GPGPU의 멀티 쓰레드를 활용한 고성능 병렬 LU 분해 프로그램의 구현☆

Implementation of high performance parallel LU factorization program for multi-threads on GPGPUs

신 봉 희* 김 영 태**
Bong-Hi Shin Young-Tae Kim

요 약

GPGPU는 원래 그래픽 계산을 위한 프로세서인 GPU를 일반 계산에 활용하여 저전력으로 고성능의 효율을 보이는 신개념의 계산 장치이다. 본 논문에서는 GPGPU에서 계산을 하기 위한 병렬 LU 분해법의 알고리즘을 제안하였다. Nvidia GPGPU에서 프로그램을 실행하기 위한 CUDA 계산 환경에서는 계산하고자 하는 데이터 도메인을 블록으로 나누고 각 블록을 쓰레드들이 동시에 계산을 하는데, 이 때 블록들의 계산 순서는 무작위로 진행이 되기 때문에 블록간의 데이터 의존성을 가지는 LU 분해 프로그램에서는 결과가 정확하지 않게 된다. 본 논문에서는 병렬 LU 분해법에서 블록간의 계산 순서를 인위적으로 정하는 구현 방식을 제안하며 아울러 LU 분해법의 부분 피벗팅을 계산하기 위한 병렬 reduction 알고리즘도 제안한다. 또한 구현된 병렬프로그램의 성능 분석을 통하여 GPGPU의 멀티 쓰레드 기반으로 고성능으로 계산할 수 있는 병렬프로그램의 효율성을 보인다.

ABSTRACT

GPUs were originally designed for graphic processing, and GPGPUs are general-purpose GPUs for numerical computation with high performance and low electric power.In this paper, we implemented the parallel LU factorization program for GPGPUs. In CUDA, which is computational environment for Nvidia GPGPUs, domains are divided into blocks, and multi-threads compute each sub-blocks simultaneously. In LU factorization program, computation order should be artificially decided due to the data dependence. To resolve the data dependancy, we suggested a parallel LU program for GPGPUs, and also explained parallel reduction algorithm for partial pivoting of LU factorization. We finally present performance analysis to show efficiency of the parallel LU factorization program based on multi-threads on GPGPUs.

☞ keyword : GPGPU, CUDA, LU, SIMT

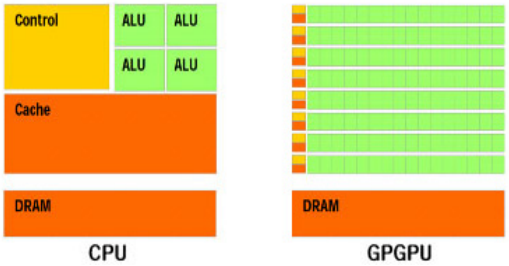
1. 서 론

LU 분해 프로그램은 연립방정식의 해를 구하는 프로그램으로서 가장 많이 사용하는 수치 계산 프로그램의 하나이다. 알고리즘의 복잡성이 2/3n3로서 많은 계산을 필요로 하기 때문에 고속의 계산이 필수적이다[1-3].

GPGPU(General Purpose Graphic Processing Unit)는 원래 그래픽을 처리하기 위한 프로세서인데 5~6년 전부터 격자 구조의 프로세서들이 일반 수치 계산에도 효율적으로 사용이 되고 있으며 저전력 기반의 신개념 계산 장치로서 많은 주목을 받고 있다[4,5].

본 논문에서는 GPGPU에서 계산을 할 수 있도록 Nvidia CUDA(Computer United Device Architecture)를 사용한 병렬 LU 분해 프로그램을 제안하였다.

* 정 회 원 : 인천대학교 컴퓨터공학부 교수
bhshin@incheon.ac.kr
** 정 회 원 : 강릉원주대학교 컴퓨터공학과 교수
ykim@kangnung.ac.kr
[2010/11/10 투고 - 2010/11/19 심사(2011/02/14 2차) - 2011/03/16 심사완료]
☆ 이 논문은 인천대학교 2010년도 자체연구비 지원에 의하여 연구되었음



(그림 1) CPU와 GPGPU의 비교

CUDA 계산 환경에서는 전체 계산 도메인인 그리드를 블록으로 나누고 각 블록을 여러 쓰레드들이 동시에 계산한다. 이 때 블록의 계산 순서는 임의로 지정이 되기 때문에 블록간의 데이터 의존성이 있는 LU 분해 알고리즘의 경우에는 블록의 계산 순서를 인위적으로 해 주어야 한다. 본 논문에서는 이러한 계산 방식을 사용하기 위해 병렬 LU 분해 알고리즘을 제안한다. 아울러 부분 피벗팅을 계산하기 위하여 분산 쓰레드간의 reduction 방식의 병렬프로그램을 제안한다.

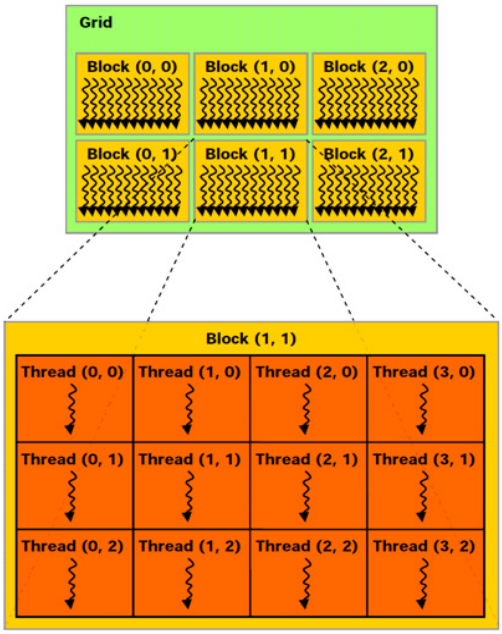
본 논문은 2장에서 병렬프로그램의 구현 방식, 3장에서는 성능에 대해서 기술하고 4장에서 결론을 맺는다.

2. 병렬 프로그램의 구현

2.1 GPGPU의 구조

GPGPU에서는 프로세서들이 격자 구조로 배치가 되어 있으며 이들을 병렬로 계산하여 계산 성능을 높인다. (그림 1)은 GPGPU 프로세서들의 격자 구조를 CPU와 비교하여 보여준다. 이 구조는 SIMD(Single Instruction Multiple Data)와 구조가 유사하지만 GPGPU에서는 쓰레드를 기반으로 하여 이를 SIMT(Single Instruction Multiple Threads)라고 부른다[6].

CUDA 병렬프로그램에서는 계산의 단위가 쓰레드 블록인데 하나의 블록이 쓰레드에 의하여 동시에 계산이 된다. (그림 2)는 계산에 할당된



(그림 2) GPGPU의 프로세서 구조

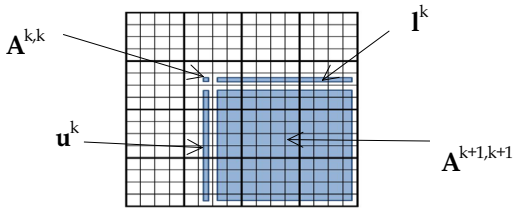
GPGPU 쓰레드의 예를 보여준다. 도메인은 그리드로 지정이 되며, 이 그리드는 사용자가 프로그램에서 블록과 쓰레드의 구조를 지정해 주면 블록으로 나누어진다. 그림에서는 4×3의 쓰레드가 동시에 3×2 크기의 그리드 중의 하나의 블록을 동시에 계산한다.

2.2 병렬 LU 분해 알고리즘

LU 분해 프로그램은 주어진 $N \times N$ 크기의 행렬 **A**를 크기가 각각 $N \times N$ 하삼각행렬 **L**과 상삼각행렬 **U**의 곱으로 분해하는 프로그램이다. 분해된 두 행렬 **L**과 **U**를 이용하여 연립방정식의 해를 쉽게 구할 수 있게 된다. 다음은 **L**과 **U**를 이용하여 연립방정식의 해를 구하는 2 단계이다.

주어진 연립방정식 **Ax** = **b**(**A**와 **b**는 상수 행렬, x는 미지수 행렬)에 대하여,

$$\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$$



(그림 3) 병렬 LU 분해 알고리즘

(i) **Ux = y**라고 하면 **Ly = b**이며, 이 식에서 **y**를 구한다.
(ii) 위에서 구한 **y**를 이용하여 **Ux = y**에서 **x**를 구한다.

LU 분해 알고리즘은 크게 일반 원소를 사용하는 방식과 서브블록으로 나누어서 계산하는 방식으로 분리된다[1-3]. [4]와 [5]에서 구현된 GPGPU 계산을 위한 병렬 LU 분해 프로그램은 서브블록으로 나누어서 행렬의 곱을 사용한 알고리즘을 사용한데 비하여 본 논문에서 구현된 병렬프로그램은 각 원소 단위로 구현된 알고리즘이다.

다음은 일반적인 LU 분해 알고리즘이다. 도메인의 크기는 $N \times N$이며 3차원 반복문에서 내부의 2중 반복문은 반복이 진행되면서 도메인의 크기가 작아지는 특징을 가지고 있다.

```
for(k=0; k<N; k++) {
    for(i=k+1; i<N; i++) {
        A(i,k) = A(i,k)/A(k,k);
        for(j = k+1;j<N; j++) {
            A(i,j) = A(i,j) - A(k,j)*A(i,k);
        }
    }
}
```

(그림 3)은 LU 분해법의 병렬 알고리즘을 보여준다. 반복문에서 k값이 0부터 N-1까지 반복하면서 $\mathbf{u}^k$을 $\mathbf{A}^{k,k}$를 사용하여 병렬로 수정한 후에 $\mathbf{u}^k$와 $\mathbf{l}^k$를 사용하여 서브도메인 $\mathbf{A}^{k+1,k+1}$을 수정하면 된다[3].

2.3 CUDA 환경에서의 병렬 LU 분해 프로그램

이 절에는 2.2에서 설명한 병렬 LU 알고리즘을 GPGPU에서 실행하기 위해 CUDA C를 사용하여 본 논문에서 제안한 병렬프로그램에 대하여 설명한다. 먼저 CUDA 환경에서 멀티 쓰레드를 사용하기 위하여 프로그램을 구현한 방식을 설명하고 이를 반복문에서의 데이터 의존도가 있는 LU 분해 프로그램에서 사용하기 위하여 프로그램을 전환하는 과정을 설명한다. 여기에서의 병렬 LU 프로그램의 방식들은 모두 본 논문에서 제안한 것이다.

2.3.1 데이터 의존도를 고려하지 않은 병렬 LU 분해 프로그램

다음은 CUDA C를 사용한 LU 분해법의 일반적인 병렬 프로그램이다. 먼저 호출 프로그램에서 블록의 크기를 쓰레드의 개수를 사용하여 정하고, 이 값을 사용하여 다시 그리드를 블록으로 나눈다.

```
dim3 dimBlock(NUM_THREADS,NUM_THREADS);
dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x,
((N+dimBlock.y-1)/dimBlock.y));
```

GPGPU에서 계산하기 위한 CUDA 함수는 다음과 같다. blockDim.x와 blockDim.y는 시스템 변수로서 2차원 블록의 크기를 나타내며, blockIdx.x와 blockIdx.y, threadIdx.x와 threadIdx.y는 각각 블록의 인덱스와 블록에서의 쓰레드의 인덱스를 나타낸다.

```
i = blockIdx.x*blockDim.x+threadIdx.x;
j = blockIdx.y*blockDim.y+threadIdx.y;

for(k=0; k<N; k++) {
    if (i > k) {
        if (j == k) A(i,j) = A(i,j)/A(k,k);
```

```
    //u 값 계산
        if (j > k) A(i,j) = A(i,j)-A(k,j)*A(i,k);
//서브도메인 계산
        }
    }
```

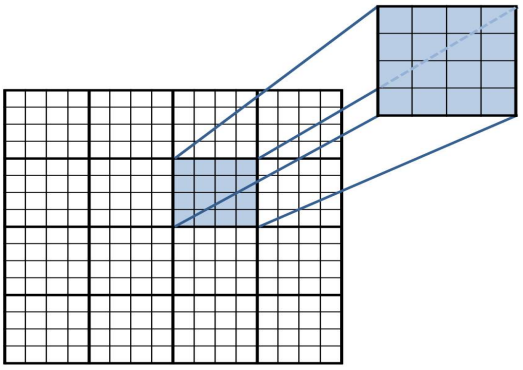
2.3.2 데이터 의존도를 고려한 병렬 LU 분해 프로그램

CUDA 프로그램에서는 여러 개의 쓰레드들이 하나의 블록을 동시에 계산함으로써 성능을 개선한다. (그림 4)는 이러한 병렬 쓰레드의 블록 계산과정을 보여 준다. 그림에서 4×4 격자의 쓰레드가 동시에 해당 블록을 계산하는데 그리드 내에서 블록들의 계산 순서는 무작위로 이루어진다[6]. 따라서 LU 분해 알고리즘의 경우에는 (그림 3)에서와 같이 $\mathbf{u}^{k}$을 먼저 수정한 후에 $\mathbf{A}^{k+1,k+1}$을 수정하여야 하는데 이러한 데이터 의존도를 해결할 수가 없다.

본 논문에서는 LU 분해 알고리즘의 데이터 의존도를 해결하기 위하여 다음과 같이 그리드를 하나의 블록으로서 사용하고 인위적으로 블록의 계산 순서를 정하도록 하였다. 따라서 프로그램에서 블록은 순차적으로 계산이 되기 때문에 데이터의 의존성이 해결이 된다. 이 방식은 SIMD 컴퓨터에서 주로 사용하는 방식으로서 SIMT에서 사용하는 방식을 보여 준다[7]. 다음은 제안된 병렬 LU 분해 알고리즘이다. 먼저 주어진 쓰레드의 개수로 계산 블록을 설정하고 그리드의 크기는 하나로 설정한다.

```
dim3 dimBlock(NUM_THREADS,NUM_THREADS);
dim3 dimGrid(1, 1);
```

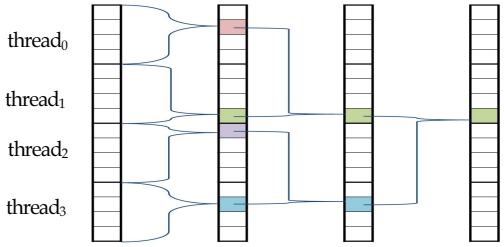
GPGPU CUDA 프로그램에서는 블록의 계산 순서를 나타내기 위하여 이중 반복문을 사용한다. 또한 블록 내어서도 $\mathbf{A}^{k+1,lk+1}$보다 $\mathbf{u}^{k}$을 먼저 계산하기 때문에 쓰레드의 동기를 호출하였다. 따라서 이 반복문에 의하여 데이터 의존성이 해결이 된다.



(그림 4) 쓰레드의 할당된 블록 계산

```
for(k=0; k<N; k++) {
    for(bj=k/blockDim.y; bj<n; bj++) {
        j = bj*blockDim.y + threadIdx.y;
        if (j == k) {
            for(bi=k/blockDim.x; bi<n; bi++) {
                i = bi*blockDim.x + threadIdx.x;
                if (i > k) {
                    A(i,j) = A(i,j)/A(k,k);
                }
            }
        }
        __syncthreads();

        if (j > k) {
            for(bi=k/blockDim.x; bi<n; bi++) {
                i = bi*blockDim.x + threadIdx.x;
                if (i > k) {
                    A(i,j) = A(i,j)-A(k,j)*A(i,k);
                }
            }
        }
        __syncthreads();
    }
}
```



(그림 5) 부분 피벗팅을 계산하기 위한 병렬 reduction

2.4 부분 피벗팅(Partial pivoting)

피벗팅은 LU 분해 알고리즘에서 수치상으로 안정적인 결과를 얻기 위해서 적용한다[3]. GPGPU에서는 각 원소의 값이 쓰레드를 통하여 분산이 되어 있기 때문에 그림과 같이 병렬 reduction을 사용하였으며 알고리즘은 (그림 5)와 같다. 먼저 각 쓰레드에서는 최소값을 얻기 위하여 해당 블록에서 순차적으로 최소값을 찾는다. 이 값을 다음에는 병렬 reduction 알고리즘을 적용하여 하나의 쓰레드에서 최종적으로 최소값을 찾고 이 값을 사용하여 두 개의 행을 바꾼다.

3. 성능 분석

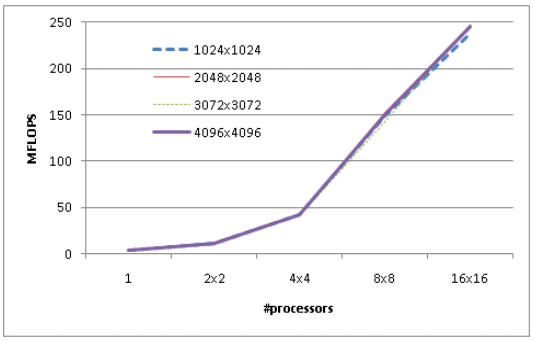
이 장에서는 GPGPU를 사용하여 본 논문에서 제안한 병렬 LU 분해법 프로그램의 성능을 기술한다.

3.1 시스템 사양

프로그램을 실행하기 위한 GPGPU는 Nvidia Tesla C1060을 사용하였다. 4GB의 메모리를 가지고 있으며 프로세서의 속도는 1.3GHz이며 최대 512개의 쓰레드를 사용할 수 있다. 또한 GPU 프로그래밍을 위하여 CUDA SDK 1.1을 사용한다.

3.2 성능 비교

호스트 컴퓨터에서는 CUDA 함수를 호출하고 결과를 기다리지 않고 다음 문장을 실행하기 때문에 실행 시간의 측정은 cudaEvent를 사용하여 함수의 시작과 끝을 event로 기록하여 측정하였다[6].

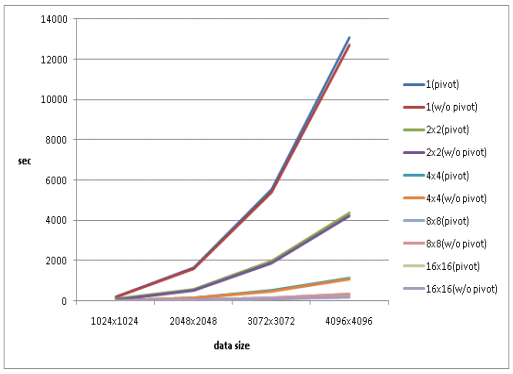


(그림 6) LU 분해 프로그램의 GPGPU를 사용한 계산 성능

(그림 6)은 병렬 LU 분해법 프로그램의 GPGPU를 사용한 계산 성능을 보여 준다. 사용한 도메인의 크기는 각각 1024×1024, 2048×2048, 3072×3072, 4096×4096으로 하였다. 쓰레드의 그리드의 크기 및 형태는 1×1, 2×2, 4×4, 8×8, 16×16으로 다양하게 계산 성능을 실험해 보았다.

그림에서 볼 수 있듯이 제안된 방법으로 성능을 측정해 본 결과 도메인의 크기와 프로세서의 크기에 비례하여 수행되는 성능(MFLOPS)도 증가함을 알 수 있다. 또한 도메인의 크기에 관계없이 성능의 효율성을 보여 주기 때문에 보다 큰 도메인을 사용할 때도 같은 성능 개선의 효과를 볼 수 있음을 알 수 있다.

(그림 7)은 피벗팅을 적용한 경우와 적용하지 않은 경우의 시간을 비교하였다. 프로세서의 개수에 많을수록 실행 시간은 많이 감소하는 것을 알 수 있다. 이것은 (그림 6)에서와 같이 제안된 알고리즘으로 병렬 LU 분해 프로그램이 성능의 효과가 있다는 것을 의미한다. 한편 부분 피벗팅의 적용 여부는 거의 차이가 없는 것으로 봐서 부분 피벗팅 자체는 성능에 거의 영향을 주지 않는 것을 알 수 있다.



(그림 7) 부분 피벗팅의 성능 비교

4. 결 론

본 논문에서는 GPGPU에서 계산을 할 수 있도록 병렬 LU 분해 프로그램을 구현하였다. Nvidia GPGPU 계산 환경인 CUDA 환경에서는 그리드(전체 데이터 도메인)를 여러 블록으로 나누고 각 블록을 할당된 쓰레드들이 동시에 계산하여 성능을 높인다. 이 때 블록의 계산 순서는 임의로 지정이 되기 때문에 블록간의 데이터 의존성이 있는 LU 분해 알고리즘의 경우에는 결과가 정확하지 않게 된다. 본 논문에서는 병렬 LU 분해 알고리즘에서 블록의 계산 순서를 인위적으로 적용하여 정확한 결과를 계산할 수 있도록 프로그램을 구현하였다.

제안된 프로그램을 Nvidia GPGPU에서 실행할 경우에는 데이터의 크기에 상관이 없이 쓰레드의 개수를 증가함에 따라 쓰레드의 개수에 비례하여 실행 시간이 감소함으로서 병렬 프로그램의 효율성을 보였다. 따라서 GPGPU가 LU 분해 프로그램에서도 효율적으로 사용될 수 있음을 보였다.

향후 연구 과제로는 GPGPU와 CPU와의 데이터의 효율적인 전송 속도의 향상을 위하여 GPGPU의 메모리 처리 등에 관해 연구할 예정이며, 관련 연구를 통하여 GPGPU를 일반적인 수치 계산의 활용하는 것이 훨씬 수월해 질 것으로 전망된다.

참 고 문 헌

[1] G. Geist, and C. Romine, ‘LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures.’, SIAM J. Sci. Stat. Comput., vol. 9, no. 4, pp. 639-649, July 1988.

[2] G. Laszewski, M. Parashar, A. Mohamed, and G. C. Fox, ‘On the Parallelization of Blocked LU Factorization Algorithms on Distributed Memory Architectures.’, Proceedings of ’92 Conference on Super Computing, 1992.

[3] Y. Kim, ’Performance Comparison of Two Parallel LU Decomposition Algorithms on MasPar Machines.’, Journal of IEEE Korea Council, Vol. 2, No. 2, pp. 247-255, 1999.

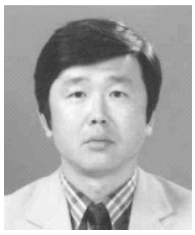
[4] N. Galoppo, N. Govindraju, M. and D. Henson, ’LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphic Hardware.’, Proceedings of 2005 Conference on Super Computing, 2005.

[5] V. Volkov and J. Demmel, ’LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs’, LAPACK Working Note 202, 2008.

[6] NVIDIA CORPORATION. 2009. Nvidia Program Guide Version 2.3.1

[7] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, ’Solving Problems on concurrent Processors Vol. 1.’, Prentice Hall, Englewood Cliffs, NJ, 1988.

◐ 저 자 소 개 ◑



신 봉 희

1977년 인하대학교 전자공학과 공학사
1981년 인하대학교 전자공학과 공학석사
1995년 단국대학교 전자공학과 공학박사
2010~현재 인천대학교 컴퓨터공학부 교수
관심분야 : 마이크로프로세서, 임베디드시스템, etc.
E-mail : bhshin@icc.ac.kr

김 영 태

1986년 연세대학교 수학과 학사
1992년 미국 Iowa State Univ. M.S.
1996년 미국 Iowa State Univ. Ph.D.
1998년~현재 강릉원주대학교 컴퓨터공학과 교수
관심분야 : 병렬처리, 초고속컴퓨팅, etc.
E-mail : ykim@kangnung.ac.kr