병렬 처리장치의 개수에 따른 스케줄링 알고리즘의 제안 및 성능평가

Proposal and Performance Evaluation of A Scalable Scheduling Algorithm According to the Number of Parallel Processors

박 경 린* 이 상 준** 이 봉 규***
Gyung-Leen Park Sang Joon Lee BongKyu Lee

요 약

병렬처리 시스템에서의 스케줄링 문제는 응용 프로그램의 병렬 수행 시간을 최소화하는 스케줄을 찾아내는 문제로, 지난 수 십년 간 중요한 연구 과제였으며, 복제 기반 스케줄링 알고리즘들은 이 문제를 해결하기 위해 제안된 비교적 새로운 접근 방법이다. 복제 기반 스케줄링 알고리즘들은 작업들을 복제함으로써, 작업들간의 통신비용을 줄이는 방법으로 대부분의 복제 기반 스케줄링 알고리즘들은 무한한 수의 처리장치의 존재를 가정한다. 이러한 가정은 현실적이지 못하므로, 본 논문은 사용 가능한 처리장치의 수에 따라 다른 스케줄을 생성하는 스케줄링 알고리즘을 제안하고 그 성능을 평가한다. 성능 평가 결과는, 작업의 수가 N일 경우에 처리장치의 수가 N개 이상이면 무한한 처리장치의 수를 가정하였을 경우와 같은 스케줄을 생성하고, 사용 가능한 처리장치의 수가 N개 이하로 감소함에 따라 병렬 수행 시간이 서서히 증가함을 보인다.

Abstract

The scheduling problem in parallel processing systems has been a challenging research issue for decades. The problem is defined as finding an optimal schedule which minimizes the parallel execution time of an application on a target multiprocessor system. Duplication Based Scheduling (DBS) is a relatively new approach for solving the problem. The DBS algorithms are capable of reducing communication overhead by duplicating remote parent tasks on local processors. Most of DBS algorithms assume an availability of the unlimited number of processors in the system. Since the assumption may not hold in practice, the paper proposes a new scalable DBS algorithm for a target system with *limited* number of processors. It is shown that the proposed algorithm with N available processors generates the same schedule as that obtained by the algorithm with unlimited number of processors, where N is the number of input tasks. Also, the performance evaluation reveals that the proposed algorithm shows a graceful performance degradation as the number of available processors in the system is decreased.

1. Introduction

Efficient scheduling of parallel programs, represented as a *Directed Acyclic Graph (DAG)*, onto processing elements of parallel and distributed computer systems are extremely difficult and important issues [1-6]. The general goal of the scheduling process is to efficiently utilize resources to optimize a set of performance criteria (e.g., to minimize program parallel execution time). Since it has been shown that the multiprocessor scheduling problem is NP-complete, many researchers have proposed scheduling algorithms based on heuristics. The scheduling algorithms can be classified into two general categories: algorithms that employ task duplication and algorithms that do not employ task duplication.

List scheduling is a main technique used in most of non-duplication scheduling algorithms [7]. The list

* 종신회원 : 제주대학교 전산통계학과 교수
** 종신회원 : 제주대학교 통신컴퓨터 공학부 교수
*** 정 회 원 : 제주대학교 전산통계학과 교수

scheduling method assigns a priority to each task node in the input DAG. A list of tasks are constructed in decreasing order of the priority. Then a ready task with the highest priority is taken from the queue and assigned to a suitable Processing Element (PE) for assignment. A ready task is one that all of its parent tasks are already scheduled. Typically, a suitable PE is one that can execute the task the earliest. The scheduling algorithms in this class differ in the way of assigning the priority to the tasks in the input DAG.

Duplication Based Scheduling (DBS) is a relatively new approach to the scheduling problem [4, 8-14]. The DBS algorithms are capable of reducing communication overhead by duplicating remote parent tasks on local processing elements. Similar to non-duplication algorithms, DBS methods have been shown to be NP-complete [15]. Thus, many of the proposed DBS algorithms are based on heuristics. In general, most of the DBS algorithms assume an availability of an unlimited n umber of processors; with few exceptions as in [16, 17]. Since the assumption may not hold in practice, we propose a scalable DBS algorithm which schedules task nodes in an input DAG to a limited number of processors in a target system. The proposed algorithm with N available processors generates the same schedule as that obtained by the algorithm with unbounded number of processors, where N is the number of task nodes in the input DAG. Our simulation study shows the graceful performance degradation of the proposed algorithm as the number of available processors is decreased.

The remainder of this paper is organized as follows. Section 2 presents the system model and the problem definition. The proposed algorithm is presented in Section 3 along with the worst case and optimality conditions. Section 4 presents our simulation study. Section 5 concludes this paper.

2. System Model and Problem Definition

A parallel program is usually represented by a Directed Acyclic Graph (DAG), which is also called a *task graph*. As defined in [12], a DAG consists of a tuple (V, E, T, C), where V, E, T, and C are the set of task nodes, the set of communication edges, the set of computation costs associated with the task nodes, and the set of communication costs associated with the edges, respectively. $T(V_i)$ is a computation cost for task V_i and $C(V_i, V_j)$ is the communication cost for edge $E(V_i, V_j)$ which connects task V_i and V_j. The edge $E(V_i, V_j)$ represents the precedence constraint between the node V_i and V_j. In other words, task V_j can start the execution only after the output of V_i is available to V_j. When the two tasks, V_i and V_j, are assigned to the same processor, $C(V_i, V_j)$ is assumed to be zero since intra-processor communication cost is negligible compared with the interprocessor communication cost. The weights associated with nodes and edges are obtained by estimation [18].

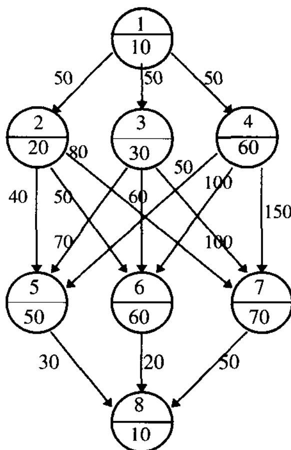
This paper defines two relations for precedence constraints. The $V_i \Rightarrow V_j$ relation indicates the strong precedence relation between V_i and V_j. That is, V_i is an immediate parent of V_j and V_j is an immediate child of V_i. The terms *iparent* and *ichild* are used to represent immediate parent and immediate child, respectively. The $V_i \rightarrow V_j$ relation indicates the weak precedence relation between V_i and V_j. That is, V_i is a parent of V_j but not necessarily the immediate one. $V_i \rightarrow V_j$ and $V_j \rightarrow V_k$ imply $V_i \rightarrow V_k$. $V_i \Rightarrow V_j$ and $V_j \Rightarrow V_k$ do not imply $V_i \Rightarrow V_k$, but imply $V_i \rightarrow V_k$.

The relation → is transitive, and the relation ⇒ is not. A node without any parent is called an *entry node* and a node without any child is called an *exit node*.

Graphically, a node is represented as a circle with a dividing line in the middle. The number in the upper portion of the circle represents the node ID number and the number in the lower portion of the circle represents the computation cost for the node. For example, for the sample DAG in Figure 1, the entry node is V_1 which has a computation cost of 10. In the graph representation of a DAG, the communication cost for each edge is written on the edge itself. For each node, *incoming degree* is the number of input edges and *outgoing degree* is the number of output edges. For example, in Figure 1, the incoming and outgoing degrees for the node V_5 are 3 and 1, respectively. A few terms are defined here for a more clear presentation.

Definition 1: A node is called a *fork node* if its outgoing degree is greater than 1.

Definition 2: A node is called a *join node* if its incoming degree is greater than 1.



(Figure 1) Sample Dag

Note that the fork node and the join node are not exclusive terms, which means that one node can be both a fork and also a join node; i.e., both of the node's incoming and outgoing degrees are greater than one. Similarly, a node can be neither a fork nor a join node; i.e., both of the node's incoming and outgoing degrees are one. In the task graph of Figure 1, nodes V_1, V_2, V_3, and V_4 are fork nodes while nodes V_5, V_6, V_7, and V_8 are join nodes.

Definition 3: The *Earliest Start Time*, $EST(V_i, P_k)$, and *Earliest Completion Time*, $ECT(V_i, P_k)$, are the times that a task V_i can start and finish its execution on processor P_k, respectively. They are denoted as $EST(V_i)$ and $ECT(V_i)$, when the information on the processors are not necessary.

Definition 4: A *message arriving time (MAT)* from V_i to V_j, or $MAT(V_i, V_j)$, is the time that the message from V_i arrives at V_j. If V_i and V_j are scheduled on the same processor P_k, $MAT(V_i, V_j)$ becomes $ECT(V_i, P_k)$. Otherwise, $MAT(V_i, V_j) = ECT(V_i, P_k) + C(V_i, V_j)$.

Definition 5: An *iparent* of a join node is called its *primary iparent* if it provides the largest MAT to the join node. The primary iparent is denoted as $V_i = PIP(V_j)$ if V_i is the primary iparent of V_j. More formally, $V_i = PIP(V_j)$ if and only if $MAT(V_i, V_j) > MAT(V_k, V_j)$, for all V_k, $V_k \Rightarrow V_j$, $V_i \Rightarrow V_j$, $i \neq k$. If there are more than one iparent providing the same largest MAT, PIP is chosen arbitrary.

Definition 6: An immediate parent node of a, join node is called the *secondary iparent* of the join node if it provides the second largest MAT to the join node. The secondary iparent is denoted as $V_i = SIP(V_j)$ if V_i is the secondary iparent of V_j. Formally, $V_i = SIP(V_j)$ if and only if

$MAT(V_i, V_j) > MAT(V_k, V_j)$, for all V_k, $V_k \Rightarrow V_j$, $V_k \neq PIP(V_j)$, $V_i \Rightarrow V_j$, $i \neq k$. If there are more than one iparent providing the same second largest MAT, SIP is chosen arbitrary.

Definition 7: The processor which has the primary iparent for V_i is called the *primary processor* of V_i.

Definition 8: The level of a node is recursively defined as follows. The level of an entry node, V_0, is zero. Let $Lv(V_i)$ be the level of V_i. Then $Lv(V_0) = 0$. $Lv(V_j) = Lv(V_i) + 1$, $V_i \Rightarrow V_j$, for non-join node V_j. $Lv(V_j) = Max(Lv(V_i)) + 1$, $V_i \Rightarrow V_j$, for join node V_j. For example, the level of node V_1, V_2, V_5, V_8 are 0, 1, 2, and 3, respectively

The topology of the target system is assumed to be a complete graph; i.e., all processors can directly communicate with each other. Thus, the multiprocessor scheduling process becomes a mapping of the task nodes in the input DAG to the processors in the target system with the goal of minimizing the execution time of the entire program. The execution time of the entire program after scheduling is called the *parallel time* to be distinguished from the completion time of an individual task node.

3. The Proposed Algorithm

3.1 Motivation

We recently proposed a new DBS algorithm, *Duplication First Reduction Next (DFRN)*, which outperforms schedulers with the same or less complexities and provides a comparable performance to the schedulers with higher complexities [14]. Most of DBS algorithms, including DFRN, assume that an unlimited number of processors are available for scheduling. This assumption makes the design of DBS algorithms simpler. Recently, scalable DBS algorithms [16, 17] are considered due to the limited number of processors in real world situations. Our objective is to propose a scalable DFRN algorithm which adjusts the extent of the duplication according to the number of processors available.

3.2 Description of the Algorithm

Figure 2 presents the high level description of the proposed algorithm, *Scalable scheduling with Duplication First Reduction Next (SDFRN)*. In this figure, the notations, P_c, P_u, PIP, IP, LN, and JN are used for the primary processor, an unused processor, the primary iparent, an iparent, the last node, and a join node, respectively. The *last node* of processor P_i is the most recent node assigned to P_i. For example, In Figure 3.(c), the last node of P_1, P_2, and P_3 are V_8, V_3, and V_2, respectively. The term, iparent, used in the algorithm in Figure 2 indicates the iparent which has the minimum EST if there are more than one iparent image across different processors. For example, in Figure 3.(a), V_3 on P_2 is identified as the iparent of its ichild since $EST(V_3, P_2) = 10$ while $EST(V_3, P_k) = 70$ for $k = 1, 4$, and 5. The primary iparent and the primary processor are used in the same way.

Note that the algorithm is presented in a generic form so that we can use any list scheduling algorithm as a node selection heuristic. The node selection heuristic decides which node is considered first for scheduling. Heavy Node First (HNF) [19] is used as the node selection heuristic in this paper. The HNF heuristic assigns the nodes in a DAG to the processors, level by level. At each level, the scheduler selects the eligible nodes for scheduling in descending order based on computational weight, with the heaviest node (i.e. the node which has the largest computation

cost) selected first. The node is selected arbitrarily if multiple nodes at the same level have the same computation cost.

Step (1) builds a priority queue according to the node selection heuristic used. Step (2) considers each node in the queue one by one in FIFO manner. Steps (3) through (10) handle the case that the node under consideration, V_i, is a non-join node. If the iparent identified in Step (4) is not the last node and there is an unused processor available as shown in Step (5), tasks scheduled on the processor up to the iparent are copied to (i.e. duplicated) the unused processor. Then V_i is scheduled onto the unused processor to make EST of V_i the same as ECT of the iparent. Otherwise, EST of V_i is increased due to the computation time of the tasks between the iparent and the last node in the schedule. If the iparent is the last node or there is no unused processor available, V_i is scheduled onto the processor where the iparent has been scheduled as shown in step (9). The SDFRN algorithm does not duplicate its parent if there is no available processors in the system even though the iparent is not the last node.

If V_i is a join node, the primary iparent of V_i and the primary processor are identified in step (12). DFRN is applied to a join node in steps (15) or (17) after handling the last node in the same way. DFRN(P_a, V_i) consists of two procedures, try_duplication(P_a, V_i) and try_deletion(P_a, V_i), as shown in steps (21) and (22). try_duplication(P_c, V_i) first tries to duplicate the iparent giving the largest MAT to V_i. The procedure recursively searches its iparent from V_i in a bottom-up fashion until it finds the parent which has already been scheduled on Pa as shown in step (24) and (25). After the duplication step, try_deletion(P_a, V_i) decides whether to delete any of the duplicated tasks based on the two conditions in step (30). Details of the two procedures

Scheduling algorithm with SDFRN

```
(1) initialize() // build a priority queue using HNF
(2) for each node Vi in the queue // in FIFO manner
(3)   if Vi is not a JN // Vi has only one IP
(4)     identify the IP
(5)     if the IP is not LN and an unused processor is
        available
(6)       copy the schedule up to the IP onto Pu
(7)       schedule Vi to Pu.
        else
(9)       schedule Vi to the PE having the IP
(10)    endif
(11)  else        // if Vi is a join node
(12)    identify PIP and Pc
(13)    if PIP is not LN and an unused processor is
        available
(14)      copy the schedule up to PIP onto Pu
(15)      DFRN (Pu, Vi)  // apply DFRN to Pu
(16)    else
(17)      DFRN (Pc, Vi)  // apply DFRN to Pc
(18)    endif
(19)  endif
(20) end for

DFRN(Pa, Vi)
(21) try_duplication(Pa, Vi)
(22) try_deletion(Pa, Vi)

try_duplication(Pa, Vi)
(23) for each Vp, (MAT(Vp, Vi) ≥ MAT(Vq, Vi), Vp ⇒ Vi,
     Vq ⇒ Vi, p ≠ q, Vp and Vq are not on Pa yet)
       // from the node giving the largest MAT to the
       node giving the smallest MAT
(24)  if there is any Vx, (MAT(Vx, Vp) ≥ MAT(Vy, Vp), Vx
      ⇒ Vp, Vy ⇒ Vp, x ≠ y, Vx and Vy are not on Pa yet)
(25)     try_duplication(Pa, Vx)
              //traces the IP which is not on Pa
(26)   else      // if all its IPs are scheduled on Pa
(27)     schedule Vp onto Pa
              //duplicates the IP which is not on Pa
(28)   endif
(29) end for

try_deletion(Pa, Vi)
(30)  delete any duplicated task Vk  if
   (i) ECT(Vk, Pa) > MAT(Vk, Vd) or
      // Vd is the ichild of Vk for which Vk is duplicated
   (ii) ECT(Vk, Pa) > MAT(SIP(Vi), Vi))
```

(Figure 2) Description of the Sdfrn Algorithm

can be found in [14]. Since the algorithm is executed for each node, N processors are enough to guarantee the necessary duplications in the algorithm, where N is the number of task nodes in the input DAG.

Since try_duplication(P_a, V_i) duplicates parents by the order of MAT, the sorting takes $O(V^2)$, which makes the complexity of the routine $O(V^2)$. try_deletion(P_a, V_i) also takes $O(V^2)$ time since it considers deletion m times and takes $O(p)$ time for calculation of EST(V_i, P_a) whenever any node is deleted, where m is the number of tasks duplicated and p is the number of deleted iparent of the node, $m \leq V$, $p \leq V$. Thus the complexity of try_deletion(P_a, V_i) becomes $O(V^2)$, which makes the complexity of DFRN(P_a, V_i) to be $O(V^2)$. The whole complexity becomes $O(V^3)$ since DFRN(P_a, V_i) is executed q times where q is the number of join nodes in the DAG, $q \leq V$.

For illustration, Figure 3 contains the schedules obtained by SDFRN algorithm with various number of processors for the sample DAG of Figure 1. In this example, P_i represents processing element i; PT is the Parallel Time of the DAG; and [EST(V_i, P_k), i, ECT(V_i, P_k)] represents the earliest starting time and earliest completion time of task i. In this example, even though there are more than 5 processors available in the system, the algorithm uses only 5 processors. Thus, the figure does not contain the cases where more than 5 processors are used.

It is proven that DFRN algorithm has the following two properties [14]. Due to lack of space, the proofs are omitted, but can be found in [14]. The performance evaluation of DFRN algorithm can be found in [20].

1) For a general input DAG, the schedule obtained by DFRN algorithm is guaranteed to be less than or equal to the length of the critical path of the input DAG.

P1: [0, 1, 10] [10, 4, 70] [70, 3, 100] [110, 7, 180] [180, 8, 190]
P2: [0, 1, 10] [10, 3, 40]
P3: [0, 1, 10] [10, 2, 30]
P4: [0, 1, 10] [10, 4, 70] [70, 3, 100] [100, 6, 160]
P5: [0, 1, 10] [10, 4, 70] [70, 3, 100] [100, 5, 150]
(a) Schedule by SDFRN with 5 processors (PT = 190)

P1: [0, 1, 10] [10, 4, 70] [70, 3, 100] [110, 7, 180] [180, 5, 230] [230, 8, 240]
P2: [0, 1, 10] [10, 3, 40]
P3: [0, 1, 10] [10, 2, 30]
P4: [0, 1, 10] [10, 4, 70] [70, 3, 100] [100, 6, 160]
(b) Schedule by SDFRN with 4 processors (PT = 240)

P1: [0, 1, 10] [10, 4, 70] [70, 3, 100] [110, 7, 180] [180, 6, 240] [240, 5, 290] [290, 8, 300]
P2: [0, 1, 10] [10, 3, 40]
P3: [0, 1, 10] [10, 2, 30]
(c) Schedule by SDFRN with 3 processors (PT = 300)

P1: [0, 1, 10] [10, 4, 70] [70, 2, 90] [90, 3, 120] [120, 7, 190] [190, 6, 250] [250, 5, 300] [300, 8, 310]
P2: [0, 1, 10] [10, 3, 40]
(d) Schedule by SDFRN with 2 processors (PT = 310)

(Figure 3) Schedules By Sdfrn Algorithm With Various Number Of Processors

2) For an input tree, the schedule obtained by DFRN is always optimal.

The two properties of DFRN algorithm are also valid for SDFRN algorithm if there are at least N processors available in the system since SDFRN with N available processors generate the same schedule as that obtained by DFRN algorithm with unlimited number of processors. However, these properties are not valid as the number of processors available is decreased below N.

4. Performance Comparison

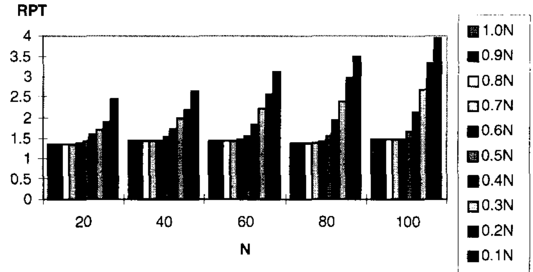
We generated 1000 random DAGs to compare the performance of SDFRN with various number of pro-

cessors available. We used three parameters the effects of which we were interested to investigate: the number of nodes, CCR (Communication to Computation Ratio), and the average degree (defined as the ratio of the number of edges to the number of nodes in the DAG). The numbers of nodes used are 20, 40, 60, 80, and 100 while CCR values used are 0.1, 0.5, 1.0, 5.0, and 10.0. CCR is the ratio of average communication cost to average computation cost. We gave a parameter value to control the degree of the nodes in the DAG and obtained the average degree from a number of resulting DAGs. 40 DAGs are generated for each case of the 25 combinations, which makes 1000 DAGs.

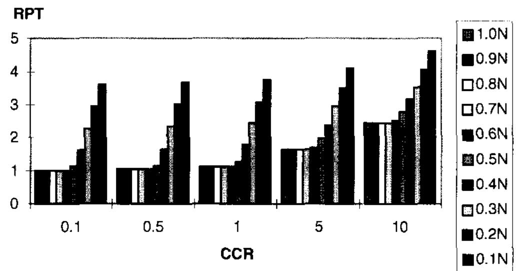
For performance comparison, we define one *normalized* performance measure named *Relative Parallel Time (RPT)*, which is a ratio of the parallel time to the low bound which is the sum of the computation costs of the nodes in the critical path. For example, if the parallel time obtained by SDFRN is 200 and the low bound is 100, RPT of SDFRN is 2.0. A smaller RPT value is indicative of a shorter parallel time. The RPT of any scheduling algorithm can not be lower than one. If RPT is less than one, the system is not executing all the tasks in the critical path, which violates the system model defined in Section 2.

Graphical representations of the performance degradation according to the number of available processors are shown in Figure 4, Figure 5, and Figure 6 with respect to N (the number of nodes), CCR, and the average degree, respectively. In these Figures, UBN stands for unbounded number of processors while aN represents $a \times$ N available processors. For example, 0.9N indicates that the number of available processors is 90% of the number of task nodes.

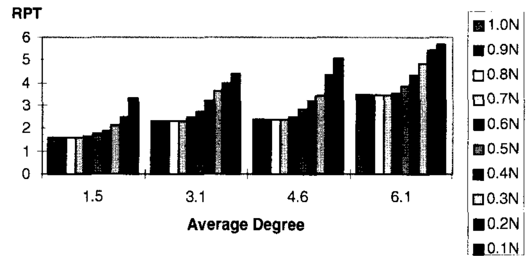
These Figures show that the SDFRN achieves comparable performance to that of DFRN with unbounded number of processors until the number of available processors is reduced to 60% of the number of task nodes, in most cases. The performance degradation becomes significant as the number of processors is decreased to less than 50% of the number of nodes. The differences in values of N, CCR, and the average degree do not change the pattern of the performance degradation but change the scale of that.



(Figure 4) Performance With Respect To N (CCR = 3.3, D = 3.5)



(Figure 5) Performance With Respect To CCR (N = 100, D = 3.8)



(Figure 6) Performance With Respect To The Average Degree (N = 100, CCR = 3.3)

5. Conclusion

This paper proposed a scalable DBS algorithm, SDFRN, which schedules the task nodes in input DAGs onto limited number of processors in target systems with graceful performance gradation as the number of available processors is decreased. SDFRN with N available processors generates the same schedule as that obtained by DFRN with unlimited number of processors, where N is the number of task nodes in input DAGs. If at least N processors are available, the schedule obtained by SDFRN is guaranteed to be shorter than or equal to the length of the critical path in the application. Also, SDFRN generates an optimal schedule for tree structured input DAGs in the case.

Our performance study showed that SDFRN achieves comparable performance to the original DFRN algorithm with unlimited number of processors until the number of available processors is reduced to about 60% of the number of nodes. The performance degradation becomes significant when the available number of processors is decreased to less than 50% of the number of nodes. The performance degradation shows similar patterns regardless of the number of nodes, CCR values, and the average degrees.

References

[1] B. Shirazi, A. R. Hurson, "Scheduling and Load Balancing: Guest Editors' Introduction," *Journal of Parallel and Distributed Computing*, Dec. 1992, pp. 271~275.

[2] B. Shirazi, A. R. Hurson, "A Mini-track on Scheduling and Load Balancing: Track Coordinator's Introduction," *Hawaii Int'l Conf. on System Sciences (HICSS-26)*, Jan. 1993, pp. 484~486.

[3] B. Shirazi, A. R. Hurson, K. Kavi, "Scheduling & Load Balancing," *IEEE Press*, 1995.

[4] B. Shirazi, H.-B. Chen, and J. Marquis, Comparative Study of Task Duplication Static Scheduling versus Clustering and Non-Clustering Techniques, *Concurrency: Practice and Experience*, Vol. 7(5), Aug. 1995, pp. 371~389.

[5] M.Y. Wu, A dedicated track on Program Partitioning and Scheduling in Parallel and Distributed Systems, *Hawaii Intl Conference on Systems Sciences*, Jan. 1994.

[6] T. Yang and A. Gerasoulis, A dedicated track on Partitioning and Scheduling for Parallel and Distributed Computation, *Hawaii Intl Conference on Systems Sciences*, Jan. 1995.

[7] T. L. Adam, K. Chandy, and J. Dickson, A Comparison of List Scheduling for Parallel Processing System, *Communication of the ACM*, Vol. 17, No. 12, Dec. 1974, pp. 685~690.

[8] I. Ahmad and Y. K. Kwok, A New Approach to Scheduling Parallel Program Using Task Duplication, *Proc. of Intl Conf. on Parallel Processing*, Vol. II, Aug. 1994, pp. 426~433.

[9] H. Chen, B. Shirazi, and J. Marquis, Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication, *Intl Conf. on Parallel and Distributed Systems*, Dec. 1993, pp. 270~275.

[10] Y. C. Chung and S. Ranka, Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors, *Supercomputing92*, Nov. 1992, pp. 512~521.

[11] J. Y. Colin and P. Chretienne, C.P.M. Scheduling with Small Communication Delays and Task Duplication, *Operations Research*, 1991, pp. 680~684.

[12] S. Darbha and D. P. Agrawal, SDBS: A task duplication based optimal scheduling algorithm, *Scalable High Performance Computing Conf.*, May 1994, pp. 756~763.

[13] B. Kruatrachue and T. G.˙ Lewis, Grain Size Determination for parallel processing, *IEEE Software*, Jan. 1988, pp. 23~32.

[14] G.-L. Park, B. Shirazi, and J. Marquis, DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems, *Intl Parallel Processing Symp.* April 1997, pp. 157~166.

[15] C. H. Papadimitriou and M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, *ACM Proc. of Symp. on Theory of Computing (STOC)*, 1988, pp. 510~513.

[16] S. Darbha and D. P. Agrawal, A Fast and Scalable Scheduling Algorithm for Distributed Memory Systems, *Proc. of Symp. On Parallel and Distributed Processing*, Oct. 1995, pp. 60~63.

[17] Y.-K. Kwok and I. Ahmad, Exploiting Duplication to Minimize the Execution Times of Parallel Programs on Message-Passing Systems, *Symp. On Parallel and Distributed Processing*, Oct. 1994, pp. 426~433.

[18] M. Y. Wu and D. D. Gajski, Hypertool: A Programming Aid for Message-Passing Systems, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 3, Jul. 1990, pp. 330~340.

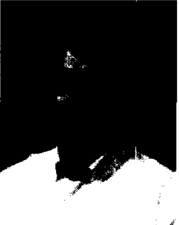
[19] B. Shirazi, M. Wang, and G. Pathak, Analysis and Evaluation of Heuristic Methods for Static Task Scheduling, *Journal of Parallel and Distributed Computing*, Vol. 10, No. 3, 1990, pp. 222~232.

[20] G. Park, B. Shirazi, J. Marquis, Modeling and Evaluation of Task Scheduling Algorithm Using Stochastic Petri Net,, *Intl Conf. on Parallel and Distributed Processing Techniques and Applications*, Vol. III, June. 2000, pp. 1269~1275. Vol. 10, No. 3, 1990, pp. 222~232.

◐ 저 자 소 개 ◑

박 경 린

1986년 중앙대학교 전자계산학과 학사
1992년 텍사스 주립대 전자계산학과 석사
1997년 텍사스 주립대 전자계산학과 박사
1998년~현재 제주대학교 자연과학대학 전산통계학과 조교수
관심분야 : 분산/병렬처리 시스템



이 상 준

1984년 중앙대학교 전자계산학과 학사
1989년 중앙대학교 전자계산학과 석사
1992년 중앙대학교 전자계산학과 박사
1993년~현재 제주대학교 공과대학 통신컴퓨터 공학부 부교수
관심분야 : 인공지능

이 봉 규

1988년 서강대학교 전자계산학과 학사
1990년 서울대학교 컴퓨터공학과 석사
1995년 서울대학교 컴퓨터공학과 박사
1998년~현재 제주대학교 자연과학대학 전산통계학과 조교수
관심분야 : 패턴인식, 병렬처리, 신경망