Performance Evaluation of Scheduling Precedence-Constrained Computations on Message-Passing Systems

M. Al-Mouhamed and A. Al-Maasarani

Abstract

Scheduling precedence graphs with communication times is the theoretical basis for achieving efficient parallelism in message-passing machines. The lack of global information on the tasks, due to communication, has lead to develop local scheduling heuristics such as the *Earliest-Task-First.* Using knowledge on computation, communication, and system topology, a class of global priority-based scheduling heuristics called *Generalized List Scheduling* is proposed. The task-level is evaluated by backward scheduling the computation over the multiprocessor by using the best local heuristic. This leads to realistic measurement of the task priority for use in forward GLS scheduling.

Experimental evaluation of local and GLS heuristics is carried out using extensive random graph generation and altering over the communication, inherent parallelism, and system topology. Analysis shows that local heuristics rely on locally maximizing the processor efficiency and gives acceptable deviations only when the inherent parallelism is large enough to cover the effective communication. This leads the local heuristics to achieve bounded speedup.

GLS scheduling is based on combining two strategies: 1) differentiate critical computation and communications from others by scheduling critical paths first, and 2) implement effective management of processor utilization in order to increase the speedup. GLS scheduling maintains acceptable relative deviation versus change in parallelism, communication, and multiprocessor topology. The time complexity of GLS heuristics is $O(pn^2)$, where p and n are the number of processors and that of the tasks, respectively.

Keywords: Bounds, Distributed Processing, Heuristics, Performance, Scheduling.

1 Introduction

A fundamental result of scheduling theory is the introduction of the list scheduling [1, 2] and the establishment of its performance guarantee. Graham's List-Scheduling is based on: 1) evaluation of the task level or length of the shortest path from starting a task to completion of the computation, and 2) minimizing the processor idle time by scheduling the ready task with the highest level on any idle processor. Empirical testing of scheduling [3] deterministic and stochastic computations has proved that list-scheduling is near optimum as the finish time deviates by at most 4% from the optimum solution that is measured using Fernandes-Bussel's [4] lower bound. The success of list-scheduling is due to the use of task-level to differentiate critical from non-critical tasks and the simplicity of the model for which the inter-task communication overhead is negligible compared to the computation. The assumption on the communication has been largely based on shared memory architectures. But due to significant communication overhead, this assumption cannot be justified [5, 6] for message-passing architectures. Therefore, the need to handle the communication and the multiprocessor topology in designing effective scheduling for message-passing systems.

Linear clustering [7] has been applied for iteratively merging the most communicating paths as an attempt to optimize the computation time. After multiple refinements the resulting graph is mapped onto the target multiprocessor using graph theoretic approach. Another method based on clustering immediate tasks [8] has been proposed for minimizing the critical path length over infinite number of processors. When the minimum critical path is found, merging operations are performed in order to match the clusters with the number of processors. Methods based on branch-and-bound [9] and simulated annealing [10] have also been used as heuristics for mapping and partitioning computations. These approaches either minimize objective functions other than the computation finish time or lack global evaluation because only sub-problems are investigated. We focus on low cost compact scheduling heuristics that can be applied to regularly and irregularly connected network of processors.

A set of $\Gamma(T_1, \ldots, T_n)$ of n tasks (T) with their precedence constraints and communication costs are to be scheduled on p identical processors so that their overall execution time is minimum. The computation can be modeled [11, 12] by using a directed acyclic task graph $G(\Gamma, \rightarrow, \mu, c)$ where $\rightarrow$, $\mu(T)$, and c(T,T') denote the precedence constraints, the task execution time, and the number of communication messages (volume of data) that are sent from task T to its immediate successor T', respectively. The multiprocessor is denoted by $S(P, R)$ where P is a set of processors and R is the the interconnection network. The time to transfer a unit of message from a processor $p(T)$ to processor $p(T')$ is denoted by $r(p(T), p(T'))$, therefore the time to transfer $c(T, T')$ messages is $c(T, T')r(p(T), p(T'))$, i.e. the communication media is assumed to be contention free. System $S(P, R)$ enables modeling loosely-coupled, regularly and irregularly connected multiprocessors.

To differentiate between critical and non-critical computation, one needs to define a global task-level that accounts for the characteristics of the computation and the multiprocessor. The multiprocessor topology and communication costs strongly affect the behavior of the computation, and therefore, knowledge on the multiprocessor is needed in evaluating the task-level ($l(T)$). The task-level is defined as the length of the shortest path from starting that task to any exit node such that all the precedence and communication constraints are satisfied. Unfortunately, the accurate evaluation of the $lst(T)$ is very difficult because: 1) the length of paths, in the task-graph, are differently affected depending on the way each path is mapped onto the processors, and 2) by definition the $l(T)$ derive from the optimum solution that is obtained using infinite or finite number of processors. To avoid these problems, only local scheduling heuristics have been proposed such the *earliest-task-first [12] and the* largest-communication-first [13]. Discarding all the communication aspects [14], or accounting for all of them, in evaluating the task-priority leads to excessively inaccurate evaluations that fail specifying the degree of criticality of the tasks with respect to the overall computation.

Our objective is to find configuration-dependent global scheduling heuristics by generalizing the concept of task-level so that to incorporate the available knowledge such as that of the computation graph, communication aspects, and multiprocessor topology. Using the generalized task-level, a number of scheduling heuristics are proposed by combining the task-level with efficient management of the processor idle times in defining a new global task-priority concept. This methodology extends the list scheduling concept, that has previously been applied to computation model $G(\Gamma, \rightarrow, \mu)$, to a new approach called *Generalized List Scheduling* (GLS) that is applied to schedule the computation model $G(\Gamma, \rightarrow, \mu, c)$ over system $S(P, R)$. Experimental evaluation of local and GLS scheduling is carried out using random generation of computation graphs and scheduling using different strategies, processor utilization techniques, and multiprocessor topologies. Analysis of local and global scheduling outlines the relative merit of each heuristic with respect to the task-selection strategy, amount of communication/computation, inherent parallelism, and multiprocessor topology.

2 Evaluation of the global task-level

Let T be a task and denote by $D(T)$ the set of predecessors of T. By considering only one predecessor task, the earliest-starting-time $est(T,p)$ of T for some processor p depends on the finish time $f(T')$ of the predecessor $T' \in D(T)$, the number of messages $c(T,T')$ sent from task T to T', and the processor p' on which task T' has been running:

$$est(T,p) = f(T) + \begin{cases} 0 & \text{if } p = p' \\ c(T',T).r(p',p) & \text{otherwise} \end{cases} \tag{1}$$

By considering all the predecessors of T, the $est(T,p)$ is the earliest time the latest message from the predecessors reaches processor p:

$$est(T,p) = \max_{T' \in D(T)} \{f(T') + c(T',T).r(p(T'),p(T))\} \tag{2}$$

As the $est(T,p)$ depends on the routing cost $r(p,p')$, then there exists a processor p^* for which task T can start at the earliest time ($est(T)$) among all the available processors:

$$est(T) = est(T,p^*) = \min_p \{est(T,p)\} \tag{3}$$

The effective earliest-starting-time ($est\ (T,p^*)$) is the least time at which T can start on some processor p^* by considering the precedence of T and the current free time $f(p)$ of every processor p:

$$est\ (T,p^*) = min_p\{\max\{est(T,p), f(p)\}\ \} \tag{4}$$

The earliest-completion-time $ect(T)$ is simply $est(T) + \mu(T)$. The earliest-completion-time provides a heuristic approach to measurement of the shortest path from starting the computation graph to completion of task T such that all the precedence constraints are preserved. The task-level of T is the length of the shortest path from starting T to completion of any exit node of the task graph. A heuristic approach to evaluate the task-level can be obtained by evaluating the earliest-completion-time of the tasks using the dual task graph, i.e. the latest-starting-time of the tasks in the original task-graph. The dual task graph is obtained by reversing all the arc direction in the original task graph. On the other hand, applying Eq. 3 to the computation of the $ect(T)$ may lead two different tasks to occupy the same activity level of one processor, i.e. one processor is implicitly assumed to evaluate more one task at a time. For this Eq. 3 does not lead to achievable evaluation of the task-levels.

An algorithm called GTT is proposed to evaluate generalized task-level or $ect(T)$ as obtained by scheduling the dual task-graph $G_d(\Gamma, \rightarrow, \mu, c)$ starting with the entry nodes, allocate them their $ect(T)$ times, and propagate this process down to every task whose predecessors in G_d have all been allocated their $ect(T)$ times until the exit nodes. The scheduling of $G_d(\Gamma, \rightarrow, \mu, c)$ over system $S(P,R)$ is performed using the best known local heuristic that uses the earliest-task-first as defined by the effective earliest-starting-time in Eq. 4. Algorithm GTL uses set B to store the tasks that have been allocated their $ect(T)$ and set A to store the tasks that have no predecessors or whose predecessors all belong to B. Function $f(p)$ denote the current finish time of processor p. We assume function $\lambda_d(T)$ is initialized to the number of predecessors of T. Algorithm GTL is the following:

```
(1) Obtain the dual task graph $G_d(\Gamma, \rightarrow, \mu, c)$
(2) Initialize: $A \leftarrow \{T : D(T) = \emptyset\}$, $est(T,p) = 0$
        for each task and each processor.
(3) While $|B| < n$ Do
        Begin
        (3.1) Select the task $T^* \in A$ and processor $p^*$ that satisfy:
                $est(T^*, p^*) = min_{T \in A}\{min_p\{est(T,p)\}\}$
        (3.2) Assign $T^*$ on $p^*$ : $p(T^*) = p^*$, $ect(T^*) = est(T^*, p^*) + \mu(T^*)$,
            $f(p^*) = ect(T^*)$, Remove $T^*$ from $A$, Append $T^*$ to $B$,
            For each $T \in A$, update $est(T, p^*) = max\{est(T, p^*), f(p^*)\}$
        (3.3) Repeat for each task $T \in S(T^*)$ : $\lambda_d(T) = \lambda_d(T) - 1$ ,
            If $\lambda_d(T) = 0$ then
            Update $A$ : $A \leftarrow A + \{T\}$,
            Evaluate the effective $est(T,p)$ for each processor $p$:
            $est(T,p) = \max\{\max_{T' \in D(T)}\{f(T') + c(T',T).r(p(T'),p)\}, f(p)\}$
        End.
```

The output of LST is the list of task-level $l(T) = ect(T)$ that represents an approximation of the shortest path from task T to any exit node of $G(\Gamma, \rightarrow, \mu, c)$. The main loop of GTL is statement 3 that executes n times because one task is allocated for each run of the body. Statement 3.1 executes at most pn times in order to select one task. Statement 3.2 updates the parameters but its last statement executes n times. Operation $\lambda_d(T) = \lambda_d(T) - 1$ in statement 3.3 executes $O(n^2)$ times but the condition $\lambda_d(T) = 0$ occurs only once for each task. The time complexity of GTL is then $O(pn^2)$.

3 Generalized List Scheduling

In this section we generalize the Graham's list scheduling by defining global priority based scheduling heuristics that incorporate the effect of inter-task communication and multiprocessor topology. The new scheduling is called *Generalized List Scheduling* (GLS). A heuristic that belong to this class consists of two steps: 1) obtain the priority list of the tasks by using algorithm GTL, 2) scheduling: among the ready-to-run tasks, select the most prior task and assign it to run at the earliest. The second step of GLS heuristics can be implemented using different strategies depending on how the task-priority is mapped to the task level.

For GLS scheduling, there are two approaches to control the scheduling process: *Processor-driven (PD) and* Graph-driven (GD). The PD approach consists of updating the set of ready-to-run (RTR) tasks when any processor completes execution of some task and becomes idle. The successors of those newly completed tasks are the only to be involved in the updating process. This leads the PD scheduler to track the increasing sequence of processor completion times. For example, algorithm [12] implements the local strategy called *earliest-task-first* by using the PD approach.

The GD approach consists of updating the set RTR following the starting of each task and only the successors of this task are involved in the updating process. This anticipating process promotes in-depth expansion of the task-graph compared to the rather horizontal expansion in case of PD. These approaches will be investigated in the evaluation.

Depending on the how task-selection maps into the generalized task level and the incurred processor idle time, we define the following GLS heuristics. Heuristic GD/HLF is Graph-driven/ Highest-level-first, i.e. highest $l(T)$ first. Heuristic PD/HLF uses the processor driven approach. Selecting tasks according to the HLF criteria may lead to increasing the processor idle time that precedes the starting of the highest level task. Therefore, a heuristic that imposes a penalty function of the idle time that precedes its starting time consists of defining the task-priority as $l(T) - est(T)$, where $l(T)$ is the length of the shortest path from starting T to any exit node as achieved by heuristic GTL and $est(T)$ is the effective earliest-starting-time. This approach leads to define the heuristics GD/HLETF and PD/HLETF that are called Highest-level-earliest- task-first. According to the level function, the selected task T satisfies $l(T) - est(T) \geq l(T_i) - est(T_i)$ for any RTR task T_i. In other words, we have: $l(T) - l(T_i) \geq est(T) - est(T_i)$, i.e. to select task T the difference in levels between T and T_i should be higher than the amount of idle time $(est(T) - est(T_i))$ that would be saved if T_i was selected first.

In the following we present algorithm GD/HLETF as one representative heuristic for the GLS class. GD/HLETF uses the sets that have been defined for in algorithm LST. The inputs to GD/HLETF are the task-graph and the list of $ect(T)$ times that are generated by LST. This heuristic consists of selecting a task T^* and a processor p^* such that $l(T^*) - est(T^*, p^*)$ is the highest among all the RTR tasks. Following the scheduling of T^* on p^*, the time at which p^* becomes free is $f(p^*) = est(T^*, p^*) + \mu(T^*)$ is used to update the $est(T, p^*)$ for all the RTR tasks, i.e. $est(T, p^*) \leftarrow \max\{est(T, p^*), f(p^*)\}$. The outputs are the starting time $st(T)$ of each task and the processor $p(T)$ on which T is assigned. Algorithm GD/HLETF is the following:

```
(1) Initialize: $A \leftarrow \{T : D(T) = \emptyset\}$, $est(T, p) = 0$ for each task $T$ and each processor $p$,
                $B \leftarrow \emptyset$, $f(p) = 0$ for each processor.
(2) While $|B| < n$ Do
        Begin
        (2.1) Select the task $T^* \in A$ and processor $p^*$ that satisfy:
                  $l(T^*) - est(T^*, p^*) = \max_{T \in A}\{l(T) - \min_p\{est(T, p)\}\}$
        (2.2) Assign $T^*$ on $p^*$ : $p(T^*) = p^*$, $st(T^*) = est(T^*, p^*)$, $f(p^*) = st(T^*) + \mu(T^*)$,
              Remove $T^*$ from $A$, Append $T^*$ to $B$,
              For each $T \in A$, update $est(T, p^*) = max\{est(T, p^*), f(p^*)\}$
        (2.3) Repeat for each task $T \in S(T^*)$ : $\lambda_d(T) = \lambda_d(T) - 1$ ,
              If $\lambda_d(T) = 0$ then
                  Update $A$ : $A \leftarrow A + \{T\}$,
                  Evaluate the effective $est(T, p)$ for each processor $p$:
                  $est(T, p) = \max\{\max_{T' \in D(T)}\{f(T') + c(T', T).r(p(T'), p)\}, f(p)\}$
        End.
```

Similar analysis to that of LST shows that the time complexity of GD/HLETF is $O(pn^2)$. Our main concern is to promote the processor utilization for global-priority based scheduling algorithm. An optimization technique that can reduce the processor idle time is to attempt filling the idle time that precedes the starting of the most prior task T by a less prior task T' provided that this operation does not lead to delay T whose scheduling decision will be postponed without affecting its earliest starting time. If such task T' is found, then updating the set RTR by eventually adding some successor T'' of T' cannot cause any delay to T because $l(T) - est(T) \geq l(T'') - est(T'')$ because $T'' \in D(T')$ implies that $l(T') - est(t') \geq l(T'') - est(T'')$. This method allows defining heuristics PD/HLETF* and GD/HLETF* that apply the above idle time optimization techniques. This approach leads to only increasing the constant in pn^2, thus leaves the time complexity as $O(pn^2)$.

4 Experimental Evaluation

The objective is to compare performance of local scheduling heuristics and the proposed approach that is based on pre-evaluation of the task-priority and generalized list scheduling. For this we consider the local heuristic PD/ETF [12] and GD/ETF that is identical to steps 2 and 3 of algorithm GTL. A heuristic called *Random* is used to randomly select tasks and assigned them to run at their earliest starting times. This is useful to compare the effect of random and deterministic task selections.

A *random graph generator* (RGG) is used for generating computation graphs with few tasks hundred tasks and with task computation time ranging from 10 to 190 time units. The average communication cost, number of level, and the number of processors are indirectly controlled using the parameters: 1) the ratio ($\alpha = c_{arc}/\mu_T$) of average communication carried by each edge (C_{arc}) to the average task computation time (μ_T), 2) the degree of parallelism ($\beta = N_T/N_L.p$) that is the average number of tasks (N_T) over the product of the average number of levels (N_L) by the number of processors p, and 3) the topology of the interconnection network that is the fully-connected (FC), the hypercube (HC), and the ring (RG).

The studied ranges of α and β is $[0-3]$ and $[0.5-4]$, respectively. The variance on C_{arc} is set to 50% of the current average of C_{arc}. Each graph has at least 6 levels and 70% of the outgoing arcs from one level are incoming arcs to the next level and the remaining 30% reach arbitrary forward levels. For each instance of these parameters, the RGG uses the uniform distribution in order to generate 500 random computation graphs that are scheduled by each of the previously defined heuristics. The shortest finish time that is achieved by some heuristic for a given task graph is denoted by (ω_{best}) and used as a reference of the optimum solution.

4.1 Deterministic Versus Random Task Selection

Figure 1 shows the average percent deviation of the finish time as achieved by *Random* over the FC topology. For low values of parallelism ($\beta = 0.5 - 1$), *Random* deviates the least compared to its deviation at higher level of parallelism because at lower parallelism the size of the set RTR is small anyway. Random task selection may increase the the finish time up to 40% on the average. Therefore, deterministic task selection is needed especially when the parallelism exceeds some threshold ($\beta \geq 2$), i.e. there are at least two tasks that compete for each processor on the average.

4.2 Global Priority and Processor-Driven

Heuristics PD/HLF and PD/HLETF generate finish times that significantly deviates from ω_{best} with increasing parallelism and communications. Typically, PD/HLF and PD/HLETF deviates by 60% and 9% on the average. Figure 2 shows the average deviation for PD/HLETF with the HC topology that is qualitatively representative for those obtained with the FC and RG. The reason for these significant deviations is that the PD approach leads global priority-based selection to inefficient management of the processor idle time. PD/HLETF with its task-priority definition $l(T) - est(T)$ greatly improves the performance of PD/HLF but still gives unacceptable deviation in the general case. The PD approach is not adequate for global priority-based selection because of unfair balancing between task-level and processor utilization.

4.3 Local Heuristics

Heuristics PD/ETF and GD/ETF have nearly the same average deviation from ω_{best} with small advantage to GD/ETF (2%). The PD and GD approaches are identical within the framework of local scheduling. However, the slight advantage of GD/ETF over PD/ETF is due to the use of the effective earliest-starting-time in GD/ETF (Eq. 4) againsd the theoretical one (Eq.3) in PD/ETF. Figure 3, 4, and 5 show the average deviation of PD/ETF from ω_{best} for the FC, HC, and RG topologies, repectively. A deviation of 5% is achieved by PD/ETF only when $\beta/\alpha \geq \epsilon_{top}$, where ϵ_{top} is a topology dependent parameter. Using the definition of α and β, we have:

$$\frac{N_T}{N_L}.\frac{\mu_T}{c_{arc}} \geq \epsilon_{top}.p \tag{5}$$

Therefore, to achieve acceptable deviation (5%) the inherent parallelism (N_T/N_L) and the communication ratio (c_{arc}/μ_T) impose a bound on the number of processors used. The local heuristics PD or GD/ETF require the parallelism to be higher than some threshold, depending on the communication, in order to achieve acceptable deviation. We conclude that local heuristics that are based on earliest-task-first rely on overlapping computation and communication as a strategy to minimize the finish time through management of the processor utilization. Therefore, these heuristics require increasing the parallelism, or decreasing the number of processors, in order to achieve acceptable global finish time. This effect appears clearly for connectivity-restricted topologies as shown on Figures 3, 4, and 5.

4.4 Generalized List Scheduling

The heuristics GD/HLF, GD/HLETF, and GD/HLETF* give acceptable deviation from ω_{best} for low to average communication ($0 \leq \alpha \leq 1.5$). To save the area of this paper, only the plots of the average deviation for heuristic GD/HLETF* are shown on Figures 6, 7, and 8 for the FC, HC, and RG topologies, respectively. While GD/HLF deviates by more than 8% for ($\alpha > 1.5$), heuristic GD/HLETF has overcome most of the deficiency of GD/HLF with respect to processor utilization because GD/HLETF slightly increases its deviation with increasing communication. For all studied levels of parallelism, the peak deviation of GD/HLETF is 4.5%, 6%, and 7.5% for the FC, HC, and RG topologies, respectively. Heuristic GD/HLETF* achieves the lowest average deviation that is nearly 2% for all studied level of parallelism and communication. This shows a clear advantage of global priority-based scheduling over the local approches. The slight deviation of GD/HLETF* compared to those of GD/HLF and GD/HLETF indicates that the major issue is to combine the task-level with efficient management of the processor idle times. This objective seems to be achieved within heuristic GD/HLETF* that maintains small deviation over the studied range of communication, parallelism, and multiprocessor topologies.

4.5 Analysis of the Distribution

Analysis of the distribution is carried out for PD/ETF and GD/HLETF* because these heuristics are representative of local and GLS scheduling, respectively. Figures 9 and 10 show the boundary for the best 50% and 90% population of the finish times versus the available parallelism (β) for PD/ETF and GD/HLETF*, respectively. Each point of the boundary plots is taken as the maximum deviation for all levels of studied communications.

While the 50% boundary for PD/ETF is at the 10% deviation level, that of GD/HLETF* does not exceed the 1.5% level. The 90% boundary is nearly about 18% for PD/ETF against 4% to 7% maximum deviation for GD/HLETF *. Changing the topology from FC, to HC, and to RG has the effect of increasing the communication requirements on the original computation but the general shape of the distributions is nearly maintained. PD/ETF is more sensitive to the inherent parallelism than GD/HLETF *. PD/ETF slightly reduces its 50% deviation boundary versus increasing parallelism while GD/HLETF * maintains constant deviation at the same boundary level. The dependency on parallelism and topology appears only at the 90% boundary level for GD/HLETF *.

5 Conclusion

Using knowledge on computation, communication, and multiprocessor topology, a class of global priority-based scheduling heuristics called *Generalized List Scheduling* or GLS has been proposed. Global task-level is heuristically evaluated by scheduling the dual task-graph, that is used to model the computation, over the target multiprocessor. Using the best known local scheduling, the task-level or length of the shortest path from starting that task to exit node is approximated to the task completion time that is achieved in scheduling the dual task graph. GLS scheduling operates on the forward task-graph by using differential task-priority concept based on the generalized task-level and the incurred processor idle time.

Experimental evaluation of local and GLS scheduling is carried out by stepping over the communication and parallelism and by considering different multiprocessor topologies. The communication media was assumed to be contention free. Analysis shows that local scheduling rely on maximizing the processor utilization in order to minimize the finish time. This strategy leads to acceptable deviation, from the best known solution, only when the available parallelism is sufficient to cover the communications, and therefore, leads to limit potential speedup.

Even with approximate task-level, GLS scheduling maintains acceptable deviation from the best known solution versus increasing parallelism, communication, and restricting the multiprocessor topology. The efficiency and compactness of GLS scheduling makes it very attractive for compile-time scheduling over regularly and irregularly connected multiprocessor topologies.

Future extension is to find more refined evaluation for the task-level by using local search or iterative pre-processing. The use of generic model for the interconnection network to account for the communication delay is interesting to make the evaluation sharper and more realistic. Finally, extension of this work to non-deterministic or incompletely specified computation such as bounding the knowledge to just few task-levels is useful generalization of GLS scheduling to dynamic environment.

References

[1] Coffman,E.G., and Denning, P.J., *'Operating Systems Theory', (eds) Prentice-Hall, 1973.*

[2] Coffman, E.G. at als, *'Computer and Job-Shop Scheduling Theory', (Eds) John Willy & Sons, 1976.*

[3] Adam, T.L., Chandy, K.M., and Dickson, J.R., *'A Comparison of List Schedules for Parallel Processing Systems', Comm. of the ACM, Vol 17, No 12, Dec. 1974, pp 685-690.*

[4] Fernandez, E.B., and Bussell, B., *'Bounds on the Number of Processors and Time for Multiprocessors Opyimal Schedules', IEEE Trans. on Comp., C-22, No. 8, Aug. 1973, pp. 745-715.*

[5] McGreary, C. and Gill, H., *Automatic Determination of Grain Size for Efficient parallel Processing', Comm. of the ACM, Vol. 32, No. 9, Sep. 1989, pp. 1073-1078.*

[6] Pase, D.M., *A Comparative Analysis of Static Parallel Schedulers where Communication Costs are Significant', Ph.D. thesis, Oregon, Jul. 1989.*

[7] Kim, S.J., and Browne, J.C., *'A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures', Proc. of the Inter. Conf. on Parallel Processing, Vol. 3, Aug. 1988, pp. 1-8.*

[8] Sarkar, V., and Hennessy, J., *'Compile-time Partitioning and Scheduling of Parallel Programs', Proc. of the SIGPLAN Symp. on Compiler Construction, Jul. 1986, pp. 17-26.*

[9] Richard Ma, P.Y., Lee, E.Y.S., and Masahiro, T., *A Task Allocation Model for Distributed Computing Systems, IEEE Trans. on Computers, Vol. C-31, Jan. 1982, pp. 41-47.*

[10] Sheild,J., *'Partitioning Concurrent VLSI Simulated Programs onto Multiprocessor by Simulated Annealing', IEEE proceedings, No. 134, Jan. 1987, pp. 24-30.*

[11] Rayward-Smith, V.J., *'UET Scheduling with Interprocessor Communication Delays', Report SYS-C86-6, Information Systems, University of East Anglia, Norwich, U.K., 1986.*

[12] Hwang, J.-J., Chow, Y.-C., Anger, F.D., and Lee, C.-Y., *'Scheduling Precedence Graphs in Systems with Interprocessor Communication Times, SIAM Computing, Apr. 1988, pp. 244-257.*

[13] Al-Mouhamed, M., *'Analysis of Macro-Dtaflow Dynamic Scheduling on Non-Uniform Memory Access Architectures', To appear in IEEE Trans. on Parallel and Distributed Systems, 1993.*

[14] Sih, G.C., and Lee, E.A., *'Scheduling to Account for Interprocessor Communication within Interconnection-Constrained Processing Network', Inter. Conference on Prallel Processing, Vol. I, 1990, pp. 9-16.*