OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

Piece-Wise Scheduling of Composite Task Graphs onto Distributed Memory Parallel Computers

T. G. Lewis
Computer Science Department
Oregon State University
Corvallis, OR 97331-3202

91-60-15

Piece-Wise Scheduling of Composite Task Graphs onto Distributed Memory Parallel Computers

T. G. Lewis
Computer Science Dept.
Oregon State University
Corvallis, OR. 97331-3202
lewis@cs.orst.edu
(503)-737-5577

Abstract

Heuristics for static scheduling of task graphs using list scheduling techniques have continued to improve by adding real-world factors such as processor speed, network transmission speed, interconnection topology, and link contention considerations to the basic task graph model. Yet, the resulting schedules do not fully model program loops and branches, startup costs for both process creation and message initiation, and a number of interesting parallel processing patterns such as meshes, tress, and supervisor/workers. In fact, improvements in the schedule may be obtained when the task graph is regular as when it contains repeated or replicated tasks, divide-and-conquer patterns of communication, or a mesh-structured pattern of computation.

In this paper we describe a limited approach to scheduling composite task graphs that considers process and message startup costs, and three regular patterns : replicated, tree, and mesh. The approach is to model programs with such regular patterns as a composite task graph, where each regular structure is a decomposable sub-task node in the task graph. Then, we compute an optimal schedule for each sub-task graph, piece the sub-tasks together, and perform an ordinary static scheduling heuristic on the pieces, to produce an overall schedule.

We define a composite task graph as a hierarchical task graph containing regular-structured sub-task graphs as components. At the top level of this hierarchy, each graph node represents either a simple task or a hierarchically decomposable sub-task graph. We propose a piece-wise scheduling algorithm that simply allocates processors to sub-task graphs according to closed-form expressions which give determine the optimal number of processors, and then uses a list scheduling algorithm to schedule the flattened graph onto these processors.

We do not address the pressing problem of loops and branches in the task graph representation, but we speculate that the technique of piece-wise scheduling introduced here can be adapted to a hybrid form of scheduling that may accommodate branches and loops.

Piece-wise scheduling is not guaranteed to yield the best global schedule. Rather, it pieces together locally optimum sub-schedules. Finding globally optimum schedules for composite task graphs remains an open problem. We present an heuristic approach that has been experimentally used to schedule small parallel programs with encouraging results. More empirical evidence is needed to determine the usefulness of this technique, but early indications are encouraging.

Introduction

The problem of scheduling parallel program tasks onto distributed-memory parallel computers has received considerable attention in recent years (see bibliography). This problem is known to be computationally intensive. The complexity of the problem rises even further when real-world factors such as process initiation time, message initiation time, transmission delay time, and time-complexity of the problem being solved are included in the analysis. Regardless, many researchers have studied restricted forms of the problem by constraining the task graph representing the parallel program or the parallel system model. Most investigators make simplifying assumptions to reduce the complexity of the problem. These assumptions ignore most of the critically important features needed to model modern parallel processor systems.

Recent *static scheduling* heuristics have been proposed to handle link contention, processor topology (some communication time delays are no longer sensitive to source-destination node distances created due to interconnection topologies, but some are), and transmission speeds of the network. Even so, many pressing problems remain in the literature: methods to handle message and process initiation costs, data parallelism (task replication), and program branches and loops. In this paper, we address the problem of statically scheduling regular task graphs onto a fully connected, distributed-memory parallel processor when message and process initiation times are included. We do *not* address the problem of program branches and loops, nor do we consider other factors such as network topology and link contention which have received attention elsewhere [ArRo91, Bala89, BeBa90, BlWe84, BlCh91, Carr90, Coff76, Cole91, DJLe89, ErSa89, HoSh88, JBGh90, Kamp89, Kant89, Lang84, Lee91, LSTa90, Robe89, Seth76, Ullm75].

A parallel program schedule must perform both a mapping of tasks onto processors, and an ordering, in time, of task executions. In the final analysis, it may be necessary to devise new scheduling *heuristics*, as opposed to optimum algorithms, that consider: 1) tasks taking a sizeable amount of time to be initiated and variable execution time given by a computational complexity formula, and 2) communication links requiring a significant message-passing startup time and taking a variable amount of communication time to pass data from one task to another.

Niether heuristic nor algorithmic solutions to scheduling programs that consider such factors have been proposed. This paper attempts a purely analytical analysis of such task graphs, to produce an analytically derived schedule. Our analytical analysis combines mapping with sequencing to form a schedule. We have simplified the analysis by assuming a fully connected interconnection network, and homogeneous processors. The results can be generalized to these more realistic conditions, but we have not attempted such a general solution, here.

In addition, our appoach does not consider the possibility of a global minimum time solution, but instead gives locally minimum solutions. That is, each analytical schedule is designed to minimize the completion time of a sub-task graph. When the schedule of a sub-task graph is combined with other sub-tasks to form a larger task graph for the complete parallel program, the combined schedules are not guaranteed to give a globally minimum execution time. That is, the solutions are only piece-wise minimal, hence we call these *piece-wise schedules*.We show one method of combining piece-wise schedules into a larger schedule, and describe some remaining problems with this approach.

Recent surveys and classifications of scheduling are given in [CaKu88, Chen90, WaCh91]. PPSE (Parallel Programming Support Environment) was inspired by early work done by Snyder (POKER), Berman (Prep-P), Wu (HYPERTOOL), Purtilo (Polylith), and others. This paper is an extension of on-going work on PPSE initiated by the author in 1988. PPSE most nearly resembles HYPERTOOL developed by Gajski and students at University of California, Irvine, and reported by Wu and Kwan. While independently developed both HYPERTOOL and PPSE suffer from lack of consideration of several real-world factors mentioned above. The mapping heuristics of Lo and Berman are related to this work, but only in that both approaches exploit task graph regularity, see Lo, Bokhari, Cybenco, McCreary, Muhlenbein, Razouk, et al., in the bibliography.

HYPERTOOL and PPSE incorporate static list scheduling techniques based on critical path analysis. HYPERTOOL takes as its input a C program and constructs a task graph that is analyzed and then

scheduled onto a hypercube. PPSE takes as its input a *design* in the form of a hierarchical dataflow graph, and schedules a flattened equivalent of the dataflow graph onto an arbitrary interconnected target machine. HYPERTOOL requires that the program be written before it can be analyzed, while PPSE does not. But, both tools automate much of the production of synchronization and message-passing code.

The PPSE approach is not as closely related to dynamic scheduling work done by a number of others such as [Cyben89, Feo86, Fost91, Gree88, JiJe90, MuSi90, PoKu87, SCMi90, VeDa90, XHYu90]. While it is possible to adapt some of these dynamic scheduling heuristics to our static heuristics to arrive at a hybrid method, our work is predominantly aimed at static scheduling. Furthermore, our work has little to do with mapping algorithms [BeSn87, Bokh81b, ChAg90, ChSh86, Efe82, Erca88, KPVe86, LeAg87, Lo84, LRGK90, MoSk90, NiKi89, SaEr87, SLHH91, ShFo90]. These are important algorithms, but scheduling differs from mapping mainly in the way tasks are scheduled to execute in a certain order. Concern for the order of execution increases the complexity of the algorithms. Finally, our approach is fundamentally aimed at practical considerations, which means we are primarily concerned with the impact of target machine factors on schedules.

The Model

We seek a unified model that considers real world factors in both the target machine and the parallel program. For distributed-memory *target* machines these parameters are: interconnection topology, processor and communication link characteristics, and message/processor startup costs. For parallel *programs* these parameters are: size of problem, computational complexity of the tasks, conditional branching, and loops.

Target Machine Model

The target machine can be described as a general system (P, $[P_{ij}]$, $[S_i]$, $[I_i]$, $[B_i]$,$[R_{ij}]$) as follows:

1. $P = \{ P_1, ..., P_m \}$ is a set of processors forming the parallel machine.
2. $[P_{ij}]$ is an mxm interconnection topology matrix.
3. S_i. $1 \le i \le m$, specifies the speed of processor p_i.
4. I_i. $1 \le i \le m$, specifies the startup cost of initiating a message on processor p_i.
5. B_i. $1 \le i \le m$, specifies the startup cost of initiating a process on processor p_i.
6. R_{ij} is the transmission rate over the link connecting two adjacent processors p_i and p_j.

This is a general model. For most of this paper, we will simplify this model by assuming a uniformity across the parallel computer system. Processors will all operate at the same speed, and message-passing parameters are equal on all links. Thus, the simplified model becomes:

1. P = number of identical processors, numbered from zero to P-1.
2. The mxm interconnection topology is assumed to be a complete graph.
3. All processors run at the same speed, denoted by Parameter α.
4. Parameter a = the startup cost of initiating a message on all processors.
5. Parameter β = the startup cost of initiating a process on all processors.
6. Parameter b = the time to transmit a unit of data across all links.

Parallel Program Task Model

The task system for a given set of resources can be defined as the system (T, <, $[D_{ij}]$, $[F_i]$) as follows:

1. $T = \{ T_1, ..., T_n \}$ is a set of tasks to be executed.
2. < is a partial order defined on T which specifies operational precedence constraints. That is $T_i < T_j$ signifies that T_i must be completed before T_j can begin.
3. $[D_{ij}]$ is an n x n matrix of communication data, where $D_{ij} > 0$ is the amount of data required to be transmitted from task T_i to task T_j, $1 \le i,j \le n$.

4. $[F_i]$ is an n vector of the amount of computations, where $F_i > 0$ is the number of instructions required to execute T_i, $1 \le i \le n$. The elements of this vector are functions which give the computational complexity of each task's execution time as a function of inputs, D_{ij}. In many cases, all execution times are alike, so we will use the simple form, F.

Execution and Communication Cost

The following parameters are required to represent the computational costs and communication costs incurred by a parallel program on a specific parallel processing system.

1. T_i : the execution time of task i when executed on a processor. It should consider the size of the tasks F_i, and process startup time, β. In general, $T_{ij} = F_i + \beta$

In most of the following analysis, this model is further simplified by making F_i a function of the computational complexity of the task. For example, $F_i(S,k)$ might be used to model the movement of S bytes of data to k replicated tasks.

2. $C(i_1,i_2,j_1,j_2)$: the communication delay between tasks i_1 and i_2 when they are executed on processing elements j_1 and j_2, respectively. It reflects the target machine performance parameters as well as the size of the data to be transmitted.

When the I/O processors take the same amount of time to initiate a message and the transmission rate is the same over the interconnection network, the formula for communication cost is

$$C(i_1,i_2,j_1,j_2)=(\alpha+\beta D_{i_1i_2})*H_{j_1j_2}$$

where H_{ij} is the number of hops between processing elements i and j. If we further assume that all links take equal time, (hops are ignored) then $H_{ij} = 1$, and the formula simplifies to

$$C(i_1,i_2,j_1,j_2) = \alpha+\beta D_{i_1i_2}$$

Our goal is to schedule a task graph like the one shown in Figure 1, such that the parallel program executes in the shortest elapsed time. Where it is appropriate, the task graph is annotated with a node number or formula specifying the task's identification and/or execution time, and size of data to be passed in a message (arcs). The static schedule is given in the form of a Gantt chart, Figure 1. The Gantt chart will be drawn so that time progresses from left-to-right and processor numbers progress from top-to-bottom.

Parallel programs are fully modelled as a task graph that can be partitioned into sub-task graphs, where each sub-task graph is a regular structure such as a fan, tree, or mesh. The partitioned sub-task graphs are scheduled independently by the algorithms presented in the following sections of this paper. Then, the schedules for the sub-task graphs are pieced together to form the full schedule. This approach is called *piece-wise scheduling* for obvious reasons. While the schedules of each sub-task graph may be optimal, the global schedule of the entire graph is not guaranteed to result in minimum execution times. Hence, this approach cannot guarantee the best overall schedule. However, this technique is useful for large programs where the pattern of parallelism may differ in different parts of the program, and there are more tasks than processors during much of the program's execution.

First, we derive analytical formulas for three regular task graphs: fan, tree, and mesh. Then, we show how to use piece-wise scheduling to derive a global schedule for larger task graphs containing fan, tree, and mesh sub-task graphs as components. The goal is to produce the shortest execution time by piece-wise scheduling of optimal sub-task graphs.

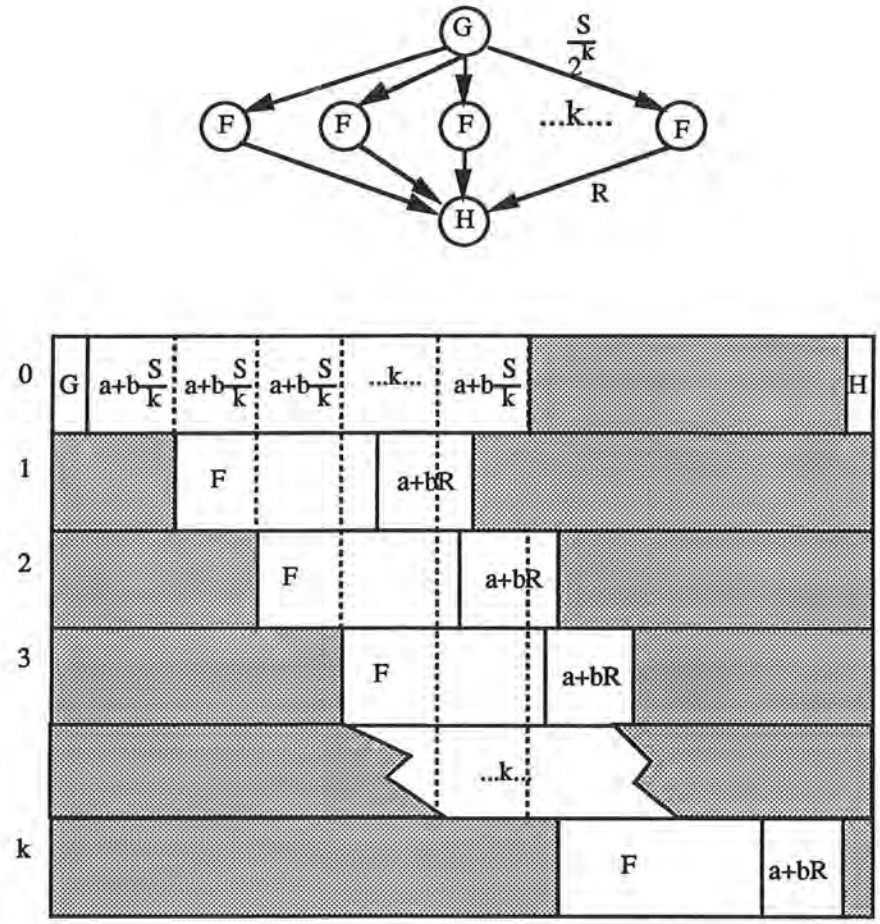


Figure 1. The Replicate Task Graph, and its Schedule as a Gantt Chart

Replicates

Perhaps the most common pattern of parallel computation is the supervisor/worker pattern shown in Figure 1. The supervisor task is modelled by subtasks G+H, where G distributes equal-sized data partitions to identical processing functions represented by the worker tasks labelled F. Such a pattern is used, for example, to search a list in parallel by breaking up the list into equal-length sub-lists, searching each sub-list in parallel, and then returning a result R to the supervisor, who performs a sequential search for the exact match. Other common operations of finding the minimum, maximum, sum, average, and so forth can be solved with this simple pattern. [This is not the only pattern that can be used to solve these kinds of problems].

The identical functions F accept input of size $\frac{S}{k}$, where we assume k replicated workers, list of size S, and even distribution of data across all workers. After a period of parallel computation, the workers each return a value of size R to the supervisor, who then completes the computation.

Suppose the execution time of G, F, and H are given as a function of the size of their inputs. Specifically, suppose G and H execute in some constant time, and F(x) is a function of the computational complexity of each worker subtask, where x is the size of the input to a worker. Finally, assume the returned value from each worker is of size $R << S$.

The Gantt chart of Figure 1 gives a possible schedule for this pattern. We assume a linear model for communication, a+bx, where a is the time to initiate a message, b is the time to transmit a unit of message, and x is the number of units in the message. The Gantt chart shows the supervisor as mapping onto processor zero, and workers 1 through k mapping onto processors 1 through k. [A slightly tighter schedule might be had if the first worker is mapped onto processor zero, also, but this depends on the amount of communication overhead].

Communication time is charged to the sender, who we assume, can only send one message at a time. The receiver is assumed to buffer all incoming messages, so they are available as soon as requested. This assumption may not be valid, in general, but from the Gantt chart we observe that H must wait on only the last worker to finish before making access to all returned messages.

The question we pose is, "what is the best value of k, considering both computational complexity of each worker, and communication costs?" That is,

Minimize : T(S,k) ; where T = total elapsed execution time for the Gantt chart
Subject to : k = 0,1,2,...,min(S,P); where P=number of processors

From the Gantt chart of Figure 1 we get an expression for T(S,k) by substituting $x=\frac{S}{k}$ and collecting terms:

Minimize : $T(S,k) = G+k(a+b\frac{S}{k})+H+F(\frac{S}{k})+(a+bR) = C+ak+F(\frac{S}{k})$; where C=G+H+a+bR+bS

This is the well-known grain-size determination problem, where we seek to determine the best packing of worker tasks onto processors, such that the computation time is balanced with the communication time. In practice, this may require placement of many workers per processor. If too many tasks are allocated to a single processor, performance decreases due to loss of parallelism. If too few are allocated, performance may also diminish due to communication overhead (a variant of the min-max problem).

Setting the first derivative of T with respect to k to zero and solving for k produces an optimum value k^*. That is, solve the following equation for k^*.

$$a+\frac{\partial T}{\partial k} = 0 \; ; \; k^* \text{ in } [0, \min(S,P)]$$

Case I. Polynomial Complexity : $F(x) = x^n$; $n>0$

When the computational complexity of each worker is O(x), the derivative of $\frac{S}{k}$ is $(-\frac{S}{k^2})$, so we get a surprising result: $k^* = \min(\sqrt[2]{\frac{S}{a}}, P)$. In general, a polynomial-complex worker results in a schedule given by $k^* = \min(\sqrt[n+1]{\frac{nS^n}{a}}, P)$.

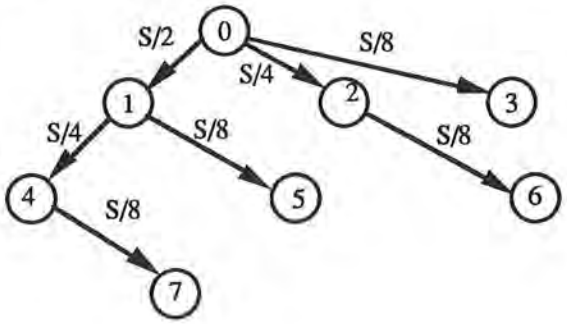
The interpretation of this result is straight-forward. The message sending startup cost can have a major impact on the optimum parallelism. However, as the complexity of each worker tasks execution time grows, communication startup time has a lessor influence. That is, as the computation to be performed increases, the grain-size increases, leading to greater benefits of parallelism.

This result also explains why relative speedup is sometimes disappointing for distributed-memory computers with high message startup costs. Using $\frac{T(S,0)}{T(S,k^*)}$ as our measure of relative speedup S_p, assuming we have an infinite number of processors, setting n=1 for simplicity, and substituting k^* :

$$S_p = \frac{C}{C+ak^*+\frac{S}{\sqrt{S/a}}} = \frac{C}{C+2\sqrt{aS}} \approx \frac{a+bR+bS}{a+bR+bS+2\sqrt{aS}} \quad \text{where } C=G+H+a+bR+bS$$

The last term above is obtained by assuming G=H=0. If S is dominant, the speedup behaves as $O(\sqrt{S})$, but high values of a and b can alter this approximation. Nonetheless, this interesting result suggests a speedup far from theoretical estimates that ignore communication costs and task initialization times (That is, G, H >0).

For $F(x) = \log_2(x)$, $k^* = 0$. A supervisor/worker pattern should not be used to parallelize a sub-polynomial algorithm. But, what is the value of k^* when $F(x) = x \log_2(x)$? It has been speculated that an $O(x \log(x))$ algorithm can be solved in O(x) time using parallel processors. We turn to this problem in the next section.



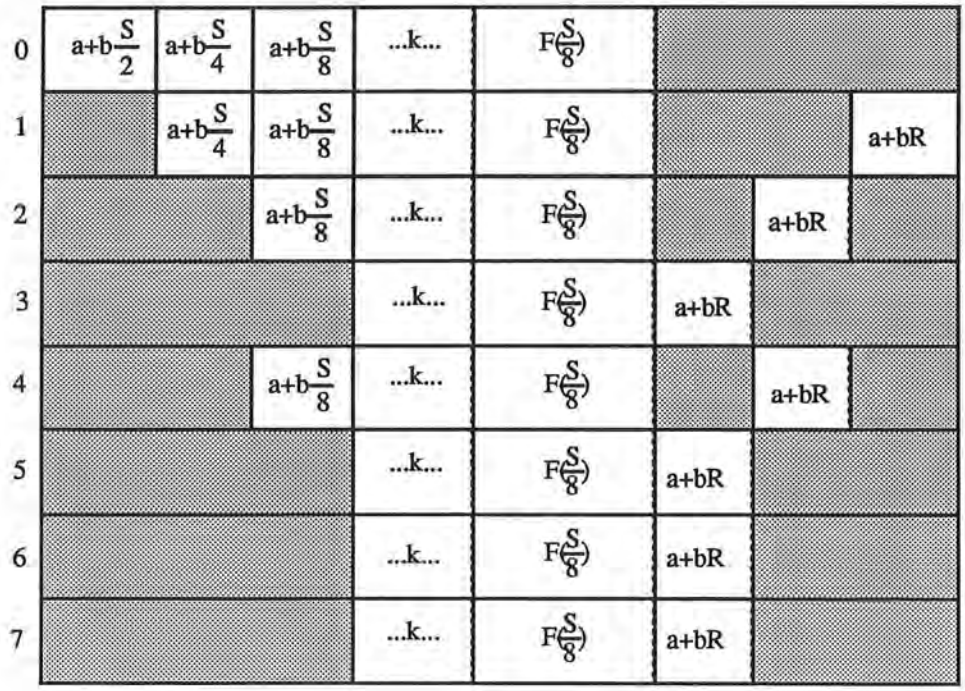


Figure 2. Divide-and-Conquer Task Graph and Gantt Chart Schedule

Divide-and-Conquer Patterns

Divide-and-conquer algorithms are very common in sequential programming, but take on a different character in parallel programming as shown in Figure 2. Here we see a tree-structured collection of workers performing a two-phase calculation: 1. data of size S is partitioned and distributed to two neighboring tasks, who in turn, partition and distribute their inputs of size S/2 to neighbors, and so forth, until some depth k is reached, and 2. the leaf nodes at level k take time F(x) to compute a result which is passed up a level to a parent task, which in turn computes a result and passes it up a level, and so forth, until the final answer reaches the root task.

A simple binary tree of tasks can be mapped onto a simple binary tree of processors, but this would be inefficient because all but the leaf processors would become idle while the leaf nodes become over burdened with work. Therefore, the mapping shown in Figure 2 is used whereby the root task shares a processor with one of its sibling tasks. This folding of tasks onto processors assures higher processor utilization, reduces communication delays, and in general improves performance of the divide-and-conquer algorithm.

In Figure 2, processor zero sends S/2 units of data to processor one, then performs a second task to divide the remaining S/2 units of data between processors two and three. Similarly, processor one sends S/4 units of data to processor four, and then performs a second task of dividing the remaining units of data between processor five, etc. Figure 2. shows three levels of division, but in general, any number of levels might be used. In fact, the question is, "what is the optimum number of levels, k^*, to obtain a minimum time divide-and-conquer algorithm?"

Minimize : T(S,k) ; where T = total elapsed execution time for the Gantt chart
Subject to : k = 0,1,2,...,min($\log_2(S)$,P); where P=number of processors

From the Gantt chart we can derive an expression for T(S,k), as follows. The total distribution phase communication time is $\left(\sum_{i=1}^{k} a+b\frac{S}{2^i}\right)$, and the total collection time is k(a+bR). All tasks compute at the same time, so only one $F(\frac{S}{2^k})$ is needed. We have assumed the collection time is zero on all processors, so the algorithm terminates as soon as all results are returned to task zero. Thus, the total elapsed time is $T(S,k) = \left(\sum_{i=1}^{k} a+b\frac{S}{2^i}\right) + k(a+bR) + F(\frac{S}{2^k})$. This expression simplifies to $bS-b(\frac{S}{2^k})+(2a+bR)k+F(\frac{S}{2^k})$.

When k=0, this parallel algorithm reduces to a serial algorithm running on one processor. When k=log(S), we would expect maximum speedup, assuming communication costs are ignored. That is, the idea speedup is of forder $\frac{O(F(S))}{O(S)}$.

$$S_p = \frac{T(S,0)}{T(S,\log(S))} = \frac{F(S)}{F(1)+(2a+bR)\log(S)+bS(\frac{S-1}{S})} \approx \frac{F(S)}{F(1)+(2a+bR)\log(S)+bS}$$

The last term is an approximation when $\frac{S-1}{S} \approx 1$, for large S. Furthermore, if we assume communication costs are negligible, a=b=0, and F(1) is negligible, so the ideal speedup is simply $\frac{O(F(S))}{O(S)}$. For example, an O(S) problem can be solved in time O(1), because F(S)=O(S), and a serial divide-and-conquer problem that can be solved in O(S log(S)), can be solved in parallel in time proportional to O(S). But, these estimates assume no communication costs.

Lets examine two realistic cases. First, suppose F(x) is linear as in the replicated pattern, and then suppose F(x) is O(x log(x)), and communication costs are considered, too.

Case I. Linear Algorithm : $F(\frac{S}{2^k}) = (\frac{S}{2^k})$.

Minimize : $T(S,k) = (\frac{S}{2^k}) - b(\frac{S}{2^k}) + (2a+bR)k + bS$

Subject to : $k = 0, 1, 2, \ldots, \min(\log(S), P)$.

Setting the derivative to zero and solving for k yields k^*, as before.

Solve : $\frac{\partial T}{\partial k} = 0 = \frac{(b-1)S(\ln 2)}{2^k} + (2a+bR)$; where ln2 is the natural logarithm of 2.

This equation has no solution within the constraints when B>1 and S>0, so $k^* = 0$. That is, the high cost of communication makes running a linear algorithm in the divide-and-conquer pattern run slower that its serial version. This is intuitive.

Case II. Quadratic Algorithm : $F(\frac{S}{2^k}) = (\frac{S}{2^k})^2$

Minimize : $T(S,k) = (\frac{S}{2^k})^2 - b(\frac{S}{2^k}) + (2a+bR)k + bS$

Subject to : $k = 0, 1, 2, \ldots, \min(\log(S), P)$.

Setting the derivative to zero and solving for k yields k^*, as before.

Solve : $\frac{\partial T}{\partial k} = 0 = \frac{bS(\ln 2)}{2^k} - \frac{2S^2(\ln 2)}{2^{2k}} + (2a+bR)$; where ln2 is the natural logarithm of 2.

When $S \neq 0$, a positive solution exists, $k^* = \log_2 S - \log_2 X$, where X is the solution to a quadratic equation: $X = \frac{b}{4} + (\frac{1}{2})\sqrt{\frac{b^2}{4} + 5.771a + 2.885bR}$. For example, suppose a=250 ms, b=10 ms/byte, R=10 bytes, and S=1000 bytes. Then $X \approx 23$, and $k^* \approx 5$. Without consideration of communication, the optimal value of $k^* \approx 10$ is twice the optimum with communication costs. Suppose b=0, then $k^* \approx 6$, which shows a small effect. Note that X = 0 when a=b=0: the result is valid only for a>0, b>0.

Case II. Logarithmic Algorithm : $F(\frac{S}{2^k}) = (\frac{S}{2^k}) \log(\frac{S}{2^k})$

An interesting case arises when attempting to match a divide-and-conquer algorithm to a divide-and-conquer schedule. Intuitively, the match is exact, but this is not the case when communication costs are included in the analysis.

Minimize : $T(S,k) = (\frac{S}{2^k})\log(\frac{S}{2^k}) - b(\frac{S}{2^k}) + (2a+bR)k + bS$

Subject to : $k = 0, 1, 2, \ldots, \min(\log(S), P)$.

Setting the derivative to zero and attempting to solve for k yields an intractable equation to solve in closed form:

Solve : $\frac{\partial T}{\partial k} = 0 = \frac{k - \log S - \log e + b}{2^k} + (2a+bR)$; where loge is the base-two logarithm of e.

The theoretical speedup can be approximated by setting k = logS, as before, and showing that communication costs play a critical role in even the predicted speedup:

$S_p = \frac{T(S,0)}{T(S,\log S)} = \frac{S\log S}{(2a+bR)\log S} = \frac{S}{(2a+bR)}$. Once again, if a=250ms, b=10ms/byte, R=10 bytes, and S = 1000, the speedup is reduced from its theoretical (linear) improvement by a factor of $\frac{600}{1000}(100) = 60\%$.

Figure 3 illustrates the sensitivity of k^* to communication cost parameters. The minimum points in these curves is where $k = k^*$. Notice how k^* quickly decreases with small increases in a,b, or R. The optimum level of the tree is very sensitive to small amounts of overhead in communication.

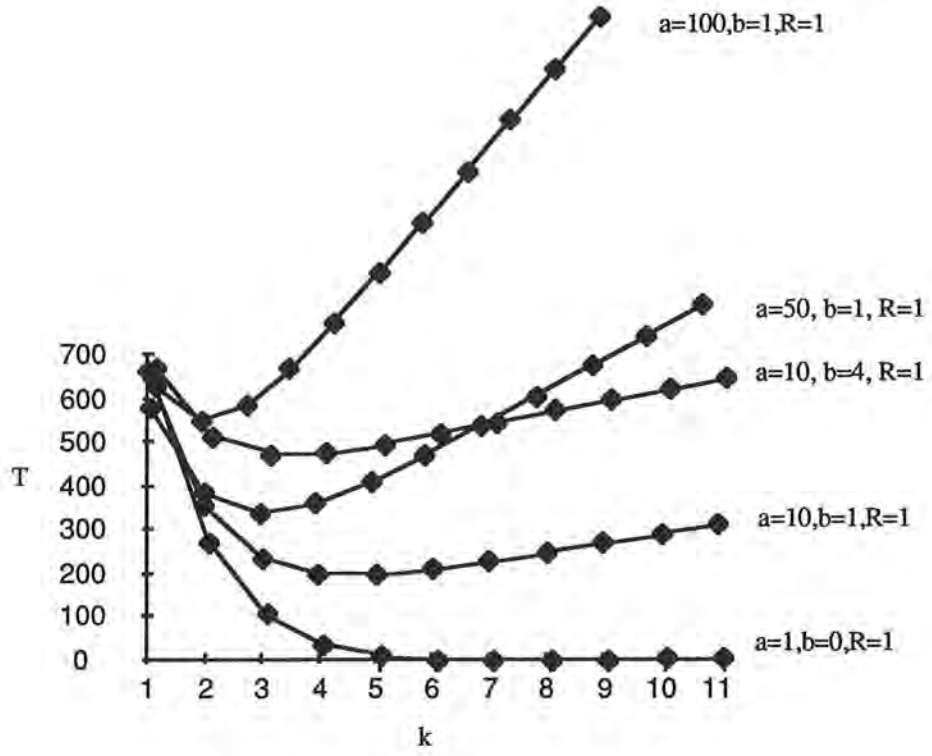


Figure 3. Time vs. Level, k for O(S logS) Algorithm Scheduled onto a Tree of Processors

Nearest Neighbor Meshes

Nearest neighbor calculations, typically done on a mesh of processors, is a regular pattern that might be sensitive to communication overhead, because every processor must communicate with one or more neighbors. The 5-point stencil of Figure 4 is an example. This pattern is used to solve the wave equation by successive overrelaxation techniques. The central grid point is replaced by the average of the four neighbors, N=North, E=East, W=West, and S=South. The 5-point stencil is sometimes called the NEWS pattern.

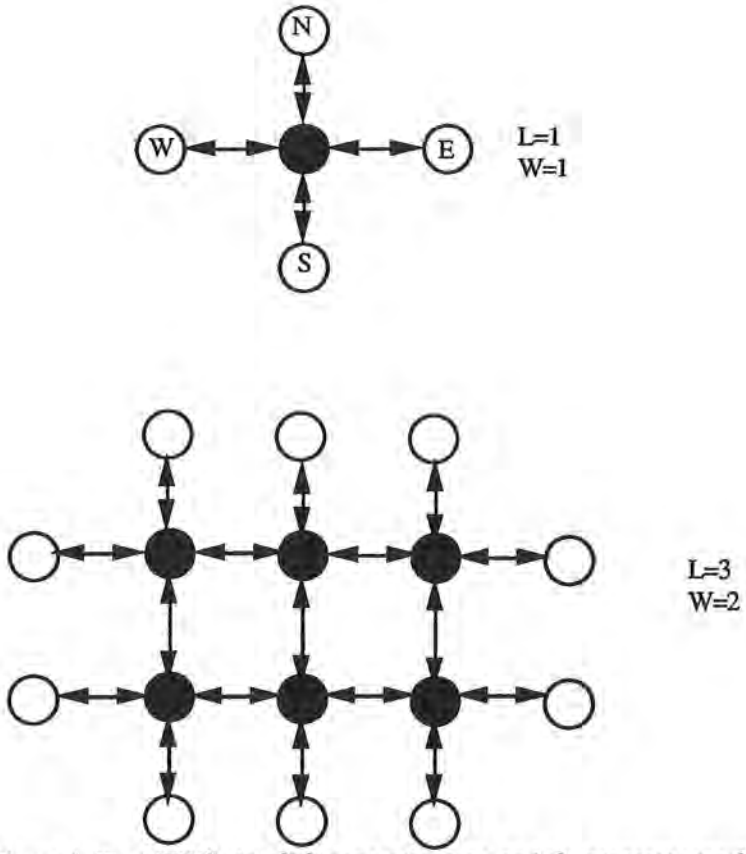


Figure 4. The NEWS stencil for one processor, and six processor mesh.

In Figure 4, the dark nodes perform calculations from data passed to them by the light nodes. Thus, in a single-node stencil, four messages are needed to do one calculation. However, in a NEWS mesh containing (2()(3)=6 calculation nodes, only 10 messages, or 1.4 messages per calculation are needed to do a calculation. As the grain size of the stencil increases, the number of messages per calculation decreases. This is due to the fact that communication occurs along the periphery, while calculation occurs across a "surface".

In general, a mesh of dimensions L by W takes 2(L+W) messages to perform LW calculations. Assuming a computational complexity at each node of F(x), where x is the size of the data passed to the calculation by incoming messages, the time to perform on "seep" of calculations across a surface of LW tasks is given by the formula:

$T(L,W) = F(LW)+2(L+W)(a+bR)$; where R is the size of the message passed along each link in the mesh.

Case I. Linear Algorithm : $F(LW) = \alpha LW+ \beta$, where α and β are constants for a particular system. The value of α relates to processor speed, and the value of β relates to process initiation time on each processor node.

Minimize : $T(L, W) = \alpha LW+\beta+4(L+W)(a+bR)$
Subject to : : $LW \leq P$, where P is the number of processors.

Suppose L=W=n, for simplicity. Then, $T(n,n) = \alpha n^2+\beta+4n(a+bR)$. The first derivative of this expression is always positive, hence T(n,n) is a monotonic increasing function in n. This means the optimum value occurs when n is as large as possible, e.g. $n = \sqrt[2]{P}$.

Relative speedup is defined as $\frac{T(n,n)}{T(1,1)}$, so $S_p = \frac{\alpha n^2+\beta+4n(a+bR)}{\alpha\frac{n^2}{P}+\beta+4(a+bR)\frac{n}{\sqrt[2]{P}}}$. We should consider two extreme cases:

Computational Intense: $4n(a+bR) << \alpha n^2+\beta$; $S_p \approx O(P)$. Alternately, when $4n(a+bR) >> \alpha n^2+\beta$ we say the algorithm is communicationally intense, and $S_p \approx O(\sqrt[2]{P})$. That is, even though communication grows more slowly than computation, communication costs can degrade performance by a factor of $\sqrt[2]{P}$.

This analysis clearly demonstrates why nearest neighbor mesh calculations perform especially well for small-grained algorithms. Scalability is not a major problem because most of the communication is overlapped. That is, communication as well as computation is performed in parallel. This is shown in Figure 5, which gives a schedule for the 5-point stencil.

	Communicate	Calculate
0	4(L+W)(a+bR)	αLW+β
1	4(L+W)(a+bR)	αLW+β
2	4(L+W)(a+bR)	αLW+β
	.....	
n-1	4(L+W)(a+bR)	αLW+β

Figure 5. Schedule for 5-Point Stencil Pattern

However, this analysis does not include the cost of data partitioning and distribution to the mesh. For example, we might use the divide-and-conquer pattern to distribute a matrix of data throughout the mesh prior to iterating the 5-point stencil. Once the iterations converge at each processor, we must collect the results back on a single (root) processor. This communication time must be included in a through analysis.

Assuming S = n2, where n is the dimension of a matrix containing 5-point stencils; R=0, and F(x)=0 for the divide-and-conquer communication delay. An analysis quite similar to the previous divide-and-conquer analysis can be performed to arrive at the following cost function to communication delay:

$T_{communication}(n,k) = ak + \frac{(2^k-1)}{(2^k)}n^2b$; where $2^k = P$ = number of processors involved in the mesh calculation.

Assuming a square matrix for simplicity, we can use the mesh formula for execution time, where I is the number of nearest neighbor iterations needed to converge to a solution, and the computational complexity at each node is assumed to be $F(x)=\alpha x^2+\beta$:

$$T_{calculation}\left(\frac{n}{\sqrt[2]{P}}, k\right) = I\left(\alpha\frac{n^2}{\sqrt[2]{P}}+\beta+4\frac{n}{\sqrt[2]{P}}(a+bR)\right)$$

Then, the total time is $2T_{communication}+T_{calculation}$, and the minimization problem is as follows.

Case I: Communication + Mesh Calculation.

Minimize : $2(\frac{(2^k-1)}{(2^k)}n^2b) + I(\alpha\frac{n^2}{\sqrt[2]{P}}+\beta+4\frac{n}{\sqrt[2]{P}}(a+bR))$

Subject to : $n > P$; $P = 2^k >> 1$ so that $\frac{P-1}{P} \approx 1$.

With the simplifying assumptions given above, we can re-write the objective function in a simplified form, with some loss of generality:

$T(n, P) = a\log_2(P)+bn^2+I(\alpha\frac{n2}{P}+\beta+\frac{n}{\sqrt[2]{P}}(a+bR))$ which produces the following equation to solve for P,

when setting the derivative to zero as before: $2a\log(e)P - (2nI(a+bR))\sqrt[2]{P} - \alpha n^2 I=0$. This equation is solved by substituting $P=Q^2$, and solving for Q by the quadratic formula. After a number of factorizations, the result can be shown to be $P = O(n^2)$. We omit the lengthy calculations here.

Nearest neighbor calculations are so efficient, they can easily absorb communication overhead costs, process and message-passing startup costs, and the effects of iteration. Communication costs dominate during the distribution phase, but have little effect on processing speed during the iterative phase. Thus, maximizing the size of the mesh pays off.

Composite Task Graphs

A composite task graph is a hierarchical task graph containing nodes and arcs as defined earlier, but with the added hierarchical structure shown in Figures 6,7, and 8. For example, in Figure 6, the top level of the hierarchical task graph shows parallel mesh and tree components. At level two, the mesh is expanded into a mesh structure, Figure 7, and the tree is expanded into a tree structure, Figure 8.

One approach to scheduling such hierarchies of task graphs is to proceed from the top, down. That is, schedule the top-level graph, assuming time and computation estimates as developed in the previous sections. Then, independently schedule the sub-task, e.g. the mesh followed by the tree. This method is simple, but not very accurate, because it cannot detect parallelism among siblings.

Another approach is to schedule from the bottom up. First, the lowest-level sub-tasks are scheduled, then the next, and so forth, until the top-level graph is scheduling using the information derived from lower level schedules. This has the advantage of making better use of the processors, but it does not recognize all the possibilities for parallelism.

A third, approach, which is used here, is to flatten the entire hierarchical graph into one large task graph, and use list scheduling techniques described in [ElLe90] and elsewhere. That is, we can transform the hierarchical graph into one that we know how to schedule. The algorithm is as follows:

1. Estimate the optimal number of processors needed for each sub-task according to the analytical formulas for each sub-task communication pattern.

2. Partition the available processors into groups, such that the parallel sub-tasks are allocated their "optimal" number of processors. {In general, this is a kind of bin packing problem that we leave as an exercise for the reader. The simple examples given here allocate the simple tasks and divide-and-conquer trees, first, then give all remaining processors to mesh calculations second. This, however, may not be the best partition].

3. Flatten the entire hierarchical composite task graph into a single level, using a depth-first traversal algorithm. For two-level graphs as described here, this means expansion of the sub-task graphs in place, while maintaining the intended graph topology.

4. Schedule the flattened static task graph using some list processing algorithm. We use the MH heuristic to schedule the flattened graph. MH considers nearest neighbors to minimize the number of hops across the network, as well as communication delays and task execution times.

A consequence of flattening and list scheduling is that the sub-tasks become merged with surrounding sub-tasks, and the uniformity of each sub-task graph is lost. This may be an advantage or a disadvantage. More work is needed to examine the effects of flattening on the globally optimum schedule. We have not attempted to answer this important question, here.

Figure 6 shows a simple composite task graph containing two parallel sub-tasks: a mesh and a divide-and-conquer tree. The sub-task graphs show the communication patterns of both distribution and collection of the results. Hence, they do not look as simple as a mesh or tree. Instead, they contain a mesh or tree embedded in a pattern of communication that first distributes the data to all tasks, followed by a collection pattern that collects data from all tasks. Tasks labelled S2, S4, S8, and ONE, R, are part of the distribution and collection patterns, respectively. In the sub-task graphs, "Param1" and "Param2" boxes show data input and output parameters for each of the sub-tasks, see Figures 7 and 8.

Tasks are scheduled by first estimating the optimal number of processors needed, then allocating processors while computing the best schedule using the bottom-up list scheduling approach. The result of scheduling the flattened graph obtained from Figures 6,7, and 8 is shown in Figure 9. The light colored part of each Gantt line shows tasks waiting for communication, and the light parts show duration of a task executing on a certain processor. The total elapsed time is 1413 time units, for the particular parameters used in this example.

The scheduling heuristics employed to obtain the Gantt chart of Figure 9 use the analytical formulas derived in the previous section to estimate the number of processors and length of time to perform a sub-task. These schedules are then adjusted to take advantage of earliest processor ready times and idle processors. Thus, the individual schedules can be compacted to reduce "empty spaces" between adjacent sub-task schedules.

[We simplified this example to conserve space: S=128, F takes 256 time units for the tree, and varies for the mesh over 180, 150, 120, and 100 units each, depending on the size of data block being processed. Tasks labelled as ONE, take 1 time unit each, because they simply pass data along the network. In the divide-and-conquer cases, S/2=64, S/4=32, S/8=16 were used along data dependency arcs.]

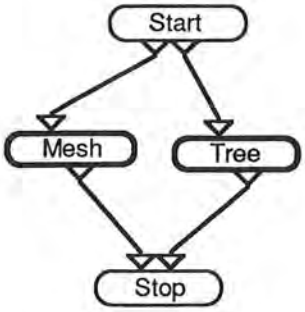


Figure 6. Example Task Graph Consisting of Parallel Tree and Mesh Sub-Task Graphs.

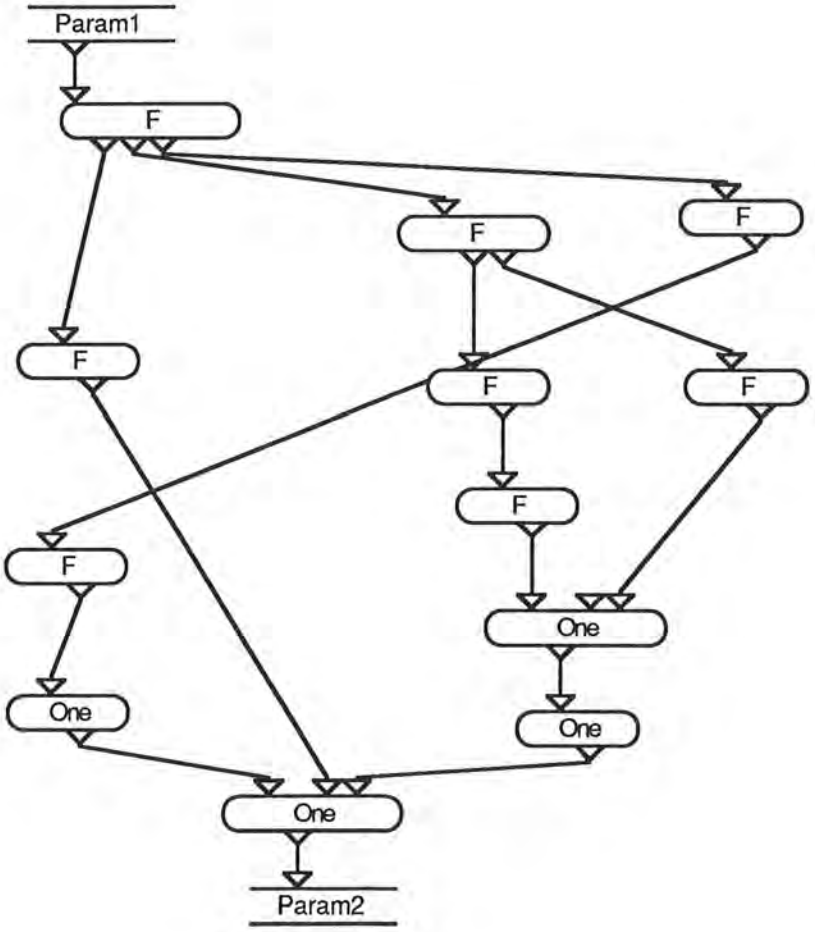


Figure 7. Mesh Sub-Task Graph Showing Communication Patterns, Only

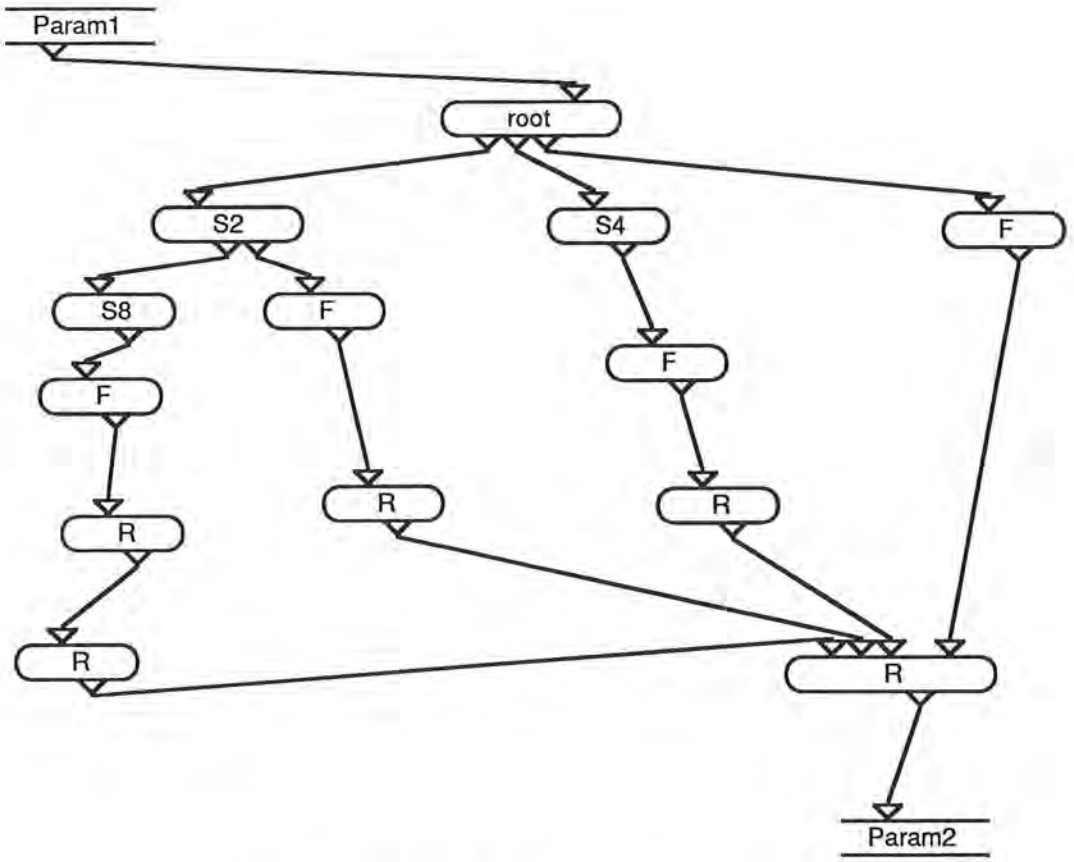


Figure 8. Tree Sub-Task Showing Communication Pattern.

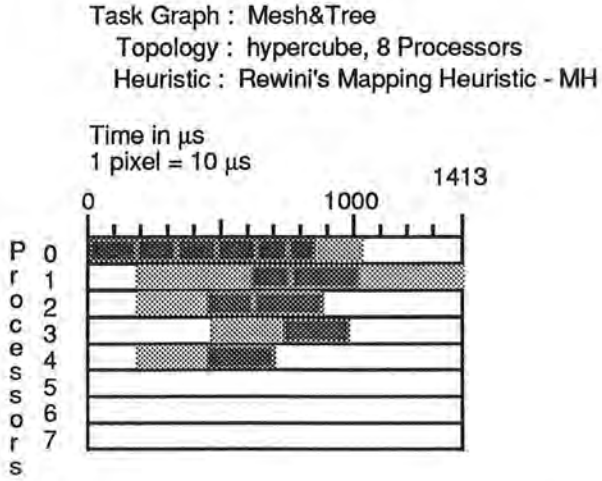


Figure 9. Gantt Chart Schedule For The Composite Task Graph of Figures 6, 7, and 8. We have used the MH algorithm and a hypercube topology consisting of 8 processors.

Conclusions

An automatic scheduling tool has been constructed for piece-wise scheduling of composite task graphs. Figures 6,7,8, and 9 illustrate the use of this tool on a small example. Further work is needed to extend the method to larger and more complex patterns of parallelism, and to calibrate parameters such as process and message startup times, processor speed, and transmission speeds of real machines. Also, further work is needed to study how to perform global optimizations.

Furthermore, this technique does not solve a number of lingering problems with static schedulers. The most pronounce is that of scheduling programs containing branches. We speculate that the piece-wise technique described here can be adapted to this problem as follows. The task graph is flattened as before, but at each branch in the program the flattened graph produces a tree of alternatives. Thus, for programs containing only two-way branches, the flattened graph becomes a binary tree of alternative flattened task graphs. The scheduler then produces a binary tree of Gantt charts, based on all paths through the program. Finally, the program dynamically selects the best schedule after each branch, and self-schedules until reaching the next branch, etc.

This brute-force method of scheduling combines static and dynamic methods into a hybrid. Such hybrid methods have been tried with success in self-scheduling loops [PoKu87]. The value of this approach remains speculative, however. Yet, it may offer a solution to a most vexing problem for parallel program schedulers.

Bibliography

[ACDi74] Adam, T., Chandy, K., and Dickson, J. A Comparison of list Schedulers for Parallel Processing Systems. *Comm. ACM*, vol. 17, December 1974, pp. 685-690.

[Andri88] Andriessen, J., Task scheduling programming system for the Delft parallel processor, *J. Microprocessor and Microprogramming*, vol 23, no 1-5, Mar 1988, pp. 283.

[ArRo91] Arkin, E. M., and R. O. Roundy, Weighted-Tardiness Scheduling on Parallel Machines With Proportional Weights, *Operations Research*, vol 39, no 1, Jan 1991, pp. 64.

[Bala89] Balakrishnan, A., Preemptive Scheduling of Hybrid Parallel Machines, Operations Research, vol 37, no 2, Mar 1989, pp. 301.

[BPNi90] Beck, M., Pingali, K. K., and A. Nicolau, Static Scheduling for Dynamic Dataflow Machines, *Journal Parallel and Distributed Computing*, vol 10, no 4, Dec 1990, pp. 279.

[BeBa90] Belkhale, K. P., and P. Banerjee, Approximate algorithms for the partitionable independent task scheduling problem, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-72.

[BeSn87] Berman, F., and L. Snyder, On mapping parallel algorithms to parallel architectures, *Journal of Parallel and Distributed Computing*, vol 4, 1987, pp. 439.

[BeSt89] Berman, F., and B. Stramm, Prep-P: Evolution and overview, TR CS89-158, Univ. Calif. San Diego, 1989.

[BlWe84] Blazewicz, J., and J. Weglarz, Scheduling independent 2-processor tasks to minimize schedule length, *Inf. Procss. Letters*, vol 18, no 5, Jun 1984, pp. 267.

[BlCh91] Blocher, J. D., and S. Chand, Scheduling of Parallel Processors: A Posterior Bound on LPL Sequencing and a Two-Step Algorithm, *Naval Research Logistics*, vol 38, no 2, Apr 1991., pp. 273.

[Bokh81a] Bokhari, S. A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in Distributed Processor System, *IEEE Transaction on Software Engineering*, vol. SE-7, no. 6, November 1981.

[Bokh81b] Bokhari, S. On the Mapping Problem, *IEEE Transactions on Computers*, C-30, 3, 1981, pp 207-214.

[Bokh88] Bokhari, S. Partitioning problems in parallel, pipelined, and distributed computing, *IEEE Transactions on Computers*, C-37, 1, 1988, pp 48-57.

[Carr90] Carreno, J., J., Economic Lot Scheduling for Multiple Products on Parallel Identical Processors, *Management Science*, vol 36, no 3, Mar 1990.

[CaKu88] Casavant, T., and J. A. Kuhl, Taxonomy of Scheduling in General Purpose Distributed Computing Systems, *IEEE Transaction on Software Engineering*, vol. SE-14, no. 2, February 1988.

[ChAg90] Chaudhary, V., and J. K. Aggarwal, Generalized Mapping of Parallel Algorithms Onto Parallel Architectures, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. II-137.

[ChSh86] Chen, M., and Shin, K. Embedment of Interacting Task Modules into a Hypercube Multiprocessor. *Proc. Second Hypercube Conf.*, Oct. 1986, pp. 121-129.

[Chen90] Cheng, T.C.E., A state-of-the-art review of parallel-machine scheduling research, *European Journal of Operational Research*, vol 47, no 3, Aug 1990, pp. 271.

[ChSi91] Cheng, T.C.E., and C. C. S. Sin, An algorithm for the N/M parallel/ Cmax preemptive due date scheduling problem, *Engineering Costs and Production Economics*, vol 21, no 1, Feb1991, pp. 743.

[ChAb81] Chou, T., and Abraham, J. Load Balancing in Distributed Systems. *IEEE Transaction on Software Engineering*, vol. SE-8, no. 4, July 1981.

[CHLE80] Chu, W., Holloway, L., Lan, M., and K. Efe, Task Allocation in Distributed Data Processing. *IEEE Computer*, Nov 1980, pp. 57-69.

[ChLa87] Chu, W., and M. Lan, Task allocation and precedence relations for distributed real-time systems, *IEEE Trans. Comput.*, C-36, 6, Jun 1987, pp. 667.

[Coff76] Coffman, E. *Computer and Job-Shop Scheduling Theory*. John Wiley & Sons. 1976.

[Cole91] Cole, R., Approximate Parallel Scheduling: Applications to Logarithmic-Time Optimal Parallel Graph Algorithms. *Information and Computation*, vol 92 no 1, May 91. pp.1.

[Cyben89] Cybenko, G., Dynamic load balancing for distributed memory multiprocessors, J. *Parallel Distrib. Comput.*, vol 7, no 2, Oct 1989, pp. 279.

[Cytr84] Cytron, R., *Compile-time Scheduling and Optimization for Asynchronous Machines*, PhD thesis, Dept. Computer Science, Univ. Illinois, Urbana-Champaign, 1984.

[Dono86] Donovan, K, *Multiprocessor scheduling with practical constraints*, PhD thesis, Univ. Central Florida, GAX86-21094, 1986.

[DJLe89] Du, Jianzhong, and Y. T. Leung, Complexity of Scheduling Parallel Task Systems., *SIAM Journal on Discrete Mathematics*, vol 2, no 4, Dec 1989, pp. 473.

[EbNi89] Ebciogiu, K., and A. Nicolau, Percolation scheduling with resource constraints, TR No 89-31, Dept. Information and Computer Science, Univ. California, Irvine, CA. 92717, 1989.

[Efe82] Efe, K., Heuristic models of task assignment scheduling in distributed systems, *COMPUTER*, July 1982, pp. 50.

[ElAl91] El-Rewini, H., and H. H. Ali, Scheduling Conditional Branching using Representative Task Graphs, Accepted for publication in *The Journal of Combinatorial Mathematics and Combinatorial Computing*, 1991.

[ElLe90] El-Rewini, H., and T. G. Lewis, Scheduling Parallel Program Tasks onto Arbitrary Target Machines, *Journal of Parallel and Distributed Computing*, vol 9, no 2, Jun 1990, pp. 138.

[Emmo90] Emmons, H., Scheduling stochastic jobs with due dates on parallel machines, *European Journal of Operational Research*, vol 47, no1, July 1990, pp. 49.

[Erca88] Ercal, F., *Heuristic approaches to task allocation for parallel computing*, PhD thesis, Ohio State Univ., GAX88-20290

[ErSa89] Erel, E., and S. C. Sarin, Scheduling independent jobs with stochastic processing times and a common due date on parallel and identical machines, *Annals of Operations Research*, vol 17, no 1/4, 1989, pp. 181.

[FTYe90] Fang, Z., Tang, P., and P-C. Yew, Dynamic Processor Self-Scheduling for General Parallel Nested Loops, *IEEE Trans. Computers*, vol39, no 7, Jul 1990, pp. 919.

[FeRu90] Feitelson, D. G., and L. Rudolph, Mapping and scheduling in a shared parallel environment using distributed hierarchical control, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-1.

[Feo86] Feo, J, *Dynamic, distributed resource allocation on regular SW-Banyans*, PhD thesis, Univ. Texas-austin, GAX86-18463

[Fost91] Foster, I., Automatic Generation of Self-Scheduling Programs, *IEEE Trans. on Parallel and Distributed Sys.*, vol 2, no 1, Jan 1991, pp. 68.

[FuKa85] Fuchs, K, and D. Kafura, Memory-constrained task scheduling on a network of dual processors, *J. ACM*, vol 32, no 1, Jan 1985, pp. 102.

[GeNg89] Geist, G.A., and E. Ng, Task Scheduling for Parallel Sparse Cholesky Factorization, *Int'l Journal of Parallel Programming*, vol 18, no 4, Aug 1989, pp. 291.

[Gonz77] Gonzalez, M. Deterministic Processor Scheduling. *Computing Surveys*, vol. 9, no. 3, September 1977.

[Gree88] Green, J., *Load balancing algorithms in a distributed processing environment*, PhD thesis, UCLA, GAX88-22294, 1988.

[GNRe90] Grunwald, D. C., Nazief, B. A. A., and D. A. Reed, Empirical Comparison of Heuristic Load Distribution in Point-to-Point Multicomputer Networks, *Proc. 5th Distributed Memory Computing Conf.*, Apr 1990, Charleston, SC., pp. 984.

[GTUr91] Gupta, A., Tucker, A., and S. Urushibara, The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications, *Performance Evaluation Review*, vol 19, no 1, May 91, pp. 120.

[HaPo91] Hariri, A.M.A., and C. N. Potts, Heuristics for scheduling unrelated parallel machines, *Computers & Operations Research*, vol 18, no 3, 1991, pp. 323.

[HoSh88] Hochbaum, D, and D. Shmoys, A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach, *SIAM J. Comput.*, vol 17, no 3, Jun 1988, pp. 539.

[HLMa90] Hoitomt, D. J., Luh, P. B., and E. Max, Scheduling Jobs with Simple Precedence Constraints on Parallel Machines, *IEEE Control Systems Magazine*, vol 10, n0 2, feb 1990, pp. 34.

[Hu61] Hu, T. Parallel Sequencing and Assembly Line Problems. *Operations Research*, vol.9, 1961, pp. 841-848.

[HXLu90] Hu, Y., Xie, Z., and X. Lu, Approaches to decentralized control job scheduling for homogeneous and heterogeneous parallel computer systems, *Future Generations Computer Systems*, vol 6, no 1, Jun 1990, pp. 91.

[JiJe90] Ji J., and M. Jeng, Dynamic task allocation on shared memory multiprocessor systems, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-17.

[JBGh90] Jiang, H., Bhuyan, L. N., and D. Ghosal, Approximate Analysis of Multiprocessing Task Graphs, Prc. 1990 Int'l Conf. on Parallel Processing, Penn State Univ., Aug 1990, pp. III-228.

[Kamp89] Kampke, T., Optimal Scheduling of Jobs with Exponential Service Times on Identical Parallel Processors, *Operations Research*, vol 37, no 1, Jan 1989, pp. 126.

[Kant89] Kantsedal, S.A., Sequential and Parallel Computations in the General Scheduling Problem, *Automation and Remote Control*, vol 50, no 12, Dec 1989, pp. 1737.

[Kauf74] Kaufman, M. T. An Almost-Optimal Algorithm for the Assembly Line Scheduling Problem. *IEEE Trans. Computers*, Vol. c-23, No. 11, Nov. 1974, pp. 1169-1174.

[Kim88] Kim, S., *A general approach to multiprocessor scheduling*, PhD thesis, Univ. Texas-austin, GAX88-16494, 1988.

[Kohl75] Kohler, W. H. A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems. *IEEE Trans. Computers*, Vol. c-15, No. 12, Dec. 1975, pp. 1235-1238.

[KPVe86] Koutsougeras, C., Papachristou, C., and R. Vemuri, Data flow graph partitioning to reduce communication cost, *SigMicro TC-Micro Newsletter*, vol. 17, no. 4, Dec 1986, pp. 82.

[Krua87] Kruatrachue, B. Static Task Scheduling and Grain Packing in Parallel Processing Systems. Ph.D. thesis, Department of Computer Science, Oregon State University, 1987.

[Kwan89] Kwan, A. W., *Programming environments for parallel programming*, TR 89-06, Dept. Information and Computer Science, Univ. California, Irvine, CA., Jan 1989.

[KBGa89] Kwan, A. W., Bic, L., and D. D. Gajski, *Improving parallel program performance using critical path analysis*, TR 89-05, Dept. Information and Computer Science, Univ. California, Irvine, CA., Jan 1989.

[Lang84] Langston, M., Performance of heuristics for a computer resource allocation problem, *SIAM J. Algebraic Discrete Methods*, vol 5, no 2, Jun 1984, pp. 154.

[LTBo91] Lau, K., Tylavsky, D.J., and A. Bose, Course Grain Scheduling in Parallel Triangular Factorization and Solution of Power System Matrices. *IEEE Trans. on Power Systems*, vol 6, no 2, May 1991, pp.708.

[LHCA88] Lee C. Y., Hwang, J. J., Chow, Y. C., and Anger, F. D. Multiprocessor Scheduling with Interprocessor Communication Delays. *Operations Research Letters*, 7, 3, 1988, pp 141-147.

[Lee91] Lee, C.Y., Parallel machine scheduling with non-simultaneous machine available time, *Discrete Applied Mathematics and Combinatorics*, vol 30, no 1, Jan 1991, pp. 53.

[LeMe87] Lee, E., and D. Messerschmitt, Static scheduling of synchronous data flow programs for digital signal processing, *IEEE Trans. Computers*, C-36, 1, Jan 1987, pp. 24-35.

[LeAg87] Lee, S., and J. Aggarwal, A mapping strategy for parallel processing, *IEEE Trans. Computers*, C-36, 4, Apr. 1987, pp. 433.

[LSTa90] Lenstra, J.K., Shmoys, D.B., and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, vol 46, no 3, Apr 1990, pp. 259.

[LiCh90] Li, K., and H. Cheng, Job scheduling in PMCS using a 2DBS as the system partitioning scheme, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-119.

[LiKa90] Li, K. C., and H. Kam, Static Job Scheduling in Partitionable Mesh Connected Systems, *Journal Parallel and Distributed Computing*, vol 10, no 2., Oct 1990, pp. 152.

[LiKe87] Lin, F., and R. Keller, The gradient model load balancing method, *IEEE Trans. Software Eng.*, vol 13, no 1, Jan 1987, pp. 32.

[Linn85] Linnemann, V. Deterministic Processor Scheduling with Communication Cost, *Fachedaling Informatik Universitat*, Frankfurt, 1985.

[LiYa90] Liu, K. J. R., and K. Yao, Multi-phase systolic architectures for spectral decomposition, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-123.

[Lo84] Lo, V. M., Heuristic Algorithms for Task Assignment in Distributed Systems. *Proc. 4th Int. Conf. Distr. Comput. Syst.*, May 1984, pp.30-39.

[LRGK90] Lo, V. M., Rajopadhye, S., Gupta, S., Keldsen, D., Mohamed, M. A., and J. A. Telle, OREGAMI: Software Tools for Mapping Parallel Computations to Parallel Architectures, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. II-88.

[Mart88] Martel, C. U., A Parallel Algorithm for Preemptive Scheduling of Uniform Machines., *Journal of Parallel and Distributed Computing*, vol 5, no 6, Dec 1988, pp. 700.

[MINi90] Masuda, T., Ishii, H., and T. Nishida, Scheduling problem on quasidentical parallel machines, *Mathematica Japonica*, vol 35, no 3, May 1990, pp. 545.

[McGi90] McCreary C., and H. Gill, Effiecient Exploitation of Concurrency Using Graph Decomposition, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. II-199.

[MoSk90] Moon, Y., and J. Sklansky, A Class of Mapping Algorithms for Hypercube Computers, *Proc. 5th Distributed Memory Computing Conf.*, Apr 1990, Charleston, SC., pp. 903.

[MKLM88] Muhlenbein, H., Kraemer, O., Limburger, F., Mevenkamp, M., and S. Streitz, Design rationale for MUPPET: A programming environment for message-based multiprocessors, *Proc. First Int'l Conf. on Supercomputing*, Lecture Notes in Computer Science 297, Springer-Verlag, Berlin, 1988.

[MuSi90] Munshi, A. A., and B. Simons, Scheduling Sequential Loops on Parallel Processors, *SIAM Journal on Computing*, vol 19, no 4, Aug 1990, pp. 728.

[MuEv89] Musier, R.F.H., and L. B. Evans, An approximate method for the production scheduling of industrial batch processes with parallel units, *Computers and Chemical Engineering*, vol 13, no 1/2, Jan 1989, pp. 229.

[NiWu] Ni, L, and C. Wu, Design tradeoffs for process scheduling in shared memory multiprocessor systems, *IEEE Trans. Software Eng.*, vol 15, no 3, Mar 1989, pp. 327.

[NiKi89] Ni, L., and C. King, On partitioning and mapping for hypercube computing, *Int'l J. Parallel Programming*, vol 17, no 6, Dec 1989, pp. 475.

[PaTs87] Papadimitriou, C., and J. Tsitsiklis, On stochastic scheduling with in-tree precedence constraints, *SIAM J. Comput.*, vol 16, no 1, Feb 1987, pp.1.

[PMMc90] Pekny, J.F., Miller, D.L., and G. J. McRae, An exact parallel algorithm for scheduling when production costs depend on consecutive system states, *Computers & Chemical Engineering*, vol 14, no 9, Sept 1990, pp. 1009.

[PoKu87] Polychronopoulos, C., and D. Kuck, Guided self-scheduling: A practical scheduling scheme for parallel supercomputers, *IEEE Trans. Comput.*, C-36, 11, Nov 1987, pp. 1374.

[Pras87] Prastein, M. Precedence-Constrained Scheduling with Minimum Time and Communication. MS thesis, University of Illinois at Urbana-Champaign, 1987.

[PRGr87] Purtilo, J., Reed, D. A., and D. C. Grunwald, Environments for prototyping parallel algorithms, *Proc. 1987 Int'l Conf. Parallel Processing*, The Pennsylvania State Univ. Press, University Park, PA., 1987.

[Rayw87] Rayward-Smith, V., The complexity of preemptive scheduling given interprocesor communication delays, *Inf. Process. Letters*, vol 25, no 2, May 1987, pp. 123.

[Razo87] Razouk, R. R., A guided tour of P-NUT (release 2.2), TR 86-25, Dept. of Information and Computer Science, Univ. California, Irvine, CA., Jan 1987.

[Robe89] Robert, Y., Optimal scheduling algorithms for parallel Gaussian elimination, *Theoretical Computer Science*, vol 64, no 2, May 1989, pp. 159.

SaEr87] Sadayappan, P., and F. Ercal, Mapping of finite element graphs onto processor meshes, *IEEE Trans. on Computers*, vol C-36, no 12, Dec 1987, pp. 1408.

[SSAh89] Sagar, G., Sarje, A., and K. Ahmed, Task allocation techniques for distributed computing systems, *J. Microcomput. Appl.*, vol 12, no 2, Apr 1989, pp. 97.

[SCMi90] Saltz, J., Crowley, K., and R. Michandaney, Run-Time Scheduling and Execution of Loops on Message Passing Machines, *Journal of Parallel and Distributed Computing*, vol 8, no 4, apr 1990, pp. 303.

[Sark87] Sarkar, V., *Partitioning and scheduling parallel programs for execution on multiprocessors*, PhD thesis, Stanford, GAX87-23080, 1987.

[SLHH91] Sartor, J. M., Lamont, G. B., Hammell II, R. J., and T. C. Hartrum, Mapping Precedence-constrained simulation tasks for a parallel environment, *Proc. 6th Distributed Memory Computing Conf.*, Portland, OR. Apr. 1991, pp. 2.

[Seot91] Seitz, C. L., *Resources in Parallel and Concurrent Systems*, ACM Press, 1991.

[Seth76] Sethi, R. Scheduling Graphs on Two Processors. *SIAM J. Computers*, vol. 5, no. 1, March 1976, pp. 73-82.

[SiLe90] Sih G. C., and E. A. Lee, Scheduling to account for interprocessor communication within interconnection-constrained processor networks, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-101.

[ShFo90] Shang, W., and J. A. B. Fortes, Time-optimal and conflict-free mappings of uniform dependence algorithms into lower dimensional processor arrays, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. II-88.

[SWPa90] Shirazi, B., Wang, M., and G. Pathak, Analysis and Evaluation of Heuristic Methods for Static Task Scheduling, *Journal Parallel and Distributed Computing*, vol 10, no 3, Nov 1990, pp. 222.

[SySo86] Synder, L., and D. Socha, POKER on the Cosmic Cube: The first retargetable parallel programming language and environment, *Proc. 1986 Int'l Conf. Parallel Processing*, IEEE Computer Society Press, Washington, DC., 1986.

[So90] So, K. C., Some Heuristics for Scheduling Jobs on Parallel Machines with Setups, *Management Science*, vol 36, no 4, Apr 1990, pp. 467.

[Sumi87] Sumichrast, R., Scheduling parallel processors to minimize setup time, *Comput. Oper. Res.*, vol 14, no 4, Oct 1987, pp. 305.

[Tang90] Tang, C.S., Scheduling batches on parallel machines with major and minor set-ups, *European Journal of Operational Research*, vol 46, no 1, May 1990, pp. 28.

[TaLi90] Tang, Z., and G. J. Li, Optimal granularity of grid iteration problems, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-111.

[Tows86] Towsley, D. Allocating Programs Containing Branches and Loops Within a Multiple Processor System, *IEEE Transaction on Software Engineering*, vol. SE-12, no. 10, October 1986.

[TyNi89] Tyrrell, A., and J. Nicoud, Scheduling and parallel operations on the transputer, *J. Microprocessor and Microprogramming*, vol 26, no 3, Oct 1989, pp. 175.

[Ullm75] Ullman, J. NP-Complete Scheduling Problems, *Journal of Computer and System Sciences*, vol. 10, 1975, pp. 384-393.

[VLLe90] Veltman, B., Lageweg, B.J., and J. K. Lenstra, Multiprocessor scheduling with communication delays, *Parallel Computing*, vol 16, no 2/3, Dec 1990, pp. 173.

[VeDa90] Venkatesh, R., and G. R. Dattatreya, Adaptive optimal load balancing of loosely coupled processors with arbitrary service time distributions, *Proc. 1990 Int'l Conf. on Parallel Processing*, Aug 1990, pp. I-22.

[WaCh91] Wang, Q., and K. H. Cheng, List scheduling of parallel tasks, *Information Processing Letters*, vol 37 no 5, Mar 1991, pp. 291.

[Weis90] Weiss, G., Approximation results in parallel machines stochastic scheduling, *Annals of Operations Research*, vol 26, no 1/4, Dec 1990, pp. 195.

[Wool] Woolsey, R. E. D., Production Scheduling Quick and Dirty Methods for Parallel Machines, *Production and Inventory Management Journal*, vol 31, no 3, pp. 84.

[WuGa88] Wu, M. Y., and D. D. Gajski, Computer-aided programming for multiprocessor systems, TR 88-19, Dept. Information and Computer Science, Univ. California, Irvine, CA. Jun 1988.

[WuGa88] Wu, M. Y., and D. D. Gajski, A programming aid for hypercube architectures, *J. Supercomputing*, vol 2, no 3, 1988.

[XHYu90] Xuejun, Y., Haibo, C., and C. Yungui, Processor Self-Scheduling for parallel loops in pre-emptive environments, *Future Generations Computer Systems*, vol 6, no 1, Jul 1990, pp. 97.

[Yu84] Yu, W. H., LU Decomposition on a Multiprocessing System with Communication Delay. Ph.D. thesis, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1984.

[ZLEa91] Zahorjan, J., Lazowska, E.D., and D. L. Eager, The Effect of Scheduling Discipline on Spin Overhead in Shared Memory Parallel Systems, *IEEE Transactions on Parallel and Distributed Sys.*, vol 2, no 2, Apr 1991, pp. 180.