SCHEDULING TO ACCOUNT FOR INTERPROCESSOR COMMUNICATION WITHIN INTERCONNECTION-CONSTRAINED PROCESSOR NETWORKS

Gilbert C. Sih
Edward A. Lee
Department of EECS
University of California
Berkeley, California 94720

ABSTRACT

Interprocessor communication (IPC) overhead can severely degrade the performance of parallel processing systems. This paper presents a unified approach to the compile-time scheduling of precedence-constrained, communicating tasks onto arbitrarily interconnected processor networks containing dedicated communication hardware. Scheduling and routing are performed simultaneously to account for limited interconnections between processors, and shared resource contention is eliminated through the scheduling of all communications as well as computations. A new scheduling heuristic called **dynamic level scheduling** is proposed, which modifies the classical list scheduling methodology to account for IPC and synchronization overhead. This technique is fast, widely targetable, and displays promising performance results.

1. Introduction

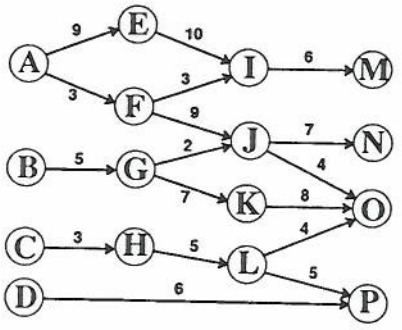
The biggest impediment to the use of parallel processing remains the scarcity of techniques for effective partitioning and scheduling of programs. The difficulties stem from the need for multiple processors to exchange intermediate results, which causes transmission/synchronization delays and contention for shared resources. This **interprocessor communication** (IPC) degrades performance in parallel processing systems. Instances of the "saturation effect", in which the addition of more processors actually decreases throughput due to excessive IPC, have been well-documented [1]. It is therefore essential that schedulers for parallel architectures incorporate IPC considerations if the full performance benefits of parallel hardware are to be attained.

The authors gratefully acknowledge the support of SRC, Cygnet, Dolby Laboratories, and the State of California Micro program.

The problem being addressed is the compile-time (static) scheduling of acyclic precedence graphs onto multiple processor architectures with limited interconnections. These precedence graphs may be derived from data flow graph algorithmic descriptions which fit the Synchronous Data Flow (SDF) model [2], which requires that the node execution times and the number of data units passed or received on every arc on each node invocation are known to the compiler. While this model is primarily suited for signal processing and some scientific computation, the domain of application can be broadened somewhat through the use of *self-timed* scheduling. See [3] for a discussion of these issues.

Possible target architectures include tightly-coupled shared bus configurations, message-passing multicomputer topologies such as meshes, rings, or hypercubes, as well as networks of processors and memories interconnected through dynamic switching networks. The architecture is assumed to contain separate communication hardware, permitting the overlap of communication with computation. Overhead introduced by IPC is accounted for, and shared resource contention is eliminated through scheduling of all communications as well as computations.

The input to the scheduling algorithm is an acyclic precedence expansion graph (APEG), which is the expanded precedence graph of the dataflow representation augmented with data transfer information. More precisely, an APEG is a finite acyclic digraph G = {N,A}. N is a set of computation nodes (tasks) $\{N_i\}$ i = 1 ... n, with known execution times, where each node is executed exactly once in each invocation of the program. A is the set of directed arcs $\{A_{ij}\}$ between nodes which define a partial order or precedence constraint (<) on N such that arc A_{ij} directed from node N_i into node N_j implies that N_i must precede N_j ($N_i < N_j$) in execution. Each arc A_{ij} also carries label D_{ij} which specifies the amount of data (in bits, bytes, or words) that N_i passes to N_j on each invocation. An example APEG and associated node execution time table is shown below in figure 1.



Execution Times

A	3
B	6
C	5
D	3
E	5
F	2
G	4
H	4
I	5
J	3
K	6
L	4
M	5
N	5
O	4
P	6

Figure 1. An APEG and node execution time table

The target architecture consists of a set of processors, $\{P_k\}$ k = 1 ... p, interconnected in some specified manner. The characteristics of the architecture must be known, so that the time needed for communication of data between two given processors can be calculated deterministically (or upper-bounded) if resource availability is guaranteed. Communication between nodes co-resident on the same processor is assumed to have no cost. Communication between nodes located on different processors entails IPC cost, because the receiving node must postpone its execution for the time interval required for data transfer. Processors are assumed to be identical in this discussion, with a treatment of heterogeneous processors deferred to a forthcoming publication.

Scheduling will be nonpreemptive, so that once a node has started execution on a processor, it will run to completion. The scheduling objective is to minimize the **schedule length**, or **makespan**, when all communication and synchronization costs have been included. This goal equivalently maximizes the **speedup**, defined as the shortest time required for sequential execution of the APEG on a single processor, divided by the time required for parallel execution on multiple processors. Even when communications are ignored, this scheduling problem falls into the class of NP-complete problems[4]. When interprocessor communications are included, the problem becomes NP-complete in the strong sense, even if there are an infinite number of processors available [5]. Hence we will rely upon heuristics.

Much related work has concentrated on task allocation, attempting to assign tasks to processors in order to minimize some objective function [1, 6, 7, 8, 9]. While these works are innovative, they either ignore precedence constraints or attempt to minimize an objective other than schedule length, and thus are not applicable in this context. A relevant approach, called linear clustering, has been proposed by Kim and Browne [10]. This technique transforms the task graph into an intermediate representation called a Virtual Architecture Graph (VAG) by iteratively clustering the most expensive (in computation and communication) paths into a single node. After successive refinement steps, the VAG is then mapped onto the specified architecture using graph theoretic techniques. A proposal by Sarkar and Hennessy attempts to initially minimize the schedule length on an infinite number of processors (critical path length) [11], by initially partitioning the graph into blocks of tasks. The initial partition places each task in a separate block, and each successive step merges the two blocks which yield the biggest decrease in critical path length. This component terminates when no further decreases in critical path length are possible. The scheduling phase proceeds to merge blocks until the number of blocks equals the number of processors. A method which employs branch and bound heuristics to prune the search space of possible schedules has been suggested by Greenblatt and Linn [12].

The scheduler has been implemented as part of an interactive design system for digital signal processing (DSP) called Gabriel, which allows rapid prototyping of new DSP algorithms using a block diagram graphical interface. The blocks span the entire range from fine-grained operators such as adders or multipliers, to medium-grained functions such as FFT's, to large-grained tasks such as a speech coder. Using feedback information from the scheduler, an algorithm designer can iteratively refine both task graph and target architecture to maximize performance. This environment imposes three constraints on the scheduling method employed. First, the interactive nature of the design approach requires that the scheduling technique execute rapidly. Second, the scheduling technique must be flexible enough to handle task graphs of arbitrary granularity, which affects the tradeoff between the amount of parallelism utilized and the amount of communication overhead incurred. Third, the scheme must be adaptable to the plethora of architectures found in digital signal processing. Our proposed scheme, called **dynamic level scheduling**, is a promising approach toward meeting these goals.

This paper is organized as follows: section 2 reviews the classical HLFET scheduling algorithm. Section 3 introduces the dynamic level scheduling strategy, and section 4 presents methods of streamlining the algorithm. Section 5 summarizes and indicates future research directions.

2. List Scheduling Algorithms

List scheduling is a technique in which tasks are assigned priorities and placed in a list, sorted in order

of decreasing priority. Nodes whose predecessors have been completed are designated as being **ready** (for execution). A global time clock serves to regulate the scheduling process. Processors which are idle at the current time are designated as being **available** (for assignment). When a processor is available, the first ready node in the list is assigned to be executed on that processor. After assignment, the processor is removed from the available processor list, the node is deleted from the priority list, and this process is repeated until the available processors have been exhausted. The time clock is then incremented until some processors finish execution of their allotted tasks and are available once again. The algorithm terminates when all nodes have been scheduled.

The most widely known list scheduling method is HLFET (Highest Levels First with Estimated Times) [13], one of the class of critical path algorithms [14]. HLFET is an extension of Hu's pioneering work [15]. In this procedure, the priority of each node is set equal to its **level**, defined as the largest sum of execution times on any directed path from the node to an endnode of the graph. List scheduling is then performed. To minimize confusion in terminology, these levels will be referred to as **static** levels. In the absence of IPC, the HLFET algorithm demonstrates near-optimal performance in almost all cases [13]. The success of this technique stems from the accurate representation of a node's priority by its static level, which causes each successive scheduling step to shorten the longest path to completion.

List scheduling with inclusion of communication delay was addressed by Yu in the context of a fully-interconnected processor network [16]. He proposed a heuristic which selects the ready node with the highest static level at each step, and schedules it on the available processor which will complete execution of the node at the earliest time. Yu also proposed advanced techniques which use combinatorial matching algorithms to pair ready nodes with available processors. Although the ideas are sound, the use of the classical list-scheduling methodology leads to an inherent flaw which is exposed in section 3.

3. Incorporating IPC Considerations

In the absence of interprocessor communication, all available task parallelism can be utilized without cost. That is, given enough processors, an optimal schedule can always be constructed by invoking all simultaneously executable nodes on different processors. IPC considerations induce a tradeoff between the amount of parallelism utilized and the amount of communication overhead incurred. In general, as the amount of computation decreases relative to the amount of IPC, the amount of parallelism which can be effectively utilized also decreases.

3.1. Handling Communication Resources

Scheduling in the presence of IPC contains two main aspects: assigning processors for computation nodes, and allocating communication resources for interprocessor data transfers. These two problems, often referred to as the "mapping problem" and the "traffic scheduling" problem respectively, have traditionally been dealt with separately. A task allocation algorithm first assigns nodes to processors in accordance with some objective function, followed by a routing algorithm which performs interprocessor traffic scheduling upon the node mapping [17, 7]. This separation is impractical, because the ease with which the traffic scheduling can be performed is directly dependent on the properties of the mapping; the best isolated assignment of nodes to processors is invariably suboptimal after simultaneous consideration of both communications and computations.

Our proposed scheduling strategy addresses both issues concurrently, trying to avert overloaded communication resources by adjusting the node-processor mapping accordingly. Just as computations are scheduled upon processors, communications are scheduled upon IPC resources by dedicating the resources used in a data transfer for the duration of the transmission. With guarantee of resource availability, the communication time can be calculated deterministically (or upper bounded) using the locations of source and destination processors, the amount of data to be transferred, and the characteristics of the communication architecture. A routing algorithm, employed by the scheduler, uses knowledge of previous resource usage to reserve a path between source and destination processors for this duration.

For illustrative purposes, consider the scheduling of the APEG from figure 1 onto the target architecture shown in figure 2, which consists of four processors interconnected through four full-duplex interprocessor links. For simplicity, assume that the time needed to communicate D units of data between any two processors is merely D time units, ignoring the fact that communication between P1 and P3 is bound to be slower than communication between P1 and P2. The upper chart in figure 3 shows a possible scheduling of nodes onto processors, while the lower chart in figure 3 shows the corresponding scheduling of communications onto links. This simultaneous consideration of spatial (routing) and temporal (scheduling communi-

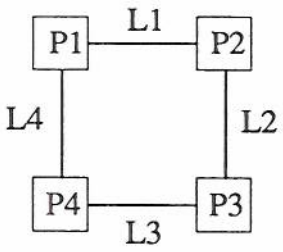


Figure 2. A 4-processor target architecture

cation time windows) aspects of IPC eliminates the possibility of shared resource contention, which ensures deterministic behavior.

The scheduling algorithm is divided into two components. The first component contains the fixed, architecture-independent scheduling routines, while the second component contains the architecture-dependent communication resource scheduling and routing routines. This division permits wide targetability without sacrificing efficiency, enabling special-purpose routines optimized for a particular architecture to be employed within the second component. A specific interface is defined at the boundary, which allows the topology dependent sections to be interchangeable. A topology dependent portion for a new architecture can be coded within a few hours.

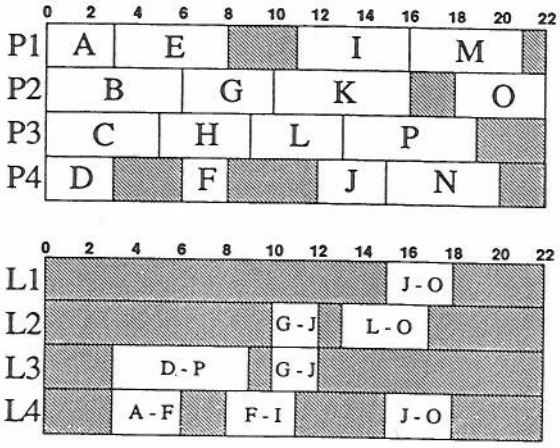


Figure 3. Time charts displaying the scheduling of nodes onto processors and communications onto links

3.2. Dynamic Levels

A list-scheduling algorithm can be decomposed into the execution of two fundamental tasks at each step: selection of the next ready node to schedule, and selection of the processor on which the chosen node should be scheduled. The HLFET scheme performs these tasks independently, leading to poor performance when IPC is taken into consideration. To improve node and processor selection, a new quantity is introduced whose value changes throughout the scheduling process. This **dynamic level**, denoted $DL(N_i, P_j, \Sigma(t))$, reflects the quality of the match between node N_i and processor P_j at state $\Sigma(t)$, where $\Sigma(t)$ encompasses both the state of the processing resources (previously scheduled nodes), and the state of the communication resources (previously scheduled data transfers) at global time t. To introduce the dynamic level concept, some notation is first defined. $SL(N_i)$ represents the static level of node i, $N_R(\Sigma(t))$ denotes the set of ready nodes at state $\Sigma(t)$, and $P_A(\Sigma(t))$ represents the set of available processors at state $\Sigma(t)$ In addition, we define $DA(N_i, P_j, \Sigma(t))$ to be the earliest time that all *data* required by node N_i is *available* at processor P_j given state $\Sigma(t)$. This quantity, calculated within the topology-dependent section of the scheduler, represents the earliest time at which all data transfers to node N_i from its immediate predecessors are **guaranteed** to have been completed with communication resource availability assured. The dynamic level can now be defined as

$$DL(N_i, P_j, \Sigma(t)) = SL(N_i) - \max[t, DA(N_i, P_j, \Sigma(t))] \quad (2)$$

The interpretation of this quantity is straightforward. The maximization term represents the earliest time that node N_i can start execution on processor P_j, because the node cannot start execution until the current time, and it cannot be invoked until all the data from its predecessors has been received. So the dynamic level $DL(N_i, P_j, \Sigma(t))$ is the difference between the static level of node N_i and the earliest time the node can start execution on processor P_j. This expression is intuitively appealing, because it simultaneously incorporates execution and communication time aspects. By evaluating dynamic levels (which may be negative) over all combinations of ready nodes and available processors, this technique hopes to find the best node-processor match for scheduling given the current state.

Before engaging in performance comparisons between algorithms, it is important to realize that performance variations can occur with changes in the graph size, structure, density, parallelism, and the relative magnitudes of node execution and internode communication times. In our experience, the only general principle seems to be that for each scheduling approach, there exist specific graph instances which will be scheduled poorly. The prevailing goal then, must be to capture some notion of "average performance". To extract general trends over a broad range of test inputs, we will use randomly-generated graphs, where the graph instances have the relative amount of data transfer and computation varied, as well as the amount of task parallelism relative to the number of

processors. We will supplement this data with results obtained from scheduling DSP algorithms on our only physical target architecture aimed at signal processing: a four-processor shared-memory multiprocessor donated by Dolby Labs.

To measure the performance improvement obtainable through dynamic levels, randomly generated task graphs containing between 50 and 250 nodes were scheduled onto a 16-processor mesh. Node execution times and nearest-neighbor communication times were chosen randomly from the same uniform distribution. Several methods for node and processor selection were investigated. The first method initially selects the available processor with smallest index and then chooses the ready node which maximizes the dynamic level with this processor. The second method initially selects the ready node with highest static level and then chooses the available processor maximizing the dynamic level with this node. The third method examines all possible combinations of ready nodes and available processors and chooses the node-processor pair yielding the highest dynamic level. The performance improvement of these three methods over the HLFET approach using independent node and processor selection are compared, where communication costs have been included in all cases.

An interesting set of curves emerges when the amount of parallelism in the APEG is varied with respect to the number of processors. The measure of parallelism being used is the ratio of the total sum of node execution times divided by the length of the critical path through the graph (the longest path from any initial node to any terminal node). This is a lower bound on the number of processors needed to execute the graph in time bounded by the critical path when interprocessor communication is excluded. Curves displaying percentage improvement in speedup obtained from using dynamic levels over the static levels employed by HLFET are shown below in figure 4, plotted against this measure of parallelism. A curve is shown for each of the aforementioned techniques, where each point represents an average taken over multiple graphs with the specified parallelism.

The initial processor selection technique exhibits little improvement when the amount of graph parallelism is small compared to the number of processors because there are very few ready nodes at each scheduling step, in many cases only a single node. The initially chosen processor is forced to choose this single ready node, therefore performing the same steps as the HLFET algorithm and delivering comparable performance. As the amount of parallelism increases, the number of ready nodes increases, permitting a better match between node and processor. Performance increases accordingly.

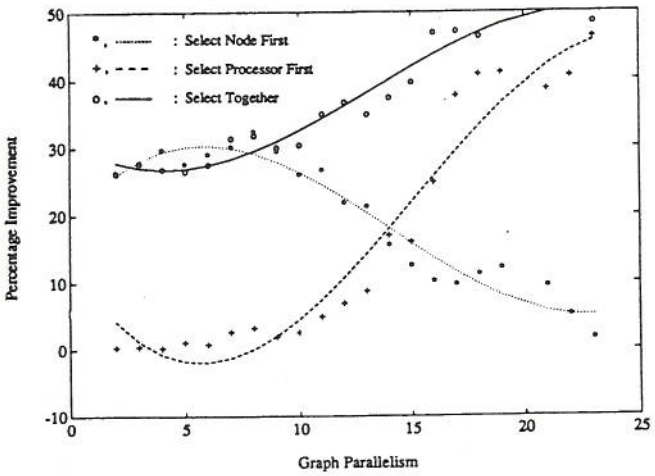


Figure 4. Percentage improvement in speedup over HLFET

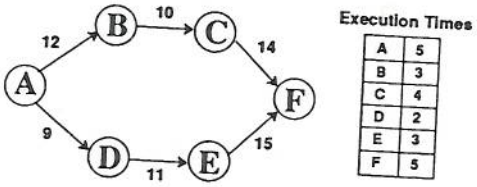
Conversely, the initial node selection technique exhibits large improvement when the number of processors exceeds the amount of graph parallelism, because there is little contention for processors; each node is able to select its "preferred" processor. As the amount of parallelism increases, parallel paths in the graph must share processing resources. As processors are successively removed from the available processor list, the node with highest static level is often forced to be executed on one of the remaining available processors for which excessive IPC is incurred. This performance degradation is exacerbated as the amount of parallelism is further increased.

Selecting the highest dynamic level ready-node, available-processor pair out of all combinations retains good improvement throughout the entire range, increasing slightly as the amount of graph parallelism increases. The increased flexibility accorded this strategy allows a more effective matching of nodes with processors. This method demonstrates superior performance over the other techniques, attaining speedup improvements of over 50% in comparison with the HLFET algorithm. However, this increased performance is realized at the price of added computational complexity.

3.3. Revising Processor Selection

While the addition of dynamic levels significantly improves performance, the algorithm still exhibits the list scheduling deficiency of being unable to idle "available" processors. Consider the graph shown in figure 5, and for simplicity, assume a 2 processor system with communication model C = D, so that the number of cycles needed for IPC equals the number of data units. The optimal schedule executes every

node on a single processor while idling the other processor completely, a solution which is unobtainable using the current list scheduling methodology. The inability to idle processors is an inherent flaw in the algorithm, which requires that all available processors be assigned nodes for execution before the global clock can be updated to replenish the supply.

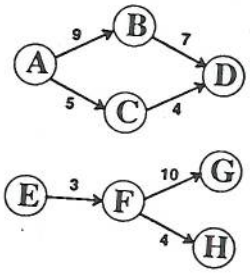


Execution Times

A	5
B	3
C	4
D	2
E	3
F	5

Figure 5. A fine-grained precedence graph

To remedy this difficulty, the fundamental operation of the algorithm is altered in a subtle but important manner. The global timeclock used to update the current time at each scheduling step is removed, so that processors are no longer classified as being "busy" or "available". All processors can now be considered candidates for scheduling at each step, which allows the same processor to be chosen in consecutive scheduling steps. To illustrate the effect of this modification more clearly, the scheduling steps taken by the algorithm both with and without the global time clock will be contrasted using the APEG shown in figure 6, which will be scheduled onto a two



Execution Times

A	3
B	3
C	3
D	7
E	2
F	3
G	5
H	6

Static Levels

A	13
B	10
C	10
D	7
E	11
F	9
G	5
H	6

Figure 6. An example acyclic precedence expansion graph

processor system interconnected by a full-duplex data link. The scheduling steps taken by the algorithm with the global clock are shown in figure 7. After nodes A and B have been scheduled on P1 and nodes E and F have been scheduled on P2, the algorithm has its global clock at time 5, when P2 is the only processor available for scheduling. After dynamic level evaluation of the three ready nodes C, G, and H with P2, the algorithm schedules node C on P2, and schedules communication A to C on the link from P1 to P2 in the interval {3,8}. The global clock is updated to time 6, when P1 becomes available. Dynamic level evaluation for nodes G and H with P1 results in node H being scheduled on P1 and communication F to H scheduled on the link from P2 to

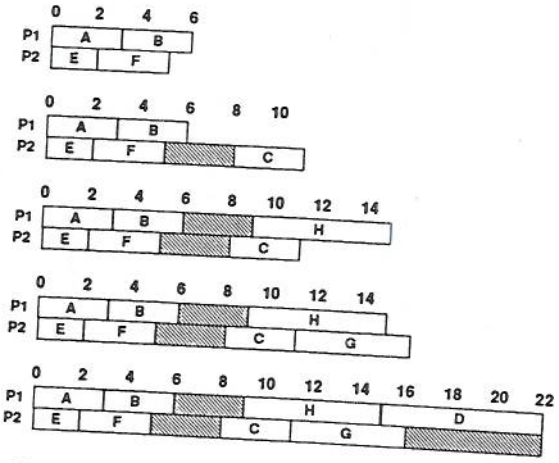


Figure 7. Scheduling progression with the global clock

P1 in the interval {5,9}. The global clock is updated to time 11, freeing P2. After evaluating dynamic levels for nodes G and D with P2, node G is scheduled on P2. Finally, node D is scheduled on P1, yielding a final makespan of 22 time units.

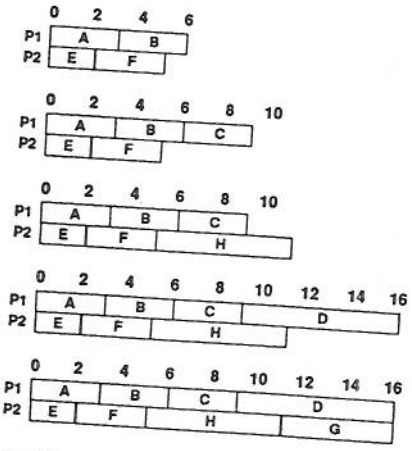
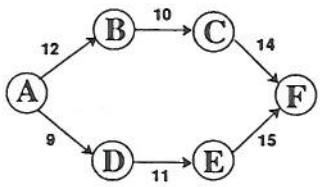


Figure 8. Scheduling progression without the global clock

The scheduling steps taken by the algorithm without the global clock, shown in figure 8, start to diverge from the previous approach after nodes A, B, E, and F have been scheduled. Dynamic levels are evaluated for nodes C, G, and H on each of processors P1 and P2, resulting in node C being matched with P1. After scheduling node C on P1, node D is immediately released into the list of ready nodes even though it can not be executed until time 9. This approach relaxes the constraint which forces all processors to be scheduled together; some processors may have nodes scheduled far in advance of other processors. The algorithm proceeds to schedule node H on P2, node D on P1, and node G on P2, resulting in an optimal schedule with makespan 16. Since additional processors are incorporated only as they are needed, this modified approach constructs schedules

node on a single processor while idling the other processor completely, a solution which is unobtainable using the current list scheduling methodology. The inability to idle processors is an inherent flaw in the algorithm, which requires that all available processors be assigned nodes for execution before the global clock can be updated to replenish the supply.

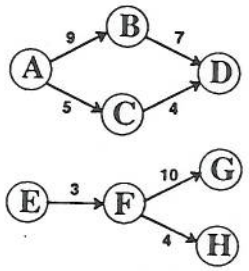


Execution Times

A	5
B	3
C	4
D	2
E	3
F	5

Figure 5. A fine-grained precedence graph

To remedy this difficulty, the fundamental operation of the algorithm is altered in a subtle but important manner. The global timeclock used to update the current time at each scheduling step is removed, so that processors are no longer classified as being "busy" or "available". All processors can now be considered candidates for scheduling at each step, which allows the same processor to be chosen in consecutive scheduling steps. To illustrate the effect of this modification more clearly, the scheduling steps taken by the algorithm both with and without the global time clock will be contrasted using the APEG shown in figure 6, which will be scheduled onto a two



Execution Times

A	3
B	3
C	3
D	7
E	2
F	3
G	5
H	6

Static Levels

A	13
B	10
C	10
D	7
E	11
F	9
G	5
H	6

Figure 6. An example acyclic precedence expansion graph

processor system interconnected by a full-duplex data link. The scheduling steps taken by the algorithm with the global clock are shown in figure 7. After nodes A and B have been scheduled on P1 and nodes E and F have been scheduled on P2, the algorithm has its global clock at time 5, when P2 is the only processor available for scheduling. After dynamic level evaluation of the three ready nodes C, G, and H with P2, the algorithm schedules node C on P2, and schedules communication A to C on the link from P1 to P2 in the interval {3,8}. The global clock is updated to time 6, when P1 becomes available. Dynamic level evaluation for nodes G and H with P1 results in node H being scheduled on P1 and communication F to H scheduled on the link from P2 to

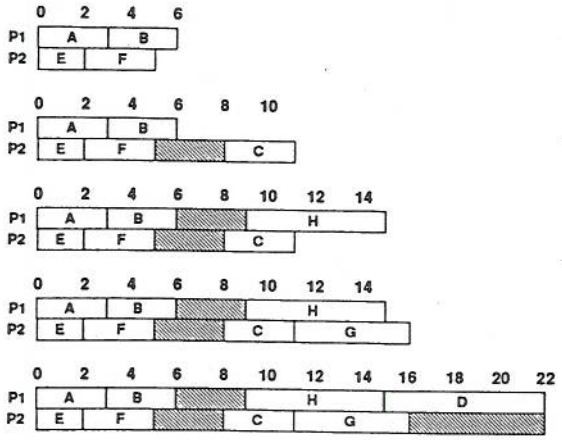


Figure 7. Scheduling progression with the global clock

P1 in the interval {5,9}. The global clock is updated to time 11, freeing P2. After evaluating dynamic levels for nodes G and D with P2, node G is scheduled on P2. Finally, node D is scheduled on P1, yielding a final makespan of 22 time units.

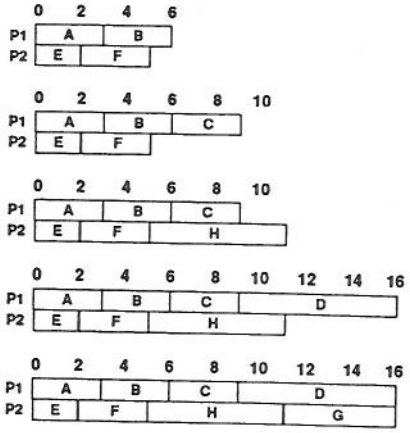


Figure 8. Scheduling progression without the global clock

The scheduling steps taken by the algorithm without the global clock, shown in figure 8, start to diverge from the previous approach after nodes A, B, E, and F have been scheduled. Dynamic levels are evaluated for nodes C, G, and H on each of processors P1 and P2, resulting in node C being matched with P1. After scheduling node C on P1, node D is immediately released into the list of ready nodes even though it can not be executed until time 9. This approach relaxes the constraint which forces all processors to be scheduled together; some processors may have nodes scheduled far in advance of other processors. The algorithm proceeds to schedule node H on P2, node D on P1, and node G on P2, resulting in an optimal schedule with makespan 16. Since additional processors are incorporated only as they are needed, this modified approach constructs schedules

which exhibit a natural "clustering" of nodes which communicate heavily, without sacrificing efficient use of the communication resources.

Removal of the global time clock necessitates a few changes in the dynamic level expression. The state of the processing and communication resources $\Sigma(t)$ loses its time dependence and is now denoted Σ. The notation $TF(P_j, \Sigma)$ is introduced to represent the time that the last node which has already been mapped onto the jth processor finishes execution. The revised dynamic level can now be represented as:

$$DL(N_i, P_j, \Sigma) = SL(N_i) - \max[TF(P_j, \Sigma), DA(N_i, P_j, \Sigma)] \quad (3)$$

The ready node and processor which maximize this expression are again chosen for scheduling, where the processor candidates now encompass the entire set.

The effect of this modification is exhibited below in figure 9 where the performance curve of the revised algorithm is shown in comparison with the global clock algorithm with dynamic levels. At modest levels of graph parallelism, the two methods exhibit comparable performance. As the amount of parallelism increases, the modified approach exhibits sharply increasing performance, due to this clustering

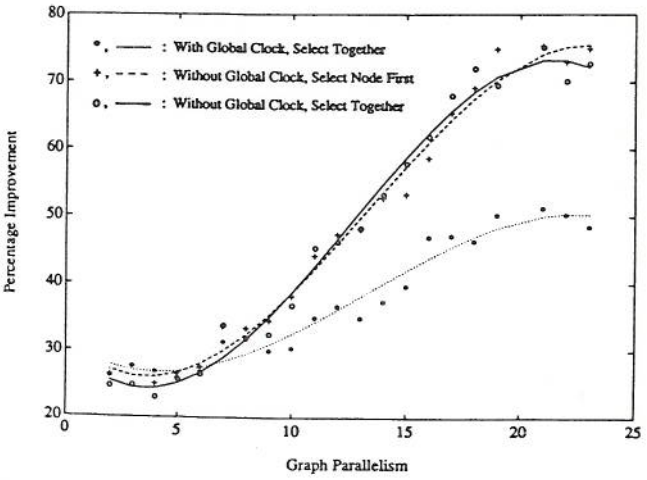


Figure 9. Percentage improvement in speedup over HLFET

phenomenon. With increased freedom in selecting processors, the modified approach displays an enhanced ability to assign each node its "preferred" processor even as the number of parallel paths increases. Speedup improvements of 75% over the HLFET algorithm were observed. As might be expected, the greater the severity of the IPC costs, the greater the performance gain obtained through the revision. Notice that the modified algorithm using initial node selection does not exhibit the performance degradation shown earlier by the global clock approach as the graph parallelism increases. Initially selecting the highest static level node and then selecting the processor through dynamic levels now exhibits nearly equal performance as simultaneous node and processor selection. A plausible explanation is that as a result of the greater freedom in processor selection, the same scheduling steps are occurring, but in a different order.

In practice, we found the speedup improvement to be even higher. When scheduling signal processing algorithms on a four-processor shared-memory multiprocessor, speedup improvements exceeding 100% compared to the HLFET algorithm were common when the parallelism greatly exceeds the number of processors.

4. Streamlining the Algorithm

The algorithm using dynamic levels without a global clock can be simplified for faster execution without an appreciable degradation in performance.

4.1. Initial Node Selection

The ability to gain analogous performance in using initial node selection as when selecting nodes and processors together yields a great savings in execution time because examination of all node-processor combinations is no longer necessary at each scheduling step. After investigation of several possibilities, the following expression was selected as a criterion for choosing the node to be scheduled from among all ready nodes: $\max_i \{SL_i + C_{adj}[\max_k (D_{ki})]\}$ The notation D_{ki} represents the number of data units passed from node k to node i, and the expression $C_{adj}(D)$ denotes the time needed to communicate D data units between adjacent processors with communication resource availability guaranteed. This criterion is therefore the node's static level plus the adjacent processor communication time for the maximum amount of data passed on any arc into the node from an immediate predecessor.

4.2. Limiting Processor Selection

To further reduce the scheduling time, the number of processors for which dynamic levels are evaluated can be reduced. For many multicomputer networks (e.g. mesh, hypercube), a scheme based on a center of mass principle is effective. This method identifies the processor locations for each predecessor of the candidate node. Using these processor positions in an appropriate coordinate system, and the number of data units passed as a weighting function, the center of mass of the data is calculated and rounded to the nearest processor location. Candidate processors are then limited to those within a fixed

radius of this center of mass processor, with a few processors located outside this range included to promote spreading of the load. Intuitively, the center of mass calculation compels each predecessor processor to pull the candidate node toward it, with an attractive force which is directly proportional to the amount of data communicated. While this technique is not intended as a panacea for all architectures, it is reasonable to assume that similar techniques which limit the number of processors at every step can be developed for each topology with only minor performance penalty.

The operations performed by the streamlined algorithm at each scheduling step are displayed in figure 10, with each operation classified as residing in the fixed or topology-dependent component.

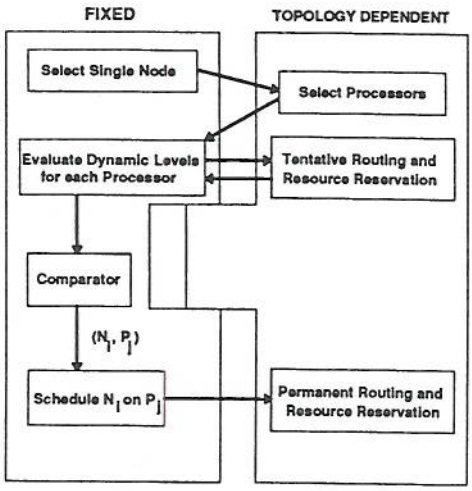


Figure 10. The algorithm specification

The simplicity of this scheme permits execution speeds suitable for a prototyping environment. A 200 node graph can be scheduled onto 16 processors in less than a minute using the current unoptimized lisp implementation.

5. Summary and Future Research

A new integrated approach called dynamic level scheduling has been presented for the compile-time scheduling of precedence graphs onto multiple processor architectures with interconnection constraints. Scheduling and routing are performed simultaneously to enable the scheduling of all communications as well as computations. Accounting for communication overheads and eliminating shared resource contention allows the scheduling of tasks with real-time constraints. This scheduling heuristic is fast, widely retargetable, applicable in arbitrary granularity scheduling environments, and displays promising performance results.

Future studies will focus on optimization techniques which were too time-consuming for the prototyping environment, but can be applied in the final design phase. An initial prepass-scheduler or limited backtracking techniques can be used to promote a more global scheduling perspective, and iterative approaches designed to reduce the scheduling bottleneck may prove beneficial. The interaction between scheduling and routing also merits further examination. Since previous communication resource reservations may block a node from being scheduled on a certain processor, the rerouting of data transfer paths may facilitate a better node-processor mapping.

6. References

1. W.W. Chu, L.J. Holloway, M.T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," *Computer*, pp. 57-69 (November 1980).
2. E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers* C-36(2)(January, 1987).
3. E.A. Lee and S. Ha, "Scheduling Strategies for Multiprocessor Real-Time DSP," *Globecom*, (November, 1989).
4. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, NY (1979).
5. V. Sarkar, "Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors," *Ph.D. Dissertation, Stanford University*, (April, 1987).
6. W.W. Chu and L.M.T. Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems," *IEEE Transactions on Computers* C-36(6) pp. 667-679 (June 1987).
7. S.W. Bollinger and S.F. Midkiff, "Processor and Link Assignment in Multicomputers using Simulated Annealing," *1988 International Conference on Parallel Processing* 1 pp. 1-7 (August, 1988).
8. K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *Computer*, pp. 50-56 (June 1982).
9. H.S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Transactions on Computers* SE-3(1) pp. 85-93 (January, 1977).
10. S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computations upon Multiprocessor Architectures," *Proceedings 1988 International Conference on Parallel Processing* 3 pp. 1-8 (August, 1988).
11. V. Sarkar and J. Hennessy, "Compile-time Partitioning and Scheduling of Parallel Programs," *Proceedings of the SIGPLAN '86 Symposium on Compiler Construction*, pp. 17-26 (July, 1986).
12. B. Greenblatt and C.J. Linn, "Branch and Bound Algorithms for Scheduling Communicating Tasks in a Distributed System," *Compcon 1987*, pp. 12-16 ().
13. T.L. Adam, K.M. Chandy, and J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," *Communications of the ACM* 17(12) pp. 685-690 (December 1974).
14. E.G. Coffman Jr., Editor, *Computer and Job Shop Scheduling Theory*, John Wiley and Sons, New York, NY (1976).
15. T.C. Hu, "Parallel Sequencing and Assembly Line Problems," *Operations Research* 9(6) pp. 841-848 (November 1961).
16. W.H. Yu, "LU Decomposition on a Multiprocessing System with Communication Delay," *Ph.D. Thesis, UC-Berkeley*, (1984).
17. R.P. Bianchini Jr. and J.P. Shen, "Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks," *IEEE Transactions on Computers* C-36(4) pp. 396-409 (April 1987).