Improving Scheduling of Tasks in a Heterogeneous Environment

Rashmi Bajaj and Dharma P. Agrawal, *Fellow*, *IEEE*

Abstract—Optimal scheduling of parallel tasks with some precedence relationship, onto a parallel machine is known to be NP-complete. The complexity of the problem increases when task scheduling is to be done in a heterogeneous environment, where the processors in the network may not be identical and take different amounts of time to execute the same task. This paper introduces a Task duplication-based scheduling Algorithm for Network of Heterogeneous systems (TANH), with complexity $O(V^2)$, which provides optimal results for applications represented by Directed Acyclic Graphs (DAGs), provided a simple set of conditions on task computation and network communication time could be satisfied. The performance of the algorithm is illustrated by comparing the scheduling time with an existing "Best Imaginary Level scheduling (BIL)" scheme for heterogeneous systems. The scalability for a higher or lower number of processors, as per their availability is also discussed. This work is shown to provide substantial improvement over existing work on the Task Duplication-Based Scheduling Algorithm (TDS).

Index Terms—Communication cost, computational cost, directed acyclic graph, heterogeneous environment, network of processors, optimal scheduling, task duplication.

1 Introduction

A growing emphasis on concurrent processing of jobs has lead to an increased acceptance of heterogeneous environment and availability of a network of processors makes a cost-effective utilization of underlying parallelism for applications like weather modeling, fluid flow, image processing, real-time and distributed database systems. Efforts are being made so that the system can automatically explore its concurrent processing capabilities. Heterogeneity in a system is introduced due to the presence of processors that have different characteristics, including speed, memory space, special processing functionalities, etc. In these applications, the data is distributed over the processors either intentionally or unknowingly and fast access of locally available data helps in achieving higher speed-ups.

The problem of optimal scheduling of tasks with required precedence relationship, in the most general case, has been proven to be NP-complete [1], [2] and optimal solutions can be found only after an exhaustive search. Such optimality can be achieved if adequate time is available, i.e., the problems are non-real-time type. Hence, many heuristics that could give an optimal solution in polynomial time have been proposed, for some very restricted cases. But, approximate optimizations are sometimes said to be an acceptable approach that can be put forth for the scheduling problem.

A well-known strategy behind efficient execution of a huge application is to partition it into multiple independent tasks and schedule such tasks over a set of available processors. A task-partitioning algorithm takes care of efficiently dividing an application into tasks of appropriate grain size and an abstract model of such a partitioned application can be represented by a Directed Acyclic Graph (DAG). Each task of a DAG corresponds to a sequence of operations and a directed arc represents the precedence constraints between the tasks. Each task can be executed on a processor and the directed arc shows transfer of relevant data from one processor to another. In this paper, it is assumed that the application is available in the form of a DAG and only the scheduling of tasks is considered here. The system is assumed to be heterogeneous, which means that a task may take different amount of execution time if run on different processors of the network. Hence, efficient scheduling of the tasks represented by a DAG over such a system becomes more involved than the case with homogeneous systems, wherein all the processors are identical and, hence, execution time of a task is independent of the processor on which it is executed. Task scheduling algorithms perform both task allocation and task sequencing, which means specifying the order of node execution on a given number of processors.

In addition, a lot of work has been done on efficient scheduling of DAGs for distributed memory machines [3]. Among various heuristics that have been proposed [4], [5], [6], [7], [8], [9], [10], [11] for the scheduling problem on homogeneous systems, the commonly categorized schemes are priority-based, cluster-based, and task duplication-based schemes [3], [11], [12], [13], [14], [15], [16], [17], [18], [19]. In all these publications, the authors assume that a separate I/O processor is available to perform computation and communications concurrently. Priority-based schemes [4], [6] assume a priority for each task that is utilized to

- *R. Bajaj is with France Telecom R&D, 801 Gateway Blvd., Suite 500, South San Franscisco, CA 94080.*
  E-mail: rashmi.bajaj@rd.francetelecom.com.
- *D.P. Agrawal is with the Center for Distributed and Mobile Computing, ECECS Department, University of Cincinnati, Cincinnati, Ohio, 45221-0030. E-mail: dpa@ececs.uc.edu.*

Manuscript received 25 Apr. 2001; revised 24 June 2003; accepted 16 July 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 114048.

assign the tasks to the different processors. However, even though simple to implement, these do not take the interprocessor communication (IPC) into account. Cluster-based schemes [5], [7], [8], [9], [10] divide the tasks into a set of clusters, each formed on the basis of interdependence of the tasks and then allocate each cluster to the processor so as to achieve a minimal communication overhead. A polynomial optimal schedule for arbitrary task graphs with unit execution time and small communication times has been introduced by Colin and Chretienne [12] and provides good performance on distributed memory architecture with identical processors. Optimal scheduling on a homogeneous system with duplication of some tasks is covered in [3]. The trade off between computation and communication in a homogeneous multiprocessor system has been exploited in [13] by doing selected duplication of tasks that could reduce the wait state while data is being transferred between processors to satisfy precedence constraints.

A new algorithm has been introduced [13] that first identifies the critical path dominant list of nodes of a given DAG and its performance is compared with five other existing algorithms when unbounded and bounded numbers of processors are assumed to be available. The algorithm is shown to provide optimal schedule if root node is duplicated at each processor. A polynomial time optimal partitioning algorithm for nested loops with uniform dependencies on a homogeneous multicomputer has been considered in [14]. The basic idea is to model startup time to be constant and being a dominating factor in computation. A detailed taxonomy for classifying various scheduling algorithms for a homogeneous multiprocessor system is given in [15] and 15 different scheduling algorithms are compared for a set of benchmark programs, primarily consisting of randomly generated application graphs. The impact of important problem parameters, are also observed for bounded and unbounded number of clusters and average degradation in the schedule length is observed with respect to the shortest schedule length called as "pseudooptimal." A new concept of scheduling scalability is also introduced and the authors conclude that a lot more work is needed to reduce the scheduling complexity.

The task duplication-based optimal scheduling algorithm proposed by Darbha and Agrawal [3] for homogeneous multiprocessors has been improved by Park and Choe [16] by allowing parent task of a join task to be scheduled on the same processor if that helps reduce the schedule length. The worst-case algorithm complexity is obtained as $\mathrm{O}(\mathrm{d}|\mathrm{V}^2|)$, where d is the maximum number of incoming edges of a join task, as opposed to $\mathrm{O}(|\mathrm{V}^2|)$ obtained in [3]. Even though, all these algorithms can be applied for a heterogeneous environment with appropriate modifications, very little has appeared in the literature because of the issue of heterogeneity of processors and the dynamically changing availability of enhanced processors. With increasing variety of heterogeneous architectures, an algorithm called bubble scheduling and allocation (BSA) is proposed in [17] that imposes finish times of tasks by migrating a task only if it can "bubble up" in a breadth-first fashion. The basic strategy is to assign the critical path to the fastest processor and the schedule length is shown to be optimal if processors are homogeneous and are fully connected or hyper cubes with homogeneous links. Beaumont et. al. [18] emphasize the importance of static scheduling and load balancing for good performance of parallel systems and showed that designing and implementing such static strategies for heterogeneous platforms are difficult. A static data distribution scheme for dense linear algebra kernels such as matrix multiplication or LU decomposition on heterogeneous platform has been presented in [19] and provides almost perfect load balancing. Task duplication based schemes [11] give flexibility to the number of available processors, as the number of required clusters may always exceed the number of processors available. Task Duplication schemes are, in general, observed to be better than the Priority-based and Cluster-based schemes. Again, these schemes give optimal or suboptimal schedules for certain specific types of DAGs. These schemes also form the basis of solutions proposed for heterogeneous processors, where the runtime of a task varies from processor to processor, which makes scheduling much harder. If some processors are lightly loaded during the execution duration of the application, they can be used in time-shared mode and the solution proposed in this paper can be applied to all such problems. Hence, we extend the work done in [20], [21], [22] and introduce the algorithm TANH and prove it to be optimal for DAGs with a simple set of conditions that are described in a later section. The TANH also gives improved performance in terms of speed-up and complexity $\mathrm{O}(\mathrm{V}^2)$, in comparison with Best Imaginary Level Scheduling (BIL) [23], a scheduling algorithm with complexity $\mathrm{O}(\mathrm{V}^2\mathrm{W}\log\mathrm{W})$, where V is the number of nodes and W is the number of processors, for a heterogeneous system. In the next section, we introduce the details of the TANH algorithm and Section 3 shows the improvement in performance for practical DAGs. Section 4 gives the conclusions.

2 Task Duplication-Based Scheduling Algorithm for Network of Heterogeneous Systems (TANH)

2.1 Definition of a DAG

A DAG is a directed acyclic graph that gives an abstract level of an application model, with multiple independent partitions and each partition represents a task and directed arcs indicate a data communication and the precedence relationship. A task is assumed to be an indivisible unit of executable work and is assumed to be nonpreemptive. The TANH algorithm schedules these tasks onto available heterogeneous processors in a fashion that results in an optimal schedule, which could be defined as a schedule with the lowest schedule length (also known as *makespan* [22]) possible for a DAG. The heterogeneity has been modeled by assuming different runtimes of tasks on different processors. The DAG is defined by a tuple (V, E, W, τ, c), where V is a set of task nodes; E is a set of communication edges; W is a set of processors; $\tau = \tau(\mathrm{j}, \mathrm{w})$, where $\mathrm{j} \in \mathrm{V}$ and $\mathrm{w} \in \mathrm{W}$, while $\tau(\mathrm{j}, \mathrm{w})$ indicates the runtime or computation cost of jth task on wth processor; $c = c(\mathrm{j}, \mathrm{k})$ with $\mathrm{j}, \mathrm{k} \in \mathrm{V}$ and $c(\mathrm{j}, \mathrm{k})$ indicates the communication cost

TABLE 1
Mathematical Description of Parameters

j, k, l	: Nodes of the DAG, also known as tasks.
w, q	: Notation for processors that belongs to set of processors W. w here denotes the favorite processor of a task j chosen from set of processors W, after the condition in equation (2) is satisfied.
PRED(j)	: Set of predecessors of any task j (i.e. all nodes that send data to node j).
SUCC(j)	: Set of successors of any task j(i.e. all nodes that j sends data to).
W	: Set of processors available.
est(j/w)	: Earliest start time of task j given that it will execute on processor w.
ect(k/w)	: Earliest completion time of task k given that it will execute on processor w.
τ(j,w)	: Execution time of the jth task on wth processor.
c(j,k)	: Communication cost from task j to task k, where j is predecessor of k.

Eq. No.	**Formulae for Calculation of Parameters**
(1)	**Earliest Start Time of entry node** $est(\text{entry}) = 0$
(2)	**Favorite Processor for any node j of the DAG** $fp_w(j) = w \in W \text{ s.t. } \underset{k \in PRED(j),\ w = fp(k),\ q \neq fp(k)}{\left((ect(k/w) + \tau(j,w)) < (ect(k/q) + c(k,j) + \tau(j,q)) \right.} \text{ where } q \in W).$ (In other words, assigning task *j* to processor *w* will yield a minimum completion time for task *j*).
(3)	**Earliest Start Time of any node j (other than the entry node) that runs over processor.** $est(j) = est(j/w) = \underset{l \in PRED(j)}{\text{Min}} \left[\underset{\substack{k \in PRED(j),\ k \neq l \\ fp(k) = w}}{\text{Max}} (ect(k), ect(l)+c(l,j)) \,\Vert\, \underset{\substack{k \in PRED(j),\ k \neq l \\ fp(k) \neq w;\ where\ w = fp(j)}}{\text{Max}} ((est(k)+\tau(k,w)),(ect(l)+c(l,j))) \right]$
(4)	**Earliest Completion Time of task j** $ect(j) = est(j) + \tau(j,fp(j)).$
(5)	**Favorite Predecessor of task j.** $fpred(j) = k \in PRED(j) \text{ s.t. } [\underset{fp(k) = w}{ect(k) \geq (ect(l) + c(l,j))} \,\Vert\, \underset{fp(k) \neq w;\ where\ w = fp(j)}{(est(k)+\tau(k,w)) \geq (ect(l) + c(l,j))}]$
(6)	**Latest Allowable Completion Time of exit node.** $lact(\text{exit}) = ect(\text{exit})$
(7)	**Latest Allowable Completion Time of any task j (other than exit node)** $lact(j) = \underset{k \in SUCC(j)}{\text{Min}} [\underset{fp(k) \neq w}{\text{Min}} (last(k) - c(j,k)), \underset{fp(k) = w;\ where\ w = fp(j)}{\text{Min}} (last(k))]$
(8)	**Latest Allowable Start Time of any task j.** $last(j) = lact(j) - \tau(j/fp(j))$

between tasks j and k. If both j and k reside on the same processor, c(j, k) is assumed to be zero. The communication cost between two nodes j and k, c(j, k) depends on the channel initialization at both sender j and receiver k in addition to the communication time on the channel. This is a dominant factor and can be assumed to be independent of the source and destination processors. The channel initialization time is assumed to be negligible. Strictly speaking, W is independent of V, E, τ, and c. Without loss of generality, it is assumed that there is one entry node to the DAG and one exit node from the DAG. In an actual implementation, we can create a pseudonode with zero execution time and with no incoming edges and with outgoing edges of zero cost to all entry nodes if there are more than one nodes with no incoming edges. It also creates a pseudoexit node if the input has multiple exit nodes. It is also assumed that dedicated communication channels are available, i.e., any amount of message passing is allowed at any given time [21], [22]. In other words, the network bandwidth is assumed to be wide enough to provide instantaneous contention-free transmission.

2.2 Parameters and Pseudocode for TANH

Necessary parameters used in the algorithm are mathematically described in Table 1. Dummy variables j, k, and l have been used to depict a node in the DAG. In the first step of the algorithm, the DAG is traversed in a top-down fashion to compute the earliest start and completion times (est(j) and ect(j)) for a task j(j ∈ V), indicating when a task could be started/completed at the earliest possible time. The earliest start time of a node, est(j), is the earliest time that a node can start execution on any one of the processors. Note that a node cannot start execution before its earliest start time. The earliest completion time of a node, ect(j), is defined as the earliest time that a node can finish execution. It is simply the sum of a node's earliest start time and its execution cost on its favorite processor. The level of a node, $level$(j), is defined to be the highest value of the summation of computation costs along different paths from the node j to the exit node and indicates the minimum possible time needed from the current task to the end of the program. The favorite processors from 1 to n, i.e., $fp1$(j) to fpn(j), where n is the number of available processors and $level$(j) for each task, are also computed in a top-down traversal. The processor is defined as favorite if using that for execution of task j(j ∈ V) results in a minimum completion time for the task j. Hence, it need not always imply that the task needs to have the lowest computation time on its favorite processor. The favorite predecessor of a node, $fpred$(j), is defined as that predecessor for whom the sum of earliest completion time and the cost of communication are largest among all the predecessors. In other words, the favorite

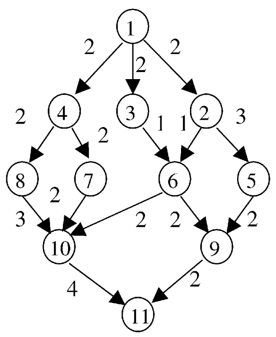


Fig. 1. Example Directed Acyclic Graph (DAG).

predecessor of a node is always the bottleneck for the execution of the node. The favorite predecessor for the task j, ($fpred$(j)), is computed as the one with the highest value of earliest completion time among the predecessors of j. While finding the $fpred$(j), we also take into account the favorite processor of j, as seen in the (5) of Table 1. Here, the predecessor of a task j is one of the nodes that are executed just one step before node j. For instance, in Fig. 1, the set of predecessors of j, (defined as PRED(j)) of task 10 are tasks 6, 7, and 8. Similarly, the set of successors of j, (defined as SUCC(j)) are those tasks that are executed just one step after task j. For instance, the set of successors of task 2 in Fig. 1, are tasks 5 and 6.

In the next stage, the latest allowable start and completion times ($lact$(j) and $last$(j)) for each node are computed in a bottom-up fashion. The quantities $lact$(j) and $last$(j), respectively, stand for the latest allowable completion time and the latest allowable start time of the task node j. The latest allowable completion time of a node is used to calculate the latest allowable start time of its successor nodes. If a node and its successor are assigned to the same processor, then the cost of communication is assumed to be zero. If they are assigned to different processors, the communication cost needs to be deducted from the $lact$ value of the node. The elements in the array $queue$ are the nodes sorted in the ascending order of $level$.

The parameters described here are different from those defined in the work related to the Task Duplication-based Scheduling Algorithm (TDS) for homogeneous systems [3], in which the mathematical equations for the est(j), $fpred$(j), and $lact$(j) did not always efficiently calculate the earliest start time or the real favorite predecessor of a task. The new equations also differ in the sense that the values are subject to constraints in terms of available favorite processors as opposed to this factor not being taken into consideration in TDS [3], thereby affecting the schedule length.

We perform a running trace of the algorithm and illustrate various stages for an example DAG given in Fig. 1. For each task (or node), the runtimes on different processors are listed in Table 2. Communication costs are indicated along the edges of the DAG in Fig. 1. The calculated values of the various tasks of the DAG in Fig. 1 are shown in Table 3. The stepwise calculation of these parameters is explained as follows:

Step 1: Calculation of earliest start time (est) of each node. Taking node 1 as an example, the est for this is zero. The est of other nodes is calculated using (3) in Table 1. For example, for node 6, whose $fp1$ is P1,

$$est(6) = \mathrm{Min}[\mathrm{Max}(ect(2), ect(3) + c(3,6)), \\ \mathrm{Max}(est(3) + \tau(3, P1), ect(2) + c(2,6))],$$

which is $\mathrm{Min}[\mathrm{Max}(9,9), \mathrm{Max}(8,10)]$. Hence, $est(6) = 9$.

TABLE 2
Runtimes of Nodes for DAG in Fig. 1

Run time of task j on processor w τ(j,w)→ ↓ task j	P1	P2	P3	P4
1	4	4	4	4
2	5	5	5	5
3	4	6	4	7
4	3	3	3	3
5	3	5	3	4
6	3	7	2	2
7	5	8	5	5
8	2	4	5	3
9	5	6	7	5
10	3	7	5	2
11	5	6	7	8

Step 2: Calculation of favorite processors $fp1$, $fp2$, $fp3$...etc. for each node (where $fp1$ implies the best choice processor, $fp2$ implies the next best choice among the processors, etc.). In Table 1, it is seen that node 1, which is the entry node, has its favorite processors $fp1$, $fp2$, $fp3$, and $fp4$ in the order P1, P2, P3, and P4. For other nodes, take, for example, node 10, the $fp1(10)$ is P4. This is because the ect of its predecessors, nodes 6, 7, and 8, is 12, 12, and 9, respectively. Thus, using the required ect's over all the processors, the processor which yields the lowest $ect(10)$ is chosen (considering this includes the computation cost of node 10 as well on each of the processors), which in this case gives the favorite processors $fp1$, $fp2$, $fp3$, and $fp4$ as P4, P1, P3, and P2, respectively.

Step 3: Calculating the earliest completion time (ect): The $ect(1)$ is calculated using (4). Here, $ect(1) = 4$, as $est(1) = 0$ and $\tau(1, P1) = 4$.

Step 4: Calculation of favorite predecessor ($fpred$). The favorite predecessor of a node is calculated using (5). Among all the predecessors of a node, a predecessor that has the highest value of ect is chosen as the $fpred$.

Step 5: Calculation of the latest allowable start time ($last$) and latest allowable completion time ($lact$) for each node: The values of latest allowable start time ($last$) and latest allowable completion time ($lact$) of the nodes is calculated in a bottom-up traversal of the DAG. These are calculated using (6), (7), and (8) of Table 1.

In the remaining stages of the algorithm, we generate the clusters and simultaneously choose a processor for their

TABLE 3
Calculated Parameters for Example DAG in Fig. 1

node	est	fp1	fp2	fp3	fp4	ect	lact	last	fpred	level
1	0	P1	P2	P3	P4	4	4	0	-1	33
2	4	P1	P2	P3	P4	9	9	4	1	27
3	4	P3	P1	P2	P4	8	8	4	1	29
4	4	P1	P2	P3	P4	7	9	6	1	26
5	9	P1	P3	P4	P2	12	15	12	2	20
6	9	P1	P3	P4	P2	12	12	9	2	22
7	7	P4	P3	P1	P2	12	14	9	4	23
8	7	P1	P4	P2	P3	9	11	9	4	20
9	14	P1	P4	P2	P3	19	20	15	5	15
10	14	P4	P1	P3	P2	16	16	14	7	15
11	20	P1	P2	P3	P4	25	25	20	9	8

```
Algorithm:  TANH main body

Input:    DAG(V, E, W, τ, c)
          pred(i): Set of Parents of Task i.
          succ(i): Set of Successors of Task i.
          τ(i,w): Computation costs of Task i on each processor w ∈ W.
Output:   Schedule
Begin:
          1. Compute est(i), ect(i), fpred(i), fp(i), level(i) for all nodes i ∈ V;
          2. Compute last(i), lact(i) for all nodes i ∈ V;
          3. Generate initial clusters /* algorithm for this part in Figure 3 */
          4. Perform duplication of tasks or processor reduction
          5. Perform message scheduling
End
```

Fig. 2. Pseudocode of TANH.

execution. A fine-tuning of the final schedule is carried out depending upon the available number of processors, in comparison with the required number of processors. The pseudocode of TANH is given in Fig. 2. As we see, in the fourth stage, if the available number of processors is higher, additional duplication is carried out. On the other hand, if the available processors are fewer, the algorithm can be designed for appropriately scaling down the number of clusters to the available number of processors. The algorithm for the situation where a processor reduction [21] is required is given in Fig. 4.[1]

Coming back to the example DAG, for generating clusters, the nodes are arranged in ascending order of *level* to generate the *queue*. Here, the generated *queue* is as follows: {11, 10, 9, 5, 8, 6, 7, 4, 2, 3, 1}. Starting from the first node in the *queue*, the cluster is generated. The first cluster, following the algorithm in Fig. 3, is generated as 11-9-5-2-1, and executed on P1. The algorithm for the cluster generation[2] (Step 3) is given in Fig. 3.[3] The rest of the algorithm follows the same procedure as explained in [21] and is reproduced for the completeness of the text. The schedule for the example DAG is shown in Fig. 5, with four processors represented along the Y-Axis while the schedule length indicated along the X-Axis. Fig. 5 is a timing diagram with the respective allocation of tasks for each processor. The schedule time indicates the time units needed for completing the task and the arrows between processors show the communication time for transmission of results from one parent node to the child node(s). Also, as seen from Table 3, the calculated earliest completion time for the exit node 11 is 25, which matches the total schedule length in the timing diagram of Fig. 5, thus an optimal makespan

1. The processor reduction step is applied so that all clusters can be accommodated in the available set of processors. If the number of available processors is less than the number of initial clusters generated, then the excess clusters will be accommodated in the pseudoprocessors. To accommodate remaining clusters, use pseudoprocessors where the runtime of a task on a pseudoprocessor is taken as the average runtime of the task. Compaction has to be carried out to get rid of these pseudoprocessors and to ensure that the number of clusters remaining can be accommodated in the available set of processors.

2. The essence of this step is to try and generate clusters, where each task is preceded by its favorite predecessor. If a predecessor has already been assigned or if it is not critical, where the criticality is determined by considering the equation $\mathrm{last}(x) - \mathrm{lact}(y) \geq c(x, y)$ (where x is the successor of task y), then, if the above condition is satisfied, then task y is not critical to task x. Each cluster will be assigned to the first available fp of the header task of that cluster, i.e., if $fp[1]$ of the task is already assigned to another cluster and $fp[2]$ is unassigned, $fp[2]$ will be selected to accommodate the current cluster.

3. Step 3 of TANH and processor reduction step have been restated from [21] for the purpose of completeness in pseudocode of the algorithm.

```
Algorithm: Processor_allocation.
Begin
  x=First element of queue;
  current processor = fp1 of x;
  assign x to current processor;
  While(NOT all tasks assigned){
      y=fpred(x);
      if x has more than one predecessors{
          if ((last(x)-lact(y))≥c(x,y)) /*y is not critical */
             y = predecessor of x, with the lowest running time on this processor;
          else if (y is already assigned)
             y = predecessor of x, with the lowest  running time on this processor;
      }
      assign y to current processor;
      x = y;
      if x is the entry node{
          x = next unassigned element in queue;
          if(available processors > required processors)
             current processor = next available fpm of x;
          else
             current processor = new pseudo processor;
      assign x to current processor;
    }
End
```

Fig. 3. Algorithm for Step 3 of TANH.

or a schedule length equal to the lowest possible completion time for the exit node is obtained.

2.3 TANH Complexity

The complexity of the first stage is $O(V(W + W \log W))$ [24], where W is the number of processors and V is the number of nodes. For most practical DAGs, $V >> W$ and V is of the order of $W \log W$. Therefore, the complexity is $O(V^2)$. The second step is a bottom up traversal and for a connected graph, the complexity is $O(E)$. The complexity of the formation of the *queue* is $V \log V$. The third step is similar to any depth first search algorithm and the order of complexity is $O(V + E)$. For a connected graph, it is $O(E)$ for the worst case. In the duplication stage, the complexity is $O(TV)$, where T is the number of tasks where the *fpred* has not been used. In the worst-case, $T = V$ and the complexity is $O(V^2)$. The compaction stage involves sorting of processors based on empty slots. Therefore, for each pass through the compaction code, it is of the order of $O(V + W \log W)$ with $W = V$ in the worst case. The number of passes required depends on the difference between the available processors (AP) and the required processors (RP) and is given by $Q = \log_2(RP/AP)$. In the worst case, $RP = V$ and $AP = 1$. So, the worst case is $Q = \log_2(V)$. Therefore, the complexity for the compaction is $O(V^2)$. Thus, the overall complexity is $O(V(W + W \log W) + E + V \log V + V + E + V^2)$. But, as mentioned above, for most practical applications, the first term can be substituted as V^2 as $V >> W$ and $V \approx W \log W$. Therefore, the complexity of the algorithm is $O(V^2)$.

```
Algorithm:  Processor_reduction;
Input:
    Initial set of clusters
    Number of Available Processors (AP)
    Number of Required Processors (RP)
Begin
    While (AP<RP){
Calculate exec(i) for each cluster;                /* Sum of computation costs of tasks in proc  i */
    Sort available processors in ascending order of exec(i);
    Temp=RP-AP;
    if( temp > RP/2)
        temp=RP/2;
    for(j=0;j<temp;j++)
        merge task lists of processors j and (temp*2-j-1);
/*Use level to determine relative positions */
/* Remaining mostly crowded pseudo processors */
decrement RP;
}
End
```

Fig. 4. Algorithm for the processor reduction step.

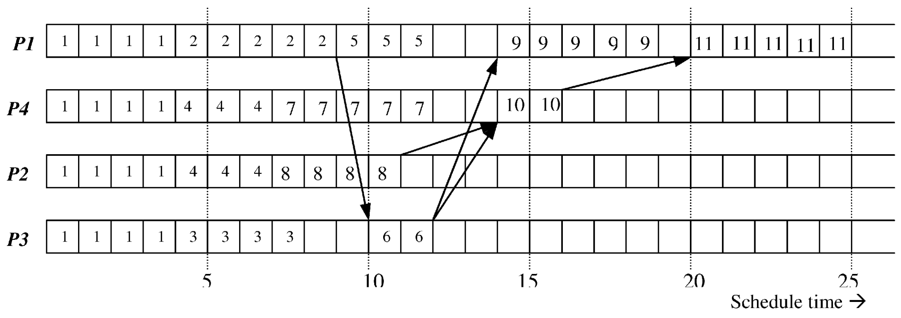


Fig. 5. Schedule for example DAG in Fig. 1.

3 Optimality of Scheduling Algorithm

3.1 Conditions

A DAG is basically a collection of fork nodes and join nodes (illustrated in Figs. 6a and 6b) interconnected with directed arcs, representing precedence relationships and data communication if two adjacent nodes are allocated for execution on two different processors. A fork node is a node that is a parent or predecessor to more than one node. A join node is a node that is a child to more than one node. These two kinds of nodes hold a special significance in determining the schedule length of a DAG. This is so because the choice of the best predecessor or successor determines the efficiency in the cluster generation and thereby affects the generation of the schedule close to the optimal solution.

Hence, once a set of conditions has been proven for the fork and join nodes, then that set is also applicable to the DAG as a whole. For this reason, we need to impose some simplistic conditions as follows: Let there be V_f fork nodes and V_j join nodes with $(V_f, V_j) \in V$. Let $est(m)$ and $est(n)$ be the earliest start times of the predecessor nodes m and n, respectively, such that m is the first favorite predecessor of task j ($fpred(j)$) and n the second $fpred(j)$ (Fig. 7b). Then, we have the following conditions:

3.1.1 Condition for Fork Nodes (V_f)

A fork node i that is not a join node (Fig. 7a) is assumed to have the same execution time on all processors. This condition, as explained later, is desirable because these nodes are critical and, hence, are potential candidates for duplication on a majority of the processors.

3.1.2 Condition for Join Nodes (V_j)

In Fig. 7b, let m be the first $fpred(j)$ and n be the second $fpred(j)$, then we have three cases, depending on the fact that the nodes m and n may or may not be fork nodes of the type that follow Condition 3.1.1.

Case 1. m and n are not fork nodes. There are three situations:

1. $\tau(n)_{\min} \geq c(m,j)$ and $\tau(m)_{\min} \geq [c(n,j) + (est(n) - est(m))]$ if $est(m) < est(n)$,
2. $\tau(m)_{\min} \geq c(n,j)$ and $\tau(n)_{\min} \geq [c(m,j) + (est(m) - est(n))]$ if $est(m) > est(n)$, or
3. $\tau(m)_{\max} \geq \tau(n)_{\min} + c(n,j)$ and $\tau(n)_{\max} \geq \tau(m)_{\min} + c(m,j)$ if $est(m) = est(n)$.

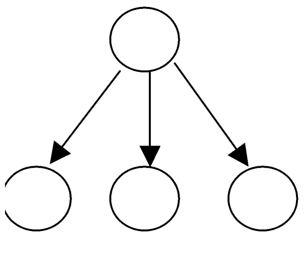
Case 2. m is a fork node; n is not a fork node. The three situations are:

4. $\tau(n)_{\min} \geq c(m,j)$ and $\tau(m) \geq [c(n,j) + (est(n) - est(m))]$ if $est(m) < est(n)$,
5. $\tau(m) \geq c(n,j)$ and $\tau(n)_{\min} \geq [c(m,j) + (est(m) - est(n))]$ if $est(m) > est(n)$ or,
6. $\tau(m) \geq \tau(n)_{\min} + c(n,j)$ and $\tau(m) \leq \tau(n)_{\min} + c(m,j)$ if $est(m) = est(n)$.

Case 3. m is not a fork node; n is a fork node. Here again, there are three situations:

7. $\tau(n) \geq c(m,j)$ and $\tau(m)_{\max} \geq [c(n,j) + (est(n) - est(m))]$ if $est(m) < est(n)$,
8. $\tau(m)_{\max} \geq c(n,j)$ and $\tau(n) \geq [c(m,j) + (est(m) - est(n))]$ if $est(m) > est(n)$, or
9. $\tau(n) \geq \tau(m)_{\min} + c(m,j)$ and $\tau(n) \leq \tau(m)_{\min} + c(n,j)$ if $est(m) = est(n)$.

It is important to mention that, in case m and n are both fork nodes, then the situation becomes similar to a homogeneous system, and optimality has already been proven for homogeneous Distributed Memory Machines (DMMs) [23]. If the aforementioned conditions are satisfied,


(a)

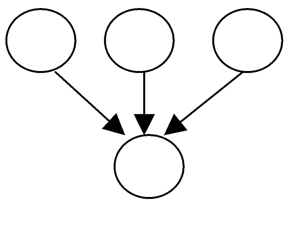
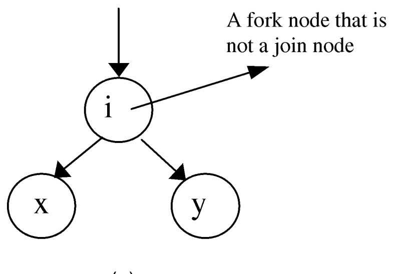
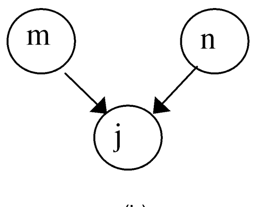

(b)

Fig. 6. (a) Fork node and (b) Join node.



(a)



(b)

Fig. 7. (a) Example fork node and (b) Join node.

then the DAG gives optimal makespan. The pseudocode for heterogeneous systems has been given in [20]. Appropriate changes in parameter calculations have been made, as given in Table 1, and the algorithm is found to lead to optimal results (proven in the next section) with the DAGs with the conditions given above.

3.2 Optimality Check

As mentioned earlier, the DAG is basically a combination of fork and join nodes. For the fork nodes, when there is just one incoming edge, the need for duplicating the fork node (Fig. 7a) results following the criticality of such a node for the execution of its successor nodes. Hence, Condition 3.1.1 is admissible and forms the basis for the conditions for join nodes (Section 3.1.2), which are necessary to show the optimality of our scheduling algorithm.

Theorem 1. *Given the join nodes that satisfy the respective conditions stated in Section 3.1.2, the TANH Algorithm for heterogeneous systems gives optimal makespan.*

Proof. Consider the example join node in Fig. 7b. According to the condition, nodes m and n are the first and second favorite processors of task j. In other words, these predecessor nodes, which complete the latest and second latest (according to (5) in Table 1), take into account the favorite processors (fps) as well, are chosen as nodes m and n in this condition. Hence, task j can start executing after m and n complete execution. This algorithm has been designed such that the predecessor nodes are assigned to separate processors and this condition supports this fact and it is proven that this results in the lowest $est(\mathrm{j})$.

In the conditions, it is assumed that P is $fp(\mathrm{m})$ and Q is $fp(\mathrm{n})$. It has been proven, for Case 1 of Condition 3.1.2, that the start time of task j cannot be lowered by assigning m and n to the same processor and, hence, the conditions would result in the lowest possible $est(\mathrm{j})$. The other predecessor tasks may have any values of communication cost and computation cost. But, in any case, it follows that since these nodes are not the first or the second $fpred$s (according to (5) of Table 1), task j has to wait till m and n are completely executed. Thus, only tasks m and n need to be considered among all the predecessors of task j. The proof for the remaining Cases 2 and 3 of Section 3.1.2 follows from the proof for Case 1, as explained below.

Case 1. Condition where m **and** n **are not fork nodes.** There are three cases possible here.

Case 1a. $est(\mathrm{m}) < est(\mathrm{n})$, from the condition stated in 1) in Section 3.1.2. Here again, two conditions are given.

Condition 1. $\tau(\mathrm{n})_{\min} \geq c(\mathrm{m},\mathrm{j})$. Here, in the worst case, if $ect(\mathrm{m}) > ect(\mathrm{n}) + c(\mathrm{n},\mathrm{j})$, i.e., $est(\mathrm{j}) = ect(\mathrm{m})$, if tasks m, n, and j are assigned to the same processor P, then, even in the best situation, $est(\mathrm{j}) = est(\mathrm{m}) + \tau(\mathrm{m},\mathrm{P}) + \tau(\mathrm{n})_{\min} = ect(\mathrm{m}) + \tau(\mathrm{n})_{\min}$. Thus, the start time of task j cannot be lower than $ect(\mathrm{m})$.

Condition 2. $\tau(\mathrm{m})_{\min} \geq [c(\mathrm{n},\mathrm{j}) + (est(\mathrm{n}) - est(\mathrm{m}))]$. For the worst case here, if $ect(\mathrm{n}) > ect(\mathrm{m}) + c(\mathrm{n},\mathrm{j})$, i.e., $est(\mathrm{j}) = ect(\mathrm{n})$, if tasks m, n, and j are assigned to the same processor Q, then, even for the best possibility, $est(\mathrm{j}) = est(\mathrm{n}) + \tau(\mathrm{n},\mathrm{Q}) + \tau(\mathrm{m})_{\min} = ect(\mathrm{n}) + \tau(\mathrm{m})_{\min}$. The start time of task j could be improved if $est(\mathrm{n}) + \tau(\mathrm{n},\mathrm{Q}) + \tau(\mathrm{m})_{\min} < est(\mathrm{m}) + \tau(\mathrm{m},\mathrm{P}) + c(\mathrm{n},\mathrm{j})$. In other words, if $est(\mathrm{n}) + \tau(\mathrm{m})_{\min} < est(\mathrm{m}) + c(\mathrm{n},\mathrm{j})$, for the ideal case that $\tau(\mathrm{n},\mathrm{Q}) = \tau(\mathrm{m},\mathrm{P})$, we get $\tau(\mathrm{m})_{\min} \geq [c(\mathrm{n},\mathrm{j}) + (est(\mathrm{n}) - est(\mathrm{m}))]$. Thus, the start time of j cannot be lowered.

Case 1b. $est(\mathrm{m}) > est(\mathrm{n})$, from the condition stated in 2) in Section 3.1.2. Here again, two conditions are given.

Condition 1. $\tau(\mathrm{n})_{\min} \geq c(\mathrm{m},\mathrm{j})$. Here again, in the worst case, if $ect(\mathrm{m}) > ect(\mathrm{n}) + c(\mathrm{n},\mathrm{j})$, i.e., $est(\mathrm{j}) = ect(\mathrm{m})$, if tasks m, n, and j are assigned to the same processor Q, then, even in the best case, $est(\mathrm{j}) = est(\mathrm{n}) + \tau(\mathrm{n},\mathrm{Q}) + \tau(\mathrm{m})_{\min} = ect(\mathrm{n}) + \tau(\mathrm{m})_{\min}$. Thus, the start time of task j cannot be lower than $ect(\mathrm{n})$.

Condition 2. $\tau(\mathrm{n})_{\min} \geq [c(\mathrm{m},\mathrm{j}) + (est(\mathrm{m}) - est(\mathrm{n}))]$. For the worst case here, if $ect(\mathrm{n}) > est(\mathrm{m}) + c(\mathrm{m},\mathrm{j})$, i.e., $est(\mathrm{j}) = ect(\mathrm{n})$, if tasks m, n, and j are assigned to the same processor P, then, in the best possibility, $est(\mathrm{j}) = est(\mathrm{m}) + \tau(\mathrm{m},\mathrm{P}) + \tau(\mathrm{n})_{\min} = ect(\mathrm{m}) + \tau(\mathrm{n})_{\min}$. The start time of task j could be improved if $est(\mathrm{m}) + \tau(\mathrm{m},\mathrm{P}) + \tau(\mathrm{n})_{\min} < est(\mathrm{n}) + \tau(\mathrm{n},\mathrm{Q}) + c(\mathrm{m},\mathrm{j})$. In other words, if $est(\mathrm{m}) + \tau(\mathrm{n})_{\min} < est(\mathrm{n}) + c(\mathrm{m},\mathrm{j})$, for the ideal case that $\tau(\mathrm{n},\mathrm{Q}) = \tau(\mathrm{m},\mathrm{P})$, we get, $\tau(\mathrm{n})_{\min} \geq [c(\mathrm{m},\mathrm{j}) + (est(\mathrm{m}) - est(\mathrm{n}))]$. Thus, the start time of j cannot be lowered.

Case 1c. $est(\mathrm{m}) = est(\mathrm{n})$, from the condition stated in 3) in Section 3.1.2. Here again, two conditions are given.

Condition 1. $\tau(\mathrm{m})_{\max} \geq \tau(\mathrm{n})_{\min} + c(\mathrm{n},\mathrm{j})$ **and Condition 2:** $\tau(\mathrm{n})_{\max} \geq \tau(\mathrm{m})_{\min} + c(\mathrm{m},\mathrm{j})$. Here, there are two cases, either of which give worst results: either $ect(\mathrm{n}) > est(\mathrm{m}) + c(\mathrm{m},\mathrm{j})$ or $ect(\mathrm{m}) > est(\mathrm{n}) + c(\mathrm{n},\mathrm{j})$. Also, since $est(\mathrm{m}) = est(\mathrm{n})$, the worst-case may occur while m, n, and j are executed on any processor apart from P and Q. Hence, both the above conditions are direct relations between the computation costs of m and n. Also, the calculated $est(\mathrm{j})$ is the lowest possible value considering that $\tau(\mathrm{m})_{\max}$ and $\tau(\mathrm{n})_{\max}$ have to be taken into account for the flexibility of m and n to be processed on any processor.

Case 2. Condition where m **is a fork node and** n **is not a fork node.** Here again, there are three cases as given by 4), 5), and 6) of Section 3.1.2. Since m is a fork node, Condition 3.1.1 states that the computation cost of m on all processors is same. Hence, $\tau(\mathrm{m})_{\max} = \tau(\mathrm{m})_{\min} = \tau(\mathrm{m})$ Hence, it follows that these equations are basically the same as for Case 1, except that $\tau(\mathrm{m})_{\max}$and $\tau(\mathrm{m})_{\min}$ reduce to $\tau(\mathrm{m})$. Hence, the proof for this case follows from the proof given for Case 1.

Case 3. Condition where m **is not a fork node and** n **is a fork node.** Here again, there are three cases as given by 7), 8), and 9) of Section 3.1.2. Since n is a fork node, Condition 3.1.1 states that the computation cost of n on all processors is same. Hence, $\tau(\mathrm{n})_{\max} = \tau(\mathrm{n})_{\min} = \tau(\mathrm{n})$. Hence, it follows that these equations are basically the same as for Case 1, except that $\tau(\mathrm{n})_{\max}$ and $\tau(\mathrm{n})_{\min}$ reduce to $\tau(\mathrm{n})$. Hence, the proof for this case also follows from the proof given for Case 1. □

Thus, from the above cases it follows that, if the DAGs fork and join nodes satisfy the conditions given above, then TANH generates a schedule with the lowest possible

TABLE 4
Complexities of Scheduling Schemes for Heterogeneous Systems

Algorithm	Proposed by	Complexity	Restrictions/ Observations
GDL	Sih & Lee [1993]	$O(wv^3 * f(w))$	Takes average execution time for a node
BIL	Hyunok Oh & Soonhoi Ha [1996]	$O(v^2 w \log w)$	Optimal for linear DAGs, higher complexity
HEURISTIC	Hui & Chanson [1997]	$O((v+e)v \log v)$	Precedence among tasks not taken into account
A^* Based	Kafil & Ahmed [1998]	$O(w^{f(v)})$	Not practical for large DAGs
HFET	Haluk, Hariri & Wu [1999]	$O(v^2 w)$	Effects of decision of processor selection not known, higher complexity
CPOP	Haluk, Hariri & Wu [1999]	$O(v^2 w)$	Effects of decision of processor selection not known, higher complexity
BSA	Kwok & Ahmad [2000]	$O(d\|v\|^2)$	Optimal only for homogeneous systems and links
TANH	Proposed here	$O(v^2)$	Optimal with simplistic node conditions, lower complexity

earliest start time and, hence, a lowest possible earliest completion time is assured.

4 Performance Analysis

4.1 Algorithm Complexity

In this section, we compare the scheduling complexity of various schemes proposed for heterogeneous systems. These are summarized in Table 4. Among various schemes, "General Dynamic Level Scheduling" (GDL) [4], assumes an average execution time of the node to be same for all the processors while performing a priority-based scheduling of tasks. The algorithm proposed by Kwok and Ahmad [17] provides an optimal solution if the system is homogeneous. The heuristic by Hui and Chanson [25] does not take precedence among tasks into account while generating a schedule. The A^*-based technique [26] from the area of artificial intelligence, is not practical for large DAGs. Among the two recent algorithms proposed, "Heterogeneous Earliest Finish Time" (HFET) [27] and "Critical Path On a Processor" (CPOP) [27], HFET is said to perform better. In HFET, it is observed that, by the time the decision of selecting the processor for a task is made, the overall effect of that decision over the schedule is not known. The scheduling scheme BIL by Oh and Ha [23] has been observed to be analytically superior to GDL and generates optimal results if the DAG is linear. Hence, based on these observations, a performance comparison of TANH versus BIL [23] is more indicative of the improved performance of TANH and this is discussed in the following section. Our proposed algorithm TANH also has a better worst-case complexity $O(v^2)$ and its performance superiority has been illustrated by comparing the schedule length for large practical DAGs (discussed in Section 4.3).

4.2 Comparison of TANH with BIL

The performance of the algorithm has been observed in terms of ACT/ECT, which is the ratio of actual completion time and the earliest completion time. This is an absolute performance indicator and has been observed over different values of the diversity and the Communication cost-Computation cost Ratio (CCR). The diversity of a task set indicates the level of computational heterogeneity in that task set. Hence, a high value of diversity indicates that there is a large variation in the task execution times in the task set. The closeness of ACT to ECT indicates the quality of the schedule. If ACT/ECT is one, it implies an optimal schedule.

To compare TANH with the Best Imaginary Level (BIL) heuristic [23], we use the DAGs generated in Gaussian elimination method of [28] and the results are given in Fig. 8. The TANH is observed to perform better than BIL in terms of schedule length. As clear from the plots, a drastic performance enhancement is observed for lower CCR. As the improvement is evident even for small DAGs, it is indicative of the significance of TANH performance.

Diversity and CCR are quantified as:

$$\text{Diversity} = \text{Average of } \left[\frac{(\text{Max node cost} - \text{Min node cost})/2}{\text{Average node cost}}\right]$$

$$\text{and CCR} = \left[\frac{\text{Average edge cost}}{\text{Average node cost}}\right].$$

Also, since the algorithm or implementation of BIL is not available, we have run our TANH algorithm for four practical DAGs, and scheduling results have been obtained.

4.3 Scheduling Results for Practical DAGs

Benchmarking has been done in [15] to specify the characteristics of random DAGs with up to 500 nodes, which have also been tested for a small system (e.g., 8-node

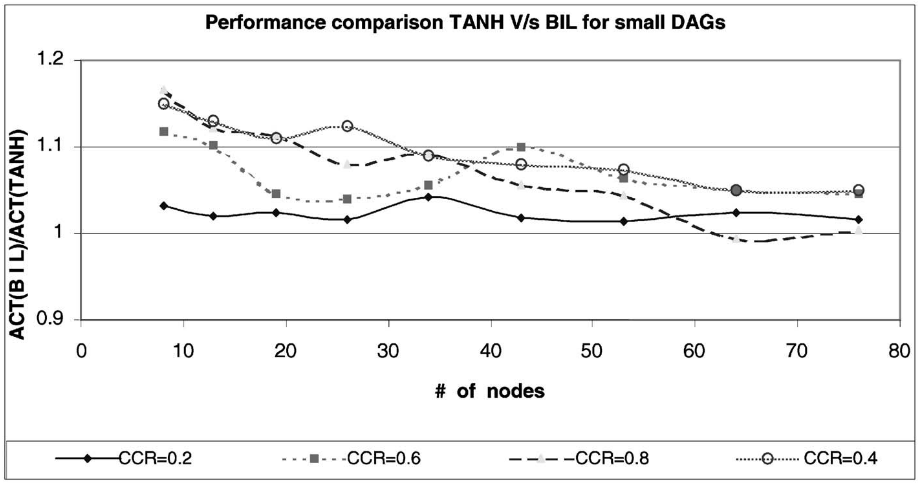


Fig. 8. Variation of (ACT(BIL)/ACT(TANH) V/s number of nodes.

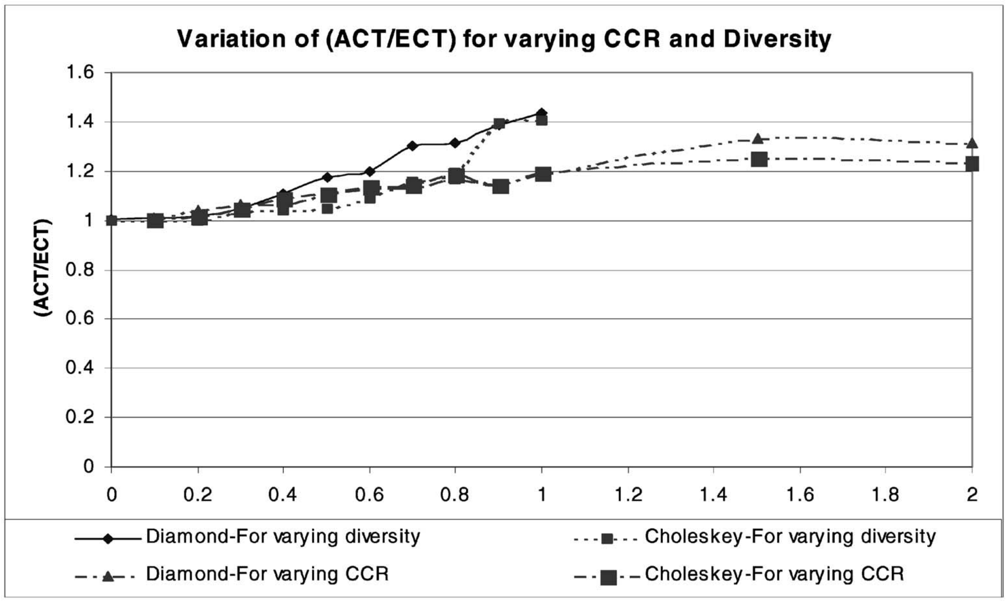


Fig. 9. Variation of (ACT/ECT) for varying CCR and diversity.

hypercube) and does not seem to be practical for large real-life applications. Therefore, we have selected four practical DAGs with up to 3,000 nodes and 25,000 edges and illustrate the usefulness of our algorithm. These are the Bellman-Ford algorithm, Master-Slave algorithm, Cholesky decomposition algorithm, and Diamond DAGs. The first three are a part of the synthetic parallel program project called ALPES and have been used in [11], [20], [21], [22], [23], [29] for observing the performance of different scheduling algorithms. The diamond DAG primarily indicates the master-slave phenomenon [3]. To reflect the heterogeneity of the processor, we use random changes in the execution time of a task over different processors. As the communication network is assumed to be the same for all processors, the same communication edge costs are used as in the original application. The input applications have been tested for different randomizing seeds and an average taken over these values for our simulation experiments to obtain the schedule time.

The four DAGs used contain about 3,000 nodes each. The numbers of edges vary from 4,000 up to 25,000. The numbers of predecessor tasks vary from 1 to more than 140 nodes. Comparison of ACT/ECT versus Diversity and CCR has been shown in Fig. 9 for all the four DAGs. The variation of ACT/ECT with respect to the number of relative processors is shown in Fig. 10. The relative number of processors is the ratio between the actual number of available processors to the critical number of processors required. The critical number of processors is calculated in accordance with the requirement of the algorithm for generating an optimal schedule. For Systolic DAG, the critical number of processors is 109, 75 for Cholesky DAG, 149 for Master-Slave DAG, and 315 for Bellman-Ford DAG.

5 Conclusions

The TANH algorithm introduced here has been proven to be optimal for DAGs by reducing the makespan. The

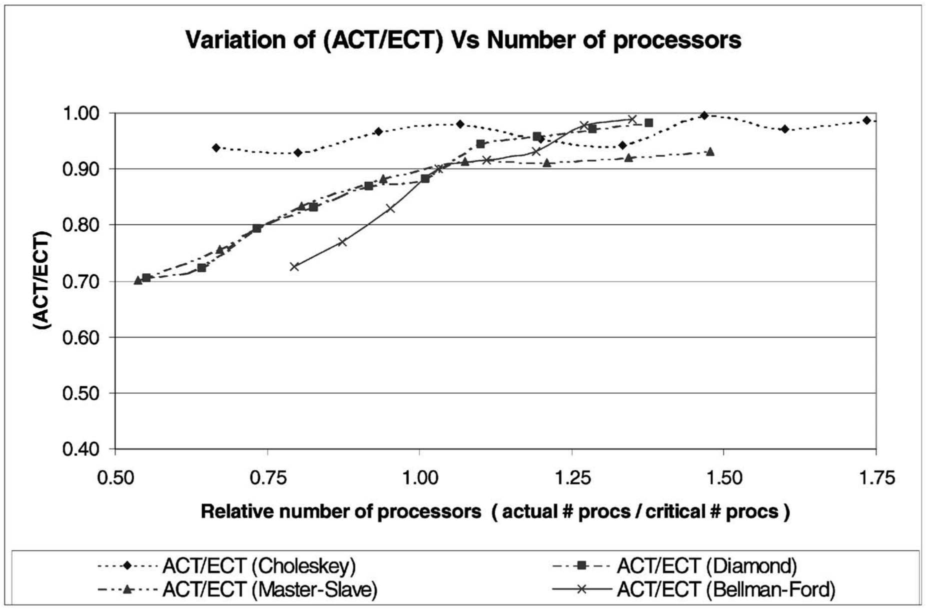


Fig. 10. Variation of (ACT/ECT) for varying number of processors.

performance of this algorithm has been observed to be substantially better that the existing scheme of BIL, the best existing scheduling algorithm for heterogeneous systems. The algorithm with a complexity of $O(V^2)$ is easily scalable upward or downward. It is also obvious that the algorithm will yield optimal results when the system is homogeneous, implying that all the processors have the same computation and other relevant characteristics.

ACKNOWLEDGMENTS

The authors wish to thank Samantha A. Ranaweera for providing a strong base for the solution to the problem. With his prior work, the authors have been able to devise an efficient scheduling algorithm and prove its optimality analytically. Thanks are also due to the referees for their useful comments. This work has been supported by the US National Science Foundation under grant numbers CCR-9902748 and ITR-0081696.

REFERENCES

[1] R.L. Graham, L.E. Lawler, J.K. Lenstra, and A.H. Kan, “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey,” *Annals of Discrete Math.*, pp. 287-326, 1979.
[2] T. Cassavant and J.A. Kuhl, “Taxonomy of Scheduling in General Purpose Distributed Memory Systems,” *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 141-154, 1988.
[3] S. Darbha and D.P. Agrawal, “Optimal Scheduling Algorithm for Distributed Memory Machines,” *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 1, pp. 87-95, Jan. 1998.
[4] G.C. Sih and E.A. Lee, “A Compile Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processors Architectures,” *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175-187, Feb. 1993.
[5] A. Gerasoulis and T. Yang, “A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs onto Multiprocessors,” *J. Parallel and Distributed Computing*, vol. 16, no. 4, pp. 276-291, Dec. 1992.
[6] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee, “Scheduling Precedence Graphs in Systems with Inter Processor Communication Times,” *SIAM J. Computing*, vol. 18, no. 2, pp. 244-257, Apr. 1989.
[7] S.J. Kim and J.C. Brown, “A General Approach to Mapping of Parallel Computations upon Multiprocessor Architectures,” *Proc. Int'l Conf. Parallel Processing*, vol. 3, pp. 1-8, Aug. 1988.
[8] S.S. Pande, D.P. Agrawal, and J. Mauney, “A New Threshold Scheduling Strategy for Sisal Programs on Distributed Memory Systems,” *J. Parallel and Distributed Computing*, vol. 21, no. 2, pp. 223-236, May 1994.
[9] S.S. Pande, D.P. Agrawal, and J. Mauney, “A Scalable Scheduling Method for Functional Parallelism on Distributed Memory Multiprocessors,” *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 4, pp. 388-399, Apr. 1995.
[10] V. Sarkar, *Partitioning and Scheduling Programs for Execution on Multiprocessors.* MIT Press, 1989.
[11] H.B. Chen, B. Shirazi, K. Kavi, and A.R. Hurson, “Static Scheduling Using Linear Clustering and Task Duplication,” *Proc. ISCA Int'l Conf. Parallel and Distributed Computing and Systems*, pp. 285-290, 1993.
[12] J.Y. Colin and P. Chretienne, “C.P.M. Scheduling with Small Computation Delays and Task Duplication,” *Operations Research*, pp. 680-684, 1991.
[13] I. Ahmad and K. Yu-Kwong, “On Exploiting Task Duplication in Parallel Program Scheduling,” *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 9, pp. 872-892, Sept. 1998.
[14] E. Hodzic and W. Shang, “On Supernode Transformation with Minimized Total Running Time,” *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 5, pp. 417-428, May 1998.
[15] K. Yu-Kwong and I. Ahmad, “Benchmarking and Comparison of the Task Graph Scheduling Algorithms,” *J. Parallel and Distributed Computing*, vol. 59, no. 2, pp. 381-422, Dec. 1999.
[16] C.I. Park and T.Y. Choe, “An Optimal Scheduling Algorithm Based on Task Duplication,” *IEEE Trans. Computers*, vol. 51, no. 4, Apr. 2002.
[17] K. Yu-Kwong and I. Ahmad, “Link-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors,” *Cluster Computing*, vol. 3, no. 2, pp. 113-124, Sept. 2000.
[18] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, “Heterogeneity Considered Harmful to Algorithm Designers,” Research Report No. 2000-24, Ecole Normale Superieure de Lyon, France, 2000.
[19] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert, “Static LU Decomposition on Heterogeneous Platforms,” Research Report No. 2000-44, Ecole Normale Superieure de Lyon, France, 2000.
[20] A. Ranaweera and D.P. Agrawal, “Task Duplication Based Scheduling Algorithm for Heterogeneous Systems,” *Proc. Int'l Parallel Processing Symp.*, pp. 445-450, 2000.

[21] A. Ranaweera and D.P. Agrawal, "A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing,* pp. 383-390, Aug. 2000.
[22] A. Ranaweera, "Task Scheduling Algorithms for Heterogeneous Systems," masters thesis, Univ. of Cincinnati, Ohio, 2000.
[23] H. Oh and S. Ha, "A Static Heuristic for Heterogeneous Processors," *Proc. European Conf. Parallel Processing (Euro-Par '96),* vol. 2, pp. 573-577, Aug. 1996.
[24] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.
[25] C.C. Hui and S.T. Chanson, "Allocating Task Interaction Graphs to Processors in Heterogeneous Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 9, pp. 908-926, Sept. 1997.
[26] M. Kafil and I. Ahmed, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency,* vol. 6, no. 3, pp. 42-51, July-Sept. 1998.
[27] H. Topcuoglu, S. Hariri, and M. Wu, "Task Scheduling Algorithms for Heterogeneous Processors," *Proc. Heterogeneous Computing Workshop,* pp. 3-15, Apr. 1999.
[28] M. Cosnard and E. Jeannot, "Compact DAG Representation and Its Dynamic Scheduling," *J. Parallel and Distributed Computing,* vol. 58, no. 3, pp. 487-514, Sept. 1999.
[29] S. Darbha and D.P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems," *J. Parallel and Distributed Computing,* vol. 46, no. 1, pp. 15-27, Oct. 1997.

Rashmi Bajaj received the BE degree in electronics and communication from University of Madras, India, in 1999 and the MSCE degree from the University of Cincinnati, Ohio, in 2001. She is a wireless engineer at France Telecom R&D in San Francisco, California. Her current research interests include short-range wireless communication technologies like WLANs and Bluetooth.

Dharma P. Agrawal is the Ohio Board of Regents Distinguished Professor of computer science and computer engineering and the founding director for the Center for Distributed and Mobile Computing in the Department of ECECS, University of Cincinnati, Ohio. He has been a faculty member at North Carolina State University, Raleigh, (1982-1998) and Wayne State University, Detroit (1977-1982). His current research interests are energy efficient routing and information retrieval in ad hoc and sensor networks, the effective handoff handling and multicasting in integrated wireless networks, interference analysis and routing in Piconets/Scatternet, the use of smart multibeam directional antennas for enhanced QoS, and the scheduling of periodic real-time applications. He has edited a tutorial text on advanced computer architecture, coedited texts entitled *Distributed Computing Network Reliability* and *Advances in Distributed System Reliability*, a self-study guide on Parallel Processing and Compiler Optimizations for Scalable Parallel Machines. Brooks/Cole has recently published his new textbook, *Introduction to Wireless and Mobile Systems*, specifically designed for computer science and engineering students. Dr. Agrawal is an editor for the *Journal of Parallel and Distributed Systems* and the *International Journal of High Speed Computing*. He has served as an editor of the *IEEE Computer* magazine, and the *IEEE Transactions on Computers*. He has been the program chair and general chair for numerous international conferences and meetings. He has received numerous certificates and awards from the IEEE Computer Society. He was awarded a "Third Millennium Medal" by the IEEE for his outstanding contributions. He has also delivered keynote speeches for five international conferences. He also has four patents in the wireless networking area. He is a fellow of the ACM and the IEEE. He has also been a Computer Science Accreditation Board visitor and an ABET team visitor. Very recently, he has been selected as a Fulbright Senior Specialist for a duration of five years.