인텔 제온 파이에서의 희소행렬 벡터 연산

이 혜지○ 임은진
국민대학교 컴퓨터공학부
hyeoiji.lee@kookmin.ac.kr, ejim@kookmin.ac.kr

SpMV on Xeon Phi

Hyeoi-Ji Lee○ Eun-Jin Im
School of Computer Science, Kookmin University of Korea

요 약

고속 연산을 필요로 하는 대량 연산 분야에서 매니코어 프로세서를 이용한 구현과 그 성능 분석 및 개선은 고속 연산의 구현 뿐 아니라, 매니코어 프로세서의 성능 이해와 미래 설계를 위해서 필수적인 연구이다. 본 연구에서는 인텔의 매니코어 연산 가속기인 제온 파이 프로세서를 사용한 희소행렬 연산을 구현하고 실제 여러 과학 분야에서 사용되는 희소행렬에 대하여 적용한 후 성능을 비교하여 희소행렬 연산의 최적화 방안을 설계한다.

1. 서 론

무어의 법칙에 따라 프로세서의 집적도는 기하급수적으로 증가하지만 Power Wall 에 부딪힌 프로세서 클락 속도는 더 이상 증가하지 않는다. 그 결과 프로세서 내 코어 개수가 기하급수적으로 늘어나고 있는 상황에서 매니코어 프로세서를 위한 활발한 소프트웨어 연구가 요구되고 있다.[1] 그 중 고속 연산을 필요로 하는 대량 연산 응용 분야에서 성능 문제에 초점을 맞출 때, 실제 연산 성능이 이론적 최고치(모든 연산 자원이 최대한 활용되는 가정)에 못 미치는 이유는 4가지로 볼 수 있다. (1) 연산 자체의 병렬성이 부족하거나 (2) 병렬성이 있더라도 충분히 표현되지 못하거나, (3) 메모리나 입출력 장치에서의 병목 현상, (4)프로세서 구조에 특화된 명령어를 활용하지 못하는 등의 문제 때문이다.

현재 연산 가속화를 위해 사용되는 매니코어 프로세서로는 NVIDIA/AMD 의 GPU 들과 Intel 의 Xeon Phi 와 같은 가속기가 대표적이다. 본 논문에서는 과학/공학 연산 분야에서 사용되는 대규모 미분 방정식의 해를 구하는데 필수적인 연산인 희소행렬 연산을 대상으로 Xeon Phi에서 구현하고 성능을 측정한다. 희소 행렬 연산은 데이터 구조가 간접 포인터로 구성되어 산술연산 대 메모리 접근 연산의 비율 (arithmetic intensity) 가 낮고, 연산의 메모리 접근이 순차적이지 못하여 메모리 계층구조를 효율적으로 활용하지 못한다는 구조적인 원인으로 연산 속도가 이론적 최고치에 훨씬 밑도는 연산이다.

* 본 논문은 2014년도 정부(미래창조과학부)의 지원으로 매니코어 기반 초고성능 스케일러블 OS 기초연구(차세대 OS 기초 연구센터) 과제로 수행된 연구임
(No. 14-824-09-011)

또한 그 연산 성능은 행렬 내 0이 아닌 원소의 분포 형태 (spyplot 으로 나타내짐) 에 따라 편차가 크다. 본 논문에서는 희소행렬-벡터곱셈 연산 (SpMV)을 Xeon Phi 프로세서 상에서 구현하고, 다양한 형태의 분포를 가지는 행렬들에 대해 그 성능을 측정하여 비교함으로써 향후 연산의 최적화 방안을 제시한다.

2. 인텔 제온 파이 구조와 OpenMP와의 기능

인텔에서 출시한 KCN(Knights Corner) 기반의 제온 파이는 MIC(Many Integrated Core)구조로서 매니코어 코프로세서이다. 최소 57개 이상의 코어로 이루어져 있으며 각 코어는 양방향성 링 버스로 연결되어 있다. 각 코어에는 SMT(Simultaneous Multi-Threading)를 비롯한 4개의 중요한 컴포넌트들로 이루어져 있다. SMT는 동시에 4개의 쓰레드를 수행 할 수 있고, VPU(Vector Processing Unit)는 512bit SIMD(Single Instruction Multiple Data) 엔진을 통해 한 번의 연산으로 16개의 일 배수 배정밀도 연산이나 8개의 이 배수 배정밀도의 소수점 연산의 수행을 지원한다. L1 캐쉬는 데이터 캐쉬와 명령어 캐쉬 2개로 구성되어 있고, L2 캐쉬는 512KB의 용량을 갖고 un-core 인터페이스로 사용된다.

그리고 각 코어마다 VPU(Vector Processing Unit)를 지원하여 512bit 길이의 FMA(Fused Multiply-Add)로 3피연산자 1연산이 가능하다. 이는 계산응용을 3피연산자 1연산 형태로 적용하여 작성한다면 성능을 향상 시킬 수 있는 기능이다. 또한 각 코어마다 4개의 하드웨어 쓰레드를 제공하는데 실행 모드에 따라 최대 사용 가능한 쓰레드 개수가 제한된다. 인텔 제온 파이에서는 총 2가지 모드로서 네이티브와 오프로드 모드로 계산응용을 실행할 수 있다. 네이티브일 경우 “코어개수*4개”의 쓰레드를, 오프로드일 경우 “(코어개수-1)*4”개의 쓰레드를 최대로 사용가능하다. 또한 각 코어마다 쓰레드를 어떻게

분배할지 compact, scatter, balanced 값으로 설정이 가능하다.(그림1)

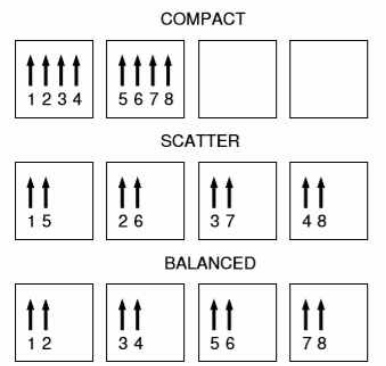

그림1. 4개의 코어 시스템 상에서 쓰레드 분배 알고리즘

3. 기존 연구

생명공학, 공학, 자연과학을 포함하는 계산과학 분야에서 많이 사용되고 있는 SpMV(Sparse Matrix-Vector multiplication)의 성능은 주어진 아키텍처와 행렬 종류에 따라 많은 영향을 받는다. 그래서 각 아키텍처에 맞게 성능을 최적화하는 연구가 진행되고 있어 지금까지는 멀티코어 기반 또는 GPU상에서의 성능 최적화 연구가 많이 진행되어 왔다.[2][3] 그에 대한 연구는 표1과 같이 크게 코드, 자료구조, 병렬 최적화로 총 3가지로 분류할 수 있다. 또한 최적화 연구와 더불어 각 아키텍처의 성능 모델링 연구가 함께 이뤄져 최적화 분석 툴 개발이 진행되고 있으며[4], 더 나아가 오토 튜닝의 연구에 포함되고 있다.[5]

표1. 희소행렬 벡터 연산 최적화에 대한 연구

코드 최적화	자료구조 최적화	병렬 최적화
SIMDization	BCSR	Row Threading
SW Pipelining	BCOO	Process Affinity
Branchless	16-bit indices	Memory Affinity
Pointer Arithmetic	32-bit indices	
PF/DMA Values&Indices	Register Blocking	
PF/DMA Ponters/Vetors	Cache Blocking	
inter-SpMV data structure caching	TLB Blocking	

4. 실험 데이터 및 환경

4.1 실험 데이터

본 실험에서는 GPU와 멀티코어 상에서 진행된 타 연구의 실험 데이터를 그대로 사용했다.[2][3] 연산 속도에 민감한 데이터로서 공학/과학 분야 시뮬레이션에서 사용되는 대규모 미분방정식의 해를 구하는데 있어서 필수적이면서도 반복적으로 사용되는 희소행렬 벡터이다. 각 데이터 정보와 사용분야는 표2에 적혀있다.

4.2 실험 환경

본 실험 환경은 Intel Xeon Phi 3210A모델에서 진행되었고 제조사에서 성능을 다음과 같이 제공한다(표3).

표3. 인텔 제온 파이 3120A 주요 성능

코어 수	57
클럭 속도	1.1 GHz
L2 캐쉬	28.5MB
최대 메모리 크기	6GB
최대 메모리 대역폭	240 GB/s
최고 배정밀도(Rpeak)	1003GFLOPs

호스트 환경은 CentOS 6.4버전에서 MPSS 3.3버전을 사용하였고 인텔 제온 파이에서는 uOS 2.6.38.8 kernel을 사용하고 MPSS 3.3버전을 사용했다. 기본적으로 OpenMP를 사용하여 offload모드에서 실험을 진행했으며, 56개 코어와 각 코어마다 4개의 쓰레드를 사용할 때 설정은 다음과 같다.[6] 그러나 쓰레드를 설정 할 때, 주의할 부분이 있다. PHI_KMP_PLACE_THREADS와 PHI_OMP_NUM_THREADS는 같이 사용하는 것을 권장하지 않는다. 실제 같이 사용하면 성능이 저하되어, 이번 실험에서는 PHI_KMP_PLACE_THREADS으로만 설정한다. 또한 컴파일는 icc를 사용하고 옵션으로는 vec-report2, openmp-report=2, openmp, offload 옵션으로 설정했다. 또한 성능은 호스트와 MIC간의 데이터 전송 시간을 제외한 Intel Xeon Phi에서의 연산 시간만

```
export MIC_ENV_PREFIX=PHI
export PHI_KMP_AFFINITY=balanced
export PHI_KMP_PLACE_THREADS=56c,t4,0O
```

을 측정하였다. (OFFLOAD_REPORT참조,[7])

표2. 실험 데이터 희소행렬

행렬	이름	#row	#col	#nonzero	응용분야
	FEM Cantilever	62,451	62,451	4,007,383	건축공학
	FEM Spheres	83,334	83,334	6,010,480	지리학
	Economics	206,500	206,500	1,273,389	경제학
	Epidemiology	525,825	525,825	2,100,225	확률론
	Protein	36,417	36,417	4,344,765	생명공학
	QCD	49,152	49,152	1,916,928	물리학
	LP	4,284	1,092,610	11,279,748	철도
	FEM/Harbor	46,835	46,835	2,374,001	CFD
	FEM/Ship	140,874	140,874	7,813,404	선박
	webbase	1,000,005	1,000,005	3,105,536	웹

5. SpMV 성능

Intel Xeon Phi에서 수행되는 연산 시간은 데이터 전송 시간을 고려한 여부에 따라 두 가지로 나눠서 측정했

다. 첫 번째는 OFFLOAD_REPORT 결과에 따라 실제 MIC에서 수행된 시간을 측정하여 성능을 측정하였고, 두 번째로는 MIC으로 연산을 수행할 행렬 데이터를 전송하는 부분까지 고려하여 연산 수행 시간을 측정하였다.

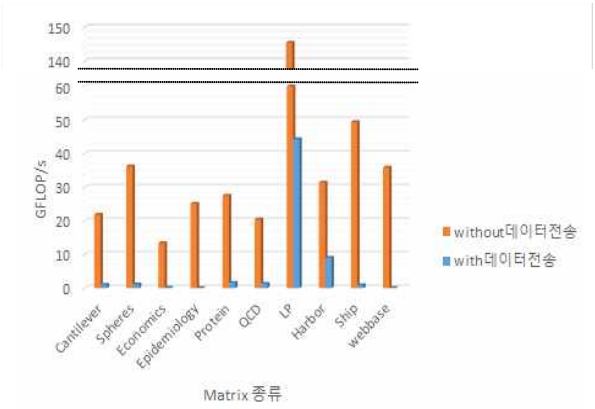

그림2. matrix 종류에 따른 SpMV성능(CSR 포맷)

6. 분석

최고 배정밀도 값(Rpeak) 대비 연산 수행(without 데이터 전송)을 고려했을 때, 최저 약 1.34%에서 최고 약 14.5%까지의 성능을 갖게 된다. 반면에 연산 행렬 전송 시간을 고려한다면 최저 약 0.01%에서 최고 약 4.42%까지의 성능을 보인다. 이를 통해 보았을 때, offload모드로 SpMV를 수행할 때, 데이터 전송에 대한 오버헤드를 줄이기 위하여 행렬의 특성에 따라 행렬 포맷을 선정하는 부분에 대한 연구가 진행되어야 필요성이 있다. 또한 행렬의 병렬성을 충분히 표현하기 위하여서 기존에 연구가 진행되던 형태에서 VPU를 어떻게 적용할지에 대한 부분에 대해 정의가 필요하다.

7. 결론 및 향후 계획

본 실험을 기반으로 Intel Xeon Phi에서의 연산 성능을 개선할 수 있는 새로운 희소행렬 포맷을 연구하고, SpMV 연산에 VPU의 활용도를 높이기 위한 최적화 방안과 더불어 쓰레드 스케줄링 기법을 개선하여 Intel Xeon Phi에 맞는 희소행렬 연산의 최적화 방안을 설계할 것이다.

8. 참고문헌

[1]asanovic, k., bodik, r., catanzaro, b., gebis, j., keutzer, k., Patterson, D., Plishker, W., shalf, j., Williams, s., and yelick, k. The Landscape of Parallel Computing Research: A View from Berkeley. technical report ucb/eecs-2006-183. eecs, university of california, berkeley, Dec. 2006. 5

[2]Samuel Williams, “Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", SC07

[3]Nathan Bell, Michael Garland, “Efficient Sparse Matrix-Vector Multiplication on CUDA”, “NVIDIA Technical Report NVR-2008-004”, Dec. 2008

[4]Ping Guo, "A Performance Modeling and Optimization Analysis Tool for Sparse Matrix-Vector Multiplication on GPUs", IEEE 2012

[5]Samuel Webb Williams, “Auto-tuning Performance on Multicore Computers”, EECS 2008

[6]https://software.intel.com/en-us/blogs/2013/02/15/new-kmp-place-threads-openmp-affinity-variable-in-update-2-compiler

[7]https://software.intel.com/en-us/node/510196