Pareto-based Soft Real-Time Task Scheduling in Multiprocessor Systems

Jaewon Oh, Hyokyung Bahn, Chisu Wu, and Kern Koh
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, South Korea
jwoh@selab.snu.ac.kr
hyokyung@summer.snu.ac.kr
wuchisu@selab.snu.ac.kr
kernkoh@june.snu.ac.kr

Abstract

We develop a new method to map (i.e. allocate and schedule) real-time applications into certain multiprocessor systems. Its objectives are 1) the minimization of the number of processors used and 2) the minimization of the deadline missing time. Given a parallel program with real time constraints and a multiprocessor system, our method finds schedules of the program in the system which satisfy all the real time constraints with minimum number of processors. The minimization is carried out through a Pareto-based Genetic Algorithm which independently considers the both goals, because they are non-commensurable criteria. Experimental results show that our scheduling algorithm achieved better performance than previous ones. The advantage of our method is that the algorithm produces not a single solution but a family of solutions known as the Pareto-optimal set, out of which designers can select optimal solutions appropriate for their environmental conditions.

1. Introduction

The availability of inexpensive, high-performance processors and memory chips has made it attractive to use multiprocessor systems for real-time applications. However, the programming of such multiprocessor systems presents a rather formidable problem. In particular, time-critical tasks must be serviced within certain preassigned deadlines dictated by the physical environment in which the multiprocessor system operates [1]. Thus, efficient methods and tools for allocation and scheduling of tasks are needed.

In general, a task can be characterized by an integer triple (S, C, D), where S is the start-time of the task, C is the execution time of the task, and D is the deadline. A precedence relation may be defined on a set of tasks. Task preemption may or may not be allowed.

A multitude of methods for allocation and scheduling of these task sets have been proposed. For the uniprocessor case, Liu and Layland obtained a necessary and sufficient condition for scheduling periodic task sets (with preemption allowed) [2]. Henn generalized the result for a single processor case to cover precedence constraints [3]. Garey and Johnson discovered optimal scheduling algorithm when there are only two processors [4]. Unluckily, the scheduling problem often becomes NP-hard whenever more than two processors are involved [1].

Task scheduling is simply the choice of a mapping of a set of tasks to a set of processors to achieve the pre-defined objectives such as the minimization of the deadline missing time and the minimization of the communication cost. Whilst this problem requires the simultaneous optimization of multiple, often competing, criteria, the solutions to the problem are usually computed by combining them into a single criterion to be optimized, according to some combining function [5, 6]. The weakness of optimizing a combination of the objectives is that if the optimal solution cannot be accepted either because the combining function used excluded aspects of the problem which were unknown prior to optimization or because we chose an inappropriate setting of the coefficients of the combining function, additional runs may be required until a suitable solution is found [7]. Therefore, we treat the problem of determining an optimal scheduling as a multiobjective problem with non-commensurable objectives. We use the Pareto-based optimization, which independently considers multiple objectives, and find a set of solutions that satisfy all of the given objectives. Moreover, when a new objective is introduced, our method can include it without interfering with existing objectives.

The success of any GA-based approach depends heavily on how well the various components of a GA incorporate the salient features of the problem under consideration. In this paper, we present GA components that are well suited to the problem of task scheduling with precedence and deadline constraints, including the problem encoding strategies, the GA operators with various heuristics, and parameter tunings.

This paper is organized as follows. In Section 2, we define the graph models for allocation and scheduling of real-time applications. Section 3 shows our scheduling algorithm. In

This work was supported by the Brain Korea 21 Project.

Section 4, we describe the results and analysis of the experiment on applications presented in [5]. Finally, we conclude in Section 5.

2. Scheduling Model

We assume that a real-time application program has been partitioned into tasks that describe subcomputations of the application. Each task may be composed of one or more subtasks.

2.1. Subtask Graph (STG)

STG is an acyclic, directed, and weighted graph, that represents the relationships among subtasks in a task. Each subtask in a task is denoted by a node in the graph and the precedence condition between two subtasks is denoted by a directed edge. An edge is directed from the sender toward the receiver. STG is denoted by:

$$STG = (ST, STR, STW, STCW, STRT, STD)$$

- ST (Subtasks) = { $t_i s_j$ }, where $t_i s_j$ indicates subtask j of task i.
- STR (Subtask Relationships) = { str_{ijkl} }, where $str_{ijkl} = 1$ if $t_i s_j$ sends output to $t_k s_l$. Otherwise $str_{ijkl} = 0$. It is assumed that if subtask s_1 sends output to s_2, this is done at the end of the s_1 execution, and must arrive before s_2 can begin executing.
- STW (Subtask Workloads) = { stw_{ij} }, where stw_{ij} indicates the size of $t_i s_j$ in terms of the number of instructions executed.
- $STCW$ (Subtask Communication Workloads) = { $stcw_{ijkl}$ }, where $stcw_{ijkl}$ indicates the volume of communication from $t_i s_j$ to $t_k s_l$.
- $STRT$ (Subtask Release Times) = { $strt_{ij}$ }, where $t_i s_j$ must start its execution after $strt_{ij}$.

The macro data flow model described in [8] will be adopted: a subtask is ready to be executed as soon as all its input data are available and it cannot be interrupted once started. However, if all input data of $t_i s_j$ are available before $strt_{ij}$, $t_i s_j$ must wait until $strt_{ij}$.

- STD (Subtask deadlines) = { std_{ij} }, where $t_i s_j$ must finish its execution within std_{ij}.

2.2. Task Graph (TG)

We assume there is no communication between tasks. TG is an acyclic, directed, and weighted graph. Its node set $V(TG)$ is a set of all subtasks in an application and its edge set $E(TG)$ is a set of all precedence relations between subtasks. This graph has the following components that are constructed from STG:

$$TG = (T, STGS, TRT, TD)$$

- T (Tasks) = { t_i }, where t_i indicates task i.

Let STG_i be the subtask graph of task i and be (ST_i, STR_i, STW_i, $STCW_i$, $STRT_i$, STD_i). Then, the union of STG_i and STG_j is defined as $STG_{i \cup j} = (ST_i \cup ST_j, STR_i \cup STR_j, STW_i \cup STW_j, STCW_i \cup STCW_j, STRT_i \cup STRT_j, STD_i \cup STD_j)$.

- $STGS$ (Subtask Graph Sets) = $\cup_i STG_i$ for every task i.
- TRT (Task Release Times) = { trt_i }, where every subtask $t_i s_j$ in task t_i must start its execution after trt_i.
- TD (Task Deadlines) = { td_i }, where every subtask in task t_i must finish its execution within td_i.

2.3. System Graph (SG)

SG is an undirected and weighted graph. Each processor in a parallel system is denoted by a node and the network connection is denoted by an undirected edge. SG has the following components:

$$SG = (P, PR, PC, NC)$$

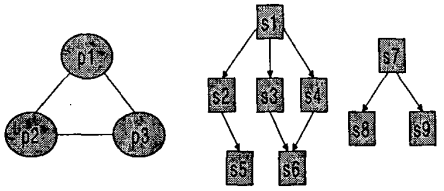
- P (Processor) = { p_i }, where p_i indicates processor i in the system.
- PR (Processor Relationship) = { pr_{ij} }, where $pr_{ij} = 1$ if processor i is connected by network with processor j. Otherwise $pr_{ij} = 0$.
- PC (Processor Capability) = { pc_i }, where pc_i indicates the (average) time of execution for one instruction on processor i.
- NC (Network Capability) = { nc_{ij} }, where nc_{ij} indicates the average time needed to transfer a unit of information from processor i to processor j.

3. Mapping Algorithm

3.1. Solution Structure and Fitness Function

In the scheduling problem, a chromosome represents one of all the possible mappings of all the subtasks into the processors. A possible solution in our method is illustrated in Figure 1.

The system graph and the task graph are shown in Figure 1-(a) and a possible allocation and scheduling is in Figure 1-(b). The system graph is a complete graph on three processors and the task graph is composed of two tasks, which have no precedence condition and are composed of six subtasks and three subtasks, respectively. A chromosome is partitioned into two parts, which represent the allocation and scheduling information of subtasks, respectively, as shown in Figure 1-(b). Chromosome A denotes the scheduling information and chromosome B the allocation information. The length of each chromosome is the total number of subtasks in a real-time application. The sequence of subtasks in chromosome A represents the scheduling order. Because there may be precedence relations among subtasks, some sequence of scheduling may not be a feasible solution. Note that to be a feasible schedule, chromosome A should be a possible topological order for the given task graph. Chromosome B determines the allocation of subtasks. In chromosome B, i-th gene (i.e. i-th character in chromosome B) denotes the processor number to which a subtask $A[i]$ is allocated. In Figure 1-(b), the second gene of chromosome B, $B[2]$ means that $s3$ ($A[2]$) is allocated to $p1$. For example, in Figure 1, $s1$, $s3$, $s5$, and $s8$, which are allocated into the processor $p1$, are scheduled to begin execution first, second, third, and last, respectively.



System Graph Task Graph

(a) Sample Task Graph and System Graph

S1	S3	S4	S2	S5	S6	S7	S9	S8

Chromosome A (Scheduling)

P1	P1	P3	P3	P1	P3	P3	P3	P1

Chromosome B (Allocation)

(b) A Feasible Solution

Figure 1. Chromosome Encoding

In a GA, we need a cost function for measuring the quality of each solution. Our cost function is a vector-valued function F that consists of an objective function component f corresponding to each objective. They are defined as follows:

$F = (f_1, f_2)$

$f_k : S \rightarrow R^+$, where $k=1,2$ and S is a set of solutions.

- f_1 (number of processors used): This objective function component denotes the total number of processors, which are used to execute subtasks in a real-time application.
- f_2 (deadline missing time): This component denotes the sum of the deadline missing times of all subtasks with the deadline requirements. It is calculated by the following formula:

$$f_2 = \sum Pos\ (stt_i - std_i)$$

$$Pos\ (x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

where stt_i is the time at which subtask i ends its execution.

Assuming a minimization problem, the following definitions apply [9]:

Definition 1. (Inferiority) Vector $\boldsymbol{u} = (u_1, \ldots, u_n)$ is said to be *inferior* to $\boldsymbol{v} = (v_1, \ldots, v_n)$ iff $\boldsymbol{v}$ is partially less than $\boldsymbol{u}$ ($\boldsymbol{v}\ p< \boldsymbol{u}$), i.e., $\forall\ i = 1, \ldots, n$, $v_i \le u_i$ and $\exists\ i = 1, \ldots, n$, $v_i < u_i$

Definition 2 (Superiority) Vector $\boldsymbol{u} = (u_1, \ldots, u_n)$ is said to be *superior* to $\boldsymbol{v} = (v_1, \ldots, v_n)$ iff $\boldsymbol{v}$ is inferior to $\boldsymbol{u}$.

Definition 3 (Domination) Let $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ be the vector values of a cost function over the solutions S_i and S_j, respectively. S_i is said to *dominate* S_j iff $\boldsymbol{v}_i$ is superior to $\boldsymbol{v}_j$.

In this paper, instead of determining the individual's fitness by combining objectives (i.e., f_1 and f_2) into a single criterion to be optimized, we assign the fitness to the individual using a scheme proposed in [9]. In the scheme, the rank of a certain chromosome corresponds to the number of chromosomes in the current population by which it is dominated. Consider, for example, an individual x_i at generation t, which is dominated by $p_i^{(t)}$ individuals in the current generation. Its current position in the individuals' rank can be given by:

$$\text{Rank}(x_i, t) = 1 + p_i^{(t)}$$

The individual' fitness is its rank in our method.

3.2. GA Operators

Selection Operator

We apply the most common practice, which gives the best chromosome in the population four times more probability to be selected than the worst chromosome. The normalized fitness function is as follows:

$$NF_i = (R_w - R_i) + (R_w - R_b)/3,$$

where R_w, R_b, and R_i are the ranks of the worst chromosome, the best chromosome, and the chromosome i, respectively, in the population.

Based on the normalized fitness function, we apply a roulette wheel selection as a selection operator [10].

Crossover Operator

We define a one-point crossover with adjustment because when the classical one-point crossover is applied to chromosome A, it may produce infeasible solutions; the sequence of subtasks in chromosome A of a new solution may not be the scheduling order. This problem is resolved by the following procedure:

Procedure 1. one-point crossover with adjustment.

1) Let $P_A1 = (P_A1[1], \ldots, P_A1[n])$ and $P_B1 = (P_B1[1], \ldots, P_B1[n])$ be chromosomes A and B of parent $P1$, and P_A2 and P_B2 be chromosomes A and B of parent $P2$, respectively. Then, generate a crossover point k randomly, $1 \le k < n$.
2) The left segments of C_A1 and C_B1 of the child $C1$ are $C_A1 := P_A1[1], \ldots, P_A1[k]$ and $C_B1 := P_B1[1], \ldots, P_B1[k]$
3) Scan the parent string P_A2 from left to right for $P_A2[i]$ such that $P_A2[i] \neq C_A1[j]$ for all j, $1 \le j \le k$.
4) $C_A1[k+1] := P_A2[i]$ and $C_B1[k+1] := P_B2[i]$. Then, increment k by 1.
5) Iterate 3 ~ 4 steps until the production of the child $C1$ is completed.

Applying this procedure to chromosome A of two parents will produce a child solution representing a possibly new topological order for the given task graph. We prove it in [10].

While the child has the same left segment as the parent $P1$ has, the child may have the right segment that is produced from the parent $P2$ but is adjusted to satisfy the precedence requirement. To give the child a chance to have the same right segment as parent $P1$ has, the new crossover procedure that is modified by swapping the left and the right in Procedure 1, is introduced. The two operators are applied alternatively. An example of a one-point crossover with adjustment is given in Figure 2, when the system graph and task graph in Figure 1-(a) are used.

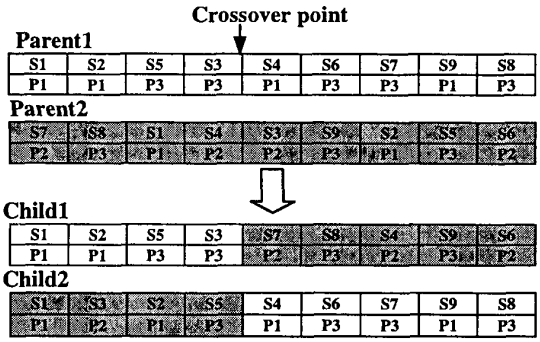


Figure 2. 1-point crossover with adjustment

Mutation Operator

Our mutation operator works by randomly selecting a gene of chromosome *B* and then replacing the value of this gene with a random number chosen from among processor numbers used by chromosome *B*.

Replacement Operator

At first if both parents dominate a new solution, the most inferior chromosome in the current population is replaced with the offspring. Otherwise, the operator checks if the offspring dominates either of two parents. If the offspring dominates one parent, the parent is replaced with the offspring. If the offspring dominates neither of two parents, the most inferior chromosome of the existing population is replaced with the offspring.

4. Experiments

4.1. Initial Population

The default population size N is 80. Each of the N initial solutions is generated by giving a randomly generated topological order for the given task graph to chromosome *A* and assigning a randomly chosen processor to each subtask.

4.2. Domination Redefinition

Applying domination defined in Section 3 caused premature convergence to a local optimum, i.e., f_1 is not minimized while f_2 is minimized and converges to 0. To resolve this problem, we redefine domination instead of applying the heuristic improvement operator.

First, a new objective function is introduced:

• f_3 (Load Imbalance): This objective function component measures how uniformly the workloads of the subtasks scheduled by one solution are distributed. The solution with a smaller value of f_3 indicates that the workloads are distributed more uniformly. It is calculated by the following formula, where N is the number of processors:

$$f_3 = \left\{ \sum_p \left(\sum_i stw_i \times pc_p \times x_{ip} - TL/N \right)^2 \right\} / N$$

$$TL = \sum_p \sum_i stw_i \times pc_p \times x_{ip}$$

where $x_{ip} = 1$ if subtask i is allocated to processor p. Otherwise, $x_{ip} = 0$.

This load imbalance component is used to reduce the number of processors used. When one solution S_i does not dominate the other S_j and S_j does not dominate S_i, the solution with a larger value of f_3 of the two solutions is given to larger fitness. The reason is the solution S with a larger value of f_3 indicates that its workloads may be more skewed on certain processors. So the genes from S may dominate the population, potentially causing convergence to solutions with the minimum number of processors used. The domination is redefined as follows.

Definition 4. (Domination) Let $v_i = (f_1(i), f_2(i), f_3(i))$ and $v_j = (f_1(j), f_2(j), f_3(j))$ be the vector values of a cost function over the solutions S_i and S_j, respectively. S_i is said to *dominate* S_j iff $(f_1(i), f_2(i))$ is superior to $(f_1(j), f_2(j))$ or ($(f_1(i), f_2(i)) = (f_1(j), f_2(j))$ and $f_3(i) > f_3(j)$).

4.3. Experimental Testbed

To validate our allocation and scheduling algorithm, we performed experiments on two real-time application programs presented in [5] and on one program into which the two programs are merged. In this paper, we show only the experiment on a program with one task and fifty subtasks on account of limited space. Refer to our technical report for detail [10].

To compare the performance of our method (RT-GA) with that of the method (RT-SA) proposed in [5], the following assumptions are taken:

1) All the processors are equal.
2) Communication time is negligible.
3) The release times of all tasks and subtasks are 0.

Example 1. The Sample Parallel Application with one task and fifty subtasks

The task graph for this application is given in Figure 3. The weight of a node denotes its workload. We want to execute this program in a parallel system whose system graph is a complete graph on nine processors with values of *pc* 1. This application must respect the following real-time constraints:

STD = { $std_{1\_11} = 30$, $std_{1\_13} = 45$, $std_{1\_26} = 75$, $std_{1\_28} = 65$, $std_{1\_45} = 145$, $std_{1\_50} = 170$ }

Allocation and scheduling produced by our algorithm is shown in Figure 4 through a Gantt chart. Both RT-GA and RT-SA produced schedules that satisfy all real-time constraints. But schedules given by RT-GA use 5 processors, while the schedule returned automatically by RT-SA uses 8 processors though two more processors could be made free by analyzing the schedule returned.

We plot the experimental results with the number of used processors on the *x*-axis and the deadline missing time on the *y*-axis in Figure 5. A logarithmic scale is used for the *y*-axis. It shows the distribution of solutions in the population over times (i.e. at populations 0, 1000, 2000, 3000, 6000, 14000, and 20000). In this population graph, the solution closest to

the origin is the near-optimal one because we assume the minimization problem. We can see that our algorithm yields new individuals that will lead to better solutions over time before population 14000. During the evolution, some subtasks scheduled by these individuals miss their deadlines even if the overall system has the capacity to meet the deadlines of all subtasks or these solutions use more processors than the optimal solution can use. After this time, solutions are converged to schedules that use five processors with the deadline missing time 0.

5. Conclusion

We addressed the problem of determining an optimal scheduling of a real-time application in certain multi processor systems such that the number of processors used and the total deadline missing time are minimized. We treat task scheduling as a multiobjective problem with non-commensurable objectives. Our approach differs from previous researches in that we seek to minimize the components of a vector-valued fitness function. In order to consider these objectives independently and search near-optimal solutions, we provide a Pareto-based GA. To improve on solutions, the existing domination relation is redefined and a heuristic crossover operator is newly defined. Experimental results show that our scheduling algorithm achieved better performance than the previous approach.

References

[1] M. L. Dertouzos and A. K. Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, Dec. 1989.

[2] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *JACM*, vol. 20, no. 1, Jan. 1973.

[3] R. Henn, "Antwortzeitgesteuerte prozessorzuteilung unter strengen zeitbedingungen," *Computing*, vol. 19, 1978, pp. 209-220.

[4] M. Garey and D. Johnson, "Two-processor scheduling with start-time and deadlines," *SIAM J. Comput.*, vol.6, 1977, pp. 416-426.

[5] M. Coli and P. Palazzari, "A New Method for Optimization of Allocation and Scheduling in Real Time Applications," *7th Euromicro Workshop on Real-Time Systems*, 1995.

[6] J. Aguilar, and E. Gelenbe, "Task Assignment and Transaction Clustering Heuristics for Distributed Systems," *Information Sciences*, Vol. 97, March 1997, pp. 199-219.

[7] C. A. Coello Coello, *An Empirical Study Of Evolutionary Techniques For Multiobjective Optimization In Engineering Design*, Ph. D. Thesis, Tulane University, 1996.

[8] V. Sarkar, *Partitioning and scheduling parallel programs for execution on multiprocessors*, Cambridge, MS, M.I.T. Press, 1989.

[9] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 416-423.

[10] Jaewon Oh, Hyokyung Bahn, Chisu Wu, and Kern Koh, "Pareto-based Soft Real-Time Task Scheduling in Multiprocessor Systems," School of Computer Science and Engineering, Seoul National University, 2000, http://drum.snu.ac.kr/~jwoh/TechReport/techRep00_9.ps.

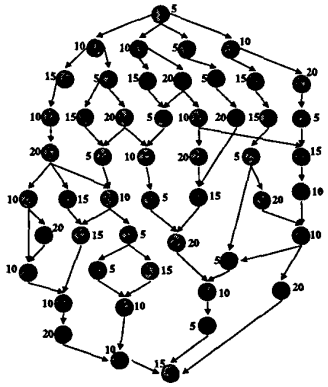


Figure 3. Task Graph for Example1

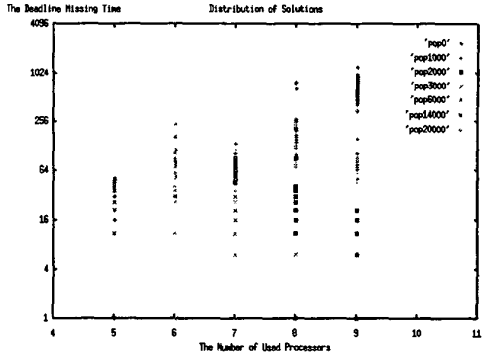


Figure 5. The Distribution of Solutions

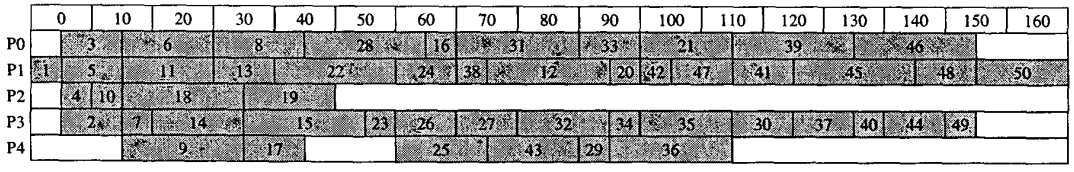


Figure 4. Gantt Charts for Example 1