The Allocation and Scheduling of Precedence- and Timing-Constrained Tasks with Communication Delays

Bo-Chao Cheng Alexander D. Stoyenko Thomas J. Marlowe Sanjoy Baruah

Real-Time Computing Laboratory
Department of Computer and Information Science
New Jersey Institute of Technology
University Heights, Newark NJ 07012 USA
{bcceng, alex, marlowe, sanjoy}@rtlab12.njit.edu*

Abstract

The problem of non-preemptively scheduling a set of n tasks on m identical processors with communication overhead subject to precedence and deadline constraints is considered. A new heuristic with the time complexity of $O(n^2m)$, Least Space-Time First (LSTF), is proposed to minimize the maximum tardiness. From simulation results, it is shown that LSTF outperforms other heuristic algorithms.

1 Introduction

Much research has been done on the allocation of tasks in a parallel or distributed processor computer with respect to different models and performance criteria [1, 8, 9, 10]. In this paper, we consider the problem of scheduling and allocating the tasks of a precedence- and timing-constrained task graph with communication delays onto a set of processors in a way that minimizes maximum tardiness τ, defined as $\tau = \max_{i=1}^{n}\{\max\{0, (\gamma_i - D_i)\}\}$ where γ_i is the completion time and D_i is the deadline of task i respectively.

There are two advantages of using τ to evaluate scheduling performance. First, a scheduling algorithm which minimizes tardiness will necessarily satisfy all deadlines for any schedulable set of tasks (since "schedulable" is equivalent to "tardiness equals zero"). Secondly, in practical systems, engineers may need information on how late the tasks may be, and how much penalty each should pay if it misses its deadline.

*Marlowe is also with the Department of Mathematics and Computer Science, Seton Hall University, South Orange, NJ 07079 USA.

When all task deadlines are 0, the problem of minimizing tardiness reduces to the problem of minimizing the makespan, which is known to be a NP-complete [5]. Therefore, the general problem of scheduling to minimize tardiness is NP-hard [4].

Now, we specify the task and hardware model formally. Suppose that we have a number of m identical processors and a set $T = \{T_1, T_2, ..., T_n\}$ of n non-preemptive tasks with each task T_i being characterized by an ordered pair (X_i, D_i), where X_i is the cpu execution requirement and D_i is the deadline of task T_i. There exist precedence constraints "$\rightarrow$" on T. Each edge, $T_i \rightarrow T_j$, stands for a precedence relationship between predecessor T_i and successor T_j. Task T_i is defined to be *ready* when all its predecessors have completed execution. A non-negative integer $\lambda(T_i, T_j)$, the data volume sent from T_i to T_j, is associated with each edge. This task model is called the enhanced directed acyclic graph (EDAG) [6] and represented as $G = G(T, \rightarrow, X, D, \lambda)$. Figure 1-a shows an example of an EDAG. Let ρ_i denote the processor hosting task T_i,

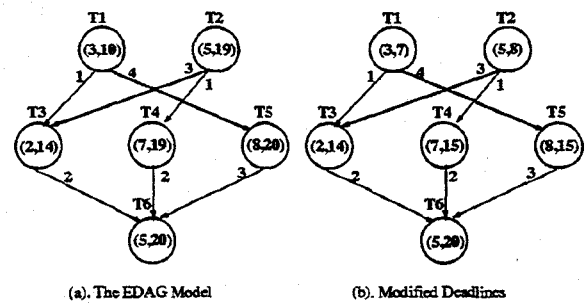

Figure 1. An example Task System.

and $\kappa(\alpha, \beta)$ denote the time to transfer a data unit from processor α to β. We assume that communication is

0-8186-7614-0/96 $5.00 © 1996 IEEE

contention-free, so $\kappa(\alpha,\beta)$ is constant; and that communication within a processor incurs no delays; i.e., when $\alpha = \beta$, $\kappa(\alpha,\beta) = 0$. The communication delay between T_i and T_j, $C_{i,j}$, is then $\kappa(\rho_i,\rho_j)*\lambda(T_i,T_j)$ time units.

2 Development of LSTF Heuristic

The Earliest Task First (ETF) heuristic [6] schedules each ready task on the processor at which it can be scheduled at the earliest time. Without deadline constraints, the makespan (ω_{ETF}) generated by this heuristic satisfies $\omega_{ETF} \leq (2 - \frac{1}{m})\omega^i_{opt} + \eta$, where ω^i_{opt} is the optimal schedule length ignoring the communication delay and η is the maximum communication requirement along all paths in G. The basic Least Space-Time First (LSTF) algorithm works well to minimize the maximum tardiness, τ, in the absence of costs for communication [3].

To enhance the basic LSTF scheduler to handle interprocessor communication delays, we have integrated LSTF with ETF. This is done by selecting the task to be executed by the basic LSTF algorithm and then assigning the chosen task to a processor by ETF. The LSTF scheduler consists of two phases as illustrated below:

1. Phase one:
   (a) compute modified deadlines in the standard way, using precedence relationships and cpu execution requirements, by Equation 1. Fig 1-b shows the result of this step on the task graph of Fig 1-a.
   $$D_i = \min_j(D_i, D_j - X_j) \qquad (1)$$
   where j ranges over all of i's successors.
   (b) assign static space-time $(D_i - X_i)$ to each task[1], where D_i is the new modified deadline of task i.
   (c) construct a ready node priority queue, R, based on space-time (with least space-time having highest priority and the head of R having least space-time).
2. Phase two: execute the worklist algorithm shown below[2]:

[1] Unlike the conventional *slack*, calculated as the difference of old deadline and execution requirement, *space-time* is a priority measurement that depends upon the precedence relation in addition to deadline and execution requirement.

[2] the function *earliest_start()* returns the identifier of the processor on which task A would be able to begin execution at the earliest time.

```
(1) While R ≠ ∅
(2) begin
(3)     Let A be the task at the head of R
(4)     P = earliest_start(A)
(5)     assign A to run on P
(6)     delete A from R
(7)     insert all ready children of A into R
(8) end
```

LSTF Scheduler

It takes $O(n)$ time to create the priority queue R, and each insert and deletemin operation can be performed in $O(\log n)$ time. Hence, the bottleneck of LSTF scheduler is in *Line (4)*, whose time complexity is $O(nm)$ [6] per instance. Since *Line (4)* is executed n times, the complexity of LSTF is $O(n^2m)$.

Before we illustrate the function *earliest_start()*, we define a number of terms. Let $\gamma(T_i)$ be the completion time of task T_i. $\Re(T_i,P)$ denotes the earliest time at which task T_i on processor P will have received all messages from all its predecessors:

$$\Re(T_i,P) = \max_{T_j}\{\gamma(T_j) + C_{j,i}\} \qquad (2)$$

where T_j ranges over all of T_i's predecessors.

Let $\varepsilon(T_i,P)$ denote the earliest time at which task T_i may begin executing on processor P — this is the earliest time $\hat{t} \geq \Re(T_i,P)$ such that processor P is free over $[\hat{t}, \hat{t} + X_i]$. The function *earliest_start()* chooses the processor P which has minimum of $\varepsilon(T_i,P)$ among the m processors.

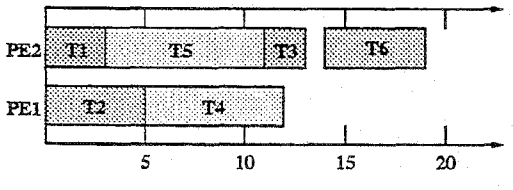

Figure 2. LSTF-generated schedule on 2 processors.

Fig 2 shows the schedule generated by LSTF when scheduling the task set shown in Fig 1 upon 2 processors, PE1 and PE2, with communication delay $\kappa(PE1,PE2) = \kappa(PE2,PE1) = 1$. Let us see how *earliest_start()* works for task 6 (T_6).

$$\begin{aligned}\Re(T_6,1) &= \max\{\gamma(T_3) + 1*2, \gamma(T_4) + 0*2, \\ &\qquad \gamma(T_5) + 1*3)\} \\ &= \max\{15,12,14\} = 15 \qquad (3)\end{aligned}$$

$$\begin{aligned}\Re(T_6,2) &= \max\{\gamma(T_3)+0*2, \gamma(T_4)+1*2, \\ &\qquad \gamma(T_5)+0*3)\} \\ &= \max\{13,14,11\} = 14 \qquad (4)\end{aligned}$$

From Equation 3 and 4, we derive that $\varepsilon(T_6,1) = 15$ and $\varepsilon(T_6,2) = 14$, so the function *earliest_start()* returns *PE2* to LSTF scheduler.

3 Simulation Study

We have implemented a work generator and a prototype symbolic simulator for a preliminary evaluation of enhanced LSTF's performance with respect to two other approaches, EDF-E and EDF-R. In either case, the next task to be executed is a ready task with the earliest deadline [7]. For EDF-E, the task is assigned to a processor according to ETF; for EDF-R, a processor is selected at random. Different applications have different performance criteria, so we compared LSTF with those approaches with respect to the number of tasks missing their deadline (Ψ) and to tardiness (τ). We study the impact of the number of task, number of processor and edge density on Ψ and τ respectively. Each of the points in the following simulation results represents the average data of 100 random graphs (with the generator initialized by time-dependent seeds).

The nodes in the task model are generated randomly and the edges are generated depending upon the specified ratio of edge to node. Each node is assigned a weight generated at random according to the non-negative binomial distribution and each task is assigned a deadline, defined by Equation 5.

$$D_i = \min((p_i + \frac{W-p_i}{m^{0.4}}), (W(\frac{1}{m})^{0.5})) \qquad (5)$$

where W is the total weight of all tasks, and
p_i is the weight of the longest path of T_i.

Figure 3 shows the relation between the number of processors and the gain ($\frac{\tau_x - \tau_{LSTF}}{\tau_{LSTF}} \times 100$), where τ_x is the maximum tardiness when scheduled by algorithm x. In this simulation, there are 440 tasks, each with a non-negative binomial weight, and 1105 edges, each with random integer weight between 1 and 5. The minimum, maximum and average deadlines of these tasks are 78, 505 and 275 respectively. The processors are fully connected to each other and $\kappa(\alpha,\beta) = 1$ for every pair of processors α and β. With increase in the number of processors, the gain increases rapidly in both cases. When the number of processors is over 15, the gain is not defined because the denominator, τ_{LSTF}, becomes 0 (all tasks meet their deadlines).

Figure 4 shows the relation between the number of processors and the gain ($\frac{\Psi_x - \Psi_{LSTF}}{\Psi_x} \times 100$) in terms

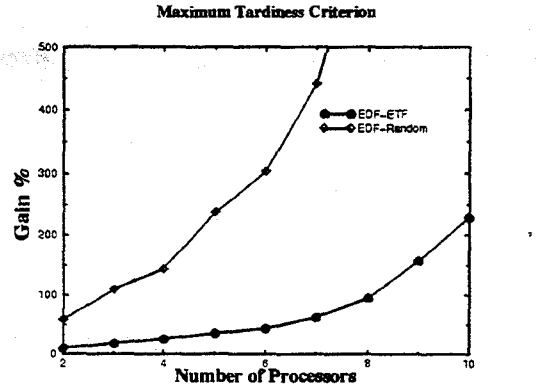

Figure 3. The relationship between the number of processors and the gain under performance criterion τ.

of Ψ_x, where Ψ_x is the number of tasks missing their deadlines when scheduled by algorithm x. This was simulated using the same set of tasks and hardware model as in Figure 3. With only a few processors, EDF-E is slightly better than LSTF, but as the number of processors increases, LSTF does increasingly better than EDF-E up to a saturation level, after which the gain evens out.

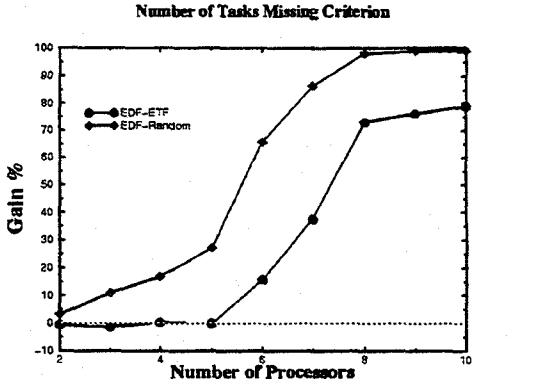

Figure 4. The relationship between the number of processors and the gain under performance criterion Ψ.

Figure 5 shows the relation between ratio of edge to node verses the gain ($\frac{\tau_x - \tau_{LSTF}}{\tau_{LSTF}} \times 100$), where τ_x is the maximum tardiness when scheduled by algorithm x. It was simulated using a task model similar to that used in Figure 3 with 361 tasks, and with minimum, maximum and average deadlines of 94, 412 and 250 respectively. The hardware model is a network of three fully connected processors with $\kappa(\alpha,\beta) = 1$ for every pair of processors $\alpha \neq \beta$. We can observe that the ratio of edge to node does not have much effect on the

gain relative to both algorithms, and therefore conclude that edge-density is not a major factor in performance at minimizing τ.

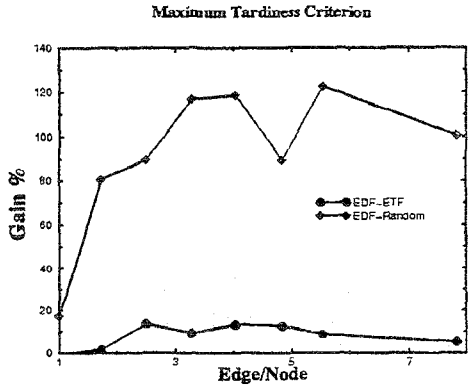

Figure 5. The relationship between the ratio of edge to node and the gain under performance criterion τ.

Table 1 shows the τ and Ψ values for varying numbers of tasks, keeping the ratio of edge to node constant at 2.49. The hardware model is the same as the model used in Figure 5. LSTF always does better than either EDF at minimizing τ, and better than EDF-R with respect to Ψ; however, we observed no observable pattern in the relationship between values of Ψ for EDF-E and LSTF.

		Task Number						
		141	197	246	284	414	571	818
LSTF	τ	51.6	91.9	114	261	219	711	485
	Ψ	32.1	68.4	78.0	219	199	516	441
EDF-E	τ	71.6	111	140	293	267	764	554
	Ψ	35.4	70.5	83.2	215	214	511	467
EDF-R	τ	185	277	356	503	617	1200	1182
	Ψ	96.0	143	182	245	327	533	654

Table 1. Performance table for varying task numbers on three fully connected processors.

4 Conclusions and Future Work

Integrated with ETF which effectively reduces communication delays, the LSTF scheduler outperforms both EDF-E and EDF-R in the sense of minimizing tardiness to schedule precedence- and timing-constrained task graphs on multiple-processors. LSTF also works better to minimize the number of tasks which miss their deadline than EDF-R.

We have a running Resource Allocation component for the DESTINATION prototype [2] based on logical and resource model. The logical model describes the functional and behavioral views. The resource model includes the description hardware, software, constraints, and human resources that represents a particular form of the system under consideration. When the software and hardware description are taken as input, DESTINATION will decide the location (assignment) and the time (scheduling) of execution. We are adding LSTF to this list of resource allocation algorithms.

Acknowledgments

This work is supported in part by the U.S. ONR Grants N00014-92-J-1367 and N00014-93-1-1047, the U.S. NSWC Grants N60921-93-M-1912, N60921-93-M-3095, N60921-94-M-1426 and N60921-94- M-1250 and the AT&T UEDP Grant 91-134. The authors thank the members of the Real-Time Computing Laboratory at New Jersey Institute of Technology for many helpful suggestions and comments.

References

[1] M. Al-Mouhamed and A. Al-Maasarani. Performance evaluation of scheduling precedence-constrained computations on message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1317–1322, 1994.

[2] C. C. Amaro and et. al. Economics of resource allocation. In *1994 Complex Systems Engineering and Assessment Technology Workshop*, 1994.

[3] B.-C. Cheng, A. D. Stoyenko, T. J. Marlowe, and S. Baruah. Lstf: A new scheduling policy for complex real-time tasks in multiple processor systems. *submitted to Automatica.*

[4] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company., 1979.

[5] J. Hoogeveen, J. Lenstra, and B. Veltman. Three, four, five, six, or the complexity of scheduling with communication delays. *Operations Research Letters*, 16(3):129–137, 1994.

[6] J.-J. Hwang and et. al. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. of Comput.*, 18(2):244–257, 1989.

[7] C. L. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *J. ACM*, 20(1):46–61, 1973.

[8] P.-Y. Ma, E. Lee, and M. Tsuchiya. A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, 31(1):41–47, 1982.

[9] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):412–420, 1995.

[10] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.