논문번호 12-04-05
28

Journal of IKEEE.Vol.16,No.4,310~315,December 2012
http://dx.doi.org/10.7471/ikeee.2012.16.4.310

GPGPU를 위한 쉐이더 명령어기반 멀티 스레드 관리 기법

Method of Multi Thread Management based on Shader Instruction for Mobile GPGPU

이 광 엽*, 박 태 룡**

Kwang-Yeob Lee*,Tae-Ryong Park**

Abstract

This thesis is intended to design multi thread mobile GPGPU optimized in mobile environment, and to verify an effective thread management method of the multi thread mobile processor. In thread management, there is no management hardware and implement with software instructions. For the verification of the multi thread management method, Lane detection algorithm was implemented to compare nVidia's CUDA Architecture and the designed GPGPU in terms of thread management efficiency. The number of thread is normalized to 48 threads. An implemented Land Detection Algorithm is composed of Gaussian filter algorithm and Sobel Edge Detection algorithm. As a result, the designed GPGPU's thread efficiency is up to 2 times higher than CUDA's thread efficiency.

요 약

본 논문에서는 모바일환경에 최적화 된 멀티 스레드 모바일 GPGPU를 설계하고, 멀티 스레드로 구성된 모바일 프로세서에서의 효과적인 스레드 관리 기법을 검증한다. 스레드의 제어에는 별도의 하드웨어 없이 소프트웨어 명령어를 기반으로 설계하였다. 스레드 관리 기법의 검증은 차선 검출 알고리즘을 구현하여 nVidia의 CUDA Architecture와 설계한 GPGPU의 스레드 관리 효율을 비교 분석한다. 스레드 효율에서는 CUDA와 비교했을 때 최대 2배까지 높은 효율을 보인다.

Key words : GPGPU; Multi Thread; Thread Management; Parallel Processing

Ⅰ. 서론

최근 모바일 프로세서의 성능이 향상됨에 따라 다양한 임베디드 장치와 그에 대한 어플리케이션 수요 및 개발이 늘어나고 있다.

모바일 프로세서의 CPU분야에서는 SAMSUNG의 Exynos, Qualcomm의 Snap Dragon등이, GPU분야에서는 Imagination의 PowerVR, ARM의 Mali등이 High Capacity를 보이는 대표적인 High-end 모바일 프로세서이다.

이러한 모바일 프로세서들의 성능향상에는 높은 동작 주파수를 갖도록 설계하는 방법도 있지만, 멀티 스레드 또는 멀티 코어로 설계하여 어플리케이션의 수행을 분배하여 실행할 수 있도록 하여 프로세서에 걸리는 부하를 줄이는 방법도 있다.

*Professor, Dept. of Computer Engineering, Seokyeong University
kylee@skuniv.ac.kr
**Professor, Dept. of Computer Engineering, Seokyeong University, Corresponding author
This work was sponsored by ETRI SW-SoC R&BD Center, Human Resource Development Project.
Manuscript received Aug. 27. 2012; revised Oct. 24. 2012 ; accepted Nov. 6, 2012

본 논문에서는 멀티 코어, 멀티 스레드로 구현한 GPGPU에 병렬 영상 처리 알고리즘을 적용하여 GPGPU의 효율적인 스레드 관리 기법을 제시한다.

II. CUDA의 스레드 관리 기법

CUDA는 nVidia의 PC기반 GeForce 3D 그래픽스 카드의 SM(Streaming Multi-processor)들을 활용하여 일반적인 응용 프로그램의 처리 및 가속을 가능하도록 하는 GPGPU 기술이다.

CUDA에서는 SIMT (Single Instruction, Multiple Thread)라는 하나의 명령어로 여러 개의 스레드를 동시에 구동시키는 명령어 구조를 사용하는데, 이 명령어는 다수의 스레드를 제어하는 장점과 동시에 다음과 같은 단점이 있다.

그림1 과같이 분기문을 실행하는 경우에는, if의 조건을 만족하여 명령어 1~3의 명령어를 실행하는 스레드들은 수행이 끝난 후에 else의 조건을 만족하여 명령어 4~5를 수행하는 스레드의 수행을 기다리지 않고서는 명령어 6을 수행할 수 없다.

이 문제는 CUDA의 SM과 내부 스레드들을 하나의 명령어로 제어하는 대신에 모든 스레드들이 동기화 되어야 분기문 이후의 명령어를 수행할 수 있기 때문에 분기문의 전체 실행시간을 예측할 수 없어서 발생한다.

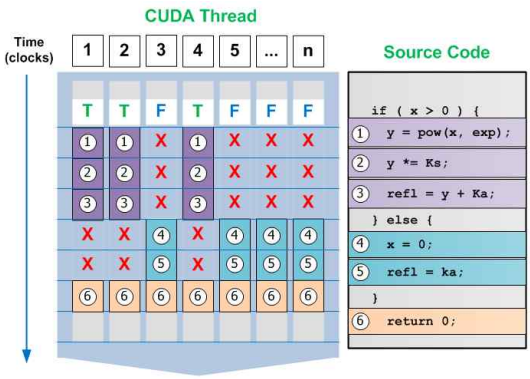

Fig 1. Flow of Conditional statement on CUDA
그림 1. CUDA의 분기문 실행 흐름도

그림1 에서 if 에 해당하는 코드가 3사이클을 소모하고, else 에 해당하는 코드가 2사이클을 소모한다면 이 분기문을 완전히 수행하기 위해서는 모든 스레드가 최소한 5사이클 이상을 소모하게 된다.

만약 여기서 else if에 의한 분기문의 분기 수가 늘어나거나, 분기문 내부에 실행할 명령어가 많아지는 경우에는 모든 스레드는 더 많은 시간을 다른 스레드의 수행을 기다리기 위해 낭비하는 것이다.

CUDA는 이러한 문제를 최소화 하기위해 워프(Warp)라는 스레드 그룹 할당을 통해 1 워프당 최대 36개 이하의 스레드를 사용할 것을 권하고 있다.

III. Meitner GPGPU

Meitner는 GPGPU기능을 탑재한 모바일 그래픽 프로세서이다. Meitner의 구조는 1개의 코어 당 12개의 스레드를 갖는데, 본 논문에서는 이를 4개의 코어를 갖는 멀티코어 구조로 총 48개의 스레드로 구성하도록 하였다.

주요 특징으로는 한 번에 2개의 명령어를 동시에 실행하는 듀얼 페이즈 가변구조 명령어 구조, 스레드 제어 하드웨어가 없이도 싱글 / 멀티 스레드 간의 전환 및 프로그램 작성이 용이하도록 다양한 분기 명령어를 통해 스레드의 실행을 제어할 수 있는 구조를 가지고 있다.

내부 레지스터 구조는 하나의 레지스터가 128bit로 구성되어 있고 이를 다시 4개의 32bit 컴퍼넌트로 분리하여 사용할 수 있는 스레드 레지스터가 스레드당 128개, 그리고 하나의 코어 내부에서 12개의 스레드가 서로 공유할 수 있는 코어 공유 레지스터가 2042개로 구성되어 있다.

코어 공유 레지스터도 스레드 레지스터와 같은 128bit 길이를 갖는 32bit의 4개 컴퍼넌트 구조로 되어 있다.

IV. Meitner의 스레드 관리 기법

Meitner도 CUDA와 마찬가지로 여러 개의 스레드를 하나의 명령어로 제어할 수 있다. 하지만 다음 명령어를 수행하기 위해서 모든 스레드가 종료되기를 기다리는 CUDA와 다르게 Meitner는 DTSP (Dynamic Threading on Single Program) 구조를 통해 각 스레드가 독립적으로 프로그램을 수행할 수 있다.

그림2 는 그림1 의 분기문을 Meitner에서 수행했을 때의 스레드들의 동작 상태를 보여준다.

각각의 스레드는 하나의 동일한 프로그램 소스코드에서 분기문을 만날 경우, 별도의 동기화 명령어가 있기 전까지는 자신이 수행해야 할 코드만을 다른 스레드의 수행을 기다리는 시간 없이 처리하도록 되어 있다.

만약 스레드 간에 동기화가 필요한 상황이 발생하면 별도의 스레드 제어기의 제어 없이 스레드가 직접 다른 스레드에게 자신의 Run / Idle 상태의 신호를 알려주고 스레드간에 이를 감지하여 동기화 타이밍을 조절할 수 있도록 설계하였다.

이러한 DTSP 구조에서는 각 스레드는 자신이 수행해야 할 명령어을 수행하며 분기문의 수행시간 예측이 가능해지고 별도의 동기화 명령어를 통해 스레드를 멈추지 않는 경우에는 CUDA와 같이 분기문을 수행할 때에 다른 스레드의 수행시간을 기다리지 않아도 된다.

따라서 CUDA의 분기문에 의한 명령어 실행 동기화와 그에 따른 성능저하가 Meitner에서는 발생하지 않는다는 것이다.

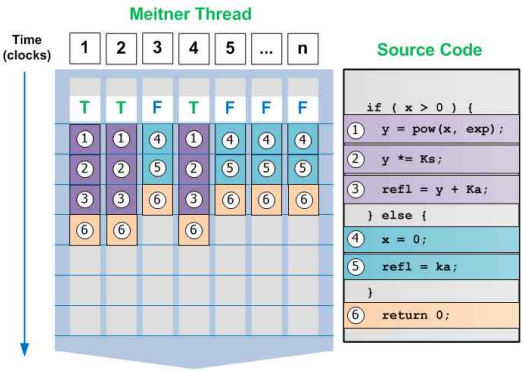

Fig 2. Flow of 분기문 on Meitner
그림 2. Meitner의 분기문 실행 흐름도

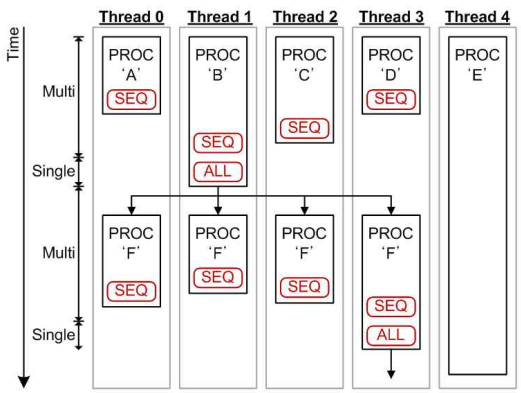

Fig 3. Thread Operation by Control Instruction
그림 3. 제어명령어에 의한 스레드 동작

그림3 은 DTSP 구조에서 명령어를 통해 스레드들이 싱글 스레드 시퀀스와 멀티 스레드 시퀀스로 전환되는 모습을 보여준다.

프로그램 초기에 각각의 스레드 0~4는 프로그램에서 자신이 수행해야할 Function A~D를 'SEQ' 명령어를 통해 Multi 스레드로 개별 수행한다. 'SEQ' 명령어에 해당하는 Function의 수행이 끝나고 동기화 명령어인 'ALL' 명령어를 수행하는 순간부터 모든 스레드 들은 동기화를 위해 자신의 상태를 다른 스레드들에게 알리고 idel 상태로 진입하게 된다.

가장 늦게 'ALL' 명령어를 수행하는 스레드 1은 다른 스레드 들이 idle 상태인 것을 확인하고 하나의 Function F를 병렬로 처리하기 위한 'SEQ' 명령어를 통해 다른 스레드들에게 실행 명령을 넘겨주게 된다. 이 과정에서 스레드 4 는 독립적으로 Function E를 수행하며 다른 스레드의 동기화 명령을 받지 않는다.

Ⅴ. 차선 검출 알고리즘

차선 검출 알고리즘은 차량의 도로 주행영상에서 차선 성분을 찾아내는 알고리즘이다.

이 알고리즘은 그림4 와 같이 입력받은 도로 주행영상에 Gaussian Filter를 세로, 가로의 순서로 적용한 후에 Sobel Edge 검출 알고리즘을 통해 이진화 영상을 만든 후에 차선 영역을 검출 하는 방식으로 진행된다.

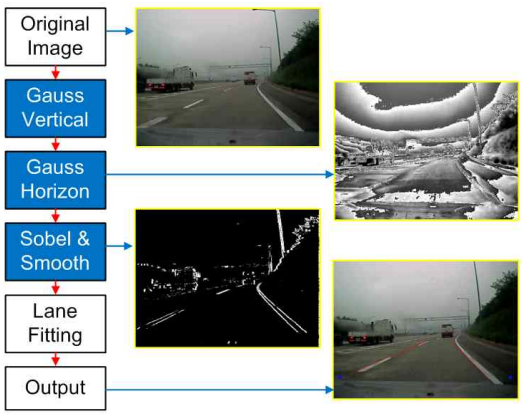

Fig 4. Flow of Lane Detection Algorithm
그림 4. 차선 검출 알고리즘의 흐름도

본 논문에서는 차선 검출 알고리즘을 CUDA와 Meitner로 구현하였다. 구현에 있어 CUDA는 CUDA 개발툴킷을 활용하여 C로 작성하였고, Meitner는

Meitner의 어셈블리 언어로 작성했다.

수행 순서는 YUV로 입력되는 블랙박스 주행영상에서 CPU는 Y값을 읽어서 GPGPU로 전달하고, GPGPU에서는 Gaussian Filter를 세로, 가로의 순서로 적용한다. Gaussian Filter를 적용한 Y값 영상에 Sobel Edge 검출 알고리즘을 이용해 외곽선을 갖는 영역을 검출 한 후 Binary 영상으로 변환하여 CPU에게 전달하면, CPU가 이를 기반으로 하여 현재 차량이 주행 중인 차선을 판별하고 이에 대한 차선 이탈 경보를 수행하도록 구성했다.

Ⅵ. 병렬 처리 기법

Gaussian Filter의 경우 가로와 세로로 대상 픽셀의 좌, 우측 각각 2개의 픽셀로 총 5개 픽셀을 필요로 하고, Sobel & Smooth Filter의 경우에는 가로로 대상 픽셀의 좌, 우 총 2개의 픽셀을 필요로 한다. 이렇게 대상 픽셀의 인접 픽셀 값을 참고하여 Filter의 연산을 통해 대상 픽셀의 값을 결정하는데, 같은 방향으로 한 번 불러들였던 픽셀들을 재사용하여 다음 픽셀의 연산에 사용하도록 되어있다.

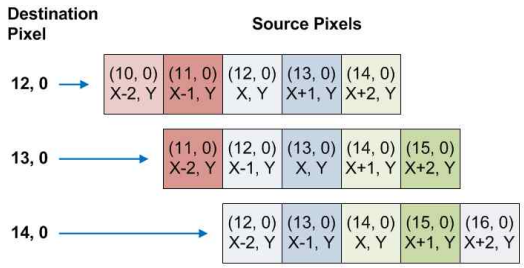

Fig 5. Pixel Reusing in Gaussian Filter
그림 5. 가우시안 필터의 픽셀 재사용

그림5 는 수평 Gaussian Filter에서 같은 Y좌표상에서 X좌표가 인접해 있는 픽셀들의 연산에 필요한 픽셀들을 표시하여 이러한 픽셀 재사용에 대한 예를 보여준다.

Filter의 적용 시에 이러한 픽셀의 재사용은 대상 이미지의 로드, 각 픽셀의 연산, 픽셀 값 저장, 이미지 저장에 걸쳐 무수한 메모리 접근을 필요로 한다.

따라서 이미지 처리에 대한 병렬화 수준을 너무 높이는 경우에는 메모리 대역폭 이상의 데이터 교환이 일어나 병렬화된 프로그램의 수행속도를 오히려 느리게 만드는 결과를 초래하게 된다.

이 문제를 해결하기 위해 각 Filter의 적용에 있어서 픽셀 재사용 단위를 크게 나누어 각각의 스레드가 픽셀 재사용이 가능한 스캔라인 단위로 픽셀을 불러들이고 연산하여 저장한다면, 메모리 연산에 대한 수행 속도를 최적화 할 수 있다.

그래서 하나의 스레드가 해당 Filter마다 가로 또는 세로 방향으로 한 라인을 구성하는 가로 - 640 픽셀, 세로 - 480 픽셀 단위로 처리하도록 설계하였다.

Ⅶ. 실험 및 분석

실험은 두 가지 GPGPU에서 차선 검출 알고리즘을 위한 Gaussian Filter (세로, 가로)와 Sobel Edge 검출 알고리즘을 구동했을 때에 수행시간을 바탕으로 진행된다.

CUDA의 경우 SM 내부의 블럭을 최대 36 스레드 이하의 단위로 사용하도록 권고하고 있으므로, 스레드 수를 늘려감에 따라 1 블럭 당 12 스레드씩 생성하여 실험한다.

실험을 위하여 CUDA는 Geforce 550Ti Graphics Card를 사용하였다. Meitner는 Xilinx Inc.의 Virtex 6 ML605 Board에 FPGA로 구현하고, CUDA와 Meitner를 i7-2600 3.4GHz CPU를 탑재한 x86 PC Main Board의 PCI Express에 연결하여 검증 소

Table 1. Specification of each GPGPUs
표 1. 각 GPGPU의 사양

-	CUDA	Meitner
# of Core	192	4
Frequency	900 MHz	100 MHz
Bandwidth	192 bit	64 bit
Memory	DDR5-2050 MHz	DDR3-1066 MHz
Mem. Size	1024 MB	256 MB
PCI-E	PCI-E 2.0 x16	PCI-E 2.0 x4

Table 2. Average Execution Time and Efficiency
표 2. 평균 수행시간과 효율

스레드	CUDA @900MHz		Meitner @100MHz	
	수행 시간 (sec.)	효율 (times)	수행 시간 (sec.)	효율 (times)
1	1.62	1	13.25	1
12	0.46	3.52	1.72	7.70
24	0.25	6.48	1.14	11.62
36	0.17	9.53	0.89	14.89
48	0.12	13.50	0.78	16.99

프트웨어의 구동결과를 바탕으로 진행한다.

표1 은 각 GPGPU의 간략한 스펙을 나타낸다.

표2 는 Gaussian Filter (세로, 가로)와 Sobel Edge 검출 알고리즘을 10 프레임 수행할 때에 각 GPGPU의 스레드 수 별로 평균 수행 시간과 스레드가 늘어남에 따른 효율을 나타낸다. 여기서 times 로 표시되는 효율는 각 GPGPU의 1개 스레드로 수행했을 때보다 멀티 스레드를 활용했을 때 얼마나 시간이 단축되는지를 기준으로 한다.

표2 의 결과에서 전체적인 수행 시간은 900MHz로 구동되는 CUDA가 100MHz로 구동되는 Meitner보다 빠른 수행 시간을 보이지만, 스레드가 늘어남에 따른 효율은 Meitner가 앞서는 것을 알 수 있다.

CUDA는 스레드 수가 늘어남에 따라 1 스레드에 비해 약 300% 포인트 수준의 고정적인 성능의 향상이 있고, Meitner는 모바일 환경을 목표로 하여 스레드의 수가 더 많이 늘어나는 것보다 작은 사이즈에서 더 좋은 효율을 낼 수 있도록 설계되어 CUDA보다 효율은 높지만 스레드 수가 늘어남에 따른 성능 향상의 폭은 점차 낮아지는 경향을 보인다.

그림6 은 표2 에서 측정한 성능을 바탕으로 스레드 숫자가 늘어남에 따른 기대성능과 늘어나는 스레드 수에 따른 실제성능의 비교이다. 이 때 기대성능은 스레드의 수가 늘어남에 따른 성능 향상 폭이 스레드 수의 배수라고 정의한다.

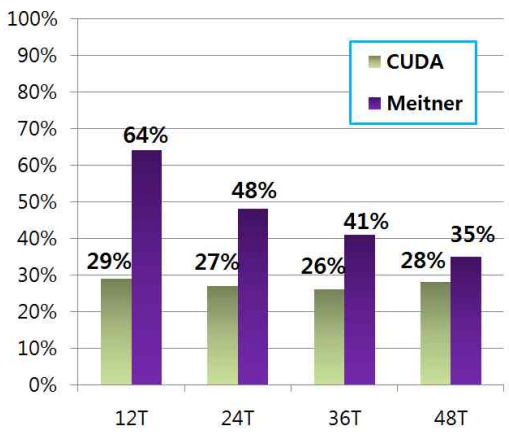

Fig 6. Comparison of Expected Power and Real Power
그림 6. 기대성능과 실제성능의 비교

그림6 에 따르면 12스레드에서 실제성능은 Meitner는 기대성능의 64%, CUDA는 29%로 Meitner가 CUDA보다 기대성능에 2배 이상 가까운 성능을 보여주고 있다. 이는 작은 SoC가 요구되는 모바일 환경에 적합하다는 결론의 근거가 된다.

CUDA가 스레드 수의 증가에 따른 비교적 고정적인 실제성능을 보이는 반면 Meitner의 스레드가 더 늘어남에 따라 실제성능이 낮아지는 것은 현재 Meitner가 64bit PLB 버스 인터페이스를 사용하고 있기 때문으로 보인다. 추후 256bit의 AXI 버스 인터페이스나 개선된 버스 인터페이스의 적용이 될 경우 고정적인 성능 향상을 보일 것으로 예상된다.

Ⅷ. 결론

이론적으로 CUDA는 스레드를 3천만 개 이상 만들어 사용할 수 있다. 하지만 이는 전력 소모와 발열량을 크게 고려하지 않는 PC환경에서 가능한 설계기법으로 모바일 환경에는 적합하지 않다.

본 논문에서 개발 중인 Meitner는 3D Graphics와 GPGPU의 기능을 수행할 수 있는 Mobile GPGPU로서 적은 전력 소모와 작은 SoC 사이즈를 통해 제한적인 환경의 GPGPU 기능수행에 있어서 CUDA보다 높은 효율을 보이는 것으로 나타났다. 본 논문에서 수행한 실험의 비교대상에 따라 다른 결과를 보일 수 있으므로 차후 다양한 모바일 GPU와의 성능 비교 실험을 요구한다.

본 논문에서 나타난 Meitner의 장점과 단점을 보완하여 Mobile GPU 분야에서도 GPGPU 기술의 발전과 활용도가 높아지기를 기대해본다.

References

[1] HyungKi Jeong, "A Design of a Multi-Threaded & Multi-Core GP-GPU using a dynamic Thread management techniques", The Graduate School of Seokyeong Universiry, August 2010.

[2] NVIDIA, "NVIDIA CUDA C Programming Guide", November 2011.

[3] NVIDIA, "CUDA API REFERENCE MANUAL", January 2012.

[4] Hongfei Yu, Wei Liu, Jianghua Pu, Bobo Duan, Huai Yuan, Hong Zhao, "Lane recognition based on location of raised pavement markers", Intelligent Vehicles Symposium (IV), July 2011.

[5] Shih-Shinh Huang, Chung-Jen Chen, Pei-Yung Hsiao, and Li-Chen Fu, "On-Board Vision System for Lane Recognition and Front-Vehicle Detection to Enhance Driver's Awareness", IEEE Intermtbnal confennu on Robdm a Automlltlon, April 2004.

BIOGRAPHY

Lee Kwang Yeob (Life member)

1985. 8 Seogang University, Dept. of Electronics Engineering(BS)
1987. 8 Yonsei University, Dept. of Electronics Engineering(MS)
1994. 2 Yonsei University, Dept. of Electronics Engineering(Ph.D)
1989~1995. 2 Hyundai Electronics Inc., Senior Researcher
1995.3~ Seokyeong Univeristy, Dept. of Computer Engineering, Professor
<Research interests> Microprocessor, Embedded System, 3D Graphics System

Park Tae Ryong (member)

1985 : Hangyang University, Dept. of Mathmatics(BS)
1987 : Hangyang University, Dept. of Mathmatics(MS)
1995 : Hangyang University, Dept. of Mathmatics(Ph.D)
1994~ : Seokyeong Univeristy, Dept. of Computer Engineering, Professor
<Research interests> Crypto Algorithm, Computer Security, Computer Arithmetic