input DAG G and maps each necessary cluster to a unique processor. Even though an unbounded number of processors is assumed, BNR does try to minimise the number of processors required as long as the makespan is not adversely affected.

2.1 Properties for *LogP* scheduling

In order to schedule tasks under a $LogP$-type model, BNR applies a couple of new *general scheduling restrictions* with regard to properties of clusters (one in each phase). In the first phase, only the owner task of a cluster may send data to a task in another cluster. This restriction does not adversely affect any makespan since, by definition, an owner task will not start execution later than any copy of itself. Also, if a non-owner task u in cluster $C(v_i)$ were to communicate with a task in another cluster, the processor to which $C(v_i)$ is allocated would incur a

send overhead after the execution of u which in turn might force task v_i to start at a time later than its earliest possible. The second phase uses the restriction that each cluster can have only one successor. If two or more clusters share the same predecessor cluster, then this phase will assign a unique copy of the predecessor cluster to each successor irrespective of whether or not the data being sent to each successor is identical. This removes the need to incur the multiple send overheads at the end of the predecessor cluster which unnecessarily delay the execution of successor clusters.

These general restrictions are new in the sense that they do not need to be applied by scheduling strategies which are used exclusively for delay model conditions (because of the assumptions made under the delay model). The purpose of these restrictions is to aid in the minimisation of the makespan, however they do incur the penalty of increasing the number of clusters required and thus the number of processors needed. Where the number of processors used is viewed as a reflection on the cost of implementing the schedule, a post-pass optimisation can be applied to relax the above restrictions and remove clusters which now become redundant (i.e. those clusters whose removal will not increase the makespan). Note that Löwe *et al.* [7] suggests that the problem of determining the optimal amount of replication is effectively NP-complete for the $LogP$ model.

2.2 The first phase

For each task $v_i \in V$ (where i is assigned topologically), BNR constructs a cluster $C(v_i)$ (with v_i being its owner) from the ancestors of v_i. The term $iancs(C(v_i))$ defines the set of immediate ancestors of tasks already in $C(v_i)$ which themselves are not in $C(v_i)$, i.e. $iancs(C(v_i)) = \{u_j \mid (u_j, t_l) \in E \wedge u_j \notin C(v_i) \wedge t_l \in C(v_i)\}$. A task $u_j \in iancs(C(v_i))$ is incorporated into cluster $C(v_i)$ if v_i's start time can be reduced. Thus, BNR includes u_j in $C(v_i)$ based on comparing the start time of v_i under the following two situations: (a) when $u_j \notin C(v_i)$; and (b) when $u_j \in C(v_i)$.

The pseudocode for BNR can be seen in Algorithm 3. The strategy visits each task $v_i \in G$, constructing cluster $C(v_i)$ and calculating v_i's *earliest schedule time* $e_s(v_i)$ (which is defined as the start time of v_i in its own cluster $C(v_i)$) as described in the manner below. Note that we assume the definition of the *earliest start time* of task v_i to mean the optimal or earliest possible start time of v_i [15]. Our objective is to attempt to find a schedule for each $v_i \in V$ such that $e_s(v_i) = e(v_i)$.

Let $C(v_i) = \langle t_1, \ldots, t_l, \ldots t_k, v_i \rangle$ be tasks of cluster v_i in execution order. These tasks are not listed in non-decreasing order of their earliest start time nor earliest schedule time but rather in an order determined by the sequence in which tasks were included in the cluster and the position of their immediate successor task at the time of their inclusion. The *critical cluster path cost*, $m(C(v_i))$, is the earliest time that v_i can start **due** to the ancestor tasks of v_i in $C(v_i)$. This is calculated simply by ignoring the communications of all tasks $\in iancs(C(v_i))$ to $C(v_i)$, as shown in lines 2 and 3 of Algorithm 1. $et(t_p, v_i)$ is the end time of this copy of task t_p in $C(v_i)$ and which is effectively never scheduled earlier than

the original (owner) task t_p in $C(t_p)$(line 3). The end time of t_1, the first task, $(et(t_1, v_i) = e_s(t_1) + \varepsilon(t_1))$ is based on its earliest schedule time even though its ancestors (when it has some) are not (yet) in the cluster. The reason for this is that BNR, in some sense, bases this value on the likely cost in future iterations, i.e. best, of course, being limited by the task's earliest schedule time.

Algorithm 1 : *critical-cluster-path* $(C(v_i))$*;*

```
1  let C(v_i) = ⟨t_1, t_2, ..., t_k⟩;   et(t_1, v_i) := e_s(t_1) + ε(t_1);
2  for p = 2, ..., k do
3     et(t_p, v_i) := max{et(t_{p-1}, v_i), e_s(t_p)} + ε(t_p);
4  m(C(v_i)) := et(t_k, v_i);
```

Algorithm 2 : *critical-ancs-path* $(v_i, C(v_i))$

```
1  maxc := 0;  define iancs(C(v_i));  let C(v_i) = ⟨t_1, t_2, ..., t_k⟩;
2  if (iancs(C(v_i)) ≠ ∅) then
3     for all u_j ∈ iancs(C(v_i)) do
4        for each (u_j, t_l) ∈ E such that t_l ∈ C(v_i) do
5           c(u_j, t_l, v_i) := e_s(u_j) + ε(u_j) + λ_s + L × ω(u_j, t_l) + λ_r;
6           for p = l, ..., k do
               c(u_j, t_l, v_i) := max{c(u_j, t_l, v_i), e_s(t_p)} + ε(t_p);
            end for each
7           if (maxc < c(u_j, t_l, v_i)) then
8              maxc := c(u_j, t_l, v_i);  mtask := u_j;
            end if
        end for all
```

Task u_j is a possible candidate to be incorporated into cluster $C(v_i)$ if $u_j \in iancs(C(v_i))$. Under situation (a), for each task u_j, BNR calculates the earliest time that v_i can start **due solely** to the chosen task u_j (known as the *critical ancestor path cost* $c(u_j, t_l, v_i)$) to be the sum of: the execution of u_j $(e_s(u_j) + \varepsilon(u_j))$; the sending overhead (λ_s) (due to the $LogP$ general scheduling restrictions, there is no need to determine the number of overheads and gaps g incurred, since BNR only permits a cluster to send one message at the expense of needing multiple copies of the same cluster); the communication delay $(L \times \omega(u_j, t_l))$; the receiving overhead (λ_r); and the computation time of tasks t_l to t_k (i.e. the critical cluster path cost for task t_l to t_k). The task u_j whose $c(u_j, t_l, v_i)$ is the largest is the immediate ancestor of cluster $C(v_i)$ who most delays task v_i. This function can be seen in Algorithm 2 which assigns *mtasks* and *maxc* to the critical immediate ancestor and its critical ancestor path cost, respectively (line 8).

If *maxc* is greater than the critical cluster path cost, $m(C(v_i))$, than task *mtask* becomes the chosen candidate for inclusion into cluster $C(v_i)$. If this is not the case, the process stops and cluster $C(v_i)$ is now complete. This forms

the condition of the main loop in the *construct-cluster* function as shown in lines 5 to 11 of Algorithm 3. Before committing $mtask$ to $C(v_i)$, it is necessary to compare the cost $maxc$ with situation (b) – the critical cluster path cost of the cluster $C(v_i) \cup \{mtask\}$. If $maxc$ is less than this new cost, task $mtask$ is not included (line 11) and the formation of cluster $C(v_i)$ is finished. Otherwise, task $mtask$ is included, critical ancestor path costs are calculated for the new set of tasks in $iancs(C(v_i))$ (line 9), the start time of v_i is updated (line 10) and the loop is repeated. Note that it is not necessary for all of the ancestors of v_i to be visited when constructing the cluster $C(v_i)$.

Algorithm 3 : *construct-cluster* (v)

```
1   if pred(v) = ∅ then e_s(v) := 0;
2   else
3      (maxc, mtask) := critical-ancs-path(v, C(v));
4      m(C(v)) := 0; e_s(v) := max{m(C(v)), maxc};
5      while (m(C(v)) < maxc);
6         C(v) := C(v) ∪ {mtask}; m'(C(v)) := m(C(v));
7         m(C(v)) := critical-cluster-path(C(v));
8         if (m(C(v)) ≤ maxc) then
9            (maxc, mtask) := critical-ancs-path(mtask, C(v));
10           e_s(v) := max{m(C(v)), maxc};
          else
11           e_s(v) := max{m'(C(v)), maxc}; C(v) := C(v) − {mtask};
      end while
    end if
```

2.3 The second phase

The second phase of the approach is to determine which of the generated clusters are needed to implement the schedule. This can be achieved by simply tracing the cluster dependencies starting with the sinks. Note that multiple copies of clusters may be required. These schedule clusters are then allocated to processors, at worst one cluster per processor. In some cases, there is scope for optimisation: clusters whose execution times do not overlap can be mapped to a single processor; depending on the communication parameters, the number of copies of a cluster could be reduced without affecting the makespan, e.g. clusters have the ability to multicast under the delay model.

2.4 Algorithm analysis

At first glance, BNR appears to be similar to that of the PLW algorithm of Palis *et al.* [14] ignoring the differences for $LogP$ scheduling. Both of these algorithms consist of two phases, utilise task replication to form clusters (one for each task in G), even growing them by adding a single task at a time. However, the key difference is the use of the *earliest schedule time* to determine the start time of

owner tasks (i.e. only for the original tasks in G and not their copies) rather than the *earliest start time* for all tasks including their copies. (Note that the earliest start time is a bound [14] since it may be impossible to actually schedule a task at this time [15]). A second difference is the execution order of tasks within a cluster. Note that PLW orders tasks in nondecreasing order of their earliest start time. The effect of these differences are highlighted by the results presented in the next section.

The complexity of PLW is $\mathcal{O}(n^2 \log n + ne)$, where $n =\mid V \mid$ and $e =\mid E \mid$. For BNR, the *critical-cluster-path* function is clearly $\mathcal{O}(n)$. Examining function *critical-ancs-path*, $\mathcal{O}(e)$ edges could emanate from tasks in $iancs(v_i)$. Given k, the number of tasks currently in cluster $C(v_i)$, there are at most $(n-k)$ tasks in $iancs(C(v_i))$. Consequently for the worst case number of edges, as the number of tasks in $C(v_i)$ grows, so the number of edges emanating from $iancs(C(v_i))$ diminishes. The complexity of *critical-ancs-path* is then $\mathcal{O}(ek)$ and the whole strategy $\mathcal{O}(n^2ek)$.

3 Results

The results produced by BNR have been compared with the clustering heuristic PLW [14] and DSC [17] using a benchmark suite of DAGs which includes out-trees (OT), in-trees (IT), diamond DAGs (D), a set of randomly generated DAGs (R) and irregular DAGs (I) taken from the literature.

A few of the experimental results obtained are shown in Tables 1, 2 and 3. The subscript of the DAGs represent the number of tasks in the respective graphs. Each heuristic produced a schedule with a (predicted) makespan M and an actual makespan S (only one value appears in the column if M and S are the same). The time for the actual makespan of the schedule was derived from a *parallel machine simulator* [4]. The required number of processors is P.

The first set of experiments focus on the delay model and fine-grained DAGs for the following reasons. The best scheduling algorithms are based on the delay model since it is the standard scheduling model. The delay model is in fact a special case under the $LogP$ model ($g = o = 0$) where tasks can exploit the property of multicasting. Although the scheduling restrictions cause $LogP$ strategies, such as the one proposed in this work, to appear greedy with respect to the number of processors required, it is important to show that the makespans produced and processors used are comparable to those of delay model approaches under their model. Scheduling approaches such as DSC and PLW have already been shown to generate good results (optimal or near-optimal) for various types of DAGs. For this experimental analysis, the graphs were assigned unit execution and unit communication costs and the latency L was set to 1, 2, 5 and 10 units for different tests.

In the 196 experiments carried out in the first set, the schedules produced by BNR were never worse than those of DSC or PLW. BNR was better than DSC in 164 cases and equal in 32, and in 166 and 30 cases, respectively, against PLW. The performances of both DSC and PLW progressively worsen in com-

	Communication Cost = 1						Communication Cost = 2					
	DSC		*PLW*		*BNR*		*DSC*		*PLW*		*BNR*	
DAG	M/S	P	M/S	P	M/S	P	M/S	P	M/S	P	M/S	P
OT_{63}	11	32	8	46	6	32	11	21	8	36	6	32
OT_{511}	17	256	11/12	344	9	256	17	171	13	308	9	256
IT_{63}	11	32	11	21	11	16	11	21	13	21	11	21
IT_{511}	17	256	17	341	17	256	17	171	19/21	341	17	171
D_{64}	22	8	23/28	32	22	6	30	5	28/37	26	28	8
D_{400}	58	20	59/76	200	58	14	78	11	70/97	164	76	14
I_{41} [11]	16	13	16	22	15	10	21	9	18/22	20	17	13
R_{13}	8	5	9	7	7	5	9	3	9	7	8	4
R_{124}	8	67	8/9	76	7	64	10	62	11	73	9	65
R_{310}	18	110	21/22	189	16	105	24/25	106	26/29	173	20	121
R_{510}	20	170	23/25	327	17	184	25	155	31/34	295	21	214

Table 1. Results for communication latency (L) equal to 1 and 2 units.

	Communication Cost = 5						Communication Cost = 10					
	DSC		*PLW*		*BNR*		*DSC*		*PLW*		*BNR*	
DAG	M/S	P	M/S	P	M/S	P	M/S	P	M/S	P	M/S	P
OT_{63}	19	13	6	32	6	32	29/32	16	6	32	6	32
OT_{511}	31/33	121	14	260	9	256	50/53	115	9	256	9	256
IT_{63}	19	13	19	21	17	12	29	16	24	9	22	11
IT_{511}	31	121	29/31	169	30	108	50/49	115	41	73	36	88
D_{64}	55	6	39/56	20	36	11	61/60	3	54/62	17	45	9
D_{400}	163	18	95/160	97	104	35	230/225	15	137/229	66	135	52
I_{41} [11]	31	5	28/26	19	23	8	52	4	34/32	8	29	8
R_{13}	13	1	12	5	11	3	13	1	12	2	12	2
R_{124}	14/16	54	16/17	61	11	60	23	53	20/21	53	15	56
R_{310}	43/45	98	41	131	29	140	73/75	95	75/79	111	42	138
R_{510}	46/44	128	51/55	226	32	229	80	116	71/82	185	47	244

Table 2. Results for communication latency (L) equal to 5 and 10.

parison with BNR as the DAGs effectively become more fine-grained due to the increasing latency value. DSC produces schedules which are, on average, 12.2%, 16%, 32.1% and 39.1% worse than those of BNR for each of the respective latency costs. For PLW these values are 20.6%, 23.9%, 25% and 25.3%. DSC generally utilises the fewest processors since it does not replicate tasks. BNR, on the whole, uses fewer processors than PLW (especially for small latencies), but where this is not true the compensation is the better makespan.

A second set of experimental results is presented in Table 3. This table contains a comparison of makespans produced by the three strategies for a group of documented irregular, non-unit cost graphs used by various researchers. BNR continues to perform well.

	DSC		*PLW*		*BNR*	
DAG	M/S	P	M/S	P	M/S	P
I_{7a} [8]	9	2	8	5	8	3
I_{7b} [9]	8	3	11/12	5	8	3
I_{10} [14]	30	3	27	3	26	3
I_{13} [1]	301	7	275	8	246	8
I_{18} [12]	530†/550	5	490/480	8	370	9

† Note that Reference [12] reports a schedule time of 460 on 6 processors.

Table 3. Results for non-unit (task and edge) cost irregular *DAG*s ($L = 1$).

4 Conclusions and future work

Based on the results obtained so far, BNR compares favourably against traditional clustering-based scheduling heuristic such as DSC and PLW which are dedicated exclusively to the delay model. Clearly, results of further experiments using graphs with a more varied range of granularities and connectivities are needed to complete the practical evaluation of the algorithm. The main objective is, however, to analyse the heuristic's behaviour under $LogP$ conditions, comparing results under these conditions with other $LogP$ scheduling algorithms. In particular, we wish to investigate MSA [2] to ascertain the benefits of its technique of bundling messages in a $LogP$ environment. In terms of obtaining theoretical bounds on makespan optimality under $LogP$ conditions, further work needs to be done although we suspect that under the delay model the bound is no worse than that of PLW.

Acknowledgments

The authors would like thank Tao Yang and Jing-Chiou Liou for providing the DSC and PLW algorithms, respectively, and for their help and advice.

References

1. I. Ahmad and Y-K Kwok. A new approach to scheduling parallel programs using task duplication. In K.C. Tai, editor, *International Conference on Parallel Processing*, volume 2, pages 47–51, Aug 1994.
2. C. Boeres and V. E. F. Rebello. A versatile cost modelling approach for multicomputer task scheduling. *Parallel Computing*, 1999. (to appear).
3. C. Boeres and V.E.F. Rebello. Versatile task scheduling of binary trees for realistic machines. In C. Lengauer, M. Griebl, and S. Gorlatch, editors, *The Proceedings of the Third International Euro-Par Conference on Parallel Processing (Euro-Par'97)*, LNCS 1300, pages 913–921, Passau, Germany, August 1997. Springer-Verlag.
4. C. Boeres, V.E.F. Rebello, and D. Skillicorn. Static scheduling using task replication for LogP and BSP models. In D. Pritchard and J. Reeve, editors, *The*

Proceedings of the Fourth International Euro-Par Conference on Parallel Processing (Euro-Par'98), LNCS 1470, pages 337–346, Southampton, September 1998. Springer-Verlag.

5. 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.
6. D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. 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.
7. J. Eisenbiergler, W. Lowe, and A. Wehrenpfennig. On the optimization by redundancy using an extended LogP model. In *Proceeding of the International Conference on Advances in Parallel and Distributed Computing (APDC'97)*, pages 149–155. IEEE Comp. Soc. Press, 1997.
8. A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16:276–291, 1992.
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. S.J. Kim and J.C. Browne. A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proceedings of the 3rd International Conference on Parallel Processing*, pages 1–8, 1988.
12. Y. K. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating tasks graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):505–521, May 1996.
13. W. Lowe and W. Zimmermann. On finding optimal clusterings of task graphs. In *Aizu International Symposium on Parallel Algorithm and Architecture Synthesis*, pages 241–247. IEEE Computer Society Press, 1995.
14. M.A. Palis, J.-C Liou, and D.S.L. Wei. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):46–55, January 1996.
15. C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328, 1990.
16. D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Question and answers about BSP. *Scientific Computing*, May 1997.
17. T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.
18. W. Zimmermann, M. Middendorf, and W. Lowe. On optimal k-linear scheduling of tree-like task graph on LogP-machines. In D. Pritchard and J. Reeve, editors, *The Proceedings of the Fourth International Euro-Par Conference on Parallel Processing (Euro-Par'98)*, LNCS 1470, pages 328–336, Southampton, September 1998. Springer-Verlag.