OPTIMAL GRANULARITY OF PARALLEL EVALUATION OF AND TREES

Guo-Jie Li and Benjamin W. Wah
Department of Electrical and Computer Engineering
and the Coördinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Avenue
Urbana, IL 61801

ABSTRACT

AND-tree evaluation is an important technique in artificial intelligence and operations research. An example is the divide-and-conquer algorithm, which can be considered as the evaluation of a precedence graph consisting of two opposing AND-trees. In this paper, the optimal granularity of parallelism of AND-tree computations is quantitatively analyzed. The efficiency analysis is based on both preemptive and nonpreemptive critical-path scheduling algorithms. It is found that the optimal grain depends on the complexity of the problem to be solved, the shape of the precedence graph, and the task-time distribution along each path. The major results consist of tight bounds on the number of processors, within which the optimal grain can be sought efficiently. In view of the optimal granularity, architectural requirements for parallel AND-tree evaluations are also discussed.

INDEX TERMS: AND trees, critical-path scheduling, divide-and-conquer algorithms, granularity, processor-time efficiency, processor utilization.

1. INTRODUCTION

A wide class of problems arising in artificial intelligence, operations research, decision making, and various scientific and engineering fields involve finding a solution of a problem, which is made up of a large number of subproblems to be solved. Solving these subproblems can be represented as AND-tree computations. Examples include evaluating arithmetic expressions, searching possible solution trees of logic programs, evaluating functional programs, scheduling operations in assembly lines, finding the extremum, merge-sorting, and quick-sorting.

There are two kinds of AND-trees, *intrees* and *outtrees*. In an intree (resp. outtree), each node has at most one immediate successor (resp. predecessor), and the root is an exit node (resp. entry node). The intrees and outtrees specify the precedence relationships among the nodes. Every node is reachable from the entry node for an outtree or can reach the exit node for an intree. In recursive computations, such as divide-and-conquer algorithms, a problem is partitioned into smaller and distinct subproblems, and the solutions are found for the subproblems and are combined into a solution for the original problem. The procedure is applied recursively until the subproblems are so small that they can be solved directly. In this way, the evaluation can be viewed as a process with two phases, the decomposition of subproblems based on an outtree and the composition of results based on an intree. Hence, the precedence graph is composed of an intree and an outtree. We call this particular graph an *outin tree*. Deterministic programs can be represented by outin trees. Functional programming, which is considered as an important programming style to resolve the von-Neumann bottleneck, deals exclusively with AND-graphs [1]. Similarly, data-flow graphs are AND-graphs [7].

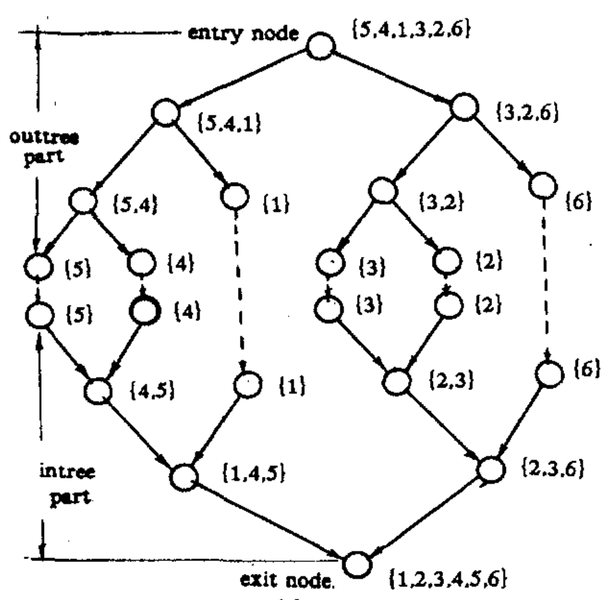
Outin trees have the following characteristics. First, there is no cycle in an outin tree. Second, in contrast to general forests, an outin tree consists of one outtree and one intree and have a one-to-one correspondence of all the leaves in the two trees. We call these leaves the *leaves of the outin tree*. In this paper, we will mainly discuss outin trees. However, the results derived apply to intrees and outtrees as well. Here, AND-trees and outin trees are used synonymously. Figure 1(a) illustrates an outin tree, which reflects the precedence relationships among tasks in a merge-sort problem shown in Figure 1(b) to sort six elements. The nonterminal nodes in the outtree part represent decompositions, each of which split a (sub-)list into two smaller sublists, whereas the nonterminal nodes in the intree part represent composition, each of which generate a sorted list based on two smaller sorted sublists.

Evaluation of outin trees naturally suggests implementation on parallel computers due to the independence of subproblems. AND-tree evaluations are important in evaluating logic programs, especially when parallel processing is used [5]. Studies conducted on parallel computers for executing divide-and-conquer algorithms can be classified into three types. First, multiprocessors that are connected in the form of a tree, especially a binary tree, can be used to exploit the potential parallelism of divide-and-conquer algorithms [13, 23]. A second approach is the virtual tree machine [2], which consists of a number of processors with private memory connected by an interconnection network, such as the binary n-cube, and a suitable algorithm to decide when and where each subproblem should be solved. The third approach is a variation of the above approaches using a common memory. All processors are connected to the memory by a common bus [14].

To evaluate an AND-tree in parallel, it is necessary to schedule the subproblems to achieve high throughput and processor utilization. An important problem is to determine the proper granularity of parallelism, that is, the minimum size of a subproblem that should be computed by a single processor. If the grain is too large, then the processors can be loosely coupled but may be under-utilized. In contrast, if the grain is too small, then the processors can be better utilized, but tight coupling may be necessary, and the communication overhead may be prohibitive. The grain must be properly chosen to obtain a proper balance between processor utilization and communication overhead.

In previous studies, one can find different points of view on the issue of granularity. Some researchers advocate a fine grain, while others suggest a coarse grain. For example, in designing the FFP machine [21], a small grain is chosen based on the hypothesis that appropriately designed small-grain multiprocessors will prove superior to large-grain ones in supporting ease and generality of parallel computations. In contrast, in

Research supported by National Science Foundation Grant DMC 85-19649.



(a)

```
Procedure mergesort
  integer low, high
  if low < high
    then call split (low, high, mid)
      call mergesort (low, mid)
      call mergesort (mid+1, high)
      call merger (low, mid, high)
  endif
end mergesort
```

(b)

Figure 1. The merge-sort problem represented as an outin tree.

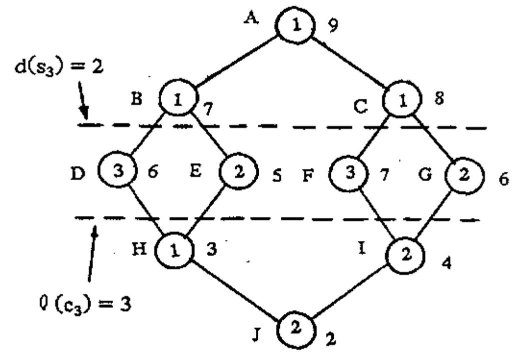
Rediflow [17], large-grain parallelism is used to minimize communication overheads. Moreover, most previous studies on granularity were discussed qualitatively. In this paper, we will analyze the optimal granularity of parallel evaluation of outin trees quantitatively. We will identify the factors that influence the optimal grain, in particular, the relationship between the optimal granularity and the problem complexity.

2. SCHEDULING PARALLEL OUTIN-TREE EVALUATIONS

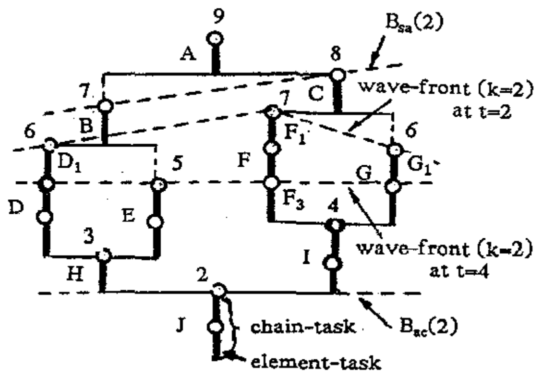
To analyze the optimal granularity of parallel evaluation of outin trees, an asynchronous model for parallel computation is adopted here. The precedence graph of an outin tree is oriented such that the entry node is at the top of the figure and the exit node is at the bottom. An arc is assumed to be always directed towards the bottom of the graph. The number inside a node is the task execution time, while the number next to a node, called its *length*, is the sum of the task execution times for nodes in the longest path from this node to the exit node. Figure 2(a) is an example of an outin tree.

The execution time of a task can be interpreted as either its maximum processing time or its expected processing time. In the former case, the worst-case time to complete the schedule is considered, while in the latter case the length of the schedule represents a rough estimate of the average time of computation. In some outin-tree problems, the execution time of each task can be predicted quite accurately. For example, in evaluating arithmetic expressions, the time to execute a primitive operation, such as a multiplication, is known. In other cases, the average execution times may have to be estimated from statistics or from previous experience. In all cases, the communication overhead is non-trivial when preemptions are allowed, and the task time should also include the overhead of preemptions.

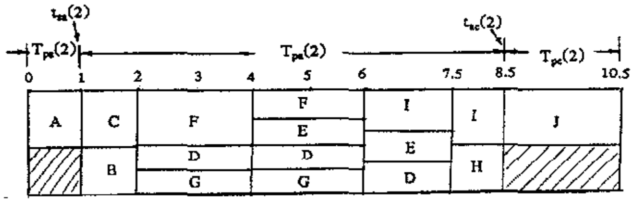
Our goal is to choose an algorithm that minimizes the maximum completion time for scheduling outin trees on a set of P identical processors, and to find the optimal granularity based on



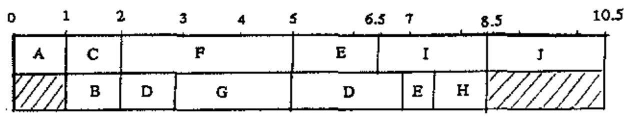
(a) Task precedence graph as outin tree.



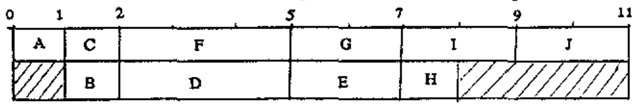
(b) Task precedence graph as e-outin tree using chain tasks.



(c) General scheduling (processor sharing) of e-outin tree in Figure 2(b).



(d) Preemptive CPS scheduling of e-outin tree in Figure 2(b).



(e) Nonpreemptive CPS scheduling of outin tree in Figure 2(a).

Figure 2. Outin tree and CPS scheduling.

this scheduling algorithm. Our scheduling problem is similar to the P/tree/C_{max} scheduling problem in which tree precedence graphs are considered [12, 6, 4]. Note that the proper granularity is related to the scheduling algorithm. If the underlying scheduling algorithm does not minimize the completion time, then the granularity found with respect to this scheduling algorithm is suboptimal. In this paper, our analysis of optimal granularity is based on optimal scheduling algorithms. The method of granularity analysis used here can be applied to choosing the best grain for other scheduling algorithms.

If preemption is allowed, P/preemption,intree/C_{max} can be solved optimally either by Muntz and Coffman's Critical Path Scheduling (CPS) algorithm in O(N) time [22], or by other polynomial-time algorithms [9]. In the CPS algorithm, the next job chosen is the one with the longest length of unexecuted jobs. This longest path is called the *critical path.* If preemption is not allowed, then optimal scheduling algorithms have been obtained only for two cases: (a) all tasks have equal execution times and the precedence relationships are in the form of an intree (Hu's algorithm) [15] and (b) when two processors are used [3]. Hu's optimal scheduling algorithm is indeed a CPS algorithm. Many other cases have been proved to be NP-hard [25, 19]. Besides efficient and optimal, the CPS algorithm is easy to implement and, consequently, is one of the most common scheduling algorithms [20, 18].

In case that the precedence graph is a tree, that all processors are identical, and that each task requires t_i, $0<t_i\leqslant t_{max}$, units of time to complete, the nonpreemptive CPS algorithm turns out to be almost-optimal in the sense that

$$T_p(k) \leqslant T_{np}(k) \leqslant T_p(k) + t_{max} \qquad (2.1)$$

where $T_{np}(k)$ and $T_p(k)$ are, respectively, the total times required by the nonpreemptive and preemptive CPS algorithms using k processors [16]. Some researchers have strived for nonpreemptive scheduling algorithms to solve scheduling problems with tree precedence [12, 8, 18]. Recently, Garey, Dolev, et al. have studied the scheduling of forests consisting of intrees and outtrees. Given a fixed number of processors, polynomial algorithms with high complexities to find an optimal schedule of these forests have been developed [10, 8].

In our research, we are interested in a special, but widely used, case of outin trees. From the point of view of optimal granularity, evaluating an outin tree can be divided into the *splitting*, *all-busy*, and *combining* phases with respect to k, the fixed number of processors. In the splitting phase, the problem is decomposed, and the number of busy processors is increased from one up to at most k−1 (the number of busy processors must always be less than k if the number of available tasks at any time is less than k). In the combining phase, the subproblems are composed, and the number of busy processors is decreased from at most k−1 to one. During these two phases, some processors are idle. In contrast, in the all-busy phase, all the k processors are busy. Schindler has proved that the schedule of a precedence graph is optimal if either the computations can be completed in only the all-busy or combining phase, or it can be partitioned into the all-busy and combining phases by a "heightline" [24]. We will show that this result can be extented to scheduling outin trees, and that the CPS algorithm guarantees the optimal preemptive scheduling and near-optimal nonpreemptive scheduling of outin-tree evaluations.

To analyze the properties of *preemptive CPS*, in short, *PCPS*, algorithms, it will be more convenient to represent a task (a node of the outin tree) of execution time t_i by a chain-task, which is t_i *element-tasks* (or *element-nodes*, or in short, *e-tasks* or *e-nodes*), each of which has one unit of execution time (see Figure 2(b)). We will use a subscript i in the task identifier to indicate the i'th e-task in the chain-task. Hence, F_2 is the second e-task of task F. The new outin tree is called the *element-outin tree* (or *e-outin tree*). For each chain-task, the e-task farthest from the exit e-node of the e-outin tree is called a *task-head* e-node. It is easy to verify that the length of the task-head e-node is the same as the length of the original multiunit task. Two e-nodes are said to be in the *same e-level* of the e-outin tree if their lengths are identical, that is, the e-level number of an e-node is equal to its length assuming that the exit e-node is in Level 0. To distinguish between nodes and levels in the original outin tree (as exemplified by Figure 2(a)) and in the e-outin tree (as exemplified by Figure 2(b)), we will use *tasks* and *levels* with respect to the original outin tree and *e-task* and *e-level* with respect to the e-outin tree in subsequent discussions.

There is another variation of preemptive scheduling algorithms called General Scheduling (GS) discipline [22], which is extremely useful in studying the granularity of parallel outin-tree evaluations. In the GS algorithm, each processor in the system is considered to have a certain amount of computing capacity rather than as a discrete unit, and this computing capacity can be assigned to tasks in any amount between zero and the equivalence of one processor. For example, if we assign half of a processor to task P_i with execution time t_i, then it will take $2\cdot t_i$ units of time to complete P_i. In the GS discipline, one processor is assigned to each of the k e-nodes farthest from the exit e-node of the e-outin tree to be evaluated. If there is a tie in the lengths among u e-nodes for the last v, u>v, processors, then v/u of a processor is assigned to each of these u e-nodes. Each time when either (a) a chain-task of the e-outin tree is completed, or (b) a point is reached where, if we continue with the present assignment, some e-nodes will be computed at a faster rate than other e-nodes that are farther from the exit e-node, then the processors are reassigned to the remaining tree according to the CPS principle. Situation (b) occurs when an e-node that is being computed has the same length as that of some unexecuted task-head e-node(s). In this case, one (or part of a) processor must be assigned to the unexecuted task-head e-node.

The GS discipline is illustrated in Figure 2(c). Muntz and Coffman have proved the equivalence between the GS and PCPS algorithms [22]. That is, if preemptions are permitted, then the "processor-sharing" capability is not needed for optimal scheduling. To illustrate this equivalence, Figure 2(d) shows the preemptive schedule for the corresponding e-outin tree in Figure 2(b). Note that in the scheduling algorithms discussed in this paper, all idle processors, if any, must be used to compute an available executable task.

In practice, preemptions are usually restricted at the beginning of a time unit, so the overhead of a practical PCPS algorithm is equal to that of Hu's algorithm, which assumes that tasks have unit execution times. From Eq. (2.1), we have [22]

$$T_p(k) = T_{gs}(k) \leqslant T_h(k) \leqslant \lceil T_{gs}(k) + 1\rceil = \lceil T_p(k) + 1\rceil \qquad (2.2)$$

where $T_h(k)$ and $T_{gs}(k)$ are the times required by Hu's and GS algorithms, respectively. Eq. (2.2) shows that the behavior of GS is very close to that of any PCPS algorithm that only allows preemptions at the beginning of a time unit. In subsequent discussions, the results will be derived without any restriction on the allowable times for preemptions. Moreover, we will use GS as a model to analyze the properties of PCPS algorithms. The granularities derived are the same as those when the PCPS algorithm is used.

At time t, an e-node is said to be *active* if either a processor or part of a processor is assigned to it. The total number of active e-nodes may be greater than the number of processors since some e-nodes may share processors. All active e-nodes form a wave-front in the e-outin-tree evaluation. Two particular times of the wave-front are of special interest: $t_{sa}(k)$ and $t_{ac}(k)$. The computation enters the all-busy phase at $t_{sa}(k)$ and enters the combining phase at $t_{ac}(k)$. In both times, the wave-fronts serve as phase-boundaries. We call the former phase-boundary $B_{sa}(k)$ and the latter $B_{ac}(k)$.

For the task graph in Figure 2(a), if the PCPS algorithm is employed, then $t_{sa}(2)=1$ and $t_{ac}(2)=8.5$ (see Figures 2(c) and 2(d)). The corresponding phase-boundaries $B_{sa}(2)$ and $B_{ac}(2)$ are indicated in Figure 2(b).

If a preemptive (resp. nonpreemptive*) CPS algorithm is applied, then the computational times required by k processors to complete the splitting, all-busy and combining phases are denoted by $T_{ps}(k)$, $T_{pa}(k)$ and $T_{pc}(k)$ (resp. $T_{nps}(k)$, $T_{npa}(k)$ and $T_{npc}(k)$). In the intree part of an e-outin tree, each e-node corresponds to a path to the exit e-node, while each e-node in the outtree part may have more than one path to the exit e-node. The longest path from an e-node to the exit e-node is selected as the *execution-path* through this e-node. For an e-node, if more than one such longest path exist, then a left-to-right orientation or any tie-breaking rule is used to break the tie. In the outtree part, if an e-node has q immediate successors, one of which is in its execution-path called the *immediate execution-successor*, then the other q−1 immediate successors serve as heads of new execution-paths, called *path-heads*. As a result, each active e-node corresponds to a unique execution-path to the exit e-node.

For example, in Figure 2(b), the execution-path from e-node A_1 is (A_1, C_1, F_1, F_2, F_3, I_1, I_2, J_1, J_2) and e-node B_1 is the head of the execution-path (B_1, D_1, D_2, D_3, H_1, J_1, J_2). Note that when k processors are used, only the topmost k−1 path-heads are active in the splitting phase. Other path-heads are active in the all-busy phase.

Let A(t,k) be the set of active e-nodes at time t when k processors are used. This set can be divided into two classes in terms of the lengths of the corresponding execution-paths. At time t, the active e-nodes whose execution-paths are the shortest among all active e-nodes belong to a subset $A_s(t,k)$, and lie in a single e-level, called the *minimal active e-level*. The other active e-nodes belong to another subset $A_h(t,k)$. (If t and k are obvious in the context, they will be omitted for brevity). For example, in Figure 2(b), when t is 2, e-nodes D_1 and G_1 belong to $A_s(2,2)$, and e-node F_1 belongs to $A_h(2,2)$.

In the following four propositions, we will show the properties of the PCPS algorithm, which are related to the optimal granularity of parallel outin-tree evaluations. It is easy to observe that the active e-nodes are executed at different rates depending on whether the assigned processor is shared or not. Let $r_i(t)$ be the processing rate in e-node per time unit of active e-node i at time t. The following proposition distinguishes the processing rates under various conditions.

Proposition 2.1: During a parallel evaluation of an outin tree with k processors, $r_i(t)$, the processing rate of active e-node i at time t, satisfies the following equations.

$$r_i(t) \begin{cases} = 1 & \text{if e-node } i \in A_h(t,k) \text{ or } |A(t,k)| \leqslant k \\ < 1 & \text{otherwise} \end{cases}$$

where |A(t,k)| is the number of active e-nodes in the set A(t,k).
Proof: This follows from the GS strategy immediately. □

Proposition 2.1 reflects the following facts. First, in the splitting and combining phases, the processing rate for any e-node is one, that is, an e-node is processed in each time unit. Second, if an active e-node is not in the minimal active e-level, then one processor (rather than part of a processor) has to be assigned to it, that is, its corresponding processing rate is one. Third, in the all-busy phase, if the number of active e-nodes is equal to the number of processors, then the processing rates for e-nodes in the minimal active e-level is also one. Only when the number of active e-nodes is larger than the number of processors used, the processing rates of e-nodes in the minimal active e-level are less than one.

Let h_{max} be the length of the critical path in the outin tree to be evaluated. Since at least one time unit is needed to complete each e-node, it is evident for any scheduling algorithm that

$$T(k) \geqslant h_{max} \tag{2.3}$$

where T(k) is the completion time using k processors under a preemptive or nonpreemptive scheduling discipline. The following proposition shows the relationship between $T_p(k)$ and the shape of the phase-boundary.

Proposition 2.2:** (a) $T_p(k) > h_{max}$ implies that all active e-nodes on the phase-boundary B_{ac} are located in the same e-level. (b) If an active e-node on the phase-boundary B_{ac} belongs to A_h, then $T_p(k) = h_{max}$.

The example in Figure 2 illustrates Property (a) above. At time t=4, e-node G_1 in the critical path "enters" the set A_s, and hereafter all active e-nodes are in the same e-level. Proposition 2.2 reflects the fact that preemptive scheduling algorithms distribute work uniformly among the available processors, thereby reducing the computational time required in the combining phase. This is the reason for a preemptive algorithm to run faster than the nonpreemptive counterpart.

To investigate the optimal granularity, we need to examine the phase-boundaries when different numbers of processors are used. The following proposition compares two boundaries with respect to k and k+1 processors. In subsequent discussions, the phase-boundary in a *single e-level* will mean that all active e-nodes on this boundary have execution-paths with the same length.

Proposition 2.3:** If phase-boundary $B_{ac}(k+1)$ is in a single e-level, then the phase-boundary $B_{ac}(k)$ must be in a single e-level.

If the phase-boundary $B_{ac}(k+1)$ is not in a single e-level, then there are k,' 0<k'<k, active e-nodes belonging to $A_h(k+1)$ on this phase-boundary. During the splitting and all-busy phases, we can partition the (k+1) processors into two groups. The first group consists of k' processors that only evaluate e-nodes in the k' execution-paths from the topmost k' path-heads to the k' e-nodes. Other (k+1−k') processors constitute the second group, which evaluate all e-nodes in the two phases except for e-nodes in the aforementioned k' execution-paths. Accordingly, we can prove the following proposition.

Proposition 2.4:** If an outin tree is evaluated by the PCPS algorithm, then $T_{ps}(k+1) \geqslant T_{ps}(k)$, and $T_{pc}(k+1) \geqslant T_{pc}(k)$.

The following theorem shows that the PCPS algorithm can be used to find the optimal preemptive schedule for outin trees.

Theorem 2.1.: PCPS is a minimum-completion-time scheduling algorithm for an outin tree.
Proof: Let $\Phi_{ps}(k)$ and $\Phi_{pc}(k)$ be the total amount of idle times in the splitting and combining phases when the PCPS algorithm is applied and k processors are used. Clearly,

$$T_p(k) = \frac{\Phi_{ps}(k) + T(1) + \Phi_{pc}(k)}{k} \tag{2.4}$$

Minimizing $T_p(k)$ implies minimizing $(\Phi_{ps}+\Phi_{pc})$. In the PCPS algorithm, once an e-node is available, that is, its predecessor node has been finished, a processor is assigned to it immediately. The time spent in the splitting phase for any schedule cannot be shorter than that in the PCPS algorithm. It also means that Φ_{ps} for the PCPS algorithm is the minimum. We now consider Φ_{pc}. If the phase-boundary B_{ac} is not in a single e-level, then $T_p(k)=h_{max}$ according to Proposition 2.2. That is, the PCPS algorithm achieves the minimum computational time T(k) according to Eq. (2.3). What we need to consider is the case when the phase-boundary B_{ac} is in a single e-level. This boundary is indicated by a dark line B in Figure 3.

Suppose that an arbitrary scheduling algorithm is used, the corresponding phase-boundary is denoted by B', which is shown as a dashed line in Figure 3. Note that it is impossible for all e-nodes on boundary B' to be beneath boundary B, otherwise at least one processor is idle before the wave-front achieves B,' which implies that B' is not a phase-boundary. In other words, at least one e-node on boundary B' is above or on boundary B_{ac}. Similarly, it is impossible for all e-nodes of boundary B' to be

* We will discuss nonpreemptive CPS algorithms in Section 4.

** Due to space limitation, the proofs of Propositions 2.2 thru 2.4, Lemmas 3.1 thru 3.4, and all lemmas and theorems in Section 4 are omitted.

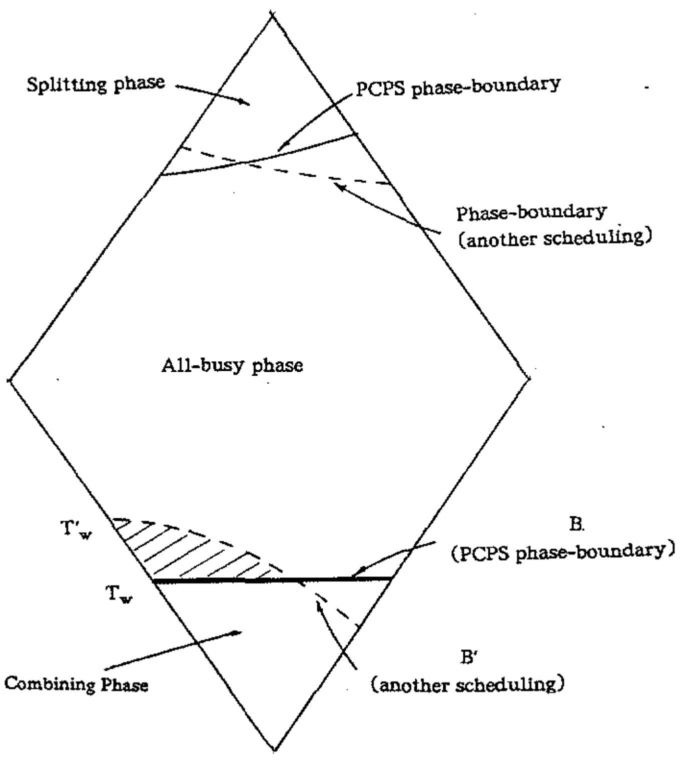


Figure 3. Proof of Theorem 2.1.

above boundary B.

Let N_{pc} and N_c' be the amount of task times in the combining phase of the PCPS and another scheduling algorithm. Let $T_c'(k)$ and $\Phi_c'(k)$ be the computational time and the total idle time in the combining phase when an arbitrary scheduling algorithm using k processors is adopted. If $T_{pc}(k)$ equals $T_c'(k)$, then $N_{pc}(k) \geqslant N_c'(k)$, hence, $(\Phi_c' - \Phi_{pc}) = (N_{pc} - N_c') \geqslant 0$. If $T_{pc}(k)$ is less than $T_c'(k)$, since at least one e-node on boundary B' is beneath or on boundary B_{ac}, $(N_c' - N_{pc})$ cannot be larger than the amount of task times for e-tasks beneath B' and above B (the shaded area in Figure 3). Since, from Proposition 2.1, the processing rate for any path in the combining phase is one, then after $(T_c' - T_{pc})$ time units, all e-nodes in the shaded area in Figure 3 must be completed. As less than k e-nodes can be completed during a time-unit in the combining phase, the amount of e-nodes in the shaded area must be less than $k(T_c' - T_{pc})$. Therefore,

$$(\Phi_c' - \Phi_{pc}) = [k(T_c' - T_{pc}) - (N_c' - N_{pc})] > 0.$$

This means that Φ_{pc}, the total idle time in the combining phase, is also minimum for the PCPS algorithm. The proof does not imply that the PCPS algorithm is the unique optimal algorithm, but rather that the amount of idle times introduced by the PCPS algorithm is the minimum. □

3. OPTIMAL GRANULARITY IN PREEMPTIVE SCHEDULING

The criteria generally used to define the optimal granularity are the processor utilization (PU), kT^2, and AT^2, where k is the number of processors, T is the computational time, and A is the area of a VLSI implementation. The complexity of divide-and-conquer algorithms in an SIMD model and the conditions to assure the optimal processor utilization have been studied [14]. However, processor utilization increases monotonically with decreasing number of processors, which means that PU achieves the maximum when one processor is used. Hence, PU is not an adequate measure for the effects of parallel processing. A more appropriate measure is the kT^2 criterion, which considers both PU and computational time, since

$$kT^2(k) = \frac{T(1)T(k)}{PU} \quad \text{where } PU = \frac{\text{speedup}}{k} \text{ and } T(k) = \frac{T(1)}{\text{speedup}}$$

To minimize kT^2 means to reduce the computational time and to maximize the processor utilization. kT^2 is linearly related to AT^2 if the area of connection wires is proportional to the area of processing elements, as in systolic arrays. Both computational time and processor utilization are important in many applications, hence, kT^2 is a good criterion to optimize. In other applications, such as real-time processing, the completion time may be more critical and the PU is a secondary consideration. In this case, a different optimization criterion may have to be used.

In this paper, we have adopted kT^2 as a criterion of processor-time efficiency to derive the optimal granularity for parallel outin-tree evaluations. That is, given an outin-tree, we need to either choose k to minimize kT^2, or given a fixed k, determine the type of outin trees (their shapes, complexities, etc.) and its proper size that can be solved most efficiently by this system.

It is difficult to find the optimal granularity with respect to kT^2 directly because the optimal granularity depends on the execution time of each task and the shape of the outin tree. We now try to find an efficient and systematic method to determine the optimal grain via an intermediate variable, the total idle time. Let $\Phi_p(k)$ (resp. $\Phi_{np}(k)$) be the total amount of idle times when a preemptive (resp. nonpreemptive) scheduling algorithm with k processors is used. $\Phi_p(k)$ takes into account the idle times in both the splitting and combining phases. Clearly, $\Phi_p(k) = [\Phi_{ps}(k) + \Phi_{pc}(k)]$ and

$$kT(k) = T(1) + \Phi(k) \tag{3.1}$$

Eq. (3.1) holds for both preemptive and nonpreemptive scheduling algorithms.

The total idle time $\Phi_p(k)$ is related to both k and kT^2. The following two lemmas show the difference between the total idle times when different number of processors are used.

Lemma 3.1:** Suppose that an outin tree is evaluated by the PCPS algorithm, then

$$[\Phi_p(k+1) - \Phi_p(k)] \leqslant h_{max} \tag{3.2}$$

where h_{max} is the length of the critical path.

Lemma 3.2:** Suppose that an outin tree is evaluated by the PCPS algorithm, then

$$[\Phi_p(k+1) - \Phi_p(k)] \geqslant [T_{ps}(k) + T_{pc}(k)] > 0 \tag{3.3}$$

Lemma 3.3:** Suppose that an outin tree is evaluated by the PCPS algorithm, then

$$T_p(k+1) \leqslant T_p(k)$$

The above lemmas reveal that when the number of processors used are increased, the total idle times must increase, and the difference of the total idle times with respect to k_1 and k_2 processors is bounded by $(k_2 - k_1)[T_{ps}(k) + T_{pc}(k)]$ and $(k_2 - k_1) \cdot h_{max}$, respectively. From these facts, we can determine the conditions under which kT^2 is either monotonically increasing or decreasing with respect to k. The following theorem shows the relation between $\Phi_p(k)$ and kT^2.

Theorem 3.1: Suppose that an outin tree is evaluated by the PCPS algorithm. $kT_p^2(k)$ is monotonically increasing with k if $[\Phi_p(k+1) - \Phi_p(k)] > T_p(k)/2$. $kT_p^2(k)$ is monotonically decreasing with k if $[\Phi_p(k+1) - \Phi_p(k)] < T_p(k)/(2 + 1/k)$.

Proof: By Eq. (3.1), we get

$$(k+1)T_p^2(k+1) - kT_p^2(k) \tag{3.4}$$
$$= \frac{[T_p(1)+\Phi_p(k+1)]^2}{k+1} - \frac{[T_p(1)+\Phi_p(k)]^2}{k}$$
$$= \frac{k[T_p(1)+\Phi_p(k+1)]^2 - k[T_p(1)+\Phi_p(k)]^2 - [T_p(1)+\Phi_p(k)]^2}{k(k+1)}$$
$$= \frac{[\Phi_p(k+1)-\Phi_p(k)]\,[2T_p(1)+\Phi_p(k+1)+\Phi_p(k)] - kT_p^2(k)}{k+1}$$
$$= \frac{[\Phi_p(k+1)-\Phi_p(k)]\,[(k+1)T_p(k+1)+kT_p(k)] - kT_p^2(k)}{k+1}$$

Since, from Lemma 3.2, $\Phi_p(k+1) > \Phi_p(k)$, hence

$$(k+1)T_p(k+1) = [T_p(1)+\Phi_p(k+1)] \tag{3.5}$$
$$> [T_p(1)+\Phi_p(k)] = kT_p(k)$$

From Eq's (3.4) and (3.5), we conclude that if $[\Phi_p(k+1)-\Phi_p(k)] > T_p(k)/2$, then $(k+1)T_p^2(k+1) > kT_p^2(k)$. By Lemma 3.1, Eq's (3.4) and (2.3), we obtain the following condition.

If $\quad [\Phi_p(k+1) - \Phi_p(k)] < \dfrac{T_p(k)}{2+1/k}$,

then $\quad [(k+1)T_p^2(k+1) - kT_p^2(k)]$

$$< \left| \frac{kT_p(k)}{2k+1}\,[2kT_p(k)+T_p(k)] - kT_p^2(k) \right| = 0 \quad \square$$

Theorem 3.1 restricts the region within which we need to find a value k that minimizes $kT_p^2(k)$. In other words, the approximate condition that adding a processor will not degrade the processor-time efficiency is that all processors will be busy at least half of the time.

In the example shown in Figure 2, $T_p(1) = 18$, $T_p(2) = 10.5$, $\Phi_p(2) = 3$, $T_p(3) = 9$, and $\Phi_p(3) = 9$. (Readers are suggested to schedule this outin tree with three processors). Since $\Phi_p(2) = 3 < T_p(1)/3 = 6$, and $[\Phi_p(3) - \Phi_p(2)] = 6 > T_p(2)/2 = 5.25$, according to Theorem 3.1, we can conclude that the use of two processors minimizes kT^2 for this outin tree.

A question about the monotonicity of $[kT_p^2(k+1) - kT_p^2(k)]$ now arises naturally. If $[kT_p^2(k+1) - kT_p^2(k)]$ is increasing monotonically with k, then $kT_p^2(k)$ is a unimodal function of k, and the optimal value of k can be found easily. This monotonicity will be proved in the following theorem.

Theorem 3.2: Suppose that an outin tree is evaluated by the PCPS algorithm, then $kT_p^2(k)$ is a concave function of k, that is, $kT^2(k)$ achieves the minimum when $k = k'$, and $kT^2(k)$ is monotonically decreasing (resp. increasing) with k when $k < k'$ (resp. $k > k'$).

Proof: To show $kT_p^2(k)$ is a concave function of k, we need to prove that its second-order difference is positive, namely, $[(k+2)T_p^2(k+2) - (k+1)T_p^2(k+1)] > [(k+1)T_p^2(k+1) - kT_p^2(k)]$. Let $\Delta(kT_p^2(k)$ denote $[(k+1)T_p^2(k+1) - kT_p^2(k)]$. Then

$$\Delta(kT_p^2(k)) = k[T_p^2(k+1) - T_p^2(k)] + T_p^2(k+1) \tag{3.6}$$
$$\Delta((k+1)T_p^2(k+1)) = (k+1)[T_p^2(k+2) - T_p^2(k+1)] + T_p^2(k+2) \tag{3.7}$$

Subtracting Eq. (3.6) from Eq. (3.7) and applying Eq. (3.1) yields

$$\Delta((k+1)T_p^2(k+1)) - \Delta(kT_p^2(k)) \tag{3.8}$$
$$= (k+2)[T_p^2(k+2) - T_p^2(k+1)] - k[T_p^2(k+1) - T_p^2(k)]$$
$$= [T_p(k+2) + T_p(k+1)]\,[\Phi_p(k+2) - \Phi_p(k+1) - T_p(k+1)]$$
$$- [T_p(k+1) + T_p(k)]\,[\Phi_p(k+1) - \Phi_p(k) - T_p(k+1)]$$

From Eq's (2.3) and (3.8) and Lemmas 3.1, 3.2 and 3.3, we conclude that $\{\Delta((k+1)T_p^2(k+1)) - \Delta(kT_p^2(k))\} > 0$. $\square$

We have found the condition under which kT_p^2 is increased or decreased based on the intermediate variable, $\Phi_p(k)$, and that kT_p^2 is a concave function. Next, we will determine the number of processors such that kT_p^2 is minimum for a given outin tree.

Note that in the original outin tree (see Figure 2(a)), each node is a multiunit task, and tasks in a level may have different lengths. If there are m(i) tasks in level i, then there are m(i) paths from level i to the exit node. Among these paths, the minimum length is denoted by $\ell(i)$. Similarly, we can define the *depth* of a node as the sum of task-times along a path from the entry node to and including this node, and denote the shortest depth from the entry node to level i by d(i).

Given k processors, we can find c_k, a particular level in the intree part of the original outin tree, such that $m(c_k)$, the number of tasks in this level, is less than k, but $m(c_k+1) \geqslant k$. This particular level is called the *minimum-all-busy level*. Likewise, in the outtree part, there is a level called the *maximum-all-busy level* and denoted by s_k, such that $m(s_k) < k$ and $m(s_k-1) \geqslant k$. By recognizing the minimum-all-busy and maximum-all-busy levels, we can roughly estimate the locations of the phase-boundaries. Recall from Proposition 2.2 that $T_p(k) = h_{max}$ if the phase-boundary B_{ac} is not in a single e-level. In this case, $(k+1)T_p^2(k+1) > kT_p^2(k)$. To achieve the minimum $kT_p^2(k)$, the number of processors should be reduced until the phase-boundary appear in a single e-level. (When $B_{ac}(k)$ is not in a single e-level but $B_{ac}(k-1)$ is, $kT_p^2(k)$ may be minimum.) This observation shows that the use of the minimum-all-busy level to estimate T_{pc} is accurate in most cases. The following lemma shows that the shortest length from the minimum-all-busy (resp. maximum-all-busy) level to the exit (resp. entry) node gives the lower-bound computational time in the combining (resp. splitting) phase.

Lemma 3.4:[**] Suppose that an outin tree is evaluated by k processors. If s_k and c_k are the maximum-all-busy and minimum-all-busy levels, then (a) $T_{ps}(k) \geqslant d(s_k)$, and (b) $T_{pc}(k) \geqslant \ell(c_k)$. Further, if the phase-boundary $B_{ac}(k)$ lies in a single e-level, then $T_{pc} = \ell(c_k)$.

This lemma is illustrated by the example in Figure 2(a). Suppose that three processors are used, the maximum-all-busy level contains tasks B and C, and the minimum-all-busy level contains tasks H and I. $T_{ps}(3)$ (resp. $T_{pc}(3)$) cannot be less than 2 (resp. 3) because if either task B or C, which are associated with the shortest depth from the entry node to this level, is not finished, then the computation cannot enter the all-busy phase. Likewise, if task H has been assigned to a processor, then the computation must have entered the combining phase. Since $B_{ac}(2)$ lies in a single e-level, $T_{pc}(2) = \ell(c_2) = 3$.

For an arbitrary outin tree,

$$T_{ps} \geqslant t_{en}, \; T_{pc} \geqslant t_{ex}, \; \text{and} \; \Phi_p \geqslant (t_{en} + t_{ex}) \tag{3.9}$$

where t_{en} and t_{ex} are the task times of the entry and exit nodes, respectively.

When more than one processor are used, some processors must be idle when the entry and exit nodes are evaluated. If the times spent in evaluating the entry and exit nodes dominate over all other computations, then parallel processing is definitely inefficient. The following corollary identifies the condition under which sequential computation is better than parallel processing.

Corollary 3.1: Suppose that an outin tree is scheduled by the PCPS algorithm and $(t_{en}+t_{ex}) > T(1)/2$, then sequential processing, i.e., k=1, achieves the minimum kT^2.

Proof: This follows from Theorem 3.1 immediately. $\square$

Having proved a series of propositions, lemmas, and theorems, the main theorem to derive the optimal granularity under the PCPS scheduling algorithm can be obtained now. In the following theorem, the region on k in which we can find the optimal granularity of parallel outin-tree evaluation is given.

Theorem 3.3: Suppose that an AND-tree is evaluated by the PCPS algorithm and k > 1, then

$$(k+1)T_p^2(k+1) > kT_p^2(k) \text{ if } k > \frac{2T_p(1)}{h_{max}} \text{ and} \tag{3.10}$$

$$(k+1)T_p^2(k+1) < kT_p^2(k) \text{ if } k < \left\{\frac{T_p(1)+t_{en}+t_{ex}}{2h_{max}} - \frac{1}{2}\right\} \tag{3.11}$$

Proof: From Lemma 3.2,

$$\Delta\Phi_p(k) \geqslant T_{ps}(k) + T_{pc}(k). \tag{3.12}$$

Since the idle time of each processor cannot be larger than $(T_{ps}+T_{pc})$, we have

$$\Phi_p(k) \leqslant (k-1)\,[T_{ps}(k) + T_{pc}(k)] \tag{3.13}$$

By Theorem 3.1, Eq's (3.1), (3.12) and (3.13), the condition that guarantees the monotonic increase of $kT_p^2(k)$ with k is

$$(k+1)T_p^2(k+1) > kT_p^2(k) \tag{3.14}$$

$$\text{if } [T_{ps}(k)+T_{pc}(k)] > \frac{T_p(1) + (k-1)[T_{ps}(k)+T_{pc}(k)]}{2k}.$$

On the other hand, when all tasks but those in the critical path can be completed by (k-1) processors during $h_{max} - [T_{ps}(k)+T_{pc}(k)]$ time-units, i.e.,

$$(k-1) > \frac{T_p(1) - h_{max}}{h_{max} - [T_{ps}(k)+T_{pc}(k)]}, \tag{3.15}$$

the phase-boundary B_{sc} must not be in a single e-level. As discussed before, the optimal grain cannot be larger than the RHS of Eq. (3.15) plus one. By Lemma 3.4, Eq's (3.14) and (3.15),

$$(k+1)T_p^2(k+1) > kT_p^2(k) \text{ if}$$

$$k > \min\left\{\frac{T_p(1)}{d(s_k)+\ell(c_k)} - 1,\ \frac{T_p(1)}{h_{max} - [d(s_k)+\ell(c_k)]}\right\} \tag{3.16}$$

The condition described in Eq. (3.10) is obtained from Eq. (3.16).

Note that $\Delta\Phi_p(k) \leqslant h_{max}$, and that $kT_p(k) > [T_p(1)+t_{en}+t_{ex}]$ according to Lemma 3.1, Eq's (3.1) and (3.9). By Theorem 3.1, the following result can be derived.

$$(k+1)T_p^2(k+1) < kT_p^2(k) \text{ if } h_{max} < \frac{T_p(1) + t_{en} + t_{ex}}{2k+1}$$

which is equivalent to Eq. (3.11). □

To find optimal granularity, we need to search the small region of k defined by Theorem 3.3. The lower and upper bounds of this region are, respectively, $\{T_p(1)/(2h_{max}) - 1\}$ and $\{2T_p(1)/(h_{max}) + 1\}$. Note that we have not made any assumption about the distribution of task times in deriving these bounds. Since kT_p^2 is a concave function of k (Theorem 3.2), the desirable number of processors can be found easily by a binary search. The binary search can be completed within about $\log_2(T_p(1)/h_{max})$ steps. Each step in the binary search tests whether $\Delta(kT_p^2(k))$ is positive. If it is, then a smaller value of k will be checked in the next step, otherwise, a larger k will be tested.

For any k inside the search region, the phase-boundary $B_{sc}(k)$ is in a single e-level, hence the location of $B_{sc}(k)$ can be uniquely determined without knowing the detailed schedule. Accordingly,

$$\Phi_p(k) = k[T_{ps}(k) + T_{pc}(k)] - [N_{ps}(k) + N_{pc}(k)]$$

where $N_{ps}(k)$ and $N_{pc}(k)$ are the amount of task times in the splitting and combining phases. From Lemma 3.4, $T_{pc}(k) = \ell(c_k)$, and $T_{ps}(k)$ can be found directly from the e-outin tree. As a result,

$$kT_p^2(k) = \frac{[T_p(1) + \Phi_p(k)]^2}{k}$$

For instance, suppose that N items need to be sorted. It is well-known that $T(1) = N\cdot\log_2 N$ if a merge-sort algorithm is used. In this case, the overhead in the intree part dominates that of the outtree part. For the intree part, $h_{max} = N + N/2 + \cdots + 1 = 2N-1$, so the lower and upper bounds of the search region can be determined from Theorem 3.3, which are close to $(\log_2 N)/4$ and $\log_2 N$, respectively. Since there are only $(3\cdot\log_2 N)/4$ candidate values in this search region, $\log_2\log_2 N$ steps of a binary search can guarantee to find the optimal grain of parallel merge sorting. For problems such as evaluating numerical or logic expressions and finding the maximum (or minimum) value, all task times are identical. Theorem 3.3 predicts that the optimal grain is between $N/(2\cdot\log_2 N)$ and $2N/\log_2 N$. Figure 4 shows the simulation results of applying a nonpreemptive CPS algorithm to a binary intree of 4096 terminal notes and $t_i=1$ for all i. Since all tasks have unit execution times, the performance of the nonpreemptive CPS algorithm is very close to that of GS algorithm (see Eq. (2.2)). In this example, kT^2 is minimum when 431 processors are used, which is between $N/(2\cdot\log_2 N)$ (=170) and $2N/\log_2 N$ (=683).

The above analysis reveals that the optimal grain of an outin-tree evaluation is related to the following parameters:

(a) T(1), the time required by a sequential evaluation, which is the sum of all task times in the outin tree;
(b) h_{max}, the length of the critical path;
(c) $d(s_k)$, the shortest depth from the entry node to s_k, the maximum-all-busy level; and
(d) $\ell(c_k)$, the shortest length from c_k, the minimum-all-busy level, to the exit node (recall that c_k and s_k depend on k).

$T(1)/h_{max}$ reflects the shape of the outin tree, while $T(1)/d(s_k)$ and $T(1)/\ell(s_k)$ reflect the distribution of the task-times. If the outin tree is "wide" and nearly balanced, i.e., $T_p(1)/h_{max}$ is large, then a fine grain is more appropriate, otherwise, a coarse grain is more suitable. Further, if tasks in levels closer to the entry and exit nodes have longer execution times, i.e., $T_p(1)/[d(s_k)+\ell(c_k)]$ is small, then the optimal grain should be larger, otherwise, a finer grain is better with respect to kT^2. Both $T(1)/h_{max}$ and $T(1)/[d(s_k)+\ell(c_k)]$ are related to the problem complexity. We will again show the influence of problem complexity on the

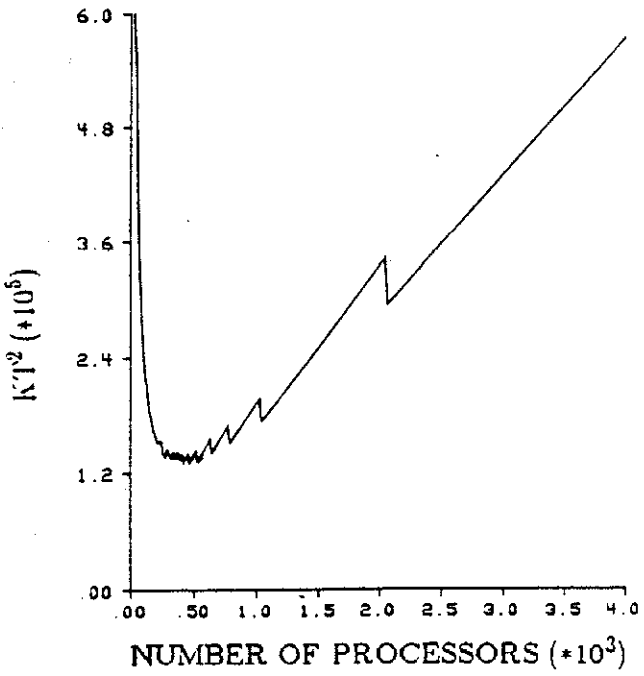


Figure 4. Simulation results to find the optimal granularity of evaluating an intree with 4096 leaves and unit execution times for all nodes.

optimal granularity in the next section, where nonpreemptive algorithms will be used.

4. OPTIMAL GRANULARITY IN NONPREEMPTIVE SCHEDULING

Nonpreemptive CPS algorithms are similar to the PCPS algorithm except that preemption is not allowed. In the nonpreemptive CPS algorithm, one processor is assigned to each of the k nodes farthest from the exit node. If there is a tie in lengths among more than one node, then a left-to-right tie-breaking rule is used to assign a processor to one of these nodes. When a task of the outin tree is completed, the free processor is assigned to the node farthest from the root in the remaining outin tree to be evaluated. Figure 2(e) illustrates an example of nonpreemptive CPS scheduling. In general, nonpreemptive scheduling is more practical due to the smaller task-switching overheads; however, it is more difficult to predict its performance and determine the optimal grain in parallel processing.

The problem of determining the optimal granularity of nonpreemptive CPS algorithm is complicated by its anomalous behavior. Graham has proved that if an AND-tree is evaluated twice by using k_1 and k_2 processors, respectively [11], then

$$\frac{T_{np}(k_1)}{T_{np}(k_2)} \leqslant \left\{ 1 + \frac{k_2 - 1}{k_1} \right\}$$

The above inequality implies that the anomaly $T_{np}(k+1)/T_{np}(k) < k/(k+1)$ is possible. In other words, $kT_{np}^2(k)$ is generally not a concave function of k and cannot be searched by a binary search or other efficient search methods.

In a special case, if the execution times of tasks of an outtree are monotonically decreasing as the tree is decomposed, then it will be shown below that $\Phi_{np}(k_2) > \Phi_{np}(k_1)$ holds for $k_2 > 2k_1$. Likewise, the same relation holds for the case when the execution times of tasks of an intree are monotonically increasing as the tree is composed. Here, the optimal granularity of outin-tree computations based on a nonpreemptive CPS algorithm can be bounded in a relatively small region. The assumption on monotonic distribution of task times is valid in divide-and-conquer algorithms.

In this section, we will develop conditions under which kT^2 is monotonically increasing or decreasing with k for the special case in which the task times are monotonically decreasing in the outtree and monotonically increasing in the intree. We will investigate the difference of the total idle times with respect to different number of processors under nonpreemptive CPS. The following lemma gives the lower and upper bounds of $[\Phi_{np}(k_2) - \Phi_{np}(k_1)]$.

Lemma 4.1:[**] Suppose that an outin tree is scheduled by a nonpreemptive CPS algorithm and that $t_i > t_j$ if task i is a predecessor (resp. successor) of task j in the outtree (resp. intree) part, then

$$[\Phi_{np}(k_2) - \Phi_{np}(k_1)] \geqslant \{(k_2 - k_1)[T_{ps}(k_1) + T_{pc}(k_1)] - k_1 t_{npa}(k_1)\} > 0 \quad \text{if } k_2 > 2k_1 \qquad (4.1)$$

$$[\Phi_{np}(k_2) - \Phi_{np}(k_2)] \leqslant \{(k_2 - k_1)[T_{ps}(k_2) + T_{pc}(k_2)] + k_2 t_{npa}(k_2)\} \quad \text{if } k_2 > k_1 \qquad (4.2)$$

where $t_{npa}(k)$ is the longest task-time among all tasks in the all-busy phase when k processors are used.

We should point out that the above lemma holds for the case in which a part of the phase-boundary is in the intree and another part is in the outtree. The above lemma is true because the task time of a node in the all-busy phase is less than either T_{ps} or T_{pc} from the assumption of monotonically distributed task times. That is, $[T_{ps}(k_1) + T_{pc}(k_1)] > t_{npa}(k_1)$ is always true regardless the location of the phase-boundary.

Similar to Theorem 3.1, we first study the relationship between kT^2 and the idle times. The following theorem gives the conditions under which kT^2 is monotonically increasing or decreasing based on the intermediate variable $\Phi_{np}(k)$.

Theorem 4.1:[**] Suppose that an outin tree is scheduled by a nonpreemptive CPS algorithm and that $t_i > t_j$ if task i is a predecessor (resp. successor) of task j in the outtree (resp. intree) part, then

$$k_2 T_{np}^2(k_2) > k_1 T_{np}^2(k_1) \text{ if } [\Phi_{np}(k_2) - \Phi_{np}(k_1)] > \left\lfloor \frac{k_2 - k_1}{2} T_{np}(k_1) \right\rfloor \text{ and } k_2 > 2k_1; \qquad (4.7)$$

$$k_2 T_{np}^2(k_2) < k_1 T_{np}^2(k_1) \text{ if } [\Phi_{np}(k_2) - \Phi_{np}(k_1)] < \left\lfloor \frac{2(k_2 - k_1)k_1}{2k_1 + 3k_2} T_{np}(k_1) \right\rfloor \text{ and } k_2 > k_1 \qquad (4.8)$$

The main theorem to find the optimal granularity can be derived from Theorem 4.1. Before this theorem is proved, the following lemma is needed.

Lemma 4.2:[**] For a given outin tree, suppose that both PCPS and nonpreemptive CPS algorithms are applied, then $[T_{nps}(k) + T_{npc}(k)] \leqslant [T_{ps}(k) + T_{pc}(k) + t_{npa}(k)]$.

The example in Figure 2 illustrates this lemma. Here, $[(T_{nps} + T_{npc}) - (T_{ps} + T_{pc})] = 1$, which is less than t_{npa} (=3).

Theorem 4.2:[**] Suppose that an outin tree is scheduled by a nonpreemptive CPS algorithm and that $t_i > t_j$ if task i is a predecessor (resp. successor) of task j in the outtree (resp. intree) part, then k, the number of processors that minimizes $kT_{np}^2(k)$, is bounded between $[T_{np}(1) + t_{en} + t_{ex}]/(8h_{max})$ and $3T_{np}(1)/[d(s_k) + \ell(c_k) - 2t_{npa}(k)]$

As an example, we can determine the area within which the optimal granularity can be found for the parallel merge-sort of N elements. In this problem, the computational overhead in the intree is dominant, so only the part of the intree has to be considered in the scheduling. From Theorem 4.2, the lower bound of the search region is $(\log_2 N)/16$, since $T_{np}(1) = N \cdot \log_2 N$ and $h_{max} < 2N$. If N is large enough, then $[d(s_k) + \ell(c_k) - 2t_{npa}(k)]$ will be larger than 1.5N, hence, the upper bound of the search region is $2 \cdot \log_2 N$.

Comparing these bounds with Theorem 3.3, we see that the range within which an optimal-grain for a nonpreemptive schedule can be found is larger than that of a preemptive schedule. Moreover, kT^2 is not monotonically decreasing or increasing with k for nonpreemptive scheduling, i.e., kT^2 is not a unimodal function of k, hence, an exhaustive search is required to find the optimal grain.

To predict the optimal order-of-magnitude granularity in general, we now briefly discuss the asymptotically optimal granularity of parallel outin-tree evaluations with nonpreemptive CPS algorithm. Let C(n) be the overhead of a node in the intree, which has n leaves rooted by this node. C(n) represents the overhead of combining the results from its immediate predecessor nodes in the intree. Likewise, let D(n) be the overhead of a node in the outtree, which has n leaves rooted by this node. D(n) represents the overhead of decomposing the given node into its immediate successor nodes in the outtree. For an outin tree with N leaves, C(N) and D(N) represent the overheads of the exit and entry nodes, respectively. Let Θ be the set of functions of the same order. For problems such as summing a set of numbers, finding the maximum of N numbers, and returning logical values to the main goal in evaluating logic programs, $C(n) = \Theta(1)$. In quicksort and merge sort, $C(n) + D(n) = \Theta(n)$.

The asymptotically optimal grain depends on the complexities of C(n) and D(n). The higher the order-of-magnitude complexity of C(n) and D(n) are, the larger the granularity is. When the order-of magnitude complexity of C(n) (and/or D(n)) is large, the time spent in the combining (and/or splitting) phase is dominating the time in the all-busy phase, and the performance gain in the all-busy phase with finer grains is negligible. In other words, a small granularity may result in underutilization of processors.

To isolate the impact of the complexities C(n) and D(n) on the optimal granularity from the shape of the outin tree, we discuss the complete binary outin tree, and assume that, for all nodes in a level of the intree (resp. outtree) part, the order-of-magnitude complexities of C(n) (resp. D(n)) are identical. This assumption enables us to estimate T(1). The following theorem gives the condition under which the asymptotically minimum kT^2 is achieved for various complexities of C(n) and D(n).

Theorem 4.3:** Suppose that a nonpreemptive CPS algorithm is applied to evaluate an outin tree of N leaves by k processors. Assume that, for all nodes in a level of outin tree, the order-of-magnitude complexities of C(n) (and D(n)) are the same and that $t_i > t_j$, if task i is a predecessor (resp. successor) of task j in the part of the outtree (resp. intree). Then the order-of-magnitude $kT_p^2(N)$ is the minimum if $\Theta(T_{nps}(k(N))) = \Theta(T_{nps}(k(N))+T_{npc}(k(N)))$.

The above theorem shows that if the number of leaves of an outin tree is very large, then, to achieve the minimum $kT_p^2(k)$, the number of processors should be chosen such that the times required by the all-busy phase and the total times required by the other two phases are approximately equal. This result also shows the relationship between the processor utilization and kT^2. Let N_{sc} be the amount of task-time in the splitting and combining phases, and $T_{nps}(k) = [T_{nps}(k)+T_{npc}(k)]$. Then, for arbitrary outin tree computations, an asymptotically optimal granularity is achieved when

$$PU(k) = \frac{kT_{npa}(k) + N_{sc}}{2kT_{npa}(k)}$$

Since $T_{nps}(k) \leqslant N_{sc} \leqslant (k-1)T_{npa}(k)$, we conclude that the corresponding processor utilization is between 0.5 and 1. In other words, when a problem is solved by a parallel divide-and-conquer algorithm and there are a large number of leaves in its outin-tree representation, to pursue more than 50% processor utilization will reduce the utilization-time efficiency. According to Theorem 4.3, the asymptotically optimal granularities with respect to various C(n)+D(n) are summarized in Table 1.

Complexity of C(n)+D(n) $1 \leqslant n \leqslant N$	Optimal Granularity	Architectural Requirements
$\Theta(\log_2^s n)$ $s \geqslant 0$	$\Theta\left(\frac{N}{\log_2^{s+1} N}\right)$	A very large number of processors; tree or other efficient interconnection
$\Theta(n^r \log_2^s n)$ $0<r<1$ $s \geqslant 0$	$\Theta\left(\frac{N^{1-r}}{\log_2^s N}\right)$	A large number of processors; tree or other efficient interconnection
$\Theta(n \log_2^s n)$ $s \geqslant 0$	$\Theta(\log_2 N)$	A small number of processors; loosely coupled; simple interconnection
$\Theta(n^p)$ $p>1$	$\Theta(1)$	Single or few processors; shared memory

Table 1. Asymptotically optimal granularities in parallel processing of outin trees with respect to order-of-magnitude kT^2 (N is the number of leaves of the outin tree).

5. CONCLUDING REMARKS

In this paper, we have derived tight bounds within which the optimal granularity of parallel AND-tree evaluations under preemptive and nonpreemptive critical path scheduling can be found. For nonpreemptive scheduling, the asymptotically optimal granularities with respect to various problem complexities have been derived. These theoretical results provide an upper bound on the number of processors to achieve the minimum kT^2 criterion.

According to our efficiency analysis, we found that the optimal granularity depends on the problem complexity, the shape of the precedence graph (balanced or skewed), and the task-time distribution along each path (random or monotonic). It is usually difficult to predict the shape and the task-time distribution. One possible way is by statistical analysis. In contrast, the complexity of a problem to be solved is generally known before the problem is solved.

The complexity of each node in the outin tree is an important factor that influences the optimal granularity. As illustrated in Table 1, if C(n)+D(n) is $\Theta(n^p)$, $p>1$, and a large number of processors are used, then the processor-time efficiency, kT^2, must be poor regardless of the capacity of the interconnection network. In this case, the time needed to evaluate a subproblem will be increased quickly during the decomposition process in the outtree and the composition process in the intree. Hence, the root and exit nodes of the tree are obvious bottlenecks. In contrast, if C(n) and D(n) are $\Theta(1)$, then the time needed to evaluate any subproblem is bounded by a constant, and the root and exit nodes will not be bottlenecks. Examples of this kind of problems include finding the maximum and evaluating an arithmetic expression. Here, a fine-grain architecture is appropriate, and a large speedup will be obtained by using a large number of processors. Tree-structured computer architectures [13, 21] and virtual-tree computers [2] are good candidates in these applications. In cases when C(n) equals either $\Theta(n)$ or $\Theta(\log^s n)$, $s \geqslant 0$, the time needed to evaluate a subproblem is increased slowly during the decomposition process in the outtree and the composition process in the intree. A medium-grain architecture will be more cost-effective. For example, to sort 4000 elements by a parallel merge-sort algorithm, using ten to twelve processors will be a good choice. For many practical problems, especially when divide-and-conquer algorithms are used, the precedence graph is nearly balanced, and the task-times of all nodes in each level are approximately equal, hence, the nonpreemptive critical path scheduling algorithm may be viewed as a parallel breadth-first search. In this case, well balanced workloads with overlapped process communications can be assigned to the processors working under an SIMD model. The optimal grains will, therefore, be close to the theoretical ones predicted in Sections Three and Four. The architectural requirements for various cases are summarized in Table 1.

In many problems, the order-of-magnitude complexities of C(n) and D(n) may be different. For example, for the quicksort algorithm, $C(n)=\Theta(1)$ and $D(n)=\Theta(n)$, that is, most of the computational overhead is spent in the decomposition phase, and the composition operation is trivial. In contrast, for the merge-sort algorithm, $C(n)=\Theta(n)$ and $D(n)=\Theta(1)$. In many logical and functional programs, the return operation is usually simple, i.e., $C(n)=\Theta(1)$, but the complexity of a subgoal or function call depends on the number of the parameters passed and the method of copying data. For these problems, the optimal grain can be determined by the part of the tree that has dominant overhead.

The shape of the AND-tree and its task-time distribution are also important factors to be considered. Let $T_p(1)/h_{max}$ be "average width" of an AND-tree. The optimal granularity is found to depend strongly on the average width. If the AND-tree is "wide," then the degree of parallelism is high and the granularity can be small. On the other hand, if the AND-tree is "narrow," then the degree of parallelism is low and the granularity is necessarily large. Here, the tree may have to be restructured to arrive at a different representation.

In many problems involving AND-trees, the trees are usually irregular, and the workloads may be data dependent. An important functional requirement is, therefore, the ability to dynamically distribute the workload in the architecture. For a computer architecture with a small granularity, an efficient interconnection network is needed. In a loosely coupled system with a coarse grain, an effective load balancing mechanism is necessary. Here, process communications may not be well overlapped with computations, and the corresponding task-times should include the communication overhead. As a result, the optimal number of processors may be less than the theoretical values predicted in Sections Three and Four.

REFERENCES

[1] J. Backus, "Can Programming be Liberated from the von Neumann Style? A Functional Style and Algebra of Programs," *Comm. of the ACM*, vol. 21, no. 8, pp. 613-641, ACM, 1978.

[2] F. M. Burton and M. M. Huntbach, "Virtual Tree Machines," *IEEE Trans. on Computers*, vol. C-33, no. 3, pp. 278-280, 1984.

[3] E. G. Coffman, Jr. and R. H. Graham, "Optimal Scheduling for Two Processors Systems," *Acta Informatica*, vol. 1, no. 3, pp. 200-213, 1972.

[4] E. G. Coffman, Jr. (ed.), *Computer and Job-Shop Scheduling Theory*, Wiley, New York, NY, 1976.

[5] J. S. Conery and D. F. Kibler, "AND Parallelism and Nondeterminism in Logic Programs," *New Generation Computing*, vol. 3, no. 1, pp. 43-70, OHMSHA Ltd. and Springer-Verlag, 1985.

[6] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

[7] J. B. Dennis, "Data Flow Supercomputers," *Computer*, vol. 13, no. 11, pp. 48-56, IEEE, Nov. 1980.

[8] D. Dolev and M. Warmuth, "Profile Scheduling of Opposing Forests and Level Orders," *SIAM Journal of Algorithm and Discrete Mathematics*, vol. 6, no. 4, pp. 665-687, Oct. 1985.

[9] M. R. Garey and D. S. Johnson, "Scheduling Tasks with Nonuniform Deadlines on Two Processors," *Journal of ACM*, vol. 23, no. 3, pp. 461-467, 1976.

[10] M. R. Garey, D. S. Johnson, R. E. Tarjan, and M. Yannakakis, "Scheduling Opposing Forests," *SIAM Journal of Algorithm and Discrete Mathematics*, vol. 4, no. 1, pp. 72-93, March 1983.

[11] R. L. Graham, "Bounds for Certain Multiprocessing Anomalies," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1563-1581, Nov. 1966.

[12] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. N. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Ann. Discrete Mathematics*, vol. 5, pp. 287-326, 1979.

[13] J. A. Harris and D. R. Smith, "Simulation Experiments of a Tree Organized Multicomputer," *Proc. 6th Annual Symp. on Computer Architecture*, pp. 83-89, IEEE/ACM, April 1979.

[14] E. Horowitz and A. Zorat, "Divide-and-Conquer for Parallel Processing," *Trans. on Computers*, vol. C-32, no. 6, pp. 582-585, IEEE, June 1983.

[15] T. C. Hu, "Parallel Sequencing and Assembly Line Problems," *Operations Research*, vol. 9, no. 6, pp. 841-848, 1961.

[16] M. Kaufman, "An Almost-Optimal Algorithm for the Assembly Line Scheduling Problem," *Trans. on Computers*, vol. C-23, no. 11, pp. 1169-1174, IEEE, 1974.

[17] R. M. Keller, F. C. H. Lin, and J. Tanaka, "Rediflow Multiprocessing," *Proc. COMPCON Spring*, pp. 410-417, IEEE, 1984.

[18] M. Kunde, "Nonpreemptive LP-Scheduling on Homogeneous Multiprocessor Systems," *SIAM Journal of Computing*, vol. 10, no. 1, pp. 151-173, Feb. 1981.

[19] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," *Proc. Discrete Mathematics*, pp. 343-362, North-Holland, 1977.

[20] E. L. Lloyd, "Critical Path Scheduling with Resource and Processor Constrains," *Journal of ACM*, vol. 29, no. 3, pp. 781-811, 1982.

[21] G. Mago, "Making Parallel Computation Simple: The FFP Machine," *Proc. COMPCON Spring*, pp. 424-428, IEEE, 1985.

[22] R. Muntz and E. Coffman, Jr., "Preemptive Scheduling of Real-Time Task on Multiprocessor Systems," *Journal of the ACM*, vol. 17, no. 2, pp. 324-338, ACM, April 1970.

[23] F. J. Peters, "Tree Machine and Divide-and-Conquer Algorithms," *Lecture Notes CS 111 (CONPAR81)*, pp. 25-35, Springer-Verlag, 1981.

[24] S. Schindler, "On Optimal Scheduling for Multiprocessor Systems," *Proc. of Princeton Conf. on Information Science and Systems*, pp. 219-223, 1972.

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