JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 2, pp. 11-19, February 2020
https://doi.org/10.9708/jksci.2020.25.02.011

# IPC-based Dynamic SM management on GPGPU for Executing AES Algorithm

Dong Oh Son*, Hong Jun Choi**, Cheol Hong Kim***

*Senior Researcher, SK Hynix Memory System R&D, Icheon, Korea
**Senior Researcher, The Attached Institute of ETRI, Daejeon, Korea
***Professor, School of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea

## [Abstract]

Modern GPU can execute general purpose computation on the graphic processing unit, and provide high performance by exploiting many core on GPU. To run AES algorithm efficiently, parallel computational resources are required. However, computational resource of CPU architecture are not enough to cryptographic algorithm such as AES whereas GPU architecture has mass parallel computation resources. Therefore, this paper reduce the time to execute AES by employing parallel computational resource on GPGPU. Unfortunately, AES cannot utilize computational resource on GPGPU since it isn't suitable to GPGPU architecture. In this paper, IPC based dynamic SM management technique are proposed to efficiently execute AES on GPGPU. IPC based dynamic SM management can increase and decrease the number of active SMs by using IPC in run-time. According to simulation results, proposed technique improve the performance by increasing resource utilization compared to baseline GPGPU architecture. The results show that AES improve the performance by 41.2% on average.

▸ **Key words**: AES(Advanced Encryption Standard), GPU(Graphics processing unit), SM(Streaming Multiprocessors), GPGPU(General-Purpose computation on the GPU)

## [요 약]

최신 GPU는 GPGPU를 활용하여 범용 연산이 가능하다. 뿐만 아니라, GPU는 내장된 다수의 코어를 활용하여 강력한 연산 처리량을 제공한다. AES 알고리즘은 다수의 병렬 연산을 요구하지만 CPU 구조에서는 효율적인 병렬처리가 이뤄지지 않는다. 따라서, 본 논문에서는 강력한 병력 연산 자원을 활용하는 GPGPU 구조에서 AES 알고리즘을 수행함으로써 AES 알고리즘 처리시간을 줄여보았다. 하지만, GPGPU 구조는 AES 알고리즘 같은 암호알고리즘에 최적화되어 있지 않다. 그러므로 AES 알고리즘에 최적화될 수 있도록 재구성 가능한 GPGPU 구조를 제안하고자 한다. 제안된 기법은 SM의 개수를 동적으로 할당하는 IPC 기반 SM 동적 관리 기법이다. IPC 기반 SM 동적 관리 기법은 GPGPU 구조에서 동작하는 AES의 IPC를 실시간으로 반영하여 최적의 SM의 개수를 동적으로 할당한다. 실험 결과에 따르면 제안된 동적 SM 관리 기법은 기존의 GPGPU 구조와 비교하여 하드웨어 자원을 효과적으로 활용하여 성능을 크게 향상시켰다. 일반적인 GPGP 구조와 비교하여, 제안된 기법의 AES의 암호화/복호화는 평균 41.2%의 성능 향상을 보여준다.

▸ **주제어**: 고급 암호화표준, 그래픽 처리 장치, 스트리밍 멀티프로세서, GPU 상의 범용 계산

• First Author: Dong Oh Son, Corresponding Author: Cheol Hong Kim
*Dong Oh Son (dongoh.son@sk.com), SK Hynix Memory System R&D
**Hong Jun Choi (chj6083@nsr.re.kr), The Affiliated Institute of ETRI
***Cheol Hong Kim (chkim22@chonnam.ac.kr), School of Electronics and Computer Engineering, Chonnam National University
• Received: 2019. 07. 19, Revised: 2019. 12. 01, Accepted: 2019. 12. 02.


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

## I. Introduction

그래픽 작업에 특화된 그래픽 처리 장치(GPU: Graphics processing unit)는 수천개의 병렬 스레드를 활용하며 고성능을 제공한다. GPU는 단일 명령 다중 스레드(SIMT: Single Instruction Multiple Threads) 모델을 사용하여 하나의 명령을 다수의 스레드를 통해 병렬적으로 처리가 가능하다. 또한, GPU는 다양한 그래픽 작업을 수행하기 위한 프로그래머블 프로세서를 포함하여 그래픽 연산에 특화되어 있다[1,2]. 최근 GPU는 그래픽 연산뿐만 아니라 범용 연산이 가능하며 이를 GPGPU(General-Purpose computation on the Graphics Processing Units)라 부른다. GPU 개발 업체들은 GPGPU를 효율적으로 활용하기 위해서 응용프로그램 인터페이스(API: Application Programming Interface)를 제공하고 있으며, 제공되는 API는 CUDA, OpenCL, ATI Stream 등이 있다[3-5]. 이처럼 최신 GPU는 GPGPU 구조를 활용하여 성능을 향상시키며 다양한 범위에서 시스템 성능을 향상시키기 위해 사용되고 있다[6-8]. 높은 성능을 제공하기 위해 GPU는 높은 클럭 주파수에 동작하며 대용량 병렬 컴퓨팅 연산을 수행한다. 예를 들어 NVIDIA사의 TITAN X GPU는 3,584개의 CUDA 코어를 사용하며, 1,531MHz의 주파수에서 동작한다[9]. 하지만 GPGPU 하드웨어 자원 보다 작은 크기의 작업들은 GPU의 하드웨어 자원을 효율적으로 활용하지 못하며, 오히려 다수의 CUDA 코어를 사용하면 성능의 감소로 이어지기도 한다. 따라서, GPGPU 구조의 하드웨어 자원보다 낮은 작업 처리량을 가진 애플리케이션은 작업량에 맞는 하드웨어 자원 할당이 필요하다.

AES(Advanced Encryption Standard)는 고급 암호화 표준으로 강력한 암호화 알고리즘이다. AES는 개발된 알고리즘에 따라 암호화와 복호화로 동작한다. 각 방식에 따라 128비트, 192비트, 그리고 256비트의 키 값을 활용하여 동작하며, 각 비트별로 10에서 14라운드로 연산이 수행된다. AES를 동작하기 위한 환경으로는 소프트웨어, 하드웨어, 그리고 펌웨어까지 다양한 환경에서 동작이 가능하다. 하지만 빠른 연산결과를 도출하기 위해서는 높은 하드웨어 성능이 요구된다. 선형적으로 명령어를 처리하는 CPU 구조에서는 병렬적으로 수행되는 AES 알고리즘을 수행하기에는 적합하지 않다. 하지만 다수의 코어를 활용하여 병렬 연산을 수행하는 GPGPU 구조에서는 AES 알고리즘을 효율적으로 수행이 가능하다.

본 논문에서는 AES의 암호화/복호화 연산의 성능 향상을 위해서 GPGPU 구조를 활용하고자 한다. 강력한 하드웨어 자원을 가진 GPGPU 구조는 AES의 암호화/복호화를 수행할 때 병렬처리를 통해 성능 향상이 가능하다. 하지만, AES의 암호화/복호화가 GPGPU 구조에 최적화 되어있지 않기 때문에 작업량에 맞는 하드웨어 자원 할당이 가능한 IPC 기반 SM 동적 관리 기법을 제안한다. 제안하는 기법은 GPGPU 구조에서 동작하는 AES의 최적화를 위해 SM의 개수를 동적으로 변경한다. 제안된 기법은 각 입력 값에 따라 동적으로 SM의 개수를 할당하여 AES의 최적화가 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 AES와 실험 대상인 GPU 구조에 대해서 자세히 설명한다. 또한, 3장에서는 CPU와 GPU 환경에서 동작하는 AES 비교와 GPU에서 동작하는 AES의 최적화를 위한 IPC 기반 동적 SM 관리 기법에 대해서 기술한다. 4장에서는 실험 환경과 실험 결과에 대해서 자세히 기술한다. 마지막으로 5장에서 본 논문의 결론을 맺는다.

## II. Background

### 1. AES(Advanced Encryption Standard) algorithm

고급 암호화 표준(AES: Advanced Encryption Standard)는 미국 표준 기술 연구소에서 제정한 암호화 방식을 말한다[10]. AES는 미국 표준 기술 연구소에서 제시한 3가지 선정 기준을 만족한다. 선정 기준 3가지는 안전성(security), 비용(cost), 그리고 구현 효율성(implementation)이다. AES는 DES(Data Encryption Standard)를 대체하기 위한 알고리즘으로 DES에 대한 공격 방식들은 AES에서 보완되었으며 DES 보다 큰 크기의 키 값을 사용하기 때문에 DES 보다 더 안전하다. AES를 공격하기 위한 통계적인 분석, 차분 공격, 그리고 선형 공격 방법들은 실패했으며 지금까지 알려진 블록 암호 알고리즘에 대한 모든 공격 방법들에 대해서 안정성을 보장하고 있다. AES는 알고리즘 설계가 매우 단순하며 구현에 있어 많은 하드웨어 자원을 요구하지 않기 때문에 저가형 임베디드 프로세서와 작은 크기의 메모리만으로도 설계가 가능하다. 또한, 구현 방법에 있어서 AES는 소프트웨어, 하드웨어, 펌웨어로 구현이 가능하다. AES는 128비트, 192비트, 그리고 256비트로 키 값으로 암호화 처리가 가능하며, 알고리즘을 통해 128비트, 192비트, 그리고 256비트의 암호문으로 출력된다. AES는 암호화를 위해서 각 비트별로 10, 12, 그리고 14라운드를 활용하며 4 X 4 크기부터 8 X 8 크기의 행렬을 활용하여 연산을 수행한다. 또한 AES는 대치(substitution), 치환(permutation), 뒤섞음(mixing), 키덧셈(key-adding)의 4가지 형태 변환을 통해 안정성을 확보할 수 있다.

## 2. GPU architecture

본 장에서는 NVIDIA에서 사용하는 GPU 용어를 인용하여 GPU 구조에 대한 배경지식을 기술하고자 한다. 다양한 GPU 제조회사들 [11,12]의 다른 용어 사용에 혼동을 방지하기 위해서 본 논문에서는 NVIDIA GPU 구조에 정의된 용어를 사용한다[13]. GPU 구조에서, 다중 스레드들은 SIMT(Single Instruction Multiple Threads) 구조를 지원하기 위하여 워프(Warp)로 알려진 묶음으로 그룹화 된다. 32개의 스레드들로 구성된 하나의 워프는 lock-step으로 처리된다[14]. GPU는 높은 성능을 제공하지만 긴 메모리 접근 지연시간으로 인한 심각한 성능 저하가 발생할 가능성 또한 포함하고 있다. 긴 메모리 접근 지연시간으로 인한 심각한 성능 저하 문제를 해결하기 위하여 GPU는 다수의 워프가 동시에 수행되는 FGMT(Fine-Grained Multi-Threading) 기법을 채택하고 있다[15]. 그러므로 GPU는 동일한 명령어를 수행하는 워프들을 그룹화한 CTA(Cooperative Thread Array)를 할당받아 연산을 수행한다[16].

그림 1은 NVIDIA GPU 기반의 기본적인 GPU 구조를 보여주고 있다. 기본적으로 GPU는 GPU 클러스터와 내부연결망 네트워크 그리고 메모리로 구성되어 있다. 하나의 GPU 클러스터는 온 칩 내부연결망 네트워크를 통해 메모리와 연결된 다수의 SM들로 구성되어 있다[3]. 매 클럭마다 CTA 스케쥴러는 하나의 CTA를 하나의 SM에 할당한다. 본 논문에서 SM을 선택하는 스케쥴링 기법은 라운드 로빈 방식을 사용한다고 가정한다.

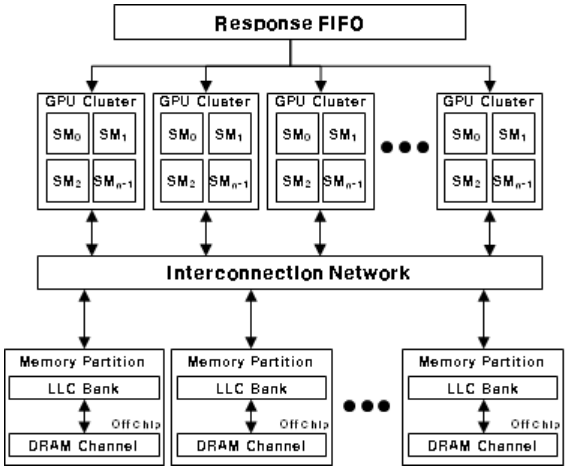


Fig. 1. Baseline GPU Architecture

그림 2는 GPU 클러스터 내부를 보다 상세하게 보여주고 있다. 응답 FIFO는 내부연결망 네트워크로부터 얻은 데이터들을 저장하고 있다. 응답 FIFO는 데이터의 종류에 따라 다른 동작을 수행한다. 만약, 응답 FIFO의 데이터가 명령어라면 응답 FIFO는 그 명령어를 SM 내부의 명령어 캐쉬로 전달할 것이다. 달리 말하면, 응답 FIFO의 데이터가 메모리 요청이라면 응답 FIFO는 그 메모리 요청을 SM 내부의 LSU(Load/Store Unit)로 보낼 것이다. 하지만 불행하게도 모든 SM이 동작 중이라면, 응답 FIFO는 멈추게 될 것이다. 그림2의 오른쪽부분은 SM 내부를 확대하여 보여준다. 다수의 스레드들을 병렬적으로 실행하기 위하여, 각각의 SM은 LSU, 명령어 캐쉬, 거대한 레지스터 파일 그리고 다양한 FU(Functional Unit)들과 공유 메모리 텍스쳐 캐쉬, 상수 캐쉬, L1 데이터 캐쉬 등의 다양한 메모리들을 포함하고 있다. 메모리 데이터는 주입 포트 버퍼를 통해 내부연결망 네트워크로 보내진다[17,18].

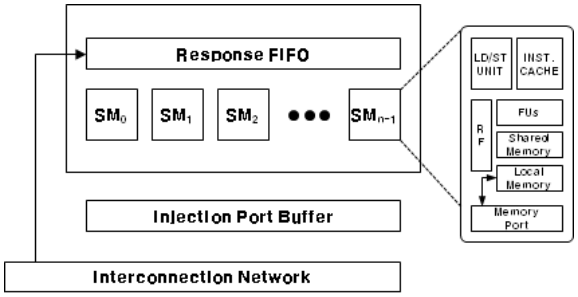


Fig. 2. GPU Cluster

## III. Exploiting GPGPU for performance improvement on AES

본 장에서는 AES를 효과적으로 활용하기 위해서 CPU와 GPU 환경에서의 성능을 비교 분석하며, GPU 환경에서 효율적으로 AES를 수행하기 위한 IPC 기반 동적 SM 관리 기법을 제안한다.

### 1. AES execution on GPU and GPU

AES는 소프트웨어, 하드웨어, 그리고 펌웨어 등 다양한 환경에서 동작이 가능하며 범용PC, 임베디드 프로세서, 그리고 ARM 등 다양한 구조에서도 동작이 가능하다. 아래 그림 3은 CPU 구조에서 동작하는 AES의 암호화 성능(IPC)과 GPGPU구조에서 동작하는 AES의 성능을 비교한 그래프이다.

실험결과 CPU 구조에서 AES의 암호화 성능은 0.73(IPC)으로 매우 낮은 성능을 보이며, GPGPU 구조에서 AES의 암호화 성능은 190.4(IPC)의 높은 성능을 보여

준다. 강력한 하드웨어 자원을 보유한 GPGPU 구조에서는 CPU와 비교하여 약 260배의 성능 향상을 보인다. 다수의 코어를 가진 GPU에서 AES 암호화 연산을 병렬적으로 수행하기 때문에 기존의 CPU 구조와 비교하여 높은 성능 향상을 보여준다.

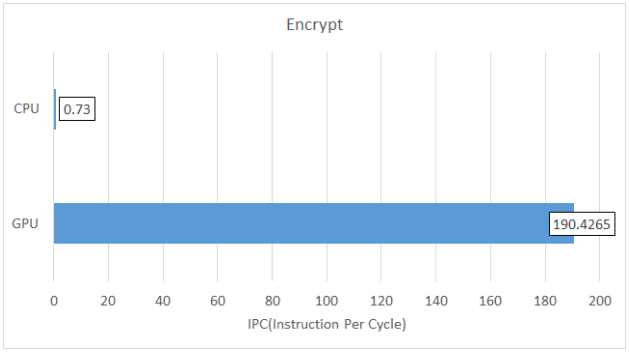


Fig. 3. Comparison of AES performance(CPU and GPU)

일반적으로 GPU의 SM의 개수가 많으면 많을수록 연산처리량이 증가하여 프로그램의 성능이 증가한다. 따라서 128개의 SM을 가진 GPGPU 구조에서 AES를 실행하면 높은 성능 향상을 보인다. 하지만 하드웨어 자원보다 낮은 연산처리량을 가지는 경우 하드웨어 자원을 효과적으로 활용하지 못하며, 각 SM에 작업을 할당하기 위해 작업을 분할하는데 추가적인 작업이 필요하다. 또한, 각 SM에서 연산이 수행 될 때 메모리 요청에 따른 메모리 스톨 등의 문제가 발생 할 수 있다. 따라서 128개의 SM을 가진 기존의 GPGPU 구조에 AES는 최적화되어 있지 않다. GPGPU 구조에서 동작하는 AES의 최적화를 위해서는 각 SM의 개수의 변화에 따른 성능 값 분석을 수행하며, 분석된 결과를 활용하여 이를 효율적으로 활용하여야 한다. 본 논문에서는 GPGPU 구조에서 동작하는 AES의 성능 향상을 위해서 각 SM의 개수 변화에 따른 AES의 성능을 분석하며, 성능 향상을 위한 IPC 기반 동적 SM 관리 기법을 제안한다.

## 2. IPC-based dynamic SM management

기존 연구[19]에 따르면 벤치마크의 특성에 따라 GPU에 할당되는 하드웨어 자원이 증가하여도 성능이 선형적으로 증가하지 않음을 알 수 있다. 하드웨어 자원의 증가는 병렬성의 증가와 메모리 접근시간 감소의 이점이 있으나 SM 스톨 또는 메모리 충돌과 같은 단점도 존재한다. 이러한 문제를 해결하기 위해 워프 단위의 스케쥴링 기법과 CTA 단위의 스케쥴링 기법이 연구되었다[19,20]. 본 논문에서는 워프, CTA가 수행되는 다수의 SM을 대상으로 하고, AES의 암호화/복호화를 GPGPU 구조에서 효율적으로 활용하기 위해서 SM의 개수를 동적으로 변경하는 IPC 기반 동적 SM 관리 기법을 제안한다.

본 논문에서 제안하는 IPC 기반 동적 SM 관리 기법은 초기 SM의 개수를 1개로 설정하고 AES 벤치마크 프로그램을 수행하며, 1,000 사이클 마다 IPC를 측정하며 SM의 개수를 2의 배수로 증가시킨다. 해당 주기마다 성능(IPC)를 측정하여 성능이 향상되는 경우 SM의 개수를 증가시키며, 기존보다 성능이 하락한 경우 SM의 개수를 증가시키지 않고 이전의 SM의 개수로 SM의 개수를 고정시킨다. 이처럼 동적으로 SM의 개수를 변경할 수 있다면, AES 벤치마크 프로그램을 수행할 때 SM 개수에 따른 최적의 성능을 유지할 수 있다. 제안된 기법은 SM의 개수를 1개부터 최대 128개까지 변경할 수 있으며, 다양한 입력 값에 적용이 가능하다.

아래 알고리즘 1은 IPC 기반 동적 SM 관리 기법에 대한 pseudo 코드이다. 처음 AES 벤치마크 프로그램이 수행되면 128개의 SM 중 1개의 SM만을 활성화 시킨다. 그리고 1,000 사이클 마다 현재까지 수행된 명령어 개수와 사이클 수를 확인하여 IPC를 측정한다. 측정된 IPC는 임시공간에 저장되며, 이전 결과와 비교하여 성능이 향상된 경우 SM의 개수를 2의 배수로 증가시킨다. 연산복잡도가 높은 AES는 하드웨어 활용률을 높이는 것이 중요함으로 SM 스톨 또는 메모리 충돌이 발생하여 성능이 하락하기 전까지 지속적으로 SM의 개수를 증가 시킨다. 제안한 알고리즘은 SM의 개수를 1개부터 최대 128개까지 증가시키며, 이전 결과와 비교하여 성능이 하락한 경우에만 SM의 개수를 이전의 개수로 변경시키고 SM의 개수를 고정시킨다.

Algorithm 1. Proposed SM management Algorithm

```
1:  function SM_Management(sm)
2:    loop
3:      r ← cycle mod 1000
4:      if r = 0 then
5:        check current_IPC
6:        if current_IPC > previous_IPC then
7:          sm.Num ← sm.Num * 2
8:        else
9:          sm.Num ← sm.Num / 2
10:         exit loop
11:       end if
12:     end if
13:   end loop
14: end function
```

제안하는 기법은 기존의 128개의 SM 개수에서 활성화시키는 SM의 개수만을 변경하기 때문에 별도의 하드웨어 공간이 필요하지 않으며 연산복잡도 또한 증가시키지 않는다.

## IV. Experiments

본 논문에서는 AES의 효율적인 암호화, 복호화를 수행하기 위해서 GPGPU를 활용하여 병렬연산을 수행하며, 성능 결과를 상세하게 측정하기 위해 다음과 같은 실험환경을 구성하였다.

우리는 제안된 기법의 성능과 전력 효율성을 평가하기 위해서 GPGPU-Sim[21]과 McPAT[22]가 통합된 GPUWattch[23] 시뮬레이터를 사용하였다. GPGPU-Sim은 GPU와 같은 병렬 구조의 다양한 측면을 고려하기 위해서 개발되었다. 본 실험에서는 ISPASS[21] 벤치마크 중에서 AES 벤치마크를 대상을 실험을 수행한다.

Table 1. Hardware parameters

| Parameter | Value |
|---|---|
| Number of GPU Clusters | 1 |
| Number of SMs per GPU Cluster | 128 |
| Maximum Number of Current CTAs/GPU Cluster | 8 |
| PTXPLUS | Enabled |
| Number of Threads/SM | 1536 |
| Number of SP Units/SM | 2 |
| Number of SFU Units/SM | 1 |
| Number of LSDT Units/SM | 1 |
| Clock (Core; Interconnection; L2; DRAM) | 700;700;700;924 (MHz) |
| Number of Packets in Ejection Buffer | 8 |
| Number of Response Packets in LDST Unit of Ejection Buffer | 2 |
| CTA Scheduling Scheme for Distribution and GPU Status Table | Round Robin Policy |
| Warp Scheduling Scheme | Round Robin Policy |

표 1과 2는 실험에서 사용된 프로세서 구성 변수와 메모리 구성 변수를 나타낸다. 본 실험에서 대상으로 하는 GPU는 NVIDIA GTX480[24]과 동일한 환경으로 구성으로 된 GPU 구조를 대상으로 한다[25]. 대상 GPU는 700 MHz에서 작동되며, 128개의 SMs로 구성 되며 각 SM은 2개의 SP 장치 그리고 명령어 캐쉬와 데이터 캐쉬로 구성된다. 각 SP 장치는 16개의 CUDA 코어로 구성되어 있다. 실험용 GPU 구조는 총 4,096개의 GPU CUDA 코어로 구성되어 있으며 정수형, 실수형 파이프라인을 독립적으로 수행한다. 또한 CTA(Cooperative Thread Array)와 SMs(Streaming Multiprocessors)을 선택하기 위한 기본 스케쥴링 방식으로 라운드 로빈 스케쥴링을 사용한다. 다음 장에서 제안된 IPC 기반 동적 SM 관리 기법에 대한 실험 결과 및 분석을 상세하게 설명한다.

Table 2. Memory parameters

| Parameter | Value |
|---|---|
| Number of Registers/SM | 32768 |
| Shared Memory/SM | 16KB |
| Constant Cache/SM | 8KB, 2-way 64byte lines, Read-only |
| Texture Cache/SM | 12KB, 24-way 128byte lines, Read-only |
| L1 Inst. Cache | 2KB, 4-way, 128byte lines |
| L1 Data Cache | 16KB, 4-way, 128byte lines |
| L2 Data Cache | 64KB, 8-way, 128byte lines |
| Number of Memory Controllers | 8 |
| Number of Memory Subpartition/Controller | 2 |
| Memory Channel Bandwidth | 4 bytes |
| DRAM Scheduler Queue Size | 16 |
| DRAM Request Queue Size | 116 |
| GDDR3 Memory Timing | tCL=12, tRP =12, tRC=40, tRAS=28, tRCD=12, tRRD=6 |

## V. Experiment result

### 1. Exploring optimal number of SM on AES execution for proper hardware resources utilization

본 논문에서는 AES의 암호화/복호화 알고리즘을 수행할 때 GPGPU에 적합한 최적의 SM의 개수를 찾기 위해 각 SM의 개수를 변화하여 IPC를 기준으로 하드웨어 자원 활용 효율성을 분석하였다. 실험방법으로는 AES의 암호화

/복호화에 128비트, 256비트의 키 값을 사용하였고 알고리즘 적용 대상으로 128 x 128 픽셀의 BMP 파일과 256 x 256 픽셀의 BMP 파일을 대상으로 실험을 수행하였다.

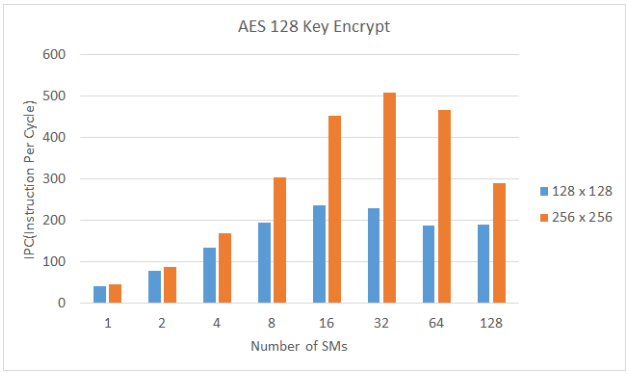


Fig. 4. AES encryption performance(128 key)

그림 4는 AES 벤치마크 중 128비트의 키 값을 갖는 알고리즘에 대한 암호화 성능을 보여준다. 실험 결과 128 x 128 픽셀의 경우 16개의 SM을 가지는 구조에서 213.7(IPC)로 최적의 성능을 보이며, 256 x 256 픽셀의 경우 32개의 SM에서 497.7(IPC)로 최적의 성능을 보인다. GPGPU는 강력한 하드웨어 성능을 보이며 대용량 연산에 특화되어 있다. 하지만 작업 처리량 보다 하드웨어 자원이 뛰어날 경우 하드웨어 활용률은 떨어지게 된다. 그림에서 보면 128 x 128 픽셀의 암호화 연산보다 256 x 256 픽셀의 암호화 연산이 높은 성능을 보임을 알 수 있다. 이러한 이유로는 128 x 128 픽셀은 하드웨어 자원을 효율적으로 활용하기에는 AES 알고리즘이 가진 작업 처리량이 작은 것으로 판단된다. 각 SM의 개수에 따른 최적의 성능 또한 128 x 128 픽셀과 256 x 256 픽셀의 차이를 볼 수 있다. 대용량의 연산을 처리하기 위해서는 SM의 개수가 매우 중요하지만 GPU가 보유하고 있는 128개의 SM을 활용하기에는 AES 알고리즘의 작업 처리량이 크지 않다. 따라서 128 x 128 픽셀의 경우 16개의 SM에서 하드웨어 자원 활용률이 최적화 되고, 256 x 256 픽셀에서는 32개의 SM에서 하드웨어 자원 활용률이 최적화 된다. 실험결과에서 알 수 있듯이 강력한 하드웨어 자원을 보유한 GPU는 자원 활용률 최적화와 전력 효율성 향상을 위해서 SM의 개수를 입력 값에 따라 동적으로 할당한다면 GPGPU 구조의 하드웨어 자원 활용률을 향상 시킬 수 있다.

그림 5는 AES 벤치마크 중 128비트의 키 값을 갖는 알고리즘에 대한 복호화 성능을 보여준다. 그림 4의 AES 암호화 성능(128 key)과 비교하여 성능에 큰 차이를 보이지 않으며 유사한 성능 분포를 보여준다. 128 x 128 픽셀의 경우 AES 암호화 성능(128 key)과 동일하게 16개의 SM에서 가장 높은 성능 236.5(IPC)를 보여주며 256 x 256 픽셀의 경우 역시 AES 암호화 성능 (128 key)과 동일하게 32개의 SM에서 가장 높은 성능 507.5(IPC)를 보여준다.

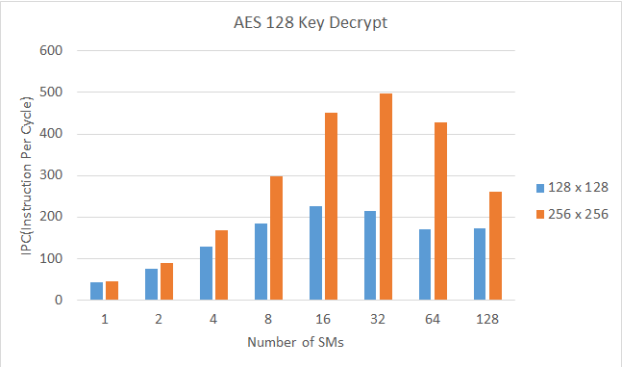


Fig. 5. AES decryption performance(128 key)

그림 6은 256비트의 키 값을 가지는 AES 암호화 성능을 보여준다. 그림 6은 그림 4와 유사한 성능 분포를 보여준다. 실험결과 128 x 128 픽셀을 대상으로 16개의 SM을 가지는 구조에서 266.8(IPC)로 최적화 된 성능을 보여주며 256 x 256 픽셀을 대상으로는 32개의 SM을 가지는 구조에서 596.8(IPC)로 최적화 된 성능을 보여준다. 전체적인 성능 분포는 그림 4와 유사하지만 성능 차이는 최대 104.3(IPC)를 보인다. 이러한 이유는 256비트의 키 값을 가지는 AES의 암호화/복호화 연산에는 더 많은 하드웨어 자원을 필요로 하며 GPGPU 구조는 이를 충분히 지원 할 수 있기 때문에 128비트 키 값보다 높은 성능을 보여준다.

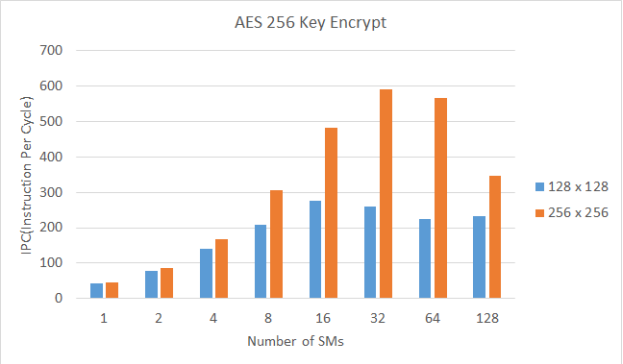


Fig. 6. AES encryption performance(256 key)

그림 7은 AES 벤치마크 중 256비트의 키 값을 갖는 알고리즘에 대한 복호화 성능을 보여준다. 그림 7 역시 그림 6의 AES 암호화 성능(256 key)과 비교하여 성능에 큰 차이를 보이지 않으며 유사한 성능 분포를 보여준다. 128 x 128 픽

셀의 경우 AES 암호화 성능(256 key)과 동일하게 16개의 SM에서 가장 높은 성능 277.0(IPC)을 보여주며 256 x 256 픽셀의 경우 역시 AES 암호화 성능(256 key)과 동일하게 32개의 SM에서 가장 높은 성능 590.6(IPC)을 보여준다.

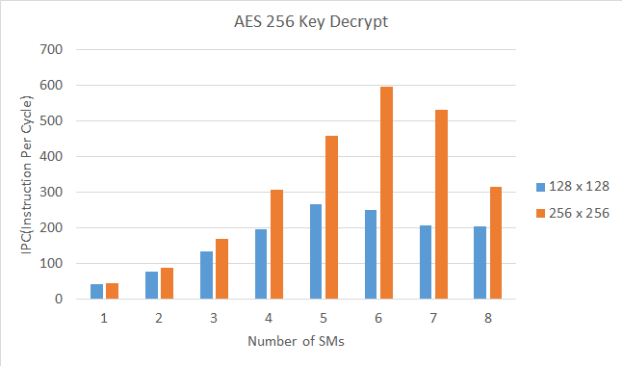


Fig. 7. AES decryption performance(256 key)

## 2. Performance analysis on AES execution by using IPC-based dynamic SM management

앞 장에서는 SM의 개수에 따른 AES의 암호화/복호화 연산의 성능 변화를 분석하였다. 분석 결과 SM의 증가에 따라 성능이 선형적으로 증가하지 않음을 확인하였다. 특정 SM 개수 이상부터는 성능이 하락하는 경우도 발생하였다. 이러한 이유는 GPU의 하드웨어 자원을 AES 알고리즘 연산에서 충분히 활용하지 못하였기 때문으로 판단된다. 따라서 본 논문에서는 GPU에서 동작하는 AES의 효율적인 연산을 위해서 IPC 기반 동적 SM 관리 기법을 사용하였다.

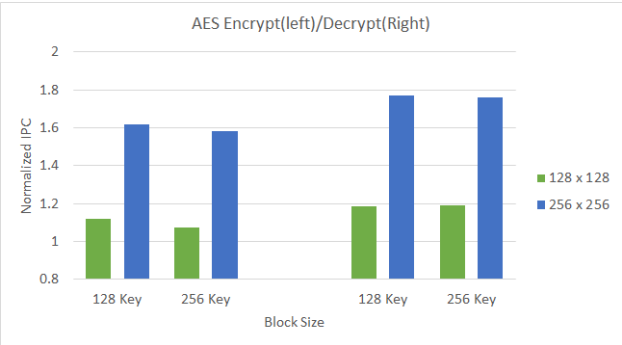


Fig. 8. IPC-based Dynamic SM management

그림 8은 제안된 기법이 적용된 AES의 암호화/복호화의 성능을 기존의 성능과 비교한 결과이다. 결과 값은 기본구조(128 SMs)의 성능을 정규화 하였다. 실험 결과 암호화/복호화 모든 경우에서 기존의 GPU 구조에서 수행한 결과보다 높은 성능을 보여준다. 8개의 그래프 중에서 왼쪽 4개의 그래프는 AES의 암호화에 대한 값을 나타내며 오른쪽 4개의 그래프는 AES의 복호화에 대한 값을 나타낸다. 암호화의 경우 128비트의 키 값에서 256비트 키 값보다 높은 성능 향상을 보여준다. 또한, 128 x 128 픽셀보다 256 x 256 픽셀에서 매우 높은 성능 향상을 보여준다. 제안된 기법이 적용된 AES 암호화는 최소 7.5%에서 61.9%의 성능 향상을 보여준다. 복호화의 경우 128비트의 키 값과 256비트의 키 값에서 별다른 성능의 차이를 보이지 않는다. 하지만, 128 x 128비트의 픽셀과 256 x 256 픽셀의 성능차이는 매우 크다. 제안된 기법이 적용된 AES 복호화는 최소 18.5%에서 77.2%의 성능 향상을 보여준다.

본 장에서는 SM의 개수 변화에 따른 AES의 암호화/복호화의 성능 분석과 본 논문에서 제안하는 기법인 IPC 기반 동적 SM 관리 기법을 적용하여 기존의 GPGPU 구조와의 성능 비교를 수행하였다. 앞서 말했듯이 강력한 하드웨어 자원을 가진 GPU를 활용할 때 SM의 개수는 매우 중요하다. 대부분의 애플리케이션의 경우 SM의 개수가 많으면 많을수록 하드웨어 자원이 향상되기 때문에 성능이 향상된다. 하지만, 보유한 하드웨어 자원보다 낮은 연산량을 수행 할 때에는 GPGPU 구조의 하드웨어 자원을 효율적으로 활용하지 못한다. 본 논문에서는 AES를 GPGPU 구조에서 사용할 때 발생하는 최적화 문제점을 해결하기 위해서 IPC를 기반으로 AES의 암호화/복호화 성능을 파악하고 가장 효율적인 SM의 개수를 동적으로 할당하는 기법을 제안하였다. 실험결과에서 보는 바와 같이 모든 경우에서 기존의 구조와 비교하여 성능이 향상됨을 확인 할 수 있다. 제안된 기법은 8가지의 입력 값에서 평균 41.2%의 높은 성능 향상을 확인하였다.

# VI. Conclusions

본 논문에서는 GPGPU 구조에서 동작하는 AES와 같은 암호알고리즘 수행 최적화를 위해 동적으로 SM의 개수를 변경하는 IPC 기반 동적 SM 관리 기법을 제안하였다. 제안된 기법을 평가하기 위해 블록 비트 값과 입력 데이터 값의 변화에 따라 총 8개의 실험을 수행하였다. AES의 암호화 보다는 복호화에서 제안된 기법의 성능 향상이 두드러졌으며, 128 x 128 픽셀의 입력 값 보다 상대적으로 연산량이 많은 256 x 256 픽셀의 입력 값에서 높은 성능 향상을 보였다. 실험 결과 7.5%에서 77.2%의 성능이 향상되었으며 평균 41.2%의 높은 성능 향상을 확인 할 수 있었다. 이러한 이유는 특정 사이클 마다 SM의 개수를 증가하여 IPC를 확인하고 기존의 IPC와 비교하여 성능이 감소하

는 경우 SM의 개수를 증가시키지 않고 최적의 SM의 개수를 동적으로 변경함으로써 AES 알고리즘을 최적화 하였기 때문이다.

GPGPU 구조는 CPU 구조와 비교하여 AES의 암호화/복호화 연산을 효율적으로 연산 할 수 있다. 본 논문에서는 GPGPU에서 동작하는 AES의 암호화/복호화에 대한 최적화를 수행하였다. 본 논문에서 제안한 기법을 GPU에 적용하여 AES와 같은 암호알고리즘의 최적화 수행을 지원한다면 높은 성능을 얻을 수 있을 것으로 기대된다.

## REFERENCES


[1] D. Luebke and G. Humphreys. “How GPUs work,” Journal of Computer, Vol. 40, No. 2, pp. 96-100, February 2007. DOI: 10.1109/MC.2007.59

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan, “Brook for GPUs: stream computing on graphics hardware,” In Proceedings of Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 777-786, August 2004. DOI: 10.1145/1186562.1015800

[3] CUDA Programming Guide Version 3.0, available at https://developer.nvidia.com/cuda-toolkit-30-downloads

[4] Khronos Group, OpenCL, available at http://www.khronos.org/opencl

[5] ATI Stream SDK, available at http://developer.amd.com/community/blog/2009/08/05/ati-stream-sdk-and-opencl

[6] General-purpose computation on graphics hardware, available at http://www.gpgpu.org

[7] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” Computer Graphics Forum, Vol. 26, No. 1, pp. 21-51, March 2007. DOI: 10.1111/j.1467-8659.2007.01012.x

[8] Y. Yang, P. Xiang, M. Mantor, and H. Zhou, “CPU-assisted GPGPU on fused CPU-GPU architectures,“ In Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, pp. 1-12, March 2012. DOI: 10.1109/HPCA.2012.6168948

[9] NVIDIA TITAN X, available at http://www.nvidia.co.kr/graphics-cards/geforce/pascal/kr/titan-x-pascal

[10] X. Zhang, and K.K. Parhi, “High-speed VLSI architectures for the AES algorithm,” In Proceedings of IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 957-967, August 2004. DOI: 10.1109/TVLSI.2004.832943

[11] NVIDA Co. Ltd., available at http://www.nvidia.com

[12] AMD(Advanced Micro Devices) Inc., available at http://www.amd.com

[13] NVIDIA’s Next Generation CUDA Compute Architecture: Fermi, available at http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf

[14] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, “Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow,” In Proceedings of International Symposium on Microarchitecture, pp. 407-420, December 2007.

[15] J. E. Thornton, “Parallel operation in the control data 6600,” In AFIPS Proceedings of FJCC, Part. 2, Vol. 26, pp. 33-40, 1964. DOI: 10.1109/MICRO.2007.12

[16] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, “Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling,” In Proceedings of the International Symposium on High Performance Computer Architecture, pp. 260-271, June 2014. DOI: 10.1109/HPCA.2014.6835937

[17] K. M. Abdalla, L. V. Shah, J. F. Duluk, T. J. Purcell, T. Mandal, and G. Hirota, “Scheduling and Execution of Compute Tasks,” US Patent US20130185725, 2013.

[18] H. Choi, D. Son, J. Kim, and C. Kim, “Concurrent warp execution: improving performance of GPU-likely SIMD architecture by increasing resource utilization,” Journal of SuperComputing, Vol. 69, No. 1, pp. 330-356, July 2014. DOI: 10.1007/s11227-014-1155-4

[19] D. Son, J. Kim, and C. Kim, “An IPC-based Dynamic Cooperative Thread Array Scheduling Scheme for GPUs,” Journal of The Korea Society of Computer and Information, Vol. 21, No. 2, pp. 9-16, February 2016. DOI: 10.9708/jksci.2016.21.2.009

[20] G. Kim, J. Kim, and C. Kim, “Latency Hiding based Warp Scheduling Policy for High Performance GPUs,” Journal of The Korea Society of Computer and Information, Vol. 24, No. 4, pp. 1-9, April 2019. DOI: 10.9708/jksci.2019.24.04.001

[21] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, “Analyzing CUDA Workloads Using a Detailed GPU Simulator,” In Proceedings of International Symposium on Performance Analysis of Systems and Software, pp. 163-174, April 2009. DOI: 10.1109/ISPASS.2009.4919648

[22] S. Li, J. H Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures,” In Proceedings of the International Symposium on Microarchitecture, pp. 469-480, January 2009. DOI: 10.1145/1669112.1669172

[23] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “GPUWattch: Enabling Energy Optimizations in GPGPUs,” In Proceedings of the International Symposium Computer Architecture, pp. 487-498, June 2013. DOI: 10.1145/2485922.2485964

[24] GTX480 NVIDIA, available at http://www.geforce.com/hardware /desktop-gpus/geforce-gtx-480

[25] M. Abdel-Majeed, D. Wong, and M. Annavaram, "Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs," In Proceedings of International Symposium on Microarchitecture, pp. 111-122, December 2013. DOI: 10.1145/2540708.2540719

## Authors

Dong Oh Son received the B.S. degree and M.S. in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2010 and 2012 respectively. He received the Ph.D. degree in computer engineering from Chonnam National University in 2016. Currently, he works at SK Hynix Memory System R&D. His research interests include computer architecture, embedded systems and GPGPU.

Hong Jun Choi received the B.S. degree and M.S. in electronics and computer engineering from Chonnam National University, Gwangju, Korea in 2009 and 2011 respectively. He received the Ph.D. degree in computer engineering from Chonnam National University in 2014. From Sept 2014 to Jan 2015, he worked as a postdoctoral researcher at Chonnam National University. Currently, he works at the Attached Institute of ETRI. His research interests include computer architecture, low-power processors and GPGPU

Cheol Hong Kim received the B.S. degree in Computer Engineering from Seoul National University, Seoul, Korea in 1998 and M.S. degree in 2000. He received the Ph.D. in Electrical and Computer Engineering from Seoul National University in 2006. He worked as a senior engineer for SoC Laboratory in Samsung Electronics, Korea from Dec. 2005 to Jan. 2007. Now is working as Professor at School of Electronics and Computer Engineering, Chonnam National University, Korea. His research interests include embedded systems, mobile systems, computer architecture, SoC design, low power systems, and multiprocessors.