Scheduling Optimization Through Iterative Refinement

Mayez Al-Mouhamed and Adel Al-Massarani *

Abstract

Scheduling computations with communications is the theoretical basis for achieving efficient parallelism on distributed memory systems. We generalize Graham's task-level in a manner to incorporate the effects of computation and communication. A new scheduling is proposed by combining task priority with efficient management of processor idle time. We also propose an optimization called Iterative Refinement Scheduling (IRS) that iteratively schedules the *forward* and *backward* computation graph. The task-level used in some scheduling iteration is obtained from the schedule generated in the previous iteration. Each iteration produces a new schedule and new task-levels. This approach enables searching and optimizing solutions as the result of using more refined task-level in each scheduling iteration. Evaluation and analysis of the results are carried out for different instances of *communication granularities* and *problem parallelism*. It is shown that solutions obtained out of few iterations statistically outperforms those generated by other recently proposed scheduling. IRS allows exploring a space of solutions whose size grows with the amount of parallelism and communication granularity. IRS enables *optimizing* the solution specially for critical instances such as fine-grain computations and large parallelism.

Keywords: Distributed memories, heuristics, message-passing, performance, scheduling

1 Introduction

Automatically extracting parallelism out of large scale scientific computations is especially useful when the programs are repeatedly executed over different data sets. Deterministic scheduling can be profitable when the execution behavior is predictable at compile-time [13] such as the class of static dataflow computations. In this case, the compiler is capable of predicting the the precedence relationships and volume of transferred data between the various computation modules.

*Department of Computer Engineering, King Fahd University of Petroleum and Minerals, P.O. Box 787, Dhahran 31261, Saudi Arabia (Email: mayez/adel@ccse.kfupm.sa.edu)

We study scheduling precedence-constrained computations on an arbitrary number of processors that are regularly or irregularly interconnected. To incorporate the effects of communication, a model of communication latency is used to evaluate the cost of transferring data between processors. Generally, the scheduling problem is NP-complete [4] except for trees. Lower bounds [2] and worst case analysis [7, 2] have been proposed for scheduling precedence computations with and without communication costs. The objective is to find efficient non-preemptive scheduling approaches that combine knowledge on the computation structure and multiprocessor in order to minimize overall finish time. Different approaches have been used for scheduling computations with communication costs which can be classified into the following categories: 1) *Searching*, 2) *Clustering*, 3) *Task-Duplication*, and 4) *Priority-Based Scheduling*.

Search-based scheduling methods using branch-and-bound [16] and simulated annealing [18] have 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 partial scheduling problems are addressed [8, 17]. Recently, genetic algorithms [5] were also applied to scheduling without communication times.

Linear clustering [8] has been applied for iteratively merging tasks along the most communicating chains in an attempt to minimize the computation time. After multiple refinements the resulting graph is mapped onto the target multiprocessor using graph theoretic approach. *Clustering* over bounded number processors [17] consists of 1) partitioning the set of tasks into clusters of sequential tasks, and 2) reducing the number of clusters by merging operations until matching the number of processors. The *Dominant Sequence Clustering* (DSC) [22] was proposed to enhance the work reported by Sarkar and Hennessy [17]. DSC is a low complexity clustering that accepts merging task T to a cluster if the distance from some entry node to T decreases as well as the current length of the dominant chain to which T belongs. The time complexities compare as $O(e(e+n))$ [17] and $O(\log n(e+n))$ [22], where n and e are the numbers of tasks and communication edges, respectively.

Another scheduling for unbounded number processors is *Dynamic Critical Path* (DCP) that was proposed by Kwok and Ahmad [11]. The DCP is a chain of immediate tasks having zero mobility. The mobility is found by using an *absolute earliest starting time* and an *absolute latest starting time* for each task. The selected DCP task T is one that has all its DCP predecessors already assigned. Only processors holding the predecessors of T are examined. For each such a processor, the algorithm searches a vacant slot to fit T with the possibility of starting earlier or delaying previously assigned tasks whithin the limit of their mobilities. Processor selection is based on looking for the potential start times of remaining tasks on each processor. This condition guarantees some processor reservation for the most critical successor. Finally, all the tasks pulled at their earliest times values. The complexity of DCP scheduling is $O(n^3)$.

Wu and Gajsky [21] proposed two algorithms that are the *modified critical path* (MCP) and *mobility directed* (MD). These algorithms use the *as-soon-as-possible* starting time ($t_s(T)$) that is the length of the longest path including computation and communication

from entry node to T. The *as-late-as-possible* completion time ($t_l(T)$) is the difference between the longest path in the graph and the length of the longest path from T to some to exit node. MCP selects the free task with the largest ($t_l(T)$) and assigns it to the processor that can start it at the earliest among all processors. MD selects the free task T with least *relative mobility* ($rm(T) = (t_l(T) - t_s(T))/\mu(T)$) and assigns it to a processor that does not increase the current critical path, i.e. none of the scheduled tasks is delayed beyond its latest time. MD uses an unbounded number of processors and its complexity is $O(pn(n+e))$.

By assuming that delays in communication are mainly due to channel latency, scheduling based on *Task Duplication* (TD) over idle processors was proposed [10] to reduce the communication. Beside the time complexity of this method ($O(n^4)$) [10] the management of duplicated data messages is another drawback. The impacts of task and processor selection as well as the profitability of task duplication were experimentally studied by Pase [15] which used Hu's task-priority [6] for these selections. The degree of task duplication can also be adjusted [12] depending on the number of available processors or their difference in speeds.

Priority-based scheduling over bounded number of processors basically uses a criterion for *task and processor selection* as the main strategy to minimize overall finish time. The basis for task selection is Graham's task-level $l(T)$ that measures the remaining computation beyond this task to finish overall computation. The task-level depends only on the task graph in the case of zero communications, i.e. longest path from a task to exit node. For non-zero communication costs the task-level depends on the mapping of tasks to processors and the implied communication cost. The way highly communicating tasks are assigned to processors may significantly affect the length of paths. In other words, the knowledge of the computation graph is not sufficient to distinguish between critical and non-critical tasks. Discarding the communication and network latency in evaluating the task-level [10, 15], or pessimistically accounting for all of them [21] leads to excessively inaccurate evaluations. To avoid these problems, only local scheduling heuristics have been proposed such as *earliest-task-first* (ETF) [7], *Largest-Communication-First* (LCF) [3], *Dynamic Level Scheduling* (DLS) [20], bounded number of processors.

In ETF task and processor selection are based on finding the earliest startable task and its best suited processor. Therefore, the main stategy of this algorithm is the local minimization of processor idle times through exploiting the offered opportunity of overlapping computation with communication. However, ETF local decision has shown to cause some performance degradation especially when is there is little opprotunity to exploit simultaneity between compuation and communication. LCF searches a task and a processor such that the largest amount of communication are supressed, but if no saving on communication is present then the task that has been ready at the earliest time is selected. In DLS, the largest sum of computations from task T to exit is considered as *static task level* $SL(T)$. The ready task that has the highest *dynamic level* $DL(T,p) = SL(T) - ST(T,p)$ is selected first, where $ST(T,p)$ denotes the earliest time at which all incomming data transfer complete for T. This requires evaluation of $ST(T,p)$ for all ready tasks and all processors. Basically, the complexity of ETF, LCF, DLS algorithms is $O(pn^2)$.

Our objective is to generalize the notion of task-level in a manner to incorporate the effects of computation, volume of transferred data, and network latency. Graham's scheduling is based on fetching ready tasks by idle processors. This concept has been widely used for scheduling computations with or without communications. Graham's scheduling leads to uniform processor scheduling and inefficiently utilizes the task-level in presence of communications. We propose a new approach called *Computation-driven scheduling* which combined with our generalized task-level enables early reservation of resources to critical computations and communications. We further extend our approach by iteratively refining the generalized task-level while exploring a space of "good" solutions. Analysis of the proposed task-level, computation-driven scheduling, and iterative refinement is presented. We carry out extensive experimental evaluation for different instances of problem granularities, inherent parallelism, and network latency.

This paper is organized as follows. Section 2 reviews Graham's scheduling concept. Derivation of the generalized task-level is presented in Section 3. The computation-driven approach is developed in Section 4. Section 5 presents the iterative refinement scheduling and analysis of the proposed approach is presented in Section 5. Evaluation of two iterations scheduling is presented in Section 6. Evaluation of the iterative refinement scheduling and comparison to other approaches are presented in Section 7. Section 8 concludes and outlines possible future extension of this work.

2 Background

A fundamental result of scheduling theory is the introduction of list-scheduling (LS) and its performance bound [4]. The objective function is to minimize the execution time of computations that can be represented by using precedence-constrained graphs with no communication times. One important aspect of LS is the use of the longest sum of computation from starting a task T to any exit node [6] as a measure of task criticality that is called the *task-level* ($l(T)$). After evaluating the task-levels, LS sorts the tasks in the decreasing order of their levels and store them in a list L. Next, each idle processor p scans L from the beginning searching for a ready to run task. If p finds a ready task T, then T is started on p and run without preemption. Otherwise, p does not start the above process until completion of some currently running task which causes other tasks to become ready.

The key point to this algorithm is the way the scheduling process is controlled and the selection function (multi-objective) which accounts for the *earliest startable task* as well as *task priority*. Graham's list-scheduling implicitly enforces the starting times of successively scheduled tasks be non-decreasing sequence in time. This in turn enables finding the well known Graham's worst case bound [4]. The finish time ω_{LS} over m processors always satisfies $\omega_c \leq \omega_{LS} \leq (2-1/p)\omega_{opt}$, where ω_c is the highest task-level or logest path and ω_{opt} is the optimum solution. Note that Graham's task-level is independent from mapping tasks to processors because of zero communication.

LS generates optimum solutions for tree-computations with equal task times. Ex-

perimental evaluation of LS [1] proved that as shown that LS generates near-optimum solutions for both deterministic and stochastic computations as it deviates from optimum by less than 5% in 90% of the cases. Unfortunately, List-scheduling does not apply [7] to scheduling of precedence-constrained computations with communications that are targeted to distributed memory systems, which is the major issue in this work.

Our objective is to investigate near-optimum scheduling approaches for problems where the volume of transferred data is one important aspect as well as the communication delays in the underlaying distributed memory system. For this, we generalize Graham's task-level, investigate more efficient methods for controlling the scheduling process, and propose an optimization approach for refining the solution.

3 Precedence computations with communications

For a precedence-constrained computation with communications, the evaluation of the shortest distance from starting a task T to any exit task depends on the mapping of the tasks to processors because the communication delays between the processors are not identical. In the following we analyze the evaluation of the *earliest starting time* of the tasks, show its dependency on the task-processor mapping problem, and discuss our approach for heuristically defining the task-level.

A set of $\Gamma(T_1, \ldots, T_n)$ of n tasks (T) with their precedence constraints and communication costs are to be non-preemptively scheduled on p identical processors so that their overall execution time is held to a minimum. The computation can be modeled [7] by using a directed acyclic task graph $G(\Gamma, \rightarrow, \mu, C)$ where $\rightarrow$, $\mu(T_i)$, and $c(T_i, T_j) \in C$ denote the precedence constraints, the task execution time, and the number of communication messages (volume of data) that are sent from task T_i to its immediate successor T_j, 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 one unit of messages (datum) from a processor p_i to processor p_j, through the interconnection network, is denoted by $r(p_i, p_j)$, where p_i and p_j are the processors that are assigned tasks T_i and T_j, respectively. The communication model is based on a linear communication latency model. Assuming that the communication media is contention-free, the time to transfer $c(T_i, T_j)$ messages is $c(T_i, T_j)r(p_i, p_j)$.

Consider the hypercube network and let $h(p_i, p_j)$ be the number of hops the packet holding the volume of data $c(T_i, T_j)$ should travel on between processors p_i and p_j. When the communication media is contention free, the time to perform the transfer can be expressed as follows:

$$c(T_i, T_j)r(p_i, p_j) = t_s + c(T_i, T_j) \times t_{hop} \times h(p_i, p_j) \tag{1}$$

where t_s and t_{hop} are the packet setup time and the time to transfer one unit of data over one hop. This approach allows incorporating a model of data transfer delays in evaluating the earliest-starting-time of the tasks.

Let T_i be a task and denote by $Pred(T_i)$ the set of predecessors of T_i. By considering only one predecessor task, the earliest-starting-time $est(T_i, p_i)$ of T_i for some processor p_i depends on 1) the completion time $ct(T_k, p_k)$ of predecessor T_k on processors p_k, 2) the number of messages $c(T_k, T_i)$ sent from T_k to T_i, and 3) the cost of routing one unit of messages from p_k to p_i.

In other words, $est(T_i, p_i)$ is defined by the relationship:

$$est(T_i, p_i) = ct(T_k, p_k) + \begin{cases} 0 & \text{if } p_i = p_k \\ c(T_k, T_i) \times r(p_k, p_i) & \text{otherwise} \end{cases} \tag{2}$$

The $est(T_i, p_i)$ will be null for every processor p_i in the case T_i has no predecessors ($Pred(T_i) = \emptyset$). By considering all the predecessors of T_i, $est(T_i, p_i)$ is the earliest time the latest message from the predecessors reaches processor p_i:

$$est(T_i, p_i) = \max_{T_k \in Pred(T_i)} \{ct(T_k, p_k) + c(T_k, T_i) \times r(p_k, p_i)\} \tag{3}$$

As $est(T_i, p_i)$ depends on the routing cost $r(p_k, p_i)$, then there exists at least one processor p^*, that is free at time $t(p^*)$, for which T_i can start at the earliest time $est(T_i, p^*)$ among all the available processors:

$$est(T_i, p^*) = min_{p_i}\{\max\{est(T_i, p_i), t(p_i)\}\} \tag{4}$$

Function $est(T_i, p_i)$ incorporates the effects of computation and communication along some chain of predecessors of T up to entry node in addition to the cost of transferring messages between these predecessors. Moreover, $est(T_i, p_i)$ explicitly depends on *how tasks are mapped to processors* as well as the implied network latency.

Denote by p_{max} the maximum number of tasks that can be made ready to run at any given time for a given computation. In general, selecting p_{max} processors with a given interconnection network for evaluating the earliest starting time does not guarantee that $est(T_i, p_i)$ represent the shortest time distance from some entry task to starting T_i. This is due to the dependence of $est(T_i, p_i)$ over the processor selection which in turn affects the cost of transferring data between immediate tasks. Even if we choose p_{max} processors in $S(P, R)$ with arbitrary interconnection network, evaluating $est(T_i, p_i)$ cannot be achievable in all cases because two tasks may simultaneously compete for the same processor in order to achieve the lowest earliest-starting-times. For the model $G(\Gamma, \rightarrow, \mu, C)$, the task-level strongly depends on the routing costs between the processors. Therefore, the evaluation of the task-level is not tractable using $G(\Gamma, \rightarrow, \mu, C)$ and multiprocessor $S(P, R)$.

Note that using Graham's model $G(\Gamma, \rightarrow, \mu)$ of the computation the task-level is independent from mapping tasks to processors, due to zero communications, which enables evaluation of the task-levels using only knowledge of the computation model.

To avoid the difficulty in the evaluation of the task-level as a quantifier of task criticality, only local scheduling heuristics have been proposed such the *earliest-task-first* [7] and the *largest-communication-first* [3] for the computation model $G(\Gamma, \rightarrow, \mu, C)$. These

heuristics are based on locally minimizing the processor idle time as a strategy toward minimizing overall finish time. On the other hand, discarding all the communication aspects [19, 15, 10], or accounting for all of them [21], in evaluating the task-level leads to excessively inaccurate evaluations that fail specifying the degree of criticality of the tasks with respect to overall computation.

Our approach is based on a *heuristic estimate of the task-level* and *refining* the estimate through an iterative scheduling process. According to this method, a primary estimate of the task-level is obtained by scheduling the reverse graph (G_r) over multiprocessor $S = S(P, R)$ by using the principle of earliest-task-first. The reverse graph G_r is identical to the original computation graph $G = G(\Gamma, \rightarrow, \mu, C)$ except that all edge directions are reversed. Note that scheduling G_r starts with the exit tasks of G and propagates up to entry tasks. Following the scheduling of G_r, the primary task-level $l(T)$ is set to the achieved completion time ($ct(T)$), i.e., $l(T) = ct(T)$ for each task T. For the original graph G, the above $l(T)$ represents an approximation of the remaining computation and communication along a directed chain $X : (T \rightarrow \ldots \rightarrow T_n)$ of tasks whose first task is T and last task T_n is any exit node of G. Clearly, $l(T)$ provides an estimate of the shortest distance from starting T to arbitrary exit node with respect to G. Note that $l(T)$ now incorporates the effects of the computation, communication, and network latency along the chain X because $l(T) = ct(T)$ results from scheduling G_r over S. The next step is to use the above task-level $l(T)$ according to the principle of *highest-level-first* in scheduling the original computation graph G over system S in an attempt to minimize overall finish time.

Based on heuristic evaluation, our approach has the advantage of incorporating the precedence constraints, communication aspect, and network latency factors in evaluating the task-levels. The important point is that the estimate of the task-level can easily be improved through an *iterative refinement process* that will be described later in this paper.

In the next section, we discuss the traditional approach for controlling the scheduling process and propose a new method, called *Computation-Driven*, which combined with our heuristic evaluation of the task-level will prove to be efficient and simple compared to existing scheduling approaches.

4 The computation-driven scheduling

In this section we analyze Graham's list scheduling and specially the way the scheduling process is controlled. Graham [4] accurately defined his approach to controlling and sequencing the List-Scheduling that will be referred to as *Processor-Driven*.

The *processor-driven* approach allows controlling the scheduling so that the starting times of successively scheduled tasks form a non-decreasing sequence in time. This approach was first proposed by Graham [4] (Section 2) and later was used in many other scheduling applications [7, 15, 19, 21] for computations either with or without communication times. When at least one processor completes execution of some task T, the successors of T are examined in order to find out whether any of them becomes ready-

to-run in which case such a successor is added to the current set of ready-to-run tasks. Clearly, only 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*. Fundamentally, this approach leads to uniform scheduling of the processors because the starting times of successively scheduled tasks, for all the processors, form a non-decreasing sequence in time. Example of well known PD scheduling are the list-scheduling and the ETF heuristic [7] that is based on the principle of *earliest-task-first*.

We propose a new approach called *Computation-Driven* (CD) to control the scheduling process as opposed to the *processor-driven* that was proposed within the framework of list-scheduling and more recently used in many other scheduling. To emphasize the notion of critical tasks, the CD uses a decision function $d(T) = F(l(T), \ldots)$ that is increasing function (F) of the task-level $l(T)$ which has been defined in Section 3. In general, function $d(T)$ is to be dynamically evaluated. The CD approach mainly consists of the following steps:

1. Initialize: evaluate the task-levels $l(T)$
2. Until there are no unscheduled tasks
    - The task T with the highest $d(T)$ is assigned to some processor p that can run T at the earliest time. The earliest time p becomes free is set to the earliest completion of T on p
    - At the starting of T on p, the set of ready-to-run tasks is immediately updated to include any successor of T whose predecessors have all been assigned to some processors but have not necessarily been completed

The important features of CD compared to PD are 1) the absence of global time, 2) the set of ready-to-run tasks is immediately updated following the starting of each task, 3) the starting times of successively scheduled tasks are not necessarily a non-decreasing sequence in time. While PD rather uniformly schedules the processors, the CD is non-uniform with respect to processor selection because the tasks of a critical chain can be sequentially scheduled on one or few processors. The PD scheduling is driven by free processors but CD scheduling is driven by the computation graph and more precisely by some chain of critical tasks that are selected based on their task-levels. On the other hand, the explicit choice of function F is necessarily driven by the need to implement different minimization strategies depending on how the task-level should be combined with other information. The formal characteristics of the CD approach will be analyzed in Section 5.1 and evaluated in Section 6.

Depending on how task-selection maps into the generalized task-level and the incurred processor idle time, we define the following CD scheduling heuristics. Heuristic CD/HLF is computation-driven/Highest-level-first for which the decision function is defined by $d(T) = l(T)$.

Selecting tasks according to an HLF criterion may lead to increasing the processor idle time that precedes the starting of the highest level task. Therefore, a heuristic function that imposes a penalty due to the idle time which precedes its starting time consists of a decision function $d(T) = l(T) - est(T,p)$, where $est(T,p)$ is the least starting time of T that can be achieved on some processor p. This approach leads to define a computation driven scheduling heuristic called *Highest-Level-Earliest-Task-First* (CD/HLETF). In the following we analyze the selection function of heuristic CD/HLETF.

Consider task T that is scheduled first by CD/HLETF whenever T satisfies $l(T) - est(T,p) \geq l(T') - est(T',p')$ for any ready task T', where p and p' are the processors on which T and T' can run at the earliest time. If $p \neq p'$, then the decision order of T and T' are independent and non-conflicting. In this case, assigning T on p at time $est(T,p)$ does not block T' from being assigned later to start at time $est(T',p')$.

Assume $p = p'$ and consider the following two cases: 1) $est(T,p) \leq est(T',p')$, and 2) $est(T,p) > est(T',p')$.

Assume $est(T,p) \leq est(T',p')$. The task-level can be written as $l(T) \geq l(T') - (est(T',p') - est(T,p)) = l(T') - \Delta_{idle}$, where Δ_{idle} is the idle time that precedes T' if T' is selected first. Since, $l(T) \geq l(T') - \Delta_{idle}$ indicates that CD/HLETF decreases the level (penalty) of T' because it incurs an idle time Δ_{idle} if is was selected first.

Assume $est(T,p) > est(T',p')$. We similarly obtain $l(T) \geq l(T') + \Delta_{idle}$, where $\Delta_{idle} = est(T,p) - est(T',p')$. This is equivalent to penalizing task T as the level of T' is increased by Δ_{idle} because the starting of T incurs an idle time Δ_{idle}. CD/HLETF imposes a penalty on the task-level (increasing or decreasing) that is the amount of relative idle time which precedes the starting of every task.

We now present algorithm (CD/HLETF) which is shown in Figure 1. CD/HLETF uses task-levels $\{l(T)\}$ as inputs and set A and B to store ready-to-run tasks and assigned tasks, respectively. Initially, A contains all tasks without predecessors and B is empty. $t(p)$ holds the earliest time p is idle. The main loop (Statement 2) repeats until there are no unscheduled tasks. It selects a task T^* and a processor p^* such that $l(T^*) - est(T^*,p^*)$ is the highest among all ready tasks for which the task's least starting time ($least\_est(T)$) is for processor p^*. Following the scheduling of T^* on p^*, the time at which p^* becomes free ($t(p^*) = est(T^*,p^*) + \mu(T^*)$) is used to update the $est(T,p)$ for all tasks of A. It also finds new $least\_est(T)$ if the current $least\_est(T)$ is modified.

For each task T, an integer $\lambda(T)$ is initially set to the number of predecessors of T. Following the scheduling of T, $\lambda(T')$ is decremented for each successor $T' \in Succ(T)$. If $\lambda(T') = 0$, then all the predecessors of T' have already been assigned and, consequently, T' becomes ready according to the CD approach. T' can start after all precedence and communication are satisfied but its inclusion in A enables it to compete early with others and possibly get scheduled (reservation). Notice that finish time of successively scheduled tasks is no more a non-decreasing sequence in time. The outputs of CD/HLETF are the completion times $\{ct(T)\}$ of the tasks and their processor assignment $\{p(T)\}$.

The main loop of CD/HLETF is statement 2 that executes n times because one task is scheduled in each run. Statement 2.1 executes at most pn times in order to select one task. Statement 2.3 updates the parameters but its last operation executes n times. Operation

Algorithm: CD/HLETF

(1) Initialize: $A \leftarrow \{T : Pred(T) = \emptyset\}$, $B \leftarrow \emptyset$,
For each $T \in A$ and each p: $est(T,p) = 0$, $least\_est(T) = 0$, $t(p) = 0$.

(2) While $|B| < n$ Do
Begin
(2.1) Select $T^* \in A$ and p^* such that $least\_est(T^*) = est(T^*, p^*)$ and
$l(T^*) - est(T^*, p^*) = \max_{T \in A}\{l(T) - \min_p\{est(T,p)\}\}$.

(2.2) Assign T^* on p^* : $p(T^*) = p^*$, $ct(T^*) = est(T^*, p^*) + \mu(T^*)$, $t(p^*) = ct(T^*)$,
Remove T^* from A, Add T^* to B,
For each $T \in A$, update: $est(T, p^*) = max\{est(T, p^*), t(p^*)\}$,
Find new $least\_est(T)$.

(2.3) Repeat for each task $T \in Succ(T^*)$: $\lambda_d(T) = \lambda_d(T) - 1$,
If $\lambda_d(T) = 0$ Then Add T to A,
For each p: $est(T,p) = \max\{\max_{T' \in Pred(T)}\{ct(T') + c(T', T).r(p(T'), p)\}, t(p)\}$.
Find $least\_est(T)$.
End

Figure 1: Scheduling algorithm CD/HLETF

$\lambda(T) = \lambda(T) - 1$ in statement 2.3 executes $O(n^2)$ times but the condition $\lambda(T) = 0$ occurs only once for each task. The time complexity of CD/HLETF is then $O(pn^2)$.

An optimization technique that can improve management processor idle time is to attempt filling the idle time that precedes the the most prior task T by another task T' provided that T does not get delayed. If such T' can be found, then updating A by eventually adding some successor T'' of T' always satisfies $d(T) > T''$, thus processor reservation is maintained for T. This approach defines heuristic CD/HLETF * which only increases the constant in pn^2.

5 Iterative refinements

A *computation-driven* scheduling heuristic (H_{cd}) uses a decision function $d(T) = F(l(T), state)$, where $l(T)$ is the task-level, F is some increasing function of $l(T)$, and *state* is some dynamic function of T and the status of the processors at the time task T is considered for possible scheduling. Clearly, $l(T)$ results from some global knowledge of the computation in order to state how critical T is compared to other competing tasks. The decision function represents some *Generalized Task-level* (GTL) because it combines an approximation of the task-level with some penalty function. An example of CD scheduling is a heuristic that selects tasks according to the principle of highest $d(T) = l(T) - est(T,p)$ among all

ready to run tasks. In this case, $est(T, p)$ represents an approximation of the shortest time distance from some entry node to T.

The principle of CD scheduling consists of using the completion time of the tasks that results from scheduling the reverse graph G_r as the quantifier of task criticality for scheduling G on the basis of the principle of highest-level-first. The completion times so utilized are only approximate but they incorporate the most critical effects along an arbitrary chain of tasks that are the computations, communications, and network latency. We expect this strategy to yield solutions that statistically minimize the finish time. Therefore, we propose extending this process by *alternatively scheduling G_r and G over system S* and passing the completion time $ct(T)$ of the tasks from one CD scheduling iteration to the next in order to search in a space of solutions. We call this approach *Iterative Refinement Scheduling* (IRS). The important point of IRS is that the task-levels used in some scheduling iteration are the task completion times that result from the very previous scheduling iteration. For this, each scheduling iteration passes its output $\{ct(T)\}$ to the next iteration for use as task-levels. The objective of this iterative process is to search solutions with shorter finish times as a result of using a more refined estimate of the task-level throughout successive refinements.

The iterative refinement scheduling is shown on Figure 2. A scheduling iteration, denoted by $H_{cd}(G, S, L) = \{ct(T)\}$, consists of scheduling graph G over system S by using heuristic H_{cd} and a list L of task-levels so that the resulting schedule is defined by the task completion times ($ct(T)$) and their processor assignments. The processor assignment is assumed but intentionally omitted from this notation. The iterative process is initialized by using a CD heuristic H_{CD} and scheduling of G_r according to the principle of earliest-task-first, i.e., highest-level-first with a decision function $d(T) = -est(T, p)$. The above scheduling can be represented by the notation $H_{cd}(G_r, S, -) = \{ct_1(T)\}$, where $ct_1(T)$ is the completion time of T following the first iteration. Next, the set of task completion times $\{ct_1(T)\}$ are used as task-levels in the second iteration. The second step is defined by $H_{cd}(G, S, \{ct_1(T)\}) = \{ct_2(T)\}$, where $\{ct_2(T)\}$ are the task completion times as achieved in the second iteration. Generally, odd numbered iterations perform scheduling of the reverse graph G_r and even numbered iterations operate on the original graph G. Formally, iterations $2k-1$ and $2k$ are defined by the successive scheduling ($k \geq 1$):

$$2k-1:\ H_{cd}(G_r, S, \{ct_{2k-2}(T, p)\}) = \{ct_{2k-1}(T, p)\}$$

$$2k:\ H_{cd}(G, S, \{ct_{2k-1}(T, p)\}) = \{ct_{2k}(T, p)\}$$

The iterative refinement is repeated until some condition is met. In the following we present an Iterative-Refinement-Scheduling algorithm which terminates if the finish time of the solution converges to some value, oscillates, or the number of iterations exceeds some bound. The iteration counter is denoted by k and ω_k denotes the finish time of the current solution. The output is the solution with the least finish time (L_{sol}).

Using one single CD heuristic, the IRS scheduling allows searching into a space of solutions. The assignment of a task T to processor p affects the solution in different

Input: Precedence-Graph $G = G(\Gamma, \rightarrow, \mu, C)$, its reverse Graph G_r, $\Gamma = \{T_1, \ldots, T_n\}$, multiprocessor model $S = S(P, R)$, and number of iterations N

Output: List $L = \{(ct(T), p(T)) : T \in \Gamma\}$ for the best solution

Algorithm : Iterative Refinement for Heuristic H_{cd}

(1) Initialize: set $ct_0(T) = 0$ for all $T \in \Gamma$, use temporary storage $G_t = G_r$,
set $\omega = \infty$ and counter $k = 1$
(2) Until (ω_k converges) or (ω_k oscillates) or ($k > N$) do
Begin
(2.1) Schedule G_t using H_{cd} : $H_{cd}(G_t, S, \{ct_{k-1}(T)\}) = \{(ct_k(T), p_k(T)) : T \in \Gamma\}$
(2.2) set $\omega_k = \max_{T \in \Gamma}\{ct_k(T)\}$, $k \leftarrow k + 1$
if $\omega_k < \omega$ then $\omega = \omega_k$, $L \leftarrow \{(ct_k(T), p_k(T))\}$, endif
if $even(k) = true$ then $G_t = G$
else $G_t = G_r$, endif
End

Figure 2: Iterative Refinement Scheduling

manner, mainly the volume of data to be transferred from the predecessors and the network latency in routing the data to p. Therefore, the size of the searching space and the number of iterations required to reach some steady state are necessarily dependent on the problem granularity (ratio of communication to computation) and the network latency. The size of the search space and the iterative process behavior will be discussed in section 7.1.

5.1 Analysis of the iterative refinement

The PD approach was initially introduced by Graham [4] and later extended [7, 15] to precedence-constrained computation with communication times. Initially at time zero, each processor p scans the list of tasks searching for a task T that can start without delay. If T is found, then p starts T, otherwise it idles until another processor completes execution of some other task.

This strategy was designed with a worst-case bound [4] in mind for problems with zero-communication. The PD approach and the bound were later generalized to computations with non-zero communication. It is proved [7] that the finish time $\omega_{PD/ETF}$ satisfies: $\omega_{PD/ETF} \leq (2 - 1/m)\omega_{opt} + C$, where m is the number of processors, ω_{opt} is the optimum solution without communication costs, and C is the sum of communication along some chain of tasks that finish at $\omega_{PD/ETF}$. The bound results from the fact that *the starting times of successively scheduled tasks is a non-decreasing sequence in time.* In the following

we prove that a schedule generated by the computation-driven approach does not satisfies the above stated bound.

Theorem 1 *The Computation-Driven scheduling does not satisfy the worst case bound of processor-driven/earliest-task-first.*

Proof The set of time points in $(0, \omega_{cd})$ can be partitioned into two subsets A and B that consist of all the time points for which: all processors are busy (A), and at least one processor is idle (B). B is the disjoint union of q open intervals $B = \cup_{1 \leq i \leq q}(b_{l_i}, b_{r_i})$ and $b_{l_1} < b_{r_1} < \ldots < b_{l_i} < b_{r_i} < \ldots < b_{l_q} < b_{r_q}$. We prove that is impossible in general to find a chain of tasks $X : T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_k$ such that T_k completes at ω_{cd} and chain X covers B. In other terms $\Sigma_{T \in X}\mu(T) + \Sigma_{T,T' \in X}c(T', T)r_{max}$ cannot always covers $\Sigma_{1 \leq i \leq q}(b_{r_i} - b_{l_i})$, where $r_{max} = \max_{p,p'}\{r(p, p')\}$.

Consider a task $T \in X$ such that its starting time $s(T) \in B$. By definition of B, there exists a processor p_ϵ that is idle in interval $I = (s(T) - \epsilon, s(T))$, where ϵ is some positive number. Denote by T' a task that starts on p_ϵ at $s(T') \geq s(T)$. Consider the case where the latest message for T reaches processor p_ϵ prior to time $s(T)$. In this case, we have $est(T, p_\epsilon) \leq s(T)$. The reason for which T was not started earlier (at time $est(T, p_\epsilon)$) on p_ϵ could be $d(T') \geq d(T)$ and T' was scheduled on p_ϵ prior to scheduling T on p even with $est(T, p_\epsilon) \leq est(T', p_\epsilon)$. CD scheduling may leave interval $(est(T, p_\epsilon), s(T))$ uncovered by the data transfer from the predecessors of T. Therefore, interval I cannot always be covered by $\mu(T)$ and $c(T', T)r_{max}$. ∎

For the processor-driven approach, the criterion behind the bound is to locally minimize the processor idle time as the main strategy toward minimizing overall finish time. The bound is useful notion but one important question is whether a heuristic that satisfies the bound is capable of generating near-optimum solutions under critical problem instances such as fine-grain or non-uniform network latency, or both.

By abandoning the strict enforcement of PD scheduling, the *computation-driven approach* applies a balanced decision function with respect to task criticality (task-level) and the incurred processor idle time. Tasks with higher levels are scheduled in-sequence along immediate chain of tasks until the priority is reversed to other chains which leads to out-of-sequence scheduling. We call this process the *effect of processor reservation* which will be clarified by the following theorems.

Theorem 2 *The decision function of CD scheduling is a decreasing function along any directed path* $(T_1 \rightarrow T_2 \rightarrow \ldots T_L)$ *in* G *or* G_r.

Proof Consider two immediate tasks T and $T' \in Pred(T)$ in G at iteration k. Let p_k and p'_k be the processors that are assigned tasks T and T' at iteration k, respectively. We need to prove that $ct_{k-1}(T') - est_k(T', p'_k) > ct_{k-1}(T) - est_k(T, p_k)$. As $T' \in Pred(T)$ and $\mu(T) \neq 0$, therefore, we have $est_k(T, p_k) > est_k(T', p'_k)$ for any immediate tasks of iteration k. Now consider the previous iteration $k - 1$ for which T is a predecessor of

T' ($T \in Pred(T')$ in G_r). In this case, $ct_{k-1}(T') > ct_{k-1}(T)$ which can be combined with $est_k(T, p_k) > est_k(T', p'_k)$ to give $ct_{k-1}(T') - est_k(T', p'_k) > ct_{k-1}(T) - est_k(T, p_k)$ for arbitrary k.

By definition, the CD decision function $d_k(T)$ is increasing function of $ct_{k-1}(T)$ at iteration k. Therefore, $d_k(T)$ decreases long any chain $(T_1 \rightarrow T_2 \rightarrow \ldots T_L)$ of immediate tasks where T_1 is an entry task and T_L is an exit task in G. The same proof applies to chains of G_r. ∎

The computation-driven approach schedules a task at each decision step. Denote by $do(T)$ the decision order of T that satisfies: $1 \leq do(T) \leq n$ for n tasks.

Theorem 3 *The decision function $d(T)$ is non-increasing function along any increasing sequence of decision orders.*

Proof We need to prove that the decision function $d(T)$ always satisfies $d(T) \geq d(T')$ whenever $do(T) < do(T')$. Assume T is scheduled at some decision order $do(T)$, we necessarily have $d(T) \geq d(T')$ for any ready task T'. Consider task $T'' \in Succ(T')$ as any successor of T'. In this case, we necessarily have $d(T') \geq d(T'')$ as shown in Theorem 2. Therefore, $d(T)$ is a non-increasing function along any increasing sequence of decision orders because $d(T) \geq d(T') > d(T'')$ whenever $do(T) > do(T')$ and T'' is any successor of any unscheduled but ready task T'. ∎

A dominant chain of tasks is a sequence of immediate tasks in a schedule such that delaying one of these tasks cause increasing in the schedule time. The CD strategy is to schedule tasks along dominant chains of tasks until the task-level drops (Theorem 3) below that of some ready tasks in which case scheduling switches to the next chain. The task-level of all tasks used in one scheduling iteration completely identifies a solution or a state. Therefore, IRS can be seen as a deterministic evolutionary process [9] that has hereditary variation and differential production.

It changes through iterations such that each new state (solution) is similar to previous state and yet different. The similarity is present because the task-level does not always significantly change, from one iteration to another, to trigger a change in the decision function $d(T)$. Reversing the decision order requires relatively significant change in task-level in a differential manner.

Each state is evaluated through mapping of task-level to completion times for all tasks. This means that inferior task assignment are discarded because "excessive" task delay in current state means "excessive" task-level in next iteration, thus partially improves the local state. Each new iteration is likely to retain some similarity with previous states and introduces variation at the same time. IRS continues until finding some balancing which corresponds to some steady local minima. We expect this corrective process to explore a space of "good" solutions that allows sharpening of the finish time.

6 Evaluation of two-iteration CD scheduling

The objective is to compare performance of local scheduling heuristics and the proposed approach which is based on pre-evaluation of the task-priority and generalized list scheduling. We compare our approach to the well known PD/ETF [7]. A heuristic called CD/R is used to randomly select ready tasks and assign them to run at their earliest starting times. This is useful to compare the effect of random and deterministic task selections. Testing includes three PD heuristics (PD/ETF, PD/HLF, and PD/HLETF) and five CD heuristics (CD/ETF, CD/HLF, CD/HLF*, CD/HLETF, and CD/HLETF*).

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 three parameters. The average communication to the average computation is denoted by $\alpha = \Sigma_{T,T'} c(T',T) r_{min} \,/\, \Sigma_T \mu(T) = c_{arc}/\mu_T$, where r_{min} is the least time to transfer a unit of data between two processors (set to 1), $\mu(T)$ is the computation time of T, c_{arc} is the average edge communication time, and μ_T is the average task time.

The graph parallelism is the average number of tasks that can be made ready to run at the same time. This can be measured by using the ratio of the sum of all computation times in the problem over the sum of computation times along the longest chain ($X_{longest}$). $X_{longest}$ is a chain of immediate tasks starting at entry node and ending at exit node such the sum of all its task times is the largest among all available chains. In other words, the graph parallelism is $\sum_{T\in\Gamma} \mu(T) / \sum_{T\in X_{longest}} \mu(T)$. We define the degree of parallelism (β) as the task graph parallelism over the number of processors (p):

$$\beta = \frac{\sum_{T\in\Gamma} \mu(T)}{p \times \sum_{T\in X_{longest}} \mu(T)} \tag{5}$$

The degree of parallelism is an indicator of the average number of tasks that can be made ready per processor. It also indicates the average number of tasks that may compete for each processor. The communication system is based on letency cost model which is applied in this study to fully connected (FC), the hypercube (HC), and the ring (RG). Note that low value of α corresponds to coarse grain tasks and high values of α correspond to fine grain computations.

The studied ranges of α and β is $[0-3]$ with a step of 0.5 and $[0.5, 1, 2, 2.5, 3, 4]$, 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 α, β, and topology (126 instances), 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_b) and used as a reference of the optimum solution. To compare the finish time (ω_h) of some heuristic H to reference ω_b we use the formula $(\omega_h/\omega_b - 1)100$ that

		FC		HC		RG	
$Heuristic$	Decision	Low	High	Low	High	Low	High
PD/ETF	$est(T)$	4	7	4	9	4	10
PD/HLF	$l(T)$	8	45	25	90	40	130
$PD/HLETF$	$l(T) - est(T)$	2.5	5	3	9	4	14
CD/R	random	16	32	20	35	25	40
CD/ETF	$est(T)$	3	4	3.5	7.5	4	8.5
CD/HLF	$l(T)$	1	6.5	3	13	4	15
CD/HLF^*	$l(T)$	1	2.5	1.5	2.5	2	3
$CD/HLETF$	$l(T) - est(T)$	0.8	3	1.5	5	2	6
$CD/HLETF^*$	$l(T) - est(T)$	0.6	1.3	0.8	1.5	1	2

Table 1: Average percentage deviation from ω_b for all the tested heuristics

represents the percent deviation of H from ω_b. The average percent deviation is obtained by averaging the deviations for a given instance of studied levels of communication (α), parallelism (β), and network topology.

Table 1 lists the average percent deviation, from ω_b, of the finish time as achieved by the studied heuristics for the fully connected (FC), hypercube (HC), and ring (RG) topologies. Entries of the table are obtained by averaging the finish time over the studied range of parallelism. The columns entitled Low and High denote the range of low communication ($0 \leq \alpha \leq 1$) and high communication ($2 \leq \alpha \leq 3$).

6.1 Random versus deterministic task selection

Although the random scheduling CD/R assigns each randomly selected task at its earliest starting time, it produces unacceptable deviation from ω_b for all studied values of communication, parallelism, and network topology. The finish time of the solutions generated by CD/R deviates from ω_b by 16% to 40% as shown in Table 1. Therefore, deterministic task selection is needed specially when increasing parallelism and communication.

6.2 The task-level under the PD approach

Heuristic PD/HLF significantly deviates from ω_b because task selection according to highest-level-first does not handle to idle time left just before starting the task. PD/HLETF overcomes some of the deficiency of PD/HLF but still have unacceptable deviation (9% and 14% for HG and RG) from ω_b because the benefit of task-level cannot be exploited with PD approach. For PD, the starting times of successively scheduled tasks is a non-decreasing sequence in time (horizontal), while, the benefit of task-level is to be able to schedule dominant tasks in sequence on some processor (vertical reservation) prior to assigning idle processors. These criteria are obviously conflicting.

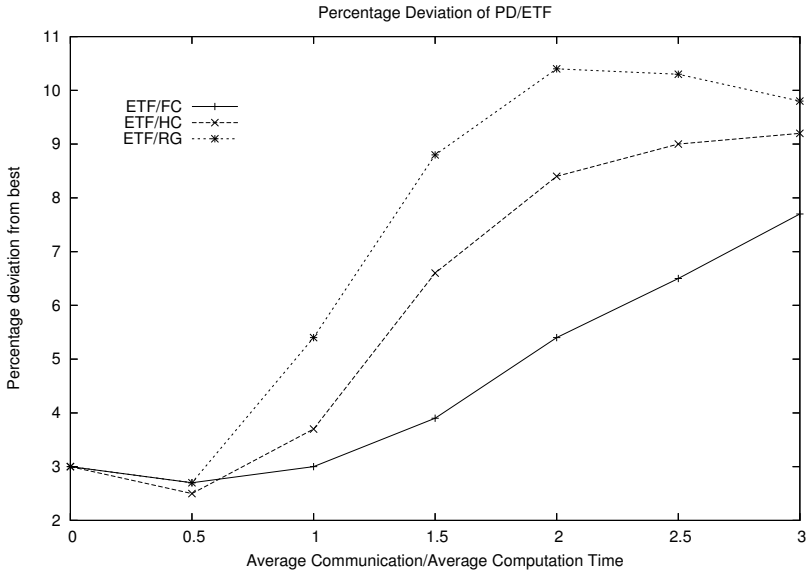

Figure 3: Average percentage deviation of PD/ETF from ω_b

6.3 Local heuristics

Heuristics PD/ETF and CD/ETF have nearly the same average deviation from ω_b with small advantage to CD/ETF (2%). The PD and CD approaches are identical within the framework of local scheduling. However, the slight advantage of CD/ETF over PD/ETF is due to the use of the effective earliest-starting-time in CD/ETF (Eq. 4) against the theoretical one (Eq. 3) in PD/ETF. Figure 3 shows the average deviation of PD/ETF from ω_b for the FC, HC, and RG topologies, respectively. A deviation of 5% or less 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, analysis of the data gives:

$$\frac{N_T}{N_L} \cdot \frac{\mu_T}{c_{arc}} \geq \epsilon_{top}.p \tag{6}$$

Therefore, to achieve acceptable deviation (5%) the inherent parallelism (N_T/N_L) and the communication ratio (c_{arc}/μ_T) imposes 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 Figure 3.

6.4 The generalized task-level under the CD approach

The heuristics CD/HLF, CD/HLF*, CD/HLETF, and CD/HLETF * give acceptable deviation from ω_b for low to average communication ($0 \leq \alpha \leq 1.5$). However, the best results are obtained for CD/HLF* and CD/HLETF* which show that the main issue is

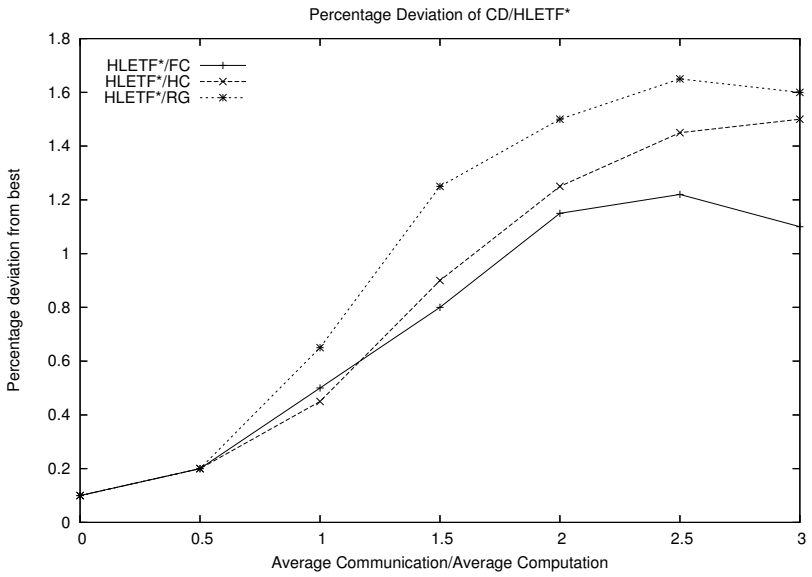

Figure 4: Average percentage deviation of CD/HLETF* from ω_b

to combine the knowledge on task-level with efficient management of the processor idle time.

The plots of the average deviation for heuristic CD/HLETF * is shown on Figure 4 for the FC, HC, and RG topologies, respectively. While CD/HLF deviates by more than 8% for ($\alpha > 1.5$), heuristic CD/HLETF overcomes most of the deficiency of CD/HLF with respect to processor utilization because CD/HLETF slightly increases its deviation with increasing communication. For all studied levels of parallelism, the peak deviation of CD/HLETF is 4.5%, 6%, and 7.5% for the FC, HC, and RG topologies, respectively. Heuristic CD/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 CD/HLETF * compared to those of CD/HLF and CD/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 CD/HLETF * that maintains small deviation over the studied range of communication, parallelism, and multiprocessor topologies.

6.5 Analysis of the distribution

Analysis of the distribution is carried out for PD/ETF and CD/HLETF * because these heuristics are the best representative of local and CD, respectively. The boundary (Figure 5) of the best 50% and 90% population of the finish time versus the available parallelism (β) is studied here. 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 CD/HLETF * does not exceed the 1.5% level. The 90% boundary is nearly about 18% for PD/ETF

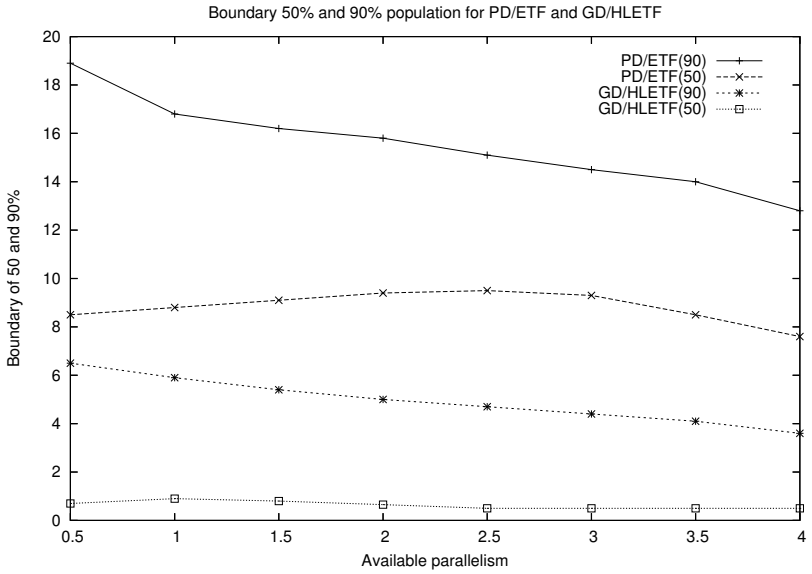

Figure 5: Boundary of 50% and 90% population for PD/ETF and CD/HLETF*

against 4% to 7% maximum deviation for CD/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 CD/HLETF *. PD/ETF slightly reduces its 50% deviation boundary versus increasing parallelism while CD/HLETF * maintains constant deviation at the same boundary level. The dependency on parallelism and topology appears only at the 90% boundary level for CD/HLETF *.

7 Evaluation of iterative refinement

The iterative refinement scheduling applies to generalized list scheduling heuristics for which the selection function $d_k(T)$ at some iteration k is increasing function of the task completion time $ct_{k-1}(T)$ (task-level) as achieved in the previous iteration $k-1$. The objective is to find shorter finish time than that generated following the first two iterations. The process relies on using a more refined estimate of the task-level throughout the iterative process. Evaluation of the iterative refinement was carried out for all the studied CD heuristics by using the previously defined random graph generation.

7.1 Iterative process behavior

Iteratively scheduling generated graphs by using a given H_{cd} is characterized by the previously defined communication parameters (α), parallelism (β), and network topology (FC, HC, and RG). For each heuristic H_{cd} and each instance of α, β, and network topology, a

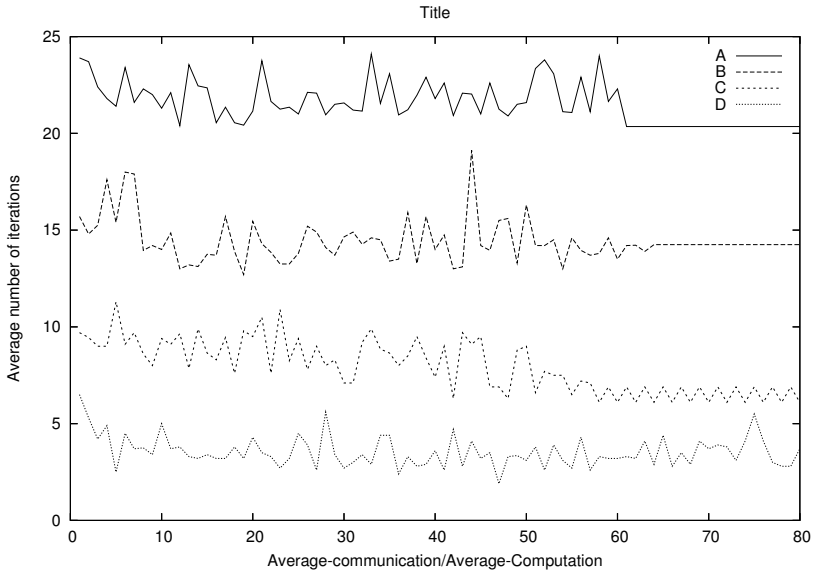

Figure 6: The four behavioral types of iterative refinement scheduling

space of solutions is explored through the iterative refinement process. Denote by (ω_{best}) the shortest finish time that is found through the iterative process by using some heuristic and for a given problem.

The first observation is that the finish time of the solutions found by the iterative refinement fluctuates but sharply tends to find solutions with shorter finish time compared to that obtained from two-iteration CD scheduling (Section 6). The iterative process behavior can be classified into four categories: a) converges to its best solution ω_{best} (Figure 6-a), b) converges to a solution other than ω_{best} (Figure 6-b), c) becomes cyclic over a number of iterations (Figure 6-c), and d) does not converge (Figure 6-d). The behavior of the four types of iterative refinement that are shown on Figure 6 is taken for the instance ($\alpha = 1$, $\beta = 2$, and FC) and heuristic $CD/HLETF^*$. The iterative refinement for $CD/HLETF^*$ gives the least improvement over its two-iteration solution because its two-iteration finish time already has little deviation from ω_b.

The average number of iterations (N_s) required to reach any of the first three states a, b, or c is characterized by 1) N_s is nearly the same for states a, b, and d, and 2) N_s strongly depends on the problem instances (α, β, and network topology). The last type (d) may converge if the iterative process is continued beyond N_s.

Increasing the communication requirements of a problem instance leads to increasing the effect of the scheduling decision on the solution finish time because of increasing the number of alternatives for scheduling immediate tasks. Therefore, N_s increases with increasing the communication parameter α. Coarse-grain computation (low α) requires the least value of N_s for any given instance of β and topology. The network topology has similar effect to that of the communication because assigning a task T to a processor p implicitly affects overall finish time due to connectivity of p and its associated communication costs.

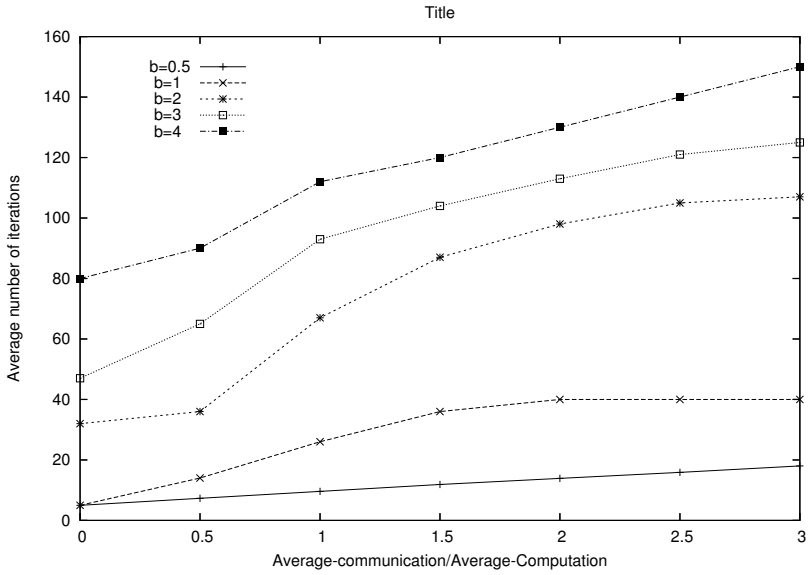

Figure 7: Effect of increasing parallelism and communication on N_{stable}

Increasing parallelism leads to increasing the average number of tasks that compete for every free processor. This in turn increases the effect on the overall finish time depending on the used task selection function. Therefore, increasing parallelism implies increasing the number of iterations required to reach any of the stable states.

Figure 7 shows the average number of iterations N_s versus increasing communication and parallelism for iteratively scheduling $CD/HLETF^*$ with the FC network topology. The plot of function $N_s = F(\alpha, \beta)$ for the HC and RG topologies are fundamentally similar with some gradual shifting due to increasing of the network communication penalties. For each network topology, the quasi-linearity of function $N_s = F(\alpha, \beta)$ enables finding analytical expressions based on experimental data.

7.2 Repetitive random versus iterative refinement

It might be thought that the improvement brought by applying the proposed iterative refinement is due to arbitrary selection of ready tasks at each iteration. To investigate this point, repetitive random (CD/R) scheduling was compared to CD iterative refinement. We compared the best finish time of the solutions generated by using an equal number of iterations in both scheduling. The iterative refinement of H_{cd} was able to significantly improves its best solution compared to its two-iteration finish time. Although a repetitive Random scheduling was able to produce better results than single-iteration Random, it was far from delivering a good schedule.

Another aspect is the study of the effect of randomly setting the task-levels for starting the iterative refinement with CD heuristics. This random-starting uses random task-levels only for the first iteration of the iterative refinement. The result is that random-starting was quickly able to generate solutions that are comparable (less than 2% deviation) to

those generated by CD iterative refinement and over different problem instances. Both iterative behaviors were comparable in finish time except in that random-starting required more iterations to converge. The experiments clearly indicate that CD iterative refinement is a *deterministic process* that searches in a space of solutions with *highly probable improvement* over the finish time.

7.3 Improvements to CD heuristics

For each instance of α, β, and studied network topology, we generate 250 computation graphs and apply the iterative refinement algorithm with each CD scheduling heuristic. The termination condition of the iterations corresponds to convergence, oscillation, or when the number of iterations reaches 100.

The performance function of the iterative refinement of a heuristic H_{cd} is the shortest finish time $\omega_{cd}(N_s)$ that is found through the iterations. The iterative refinement is compared to the the finish time ($\omega_{cd}(2)$) generated following two-iteration scheduling (Section 6) by using the formula ($\omega_{cd}(2)/\omega_{cd}(N_s) - 1$). This enables measuring the average percent improvement due to iterative scheduling. Figure 8 shows the average percent improvement for heuristics $PD/HLETF$, CD/HLF, and $CD/HLETF^*$ with the FC network topology. Heuristics (PD/HLETF, CD/HLF, and PD/HLF (not shown)) that gave unacceptable deviations or performed poorly (PD/HLF) in the two-iteration scheduling achieved impressive improvement through the iterative refinement. The iterative refinement achieves greater improvements in the upper part (I_{upper}) of the studied range of communication α and parallelism β as well as for the HC and RG network topologies.

Evaluation of two-iteration scheduling indicates that the most deviation from ω_b, for a given heuristic, is always located in (I_{upper}) because of the need for efficient overlapping between computation and communication. I_{upper} corresponds to the largest studied solution-space as $N_s(\alpha, \beta)$ increases with increasing communication and parallelism for a given network topology. The iterative refinement is able to significantly improve the solution generated by the two-iteration scheduling specially for problem instances within (I_{upper}) as shown in Figure 8. The greatest improvement is obtained for those heuristics whose two-iteration scheduling significantly deviates from ω_b. The more the CD heuristic deviates from ω_b, the more is the expected improvement with its iterative refinement.

7.4 Comparison to other approaches

Pase [15] experimentally studied 12 scheduling heuristics ($S_1 - S_{12}$) including the *task duplication* technique ($O(pn^2)$) [10] and PD/ETF [7] (S_2). His heuristic (S_1) assigns priority from the graph bottom and selects the task that is closest to top. The priority function (page 17 of [15]) is evaluated based on task computation times but neither account for the communication edges nor the network latency. He founds that S_1 and PD/ETF are among the best heuristics and both outperform the TD scheduler of [10]. Our study indicates that the two-iteration scheduling as well as the iterative refinement significantly

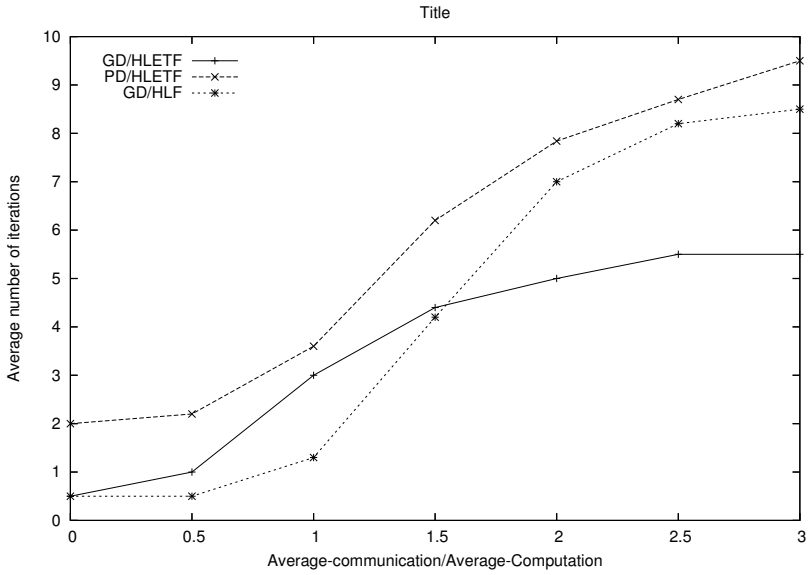

Figure 8: Improvement due to iterative refinement over two-iteration scheduling

	Average Deviation			Iterative Refinement	
Comm/Comp	DSC/ETF	DSC/Sarkar	HLETF/ETF	HLETF/ETF	Iter.
0.1-0.3	0.06	5.56	2.8	5.73	5
0.83-1.25	3.3	20.74	3.27	6.87	22
3.3-10	2.36	19.39	6.97	12.02	37

Table 2: Comparison of DSC, Sarkar, PD/ETF, CD/HLETF*, and iterative CD/HLETF*

outperforms PD/ETF versus change in communication, parallelism, and network topology. All the scheduling heuristics, including ours, fundamentally have identical number of steps ($O(pn^2)$) but differ slightly in the constant.

The mobility intervals used as task-priority used by Wu [21] ($O(n^3)$) are too inaccurate because they incorporate all the communication carried by the edges and do not address the processor selection problem.

The heuristic called *Dominant Sequence Clustering* (DSC) [22] was proposed for scheduling DAGs on unbounded number of completely connected (FC) processors. DSC scheduling improves the clustering approach presented in Sarkar and Hennessy [17] but slightly outperform (3.3%) PD/ETF as reported by Yang and Gerasoulis [22] who studied scheduling with the FC network. Heuristic PD/ETF is used as reference in [22] and this work, therefore we can compare our work to that reported in [22, 17]. DSC ($O(n \log n)$), PD/ETF, and Sarkar clustering have been studied over an unbounded number of processors and their comparison is shown in columns 2 and 3 of Table 2 as reported in [22]. The second step of DSC is to merge clusters in order to match the number of clusters with that of the processors. The second step is likely to increase the finish time that results from the use of an unbounded number of processors. Therefore, Table 2 represents the best results of DSC that can be compared to PD/ETF.

K/TD	P/TD	Sarkar	DSC	Pase	ETF	CD
$O(n^4)$	$O(n^3(nlogn+e))$	$O(n(n+e))$	$O((n+e)logn)$	$O(pn^2)$	$O(pn^2)$	$O(pn^2)$

Table 3: Comparison of time complexities

The deviations shown in Table 2 represent the average percent improvement of H_X over H_Y that is evaluated by $(1-\omega_X/\omega_Y)100$. DSC and two-iteration CD/HLETF* nearly achieve the same improvement over PD/ETF in the low to medium communication range (coarse grain) but CD/HLETF* outperforms DSC for fine grain tasks ($3.3 \leq \alpha \leq 10$). The use of the iterative refinement with CD/HLETF* significantly outperforms all the above scheduling specially for problem instances (fine-grain) where it is hard to find good solutions. The two-iteration scheduling (HLETF*) and its iterative refinement largely outperforms PD/ETF over all studied range of parallelism, communication, and network topology. The cost of iterative refinement is linear with the cost of CD scheduling.

Finally, Table 3 compares the time complexities of Kruatrachue's [10] and Papadimitriou's [14] task-duplication scheduling, Sarkar and Hennessy's [17] and Yang and Gerasoulis's clustering [22] over unbounded number of processors, Pase's scheduling [15], Hwang and others' earliest-task-first [7], and our proposed computation-driven scheduling.

8 Conclusion

We have addressed the problem of finding efficient scheduling heuristics for precedence-constrained computations with communication times targeted on distributed memory systems.

The task-level is fundamental to differentiate critical from non-critical computations and communications. We generalized the notion of task-level in a manner to incorporate the effects of computation, volume of transferred data, and network latency. The approximate task-level was set to the task completion time that was obtained by backward scheduling the computation graph. We proposed a new approach called *Computation-driven scheduling* which combined with our generalized task-level enables early reservation of resources to critical computations and communications. The next step was to use the task-level in forward scheduling according to the principle of *highest-level-first*.

The task-level can easily be improved through an *iterative refinement process* that consists of alternatively scheduling the computation graph and its associated reverse over a number of iterations. Information on task-levels passes from one iteration to another in order to refine the tasks-level and, consequently, optimizes the solution.

We carried out extensive experimental evaluation for different instances of problem granularities, inherent parallelism, and network latency for the fully connected, cube, and ring. It is found that our two-iteration scheduling outperforms all known scheduling heuristics for the studied granularities, parallelism, and networks. The iterative refinement scheduling was shown to explore a space of solutions whose size grows with the amount of parallelism and communication granularity. Solutions generated by our iterative refinement largely outperform all known approaches specially for fine-grain problems

where other approaches fail.

The iterative refinement is a low cost scheduling approach that can be implemented as a compiler optimization for programming distributed memory systems. It is specially useful for large-scale programs that are compiled once but repeatedly executed over different data sets. Future extension to this work would address the problem of finding generic model of network latency to incorporate the effect of contentions. Real-time monitoring of the iterative refinement scheduling may present an advanced approach for synthesizing "good" solutions in the presence of all system's effects.

References

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Comm. of the ACM*, 17, No 12:685–690, Dec 1974.

[2] M. Al-Mouhamed. Lower bounds on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Trans. on Software Engineering*, 16, No 12:1390–1401, 1990.

[3] M. Al-Mouhamed. Analysis of macro-dataflow dynamic scheduling on non-uniform memory access architectures. *IEEE Trans. on Parallel and Distributed Systems*, 19, No 3:875–888, Nov 1993.

[4] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. on Applied Mathematics*, 17:416–429, 1969.

[5] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 5, No 2:113–120, Feb 1994.

[6] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, No 9:841–848, May 1961.

[7] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C. Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Computing*, pages 244–257, Apr 1989.

[8] S. J. Kim and J. C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. *Proc. of the Inter. Conf. on Parallel Processing*, 3:1–8, Aug 1988.

[9] R. M. Kling and P. Banerjee. ESP: Placement by simulated evolution. *IEEE Trans. on Computer-Aided Design*, 8, No 3:245–256, Mar 1989.

[10] B. Kruatrachue. Static task scheduling and grain packing in parallel processing systems. *Ph.D. Thesis, Department of Computer Science*, 1987. Oregon State University.

[11] Y.-K. Kwok and I. Ahmad. Dynamic critical path scheduling: An effective technique for scheduling task graphs onto multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 7, No 5:506–521, 1996.

[12] Y.-K. Kwok and I. Ahmad. On using task duplication in parallel program scheduling. *Under review with IEEE Trans. on Parallel and Distributed Systems*, 1996.

[13] E. A. Lee and J. C. Bier. Architectures for statically scheduled dataflow. *J. of Parallel and Distributed Computing*, 10:333–348, 1990.

[14] C. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Computers.*, pages 322–328, 1990.

[15] D. M. Pase. A comparative analysis of static parallel schedulers where communication costs are significant. *Ph.D. Thesis, Oregon*, Jul 1989.

[16] P. Y. Richard Ma, E. Y. S. Lee, and T. Masahiro. A task allocation model for distributed computing systems. *IEEE Trans. on Computers*, C-31:41–47, Jan 1982.

[17] V. Sarkar and J. Hennessy. Compile-time partitioning and scheduling of parallel programs. *Proc. of the SIGPLAN Symp. on Compiler Construction*, pages 17–26, Jul 1986.

[18] J. Sheild. Partitioning concurrent VLSI simulated programs onto multiprocessor by simulated anealling. *IEEE proceedings*, 134:24–30, Jan 1987.

[19] G. C. Sih and E. A. Lee. Scheduling to account for interprocessor communication within interconnection constrained processing network. *Inter. Conf. on parallel Processing*, I:9–16, 1990.

[20] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. on Parallel and Distributed Systems*, 10, No 2:175–187, Feb 1993.

[21] M.-Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. on Parallel and Distributed Systems*, 1, No 3:330–343, Jul 1990.

[22] T. Yang and A. Gerasoulis. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. on Parallel and Distributed Systems*, 5, No 3:951–967, Sep 1994.