Scheduling Optimization Through Iterative Refinement

Mayez Al-Mouhamed and Adel Al-Masarani
Computer Engineering Department
College of Computer Science and Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Abstract

Scheduling DAGs with communication times 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, data size, and network latency. A new scheduling that uses the proposed task-level to make early reservation of resources for critical computation and communication is proposed. We also propose an optimization called Iterative Refinement Scheduling (IRS) that alternatively schedules the computation graph and its associated reverse. The task-level used in some scheduling iteration is the task's starting time that is achieved in the very previous iteration. IRS 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 problem granularities, parallelism, and network latency such as the fully connected, hypercube, and ring. The finish time obtained from two-iteration scheduling outperforms those generated by other recently proposed scheduling as well as the clustering approaches.

IRS allows exploring a space of solutions whose size grows with the amount of parallelism and communication granularity. Solutions generated by IRS largely outperform all known approaches specially for fine-grain problems where other approaches fail. IRS enables optimizing the solution specially for critical instances such as fine-grain DAGs and large parallelism. The time complexity of one IRS iteration is $O(pn^2)$.

1 Introduction

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 such as the class of static dataflow computations. In that case, the compiler is capable of predicting the computation and communication requirements as well as the precedence relationships and volume of transferred data between the various computation modules.

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 the processors. Generally, the scheduling problem is NP-complete [5] except for trees. Lower bounds [1] and worst case analysis [8, 1] have been proposed for scheduling precedence computations with and without communication costs. The objective is to find efficient nonpreemptive scheduling approaches that combine knowledge on the computation structure and target 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: *Searching*, *Clustering*, *Task-Duplication*, and *Priority-Based Scheduling.*

Methods based on branch-and-bound [13] and simulated annealing [15] have also been used as heuristics for mapping and partitioning computations. These approaches either minimize objective functions other than the computation finish time or lack global evaluation because only partial scheduling problems are addressed [9, 14]. Recently, genetic algorithms [6] were also applied to scheduling without communication times.

Linear clustering [9] 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 unbounded number processors [14] 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) [18] was proposed to enhance the work reported by Sarkar and Hennessy [14]. 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 $O(e(e+n))$ [14] and $O(\log n(e+n))$ [18], where n and e are the numbers of tasks and communication edges, respectively.

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 task duplication were experimentally studied by Pase [12] by evaluating the Hu's task-priority [7] as the sum of task times from the graph bottom.

Priority-based scheduling basically uses a selection criterion in order to assign a ready task to a processor so that to meet some local strategy as an attempt to minimize overall finish time. The notion of task-level is one of fundamental priority information because it allows differentiating critical from non-critical tasks. Unfortunately, the evaluation of the task-level [3] for precedence-constrained computations with communication times is not tractable because the task-level strongly depends on the way the tasks are mapped to the processors and the implied communications. In other words, the knowledge of the computation structure is not sufficient to distinguish between critical and non-critical tasks. Discarding the communication and network effects in evaluating the task-level [10, 12], or pessimistically accounting for all of them [17] leads to excessively inaccurate evaluations. To avoid these problems, only local scheduling heuristics have been proposed such as the principles of *earliest-task-first* [8], *largest-communication-first* [2], etc. Local scheduling heuristics minimize the processor idle time that is one strategy towards minimizing overall finish time.

Our objective is to find efficient scheduling heuristics for precedence-constrained computations with communication times targeted to distributed memory systems. We 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 provides additional insite for the characterizing of our scheduling and its search space and enables comparisons with other approaches.

The evaluation of task-level is developed in Section 2. Section 3 presents the computation driven scheduling and the proposed method for iterative refinement. Section 4 presents one computation-driven algorithm and its time complexity. Section 5 presents the evaluation of iterative scheduling over fully-connected, Hypercube, and ring topologies. Section 6 compares our results to other approaches and Section 7 concludes about this work.

2 The task-level

A fundamental result of scheduling theory [4] is the introduction of list-scheduling (LS) and its bound [5]. The objective function is to minimize the execution time of precedence-constrained computations with no communication times. LS implicitly enforces the starting times of successively scheduled tasks be non-decreasing sequence in time which enabled finding the worst case bound. The task-level is independent from mapping the tasks to processors because of zero communication.

A major issue in this work is the evaluation of task-level in the case of non-zero communication which is not tractable The task-level depends on the mapping of the tasks to processors because the communication delays between the processors are not identical.

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 processors so that overall execution time is minimum. The computation can be modeled [8] by using a directed acyclic graph $G(\Gamma, \rightarrow, \mu, C)$ where $\rightarrow$, $\mu(T)$, and $c(T', T) \in C$ denote the precedence constraints, the task execution time, and the volume of data sent from task T' to its 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 one unit of data from a processor p' to p, through the interconnection network, is $r(p', p)$, where p and p' are the processors that are assigned tasks T and T', respectively. Assuming that the communication media is contention-free, the time to transfer $c(T', T)$ messages is $c(T', T)r(p(T'), p(T))$.

Let T be a task and denote by $D(T)$ the set of predecessors of T. The earliest-starting-time $est(T, p)$ of T for processor p depends on: 1) the completion time $ct(T', p')$ of predecessor T' on p', 2) the number of messages $c(T', T)$ sent from T' to T, and 3) the cost of routing one unit of messages from p' to p:

$$est(T, p^*) = min_p\{\max_{T' \in D(T)}\{ct(T', p') + c(T', T).r(p, p')\}\} \tag{1}$$

where p^* ia a processor that can start T at the earliest time. Note that $est(T, p) = 0$ for every p if $D(T) = \emptyset$. $est(T, p)$ depends on *how tasks are mapped to processors* as well as the implied network latency.

In our approach we consider $est(T, p)$ as an *estimate of the distance from entry to* T and propose *refining* the estimate through an iterative scheduling process based on *forward/backward* passes. We first schedule the reverse graph (G_r) over $S = S(P, R)$ 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.

Following the scheduling of G_r, the primary task-level $(l(T))$ is set to the achieved completion time $(ct(T))$ to be used as task priority in the next forward scheduling. $l(T)$ 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 any chain of immediate tasks that starts at T and finishes at some exit node. This forward/backward scheduling is repeated in an attempt to minimize overall finish time.

In the next section, we discuss Graham's method 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 compared to existing scheduling approaches.

3 The computation-driven

In list scheduling [5] when at least one processor completes execution of T, the successors of T are examined in order to find out whether some of them become ready-to-run. Only the successors of those newly completed tasks are involved in the updating process. We call this approach Processor-Driven (PD) because the scheduler tracks the increasing sequence of processor completion times using a *global time*. Mainly, PD leads to uniform scheduling of the processors because the starting times of successively scheduled tasks form a non-decreasing sequence in time. This strategy was designed with a worst-case bound [5, 8] in mind. Examples of PD scheduling are list-scheduling, the ETF heuristic [8], and others [12, 16, 17].

We propose an approach called *Computation-Driven* (CD) that differ from the PD by: 1) a dynamic decision function $d(T) = F(l(T), \ldots)$ that is increasing function of the task-level, 2) the task with the highest $d(T)$ is assigned to run at the earliest, and 3) any successor of T, whose predecessors have all been assigned to some processors but not necessarily completed, is added to the set of ready-to-run. Tasks with higher levels are scheduled in-sequence along chain of immediate tasks until the priority is reversed to the benefit of other chains which leads to out-of-sequence scheduling, i.e. effect of processor reservation. This can be clarified by the following theorems.

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'. ■

In the following we introduce the notion of dominant chain of tasks that can only be defined with respect to a given schedule because the distance from starting a task to completion of the computation is dependent on processor assignment.

Let p and p_i be the processors that are assigned tasks T and $T_i \in Pred(T)$, respectively. Denote by $lmt(T_i, T, p)$ the earliest time the *last-message* $c(T_i, T)$ for task T reaches processor p. Formally, we have $lmt(T_i, T, p) = ct(T_i) + c(T_i, T)r(p_i, p)$ but if $p(T_i) = p$ then $lmt(T_i, T, p)$ will be reduced to $ct(T_i)$ as zero messages are sent from T' to T within p. Following the selection of a task T, H_{cd} always assigns T to run on some processor p that can start T at the earliest time among all the processors.

By definition of $est(T,p)$, T cannot start earlier than the largest $lmt(T_i,T,p)$ among all its predecessors and provided that processor p is free. CD scheduling guarantee that the selected task is scheduled at the earliest time among all free processors. Since, T is scheduled by some H_{cd} on p, it should be that $est(T,p) = \max\{\max_T\{lmt(T_i,T)\}, t(p)\}$, where $t(p)$ is the earliest time p has to become free when T was assigned to p.

Definition 1 *For an arbitrary CD schedule, a predecessor T' of T is said to be dominant with respect to T if its last-message time for T is the highest among all predecessors of T: $lmt(T',T,p) = \max_{T_i \in Pred(T)}\{lmt(T_i,T,p)\}$.*

Definition 2 *For an arbitrary CD schedule, a chain of immediate tasks (directed path) $X: T_1 \to T_2 \ldots T_L$ is said to be dominant if $ct(T_L) = \omega_{cd}$ and for every task $T \in X$ such that $Pred(T) \neq \emptyset$ there exists at least one dominant predecessor for T in X.*

For any CD heuristic, each sequence of increasing decision orders is necessarily a non-increasing sequence of task-levels. The strategy of computation-driven is to schedule tasks along dominant chain of tasks until the task-level drops (Theorem 3) below that of some ready tasks in which case scheduling switches to the next chain. Dominant chain of tasks with substantially higher task-levels are sequentially scheduled in an attempt to make *early processor reservation*, i.e., non-uniform task scheduling. The above tasks are implicitly assigned by CD to the most suited cluster of processors with the lowest network latency because the selected tasks are assigned to run at the earliest time. The scheduling process switches to other paths when the value of the decision function has decreased up to the level of those unscheduled but ready tasks in which case the reservation process is restarted for another chain.

CD scheduling becomes uniform with respect to processor selection and ready tasks only when the remaining computations (distance to exit) for all ready tasks are nearly identical. The computation-driven scheduling tends to reduce the difference in task-level between unscheduled tasks prior to switching to uniform scheduling.

The processor-driven always proceeds in uniform scheduling because this is the best strategy in the absence of information to differentiate dominant from non-dominant chain of tasks. The CD approach exploits the task-level as an indicator of remaining distance along each chain of tasks and performs non-uniform scheduling for dominant chains in an attempt to equalize the remaining computation and communication.

The objective of CD scheduling is to minimize the finish time based on the estimated task-levels and the incurred processor idle time. The iterative refinement exploits the available completion times of the tasks in one iteration in order to yield some refinement by using the achieved completion times as task-levels for the next iteration.

Consider a dominant chain of tasks $X: T_1 \ldots \to T_{u-k} \to \ldots T_u$ that belongs to the schedule generated by some iteration and let X' be any non-dominant chain. Using definitions 1 and 2, a number of terminal tasks of X ($T_{u-k} \to \ldots T_u$) finish later than the terminal task (T_w) of X'. Due to dependence over the processor assignment, a dominant chain in one iteration may become non-dominant in another iteration. One of the reasons for which non-critical chains may become dominant is the accumulated delays along the starting of its tasks due to inaccurate task-level.

In the next iteration, the dependence edges are reversed and the ready tasks include both T_u and T_w at the starting of the scheduling. The effect on the next iteration is that up to some level k the value of the decision function for the tasks of X exceeds that of T_w, i.e., $d(T_u) > d(T_{u-1}) > \ldots > d(T_{u-k}) > d(T_w)$. Some starting tasks of chain X are implicitly given priority over non-dominant chains. The result is that sub-chain $X_{top}: T_u \to \ldots T_{u-k}$ is non-uniformly scheduled with the least delay because there is no sharing with other chains.

Processors that are not used by X_{top} can be used later for other chains at higher decision orders. The starting times of successively scheduled tasks are not constrained by any order under the computation-driven scheduling. In the following we consider two typical cases.

If the impact of giving sub-chain X_{top} more priority leads another chain X' to become dominant, then, chain X was given excessive priority compared to X' and some correction is needed. The same reasoning leads chain X' to be given more priority in the next iteration in order to correct the condition for which X' was delayed by giving excessive priority to X.

Now assume X_{top} and X' both become dominant and, consequently, the difference between their task completion times is not large enough to trigger in-depth non-uniform scheduling for either one. In this case, both chains will be uniformly scheduled in the subsequent iteration and the process will be continuously repeated.

The iterative refinement process continues in either cases until finding a balancing between the present chains that corresponds to some steady solution. We expect this corrective process to explore a space of "good" solutions that allows sharpening the finish time generated by simple application of CD scheduling.

4 Example of a CD heuristic

In the following we present algorithm CD/HLETF as one representative heuristic for the CD class. CD/HLETF operates as follows. Initialization consists of setting the task-levels $l(T)$ as the completion times of the previous scheduling iteration. The main loop repeats until there are no unscheduled tasks. The task T^* with the highest $d(T^*) = l(T^*) - est(T^*, p^*)$ 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

CD/HLETF has the task-levels $\{l(T)\}$ as inputs and uses 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. CD/HLETF consists of selecting a task T^* and a processor p^* such that $l(T^*) - est(T^*, p^*)$ is the highest among all the tasks of A. 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 whose earliest-starting-time can only be achieved on p^*. 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

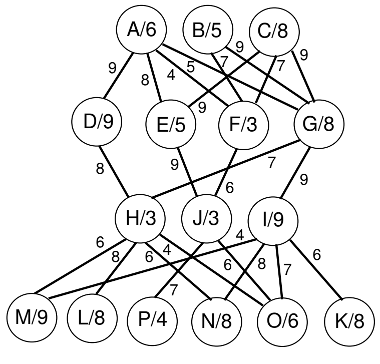

Figure 1: Example of DAGs

ready. Variable $t(p)$ holds the earliest time processor p is idle. The outputs of CD/HLETF are the completion times $\{ct(T)\}$ of the tasks and their processor assignment $\{p(T)\}$. Algorithm CD/HLETF is the following:

Algorithm: CD/HLETF

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

(2) While $|B| < n$ Do
      Begin
      (2.1) Select $T^* \in A$ and $p^*$ :
            $l(T^*) - est(T^*, p^*) =$
            $\max_{T \in A}\{l(T) - \min_p\{est(T,p)\}\}$
      (2.2) Assign $T^*$ on $p^*$ : $p(T^*) = p^*$,
            $ct(T^*) = t(p^*) = est(T^*, p^*) + \mu(T^*)$
            remove $T^*$ from $A$, add $T^*$ to $B$
            For each $T \in A$, update:
            $est(T, p^*) = max\{est(T, p^*), t(p^*)\}$

      (2.3) Repeat for each task $T \in Succ(T^*)$ :
            $\lambda(T) = \lambda(T) - 1$ ,
            If $\lambda(T) = 0$ Then update $A$ :
                XS$A \leftarrow A + \{T\}$,
                evaluate $est(T,p)$ for each $p$:
                $est(T,p) =$
                $\max\{\max_{T' \in Pred(T)}\{ct(T')+$
                $c(T',T).r(p(T'),p)\}, t(p)\}$
      End
```

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 $\lambda(T) = \lambda(T) - 1$ 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 is then $O(pn^2)$.

Using the DAG shown in Figure 1, Figure **??** shows the schedules generated by: a) algorithm ETF [8], a) the first iteration (backward) of IRS scheduling, c) the second iteration (forward) of IRS. Figure refsched shows the set of ready-to-run tasks and decision order for the second iteration (forward) of IRS scheduling in which lst denotes the task level.

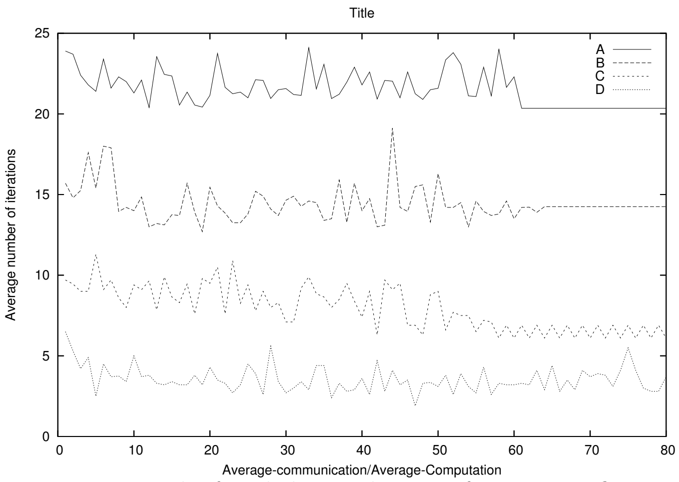

Figure 2: The four behavioral types of iterative refinement scheduling

5 Evaluation of IRS

A *random graph generator* is used to generate graphs with few hundred tasks with different computation and communication setting. The following three parameters are used for setting the generated problems and multiprocessor. The *average communication to average computation* is denoted by $\alpha = \Sigma_{T,T'} c(T',T) r_m \,/\, \Sigma_T \mu(T) = c_{arc}/\mu_T$, where r_m is the least time to transfer a unit of data between two processors (set to 1). The *degree of parallelism* ($\beta = N_T/pN_L$) 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. The *topology of the interconnection network* that is the fully connected (FC), the hypercube (HC), and the ring (RG). Note that low and high values of α corresponds to coarse and fine grain computations, respectively.

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. For each instance of α, β, and topology (126 instances), we use the uniform distribution to generate 250 random computation graphs.

The finish time of the solutions found by the iterative refinement fluctuates but sharply tends to find solutions with shorter finish time. The iterative process behavior can be classified into four categories: a) converges to its best solution ω_{best} (Figure 2-a), b) converges to a solution other than ω_{best} (Figure 2-b), c) becomes cyclic over a number of iterations (Figure 2-c), and d) does not converge (Figure 2-d). The behavior of the four types of iterative refinement that are shown on Figure 2 is taken for the instance ($\alpha = 1$, $\beta = 2$, and FC) and heuristic $CD/HLETF$. Similar results are obtained for the HC and RG topologies.

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

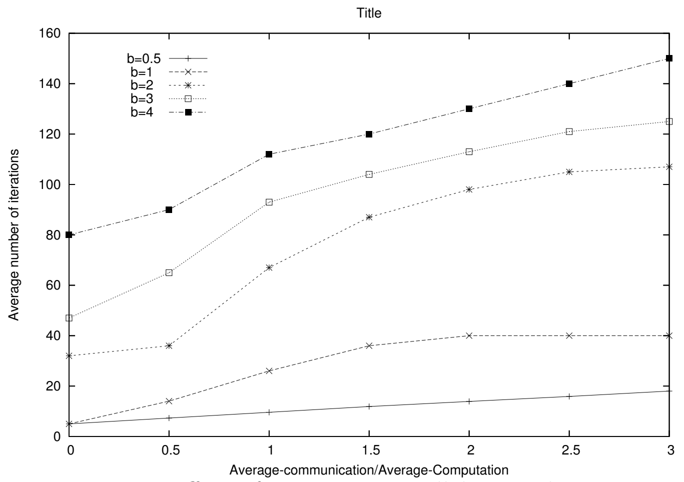

Figure 3: Effect of increasing parallelism and communication on N_{stable}

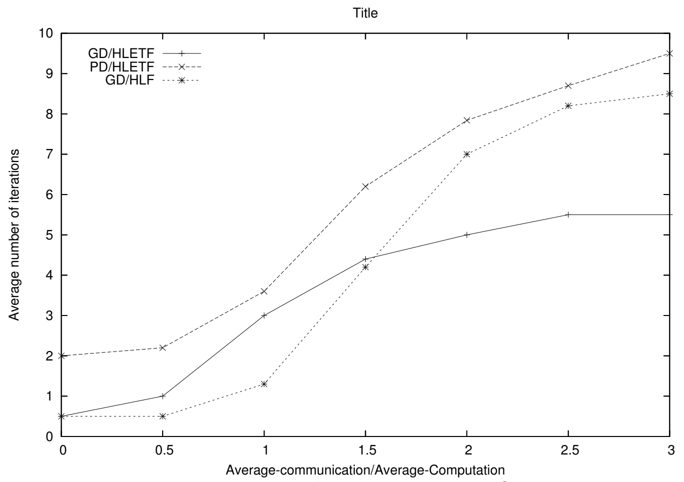

Figure 4: Improvement due to iterative refinement over two-iteration scheduling

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.

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 3 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.

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 scheduling was compared to CD iterative refinement over the same number of iterations. Although a repetitive Random scheduling was able to produce better results than single-iteration, it was far from delivering a good schedule. 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.

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 generated following two-iteration scheduling by using the formula $(\omega_{cd}(2)/\omega_{cd}(N_s) - 1)$. This enables measuring the average percent improvement due to iterative scheduling. Figure 4 shows the average percent improvement for the FC topology which is due to 20 refinement iterations over two-iteration scheduling for heuristics: 1) $PD/HLETF$ that is *processor-driven* and uses $d(T) = l(T) - est(T)$, 2) CD/HLF that is *computation-driven* and uses $d(T) = l(T)$, and 3) $CD/HLETF$ that is *computation-driven* and uses $d(T) = l(T) - est(T)$. 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. I_{upper} corresponds to the upper range of communication and inherent parallelism in the studied problems which corresponds to fine grain task graphs ($2 \leq \alpha \leq 3$) and large inherent parallelism ($\beta = 4$).

The iterative refinement is able to significantly improve the solution generated by the two-iteration scheduling specially for fine grain tasks, relatively large communication, and large inherent parallelism.

6 Comparison

Pase [12] experimentally studied 12 scheduling heuristics ($S_1 - S_{12}$) including the *task duplication* technique ($O(pn^2)$) [10] and PD/ETF [8] (S_2). The priority function (page 17 of [12]) 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 outperforms PD/ETF versus change in communication, parallelism, and network topology. All the scheduling heuristics, including ours, fundamentally have identical number of steps ($O(pn^2)$).

The mobility intervals used as task-priority used by Wu [17] ($O(n^3)$) 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* [18] was proposed for scheduling DAGs on unbounded number of completely connected (FC) processors. DSC scheduling improves the clustering approach presented in Sarkar and Hennessy [14] but slightly outperform (3.3%) PD/ETF as reported by Yang and Gerasoulis [18] who studied scheduling with the FC network. 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 the table below. 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, the above table represents the best results of DSC that can be compared to PD/ETF.

We compare the relative merit of IRS scheduling and other approaches. The deviations shown in the table below represent the average percent improvement of H_X over H_Y that is evaluated by $(1 - \omega_X/\omega_Y)100$ and each column represent the range for the ratio of average communication over average computation:

Comm/Comp	0.1-03	0.83-1.25	3.3-10
DSC/ETF	0.06	3.3	2.36
DSC/Sakar	5.56	20.74	19.39
HLETF/ETF	2.8	3.27	6.97
IRS/ETF	5.73(5)	6.87(22)	12.02(37)

The additional numbers in the last row represent the number of IRS iterations to achieve the result. 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.

In the following we compare the time complexity of scheduling heuristics. The time complexities of Kruatrachue's [10] and Papadimitriou's [11] task-duplication scheduling which are $O(n^4)$ and $O(n^3(nlogn + e))$, respectively. The heuristic presented by Sarkar and Hennessy [14] is $O(n(n + e))$. Yang's and Gerasoulis's clustering [18] over unbounded number of processors has a time complexity of $O((n + e)logn)$, where e is the number of edges. Finally, Pase's scheduling [12], Hwang's and others Earliest-Task-First [8], and our proposed Computation-Driven Scheduling have $O(pn^2)$.

7 Conclusion

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 task-level enables early reservation of resources to critical computations and communications. The next step was to use the task-level in forward scheduling.

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 levels of granularities, parallelism, and network latency. 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.

8 Acknowledgments

The authors would like to acknowledge support from the College of Computer Science and Engineering and the Research Committee at the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

References

[1] 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, Dec 1990.

[2] 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.

[3] M. Al-Mouhamed and A. Al-Maasarani. Performance evaluation of scheduling precedence-constrained computations on message-passing systems. *IEEE Trans. on Parallel and Distributed Systems*, 5, No 12:1317–1322, Dec 1994.

[4] E.G. Coffman et al. *Computer and Job-Shop Scheduling Theory.* John Willy and Sons, 1976.

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

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

[7] T.C. Hu. Parallel sequencing and assembly lines problems. *Operations Research*, 9:841–848, 1961.

[8] J.-J. Hwang, Y.-C. Chow, F.D. Anger, and C.Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Computers*, pages 244–257, Apr 1988.

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

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

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

[13] 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.

[14] 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.

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

[16] G.C. Sih and E.A. Lee. Scheduling to account for interprocessor communication within interconnection constrained processing network. *Proc. of the ICPP*, I:9–16, 1990.

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

[18] 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.