전역 이기종 환경에서의 정적 태스크 스케줄링의 비교 연구

김 정 환†

요 약

DAG(Directed Acyclic Graph) 기반의 스케줄링을 포함한 많은 스케줄링 문제들은 대부분 NP-Complete로 알려져 있으며, 따라서 휴리스틱에 기반한 많은 알고리즘 연구들이 진행되어 왔다. 이 중 HEFT와 CPOP은 이기종 환경에서 효과적인 알고리즘으로 알려져 있다. 본 논문의 이전 연구에서는 이기종 환경이 보다 현실성 있는 전역 네트워크로 구성된 경우에 대해 효과적인 3개의 알고리즘(CPOC, eCPOC, eCPOP)을 제안한 바 있다. 본 논문에서는 이들 총 5개의 알고리즘에서 사용하는 휴리스틱을 체계적으로 분석하고, 다양한 벤치마크를 사용한 실험을 통해 비교 분석하였다. 실험 결과 전역 이기종 환경에서 eCPOC이 가장 우수한 성능을 보여주었고, 또한 제안된 3개의 알고리즘에서 사용하는 휴리스틱들이 전역 이기종 환경에서 효과적임이 확인되었다.

키워드 : 태스크 스케줄링, 이기종, 휴리스틱, 전역 네트워크, DAG

Comparative Study on Static Task Scheduling Algorithms in Global Heterogeneous Environment

Junghwan Kim†

ABSTRACT

Most scheduling problems including DAG(Directed Acyclic Graph)-based are known to be NP-complete, so many heuristic-based scheduling algorithms have been researched. HEFT and CPOP are such algorithms which have been devised to be effective in heterogeneous environment. We proposed, in the previous research, three scheduling algorithms which are effective in realistic global heterogeneous environment: CPOC, eCPOPC and eCPOP. In this paper, the heuristics which are used in the above five algorithms will be systematically analyzed. Those algorithms will be also studied experimentally using various benchmarks. Experimental results show that the eCPOC generates better schedules than any other algorithms and the heuristics which are used in the proposed algorithms are effective in the global heterogeneous environment.

Key Words : Task Scheduling, Heterogeneous, Heuristic, Global Network, Directed Acyclic Graph

1. 서 론

태스크 병렬성은 병렬 컴퓨팅 분야에서 데이터 병렬성과 함께 병렬 프로그램의 성능 향상을 위한 중요한 요소로 인식되어 왔다[1-4]. 태스크 병렬성의 활용을 위해서는 응용 프로그램에 내재되어 있는 병렬성을 자동적으로 추출하거나 혹은 프로그래머에 의해 직접적으로 병렬성을 표시하는 것이 필요하다. 추출된 병렬성은 대개 그래프 형태로 표현되며, 대표적으로 DAG(Directed Acyclic Graph)가 많이 활용된다.

DAG를 구성하는 각각의 태스크들은 개별 컴퓨팅 노드(또는 프로세서)에 할당되고, 또 할당된 프로세서에서 적절한 순서에 맞춰 실행되는 것이 필요하다. 이러한 과정이 바로 태스크 스케줄링이며, 스케줄 길이를 최소화하고자 하는 이 스케줄링 문제는 극히 제한된 경우를 제외하고는 NP-Complete으로 알려져 있다[5, 6]. 특히, 이기종 태스크 스케줄링 문제는 이기종으로 구성된 클러스터 혹은 그리드와 같이 현실적으로 존재하는 많은 플랫폼에서 그 해결이 요구되지만, 다른 스케줄링 문제와 마찬가지로 NP-Complete이므로 서브 최적화 또는 휴리스틱에 기반한 방법들이 연구되어 왔다.

태스크 스케줄링은 스케줄링 시점에 따라 정적 또는 동적 기반의 방법들이 존재하는데, 본 논문에서는 기존의 정적 스케줄링 알고리즘 중 두 개 알고리즘과 본 논문의 이전 연구에서 제안한 3개의 알고리즘에 대해 비교 및 실험을 통한 분석을 시도하고자 한다. 즉, 비교 대상의 기존 알고리즘으로 HEFT와 CPOP[7], 제안된 알고리즘으로 eCPOP, CPOC, eCPOC[8] 등 총 5개의 알고리즘에 대해 분석한다.

HEFT와 CPOP을 비교 대상으로 선택한 이유는 다음과 같다. HEFT는 매우 단순한 두 개의 휴리스틱에 기반을 두고 있지만, 좋은 스케줄 결과를 보여준다. 따라서 HEFT보다 좋은 성능을 보여주는 알고리즘을 개발하는 것이 하나의 연구

† 종신회원 : 건국대학교 자연과학대학 컴퓨터·응용과학부 조교수
논문접수 : 2006년 2월 28일, 심사완료 : 2006년 4월 6일

목표가 될 수 있다. 그러나, HEFT는 실험 환경이 두 가지 관점에서 보다 현실성을 반영할 필요가 있다. 첫째, 생성된 태스크 수행 시간과 프로세서의 성능은 완전히 독립적인 문제를 안고 있다. 둘째, 프로세서와 프로세서 사이의 통신 시간 또한 완전히 무작위적, 독립적이며 또한 비 대칭적이다. 현실세계에서는 어떤 프로세서 그룹은 매우 근접해 있어 통신 시간 또한 상대적으로 적게 소요될 수 있으며, 대칭적이다.

제안된 알고리즘은 보다 현실성 있는 환경을 고려하여 도출되었다. 예를 들어 그리드와 같은 환경에서는 어떤 프로세서 그룹은 근접해 있는 반면, 프로세서 그룹들 간에는 매우 떨어져 있는 전역 네트워크에 의해 연결되어 있을 수 있다. 또한 프로세서들이 이기종이기 때문에 태스크들의 수행 시간은 프로세서에 따라 달라지지만, 이것이 프로세서 자체의 성능 차이와 완전히 무관하지 않다.

한편, 일반적으로 임계 경로(critical path)가 스케줄 길이를 결정하므로 임계 경로에 대한 고려가 중요한 역할을 할 것으로 생각된다. 그러나, 임계 경로를 고려하는 CPOP의 스케줄 결과는 HEFT의 비해 만족스럽지 못하다. 이러한 문제의식으로부터 CPOP 알고리즘에서 사용하는 휴리스틱을 분석하게 되었으며, 이를 토대로 새로운 휴리스틱을 첨가한 알고리즘을 제안하였다. 제안된 알고리즘은 임계 경로를 고려하되, 앞서 언급한 현실적인 환경에 적합한 방법을 사용함으로써 HEFT보다 좋은 스케줄 결과를 생성한다.

이 논문에서 알고리즘들의 비교 분석은 크게 두 가지 방식을 사용한다. 먼저 각 알고리즘들에서 내재적으로 사용되고 있는 몇 가지 휴리스틱들을 체계적으로 구분 및 분석할 것이고, 다음으로 벤치마크로부터 추출된 태스크 그래프를 활용, 성능 비교 및 평가 결과를 제시할 것이다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 먼저, 2절 관련 연구에서는 기존 태스크 스케줄링 알고리즘 연구들에 대해 소개한다. 3절에서는 알고리즘 비교 분석에 사용될 휴리스틱들에 대해 체계적으로 기술하고, 4절에서는 기존 2개와 제안된 3개, 총 5개의 알고리즘들에 대해 사용된 휴리스틱 관점에서 분석 비교한다. 5절에서는 실험 방법을 기술하고, 실험 결과를 토대로 알고리즘들에 대해 비교한다. 마지막으로 6절에서는 연구 결과에 대한 요약 및 결론을 제시한다.

2. 관련 연구

정적 태스크 스케줄링 알고리즘은 몇 가지 부류로 구분되는데, 이는 TDB(Task Duplication-Based)[9], UNC(Unbounded Number of Clusters)[10-12], BNP(Bounded Number of Processors)[13], APN(Arbitrary Processors Network)[14, 15] 등과 같다.

TDB는 하나의 태스크를 여러 프로세서에 중복 할당 함으로써, 태스크들 사이의 통신 시간을 줄이는 휴리스틱에 근거를 둔 알고리즘들이다. TDB가 아닌 알고리즘은 다시 UNC와 BNP로 분류될 수 있다. UNC는 프로세서 개수의 제한이 없는 것을 가정하고 진행하는 알고리즘을, BNP는 프로세서 개수의 제한을 가정한 알고리즘을 말한다.

UNC 부류의 알고리즘은 클러스터링이라는 기법을 사용하는데, 이는 클러스터(태스크들의 클러스터)를 단계적으로 병합해 나가는 방법이다. 병합은 전체 종료 시간의 단축을 기대할 수 있는 동안 계속 단계적으로 진행된다. 병합이 종료된 후 클러스터 개수의 제한은 없지만, 실제로는 클러스터의 개수가 실제 프로세서의 개수보다 많을 수 있으므로, 클러스터를 프로세서에 적절히 매핑하는 과정이 추가적으로 필요하다.

한편, BNP가 프로세서 간 통신에 있어 충돌이 없다고 가정하는데 반해, APN은 통신 선로에서의 충돌을 전제한다. 따라서 APN에서는 태스크의 스케줄링 이외에 통신 선로에서의 메시지 스케줄링을 포함한다.

본 연구는 BNP에 부류에 속하면서 이기종 환경을 고려한 기존의 HEFT 및 CPOP 알고리즘과 새로 제안한 CPOC, eCPOC, eCPOP 등 3개의 알고리즘을 비교한다. HEFT 및 CPOP이 단순 이기종 형태의 네트워크 환경을 가정한데 반해, 제안된 알고리즘들은 프로세서들이 클러스터 형태로 군집된 형태가 다시 전역 네트워크를 구성하는 환경을 가정한다.

한편, 이들 알고리즘은 모두 리스트 스케줄링 방법을 사용한다. 리스트 스케줄링은 먼저전위자(predecessor)가 없는 태스크(시작 태스크)를 준비 큐(ready queue)에 넣고 시작된다. 다음으로 준비 큐에 있는 태스크들 중 우선 순위가 가장 높은 태스크를 추출하여 이 태스크를 위한 가장 적합한 프로세서를 선택한다. 이때 알고리즘에 따라 우선 순위를 결정하는 방법과 프로세서를 결정하는 방법이 달라진다.

3. 휴리스틱

하나의 알고리즘에는 여러 개의 휴리스틱이 단계별로 사용될 수 있으며, 또한 알고리즘들 사이에도 단계별로 공통적 혹은 서로 다른 휴리스틱을 사용할 수 있다. 본 논문에서 비교하는 5개의 스케줄링 알고리즘은 모두 리스트 스케줄링 방법에 기반을 두고 있다. 따라서, 태스크 우선순위를 결정하는 단계와 태스크를 프로세서에 할당하는 단계로 구분하여 각 단계별로 사용되는 휴리스틱들을 본 논문에서는 다음과 같이 분류하여 보았다.

3.1 태스크 우선순위 결정 휴리스틱

태스크 우선순위 결정(task prioritizing)은 리스트 스케줄링의 첫 단계로서, 준비 큐에 있는 태스크들 중 스케줄 대상이 되는 태스크를 선택하는 우선순위를 부여하는 일이다. 태스크 우선순위 결정을 위해 다양한 값들을 이용하는 연구들이 진행되어 왔으나, 여기서는 다음 두 식을 소개하기로 한다.

먼저 $rank_u(i)$는 태스크 i의 상향 rank로서, 이는 현재 태

$$rank_u(i) = \overline{w}_i + \max_{j \in succ(i)} (\overline{c}_{ij} + rank_u(j)) \quad (1)$$

$$rank_d(i) = \max_{j \in pred(i)} (rank_d(j) + \overline{w}_j + \overline{c}_{ji}) \quad (2)$$

스크 i로부터 마지막 태스크에 이르기까지 남은 최대 소요 시간을 평가한 것이다. 상향 rank라는 말은 제일 밑바닥, 즉 마지막 태스크로부터 거슬러 올라가면서 계산하기 때문에 명명된 것이다. 여기서 $succ(i)$는 태스크 i의 바로 다음 후위자(successor)를 의미한다. 또한 $\overline{c}_{ij}$는 edge (i, j)의 평균 통신 시간을, $\overline{w}_i$는 태스크 i의 평균 수행 시간을 각각 의미한다.

다음으로 하향 rank, 즉 $rank_d(i)$는 시작 태스크로부터 태스크 i직전에 이르기까지의 소요 시간을 의미한다. 또한 $pred(i)$는 태스크 i 바로 직전의 전위자(predecessor)를 의미한다.

하향 rank와 상향 rank는 각각 *t-level* (top level)과 *b-level* (bottom level)에 견주어질 수 있다. 이제 본 논문에서 알고리즘 비교를 위해 사용될 태스크 우선순위 결정 휴리스틱(Task prioritizing Heuristic)을 정리하면 다음 두 가지와 같다.

- TH1: $rank_u(i)$ 값이 높은 태스크를 먼저 스케줄링.
- TH2: $rank_u(i) + rank_d(i)$ 값이 높은 태스크를 먼저 스케줄링.

이 두 개의 휴리스틱은 [7]에서 제안된 HEFT와 CPOP 알고리즘에서 각각 사용되고 있다.

3.2 프로세서 할당 휴리스틱

태스크 우선순위 결정 단계에서 각 태스크의 우선 순위를 부여하고 나면, 다음 단계로 우선 순위가 가장 높은 태스크를 선택하여 프로세서에 할당하게 된다. 이때 가장 적합한 프로세서를 선택하고자 하는 것이 프로세서 할당 휴리스틱이다. 본 논문에 소개된 5개의 알고리즘은 다음 4가지 종류의 프로세서 할당 휴리스틱(Processor selection Heuristic)들의 하나 또는 그 이상의 복합적인 형태라고 할 수 있다.

- PH1: EFT(Earliest Finish Time)을 보장하는 프로세서에 할당.
- PH2: 모든 CPTs(Critical-Path Tasks)는 한 개의 최적 프로세서에 할당.
- PH3: 모든 CPTs(Critical-Path Tasks)는 한 개의 최적 클러스터에 할당.
- PH4: IPCPT(Immediate Predecessor of CPT)는 자신의 후위자인 CPT의 EST(Earliest Start Time)을 보장하는 프로세서에 할당.

위 4개의 휴리스틱 중 PH1과 PH2는 [7]의 HEFT와 CPOP 알고리즘에서 각각 사용되는 것이며, PH3와 PH4는 본 연구에서 추가적으로 개발된 휴리스틱이다.

프로세서들은 이기종이므로 동일 태스크라 할지라도 어느 프로세서에 할당하느냐에 따라 수행 시간이 달라질 수 있다. PH1은 태스크의 종료시간이 가장 빠른 시간, 즉 EFT를 나타내는 프로세서에 할당하는 휴리스틱이다. 프로세서의 개수가 p개이면, 각 프로세서에 태스크를 할당해봄으로써 가장 종료시간이 빠른 프로세서를 찾을 수 있으므로 여기에 소요되는 시간은 $O(p)$라고 할 수 있다. 따라서 전체 태스크 개수가 n이면 이 휴리스틱의 총 소요시간은 $O(n \cdot p)$가 된다. 이 휴리스틱은 간단하면서도 개연성이 높은 믿음을 반영하고 있다. 그러나, 개별 태스크의 종료 시간을 단축시키려는 시도가 전체 스케줄 길이의 단축에 반드시 유리하다는 보장은 없다.

CP(Critical Path), 즉 임계 경로는 스케줄 길이를 결정한다는 점에서 중요한 요소로 생각된다. 물론 태스크의 수행 시간은 할당된 프로세서에 따라 달라지므로 실제 CP는 모든 스케줄이 완료된 상태에서 얻어질 수 있다. 여기서는 태스크 수행 시간과 태스크 간의 통신 시간을 각 프로세서에 할당된 경우에 대해 평균 값으로 계산하고, 이를 토대로 CP를 계산한다. $rank_u(i) + rank_d(i)$ 값이 가장 큰 task i들이 CP를 구성하는 태스크, 즉 CPT(Critical-Path Task)가 된다. PH2는 CPT를 가급적 빨리 끝내는 것이 전체 스케줄 길이를 단축하는데 유리하다고 보고, CPTs의 수행 시간 합이 가장 작은 프로세서에 할당한다. 이는 CPT 간의 통신 시간을 0으로 한다는(edge-zeroing) 이점도 갖고 있다. CPTs가 이미 결정되어 있다고 할 때, 이를 위한 최적 프로세서를 결정하는데 소요되는 시간은 $O(p \cdot |CPTs|)$이다. 여기서 $|CPTs|$는 CPTs의 개수를, p는 프로세서의 개수를 의미한다. 태스크의 수행 시간이 태스크x프로세서 행렬로 주어진다고 했을 때, 특정 프로세서의 CPTs의 수행 시간 계산은 $O(|CPTs|)$의 시간이 소요되기 때문에 위와 같이 전체 소요시간이 계산된다. CP가 여러 경로를 포함하거나 또는 CPTs의 개수가 많은 경우 등 병렬성을 내포하고 있을 경우, 단일 프로세서만으로 수행하는 PH2는 스케줄 길이를 오히려 늘일 수 있다.

휴리스틱 PH2가 단일 프로세서에 모든 CPTs를 할당하는데 비해, 휴리스틱 PH3는 CPTs를 프로세서의 집단, 즉 클러스터에 할당한다. 이는 앞서 언급한 CPTs들의 병렬성을 활용하지 못하는 단점을 보완한 것이다. 다만, PH2에서는 CPTs의 단일 프로세서 할당이므로 이들 간의 통신 시간이 0이었으나, PH3는 그렇지 못하다. 만일 클러스터 내의 프로세서들이 충분히 물리적으로 인접해 있거나 또는 빠른 네트워크로 연결되어 있다면, 이러한 통신 시간의 영향은 최소화할 수 있을 것이다. 전체 프로세서들을 서로 간의 통신 지연시간(latency)에 기반하여 클러스터들로 이미 구분하였다고 하고, CPTs도 이미 결정되어 있다고 할 때, PH3의 소요 시간은 $O(c p_c |CPTs|)$와 같다. 여기서 p_c는 클러스터 1개를 구성하는 프로세서의 최대 개수, c는 클러스터의 개수를 의미한다. 물론 $c < p$이고 $p_c < p$ 임이 예상된다. $p_c \approx p / c$라고 가정할 때, PH3의 소요시간은 $O(p \cdot |CPTs|)$로 PH2와 동일하다고 할 수 있다.

CP가 전체 스케줄 길이를 결정하는 중요한 요소라고 한다면, CPTs와 다른 태스크들을 동일한 관점에서 프로세서에 할당하는 것 보다는 CPTs를 보다 우선적으로 배려하는 휴리스틱이 가능하다. 즉, CPTs가 아닌 태스크들은 자신의 종료 시간을 단축시키는데 초점을 두고 프로세서를 선택하기 보다는 CPTs의 종료 시간을 단축시키는데 초점을 두는 것이 바람직하다. 휴리스틱 PH4는 이러한 관점에서 제안된 것이다.

CPTs는 특정 프로세서 혹은 특정 클러스터에 할당되어 수행 시간이 어느 정도 예측 가능하므로, 종료 시간 단축과 시작 시간 단축은 동일한 의미로 간주할 수 있다. 한편, CPT의 바로 직접적인 전위자, 즉 IPCPT(Immediate Predecessor of CPT)가 아니면 CPT의 시작 시간 계산은 불가능하므로, 대상은 CPT의 직접적인 전위자로 국한한다. 즉, IPCPT의 경우 자신의 종료 시간이 가장 빠른 프로세서를 선택하는 것이 아니라, 자신의 후위자인 CPT의 시작 시간을 가장 빠르게 할 것으로 예상되는 프로세서에 할당된다. 자신의 후위자가 CPT인지 여부 혹은 후위자 중 가장 상위 레벨의 CPT를 찾는데 소요되는 시간은 $O(d)$이고, 여기서 d는 후위자의 개수(degree)를 의미한다. 따라서 PH4의 소요 시간은 태스크를 각 프로세서에 할당해보고 후위 CPT의 시작 시간을 조사해 보는데 걸리는 시간이므로 $O(d \cdot p)$가 된다.

4. 알고리즘 비교

본 논문에서의 분석하고자 하는 알고리즘은 기존 2개와 제안한 3개 알고리즘 등 총 5개의 알고리즘으로서, 앞서 언급한 태스크 우선 순위 결정 휴리스틱(TH)과 프로세서 할당 휴리스틱(PH)들을 복합적으로 사용하고 있다. 이 절에서는 사용하고 있는 휴리스틱 관점에서 각 알고리즘을 비교 분석한다.

4.1 비교 대상 알고리즘

이 절에서는 Topcuoglu 등에 의해 제안된 두 개 알고리즘, HEFT와 CPOP[7]을 분석한다.

4.1.1 HEFT(Heterogeneous Earliest Finish Time)

HEFT 알고리즘은 각 단계에서 다음 휴리스틱을 사용한다.

- 태스크 우선 순위 결정 단계: TH1
- 프로세서 할당 단계: PH1

태스크 i의 우선 순위는 TH1에 의해 결정된다. 즉 $rank_u(i)$가 높은 값일수록 중요한 태스크로 간주하여 먼저 스케줄링함을 의미한다. $rank_u(i)$는 태스크 i로부터 마지막 종료 태스크에 이르기까지 남아 있는 평균 소요시간을 의미하므로, 이 값이 높을수록 우선적으로 스케줄링하는 것은 직관적으로 타당한 휴리스틱이다.

TH1에 의해 가장 우선 순위가 높은 태스크가 선택되면, 그 태스크가 수행될 프로세서를 PH1에 의해 결정한다. 즉, 태스크의 종료 시간이 가장 빠른 프로세서를 선택한다. 이는 우선 순위가 높은 태스크일수록 빨리 종료하도록 하는 휴리스틱으로써 직관적으로 받아들일 수 있는 방법이다.

HEFT는 간단한 두 개의 휴리스틱을 사용함으로써 기존의 다른 알고리즘에 비해 스케줄링 수행 부담이 적다. 그럼에도 불구하고 이 두 휴리스틱을 사용한 HEFT는 기존의 다른 알고리즘에 비해 매우 효과적인 것으로 나와 있다[7]. 하지만, 이 알고리즘 역시 최적의 스케줄을 생성할 수는 없다. 이를 사례별로 직접 검토한 결과 스케줄 길이를 보다 단축시키지 못하는 데에는 PH1의 문제에 일정 부분 기인한 것으로 생각된다. 현재의 태스크에 대해 가장 빨리 종료할 수 있는 프로세서를 선택하는 것은 현 시점까지의 종료 시간을 가장 단축시킬 수 있는 방법이긴 하지만, 그렇다고 해서 전체 스케줄 길이가 단축된다는 보장은 없다. 즉, 이것은 greedy 방법과 유사하게 지역적으로 최적인 해에 빠질 수 있다. 전체 스케줄 길이를 단축하기 위해서는 경우에 따라 현재 태스크의 종료 시간이 늦춰지는 것이 유리할 수도 있다.

4.1.2 CPOP(Critical-Path-On-a-Processor)

CPOP 알고리즘의 각 단계에서 사용하는 휴리스틱은 다음과 같다.

- 태스크 우선 순위 결정 단계: TH2
- 프로세서 할당 단계: PH2 + PH1

태스크 i의 우선 순위는 TH2에 의해 결정된다. 즉 $rank_u(i) + rank_d(i)$가 높은 값일수록 중요한 태스크로 간주하여 먼저 스케줄링한다. 이 값은 시작 태스크로부터 현재 태스크 i 직전까지의 평균 소요시간과 다시 태스크 i로부터 마지막 종료 태스크에 이르기까지의 평균 시간의 합을 의미한다. 즉, 시작 태스크에서 종료 태스크에 이르는 경로 중 태스크 i를 관통하는 경로의 평균 길이를 나타낸다. 따라서 이 값이 가장 큰 태스크들은 CPT, 즉 임계 경로에 있는 태스크들이라고 할 수 있다. CPOP은 CPTs에 가장 높은 우선 순위를 부여하고, 다음으로 경로 길이가 긴 경로에 속해 있는 태스크들에 대해 차례대로 높은 우선 순위를 부여한다.

프로세서 할당 단계는 두 가지 휴리스틱이 복합적으로 사용된다. 먼저, TH2에 의해 가장 우선 순위가 높은 태스크, 즉 임계 경로에 있는 태스크(CPTs)가 선택되면 이들 태스크는 최적의 한 개 프로세서에 할당한다(PH2). 이때 CPTs를 위한 프로세서만이 단지 결정되며, 각 태스크의 실제 수행 시작 시간은 결정될 수 없다. 다음으로 리스트 스케줄링 방식에 따라 시작 태스크로부터 시작하여 준비 큐에 있는 태스크들 중 가장 우선 순위가 높은 태스크를 차례대로 선택하되, 선택한 태스크가 CPT라면 앞서 언급한 PH2에 의해 결정된 프로세서에 할당하고, 그렇지 않다면 PH1에 의해 프로세서를 결정한다.

CPOP은 CPTs를 신속히 처리하는 방법으로서 한 개의 프로세서에 모두 할당함으로써 통신 시간을 제거하는 휴리스틱(PH2)를 사용하고 있다. 임계 경로는 전체 스케줄 길이를 결정할 수 있는 중요한 요소이므로 이는 직관적인 타당성을 갖고 있다. 그러나, 문헌에 나온 실험 결과는 CPOP이 HEFT에 비해 만족스럽지 못한 결과를 보여주고 있으며[7], 본 연구에서 시행한 실험에서도 역시 마찬가지였다. 이는 모든 CPTs를 하나의 프로세서에 할당하는 PH2가 통신 시간은 없앨 수 있지만 대신 CPTs의 수행을 직렬화함으로써, 병렬 수행을 통한 효과를 없애는 단점을 갖고 있기 때문인 것으로 생각된다.

4.2 제안 알고리즘

직관적으로 임계 경로는 전체 스케줄에 있어 중요한 요소

로 생각되므로 본 논문에서는 임계 경로에 있는 태스크들, 즉 CPTs에 관련된 휴리스틱을 추가적으로 고안함으로써 몇 가지 알고리즘들을 제시한다. 이는 CPOP의 단점인 CPTs의 직렬화 문제의 해결과 프로세서 할당 단계에서 CPTs들에 대해 보다 우선적인 고려를 포함한다.

4.2.1 CPOC(Critical-Path-On-a-Processor)

CPOC 알고리즘의 각 단계에서 사용하는 휴리스틱은 다음과 같다.

- 태스크 우선 순위 결정 단계: TH2
- 프로세서 할당 단계: PH3 + PH1

CPOP과 다른 점은 PH2 대신 PH3를 사용하고 있는 점이다. 즉, CPTs를 단지 한 개의 프로세서에 모두 할당함으로써 CPTs의 직렬화를 야기하는 CPOP의 단점을 보완하였다. PH3에 의해 CPTs는 한 개의 프로세서가 아니라 한 개의 클러스터에 할당되며, 이를 통해 낮은 통신 시간과 증가된 병렬성을 추구할 수 있다. 여기에는 클러스터 밖의 전역 네트워크에서의 통신 시간에 비해 클러스터 내의 프로세서 간의 통신 시간이 상대적으로 적다는 가정을 하고 있다.

4.2.2 eCPOC(enhanced Critical-Path-On-a-Cluster)

eCPOC 알고리즘의 각 단계에서 사용하는 휴리스틱은 다음과 같다.

- 태스크 우선 순위 결정 단계: TH2
- 프로세서 할당 단계: PH3 + PH1 + PH4

eCPOC은 CPOC 알고리즘에 PH4를 추가하였다. 이는 앞서 언급하였듯이 개별 태스크의 종료 시간의 단축을 목표(PH1)로 하는 것이 아니라, 그 태스크의 직접 후위자가 CPT인 경우 해당 CPT의 시작 시간을 앞당기는 것을 목표로 하여 프로세서를 선택하는 휴리스틱(PH4)이다. 어떤 태스크의 직접 후위자가 CPT가 아니라면, 기존의 PH1 휴리스틱을 적용한다.

4.2.3 eCPOP(enhanced Critical-Path-On-a-Processor)

eCPOP 알고리즘의 각 단계에서 사용하는 휴리스틱은 다음과 같다.

- 태스크 우선 순위 결정 단계: TH2
- 프로세서 할당 단계: PH2 + PH1 + PH4

eCPOP은 CPOP 알고리즘에 PH4를 추가하였다. 제안된 CPOC에 PH4를 추가하여 eCPOC알고리즘을 제안하였듯이, 기존에 나온 CPOP 알고리즘에 PH4를 추가하여 이 휴리스틱의 효과를 살펴보기 위함이다. 다른 휴리스틱은 CPOP 알고리즘과 동일하다.

5. 실험 및 평가

이 절에서는 몇 가지 벤치마크 프로그램들로부터 생성된 태스크 그래프를 사용하여 수행한 실험과 알고리즘별 결과에 대한 비교 분석을 기술한다.

5.1 실험 환경 생성

분석할 알고리즘은 모두 이기종 프로세서 환경에서 수행되는 태스크 그래프의 스케줄링을 목표로 하고 있으므로, 실험을 위한 적절한 이기종 환경을 생성하는 것이 필요하다.

기존 알고리즘인 HEFT와 CPOP은 모두 태스크 그래프 생성 시 이러한 환경을 생성하여 실험하고 있다[7]. 개별 태스크는 프로세서마다 수행 시간이 다르게 주어지는데, 이때 주어지는 시간은 균일 분포(uniform distribution)를 따르는 무작위적인 시간이다. 또한 태스크와 태스크 사이의 통신 시간과 태스크 자체의 수행 시간 비율을 나타내는 척도, CCR(Communication to Computation Ratio)이 매개변수로 주어지면, 이 값을 평균으로 하여 적절한 통신 시간을 생성한다. 이를 정리하면 다음과 같다.

- β : 프로세서의 이질성 척도. 이 값이 커지면 무작위적으로 발생되는 태스크 수행 시간의 범위가 따라서 커진다.
- CCR : 통신 시간과 태스크 수행 시간의 비율. $CCR >> 1$ 이면 통신 시간이 차지하는 비율이 계산 시간에 비해 상대적으로 높다는 것을 의미한다.

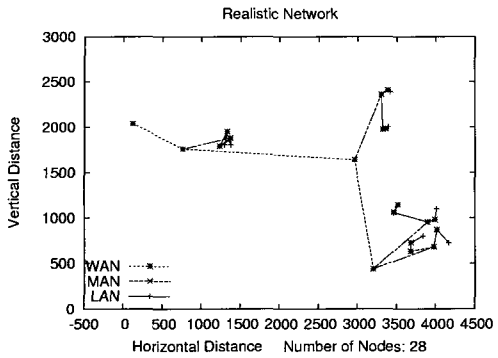
그런데, 여기서 β 값은 실제로는 프로세서의 상대적 이질성을 나타내는 것이 아니다. 왜냐하면, β 는 매 태스크 수행 시간을 생성할 때마다 무작위 분포의 범위를 정하기 위해서 사용되기 때문이다. 즉, β 는 프로세서의 성능을 생성하기 위해 사용되는 것이 아니라 태스크의 수행 시간을 생성하기 위해 사용된다. 가령, 태스크 A가 프로세서 P1보다 프로세서 P2에서 빠르게 수행될지라도, 태스크 B의 경우에서는 반대가 될 수 있기 때문이다.

이러한 문제를 해결하기 위해 본 논문에서는 새로운 매개변수, γ 를 추가하였으며, β 의 의미는 프로세서 속도의 상대적 이질성을 실질적으로 나타내도록 하였다.

- β : 프로세서 속도의 상대적 이질성 척도. 이 값은 프로세서의 상대적 속도를 생성하기 위해 사용된다.
- γ : 프로세서의 구조적 이질성 척도. 태스크별로 특정 프로세서에 따라 성능 차이가 날 수 있음을 반영하기 위한 매개변수이다. 이 값이 커지면 프로세서에 따라 태스크의 수행 시간 차이가 커진다.

가령, 어떤 프로세서는 벡터처리 기능이 있어 특정 태스크(루프 구조 등)를 보다 빠르게 처리할 수 있는 반면, 다른 종류의 태스크에 대해서는 처리 속도가 느릴 수 있다. 이러한 점의 이질성을 나타내는 척도가 γ 이다.

한편, 기존 HEFT와 CPOP에서는 프로세서와 프로세서의 통신 시간을 단순히 무작위적으로 생성하였는데, 이는 두 프로세서 사이의 통신 시간이 비대칭적이고 비현실적인 문제를 안고 있다. 예를 들어 프로세서 P1과 P2, P2와 P3 사이의 지연시간이 각각 10 us일 때, P1-P2-P3의 경로가 존재한다면, 프로세서 P1과 P3 사이의 지연시간은 20 us로 예상할 수 있



(그림 1) 생성된 전역 네트워크의 예

다. 그러나, 각 프로세서 쌍에 대해 무작위적으로 지연시간을 생성할 경우 P1과 P3 사이에 비현실적인 값, 가령 200 us이 부여되는 상황이 생길 수 있다.

따라서 본 연구에서는 먼저 프로세서들을 연결하는 네트워크를 생성하고, 이 네트워크 하에서 프로세서 사이의 경로와 거리 등을 고려한 지연시간과 대역폭을 생성하도록 하였다. 네트워크 그래프의 생성은 [16]의 도구를 사용하였다. 그림 1은 이 도구를 사용하였을 때 생성된 네트워크 그래프의 예를 보여준다. 이는 프로세서들이 군집을 이루어 여러 개의 클러스터를 형성한 형태로 전역 네트워크이다. 본 실험에서는 그림 1과 같이 다수의 클러스터가 모여 전역 네트워크를 구성하는 환경을 대상으로 한다. 이는 그리드와 같은 컴퓨팅 환경을 잘 반영한다고 생각된다. 실험을 위해 8, 16, 24, 32, 40 노드로 각각 구성된 총 49 종류의 네트워크를 생성하여 사용하였다.

[16]의 도구에 의해 생성된 네트워크는 노드와 노드 사이의 거리 정보만이 주어진다. 본 실험에서는 자체 제작한 변환 프로그램을 통해 이러한 거리 정보를 지연시간 행렬과 대역폭 행렬로 변환한다. 이들 행렬은 노드와 노드 사이의 지연시간과 대역폭을 각각 나타낸다.

<표 1>은 네트워크 규모(WAN, MAN, LAN)별로 대역폭과 고정 지연시간, 그리고 단위 거리당 지연시간을 나타낸다. <표 1>과 같이 주어진 매개변수를 사용하여 생성된 네트워크의 모든 노드-노드 쌍에 대해 지연시간과 대역폭을 계산하여 행렬을 만든다. 이때, 노드와 노드 사이의 지연시간은 Floyd-Warshall 최단 경로 알고리즘을 활용하여, 경로 상의 지연 시간을 합산하였을 때 가장 작은 값으로 결정한다. 노드와 노드 사이의 대역폭도 유사한 방법을 사용하지만, 다른 점은 경로 상의 대역폭을 합산하는 것이 아니라 경로 상의 대역폭 중 가장 작은 값으로 결정하는 것이다.

〈표 1〉 대역폭 및 지연시간 변수

	bandwidth (MB/s)	fixed delay (ms)	delay / unit distance(ms/unit)
WAN-to-WAN	1000	15	0.1
WAN-to-MAN	100	100	0.01
MAN-to-MAN	100	10	0.1
MAN-to-LAN	100	5	0.01
LAN-to-LAN	1	1	0.01

5.2 벤치마크

실험에 사용된 벤치마크는 FFT, GE, MD, robot, sparse, fpppp, rand50 등 총 7종류이며, 기존 문헌과 연구들로부터 DAG 형태의 태스크 그래프를 인용하였다. FFT, GE, MD는 [7]에 제시된 DAG 그림을 수작업을 통해 본 실험을 위한 입력 형태로 변환하였으며, robot, sparse, fpppp, rand50은 [17, 18]에서 자체 포맷(STG, Standard Task Graph)으로 표현된 그래프를 본 실험에서 제작한 변환기를 사용하여 변환, 활용하였다.

FFT(Fast Fourier Transform)와 GE(Gaussian Elimination)는 정형화된 그래프이며, 문제 크기를 FFT의 경우 4, 8, 그리고 GE의 경우 5, 10을 사용하였다. MD(Molecular Dynamics)의 경우 불규칙적인 형태를 보이는 그래프로 문제 크기는 고정되어 있다. Robot(robot control), sparse(sparse matrix solver), fpppp(SPEC fpppp), rand50은 모두 STG 포맷인데, 이 중 rand50의 경우 특정 응용 프로그램으로부터 추출된 그래프가 아니라, 무작위적으로 생성된 그래프 집합이다.

위의 벤치마크에 대해서 앞서 언급한 β, γ, CCR 등의 매개변수를 적용하여 각 종류별로 625개, 총 5625개의 그래프를 생성하여 실험에 사용하였다.

5.3 성능 평가 척도

스케줄링 알고리즘들을 비교할 수 있는 평가하기 위한 다양한 척도들이 기존에 제시된 바 있다. 각 알고리즘간의 우열 비교, 최적 값을 생성한 개수 비교, 최적 값보다 덜 좋은 정도, 사용된 프로세서 개수, 알고리즘 수행 시간, 스케줄링의 확장성 등이 [19]에 제시되어 있다. 또한 [7]에는 정규화된 스케줄 길이, 속도 향상(speedup) 등이 제시되어 있다. 본 실험에서는 이 중 각 알고리즘간의 우열 비교와 정규화된 스케줄 길이(SLR, Schedule Length Ratio)의 비교를 활용하고자 한다.

각 알고리즘 간의 우열 비교는 모든 알고리즘 쌍에 대해 스케줄 길이가 더 나은 경우(짧은 경우)의 개수, 동일한 경우의 개수, 안 좋은 경우의 개수를 산출하여 비교한다. SLR은 태스크 그래프별로 스케줄 길이가 달라 절대 비교가 어려운 점을 고려하여, 스케줄 길이를 정규화한 값이다. SLR의 최적의 값은 1이며, 모든 알고리즘의 SLR값은 1보다 작을 수 없다.

5.4 성능 분석

<표 2>는 eCPOC, CPOC, eCPOP과 기존의 CPOP, HEFT에 대해 각 쌍 별로 성능을 비교한 것이다. 표의 각 항목 3개의 수치는 알고리즘끼리 서로 비교하였을 때, 스케줄 길이가 안 좋은 경우, 동일한 경우 나은 경우를 각각 나타낸다. 이 표는 robot, sparse, fpppp 등 총 3개의 벤치마크에 대해 앞서 언급한 다양한 네트워크 구성과 매개변수를 적용하여 실험한 결과를 합산한 것이다.

〈표 2〉 알고리즘별 비교

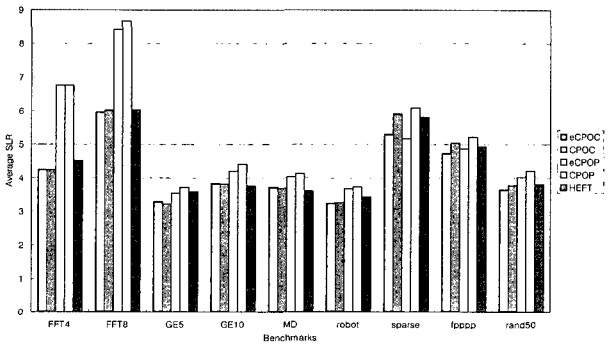
Algorithms		eCPOC	CPOC	eCPOP	CPOP	HEFT
eCPOC	<	0	3579	7092	3744	7288
	=	18375	3178	959	163	231
	>	0	11618	10324	14468	10856
CPOC	<		0	10958	6033	10069
	=		18375	69	1119	272
	>		0	7348	11223	8034
eCPOP	<			0	3193	8088
	=			18375	2771	39
	>			0	12411	10248
CPOP	<				0	11702
	=				18375	78
	>				0	6595
HEFT	<					0
	=					18375
	>					0

eCPOC의 경우 다른 모든 알고리즘과 비교했을 때 좋은 경우가 안 좋은 경우보다 많아 우수한 결과를 보였다. CPOP은 반대로 다른 모든 알고리즘에 대해 안 좋은 경우가 더 많았다. CPOC은 CPOP에 비해서는 좋으나, 나머지 3개에 대해서는 낮은 성능을 보였다. eCPOP의 경우 eCPOC에 대해서는 상대적으로 낮은 성능을 보이고, 나머지 알고리즘보다는 우수하였다. HEFT는 eCPOC과 eCPOP을 제외한 나머지에 대해 상대적으로 우수하였다. 순위를 정해보면, eCPOC, eCPOP, HEFT, CPOC, CPOP의 순서와 같다.

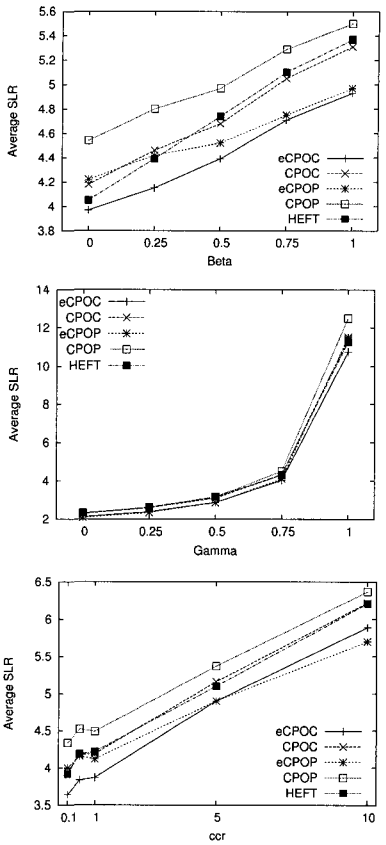
<표 2>의 각 알고리즘 쌍의 비교에서 스케줄 길이가 동일한 경우는 적고, 스케줄 길이가 더 좋거나 아니면 그렇지 않은 경우로 치우쳐 있음을 알 수 있다. 가령, eCPOC은 HEFT보다 더 좋은 경우가 10856번 나오지만, 안 좋은 경우도 7288번 나온다. 따라서 단순히 경우의 수만을 비교할 경우, 안 좋은 경우에서 실제로 얼마만큼 안 좋은 지 여부를 고려하지 않기 때문에 실제 알고리즘의 우수성을 언급하기 어려운 측면이 있다.

(그림 2)는 벤치마크별로 SLR 척도를 비교한 것이다. SLR 값은 스케줄 길이를 정규화한 것이므로 값이 작을수록 더 좋은 결과를 의미한다. 전반적으로 eCPOC이 좋은 성능을 보여주고, CPOP은 가장 안 좋은 성능을 나타냈다.

(그림 3)은 β, γ, *CCR*의 증가에 따른 SLR 값의 변화를 알



(그림 2) 벤치마크별 알고리즘 비교



(그림 3) β, γ, CCR의 증가에 따른 알고리즘 비교

고리즘별로 비교해 보여주고 있다. 이 결과는 robot, sparse, fpppp 3개의 벤치마크에 대한 실험을 합산하여 나타낸 것이다. β의 경우 값이 커짐에 따라, 즉 프로세서 속도의 이질성이 커짐에 따라 전반적으로 SLR 값이 증가하는 경향을 보여준다. 또 CPOP과 eCPOP의 경우 β값이 증가함에 따라 다른 알고리즘과의 성능 차이를 좁혀가는 경향을 보여준다. 이는 두 알고리즘이 CPTs를 한 개의 프로세서에 할당하는 PH2 휴리스틱을 사용하는데, 프로세서 속도의 이질성이 커지면 CPTs가 보다 성능이 좋은 프로세서에 할당될 가능성이 높아지기 때문인 것으로 분석된다. γ 의 경우 값이 커짐에 따라 모든 알고리즘에 있어 SLR 값이 급격히 커지는 양상을 보이고 있다. 이는 실질적으로 스케줄 길이가 증가하기보다는 SLR의 분모 부분이 급격히 감소하기 때문인 것으로 분석된다. 프로세서의 구조적 이질성을 나타내는 γ 값이 증가하면 각 태스크가 보다 적은 수행 시간에 종료될 수 있는 프로세서가 존재할 가능성이 높아지고, 이는 스케줄 길이의 하계(lower bound)를 나타내는 SLR의 분모를 작게 만들기 때문이다. *CCR* 역시 증가하면 모든 알고리즘들이 전반적으로 SLR 값이 증가한다. 이는 통신 시간의 비중이 높아지므로 스케줄 길이가 늘어날 것이라는 직관적인 예상과도 일치한다.

6. 결 론

본 연구에서는 이기종 환경에 적용할 수 있는 정적 태스

크 스케줄링 알고리즘들에 대한 비교 분석을 시도하였다. 기존에 고안되어 우수한 성능을 나타낸 HEFT 및 CPOP 알고리즘과, 이를 개선하여 새로 제안한 eCPOP, CPOC, eCPOC 등 총 5개의 알고리즘을 비교하였다.

비교 연구에서는 크게 두 가지 방식을 사용하였다. 첫째, 알고리즘을 단순 기술하거나 특징들을 열거하는 대신 알고리즘에서 사용되는 휴리스틱들을 체계적으로 분류하고, 이들 휴리스틱을 사용하는 경우의 효과에 대해 논하였다. 또한 각 휴리스틱의 시간 복잡도에 대해 분석하였다. 둘째, 벤치마크로부터 추출된 태스크 그래프를 통해 각 알고리즘들의 성능을 비교하였다.

휴리스틱 측면에서 살펴보면, CPOP에서 PH2가 사용되었던 것에 비해 CPOC은 이를 PH3로 대체함으로써 개선을 시도하였다. 또한 eCPOP과 eCPOC의 경우, 휴리스틱 PH4가 추가적으로 사용되었다. PH2는 임계 경로 상의 모든 태스크(CPTs)를 한 개의 프로세서에 할당하는 것에 반해, PH3는 한 개의 클러스터를 할당함으로써 병렬성을 추구하였다. 또한 PH1의 경우 프로세서 할당 단계에서 모든 태스크들을 단순히 동등하게 취급했던 것에 비해, PH4는 CPT의 시작 시간 단축을 우선적으로 고려하였다.

실험은 총 7종류의 벤치마크로부터 추출된 태스크 그래프에 대해 다양한 변수들을 적용하여 진행하였다. 또한 이기종 환경도 무작위적인 프로세서 연결로 구성된 네트워크가 아니라, 보다 현실성있는 전역 네트워크 환경을 생성하여 구성하였다. 실험 결과 PH3와 PH4가 효과적임을 알 수 있었으며, 이는 eCPOC이 가장 좋은 성능을 보이고 또한 CPOC 및 eCPOP도 비교적 우수한 성능을 나타내는 것으로 확인할 수 있었다. 알고리즘 간의 우열 비교 및 SLR 이외에 다른 성능 척도에 대한 측정과 휴리스틱들의 다른 조합에 대한 실험은 향후 연구 과제로 남는다.

참 고 문 헌

[1] Shankar Ramaswamy, Sachin Sapatnekar and Prithviraj Banerjee, "A Framework for Exploiting Task and Data Parallelism on Distributed Memory Multicomputers," IEEE Transaction on Parallel and Distributed Systems, Vol.8, No.11, pp.1098-1116, Nov., 1997.

[2] Henri E. Bal and Matthew Haines, "Approaches for Integrating Task and Data Parallelism," IEEE Concurrency, pp.74-84, July-Sept., 1998.

[3] C. Boeres and A. Lima, "Hybrid Task Scheduling: Integrating Static and Dynamic Heuristics," Proc. of the 15th Symp. On Computer Architecture and High Performance Computing (SBAC-PAD '03), 2003.

[4] O. Beaumont, A. Legrand and Y. Robert, "Scheduling Strategies for Mixed Data and Task Parallelism on Heterogeneous Clusters and Grids," Proc. of the 11th Euromicro Conf. on Parallel, Distributed and Network-Based Processing (Euro-PDP'03), 2003.

[5] H. El-Rewini and H. H. Ali, "Task Scheduling in Multiprocessing Systems", IEEE Computer, pp.27-37, Dec., 1995.

[6] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs," ACM Computing Surveys, Vol.31, No.4, pp.407-471, Dec., 1999.

[7] H. Topcuoglu, S. Hariri and M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," IEEE Trans. on Parallel and Distributed Systems, Vol.13, No.3, pp.260-274, March, 2002.

[8] Junghwan Kim, Jungkyu Rho, Jeong-Ook Lee and Myeong-Cheol Ko, "CPOC: Effective Static Task Scheduling for Grid Computing," Lecture Notes in Computer Science, 3726, pp.477-486, 2005.

[9] I. Ahmad and Y. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," IEEE Trans. on Parallel and Distributed Systems, Vol.9, No.9, pp.872-892, 1998.

[10] M. Wu and D. D. Gajski, "Hypertool: A Programming Aid for Message-passing Systems," IEEE Trans. on Parallel and Distributed Systems, Vol.1, No.3, pp.330-343, 1990.

[11] 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, 1994.

[12] Y. Kwok and I. Ahmad, "Dynamic Critical-path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessor," IEEE Trans. on Parallel and Distributed Systems, Vol.7, No.5, pp.506-521, 1996.

[13] J. Hwang, Y. Chow, F. D. Anger and C. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," SIAM J. Comput., Vol.18, No.2, pp.244-257, April, 1989.

[14] G. C. Sih and E. A. Lee, "A Compile-time Scheduling Heuristic for Interconnection-constrained Heterogeneous Processor Architectures," IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.2, pp.75-87, Feb., 1993.

[15] H. El-Rewini and H. H. Ali, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," Journal of Parallel and Distributed Computing, Vol.9, No.2, pp.138-153, 1990.

[16] M. B. Doar, "A Better Model for Generating Test Networks," IEEE Global Telecommunications Conference, Nov., 1996.

[17] Takao Tobita and Hironori Kasahara, "A Standard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms," Journal of Scheduling, 5, pp.379-394, 2002.

[18] http://www.kasahara.elec.waseda.ac.jp/schedule/index.html

[19] Yu-Kwong Kwok, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," Journal of Parallel and Distributed Computing, 59, pp.381-422, 1999.

김 정 환

e-mail : jhkim@kku.ac.kr

1991년 서울대학교 계산통계학과(학사)

1993년 서울대학교 전산과학과(이학석사)

1999년 서울대학교 전산과학과(이학박사)

1999년~2000년 삼성전자 통신연구소 연구원

2001년~현재 건국대학교 자연과학대학 컴퓨터·응용과학부 조교수

관심분야 : 병렬처리, 분산 시스템, 결함 내성, 그룹 통신