OpenMP를 이용한 빠른 정렬의 멀티 코어 구현

*성백상, 성원용
서울대학교 전기공학부
e-mail : *bsseong@dsp.snu.ac.kr, wysung@dsp.snu.ac.kr*

Implementation of Quicksort for Multi-Core Systems using OpenMP

*Becksang Seong and Wonyong Sung
School of Electrical Engineering, Seoul National University

(이 논문은 지식경제부 출연금으로 ETRI와 시스템반도체산업진흥센터에서 수행한 ITSoC 핵심설계인력양성사업과 교육과학기술부의 재원으로 한국학술진흥재단에서 수행하는 BK21 프로젝트의 지원을 받아 수행된 연구입니다.)

Abstract

Quicksort is the fastest among sorting algorithms which use comparison of keys, and it can be parallelized easily due to its algorithmic characteristics. In this paper, we implemented the parallelized quicksort which utilizes shared memory programming, OpenMP. We improved the algorithm of hyperquicksort and obtained the speed-up of mean 250% with a quad-core based system.

I. 서론

빠른 정렬(quicksort)은 매우 많은 분야에서 응용이 되고 있으며, 최근의 멀티코어 CPU를 이용한 좋은 프로그램의 개발이 필요하다. 병렬처리 프로세서를 위하여 하이퍼퀵소트(hyperquicksort)와 같은 알고리즘이 개발되었지만, 이 알고리즘은 OpenMP 보다는 MPI, 메시지 전달 모델(message passing model)에 최적화 되어 있다.

II. 본론

2. 1 하이퍼퀵소트[1]

하이퍼퀵소트는 종래의 빠른 정렬의 병렬도를 극대화하기 위하여 만들어진 알고리즘이다. 그러나 하이퍼퀵소트는 각 프로세스 사이의 개별적인 통신(communication)에 초점을 맞춘 알고리즘이기 때문에 MPI에 최적화 되어 있다고 할 수 있다. 이러한 알고리즘을 OpenMP에 최적화시키기 위해서는 공유 메모리 방식에 맞도록 알고리즘을 수정할 필요가 있다.

2. 2 OpenMP를 이용한 빠른 정렬

OpenMP를 이용하여 빠른 정렬의 병렬도를 극대화하기 위하여 다음과 같은 알고리즘을 사용하였다.

2. 2. 1 입력 데이터의 분할(partition)

각각의 프로세서에 작업을 할당하기 위하여 입력 데이터를 프로세서의 수에 맞게 분할해야 한다. 이 때 각 프로세서들에 할당되어 있는 작업량은 거의 같아야 한다. 즉, 로드 균형(load balance)이 맞아야 한다. 따라서 중앙값(median)을 사용한 분할 알고리즘(partition algorithm)을 사용하는 것이 바람직하다[2].

2. 2. 2 분할된 데이터를 배열에 저장

중앙값을 기준으로 분할된 데이터를 프로세서의 수에 해당하는 개수의 배열에 각각 저장한다. 각 배열에 저장된 값들은 서로 독립적(independent)이라고 할 수 있다. 왜냐 하면 앞의 단계에서 분할을 했기 때문이다.

2. 2. 3 빠른 정렬

위의 단계를 거친 후 각 배열들은 서로 독립적이기 때문에 모든 프로세서가 빠른 정렬을 동시에 수행할 수 있다. 즉, 병렬화가 가능하다.

정렬 방식 / 입력 크기	일반 빠른 정렬	OpenMP 사용	속도 증가
10M	2.94 sec	1.60 sec	×1.83
15M	5.84 sec	2.72 sec	×2.15
20M	9.69 sec	4.13 sec	×2.34
25M	14.52 sec	5.66 sec	×2.56
30M	20.32 sec	8.29 sec	×2.45
35M	27.06 sec	9.84 sec	×2.75
40M	34.71 sec	12.20 sec	×2.84
45M	43.36 sec	14.80 sec	×2.92
50M	52.95 sec	17.73 sec	×2.98
평균			×2.54

표 1. 일반 빠른 정렬과 OpenMP를 이용한 빠른 정렬의 쿼드 코어에서의 실행 시간 비교

정렬 방식 / 입력 크기	일반 빠른 정렬	OpenMP 사용	속도 증가
10M	2.69 sec	1.55 sec	×1.74
15M	5.34 sec	3.00 sec	×1.78
20M	8.86 sec	4.86 sec	×1.82
25M	13.22 sec	7.14 sec	×1.85
30M	18.38 sec	9.84 sec	×1.87
35M	24.53 sec	13.09 sec	×1.87
40M	31.41 sec	16.63 sec	×1.89
45M	37.30 sec	20.70 sec	×1.80
50M	47.97 sec	25.09 sec	×1.91
평균			×1.84

표 2. 일반 빠른 정렬과 OpenMP를 이용한 빠른 정렬의 듀얼 코어에서의 실행 시간 비교

2. 2. 4 병합(merge)

각 프로세서에서 정렬된 데이터들을 병합한다.

2. 3 하이퍼퀵소트와의 차이점

하이퍼퀵소트의 경우 데이터들을 각 프로세서의 지역 메모리(local memory)에 먼저 저장을 하지만, 본 개발된 알고리즘은 공유 메모리상에서 미리 분할된 데이터를 배열을 사용하여 각 프로세서에 할당한다는 점에서 가장 큰 차이점을 가지고 있다. 또한, 하이퍼퀵소트의 경우 입력 데이터를 각 지역 메모리에 저장한 후 분할을 하고, 분할된 데이터들을 각 프로세서들 끼리 교환(swapping)하여 의존도(dependency)를 제거하였지만, 본 알고리즘에서는 데이터를 각 프로세서에 할당하기 전에 분할을 먼저 함으로써 각 프로세서들 사이의 데이터 교환이 필요 없게 되고, 그로 인하여 오버헤드를 줄일 수 있다.

Ⅲ. 구현

본 알고리즘을 C 언어로 구현하여 실험을 하였다. OpenMP를 사용한 빠른 정렬과 사용하지 않은 빠른 정렬의 실행시간을 비교한 것이 표 1과 표 2에 나타나 있다.

표 1의 실험에 쓰인 CPU는 Intel(R) Core2Quad 2.40GHz이다. OpenMP를 사용했을 경우 속도가 세 배 가까이 증가한 다는 것을 알 수 있다. 속도 증가가 네 배에 못 미치는 이유는 분할과 중앙값을 구하기 위한 오버헤드 때문이다. 입력 데이터의 크기가 커질수록 오버헤드의 영향이 줄어드는데, 그 이유는 분할 필요한 실행 시간은 $O(n)$인데 비하여 빠른 정렬을 하는 데 필요한 실행 시간은 $O(nlogn)$이기 때문이다.

표 2에는 듀얼 코어(dual core)를 사용했을 경우의 실험 결과이다. 실험에 쓰인 CPU는 Intel(R) Core2Duo 2.66GHz이다. 여기에서는 위에서 언급한 오버헤드가 크게 줄어들기 때문에, 두 배 가까운 속도 향상을 얻을 수 있었다.

입력 크기	시간 측정 대상	실행 시간
10M	중앙값 측정과 분할	0.84 sec
	배열에 저장	0.05 sec
	합계	0.89 sec
50M	중앙값 측정과 분할	4.07 sec
	배열에 저장	0.23 sec
	합계	4.30 sec

표 3. 입력 데이터의 분할과 분할된 데이터를 배열에 저장하는데 걸린 시간(쿼드 코어에서 측정)

표 3에는 입력크기가 10M와 50M일 때의 오버헤드에 해당하는 부분의 측정 시간이 나타나 있다(쿼드 코어에서 측정).

Ⅳ. 결론

빠른 정렬의 알고리즘을 OpenMP에 최적화 되도록 수정할 수 있으며, 이 때 중앙값을 이용한 분할 알고리즘이 유용하게 쓰인다. 본 논문에서는, 수정된 알고리즘을 C 언어로 구현한 결과, 쿼드코어(quad-core)상에서 세 배에 가까운 속도 향상을 얻을 수 있었다. 코어의 개수만큼의 속도 향상을 가져올 수 없는 이유는 중앙값과 분할에 필요한 오버헤드 때문이다.

참고문헌

[1] Michael, J. Quinn. *Parallel programming in C with MPI and OpenMP*. Mc Grow Hill Companies, Inc., 2004.

[2] Thomas, H. Cormen, Charles, E. Leiserson, and Ronald, L. Rivest. *Introduction to algorithms* (2nd Ed.). Mc Grow Hill Companies, Inc., 2001.