Highly Constrained Task Scheduling on Multiprocessing Systems

Abdelmageed Elsadek

The Cabinet Information and Decision Support Center

Mostafa A. Azim

Arab Academy for Science and Technology and Maritime Transpor

Abstract

The problem of non-preemptively scheduling a set of m tasks on n processors with communication overhead subject to precedence, memory and deadline constraints is considered. A new heuristic with the time complexity of $0(m^2n)$, Augmented Least Space-Time First (LSTF), is proposed to minimize the maximum tardiness. The efficiency of the augmented LSTF using a large number of randomly-generated graphs and three real-world structures is compared with that of the augmented Earliest Deadline First- Earliest Task First (EDF-E) that schedules each ready task on the processor at which it can be scheduled at the earliest time and with that of EDF-R that select the processor at random. The result of the comparisons, which are based on the maximum tardiness and the number of tasks that miss their deadlines, indicates that the augmented LSTF outperforms both EDF-E and EDF-R.

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, 2, 3, 4]. In this paper, we consider the problem of scheduling and allocating the tasks of a precedence, memory and timing-constrained task graph with communication delays onto a set of processors in a way that minimizes maximum tardiness τ, defined as $\tau = \max^{m}_{i=1} \{\max\{0, (\gamma_i - D_i)\}\}$ 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.

When all task deadlines are 0, the problem of minimizing tardiness reduces to the problem of minimizing the make-span, which is known to be a NP-complete [5]. Therefore, the general problem of scheduling to minimize tardiness is NP-hard [6].

Now, we specify the task and hardware model formally. Suppose that we have a number of n processors and a set T = { T_1, T_2 ……., T_m } of m non-preemptive tasks with each task Ti being characterized by an ordered pair $(D_i - X_i)$ where X_i is the CPU execution requirement and D_i is the deadline of task Ti. There exist precedence constraints "→" on T. Each edge, T_i,→ T_j, ands for a precedence relationship between predecessor T_i and successor T_j. Task T_i is defined to be ready when all its predecessor have completed execution. A non-negative integer λ_{ij}, 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) [7] and represented as G = G(T, →, X, D, λ). Fig. 1-a shows an example of an EDAG. In addition, a task assignment vector A is defined to be A:T→P, where A(i) =j if task Ti is assigned to processor p_j, 1≤ i ≤ m, 1≤ j ≤ n. For a certain assignment, the Task Set (TS_j) of a processor j can now be defined as the set of tasks allocated to that processor [8, 9]:

$$TS_j = \{i \mid A(i)=j\} \qquad j = 1, ...,n$$

Also, a vector B is defined to be a vector whose elements b_i denote the amount of memory needed to process task T_i and Q is a vector whose elements q_j represent the maximum capacity of local memory associated with processor p_j. In cases where memory is constrained, the following inequality must hold at each processing node of the system [10, 11 ,12]

$$\sum_{\substack{1 \le i \le m \\ i \in TS_j}} b_i \le q_j \qquad (1)$$

Let κ(α, β) donate the time to transfer a data unit from processor α to β. We assumed that communication is contention-free between the processors, so κ(α, β) is constant and equal to κ; and that communication within a processor incurs no delays; i.e., when α = β, κ(α, β) = 0. The communication delay between T_i and T_j, C_{ij}, is then equal κ* λ_{ij} time units if T_i and T_j are assigned to different processors.

2. Augmented LSTF Heuristic

The Earliest Task First (ETF) heuristic [7] schedules each ready task on the processor at which it can be scheduled at the earliest time. Without deadline constraints, the make-span (ω_{ETF}) generated by this heuristic satisfies

$$\omega_{ETF} \le (2 - 1/n)\, \omega_{iopt} + \eta$$

where ω_{iopt} 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 [13].

To enhance the basic LSTF scheduler to handle interprocessor communication delays, we have integrated LSTF with ETF and we added the memory constraints as well as the precedence and deadline constrained. 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 blew:

Phase one:

(a) Compute modified deadlines in the standard way, using precedence relationships and CPU execution requirements, by Equation 2. Figure 1-b shows the result of this step on the task graph of Figure 1-a.

$$D_i = \min_j (D_i, D_j - X_j) \quad (2)$$

Where $_j$ ranges over all of i's successors.

(b) Assign static space-time ($D_i - X_i$) to each task, where D_i s the new modified deadline of task i. 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.

(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).

Phase two:

execute the worklist algorithm shown below. It should be noted that, the function earliest_start() returns the identifier of the processor on which a task would be able to begin execution at the earliest time.

(1) While $R \neq \emptyset$

(2) begin

(3) Let T_i be the task at the head of R

(4) P_j = earliest_start(T_i) such that equation 1 holds

(5) assign T_i to run on P_j i.e $T_i \in TS_j$

(6) delete T_i from R

(7) insert all ready children of T_i into R

(8) End

Augmented LSTF Scheduler algorithm

It takes 0 (m) time to create the priority queue R, and each insert and delete operation can be performed in 0 (log m) time. Hence, the bottleneck of LSTF

schedule is in Line (4), whose time complexity is 0 (m^2n) [6] per instance. Since Line (4) is executed n times, the complexity of LSTF is 0(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. R (T_i, P) denotes the earliest time at which task T_i on processor P will have received all messages from all its predecessors:

$$R(T_i, P) = \max_{Tj} \{\gamma(T_j) + C_{ji}\} \quad (3)$$

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 being executing on processor P – this is the earliest time $t' \geq R(T_i, P)$ such that processor P is free over $[t', t' + X_i]$. The function earliest_start() chooses the processor P which has minimum of $\varepsilon(T_i, P)$ among the n processors.

Figure 2 shows the schedule generated by LSTF when scheduling the task set shown in Figure 1 upon 2 processors, P_1 and P_2, with communication delay $\kappa(P_1, P_2) = \kappa(P_2, P_1) = 1$. Let us see how earliest_start() works for T_6.

$$R(T_6, P_1) = \max \{\gamma(T_3) + 1 * 2, \gamma(T_4) + 0 * 2, \gamma(T_5) + 1 * 3)\}$$
$$= \max \{15, 12, 14\} = 15 \quad (4)$$
$$R(T_6, P_2) = \max \{\gamma(T_3) + 0 * 2, \gamma(T_4) + 1 * 2, \gamma(T_5) + 0 * 3)\}$$
$$= \max \{13, 14, 11\} = 14 \quad (5)$$

From Equations 4 and 5, we derive that $\varepsilon(T_6, P_1) = 15$ and $\varepsilon(T_6, P_2) = 14$, so the function earliest_start() returns P_2 to LSTF scheduler.

3. Empirical analysis for random systems

For a preliminary evaluation of the augmented LSTF's performance, we compared its allocation results with respect to two other approaches allocations, EDF-E and EDF-R after augmenting them to take into account the memory constraints. In either case, the next task to be executed is a ready task with the earliest deadline [14,15]. For EDF-E, the task is assigned to a processor according to ETF; for EDF-R, a processor is selected at random. The comparison is based on two parameters, the number of tasks missing their deadline (Ψ) and tardiness (τ).

For empirical evaluation, nine categories of randomly generated graphs were created in a manner to represent programs of varying structure and sparsity. The method of generating random task graphs used in this research is based upon the probabilistic construction of a module graph's boolean adjacency matrix. The adjacency matrix for such graphs is a ($m \times m$) matrix whose elements, a_{ij} (where $1 \leq i < m$ and $1 \leq j < m$) are defined to be: $a_{ij}=1$ if there is a data/control dependency directed from task T_i to task T_j, and $a_{ij}=0$ if no dependency exists between task T_i and task T_j.

To generate random structures, the adjacency matrix is first constructed with all its diagonal elements being set to zero (a_{ij}=0 for all i=j). Then each of the remaining elements of the matrix are determined individually as part of a Bernoulli process with the parameter, ρ, representing the probability of a "success". For each ordered pair, (i,j), where i<j when the Bernoulli trial is a success, then a_{ij}=1 and a_{ji}=1, in case of a failure a_{ij}=0 and a_{ji}=0. The parameter ρ can also be considered to be the sparsity of the task graph where the sparsity is defined to be the expected portion of the possible m(m-1)/2 data/control dependencies represented by the edges of the task graph. If the sparsity=1 then the random graph generation routine creates fully connected program graphs, and if it is set equal to zero it creates an embarrassingly parallel one, values of the sparsity that lie between these two extremes generally produce program graphs that possess intermediate structures.

In the empirical analysis which follows, nine categories of 100 node random graphs (with each category representing a distinct graph sparsity, ρ , which ranges from 0.01 to 0.09 in increments of 0.01) were constructed with 50 program graphs being generated for each category. The random graphs were created with module/task execution times, X_i, being picked randomly (from a uniform distribution) within the interval from 1 to 500, and the λ_{ij} data volume between T_i and T_j being chosen in a similar manner in the range from 1 to 20. Each task is assigned a deadline, defined by:

$$D_i = \min((L_i + ((W-L_i) / n^{0.4}), (W(^1/_n)^{0.5}))) \quad (6)$$

Where W is the total weight of all tasks, and

L_i is the weight of the longest path of T_i.

Table 1 illustrates the non-constrained case where all processors are assumed to be identical and there are sufficient processor resources (memory) available to allow any number of tasks to be assigned to any given processing element. Table 2 represents the case where all processors memory resource limits, q_j ($1 \le j \le n$), are equal to ceil(E[($\sum_{1 \le i \le m} b_i$)/n]) = ceil(1050/16) = 66, where b_i is the amount of memory needed by module/task T_i and it is randomly generated using the uniform distribution in the range from 1 to 20. Table 3 shows the case where resource constraints are applied in a nonuniform manner in which each processor is assigned a unique resource limit q_j defined in this case to be $q_j = |\chi_j - 0.5| k + C$, where $0 \le \chi_j \le 1$ is a uniformly generated random number, k = ceil(E[($\sum_{1 \le i \le m} b_i$)/n]) (which is 66 in this case) and C is chosen to be the smallest integer constant that allows feasible allocations (which was found to be 10 in this application).

From these tables, it can be observed that the proposed augmented LSTF algorithm produce allocations which are consistently better than the two other algorithms.

4. Analysis for real-world structures

The ultimate usefulness of any task allocation heuristic is how well it performs when applied to a representable set of real-world problems. With this in mind, the aforementioned heuristics were also applied to three large-scale real-world structures.

The first structure which was considered is based upon a simulation of NASA's proposed National Launch System (NLS) [16]. The NLS simulation is a real-time guidance and control type simulation of the U.S. heavy launch vehicle which was proposed in the early 1990's. It is made up of 586 communicating modules which can be represented by a program graph which has a sparsity of 0.0031.

The second structure is a simulation of a Space Shuttle Main Rocket Engine (SSME). The SSME simulation [17] is a complex continuous simulation that produces the flow rates, pressures, thrusts, and temperatures of ten subsystem components associated with SSME operation. The simulation is designed to operate in real-time and respond to appropriate commands issued by the shuttle's main computer system. It is composed of 131 communicating modules with a module graph sparsity of 0.0195.

The third structure is a simulation of a 6-degree-of-freedom robot system [18]. This system consists of rigid bodies connected by ideal revolute joints. Every joint is driven by a torque, produced by the electro-magnetic field of a current-controlled DC-motor and transformed by gear-boxes. The motors are controlled by decentralized cascade controllers. This simulation is composed of 677 communicating modules and has a sparsity of 0.0071.

Table 4 reflects the results of applying the proposed augmented LSTF algorithm and EDF-E and EDF-R algorithms to the three real-world structures using the task and communication timings which were derived from actual execution time profiles made on a SGS Thompson Transputer system. The superiority of the augmented LSTF algorithm is evident.

5. Conclusions and Future Work

The efficiency of the augmented LSTF using a large number of randomly-generated graphs and three real-world structures is compared with that of the augmented EDF-E and EDF-R. The result of the comparisons, based on the maximum tardiness and the number of tasks that miss their deadlines, indicate that the augmented LSTF outperforms both EDF-E and EDF-R.

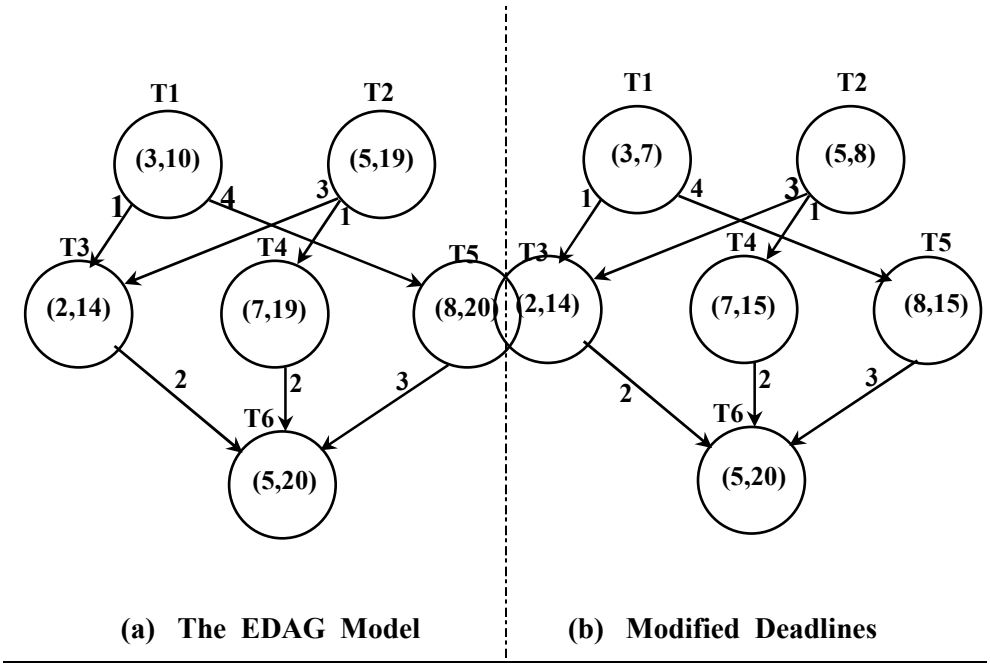

Figure 1. An example Task System.

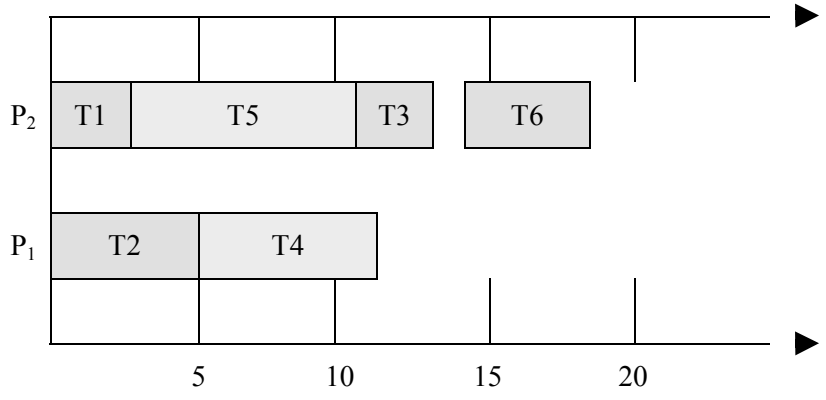

Figure 2. LSTF generated schedule on 2 processors.

Table 1: measured performance on a 16 processor non-constrained system

p (sparse)	Augmented LSTF		Augmented EDF-E		Augmented EDF-R	
	τ	Ψ	τ	Ψ	τ	Ψ
0.01	26	3	42	4	76	6
0.02	31	3	53	5	89	7
0.03	39	4	63	5	113	8
0.04	54	4	79	6	126	9
0.05	71	5	97	6	147	9
0.06	94	5	133	6	164	10
0.07	127	6	163	7	192	11
0.08	154	6	182	7	223	12
0.09	178	7	211	9	289	13

Table 2: measured performance on a highly constrained uniform16 processor system

p (sparsity)	Augmented LSTF		Augmented EDF-E		Augmented EDF-R	
	τ	Ψ	τ	Ψ	τ	Ψ
0.01	30	4	45	4	79	5
0.02	36	5	60	7	97	9
0.03	43	5	70	8	118	11
0.04	61	5	88	8	134	12
0.05	80	7	113	9	156	14
0.06	101	7	142	11	181	17
0.07	138	8	176	11	213	17
0.08	173	8	203	13	254	19
0.09	194	9	247	15	313	21

Table 3: measured performance on a highly constrained non-uniform 16 processor system

p (sparsity)	Augmented LSTF		Augmented EDF-E		Augmented EDF-R	
	τ	Ψ	τ	Ψ	τ	Ψ
0.01	30	4	45	4	79	5
0.02	36	5	60	7	97	9
0.03	43	5	70	8	118	11
0.04	61	5	88	8	134	12
0.05	80	7	113	9	156	14
0.06	101	7	142	11	181	17
0.07	138	8	176	11	213	17
0.08	173	8	203	13	254	19
0.09	194	9	247	15	313	21

Table 4: measured 16 processor speedup for original real-world applications

Application	Augmented LSTF		Augmented EDF-E		Augmented EDF-R	
	τ	Ψ	τ	Ψ	τ	Ψ
NLS	38	14	54	35	89	82
SSME	21	8	53	19	67	31
6-D Robot	41	18	63	41	103	84

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] 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.

[3] K. Ramamritham. “Allocation and scheduling of precedence-related periodic tasks”. IEEE Transactions on Parallel and Distributed Systems, 5(4):412-420, 1995.

[4] T. Yang and A. Gerasoulis. “Scheduling parallel tasks on an unbounded number of processors”. IEEE Transactions on Parallel and Distributed Systems, 5(9):951-967, 1994.

[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] M. Garey and D. Johnson. “Computers and Intractability: A Guide to the Theory of NP-Completeness”. W. H. Freeman and Company., 1999.

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

[8] Farzad Ghannadian, Cecil O. Alford, and Ron Shonkwiler, “Application of the genetic algorithm to multiprocessor scheduling”, Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 95).

[9] Gylys, V. B., and Edwards, J. A., "Optimal partitioning of work load for distributed system", Procs. Compcon Fall 76, pp. 353-357.

[10] Abdelmageed Elsadek Abdelrazek “Task Allocation and Reallocation for Fault Tolerance in Heterogeneous Distributed Computing Systems”, Ain Shams University, Faculty of Engineering, Vol. 37, No. 2, June 30, 2002, Egypt.

[11] Abdelmageed Elsadek, Gamal I. Selim, and B Earl Wells,“Task Allocation and Reallocation for Fault Tolerance in Distributed Computing Systems” Ain Shams University, Faculty of Engineering, Vol. 36, No. 4, Dec 31, 2001, , Egypt.

[12] Abdelmageed Elsadek, B. Earl Wells, “Heuristic Model for Task Allocation in a Heterogeneous Distributed Computing Systems,” the International Journal of Computers and their Applications (IJCA), Vol. 6, No. 1, pp. 1-13, March, 1999, USA.

[13] 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. SIAM J. of Comput., 20(2), 1989.

[14] C. L. Liu and J. Layland. “Scheduling algorithm for multiprogramming in a hard real-time environment”. J. ACM, 20(1):46-61, 1993.

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

[16] John M. Hanson, M. Wade Shrader, H. P.Chang, and S. E. Freeman, “Guidance and dispersion studies of National Launch System ascent trajectores”, AIAA paper 92-4306, Proceedings of the 1992 AIAA Guidance and Control Conference.

[17] B. Earl Wells, Kenneth G. Ricks, and John M. Weir “Parallel simulation of a large scale aerospace system in a multicomputer environment” to appear in IEEE Transactions on Aerospace and Electronic systems, April 1997.

[18] M. Otter, H. Elmqvist, and F. E. Cellier, “ Modeling of multibody systems with the object-oriented modeling language Dymola”, Proc. NATO/ASI, Computer-Aided analysisof rigid and flexible mechanical systems, troia, Portugal, June 27-July 9, 1993. Also in Nonlinear Dynamics, 9:91-112, 1996, Kluwer Academic publis.