JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 3, pp. 597-603, Mar. 2019

멀티코어 및 매니코어 아키텍처에서 시스템 특징 및 성능 분석

윤 준 원[1] · 송 의 성[2*]
[1]한국과학기술정보연구원 슈퍼컴퓨팅본부
[2]부산교육대학교 컴퓨터교육과

System Characteristics and Performance Analysis in Multi and Many-core Architectures

JunWeon Yoon[1] · Ui-Sung Song[2*]
[1]Department of Supercomputing Center, KISTI, Daejeon 34141, Korea
[2]Department of Computer Education, Busan National University of Education, Busan 47503, Korea

[요 약]

최근 x86 기반의 시스템은 단일 CPU에 20개 이상의 코어가 장착될 정도로 멀티 및 매니코어 아키텍처 환경이 일반화 되고 있으며, 다양한 어플리케이션의 계산 수행 성능을 높이기 위해 GPU, Intel Phi 등 가속기 기반의 환경도 보편화 되고 있다. 이런 다양한 아키텍처에서 최적의 계산 수행 결과를 얻기 위해서는 어플리케이션의 특징에 맞는 시스템 환경을 선택해야 한다. 실제 현업 연구자들의 어플리케이션 성능은 클러스터 환경을 구성하는 CPU, 메모리, 스토리지, 네트워크 등의 특성에 따라 성능이 달라진다.

본 연구에서는 최근 이슈가 되는 멀티코어 및 매니코어 아키텍처에서의 시스템 특징을 연구하고 성능 분석을 수행한다. 이 실험을 위해 클러스터 성능 벤치마크 도구들을 활용하여 프로세서, 메모리, 캐시, 파일시스템 등의 요소별 특징을 파악하고 비교한다. 이로써 다양한 멀티 및 매니코어 아키텍처의 특징을 구분하고 어플리케이션 수행에 필요한 시스템 환경을 제시할 수 있다.

[Abstract]

The x86-based system is now becoming more common in multi and many-core architecture environments, with more than 20 cores on a single CPU. Also, accelerator-based architectures such as GPU and Intel Phi are also popular in order to improve the computational performance in various applications. In each of these architectures, the system environment that matches the application characteristics must be selected in order to obtain the optimum result for application execution. Application performance of real-world researchers depends on the characteristics of the CPU, memory, storage, and network that make up the cluster environment.

In this paper, we analyze the system characteristics and execute the performance experiments in multi and many-core architectures. For this experiment, we use cluster performance benchmarking tools to identify and compare elemental features such as processor, memory, cache, and file system. This makes it possible to distinguish the characteristics of various multi and man-core architectures and to present the system environment required for application execution.

색인어 : 매니코어, 멀티코어, 가속기, GPU, Intel Phi, 벤치마크
Key word : Many-core, Multi-core, Accelerator, GPU, Intel Phi, Benchmark

http://dx.doi.org/10.9728/dcs.2019.20.3.597

Received 15 January 2019; **Revised** 18 February 2019
Accepted 20 March 2019

***Corresponding Author; Ui-Sung Song**

Tel: +82-51-500-7326
E-mail: ussong@bnue.ac.kr

Ⅰ. 서 론

최근 x86 기반의 멀티코어(multi-core) CPU가 대중화 되고 더욱이 가속 프로세서(Intel PHI, NVIDIA GPU 등)가 탑재되면서 단일 노드의 성능이 급격히 향상되고 있다. 특히 Intel Xeon Phi 제품군의 2세대 버전인 Knights Landing(KNL)은 부팅이 가능한 호스트 프로세서로 60~70여개의 코어가 집적된 매니코어(many-core) 아키텍처이다[1].

무어의 법칙에 따르면 반도체 장치의 성능향상은 2년마다 성능이 2배로 증가한다는 개념 이였으나 2,000년 중반부터는 클럭 주파수를 높이는 개별 프로세서의 향상에서 벗어나 복수개의 프로세서 코어를 하나의 CPU에 집적하는 매니코어 환경으로 변화되고 있다[2]. 최근에는 병렬 연산을 증대하기 위해 가속기 기반의 이기종(Heterogenous) 아키텍처가 TOP500 슈퍼컴퓨터 리스트 내에도 대다수 포진되어있다. NVIDIA의 GPU, Intel Xeon Phi coprocessor가 대표적인 예로 앞으로도 고성능 프로세서에 대한 매니코어 환경이 더욱 가속화 될 것이다. 최근 슈퍼컴퓨터 TOP500 리스트에 등재된 슈퍼컴퓨터의 타입을 보면 대표적으로 다음의 3가지 시스템으로 구분된다[3].

- Intel 또는 AMD의 CPU 프로세서 기반 시스템
- NVIDA 또는 AMD의 GPU 기반의 이기종 시스템
- Intel Xeon Phi 기반의 Many-core 시스템

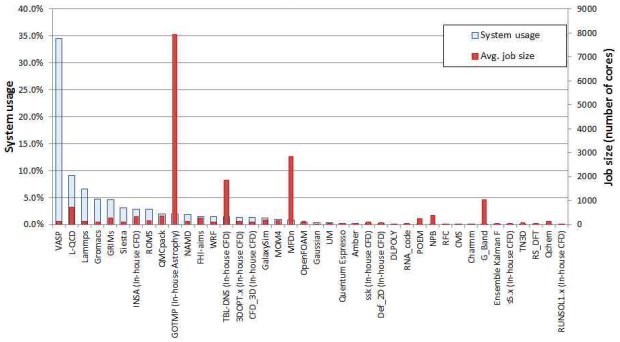


그림 1. HPC 어플리케이션 및 시스템 사용량
Fig 1. HPC Applications and System Usages
[Source: KISTI Tachyon2 Supercomputer]

HPC 환경은 다양한 어플리케이션들이 자원 요구사항에 맞게 수행되고 있다. (그림 1은) HPC의 병렬환경에서 수행됐던 어플리케이션과 사용량을 보여주고 있으며, 대표적으로 양자역학(VASP, Gaussian), 분자동역학(Lammps,Amber), 구조역학(Abaqus) 등 있으며 각각의 어플리케이션들은 CPU, 메모리, 파일 I/O 등의 특성에 따라 큰 성능차이를 보인다.

본 연구에서는 멀티코어와 매니코어 환경에서의 시스템 특징을 조사하고 각 요소별 성능 분석을 수행하여 제시함으로서 (그림 1)과 같이 대규모 병렬처리를 요구하는 어플리케이션 사용자에게 중요한 지침이 되고자 한다.

Ⅱ. 관련 연구

멀티코어(multi-core)와 매니코어(many-core) 차이는 칩셋에 구성된 코어 내의 메모리 구성 방식이 공유 메모리 또는 분산 공유 메모리에 따라 구분하기도 한다. 즉, 멀티코어의 경우 모든 코어가 동일한 메모리 영역을 공유하고 매니코어의 경우 코어 내의 그룹에서는 메모리를 공유하나 그룹과 그룹 간에는 메모리를 공유하지 않는 개념을 둔다[4]. 최근에는 그 경계가 모호하나 본 연구에서는 Intel Xeon 기반의 CPU를 멀티코어 Xeon Phi 아키텍처 또는 GPU 경우를 매니코어로 지칭하기도 한다.

HPC 환경은 대규모의 계산 자원들이 고속의 인터커넥션 네트워크로 연결되어 있으며, OS부터 이를 관리하는 다양한 시스템 소프트웨어들이 설치되어 있다. 이렇게 유기적으로 구성된 HPC 시스템에서 작업들이 신뢰성 있게 통신하고 수행되려면 각 구성 요소의 상태와 성능을 확인하고 보장해야 한다.

본 연구에서는 최근 HPC 환경에서 주로 사용되고 있는 멀티 및 매니코어 아키텍처의 특징을 파악하고 성능을 분석한다.

2-1 Intel 멀티 및 매니코어 아키텍처

일반적으로 CPU 기반의 마이크로 아키텍처는 코어마다 L1, L2 캐시가 있고 모든 코어가 공유하는 L3 캐시로 구성된다. <표 1>은 Intel CPU 아키텍처 세대별 최대 코어 수와 쓰레드 수의 증가를 보여준다.

표 1. 세대별 Intel CPU 최대 코어 및 쓰레드 수
Table 1. The maximum number of cores and threads by Intel CPU generation

CPU Model	Max Core Count	Thread Count
Westmere	10	20
IvyBridge	15	30
Haswell	18	36
Broadwell	24	48
Skylake	28	56
Knight Landing	72	288 (4threads/core)

이전 세대인 Intel Xeon 프로세서 계열(Haswell 및 Broadwell)에서는 프로세서, 코어, LLC (Last Level Cache), 메모리 컨트롤러, IO 컨트롤러 및 소켓 간 QPI(Quick Path Interconnect) 포트는 링 아키텍처를 사용하여 연결된다[5]. 이 방식은 지난 몇 세대 동안 인텔 멀티 코어 CPU에 적용되었다. 각 세대마다 CPU의 코어 수가 증가하면 액세스 대기 시간이 늘어나고 코어 당 사용 가능한 대역폭이 줄어들면서 (그림 2)와 같이 칩을 두 부분으로 나누고 두 번째 링을 도입하여 거리를 줄이고 대역폭을 추가함으로써 완화되었다. 그러나 인텔이 그

동안 사용해온 링 버스는 코어 수와 코어 증가 등으로 확장시 링 버스를 여러 개 사용해 설계가 복잡해지고 그에 따른 데이터 전송이나 지연 현상으로 효율이 하락하는 문제가 발생했다. 또한 인터페이스의 대역폭과 클럭 속도 증가에 따라 비대해지고 효율이 낮아지는 매니코어 확장을 위해 새로운 설계 방식이 필요해졌다.

즉, 인텔 프로세서 아키텍처의 당 코어 증가와 높은 메모리 및 I/O 대역폭을 사용하면서 칩 내의 인터커넥트에 링 기반 아키텍처에서 성능 제한이 발생할 수 있다. 이에 Intel Xeon 프로세서 Scalable 제품군은 이전 링 기반 아키텍처와 관련된 대기 시간 및 대역폭 제약을 완화하기 위해 메시 아키텍처를 도입했다. 메시 아키텍처는 높은 대역폭과 낮은 지연시간, 전력 효율 향상, 멀티코어 확장에 이점을 가진다[6].

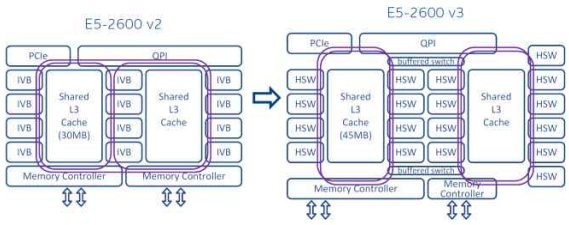


그림 2. 인텔 제온 온다이 인터커넥트 향상
Fig 2. Intel Xeon On-Die Interconnect Enhancements

Ⅲ. 시스템 환경

멀티코어/매니코어 환경의 성능 측정은 KNL 노드(Xeon Phi 7250)와 SKL 노드(Xeon Gold 6148)로 구성되어 있는 리눅스 클러스터에서 수행하였다. 클러스터를 구성하는 각 계산노드는 (그림 3)과 같이 100Gbps의 고속 인터커넥션 네트워크로 연결 되어 있다.

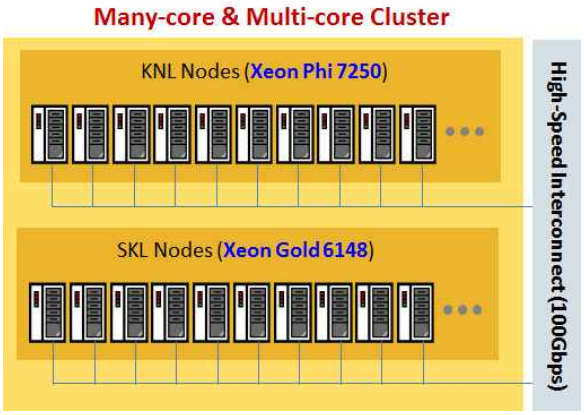


그림 3. 멀티코어/매니코어 클러스터 시스템
Fig 3. Multi-core/Many-core Cluster System

3-1 KNL 노드

KNL 노드는 Xeon Phi 7250 프로세서와 96GB DDR 메모리가 장착되어 있는 매니코어 시스템으로 (그림 4)와 같이 프로세서 내에 68개의 CPU 코어가 34개의 타일로 구성되어 있으며, 각 타일은 2 개의 CPU 코어와 1MB L2 캐시가 공유하는 형태로 구성되며 2D 메시에 배치된다[7].

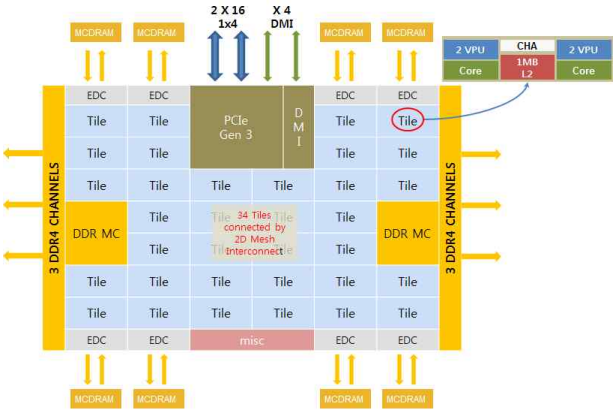


그림 4. 인텔 제온 파이 7250 아키텍쳐
Fig 4. Intel Xeon Phi 7250 architecture

KNL 프로세서에는 6 개의 DDR 채널이 있으며 8개의 Multi-Channel Dynamic Random Access memory (MCDRAM) 컨트롤러가 있다. MCDRAM은 16GB 온패키지 High-Bandwidth Memory(HBM)로 L3 캐시, 플랫, 하이브리드형태의 3가지 모드로 작동한다. 이론성능으로 DDR4(90GB/s) 메모리에 비해 MCDRAM(490GB/s)은 약 5배의 높은 대역폭을 갖는다. KNL 노드의 각 타일은 CPU는 캐시 일관성의 성능 등을 감안하여 NUMA로 클러스터링 된다. NUMA 통신은 All2All, SNC-2/4, Hemisphere, Quadrant 5 가지 모드로 구성 할 수 있다.

3-2 SKL 노드

SKL 노드는 Intel Skylake 프로세서가 듀얼 소켓에 장착되어 있으며 DDR4 192GB 메모리가 6채널로 구성되어 있다. 프로세서는 기본 클럭 속도가 2.4GHz이며 20개의 코어를 가진 Xeon Gold 6148 모델이다. Skylake는 Broadwell과 동일한 14nm 제조 공정 기술을 사용하나 소켓 당 코어의 수가 증가하면 대기 시간이 증가하고 대역폭이 제한된다는 단점이 있다. 이 문제를 완화하기 위해 Skylake는 (그림 5)와 같이 수직 및 수평 경로 배열을 갖는 메시 아키텍처를 사용하여 최단 경로를 통해 하나의 코어에서 다른 코어로 통신 할 수 있다[8]. 두 개의 소켓은 이전 세대의 Intel Xeon 프로세서에서 사용되었던 Intel Quick Path Interconnect(QPI)보다 향상된 대역폭과 성능을 제공하는 두 개의 Intel Ultra Path Interconnect(UPI) 링크로 연결된다.

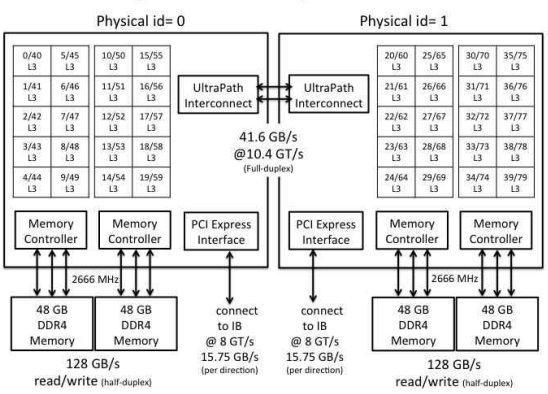


그림 5. 제온 스카이레이크 프로세서 구성도
Fig 5. Xeon Skylake Processor Configuration
[Source: NASA High-End Computing Capability(HECC)]

UPI는 초당 10.4 기가 (GT)의 속도로 실행되며, 각 링크에는 두 방향에 대한 별도의 레인이 있다. 총 양방향 대역폭 (2 링크 x 2 방향)은 초당 41.6 GB/s 이다[9].

Ⅳ. 성능 분석

4-1 연산 성능

이론성능(Rpeak)은 CPU 코어 수, 클록 주파수 및 부동 소수점 명령어를 곱하여 얻은 시스템의 이론상 최대 성능이다. 기본적으로 사용되는 운영 체제 커널 또는 운영 프로그램으로 인해 시스템 부하가 발생하는데 이를 감안한 실측성능 (Rmax)은 이론성능 (Rpeak) 보다 낮다. KNL노드(식 1)와 SKL 노드(식 2)의 이론성능 (Rpeak)은 다음과 같이 얻을 수 있다.

$$KNL\ Node = 68\,cores \times 32\,FPU \times 1.4\,GHz = 3{,}046.4\ GFlops \quad (1)$$

$$SKL\ Node = 40\,cores \times 32\,FPU \times 2.4\,GHz = 3{,}072\ GFlops \quad (2)$$

실측성능은 Linpack 벤치마크를 이용하여 측정할 수 있는데 이는 LU 인수분해를 사용하여 컴퓨터가 얼마나 빨리 선형 방정식 $Ax = b$를 풀 수 있는지를 결정한다[10]. 프로그램을 실행하려면 MPI (Message Passing Interface) 라이브러리 [11]와 BLAS, CBLAS 또는 ATLAS [12]와 같은 매트릭스 곱셈을 제공하는 수학 라이브러리가 있어야 한다. 시스템의 속성은 입력 변수 파일 (HPL.dat)을 조정하여 설정한다.

KNL노드에서 메모리 모드를 Flat으로 고정하고 클러스터 모드를 변경하면서 단일 노드부터 16노드까지 확장 테스트를 수행하였다. All2All의 경우 이론성능 대비 실측성능이 약 60%를 나타냈다. SNC2, SNC4는 OS 레벨에서 NUMA 모드의 칩을 절반 또는 사분면의 도메인 나눈다. 따라서 각 도메인에 맞게 어플리케이션을 최적화하여 수행하지 않으면 (그림 6)과 같이 크게 성능이 저하된다. 더욱이 Xeon Phi 7250 모델의 경우 34개의 타일이 4사 분면에 균등하게 분산 될 수 없기 때문에 SNC4의 경우 성능 저하가 더욱 심해진다.

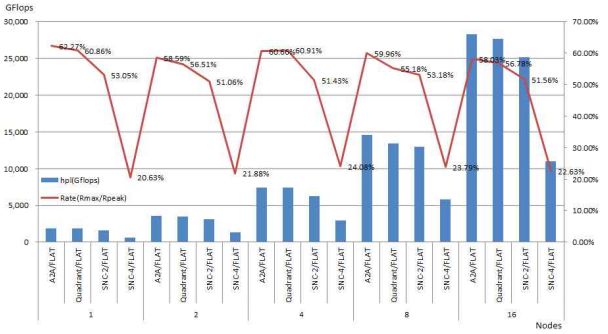


그림 6. 클러스터 모드 KNL(제온 파이 7250) 노드 HPL 벤치마크
Fig 6. KNL Node(Xeon Phi 7250) HPL Benchmark by Cluster Modes

(그림 7)은 SKL 노드에서 마찬가지로 단일노드에서 16노드까지 확장성 테스트를 수행하였으며 이론성능 대비 실측성능은 약 51%~66%를 나타냈다. 노드의 수가 증가할수록 노드간 통신의 부하로 성능 비율이 조금씩 낮아졌다.

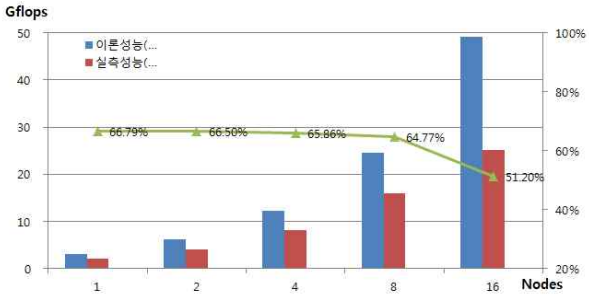


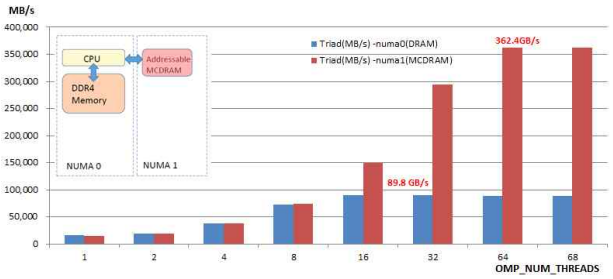
그림 7. SKL 노드(Xeon Gold 6148) HPL 벤치마크
Fig 7. SKL 노드(Xeon Gold 6148) HPL Benchmark

4-2 메모리 성능

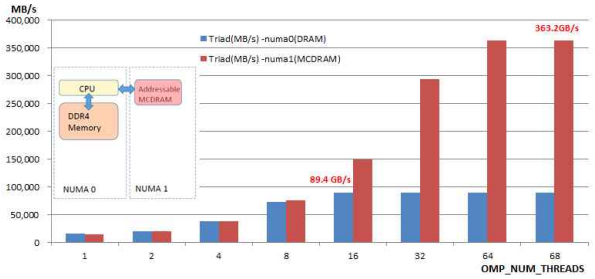
단일노드에서 프로세서와 메모리 간의 대역폭을 측정하기 위해 STREAM 벤치마크를 사용하였다[13]. 성능 테스트는 단일노드에 구성된 코어의 개수 내에서 쓰레드 수 (OMP_NUM_THREADS)를 증가시키면서 수행하였다.

KNL 노드는 앞서 언급했듯이, DDR4 메모리와 온칩 고속 메모리인 MCDRAM의 2가지 유형의 메모리가 장착되어있다. 두 형태의 메모리는 numactl 명령을 사용하여 아래와 같이 membind 옵션을 통해 직접 액세스 할 수 있다. 물리적인 대역폭은 MCDRAM (490GB/s), DDR4(90GB/s) 이며 아래와 같이 각각은 node 0와 node 1로 할당된다. (그림 8)은 All2All 및 Quadrant 모드에서의 성능 비교를 보여준다.

- *node 0 (DDR4, 98207 MB) #numactl --membind 0 stream*
- *node 1 (MCDRAM, 16384 MB) #numactl --membind 1 stream*



(a) Cluster Mode: All2All, Memory Mode: Flat



(b) Cluster Mode: Quadrant, Memory Mode: Flat

그림 8. KNL 노드에서 STREAM 성능 테스트
Fig 8. STREAM Performance Test on the KNL Node

KNL 노드의 경우 클러스터 모드에 따른 차이는 크게 나타지 않았다. 대역폭은 MCDRAM의 경우 이론성능(490GB/s) 대비 약 360GB/s, DDR4의 경우는 이론성능(90GB/s) 대비 약 89GB/s의 성능을 나타냈다(ARRAY_SIZE=134217728)[14]. SKL 노드의 경우 (그림 5)에서 보듯이 2개의 CPU가 장착되어 있으며 총 메모리 대역폭은 256GB/s(2x128)이나 두 소켓을 연결하는 UPI 대역폭이 41.6GB/s로 쓰레드 개수가 커질수록 병목현상이 발생하게 된다. (그림 9)와 같이 SKL노드의 STREAM의 경우 최대 성능은 177.2GB/s(ARRAY_SIZE=64000000)를 나타냈다.

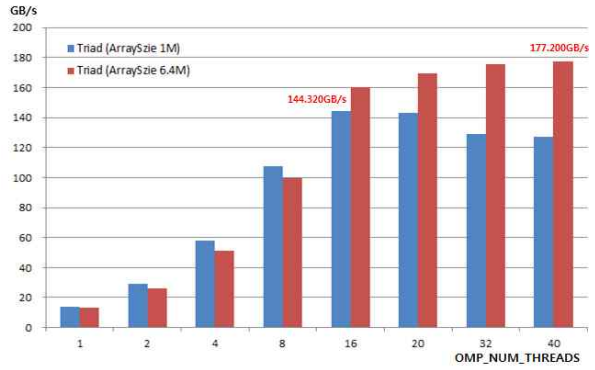


그림 9. SKL 노드에서 STREAM 성능 테스트
Fig 9. STREAM Performance Test on the SKL Node

4-3 캐시 성능

캐시 기능은 디스크에서 매번 데이터를 읽는 대신 메모리 버퍼에 캐싱함으로써 파일 시스템 읽기 성능을 향상시킬 수 있다. 즉, 디스크에서 항상 읽는 대신 고정 크기 메모리 버퍼 (블록 버퍼 캐시라고 함)에 데이터를 저장하여 성능 부하를 줄일 수 있다. 클러스터 컴퓨팅 환경에서 캐싱 성능을 측정하기 위해서 단일 노드에서 파일 크기를 증가시키며 수행한다.

파일 크기는 (식 3)과 같이 결정된다. 전송 크기는 작업 당 전송 프로세스(단일 I/O 호출)의 데이터 크기이며 지정된 블록 크기만큼 전송된다. 따라서 블록 크기는 전송 크기 보다 크거나 같다[15]. 응용 프로그램의 데이터는 일련의 세그먼트로 구성된다.

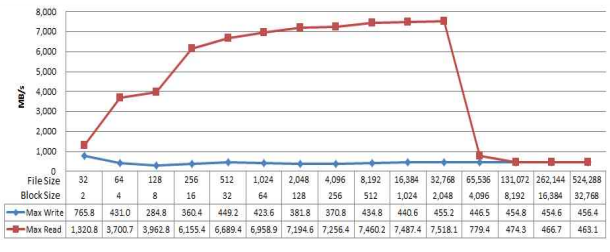
$$FileSize = BlockSize \times NumTasks \times SegmentCount \quad (3)$$

캐시 효과 테스트는 병렬파일시스템의 I/O 성능을 측정하는 IOR 벤치마크를 사용하였다[16]. 이 도구는 MPIIO, POSIX 등의 인터페이스를 사용하며, 수행시 액세스 패턴을 설정하여 병렬파일시스템의 성능을 측정하게 된다. MPI를 이용하여 병렬파일시스템이 마운트 된 클러스터 구성에서 읽기(read), 쓰기(write) 성능을 측정하게 된다. 수행하는 프로세스가 한개의 파일을 생성(Single Shared File)하는 방식과 각각의 프로세스가 개별 파일을 생성(File Per Process)방식의 두 가지 옵션을 선택하여 수행한다[17].

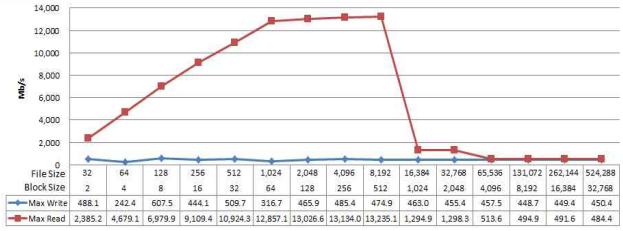
표 2. 블록사이즈 증가에 따른 캐시 효과 테스트
Table 2. Cache effect test with increasing the block size

* Use 16(KNL) & 10(SKL) processes and repeat with changing the Block Size(MB)

```
for bs in 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768
do
    mpirun -NP IOR.posix -a POSIX -b ${bs}m –o
    /PFS/iorDatat-$JOB_ID -t 2m –v –w –r
done
```



(a) Cache effect using DDR4(98207 MB)

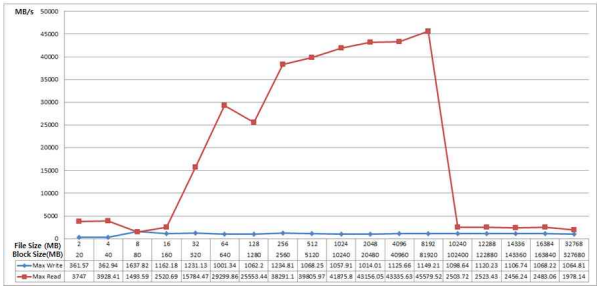


File Size	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768	65,536	131,072	262,144	524,288
Block Size	2	4	8	16	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,768
Max Write	488.1	242.4	607.5	444.1	509.7	316.7	465.9	485.4	474.9	463.0	455.4	457.5	448.7	449.4	450.4
Max Read	2,385.2	4,679.1	6,979.9	9,109.4	10,924.3	12,857.1	13,026.6	13,134.0	13,235.1	1,294.9	1,298.3	513.6	494.9	491.6	484.4

(b) Cache effect using MCDRAM(16384 MB)

그림 10. KNL 노드에서 블록사이즈 증가에 따른 캐시 성능
Fig 10. Cache Performance with the Block Size Increasing on the KNL node

KNL 노드의 경우 메모리 성능 테스트와 마찬가지로 단일노드에서 MCDRAM(16GB)과 DDR4(96GB)를 나누어 테스트 한다. 노드당 16개의 프로세스를 사용하였고 블록사이즈(BlockSize)는 2MB~32GB, 파일사이즈(FileSize)는 32MB~512GB 까지 증가시키며 테스트를 수행하였다. (그림 10)의 (a)는 DDR4를 바인드하여 테스트한 경우로 파일 크기가 약 64GB에서 (b) MCDRAM은 약 16GB에서 캐시 효과로 인한 성능이 급격히 저하된다.



File Size (MB)	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	10240	12288	14336	16384	32768
Block Size(MB)	20	40	80	160	320	640	1280	2560	5120	10240	20480	40960	81920	102400	122880	143360	163840	327680
Max Write	361.57	362.94	1637.82	1162.18	1231.13	1001.34	1062.2	1234.81	1068.25	1057.91	1014.01	1125.66	1149.21	1098.64	1120.23	1106.74	1068.22	1064.81
Max Read	3747	3928.41	1499.59	2520.69	15784.47	29299.86	25553.44	38291.1	39805.97	41875.8	43156.05	43385.63	45579.52	2503.72	2523.43	2456.24	2483.06	1978.14

그림 11. SKL 노드에서 블록사이즈 증가에 따른 캐시 성능
Fig 11. Cache Performance with the Block Size Increasing on the SKN node

SKL 노드의 경우 20개의 코어로 구성된 2개의 CPU와 192GB의 DDR4 메모리가 장착되어있다. 노드당 10개의 프로세스를 사용하였고 블록사이즈(BlockSize)는 2MB~32GB, 파일사이즈(FileSize)는 20MB~320GB 까지 증가시키며 테스트를 수행하였다. 하나의 CPU에 16GB 메모리가 6채널로 96GB가 할당되어 있으며 96GB 이상의 경우 UPI(Intel Ultra Path Interconnect)를 통해 상대 CPU의 메모리에서 읽어야 하므로 성능이 낮아진다. 이 실험에서는 10개의 코어, 즉 1개의 CPU 내의 코어들을 사용하였기 때문에 (그림 11)과 같이 파일크기가 약 100GB 미만에서 캐시 효과로 인한 성능이 급격히 저하됨을 볼 수 있다.

Ⅴ. 결 론

병렬 컴퓨팅은 대규모 문제를 해결하기 위한 적합한 솔루션이다. 병렬 컴퓨팅 수행을 위해 대부분 계산노드를 고속의 네트워크로 연결하여 계산 자원을 엮는 클러스터 환경을 구축한다.

본 논문은 대형 HPC 클러스터를 구성하는 x86 기반의 멀티코어(multi-core) CPU와 매니코어(many-core) 아키텍처 특징을 분석하고 벤치마크 도구를 사용하여 다양한 시스템 구성 요소의 성능을 확인했다. Intel Xeon Phi 제품군의 2세대 버전인 KNL 노드는 호스트타입의 프로세서로 68개의 코어가 집적된 매니코어(many-core) 아키텍처이다. 실험을 통해서 확인했듯이 여러 타일내의 L2 캐시 일관성(Cache Coherence)을 보장하기 위한 성능감소가 있다. 이를 위해 NUMA로 클러스터링 모드를 제공한다. KNL 노드는 고속의 온칩메모리인 16GB MCDRAM을 장착하고 있어 상대적으로 용량이 큰 L3 캐시로 사용할 수 있다. 본 연구에서는 MCDRAM과 DDR4의 메모리 성능을 각각 확인하였고 이를 고려하여 어플리케이션의 수행 환경을 설정한다면 최적의 성능을 얻을 수 있다. SKL 노드는 KNL 노드와 이론 연산 성능은 비슷하지만 실측 연산 성능, 메모리 성능, 캐시 효과에서 더 높은 성능 수치를 얻었다. 앞서 언급한 KNL노드가 가지는 캐시 메모리 버스와 같은 제한 자원에 오버헤드로 기인된다. 그러나 시장 가격이 SKL 노드에 비해 저렴하기 때문에 보다 확장성 있는 클러스터 구성이 가능하다.

이렇게 시스템의 성능 분석을 통해 어플리케이션이 수행되는 계산 자원의 특성을 파악할 수 있고 이를 바탕으로 확장성 있는 병렬연산을 지원할 수 있다. 또한 시스템의 성능 이슈나 장애가 발생하는 경우 요소별 벤치마크 성능 자료를 근거로 문제의 원인을 신속하게 파악하고 대처할 수 있다.

감사의 글

이 논문은 2018년도 부산교육대학교 교내 연구 과제로 지원을 받아 수행된 연구임.

참고문헌

[1] Rosales, Carlos, et al, “A comparative study of application performance and scalability on the Intel Knights Landing processor,” in *International Conference on High Performance Computing. Springer, Cham,* pp. 307-318, 2016.

[2] Y. JunWeon, S. Ui-Sung, “Build the Teaching Practice System based on Cloud Computing for Stabilization through Performance Evaluation”, *The Journal of Digital Contents Society*, Vol. 15, No. 5, pp. 595-602, 2014.

[3] Dongarra JJ, Meuer HW, Strohmaier E, Top500 supercomputer sites. URL http://www.top500.org/ (updated every 6 months), 2018.

[4] Diaz, J., Munoz-Caro, C., & Nino, A. “A survey of parallel programming models and tools in the multi and many-core era,” in *IEEE Transactions on parallel and distributed systems*, 23(8), pp. 1369-1386, 2012.

[5] Kumashikar, M. K, Bendi, S. G, Nimmagadda, S Deka, A. J, Agarwal, A, “14nm Broadwell Xeon® processor family: Design methodologies and optimizations,” in *Solid-State Circuits Conference (A-SSCC)*, pp. 17-20, 2017.

[6] Doweck, J, Kao, W. F, Lu, A. K. Y, Mandelblat, J, Rahatekar, A, Rappoport, L, Yoaz, A, “Inside 6th-generation Intel Core: new microarchitecture code-named Skylake,” *IEEE Micro*, Vol 2, pp. 52-62, 2017.

[7] Li S, Raman K, Sasanka R., “Enhancing application performance using heterogeneous memory architectures on a many-core platform,” in *High Performance Computing & Simulation (HPCS)*, pp. 1035-1042, 2016.

[8] NASA High-End Computing Capability(HECC) [Internet]. Available: https://www.nas.nasa.gov/hecc/

[9] Tanabe, N., & Endo, T., “Exhaustive evaluation of memory-latency sensitivity on manycore processors with large cache,” in *Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications*, pp. 27-34, 2018.

[10] Petitet A, “HPL-a portable implementation of the high-performance Linpack benchmark for distributed-memory computers,” 2004. Available: http://www.netlib.org/benchmark/hpl/.

[11] Gropp W, Lusk E, Skjellum, A, Using MPI: portable parallel programming with the message-passing interface 1, MIT press, 1999.

[12] Whaley RC, ATLAS (Automatically Tuned Linear Algebra Software). Encyclopedia of Parallel Computing: pp. 95−101, 2011, Online: https://doi.org/10.1007/978-0-387-09766-4_85.

[13] John D, McCalpin D STREAM: Sustainable memory bandwidth in high performance computers, 2015. Online: http://www.cs.virginia.edu/stream.

[14] Sodani A, “Knights Landing (KNL): 2nd generation Intel Xeon Phi processor,” in *Hot Chips 27 Symposium (HCS)*, pp. 1-24, 2015.

[15] Shan H, Shalf J, “Using IOR to analyze the I/O performance for HPC platforms”, 2007.

[16] Loewe W, McLarty T, Morrone C, IOR benchmark, 2012. Online: https://sourceforge.net/projects/ior-sio.

[17] Devendran D, Byna S, Dong B, Van Straalen B, Johansen H, Keen, N, Samatova NF, “Collective I/O Optimizations for Adaptive Mesh Refinement Data Writes on Lustre File System”, 2016.

윤준원(JunWeon Yoon)

2002년～2004년 : 고려대학교 대학원 컴퓨터학과(이학석사)
2010년～2018년 : 고려대학교 대학원 컴퓨터학과(공학박사)
2005년~현 재 : KISTI 국가슈퍼컴퓨팅본부 선임연구원

※관심분야: 분산컴퓨팅, 결함포용시스템, 슈퍼컴퓨팅, 병렬파일시스템, 배치스케줄링, 벤치마크(BMT)

송의성(Ui-Sung Song)

1991년～1997년 : 고려대학교 컴퓨터학과(학사)
1998년～1999년: 고려대학교 대학원 컴퓨터학과(이학석사)
2000년～2005년: 고려대학교 대학원 컴퓨터학과(이학박사)
2006년～현 재: 부산교육대학교 컴퓨터교육과 교수

※관심분야: 컴퓨터교육, 교육용로봇교육, 컴퓨터네트워크, 스마트러닝