병렬계산의 스케쥴링에있어서 유전자알고리즘에관한 연구

(A study on the Genetic Algorithms for the scheduling of Parallel Computation)

성 기 석, 박 지 혁
강릉대학교 산업공학과

Kiseok Sung, Jeehyuk Park
Industrial Engineering Department
Kangnung National University

Abstract

For parallel processing, the compiler partitions a loaded program into a set of tasks and makes a schedule for the tasks that will minimize parallel processing time for the loaded program. Building an optimal schedule for a given set of partitioned tasks of a program has known to be NP-complete.

In this paper we introduce a GA(Genetic Algorithm)-based scheduling method in which a choromosome consists of two parts of a string which decide the number and order of tasks on each processor. An additional computation is used for feasibility constraint in the chromosome.

By granularity theory, a partitioned program is categorized into coarse-grain or fine-grain types. There exist good heuristic algorithms for coarse-grain type partitioning. We suggested another GA adaptive to the coarse-grain type partitioning. The infeasibility of chromosome is overcome by the encoding and operators.

The number of processors are decided while the GA find the minimum parallel processing time.

1. 서론

병렬처리(parallel processing) 시스템에서 프로그램이 부하(load)되면 compiler는 target code를 생성하게되는데, 그 과정은 적절한 프로그램 분할, 수행될 프로세서수의 결정 그리고 스케쥴링의 세 과정으로 이루어진다. 이 세 가지 문제는 모두 NP-complete임이 알려져있다.[1][2][3]

프로그램 분할은 프로그램을 작은 조각 프로그램(태스크)으로 나누어 그 종속관계를 규정짓고 각 태스크의 처리시간과 태스크간 자료의 이동시간을 산출하는 과정이다. 스케쥴링은 주어진 프로세서 안에서 병렬처리시간(PPT:Parallel Processing Time)을 최소화하기 위해서 각 태스크의 수행 시작 시각과 수행되어야할 프로세서를 배정(allocation)하는 과정이다.

분할된 프로그램은 입자이론(granularity theory)에의하여 coarse grain과 fine grain의 두가지형태로 나뉜다. Coarse grain 형태인 경우 최소 PPT는 태스크들의 linear clustering에의하여 구해질 수 있다. Linear clustering을 이용하면 worst-case인 경우에도 최소 시간의 두배를 넘지 않는다는 것이 알려져 있다.[4][5] 이러한 이유로 병렬계산모델에대해 소개된 지금까지의 해법들은 대부분 coarse grain 대상의 heuristic 기법들이다. 이러한 heuristic 기법들은 모든 유형의 프로그램에서 최소 PPT를 얻을 수 있다는 보장이 없으며 분할된 프로그램이 coarse grain 형태이어야 한다.

본 연구에서는 Job shop이나 TSP(Traveling Salesman Problem)의 스케쥴링과 같은 combinatorial 문제의 해법에 효율적이라고 알려진 유전자알고리즘(GA:Genetic Algorithm)을 병렬계산의 스케쥴링에 적용하고자 한다.[6][7][8][9] Fine grain과 coarse grain 모두에 적용 가능한 GA와, coarse grain의 경우 linear clustering을 이용한 GA 두 가지를 소개한다.

2. 병렬계산모델(Parallel Computation Model)

프로세서의 수가 m개, 태스크가 k개 있다고 하자. 그리고, 각 프로세서는 배정된 태스크가 있으면, 다른 태스크를 수행중이지 않는 이상 반드시 그 태스크를 수행한다고 하자(work-preserving). 그러한 조건하에서 각 태스크를 각 프로세서에 배정하고, 배정된 프로세서 내에서 각 태스크의 시작시간을 결정하는 것을 병렬계산의 스케쥴링이라 한다.

예를 들어 m개의 프로세서와 n개의 태스크로 구성된 병렬계산모델을 보자. 모든 태스크가 각각의 프로세서로 배정되고 해당 프로세서 안에서 태스크간의 우선 순위가 결정되었다면 하나의 스케쥴이 생성되며 이때의 태스크 수행 시작 시각($t[i,k]$)과 스케쥴 완성시간(T_0)은 다음과 같이 계산된다.

M : processors, $|M|=m$
N : tasks, $|N|=n$
C : $c_{ij} \in C$, $i=1,\cdots,n$, $j=1,\cdots,n$
태스크의 처리시간 iff $i=j$
태스크 i와 j사이의 자료이동시간 iff $i \neq j$
$IP[i, j]$: 1 iff j의 긴급선행(imminent precedent) 태스크가 i
0 iff 기타
T : $t[i, k] \in T$, $i=1,\cdots,n$, $k=1,\cdots,m$
work-preserving을 만족하는 k 프로세서에서 i 태스크의 수행 시작시각

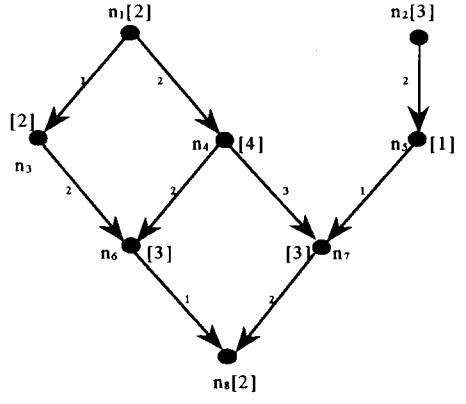
$$t[i,k] = \max\{t[j,k] + C_{ji}, IP[l,i] \cdot (t[l,r] + C_{il} + C_{li})\}, k \neq r$$

$$T_0 = \max\{t[i,k] + C_{ii}\}$$

위의 c_{ij}에서 $i \neq j$일 때 i와 j 태스크가 동일 프로세서에 배정되었다면 자료이동시간은 0이된다. 그림1.(a)는 태스크가 8개인 병렬계산모델의 한 예이고 여기서 arc 위의 숫자는 자료이동시간을, node 위의 숫자는 해당 태스크의 처리시간을 의미한다. 이러한 도식을 DAG(Directed acycle weighted task graph)라하고 이 문제의 C행렬은 그림1.(b)와 같다.

그림1의 문제에서 각 태스크의 프로세서로의 배정과 각 프로세서 안에서의 task순서가 {$n_1 \rightarrow n_2 \rightarrow n_3$}, {$n_4 \rightarrow n_5 \rightarrow n_7$}, {$n_6 \rightarrow n_8$}라 하자. 이 스케쥴의 DAG는 그림2(a)와 같으며 그림2(b)는 $t[i, k]$를 이용하여 Gantt chart로 표현한 모습이다. 이 스케쥴의 완성시간(T_0)은 16이 된다.

3. 일반적인 GA



(a) a DAG

(b) C 행렬

* [ ]는 태스크 node의 처리시간

그림 1. 8개 태스크를 갖는 병렬계산모델의 예

3.1 Encoding

여기서 chromosome은 두 부분으로 구성된다. 하나는 태스크 sequence로 태스크간의 순서를 의미하며 다른 하나는 각 프로세서에 배정되는 태스크 수를 이진으로 표현한다. 이진 열의 길이는 프로세서의 개수를 모두 표현 가능하도록 설정한다. 아래는 8개의 태스크와 4개의 프로세서가 있을 경우의 예이다.

3 1 2 4 5 7 8 6 | 011 | 011 | 010 | 101
↪task sequence↩ ↘ ↘ ↘ ↘
3 3 2 4

위 chromosome에서 {3, 1, 2}, {4, 5, 7}, {8, 6}이 각각의 프로세서에 배정되며 3→1→2, 4→5→7, 8→6이 수행순서가 된다. 이때 태스크가 모두 8개이므로 나머지 한 프로세서는 유휴(idle)상태가 된다.

3.2 Feasibility 유지

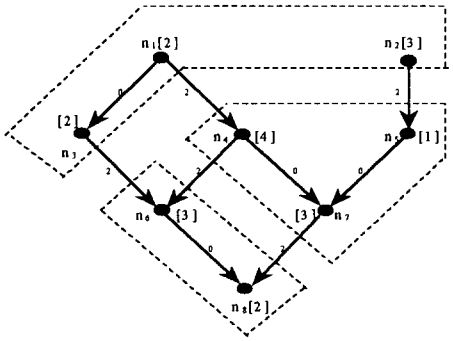
각 프로세서에 배정 받은 태스크들은 그 sequence에서의 위치에 따라 해당 프로세서에서의 수행순서가 결정되는데 어떤 경우 문제의 선후 관계 제약(precedence constraint)에 위배되는 sequence가 생성될 수 있다. 그러므로 보유하고있는 정보는 그대로 유지시키면서 feasibility를 만족하도록 chromosome에서 각 프로세서의 태스크 sequence를 수정해야한다.

선후관계는 C행렬에 나타나있다. 수정하는 과정은 두 번째부터 끝 task까지 순차적으로 진행한다. i번째 태스크에대해서 i-1번째 이전에 있는 k번째 태스크가 i번째 태스크뒤에 와야한다면, k ~ i-1번째 태스크들과 i번째 태스크를 교체한다.

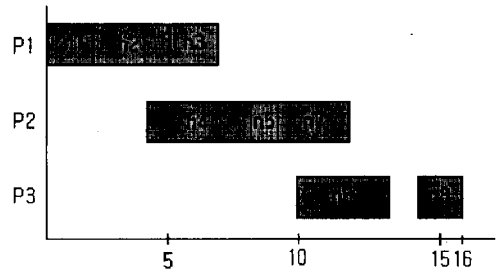
이 방법을 이용하여 그림1문제의 feasibility를 만족하도록 위의 chromosome을 수정하면 각 프로세서 안에서의 태스크 sequence는 다음과 같이 수정된다.

3 1 2 → 1 3 2 → 1 2 3*
4 5 7*

$$\begin{array}{c|cccccccc} & n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 \\ \hline n_1 & 2 & & 1 & 2 & & & & \\ n_2 & - & 3 & & & 2 & & & \\ n_3 & - & - & 2 & & & 2 & & \\ n_4 & - & - & - & 4 & & 2 & 3 & \\ n_5 & - & - & - & - & 1 & & 1 & \\ n_6 & - & - & - & - & - & 3 & & 1 \\ n_7 & - & - & - & - & - & - & 3 & 2 \\ n_8 & - & - & - & - & - & - & - & 2 \end{array}$$



(a) a scheduled DAG



(b) Gantt chart

그림2. 그림1 스케쥴의 예

8 6 → 6 8*
1 2 3 4 5 7 6 8 | 011 | 011 | 010 | 101*

*는 feasibility가 모두 만족되었음을 의미하며 이 때의 chromosome은 유일한 하나의 스케쥴과 mapping되고 앞절의 *t[i,k]*를 이용하여 구해진다.

3.3 Selection

crossover를 수행하게될 부모 chromosome들은 적합도(fitness value)를 기준으로 선택된다. 병렬계산의 스케쥴링문제는 최소 PPT를 갖는 스케쥴을 구하는 것이 목표이므로 chromosome에 대한 스케쥴의 PPT가 작을수록 적합도는 높아지게 된다. chromosome들의 적합도로 이루어진 확율분포에 난수를 발생시킴으로서 부모를 선택한다.

3.4 Operators

문제의 성격에 따라 여러 가지의 operator를 도입하여 사용할 수 있지만 여기선 가장 기본이 되는 crossover와 mutation operator만을 소개하기로 한다.

crossover는 chromosome의 두 부분을 따라 따로 수행시키는데 태스크 sequence부분에는 순서결정문제에 매우 호율적이라고 알려진 PMX(Partially Mapped Crossover)와 OX(Order Crossover)중 OX를, 나머지 이진부분에는 임의의 한 점을 기준으로 한 교체(swap) crossover를 한다.[6][9]

P_1 : 12 | 345 | 768 011 011 010 101 *
× × ↕
P_2 : 36 | 827 | 154 011 011 010 101 *

⇓

O_1 : 27 | 345 | 168 011 011 011 100
O_2 : 45 | 827 | 613 011 010 110 101

mutation은 확율적으로 양쪽부분 혹은 어느 한쪽 부분을 대상으로 하고 태스크 sequence부분은 임의의 두 gene에대한 교체를, 이진부분에서는 임의의 한 gene을 선택하여 0은 1로, 1은 0으로 교체한다. 아래는 두 부분 모두에 mutation이 적용되는 예이다.

2 7 3 4 5 1 6 8 011 011 011 100

⇓

2 1 3 4 5 7 6 8 001 011 011 100

위 두 가지 operator를 거친 chromosome들 또한 앞절의 *Feasibility 유지*과정을 거치게된다.

4. Coarse grain형태의 문제를 위한 GA

coarse grain모델의 경우 각 프로세서에 배정된 태스크들이 linear clustering을 만족할 때, 프로세서 안에서의 태스크순서가 자연적으로 결정된다. 제안하고자하는 GA는 crossover와 mutation operator가 linear clustering을 유지한다. 따라서 초기집단(initial population)만 linear clustering을 유지하도록 한다면, feasibility를 유지하기 위한 별도의 계산과정이 필요 없다.

selection방법은 3절과 같으며 스케쥴 완성시간(T_0)은 3절의 유전자알고리즘과 마찬가지로 2절의 *t[i,k]*를 이용하여 구한다. encoding과 operators는 다음과 같다.

4.1 Encoding

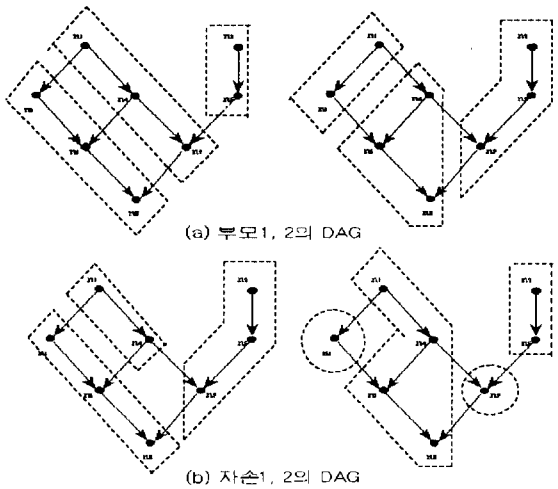
chromosome의 길이는 태스크의 수와 일치하며 chromosome에서 i번째 숫자는 i번째 태스크가 배정되는 프로세서를 의미한다. 그림1 문제를 linear clustering을 만족하도록 표현한 chromosome의 예이다.

n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_8
a string : 1 3 2 1 3 2 1 2*

위 chromosome에서 프로세서에 배정된 태스크와 그 순서가 {1→4→7}, {3→6→8}, {2→5}임을 알 수 있다.

4.2 Operators

crossover는 다음순서를 따른다:
1)모든 태스크를 배정되지 않은 집합 S_1으로둔다.



(a) 부모1, 2의 DAG

(b) 자손1, 2의 DAG

그림3. crossover의 예

2)S_1중 임의의 한 태스크를 선택하여 태스크 t라하자. 부모1에서, 태스크 t와 같은 프로세서에 배정받은 선행(후행) 태스크들과 부모2에서, 태스크 t와 같은 프로세서에 배정 받은 후행(선행) 태스크들을 자손1(2)에서 동일한 하나의 프로세서에 배정한다. 이때 해당 태스크가 자손 chromosome에서 이미 프로세서를 배정 받고 있으면 그대로 둔다.
3)자손1을 기준으로 프로세서에 배정된 태스크를 S_1에서 제외시킨다. S_1에 남아있는 태스크가 하나도 없는 경우는 4)로가고 그렇지 않으면 2)로간다.
4)자손2중 배정받지않은 태스크를 내림차순으로 정렬하여 집합 S_2로 둔다.
5)S_2의 첫 번째 태스크를 태스크 k라 하자. 태스크 k의 긴급 선후 태스크들에게 배정된 프로세서들을 집합 P에 둔다. P에 있는 프로세서가 chromosome에서 태스크 k의 프로세서와 태스크 k의 긴급 선후 태스크들의 프로세서사이에 존재하면 P에서 제외시킨다. P에서 임의의 한 프로세서를 선택하여 태스크 k의 프로세서로 한다.
6)S_2에서 태스크 k를 제외시키고 S_2에 남아있는 태스크가 없으면 끝내고 있으면 5)로간다.

다음 두 부모(P_1, P_2)의 경우 태스크가 6→4→2로 선택된다면 위의 crossover operator를 사용하여 다음과 같은 자손(O_1, O_2)이 나오며 그림3은 이를 DAG로 표현한 것이다.

$$P_1 : \ 1\ 3\ 2\ 1\ 3\ 2\ 1\ 2^*$$
$$P_2 : \ 2\ 1\ 2\ 3\ 1\ 3\ 1\ 3^*$$

$$\Downarrow$$

$$O_1 : \ 2\ 3\ 1\ 2\ 3\ 1\ 3\ 1^*$$
$$O_2 : \ 1\ 3\ 4\ 1\ 3\ 1\ 2\ 1^*$$

자손2에서 태스크 1과 3은 앞의 알고리즘에서 4), 5), 6)과정을 거치어 프로세서 1, 4를 각각 배정 받은 것이다.

mutation은 임의의 한 태스크를 선택하여 긴급선후 태스크들의 프로세서들중 하나를 임의로 배정한다.

단, mutation하는 태스크가 배정 받은 프로세서와는 다른 프로세서이어야 하며, mutation에의해서 끊긴 태스크들은 별도의 프로세서에 배정한다. 위의 자손2를 태스크 6에대해 mutation하면 다음과 같다.

$$1\ 3\ 4\ 1\ 3\ 1\ 2\ 1^*$$

5. 결론

본 연구에서 fine grain과 coarse grain형태의 병렬계산의 스케쥴링에 모두 적용 가능한 GA와 coarse grain형태인 경우 linear clustering을 이용한 GA를 제안하였다.

제안된 GA들은 사용되어야할 프로세서의 최적개수와 PPT를 최소화하는 스케쥴을 구하여준다. 제안한 첫 번째 GA는 feasibility를 유지시키기 위해서 별도의 계산을 할애하므로 fine grain형태의 문제인 경우 feasibility를 유지하는 적절한 encoding방법과 operators의 고안이 필요하다. 제안된 두 번째 GA가 feasibility와 linear clustering을 유지하면서 세대진행을 하므로 coarse grain형태의 문제에 대해서는 첫 번째 GA보다 효율이 높으리라 기대된다. Compiler는 입자이론에의하여 어떤 형태로 프로그램이 분할되었는지 살펴보아 coarse grain형태라면 두 번째 GA를 사용할 수 있을 것이다.

이 연구에서는 하나의 프로그램이 병렬처리시스템에 부하되었을때를 대상으로 하므로 앞으로 다중프로그램이 병렬처리시스템에 부하되었을 경우 GA를 이용하는 방안이 다루어져야 할 것이다.

참고문헌

1. Chretienne P., "Task scheduling over distribute memory machines," Workshop on Parallel and Distributed Algorithms, North-Holland, Amsterdam, 1989.
2. Lenstra J.K. and Rinnooy Kam A.H.G., "Complexity of scheduling under precedence constraints," Oper. Res., Vol.26, 1978.
3. Coffman E.G. and Deming P.J., *Operation Systems Theory*, Prentice-Hall, Engflewood Cliffs, NJ, 1973.
4. Gerasoulis A. and Yang T., "On the granularity and clustering of directed acyclic task graphs," IEEE Trans. Parallel Distrib. Syst., Vol4, 686-701, 1993.
5. Chretienne et al., *Scheduling Theory and its Applications*, John Wiley & Sons, 1995.
6. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, NY, 1989.
7. Guoyong Shi, "A genetic algorithm applied to a classic job-shop scheduling problem," International Jour. of Syst. Sci., Vol.28, No.1, 25-32, 1997.
8. Colin R. Reeves, "A genetic algorithm for flowshop sequencing," Computers Ops. Res. Vol.22, No.1, 5-13, 1995.
9. Zbiqniew Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, SPringer-Verlag, 1995.