성능비대칭적인 멀티프로세서를 위한 OpenMP 의 로드밸런싱 향상 기법

김병규*, 김지민*, 이평화*, 유민수**
*한양대학교 전자컴퓨터통신공학과, **한양대학교 컴퓨터공학부
e-mail : { bkkim, jmkim, phlee, msryu}@hanyang.ac.kr

A Load Balancing Technique for OpenMP for Performance-Asymmetric Multiprocessors

Byung-Kyu Kim*, Ji-Min Kim*, Pyoung-Hwa Lee*, Min-Soo Ryu**
*Dept. of Electronics Computer Engineering, Han-Yang University
**Dept. of Computer Engineering, Han-Yang University

요 약

최근 이기종 멀티프로세서 시스템에서의 병렬화를 위해 범용 CPU 와 다른 컴퓨팅 장치들간의 다양한 연동 기술들이 부각되고 있다. 멀티프로세서 프로그래밍 모델인 OpenMP 는 가장 널리 사용되는 병렬 프로그래밍 언어이지만 기존 OpenMP 의 작업 할당 정책으로는 프로세서간 로드밸런싱을 문제를 해결할 수 없다는 한계점을 가지고 있다. 본 논문에서는 기존 OpenMP 의 작업할당 문제를 해결할 수 있는 알고리즘을 제안한다. 제안하는 알고리즘은 SMP(Symmetric Multi Processing) 구조뿐만 아니라 AMP(명령어 구조는 같으나 동작 속도가 다른 이질 멀티프로세서 구조)에서도 작업부하균형을 효과적으로 실행할 수 있다.

1. 서론

최근 코어의 개수가 증가함에 따라 멀티프로세서 환경에서의 병렬 프로그래밍 기법이 지속적으로 연구되고 있다. OpenMP 는 SMP(symmetric multi-processors) 구조에서 가장 널리 사용되는 병렬 프로그래밍 언어로써 사실상의 표준으로 인식되고 있다. 그러나 기존 OpenMP 에서의 데이터 분할은 병렬 쓰레드들간의 시작시점이 서로 상이하거나 프로세서 동작 속도가 서로 다른 멀티프로세서를 고려하지 않기 때문에 프로세서간 부하균형(load-balancing)에 어려움을 가지고 있다.

본 논문에서는 프로세서들이 서로 상이한 동작 속도를 가지는 경우 OpenMP 를 확장하여 효율적으로 작업분배를 수행하는 기법을 제안한다.

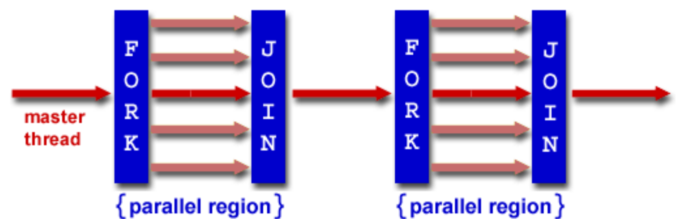
제안하는 기법은 프로세서의 실제 동작 속도 비율을 고려하여 작업을 분배할 수 있으며 병렬 프로그램의 추가적인 코드 변환을 최소화 할 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 본고의 주요 내용인 OpenMP 에 대해 살펴보고, 3 장에서는 제안하는 스케줄링 기법을 소개한다. 4 장에서는 논문의 내용을 요약하여 결론을 내리고, 앞으로 남은 과제들에 대해 간략히 살펴보도록 한다.

2. OpenMP 개요 및 문제점

OpenMP 는 컴파일러 지시어 기반의 병렬 프로그래밍 API(Application Programming Interface)이다. 이는 별도의 병렬 프로그래밍 방법이 필요하지 않고 순차적으로 작성된 프로그램에 병렬적으로 처리할 부분을 정하여 지시어를 추가함으로써 원하는 부분을 병렬적으로 처리할 수 있다는 것을 말한다.

OpenMP 는 포크-조인(fork-join) 모델을 사용한다. 마스터 쓰레드(Master Thread)는 지시어로 처리되지 않는 영역을 혼자 수행하다가 지시어를 만나면 쓰레드를 생성하여 쓰레드 별로 독립적으로 수행하게 된다. 병렬처리가 끝나는 부분에서는 동기화를 위해 모든 쓰레드들이 암시적인 배리어(Implicit Barrier)에 도달할 때까지 대기한다[1,6].



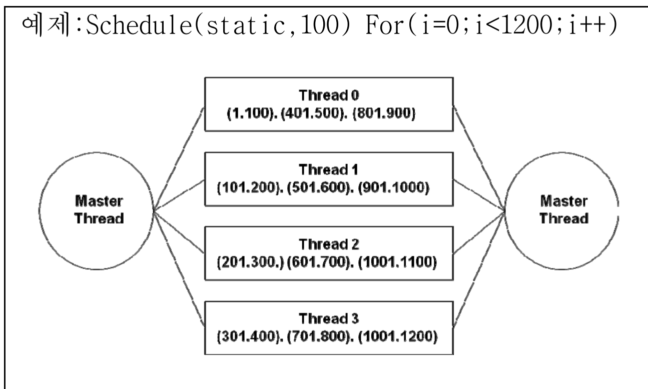
(그림 1) 포크-조인(Fork-Join) 모델

2.1. OpenMP 의 작업할당 정책

OpenMP 에서는 루프의 스케줄을 위해 schedule 보조 지시어를 제공한다. schedule 보조 지시어는 프로그래머에 의해 지정되며 크게 static, dynamic 2 가지 타입을 가지고 있다. 각 타입에서는 루프(loop)의 작업 할

당을 위해 chunk 를 이용하며 이는 프로그래머에 의해 정의되는 값이다. OpenMP 런타임은 정의된 chunk 크기를 기준으로 총 처리할 루프 반복 횟수를 분할하여 각 병렬 쓰레드에 할당하여 처리한다. OpenMP 의 작업할당 정책을 결정하는 Static 과 dynamic 지시어의 차이는 다음과 같다[1,2,3,4,5].

- Static: static 에서 chunk 크기를 지정하지 않는 경우, 각 쓰레드에게 가능한 균등한 비율로 chunk 를 나누어 할당한다. 즉, 처리할 작업량이 N 개이고 생성된 쓰레드 수가 P 개이면 N/P 크기의 작업량이 각 쓰레드에 할당된다. 만약 chunk 크기를 지정할 경우, 각 쓰레드들은 지정된 chunk 크기만큼 라운드로빈(Round-Robin) 방식으로 작업을 정적으로 할당 받아 처리한다.

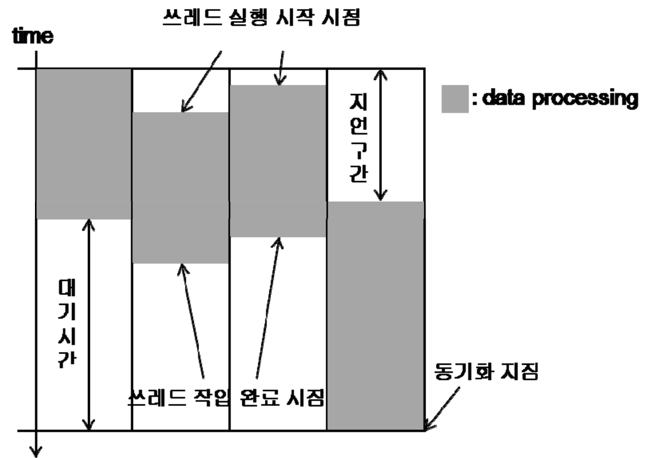


- Dynamic: 각 쓰레드에게 동일한 chunk 크기를 할당하는 것은 static 과 동일하다. 그러나 Dynamic 은 static 과 달리 사전에 모든 작업을 할당하지 않는다. Dynamic 에서는 생성된 각 쓰레드에 지정된 chunk 크기를 동적으로 할당한다. 즉, 할당된 작업을 먼저 끝낸 쓰레드는 남아 있는 작업에서 chunk 크기만큼 동적으로 할당 받아 처리한다.

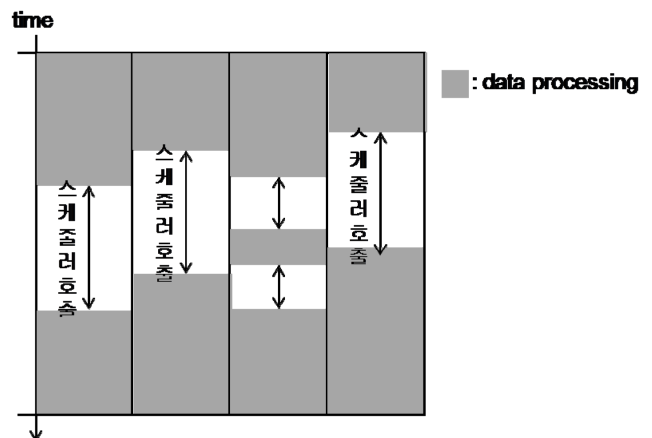
2.2. OpenMP 스케줄러의 문제점

OpenMP 에서 각 쓰레드들은 할당된 작업을 먼저 종료 하더라도 동기화를 위해 모든 쓰레드들이 작업을 종료할 때까지 대기한다. 이것은 쓰레드간의 동기화를 위해 포크 후 조인 시 암시적인 배리어가 발생하기 때문이다. 이때 각 쓰레드가 모두 동일하게 종료 할 수 없게 되고 (그림 2.a)에서와 같이 모든 쓰레드들은 동기화로 인한 대기시간을 가져야만 한다.

Dynamic 타입의 스케줄링은 chunk 만큼의 작업을 먼저 끝낸 쓰레드가 스케줄러를 호출하여 다음 chunk 를 할당 받는다. 이때 쓰레드가 처리해야 할 작업의 양이 큰 경우 스케줄러를 호출하는 비용은 크게 영향을 주지 않고 작업을 처리할 수 있지만, 처리해야 할 작업량이 적고 chunk 의 개수가 많을 경우 각 쓰레드들은 잦은 스케줄러 호출로 인한 오버헤드를 가짐으로써 순차적인 코드보다 더 많은 작업시간을 사용한다. (그림 2.b).



a. OpenMP 동기화로 인한 로드밸런싱 문제점



b. Dynamic 스케줄링에서의 오버헤드 문제점

(그림 2) OpenMP 스케줄러 문제점

3. 제안하는 작업 분할 기법

본 장에서는 OpenMP 에서 발생하는 문제점을 해결하기 위한 새로운 스케줄링 기법을 제시한다. 이상적인 작업분배 비율은 프로세서 속도비율로 작업을 분배하는 것이다. 그러나 실제 프로세서간 속도 비율은 캐시 등의 영향으로 이상적인 동작 속도 비율로 처리되지 않는다. 또한 운영체제의 스케줄링 영향으로 각 프로세서에서 실행되는 쓰레드들의 시작 시점이 서로 다르기 때문에 이상적인 비율 분배는 앞서 언급한 로드밸런싱 문제를 야기한다.

제안하는 기법은 프로세서의 실제 동작 속도 비율을 추출하여 이를 기준으로 작업 할당을 수행하며 각 쓰레드들의 시작 시점에 관계없이 효율적인 로드밸런싱을 수행할 수 있다.

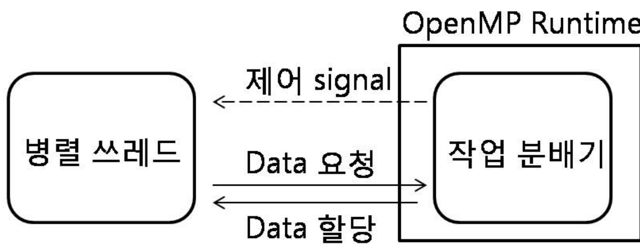
3.1. 작업 분배 알고리즘

그림 3 은 제안하는 알고리즘으로써 먼저 m 개의 서로 다른 속도를 가진 코어와 i 개의 작업량이 있을 때 코어 속도의 비율이 $C_1 : C_2 : ... : C_n$ 라면 i 개의 작업을 $C_1 : C_2 : ... : C_n$ 의 이상적인 비율로 분할한다.

```
Begin
   프로세서의 속도 비율에 따라 데이터를 분할하여 각 해당 프로세서로 할당
   Loop begin
      임의의 프로세서에서 할당된 데이터 처리가 완료될 때까지 대기
      먼저 종료된 프로세서의 쓰레드는 작업 분배기에 알림
      작업 분배기는 작업이 남은 쓰레드의 작업을 회수하여 먼저 종료된 프로세서로 할당
   Loop end
Exit
```

(그림 3) 제안하는 알고리즘

각 쓰레드는 할당 받은 작업을 실행한 후 임의의 한 쓰레드가 종료되면 작업 분배기에 작업 시간과 처리량을 전송한다. 이때 작업 분배기는 나머지 프로세서의 실행 시간에 따른 실제 데이터 처리 속도 비율을 기준으로 남아있는 작업을 할당하는데 이용된다.



(그림 4) 제안하는 알고리즘

제안하는 알고리즘에서는 병렬 쓰레드와 작업 분배기 두 가지의 컴포넌트 환경으로 구성할 수 있다.

병렬 쓰레드는 할당 받은 작업을 완료하는 경우 대기중인 작업 분배기를 깨운 후 자신의 작업 처리속도 정보를 넘겨주며 작업 분배기의 작업 할당을 대기한다.

작업 분배기는 Cilk 의 Work-Stealing[2,3,4] 기법과 유사하게 작업이 남은 쓰레드를 임의로 선정하여 작업을 회수(steal)하고 먼저 종료된 쓰레드에 할당한다. 이때 쓰레드의 남은 작업량이 종료된 쓰레드의 비율에 따른 chunk 크기가 두 배 이상일 경우 작업 분배기가 chunk 크기의 작업을 회수하여 먼저 종료된 쓰레드에 할당한다. Chunk 의 크기가 두 배 이하일 경우 다른 쓰레드의 잔여 작업량을 체크하고 위와 같은 작업을 반복 수행한다.

```
------------------------------------------ Job_Assingner Pseudo code ------------------------------------------

Begin Main function
   do receive speed parameters from all processor;
   do calculate speed ratio for all processor;
   for (Is data to process remained?)
      do Partition the data by processors speed ratio into different size chunks;
      assign the different size chunks into each correspondent processor;
      wait until finishing data processing on any processor;
      do send Signal to all processor;
      for
         if Is all speed parameters received from all processors?
            do calculate speed ratio for all processor;
            break;
         else continue;
Exit Main function

Begin Signal_Handler function
   do receive speed parameter for calculating speed ratio;
Exit Signal_Handler function
```

(그림 5) 런타임 데이터 할당 의사코드

```
------------------------------------------ Thread Part Pseudo code------------------------------------------

// variables declaration
Speed_value = CPU_Speed;
Total_Processing_Time , StartTime, EndTime, Loop_Counter = 0;

Begin Main function
   do pass Speed_value to Job_Assinger for calculating speed ratio
   for
   {
      do Loop_Counter = 0;
      do Total_Processing_Time = 0;
      wait until finishing Job_assignment by Job_Assinger
      if (Is not data to process?) break;

      //Loop variables initial_value, end_value is initialized by Job_Assinger
      for (i= initial_value;   i <= end_value; i++)
      {
         assign current time into variable StartTime;
         do processing data;
         assign current time into variable End Time;
         do Total_Processing_Time += EndTime – StartTime;
         do Loop_Counter++;
      }
      do send Signal to Job_Assinger with speed parameter Loop_Counter/Total_Processing_Time;
   }
Exit Main function

Begin Signal_Handler function
   assign current loop value i  into End_value;
Exit Signal_Handler function
```

(그림 6) 쓰레드 실행 의사코드

Example) 한 시스템에서 500MHz, 1GHz 의 클락 속도를 가진 2 개의 코어가 있다고 가정하자. 300 개의 작업량을 가진 루프를 처리해야 할 때 코어 속도에 따라 1:2 비율로 작업을 분배한다. 이때 500MHz 는 100 개의 작업량을, 1GHz 에는 200 개의 작업량을 가진다. 각각의 코어에 쓰레드를 생성하여 작업을 처리를 하고 먼저 작업이 끝난 쓰레드를 기준으로 런타임에서 작업량의 비율을 계산한다. 종료가 되지 않은 쓰레드의 나머지 작업을 런타임에서 실행 시간에 따른 처리량의 비율을 통해 작업을 재 분배하여 각각의 쓰레드를 통해 작업을 마친다.

4. 향후 계획

본 논문에서는 기존 OpenMP 의 작업 할당 정책의 로드밸런싱 문제와 잘못된 schedule 지시어 사용에 따른 잦은 스케줄러의 호출로 인한 오버헤드 문제를 제기하였다. 이러한 문제점을 해결하기 위해 작업 분배기 컴포넌트를 두고 이를 이용하여 코어의 실제 처리 속도에 따른 비율을 이용한 작업할당 알고리즘을 제안하였다.

제안하는 알고리즘은 SMP(Symmetric Multi Processing) 구조뿐만 아니라 AMP(명령어 구조는 같으나 동작 속도가 다른 이질 멀티프로세서 구조)에서도 작업부하균형을 효과적으로 실행할 수 있다.

향후 제안하는 알고리즘이 플랫폼에 따라 프로그램을 변환하는 변환기를 제공하여 이질 멀티프로세서에 범용적으로 적용될 수 있도록 개선하는 연구를 수행할 계획이다.

참고문헌

[1] http://www.openmp.org/mp-documents/OpenMP3.1.pdf
[2] http://supertech.csail.mit.edu/cilk/manual-5.4.6.pdf
[3] Acar, U.A., Blelloch, G.E., Blumofe, R.D. "The data locality of work stealing," SPAA 2000: Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures, pp. 1–12. ACM, New York, 2000
[4] ROBERT D. BLUMOFE, CHRISTOPHER F. JOERG, BRADLEY C. KUSZMAUL, CHARLES E. LEISERSON, KEITH H. RANDALL, AND YULI ZHOU, "Cilk: An Efficient Multithreaded Runtime System," 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 207-216, 1995
[5] D.S.Henty. "Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling," Proceedings of the 2000 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2000
[6] 전우철, 하순회, "단일 칩 다중 프로세서상에서 운영체제를 사용하지 않은 OpenMP 구현 및 주요 디렉티브 변환," 정보과학회논문지 시스템 및 이론 제 34 권 제 4 호, 2007