ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 21-03-11
68

j.inst.Korean.electr.electron.eng.Vol.25,No.3,479~484,September 2021
https://doi.org/10.7471/ikeee.2021.25.3.479

작은 크기의 Warp 스케쥴러 기반 SIMT구조 고성능 모바일 GPGPU 설계

Design of a High-Performance Mobile GPGPU with SIMT Architecture based on a Small-size Warp Scheduler

이 광 엽*★

Kwang-Yeob Lee*★

Abstract

This paper proposed and designed a structure to achieve high performance with a small number of cores in GPGPU with SIMT structure. GPGPU for application to mobile devices requires a structure to increase performance compared to power consumption. In order to reduce power consumption, the number of cores decreased, but to improve performance, the size of the warp scheduler for managing threads was set to 4, which was greatly reduced than 32 of general GPGPU. Reducing warp size can reduce the number of idle cycles in pipelines and efficiently apply memory latency to reduce miss penalty when accessing cache memory. The designed GPGPU measured computational performance using a test program that includes floating point operations and measured power consumption through a 28nm CMOS process to obtain 104.5GFlops/Watt as a performance per power. The results of this paper showed about four times better performance per power compared to Tegra K1 of Nvidia

요 약

본 논문은 SIMT구조의 GPGPU에서 적은 core수로 고성능을 달성하기 위한 구조를 제안하고 설계하였다. 모바일기기에 적용하기 위한 GPGPU는 소모전력대비 성능을 높이기 위한 구조가 필수적이다. 소모전력을 줄이기 위해서 core수가 줄어든 대신 성능을 높이기 위해 thread를 관리하기 위한 warp scheduler의 size를 4로 하여 일반적인 GPGPU의 32 보다 크게 줄였다. Warp size를 적게 되면 pipeline의 idle cycle수를 줄일 수 있고 cache 메모리 접근시 miss penalty를 줄이기 위한 memory latency 적용이 효율적이다. 설계된 GPGPU는 부동소수점 연산을 포함하는 테스트 프로그램으로 연산 성능을 측정하고 28nm CMOS공정으로 소비전력을 측정하여 전력당 성능지수로 104.5GFlops/Watt를 얻었다. 본 논문의 결과는 Nvidia의 Tegra K1과 비교하였을 때 약 4배 우수한 전력당 성능지수를 보였다.

Key words : SIMT, GPGPU, Warp size, memory latency, FLOPS, Performance Per Watt

*Dept. of Computer Eng., Seokyeong University
★Corresponding author
E-mail : kylee@skuniv.ac.kr, Tel : +82-2-940-7745
※ Acknowledgment
Manuscript received Sep. 17, 2021; revised Sep. 23, 2021; accepted Sep. 23, 2021.
This work was supported by Seokyeong University in 2020.

Ⅰ. 서론

GP-GPU(General Purpose computing on Graphic Processing Unit)는 그래픽 처리 용도만이 아닌 CPU와 같이 범용 연산이 가능한 프로세서 형태로 발전되고 있다. GPGPU는 수백~수천개의 core구조로 되어 있어 4~16 core를 갖는 CPU에 비하여 병렬처리 연산에 매우 뛰어나 최근 인공지능, 메타버스, 비트코인, 암호화 등에서 활용도가 넓어지고 있다. 많은 수의 core를 병렬처리할 수 있는 것은

SIMT(Single Instruction Multiple Thread)기반에서 thread의 병렬실행이 가능하기 때문이다. Thread를 관리할 때 warp이라는 묶음을 단위로 scheduling하여 실행이 되는데 이때, thread 묶음의 크기가 warp size이며 size의 크기에 따라 warp scheduling 및 GPGPU의 성능이 달라진다. Warp size는 응용분야에 따라 장단점이 있지만 일반적으로 크기가 커지면 병렬의 수는 늘어나는 대신 불필요한 대기시간(idle cycle)이 늘어나고 core의 효율성을 떨어뜨릴 수 있다[1].

본 논문에서는 모바일 응용을 목적으로 GPGPU를 개발하였기 때문에 하드웨어 효율성을 위해 적은 수의 core를 사용하게 되며 따라서, warp size도 적은 수를 택하여 설계하였다. 작은 크기의 warp scheduler의 장점을 살리기 위해 scheduling 방법과 cache memory access 방법을 이용하여 성능을 최대화 하였다.

GPGPU 관련하여 가장 선도적인 기술을 보유하고 있는 Nvidia사의 GPGPU는 CUDA core라는 구조를 기반으로 하고 있으며 CUDA core의 warp size는 32 크기를 갖고 있다. 본 논문에서는 Nvidia사의 Tegra K1[2]과 소비전력과 부동소수점 연산능력을 비교 함으로써 설계된 작은 크기의 warp scheduler를 갖는 GPGPU의 성능의 우수함을 입증하였다.

II. 작은 크기 Warp Scheduler를 갖는 SIMT 구조

1. Thread 병렬처리 기술

프로그램을 thread로 처리하는 경우 process로 처리하는 것보다 병렬처리에 매우 용이하다. Thread는 Process내에서 동작되는 여러 실행의 흐름으로 그림 1과 같이 Process내의 주소 공간이나 자원(code, data, heap영역)을 thread끼리 공유하면서 실행된다.

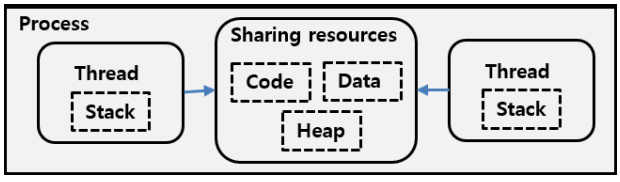

Fig. 1. Sharing resources of threads in the process.
그림 1. Process내 여러 thread들의 자원공유

Multi-Process는 그림 2와 같이 운영체제로 부터 process마다 독립적인 자원(code, data, heap영역)이 할당되기 때문에 자원의 낭비와 process변경시 Context Switching이라는 오버헤드가 발생하여 실행시간 지연 부담을 갖는다.

Thread는 Process에 비하여 적은 수의 명령어를 갖기 때문에 Process보다 많은 수의 thread가 발생하는 단점은 있지만 GPGPU와 같이 다수의 core를 이용하여 thread 병렬처리 수를 높이면 Muli-process가 갖는 단점을 보완하면서 thread 장점을 높일 수 있는 병렬처리 기술이다.

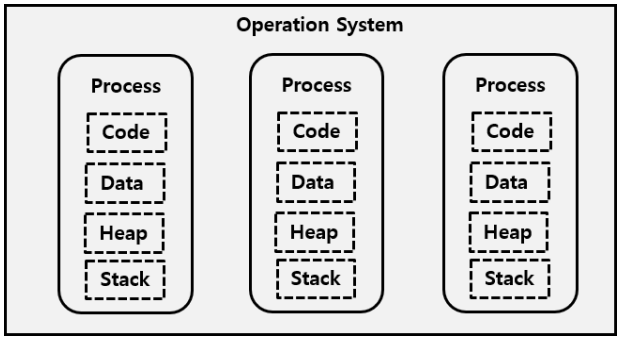

Fig. 2. Private resources of Multi-process.
그림 2. Multi-Process의 독립적인 자원

2. SIMT구조

Multi-thread는 Multi-core를 사용하여 공유자원의 갯수를 늘리게 되면 자원의 효율을 최대로 사용하며 thread와 thread의 전환에 필요한 지연시간이 없기 때문에 속도를 빠르게 할 수 있다.

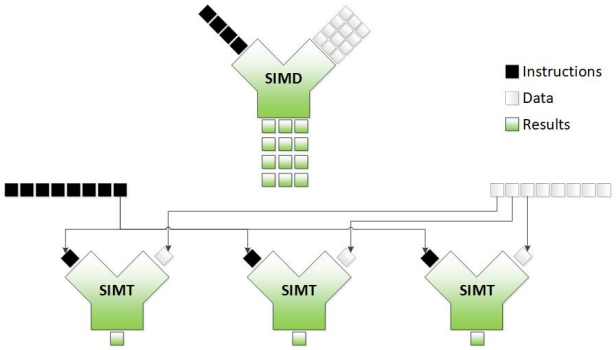

Fig. 3. Comparison of SIMD and SIMT.
그림 3. SIMD와 SIMT의 비교

GPGPU는 기존의 데이터 병렬처리 개념인 SIMD(Single Instruction Multiple Data)에 Multi-thread의 장점을 활용한 SIMT(Single Instruction Multiple

Thread) 기술을 사용한다. 그림 3은 SIMD와 SIMT의 차이를 설명한다. GPGPU의 SIMT는 명령어별로 각각 별도의 데이터를 사용하기 때문에 병렬처리에서 SIMD에 비하여 더욱 성능을 개선할 수 있다[3].

3. Thread단위 Warp Scheduler 설계

SIMT구조 GPGPU에서 Multi-thread를 관리하기 위해 Warp Scheduler를 두게 된다. Warp Scheduler는 Instruction Cache, Instruction Fetch, Stream Processor 사이에서 thread간의 자원충돌을 회피하기 위해 중재(arbitration) 역할을 하게 된다.

Warp Scheduler는 Host CPU로 부터 warp mask, thread mask 정보를 받아 어떤 warp이 활성화 되었는지 확인하고 활성화된 warp은 명령어 페치를 위한 PC(Program Counter)주소를 전달하고 이 주소는 명령어 캐시로 전달되어 실행 명령어를 페치하게 된다.

이때, warp scheduler는 여러 개의 thread를 묶음으로 하기 때문에 warp단위 활성화를 찾게 된다. Nvidia사의 GPGPU에서는 Fermi, Kepler 구조 모두 Threads/ Warp =32로 설계되어 있다.

Warp size는 idle cycle, coalescing rate, IPC (Instruction Per Cycle)에 영향을 주는데 선행연구[1]를 보면 그림 4와 같이 4개의 benchmark에서 warp size=8이 size=64에 비해 20% 이상 idle cycle이 개선되는 것을 볼 수 있다.

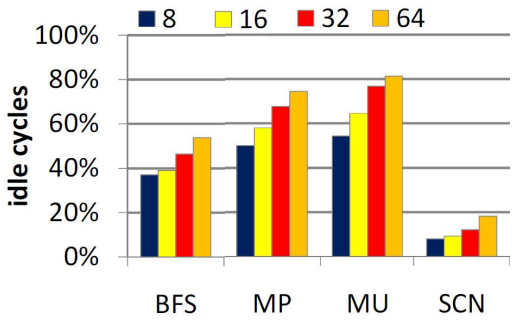

Fig. 4. Idle cycles for different warp sizes.
그림 4. warp크기에 따른 idle cycles

본 논문에서는 Threads/ Warp=4로 설계하여 warp size를 작게 하면서도 thread mask에 따라 thread 단위로 개별적 활성화를 통하여 다음과 같이 2가지 방법으로 GPGPU의 고성능화를 달성하게 된다.

- Pipeline idle cycles을 줄인다.
- Memory latency을 줄인다.

1) Pipeline idle cycles을 줄이는 설계

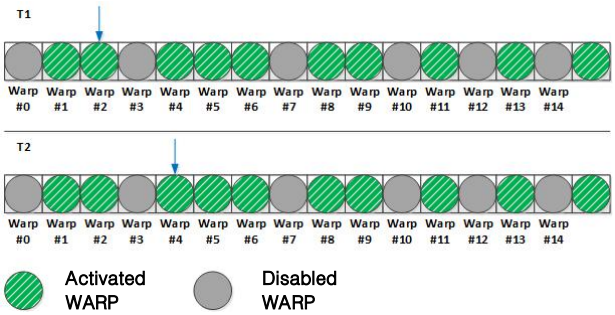

Fig. 5. Round Robin Warp Scheduler.
그림 5. Round Robin방식의 Warp Scheduler

Warp기반의 Multi-threading에서는 여러 개의 warp이 동시에 명령어 요청을 하게 되고 warp scheduler는 그중 하나를 선택하여 명령어 페치 유닛에 전달하게 되는데 본 연구에서는 활성화된 warp들을 라운드 로빈(Round Robin)방식으로 선택한다. 라운드 로빈 방식을 채택한 이유는 어느 한쪽의 warp만 연속해서 수행하게 될 때 해당 warp에 포함된 thread 간에 dependency가 존재할 때 stream process내부 pipeline stall에 의한 idle cycle이 발생한다. 이에 대한 해결방안은 그림 5와 같이 idle 발생이 감지되면 해당 warp은 비활성화 되고 warp scheduler는 라운드 로빈 방식으로 다음 활성 warp으로 명령어 페치 권한을 넘기게 된다[4].

2) Memory latency을 줄이는 설계

메모리 접근 명령어가 처리될 때 캐시 miss시 stream processor의 pipeline stall이 발생하며 이때 miss penalty를 처리하기 위한 cycle수는 경우에 따라 수백 cycle이 될 수 있다. 본 논문에서는 이러한 문제를 작은 크기의 warp scheduling으로 그림 6과 같이 해결하였다.

한 warp의 쓰레드중 캐시 miss가 발생하면 데이터가 hit 될 때까지 그 warp은 대기열에 대기하고 다음 warp의 메모리 요청을 처리한다. 대기열에는 아직 thread의 miss가 처리되지 않은 warp들이 대기하고 있다. 다른 warp의 명령어를 처리하는 중에 대기하고 있는 warp들을 라운드 로빈 방식으로 해당 warp의 thread가 hit 되었는지 검사한다. Thread의 miss가 모두 처리된 warp은 write back 모듈로

보내진다. 한 Warp에서 Miss가 발생해도 다음 Warp의 명령어를 처리하므로 파이프라인이 정지되지 않고 연속해서 수행될 수 있다. 또한, 긴 싸이클을 소모하는 메모리 접근 지연 시간을 숨길 수 있다. 본 연구의 작은 크기 warp가 달리 큰 크기의 warp인 경우 miss시 대기시간이 길어져 memory latency를 숨기지 못하는 경우가 발생하여 전체적으로 GPGPU의 성능 저하를 막지 못하게 된다[5][6].

III. Perpormance Measurement

SIMT구조 GPGPU에서 warp size를 작게하여 얻어지는 성능 이득을 입증하고 큰 size의 warp을 갖는 Nvidia사의 GPGPU와 비교하고자 한다. 부동소수점 연산능력과 소비전력 대비 성능을 기준으로 측정하고 비교한다.

1. 부동소수점 연산능력(FLOPS)

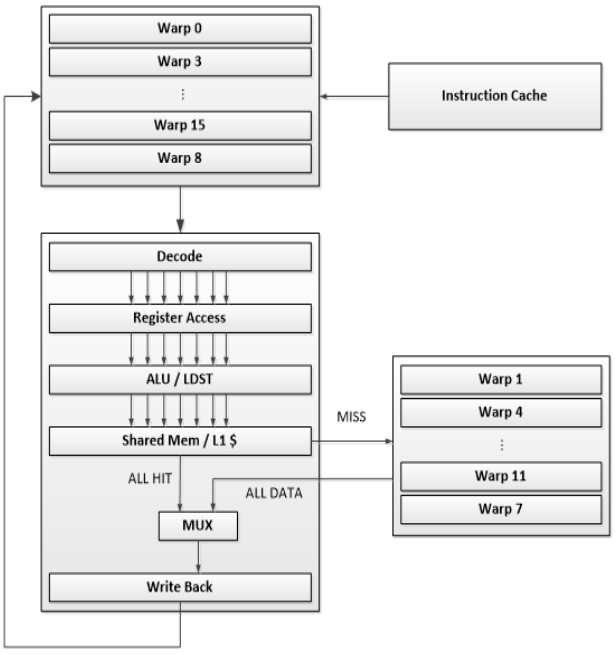

Fig. 6. Warp scheduling for reducing memory latency.
그림 6. 메모리 지연 감소를 위한 warp scheduling

부동소수점 연산능력은 부동소수점(Floating Point Operation)만으로 구성된 프로그램 실행 시 단위 시간(sec)당 수행되는 부동소수점명령어의 연산 Operation 수를 측정하며 파이프라인 클럭 사이클당 연산 Operation수에 실행 주파수를 곱하여 얻는다.

테스트 프로그램은 표 1과 같이 get_tid()함수를 이용하여 thread 번호(thread identification : tid)를 초기 주소값으로 받아 Buffer메모리에 있는 값을 '1'씩 증가 하는 과정을 반복하는 부동소수점 덧셈 능력 측정과 Buffer메모리에 있는 값을 '3.0'씩 곱셈 하는 과정을 반복하는 부동소수점 곱셈능력을 측정한다.

Table 1. Floating Point Test Program.
표 1. 부동소수점 테스트 프로그램

(a)Test Program for Floating Point Addition

```
int main( int argc, char** argv ){
        float* pBuffer = (float*)argv;
        int tid = get_tid();
        float a=0;
        float tt = 1;
        a = pBuffer[tid] + tt;
        a += tt; //loop
        pBuffer[0] = a;    }
```

(b)Test Program for Floating Point Multiplication

```
int main( int argc, char** argv ){
        float* pBuffer = (float*)argv;
        int tid = get_tid();
        float a = 0;
        a = pBuffer[tid] * 1.0;
        a *= 3.0; //loop
        pBuffer[0] = a;    }
```

테스트 프로그램을 GPGPU RTL(VerilogHDL)수준 코드로 작성하여 TestDrive Profiling Master에서 시뮬레이션 진행 후 표 2의 결과를 확인 하였다.

Table 2. Measurement results of FLOPS.
표 2. 부동소수점 연산능력 측정결과

Measurement Results
- Outputs result of an addition(fadd, faddi) or a multiplication (fmul, fmuli) per clock cycle (Applied pipeline structure) - One core of GPGPU consists of 16 ALUs, and 16 ALUs per clock cycle are simultaneously calculated and output. - When a GPGPU core is operated in 500 MHz clock, the floating point addition calculation amount for 1 second. = 16(ALU) x 500MHz(the number of clock cycle per second) = 8G Flops

2. 소비전력 대비 성능 측정

전력은 크게 정적전력(static power)과 동적전력(dynamic power)로 구성된다. 정적전력은 트랜지스터가 switching(turn on or turn off) 동작을 하지 않을 때 여러 경로로 소모되며 대부분은 source-to-drain subthreashold leakage 이다. 또한, diffusion layer와 substrate 사이에서 누설되는 전류가 요인이 된다. 결국, 정적전력은 누설전력(leakage power)이라고 한다.

동적전력은 회로가 동작할 때 소모되는 전력으로 switching power와 internal power로 구성된다. 회로에서 소모되는 총전력은 위 성분을 합하게 되며 다음과 같은 식 (1)로 계산된다.

$$P_{\text{total}} = P_{\text{leakage}} + P_{\text{switching}} + P_{\text{internal}} \quad (1)$$

소모전력 측정은 GPGPU core의 Verilog HDL 코드를 Synopsys 합성도구툴로 RTL회로 생성하고 생성된 RTL 회로를 Global Foundries의 CMOS 28nm공정(GF28SLP)을사용한 ASIC라이브러리로 변환한뒤 Synopsys 툴을 이용한 측정하였고 표 3과 같은 결과를 얻었다.

Table 3. Measurement result of Power consumption.
표 3. 소비전력 측정결과

$$P_{\text{total}} = P_{\text{leakage}} + P_{\text{switching}} + P_{\text{internal}}$$
$$= 116.5567\text{uW} + 4.8443\text{mW} + 71.5197\text{mW}$$
$$= \underline{\mathbf{76.4791\ mW}}$$

위 결과를 이용하여 소비전력 대비 성능을 계산하면 다음 식(2)와 같다.

$$GFLOPS/Watt = 8\,GFLOPS/76.5mW = 104.5\,GFLOPS/Watt \quad (2)$$

3. 모바일 GPGPU 성능 비교

Nvidia사는 GPGPU의 근간이 되는 Tesla구조를 기반으로 Kepler, Maxwell 등으로 GPGPU를 발전시켜 가장 고성능의 GPGPU 기술을 선도하고 있다. Nvidia의 GPGPU는 CUDA라고 하는 병렬처리 프로그래밍 플랫폼과 stream processor core 하드웨어 플랫폼을 기반으로 GPGPU를 설계한다[7].

이 가운데 Nvidia의 모바일 GPGPU로 Kepler구조를 기반으로 하는 Tegra K1과 본 논문에서 개발한 GPGPU의 성능을 비교하면 다음 표 4와 같다.

Table 4. omparison of Performance Per Watt.
표 4. 전력당 성능 비교

Items	Tegra K1	Ours
Warp Size	32	4
GPU cores	192	16
Process	28nm	28nm
Core Frequency	2.5GHz	500MHz
FLOPS	290GFlops	8GFlops
Power	11Watt	0.765mWatt
Performance Per Watt	26GFlops/W	104.5GFlops/W

Ⅳ. Conclusion

본 논문은 모바일기기에서 그래픽 및 인공지능 처리에 적합한 GPGPU를 설계하였으며 전력소모와 직결되는 하드웨어 크기를 최소화 하기위해 core수를 줄인 구조를 제안하였다. Core수를 줄이고 병렬처리를 강화하기 위해서는 SIMT구조에서 핵심이 되는 warp의 size를 적게 채택하였는데 size가 적으면 동시에 진행되는 thread개수가 적어 병렬처리 성능은 떨어질 수 있다.

그러나 본 논문에서는 pipeline cycle 지연을 줄이고 memory latency를 줄임으로써 적은 수의 core에 비하여 성능을 높일 수 있어 비교 기술인 Nvidia Tegra K1에 비하여 전력대비 성능이 약 4배 증가하였다.

References

[1] Ahmad Lashgar, A. Baniasadi, & A. Khonsari. “Investigating Warp Size Impact in GPUs,” *Computer Science,* 2012. ArXiv:1205.4967.

[2] Gaurav Mitra, Andrew Haigh, Luke Angove, Anish Varghese, “Experiences Using Tegra K1 and X1 for Highly Energy Efficient Computing,” *GTC 2016,* 2016. DOI: 10.1109/HPCSim.2016.7568401

[3] Kwan Ho Lee, Chi Yong Kim, “A Design of a High Performance Stream Processor without Superscalar Architecture,” *J.inst.Korean. electr. electron.eng,* Vol.21, No.1, pp.77-80, 2017. DOI: 10.7471/ikeee.2017.21.1.77

[4] Kwan Ho Lee, Chi Yong Kim, “Thread Distribution Method of GP-GPU for Accelerating

Parallel Algorithms," *J.inst.Korean. electr. electron. eng*, Vol.21, No.1, pp.92-95, 2017.
DOI: 10.7471/ikeee.2017.21.1.92
[5] Wilson W. L. Fung, Ivan Sham, George Yuan, Tor M., "DynamicWarp Formation and Scheduling for Efficient GPU Control Flow," *40th Annual IEEE/ACM International Symposium(MICRO 2007)*, pp.407-420, 2007.
DOI: 10.1109/MICRO.2007.30
[6] Kwang Yeob Lee, "Implementation of a Memory Operation System Architecture for Memory Latency Penalty Reduction in SIMT Based Stream Processor," *J.inst.Korean. electr. electron.eng,* Vol.18, No.3, pp.392-397, 2014. DOI: 10.7471/ikeee.2014.18.3.392
[7] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with CUDA," *IEEE Micro*, Vol.28, no.4, pp.13-27, 2008. DOI: 10.1109/MM.2008.57

BIOGRAPHY

Lee Kwang-yeob (Life Member)

1985 : BS degree in Electronics Engineering, Sogang University.
1987 : MS degree in Electronics Engineering, Yonsei University.
1994 : PhD degree in Electronics Engineering, Yonsei University.
1989~1995.2 : Senior Researcher, Hyundai Electronics Inc.
1995.3~present : Professor, Dept. of Computer Engineering, Seokyeong University.