Copyright 1997 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

In *Proc. of the Int'l Conf. on Parallel Processing*, August 11–15, 1997, Bloomingdale, Illinois, pp. 158–165.

A Parametrized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System

Jan Jonsson *

Department of Computer Engineering
Chalmers University of Technology
S-412 96 Göteborg, Sweden
janjo@ce.chalmers.se

Kang G. Shin

Real-Time Computing Laboratory
Dept. of Elec. Engr. and Computer Science
University of Michigan, Ann Arbor, MI 48109
kgshin@eecs.umich.edu

Abstract

In this paper we experimentally evaluate the performance of a parametrized branch-and-bound (B&B) algorithm for scheduling real-time tasks on a multiprocessor system. The objective of the B&B algorithm is to minimize the maximum task lateness in the system. We show that a last-in-first-out (LIFO) vertex selection rule clearly outperforms the commonly used least-lower-bound (LLB) rule for the scheduling problem. We also present a new adaptive lower-bound cost function that greatly improves the performance of the B&B algorithm when parallelism in the application cannot be fully exploited on the multiprocessor architecture. Finally, we evaluate a set of heuristic strategies, one of which generates near-optimal results with performance guarantees and another of which generates approximate results without performance guarantees.

1 Introduction

Since its introduction in the field of artificial intelligence, the branch-and-bound (B&B) strategy has been successfully used for finding optimal or near-optimal solutions to the problem of scheduling tasks on multiprocessor architectures. Recent work includes B&B strategies for task scheduling on distributed real-time systems [1], digital signal processing systems [2], and fault-tolerant systems [3]. The B&B strategy is an efficient method for searching the solution space of a scheduling problem. The solution space is often represented by a search tree where each vertex in the tree represents either a complete or a partial solution to the problem. With the aid of intelligent rules for selecting vertices to explore/expand and pruning (deleting) vertices that do not lead to an optimal solution, the complexity of the search can be drastically reduced as compared to that of an exhaustive implicit enumerative search. However, because the inherent exponential complexity of the B&B strategy cannot be completely eliminated, its applicability is in general restricted to small systems. When the application characteristics and/or the processing architecture are known and can be exploited very efficiently, however, the B&B strategy can perform well even for large systems [4, 5].

In this paper, we show how the B&B strategy can be used for non-preemptive scheduling of precedence-constrained tasks on a multiprocessor system subject to individual task deadlines. In particular, we show how a B&B algorithm can be applied to minimize the maximum *task lateness*, that is, the difference between a task's completion time and its deadline. For hard real-time systems where all tasks must be scheduled to meet their deadlines, the maximum task lateness indicates the scalability of the scheduled system workload. To evaluate the various aspects of the B&B strategy, we adopt a parametrized notation introduced by Kohler and Steiglitz [6].

Because the addressed scheduling problem is usually NP-complete [7], many heuristic approaches have been proposed to solve the problem in an efficient manner. For independent tasks on one processor, efficient B&B algorithms were proposed by Baker and Su [8], and McMahon and Florian [9]. A generalization of these algorithms was proposed by Lageweg *et al.* [10] for the case of precedence-constrained tasks. Optimal polynomial-time algorithms for precedence-constrained tasks on one processor have been proposed for the case when all release times are equal [11], as well as for preemptive scheduling [12]. For the multiprocessor case, on the other hand, only a few B&B algorithms have been reported in literature, addressing the problem of optimizing the schedules for deadline-constrained tasks. Peng and Shin [1] proposed an algorithm to minimize the *system hazard* (maximum normalized task response time), and Hou and Shin [4] proposed an algorithm to maximize the *probability of no dynamic failure* (all tasks meet their deadlines in the presence of component failures). The algorithms in [1, 4] both utilize a version of the optimal polynomial-time algorithm in [12] to minimize their performance measures. Recently, Abdelzaher and Shin [5] presented an algorithm for improving an initial solution where the assignment of tasks to processors is fixed and known beforehand.

The following features distinguish our B&B approach

*This work was done while the author was at the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. He was supported in part by a grant from the Volvo Research Foundation.

from others. First, we consider all possible permutations of scheduling tasks on a set of processors subject to a given set of precedence constraints. This is in contrast to the methods proposed in [1, 4] where the order of tasks is irrelevant as long as the precedence constraints are taken into account. Whenever this latter characteristic of the scheduling operation applies, all redundant vertices can be pruned from the search tree. Second, we allow related tasks to be assigned to different processors. This allows us to efficiently exploit the inherent parallelism in the application and hence increase the likelihood of meeting task deadlines. This flexibility in task distribution is not offered by task assignment techniques like those in [1, 5]. Third, by using non-preemptive run-time scheduling and hence allowing preemptions to occur only at task boundaries, we can easily include the cost of hardware context switches and still find the optimal solution. The presence of a non-negligible context switch overhead makes it very hard to find feasible schedules if unconstrained preemption is allowed.

The main contributions of this paper are as follows.

C1. We show that a last-in-first-out (LIFO) vertex selection rule outperforms the least-lower-bound (LLB) rule by at least an order of magnitude for the addressed problem. This is an interesting result since it shows that the LLB rule, the "default" rule in many B&B strategies, is not necessarily the best for all multiprocessor scheduling problems.

C2. We propose an efficient lower-bound function for the B&B algorithm under realistic assumptions on processor contention during scheduling. An experimental evaluation shows that our lower-bound function outperforms other techniques by almost an order of magnitude when task parallelism cannot be fully exploited on the system. Because the B&B strategy is computationally tractable only for systems with a relatively small number of processors, the quality of the proposed lower-bound function is all the more important.

C3. We present techniques that can significantly reduce the number of searched vertices at the price of either near-optimal results with performance guarantees or approximate results without performance guarantees. In particular, we find that by scheduling tasks in a fixed depth-first order, good approximate results can be attained at a very low computational cost.

The rest of the paper is organized as follows: Section 2 describes the assumed system and states the problem. Section 3 discusses the parametrized B&B algorithm. Section 4 describes the experimental setup. Section 5 presents the experimental evaluation. Section 6 discusses complementary results, and Section 7 summarizes the results in this paper.

2 System Models

2.1 The multiprocessor system

The multiprocessor system consists of a set $\mathcal{P} = \{p_q : 1 \leq q \leq m\}$ of identical processors. The processors communicate using an interconnection network. We assume that the interconnection network is an arbitrary topology that could include dedicated as well as shared links. The communication between two tasks residing on the same processor is done via accessing shared memory and its cost is assumed to be negligible. The communication cost associated with a message between two tasks on different processors is expressed as the product of the message length and the "nominal communication delay." The nominal delay is the worst-case communication delay that reflects the scheduling strategy used by the underlying interconnection network. We assume that the system is so designed that communication in the network can take place concurrently with processor computation.

2.2 The task system

We consider a real-time application that consists of a set $\mathcal{T} = \{\tau_i : 1 \leq i \leq n\}$ of tasks. Each task $\tau_i \in \mathcal{T}$ is characterized by a 4-tuple $\langle c_i, \phi_i, d_i, T_i \rangle$. The worst-case execution time c_i includes various architectural overheads such as the cost for cache memory misses, pipeline hazards and context switches. We also assume that the cost for packetizing and depacketizing messages are constant and included in the worst-case execution time of communicating tasks. The phasing ϕ_i is the earliest time at which the first invocation of the task will occur, measured relative to some fixed origin of time. The relative deadline d_i is the time within which the task must complete its execution, once it has been invoked. The period T_i is the time interval between two consecutive invocations of τ_i.

Let τ_i^k denote the kth invocation of the task, $k \in \mathbf{Z}^+$. The dynamic behavior of τ_i^k is then characterized by the pair (a_i^k, D_i^k), where the absolute arrival time $a_i^k = \phi_i + T_i(k-1)$ is the earliest time at which τ_i^k is allowed to start its execution, and the absolute deadline $D_i^k = a_i^k + d_i$ is the time by which τ_i^k must complete its execution.

Precedence constraints between tasks in a task set $\mathcal{T}$ are represented by an irreflexive partial order $\prec$ over $\mathcal{T}$. If task τ_j cannot begin its execution until task τ_i has completed its execution, we write $\tau_i \prec \tau_j$. In this case τ_i is said to be a *predecessor* of τ_j, and, conversely, τ_j a *successor* of τ_i. In addition, whenever $\tau_i \prec \tau_j$ and the condition $\neg(\exists \tau_k : (\tau_i \prec \tau_k) \wedge (\tau_k \prec \tau_j))$ holds, we write $\tau_i \prec\!\!\cdot\, \tau_j$. In this case, τ_i is said to be a *direct predecessor* of task τ_j and τ_j a *direct successor* to τ_i. A task which has no predecessors is called an *input task*, and a task which has no successors is called an *output task*.

The activities associated with message transfer from task τ_i to task τ_j are handled by a *communication channel* $\chi_{i,j}$ characterized by the tuple $\langle m_{i,j}, a_{i,j}, d_{i,j} \rangle$, where $m_{i,j}$ denotes the maximum message size, $a_{i,j}$ the message arrival time, and $d_{i,j}$ the relative deadline of the message. The real communication cost for sending a message depends on the communication scheduling strategy employed in the system and cannot be determined until the tasks have been assigned to processors.

The computational and communication demands of a task set, and intertask precedence constraints, are represented by a directed acyclic task graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. $\mathcal{N}$ is a set of nodes representing the tasks in the set $\mathcal{T}$. $\mathcal{A}$ is a set of directed arcs representing the precedence constraints between the tasks in $\mathcal{T}$, that is, if $\tau_i \prec \tau_j$ then $(\tau_i, \tau_j) \in \mathcal{A}$. Each node in $\mathcal{N}$ is annotated with a non-negative weight representing the computational demand of the corresponding task. For those arcs in $\mathcal{A}$ that represent communication channels, a non-negative weight is used for representing the message size.

A time-driven non-preemptive multiprocessor schedule for a task set $\mathcal{T}$ and a multiprocessor architecture $\mathcal{P}$ is the mapping of each task $\tau_i \in \mathcal{T}$ to a *start time* s_i and a processor $p_i \in \mathcal{P}$. The task is then scheduled to run without preemption on processor p_i in the time interval $[s_i, f_i]$, with its *finish time* being $f_i = s_i + c_i$. The time interval $[a_i, D_i]$, denoted by w_i, is called the *execution window* of τ_i. For periodic tasks, the static task parameters are assumed to satisfy $d_i \leq T_i$, that is, the execution windows of two invocations of the same task don't overlap in time. Furthermore, the execution time c_i cannot exceed the length $|w_i|$ of any execution window.

The schedule is said to be *valid* if (i) the conditions $s_i \geq a_i$ and $f_i \leq D_i$ are satisfied for each task τ_i, and (ii) all precedence constraints defined by the partial order $\prec$ over $\mathcal{T}$ are met. A task set is said to be *feasible* if there exists a valid schedule for the task set. We say that a task set is *schedulable* by a scheduling algorithm $\mathcal{S}$ if it produces a valid schedule for the task set.

2.3 Problem statement

For the system described in the previous sections, we want to find a schedule for a given set of tasks — a one-to-one mapping from each task to its assigned processor and start/stop times — such that the maximum task lateness $L_{max} = \max\{f_i - D_i : \tau_i \in \mathcal{T}\}$ is minimized. As a secondary performance measure we want to minimize the number of vertices searched, indicating the computational complexity of the B&B algorithm.

3 B&B Algorithm

The search for a solution to the multiprocessor scheduling problem is performed with the aid of a *search tree* that represents the solution space of the problem, that is, all possible permutations of task–to–processor assignments and schedule orderings. Each vertex in the search tree represents one specific task–to–processor assignment and schedule ordering, and one or many vertices represent the optimal solution whenever one exists. The *root vertex* of the search tree represents an empty schedule and each of its descendant vertices (children) represents the scheduling of one specific task on one specific processor. The children of each of these child vertices represents the scheduling of yet another task on one processor.

A *goal vertex* in the search tree represents a complete solution where all tasks have been scheduled on the processors. An "acceptable" complete solution is also called a *feasible* solution. An *intermediate vertex* represents a partially-complete schedule. The *level* of a vertex is the number of tasks that have been assigned to any processor in the current schedule. The *cost* of a vertex is the quality of the schedule represented by the vertex; in this paper, it is the maximum task lateness for the schedule represented by the vertex.

When no precedence constraints exist between tasks, the number of goal vertices in a search tree is $n!m^n$ for a multiprocessor system with n tasks and m processors. If precedence constraints exist, the number of goal vertices can be greatly reduced. If the maximum number of child vertices of a vertex is k, then the number of goal vertices in the search tree is at most $k^n m^n$. Because of the exponentially growing number of vertices in the search tree, vertices are normally not generated until the B&B algorithm needs to explore them. Whenever a new vertex is generated and it could lead to an optimal solution, it will be referred to as an *active vertex*. The order in which the vertices in the search tree will be explored is governed by a set of rules that are often heuristic. As will be demonstrated later, the choice of rules dictates the performance of the B&B algorithm. The power of the B&B strategy lies in alternating branching and bounding operations on the set of active vertices. Branching refers to the process of generating the child vertices of an active vertex, while bounding refers to the process of evaluating the cost of new child vertices.

To describe our B&B algorithm, we will use the parametrized notation introduced by Kohler and Steiglitz [6]. A B&B algorithm for solving a permutation problem such as the multiprocessor scheduling problem can then be characterized by a 9-tuple $\langle B, S, E, F, D, L, U, BR, RB \rangle$, where B is the vertex branching rule, S the vertex selection rule, E the vertex elimination rule, F the characteristic function, D the vertex domination rule, L the lower-bound cost function, U the upper-bound solution cost, BR an inaccuracy limit for the cost of a feasible solution, and RB are upper bounds on the time and space resources that are available for solving the problem.

F is used for eliminating partial solutions that will not lead to a valid complete solution. D is used for comparing partial solutions and eliminating "inferior" partial solutions. While F and D have been shown to be very efficient in reducing the complexity of B&B algorithms, we have chosen not to use them, in order to preserve our results as general as possible. As has been successfully demonstrated in [1, 2, 4], the effects of F and D are most powerful when they are designed with a specific processor scheduling strategy in mind.

BR is an indicator on how far the cost for a feasible solution is allowed to deviate from that of the optimal solution. This limit is useful when near-optimal results with performance guarantees [13, pp. 121-151] are desired. When

```
Algorithm BRANCH-AND-BOUND:
1. initialize active set AS with root vertex;
2. set best vertex v_u to root vertex;
3. while { AS ≠ ∅ } loop
4.   select a vertex v_b in AS according to vertex
       selection rule S;
5.   if { stop condition for S } then
       break loop;
6.   generate a set DB of child vertices to vertex v_b
       according to vertex branching rule B;
7.   calculate the cost L(v) for each vertex v in DB
       using the lower-bound function L;
8.   eliminate vertices in DB and AS according to
       vertex elimination rule E;
9.   move all remaining vertices in DB to AS;
10. end loop;
```

Figure 1: The branch-and-bound algorithm.

performance guarantees are desired, the cost L_{opt} for the optimal solution and the cost L_{acc} for a feasible solution are related as given by the relation $|L_{opt}| \leq |L_{acc}| \leq (1+BR)|L_{opt}|$. This means that when the inaccuracy limit $BR = 0\%$, the only feasible solution is the optimal one.

The resource bound RB can be viewed as a triple $\langle \text{TIMELIMIT}, \text{MAXSZAS}, \text{MAXSZDB} \rangle$ where TIMELIMIT is the maximum allowable time to find a solution, MAXSZAS is the maximum allowable size of the set of active vertices, and MAXSZDB is the maximum allowable number of child vertices of any vertex. Should the time limit be exceeded, the algorithm either fails or terminates with the best solution found so far. Should any of the storage bounds be exceeded, the algorithm must dispose of one or more of the active intermediate solutions, thereby running the risk of missing the optimal solution.

In [6] it is proven that (i) one cannot lose by eliminating those newly-generated vertices that exceed an upper-bound solution cost, and that (ii) one cannot lose by using a better solution as the initial upper bound. Similar results have been reported for the well-known A* algorithm [14].

3.1 Algorithm

Figure 1 shows a pseudo-code form of the parametrized B&B algorithm. The algorithm takes a task graph as the input and produces an annotated task graph containing information about task start and finish times for the best schedule. In this algorithm, an *active set* AS is used to hold all the active vertices.

The algorithm in Figure 1 differs slightly from the one proposed in [6] in that goal vertices are never inserted into the set of active vertices. Instead, a goal vertex either becomes the new best vertex if its cost is lower than the cost of the currently best vertex, or it will be pruned. Using this strategy, many unnecessary insertions into the active set are avoided. The different steps of the algorithm are described below in detail.

Initialize active set (Step 1 – 2): In this step the root vertex is created and initialized with an empty schedule. The cost of the root vertex is set according to the upper-bound cost U. The root vertex is then inserted into the active set AS and a variable v_u (best vertex) is set to the root vertex.

Select a vertex to explore (Step 4): A branch vertex v_b is selected among the active vertices according to the vertex selection rule S.

Generate child vertices (Step 6): A set DB of child vertices is generated according to the vertex branching rule B.

Calculate child vertex costs (Step 7): A lower bound on the cost of each vertex in DB is calculated using a lower-bound function L. Each vertex's set of tasks is scheduled on their processors and the overhead introduced by tasks not yet scheduled is estimated.

Eliminate vertices (Step 8): The vertices in DB and AS are now inspected as to whether they are capable of guiding the search to a feasible or optimal solution. The vertex elimination rule E determines which vertices to keep and which ones to prune.

Move remaining child vertices (Step 9): Move the child vertices in DB after the vertex elimination step applied to AS.

Repeat until no vertices remain (Step 3, 5 and 10): The main loop in the algorithm is repeated until no vertices are left in AS or until the stop condition for the vertex selection rule is satisfied. Unless the best vertex v_u is the root vertex, in which case the algorithm has failed to find a solution, v_u will be the optimal solution.

3.2 Vertex selection rule S

From the set AS of currently-active vertices, the vertex rule S selects the next candidate vertex to be explored by the B&B algorithm.

The following three vertex selection rules are commonly used in literature:

- The Least-Lower-Bound (LLB) rule, S_{LLB}, selects the vertex $v \in AS$ that has the least lower-bound cost $L(v)$. The stop condition for S_{LLB} is reached when the lower-bound cost of the selected vertex is equal to, or higher than, the current upper-bound cost $L(v_u)$.
- The First-In-First-Out (FIFO) rule, S_{FIFO}, selects the vertex $v \in AS$ that was generated first. The stop condition for S_{FIFO} is reached when AS is empty.
- The Last-In-First-Out (LIFO) rule, S_{LIFO}, selects the vertex $v \in AS$ that was generated last. The stop condition for S_{LIFO} is reached when AS is empty.

The FIFO strategy is very inefficient for multiprocessor scheduling because its objective is to search for a solution at the lowest possible vertex level. For a multiprocessor scheduling problem where all goal vertices are at the same level, the FIFO strategy will generate all intermediate vertices before finding any complete solution. Since this is a very ineffective strategy, we will not discuss or analyze S_{FIFO} any further, but instead focus on S_{LLB} and S_{LIFO}.

3.3 Vertex branching rule B

The vertex branching rule is instrumental in guiding the traversal of the search tree, based mainly on task precedence constraints. For each vertex in the search tree, there is a set of *ready* tasks; a task is said to be ready when all of its predecessors have been scheduled. One important factor to take into account at this stage is the *commutativity* of the underlying processor scheduling operation. A processor scheduling operation is said to be *commutative* if the order in which tasks are scheduled on the processors doesn't matter to the final scheduling result. Whenever commutativity applies, it is possible to identify those vertices that will yield the same scheduling result regardless of the order in which they are explored. All but one of these vertices can then be pruned before the algorithm is applied, which will significantly reduce the complexity of the algorithm. An example of a commutative processor scheduling operation is the one proposed by Baker *et al.* [12]. This operation has been successfully used in the B&B algorithm proposed in [1, 4]. It should be noted, however, that commutativity only applies for the B&B algorithms in [1, 4] under the assumption that the ideal preemptive scheduling strategy in [12] is used and that interprocessor communication scheduling is also commutative. For a non-preemptive scheduling strategy, the single-processor scheduling problem would be NP-complete [13], thereby effectively prohibiting the use of such strategies as those used in [1, 4].

Specifically, we will evaluate the following vertex branching rules.

- The Depth-First (DF) rule, B_{DF}, selects the task to use for generation of a child vertex from the head of a list sorted according to a depth-first traversal of the task graph.
- The Breadth-First-One-Task (BF1) rule, B_{BF1}, selects the task to use for generation of a child vertex from the head of a list sorted according to the level of a task. The level of a task is calculated in the same manner as in [4].
- The Breadth-First-All-Tasks (BFn) rule, B_{BFn}, selects all available tasks to use for generation of child vertices.

Both DF and BF1 are unable to guarantee an optimal solution unless processor and communication scheduling operations are commutative, whereas BFn guarantees to find an optimal solution if any.

3.4 Upper-bound solution cost U

An upper-bound solution cost is used to initialize the root vertex. The more accurate the upper-bound cost is, the faster the B&B algorithm will get because more vertices can be pruned at each step.

3.5 Lower-bound cost function L

A lower-bound cost function is used for the bounding operation where "pessimistic" estimates on the maximum task lateness are calculated for newly-generated vertices. So, it is important to estimate the lateness for those tasks not yet scheduled. The lower bound $\hat{L}$ on the maximum task lateness for the task set $\mathcal{T}$ is defined as

$$\hat{L} = \max\{\hat{f}_i - D_i \ : \tau_i \in \mathcal{T}\}$$

where $\hat{f}_i$ is the estimated finish time of task τ_i.

We will evaluate two lower-bound cost functions. The first function, L_{LB0}, is similar to the one used in [4]. Starting with the output tasks, task τ_i's estimated finish time $\hat{f}_i$ is defined recursively as:

$$\hat{f}_i := \begin{cases} f_i \\ \max(\{\hat{f}_i\} \cup \{\max\{\hat{f}_j, a_i\} + c_i : \tau_j \prec \tau_i\}) \end{cases}$$

The first case in this equation applies when τ_i has been assigned to and scheduled on a processor; otherwise, the second case applies.

The second lower-bound function, L_{LB1}, is similar to L_{LB0} but also takes processor contention into account. Starting with the output tasks, task τ_i's estimated finish time is defined recursively as:

$$\hat{f}_i := \begin{cases} f_i \\ \max(\{\hat{f}_i\} \cup \{\max\{\hat{f}_j, a_i, \ell_{min}\} + c_i : \tau_j \prec \tau_i\}) \end{cases}$$

Here, ℓ_{min} is the earliest time at which a new task can be scheduled on any processor.

3.6 Vertex elimination rule E

The vertex elimination rule is applied after calculating a lower-bound cost for all newly-generated vertices. Its main use is in the elimination of vertices in DB and/or AS that will not lead to a feasible solution.

We will evaluate one such rule, called the Upper-Bound-Cost-to-DB-and-AS (U/DBAS) rule, $E_{\mathrm{U/DBAS}}$. This rule compares the cost of every vertex in DB and AS with the cost of the current upper-bound cost of vertex v_u and removes all those vertices with equal or higher costs. A pseudo-code form of the algorithm for this rule is shown in Figure 2.

```
Algorithm U/DBAS:
1. set v̂_u to the goal vertex in DB with the lowest cost;
2. if { v̂_u exists and L(v̂_u) < L(v_u) } then
3.   remove v̂_u from DB;
4.   set v_u := v̂_u
5. end if;
6. prune each vertex v in DB/AS for which L(v) ≥ L(v_u);
```

Figure 2: The vertex elimination rule $E_{\mathrm{U/DBAS}}$.

4 Experimental Setup

The experimental platform used consists of a shared-bus homogeneous multiprocessor system whose size ranges from 2 to 4 processors. The shared bus is time-multiplexed in such a way that the communication cost between two processors is one time unit per transmitted data item.

4.1 Workload

In the experiments[1], a set of task graphs were generated using a random task graph generator. Each task graph contains between 12 and 16 tasks. Task execution times were chosen randomly according to a uniform distribution with mean execution time of 20 time units. Task execution times are allowed to deviate by at most ±99% from the mean execution time. An end-to-end deadline was chosen for each input–output task pair in the generated graph in such a way that the overall laxity ratio of the end-to-end deadline to the accumulated task graph workload corresponds to 1.5. The precedence constraints in the task graph were also randomly generated. The number of successors/predecessors to each task was chosen at random to be in the range of 1 to 3, and the depth of the task graph was chosen to be between 8 and 12 levels. The number of data items in each message passed between a pair of tasks was chosen in such a way that the communication–to–computation cost ratio (CCR) (of the average message communication cost to the average task execution time) is 1.0.

4.2 Deadline assignment

To assign arrival times and deadlines to each task in $\mathcal{T}$, we used the deadline assignment technique proposed in [16]. Each series of direct successors between an input–output task pair is assigned *slices*, non-overlapping execution windows, of the task pair's end-to-end deadline. The slicing technique is suitable for distributing end-to-end deadlines in real-time systems, because it allows individual tasks to be scheduled independently of each other, thereby allowing for different scheduling strategies on different processors.

4.3 Task scheduling

We used a task scheduling strategy that assumes a non-preemptive time-driven processor run-time model. A new task is scheduled on a processor at the earliest possible start time, while taking into account possible interprocessor communication costs and arrival time constraints of the task, but later than all tasks that have previously been scheduled on the processor. This scheduling strategy exhibits a time complexity that is quadratic in the number of tasks in the system. Unfortunately, the simplicity of the scheduling operation implies that it is not commutative.

4.4 Upper-bound estimation

An initial upper-bound cost U for the B&B algorithm was derived by applying a polynomial-time Earliest-Deadline-First (EDF) algorithm. For each scheduling step, the EDF algorithm selected one task from all schedulable tasks. The task with the closest absolute deadline was selected, and then scheduled on the processor that yielded the earliest start time. The set of schedulable tasks was then updated.

[1] All modeling and simulation experiments were performed within FEAST [15], a framework for evaluation of allocation and scheduling techniques for distributed hard real-time systems.

5 Experimental Evaluation

In this section we evaluate the performance of the B&B strategy presented in Section 3. Unless specifically noted, the following B&B parameter choices were used: $BR =$ 0%, TIMELIMIT $=$ 4 hours, MAXSZAS $= \infty$, and MAXSZDB $= \infty$. The observed performance indices were (i) the number of searched vertices and (ii) the maximum task lateness. In Figure 3, the plots for the number of searched vertices and the maximum task lateness are located in the upper and lower part in the figure, respectively. As a reference, we have also included in all plots the results obtained for the greedy EDF algorithm in Section 4.4.

In all the plots presented, every reported value is an average of the observed performance data, taken over the set of simulation runs, one for each parameter combination. The number of simulation runs were chosen in such a way that a 90% (95%) confidence level could be achieved for a maximum error within 10% (0.5%) of the average values reported for the number of generated active vertices (maximum task lateness). Simulations that exceeded the time limit as defined by TIMELIMIT were removed from the evaluation. These simulations were found to constitute less than 1% of all simulation runs for each reported value and the confidence levels presented above have been calculated without including the removed simulations.

5.1 Effect of vertex selection rule

Figure 3(a) illustrates how the performance of the B&B algorithm is affected by different choices of vertex selection rule S. Each plot shows the performance of the algorithm when vertex selection was made according to the S_{LLB} and S_{LIFO} rule, respectively.

As can be seen in the upper plot, S_{LIFO} outperforms S_{LLB} by more than an order of magnitude for all system sizes. This is a totally unsuspected result since S_{LLB} has been predominantly used in almost all B&B scheduling algorithms for real-time scheduling. For instance, when scheduling for minimized makespan (schedule length), a good lower-bound cost for an "early" vertex (at a low level in the search tree) is an indicator for a good complete solution. This correlation between the cost of an early vertex and a goal vertex is not necessarily provided when scheduling to minimize task lateness.

The performance of S_{LIFO} differs from that of the greedy EDF algorithm by a little more than an order of magnitude for a small system, and up to two orders of magnitude for a larger system. As can be seen in the lower plot, however, the B&B algorithm yields 5% better (more negative) task lateness for a small system and approximately 3% better task lateness for larger systems.

5.2 Effect of lower-bound function

Figure 3(b) illustrates how the performance of the B&B algorithm is affected by different choices of lower-bound function L. Each plot shows the performance of the algorithm when the lower-bound cost was calculated according

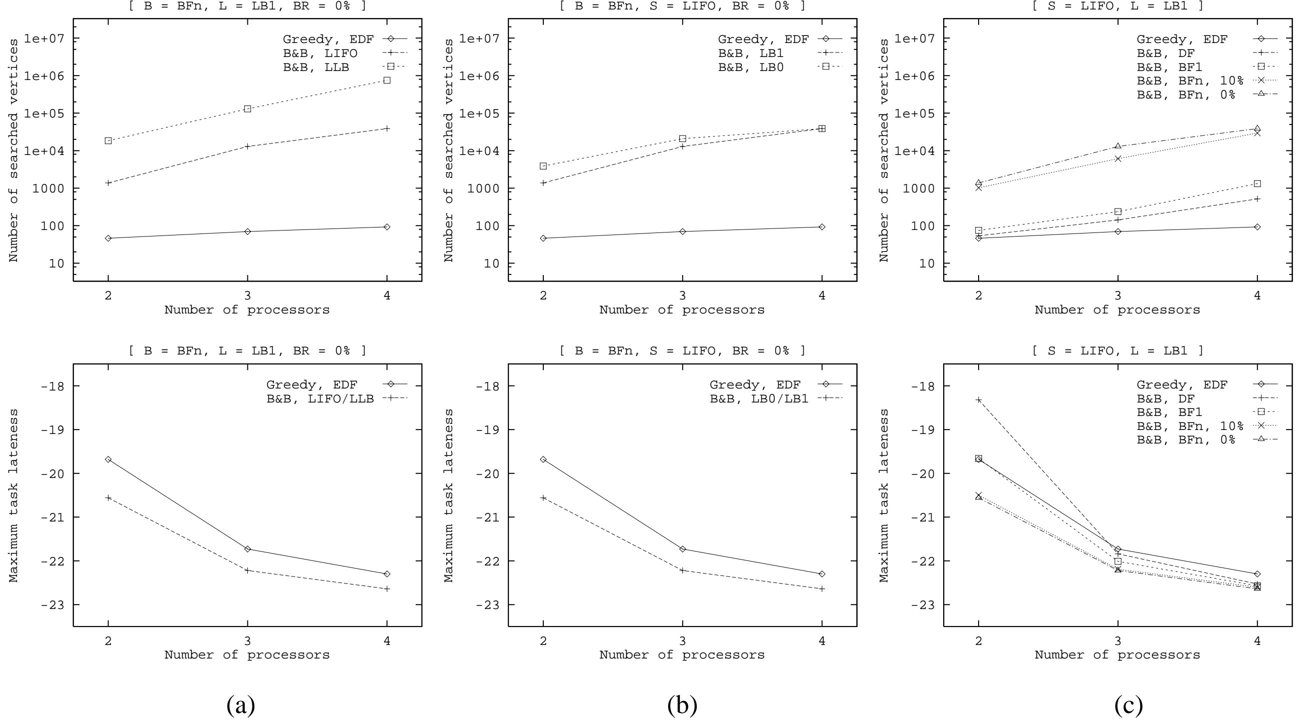

Figure 3: Performance as a function of: (a) vertex selection rule; (b) lower-bound function; (c) approximation strategy.

to L_{LB0} and L_{LB1}, respectively.

As can be seen in the upper plot, L_{LB1} outperforms L_{LB0} by approximately half an order of magnitude for a small system. As the system size increases, however, the performance of the two functions converge, since for larger systems the application parallelism can be exploited better and the adaptive characteristics of L_{LB1} become less significant.

5.3 Effect of approximation strategy

Figure 3(c) illustrates how the performance of the B&B algorithm is affected by different choices of approximation method. Each plot shows the performance of the algorithm when the lower-bound cost was calculated according to L_{LB1}. Results for the approximate vertex branching rules B_{DF} and B_{BF1} are shown, as are two versions of the branching rule B_{BFn}: one near-optimal with $BR = 10\%$, and one optimal with $BR = 0\%$. Also included in the plots are the results obtained for the optimal B&B algorithm with $S = S_{\mathrm{LIFO}}$.

As can be seen in the upper plot, the approximate vertex branching rules outperform near-optimal ones by a little more than an order of magnitude. Among the approximate vertex branching rules, B_{DF} yields a small but significant performance gain over B_{BF1}, but this advantage comes at the price of worse (less negative) task lateness as can be seen in the lower plot. It is worth noting that, for a small system, the task lateness for B_{DF} is worse than that for the greedy EDF algorithm. This can be attributed to the fact that if the application parallelism exceeds the available processing parallelism, a depth-first traversal will schedule some input tasks much later than a breadth-first traversal would do. As input tasks are delayed, their successor tasks will be delayed, hence increasing the potential for worsening task lateness. As the system size increases, more parallelism can be exploited in the system and the task lateness of both approximative vertex branching rules converge to that of the optimal. Also apparent in the plots is that the performance gain attained for B_{BFn} with performance guarantees is at best 100% as compared to an optimal version, while the maximum task lateness is kept very close to the optimal one.

6 Discussion

Due to space limit, we have omitted the results of some complementary experiments. We will briefly summarize these results here. We have evaluated the B&B strategy for task graphs with varying degrees of parallelism. Using the same basic experimental setup in Section 4, we found that when the parallelism in the task graph increases, a lower-bound cost function that takes processor contention into account will give even better performance for the B&B algorithm. We have also evaluated the B&B strategy for different values on the communication–to–computation cost ratio (CCR). Here, we found that lower CCR gives better B&B algorithm performance. This is because the cost estimates derived by lower-bound cost functions will be more accurate, thus making the B&B algorithm converge faster.

We have also investigated the impact of an upper-bound

solution cost on the performance of the B&B algorithm. We found that by using a greedy algorithm to derive approximate initial upper-bound costs, the performance of the B&B algorithm was improved by more than 200% as compared to an approach where the initial upper-bound cost was set to a positive value.

Finally, we remark on the practicality of the B&B strategy. In our simulations, the size of the simulated system had to be restricted because of limitations in the hardware simulation milieu. All simulations were made using a SPARCstation-4 machine with 64 Megabyte primary memory, running the SunOS 5.5 operating system. For some simulations, such large amounts of active nodes had to be maintained in virtual memory that machine performance was degraded significantly because of system thrashing, that is, constant shuffling of virtual memory pages to and from the machine's hard disk swap area. This behavior was particularly apparent during those simulations that employed the LLB vertex selection rule because of its quite random (with respect to virtual memory location) access pattern among the active vertices. For those simulations that employed the LIFO vertex selection rule, system thrashing did not occur since the access pattern for this rule matches very well the page replacement strategy used by the SunOS operating system, that is, LRU. This, of course, is another reason for choosing the LIFO rule over the LLB rule.

7 Conclusions

In this paper we have evaluated how effective the B&B strategy can be in finding optimal multiprocessor schedules of precedence-constrained tasks. In particular, we found that if intelligent rules for vertex selection, branching and elimination are used in the B&B algorithm, the problem of minimizing the maximum task lateness can be solved within reasonable time for moderate-size systems. For larger system sizes where machine and time limitations become a major concern, approximate versions of the B&B algorithm are viable alternatives to faster but less accurate greedy algorithms.

References

[1] D.-T. Peng and K. G. Shin, "Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems," *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, New Port Beach, California, June 1989, pp. 190–198.

[2] K. Konstantinides, R. T. Kaneshiro, and J. R. Tani, "Task Allocation and Scheduling Models for Multiprocessor Digital Signal Processing," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, no. 12, pp. 2151–2161, Dec. 1990.

[3] S. M. Shatz, J.-P. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Systems," *IEEE Trans. on Computers*, vol. 41, no. 9, pp. 1156–1168, Sept. 1992.

[4] C.-J. Hou and K. G. Shin, "Replication and Allocation of Task Modules in Distributed Real-Time Systems," *Proc. of the IEEE Int'l Symposium on Fault-Tolerant Computing*, Austin, Texas, June 15–17, 1994, pp. 26–35.

[5] T. F. Abdelzaher and K. G. Shin, "Optimal Combined Task and Message Scheduling in Distributed Real-Time Systems," *Proc. of the IEEE Real-Time Systems Symposium*, Pisa, Italy, Dec. 5–7, 1995, pp. 162–171.

[6] W. H. Kohler and K. Steiglitz, "Enumerative and Iterative Computational Approaches," *Computer and Job-Shop Scheduling Theory*, E. G. Coffman, Jr, Ed., chapter 6, pp. 229–287. Wiley, New York, 1976.

[7] P. Brucker, M. R. Garey, and D. S. Johnson, "Scheduling Equal-Length Tasks under Treelike Precedence Constraints to Minimize Maximum Lateness," *Mathematics of Operations Research*, vol. 2, no. 3, pp. 275–284, Aug. 1977.

[8] K. R. Baker and Z.-S. Su, "Sequencing with Due-Dates and Early Start Times to Minimize Maximum Tardiness," *Naval Research Logistics Quarterly*, vol. 21, no. 1, pp. 171–176, Mar. 1974.

[9] G. McMahon and M. Florian, "On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness," *Operations Research*, vol. 23, no. 3, pp. 475–482, May/June 1975.

[10] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Minimizing Maximum Lateness on One Machine: Computational Experience and Some Applications," *Statistica Neederlandica*, vol. 30, no. 1, pp. 25–41, 1976.

[11] E. L. Lawler, "Optimal Sequencing of a Single Machine Subject to Precedence Constraints," *Management Science*, vol. 19, no. 5, pp. 544–546, Jan. 1973.

[12] K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints," *Operations Research*, vol. 31, no. 2, pp. 381–386, Mar./Apr. 1983.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[15] J. Jonsson and J. Vasell, "Evaluation and Comparison of Task Allocation and Scheduling Methods for Distributed Real-Time Systems," *Proc. of the IEEE Workshop on Real-Time Applications*, Montreal, Canada, Oct. 21–25, 1996, pp. 226–229.

[16] J. Jonsson and K. G. Shin, "Deadline Assignment in Distributed Hard Real-Time Systems with Relaxed Locality Constraints," *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, Baltimore, Maryland, May 27–30, 1997, pp. 432–440.