멀티코어 시스템에서 병렬 프로그램 컴파일 성능 분석

김지홍*, 엄영익*
*성균관대학교 정보통신대학
e-mail : {jjilong, yieom}@skku.edu

A Performance Analysis of Parallel Program Compilation on Multi-core Systems

Jeehong Kim*, Young Ik Eom*
*College of Information and Communication Engineering, Sungkyunkwan University

요 약

멀티코어 환경이 보편화됨에 따라 병렬프로그래밍 기법과 이에 대한 컴파일 기술의 중요성이 더해지고 있다. 하지만 각 병렬 프로그래밍 기법과 컴퓨팅 환경에 따라 컴파일 기술이 개발되어야 하는 단점이 있다. 따라서 본 논문에서는 다양한 병렬 프로그래밍 기법과 컴퓨팅 환경을 지원할 수 있는 범용 컴파일러의 개발을 위해 병렬프로그래밍 기법과 컴파일러에 따른 병렬 프로그램의 컴파일 성능을 비교하고, 높은 수행 성능을 가진 범용 병렬 프로그램 컴파일러의 설계 방안을 모색한다.

1. 서론[1]

최근 컴퓨팅 환경에서의 프로세서는 싱글코어에서 멀티코어을 지나 매니코어로 진화하고 있다. 하지만, 이러한 프로세서의 코어 증가로 인한 기존 프로그램의 성능 향상은 미비하다. 이는 기존 프로그램이 싱글코어 프로세서에 최적화되어 개발되어, 멀티코어 프로세서에서의 병렬화 프로세싱을 고려하지 않았기 때문이다. 이를 극복하기 위해 OpenMP [1], Threading Building Block(TBB) [2]과 같은 병렬 프로그래밍 기법이 개발되었다. 하지만, 싱글코어 시스템의 기존 프로그램을 병렬 프로그래밍 기법으로 다시 개발해야하는 단점이 있다. 또한, 각 프로그램이 병렬 프로그래밍 기법으로 개발된다 하더라도, 각 병렬화 프로그래밍 기법을 지원하는 컴파일러의 개발이 각기 다른 컴퓨팅 시스템에 따라 동반되어야 하는 제한점이 존재한다.

이와 같은 병렬 프로그램의 단점은 다양한 하드웨어/소프트웨어 시스템에서 여러 병렬 프로그래밍 기법을 지원하는 범용 컴파일러의 개발로 극복할 수 있다 [3]. 여러 컴퓨팅 시스템을 위한 새로운 범용 병렬 프로그램 컴파일러의 개발은 많은 개발 자원을 필요로 한다. 이러한 문제는 충분히 발전되어온 기존의 범용 컴파일러 프레임워크를 기반으로 병렬 프로그램에 대한 범용 컴파일러를 개발함으로써 해결할 수 있다.

병렬 프로그래밍 기법을 지원하는 기존의 범용 컴파일러 프레임워크로는 GNU Compiler Collection(GCC) [4], Low Level Virtual Machine(LLVM) [5]이 있다. 이와 같은 범용 컴파일러는 중간언어(intermediate representation)를 이용하여 다양한 타겟 아키텍처의 기계어로의 변환 뿐아니라, 컴파일 과정에서 풍부한 최적화 기법을 적용할 수 있는 이점이 있다. 하지만, 이러한 기존의 컴파일러는 병렬화 기법에 따라 다른 성능을 보인다.

따라서 본 논문에서는 마이크로 벤치마크를 통해서 병렬프로그래밍 기법과 컴파일러에 따른 병렬 프로그램의 컴파일 성능을 비교하고, 높은 수행 성능을 가진 범용 병렬 프로그램 컴파일러의 설계 방안을 모색한다.

본 논문의 구성은 다음과 같다. 2 장에서는 병렬 프로그래밍 기법과 병렬 프로그래밍을 지원하는 컴파일러의 특성에 대해 설명한다. 3 장에서는 마이크로 벤치마크를 이용한 병렬 프로그램 컴파일 성능 측정 방법을 설명한다. 4 장에서는 실험 결과를 평가하고 이를 통해 범용 병렬 프로그램 컴파일러의 설계방안을 모색한다. 마지막으로 5 장에서 결론을 맺는다.

2. 병렬 프로그래밍과 컴파일러

2.1 병렬 프로그래밍

대표적인 병렬 프로그래밍 기법으로는 OpenMP [1], TBB [2] 가 있다. OpenMP [1] 는 여러 개의 프로세스가 공유된 메모리를 참조하는 환경에서 다중 스레드 병렬 프로그래밍을 위한 표준 스펙이다. 프로파일링을 통해 Hot 영역을 식별하고, 데이터의 유효 범위 등을 예측하여 컴파일러 지시자의 설정에 따라 다중 쓰레드 코드를 생성한다. Threading Building Block(TBB) [2]은 Intel 에서 개발한 C++ 병렬 프로그래밍 라이브러리이다. 단순히 기존의 쓰레드를 대체하는 것이 아니라, 플랫폼의 세부사항과 쓰레드 처리 매커니즘을

[1] "본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음" (NIPA-2013-(H0301-13-3001))

추상화한 태스크(task)를 기반으로 프로세서의 리소스를 효율적으로 사용하기 위해, 태스크를 쓰레드에 분할 정복(divide and conquer) 스케줄링하여 많은 테스크를 분산하여 병렬처리한다.

2.2 병렬 프로그래밍 지원 컴파일러

병렬 프로그래밍을 지원하는 컴파일러 대표적으로 GCC 와 LLVM 이 있다. GNU Compiler Collection(GCC) [4] 는 GNU 프로젝트의 일환으로 개발된 컴파일러로써 가장 널리 쓰이고 있다. YACC 기반의 파서가 프로그램을 분석하고. Generic 트리를 생성 및 최적화하여 중간언어를 생성한다. 생성된 중간언어는 후반부 컴파일러를 통해 타겟 아키텍처의 기계어로 변환하는 방식으로 동작한다. GCC 는 대부분의 병렬 프로그래밍 기법을 지원하기 때문에 병렬 프로그래밍 컴파일러로 많이 사용된다. LLVM [5] 은 모듈형태로 설계된 컴파일러 프레임워크이다. 전반부 컴파일러를 통해 장치 독립적인 중간언어인 LLVM IR 을 생성하고 이를 이용하여, 다양한 최적화를 수행하고 후반부 컴파일러를 통해 정적/동적 컴파일 방법으로 장치에 특화된 최적화 및 타겟 아키텍처의 기계어를 생성한다. LLVM 은 TBB 에 대해 병렬 프로그래밍 컴파일을 지원하고, OpenMP 에 대한 지원은 개발 진행중이다.

3. 실험

병렬 프로그래밍 기법과 컴퓨팅 환경에 독립적으로 높은 수행 성능을 가진 범용 병렬 프로그램 컴파일러의 설계 방안을 모색하기 위해, Pi 를 계산하는 마이크로 벤치마크를 OpenMP 와 TBB 로 32 개의 쓰레드에서 병렬 처리하도록 동일하게 작성하여 컴파일 성능을 측정하였다. OpenMP 로 작성된 마이크로 벤치마크는 구현방식에 따른 의존성을 줄이기 위해 Static 스케줄링 방식으로, OpenMP reduction 을 이용하여 각 쓰레드에서 수행한 결과를 Join 하였다. TBB 의 경우 grain size 는 auto_partitioner API 를 이용하였다. 실험은 Intel core i7-2600 3.4GHz, 메모리 8GB 의 하드웨어, Ubuntu 12.04 32bit 에서 GCC 4.6.3, LLVM 3.2 버전의 컴파일러를 이용하여 수행하였다. OpenMP 의 경우 LLVM 의 하부 프로젝트인 Dragonegg [6]를 이용하여 컴파일 성능을 측정하였다. 또한 최적화 레벨에 따른 컴파일 코드의 최적화 정도를 비교하기 위해 추가 실험을 수행하였다.

4. 평가

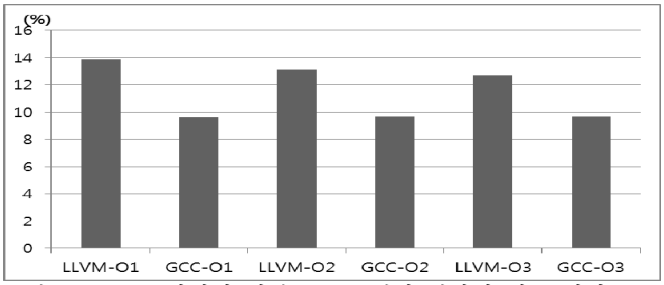

그림 1. TBB 로 작성된 병렬 프로그램의 컴파일 성능 향상도

그림 1 은 TBB 로 작성된 병렬 프로그램의 최적화 레벨에 따른 컴파일 성능의 향상도를 보여준다. 같은 최적화 레벨에서 LLVM 으로 컴파일된 병렬 프로그램의 성능이 약 4% 향상되었다. OpenMP 로 작성된 병렬 프로그램의 컴파일 성능은 같은 최적화 레벨에서 GCC 와 LLVM 으로 컴파일된 병렬 프로그램의 성능이 유사했다.

그림 2 은 TBB 로 작성된 병렬프로그램의 컴파일 수행시간을 나타낸다. LLVM 의 컴파일 수행시간이 GCC 의 컴파일 수행시간보다 약 8 배 느린 것을 확인하였다. 그림 1 와 그림 2 을 분석한 결과, TBB 로 작성된 병렬 프로그램을 LLVM 으로 컴파일할 경우 더높은 최적화 효율을 보이는 것을 확인할 수 있다. OpenMP 로 작성된 병렬 프로그램의 컴파일 수행시간은 LLVM 과 GCC 의 경우 유사했다.

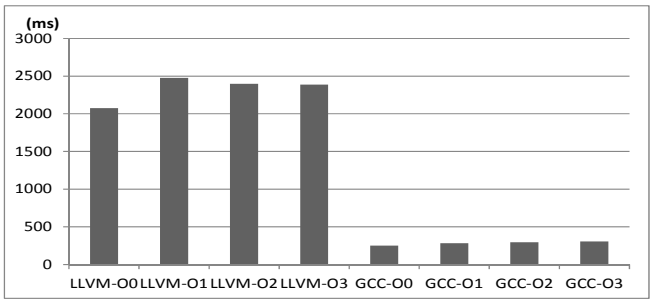

그림 2. TBB 로 작성된 병렬프로그램의 컴파일 시간

5. 결론

다양한 병렬 프로그래밍 기법과 컴퓨팅 환경을 지원할 수 있는 범용 컴파일러의 개발을 위해 병렬 프로그래밍 기법과 컴파일러에 따른 병렬프로그램의 컴파일 성능을 비교, 분석하였다. 이를 통해 OpenMP 로 작성된 병렬 프로그램의 경우 LLVM 과 GCC 이 유사한 컴파일 성능을 보이며, TBB 로 작성된 병렬 프로그램의 경우 LLVM 의 컴파일 성능이 GCC 보다 4% 향상된 것을 볼수 있었다.

참고문헌

[1] L. Dagum and R. Menon, “OpenMP: an industry standard API for shared-memory programming,” Computational Science and Engineering, IEEE, vol. 5, no. 1, pp.46-55, 1998

[2] J. Reinders, “Intel threading building blocks: outfitting C++ for multi-core processor parallelism,” O’Reilly Media, 2010

[3]ECMA International, Common Language Infrastructure(CLI), 4th edition, Dec, 2002

[4] R. M. Stallman, “Using and porting the gnu compiler collection,” Free Software Foundation, vol. 29, pp. 02111-1307 1989

[5] C. Lattner, V. Adve, “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation”, In international symposium on Code Generation and Optimization, pp.75-86, 2004

[6] Dragonegg - Using LLVM as a GCC backend, http://dragonegg.llvm.org/