Performance Evaluation of Scheduling Precedence Constrained Computations on Message Passing Systems

Mayez Al-Mouhamed and Adel Maasarani

Department of Computer Engineering
King Fahd University of Petroleum and Minerals (KFUPM)
31261 Dhahran, Saudi Arabia

Abstract: Using knowledge on computation communication and multiprocessor topology, a class global priority based scheduling heuristics, called generalized list scheduling (GLS) is proposed. Task-priority is defined as the completion of time of the task following backward scheduling the computation over the multiprocessor but using the best local heuristic. GLS scheduling consists of using the task priority in forward, graph driven scheduling. Evaluation of local (ETF) and GLS heuristics is carried out by altering over the communication parallelism and system topology. Analysis shows that local heuristics relying on locally maximizing the efficiency and gives accessible solutions only when the parallelism is large enough to cover the communication (bounded speedup). GLS scheduling outperforms the local approaches versus change in parallelism, communication, and network topology. The time complexity of GLS is O(pn^2) where p and n are the number of processors and that of the tasks, respectively.

Index terms: bounds, distributed systems, heuristics, performance, scheduling.

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

Mayez Al-Mouhamed and Adel Al-Maasarani

***Abstract*—Using knowledge on computation, communication, and multiprocessor topology, a class of global priority-based scheduling heuristics, called *Generalized List Scheduling* (GLS) is proposed. Task-priority is defined as the completion time of the task following backward scheduling the computation over the multiprocessor by using the best local heuristic. GLS scheduling consists of using the task-priority in forward, graph-driven scheduling.**

Evaluation of local (ETF) and GLS heuristics is carried out by altering over the communication, parallelism, and system topology. Analysis shows that local heuristics rely on locally maximizing the efficiency and gives acceptable solutions only when the parallelism is large enough to cover the communication (bounded speedup). GLS scheduling outperforms the local approaches versus change in parallelism, communication, and network topology. The time complexity of GLS heuristics in $O(pn^2)$, where p and n are the number of processors and that of the tasks, respectively.

***Index Terms*— Bounds, distributed systems, heuristics, performance, scheduling.**

I. Introduction

A fundamental result of scheduling theory is the introduction of the list scheduling [3] 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. List-scheduling [1] is found to be near optimum as the finish time deviates by at most 4% from the optimum solution. 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. But due to significant communication overhead, this assumption cannot be justified [7] 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 [5] has been applied for iteratively merging the most communicating paths as an attempts 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.

By assuming that delay in communication is mainly due to channel latency, a scheduling based on *Task Duplication* (TD) over idle processors was proposed [6] to reduce the communication. The impact of task and processor selection and TD were experimentally studied in [7] by evaluating the task-priority as the sum of task times (Hu's approach) from the graph bottom. In [10], the ASAP and ALAP times are evaluated by adding up all the computation and communication times along the corresponding paths. The task with the smallest mobility (ALAP-ASAP) is selected first.

Manuscript received October 31, 1992; revised December 27, 1993.

The authors are with the Computer Engineering Department, King Fahd University of Petroleum and Minerals, P.O. Box 787, Dhahran 31261, Saudi Arabia.

IEEE Log Number 9405071.

These approaches either minimize objective functions other than the computation finish time and lack global evaluation because only sub-problems are investigated [5, 8], discarding the communication and network effects in evaluating the task-priority [6, 7], or pessimistically accounting for all of them [10].

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. 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 precedence-constrained computations with communication times over an MIMD message-passing system. Experimental evaluation of local and GLS scheduling is carried out with respect to task-selection strategy, amount of communication/computation, inherent parallelism, and multiprocessor topology.

Section II defines the notation and the problem. Sections III and IV presents the proposed task-level and scheduling, respectively. Section V presents the evaluation and Section VI conclude about this work.

II. Background

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 [4] 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 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 $c(T, T')$ message is $c(T, T')r(p(T), p(T'))$, i.e., the communication media is assumed to be contention free. Parameter $r(p, p')$ is the topology factor that is proportional to the number of hops between p and p'.

To differentiate between critical and noncritical computation, one needs to define a global task-level that accounts for the characteristics of the computation and the multiprocessor. The task-level $(l(T))$ is defined as the length of the shortest path from starting the task to any exit node such that all the precedence and communication constraints are satisfied. Unfortunately, the accurate evaluation of the $l(T)$ 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. To avoid these problems, only local scheduling heuristics have been proposed such the *earliest-task-first* [4] and the *largest-communication- first* [2]. Discarding all the communication aspects [9], [7], [6], or accounting for all of them [10], 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.

III. 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 $\mathrm{est}(T,p)$ 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:

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

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

$$\mathrm{est}(T,p) = \max_{T' \in D(T)} \{f(T') + c(T',T)\cdot r(p(T'),p(T))\}$$

where $f(T') = \mathrm{est}(T',p(T')) + \mu(T')$. In the event $D(T) = \emptyset, \mathrm{est}(T,p) = 0$ for every p. As the $\mathrm{est}(T,p)$ depends on the routing cost $r(p,p')$, then there exists a processor p^* for which task T can start at the earliest time $(\mathrm{est}(T))$ among all the available processors:

$$\mathrm{est}(T) = \mathrm{est}(T,p^*) = \min_p\{\mathrm{est}(T,p)\}. \tag{1}$$

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

$$\mathrm{est}(T,p^*) = \min_p\{\max\{\mathrm{est}(T,p),t(p)\}\}. \tag{2}$$

The earliest-completion-time $\mathrm{ect}(T)$, that is $\mathrm{est}(T) + \mu(T)$, 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. A heuristic approach to evaluate the task-level can be obtained by evaluating the earliest-completion-time $\mathrm{lct}(T)$ that results from backward scheduling the graph over the multiprocessor.

An algorithm called LST is proposed to evaluate the *the latest-completion-time* that defines the task-level in the proposed scheduling. LST operates as follows: 1) Obtain the dual graph (G') by reversing all arc direction in the original graph (G), 2) Schedule G' over system $S(P,R)$ such that the earliest startable task is scheduled first, 3) for every task T, the achieved $\mathrm{ect}(T)$ time becomes the task-level $l(T)$. According to this method, $l(T)$ represents an approximation of the shortest path from task T to any exist node of G that accounts for the precedence constraints, the communication (terms $c(T,T')$), and the network latency (term $r(p,p')$). Each task requires at most pn steps to evaluate its $\mathrm{est}(T,p)$ time and n tasks require $O(pn^2)$ that is the time complexity of algorithm LST.

IV. 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 LST, 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 involved in the updating process. This leads the PD scheduler to track the increasing sequence of processor completion times using a global time. For example, algorithm [4] 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 without global time. 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 how task-selection maps into the generalized task level and the incurred processor idle time, we define the following GLS heuristics. Heuristics GD/HLF is Graph-driven/Highest-level-first, i.e., highest $l(T)$ first. 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 $l(T) - \mathrm{est}(T)$, where $l(T)$ is the length of the shortest path from a starting T to any exit node as achieved by heuristic LST and $\mathrm{est}(T)$ is the effective earliest-starting-time. This approach leads to define heuristic GD/HLETF that is called Highest-level-earliest-task-first. According to the level function, the selected task T satisfies $l(T) - \mathrm{est}(T) \geq l(T_i) - \mathrm{est}(T_i)$ for any RTR task T_i. In other words, we have: $l(T) - l(T_i) \geq \mathrm{est}(T) - \mathrm{est}(T_i)$, i.e., to select task T the difference in levels between T and T_i should be higher than the amount of idle time $(\mathrm{est}(T) - \mathrm{est}(T_i))$ 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. The inputs to GD/HLETF are the task-graph and the list of $\mathrm{ect}(T)$ times that are generated by LST. This heuristic consists of selecting a task T^* and a processor p^* such that $l(T^*) - \mathrm{est}(T^*,p^*)$ is the highest among all the RTR tasks. Following the scheduling of T^* on p^*, the time at which p^* becomes free is $f(p^*) = \mathrm{est}(T^*,p^*) + \mu(T^*)$ is used to update the $\mathrm{est}(T,p^*)$ for all the RTR tasks, i.e. $\mathrm{est}(T,p^*) \leftarrow \max\{\mathrm{est}(T,p^*), f(p^*)\}$. The outputs are the starting time $\mathrm{est}(T)$ of each task and the processor $p(T)$ on which T is assigned. Algorithm GD/HLETF is the following:

```
(1) Initialize: $A \leftarrow \{T : D(T) = \emptyset\}, B \leftarrow \emptyset$, for each $T$ and
      each $p$: $\mathrm{est}(T,p) = 0, t(p) = 0$
(2) While $|B| < n$ Do
      Begin
        (2.1) Select $T^* \in A$ and $p^*$: $l(T^*) - \mathrm{est}(T^*,p^*)$
                $= \max_{T \in A}\{l(T) - \min_p\{\mathrm{est}(T,p)\}\}$
        (2.2) Assign $T^*$ on $p^*$: $p(T^*) = p^*, t(p^*) = \mu(T^*)$
                $+\mathrm{est}(T^*,p^*)$, remove $T^*$ from $A$, add $T^*$ to $B$,
              For each $T \in A$, update:
              $\mathrm{est}(T,p^*) = \max\{\mathrm{est}(T,p^*), t(p^*)\}$
        (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 $\mathrm{est}(T,p)$ for each $p$:
                  $\mathrm{est}(T,p) = \max\{\max_{T' \in D(T)}\{f(T')$
                      $+ c(T',T)\cdot r(p(T'),p)\}, t(p)\}$
      End
```

Similar analysis to that of LST shows that the time complexity of GD/HLETF is $O(pn^2)$. 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 $l(T) - \mathrm{est}(T) \geq l(T'')\mathrm{est}(T'')$ because $T'' \in D(T')$ implies that $l(T') - \mathrm{est}(T') \geq l(T'') - \mathrm{est}(T'')$. This method allows defining heuristic GD/HLETF* that applies the above idle time optimization techniques. This approach leads to only increasing the constant in pn^2, thus leaves the time complexity as $O(pn^2)$.

V. 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. We compare our approach to the well known PD/ETF [4]. 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 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 ($\alpha = c_{\mathrm{arc}}/\mu_T$) of average communication carried by each edge (C_{arc}) to the average task computation time (μ_T), 2) the degree of parallelism ($\beta = N_T/N_{L\cdot p}$ 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 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.

A. Deterministic/Random Selection

Fig. 1 shows the average percent deviation of the finish time as achieved by *Random* over the FC topology. Random task selection may increase the finish time up to 40% on the average. Therefore, deterministic task selection is needed specially 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. The peaks and valleys on the presented plots are due to fluctuation of the reference (ω_{best}) that is an approximation of the optimum solution.

B. 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 (2) against the theoretical one (1) in PD/ETF. Figs. 2–4 show the average deviation of PD/ETF from ω_{best} for the FC, HC, and RG topologies, respectively. A deviation of 5% is achieved by PD/ETF only when $\beta/\alpha \geq \epsilon_{\mathrm{top}}$, where ϵ_{top} is a topology dependent parameter. Using

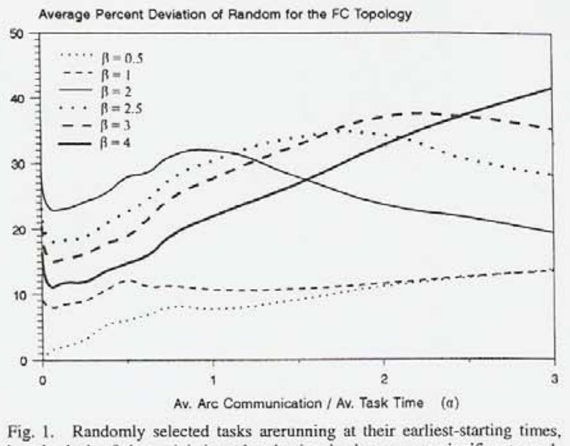

Fig. 1. Randomly selected tasks arerunning at their earliest-starting times, but the lack of deterministic task selection leads to waste significant parallelism.

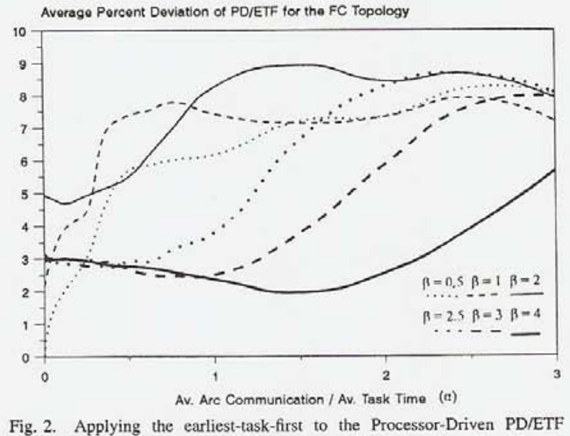

Fig. 2. Applying the earliest-task-first to the Processor-Driven PD/ETF approach leads to acceptable performance (5%) only when the inherent parallelism is sufficient to cover the communication($\beta/\alpha \geq 1.5$ for FC).

the definition of α and β, analysis of the data gives:

$$\frac{N_T}{N_L} \cdot \frac{\mu_T}{C_{\mathrm{arc}}} \geq \epsilon_{\mathrm{top}} \cdot p. \tag{3}$$

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. 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. Therefore, these heuristics require increasing the parallelism, or decreasing the number of processors, in order to achieve acceptable deviations as shown on Figs. 2–4.

C. 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 Figs. 5–7 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

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.

III. 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 $\mathrm{est}(T, p)$ 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:

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

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

$$\mathrm{est}(T,p) = \max_{T' \in D(T)} \{f(T') + c(T',T)\cdot r(p(T'), p(T))\}$$

where $f(T') = \mathrm{est}(T', p(T')) + \mu(T')$. In the event $D(T) = \emptyset, \mathrm{est}(T,p) = 0$ for every p. As the $\mathrm{est}(T, p)$ depends on the routing cost $r(p, p')$, then there exists a processor p^* for which task T can start at the earliest time $(\mathrm{est}(T))$ among all the available processors:

$$\mathrm{est}(T) = \mathrm{est}(T, p^*) = \min_p \{\mathrm{est}(T,p)\}. \tag{1}$$

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

$$\mathrm{est}(T,p^*) = \min_p \{\max\{\mathrm{est}(T,p), t(p)\}\}. \tag{2}$$

The earliest-completion-time $\mathrm{ect}(T)$, that is $\mathrm{est}(T) + \mu(T)$, 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. A heuristic approach to evaluate the task-level can be obtained by evaluating the earliest-completion-time $\mathrm{lct}(T)$ that results from backward scheduling the graph over the multiprocessor.

An algorithm called LST is proposed to evaluate the *the latest-completion-time* that defines the task-level in the proposed scheduling. LST operates as follows: 1) Obtain the dual graph (G') by reversing all arc direction in the original graph (G), 2) Schedule G' over system $S(P, R)$ such that the earliest startable task is scheduled first, 3) for every task T, the achieved $\mathrm{ect}(T)$ time becomes the task-level $l(T)$. According to this method, $l(T)$ represents an approximation of the shortest path from task T to any exist node of G that accounts for the precedence constraints, the communication (terms $c(T, T')$), and the network latency (term $r(p, p')$). Each task requires at most pn steps to evaluate its $\mathrm{est}(T, p)$ time and n tasks require $O(pn^2)$ that is the time complexity of algorithm LST.

IV. 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 LST, 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 involved in the updating process. This leads the PD scheduler to track the increasing sequence of processor completion times using a global time. For example, algorithm [4] 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 without global time. 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 how task-selection maps into the generalized task level and the incurred processor idle time, we define the following GLS heuristics. Heuristics GD/HLF is Graph-driven/Highest-level-first, i.e., highest $l(T)$ first. 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 $l(T) - \mathrm{est}(T)$, where $l(T)$ is the length of the shortest path from a starting T to any exit node as achieved by heuristic LST and $\mathrm{est}(T)$ is the effective earliest-starting-time. This approach leads to define heuristic GD/HLETF that is called Highest-level-earliest-task-first. According to the level function, the selected task T satisfies $l(T) - \mathrm{est}(T) \geq l(T_i) - \mathrm{est}(T_i)$ for any RTR task T_i. In other words, we have: $l(T) - l(T_i) \geq \mathrm{est}(T) - \mathrm{est}(T_i)$, i.e., to select task T the difference in levels between T and T_i should be higher than the amount of idle time $(\mathrm{est}(T) - \mathrm{est}(T_i))$ 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. The inputs to GD/HLETF are the task-graph and the list of $\mathrm{ect}(T)$ times that are generated by LST. This heuristic consists of selecting a task T^* and a processor p^* such that $l(T^*) - \mathrm{est}(T^*, p^*)$ is the highest among all the RTR tasks. Following the scheduling of T^* on p^*, the time at which p^* becomes free is $f(p^*) = \mathrm{est}(T^*, p^*) + \mu(T^*)$ is used to update the $\mathrm{est}(T, p^*)$ for all the RTR tasks, i.e. $\mathrm{est}(T, p^*) \leftarrow \max\{\mathrm{est}(T, p^*), f(p^*)\}$. The outputs are the starting time $\mathrm{est}(T)$ of each task and the processor $p(T)$ on which T is assigned. Algorithm GD/HLETF is the following:

```
(1) Initialize: $A \leftarrow \{T : D(T) = \emptyset\}, B \leftarrow \emptyset$, for each $T$ and
    each $p$: $\mathrm{est}(T,p) = 0, t(p) = 0$
(2) While $|B| < n$ Do
        Begin
        (2.1) Select $T^* \in A$ and $p^*$: $l(T^*) - \mathrm{est}(T^*, p^*)$
              $= \max_{T \in A}\{l(T) - \min_p\{\mathrm{est}(T,p)\}\}$
        (2.2) Assign $T^*$ on $p^*$: $p(T^*) = p^*, t(p^*) = \mu(T^*)$
              $+\mathrm{est}(T^*, p^*)$, remove $T^*$ from $A$, add $T^*$ to $B$,
              For each $T \in A$, update:
              $\mathrm{est}(T, p^*) = \max\{\mathrm{est}(T, p^*), t(p^*)\}$
        (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 $\mathrm{est}(T,p)$ for each $p$:
                    $\mathrm{est}(T,p) = \max\{\max_{T' \in D(T)}\{f(T')$
                          $+ c(T',T) \cdot r(p(T'), p)\}, t(p)\}$
        End
```

Similar analysis to that of LST shows that the time complexity of GD/HLETF is $O(pn^2)$. An optimization technique that can reduce

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.

III. 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 $\text{est}(T,p)$ 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:

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

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

$$\text{est}(T,p) = \max_{T' \in D(T)} \{f(T') + c(T',T)\cdot r(p(T'),p(T))\}$$

where $f(T') = \text{est}(T',p(T')) + \mu(T')$. In the event $D(T) = \emptyset, \text{est}(T,p) = 0$ for every p. As the $\text{est}(T,p)$ depends on the routing cost $r(p,p')$, then there exists a processor p^* for which task T can start at the earliest time $(\text{est}(T))$ among all the available processors:

$$\text{est}(T) = \text{est}(T,p^*) = \min_p\{\text{est}(T,p)\}. \tag{1}$$

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

$$\text{est}(T,p^*) = \min_p\{\max\{\text{est}(T,p),t(p)\}\}. \tag{2}$$

The earliest-completion-time $\text{ect}(T)$, that is $\text{est}(T) + \mu(T)$, 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. A heuristic approach to evaluate the task-level can be obtained by evaluating the earliest-completion-time $\text{lct}(T)$ that results from backward scheduling the graph over the multiprocessor.

An algorithm called LST is proposed to evaluate the *the latest-completion-time* that defines the task-level in the proposed scheduling. LST operates as follows: 1) Obtain the dual graph (G') by reversing all arc direction in the original graph (G), 2) Schedule G' over system $S(P,R)$ such that the earliest startable task is scheduled first, 3) for every task T, the achieved $\text{ect}(T)$ time becomes the task-level $l(T)$. According to this method, $l(T)$ represents an approximation of the shortest path from task T to any exist node of G that accounts for the precedence constraints, the communication (terms $c(T,T')$), and the network latency (term $r(p,p')$). Each task requires at most pn steps to evaluate its $\text{est}(T,p)$ time and n tasks require $O(pn^2)$ that is the time complexity of algorithm LST.

IV. 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 LST, 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 involved in the updating process. This leads the PD scheduler to track the increasing sequence of processor completion times using a global time. For example, algorithm [4] 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 without global time. 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 how task-selection maps into the generalized task level and the incurred processor idle time, we define the following GLS heuristics. Heuristics GD/HLF is Graph-driven/Highest-level-first, i.e., highest $l(T)$ first. 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 $l(T) - \text{est}(T)$, where $l(T)$ is the length of the shortest path from a starting T to any exit node as achieved by heuristic LST and $\text{est}(T)$ is the effective earliest-starting-time. This approach leads to define heuristic GD/HLETF that is called Highest-level-earliest-task-first. According to the level function, the selected task T satisfies $l(T) - \text{est}(T) \geq l(T_i) - \text{est}(T_i)$ for any RTR task T_i. In other words, we have: $l(T) - l(T_i) \geq \text{est}(T) - \text{est}(T_i)$, i.e., to select task T the difference in levels between T and T_i should be higher than the amount of idle time $(\text{est}(T) - \text{est}(T_i))$ 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. The inputs to GD/HLETF are the task-graph and the list of $\text{ect}(T)$ times that are generated by LST. This heuristic consists of selecting a task T^* and a processor p^* such that $l(T^*) - \text{est}(T^*,p^*)$ is the highest among all the RTR tasks. Following the scheduling of T^* on p^*, the time at which p^* becomes free is $f(p^*) = \text{est}(T^*,p^*) + \mu(T^*)$ is used to update the $\text{est}(T,p^*)$ for all the RTR tasks, i.e. $\text{est}(T,p^*) \leftarrow \max\{\text{est}(T,p^*), f(p^*)\}$. The outputs are the starting time $\text{est}(T)$ of each task and the processor $p(T)$ on which T is assigned. Algorithm GD/HLETF is the following:

```
(1) Initialize: $A \leftarrow \{T : D(T) = \emptyset\}, B \leftarrow \emptyset$, for each $T$ and
    each $p$: $\text{est}(T,p) = 0, t(p) = 0$
(2) While $|B| < n$ Do
      Begin
      (2.1) Select $T^* \in A$ and $p^*$: $l(T^*) - \text{est}(T^*,p^*)$
            $= \max_{T \in A}\{l(T) - \min_p\{\text{est}(T,p)\}\}$
      (2.2) Assign $T^*$ on $p^*$: $p(T^*) = p^*, t(p^*) = \mu(T^*)$
            $+\text{est}(T^*,p^*)$, remove $T^*$ from $A$, add $T^*$ to $B$,
            For each $T \in A$, update:
            $\text{est}(T,p^*) = \max\{\text{est}(T,p^*), t(p^*)\}$
      (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 $\text{est}(T,p)$ for each $p$:
                  $\text{est}(T,p) = \max\{\max_{T' \in D(T)}\{f(T')$
                        $+ c(T',T)\cdot r(p(T'),p)\}, t(p)\}$
      End
```

Similar analysis to that of LST shows that the time complexity of GD/HLETF is $O(pn^2)$. An optimization technique that can reduce

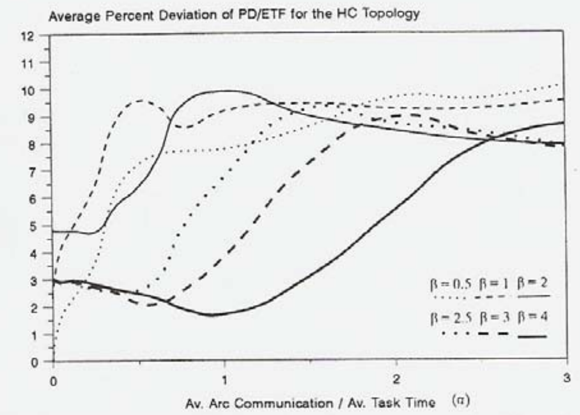

Fig. 3. Increasing the interprocessor communication penalty from FC to HC leads (PD/ETF) to deviation within 5% only when the ratio of parallelism to communication exceeds 2.3.

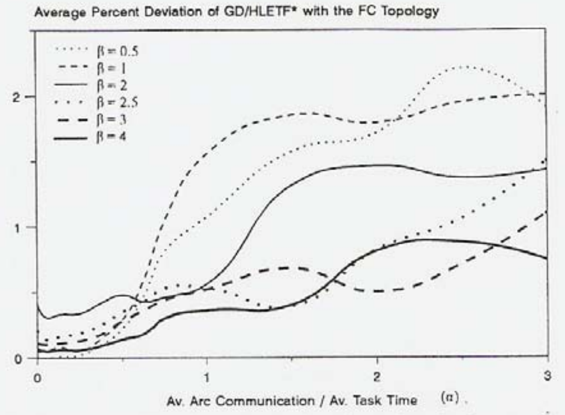

Fig. 5. Attempt filling idle times with less-prior ready tasks improves the performance of HLETF by 2% over HLETF because of better overlapping of computation and communication.

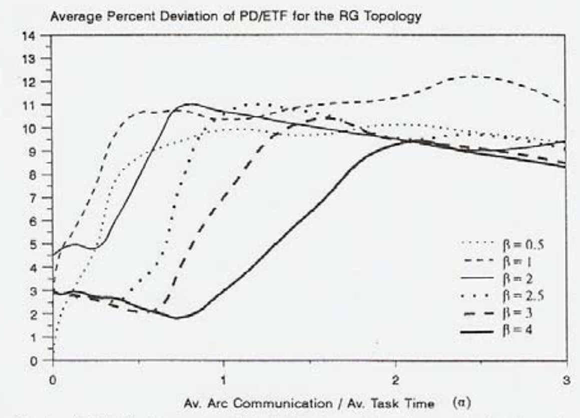

Fig. 4. PD/ETF gives acceptable finish time only when parallelism largely exceeds the average communication($\beta/\alpha \geq 4$) due to excessive interprocessor communication penalty of the ring.

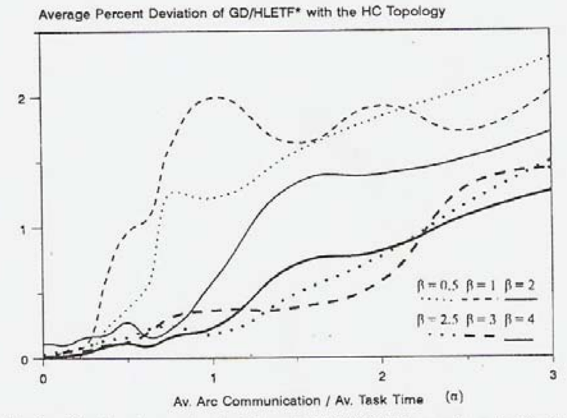

Fig. 6. The bound on the deviation of GD/HLETF is nearly maintained for the HC. Higher parallelism ($\beta \geq 1$) gives more opportunity to overlap computation with communication and results in shorter finish times.

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 approaches. 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.

D. 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. Figs. 9 and 10 show the boundary of the best 50% and 90% population of the finish time versus the available parallelism (β) for heuristics PD/ETF and GD/HLETF*. Each point of the boundary 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*.

E. Comparison to Other Contributions

The task priority used in [10] is too unnacurate because it incorporates all the communication carried by the edges and does not address the processor selection problem. Its time complexity is $0(n^3)$.

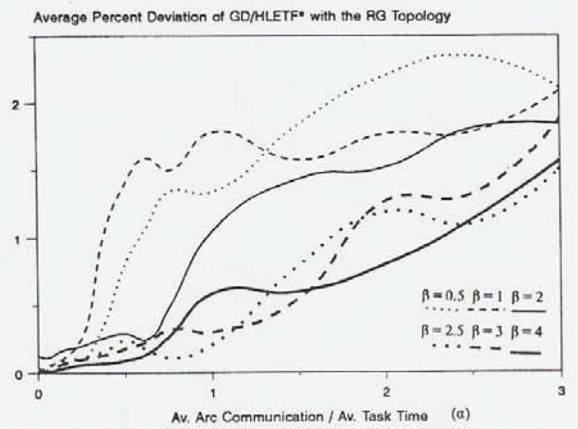

Fig. 7. The bound on the deviation of GD/HLETF is nearly maintained within 3% regardless of the level of parallelism, communication, and topology.

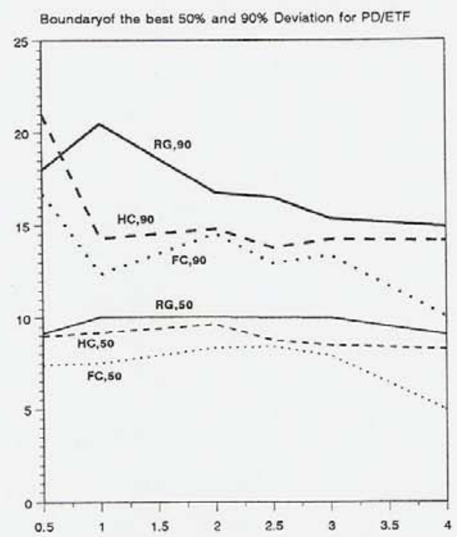

Fig. 8. The boundary indicates maximum deviation for all levels of communication (α) under each instance of parallelism (β) and topology.

Pase [7] experimentally studied 12 heuristics ($S_1 - S_{12}$) including the TD technique ($O(pn^2)$) [6] and ETF [4] (S_2). His heuristic (S_1) assigns priority from the graph bottom and select the task that is closest to top. He founds that S_1 and ETF are among the best heuristics and both outperform the TD scheduler of [6]. Our study indicates that the proposed GLS outperforms ETF (S_2) versus change in communication, parallelism, and network topology. Comparing time complexity, all these heuristics fundamentally have identical number of steps ($O(pn^2)$) but slightly differ in the constant.

VI. 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. The task-level is evaluated as the completion time of the tasks that results from backward scheduling the computation over the multiprocessor. 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.

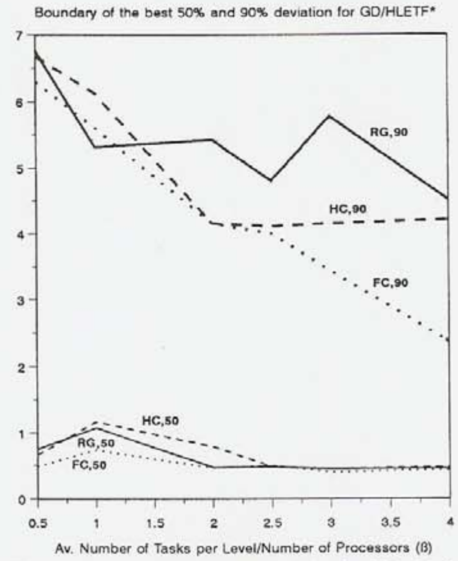

Fig. 9. The boundary indicates maximum deviation for all levels of communication (α) under each instance of parallelism (β) and topology.

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. 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 superiority of graph-driven over processor-driven has been shown. 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.

References

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Commun. ACM*, vol. 17, no. 12, pp. 685–690, Dec. 1974.
[2] M. Al-Mouhamed, "Analysis of macro-dataflow dynamic scheduling on non-uniform memory access architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 3, pp. 875–888, Nov. 1993.
[3] E. G. Coffman, Jr. *et al.*, *Computer and Job-Shop Scheduling Theory*. New York: John Willy, 1976.
[4] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C. Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM Computing*, pp. 244–257, Apr. 1988.
[5] S. J. Kim and J. C. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in *Proc. Int. Conf. Parallel Processing*, vol. 3, pp. 1–8, Aug. 1988.

[6] B. Kruatrachue, "Static task scheduling and grain packing in parallel processing systems," Ph.D. thesis, Dep. of Comput. Sci., Oregon State Univ., 1987.
[7] D. M. Pase, "A comparative analysis of static parallel schedulers where communication costs are significant," Ph.D. thesis, Oregon Graduate Inst. of Sci. and Technol., July 1989.
[8] V. Sarkar and J. Hennessy, "Compile-time partitioning and scheduling of parallel programs," in *Proc. SIGPLAN Symp. Compiler Construction*, July 1986, pp. 17-26.
[9] G. C. Sih and E. A. Lee, "Scheduling to account for interprocessor communication within interconnection constrained processing network," in *Int. Conf. Parallel Processing*, vol. 1, 1990, 9–16.
[10] M.-Y. Wu and D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 330–343, July 1990.