2000 년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

OpenMP 명세에 대한 고찰 및 분석

이종우*, 박찬영*
*한림대학교 정보통신공학부
e-mail : jwlee44@sun.hallym.ac.kr

Survey and Analysis of OpenMP Specifications

Jong-Woo Lee*, Chan-Young Park*
*Division of Information and Communication Engineering, Hallym University

요 약

메시지 전달 방식과 공유 메모리 방식은 병렬 컴퓨터 시스템을 위한 대표적인 아키텍처이다. 이 중 공유 메모리 방식은 프로그래밍의 용이함으로 인해 메시지 전달 방식에 비해 많이 채택되고 있는 실정이다. 하지만 하드웨어 벤더마다 각기 다른 공유 메모리 프로그래밍 인터페이스를 제공하기 때문에, 코드 호환성이 주 관심사인 경우에는 프로그래밍의 불편함을 감수하면서 MPI 나 PVM 등을 이용한 메시지 전달 구조를 채택하는 경우가 자주 발생한다. 본 논문에서는 공유 메모리 병렬 컴퓨터 시스템을 위한 프로그래밍 인터페이스 표준인 OpenMP 명세에 대해 고찰·분석한 결과를 제시한다. OpenMP 명세의 등장 배경 및 발전 과정 등을 기술하고, OpenMP 명세의 분분별 규정 내용을 요약한다. 또한 OpenMP 명세에 따라 기존 C 프로그램을 수정한 예도 보인다. 본 논문의 목적은 OpenMP 라는 공유 메모리 프로그래밍 인터페이스 표준을 소개하고, 이에 대한 관심을 높임으로써 관련 연구를 활성화 시키는데 있다.

1. 서론

공유 메모리 프로그래밍 모델을 표준화하기 위한 노력은 과거 ANSI X3H5 표준안[1]이 그 시초라고 할 수 있다. 하지만 MPP 구조를 갖는 분산 메모리 시스템이 각광을 받으면서 X3H5 는 병렬 프로그래밍의 표준으로 그리 많이 채택되지는 못했다. 여러 하드웨어 제작사들에 의해 지원되기는 하였으나, X3H5 는 근본적으로 루프(loop) 병렬성 이외의 부분에서는 많은 약점을 지니고 있었기 때문이다. 결과적으로 이 모델을 채택한 응용들은 병렬 시스템의 확장성을 확보하는 데에 실패하였다.

메시지 전달 프로그래밍 모델은 MPI 의 등장으로 인해 효과적으로 표준화되어 호환성 있는 병렬 프로그램 작성에 기여함으로써 메시지 전달 프로그램 작성의 표준으로 자리 잡았다고 할 수 있다. 하지만 메시지 전달은 프로그래밍 자체가 쉽지 않다는 단점을 갖고 있다. 프로그램의 자료 구조들이 프로그래머에 의해 인위적으로 분할되어야 하고, 분할된 자료 구조들을 처리하는 프로세스들 또한 프로그래머에 의해 직접 병렬화 되어야 한다. 게다가 최근의 다중처리기 시스템들은 하드웨어적인 캐쉬 일관성 유지 메커니즘을 제공하기 때문에, 메시지 전달 방식은 점차적으로 그 효용성을 잃고 있는 추세이다.

Pthreads 는 스레드 수준의 공유 메모리 프로그래밍을 위한 POSIX 표준이다. 하지만 결정적으로 이 표준안에는 Fortran 언어가 포함되어 있지 않다. 또한 규정 자체가 응용 프로그래머를 위한 것이라기 보다는 시스템 프로그래머를 위한 시스템 수준을 목표로 하고 있다. 태스크 병렬성은 고려하고 있지만 데이터 병렬성은 지원되지 않는다. 이 외에도 병렬 계산을 위해 여러 언어들이 등장하였지만 널리 사용된 것은 별로 없는 실정이다. Fortran 의 변형된 형태로 다중 처리에 적합하게 개발된 HPF 정도가 현재 사용되고 있는 실정이다.

현재 정부 산하 연구소나 과학 계산용 소프트웨어 개발자들에 의해 개발된 많은 프로그램들이 병렬화되어야 할 필요성을 갖고 있음에도 불구하고 병렬화에 어려움을 겪고 있는 것도 편하고 표준화된 병렬 프로그래밍 환경이 없기 때문이라고 할 수 있다. 이러한 필요성에서 등장한 것이 OpenMP 명세인데 이는 공유 메모리 프로그래밍 환경을 지향하고 있으며, 컴파일러와 라이브러리 수준에서 지원되어야 할 인터페이스들을 규정하고 있다. 또한 OpenMP 규정에는 Fortran 뿐

만 아니라 C/C++ 언어에 대한 API 도 포함되어 있다. 표 1 은 여러 병렬 프로그래밍 표준 모델들을 비교한 결과[2]를 보이고 있다.

표 1. 여러 병렬 프로그래밍 표준 모델의 비교 결과

	X3H5	MPI	Pthreads	HPF	OpenMI
확장성	no	yes	sometimes	yes	yes
근계적 병렬화	yes	no	no	no	yes
이식성	yes	yes	yes	yes	yes
포트란 지원	yes	yes	no	yes	yes
High level	yes	no	no	yes	yes
데이터 병렬성	yes	no	no	yes	yes
성능 고려	no	yes	no	tries	yes

2. OpenMP

OpenMP 는 컴파일러 디렉티브(directive)들과 런타임 라이브러리 루틴들의 집합이라고 할 수 있다. 공유 메모리 병렬성을 표현할 수 있도록 기존의 컴파일러 기능과 런타임 라이브러리 루틴을 확장하도록 요구하고 있는 것이다. 특정 언어를 사용하도록 하는 제약은 없으며, 현재 포트란과 C/C++ 언어를 위한 OpenMP 패키지들이 출시되어 있다. OpenMP 에 대한 상세 자료는 관련 홈페이지인 http://www.openmp.org 에서 얻을 수 있다.

2.1 OpenMP 의 특징 요약

OpenMP 의 각 부분별 규정을 살펴보기에 앞서 OpenMP 의 특징을 요약하면 다음과 같다.

- OpenMP 는 널리 통용되는 산업 표준으로서 이 표준을 준수하여 제작된 프로그램은 OpenMP 를 지원하는 어떤 플랫폼에서도 수행 가능하다.
- OpenMP 를 사용하면 기존 병렬 프로그램을 새로 설계할 필요 없이 쉽게 공유 메모리 병렬 프로그램으로 수정할 수 있다.
- 병렬화의 단위를 프로그래머가 자유로이 설정할 수 있으며 풍부한 컴파일러 디렉티브를 이용해 컴파일러에게 부담을 주지 않으면서 다양한 형태의 병렬 코드를 생성할 수 있다.
- 순차 코드를 병렬 코드로 변환하는 데에 필요한 거의 모든 기능들을 지원하기 때문에 병렬 코드로 완벽하게 변환할 수 있다.

다음 절에서는 C/C++ 명세를 중심으로 OpenMP 의 부분별 규정 내역을 기술한다.

2.2 OpenMP 의 부분별 규정 – 컴파일러 디렉티브

OpenMP 에서는 컴파일러 디렉티브를 정의하기 위해 #pragma 를 사용한다. 각 디렉티브는 "#pragma omp *directive-name* [*clause*[*clause*]...]" 형태로 정의된다.

2.2.1 Parallel 디렉티브

```
#pragma omp parallel [clauses]
{    /* ... */    }
```

의 형태로 정의되며, 병렬 실행의 기본 코드 단위를 설정하기 위해 사용된다. 종속절(clause)에는 `if`, `shared`, `private`, `firstprivate`, `copyin`, `reduction` 등의 문장이 올 수 있다. 예를 들어, `if (expression)`이 종속절 부분에 있을 경우 `expression` 이 참이어야만 코드 블록이 병렬 수행된다. 거짓이면 순차적으로 실행된다. 종속절에 대한 자세한 설명은 [3]을 참고하기 바란다.

2.2.2 For 디렉티브

```
#pragma omp for [clauses]
for (var=lower_bound; var<upper_bound; var+=increment)
    statement;
```

의 형태로 정의되며, `for` 문에 의한 반복 수행들을 여러 스레드들에게 분배하기 위해 사용된다. 종속절에는 `private`, `firstprivate`, `lastprivate`, `reduction`, `ordered`, `schedule`, `nowait` 등의 문장들이 올 수 있다. 예를 들어, `schedule(kind [, chunk_size])`은 루프를 스케줄링하기 위해 사용할 스케줄러와 스케줄 단위를 설정한다. 각 스레드들에 할당된 루프들 사이에는 묵시적 barrier 가 존재하며 `nowait` 종속절을 사용하면 묵시적 barrier 를 없앨 수 있다.

2.2.3 Sections 디렉티브

`Sections` 디렉티브는 여러 스레드들에게 서로 다른 코드 블록들을 동시에 실행시키기 위해 사용되며, 다음과 같이 정의된다.

```
#pragma omp sections [clauses]
{
    [#pragma omp section]
      structured-block
    [#pragma omp section
      structured-block
      ....
    ]
}
```

`Sections` 디렉티브로 분리된 각 *structured-block* 간에도 묵시적 barrier 가 존재한다. 종속절에는 `private`, `firstprivate`, `lastprivate`, `reduction`, `nowait` 등의 문장이 올 수 있다.

2.2.4 Single 디렉티브

병렬 수행 코드로 지정된 블록 내의 어떤 코드 부분이 반드시 하나의 스레드에 의해서만 수행되어야 하는 경우 해당 코드 블록을 지정할 때 사용된다.

```
#pragma omp single [clauses]
  structured-block
```

종속절에는 `private`, `firstprivate`, `nowait` 등의 문장이 올 수 있다.

2.2.5 혼합 디렉티브

혼합 디렉티브로는 `parallel for` 디렉티브와 `parallel sections` 디렉티브가 존재하며 의미는 두 디렉티브의 의미를 합해 놓은 것과 같다. 주의할 점은 `parallel single` 디렉티브는 존재할 수 없다는 점이다.

2.2.6 마스터 & 동기화 디렉티브

```
#pragma omp master
   compound-statement
```

마스터 디렉티브로 지정된 문장들은 응용 프로그램 수행에 참가한 스레드들 중 마스터 스레드에 의해서만 실행된다. 한편 스레드 간 동기화를 위해서 OpenMP 는 `critical`, `barrier`, `atomic`, `flush`, `ordered` 디렉티브들을 지원한다.

- `#pragma omp critical [(name)]`
  compound-statement
  단 하나의 스레드 만이 수행할 수 있도록 보장한다. 여기서 *name* 은 동기화 객체 이름 역할을 한다.
- `#pragma omp barrier`
  모든 스레드들이 이 지점에 도달할 때까지 기다린다.
- `#pragma omp atomic`
  `var` *op* = *expression*
  치환 연산이 atomic 하게 이루어지도록 보장한다.
- `#pragma omp flush [(`*list*`)]`
  여러 스레드들에 의해 공유되고 있는 변수들(*list* 에 지정)을 최신 값으로 갱신한다.
- `#pragma omp ordered`
  compound-statement
  `for` 디렉티브 내에서만 사용될 수 있으며 지정된 복합 문장들이 나열된 순서대로 반드시 수행되는 것을 보장한다.

2.2.7 Threadprivate 디렉티브

`#pragma omp threadprivate (`*list*`)` 디렉티브는 *list* 에 지정된 전역 변수에 대해 스레드마다 독자적인 복사본을 유지하고자 할 때 사용된다. 전역 변수의 복사본을 보유한 스레드가 그 변수를 수정하면 자신의 복사본에만 영향을 미친다.

2.3 OpenMP 의 부분별 규정 – 런타임 라이브러리

OpenMP 에 규정된 실행-시간 라이브러리들은 크게 두가지 종류로 분류할 수 있다. 첫째는 실행 환경(execution environment) 관련 함수들이고, 둘째는 동기화(lock) 함수들이다. 표 2 와 표 3 은 이들 각각에 속한 함수들의 종류와 간략한 기능 요약을 보이고 있다.

2.4 OpenMP 의 부분별 규정 – 환경 변수

OpenMP 에서는 총 4 개의 환경 변수를 규정하고 있는데, 대부분 실행-시간 라이브러리 함수로 지정되지 않았을 경우에 대비한 디폴트 행동을 제어하기 위해 사용된다.

표 2. OpenMP 의 실행 환경 함수 종류 및 기능

함수 이름	기능
omp_set_num_threads	병렬 코드 블록에 사용될 스레드 개수 지정
omp_get_num_threads	현재 병렬 코드 블록에 참가하도록 설정되어 있는 스레드 개수를 얻는다.
omp_get_max_threads	병렬 코드 블록에 지정 가능한 스레드 최대 개수를 얻는다.
omp_get_thread_num	실행 중인 스레드의 번호를 얻는다. 스레드 번호는 0 ~ omp_get_num-threads()-1 중의 하나일 것이다.
omp_get_num_procs	프로그램에 할당 가능한 처리기의 최대 개수를 얻는다.
omp_in_parallel	현재 수행 중인 부분이 병렬 수행 부분이면 true 를 리턴한다.
omp_set_dynamic	어떤 병렬 수행 구간에 대해 그 구간에서 사용 가능한 스레드의 개수를 동적으로 조정한다.
omp_get_dynamic	스레드 개수가 동적으로 조정된 구간이면 true 를, 아니면 false 리턴.
omp_set_nested	병렬 수행 구간이 중첩되는 것에 대한 허용 여부를 결정
omp_get_nested	중첩 병렬 수행이 허용된 구간이면 true 를 리턴함.

표 2. OpenMP 의 동기화(lock) 함수 종류 및 기능

함수 이름	기능
omp_init_lock	하나의 simple lock 을 초기화한다.
omp_destroy_lock	지정된 simple lock 을 파괴한다.
omp_set_lock	지정된 simple lock 을 얻을 때까지 기다린다.
omp_unset_lock	지정된 simple lock 을 반환한다.
omp_test_lock	지정된 simple lock 의 획득 가능 여부를 검사한다.
omp_init_nest_lock	하나의 중첩가능 lock 을 초기화한다.
omp_destroy_nest_lock	지정된 중첩가능 lock 을 파괴한다.
omp_set_nest_lock	지정된 중첩가능 lock 을 얻을 때까지 기다린다.
omp_unset_nest_lock	지정된 중첩가능 lock 을 반환한다.
omp_test_nest_lock	지정된 중첩가능 lock 의 획득 가능 여부를 검사한다.

- `OMP_SCHEDULE` : `for`, 또는 `parallel for` 디렉티브를 제어하기 위해 사용.

- OMP_NUM_THREADS : 병렬 블록에 참가할 수 있는 디폴트 스레드 개수.
- OMP_DYNAMIC : 병렬 블록에 참가한 스레드 수의 동적 조정 가능 여부를 나타냄.
- OMP_NESTED : 중첩된 병렬 블록 허용 여부를 나타냄.

3. OpenMP 프로그래밍 예

다음에 보일 OpenMP 프로그래밍 예는 Lund 대학에서 OpenMP 를 구현한 OdinMP/CCp 컴파일러[4]를 이용해 기존 C 프로그램을 OpenMP C 프로그램으로 수정한 경우를 보이고 있다. 그림 1 은 기존 C 프로그램을, 그림 2 는 수정된 C 프로그램을 각각 보이고 있다.

```
#define N 10000
typedef int array_t [N];
extern void foo(), void bar();
int main(int argc, char **argv) {
  int i;
  array_t v;

  /* this could be parallelized */
  for (i = 0; i < N; i++) { v[i] = 0; }
  printf("initialized array\n");
  /* these could run in parallel */
  foo();
  bar();
}
```

그림 1. OpenMP 로 수정되기 전의 C 프로그램

```
#define N 10000
typedef int array_t [N];
extern void foo(), void bar();
int main(int argc, char **argv) {
  int i;
  array_t v;

  #pragma omp parallel {
    #pragma omp for
    for (i = 0; i < N; i++) { v[i] = 0; }
  }
  printf("initialized array\n");
  #pragma omp parallel {
    #pragma omp sections {
      #pragma omp section { foo(); }
      #pragma omp section { bar(); }
    }
  }
}
```

그림 2. OpenMP 규정으로 수정된 병렬 프로그램

현재 상용화된 OpenMP 컴파일러로는 Compaq 사의 Digital Fortran, SGI 사의 MIPSpro 7.3 컴파일러 등이 있으며, 프리웨어로는 OdinMP[4], Omni 컴파일러 등이 있다. 이들 제품에 대한 정보는 http://www.openmp.org 의 resources 섹션에 모두 포함되어 있다.

4. 결론

우리는 본 논문에서 공유 메모리 프로그래밍 API 표준안인 OpenMP 에 대해 살펴보고, OpenMP 의 여러 규정들을 분석하였다. 공유 메모리 프로그래밍 방식이 메시지 전달 방식에 비해 편하고 쉽지만, 표준 API 의 부재로 인하여 호환성 문제가 발생하기 마련이었다. 이러한 상황에서 본 논문에서 살펴본 OpenMP 는 좋은 대안이 될 수 있을 것이다. 하지만 아직 OpenMP 를 지원하는 컴파일러 시스템들이 그리 많이 출시되지 않았고, 이미 출시된 OpenMP 시스템들도 고가의 시스템들이어서 관련 연구가 시급한 실정이다. 본 논문이 기여하는 바는 OpenMP 에 대한 소개를 통해 향후 관련 연구를 활성화시킬 수 있을 것이라는 점이다.

참고 문헌

[1] B. Leasure, ed.. Parallel Processing Model for High Level Programming languages. Draft Proposed American National Standard for Information Processing Systems. April 5, 1994.

[2] The OpenMP ARB. OpenMP: A Proposed Industry Standard API for Shared Memory Programming. White Paper, October 1997. Also available on the resources section at http://www.openmp.org.

[3] The OpenMP ARB. OpenMP C and C++ Application Program Interface, Version 1.0 – October 1998. OpenMP Consortium Document Number 004-2229-001.

[4] Christian Brunschen. OdinMP/CCp – A Portable Compiler for C with OpenMP to C with POSIX threads. MS. Thesis, Dept. of Information Technology, Lund University, July 1999.