An Analysis of Scheduling Coarse-Grain Iterative Task Computation on Message-Passing Architectures

Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106
tyang@cs.ucsb.edu

Abstract

Scheduling task computation and predicting its performance on message-passing architectures are important for program parallelization. In this paper we consider the scheduling of a weighted iterative task graph (ITG) in the presence of communication overhead. An ITG represents a sequence of parallel task computation. Each iteration contains the execution of a set of tasks and there exists task dependence within an iteration and between iterations. We provide a lower bound for optimal scheduling when the weights of iterative task graphs change monotonically in the course of iterations. We devise a valid schedule without searching task instances of all iterations, examine the asymptotic performance of this schedule compared to an optimal solution. Finally, we present case studies and experimental results on nCUBE-2 to verify our solutions.

1 Introduction

The problem of NP-hard scheduling has been addressed extensively in the literature. A basic representation of parallel computation is a directed acyclic graph (DAG) which represents the dependence between the instances of task computation. DAG scheduling has been used in compiling functional parallelism [PAM93, S89a, WF93], and programming tools [EL90, WG88, YG92]. Many scheduling algorithms for weighted DAGs with the presence of communication have been proposed, e.g. [EL90, GY93a, KB88, S89a, WG88, YG92]. The main optimization is to minimize parallel time by balancing load, reducing communication overhead, and overlapping communication with computation.

An iterative task graph (ITG) represents a sequence of parallel task computation. Each iteration contains the execution of a DAG and there exists task dependence within an iteration and between

iterations. ITGs can model scientific programs which involve loop and coarse grain task computation. For loop parallelism, the iteration space of a loop is composed of all instances of the loop body, which is a DAG. Using DAG scheduling techniques to obtain a near-optimal mapping solution requires a searching of the entire iteration space, which may not be feasible when loop bounds are too large or unknown. Loop parallelism can be explored by various loop transformation methods [W89]. An estimation of parallel time or speedup can facilitate the selection and evaluation of transformation techniques. Notice that the parallel time of a transformed program depends on the number of processors and the weight of computation and communication delay.

In this paper, we consider the scheduling of weighted ITGs containing task and loop parallelism. The goal is to devise a schedule that satisfies dependence for all instances of tasks and minimize the parallel time without searching the entire iteration space. Such a result would be useful to predict the performance of program parallelization on distributed memory machines and guide program transformation and code generation.

Mapping ITGs on message-passing architectures involves the examination of dependence cycles. Our work has also been motivated by the scheduling of fine grain instruction-level parallelism studied in the context of software pipelining [AN88, LAM88, MS90, GS92, PM91, REI68, VGN92, ZS89] in which computation between loop iterations are overlapped. For executing coarse-grain task parallelism on message-passing architectures, pipelined computation can achieve a good performance if computation and communication are properly orchestrated. In deriving a near-optimal solution, we need to consider the characteristics of asynchronous parallelism based on macro-dataflow computation model, the effect of communication delay, and weight variations in the course of iterations. We need to overlap not only the computation between iterations, but also communication with computation.

In Section 2, we give a definition of DAG and ITG scheduling, and some basic terminologies. In Section 3, we give a lower bound for optimal ITG scheduling. In Section 4 we derive a valid schedule when the number of processors is equal to the number of tasks within the iteration body, and the weight functions are monotonically increasing or remain the same in the course of iterations. We compare the asymptotic performance of this solution to that of any optimal result derived by searching the expanded task graph. Then we discuss loop unrolling to improve scheduling performance if more processors are available. In Section 5, we give a solution for task graphs with monotonically-decreasing weight functions. In Section 6, we provide an analysis on scheduling two numerical computing graphs and give the experiment results that verify the correctness of analytic results.

2 Problem Definition

A directed acyclic task graph (DAG) contains a set of task nodes and communication edges between tasks represent their data dependence. If task T_y depends T_x, then task T_y cannot start to execute

until T_x has completed its computation and the data produced by T_x is available at the local memory of the processor that executes T_y. We use the macro dataflow model for task computation [S89a]: the execution of task T_x is triggered by the arrival of all data items that T_x needs. At the completion of its execution the data are sent in parallel to successor tasks. Each task has its computation weight and the communication cost between tasks is zero if they are mapped to the same processor. Certain communication delay occurs in sending a data item from one processor to another. For example, the delay of processor-to-processor communication is usually modeled as $s + tm$ where s is the startup cost and m is the size of a message to be transferred and t is the transmission speed. The following table gives the communication and single-precision computation speeds of two machines in microseconds taken from Dunigan [Dun91].

	startup time (s)	trans. speed (t)	add or mult
iPSC/860	136 μs	0.4 μs /byte	0.1 μs
nCUBE-II	160 μs	0.6 μs /byte	0.8 μs

The physical distance between processors also affects communication delay, especially in a store-forward routing network; however, in a wormhole routing network the cost is not so sensitive to processor distances. Kumar et.al. [K94] discussed formulas for communication delay in a store-forward or wormhole routing network. In this paper, we assume that the worst-case cost is used in estimating task communication delay. Sarkar [S89a] discussed methods for estimating communication and computation cost.

Given a DAG, a schedule for this DAG is defined by a processor assignment mapping $PA(T_j)$ and a starting time mapping $ST(T_j)$ for all tasks T_j. Fig. 1(a) shows a weighted DAG with all computation cost equal to 1. Communication weights are labeled in the edges. Fig. 1(b) shows a processor assignment using 2 processors. Fig. 1(c) shows a *Gantt chart* of a schedule for this DAG. The Gantt chart completely describes the schedule since it defines both the processor assignment and the starting time for each task. The scheduling problem with nonzero communication delay has been shown to be NP-complete for a general task graph, by Chretienne [Ch89], Papadimitriou and Yannakakis [PY90], and Sarkar [S89a].

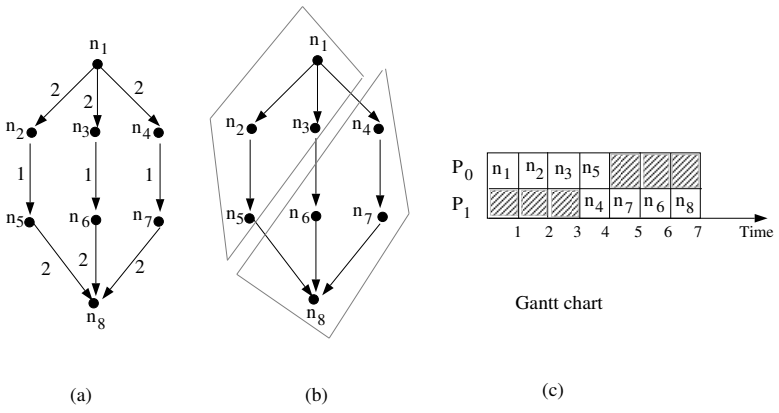

Figure 1: (a) A DAG with node weights equal to 1. (b) A processor assignment of nodes. (c) The Gantt chart of a schedule.

An iterative task graph (ITG) G contains v tasks $T_1, T_2, \cdots, T_v$, and e dependence edges. This graph is repeatedly executed N iterations. The expanded graph $E(G, N)$ is the DAG resulted from such an execution. Let the instance of task T_i at iteration k be T_i^k $(0 \leq k \leq N-1)$. Thus $E(G, N)$ contains all $N * v$ task instances T_i^k $(1 \leq i \leq v, 0 \leq k \leq N-1)$.

Some definitions on ITG scheduling are listed below.

- Let (T_i, T_j) be a dependence edge from T_i to T_j in G, the iteration delay is a distance function $d_{i,j}$. Namely, task instance $T_j^{k+d_{i,j}}$ depends on T_i^k.

  Some dependence graphs may contain imprecise dependence direction information [W89] instead of precise distance values. If $d_{i,j} =' +'$, there exists q $(q \geq 1)$ such that task instance T_j^{k+q} depends on T_i^k. If $d_{i,j} =' *'$, there exists q $(q \geq 0)$ such that T_j^{k+q} depends on T_i^k. Notice that a negative distance is not plausible.

- $succ(T_i)$ is the set of tasks immediately depending on T_i in G.

- The weight function of task T_i^k is $\tau_i(k)$.

- The communication cost from T_i^k to $T_j^{k+d_{i,j}}$ is $c_{i,j}(k)$. $c_{i,j}(k)$ is 0 if these two tasks are assigned to the same processor. We assume that the cost of fetching data items from a local memory or communication buffer is negligible compared to the cost of transferring a message from one processor to another.

- $ST(T_i^k)$ is the starting time of task T_i^k. $CT(T_i^k)$ is the completion time of task T_i^k, defined as $ST(T_i^k) + \tau_i(k)$.

- $PA(T_i^k)$, the processor location of T_i^k.

For example, a partitioned BLAS-3 program for solving a lower triangular system $Ax = b$ is listed in Fig. 2. A is a lower triangular matrix of $n \times n$, The matrix A is partitioned into $N \times N$ submatrices where $n = N \times h$. Each submatrix $A_{i,j}$ is of dimension $h \times h$. The column vectors x and b of dimension n are partitioned into N subvectors, $X_1, X_2, \cdots X_N$ and $B_1, B_2, \cdots, B_N$. Fig. 3 is an ITG for this program. The dependence from S to T_q broadcasts a partial solution vector X_j. The dependence from T_q to S is to send one subvector B_{j+1} to S[1].

Fig. 4 is another example of a partitioned loop program whose loop body contains three tasks. Fig. 5(a) is the ITG of this program. (b) is the expanded task graph of G representing the iteration space of this program. (c) is a schedule of this program on 3 processors. We assume that the communication delay between tasks assigned in different processors is 1 time unit. The task weight functions are $\tau_x(k) = k + 1$, $\tau_y(k) = 1.5k + 1$ and $\tau_z(k) = 2k + 1$.

[1]The dependence is actually overestimated since only one edge for each iteration is needed while in Fig. 3 there are P edges from $T_1, T_2, \cdots, T_P$ to S. For this case, it is not easy to summarize inter-iteration dependence accurately. In fact, current dependence analysis algorithms [W89] may overestimate dependence since accurate estimation may be intractable; however, overestimation does not affect the correctness of parallel execution.

For $j = 0$ to $N-1$

S^j: Solve $A_{j+1,j+1}X_{j+1} = B_{j+1}$.

for q= 1 to P

T_q^j

for $i = \lceil \frac{j+1-q}{p} \rceil * p + q$ to N step P

$B_i = B_i - A_{i,j+1} * X_{j+1}$.

endfor

endfor

endfor

Figure 2: A partitioned program for solving a triangular system.

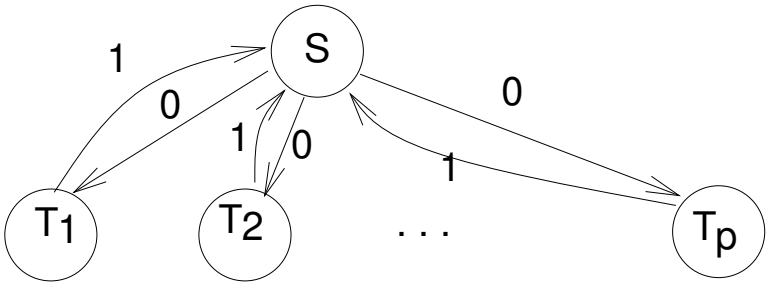

Figure 3: An ITG for solving a triangular system.

do k = 0, N-1

T_x^k : $a[k] = f_1(b[k-2])$;

$T_y^k : b[k] = f_2(b[k-1], a[k], c[k-1])$;

$T_z^k : c[k] = f_3(b[k])$;

enddo

Figure 4: A partitioned program.

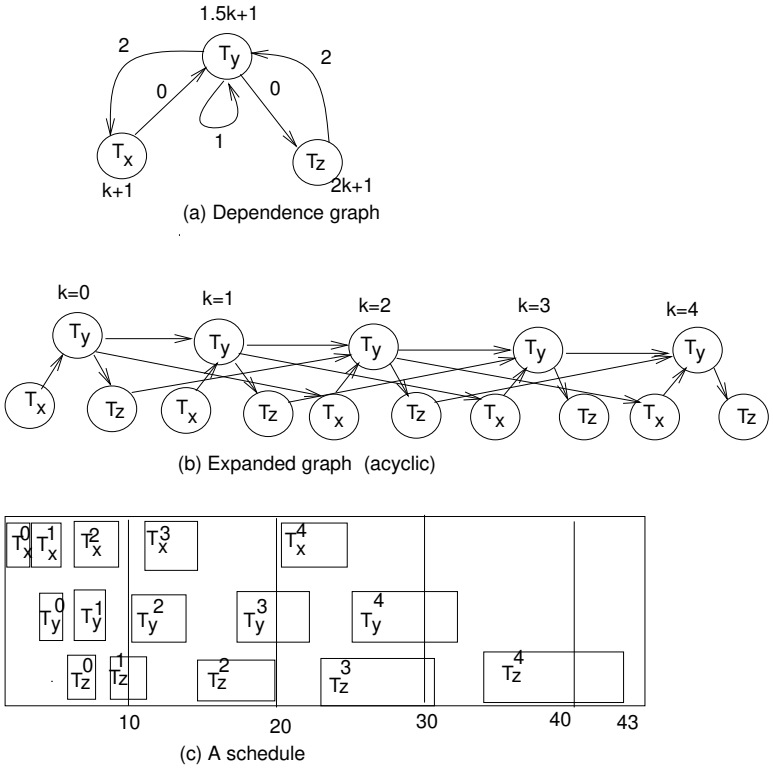

Figure 5: (a) The ITG G for Fig. 4. (b) The expanded graph $E(G,5)$. (c) A valid schedule.

A scheduling algorithm will determine $PA(T_i^k)$ and $ST(T_i^k)$. A valid schedule must satisfy the following constraint:

- **Resource constraint** (R1):

  $1 \leq PA(T_i^k) \leq p$ where p is the given number of processors.

  And if $PA(T_i^k) = PA(T_j^t)$ for two different task instances T_i^k and T_j^t in $E(G,N)$, then $ST(T_i^k) - CT(T_j^t) \geq 0$ or $CT(T_i^k) - ST(T_j^t) \leq 0$.

- **Dependence constraint** (R2):

  If a dependence delay from T_i to T_j is an integer d, then for any two tasks T_k^{k+d} and T_i^k in $E(G,N)$, $ST(T_j^{k+d}) - ST(T_1^k) \geq \tau_i(k) + c_{i,j}(k)$.

If weights of G change irregularly during iterations, it is difficult to obtain a valid uniform schedule for each task $ST(T_i^k)$ without examining the entire expanded graph of G. In practice, weight functions usually possesses certain regularity depending on iteration numbers. In this paper we first consider weights monotonically increase or remain the same from one iteration to another. More specifically,

- $\tau_i(k) = a_i k + b_i$ where $a_i \geq 0$ and $b_i \geq 0$.

- $c_{i,j}(k) = u_{i,j}k + v_{i,j}$ where $u_{i,j} \geq 0$ and $v_{i,j} \geq 0$.

We consider the case when the weights decrease monotonically in Section 5.

The following terms will be used in the rest of this paper. Let

$$A = \sum_{i=1}^{v} a_i, \quad B = \sum_{i=1}^{v} b_i,$$

$$a^{max} = \max_{i=1}^{v} a_i, \quad a^{avg} = \frac{A}{v}, \quad b^{max} = \max_{i=1}^{v} b_i, \quad b^{avg} = \frac{B}{v}.$$

Let C be a cycle in graph G, without the loss of generality, assume that $C = T_1, T_2 \cdots T_j, T_1$. Let

$$A1_C = \sum_{i=1}^{j} a_i, \quad B1_C = \sum_{i=1}^{j} b_i$$

$$A2_C = a_j + u_{j,1} + \sum_{i=1}^{j-1}(a_i + u_{i,i+1}), \quad B2_C = b_j + v_{j,1} + \sum_{i=1}^{j-1}(b_i + v_{i,i+1}),$$

$$A2 = \max_{\text{all cycles C}} A2_C, \quad B2 = \max_{\text{all cycles C}} B2_C.$$

3 Lower bounds of ITG scheduling with precise dependence

Given p processors and a DAG, a lower bound of parallel time for executing this graph is $max(\frac{Seq}{p}, L_{crit})$ where Seq is the summation of the total computation weights and L_{crit} is the length of the critical path (the longest path) in this DAG. We examine performance bounds for scheduling an ITG G below.

Theorem 1 *Given an ITG with precise dependence and p processors, let C be a cycle in G, D_C is the summation of distances in this cycle. Let $M_C = \lfloor \frac{N-1}{D_C} \rfloor$.*

$$PT_p \geq \frac{0.5 * A * N(N-1) + BN}{p} \quad \textit{and} \quad PT_p \geq \max_{\textit{all cycles } C} (M_C * B1_C + 0.5 M_C (M_C - 1) * A1_C * D_C).$$

Proof: The total computation weight of $E(G, N)$ is:

$$Seq = \sum_{k=0}^{N-1} \sum_{i=1}^{v} (a_i * k + b_i) = A * N(N-1)/2 + B * N.$$

Thus a performance bound with respect to the computation load is

$$PT_p \geq \frac{A * N(N-1)/2 + B * N}{p}.$$

Next we examine a dependence path in $E(G,N)$. Assume that there is a dependence cycle C in G, $C = T_1, T_2, \cdots T_j, T_1$. Let $d_{i,i+1} = d_i$ for $i = 1, \cdots j-1$ and $d_j = d_{j,1}$. The summation of dependence distance is

$$D_C = d_{j,1} + \sum_{i=1}^{j-1} d_{i,i+1} = \sum_{i=1}^{j} d_i.$$

Then a path in $E(G,N)$ is

$$T_1^0, T_2^{d_1}, T_3^{d_1+d_2}, \cdots, T_j^{d_1+d_2+\cdots+d_{j-1}}, T_1^{D_C}.$$

Then another path is

$$T_1^{D_C}, T_2^{D_C+d_1}, T_3^{D_C+d_1+d_2}, \cdots, T_j^{D_C+d_1+d_2+\cdots+d_{j-1}}, T_1^{2D_C}.$$

Then we can start more paths from $T_1^{2D_C}$, $T_1^{3D_C}$ and so on. The length of the q-th path ($q = 0, 1, 2, \cdots$) is

$$L(q) = \sum_{i=1}^{j}(a_i(q * D_C + \sum_{t=1}^{i} d_t)) + b_i \geq q * A1_C * D_C + B1_C.$$

The total number of such paths are $M_C = \lfloor \frac{N-1}{D_C} \rfloor$, i.e. $0 \leq q < M_C$. Notice that these M_C paths constitute one path starting from T_1^0 to $T_1^{M_C * D_C}$. The length of the critical path in $E(G,N)$:

$$L_{crit} \geq \sum_{q=0}^{M_C-1} L(q) \geq M_C * B1_C + M_C(M_C-1)/2 * A1_C * D_C.$$

■

Example: In Fig. 5(a) there a dependence cycle T_y, T_z, T_y with $D_C = 2$. A dependence path in $E(G,N)$ expanded from this cycle is $T_y^0, T_z^0, T_y^2, T_z^2, T_y^2, T_z^2, T_y^4$ when $N = 5$ and $M_C = 4/2 = 2$.

When $a^{avg} = 0$, the bound is reduced to $L_{crit} \geq M_C * B1_C = \lfloor \frac{N-1}{D_C} \rfloor B1_C$ which is a lower bound for any cyclic scheduling [REI68, LAM88, PM91].

4 Scheduling on v processors

We present a schedule for an ITG with precise dependence when the number of processors is v (equal to the number of tasks in the loop). We will provide an analysis of its asymptotic performance. Then we use loop unrolling to improve the scheduling performance if more processors are available. We also discuss a solution for dealing with an imprecise dependence graph.

4.1 Schedule Determination

We structure a schedule as follows: The instances of task T_i are assigned to processor i. Task T_i^{k+1} is executed after T_i^k. We express the starting time of task T_x^k as follows and we will compare this

schedule with an optimal schedule in Section 4.2.

$$ST(T_x^k) = \alpha_x + (\beta_x + r)k + \frac{H}{2}k^2. \quad (1)$$

The following coefficients are to be determined: two nonnegative values α_x and β_x corresponding to each task, and two nonnegative coefficients H and r. Notice that these coefficients may not be integral.

We compute the coefficients in two steps described below.

Step 1 For each dependence edge (T_x, T_y) $(T_x \neq T_y)$ we setup the following inequalities called $F1$:

$$\beta_y - \beta_x + dH \geq a_x + u_{x,y}.$$

We call a procedure $LPATH(F1, v, H, a^{max})$(see the appendix for the details) to determine values of β_x $(1 \leq x \leq v)$ and H in a complexity of $O(ve \log \frac{A2}{\epsilon})$ satisfying the above inequalities and also $H \geq a^{max}$. ϵ is the desired accuracy in finding the minimum value H.

Step 2 For each dependence edge (T_x, T_y) $(T_x \neq T_y)$ in G, we setup the following inequalities called $F2$:

$$\alpha_y - \alpha_x + r * d_{x,y} \geq b_x + v_{x,y} - \beta_y d_{x,y} - \frac{Hd^2}{2}.$$

Let $r_1 = max(0, b_x - \beta_x)$. We call $LPATH(F2, v, r, r_1)$ in a complexity of $O(ve \log \frac{B2}{\epsilon})$ to determine r and α_x $(1 \leq x \leq v)$ that satisfy the above inequalities and $r \geq r_1$.

The total complexity of this algorithm is $O(ve \log \frac{A2*B2}{\epsilon^2})$. Now we show that the above algorithm produces a legal solution.

Theorem 2 *The schedule determined by the above algorithm is legal, i.e. satisfying conditions R1 and R2.*

Proof: Condition R1. Since all and only instances of the task T_x are executed in processor x, the following condition is sufficient to make R1 true:

$$ST(T_x^{k+1}) - ST(T_x^k) \geq \tau_x(k) = a_x k + b_x. \quad (2)$$

That is

$$ST(T_x^{k+1}) - ST(T_x^k) = \beta_x + r + Hk \geq a_x k + b_x.$$

Since in Step 1, $H \geq a_x$ and in Step 2, $r \geq r_1 \geq max(0, b_x - \beta_x)$, Inequality (2) holds true for every processor.

Condition R2. We first show that all self-dependence is satisfied automatically. Notice that instances T_x^k of task T_x are executed in the same processor in an order of $k = 0, 1 \cdots$. When task T_x in G has a self dependence, for any integer d ($d \geq 1$), $ST(T_x^{k+d}) - CT(T_x^k) \geq \tau(k)$.

Next we examine examine non-self dependence. For each edge (T_x, T_y) in G $(T_x \neq T_y)$, let $d = d_{x,y}$. We need to show:

$$ST(T_y^{k+d}) - ST(T_x^k) \geq \tau_x(k) + c_{x,y}(k) = a_x k + b_x + u_{x,y} k + v_{x,y}. \quad (3)$$

Using expression (1), $ST(T_y^{k+d}) - ST(T_x^k)$ can be expanded as:

$$(\beta_y - \beta_x + dH)k + \alpha_y - \alpha_x + (\beta_y + H + r)d + \frac{Hd^2}{2}.$$

Notice that in Step 2, β_x $(1 \leq x \leq v)$ and H are solved such that the following condition is true:

$$\beta_y - \beta_x + dH \geq a_x + u_{x,y}$$

and

$$\alpha_y - \alpha_x + (\beta_y + H + r)d + \frac{Hd^2}{2} \geq b_x + v_{x,y}.$$

Thus inequality (3) is true for every dependence edge.

■

Example: For Fig. 5(a), we have the following inequalities $F1$ in Step 1:

$$\beta_y - \beta_x \geq 1, \quad \beta_x - \beta_y + H2 \geq 1.5,$$
$$\beta_z - \beta_y \geq 1.5, \quad \beta_y - \beta_z + H2 \geq 2.$$

Notice that $v = 3$, all communication weights are 1 and $a^{max} = 2$. The results of procedure $LPATH(F1, 3, H, 2)$ are $H = 2$, $\beta_x = 0$, $\beta_y = 1$ and $\beta_z = 2.5$.

In Step 2, notice that all $b_i = 1$ and $v_{i,j} = 1$. We have the following inequalities $F2$:

$$\alpha_y - \alpha_x \geq 1 + 1, \quad \alpha_x - \alpha_y + r2 \geq 1 + 1 - 0 * 2 - 2^2,$$
$$\alpha_z - \alpha_y \geq 1 + 1, \quad \alpha_y - \alpha_z + r2 \geq 1 + 1 - 1 * 2 - 2^2.$$

$r_1 = max(0, b_x - \beta_x) = 1$ and procedure $LPATH(F2, 3, r, 1)$ produces a solution $r = 1$, $\alpha_x = 0$, $\alpha_y = 2$ and $\alpha_z = 4$. We have the following schedule:

$$ST(T_x^k) = k + k^2,$$
$$ST(T_y^k) = 2 + 2k + k^2,$$
$$ST(T_z^k) = 4 + 3.5k + k^2.$$

Thus the parallel time is $PT = ST(T_z^{N-1}) + \tau_z(N-1) = 4 + 3.5(N-1) + (N-1)^2 + 2(N-1) + 1$. The sequential time

$$Seq = \sum_{k=0}^{N-1}(k + 1 + 1.5k + 1 + 2k + 1) = 4.5(N-1)N/2 + 3(N-1).$$

When N is large, the speedup is approximately equal to $4.5/2 = 2.25$.

4.2 Performance Analysis

We estimate the asymptotic speedup of the above method, and compare its performance with that of an optimal DAG schedule for the entire expanded graph $E(G, N)$.

The performance of scheduling depends on the granularity of task graphs. A definition and analysis on granularity of a DAG is given in [GY93a] and we use a slightly different definition to quantify the granularity of an ITG G as:

$$g = \min_{T_x \in G}\{\min_{T_y \in succ(T_x)} \frac{a_x}{u_{x,y}}\}.$$

Theorem 3 *If a^{max} and b^{max} are bounded constants, let*

$$Q(G) = \max_{all\ cycles\ C} \frac{A1_C}{D_C}, \quad Q2(G) = \max_{all\ cycles\ C} \frac{A2_C}{D_C}.$$

Then the asymptotic speedup is:

$$\lim_{N\to\infty} \frac{Seq}{PT} = \frac{A}{H} = \frac{a^{avg} * v}{max(a^{max}, \epsilon + Q2(G))}.$$

The asymptotic performance bound is:

$$\lim_{N\to\infty} Sup\{\frac{PT}{PT_{opt}}\} = \max(\frac{a^{max}}{max(a^{avg}, Q(G))}, 1 + 1/g + \frac{\epsilon}{max(a^{avg}, Q(G))}).$$

Proof: From step 1,

$$H = \max(a^{max}, \epsilon + Q2(G)).$$

Since $A2_C \leq A1_C + 1/gA1_C$,

$$H \leq max(a^{max}, \epsilon + (1 + 1/g)Q(G)).$$

Since

$$PT = \max_{T_x \in G}\{\alpha_x + (\beta_x + r)(N-1) + \frac{H}{2}(N-1)^2 + \tau_x(N-1)\},$$

when N is very large,

$$PT = \frac{H}{2}N^2 + t(N)$$

where $t(N)$ is a function in an order of N. That is, $\lim_{N\to\infty} \frac{t(N)}{N^2} = 0$.

Since $Seq = A/2 * N^2 + t1(N)$ where $t1$ is a function in an order of N,

$$\lim_{N\to\infty} \frac{Seq}{PT} = \frac{A}{H} = \frac{a^{avg} * v}{max(a^{max}, \epsilon + Q2(G))}.$$

Next we examine the performance bound.

$$\frac{PT}{PT_{opt}} \leq 0.5N^2 \max(\frac{a^{max}}{PT_{opt}}, \frac{Q(G)}{PT_{opt}}(1 + 1/g) + \frac{\epsilon}{T_{opt}}) + \frac{t(N)}{T_{opt}}.$$

By Theorem 1,

$$PT_{opt} \geq 0.5(N-1)^2 A/v = 0.5N^2 a^{avg} + t2(N)$$

and

$$PT_{opt} \geq \max_C M_C(M_C - 1)/2A_C * D_C = 0.5N^2 Q(G) + t3(N)$$

where $t2$ and $t3$ are functions in an order of N. Recall that $M_C = \lfloor \frac{N-1}{D_C} \rfloor$.

$$\lim_{N\to\infty} \frac{PT_{opt}}{N^2} \geq 0.5max(a^{avg}, Q(G)).$$

Thus

$$\lim_{N\to\infty} \frac{PT}{PT_{opt}} \leq \max(\frac{a^{max}}{max(a^{avg}, Q(G))}, 1 + 1/g + \frac{\epsilon}{max(a^{avg}, Q(G))}).$$

■

Example: For Fig.5, we have

$$Q(G) = 3.5/2 = 1.75, \quad a^{avg} = 4.5/3 = 1.5, \quad a^{max} = 2, \quad 1/g = 0.$$

Notice that the communication cost is 1, thus $g = \infty$. Then the scheduling performance is bounded as:

$$\frac{PT}{PT_{opt}} \leq max(\frac{2}{1.75}, 1 + \frac{\epsilon}{1.75}).$$

For a sufficiently small ϵ, we have: $\frac{PT}{PT_{opt}} \leq 1.143$. The asymptotic speedup is 4.5/2=2.25.

When $a^{avg} = 0$, we can show that:

Theorem 4 *If all $a_x = 0$ and b^{max} is a bounded constant, then let*

$$g_b = \min_{T_x \in G}\{\min_{T_y \in succ(T_x)} \frac{b_x}{v_{x,y}}\}, \quad Q_b(G) = \max_{all\ cycles\ C} \frac{B1_C}{D_C}, \quad and \quad Q_b2(G) = \max_{all\ cycles\ C} \frac{B2_C}{D_C}.$$

Then

$$\lim_{N\to\infty} \frac{Seq}{PT} = \frac{B}{r} = \frac{b^{avg} * v}{max(b^{max}, \epsilon + Q_b2(G))},$$

and

$$\lim_{N\to\infty} Sup\{\frac{PT}{PT_{opt}}\} = \max(\frac{b^{max}}{max(b^{avg}, Q_b(G))}, 1 + 1/g_b + \frac{\epsilon}{max(b^{avg}, Q_b(G))}).$$

4.3 Unrolling loop to improve the performance

From the analysis in the previous subsection, we can see that for coarse grain parallelism with $g \geq 1$, and if the variation of coefficient a in a task weight function is small, then the performance of ITG scheduling can reach 50% of the optimum when N is large.

However, when $\frac{a^{max}}{max(a^{avg}, Q(G))}$ is getting large, the scheduling performance deteriorates. This is because processors that execute the tasks T_x with $a_x = a^{max}$ are overloaded.

Loop unrolling is a loop transformation technique to increase the number of tasks within the loop body, which can lead to an optimal solution for software pipelining [AN88, PM91, CS93] when there is a sufficient number of processors. We examine how loop unrolling could improve the scheduling performance of ITG on message-passing architectures when $a^{max} >> max(a^{avg}, Q(G))$ and more processors are available.

For the example of Fig 4, the loop unrolled by 2 is shown in Fig. 6 for which we can obtain a schedule using 6 processors with $PT = \frac{7}{8}N^2 + O(N)$ and $\lim_{N\to\infty} \frac{PT}{PT_{opt}} = 1$.

do k = 0, $\lfloor \frac{N-1}{2} \rfloor$
  $T^k_{x,1} : a[2k] = f_1(b[2k-2])$;
  $T^k_{y,1} : b[2k] = f_2(b[2k-1], a[2k], c[2k-1])$;
  $T^k_{z,1} : c[2k] = f_3(b[2k])$;
  $T^k_{x,2} : a[2k+1] = f_1(b[2k-1])$;
  $T^k_{y,2} : b[2k+1] = f_2(b[2k], a[2k], c[2k])$;
  $T^k_{z,2} : c[2k+1] = f_3(b[2k])$;
enddo
do k = $2 * \lfloor \frac{N-1}{2} \rfloor$ to N
  $T^k_x : a[k] = f_1(b[k-2])$;
  $T^k_y : b[k] = f_2(b[k-1], a[k], c[k-1])$;
  $T^k_z : c[k] = f_3(b[k])$;
enddo

Figure 6: An example of unrolled loop.

In general if we enroll a loop by f times, the number of iterations is reduced from N to $\lfloor N/f \rfloor$ and the number of tasks in the loop body is increased by f times. The unrolled graph is called G^f.

For G^f, we apply the two-step method in Section 4.1 using fv processors. We can show that

$$\lim_{N\to\infty} Sup\{\frac{PT}{PT_{opt}}\} = \max(\frac{a^{max}/f^2}{max(a^{avg}/f, Q(G))}, 1 + \frac{1}{g} + \frac{\epsilon}{max(a^{avg}/f, Q(G))}).$$

Assume that ϵ is very small. To have the tightest bound, we choose f such that

$$\frac{a^{max}/f^2}{a^{avg}/f} \leq W \quad \text{or} \quad \frac{a^{max}/f^2}{Q(G)} \leq W$$

where

$$W = 1 + \frac{1}{g}.$$

Thus the minimum f that we should choose is:

$$f = \lceil min(\frac{a^{max}}{W * a^{avg}}, \sqrt{\frac{a^{max}}{W * Q(G)}}) \rceil.$$

For such an unrolling factor f, we have

$$\lim_{N \to \infty} Sup\{\frac{PT}{PT_{opt}}\} = 1 + \frac{1}{g} + \frac{\epsilon}{max(a^{avg}/f, Q(G))}.$$

4.4 Dealing with imprecise dependence graphs

If a dependence graph has imprecise direction dependence, we first transform this graph G to a precise dependence graph G^t as follows: map the dependence direction x of each dependence edge in G as

$$DMap(x) = \begin{cases} x & \text{when x is an integer constant} \\ 1 & \text{when } x = + \\ 0 & \text{when } x = *. \end{cases}$$

Then we apply the scheduling algorithm described in Section 4.1 and we show the result is still correct for G.

Theorem 5 *Let G^t be a precise dependence graph transformed from an imprecise graph G, a valid schedule of G^t using the method of Section 4.1 must be valid for G.*

Proof: For a dependence edge from task T_x and T_y in G with direction $'*'$, this dependence distance of this edge will be 0 in G^t. In a valid schedule for G^t, $ST(T_y^{k+d}) - ST(T_x^{k+d}) \geq \tau_x(k+d) + c_{x,y}(k+d)$ for any integer d ($d \geq 0$).

Since $ST(T_x^{k+d}) \geq CT(T_x^k) = ST(T_x^k) + \tau_x(k) \geq ST(T_x^k)$, $\tau_x(k+d) \geq \tau_x(k)$ and $c_{x,y}(k+d) \geq c_{x,y}(k)$

We have $ST(T_y^{k+d}) - ST(T_x^k) \geq \tau_x(k) + c_{x,y}(k)$.

When the dependence direction is '+', for any $d \geq 1$, $ST(T_y^{k+d}) - ST(T_x^{k+d-1}) \geq \tau_x(k+d-1) + c_{x,y}(k+d-1)$. Then using the similar inference, we can show $ST(T_y^{k+d}) - ST(T_x^k) \geq \tau_x(k) + c_{x,y}(k)$.

■

5 Monotonically decreasing weight functions

In this section, we discuss the schedule determination when the weight functions monotonically decrease. We assume that the weight functions of G are:

- $\tau_i(k) = -a_i k + b_i$ where $a_i \geq 0$ and $b_i \geq a_i * (N-1)$ so that there are no negative weights.
- $c_{i,j}(k) = -u_{i,j}k + v_{i,j}$ where $u_{i,j} \geq 0$ and $v_{i,j} \geq u_{i,j}(N-1)$.

We can design a solution for this case as follows:

Step 1. We schedule this graph in a backward manner, in the sense that we first reverse the dependence edge of this graph, the new graph is G^r. We construct the nonnegative weight functions of this new graph as:

- $\tau_x^r(k) = \tau_x(N-k-1) = -a_x(N-k-1) + b_x = a_x * k + b_x - a_x * (N-1)$.
- $c_{y,x}^r(k) = c_{x,y}(N-k-1-d) = -u_{x,y}(N-k-1-d) + v_{x,y} = u_{x,y}k + u_{x,y}d + v_{x,y} - u_{x,y}(N-1)$.

Step 2. Now the new graph has monotonically increasing weight functions. We use the result of section 4.1. We derive the starting time for each task T_x^k denoted as $ST(G^r, T_x^k)$. It satisfies:

$$ST(G^r, T_x^{k+1}) - ST(G^r, T_x^k) \geq \tau_x^r(k).$$

For every edge (T_y, T_x) in G^r (corresponding to $(T_x, T_y) \in G$),

$$ST(G^r, T_x^{k+d}) - ST(G^r, T_y^k) \geq \tau_y^r(k) + c_{y,x}^r(k).$$

Let the length of this schedule be L.

Step 3. For the original task graph G, we define a schedule

$$ST(G, T_x^k) = L - ST(G^r, T_x^{N-k-1}) - \tau_x^r(N-k-1).$$

Notice that the reversed schedule has the same length as the original schedule. Thus we can apply the same performance analysis as we have done in Section 4.2. We examine the validness of this approach below.

Theorem 6 *The reversed schedule is legal for the original graph G.*

Proof

For **condition R1**

$$ST(G, T_x^{k+1}) - ST(G, T_x^k) = ST(G^r, T_x^{N-k-1}) - ST(G^r, T_x^{N-k-2}) + \tau_x^r(N-k-1) - \tau_x^r(N-k-2).$$

Thus

$$ST(G, T_x^{k+1}) - ST(G, T_x^k) \geq \tau_x^r(N-k-2) + \tau_x^r(N-k-1) - \tau_x^r(N-k-2) = \tau_x^r(N-k-1) = \tau_x(k).$$

For **condition R2**, for each edge (T_x, T_y) in G:

$ST(G, T_y^{k+d}) - ST(G, T_x^k) = ST(G^r, T_x^{N-k-1}) - ST(G^r, T_y^{N-k-d-1}) + \tau_x^r(N-k-1) - \tau_y^r(N-k-d-1)$.

Thus

$ST(G, T_y^{k+d}) - ST(G, T_x^k) \geq \tau_x^y(N-k-d-1) + c_{y,x}^r(N-k-d-1) + \tau_x^r(N-k-1) - \tau_y^r(N-k-d-1)$.

Then we have $ST(G, T_y^{k+d}) - ST(G, T_x^k) = \tau_x(k) + c_{x,y}(k)$.

■

6 Case Studies and Experiments

In this section, we examine the performance of this scheduling solution on two cases and also present the experiments on nCUBE-2. For these two numerical computing problems, we use the partitioned loop programs and analyze ITGs to evaluate the performance of parallel execution. In the implementation of parallel programs, we let each processor execute instances of one task in an increasing order of the iteration number. The execution of each task follows data-driven computation model. As soon as the data needed by a task is available in the local memory, this task starts computation and then it sends the produced results to its successors assigned in different processors. Such a method has been used in [YG92]. We will compare the actual performance with the predicted result using the method of Sections 4 and 5.

6.1 Solving a triangular system

In Fig.3, the weight of task S^j is r^2w where w is the time for addition or multiplication. The weight of task T_q^j is about $\frac{(N-j-1)2r^2w}{P}$. There are total $P+1$ tasks in the loop body and $P+1$ processors are used.

The communication from T_q to S is to send a subvector of B and the cost is approximated as $s + t * h$ where s is the startup time and t is the transmission speed [Dun91]. The communication from S to T_q is to broadcast X_{j+1}. The broadcasting communication cost is estimated as $\log(P+1)(s + t * h)$ on nCUBE-2.

The weight function is monotonically decreasing. Using our method in Sections 4 and 5, we estimate the parallel time as

$$PT = h^2w + \log(P+1)(s+ht) + (\frac{h^2w}{P} + r^2w + (1+\log(P+1))(s+ht))(N-1) + \frac{h^2w}{P}(N-1)^2.$$

We implement this schedule on nCUBE-2. Processor 0 executes task S and processor q executes task T_q $(1 \leq q \leq P)$. Each processor executes its tasks in an increasing order of iteration number

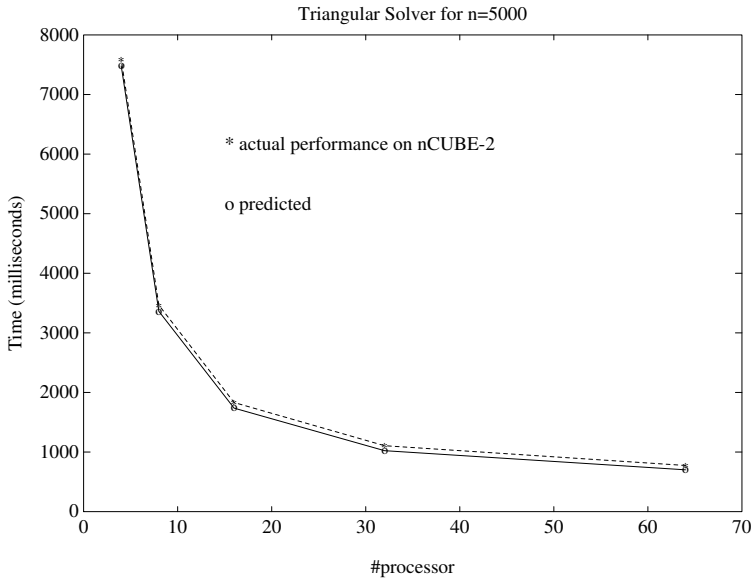

Figure 7: The performance of the triangular solver on nCUBE-2.

j. For $n = 5000$ with single precision, we use $N = 250$ and $h = 20$. The sequential time is 21890 milliseconds. Since the number of operations for the sequential algorithm is $n^2 w$, we estimate $w = 0.8756\mu s$. For nCUBE-2, $s = 160\mu s$ and $t = 0.6\mu s$ per byte [Dun91]. In Fig. 7, we plot the predicted performance by using these parameters in the above formula and compare it to the actual performance in Fig.7. The predicted performance is close to the actual parallel time in nCUBE-2. We list the speedup for $n = 5000$ single precision and $n = 10000$ with double precision[2] in the following table.

#processor	8	16	32	64
$n = 5000$, $N = 250$	6.3	12.0	19.8	28.3
$n = 10000$, $N = 500$	6.8	13.5	24.5	39.3

6.2 Matrix multiplication

A matrix multiplication ($C = A \times B$) program is listed below. A is a $n \times n$ matrix. B is a $n \times n$ lower triangular matrix. We divide the matrix B into n rows, the sizes of rows $B_1, B_2, \cdots, B_n$ are $1, 2, \cdots n$ respectively. We also divide the matrices A and C in a row-wise manner, see Fig. 8a. Let $h = \lceil \frac{n}{P} \rceil$.

[2]Due to the limit in the memory size, we use the test data in which the elements of matrix A are equal, thus only one submatrix $A_{i,j}$ is stored. This should not affect the performance data.

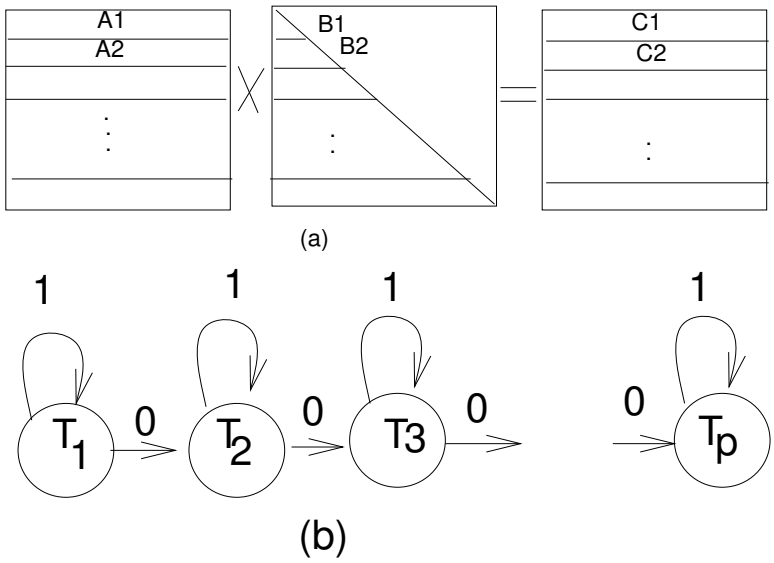

Figure 8: (a) The matrix multiplication problem: $C = A \times B$. B is a lower triangular matrix. (b) A dependence graph for matrix multiplication.

```
For k = 0 to n − 1
   for q= 1 to P
T_q^k: if q = 1 Generate row B_{k+1}; endif
      for i = h ∗ (q − 1) + 1 to min(h ∗ q, n)
         for j = 1 to k + 1
            c_{i,j} = c_{i,j} + a_{i,k+1} ∗ b_{k+1,j};
         endfor
      endfor
   endfor
endfor
```

The dependence graph is shown in Fig. 8(b). Each task T_q^k uses row B_{k+1} and row A_i to update row C_i where $r(q-1)+1 \leq i \leq rq$. This program allows the matrix B generated on the fly, so that the program can release the space of row B_{k+1} immediately after it is used by every task T_q^k. This is useful when processors of a parallel machine do not have a sufficient space to store the entire matrix B. Task T_1^k is responsible to generate row B_{k+1} to be used by tasks T_q^k where $1 \leq q \leq P$. Since row B_{k+1} has $(k+1)$ nonzero elements, the weight of task T_q^k is $(k+1)2wh$ $(2 \leq q \leq P)$, where w is the time for a multiplication or subtraction. We assume the time for generating a row B_{k+1} costs $(k+1)w/2$. The weight of T_1^k is $(k+1)2wh+(k+1)w/2$. We also assume that broadcasting primitive is not used, task T_1^k will send row B_{k+1} to T_2^k, T_2^k sends to T_3^k and so on. The communication cost for sending B_{k+1} is $s+(k+1)t$. In nCUBE-2, we use the gray-code method to embed a hypercube to a linear processor array.

The weight functions are monotonically increasing. We execute this graph on P processors. Using the solution in Section 4, we can determine the parallel time as

$$PT = (P-1)(s+t+2wh) + w/2 + (2wn + (P-1)t + w)(n-1)^2 + (hw + w/4)(n-1)^2 + 2whn.$$

$$Seq = \sum_{k=0}^{n-1}((k+1)w/2 + \sum_{i=1}^{q}(k+1)2hw) = \frac{n(n+1)}{4}w + n^2(n+1)w.$$

When n is large, the speedup approaches to P. We examine the actual performance of this schedule on nCUBE-2 machine for $n = 500$ and $n = 1000$ with single precision. Again the predicted and actual performance are very close and the result is listed in Fig. 9.

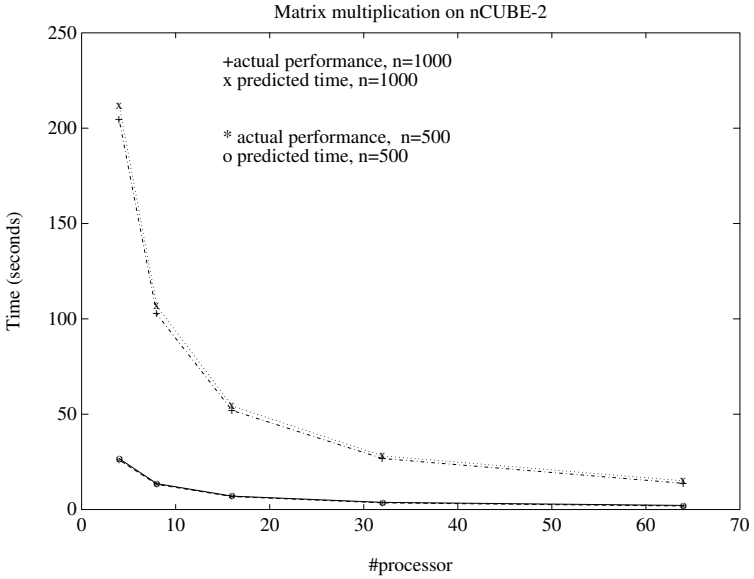

Figure 9: The performance of a matrix multiplication on nCUBE-2 for $n = 500$ and $n = 1000$.

The speedup of this implementation on nCUBE-2 when $n = 500$ and $n = 1000$ is listed below.

#processor	4	8	16	32	64
$n = 500$	4	8	15.45	29.9	56.4
$n = 1000$	4	8	16	31.4	61.2

7 Related Work

This work is related to the DAG scheduling for task parallelism on message-passing machines and software pipelining on instruction-level parallelism. DAG is the fundamental representation of task computation and the expanded graph in our model is a DAG. Exploring task parallelism through the asynchronous execution of task computation in message-passing machines has been studied in many DAG scheduling algorithms and systems, for example, El-Rewini and Lewis [EL90], Gerasoulis and Yang [GY93a, YG92], Kim and Browne [KB88], Sarkar [S89a], Shirazi, Wang and Pathak [SWP90], Wolski and Feo [WF93], Wu and Gajski [WG88]. The common model is the macro-dataflow task computation model, the execution of task computation is triggered when the data items are available in the local memory. We use the same model of computation but do not search the entire expanded task graph to produce a schedule, which is suitable when task graphs are

executed iteratively. Gerasoulis and Yang [GY93a] proposed a definition of granularity to analyze the impact of granularity on DAG scheduling. We use a similar definition to quantify the granularity of an ITG.

The earliest work on software pipelining has been Reiter [REI68]. Lam [LAM88] developed pipelining algorithms for compiling on VLIW machines. Aiken and Nicolau [AN88] proposed an algorithm that unrolls a loop and finds an optimal schedule. Optimal loop unrolling has also been studied by Parhi and Messerschmitt [PM91] and Chao and Sha [CS93]. Zaky and Sadayappan [ZS89] formulated the scheduling problem using a path algebra to find the optimal solutions. Munshi and Simons [MS90] examined the complexity of instruction scheduling and proposed heuristic solutions. Van Dongen, Gao and Ning [VGN92] proposed the scheduling solution on an unbounded number of processors. Gasperoni and Schweigelshoh [GS92] proposed scheduling algorithms for a bounded and unbounded number of processors. We extend the above work to consider the scheduling of coarse-grain task parallelism with nonzero communication. For exploring instruction-level parallelism on a fine-grained architecture, a fixed weight for each instruction can be assumed, operations of processors usually can be globally synchronized. As a result, a periodic schedule with a fixed iteration interval can be derived. For a message-passing architecture, global synchronization is expensive and coarse grain partitioning is usually desired since message communication suffers significant cost. Thus a task executes a chunk of instructions, loop, or subroutines. Computation weights vary during iterations and we may not be able to derive a fixed iteration interval. Since communication overhead affects the performance significantly, it is needed to overlap communication with computation to hide network communication latency.

8 Concluding Remarks

We have presented a schedule for mapping weighted ITGs on message-passing machines and an analysis on the asymptotic performance of this schedule. The task computation in this schedule is synchronized only between pairs of tasks through message communication to ensure the data dependence. Thus the execution of this schedule on a target machine is asynchronous and computation and communication are overlapped. The performance of such scheduling is affected by the graph granularity and weight variation between tasks in the course of iterations.

The scheduling solution is simple since each processor executes instances of one task, it is relatively easy to be implemented using techniques for executing a DAG schedule in [YG92]. The analytic result could be used to evaluate the asymptotic performance of parallel execution. Notice that when the loop bound of an ITG is unknown and coefficients of weight functions are constants, we are still able to conduct the performance evaluation since expressions for estimating the speedup and performance bound described in section 4.2 are independent from the loop bound. This paper assumes that the number of processors is equal to the number of tasks within the loop body and unrolling loop requires an additional number of processors. We still need to investigate the solutions for mapping iterative tasks on a limited number of processors and consider more complicated cost

functions.

Acknowledgments

This work was supported in part by a startup fund from University of California at Santa Barbara. We would like to thank Apostolos Gerasoulis, Oscar Ibarra and Vivek Sarkar for many valuable discussions and suggestions. We are grateful to Apostolos Gerasoulis and Rutgers University for providing us an access to their nCUBE-2 machine.

References

[AN88] A. Aiken and A. Nicolau, Optimal Loop Parallelization, *SIGPLAN 88 Conf. on Programming Language Design and Implementation.* pp.308-317.

[CS93] L. F. Chao and E. H. Sha, Unified static scheduling on various models. *Proc. ICPP 1993*, Vol II, pp 231-235.

[Ch89] P. Chretienne, Task Scheduling over Distributed Memory Machines, *Proc. of Inter. Workshop on Parallel and Distributed Algorithms*, (North Holland, Ed.), 1989.

[Dun91] T. H. Dunigan, Performance of the INTEL iPSC/860 and nCUBE 6400 Hypercube, ORNL/TM-11790, Oak Ridge National Lab., TN, 1991.

[EL90] H. El-Rewini and T. G. Lewis, Scheduling Parallel Program Tasks onto Arbitrary Target Machines, *Journal of Parallel and Distributed Computing*, Vol. 9, 1990, pp. 138-153.

[GS92] F. Gasperoni and U. Schweigelshoh Scheduling Loops on Parallel Processors: A simple algorithm with close to optimum performance. *Proc. of CONPAR 92* , pp. 613-624.

[GY93a] A. Gerasoulis and T. Yang, On the Granularity and Clustering of Directed Acyclic Task Graphs, *IEEE Trans. on Parallel and Distributed Systems.*, Vol. 4, no. 6, June 1993, pp 686-701.

[K94] V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to Parallel Computing*, Benjamin/Cummings Pub, 1994.

[LS81] C. E. Leiserson and J. B. Saxes, Optimizing synchronous systems, IEEE 22nd Annual Symp. Foundations of Computer Science, Oct 1981. pp. 23-35.

[LAM88] M. Lam, Software pipelining: an effective scheduling technique for VLIW machines, *Proc. of SIGPLAN 1988 Conf. on Programming Language Design and Implementation,* ACM, June 1988, pp.318-328.

[KB88] S.J. Kim and J.C. Browne, A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures, *Proc. of Inter. Conf. on Parallel Processing*, Vol. 3, 1988, pp. 1-8.

[PAM93] S. S. Pande, D. P. Agrawal and J. Mauney, A scalable scheduling scheme for functional parallelism on distributed memory multiprocessor systems, Tech. Report, North Carolina State Univ. 1993.

[PY90] C. Papadimitriou and M. Yannakakis, Towards on an Architecture-Independent Analysis of Parallel Algorithms, *SIAM J. Comput.*, Vol. 19, 1990, pp. 322-328.

[MS90] A. Munshi and B. Simons, Scheduling loops on processors: algorithms and complexity, *SIAM J. of Computing,* Vol 19, 1990, pp.728–741.

[PM91] K. K. Parhi and D. G. Messerschmitt, Static rate-optimal scheduling of iterative dataflow programs via optimum unfolding, *IEEE Trans. on Computers*, 40:2, 1991, pp. 178-195.

[REI68] R. Reiter, Scheduling parallel computations, *Journal of ACM*, Oct 1968, pp. 590-599.

[S89a] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, MIT Press, 1989.

[S89b] V. Sarkar, Determining average program execution times and their variance, *Proc. of 1989 SIGPLAN*, ACM, pp. 298-312.

[SWP90] B. Shirazi, M Wang, G. Pathak, Analysis and evaluation of heuristic methods for static task scheduling. *J. of Parallel and Distributed Computing*, Nov. 1990, vol.10, (no.3), pp.222-232.

[VGN92] V. H. Van Dongen, G. R. Gao and Q. Ning A polynomial time method for optimal software pipelining. *Proc. of CONPAR 92,* pp. 613-624.

[W89] M. Wolfe, *Optimizing Supercompilers for Supercomputers,* MIT Press, Boston, 1989.

[WF93] R. Wolski and J. Feo, Program Partitoning for NUMA Multiprocessor Computer Systems, Tech. Report, Lawrence Livermore Nat. Lab., 1992. *J. of Parallel and Distributed Computing*, 1993.

[WG88] M. Y. Wu and D. Gajski, Hypertool: A programming aid for message-passing systems, *IEEE Trans. on Parallel and Distributed Systems,* vol. 1, no. 3, pp.330-343, 1990.

[ZS89] A. Zaky and P. Sadayappan, Optimal static scheduling of sequential loops on multiprocessors. *Proc. of ICPP 1989*, Vol 3, 130-137

[YG91] T. Yang and A. Gerasoulis, A fast static scheduling algorithm for DAGs on an unbounded number of processors, *Proc. of Supercomputing '91*, IEEE, pp. 633-642.

[YG92] T. Yang and A. Gerasoulis, PYRROS: Static Task Scheduling and Code Generation for Message-Passing Multiprocessors, *Proc. of 6th ACM Inter. Confer. on Supercomputing*, Washington D.C., 1992, pp. 428-437.

Appendix

A Procedure $LPATH$

The basic procedure $LPATH$ used in the algorithm is described below.

- $LPATH(F, v, r, b)$. The input is a set of inequalities F with forms

$$f_j - f_i + d_{i,j} * r \geq w_{i,j}$$

where j and i are integers from 1 to v. $w_{i,j}$ and b are nonnegative constants. $d_{i,j}$ are nonnegative integer constants. The output is the nonnegative values for f_i $i = 1, \cdots v$, and a value for r such that $r \geq b$.

The solution of above inequalities can be solved by any single-source shortest path algorithm, an approach used in systolic network retiming [LS81]. Notice that in our case, the solution may not be integer, and also we impose a lower bound for r. The above inequalities are transformed as:

$$f_j - f_i \geq w_{i,j} - d_{i,j} * r.$$

We construct a directed graph L of v nodes. Each inequality in F corresponds an edge in L. The weight of this edge is $w_{i,j} - d_{i,j} * r$. Notice that r is to be solved. If L is a DAG, then then we set $r = b$. We use the topological searching to derive the solution.

In a cyclic graph, for each cycle C we add weights of edges in this cycle together.

$$0 \geq \sum_{(i,j) \in C} w_{i,j} - d_{i,j} * r, \qquad \text{thus } r \geq \frac{\sum_{(i,j) \in C} w_{i,j}}{\sum_{(i,j) \in C} d_{i,j}}.$$

The minimum value of r that satisfies this condition is

$$R = \max_{\text{all cycles C } C} \frac{\sum_{(i,j) \in C} w_{i,j}}{\sum_{(i,j) \in C} d_{i,j}}.$$

We are not able to enumerate all cycles. But given a value for r, we can use the shortest path algorithm to determine if there is a solution. Note that the input inequalities we use guarantee that $\sum_{(i,j) \in C} d_{i,j} \geq 1$. Thus $R \leq \sum w_{i,j}$. Let $Q = \sum w_{i,j}$. We can use a binary searching to find the value R for r with an accuracy of ϵ in a time complexity of $v|F| \log \frac{Q}{\epsilon}$. r satisfies that $R \leq r \leq R + \epsilon$.