ARM11MPCore에서 POSIX 쓰레드를 이용한 OpenMP 구현

이재원°1, 전우철 2, 하순회 1

1 서울대학교 전기컴퓨터공학부 통합설계및병렬처리연구실

{jwlee,sha}@iris.snu.ac.kr

2 서울대학교 기계항공공학부 bk21 체세대 기계항공 시스템 창의 설계 인력양성 사업단

wcjeun@iris.snu.ac.kr

OpenMP Implementation using POSIX thread library on ARM11MPCore

Jaewon-Lee°1, Woochul-Jeun 2, Soonhoi Ha1

1CAP Lab, School of EECS, Seoul National Univ.

2RTOS Lab, School of MAE, Seoul National Univ.

요 약

멀티프로세서 환경에서 OpenMP는 MPI 에 비해 병렬 프로그래밍을 쉽게 할 수 있다는 장점을 가지고 있고, OpenMP는 표준이 없는 병렬 프로그래밍 세계에서 실질적인 표준으로써 인정받고 있다. OpenMP는 대상 플랫폼에 따라 OpenMP 구현을 다르게 해야 하기 때문에 새로운 프로세서가 등장하면 그에 맞는 OpenMP구현을 만들어야 한다. 이 논문에선 다중 프로세서 시스템-온-칩 시스템인 ARM11MPCore 시스템 위에 POSIX 쓰레드에 기반하여 OpenMP 환경을 구축하고 그 성능을 측정한다.

1. 서 론

현재 클락 수를 높이거나 집적 도를 높이는 등의 방법으로는 더 이상의 성능 향상을 꾀하기가 힘들어지고 있다. 이에 따라 다수의 코어를 사용하여 수행 속도를 향상시키는 다중 프로세서 시스템-온-칩(Multiprocessor System-on-Chip) 시스템들에 대한 연구가 활발히 진행되고 있다. 이런 여러 다중 프로세서 시스템-온-칩 시스템들 중 ARM사의 ARM11MPCore [3]는 전력 관리와 인터럽트 분배기 (interrupt distributor)를 이용한 여러 프로세서 관리의 편의성, 그리고 Snoop Control Unit을 이용한 각 프로세서간의 캐쉬 메모리 일관성 유지 용이, 멀티미디어 어플리케이션에 강점 등으로 잘 알려져 있다.

하지만 이런 다중 프로세서 시스템을 위해서는 일반적인 순차적 프로그래밍과는 다른 병렬적 프로그래밍이 필요하다. 이런 병렬적인 프로그래밍의 모델에는 메시지 전달(message-passing) 모델과, 공유 메모리 모델이 있다. 메시지 전달 모델은 각각 프로세서들이 자신만의 고유한 메모리 영역을 가지고 있는 상태에서, 데이터 교환이 필요할 경우에 다른 프로세서와 메시지를 통해 교환을 하는 방식이다. 이 방식은 어떤 데이터를 넘겨야 할 지를 프로그래머가 세세히 지정해 줄 수 있어서 최적화를 하는데 유리하지만 모든 데이터의 전송을 관리해야만 하기 때문에 프로그래밍하기가 매우 어렵다는 단점을 가지고 있다. 이 방식은 물리적으로 분산되어 있는 클러스터 시스템에서 널리 쓰이고 있고, 현재 가장 표준으로 인정받는 인터페이스는 메시지 전달 인터페이스 (MPI: Message Passing Interface) [3] 이다. 또 다른 방식인 공유 메모리 모델은 모든 CPU들이 같은 메모리 공간을 볼 수 있도록 별도의 메모리 일관성 모델을 요구한다. 하지만 이로 인해 응용 프로그래머가 메모리 일관성에 대해 신경 써야 할 여지는 줄어들어 메시지 전달 모델보다는 프로그래밍을 매우 쉽게 할 수 있다. 따라서 공유 메모리 모델에서는 OpenMP[2]가 사실상의 표준의 자리를 차지하고 있다.

이런 OpenMP를 사용하기 위해서는 목표 플랫폼에 맞는 OpenMP 구현과 그 구현에 맞게 C코드를 변환해 주는 OpenMP 변환기 (translator) 가 필요하다 [11]. 사실 OpenMP 인터페이스는 OpenMP 디렉티브를 컴파일러가 어떻게 처리해야 하는지에 대해서만 정의

하기 때문에 OpenMP 디렉티브의 직접적인 구현은 구현하는 사람과 구현 대상 플랫폼에 따라 다르다. 결국 여러 대상 플랫 폼에 대해 여러 개의 OpenMP 구현이 존재 한다. 이렇게 OpenMP를 대상 플랫폼에 맞게 구현한 논문이 여러 가지가 있다[5-11]. 다중 프로세서 시스템-온-칩 시스템의 경우는 POSIX 쓰레드를 이용해서 OpenMP를 구현한 경우[9]와 OS없이 Cradle MPSoC 보드에서 OpenMP 환경을 구축한 사례가 있다[8,9,11].

본 논문은 다중 프로세서 시스템-온-칩 시스템인 ARM11MPCore 시스템 위에 POSIX 쓰레드에 기반한 방법으로 OpenMP 환경을 구축하고 그 성능을 측정한다.

논문의 구성은 다음과 같다. 2 장에서는 OpenMP와 , 대상 시스템인 ARM11MPCore 관한 명세에 대해 소개한다. 3장에선 본 논문에서 구현한 ARM11MPCore용 OpenMP 환경을 설명한다. 4장에선 이 시스템 위에서의 실험 결과를 분석하고 마지막 5장에선 결론과 앞으로 연구해야 할 일에 대해 소개한다.

2. 배경 설명

2.1 ARM11MPCore

ARM11MPCore 시스템은 ARM사의 ARM11MPCore 로직 타일과 에뮬레이션 보드 (EB : Emulation Board)를 결합하여 함께 동작하도록 되어있다[3]. ARM11MPCore는 200MHz ARM11 프로세서 4개와, 32KB L1 명령어 캐쉬, 32KB L1 데이터 캐쉬, 1MB L2 공유 캐쉬, 인터럽트 분배기를 내장하고 있다. EB는 Character LCD나 마우스, 키보드 등의 기타 주변장치와 256MB DDR SDRAM, 64 MB 의 NOR flash, 64Mb NAND Disk-on-Chip flash를 가지고 있다. 4개의 프로세서들이 개별적으로 가지고 있는 L1 캐쉬의 일관성은 SCU (Snoop Control Unit) 가 관리한다. ARM11MPCore의 간략한 구조도[3]는 그림 1과 같다.

ARM11MPCore는 어플리케이션의 동작 상태에 따라 CPU들을 완전히 멈추도록 할 수 도 있어서, 성능과 전력 소모를 조절 할 수 있다. 또한 각 프로세서에 대해 SMP[Symmetric Multi-Processor] 모드와 AMP [Asymmetric Multi-Processor] 모드 중 선택이 가능하며, 프로그래머가 실제 사용되는 코어 수에 관계없이 하나의 프로세서로 생각할 수 있게 해주는 프로세서 인터페이스를 제공한다.

현재 ARM사 홈페이지에서 ARM11MPCore 에서 수행 가능한 SMP 리눅스를 배포하고 있다[4]. SMP 리눅스 상에서는 생성된 쓰레드를 자동으로 다른 프로세서에 매핑하는 기능을 제공하기 때문에, 여러 CPU들에 대한 특별한 고려 없이 쓰레드를 띄우는 것만으로도 Multi core 환경을 이용할 수 있게 된다.

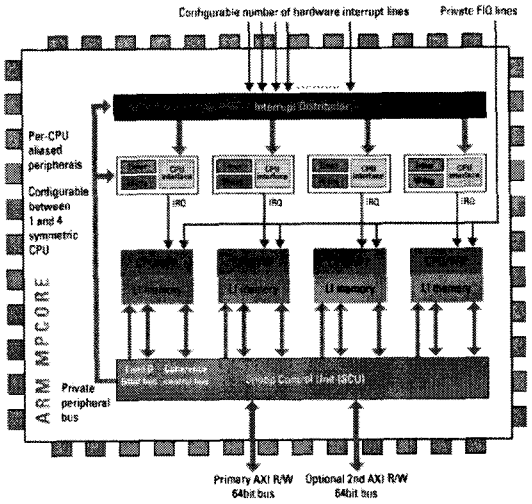

그림 1 MPCore 구조도

2.2 OpenMP

OpenMP는 C/C++, Fortran 언어에 대해 컴파일러 디렉티브를 어떻게 처리 할지를 명세 한다. 이 명세를 통하여 간단한 컴파일러 디렉티브의 삽입만으로 순차적인 코드를 병렬적인 코드로 바꿀 수 있다는 장점을 가지고 있다[13].

OpenMP 환경은 OpenMP 변환기와 OpenMP구현의 두 가지 부분으로 구성된다. OpenMP 변환기는 OpenMP 디렉티브가 달려있는 소스 코드를 OpenMP 구현에 맞는 코드로 바꿔주는 역할을 하고, OpenMP 구현은 OpenMP의 목적 플랫폼 안에서 변환된 코드가 수행될 수 있는 환경을 제공해준다. 이런 플랫폼 의존적인 성격 때문에 OpenMP 구현은 목적 플랫폼에 따라 새롭게 만들어야 한다 [11].

OpenMP 변환기를 통해 생성된 코드는 fork/join 모델을 따른다. 먼저 병렬 구간을 만나기 전까지의 순차적인 부분은 하나의 마스터 쓰레드 (master thread)가 수행한다. 그러다가 PARALLEL 디렉티브를 만나게 되면 자식 쓰레드(child thread)를 생성 후 정해진 규칙에 따라 일을 분배해 주게 된다. 그리고 나서 병렬 구간이 끝나게 되면 자식 쓰레드를 join 시킨다. 그 이후 마스터 쓰레드는 PARALLEL 구간의 다음 부분을 진행시킨다.

3. 구현

OpenMP 변환기는 기존의 OpenMP 변환기를

사용하였다 [11]. OpenMP 구현은 클러스터 시스템을 위한 기존의 OpenMP 구현[12]을 POSIX 쓰레드만으로 동작하도록 수정하여 사용하였다. 즉, 클러스터 PC들간 자료 교환 때 쓰이던 메시지 전달 인터페이스로 구현된 부분을 모두 제거하고, 현재 ARM LINUX 에서 지원하는 POSIX 쓰레드를 사용하도록 수정하였다.

4. 실험

4.1 EPCC

EPCC[14]는 각각의 OpenMP 디렉티브들의 오버헤드를 측정하는 프로그램이다. 우리가 만든 MPCore용 OpenMP 디렉티브를 분석해보기 위해서 EPCC 프로그램을 수행한 결과는 표 1 과 같다.

이 결과를 보면 코어의 수가 증가함에 따라 각 OpenMP 디렉티브들의 오버헤드가 함께 증가하는 것을 볼 수 있다. 이런 결과가 나타나는 이유는 코어의 수가 증가하면 여러 쓰레드 들이 하나의 공유 변수에 동시에 접근할 경우 쓰레드들 간의 충돌이 일어나게 되기 때문이다.. 여기에 수행 시 각 쓰레드들 간의 동기화 시에도 다른 쓰레드가 일을 끝마칠 때까지 기다리는 시간이 늘어나게 되어 코어수의 증가, 즉 쓰레드 수의 증가는 오버헤드의 증가로 나타나게 되는 것이다.

표 1 EPCC 테스트 결과 (단위: us)

디렉티브 \ 코어수	1	2	4
PARALLEL	531.3	18789.9	21931.7
FOR	19.5	99.5	176.3
BARRIER	12.3	84.2	154.5
SINGLE	14.3	85.5	158.6
CRITICAL	2.2	29.3	20.8
ATOMIC	1.36	29.9	37.1
REDUCTION	557.2	19588.1	21535.0

ARM11MPCore 환경에서 PARALLEL 디렉티브와 REDUCTION 디렉티브의 오버헤드가 다른 디렉티브 들에 비해 크게 나타난다. 이 것은 프로그래머가 이 디렉티브 들을 쓸 때의 비용을 예측하여야 OpenMP 로 프로그래밍을 하였을 때 좋은 성능을 얻게 될 수 있음을 의미한다.

4.2 MATMULT

표 2 는 2차 정방 행렬을 곱하는 간단한 곱셈 예제에 OpenMP를 적용하여 코어 수에 따른 수행시간을 측정한 결과이다. 예제에는 결과물을 검증하는 부분을 두어 나온 결과 값이 맞는지를 확인하였다.

행렬 곱셈 예제는 단 한번만 쓰레드를 생성하여 각자 프로세서에서 연산을 하고 그 결과를 합치기 때문에 쓰레드 생성 오버헤드를 크게 발생시키지 않아서 코어 수 증가에 따른 성능 개선 효과를 볼 수 있다.

하지만 행렬의 크기가 작은 64* 64일 때, 코어 수를 두 개에서 네 개로 늘리면 성능저하가 나타난다. 이것은 쓰레드 생성으로 인한 오버헤드가 병렬적으로 수행함으로써 얻게 되는 비용 절감보다 크기 때문에 수행시간이 증가 한다는 결론을 얻게 된다. 이 것은 4.1의 EPCC 테스트 결과에서 PARALLEL 디렉티브의 오버헤드가 의미하는 바와 같다.

표 2 행렬 곱셈 (단위:sec)

행렬 크기 \ 코어수	1	2	4
64 * 64	0.049	0.038	0.062
128 * 128	0.274	0.150	0.093
256 * 256	2.308	1.258	0.821

4.3 CG (conjugate gradient computation) 예제

표 3은 NAS 벤치마크 프로그램[15]에서 쓰이는 CG 예제에 OpenMP 디렉티브를 적용시킨 후 실행한 결과이다.

표 3 CG (단위:sec)

코어수	1	2	4
시간	530.8	481.5	481.7

CG 예제에서는 코어 수에 따른 성능 향상을 뚜렷하게 관측 할 수 없었다. CG 예제는 쓰레드 생성 횟수가 굉장히 많다. 이에 관한 OS 가 쓰레드 생성시 4 개의 코어 중 랜덤 하게 선택하여 일을 할당 시켜 버리기 때문에 L1 캐쉬의 Locality 를 이용하기도 힘들다. 그렇기 때문에 코어 수를 늘린다고 해서 그 것이 바로 수행속도 증가로 연결되지 않는다.

4.4 H.263 인코더

이 논문에서 쓰인 OpenMP 시스템이 좀 더 복잡한 응용 프로그램에서도 동작하는지를 확인하기 위하여 176*144 크기의 QCIF 포맷 동영상을 OpenMP 디렉티브를 적용시킨 H.263 인코더를 통해 인코딩 하였다.(표 4)

수행 결과가 올바름을 확인하기 위해 Output 으로 나온 파일들을 H.263 디코더를 통하여 본 결과 원래 이미지와 일치하는 것을 확인하였다. 하지만 수행시간은 코어의 수가 늘어남에 따라 같이 증가하는 경향을 보였다. 이 것의 원인은 H.263 인코더를 이용해 인코딩한 동영상 파일의 해상도가 너무 작아 병렬처리를 통해 얻는 이득이 쓰레드 생성시에 드는 오버헤드보다 작기 때문으로 추정된다. 좀 더 큰 해상도의 동영상을 인코딩 한다면 코어 수를 증가 시킴에 따라 성능 향상을 기대할 수 있을 것이다.

표 4 QCIF 동영상의 H.263 인코딩 (단위 : 초)

코어수	1	2	4
수행 시간	17.164	17.967	18.113

5. 결론

이 논문에선 다중 프로세서 시스템-온-칩 시스템인 ARM11MPCore 시스템 위에 POSIX 쓰레드에 기반하여 OpenMP 환경을 구축하고 그 성능을 측정하였다. 또한 우리가 구현한 OpenMP 환경을 검증하기 위해 H.263 같은 복잡한 예제들도 돌려보았고 정상적으로 동작함을 확인하였다.

또한 ARM11MPCore 에서 OpenMP 를 이용한 병렬처리 프로그램을 돌릴 때 쓰레드 생성 횟수나 병렬처리 구간의 크기에 따라 성능이 향상되거나 저하되는 것을 확인할 수 있었다.

앞으로 남은 과제는 EPCC 에서 나온 PARALLEL 디렉티브 오버헤드를 토대로 ARM11MPCore 시스템의 OpenMP 를 쓴 응용 프로그램 내에서 코어 수 증가가 성능 향상으로 이어지도록 하는 것이다. 또한 OS 없는 환경 하에서 ARM11MPCore 용 OpenMP 환경을 구축하여 OS 를 사용할 때와 비교하여 OS 스케줄로 인한 오버헤드를 측정해 보는 것이다.

6. 감사의 글

본 연구는 BK21 프로젝트, 과학기술부 도약연구지원사업(R17-2007-086-01001-0), 삼성전자에 의해 지원되었다. 또한 서울대학교 컴퓨터신기술연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 또한 한국전자통신연구원의 SoC 핵심설계인력양성사업에 의해 부분적으로 지원되었다.

참고 문헌

[1] Message Passing Interface Forum, "MPI: A message -passing interface standard", *International Journal of Supercomputer Applications and High Performance Computing*, Vol.8, No.3/4, pp159-416, 1994

[2] OpenMP Architecture Review Board, "OpenMP C and C++ application program interface" http://www.openmp.org

[3] ARM11MPCore http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html

[4] ARM Linux operating system http://www.arm.com/linux/linux_download.html

[5] Christian Brunschen and Mats Brorsson, "OdinMP/CCp - A portable implementation of OpenMP for C,"(EWOMP'99) In the proceeding of first European Workshop on OpenMP, Lund, Sweden, Sept.1999. pp.21-26

[6] Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano, and Yoshio Tanaka, "Design of OpenMP Compiler for an SMP Cluster," EWOMP'99, In the proceeding of 1st EWOMP,Lund, Sweden, Sept. 1999, pp,32-39.

[7] vassilios V. Dimakopoulos and Elias Leontiadis,"A portable C compiler for OpenMP V.2.0" EWOMP 2003, In the proceeding of 5th European Workshop on OpenMP, Aachen, Germany, Sept.2003, pp.5-11

[8] Feng Liu and Vipin Chaudhary, "A Practical OpenMP Compiler for a System-on-Chips," WOMPAT 2003, LNCS 2716,pp. 54-68, 2003.

[9] Yoshihiko Hotta, Mitsuhisa Sato, Yoshihiro Nakajima, Yoshinori Ojima, "OpenMP Implementation and Performance on Embedded Rene as M32R Chip Multiprocessor," EWOMP 2004, Stockholm, Sweden, Oct. 2004, pp.37-42

[10] Feng Liu and Vipin Chudhary, "Extending OpenMP for Heterogeneous Chip Multiprocessors," Proceedings of the 2003 International Conference on Parallel Processing, Kaohsiung, Taiwan, Oct. 2003, pp.161-

[11] Woo-chul jeun and Soonhoi Ha, "Implementation and Translation of Major OpenMP Directives for Chip Multiprocessor without using OS)

[12] Yang-Suk Kee, Jin-Soo Kim, and Soonhoi Ha, "ParADE: An OpenMP Programming Environment for SMP Cluster Systems," ACM/IEEE Supercomputing (SC'03), Nov 12-15,2003

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

[14] EPCC OpenMP Micro benchmarks 1.0, http://www.epcc.ed.ac.uk/research/openmpbench.1999

[15] NAS Parallel Benchmarks , http://www.nas.nasa.gov/Resources/Software/npb.html