병렬 처리 시스템을 위한 효율적인 복제 중심 스케줄링 알고리즘

박 경 린† · 추 현 승††

요 약

다중 처리기 시스템에서의 병렬 처리를 위한 스케줄링 문제는 지난 수 십년 동안 중요한 연구 과제가 되어왔다. 다중 처리기 스케줄링 문제(multiprocessor scheduling problem)란 다중 처리기 시스템에서 병렬 수행 시간(parallel execution time)을 최소화 할 수 있는 최적의 스케줄을 구하는 문제로 정의된다. 복제 중심 타스크 스케줄링은 이러한 문제를 풀기위한 비교적 새로운 접근 방법이다. 이 논문은 복제 중심 스케줄링 알고리즘들을 타스크 복제방법에 따라서 전체 복제와 부분 복제의 두 가지로 분류하고, 그 두 가지 방법의 장점들을 결합한 새로운 스케줄링 알고리즘을 제안한다. 시뮬레이션 결과는 이 논문에서 제안된 스케줄링 알고리즘이 비슷한 복잡도(time complexity)를 갖는 다른 스케줄링 알고리즘보다 우수함을 보여준다.

An Efficient Duplication Based Scheduling Algorithm for Parallel Processing Systems

Gyung-Leen Park† · Hyun-Seung Choo††

ABSTRACT

Multiprocessor scheduling problem has been an important research area for the past 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 multiprocessor scheduling problems. This paper classifies DBS algorithms into two categories according to the task duplication method used. The paper then presents a new DBS algorithm that extracts the strong features of the two categories of DBS algorithms. The simulation study shows that the proposed algorithm achieves considerable performance improvement over existing DBS algorithms with similar time complexity.

1. 서 론

병렬 처리 시스템에 있어서, 응용 프로그램을 효과적으로 분할(partition)하고 스케줄링하는 문제는 지난 수 십년 동안 중요한 연구 과제였다[1-7]. 분할 과정에서 어떤 응용 프로그램이 병렬 처리 시스템에서의 수행을 위해 타스크(task)들로 분해가 되면, 그 응용 프로그램은 타스크들과 그 타스크들간의 선후 제약(precedence constraint)을 나타내는 비순환 방향 그래프(DAG: Directed Acyclic Graph) 혹은 타스크 그래프(task graph)로 표현이 된다. 다중 처리기 스케줄링 문제(multiprocessor scheduling problem)는 타스크 그래프에 나타난 선후 제약을 위배하지 않으면서 응용 프

† 정 회 원 : 세주대학교 자연과학대학 전산통계학과 교수
†† 정 회 원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 1999년 3월 24일, 심사완료 : 1999년 7월 13일

프로그램의 병렬 수행 시간(parallel execution time)을 최소화 할 수 있는 최적의 스케줄을 찾아내는 문제이다.

이 다중 처리기 스케줄링 문제는 이미 NP-complete로 증명이 되었기 때문에 휴어리스틱(heuristics)에 기반한 스케줄링 알고리즘들이 주 연구 과제가 되어 왔으며[8], 이러한 휴어리스틱에 기반한 스케줄링 알고리즘들은 크게 복제 중심 스케줄링 알고리즘들(DBS: Duplication Based Scheduling)과 타스크 복제를 허용하지 않는 스케줄링 알고리즘들로 구분 할 수 있다. 복제 중심 스케줄링은 타스크들의 복제를 통하여 처리기간의 통신 오버헤드를 줄일 수 있는 장점이 있는 반면, 데이터 분산과 통합에 관한 다른 오버헤드를 포함하는 단점이 있다[9-18]. 타스크 복제를 허용하지 않는 스케줄링 방법은 다시 크게 클러스터링 알고리즘(clustering algorithm)[19,20]과 리스트 스케줄링 알고리즘(list scheduling algorithm)[1,21]으로 나눌 수 있다.

복제 중심 스케줄링 방법은 다중 처리기 스케줄링 문제를 해결하기 위하여 비교적 최근에 제안된 접근 방법으로써, 로칼(local) 처리기에 스케줄 되지 않은 선행 작업들을 로칼 처리기에 복제함으로써 처리기들간의 통신 오버헤드를 줄이는 기법이다. 복제 중심 스케줄링 문제 역시 NP-complete로 증명이 되었기 때문에 휴어리스틱에 기반한 많은 복제 중심 스케줄링 알고리즘들이 제안되었는데[9-18], 본 논문은 타스크 복제 방법에 따라 선택 복제 스케줄링(SFD: Scheduling with Full Duplication) 알고리즘들[12-14]과 부분 복제 스케줄링(SPD: Scheduling with Partial Duplication) 알고리즘들[15,16,18]로 분류한다.

부분 복제 스케줄링과 전체 복제 스케줄링의 두 접근 방법사이에는 복잡도(스케줄을 작성하는데 걸리는 시간)와 성능(작성된 스케줄의 길이)이라는 타협 관계(trade-off)가 존재하는데, 이 논문에서는 부분 복제 스케줄링 방법의 복잡도와 유사한 복잡도로 전체 복제 스케줄링 방법의 성능과 유사한 성능을 가지는 새로운 복제 중심 스케줄링 알고리즘을 제안한다. 시뮬레이션 결과는 이 논문에서 제안된 스케줄링 알고리즘이 같거나 작은 복잡도를 가지는 스케줄링 알고리즘들을 성능면에서 월등하게 능가하며, 더 큰 복잡도를 가지는 스케줄링 알고리즘에 대해서는 미미하게 열등한 성능을 가짐을 보여준다. 또한 성능 향상은 통신 비용 대 연산 비용의 비율(CCR: Communication to Computation Ratio)이 커질수록 더욱 커짐을 보여준다.

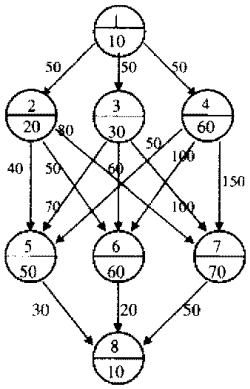
이 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장에서는 시스템 모델과 해결하려는 문제, 그리고 이 논문에서 사용되는 용어들이 체계적으로 정의된다. 3장에서는 기존의 스케줄링 알고리즘들이 간략하게 정리되고, 이 논문에서 제안된 알고리즘은 4장에서 제시된다. 5장에서는 제안된 스케줄링 알고리즘과 기존의 스케줄링 알고리즘들과의 성능비교를 보여주며, 6장에서 결론을 맺는다.

2. 시스템 모델과 문제 정의

병렬 프로그램은 통상 타스크 그래프라고 불리우는 비순환 방향 그래프로 표현이 된다[16]. 타스크 그래프는 (V, E, T, C)의 튜플로 정의되는데, 여기서 V는 타스크 노드들의 집합, E는 엣지들의 집합, T는 각 타스크 노드들의 수행 시간들, C는 각 엣지들에 할당되어진 일련의 통신 비용들로 정의된다. $T(V_i)$는 타스크 V_i의 수행에 필요한 연산 시간을 나타내고, $C(V_i, V_j)$는 타스크 V_i와 V_j를 연결하는 엣지 $E(V_i, V_j)$에 대한 통신 비용, 즉 타스크 V_i에서 타스크 V_j에 메시지를 보내는데 필요한 시간을 나타낸다. 여기서 $E(V_i, V_j)$는 타스크 V_i와 V_j 간의 선후 제약을 나타낸다. 즉, 타스크 V_j는 타스크 V_i가 수행을 끝내고 타스크 V_i의 연산 결과를 포함하는 메시지가 타스크 V_j에 도착한 다음에야 수행이 가능함을 의미한다. 만약에 두 타스크 V_i와 V_j가 같은 처리기에서 수행이 될 경우에는 $C(V_i, V_j)$는 0으로 가정한다. 왜냐하면 같은 처리기내에서 수행되는 두 타스크 간의 통신 비용은 다른 처리기에서 수행되는 두 타스크간의 통신 비용에 비하여 무시될 수 있을 만큼 작기 때문이다. 각 노드와 엣지에 할당되는 비용들은 어떤 측정치에 의존한다[6].

본 논문에서는 선후 제약에 관하여 두 개의 관계를 정의하여 사용한다. $V_i \rightarrow V_j$ 관계는 강한 선후 관계(strong precedence relation)를 나타낸다. 즉, 타스크 V_i는 타스크 V_j의 직접 선행 작업임을 나타낸다. $V_i \rightarrow V_j$ 관계는 약한 선후 관계를 나타낸다. 즉, 타스크 V_i가 타스크 V_j의 선행 작업임을 나타내지만 타스크 V_i가 타스크 V_j의 직접 선행 작업임을 요구하지 않는다. 관계 $\rightarrow$는 추이적(transitive)이지만 관계 $\Rightarrow$는 추이적이 아니다. 선행 작업이 없는 노드를 입구 노드(entry node), 후행 작업이 없는 노드를 출구 노드(exit node)라고 한다.

타스크 그래프에서 노드는 원으로 표시되고, 그 원을 상하로 나누는 선을 기준으로 위에 표시된 숫자는 그 타스크의 ID 번호를, 아래에 표시된 숫자는 그 타스크의 연산 비용을 표시한다. 예를 들면, (그림 1)에 있는 타스크 그래프에서 입구 노드는 V_1이고 그 노드의 연산 비용은 10이다. 즉, 입구 노드 V_1의 수행 완료에 10 단위시간이 요구된다. 통신 비용은 엣지위에 숫자로 표시되고, 각 노드에 대하여 진입 차수(indegree)는 그 노드를 향한 엣지들의 수를, 출력 차수는 그 노드에서 나가는 엣지들의 수를 의미한다. (그림 1)의 예에서, 노드 V_5의 진입 차수와 출력 차수는 각각 3과 1이다.



(그림 1) 타스크 그래프의 예

좀더 명확한 논의를 위하여 이 논문에서 사용되는 용어들을 아래와 같이 정의한다.

정의 1: 출력 차수가 1 보다 큰 노드를 포크노드(fork node)라고 한다.

정의 2: 진입 차수가 1 보다 큰 노드를 조인노드(join node)라고 한다.

정의 3: 가장 빠른 시작 시간(Earliest Start Time) 혹은 시작 시간, EST(V_i, P_k)과 가장 빠른 종료 시간(Earliest Completion Time) 혹은 종료 시간, ECT(V_i, P_k)은 각각 타스크 V_i가 처리기 P_k에서 수행을 시작하는 시간과 수행을 종료하는 시간을 의미한다.

정의 4: 관계 $V_i \Rightarrow V_j$에 대하여, 타스크 V_i에서 보낸 메시지가 타스크 V_j에 도착하는 시간을 메시지 도착 시간(Message Arriving Time)이라 하며 MAT(V_i, V_j)로 표시한다.

정의 5: 어떤 조인노드에 대하여, 그 노드에 가장 큰 메시지 도착 시간을 제공하는 직접선행 작업을 임계 선행 작업(CIP: Critical Immediate Predecessor)이라고 하며 조인노드 V_j의 임계 선행 작업이 V_i일 경우 V_i = CIP(V_j)로 표시한다.

정의 6: 어떤 조인노드에 대하여, 그 노드에 두번째로 큰 메시지 도착 시간을 제공하는 직접 선행 작업을 결정 선행 작업(DIP: Decisive Immediate Predecessor)이라고 하며 조인노드 V_j의 결정 선행 작업이 V_i일 경우 V_i = DIP(V_j)로 표시한다.

정의 7: CIP(V_i)가 스케줄 되어 있는 처리기를 V_i의 임계 처리기(CP: Critical Processor)라고 한다.

정의 8: 타스크 그래프상에서 임계 경로(critical path)란 입구 노드에서 출구 노드까지의 길이(path length)가 가장 긴 경로를 말하는데, 임계 경로의 길이는 그 경로에 속한 연산 비용과 통신 비용의 합이다. 임계 경로 연산 길이(CPLEC: Critical Path Length Excluding Communication cost)는 경로 길이를 그 경로에 속한 연산 비용들만의 합으로 계산 할 경우의 경로 길이를 의미한다.

정의 9: 노드의 레벨(level)은 다음과 같이 재귀적으로 정의된다. 먼저 입구 노드 V_k의 레벨은 0으로 정의되며 $Lv(V_k) = 0$으로 표시된다. 어떤 노드 V_j에 대하여 만약 V_j가 조인노드가 아니고 $V_i \Rightarrow V_j$관계를 만족시키는 V_i가 있다면 $Lv(V_j) = Lv(V_i) + 1$이 된다. 만약 V_j가 조인노드라면, $V_i \Rightarrow V_j$관계를 만족시키는 모든 V_i에 대하여 $Lv(V_j) = Max(Lv(V_i)) + 1$이 된다.

기존의 다른 복제 중심 스케줄링 알고리즘의 경우와 같이 이 논문에서는 대상 시스템내의 처리기의 개수는 무한하다고 가정하며[17], 유한한 수의 처리기에 대한 적용은 따로 [10]에 정리되어 있다. 대상 시스템의 토폴로지는 완전 연결이라고 가정하며(모든 처리기들은 직접 통신할 수 있다) 통신 비용의 차이는 처리기들 사이의 거리 때문이 아니라 타스크들 간에 전송해야 할 메시지의 양의 차이에 기인한다. 완전 연결이 아닌 토폴로지들에 관한 이슈(issue)들은 [22]에 정리되어 있다. 이제 다중 처리기 스케줄링 문제는 타스크 그래프에 있는 노드들을 병렬 수행 시간을 최소화할 수 있도록 시스템내의 처리기들에 할당하는 문제로 모델링 된다. 여기서 병렬 수행 시간은 스케줄의 길이, 즉 전체 응용 프로그램의 수행을 종료하는데 필요한 시간을 말하며, 전체 프로그램의 수행 시간을 개개의 타스크

들의 수행 시간들과 구분하기 위해 사용된다.

3. 기존의 스케줄링 알고리즘들

3장에서는 1장에서 보인 분류에 따라 각 종류에 속하는 대표적인 스케줄링 알고리즘들의 동작 원리를 간략하게 소개하며, 여기서 소개되는 스케줄링 알고리즘들은 5장의 성능 비교에서 사용된다.

3.1 Heavy Node First(HNF) 알고리즘

HNF 스케줄링 알고리즘[1]은 타스크 그래프의 각 노드에 대하여 낮은 레벨의 노드들부터 하나씩 처리기에 스케줄 해 나간다. 같은 레벨에 속하는 노드들에 대하여, HNF 스케줄러는 먼저 연산 비용이 가장 높은 노드를 찾으며 같은 연산 비용을 가지는 노드들에 대해서는 그들 중의 하나가 무작위로 선택된다. 선택된 노드는 가장 빠른 수행 시작 시간을 제공하는 처리기를 찾아 할당되며 결국 이러한 작업이 각 레벨의 모든 노드에 대하여 행해진 다음에 전체 스케줄이 얻어진다. HNF 스케줄러의 설계 동기는 연산 비용이 큰 노드들을 먼저 스케줄함으로써 전체 스케줄의 길이를 줄이려는 데 있다.

3.2 Linear Clustering(LC) 알고리즘

LC 알고리즘[19]은 먼저 타스크 그래프에서 임계 경로를 찾은 다음 그 경로에 속한 모든 노드들로 한 클러스터를 형성한다. 그 클러스터에 속한 노드들과 그 노드들과 인접한 엣지들은 원래의 타스크 그래프에서 제거되고, 노드들과 엣지들이 제거된 타스크 그래프에 대하여 모든 노드가 제거될 때까지 같은 과정이 반복된다. 결과적으로 생성된 각 클러스터는 각각의 처리기들에 할당된다.

3.3 Fast and Scalable Scheduling(FSS) 알고리즘

FSS[18]알고리즘은 먼저 입력 타스크 그래프를 탐색하면서, 각 노드의 가능한 수행 시작 시간과 종료 시간을 계산한다. 입구 노드일 경우 시작 가능 시간은 0이 되고, 종료 시간은 그 노드의 연산 비용이 된다. 타스크 그래프상의 어떤 노드가 조인노드가 아닐 경우에는 필요한 경우 선행 작업들의 복제가 이루어 지고, 결국 그 노드의 시작 시간은 그 선행 작업의 종료 시간이 된다. 조인노드일 경우에는 임계 선행 작업이 아닌 다른 선행 작업들의 복제는 시도하지 않으며, 결국 그 노드의 시작 시간은 임계 선행 작업의 종료 시간과 결정 선행 작업으로부터의 메시지 도착 시간 중 큰 값이 된다.

3.4 Critical Path Fast Duplication(CPFD) 알고리즘

CPFD알고리즘[12]은 먼저 타스크 그래프상의 노드들을 Critical Path Node(CPN), In-Branch Node(IBN), 그리고 Out-Branch Node(OBN)의 3개의 범주로 나눈다. CPN은 임계 경로에 속하는 노드들을 말하고, IBN은 그 노드로부터 임계 경로에 이르는 경로가 존재하는 그러한 노드들을 가리킨다. OBN는 CPN도 IBN도 아닌 다른 모든 노드들을 가리킨다. CPFD알고리즘은 CPN들을 가장 먼저 스케줄 하는데, 만약 어떤 CPN의 선행 작업이 되는 IBN중에서 아직 스케줄 되지 않은 노드가 있을 경우에는 그 노드를 그 CPN 보다 먼저 스케줄한다. CPN과 IBN들이 모두 스케줄된 다음에 OBN들을 스케줄한다. CPFD알고리즘의 설계 동기는 임계 경로에 속하는 CPN들을 상대적으로 먼저 스케줄함으로써, 병렬 수행 시간을 줄이려는 것이다.

3.5 비 교

각 알고리즘들의 실제 동작 결과를 보이기 위하여 (그림 2)에서는, (그림 1)의 타스크 그래프에 대하여 각 스케줄링 알고리즘들이 작성한 스케줄들을 보인다. 이 예에서, P_i는 처리기 i를 나타내고, PT는 병렬 수행 시간을 나타낸다. 어떤 상수 a, b, c에 대한 [a, b, c]의 표현은 [EST(V_i, P_k), i, ECT(V_i, P_k)], 즉 타스크 V_i의 처리기 P_k에서의 수행 시작 시간과 종료 시간을 나타낸다. 예를 들어 (그림 2) (a)에서, 첫 번째 행의 [0, 1, 10]과 [10, 4, 70]은 처리기 P1에서 타스크 V_1이 시간 0에 수행을 시작하여 시간 10에 그 수행을 종료하고, 타스크 V_4는 시간 10에 수행을 시작하여 시간 70에 그 수행을 종료함을 나타낸다. DFRN 스케줄링 알고리즘의 동작은 4장에서 자세히 기술되지만, 비교를 위하여 DFRN 알고리즘에 의한 스케줄도 (그림 2) (d)에 포함되었다.

P1: [0, 1, 10][10, 4, 70][190, 7, 260][260, 8, 270]
P2: [60, 3, 90][170, 6, 230]
P3: [60, 2, 80][160, 5, 210]
(a) HNF 알고리즘에 의한 스케줄 (PT = 270)

P1: [0, 1, 10][10, 4, 70][140, 7, 210][210, 8, 220]
P2: [0, 1, 10][10, 3, 40]
P3: [0, 1, 10][10, 2, 30]
P4: [0, 1, 10][10, 4, 70][100, 6, 160]
P5: [0, 1, 10][10, 4, 70][110, 5, 160]
(b) FSS 알고리즘에 의한 스케줄 (PT = 220)

P1: [0, 1, 10][10, 4, 70][190, 7, 260][260, 8, 270]
P2: [60, 3, 90][120, 5, 170]
P3: [60, 2, 80][170, 6, 230]
(c) LC 알고리즘에 의한 스케줄 (PT = 270)

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]
(d) DFRN 알고리즘에 의한 스케줄 (PT = 190)

P1: [0, 1, 10][10, 4, 70][70, 3, 100][100, 5, 150]
P2: [0,1,10][10,3,40][40,4,100][110,7,180][180,8,190]
P3: [0, 1, 10][10, 2, 30][30, 4, 90][100, 6, 160]
(e) CPFD 알고리즘에 의한 스케줄 (PT = 190)

(그림 2) 여러 가지 스케줄링 알고리즘들에 의해 작성된 스케줄들

4. 제안된 알고리즘

4.1 설계 동기

전체 복제 스케줄러는 일반적으로 좋은 성능에도 불구하고 긴 스케줄 작성 시간으로 인하여, 많은 수의 타스크들로 이루어지는 응용 프로그램에 적합하지 못하다는 단점이 있다. 많은 수의 타스크들로 이루어지는 커다란 응용 프로그램들을 위하여, 전체 복제 스케줄링 알고리즘 못지않은 병렬 수행 시간을 제공하면서, 스케줄 작성 시간은 보다 짧은 효율적인 스케줄링 알고리즘의 필요성은 이번 연구의 목적이 되었고, 그 목적은 DFRN(Duplication First Reduction Next)라는 새로운 접근 방법에 의해서 달성되었다.

부분 복제, 전체 복제, 그리고 DFRN 스케줄링 알고리즘의 차이는 조인노드에 대한 스케줄링 방식에 기인한다. 부분 복제 알고리즘은 임계 선행 작업이외의 다른 선행 작업들에 대한 복제를 시도하지 않는다는 점에서 전체 복제와 DFRN 스케줄링 알고리즘과 다르며, 전체 복제 스케줄링 알고리즘과 DFRN 알고리즘의 주된 차이점은 다음과 같다.

첫째, 전체 복제 스케줄링 알고리즘이 가능한 타스크 복제들의 영향을 먼저 재귀적으로 계산한 다음에 그 중 짧은 시작 시간을 제공하는 타스크 복제가 이루어 지는 반면에, DFRN 알고리즘은 현재 스케줄 하려는 타스크의 임계 처리기에 스케줄 되어 있지 않은 모든 선행 작업들을 아무런 계산 과정 없이 먼저 복제한 다음에 DFRN 알고리즘에서 제공하는 두 가지 조건중의 하나라도 만족시키는 선행 작업들을 제거한다. 타스크 복제 과정에서 가장 많은 시간을 요구하는 재귀적 수행 시작 시간 계산 과정을 생략하고, 제거 조건 검증을 통해 나중에 타스크들을 제거함으로써 스케줄 생성 시간을 줄일 수 있다.

둘째, 전체 복제 스케줄링 알고리즘들은 현재 스케줄 하려는 타스크의 선행 작업이 스케줄 되어 있는 모든 처리기에 대하여 타스크 복제를 시도하는 반면, DFRN 알고리즘은 오직 그 타스크의 임계 처리기에의 타스크 복제만을 시도한다. 실험 결과[9]에서 대부분의 경우에 타스크 복제 후에 임계 처리기에서의 종료 시간이 다른 처리기에서의 종료 시간보다 짧았기 때문에, 임계 처리기가 조인노드에 대한 가장 적합한 처리기라고 가정하고 타스크의 복제는 임계 처리기에 대해서만 이루어진다.

5장의 성능 비교에서 보이는 바와 같이, 이러한 두 가지의 차이점이 DFRN 알고리즘으로 하여금, 전체 복제 스케줄링 알고리즘들과는 비교할 수 없이 짧은 수행 시간으로 그들과 비교할만한 성능을 보이게 한다. 반면에 부분 복제 스케줄링 알고리즘에 대하여는 괄목할만한 성능의 향상을 보인다.

4.2 제안된 알고리즘의 동작

DFRN 알고리즘은 그림 3에 제시되어 있으며, 이 알고리즘의 기술에서 사용되는 기호들의 정의는 다음과 같다.

- **LN**: 마지막 노드(Last Node). 스케줄링 과정의 어떤 시점에서, 마지막 노드란 어떤 처리기 P_i에 가장 최근에 할당된 타스크 노드를 말한다.
- P_c: 임계 처리기(Critical Processor), P_u: 현재까지 사용되지 않은 처리기(Unused Processor).
- **CIP**: 임계 선행 작업(Critical Immediate Predecessor)
- **IP**: 직접 선행 작업(Immediate Predecessor).
- **JN**: 조인노드(Join Node)들의 집합.
- **V[P_k]**: 처리기 P_k에 스케줄 되어 있는 타스크들의 집합

DFRN을 이용한 스케줄링 알고리즘

```
(1) 초기화()          /* HNF를 이용하여 타스크 노드들의priority
                         queue를 생성. */
(2) for 큐에 있는 각 노드 Vi에 대하여   /* FIFO(First In First
    Out) 방식으로 */
(3)     If (Vi ∉ JN) /* Vi가 오직 하나의 직접 선행 작업
        만 가질 경우 */
(4)         IP와 IP가 스케줄 되어 있는 처리기 Pk를 식
            별한다.
(5)         if (IP = LN(Pk))      /* IP가 Pk의 마지막 노드이
                                     면 */
(6)             Pk에 Vi를 스케줄한다.
(7)         else    /* IP가 Pk의 마지막 노드가 아니면 */
(8)             사용되지 않은 처리기 Pu에 Pk에 있는 IP까
                지의 스케줄을 복제한다.
(9)             Vi를 Pu에 스케줄한다.
(10)        endif
(11)    else    /* Vi 가 조인노드이면 */
(12)        CIP와 CIP가 스케줄 되어 있는 처리기Pc 를 찾아
            낸다
(13)        if (CIP = LN(Pc))     /*CIP가 Pc의 마지막 노드
                                     이면 */
(14)            DFRN (Pc ,Vi)     /* DFRN 알고리즘을 Pc
                                     에 적용 */
(15)        else    /* CIP 가 마지막 노드가 아니면 */
(16)            사용되지 않은 처리기 Pu에 Pc에 있는 CIP까
                지의 스케줄을 복제한다.
(17)            DFRN (Pu, Vi)     /* DFRN 알고리즘을 Pu
                                     에 적용 */
(18)        endif
(19)    endif
(20) end for

DFRN(Pa, Vi)
(21) try_duplication(Pa, Vi)    /* 타스크 복제 */
(22) try_deletion(Pa, Vi)       /* 조건에 맞지 않는 타스크 제거 */

try_duplication(Pa, Vi)
(23) for each Vp (Vp ⇒ Vi, Vq ⇒ Vi, p ≠ q, Vp ∉ V[Pa], Vq ∉
     V[Pa], MAT(Vp ,Vi)≥MAT(Vq,Vi))
          /*제공하는 MAT의 내림차순으로 한 노드씩 선택*/
(24)     if∃Vx(Vx⇒Vp, Vy⇒Vp, x≠y,Vx ∉ V[Pa], Vy ∉ V[Pa],
         MAT(Vx,Vp) MAT≥(Vx,Vp))
         /* 만약 Pa에 스케줄 되지 않은 Vp의 IP가 있으면
         MAT의 내림차순으로 한 노드씩*/
(25)         try_duplication(Pa, Vx) /* Pa에 스케줄 되어 있지
             않은 IP들을 복제한다 */
(26)     else       /* 모든IP들이Pa에 스케줄 되어 있으면 */
(27)         Vp를 Pa에 스케줄한다. /* Pa에 스케줄 되어 있지
             않은 IP를 복제 */
(28)     endif
(29) end for

try_deletion(Pa, Vi)
(30) 복제된 타스크 중에서 다음 중 한 가지 조건을 만족하는 타
     스크Vk는 제거한다
     (i) ECT(Vk , Pa) > MAT(Vk, Vd )
     (ii) ECT(Vk , Pa) > MAT(DIP(Vi), Vi)
     /* Vk는 Vd의 직접 선행 작업으로서 Vd를 위하여 복제된 노
     드 */
```

(그림 3) DFRN 스케줄링 알고리즘

(그림 3)의 DFRN 알고리즘은 포괄적으로 되어 있어서, 노드 선택 알고리즘으로 리스트 스케줄링에서 사용하는 어떠한 우선순위 부여 방법도 선택하여 사용할 수 있다. 여기서 노드 선택 알고리즘이란 타스크 그래프상의 많은 노드들 중에서 어느 노드를 먼저 스케줄할 것인지를 결정하는 알고리즘을 말한다. 이 논문에서는 노드 선택의 우선순위를 결정하기 위하여 HNF(Heavy Node First)를 사용한다. 어떤 특정 노드 선택 알고리즘의 선택에 따른 영향은 [11]에 정리되어 있다.

스텝(1)에서 초기화() 서브루틴은 인접 행렬(adjacency matrix) 혹은 인접 리스트(adjacency list)의 형태로 되어있는 입력 타스크 그래프를 읽고, 각 타스크 노드의 레벨을 확인한다. 먼저 각 노드들은 HNF노드 선택의 원칙에 의하여 우선순위 큐(Priority Queue)에 자신의 레벨에 따라 오름차순으로 정렬되고, 같은 레벨의 노드들은 그 노드의 연산 비용에 따라 내림차순으로 정렬된다. 스텝(2)에서는 스텝(1)에서 정렬된 노드들 중에서 처음부터 하나씩 선택한다. 현재 선택된 노드 V_i는 조인노드일 수도 있고, 그렇지 않을 수도 있다. 조인노드가 아닐 경우에는 스텝(4)에서 스텝(9)까지가 수행이 된다. 스텝(4)에서 확인된 노드 V_i의 직접 선행 작업이 마지막 노드일 경우는 스텝(6)가 그 노드 V_i를 그 직접 선행 작업의 바로 뒤에 스케줄한다. 만약 직접 선행 작업이 마지막 노드가 아닐 경우는, 그 직접 선행 작업이 스케줄된 처리기에 있는 스케줄 중에서 첫 노드부터 그 직접 선행 작업까지의 스케줄을 사용되지 않은 처리기 P_u에 복제한 다음, 현재 선택된 노드 V_i를 그 처리기에 스케줄한다. 이렇게 함으로써, 노드 V_i의 시작 시간은 직접 선행 작업의 종료 시간이 된다. 타스크 복제를 하지 않을 경우에는, 노드V_i의 시작 시간은 직접 선행 작업과 그 처리기의 마지막 노드 사이의 연산 시간으로 인해 증가된다.

현재 선택된 노드 V_i가 조인노드인 경우에는, 스텝(12)에서 노드 V_i의 임계 선행 작업과 임계 처리기가 식별된다. 마지막 노드가 위에서 설명된 것과 같은 방법으로 처리되고, DFRN은 스텝(14) 혹은 스텝(17)에서 실행된다. DFRN(P_a, V_i) 서브루틴은 스텝(21)과 (22)에 나타난 것처럼 try_duplication(P_a, V_i)와 try_deletion (P_a, V_i)의 두 서브루틴으로 구성된다. try_duplication (P_a, V_i) 서브루틴은 가장 큰 메시지 도착 시간의 원인이 되는 직접 선행 작업을 찾아 복제를 시도하는데,

스텝(24)과 스텝(25)에서 노드 V_i에 대하여 처리기 P_a에 스케줄 되어 있는 선행 작업을 찾을 때까지 선행 작업들을 타스크 그래프에서 상향 탐색한다. 처리기 P_a에 스케줄 되어있는 선행 작업을 찾았을 때 스텝(27)에서 탐색을 멈추고 그때까지 탐색 된 선행 작업들을 P_a에 복제한다. 결과적으로 $V_k \rightarrow V_i$, $V_k \notin V[P_a]$를 만족하는 V_k가 P_a에 복제된다. 같은 과정이 다음으로 큰 메시지 도착 시간을 제공하는 직접 선행 작업에 대하여 반복되며, 결국 메시지 도착 시간의 내림차순으로 V_i의 모든 직접 선행 작업에 대하여 반복된다.

서브루틴 try_duplication(P_a, V_i)의 수행이 끝난 다음에 try_deletion(P_a, V_i) 서브루틴은 스텝(30)에 주어진 조건에 따라서 복제된 타스크들의 제거 여부를 결정한다. 첫번째 조건은 다른 처리기에서 수행이 완료된 같은 타스크의 출력이 메시지를 통하여 복제된 타스크의 출력보다 먼저 도착하는 경우로써, 이 경우에 복제된 타스크는 불필요하므로 제거된다. 두 번째 조건은 타스크 복제가 더 이상 EST(V_i, P_c)를 감소시키지 않는 경우이다. 타스크 복제 없이 V_i를 결정 선행 작업 CIP(V_i)와 같은 처리기에 스케줄 할 경우에 EST(V_i, P_c) = Max (ECT(CIP(V_i), P_c), MAT(DIP(V_i), V_i))가 되는데, 복제된 타스크 V_k의 수행 종료 시간이 이미 MAT(DIP(V_i), V_i))보다 크다면 그 타스크의 복제는 더 이상 V_i 의 수행 시작 시간을 감소시키지 않기 때문이다.

V개의 노드들을 가지는 입력 타스크 그래프에 대하여, 스텝(1)에서 그 노드들을 정렬하는데 사용하는 알고리즘에 따라서 $O(V^2)$ 혹은 $O(V\log_2 V)$의 시간이 요구된다. 이 시간은 HNF라는 노드 선택 알고리즘을 수행하는데 필요한 시간이다. 스텝(4)에서 스텝(9)까지 O(V)의 시간이 걸리고, 스텝(12)가 임계 선행 작업과 임계 처리기를 식별하는데 O(V) 의 시간이 요구된다. 서브루틴 try_duplication(P_a, V_i)은 메시지 도착 시간이 큰 순서로 복제를 하는데, for 문이 여러 번 반복되더라도 정렬될 전체 노드의 수는 V를 초과할 수가 없으므로, heap과 같은 자료구조를 사용할 경우에는 for 문의 실행 횟수에 관계없이 복잡도는 $O(V\log_2 V)$가 된다. 서브루틴 try_deletion(P_a, V_i)의 복잡도는 $O(V^2)$이므로 결국 서브루틴 DFRN (P_c ,V_i)의 수행 시간은 $O(V^3)$이 되고, DFRN(P_c,V_i)은 입력 타스크 그래프내의 조인노드의 수만큼 수행이 되므로, 조인노드의 수를 q이라고 하면($q \leq V$), 전체 알고리즘의 수행 시간은 $O(qV^2)$, 즉 $O(V^3)$이 된다.

5. 성능 비교

DFRN 알고리즘과 기존의 스케줄링 알고리즘의 성능 비교를 위하여 1000개의 무작위 타스크 그래프가 생성되었다. 타스크 스케줄링 알고리즘들간의 성능 비교에 영향을 줄 수 있는 타스크 그래프의 특성으로는 노드의 수, 통신 비용과 연산 비용의 비율(CCR: Communication to Computation Ratio), 그리고 평균 진입 차수를 들 수 있는데, 이들의 값은 응용에 따라 매우 다양하게 나타나므로[17], 본 논문에서는 이러한 특성들을 타스크 그래프 생성시에 파라미터로 사용하여, 이 값들의 변화에 따른 스케줄링 알고리즘들간의 성능 비교를 보인다.

노드의 수로는 20, 40, 60, 80, 100개의 5가지가 사용되었으며, CCR값으로는 0.1, 0.5, 1.0, 5.0, 10.0의 5가지가 사용되었다. CCR이란 평균 연산 비용에 대한 평균 통신 비용의 비율로서, CCR 값이 높을수록 통신 오버헤드가 큰 경우를 나타내고, 이는 타스크의 크기(granularity)가 작음을 의미한다[17]. 결과적으로 생성된 타스크 그래프를 평균 진입 차수에 따라 분류한 다음 그 평균 진입 차수에 따른 성능 변화 또한 관찰되었다. 5가지 노드의 수와 5가지 CCR값에 따른 25가지 경우에 대하여, 각 경우 당 40개의 타스크 그래프를 무작위로 생성, 결국 1000개의 타스크 그래프에 대하여 성능 비교가 이루어 졌다.

서로 다른 스케줄링 알고리즘들간의 성능 비교를 위하여 상대 병렬시간(RPT: Relative Parallel Time)[12]이라는 성능 평가 척도를 사용하였는데, 이는 CPLEC에 대한 병렬 수행 시간의 비율로 정의된다. 예를 들어, 어떤 스케줄링 알고리즘에 의하여 얻어진 병렬 수행 시간이 200이고 CPLEC가 100 이라면, RPT값은 2.0이 된다. CPLEC값은 병렬 수행 시간의 최소 한계(lower bound)이므로, RPT값은 1보다 작을 수 없고, RPT값이 클수록 상대적으로 긴 병렬 수행 시간을 의미한다.

성능 평가의 목적중의 하나는 서로 다른 스케줄링 알고리즘들간에 성능(얻어진 스케줄의 길이)과 수행 시간(스케줄을 작성하는데 걸리는 시간)과의 타협 관계(trade-off)를 관찰하는 것이다. 여기서 수행 시간은 Sun Sparc10 워크스테이션에서 스케줄을 작성할 경우

에, time 명령에 의해서 얻어진 user time을 사용하였다. 각 스케줄링 알고리즘이 스케줄 작성 시간은 <표 1>에 요약되어 있다.

〈표 1〉 수행 시간 비교 (단위 : 초)

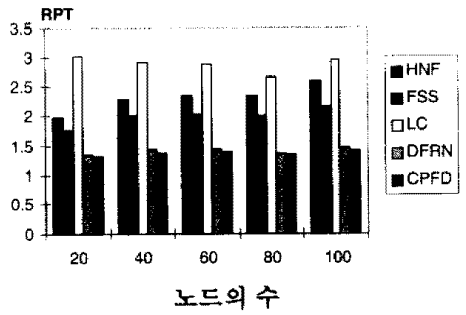
노드 수	HNF	FSS	LC	CPFD	DFRN
100	0.3	0.01	4.16	15.17	**0.48**
200	1.29	0.13	25.00	222.96	**3.23**
300	3.18	0.23	77.54	894.77	**8.24**
400	5.97	0.34	177.14	2782.56	**17.3**

<표 2>는 각 쌍의 스케줄링 알고리즘들 사이의 병렬 수행 시간의 비교 결과를 나타낸다. 표의 각 셀(cell)은 어떤 상수 a, b, c에 대하여 "> *a*, = *b*, <*c*"의 세가지 요소의 형태로 구성되어 있는데 그 의미는 1000번의 스케줄 작성에 대하여, 그 행에 표시된 스케줄링 알고리즘이 그 열에 표시된 스케줄링 알고리즘보다 긴 스케줄을 작성하는 경우가 *a*번, 같은 길이의 스케줄 *b*번, 짧은 스케줄을 작성하는 경우가 *c*번이라는 것이다. 예를 들어, DFRN알고리즘과 HNF 알고리즘과의 비교 결과는 6번째 행의 DFRN과 2번째 열의 HNF 알고리즘이 교차하는 부분에 표시되어 있다. 이 경우에 6번째 행, 2번째 열을 보면 "> 2, = 22, <976"으로 표시되어 있는데 이는 1000개의 무작위 입력 타스크 그래프에 대하여, DFRN 알고리즘이 HNF알고리즘보다 긴 스케줄을 작성하는 경우가 2번, 같은 길이의 스케줄 22번, 짧은 길이의 스케줄을 작성하는 경우가 976번이라는 의미이다. 이 비교는 HNF알고리즘에 DFRN 알고리즘을 적용하여 타스크 복제를 시도하는 경우에, 97.6%의 경우에 스케줄의 길이를 줄일 수 있다는 사실을 보여준다.

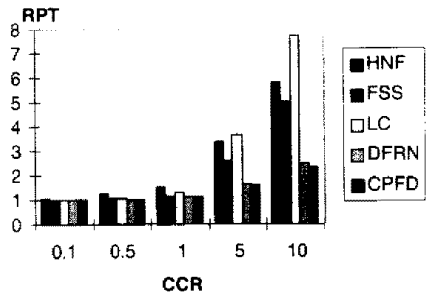
〈표 2〉 병렬 수행 시간의 비교

	HNF	FSS	LC	CPFD	DFRN
HNF	> 0 = 1000 <0	> 885 = 48 <67	> 587 = 39 <374	> 978 = 22 <0	**> 976** **= 22** **<2**
FSS	> 67 = 48 <885	> 0 = 1000 <0	> 27 = 165 <808	> 575 = 425 <0	**> 567** **= 430** **<3**
LC	> 374 = 39 <587	> 808 = 165 <27	> 0 = 1000 <0	> 829 = 171 <0	**> 829** **= 171** **<0**
CPFD	> 0 = 22 <978	> 0 = 425 <575	> 0 = 171 <829	> 0 = 1000 <0	**> 27** **= 685** **<288**
DFRN	**> 2** **= 22** **<976**	**> 3** **= 430** **<567**	**> 0** **= 171** **<829**	**> 288** **= 685** **<27**	**> 0** **= 1000** **<0**

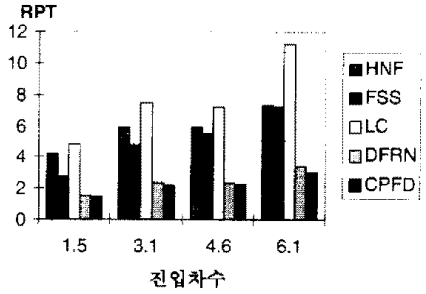
(그림 4, 5, 6)의 그래프들은 각각 입력 타스크 그래프를 구성하는 노드의 수, CCR 값, 평균 진입 차수에 대한 성능 비교를 보여준다. (그림 4)에서 보여주는 것과 같이 노드의 수는 스케줄링 알고리즘간의 상대적인 성능 비교에 커다란 영향을 끼치지 않는다. 즉, 성능 비교는 노드의 수에 상관없이 비슷한 형태(pattern)를 보인다. 이 그림에서 DFRN 알고리즘은 같거나 작은 복잡도를 가지는 다른 스케줄링 알고리즘들을 성능면에서 눈에 띄게 압도하며, 높은 복잡도를 가지는 CPFD 알고리즘에 대하여는 아주 작은 차이로 열등한 성능을 보임을 알 수 있다. (그림 5)에서 CCR값은 상대적인 성능 비교에서 아주 큰 영향을 미침을 보여준다. CCR 값이 증가할수록 성능의 차이는 더욱 커지고, CCR값이 아주 작을 경우에는 성능의 차이가 미미함을 볼 수 있다. (그림 6)에서는 평균 진입 차수의 변화가 성능 비교 결과의 패턴에 별다른 영향을 끼치지 않음을 보여준다.



(그림 4) 노드 수에 대한 성능 비교



(그림 5) CCR 값에 대한 성능 비교



(그림 6) 진입 차수에 대한 성능 비교

6. 결 론

본 논문은 전체 복제와 부분 복제 스케줄링 방법의 장점을 취합하고 단점은 보완한 새로운 방법의 복제 중심 스케줄링 알고리즘을 제안한다. 이 논문에서 제안된 알고리즘은 전체 복제 스케줄링 알고리즘들처럼 조인노드에 대하여 선행 작업들의 복제를 시도하나, 이 복제 과정에서 DFRN이라는 새로운 접근 방법을 사용하여 전체 복제 스케줄링 알고리즘이 필요로 하는 긴 재귀적 계산 과정을 단축시킴으로써, 많은 수의 타스크들로 이루어지는 큰 응용 프로그램에 적합한 스케줄 작성 시간을 제공한다.

1000개의 무작위 타스크 그래프들에 대한 성능 평가 결과는 DFRN 알고리즘이 같거나 작은 복잡도를 가지는 스케줄링 알고리즘들을 성능면에서 월등하게 능가하며, 더 큰 복잡도를 가지는 스케줄링 알고리즘에 대해서는 미미하게 열등한 성능을 가짐을 보여준다. 또한 성능 향상의 폭은 CCR값이 커질수록 더욱 커짐을 보여준다. 전체 복제 스케줄링 알고리즘들 중에서 가장 성능이 뛰어난 것으로 보고된 CPFD 스케줄러[12]와의 비교 결과는, DFRN알고리즘이 CPFD 알고리즘 수행 시간의 0.6%의 수행 시간으로 68.5%의 입력 타스크 그래프에 대하여 짧거나 같은 병렬 수행 시간을 제공함을 보여준다. 이러한 수행 시간의 차이는 노드의 수가 증가할수록 더욱 커진다는 점을 고려할 때, DFRN스케줄링 알고리즘은 많은 수의 타스크들로 이루어지는 커다란 응용 프로그램에 적합한 스케줄러가 될 수 있다.

참 고 문 헌

[1] 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.

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

[3] 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.

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

[5] 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.

[6] M.Y. Wu, A dedicated track on "Program Partitioning and Scheduling in Parallel and Distributed Systems," in the *Hawaii Int'l Conference on Systems Sciences*, Jan. 1994.

[7] T. Yang and A. Gerasoulis, A dedicated track on "Partitioning and Scheduling for Parallel and Distributed Computation," in the *Hawaii Int'l Conference on Systems Sciences*, Jan. 1995.

[8] 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.

[9] Gyung-Leen Park, B. Shirazi, and J. Marquis, "DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Systems," *International Parallel Processing Symposium*, pp.157-166, Geneva, Switzerland, April 1997.

[10] Gyung-Leen Park, B. Shirazi, and J. Marquis, "A Scalable Task Duplication Scheduling for Message Passing Systems," *International Conference on Parallel and Distributed Systems*, pp. 122-129, Barcelona, Spain, June 1997.

[11] Gyung-Leen Park, B. Shirazi, and J. Marquis, "Comparative Study of Static Scheduling with Task Duplication for Message Passing Multi-

computer Systems," *International Symposium on Solving Irregularly Structured Problems in Parallel*, pp.13-134, Paderborn, Germany, June 1997.

[12] I. Ahmad and Y. K. Kwok, "A New Approach to Scheduling Parallel Program Using Task Duplication," *Proc. of Int'l Conf. on Parallel Processing*, Vol.II, Aug. 1994, pp.47-51.

[13] H. Chen, B. Shirazi, and J. Marquis, "Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication," *Proc. of Int'l Conf. on Parallel and Distributed Systems*, Dec. 1993, pp.270-275.

[14] Y. C. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors," *Proc. of Supercomputing'92*, Nov. 1992, pp.512-521.

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

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

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

[18] 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.

[19] S. J. Kim and J. C. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," *Proc. of Int'l Conf. on Parallel Processing*, Vol.III, 1988, pp.1-8.

[20] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel tasks on an Unbounded Number of Processors," *IEEE Trans. On Parallel and Distributed Systems*, Vol.5, No.9, pp.951-967, Sep. 1994.

[21] Gyung-Leen Park, B. Shirazi, J. Marquis, and Hyunseung Choo, "Decisive Path Scheduling: A New List Scheduling Method," *International Conference on Parallel Processing*, pp.472-480, Chicago, USA, Aug. 1997.

[22] Hyunseung Choo, Hee Yong Youn, Gyung-Leen Park, and Behrooz Shirazi, "Efficient Processor Allocation Scheme for Multi Dimensional Interconnection Networks," *26th International Conference on Parallel Processing*, pp.114-117, Chicago, USA, Aug. 1997

박 경 린

e-mail : glpark@cheju.cheju.ac.kr

1986년 중앙 대학교 전자계산학과 졸업(학사)

1988년 중앙 대학교 전자계산학과 대학원(공학석사)

1992년 텍사스 주립대(알링턴) 전산공학과 대학원(공학석사)

1997년 텍사스 주립대(알링턴) 전산공학과 대학원(공학박사)

1998년~현재 제주대학교 자연과학대학 전산통계학과 전임강사

관심분야 : 분산/병렬 처리 시스템, 오류 허용 시스템, 성능 평가 등

추 현 승

e-mail : choo@yurim.skku.ac.kr

1988년 성균관대학교 이과대학 수학과 졸업(학사)

1990년 텍사스 주립대(달라스) 전자계산학과(공학석사)

1996년 텍사스 주립대(알링턴) 전산공학과(공학박사)

1997년~1998년 특허청 심사4국 컴퓨터심사담당관실 심사관

1998년~현재 성균관대학교 전기전자 및 컴퓨터공학부 전임강사

관심분야 : ATM, 병렬 및 분산 처리, 알고리즘 해석, 고속통신망 등