OpenMP를 통한 쿼드 트리 검색 병렬화 및 성능 분석

성운*, 박준석*
*인하대학교 컴퓨터정보공학부
e-mail : wo2n@naver.com

Quad-tree Search Parallelization using OpenMP and Performance Analysis

Woon Sung*, JoonSeok Park*
*Dept of Computer and Information Engineering, Inha University

요 약

OpenMP를 이용할 경우 컴파일러 디렉티브의 삽입으로 병렬화가 가능하다. 본 논문은 쿼드 트리를 이용한 데이터베이스 검색 프로그램을 OpenMP로 병렬화하여 실험을 진행한다. 실험을 통해 논리코어의 개수와 쓰레드 개수에 따른 상관관계가 적다는 것을 알 수 있다. 또한 쓰레드 개수 증가에 따른 오버헤드 발생이 성능에 영향을 준다는 사실을 알 수 있다. 쿼드 트리 자료구조를 이용한 데이터베이스 검색 프로그램을 OpenMP를 이용해 병렬화할 경우 논리적 코어의 개수가 8개, 쓰레드 개수가 16개일 경우에 1.8배의 최대의 성능향상을 이룬다.

1. 서론

무어의 법칙이 소개된 이후 매년 프로세서의 성능은 기하급수적으로 향상되었다. 그러나 최근에는 집적도를 더 이상 높일 수 없는 물리적인 한계와 클록 스피드를 높임으로써 발생하는 발열 문제 등으로 인해 프로세서 자체의 성능 향상을 기대하기 어렵다.

프로세서 성능 향상의 대안으로 등장한 것이 멀티코어[1]다. 클록 스피드와 직접도를 높이기 보다는 코어 수를 늘려 병렬로 연산함으로써 성능 효율성을 향상시킨다. 멀티코어를 효율적으로 이용하기 위해서는 멀티코어 환경에 최적화 된 프로그램이 필요하다. 프로그램이 여러 개의 코어로 연산을 분산해 병렬로 작동해야 한다. 하지만 프로그래머가 병렬로 프로그램을 작성하기는 매우 어렵다.

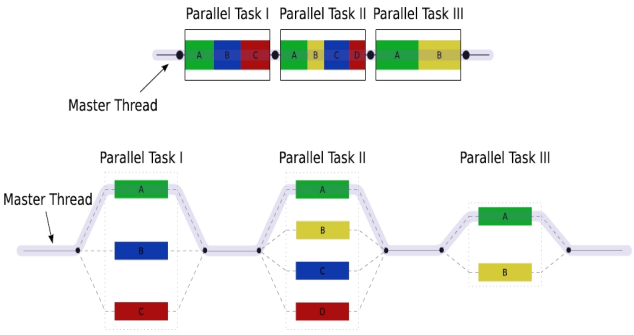
트리는 컴퓨터 응용에서 대표적으로 사용되는 자료구조다. n-체 문제(n-body problem)[2], 유전자의 염기 서열 검색[3], 데이터베이스의 구성 및 검색등의 응용에 사용한다. 그러나 동적할당을 이용한 연결 리스트 트리를 사용할 경우 재귀적인 접근으로 순차적인 실행이 보장되어야 한다는 단점이 있다.

본 논문에서는 트리 자료구조를 이용한 데이터베이스에서 특정 범위에 포함되는 단말 원소들을 검색하는 프로그램을 OpenMP[4]를 이용하여 병렬화를 진행한 후 실험 및 성능 분석을 수행하여 코어의 개수와 쓰레드 개수의 관계에 대해 파악한다. 또한, 여러 개의 쓰레드로 프로그램을 병렬화 할 경우 발생하는 오버헤드의 영향을 파악하여, 향후 OpenMP를 통한 병렬화시 성능예측 모델 연구의 기초 자료로 활용한다.

본 논문의 구성은 다음과 같다. 2장은 병렬화에 사용된 기술인 OpenMP에 대해 소개한다. 3장에서는 프로그램을 병렬화 하는 과정과 성능 변화를 분석하기 위해 실험 내용 및 실험 결과를 분석하여 기술한다. 4장과 5장에서는 기존 관련 연구에 대한 소개와 결론 및 향후 연구 방향에 대해 논한다.

2. 병렬화 기술

2.1 OpenMP



(그림 1) OpenMP의 fork/join 모델

OpenMP는 공유 메모리 다중 처리 프로그래밍 API로 C/C++,포트란 언어와, 유닉스 및 마이크로소프트 윈도 플랫폼을 비롯한 여러 플랫폼을 지원한다. OpenMP는 간단한 컴파일러 디렉티브의 삽입만으로 순차적인 코드를 병

렬화된 코드로 바꿀 수 있다[5].

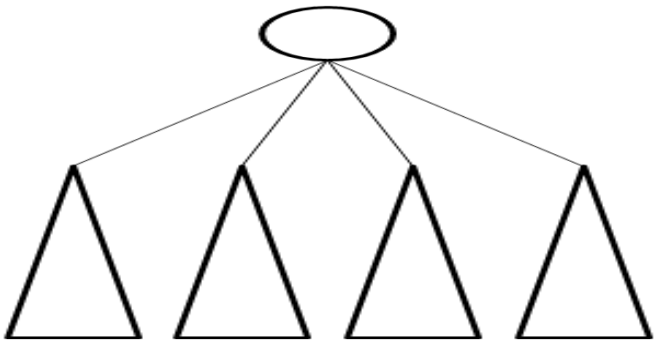
OpenMP의 수행모델은 그림 1의 fork/join 모델이다. 처음 프로그램은 마스터 쓰레드로 동작하며 디렉티브를 만나게 되면 쓰레드를 생성하여(fork) 쓰레드별로 독립적으로 수행하고 디렉티브로 선택한 영역이 끝나게 되면 다시 하나의 마스터 쓰레드로 합쳐져서(join) 순차 코드를 수행하는 형태이다. 쓰레드간 동기화를 위해 방벽(barrier)를 사용하여 동기화 지점을 설정할 수 있으며 특정 디렉티브를 사용하지 않으면 병렬화 영역이 끝나는 지점에 암묵적 방벽(implicit barrier)가 설정되어 있어 병렬화 영역 안에서 독립적으로 수행되는 모든 쓰레드들이 끝나기를 기다렸다가 하나의 마스터 쓰레드로 합쳐진다.

3. 병렬화 및 실험

3.1 병렬화

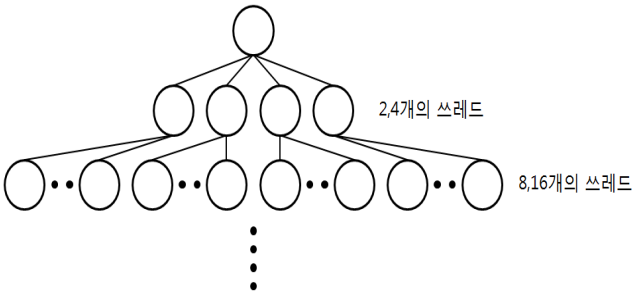
쿼드 트리의 노드는 내부 노드와 단말 노드로 구성된다. 내부 노드는 자신을 포함하는 범위를 가지고 있으며 어떤 데이터가 자신의 범위 안에 있을 경우 반드시 자신이나 하부트리가 범위 안의 데이터를 포함한다. 내부 노드는 다른 내부 노드나 단말의 노드를 자식노드로 가질 수 있으며 자식 노드들은 부모 노드의 범위를 4 등분한다. 특정 위치 정보를 저장하고 있는 데이터를 단말 노드를 삽입 하면서 구성된 트리구조를 검색하는 것이 본 논문에서 병렬화를 진행한 프로그램이다.

특정 범위를 나타내는 질의를 받으면 프로그램은 쿼드트리 안에서 질의 범위에 포함되는 범위 데이터를 가지고 있는 단말노드를 검색한다.



(그림 2) 쿼드트리의 구조 - 부모노드와 4개의 하부트리

그림 2을 보면 부모 노드가 있으면 그 부모 노드가 가지는 범위를 사등분하여 4개의 자식노드가 존재하며 그 자식노드들은 또 자신을 부모노드로 하는 하부 트리를 가진다. 이와 같은 특성을 이용하여 특정범위에 대한 질의를 받으면 루트노드에서 쓰레드의 수만큼 하부트리를 확보한 다음 하부트리 검색을 독립적으로 수행한다. 각각의 하부트리의 데이터는 독립적이며 하부트리 하나당 쓰레드 하나가 재귀적으로 하부트리내의 노드를 검색한다. 검색이 진행되는 동안 검색결과를 저장할 변수는 각 쓰레드별로 따로 관리한다. 하나의 공유변수에 대해 적절한 동기화 없이 쓰기 연산을 할 경우 경합[6]이 발생하여 올바른 결과가 나오지 않는다. 쓰레드들이 검색이 끝나고, 즉 병렬화 구간이 끝난 후 암묵적 방벽를 통하여 동기화 지점을 만들어 모든 쓰레드들의 동작이 끝나면 하나의 마스터 쓰레드로 합쳐진다. 검색결과를 저장하는 변수들이 쓰레드별로 관리되며 검색이 끝날 시 동기화를 통하여 변수들을 하나로 통합하여 결과를 보여준다.



(그림 3) 쓰레드 개수에 따른 병렬화 시작 깊이

OpenMP에서 제공하는 서브루틴을 이용하여 동작 쓰레드 수를 설정 할 수 있다. 그림 3과 같이 2개의 쓰레드로 병렬화를 한다면 병렬화 영역이 깊이 1인 노드들부터 시작한다. 4개의 노드들이 있기 때문에 2개 노드의 하부트리를 하나의 쓰레드가 검색한다. 4개의 쓰레드로 병렬화를 한다면 깊이 1인 노드부터 병렬화가 시작되어 각각 하나의 노드들의 하부트리가 각각 하나의 쓰레드로 검색된다. 이와 같이 동일한 깊이의 노드들이 가지는 하부트리의 개수에 따라 쓰레드 개수를 2, 4, 8, 16, 32, 64개 등으로 확장 할 수 있다.

3.2 실험 내용

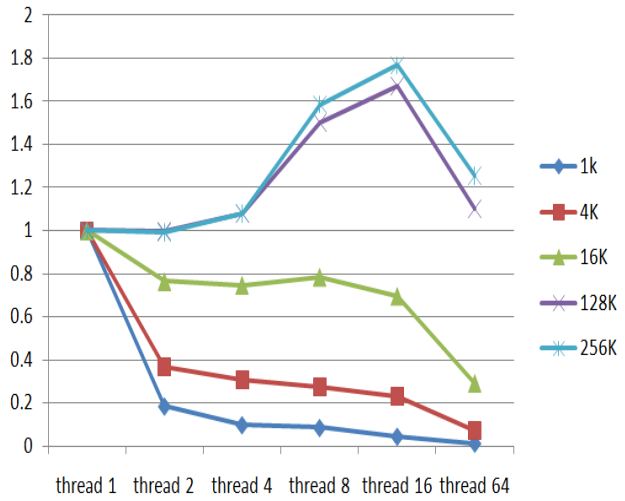
실험에 사용한 CPU는 Intel i7-2600K 3.40GHz이다. 운영체제는 Windows 7 64bit이며 컴파일러는 Visual C++ 2010 컴파일러를 사용하였다. 최적화 옵션은 /O2 옵션을 사용한다. OpenMP사용을 위해 /openmp 옵션을 사용하였다. Visual C++ 2010 컴파일러에서 사용하는 OpenMP 버전은 2.0이다.

쿼드트리를 구성하는 단말노드의 수를 1024(1K)개, 4096(4K)개, 16384(16K)개, 131072(128K)개, 262144(256K)개 로 다양하게 실험을 진행한다. 또한 쓰레드의 수를 1, 2, 4, 8, 16, 64개로 각각 병렬화된 프로그램을 BIOS설정을 통해 논리코어의 개수를 조절하여 1, 2, 4, 6, 8개의 논리코어에서 실험을 수행한다.

본 논문의 실험은 코어 개수와 쓰레드 개수의 관계를 파악하고 쓰레드 수에 따른 병렬화 오버헤드를 알아내는데 집중하기 위해 캐시 메모리의 영향을 최소화한다. 질의를 수행하기 전에 캐시 메모리의 데이터를 검색 연산과 관련이 없는 데이터로 교체한다. 또한, 여러 개의 질의가 연속적으로 발생하는 상황을 재현하기 위해 4개의 질의를 연속적으로 수행 후 시간을 측정한다.

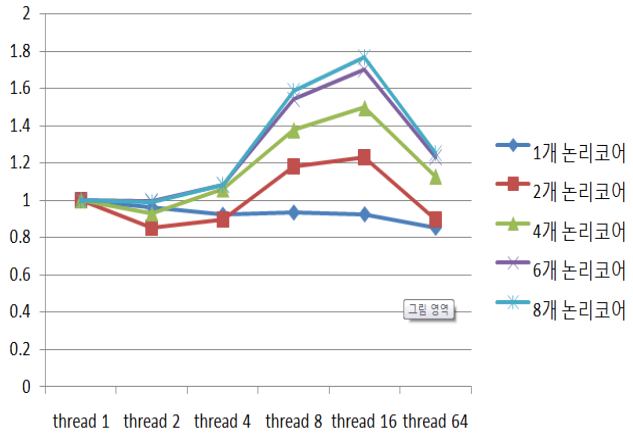
3.3 실험 결과

그림 4의 실험은 논리코어 8개를 사용하는 환경에서 진행한다. 쿼드 트리 크기별로 검색 성능 향상을 비교하여 각각 쓰레드 1개로 순차적으로 수행될 경우의 성능을 1로 하여 병렬화 성능향상을 비교한다. 1K, 4K, 16K개의 단말노드로 이루어진 쿼드트리에서는 검색의 쓰레드 개수를



(그림 4) 쿼드트리 크기별 병렬화 성능향상 비교

늘려 병렬화 하여도 쓰레드 생성과 제어에 따른 오버헤드가 병렬화 이득보다 크다. 128K, 256K개의 단말노드를 가진 쿼드 트리의 경우 쓰레드 개수가 16개 일 경우 약 1.6배와 1.8배의 성능향상 효과가 있다. 이 실험결과를 기초로 하여 논리 코어 개수별 실험은 병렬화 성능을 명확히 비교할 수 있도록 256K개의 단말노드로 이루어진 쿼드 트리에서의 검색 성능을 비교한다.



(그림 5) 논리코어 개수별 쓰레드 개수에 따른 병렬화 성능 향상 비교

그림 5는 각각 쓰레드 1개로 순차적으로 수행되는 성능을 1로 하여 코어 개수별 스레드 개수에 따른 성능향상을 비교한 결과이다. 논리코어의 개수가 2개 이상이라면 논리코어의 개수와 상관없이 일정한 쓰레드 개수에서 병렬화가 가장 효율적이다. 1개의 논리코어의 경우는 쓰레드가 많아질수록 쓰레드의 생성과 제어에 따른 오버헤드가 발생하여 성능이 나빠진다. 이후 2개 이상의 논리코어 환경부터는 논리코어의 개수와 상관없이 모두 쓰레드 개수가 16개일 때 성능향상의 효과가 가장 크다. 쓰레드 개수가 16개보다 많을 경우 쓰레드 생성과 제어의 오버헤드가 병렬화의 이득보다 더 커져 성능이 나빠진다.

4. 관련 연구

최근 들어 멀티코어 환경이 급속도로 대중화되면서 병렬 프로그램과 OpenMP를 이용한 병렬화에 대한 많은 연구가 진행되고 있다. Donghwan Jeon등은 순차적으로 수행되는 프로그램에서 병렬화 시 성능향상을 얼마나 가질 수 있는지 측정하는 프로그램인 Kismet에 대해 소개한다[7]. Tom St. John등은 트리 구조 메모리 모델에서 BFS를 병렬로 수행하는 방법을 제안한다[8]. Xavier Teruel등은 OpenMP 3.0에서부터 지원되는 테스크(task)에 관하여 분석을 수행한다[9]. Karl Furlinger와 Michael Gerndt은 OpenMP 어플리케이션에서 오버헤드와 확장성의 특징을 분석하는 방법론을 제시한다[10].

5. 결론 및 향후 연구 계획

최근 들어 멀티코어 프로세서가 대중화되고 처리되는 데이터들의 용량이 급속도로 커지기 시작하면서 병렬 프로그래밍에 대한 요구가 높아지고 있다. OpenMP는 컴파일러 디렉티브의 삽입만으로 병렬화를 간단하게 할 수 있다는 장점이 있지만 그에 따른 오버헤드가 존재한다.

본 논문에서는 순차적으로 수행되어야 하는 트리 자료구조를 이용한 데이터베이스 검색 프로그램을 OpenMP를 이용해 병렬화 하였다. 병렬화된 프로그램에서 논리코어의 개수와 쓰레드 개수의 관계를 파악하기 위한 실험을 진행하였다. 또한, 여러 개의 쓰레드로 병렬화를 진행할 경우 발생하는 오버헤드의 영향에 대해서 파악하기 위한 실험을 진행하였다. 실험 결과 코어 개수와 쓰레드 개수와의 관계는 미미하며 여러 개의 쓰레드로 병렬화 할 경우 쓰레드 생성과 제어에 따른 오버헤드가 발생하여 성능에에 영향을 준다는 것을 알 수 있었다.

본 논문의 연구 결과를 기초 삼아 향후 연구를 통해 OpenMP로 병렬화 시 성능향상 예측 모델을 확립하고자 한다. 이를 통해 병렬화 시 적정한 쓰레드 개수를 설정하는 의사 결정 모델을 적용하고 궁극적으로 OpenMP를 이용한 병렬화 시 최적의 성능향상이 이루질 수 있는 가이드라인과 개발 플로우를 구축하는 것이 목표이다.

논문 사사(acknowledgment)

본 논문은 2011년도 정부(교육과학기술부)의 한국연구재단의 기초 연구사업 지원에 의한 연구결과(No. 2011-0024909)입니다.

참고 문헌

[1] Multi-core processor , http://en.wikipedia.org/wiki/Multi-core_processor.

[2] Lars Nyland, Mark Harris and Jan Prins "GPU Gem3, Fast N-body simulation with cuda" Nvidia, pp 677-695.

[3] L. Duret, S. pencel, M. Gouy, F. Rechenmanm and G. Perriere " Tree pattern matching in phylogenetic trees : automatic search for orthologs or paralogs in homologous gens sequence databases" BIOINFORMATICS, Vol. 21, Issue 11, pp.2596-2603, 2005.

[4] OpenMP, http://en.wikipedia.org/wiki/OpenMP.

[5] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald and Ramesh Menon "Parallel Programming in OpenMP" Academic Press, 2001.

[6] Netzer, R. H. B., and B. P. Miller "What Are Race Conditions? Some Issues and Formalizations" ACM LOPLAS, Vol 1, Issue 1, March 1992.

[7] Donghwan Jeon,Saturnino Garcia,Chris Louie and Michael Bedford Taylor "Kismet: Parallel Speedup Estimates for Serial Programs" OOPSLA 11, Vol 46, Issue 10, pp.519-536, 2011.

[8] Tom St. John, Jack B. Dennis, and Guang R. Gao "Massively parallel Breadth First Search Using a Tree-Structured Memory Model", PMAM 2012, February 26.

[9] Xavier Teruel, Christopher Barton, Alejandro Duran , Xavier Martorell, Eduard Ayguade,Priya Unnikrishnan, Guansong Zhang and Raul Silvera "OpenMP Tasking Analysis for Programmers" CASCON'09, pp.32-42, 2009.

[10] Karl Furlinger and Michael Gerndt "Analyzing Overheads and Scalability Characteristics fo OpenMP Applications", VECPAR'06, July 2006.