

인텔 제온 파이를 활용한 푸아송 방정식 풀이의 병렬화

조규남*, 서재민*, 김도형*, 류훈**, 정창성*

*고려대학교 전기전자공학과

**한국과학기술정보연구원

e-mail : *{mystous, onlytjwo, 2015010681, csjeong}@korea.ac.kr, **elec1020@kisti.re.kr

Parallelization of Poisson equation solver on Intel Xeon Phi environment

Kyu Nam Cho*, Jae Min Seo*, Do-Hyeong Kim*, Hoon Ryu**, Chang-sung Jeong*

*Dept. of Electrical Engineering, Korea University

**Korea Institute of Science and Technology Information

요약

코프로세서(Co-processor)를 사용한 병렬 처리 시스템은 멀티코어 프로그래밍과 함께 과학기술 계산 분야 프로그램 개발에 많이 사용되고 있다. 본 연구에서는 CPU 기반의 코프로세서인 인텔 제온 파이 환경에서의 푸아송 방정식 해법을 병렬화 하였다. 본 연구를 통해서 인텔 제온 파이 활용 가능성을 확인하고, 일반적인 병렬화 기법이 인텔 제온 파이 환경에서도 적합한지를 확인하였다.

1. 서론

코프로세서(Co-processor)를 이용한 병렬처리 시스템은 GPGPU 의 출현으로 각광 받게 되었고, 최근까지도 연구가 활발히 진행되고 있는 분야이다. 코프로세서를 활용하면 연산 노드의 추가나 확장 없이 계산 능력을 증가 시킬 수 있다는 장점이 있다. 이는 전력 적인 측면이나 공간적인 측면에서의 이득을 취할 수 있다. 기존 GPGPU 를 활용할 경우 계산상의 이득을 취할 수 있지만 기존 코드를 GPGPU 에 동작할 수 있도록 수정하는 작업이 필요하고 이를 위한 별도의 지식이 요구 된다. 인텔 제온 파이[1]의 경우 기존 병렬 시스템을 위한 구현 방법을 큰 변경 없이 적용할 수 있는 장점이 있다. 본 연구에서는 오프로딩(Offloading) 방식을 적용하여 순차코드 대비 308%, OpenMP 코드 대비 126%의 성능 향상을 확인하였다. 본 연구로 인텔 제온 파이 병렬화를 활용한 푸아송 방정식 풀이가 높은 성능을 가질 수 있음을 증명하였다.

2. 관련연구

2.1 푸아송(Poisson) 방정식

푸아송 방정식은 다양한 공학분야에서 사용되는 2 차 편미분 방정식이다. 전자공학 및 물리학에서는 전하량의 밀도가 주어졌을 때, 이를 바탕으로 전위 분포를 구하는데 사용될 수 있다. 분포함수 u , RHS(Right hand side)의 조건 분포함수를 f 라고 하면 푸아송 방정식은 식 2.1로 정의 된다.

$$\nabla^2 u = f(x, y, z) \quad (2.1)$$

1 차원 푸아송 방정식은 수식을 통해서 해를 구할 수 있다. 2 차원 이상의 푸아송 방정식은 유한차분법(Finite difference method)을 통해서 주어진 수식을 선형화 시킬 수 있다. 유한차분법은 미분 방정식의 미분 항을 근사식으로 치환하는 방법으로 유한차분법을 통해 미분방정식은 $Ax=b$ 형태의 선형연립 방정식을 구할 수 있다[2].

이렇게 주어진 선형방정식은 가우스 소거법 같은 직접법이나 SOR(Succesive over-relaxation)과 같은 반복법(Iterative method)을 통해서 풀 수 있다. 반복법은 대용량의 데이터에 동일한 연산이 반복되는 특성을 이용하여 병렬 시스템에 적용할 수 있다. 본 연구에서는 반복법에 해당되는 컬레기울기법(CG-Conjugate Gradient)을 사용하여 푸아송 방정식의 해를 구한다.

2.2 컬레기울기법(CG-Conjugate Gradient)

컬레기울기법[3]은 대칭(Symmetric)성을 가지는 양의 정부호 행렬(Positive-definite matrix)의 해를 찾는 수치 기법이다. 병렬화를 통해서 빠른 풀이가 가능하다. 푸아송 방정식을 유한차분법을 통하여 선형화 시키면 컬레기울기법으로 풀이가 가능한 형태의 행렬이 도출된다. 컬레기울기법의 알고리즘은 아래와 같다.

알고리즘 1. 컬레기울기법 알고리즘

$$r_0 := b - Ax_0$$

$$p_0 := r_0$$

For k := 0 to max_iteration

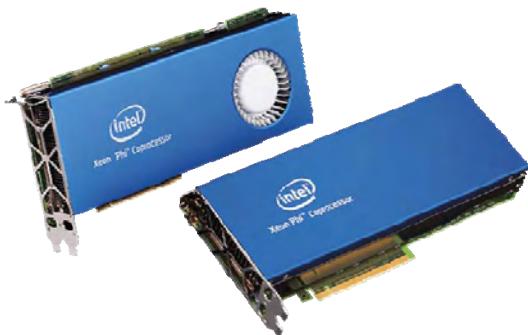
```

 $a_k := r_k^T r_k / p_k^T A p_k$ 
 $x_{k+1} := r_k - a_k A p_k$ 
if  $r_{k+1} < tolerance$ 
    exit For
 $b_k := r_{k+1}^T r_{k+1} / r_k^T r_k$ 
 $p_{k+1} := r_{k+1} + b_k p_k$ 
 $k := k + 1$ 
End

```

2.3 인텔 제온 파이

제품별로 조금의 차이는 존재 하지만 인텔 제온 파이는 코어(Core)당 4 개의 하드웨어 쓰레드(Thread)를 탑재한 최대 61 개 x86 코어를 활용하여 만들어진 코프로세서이다. 연산 속도는 최대 1.2 배정밀도(Double precision) TeraFLOPS 이다[1]. GPGPU 와는 달리 별도의 프로그래밍 모델(CUDA)을 통해서가 아닌 일반적인 병렬 프로그래밍 모델인 MPI(Message Passing Interface), OpenMP, TCP/IP 을 사용하여 프로그램을 작성할 수 있다.



(그림 1) 인텔 제온 파이 카드(출처:Intel Korea)

인텔 제온 파이를 활용한 프로그래밍에는 디바이스에 실행 파일이 직접 실행되는 기본(Native) 모드, 병렬화가 필요한 연산과 데이터만을 전송하여 실행하는 오프로딩(Offloading) 모드 그리고 호스트와 코프로세서가 작업을 공유하는 대칭모드 세 가지 방식이 사용될 수 있다.

인텔제온파이가 가지는 병렬컴퓨팅의 장점[4]은 몇 가지로 요약될 수 있는데, 그 중 가장 특징적인 것은 아래와 같다.

- 512KB SIMD width
- FMA(Fused Multiply-Add)
- 향상된 VPU(Vector Processing Unit)
- Gather, Scatter instruction

3. 병렬화 구현

본 연구에서는 컬레기울기법을 사용하여 작성되어 있는 푸아송 방정식 풀이 순차 코드를 오프로딩(offloading) 모드를 통하여 구현하였다. 병렬화는 다음 3 단계를 거쳐서 진행되었다.

- 1) OpenMP 적용
- 2) 메모리 얼라인먼트 조정

3) 오프로딩(Offloading) 문법(directive) 적용

먼저 OpenMP 를 사용하여 반복문을 병렬화 하였다. OpenMP 는 인텔 제온 파이뿐 아니라 CPU 병렬화에서 사용되는 기법이다. 인텔 제온 파이에서는 CPU 병렬화에서 사용되는 OpenMP 문법을 사용하여 간단하게 병렬화를 진행할 수 있다.

두 번째 단계는 메모리 얼라인먼트 조정이다. 이는 변수의 메모리를 할당할 때 CPU 의 캐시라인에 적합한 크기로 메모리의 영역을 할당해준다. 이렇게 함으로써 벡터라이제이션이 효율적으로 적용되어 성능향상 효과를 볼 수 있다. 특히 인텔 제온 파이의 경우 VPU 의 성능이 제온 CPU 등에 좋은 성능을 보여주기 때문에 매우 전체적인 성능 향상에 도움을 줄 수 있다.

마지막으로는 오프로딩 문법 적용이다. 오프로딩을 위해서는 오프로딩으로 동작할 계산 연산 지정과 해당 연산에서 사용될 데이터를 호스트에서 디바이스(인텔 제온 파이)로 전송하는 문법이 있다. 아래 코드는 오프로딩 문법은 간략히 표현한 코드이다.

```

double *b = (double*)_mm_malloc(n*
sizeof(double), 64);

#pragma offload target(mic)\n
    in(b(n) alloc_if(1) free_if(0))\n
#pragma omp parallel for\n
    for(int i ; i < n ; ++i )\n
    ...\n\n
#pragma offload_transfer target(mic)\n
    nocopy(*b: alloc_if(0) free_if(1))

```

오프로딩 문법은 인텔 제온 파이에서 실행될 연산, 전송될 데이터를 정의하는 부분과(offload target) 전송된 데이터의 메모리를 해제하는 부분으로 이루어져 있다.

이런 절차를 통해 순차코드를 오프로딩으로 인텔 제온 파이에서 실행할 수 있다.

4. 성능평가

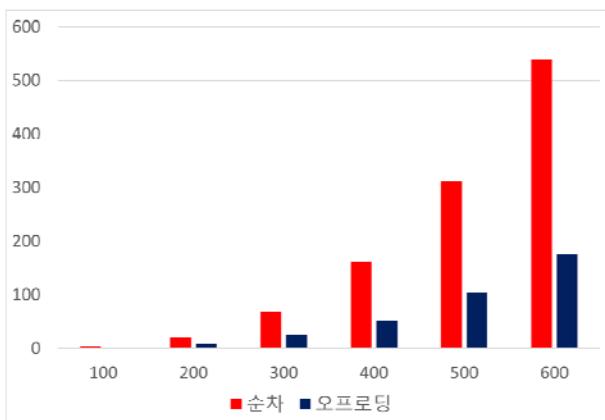
본 연구에서는 병렬화된 푸아송 방정식 해법을 검증하기 위하여 아래와 같은 환경에서 실험을 진행하였다. 표 1은 순차 코드 및 병렬화된 코드가 동작한 시스템의 환경이다.

<표 1> 성능평가 환경

호스트	디바이스
Intel Xeon E5-2680 20 CPU 2.8GHz(1CPU) 256GB	Intel Xeon Phi 7120P 61 Core 1.2GHz(1Core) 16GB

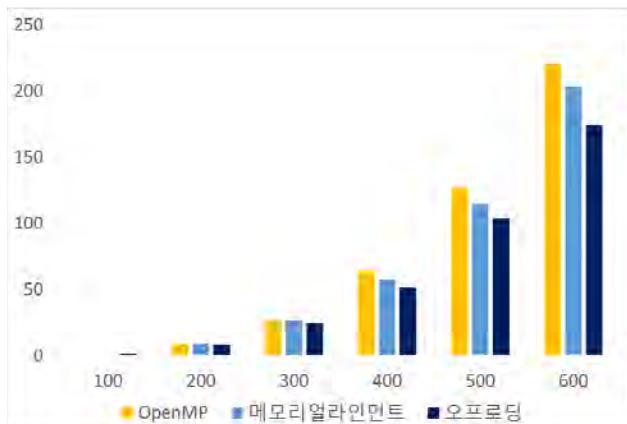
5. 결론 및 향후 연구

실험은 순차코드를 각각 100, 200, 300, 400, 500, 600개의 메쉬(Mesh)를 인자값으로 입력하여 풀이 시간을 측정하였다. 푸아송 방정식은 3 차원 푸아송 방정식을 사용하였다. 순차코드의 대조군으로는 병렬화 구현의 각 단계를 차례로 적용한 OpenMP 적용 코드, 메모리 얼라인먼트 조정 코드 그리고 마지막으로 오프로딩이 적용된 코드로 각기 순차코드와 동일한 데이터로 시간을 측정하였다. 모든 코드는 컴파일러 최적화 옵션-O3를 적용하였다. OpenMP 와 메모리 얼라인먼트 수정 코드는 모두 OMP_NUM_THREADS=20 이 적용되었다. 메모리 얼라인먼트는 _mm_malloc 함수를 사용하여 64byte 로 메모리 얼라인먼트를 조정하였다. 오프로딩은 쓰레드 240 개 MIC_KMP_AFFINITY = scatter로 실험하였다. 성능 측정 결과 오프로딩으로 구현된 푸아송 풀이의 실행 시간은 순차 코드 대비 308%의 성능 개선을 보였고, OpenMP 대비 126%, 메모리 얼라인먼트 조정 대비 116%의 성능 향상을 보였다. 위 성능 평가의 결과 중 순차코드와 오프로딩 적용 코드의 성능 비교표는 그림 2 와 같다. 가로축은 메쉬의 개수이며, 세로축은 실행시간이다. 세로축은 수치가 적은 측이 더 높은 성능을 보인 것이다.



(그림 2) 순차코드와의 성능 비교

그림 3 은 OpenMP 적용 코드, 메모리 얼라인먼트 조정 코드와 오프로딩 적용 코드들의 성능 비교이다.



(그림 3) 병렬화 코드간 성능 비교

본 연구에서는 푸아송 방정식의 풀이 순차 코드를 오프로딩 방식을 사용하여 인텔 제온 파이에 적용하였다. 그 결과 순차 코드 대비 308%의 성능 향상을 보였고, 다른 병렬화 기법인 OpenMP 및 메모리얼라인먼트 적용 코드에 대비해서도 각각 126%, 116%의 성능 향상을 보였다. 이를 통해서 우리는 인텔 제온 파이를 통한 병렬화 가능성을 확인 할 수 있었다. 오프로딩의 경우 연산 과정이 호스트 컴퓨팅 자원을 활용하지 않고 인텔 제온 파이의 컴퓨팅 자원만을 사용한다. 또한 데이터 전송을 위하여 많은 시간을 사용하게 된다. 우리는 향후 호스트와 인텔 제온 파이의 컴퓨팅 자원을 모두 사용하는 방법인 대칭 모드를 사용하여 성능을 극대화 할 수 있는 방법을 연구할 예정이다. 또한 인텔 제온 파이의 차별화된 장점인 확장된 512KB SIMD 의 사용과 VPU 의 성능을 최대로 사용할 수 있는 벡터라이제이션 기법 및 FMA 를 적용하여 병렬컴퓨팅에서의 인텔 제온 파이의 활용성을 확인하도록 하겠다.

Acknowledgement

본 연구는 2015 년도 BK21 플러스 사업과, 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터 육성 지원사업의 연구결과로 수행되었음(HITP-2015-H8501-15-1004)

참고문헌

- [1] “xeon phi.” <http://www.intel.co.kr/content/www/kr/ko/processors/xeon/xeon-phi-detail.html>.
- [2] 송창근, “노이만 경계 조건을 갖는 푸아송 방정식의 개선된 직접 병렬 해법,” (구)정보과학회논문지, vol. 21, pp. 706–714, 1994.
- [3] “Conjugate gradient method.” https://en.wikipedia.org/wiki/Conjugate_gradient_method.
- [4] R. Rahman, *Intel® Xeon Phi™ Coprocessor Architecture and Tools*. Berkeley, CA: Apress, 2013.