論文
9-2-4

Do-Loop 병렬수행 알고리즘의 문제점 분석 및 고찰 (A Analysis and Consideration About Problems of Do-Loop Parallel Processing Algorithm)

송 월 봉(Worl-Bong Song)[1)]

요 약

병렬 스케쥴링의 목적은 다중프로세서 병렬시스템 환경에서 병렬성을 가진 응용 프로그램에 대하여 최소 동기화 오버헤드(Synchronization overhead) 및 병목현상(Bottleneck) 그리고 부하균등(Load balance)을 달성하도록 스케쥴링을 수행하는데 있다. 본 논문에서는 기존의 대표적인 병렬수행 스케쥴링 알고리즘들을 분석하고 각 방법들로부터 문제점들을 도출한다. 이는 향후 효율적인 알고리즘을 설계하는데 도움이 될 것이다.

Abstract

The purpose of a parallel scheduling is to carry out the scheduling with the minimum synchronization overhead and bottleneck under a multiprocessor environment and to perform load balance for a parallel iteration. In this paper, analyse the conventional parallel scheduling methods and drive the problems from each method in order to achive the minimum scheduling overhead and load balance. These problems will go far toward solving the design of effective algorithm.

논문 접수 : 2008. 6. 4.
심사 완료 : 2008. 6. 16.

1) 시립인천전문대학 컴퓨터정보과 교수

※본 논문은 인천전문대학 연구지원비에 의한 것임.

1. 서론

본 논문에서는 지금까지 제안된 기존의 병렬 루프 스케쥴링 알고리즘들에 대한 문제점들을 도출하고 이들 문제점들을 분석하여 병렬 루프 스케쥴링 알고리즘에서 이를 개선하기 위한 개선책들에 대하여 제안하고자 한다.

공유메모리 다중 프로세서 시스템에서 효과적인 병렬 루프 스케쥴링 알고리즘을 수행하기 위해서는 다음과 같은 몇 가지 사항들을 고려하여 알고리즘에 반영해야 한다[8].

1) 스케쥴링을 수행할 idle 프로세서에게 공유변수 메모리로부터 반복들에 대한 chunk를 할당하기 위해 chunk의 크기를 결정할 때, 발생되는 동기화 오버헤드(synchronization overhead)를 최소로 유지해야 한다[2,3,5,8].

2) 공유변수 메모리로부터 계산된 chunk를 idle 프로세서의 지역 메모리에 할당할 때, 소요되는 전송시간0000000000000000000에 따른 통신 오버헤드(communication overhead)를 최소로 유지해야 한다.[4]

3) idle 프로세서들이 공유변수 메모리로부터 반복들에 대한 chunk를 계산하기 위해 공유변수 메모리에 접근할 때, 동시에 여러개의 idle 프로세서들이 접근하려고 함으로써 발생되는 대기 시간(waiting time) 즉, 최소의 병목현상을 유지해야 한다[7].

4) 각 idle 프로세서에게 할당된 병렬반복들에 대한 chunk들의 실행 시간에 기인하여 부하균등(load balance)을 달성해야 한다[1,6].

그러나 기존의 병렬 루프 스케쥴링 알고리즘들은 오버헤드를 줄이고 부하균등을 달성하기 위해 단지 공유변수 메모리로부터 각 반복들에 대한 chunk의 크기를 조절하여 스케쥴링을 수행함으로써 이러한 관점에서 모두 많은 결함들을 가지고있다. 특히, 루프의 각 반복들의 실행시간이 가변인 경우 즉, 불규칙적인 루프에서는 이러한 결함들이 더 많이 발생된다.

2. 알고리즘 분석 1

2.1 Self-Scheduling

Self-Scheduling 알고리즘은 스케쥴링을 수행하는 모든 프로세서들이 idle할 때마다 공유변수 메모리로부터 하나의 반복만을 취하여 스케쥴링을 수행하기 때문에 반복들의 수인 N번의 연산 동작이 필요하다. 따라서 동기화 오버헤드가 병렬 루프의 인덱스 수에 비례하여 발생한다. 또한, 각 반복들의 실행시간 분포가 작을 때, 스케쥴링을 수행하는 각 프로세서들은 루프 인덱스 변수와 같은 공유변수 메모리에 상호 배타적으로 빈번하게 접근하게 되어 프로세서들의 메모리 경쟁과 그에 따른 병목현상이 과도하게 발생할 수 있다. 그러므로 프로세서들은 오히려 반복들을 실행하는 시간 보다 더 많은 시간을 공유변수 메로리로부터 반복들을 취하기 위해 경쟁하는데 실행시간이 거의 모든 시간을 소비한다.

```
DOALL I=1,N
  IF (C) THEN
         {20}
         ELSE {6000}
  ENDIF
ENDOALL
```

[그림 1] 불규칙 루프 예

[Fig. 1] A example of irregular loop

```
DOALL   1   I1 = 1, 15
 DOALL   2   I2 = 1, 17
  DOALL   3   I3 = 1, 17
   DOALL   4   I4 = 1, 25
          {20}
   4 CONTINUE
  3 CONTINUE
 2 CONTINUE
1 CONTINUE
```

[그림 2] 규칙 루프 예

[Fig. 2] A example of regular loop

2.2 Guided Self-Scheduling

Guided Self-Scheduling 알고리즘은 각 프로세서들이 공유변수 메모리에 남아있는 반복의 1/p개를 idle 프로세서에게 할당하기 때문에 모든 반복들이 같은 실행시간을 가진다면, 최적의 오버헤드와 부하균등을 달성할 수 있다. 그러나 프로세서들이 반복의 끝에서 공유변수 메모리에 대한 과도한 경쟁을 유발할 가능성이 높다. 또한, Guided Self-Scheduling 알고리즘은 각 반복들의 실행시간 분포가 불규칙적인 경우에는 특히, 각 반복들의 실행시간이 선형적으로 감소하거나, 처음 몇몇 반복들의 실행시간이 큰 경우에는 프로세서에게 반복들의 초기 할당이 너무 크기 때문에 ⌈N/P⌉ 실행 종료시간에 따른 부하불균등을 유발하여 최적의 실행시간 내에 끝나지 않는다.

2.3 Factoring

Factoring 알고리즘은 Guided Self-Scheduling알고리즘에서 초기에 부하불균등을 유발하는 큰 chunk를 감소시키기 위해 ⌈N/P⌉ 의 chunk를 ⌈N/2P⌉ 크기인 P개의 배치(batch)로 하여 스케쥴링을 수행하였다. 그러므로 idle 프로세서에게 반복들의 할당이 여러 단계에 거쳐 진행되며, 각 단계 동안 공유변수 메모리에 남아있는 반복들의 부분 집합만을 할당하기 때문에 Guided Self-Scheduling 알고리즘에 비해 오버헤드가 더 좋지 않다. 또한 불규칙적인 반복들에 대해서 초기 ⌈N/2P⌉ 크기인 P개의 배치들 중 1/2 개 이하의 배치들의 반복 실행시간이 나머지 배치들의 반복 실행시간과 분산 차이가 클 때는 부하불균등을 유발하여 나쁜 종료시간을 가진다

2.4 Trapezoid Self-Scheduling

Trapezoid Self-Scheduling 알고리즘은 처음에 동기화 오버헤드를 줄이기 위해 idle 프로세서들에게 반복들의 큰 chunk(N/2P)를 할당하고 부하균등을 달성하기 위해 나머지 반복들에 대해 선형적으로 감소하는 chunk 크기 즉, δ만큼씩 감소하는 chunk의 크기를 할당하였다. 그러므로 연속된 N개의 chunk 크기는 항상 불변이다. 그러나 Trapezoid Self-Scheduling 알고리즘은 각 반복들의 실행시간이 불규칙적인 경우 크기가 작은 chunk가 큰 chunk보다 더 많은 실행시간을 가짐으로써 부하불균등을 유발할 수 있다는 점을 고려하지 못하고있다.

2.5 Affinity Scheduling

Affinity Scheduling 알고리즘은 초기에 정적으로 P개의 프로세서에게 연속된 반복들의 ⌈N/P⌉ 의 Chunk를 할당하여 동기화 오버헤드를 줄이려고 하였다. 그러나 처음 정적 으로 할당하여 동기화 오버헤드를 줄이려고 하였다. 그러나 처음 정적으로 할당할 때, 반복들의 국부성을 고려하지 않고 할당함으로써 불규칙적인 반복인 경우에는 각 프로세서에게 할당된 반복들의 실행시간 분포 차가 많아서 부하불균등을 유발할 확률이 높다. 또한, Affinity Scheduling 알고리즘에서는 부하불균등이 발생했을 때, 반복들이 가장 많이 남아있는 프로세서를 찾은 후, 남아있는 반복의 1/P를 위하여 실행함으로써 부하균등을 유지하려고 하였다. 그러나 부하균등을 달성하기위하여 가장 많이 남아있는 공유변수 메모리로부터 너무 적은 반복만을 취하기 때문에 공유변수 메모리의 접근 빈도수가 많아지고 따라서, 과도한 동기화 오버헤드를 유발할 가능성이 높다. 결과적으로 기존의 스케쥴링 알고리즘들은 많은 동기화 오버헤드를 유발하지 않고 반복들에 대한 부하균등을 달성하기 위해 시도되었다.

3. 알고리즘 분석 2

그러나 이들 알고리즘은 개개의 반복이 모든 프로세서에서 같은 실행시간을 가진다는 가정하에 각 반복에 대해 서로 독립된 임의 변수로 취급하고 스케쥴링을 수행하였다. 따라서, 각 반복들이 가변의 실행시간을 가진 불규칙적인 경우에는 작은 크기의 chunk가 더 큰 chunk

보다 더 많은 실행시간을 가질 수 있기 때문에 부하불균등이 발생된다. 또한 공유변수 메모리로부터 반복된 chunk의 계산과 계산된 chunk의 idle 프로세서에 할당은 과도한 오버헤드를 유발하여 전체적으로 스케쥴링에 나쁜 영향을 준다. [그림 1]과 [그림 2]를 가지고 기존의 스케쥴링 알고리즘들에 대해 4개의 프로세서를 가진 시스템에서 idle 프로세서에게 할당되는 반복들의 크기를 종합해 보면 <표 1>및 <표 2>와 같다. 단, 중괄호 속의 숫자인 직선 코드들의 실행시간이 모두 같고, [그림 1]에서 반복들의 수를 N=124로 가정한 결과이다.

스케쥴링 알고리즘 구분	N=124, P=4
Self-Scheduling	1,1,1,1,. . . .
Guided Self-Schedulin	31,24,18,13,10,7,6,4,3,2,2,1,1,1,1
Factoring	16,16,16,16,8,8,8,8,5,5,5,5,2,2,2,2,1,1,1,1
Trapezoid Self-Scheduling	15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
Affinity Scheduling	31,31,31,31

<표 1> [그림 1]에서 반복들의 수를 N=124로 가정한 결과

<Tab.1> The result of iteration number N=124

스케쥴링 알고리즘	N=108375, P=4
Self-Scheduling	1,1,1,. . . .
Guided Self-Scheduling	27094,20321,15240,11430,8573,6430,4822,3617,2712,2034 1526,1144,858,644,483,362,272,204,153,114,86,64,48,36,27 21,15,12,9,6,5,4,3,2,1,1,1,1
Factoring	13547,13547,13547,13547,6774,6774,6774,6774,3387,1693 1693,1693,1693,847,847,847,847,423,423,423,423,212,212 212,212,106,106,106,106,53,53,53,53,26,26,26,26,13,13,13
Trapezoid Self-Scheduling	13,7,7,7,7,3,3,3,3,2,2,2,2,1,1,1,1 13546,12643,11740,10837,9934,9031,8128,7225,6322,5419 4516,3613,2710,1807,904,1
Affinity Scheduling	27094,27094,27094,27093

<표 2> [그림 2]의 기존 스케쥴링 알고리즘 chunk크기

<Tab. 2> The scheduling algorithm chunk size of [fig. 2]

3.1 self-Scheduling 알고리즘

1개의 chunk 크기를 갖는 self-Scheduling 알고리즘의 경우 <표 1>에서 불규칙적인 실행시간을 갖는 반복들에 대해 각각의 프로세서들은 60600의 같은 실행시간을 갖는다. 또한 <표 2>에서는 규칙적인 실행시간을 갖는 반복들에 대해 27094 ,27094, 27094, 27093개의 반복들을 실행함으로써 모든 프로세서들이 1개의 반복내에 수행을 완료하여 부하균등을 달성한다. 그러나 <표 1>및 <표 2>에서 모든 프로세서들이 각각124번과 108375번의 배타적인 공유변수 메모리에 접근이 발생하여 동기화 오버헤드와 병목현상에 대한 부담이 크다.

3.2 Guided Self-Scheduling 알고리즘

Guided Self-Scheduling 알고리즘은 <표 2>의 규칙적인 실행시간을 갖는 반복에서는 처음 할당된 chunk의 실행시간이 541880으로 나머지 반복들의 실행시간 1625620의 0.33%에 불과하여 부하균등을 달성 할 수 있다. 그러나 <표 1>의 불규칙적인 실행시간을 갖는 반복에서는 첫 번째 chunk인 21개의 반복들에 대한 실행시간이 240540으로 나머지 반복들(93개 반복)의 실행시간 1860의 129.32%에 달하여 심각한 부하불균등이 발생된다.

3.3 Factoring 알고리즘

Factoring 알고리즘은 <표 2>의 규칙적인 실행시간을 갖는 경우 각 프로세서에게 처음

할당된 배치의 실행시간이 모두 270940으로 동일하며, 이들은 나머지 반복들의 실행시간 1083740의 0.25%에 불과하여 부하균등을 당성할 수 있다. 그러나 <표 1>의 불규칙적인 실행시간의 경우 각 프로세서에게 처음 할당된 4개 배치(16개 반복)의 실행시간이 각 240,240,320,320,
320으로 첫 번째 배치의 실행시간이 다른 배치의 750.75%에 달하며, 나머지 반복들의 실행시간 1200의 200.2%로 심각한 부하불균등이 발생된다. 또한 공유변수 메모리의 과도한 접근으로 오버헤드에 대한 부담이 크다.

3.4 Trapezoid Self-Scheduling 알고리즘

Trapezoid Self-Scheduling 알고리즘은 <표 2>의 규칙적인 반복의 경우 처음 할당된 chore(chunk)의 실행시간이 270920으로 나머지 반복들의 실행시간 270920으로 나머지 반복들의 실행시간 2167500의 0.13%에 불과하여 부하균등을 달성할 수 있다. 그러나 <표 1>의 불규칙적인 반복의 경우 처음 할당된 chore(15개 반복)의 실행시간이 240220으로 나머지 반복들(109개 반복)의 실행시간 2180의 110.19%로 심한 부하불균등을 유발한다. 따라서, 선형적으로 감소하는 chore의 크기만으로 부하균등을 달성할 수 없다.

3.5 Affinity Scheduling 알고리즘

Affinity Scheduling 알고리즘은 <표 2>의 규칙적인 반복의 경우 chunk의 실행시간이 각각 541880, 541880, 541880, 541860으로 부하균등을 달성한다. 그러나 표 3.9의 불규칙적인 반복에서는 첫 번째 할당된 chunk의 실행시간이 240540으로 다른 chunk들의 실행시간 620의 387.96%로 기존의 스케쥴링 알고리즘들 중 가장 심한 부하불균등이 발생한다. 이러한 부하불균등으로 인해 다른 프로세서의 메모리에 자주 접근하게 되어 동기화 오버헤드에 대한 부담이 크다. <표 3>과 <표 4>는 [그림 1]과 [그림 2]의 프로그램을 수행할 때, 각각의 스케쥴링 알고리즘에 따른 공유변수 메모리의 접근 횟수를 나타낸다.

스케쥴링 알고리즘	공유변수 메모리 접근 횟수	비 고
Self-Scheduling	124	124 오퍼레이션 발생
Guided Self-Scheduling	15	부하불균등
Factoring	20	부하불균등, 오버헤드과다
Trapezoid Self-Scheduling	15	부하불균등
Affinity Scheduling	12	부하불균등, 오버헤드 과다

<표 3>[그림 1]의 기존 알고리즘 공유메모리 접근 수

<Tab. 3> The share-memory access number about existing algorithm of [fig. 1]

스케쥴링 알고리즘	공유변수 메모리 접근횟수	비 고
Self-Scheduling	108375	108375 오퍼레이션
Guided Self-Schedulin	38	부하균등, 오버헤드 과다
Factoring	57	부하균등, 오버헤드 과나
Trapezoid Self-Scheduling	16	부하균등
Affinity Scheduling	0	부하균등, 오버헤드 감소

<표 4>[그림 2]의 기존 알고리즘 공유메모리 접근 수

<Tab. 4> The share-memory access number about existing algorithm of [fig. 2]

결과적으로 기존의 병렬 루프 스케쥴링 알고리즘들은 모든 반복들의 실행시간이 불변인 경우에는 대부분 좋은 실행 결과를 보이고 있다. 그러나 실행시간들이 가변인 경우에는 심각한 부하불균등과 오버헤드를 유발하여 전체 스케쥴링 성능에 나쁜 영향을 준다.

4. 결론

이러한 문제들에 대한 개선책은 다음과 같다.

1) 각 반복들의 실행시간에 따른 반복들의 국부성을 고려해야 한다.

2) 각 프로세서들은 자기 자신의 단일 인덱스로서 독립적인 스케쥴링을 수행하여 다음의 요소들을 개선한다.

① 공유변수 메모리로부터 반복의 chunk에 대한 연산시간에 따른 동기화 오버헤드를 감소시킨다

② 공유변수 메모리와 프로세서의 지역메모리 사이에 반복들의 전송에 따른 통신 오버헤드를 감소시킨다.

③ 프로세서들이 상호 배타적으로 공유변수 메모리에 접근하기 위한 병목현상을 줄인다.

3) 각 프로세서에게 할당된 chunk에 대한 부하균등을 달성한다.

향후 과제로는 1)을 개선하기 위해 모든반복들의 국부성을 고려한 반복들에 대해 분해 방법을 제시하고자하며, 2)를 개선하기 위해 프로세서 동족성을 고려하여 각 프로세서의 지역메모리에 분해된 반복들의 chunk를 정 적으로 할당하고 각 프로세서는 자신의 지역메모리로부터 단일 루프 인덱스를 가지고 스케쥴링을 수행하는 방법 및 3)을 개선하기 위해 프로세서가 idle하면, 실행되지 않은 반복들이 가장 많이 남아있는 프로세서로부터 반복들의 대해 동적으로 재 분해 작업을 수행한 후 계속 스케쥴링을 수행하는 방법을 연구하고자 한다.

참고문헌

[1] Fang, Z., P. C Yew, P. Tang, and C. Q. Zhu, "Dynamic processor self-scheduling for general parallel nested loops," in proc. 1987 Int, Conf. parallel processing, PP. 1-10, August. 1987.

[2] Hummel, S. F., E. Schonberg, and L. E. Flynn. "Factoring : A practical and robust method for scheduling parallel loops, "Comm. of the ACM35(8) PP. 90-101, August. 1992

[3] Kruskal, C. and A. Weiss, "Allocating independent subtasks on parallel processings, "IEEE Transactions on Software Engineering, SE-11, October 1999.

[4] Markatos, E. P., T. J. Le Blanc, "Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessor," IEEE Transactions on parallel and Distributed Systems, Vol. 5, no. 4 April 1994.

[5] Polychronopoulos, C. D. and D. Kuck, "Guided self-scheduling : A Practical self-scheduling scheme for parallel supercomputers, "IEEE Trans. Comput. Vol. C-36 PP. 1425-1439, December. 1987.

[6] Tang, P. and P. C. Yew, "Processor self-scheduling for multiple-nested parallel loops," in proc. 1986 Int. Conf. parallel processing, PP. 528-535, August. 1986.

[7] Tzen, T. H. and L. M. Sni. "Trapezoid Self-Scheduling : A Practical Scheduling scheme for parallel compilers," Tech. Rep. MSU-CPS-ACS-27 Michigan State Univ., 1990.

[8] 이영규, 박두순, "공유 메모리에서 개선된 병렬 루프 스케쥴링 알고리즘," 한국멀티미디어학회 춘계학술발표논문집, 제3권 1호, PP. 453-457, 2000.

송월봉

1974년 숭실대 공학사
1982년 한양대 공학석사
1998년 순천향대학교 공학박사(전산학)
1978년~현재 시립인천전문대학 컴퓨터정보과 교수
관심분야 : 병렬처리컴파일러, 알고리즘