특집 02

매니코어 시스템에서의 병렬 프로그래밍 최적화를 위한 분석 도구 및 벤치마크 성능 실험

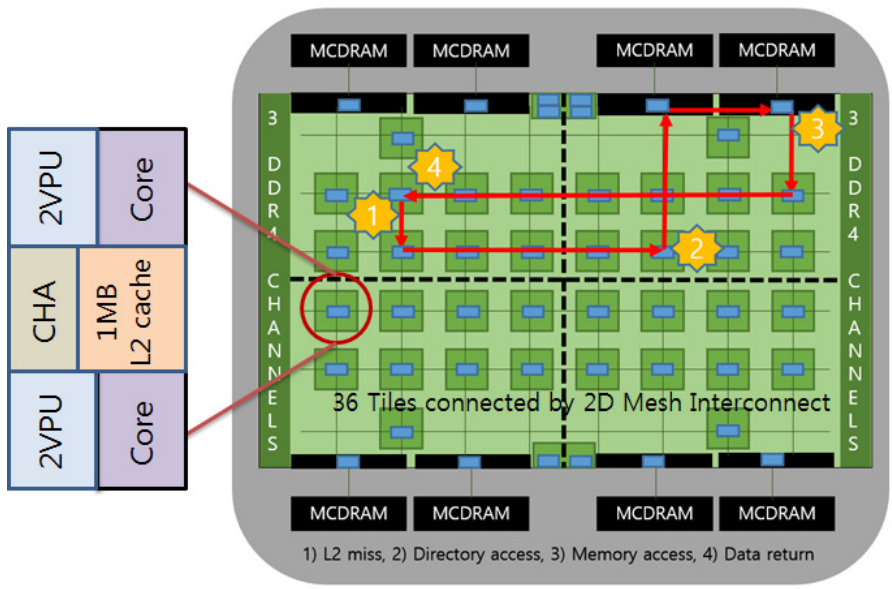
노승우·최지은·남덕윤·박근철·박찬열 (한국과학기술정보연구원)

목 차

1. 서 론

슈퍼컴퓨터는 하드웨어적으로 크게 컴퓨팅, 스토리지, 인터커넥트 부분으로 구분하여, 해당 성능을 높이는 데에 집중한다. 슈퍼컴퓨터를 구축하는 데에 있어서 클러스터 형태가 주류가 된 이후, 스토리지 및 인터커넥트의 성능 향상에 비해 컴퓨팅 부분의 성능 향상은 발전이 더딘 상황이었다. 이는 컴퓨팅 성능이 프로세서 성능 발전과 함께 이루어져 왔기 때문이다. 프로세서의 성능, 즉 반도체 집적회로 성능의 발전은 무어의 법칙으로 이야기하듯이 18개월에서 24개월마다 두 배씩 증가해 왔으나, 몇 년 전부터 무어의 법칙이 더 이상 지켜지기 힘들 것이라는 소식이 들려왔다. 이러한 이유는 고집적을 위한 비용의 증가, 고밀도에 따른 발열, 선로 선폭 및 회로 간격의 감소로 인한 데이터 간섭 현상 발생 가능성 등 다양한 현상들 때문이었다. 또한 멀티코어 성능 향상의 한계는 몇 해 전부터 단일 CPU의 클럭 속도가 3GHz에서 더 이상 향상되지 않고 있는 점에서 어느 정도 예상되던 부분이었다. 이러한 상황에서 컴퓨팅 성능은 프로세서의 클럭 속도 향상이 아닌, 인텔 제온파이나 NVIDIA GPU와 같이 별도 카드 타입의 가속기 형태가 나옴으로서 향상을 꾀할 수 있었다.

최근의 프로세서 성능 향상 방법은 멀티코어에서도 그렇게 해 왔지만 매니코어의 방식으로 (비록 클럭 속도가 낮은 코어이기는 하지만) CPU 내 코어 수를 늘리고, 프로세서 패키지 내에 추가적인 메모리를 부착하는 형태로 이루어지고 있다. 인텔에서는 2016년에 매니코어 프로세서인 나이츠랜딩 (Knights Landing; KNL)을 가속기 타입이 아닌 호스트 프로세서 타입으로 출시했다. 여기에서 우리는 앞으로 고속 메모리를 프로세서 패키지 안에 포함한 형태의 프로세서를 자주 보게 될 것으로 예상된다. 이는 '레지



(그림 1) KNL 프로세서의 하드웨어 구조 및 Quadrant 모드 캐시 미스 라우팅[2]

스터-캐시-메모리-디스크'의 저장 매체 계층이 캐시와 메모리 사이에 위치하는 온-패키지 메모리 또는 고대역폭 메모리라 불리는 고속 메모리가 위치하여 '레지스터-캐시-고대역폭 메모리-기존 메모리'의 계층 구조가 된다는 의미이다. 결국 슈퍼컴퓨터와 같이 컴퓨팅 요소의 모든 부분을 최적화하고 고속화하고자 하는 분야에서는 고속 메모리가 추가로 들어간 프로세서를 제대로 분석하고, 활용할 수 있어야 한다.

이에 본고에서는 고속 메모리가 추가된 인텔 KNL을 간략히 소개한 후, 병렬 프로그래밍을 위한 최적화된 환경을 분석하기 위해 활용하는 성능 모니터링 방법과 주요 분석 도구에 대해서 설명한다. 그리고 KNL 기반의 시스템 테스트베드를 활용하여 주요 벤치마크 실험을 실행하고 성능을 측정한다. 이를 통해 KNL 고속 메모리 크기와 코어 수, 스레드와 MPI 프로세스 수의 조합 등이 병렬 프로그램 성능에 끼치는 영향을 살펴보고 분석한다.

2. 매니코어 시스템

대표적인 매니코어 시스템인 인텔 제온파이 시스템은 매우 많은 수의 코어와 하드웨어 스레드 구조 때문에 기존의 프로세서와는 다른 특징을 가진다. 즉, 지금까지는 적은 수의 크고 정교한 코어를 만드는 방식이었다면, 반대로 제온파이는 더 많은 수의 단순한 코어를 만드는 접근 방식을 채용했다. 이러한 아이디어는 트랜지스터의 대부분이 산술 연산에 사용될 수 있다는 점에 착안했다. 이 접근법으로 제 1세대인 나이츠 코너(Knights Corner, KNC)를 시작으로, 2016년 제2세대인 나이츠 랜딩(Knights Landing, KNL)이 공식 출시되었다. KNL CPU는 KNC와 달리 AVX2 명령어 셋 확장을 통해 완전히 인텔 제온 ISA(Instruction Set Architecture)와 호환이 가능해졌기 때문에, 독립적으로 운영체제를 실행할 수 있다[1,2].

2.1 시스템 구조

KNL 플랫폼은 코프로세서, 호스트프로세서, 전용 통합 네트워크(옴니패스)를 가진 호스트프로세서의 총 3가지 유형을 가진다. 프로세서 구조는 1개의 타일 당 1MB의 L2 캐시를 공유하는 2개의 코어와 캐시 동기화를 위한 Cache/Home Agent(CHA)로 구성되며, 각 타일은 2D 메시 구조로 최대 36개 타일을 이룬다. 한 코어는 1.3~1.4GHz의 클럭 속도로, 최대 동시 4개의 멀티쓰레딩(4 SMT)을 지원하며 2개의 AVX-512 벡터처리 장치(Vector Processing Unit)를 가지고 있다. 메모리 구조는 8개의 고속 메모리 컨트롤러에 의해 접근되는 16GB 고대역폭 메모리(High Bandwidth Memory, HBM)와 2개의 3채널 메모리 컨트롤러에 의해 접근되는 최대 384GB의 DDR4 메모리로 구성된다. I/O 구조는 36개의 PCIe 선로와 추가적인 옴니패스(omnipath) 컨트롤러로 구성된다. 옴니패스는 인텔 전용 고속 네트워크 구조로 최대 100 Gb/s의 속도를 보장한다.

2.2 메모리 구조

KNL의 고대역폭 메모리(HBM)인 MCDRAM (Multi-Channel Dram)은 flat, cache, hybrid 형태의 3가지의 다른 모드로 사용할 수 있다. flat 모드는 MCDRAM이 별도의 물리적 공간을 가지도록 하며, DDR4와는 별도로 원하는 데이터를 저장할 수 있다. 이 모드는 가장 높은 대역폭과 가장 작은 지연시간을 제공하는 반면, 효과적으로 사용하기 위해 응용 프로그램의 수정이 필요할 수 있다. cache 모드는 MCDRAM을 L3 캐시로 사용할 수 있어, 응용프로그램의 수정이 필요 없다. 단점은 캐시 미스가 발생했을 때 데이터가 크면 클수록 지연이 커진다. hybrid 모드는 cache 모드와 flat 모드를 혼합하여 사용한다. 적합한 설정방법은 전체 응용프로그램의 크기와 데이터의 수정, 분리 유무에 따라 적절하게 설정한다.

2.3 캐시 클러스터링 모드

KNL의 각 타일 내의 모든 L2 캐시는 MESIF 프로토콜을 사용하는 메시에 의해 일관성있게 유지된다. 메시의 수직 및 수평 링크는 양방향 링크이며, 각 타일은 캐시 동기화를 위해 캐시 라인의 칩 상태와 위치를 식별하는 분산된 태그 디렉토리(Tag Directory, TD)를 가지고 있다. 이러한 캐시 동기화의 복잡성을 관리하고 주어진 계산 응용 프로그램에 대해 최적의 작동 모드를 설정하기 위해 KNL은 물리 메모리 주소 매핑 방법에 따라서 All2All, Quadrant/Hemisphere, SNC(Sub-NUMA-Clustering)의 3가지 클러스터링 모드를 제공하며, 이중 하나에서 작동한다. 이러한 모드는 바이오스나 인텔에서 제공해주는 명령어를 통해 콘솔에서 변경 가능하지만, 적용을 위해서는 시스템 재시작이 필요하다.

All2All 모드는 타일, TD, 메모리 채널 사이에 의존성이 없어, 메모리 주소가 칩의 모든 TD에 균일하게 분산되기 때문에 가장 지연이 크다. Quadrant/hemisphere 모드는 가상의 4개 또는 2개 부분으로 나뉘어지고, 메모리 주소가 같은 부분의 TD로 해시된다. 즉, 같은 위치에 TD와 메모리 채널이 위치하기 때문에 All2All 모드에 비해 더 높은 성능을 가진다. SNC 모드는 2개(SNC2) 또는 4개(SNC4)의 사분면을 모두 가상 NUMA 클러스터로 나타내어, 2 또는 4 소켓 제온 서버처럼 보이게 한다. 이 모드는 타일, TD, 메모리 채널이 모두 같은 소켓에 위치하므로 가장 좋은 성능을 보인다. 하지만 캐시 트래픽이

NUMA 경계를 넘는 경우에는 Quadrant /Hemisphere 모드를 사용하는 것보다 효율적이지 않다. (그림 1)은 가장 많이 사용되는 Quadrant 모드에서의 캐시 미스 라우팅 상황을 보여준다.

3. 성능 분석 도구

3.1 성능 모니터링

제온파이 프로세서의 성능 모니터링은 하드웨어 퍼포먼스 유닛(Hardware Performance Units)을 사용하여 많은 정보를 얻을 수 있다. 하드웨어 퍼포먼스 카운터(Hardware Performance Counter)는 하드웨어 관련 이벤트를 수집하는 레지스터로 프로세서의 성능 모니터링 시 사용된다. 하드웨어 퍼포먼스 카운터에서 수집 가능한 성능 관련 이벤트들은 크게 타일(Tile) 이벤트와 언타일(Untile) 이벤트로 나누어진다. 타일 모니터링은 타일 내의 Core, VPU, 캐시 관련된 이벤트를 집계(Counting)하고, 언타일 모니터링은 타일 밖의 요소들과의 통신 및 메모리, 캐시 관련 이벤트를 대상으로 진행된다. <표 1>은 KNL 프로세서의 하드웨어 퍼포먼스 카운터(HPC) 주요 이벤트를 보여준다.

3.2 분석 도구

리눅스 커널 2.6 이상에서는 하드웨어 퍼포먼스 카운터를 이용한 성능 분석을 위해 Perf[5]를 제공한다. Perf를 이용하면 성능 관련 이벤트에 대한 추적(tracing)과 집계가 가능하다. PAPI (Performance Application Programming Interface)[6]는 주로 사용자가 직접 실행 프로그램을 최적화를 할 때 코드 삽입(Instrumentation)의 형태로 하드웨어 퍼포먼스 카운터를 프로파일링 한다. 이외에도 Intel Vtune[7], ARM HPC tools(과거 Allinea)[8], Oregon 대학에서 개발한 TAU[9] 등이 하드웨어 퍼포먼스 카운터를 이용하는 주요 성능 분석 도구이다. 이들은 사용자를 위해 응용 프로그램 실행부터 결과 분석까지 하나의 그래픽 인터페이스를 제공하고 분석 결과 시각화 툴을 포함하기도 한다.

인텔의 Vtune은 고성능 분석(HPC analysis)을 위해 CPU Utilization, memory access, FPU Utilization의 3가지 주요 지표의 분석 결과를 제공한다. Vtune은 대부분의 최신 프로세서까지 지원하기 때문에 단일 인터페이스로 다양한 프

〈표 1〉 KNL 프로세서의 HPC 이벤트[3,4]

HPC 이벤트 유형		주요 이벤트
Tile		Unhalted reference clock cycles, Retired Instructions, L1 and L2 Hit/Miss Loads, Branch-*, DTLB / UTLB-*, L2Q-*
Untile	EDC	EDC Hit/Miss, MCDRAM Clock(ECLK), All read/wire request in MCDRAM cache (RPQ, WPQ),
	MC	DDR4 clock(DCLK), Memory controller -*: Column Access Strobe(CAS)
	CHA	IPQ-*, IRQ-*, ISMQ-*, RxR-*, TOR-*, Transgress-*, TxR-*
	CMS	Agent0-*, Agent1-*
	M2PCIe	RxR-*, TxC-*
	IRP	Outbound request queue-*, Write cache occupancy, Coherent Ops.

로세서의 성능 분석을 진행할 수 있다는 점이 장점이다.

ARM의 HPC tools은 성능 분석을 통해 실행 프로그램의 병목지점 추적을 돕는다. 또한 시스템에서 실행된 프로그램의 계산, 통신, I/O 연산 결과를 분석하여 보고서(Arm Performance Reports[10]) 형태로 제공하는 것이 특징이다.

TAU의 경우 routines, loops과 메모리 관련하여 소스 코드에 삽입하여 성능 분석하기에 유용하고, 하드웨어 성능 카운터를 프로파일링하여 각 routine들의 소요 시간을 분석한다. TAU의 수많은 시각화 툴은 사용자들이 다양한 성능 분석 결과를 얻을 수 있도록 한다.

4. 주요 벤치마크 성능 실험 및 분석

슈퍼컴퓨터의 성능을 측정하기 위한 다양한 시스템 벤치마크 프로그램들이 존재한다. 본 장에서는 대표적으로 가장 널리 사용되는 벤치마크 프로그램인 STREAM, HPLinpack, HPCG를 대상으로 KNL 기반 테스베드 성능 측정 결과를 비교하고 분석해 본다.

4.1 테스트베드 구성

사용한 테스트베드는 1대의 로그인 노드와 총 14대의 계산 노드로 구성되어 있다. 이중 10대의 계산 노드는 인텔 제온파이 프로세서 7250 기반으로 관련 상세 스펙 및 소프트웨어, 환경설정, 및 사용한 벤치마크는 <표 2>와 같다.

4.2 STREAM 벤치마크

가장 기본적으로 널리 사용되고 있는 메모리 성능 벤치마크 프로그램으로, 초기 버전은 1991년 Texas Advanced Computing Center(TACC)의 John D. McCalpin에 의해 개발되었다[11]. 그는 STREAM 벤치마크를 “지속가능한 메모리 대역폭(MB/s)과 단순한 벡터 커널(Copy, Scale, Add, Triad) 계산을 수행하고 측정하기 위한 단순한 종합 벤치마크 도구”라고 정의한다[12]. 인텔 STREAM 최적화 공식 문서에 따르면 KNL 7250의 Quadrant, Flat 모드에서 STREAM Triad 기대 성능은 MCDRAM 최대 475 ~ 490 GB/s, DDR4 최대 90 GB/s 이다. 본 절에서는 인텔 최적화 문서[13]에 따른 기대 성능과 실제

〈표 2〉 제온파이 기반 테스트베드 환경설정

하드웨어	
플랫폼	Intel S7200AP
프로세서	Intel XeonPhi 7250 1.40GHz, 68C
가용 메모리	13 of 16 GB (MCDRAM), 90 of 96 GB (DDR4)
네트워크	Intel Omni-Path 100GB/s

소프트웨어	
운영체제	CentOS 7.3
커널	3.10.0-514.26.2.el7.x86_64
스케줄러	PBS v14.2
파일시스템	Lustre v2.7

환경설정	
메모리 모드	Flat Mode
클러스터 모드	Quadrant

벤치마크	
STREAM	v5.10
HPLinpack	Intel Optimized HPL v2.1 Netlib HPL v2.2
HPCG	Intel MKL 2018(HPCG v3.0)

성능을 검증 해보고 CPU와 메모리 크기의 변화에 따른 성능 결과를 비교하고 분석한다.

본 벤치마크는 메모리 대역폭 점수 결과 인용을 위해 개발자가 정의한 실행규칙을 따르는 표준 버전과 사용자 또는 공급 업체가 수정 된 소스 코드를 기반으로 결과를 제출하는 비표준(튜닝) 버전의 두 가지 범주로 나뉜다. 표준 버전에서 STREAM의 배열 요소 크기는 일반적으로 사용가능한 마지막 레벨 캐시(Last Level Cache, LLC) 메모리 크기의 4배 이상이거나 10 MB 중 큰 값이어야 한다[13]. STREAM 표준에 따라 L2 캐시를 LLC로 고려하였을 때, KNL 7250의 각 배열 크기 요소를 계산해보면, 배열이 더블 타입(8byte) 이므로, 34MB * 4 / 8 = 17,000,000 이 되며, 각 배열 당 메모리 크기는 136 MB로 A,B,C 세 배열 전체가 차지하는 메모리 크기는 408 MB 이다. 그리고 MCDRAM을 LLC로 사용한다면, 각 배열 크기 요소는 16GB * 4 / 8 = 8,000,000,000, 각 배열당 메모리 크기는 64 GB, 전체 메모리 크기는 192 GB가 된다.

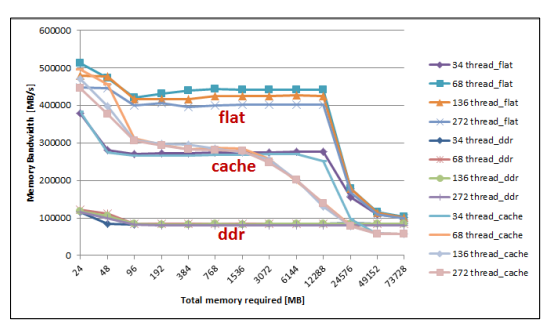
본 실험에서는 STREAM의 최신 표준 공식 버전인 5.10을 이용하였으며, 4가지 벡터 커널 중 가장 복잡한 시나리오인 Triad 커널 계산을 이용하여 실험을 진행하였다. Triad는 복사(Copy, a=b), 곱셈(Scale, a*SCALAR), 덧셈(Add, a+b)가 모두 적용된 벡터 커널로 위의 <표 3>과 같다.

STREAM 벤치마크의 구현 버전은 일반적으로 C와 Fortran의 2가지 버전이 존재하며, 여기서는 C 버전과 인텔 버전의 컴파일러(ICC)를 사용하였다. 인텔에서 권고되는 STREAM 최적화 문서를 기준으로, 정적 메모리 할당, 2MB의 메모리 정렬, 그리고 컴파일러 옵션을 사용한다[12]. 그리고 메모리 모드(cache, ddr, mcdram), STREAM 배열 크기(STREAM_ARRAY_SIZE)와 스레드 개수(OMP_NUM_THREADS)가 성능에 미치는 영향을 살펴보기 위해 각 요소의 변화에 따른 성능 결과를 측정하였다. 이 때, 이용가능한 모든 코어 위에서 스레드가 실행되도록 OpenMP 환경변수인 KMP_AFFINITY를 scatter로 설정한다. 실험 결과는 (그림 2)와 같다.

〈표 3〉 STREAM Triad 벡터 커널

```
#pragma parallel for
    for (i =0; i<N; i++) {
    a[i] = b[i] + c[i] * SCALAR;
 }
```

본 실험에서 벡터의 배열 크기를 최소 1,000,000 부터 최대 3.072,000,000 까지 제곱하여 증가시켰을 때, 전체 할당 메모리 크기는 약 24 MiB 부터 73,728 MiB 이다. 이때 소모되는 전체 메모리 크기가 34 MiB 와 16 GiB 일 때 큰 폭으로 성능이 두 단계 떨어짐을 확인할 수 있다. 그 이유는 모든 코어의 L2 캐시 합의 크기가 34 MiB 이기 때문에 모든 백터 크기가 L2 캐시의 크기보다 클 때 성능이 한번 떨어지며 이후에 MCDRAM의 크기인 16 GiB 보다 클 때 또 한번 성능이 떨어진다. 실험 결과를 살펴보면 먼저 각 메모리 모드는 ddr < cache < flat, 스레드 개수는 34 < 272 < 136 < 68 순으로 성능이 높



(그림 2) STREAM(Triad) 벤치마크를 사용한 메모리 대역폭 비교 실험

아짐을 확인할 수 있다. 즉, flat 모드 68 스레드의 경우 표준 성능이 약 450,000 MB/s로 가장 높고, cache 모드 68 스레드의 표준 성능은 약 260,000 MB/s, ddr 모드 68 스레드의 표준 성능은 약 85,000 MB/s로 위에서 언급한 기대 성능치와 거의 비슷하다.

4.3 Linpack 벤치마크

Linpack 벤치마크는 분산 메모리 컴퓨터에서 무작위 계수를 갖는 선형 대수 방정식의 대규모 시스템을 생성하고 해결하기 위해 1980년대 초 Jack Dongrra 등에 의해 개발되었다[14]. 즉, 선형 방정식 Ax * b의 조밀한 n * n 시스템을 풀고, 시스템을 분석하고 해결하는데 걸리는 시간을 측정하고, 그 시간을 성능 속도로 변환하고 결과의 정확성을 테스트한다. 이때 시스템의 성능은 초당 부동소수점 연산(FLOP/s)으로 표현된다.

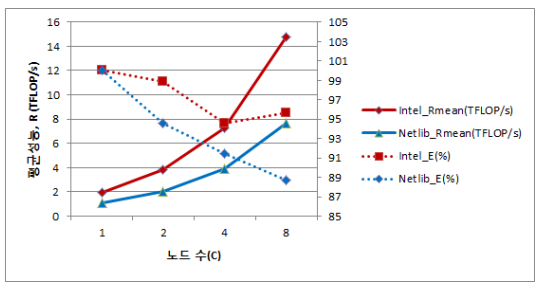
본 벤치마크의 최신버전인 HPLinpack은 결과를 표준화하기 위해 이식 가능한 구현체로서의 소프트웨어 패키지인 HPL 코드가 사용된다[15]. 본 코드는 실제 병렬 고성능 컴퓨팅 시스템(HPC)의 부동 소수점 연산의 성능 측정을 위한 산업 표준 테스트로, 세계에서 가장 강력한 컴퓨터 시스템의 목록을 정기적으로 업데이트하는 TOP 500 프로젝트의 기준으로 활용되고 있다. 본 절에서는 제온파이 테스트베드를 기반으로 인텔에서 소스 최적화한 HPLinpack 벤치마크와 기존 Netlib의 HPLinpack를 이용하여 단일 노드 및 클러스터 환경에서의 성능을 측정하고 비교 분석한다.

HPL은 HPL.dat라는 입력 텍스트 파일을 통해서 사용자가 벤치마크 매개변수를 조정하고, 인수분해 알고리즘을 선택할 수 있다. 이 중 성능에 가장 크게 영향을 미치며 중요한 매개변수인 N, NB, P와 Q에 초점을 맞추고 나머지 매개변수는 기본 값을 유지한다. 여기서 N은 문제크기이며, HPL에 의해 사용되는 메모리 크기를 나타낸다. 이 값은 최대한 메모리 크기에 맞추는 것이 적절하나, 시스템 프로그램 등의 필수 사용 메모리를 제외하고, 일반적으로 전체 사용가능한 메모리의 약 80% 정도로 할당하는 것을 추천하며, NB는 분배되는 데이터의 블록 크기로 인텔에서는 제온파이 프로세서에 최적화된 값으로 336을 제시한다[16]. P와 Q는 분산되어지는 격자의 프로세스 행과 열의 수를 나타내며, P와 Q의 곱은 MPI 프로세스의 수와 같아야 한다. P와 Q값은 서로 같거나 Q값이 크고, 보통 두 값의 차이가 적을수록 성능이 좋다.

본 실험은 인텔에서 최적화한 정적 메모리 HPL 코드와 코드 수정을 하지 않은 기존 Netlib HPL 소스에 인텔 컴파일러 최적화만을 실시하였다. 두 코드는 단일 노드에서는 공유 메모리 형태의 272개의 스레드 방식을 사용하고 다중 노드 간에는 MPI 통신 모델을 사용한다. 실험 노드의 개수(C)를 1, 2, 4, 8로 증가시켰을 때, 인

〈표 4〉 인텔 최적화 HPLinpack 성능 및 병렬 확장성 효율 비교

노드(사용 코어)	n	P(#)	Q(#)	Memory(%)	Rmean(TFLOP/s)	E(%)
1(68)	100000	1	1	75%	1.93	100%
2(136)	140000	1	2	74%	3.81	98.9
4(272)	210000	2	2	84%	7.30	94.6
8(544)	290000	2	4	81%	14.75	95.6



(그림 3) HPLinpack 소스 최적화에 따른 성능 비교

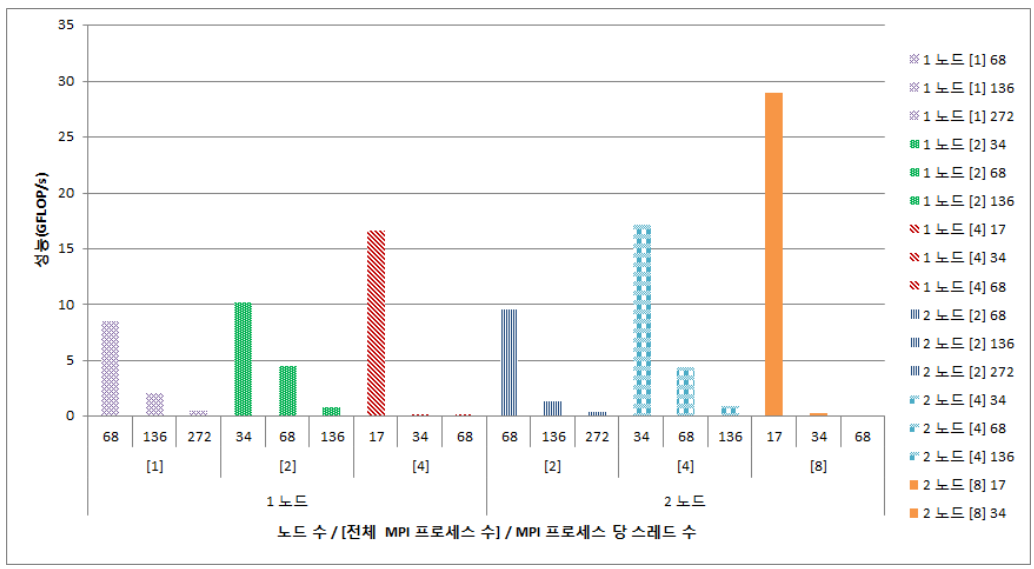
텔 코드에서의 최대 성능 값은 <표 4>와 같다. 각 실험에서 블록 크기는 최소 10,000 부터 최대 메모리로 할당이 될 때까지 10,000씩 증가시키며, 그 중에서 가장 성능이 높은 블록 크기를 선택하였다. 실험 결과, 10대 노드 미만까지는 확장 성능 효율이 약 95% 이상으로 유지된다. 또한 각 블록 크기가 실제적으로 전체 메모리의 약 80% 전후로 할당되었을 때 성능이 가장 높게 나온다. 본 실험 결과는 미국의 제온파이 기반 Colfax 클러스터에서 실험한 2017년 7월 벤치마크 보고서와 거의 비슷하다[17]. (그림 3)은 인텔에서 소스를 최적화한 버전과 기존 Netlib의 HPL을 비교하여 노드 수에 따른 성능을 측정한 그래프이다. 최적화에 따라서 최대 약 50%의 성능과 7%의 병렬 효율성 차이를 확인할 수 있으며, 시스템 구조에 맞는 병렬 프로그램 최적화가 중요한 요소임을 알 수 있다.

4.4 HPCG 벤치마크

HPCG(High Performance Conjugate Gradients) 벤치마크는 다양한 클러스터 응용 프로그램에 보다 잘 부합하는 기준을 제공함으로써 TOP500 시스템 순위에서 사용되는 고성능 LINPACK 벤치마크를 보완하기 위해 2013년 Michael Heroux, Jack Dongarra, Piotr Luszczek에 의해 제안되었다[18,19]. Jack Dongrra는 “기존의 LINPACK 벤치마크는 로컬 대역폭과 네트워크에 충분한 스트레스가 주어지지 않아 일반적으로 얻을 수 없는 성능레벨”이라고 한계점을 언급하였다. 반면에 HPCG는 희소행렬의 계산 같은 실세계 응용의 데이터 접근 패턴을 모델링하기 위한 것으로 슈퍼컴퓨터의 메모리 하위 시스템 및 내부 상호 연결의 한계가 컴퓨팅 성능에 미치는 영향을 테스트 한다. 따라서 HPCG 테스트는 일반적으로 컴퓨터의 최대 성능 중 극히 일부만을 수행하기 때문에 성능 측정을 위해 많은 시간을 필요로 하지 않는다. 본 절에서는 인텔에서 최적화한 버전의 HPCG[20]를 가지고, 입력 변수인 문제크기와 실행 시간에 따른 성능 차이를 확인하고 분석한다.

HPCG 방식은 3D 도메인의 각 그리드 지점(x,y,z)에서 27 포인트 스텐실을 사용하여(한 점에서의 방정식이 그 값과 주변 26 지점에 따라 달라짐) 논리적으로 전역적이며 물리적으로 분산된 희소 선형시스템을 생성한다. 세부적으로는 SPMV(Sparse Matrix-Vector Multiplication), SymGS(Symmetric Gauss-Seidel), WAXPBY (Scaled vector addition), DDOT(Dot Product)의 네가지 계산 블록과 MPI_Allreduce와 Halos Exchange의 두 통신 블록을 사용한다.

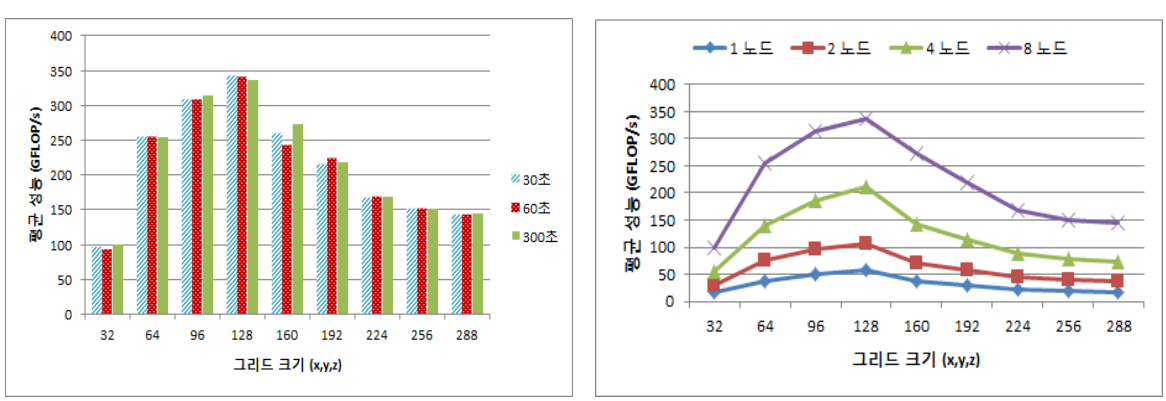
HPCG의 입력 변수는 2가지로 지역 도메인의 그리드 크기(x,y,z)와 실행시간(t)이 있다. HPCG 공식 문서에 따르면 실행에 적합한 문제 크기, 즉 그리드 크기는 CPU 캐시 보다 커야하고 총 메모리의 1/4 이상을 차지해야한다. 이때 그리드 크기는 최소 4이며, 4의 배수여야 한다. 또한 Top 500의 결과로 활용하기 위해서는 실행 시간을 최소 30분 이상 실행하도록 권고한다. HPLinpack과 마찬가지로 HPCG도 OpenMP / MPI 방식의 하이브리드 모델을 사용한다. 먼저 MPI 프로세스와 OpenMP 스레드 개수에 따른



(그림 4) 노드, MPI 프로세스, 스레드 개수의 조합에 따른 성능 비교

성능 변화를 직접 확인해 보기 위해 그리드 사이즈를 32로 작게 선택하고, 한 노드와 두 노드를 대상으로 성능을 측정한 결과는 (그림 4)와 같다. 결과에서 살펴볼 수 있듯이, 노드 당 MPI 프로세스 수는 4개 (2 노드의 경우는 8개), MPI 프로세스 당 스레드 수는 17 일 때 가장 좋은 성능을 보여준다.

본 조합의 결과를 기반으로 시간과 그리드 크기의 변화에 따른 실험 결과는 (그림 5)에 나타나 있다. 왼쪽 그림은 8 노드 OpenMP/MPI 실험으로 각 그리드 크기를 32에서 288까지 32 간격으로 증가시켰을 때, 30초, 60초, 300초별 성능을 비교한 결과로, 시간의 길이에 크게 변화가 없음을 알 수 있다. 즉, 단시간의 성능측정으로도 정확한 시스템 성능 측정이 가능하다[21]. 오른쪽 그림은 노드 수 증가에 따라 성능이 거의 선형적으로 비례하여 증가함을 확인할 수 있으며, 초기에 그리드 크기와 성능이 같이 증가하다, 그



(그림 5) 시간 및 그리드 크기의 변화에 따른 성능 결과

리드 크기가 128에 이르렀을 때 최대 성능 약 340 GFlop/s 에 이르고, 다시 서서히 떨어진다. 이는 그리드 크기가 MCDRAM의 최대 크기인 16 GB를 벗어나서, 일반 DDR 메모리를 사용하기 때문이다.

5. 결 론

본고에서는 인텔 차세대 매니코어 시스템인 KNL을 활용한 성능 분석과 벤치마크 실험을 위해 먼저 KNL에 대해서 간략히 소개한 후 대표적인 분석 도구에 대해 살펴보았다. 그리고 KNL 기반 테스트베드를 활용하여 3가지 주요 벤치마크 프로그램을 통해 성능을 측정하고 분석하여 성능에 영향을 미치는 요소들을 살펴보고 시스템 구조에 맞는 병렬 프로그램 최적화가 필요함을 보였다. 첫 번째로 STREAM 벤치마크의 배열 크기를 변화 시키면서 최적의 메모리 모드와 스레드 개수를 분석하였다. STREAM 벤치마크를 사용하여 고대역폭 온칩 메모리를 포함하는 인텔 제온파이의 메모리 성능을 측정한 결과 코어당 1 스레드로 최대 사용했을 때 가장 높은 성능을 보였으며, 실제 성능이 인텔 최적화 문서에 따른 기대 성능과 거의 비슷함을 확인하였다. 또한 인텔과 Netlib의 두 가지 HPLinpack 벤치마크 버전을 이용하여 단일 노드 및 클러스터 환경에서의 실험 결과, 블록크기가 전체메모리의 약 80%정도 할당 되었을 때 최고 성능을 보였으며 노드 수 증가에 따라 거의 선형적인 성능 증가를 보였다. 또한 최적화에 따라서 약 50%의 성능과 7%의 병렬 효율성 차이를 보였다. 마지막으로 HPCG 벤치마크 실험을 통해 MPI 프로세스 및 OpenMP 스레드 개수에 따른 성능 분석 결과 노드 당 MPI 프로세스는 4, 전체 스레드 수가 68일 때 가장 좋은 성능을 보임을 확인하였다. 또한 시간의 길이에 관계없이 단시간의 성능 측정으로도 거의 정확한 시스템 성능 측정이 가능함을 보였다.

참 고 문 헌

[1] http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017/

[2] A. Sodani, “Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor.” Hot Chips 27 Symposium (HCS) IEEE 2015, 2015.

[3] 최지은, 박근철, 남덕윤, “차세대 매니코어 프로세서 기반 성능 모니터링 이벤트를 활용한 응용 특성 분석 기법”, 2017년 한국소프트웨어종합학술대회 논문집, 2017년 12월.

[4] Harini R. “Intel(R) Xeon(R) Phi(TM) Processor Performance Monitoring Reference Manual” published on November 2, 2015, updated March, 2017.

[5] Arnaldo Carvalho de Melo, “The New Linux ‘Perf’ tools”, presentation from Linux Kongress, September, 2010.

[6] PAPI, http://icl.cs.utk.edu/papi/

[7] VTune, https://software.intel.com/en-us/intel-vtune-amplifier-xe

[8] HPC Tools, https://developer.arm.com/products/software-development-tools/hpc/arm-performance-reports

[9] S. Shende and A. D. Malony., “The TAU Parallel Performance System”, International Journal of High Performance Computing Applications, Volume 20 Number 2, pp 287-311, 2006.

[10] ARM Performance Reports, https://www.arm.com/products/development-tools/hpc-tools/cross-platform/performance-reports

[11] McCalpin, John D.: "STREAM: Sustainable Memory Bandwidth in High Performance Computers", a continually updated technical report (1991-2007), available at: "http://www.cs.virginia.edu/stream/"

[12] McCalpin, John D., 1995: "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

[13] https://software.intel.com/en-us/articles/optimizing-

memory-bandwidth-in-knights-landing-on-stream-triad
[14] J. J. Dongarra, P. Luszczek, and A. Petitet "The LINPACK Benchmark: Past, Present, and Future." Concurrency and Computation: Practice and Experience vol. 15, no. 9, pp. 803-820, August, 2003.
[15] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary "HPL-A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers", 2016. http://www.netlib.org/benchmark/hpl/
[16] https://software.intel.com/en-us/node/725943
[17] https://colfaxresearch.com/hpl-on-xeon-phi-x200/
[18] Hemsoth, Nicole (June 26, 2014). "New HPC Benchmark Delivers Promising Results". HPCWire. Retrieved 2014-09-08.
[19] Dongarra, Jack; Heroux, Michael (June 2013). "Toward a New Metric for Ranking High Performance Computing Systems" (PDF). Sandia National Laboratory. Retrieved 2016-07-04.
[20] https://software.intel.com/en-us/mkl-linux-developer-guide-getting-started-with-intel-optimized-hpcg
[21] http://en.community.dell.com/techcenter/high-performance-computing/b/general_hpc/archive/ 2017/01/17/hpcg-performance-study-with-intel-knl

저 자 약 력

노 승 우

이메일 : seungwoo0926@kisti.re.kr

- 2009년 서울시립대학교 전자전기컴퓨터공학부 (학사)
- 2011년 서울시립대학교 전자전기컴퓨터공학과 (석사)
- 2011년~2013년 서울시립대 UGL Soft / 선임연구원
- 2011년~현재 한국과학기술정보연구원(KISTI) / 선임기술원
- 관심분야 : 고성능컴퓨팅, 분산컴퓨팅, 시스템 벤치마킹, 프로파일링

최 지 은

이메일 : jieun1205@kisti.re.kr

- 2014년 숙명여자대학교 컴퓨터과학부 (학사)
- 2016년 숙명여자대학교 컴퓨터과학과 (석사)
- 2017년~현재 한국과학기술정보연구원 / 기술원
- 관심분야 : 고성능컴퓨터, 시스템 프로파일링

남 덕 윤

이메일 : dynam@kisti.re.kr

- 1999년 포항공과대학교 컴퓨터공학과(학사)
- 2001년 한국정보통신대학교 공학부(석사)
- 2006년 한국정보통신대학교 공학부(박사)
- 2004년~현재 한국과학기술정보연구원(KISTI) / 선임연구원
- 관심분야 : 분산시스템, 슈퍼컴퓨팅, 저전력컴퓨팅

박 근 철

이메일 : gcpark@kisti.re.kr

- 1998년 중앙대학교 컴퓨터공학과 (학사)
- 2000년 중앙대학교 컴퓨터공학과 (석사)
- 1998년~2000년 (주)넥센
- 2000년~2005년 (주)창성정보시스템
- 2005년~2005년 (주)이스톰
- 2006년~현재 한국과학기술정보연구원(KISTI) / 선임연구원
- 관심분야 : 분산컴퓨팅, 프로파일링, 작업최적화

박 찬 열

이메일 : chan@kisti.re.kr

- 1993년 고려대학교 수학과 (학사)
- 1995년 고려대학교 전산학과 (석사)
- 2000년 고려대학교 전산학과 (박사)
- 2010년~2011년 누욕주립대 Visiting Scholar
- 2002년~현재 한국과학기술정보연구원 (KISTI) / 책임연구원
- 관심분야 : 분산시스템, 슈퍼컴퓨팅, 저전력컴퓨팅