JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 8, pp. 1-11, August 2021
https://doi.org/10.9708/jksci.2021.26.08.001

Exploiting Hardware Events to Reduce Energy Consumption of HPC Systems

Yongho Lee*, Osang Kwon*, Kwangeun Byeon*, Yongjun Kim*, Seokin Hong**

*Student, Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
*Student, Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
*Student, Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
*Student, Dept. of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
**Professor, Dept. of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon, Korea

[Abstract]

This paper proposes a novel mechanism called Event-driven Uncore Frequency Scaler (eUFS) to improve the energy efficiency of the HPC systems. UFS exploits the hardware events such as LAPI (Last-level Cache Accesses Per Instructions) and CPI (Clock Cycles Per Instruction) to dynamically adjusts the uncore frequency. Hardware events are collected at a reference time period, and the target uncore frequency is determined using the collected event and the previous uncore frequency. Experiments with the NPB benchmarks demonstrate that the eUFS reduces the energy consumption by 6% on average for class C and D NPB benchmarks while it only increases the execution time by 2% on average.

▸ **Key words**: HPC, DFS (Dynamic Frequency Scaling), Uncore, Power Management, Low Power

[요 약]

본 논문에서는 HPC 시스템의 에너지 효율을 향상시키기 위해 Event-driven Uncore Frequency Scaler (eUFS)라는 새로운 전력관리 메커니즘을 제안한다. eUFS는 LAPI (LLC accesses Per Instructions) 및 CPI (Clock Cycles Per Instruction)와 같은 하드웨어 이벤트를 활용하여 언코어 주파수를 동적으로 조정한다. 기준 시간을 주기로 해당 하드웨어 이벤트를 취합하고, 취합한 이벤트와 이전 언코어 주파수를 이용해 목표 언코어 주파수를 결정한다. NPB 벤치마크를 사용한 실험을 통해 본 논문에서 제안하는 UFS 메커니즘은 C/D class NPB 벤치마크에 대해 평균 6%의 에너지 소비를 감소시키는 것으로 확인되었고 실행시간 증가는 평균 2% 수준인 것으로 확인되었다.

▸ **주제어**: 고성능 컴퓨팅 시스템, 동적 주파수 제어, 언코어, 전력관리, 저전력

• First Author: Yongho Lee, Corresponding Author: Seokin Hong
*Yongho Lee (jhyn205@g.skku.edu), Dept. of Electrical and Computer Engineering, Sungkyunkwan University
*Osang Kwon (osang915@g.skku.edu), Dept. of Electrical and Computer Engineering, Sungkyunkwan University
*Kwangeun Byeon (kebyun@g.skku.edu), Dept. of Electrical and Computer Engineering, Sungkyunkwan University
*Yongjun Kim (yongjunkim@g.skku.edu), Dept. of Electrical and Computer Engineering, Sungkyunkwan University
**Seokin Hong (seokin@skku.edu), Dept. of Semiconductor Systems Engineering, Sungkyunkwan University
• Received: 2021. 07. 28, Revised: 2021. 08. 20, Accepted: 2021. 08. 23.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

최근 고성능 컴퓨팅 시스템의 성능은 빠른 속도로 향상되고 있다. 하지만, 고성능 컴퓨팅을 위해 전력 소모량도 함께 증가하고 있고, 고성능 컴퓨팅 시스템에 제공되는 전력량은 한정되어있기 때문에 시스템의 최대 성능을 사용하지 못하거나 전력량을 초과하여 사용하게 된다. 고성능 컴퓨팅 시스템의 전력 소모량을 줄이기 위해 CPU의 클럭 주파수를 동적으로 조절하는 방법이 일반적으로 사용된다. 하지만 이러한 방법은 CPU의 클럭 주파수를 낮추어야 하므로 CPU 성능을 감소시키는 문제가 있다.

최근 인텔 CPU에서는 하드웨어 구성요소를 코어(Core, ALU, FPU, L1/L2 Cache)와 언코어(Uncore, Last-Level Cache, IMC, QPI)로 구분하여 서로 다른 클럭 도메인을 사용하고 있다. 이를 통해 코어와 언코어의 클럭 주파수를 개별적으로 제어할 수 있다. 언코어의 클럭 주파수는 BIOS 및 운영체제의 설정에 따라 최대, 최소, 또는 자동 (UFS, Uncore frequency scaling)으로 선택된다 [1]. UFS 기능은 CPU의 클럭 주파수와 비례적으로 언코어 주파수를 자동으로 선택한다. 하지만 자동으로 선택했을 때 코어 및 언코어에 대한 즉각적인 반응을 하지 못해 속도 저하 및 전력 낭비될 가능성이 있다 [17].

본 논문에서는 하드웨어 이벤트 정보를 바탕으로 언코어 부하 수준을 예측하여 언코어의 클럭 주파수를 동적으로 조절하는 Event-driven Uncore Frequency Scaler (eUFS) 기술을 제안한다. 언코어의 클럭 주파수가 CPU의 성능에 미치는 영향은 실행되는 어플리케이션의 특성에 따라 매우 다르다. 또한, 어플리케이션의 각 실행 페이즈(Phase)에 따라서도 언코어 주파수에 대한 CPU 성능의 민감도에 차이가 있다. 따라서 eUFS는 하드웨어 이벤트 정보를 바탕으로 언코어 부하가 높은 어플리케이션의 실행 단계를 예측하고 부하의 수준에 따라 언코어의 클럭 주파수를 단계적으로 조절한다.

NPB 벤치마크를 사용한 실험 결과를 통해 eUFS 기술이 성능 저하를 최소로 하면서도 컴퓨팅 시스템의 전력 소모를 평균 8% 감소시키고 에너지 소모는 평균 6% 감소시킬 수 있는 것으로 평가되었다.

2장에서는 언코어에 대한 기본적인 설명과 언코어 주파수에 따른 실행 시간과 소모 전력의 상관 관계를 분석한다. 3장에서는 제안하는 Event-driven Uncore Frequency Scaler에 대하여 설명한다.

4장에서는 NPB 벤치마크를 사용한 실험 결과를 통해 Event-driven Uncore Frequency Scaler의 효율성에 대해 논한다.

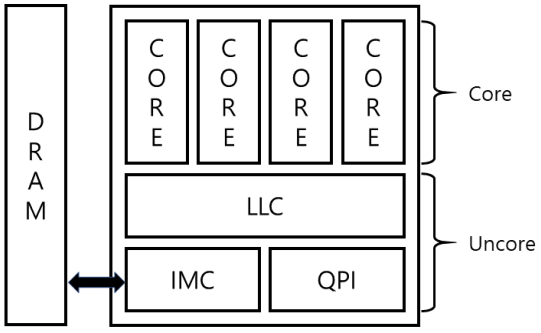


Fig. 1 CPU Structure

II. Background

1. Uncore

Figure 1은 전형적인 서버 프로세서의 구조를 보여준다. 프로세서는 크게 코어 부분과 언코어 부분으로 구성되어 있다. Core에는 계산을 담당하는 ALU, FPU 등과 L1, L2 Cache 등으로 구성되어 있다. 반면에 언코어 부분은 LLC(Last-Level Cache)와 함께 QPI(Quick path interconnect) 컨트롤러, IMC(Memory Controller)로 구성된다 [13].

이전 인텔 프로세서에는 고정된 언코어 주파수 또는 코어-언코어가 공통된 클럭 주파수를 사용했다. 그러나 새로운 세대의 프로세서에서 코어의 수와 LLC의 크기가 증가하고, 더 복잡한 형태의 메모리 컨트롤러가 사용됨에 따라 언코어 부분이 전체 CPU 칩 면적의 30% 이상을 차지하고 있으며 [17] 그에 따라서 전력 소모량도 점차 증가하고 있다 [18, 19]. 따라서 인텔 하스웰 마이크로아키텍처를 시작으로 코어, 언코어 주파수 도메인이 분리되어 독립적으로 설정할 수 있다. 언코어의 전력 소모량을 감소시키기 위해 인텔의 CPU에는 UFS 기술이 탑재되어 있다. 하지만 UFS 기술은 어플리케이션의 단계 변화에 대해 즉각적인 반응을 하지 못해 잘못된 코어-언코어 주파수를 설정하여 속도 저하 및 전력 낭비될 가능성이 있다. 언코어 주파수는 MSR(Model Specific Register)를 통해 설정할 수 있으며 msr-safe 모듈을 사용해 설정할 수 있다 [20].

2. Observation

어플리케이션의 특성 (데이터 전송 대비 연산량 등)에 따라 언코어 주파수가 성능에 미치는 영향이 매우 다르다. 하나의 어플리케이션 실행은 여러 단계의 실행 페이즈로 구분할 수 있으며 각 페이즈마다 필요로 하는 하드웨어 자원

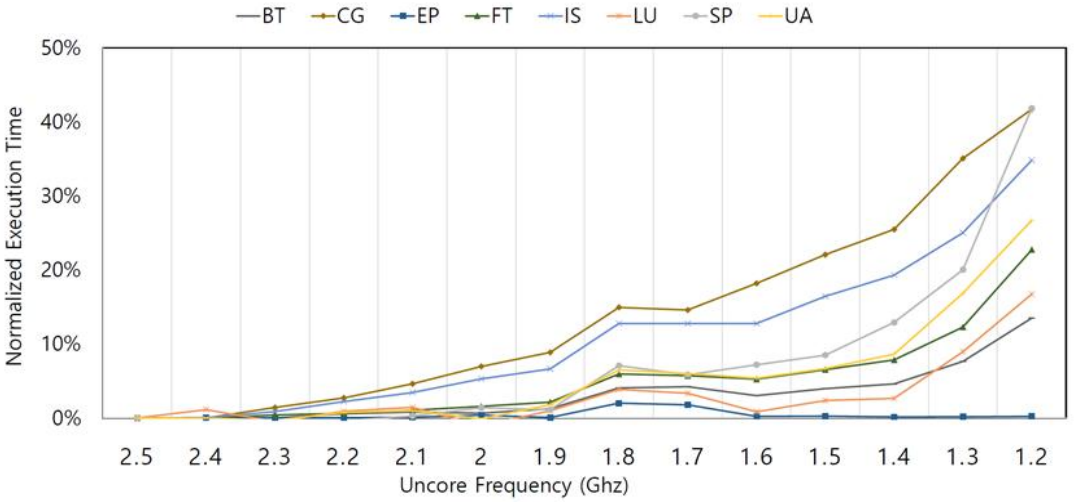


Fig. 2. Impact of Uncore Frequency on Execution time

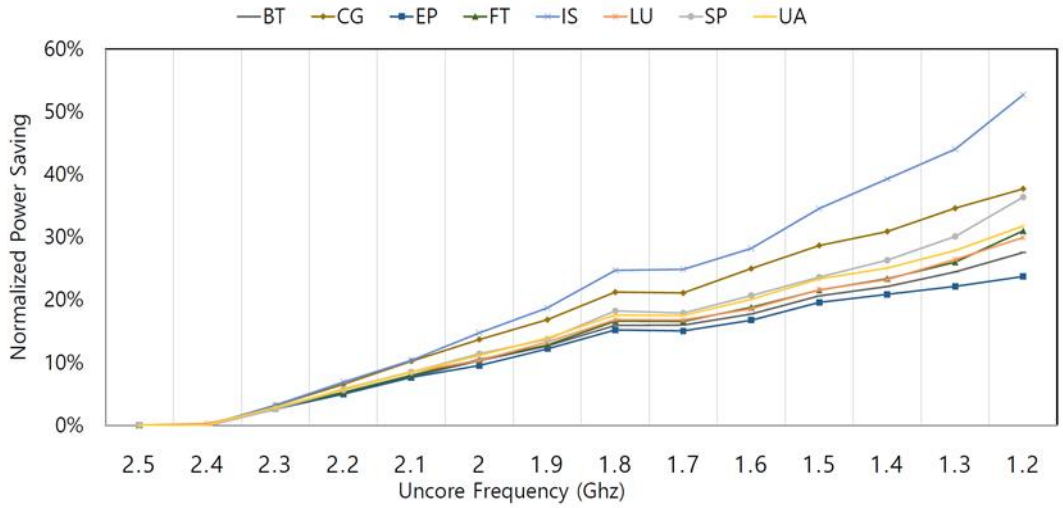


Fig. 3. Impact of Uncore Frequency on Power saving

에 차이가 있기 때문에 언코어 주파수에 따른 민감도에도 차이가 있다. 본 절에서는 언코어 주파수가 어플리케이션의 실행시간 및 파워 소모에 미치는 영향성에 대해 논한다.

Table 1. Experiment Environment

CPU	2x Intel® Xeon® Gold 5215 CPU @ 2.50GHz
DRAM	4x 128G Intel® Optane™ Persistent Memory
OS	CentOS 7.8/ Linux 5.4.83
Hyper-Threading	On
Turbo boost	Off
CPU driver	acpi-cpufreq
Governor	Performance
OMP_NUM_THREADS	16

2.1. Observation Environment

Table 1은 본 연구에서 사용한 컴퓨팅 시스템 구성을 보여준다. 2개의 Intel Xeon Gold 5215 (Cascade 마이크로아키텍처)와 512GB의 Intel® Optane™ Persistent Memory를 탑재했다. Optane 메모리는 메모리 모드로 동작하며 [16] 128GB DDR4 RAM(8x 16G DDR4 ECC DIMM@ 2666 Mhz) 은 캐시로 동작한다. CPU의 터보 부스트는 사용하지 않으며, 하이퍼 스레딩은 사용한다. CPU 드라이버로 acpi-cpufreq를 사용하며, CPU 가버너는 Performance를 사용한다. 언코어 주파수는 1.0GHz ~ 2.5GHz로 설정될 수 있다.

실험에 사용할 워크로드로써 NAS Parallel Benchmark (NPB) [12]를 사용하였다. NPB-3.4.1 OpenMP버전의 'BT', 'CG', 'EP', 'FT', 'LU', 'MG', 'SP', 'UA'를 사용하였으며 C와 D Class 워크로드에 대해 스레드는 16개를 설정

하여 실행하였다. 모든 NPB 벤치마크는 'gcc-8.3.1'으로 컴파일되었고, 'Linux-5.4.83' 기반 'Centos 7.8'에서 실행한다. 각 워크로드마다 10회씩 실행되었으며, 최소, 최대 실행 시간을 제외한 8회 실행의 평균을 사용했다.

2.2. Uncore Frequency vs Performance

Figure 2 그래프는 NPB 벤치마크별로 언코어 주파수에 따른 어플리케이션의 성능 변화를 정규화된 실행시간으로 보여준다. 언코어 주파수(2.5GHz ~ 1.2GHz)를 달리하였을 때의 실행시간을 최대 언코어 주파수(2.5GHz)에서의 실행시간(Execution time)을 기준으로 정규화하였다.

대부분의 NPB 벤치마크는 언코어 주파수가 낮을수록 성능이 감소한다. 메모리 집약적(Memory intensive)인 'CG' 벤치마크 [21]에서는 언코어 주파수가 낮을수록 더 높은 성능 저하가 나타나며, 최소 언코어 주파수에서의 성능이 최대 언코어 주파수와 비교했을 때 42% 낮은 것으로 확인되었다. 이에 반해 컴퓨팅 집약적(Compute Intensive)인 벤치마크인 'EP' [22]에서는 최소 언코어 주파수에서의 성능과 최대 언코어 주파수에서의 성능 차이가 매우 적은 것으로 확인되었다.

2.3 Uncore Frequency vs Power Saving

Figure 3 그래프는 NPB 벤치마크별로 언코어 주파수에 따른 시스템 전력 소모량 변화를 보여준다. 각 언코어 주파수(2.5GHz ~ 1.2GHz)에 따른 전력 소모량을 최대 언코어 주파수(2.5GHz)에서의 전력소모량을 기준으로 정규화하였다.

모든 NPB 벤치마크는 언코어 주파수가 낮을수록 전력 소모량이 낮음을 알 수 있다. 'IS' 벤치마크는 최소 언코어 주파수일 때 전력 소모량이 가장 낮았고, 최대 언코어 주파수 대비 약 53% 전력 소모량이 낮은 것으로 확인 되었다. 또한 'EP' 벤치마크는 최소 언코어 주파수일 때 약 24% 전력소모가 감소되었으며, 실험에 사용된 벤치마크 중 Event-driven Uncore Frequency Scaler 적용에 따른 전력 소모 절감 효과가 가장 낮은 것으로 확인되었다.

2.4 Possibility of Power Saving with Uncore Frequency Scaling

절 2.2의 결과를 통해 언코어 주파수에 대한 민감도(Sensitivity)는 NPB 벤치마크별 편차가 있으며, 언코어 주파수를 변경하더라도 성능에 영향을 받지 않는 워크로드가 존재한다는 것을 확인할 수 있다. 절 2.3의 결과를 통해 코어 주파수뿐만 아니라 언코어 주파수 scaling을 사용하여 전력 소모량 절감이 가능하다는 것을 확인할 수 있다. 즉, 언코어 주파수에 대해 민감도가 낮은 벤치마크나 민감도가 낮은 페이즈에 대해 낮은 언코어 주파수를 적용한다면 성능에 대한 영향을 최소화하면서도 전력 소모량을 감소시킬 수 있음을 알 수 있다.

3. Limitation of Prior Works

Figure 4는 'LU' 벤치마크를 실행했을 때 CPI (Clock Per Instruction), 캐시 접근 횟수 (Cache Reference), DRAM 파워 변화를 보여준다. 언코어 주파수를 동적으로 제어하는 기존 연구에서는 DRAM 파워와 IPC를 사용하여 어플레케이션의 실행 페이즈별 언코어 부하 수준을 예측한다 [2]. 하지만 DRAM 파워는 어플리케이션의 초기 페이즈에 대해서는 언코어의 부하 수준을 예측할 수 있는 정보를 제공해 주지만 실질적인 동작 상태에서는 페이즈에 대해 정확한 정보를 제공해 주지 못한다. 그 이유는 언코어는 메모리 컨트롤러뿐만 아니라 LLC, QPI 등 복합적 요소를 포함하고 있기 때문이다. 실질적인 메모리 접근 빈도가 낮아 메모리 컨트롤러에 대한 부하 수준이 낮더라도 L3 캐시 접근 빈도가 높다면 언코어의 전체적인 부하 수준은 높을 수 있다. 따라서 DRAM 파워 데이터로만 페이즈를 정확히 예측하는 것은 어렵다.

III. The Proposed Scheme

1. Phase Detection with Hardware Events

동적 언코어 주파수 스케일링을 위해서는 어플리케이션의 페이즈를 실시간으로 검출해야 한다. 본 연구에서는 운영체제 수준에서 확인할 수 있는 하드웨어 이벤트 정보를 바탕으로 페이즈를 정확히 검출하는 방법을 제안한다. 페이즈 검출에 사용할 하드웨어 이벤트를 선택하기 위해 Perf tool [15]을 사용하여 다양한 하드웨어 이벤트에 대

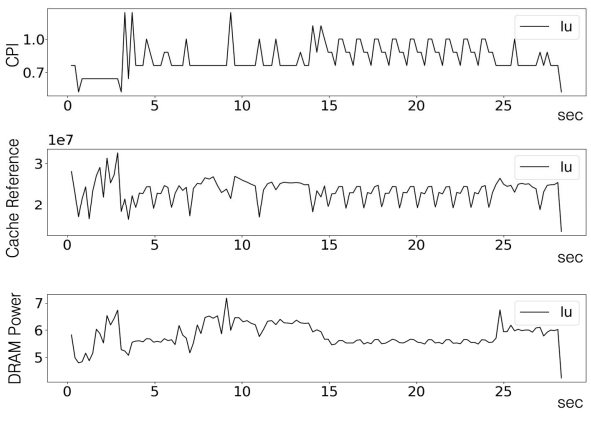
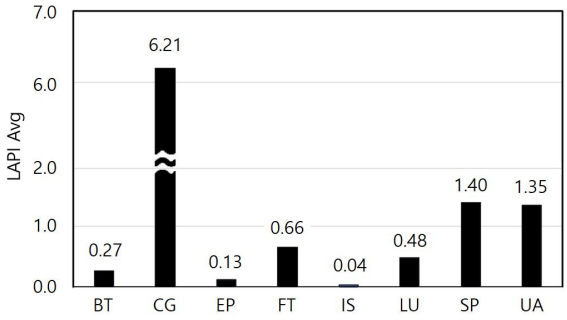


Fig. 4. Events when running 'LU' benchmark

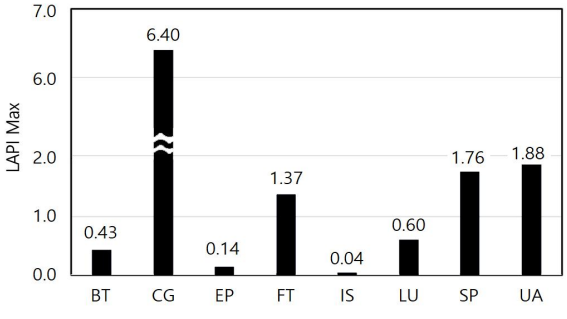
한 프로파일링을 진행하였다. 수집된 이벤트 데이터와 절 2.2와 절 2.3에서 소개한 성능 및 전력 소모량에 대한 언코어 주파수 영향성 결과의 상관관계를 파악하여 언코어 부하 수준과의 연관성이 높은 이벤트를 선정하였다.

첫 번째 이벤트는 CPI(Cycles Per Instruction, 명령어 당 평균 클럭 사이클 수)이다. 언코어 구성요소에 대한 접근 빈도가 높아지면 CPU의 실행 파이프라인이 빈번하게 멈추게 되어 (Pipeline Stall) CPI 수준이 증가하게 된다. 따라서 언코어 부하 수준과 연관도가 높을 수 있다. 하지만 CPI는 언코어 부하 증가 이외에도 다른 요인 (분기 명령어 예측 실패 등)에 의해 증가할 수 있기 때문에 다른 이벤트와 보완적으로 활용되어야 한다.

두 번째 이벤트는 LAPI (LLC Access Per Instructions, 명령어 당 LLC 접근 횟수)이다. LAPI는 페이즈를 보다 정밀하게 검출하기 위해서 활용될 수 있다. 그 이유는 언코어의 주요 구성요소인 LLC와 메모리 컨트롤러에 대한 접근 빈도가 높아질수록 LAPI 수준이 높아지기 때문이다. LAPI는 상대적인 지표가 아닌 절대적인 지표이기 때문에 언코어의 절대적인 부하 수준을 예측하는 데 활용될 수 있다. 앞서 설명한 것과 같이 CPI는 언코어의 부하 수준이 낮은 상황에 대해서도 증가할 수 있기 때문에 LAPI와 같이 사용한다면 어플리케이션의 각 페이즈를 보다 정확하게 검출할 수 있다.



(a) Average LAPI of NPB Benchmarks



(b) Max LAPI of NPB Benchmarks

Fig. 5. LAPI of NPB Benchmarks

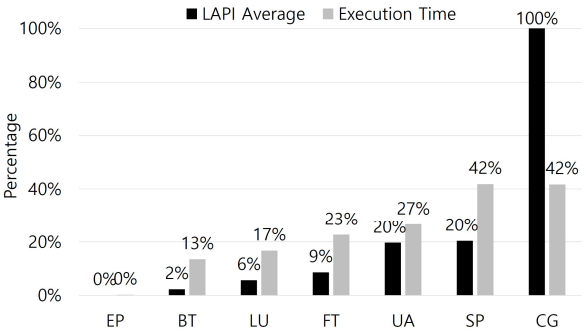


Fig. 6. Relationship between average LAPI and execution time at lowest uncore frequency

Figure 5 그래프는 NPB 벤치마크의 평균, 최대 LAPI를 나타낸다. 메모리 집약적인 벤치마크 중 하나인 'CG' 벤치마크의 평균 LAPI는 6.21%이고 최대 LAPI는 6.40% 수준이다. 이 수치는 다른 벤치마크보다 높은 수준이다. 그에 반해 컴퓨팅 집약적인 벤치마크 중 하나인 'EP' 벤치마크의 평균 LAPI는 0.13%이고 최대 LAPI는 0.14% 수준이다. Figure 6는 LAPI가 높을수록 언코어에 대해 민감도가 높다는 사실을 나타낸다. LAPI_Average는 'EP' 벤치마크의 LAPI를 기준으로 정규화한 각 벤치마크의 LAPI 수치이다. Execution_time은 최소 언코어 주파수에서의 정규화된 실행시간 (Figure 2)를 나타낸다. Figure 6에서 볼 수 있듯이 'EP', 'BT', 'LU', 'FT', 'UA', 'SP', 'CG' 순서로 평균 LAPI 수준이 높은 것을 알 수 있으며 실행시간도 'EP', 'BT', 'LU', 'FT', 'UA', 'SP', 'CG' 순서로 증가한다는 것을 알 수 있다.

2. Uncore Frequency Scaler

Figure 7은 Event-driven uncore frequency scaler (eUFS)의 구조를 나타낸다. eUFS는 어플리케이션이 실행될 때 각 실행 페이즈에서의 CPI와 LAPI를 수집하고 이를 바탕으로 다음 실행 페이즈에서의 언코어 주파수를 결정한다. eUFS는 언코어 주파수를 7단계로 제어한다. 본 연구에서 사용된 컴퓨팅 시스템에서는 언코어 주파수를 최소 1.0GHz에서부터 최대 2.5GHz로 설정할 수 있기 때문

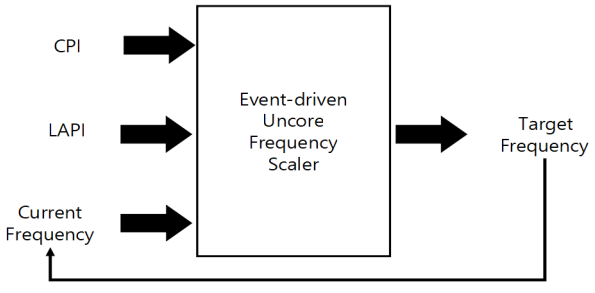


Fig. 7. eUFS's Structure

에 “2.5GHz, 2.3GHz, 2.1GHz, 1.9GHz, 1.7GHz, 1.5GHz, 1.3GHz“로 7단계를 구성했다. 기준 시간(200ms) 주기로 이벤트를 취합하고 목표 언코어 주파수(Target Uncore Frequency)를 결정한다. 언코어 주파수를 조정하기 위해서 Model Specific Register(MSR) 을 사용한다. 본 연구에서 사용한 시스템의 CPU는 인텔 Cascade Lake 마이크로아키텍처로 되어 있기 때문에 0x620H [14:8][6:0] 값을 수정해 목표 언코어 주파수 값을 설정한다 [14].

Algorithm 1. eUFS Algorithm

```
1: loop
2:    current_CPI <- measure_CPI()
3:    current_LAPI <- measure_LAPI()
4:    if current_CPI > past_CPI then
5:       if current_LAPI > 1.1 * past_LAPI and
                current_LAPI >0.8 then
6:          MAX FREQ
7:       else
8:          INCREASE FREQUENCY
9:    else
10:      if current_LAPI > 1.1 * past_LAPI and
                current_LAPI > 0.8 then
11:         INCREASE FREQUENCY
12:      else if current_LAPI < 0.8 * past_LAPI then
13:         DECREASE FREQUENCY
14:      else if current_LAPI < 2 then
15:         DECREASE FREQUENCY
```

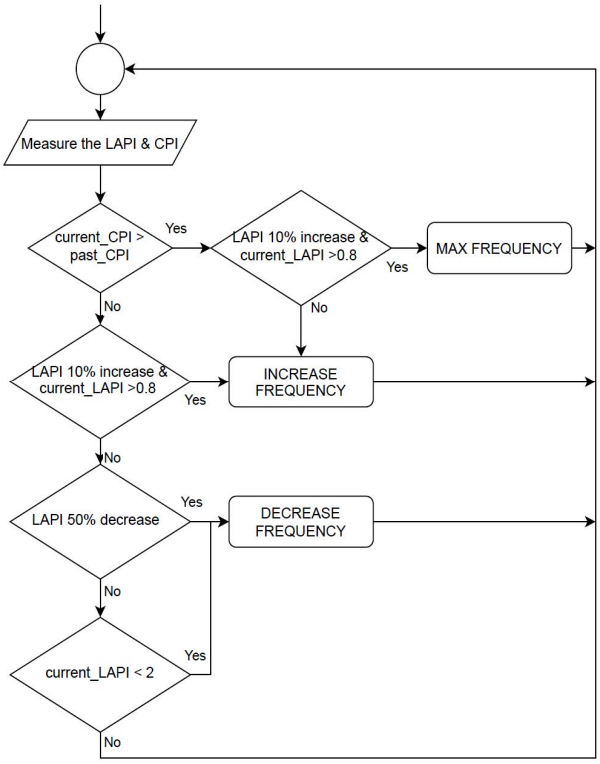


Fig. 8. eUFS flowchart

Algorithm 1과 Figure 8은 eUFS의 전체적인 동작을 설명한다. 현재 페이즈의 CPI와 LAPI를 취합하여 과거 페이즈의 CPI 및 LAPI와 비교한다. 만약 현재 페이즈의 CPI가 이전 페이즈보다 증가했다면 목표 언코어 주파수 Algorithm 1과 Figure 8은 eUFS의 전체적인 동작을 설명한다. 현재 페이즈의 CPI와 LAPI를 취합하여 과거 페이즈의 CPI 및 LAPI와 비교한다. 만약 현재 페이즈의 CPI가 이전 페이즈보다 증가했다면 목표 언코어 주파수를 증가하는 방향으로 선택한다. 이때 LAPI가 이전 페이즈에 비해 10% 증가하고 LAPI의 절대값이 0.8을 초과한다면 목표 언코어 주파수를 최대 주파수로 설정한다. 하지만 위의 조건에 부합하지 않다면 목표 언코어 주파수는 현재의 주파수보다 한 단계 높은 주파수로 설정한다. CPI가 증가하지 않은 상황에서 LAPI가 이전 페이즈에 비해 10% 증가했다면, LAPI의 절대값이 0.8을 초과했을 때 목표 언코어 주파수를 현재의 주파수보다 한 단계 높은 주파수로 설정한다.

만약 위의 조건에서 목표 언코어 주파수를 증가시키지 않았는데 LAPI가 이전 페이즈에 비해 20% 감소하거나 현재 LAPI의 절대값이 2 미만일 때는 목표 언코어 주파수를 현재 주파수보다 한 단계 낮은 주파수로 설정한다.

IV. Experiments

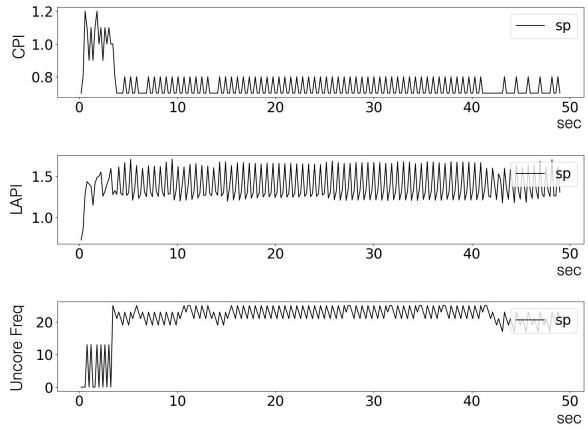
1. Experimental Methodology

Table 1은 본 연구에서 사용한 컴퓨팅 시스템 구성을 보여준다. 하드웨어 및 벤치마크 구성은 Observation 절에서 설명한 것과 동일하다.

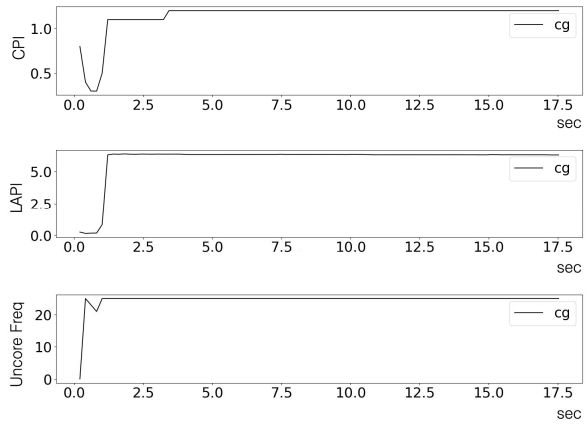
실험에서는 소프트웨어의 성능을 분석하고 이벤트 데이터를 취합하기 위해, Perf tool을 사용했다. 해당 도구는 시스템의 주요 프로파일링 데이터를 수집하여 분석 및 해석을 가능하도록 한다. LAPI를 계산하기 위해 Perf가 제공하는 성능 지표 중 cache-reference, Instructions 를 사용하였다. CPU의 평균 전력 소모량과 전체 에너지 소모량은 Perf가 제공하는 정보 중 energy-pkg를 사용하여 측정하였다.

2. Phase Detection Accuracy

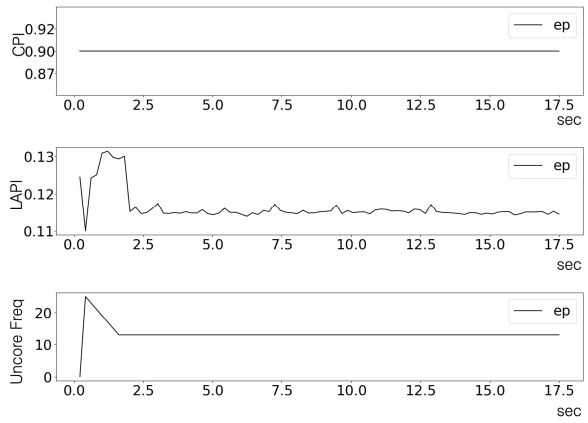
Figure 9 그래프는 논문에서 제안하는 Event-driven Uncore Frequency Scaler(eUFS)를 적용하고 SP, CG, LU 벤치마크를 실행했을 때의 CPI, LAPI, 언코어 주파수를 보여준다.



(a) Uncore Frequency, CPI, and LAPI changes for SP



(b) Uncore Frequency, CPI, and LAPI changes for CG



(c) Uncore Frequency, CPI, and LAPI changes for EP

Fig. 9. Uncore Frequency, CPI, and LAPI changes

'SP' 벤치마크에서는 CPI와 LAPI의 변화로 인해 언코어 주파수가 200ms 간격으로 수정된다. 메모리 집약적 워크로드인 'CG'에서는 실행 초기부터 언코어 주파수가 최대 주파수로 설정되어 종료될 때까지 유지된다. 그에 반해 컴퓨팅 집약적 워크로드인 'EP'에서는 언코어 주파수가 실행 초기에 바로 최저 수준으로 설정되어 종료될 때까지 최저 수준을 유지하는 것을 확인할 수 있다.

3. Performance

Figure 10과 Figure 11에서 상단 그래프는 C, D class에 대해 각 NPB 벤치마크에 대한 실행시간 증가량을 보여준다. 베이스라인에서는 CPU 가버너를 Performance 모드로 설정하여 언코어 주파수가 최고 수준으로 설정되도록 하였다.

NPB 벤치마크를 C class로 설정하였을 때, eUFS는 모든 벤치마크에 대해 평균 2% 실행시간이 증가하는 것으로 확인되었다. 'BT' 벤치마크에서는 eUFS 적용으로 인해 평균 6% 실행시간이 증가 되었으며 가장 좋지 않은 성능을 보여준다. 'LU' 벤치마크의 실행시간 증가율은 평균 5% 수준이고 'FT', 'IS', 'UA' 벤치마크에 대해서는 평균 2% 실행시간이 증가하였다. 'CG', 'EP', 'SP'에 대해서는 eUFS를 적용하더라도 베이스라인과 동일한 성능을 보임을 알 수 있었다.

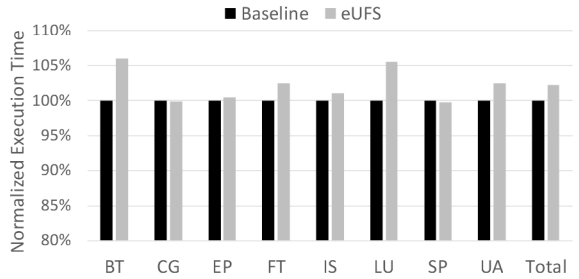
NPB 벤치마크를 D class로 설정하였을 때도 모든 벤치마크에 대한 실행시간 증가율이 평균 2% 수준인 것으로 확인되었다. 'BT' 벤치마크에서 평균 5% 실행시간이 증가하였고 'FT', 'LU' 벤치마크에 대해 평균 3% 실행시간이 증가되는 것을 확인할 수 있었다. 나머지 벤치마크에 대해서는 C class와 마찬가지로 eUFS를 적용했을 때의 성능이 베이스라인과 동일하였다.

위의 결과를 통해 벤치마크에 따라 eUFS의 성능 오버헤드는 최대 5%, 평균 2%, 최소 0% 수준인 것으로 확인되었다.

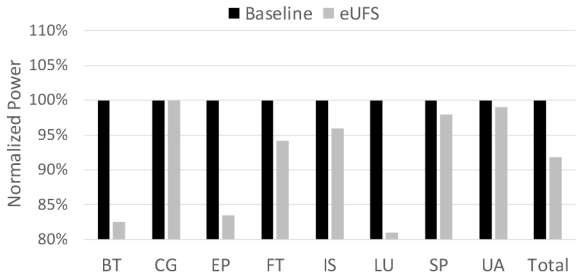
4. Power Consumption

Figure 10과 Figure 11의 중단 그래프는 각 NPB 벤치마크에 대한 CPU Package의 평균 전력 소모량을 나타낸다. NPB 벤치마크를 C class로 설정했을 때, eUFS는 소비 전력을 평균 8% 감소하는 것으로 확인되었다. 'LU' 벤치마크에서 대해서는 평균 19% 소모 전력 감소가 있으며 가장 높은 소모 전력 감소율을 보인다. 다음으로 'BT', 'EP' 벤치마크에 대해 높은 수준의 전력 소모 감소 효과를 보였으며, 평균 17% 전력소모를 감소시킬 수 있는 것으로 평가되었다. 'FT', 'SP', 'UA' 벤치마크에 대해서는 1 ~ 6% 수준의 전력 소모 감소효과가 있었으며, 메모리 집약적 벤치마크인 'CG'에 대해서는 전력 소모 감소효과가 없는 것으로 평가되었다.

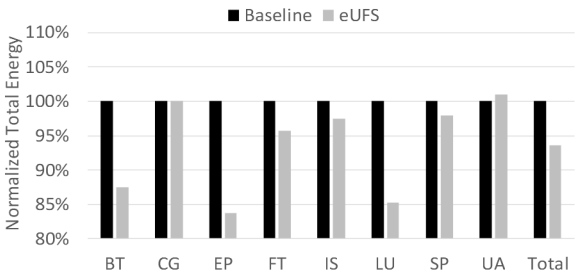
NPB 벤치마크를 D class로 설정했을 때도 유사한 양상을 보였다. 모든 벤치마크에 대해 eUFS는 평균 7% 전력 소모를 감소시켰고, 컴퓨팅 집약적인 벤치마크인 'EP'에 대해서는 평균 18% 전력 감소를 감소시켰다. 'BT', 'LU' 벤치마크에 대해서는 평균 15%, 10%의 전력 감소 효과가 있었다. Class C와 마찬가지로 메모리 집약적 벤치마크인 'CG'에 대해서는 전력소모 감소효과가 없었다.



(a) Execution time in Class C workload

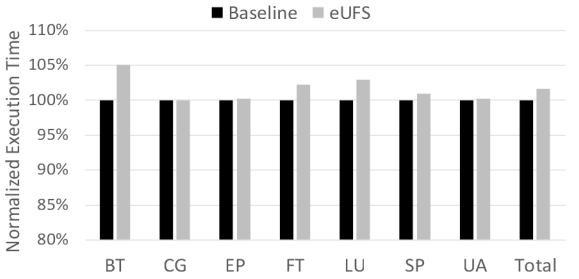


(b) Power in Class C workload

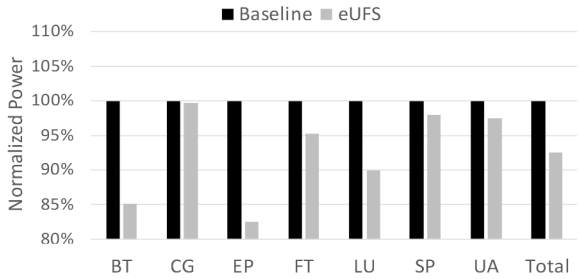


(c) Total energy consumption in Class C workload

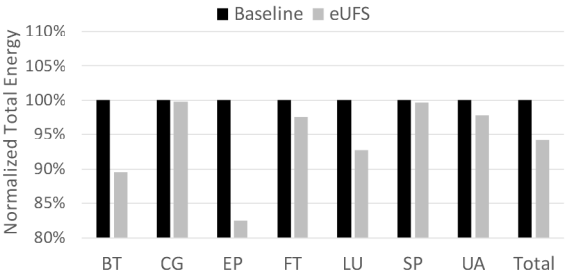
Fig. 10. Execution time, Power, Total energy consumption for Class C workload



(a) Execution time in Class D workload



(b) Power in Class D workload



(c) Total energy consumption in Class D workload

Fig. 11. Execution time, Power, Total energy consumption for Class D workload

5. Total Energy

Figure 10과 Figure 11의 하단 그래프는 각 벤치마크에 대한 총 에너지 소모량을 나타낸다. NPB 벤치마크에 C Class를 적용했을 때 모든 벤치마크에 대해 평균 6% 총 에너지 소모량이 감소되었다. ‘EP’ 벤치마크에 대해서는 평균 17% 에너지 소모가 감소되었으며 가장 높은 에너지 감소율을 보인다. 다음으로 ‘LU’, ‘BT’ 벤치마크에 대해서는 eUFS를 적용했을 때 평균 15%, 13%의 에너지 소모가 감소되었다. 메모리 집약적인 벤치마크인 ‘CG’에 대해서는 절 4에서 논했던 것과 같이 전력 소모량 감소효과가 없었다. 따라서 eUFS를 적용했을 때 전체 에너지 소모 또한 감소하지 않았다.

NPB 벤치마크에 D Class를 적용했을 때 모든 벤치마크 결과에 대해 평균 6% 총 에너지 감소효과가 있었다. ‘EP’ 벤치마크에 대해 평균 18% 에너지 소모가 감소되었으며 ‘BT’, ‘LU’ 벤치마크에 대해서는 각각 평균 10%, 7% 에너지 감소 효과를 보였다.

V. Related Works

최근 마이크로프로세서에는 전력 소모량을 제한할 수 있는 다양한 기능이 포함된다. 직접적인 전원관리 요소를 포함하거나 DVFS (Dynamic Voltage Frequency Scaling)와 같은 간접적으로 전력을 변화시킬 수 있는 기능을 제공한다. 이때의 DVFS는 주로 코어 주파수를 결정하는데 사용된다 [3]. DVFS를 활용하여 CPU idle time에 코어 주파수를 줄이기도 했다 [6]. 어플리케이션이 작동하는 과정에 사용한 전력 소모량을 기반으로 DVFS와 DCT(Dynamic Concurrency throttling)을 주기적으로 수행하여 어플리케이션의 임계 경로를 빠르게 수행할 수 있도록 돕는 런타임 시스템이 있다 [5]. 하지만 전력 소모량을 관리할 때 DVFS 기반의 해결책은 언코어의 전력 소비량은 고려하지 않았다 [4]. 코어가 발전할수록 언코어의 역할은 중요하고, 단순히 코어뿐 아닌 언코어의 전력도 효

율적으로 관리할 필요가 있다. [9, 10]는 DVFS와 UFS를 사용한 에너지 절약에 대한 연구를 제시한다. 코어와 언코어 주파수 모두를 조절하지만 특정한 어플리케이션을 대상으로 하는 한계점이 존재한다.

[7]은 최적의 언코어 주파수를 결정하는 기계 학습 기술을 제안한다. 또한 어플리케이션의 특성이 전력 소모를 줄이는데 영향을 미친다는 결론을 보여준다. [7]과 비슷한 방법으로 최적의 언코어 전력 관리를 적용하기 위해 인공 신경망을 설계하는 연구도 존재한다 [8]. [2]는 본 연구와 유사한 방식으로 접근했다. 언코어의 전력을 효율적으로 관리를 위해 성능 저하 없이 언코어 주파수를 낮게 결정하여 전력을 절약하는 런타임 시스템인 UPSCavenger을 제안한다. DRAM Power와 IPC(instructions per cycle) 정보를 사용하여 어플리케이션의 컴퓨팅 집약적, 메모리 집약적을 판단하는 알고리즘을 설계했다. 어플리케이션의 페이즈를 구분하고 각 페이즈마다 언코어 주파수를 결정할 수 있도록 했다. 직접 설계한 UPSCavenger를 각 소켓에 할당하고, 에이전트를 통해 DRAM power와 IPC를 취합하고 결과를 기반으로 언코어 주파수를 결정하는 과정을 반복한다. [11]은 [2]의 아이디어와 유사하지만 다른 하드웨어 이벤트를 통해 접근한다. 또한, Measurement, Regulator 두가지 모듈을 구분하고 있다는 특징을 가지며, FLOP/s, memory bandwidth를 포함한 다양한 이벤트를 수집하고 어플리케이션 페이즈를 구분했다.

VI. Conclusions

최근 컴퓨팅 성능의 향상으로 인한 전력 소모가 급격히 증가하고 있다. 따라서 최소 성능 저하를 가지며 소모 전력을 감소시키기 위해 많은 연구와 개발이 진행되고 있다. 본 논문에서는 LAPI(LLC Access Per Instructions)와 CPI 정보를 바탕으로 언코어의 부하가 높은 어플리케이션 페이즈를 탐지하고 이를 바탕으로 언코어 주파수를 조정하는 Event-driven Uncore Frequency Scaler (eUFS)를 제안했다. NAS Parallel Benchmark를 사용한 실험 평가를 통해 eUFS가 평균 8%의 전력 소모를 감소시키고 에너지 소모는 평균 6% 감소시킬 수 있는 것으로 평가되었다.

ACKNOWLEDGEMENT

This study was partially supported by Korea Institute of Science and Technology Information (KISTI) Grant (No. K-20-L02-C08-S01) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1G1A1011403).

REFERENCES

[1] Hackenberg, D., Schöne, R., Ilsche, T., Molka, D., Schuchart, J., & Geyer, R. An energy efficiency feature survey of the intel haswell processor. IEEE international parallel and distributed processing symposium workshop, pp. 896-904, May 2015. DOI: 10.1109/IPDPSW.2015.70

[2] Gholkar, N., Mueller, F., & Rountree, B. “Uncore power scavenger: A runtime for uncore power conservation on hpc systems.” Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-23, November 2019, DOI: 10.1145/3295500.3356150

[3] Guide, Part. "Intel® 64 and ia-32 architectures software developer’s manual." Volume 3B: System programming Guide, Part 2.11, 2021.

[4] Loh, G. “The cost of uncore in throughput-oriented many-core processors.” Workshop on Architectures and Languages for Throughput Applications, pp. 1-9, June 2008

[5] Marathe, A., Bailey, P. E., Lowenthal, D. K., Rountree, B., Schulz, M., & de Supinski, B. R. “A run-time system for power-constrained HPC applications.” International conference on high performance computing, pp. 394-408, July 2015, DOI: 10.1007/978-3-319-20119-1_28

[6] Freeh, V. W., Kappiah, N., Lowenthal, D. K., & Bletsch, T. K. “Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs.” Journal of Parallel and Distributed Computing, 68(9), pp. 1175-1185, November 2008 DOI: 10.1016/j.jpdc.2008.04.007

[7] Bekele, S. A., Balakrishnan, M., & Kumar, A. “ML guided energy-performance trade-off estimation for uncore frequency scaling.” 2019 Spring Simulation Conference (SpringSim), pp. 1-12, April 2019, DOI: 10.23919/SpringSim.2019.8732878

[8] Won, J. Y., Chen, X., Gratz, P., Hu, J., & Soteriou, V. (2014, February). “Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management.” 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 308-319, February 2014, DOI:

10.1109/HPCA.2014.6835941

[9] Sundriyal, V., Sosonkina, M., Westheimer, B., & Gordon, M. “Core and uncore joint frequency scaling strategy. Journal of Computer and Communications”, 6(12), pp. 184-201, December 2018, DOI: 10.4236/jcc.2018.612018

[10] Sundriyal, V., Sosonkina, M., Westheimer, B. M., & Gordon, M. “Comparisons of core and uncore frequency scaling modes in quantum chemistry application GAMESS.” Proceedings of the High Performance Computing Symposium, pp. 1-11, April 2018, DOI: 10.13140/RG.2.2.15809.45923

[11] André, E., Dulong, R., Guermouche, A., & Trahay, F. “DUF: Dynamic Uncore Frequency scaling to reduce power consumption”, 2020

[12] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., ... & Weeratunga, S. K. “The NAS parallel benchmarks summary and preliminary results.” Supercomputing'91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing, pp. 158-165, November 1991, DOI: 10.1145/125826.125925

[13] Hill, D. L., Bachand, D., Bilgin, S., Greiner, R., Hammarlund, P., Huff, T., ... & Safranek, R. “THE UNCORE: A MODULAR APPROACH TO FEEDING THE HIGH-PERFORMANCE CORES". Intel Technology Journal, 14(3), 2010.

[14] Guide, Part. "Intel® 64 and ia-32 architectures software developer’s manual." Volume 4B: Model- specific registers Part 2.17, 2021.

[15] De Melo, A. C. “The new linux’perf’tools”. Slides from Linux Kongress, Vol. 18, pp. 1-42, September 2010.

[16] Zhu, G., Han, J., Lee, S., & Son, Y. “An Empirical Evaluation of NVM-aware File Systems on Intel Optane DC Persistent Memory Modules.” 2021 International Conference on Information Networking (ICOIN) (pp. 559-564), January 2021, DOI: 10.1109/ICOIN50884.2021.9333911

[17] Schöne, R., Ilsche, T., Bielert, M., Gocht, A., & Hackenberg, D. “Energy efficiency features of the intel skylake-sp processor and their impact on performance.” In 2019 International Conference on High Performance Computing & Simulation (HPCS), pp. 399-406, July 2019, DOI: 10.1109/HPCS48598.2019.9188239

[18] Cheng, H. Y., Zhan, J., Zhao, J., Xie, Y., Sampson, J., & Irwin, M. J. “Core vs. uncore: The heart of darkness.” 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1-6, June 2015, DOI: 10.1145/2744769.2647916

[19] Subramaniam, B., & Feng, W. C. “Towards energy-proportional computing for enterprise-class server workloads.” Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, pp. 15-26, April 2013, DOI: 10.1145/2479871.2479878

[20] Wang, Y., Zhang, W., Hao, M., & Wang, Z.. “Online Power Management for Multi-cores: A Reinforcement Learning Based Approach” IEEE Transactions on Parallel and Distributed Systems, June 2021, DOI: 10.1109/TPDS.2021.3092270

[21] Takouna, I., Dawoud, W., & Meinel, C. “Analysis and simulation of hpc applications in virtualized data centers.” 2012 IEEE International Conference on Green Computing and Communications, pp. 498-507, November 2012, DOI: 10.1109/GreenCom.2012.80

[22] Mankodi, A., Bhatt, A., & Chaudhury, B. “Evaluation of Neural Network Models for Performance Prediction of Scientific Applications.” 2020 IEEE REGION 10 CONFERENCE (TENCON), pp. 426-431, November 2020, DOI: 10.1109/TENCON50793.2020.9293788

Authors

Yongho Lee received the B.S degrees in Computer Science and Engineering from Kyungpook National University, Korea, in 2021. He is currently an M.S student in Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea. He is interested in Computer architecture, Heterogeneous memory system and Non-volatile memory system

Osang Kwon received the B.S degrees in Computer Science and Engineering from Kyungpook National University, Korea, in 2021. He is currently an M.S student in Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea. He is interested in Computer architecture, Heterogeneous memory system and Virtual memory

Kwangeun Byeon received the B.S degrees in Computer Science and Engineering from Kyungpook National University, Korea, in 2021. He is currently an M.S student in Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea. He is interested in Computer architecture, Domain specific architecture design and Reliable computing

Yongjun Kim received the B.S degrees in Computer Science and Engineering from Kyungpook National University, Korea, in 2021. He is currently an M.S student in Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea. He is interested in Computer architecture, Heterogeneous memory system and GPU-Memory system

Seokin Hong received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2015. From 2015 to 2017, he had been a senior engineer at Samsung Electronics. In 2017, he moved to IBM T.J. Watson Research Center where he worked on secure processor architectures and emerging memory/storage systems. He is currently an assistant professor at Sungkyunkwan University. His current research interests include the design of low power, reliable, and high-performance processor architectures and memory systems. He received Best Paper Awards from International Conference on Computer Design (ICCD) in 2010 and Design Automation and Test in Europe (DATE) in 2013.