ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

GP-GPU의 캐시메모리를 활용하기 위한 병렬 블록 LU 분해 프로그램의 구현

Implementation of parallel blocked LU decomposition program for utilizing cache memory on GP-GPUs

김 영 태[1*] 김 두 한[1] 유 명 한[1]
Youngtae Kim Doo-han Kim Myoung-han Yu

요 약

GP-GPU는 그래픽 처리를 위한 GPU의 다중쓰레드를 일반 수치 계산에 활용하여 초고속으로 계산하는 장치이다. GP-GPU에서는 CPU의 캐시메모리와는 달리 다중쓰레드가 공유하는 공유메모리의 형태로 캐시메모리를 제공하며, 공유메모리는 사용자 프로그램에서 직접 제어할 수 있다. 본 연구에서는 GP-GPU의 캐시메모리를 사용하여 계산 성능을 향상시키기 위한 블록 구조의 병렬 LU 분해 프로그램을 구현하였다. Nvidia CUDA C로 구현된 병렬 블록 LU 분해 프로그램은 동일한 GP-GPU 상에서 일반 LU 분해 프로그램에 비교하여 7~8배 이상의 속도 개선을 보였다.

☞ 주제어 : LU 분해 프로그램, GP-GPU, Nvidia CUDA, 병렬프로그램

ABSTRACT

GP-GPUs are general purposed GPUs for numerical computation based on multiple threads which are originally for graphic processing. GP-GPUs provide cache memory in a form of shared memory which user programs can access directly, unlikely typical cache memory. In this research, we implemented the parallel block LU decomposition program to utilize cache memory in GP-GPUs. The parallel blocked LU decomposition program designed with Nvidia CUDA C run 7~8 times faster than nun-blocked LU decomposition program in the same GP-GPU computation environment.

☞ keyword : LU decomposition, GP-GPU, Nvidia CUDA, Parallel program

1. 서 론

GP-GPU(General Purposed Graphic Processing Unit)는 그래픽 계산 장치를 일반 계산에 사용하는 장치로서, 격자 구조로 배열된 다중 프로세서를 사용하여 행렬을 병렬로 계산할 수 있는 저전력, 고성능의 초고속 계산 장치이다[1][2][3][4]. 캐시메모리는 메인메모리보다 작지만 속도 면에서 현저히 빠른 메모리로서 메인메모리에 상주하는 데이터의 일부를 저장하여 프로세서가 메인메모리를 접근하지 않고 직접 데이터를 접근할 수 있는 저장장치이다. 한편 GP-GPU는 수백 개의 다중 쓰레드를 사용하는 병렬 계산의 특성상 시스템을 통하여 캐시메모리를 자동으로 사용하기가 어렵다. 따라서 일반적인 CPU와는 달리 응용 프로그램에서 직접 사용할 수 있는 (다중쓰레드들이 공유하는) 공유메모리의 형태로 캐시메모리를 제공한다. 캐시메모리는 메모리 처리 속도를 줄임으로써 계산 성능을 향상시키기 때문에 프로그램에서 캐시메모리를 최대한 활용하는 것이 성능에 있어서 매우 중요하다[5].

본 연구에서는 Nvidia GP-GPU의 캐시메모리(이하 공유메모리)를 사용하여 프로그램의 계산 시간을 단축하기 위하여 병렬 블록 LU 분해 프로그램을 구현하였다. LU 분해 프로그램은 2차원 행렬을 하삼각행렬(L)과 상삼각행렬(U)의 곱으로 표현하는 프로그램으로서 연립방정식을 계산하기 위한 가우스 소거법, 역행렬 및 행렬의 determinant 등을 계산할 수 있는 선형 대수의 가장 기본적인 프로그램 중의 하나이다[6]. 구현된 병렬 블록 LU 분해 프로그램은 블록을 사용하지 않는 LU 분해 프로그램과 동일한 복잡도인 $O(\frac{1}{3}n^3)$와 계산 결과를 갖지만, 행렬을 블록으로 나누어서 삼각행렬의 시스템 해법 계산

1 Department of Computer Science, Gangneung-Wonju National University, Wonju, 220-711, Korea.
* Corresponding author (ykim@gwnu.ac.kr)
[Received 1 August 2013, Reviewed 5 August 2013, Accepted 14 October 2013]

http://dx.doi.org/10.7472/jksii.2013.14.6.41

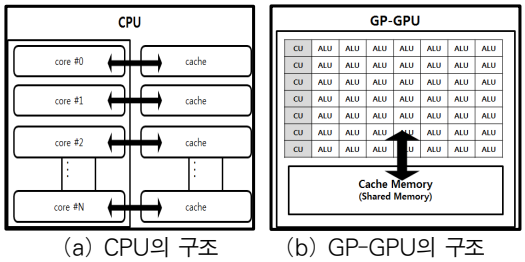
및 행렬의 곱을 사용하기 때문에 이 과정에서 나누어진 블록을 공유메모리에 저장하여 계산 시간을 단축할 수가 있다. 본 연구에서의 병렬 블록 LU 분해 프로그램은 동일한 GP-GPU 계산 환경에서 기존 LU 분해 프로그램과 비교하여 괄목할만한 속도 개선을 이루었다.

본 논문에서는 2장에서 병렬 블록 LU 분해 프로그램에 대해 설명하고, 3장에서는 병렬 블록 LU 분해 프로그램의 성능을 기존의 LU 분해 프로그램과 비교하여 속도 개선에 대하여 알아보며, 4장에서는 결론으로 맺는다.

2. 병렬 블록 LU 분해 프로그램

2.1 GP-GPU와 CUDA

GP-GPU는 그래픽을 처리하기 위한 장치로서 수백 개의 코어들이 격자 구조로 배치되어 있으며 이를 다중쓰레드를 통하여 실행하기 때문에 행렬의 계산에 있어서 매우 효율적인 구조이다. 그림1은 CPU와 GP-GPU 구조의 차이점을 보여준다. (a)에서와 같이 멀티코어로 이루어진 CPU는 각 코어가 별도의 캐시메모리와 연결되어 있지만 (b)에서의 GP-GPU에서는 여러 개의 코어들이 사용할 수 있는 메인메모리인 global 메모리와 쓰레드 블록이 사용할 수 있는 별도의 캐시메모리를 제공한다.



(a) CPU의 구조 (b) GP-GPU의 구조

(그림 1) CPU와 GP-GPU의 구조 비교

(Figure 1) Structure comparison of CPU and GP-GPU

일반적으로 CPU에서는 운영체제를 통하여 캐시메모리를 자동으로 관리하는 반면에 GP-GPU에서는 운영체제에서 각각의 코어에 캐시메모리를 자동으로 처리하기가 어렵기 때문에 응용프로그램에서 모든 코어들이 공유하는 공유메모리 형태의 캐시메모리를 직접 사용하게 된다. 따라서 응용프로그램에서 공유메모리를 효율적으로 활용하는 것이 전체 계산 성능에 매우 중요한 역할을 한다. 참고로, [1]에서 제안한 행렬의 곱 프로그램을 구현하여 비교한 결과, GP-GPU의 공유메모리를 사용한 경우가 global 메모리를 사용한 경우와 비교하여 최대 9배의 계산 속도 개선의 차이를 보였다.

GP-GPU를 처리할 수 있는 프로그램 언어로는 OpenCL, OpenGL, CUDA(Computer United Device Architecture) 등을 사용할 수 있다[1]. 이중 Nvidia에서 개발한 CUDA 모델은 고급 프로그램 언어인 C와 Fortran 등을 제공하기 때문에 일반 계산에서 가장 많이 사용하는 방식으로 본 연구에서도 CUDA C 프로그램 언어를 사용하였다.

2.2 Nvidia CUDA 환경에서의 LU 분해 프로그램

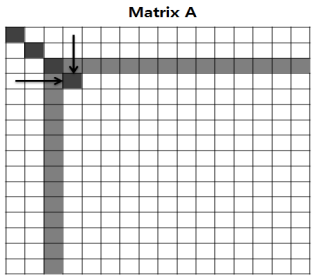
LU 분해 프로그램은 주어진 행렬 A에 대해 A=LU 형태로 하삼각행렬 L과 상삼각행렬 U의 두 개의 행렬로 나누는 프로그램이다. LU 분해 프로그램은 연립방정식의 해를 구하는 가우스 소거법과 거의 동일한 알고리즘을 사용하지만 분해된 L 및 U 삼각행렬을 사용하여 연립방정식의 해 뿐만 아니라 역행렬 및 행렬의 determinant 등을 계산할 수 있기 때문에 선형 대수 분야에서는 가장 대표적인 프로그램 중의 하나이다[6].

본 연구에서는 LU 분해 프로그램 구현을 위해 GP-GPU에서의 프로그래밍 언어로 CUDA C를 사용하였다. CUDA C에서는 먼저 동시에 계산하고자 하는 쓰레드 블록을 쓰레드의 개수로 설정하고, 전체 계산 도메인을 나타내는 그리드는 쓰레드 블록으로 나뉘어져 해당 영역들이 순차적으로 반복 계산한다. 이 때 그리드를 계산하기 위한 쓰레드 블록들의 계산 순서는 임의로 결정된다[1]. 다음은 LU 분해 프로그램에서 쓰레드 블록과 그리드를 정의하는 코드이다. 코드에서 그리드의 크기를 하나의 도메인으로 정의하는 이유는 데이터 의존도를 해결하기 위한 블록의 계산 순서를 프로그램에서 결정하기 위한 것이며 이 방식은 블록을 사용하지 않는 병렬 LU 분해 프로그램을 위하여 [7]에서 제안한 바 있으며 병렬 블록 LU 분해 프로그램에서도 동일한 방식을 사용한다.

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

그림 2는 블록을 사용하지 않는 LU 분해 프로그램의 병렬성을 보여 준다. 그림에서와 같이 프로그램의 대각선의 원소들을 기준으로 해당 열을 수정하고 이를 사용하

여 화살표가 표현하는 바와 같이 부분 도메인의 원소들을 수정하게 되며 반복이 진행되면서 계산하는 행렬의 크기는 작아지게 된다[7][8].



(그림 2) LU 분해 프로그램
(Figure 2) LU decomposition program

다음 코드는 그림 2의 방식을 구현한 LU 분해 프로그램이다. 코드에서 ①은 열을 수정하고, ②에서는 부분 도메인을 수정한다. CUDA C 프로그램에서는 행렬의 각 원소에 대한 반복문이 묵시적으로 이루어져서 반복문이 없지만, 이 코드에서 i와 j에 대한 반복문이 존재하는 이유는 앞에서 설명한 바와 같이 데이터의 의존성를 해결하기 위하여 명시적 반복문을 통하여 부분 행렬을 순서대로 계산하기 때문이다.

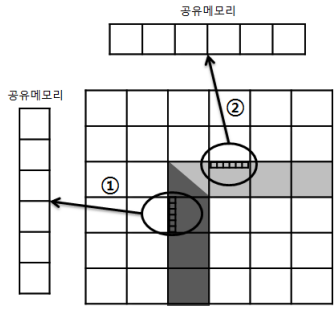
```
for(k=0; k<N; k++) {
    for(i=k+1; i<N; i++) {
        A(i,k) = A(i,k)/A(k,k);.....①
        for(j = k+1;j<N; j++) {
            A(i,j) = A(i,j) - A(k,j)*A(i,k);.....②
        }
    }
}
```

2.3 병렬 블록 LU 분해 프로그램

블록 LU 분해 프로그램은 주어진 행렬을 균등한 블록으로 나누어서 블록 단위로 삼각행렬의 시스템 해법 및 행렬의 곱 프로그램을 사용하여 계산 하는 방식이다[8][10]. 이 때 블록의 크기는 임의로 결정할 수 있는데 본 연구에서는 공유메모리를 최적으로 사용하기 위하여 블록의 크기를 GP-GPU에서의 쓰레드 블록의 크기로 정하였다. 다음의 알고리즘은 블록 LU 분해 프로그램의 기본 형태를 보여 준다. 반복문의 수는 블록의 수이며 반복문 내에서의 LU 분해 프로그램은 블록을 사용하지 않은 LU 프로그램을 말하며 부분 피벗팅을 포함한다.

```
for(B = 0; B < nBlocks; B++){
    블록을 사용하지 않는 LU 분해;
    삼각행렬 시스템의 해;
    행렬의 곱;
}
```

그림 3은 병렬 블록 LU 분해 프로그램에서의 LU 분해와 삼각행혈 시스템 해법의 알고리즘을 보여 준다. 그림에서 볼 수 있듯이 그림 2의 LU 분해 프로그램과는 달리 각 원소를 계산하는 것이 아니라 블록 단위로 계산하게 된다. 그림에서 ①은 블록을 사용하지 않는 LU 프로그램의 행렬이며 ②는 삼각행렬의 시스템 해에서의 행을 보여 준다. 이 때 계산에 사용되는 행렬의 일부분을 공유메모리에 저장하여 사용하므로 성능을 향상시킨다.



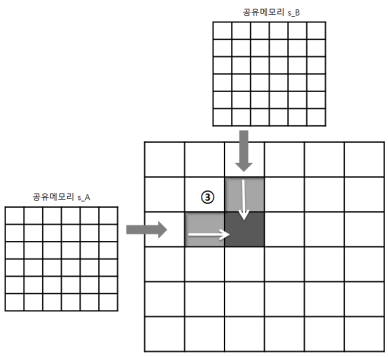
(그림 3) LU 분해와 삼각행렬 시스템 해법
(Figure 3) LU decomposition and triangular system solver

다음 CUDA C 프로그램은 그림 3에서 ②의 삼각행렬 시스템 해 프로그램이다. 프로그램에서 scol로 표현된 1차원 공유메모리의 행렬에 해당 행을 ①과 같이 저장하고 이를 쓰레드 블록이 공유하여 다음 계산인 ②에서 사용할 수 있다. 코드에서 i와 j에 대한 반복문이 없는 이유는 CUDA C에서는 반복문을 생략하기 때문이다[1].

```
i = B*blockDim.x + threadIdx.x;
for(kk=0; kk<blockDim.x; kk++) {
    k = B*blockDim.x + kk;
    scol(threadIdx.x) = A(i,k);.....①
```

```
    for(bi=B+1; bi<nBlocks; bi++) {
      j = bi*blockDim.y + threadIdx.y;
        if (j > k) {
          if (i > k) {
            A(i,j) = A(i,j)-A(k,j)*scol(threadIdx.x);..②
          }
        }
    }
}
```

그림 4는 블록 LU 분해 프로그램에서의 행렬의 곱 프로그램을 보여 준다. ③에서의 화살표가 표시하는 데로 각 블록은 행렬의 곱을 통하여 수정이 되며, 이 때 계산에 사용되는 부분 행렬은 행렬을 수정할 때 반복하여 사용이 되기 때문에 해당 블록을 2차원의 공유메모리에 저장하여 성능을 향상 시킬 수 있다.



(그림 4) 블록 LU에서의 행렬의 곱
(Figure 4) Matrix multiplication in the blocked LU decomposition

다음 프로그램은 그림 4에서의 알고리즘을 구현한 CUDA C 프로그램이다. 프로그램에서 ①과 ②에서와 같이 해당 블록을 2차원 공유메모리인 s_B와 s_A에 각각 저장을 하고 이 후에 ③에서의 행렬의 곱 프로그램에서 사용하게 된다.

```
for(bi=B+1; bi<nBlocks; bi++) {
   i = bi*blockDim.x + threadIdx.x;
   s_B(threadIdx.x,threadIdx.y) =
            A(i,B*blockDim.y+threadIdx.y);.....①
   for(bj=B+1; bj<nBlocks; bj++) {
      j = bj*blockDim.y + threadIdx.y;
      s_A(threadIdx.x,threadIdx.y) =
               A(B*blockDim.x+threadIdx.x,j);....②
      for(k=0; k<blockDim.x; k++) {
         A(i,j) -= s_A(threadIdx.x,k) *
                  s_B(k,threadIdx.y);.....③
      }
   }
}
```

2.4 부분 피벗팅

부분 피벗팅은 LU 분해 프로그램의 반복에서 해당 열에서 절대값이 가장 큰 행을 축이 되는 행과 교환함으로써 계산 결과를 좀 더 안정되게 한다[6]. 블록 LU 분해 프로그램에서의 부분 피벗팅은 [7]에서의 방식과 동일한 방식을 사용하였다.

3. 성능평가

3.1 시스템 사양

병렬 블록 LU 분해 프로그램의 성능 평가를 위해 두 개의 다른 GP-GPU를 사용하였으며 사양은 표1과 같다. Nvidia Tesla C1060의 프로세서의 수는 30개이며 각 프로세서는 8개의 코어로 구성이 되어 있고 최대한 512 쓰레드를 동시에 사용할 수 있다. 반면에 Tesla C2075는 8개의 코어로 구성이 된 14개의 프로세서로서 최대한 1,024개의 쓰레드를 사용할 수 있다. 다중 쓰레드가 사용할 수 있는 캐시(공유)메모리는 C1060이 16K bytes이고 C2075가 48K bytes이다. Tesla C2075가 Tesla C1060보다 코어의 수는 적지만, 최대 병렬 쓰레드의 수가 많고 메모리 처리 속도가 향상이 되어 계산량이 많은 경우에 보다 효율적인 성능을 보인다. 본 논문에서는 성능 평가를 위하여 두 GP-GPU를 사용함으로써 성능의 이식성을 보이고자 한다.

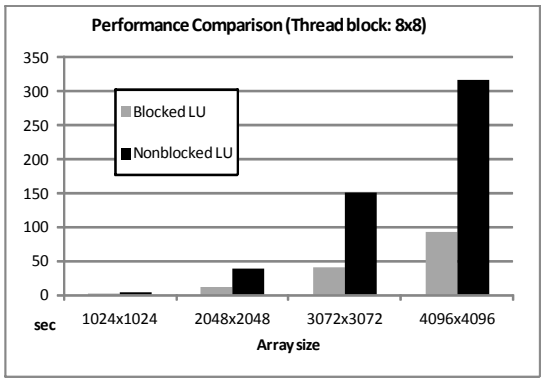
(표 1) 성능 비교에 사용된 GP-GPU의 사양
(Table 1) System specification of GP-GPUs used in performance comparison

	Tesla C1060	Tesla C2075
Clock frequency	1.3 GHz	1.15 GHz
multiprocessors의 수	30 개	14 개
cores의 수	240 개	112 개
최대 쓰레드 블록의 수	512 개	1024 개
(블록당) 공유메모리의 크기	16K bytes	48K byres
global memory의 크기	4G bytes	4G bytes

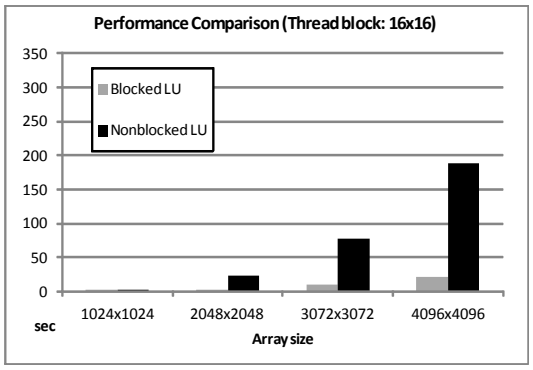
3.2 성능 비교

병렬 블록 LU 분해 프로그램의 성능 평가를 위하여 블록을 사용하기 않은 병렬 LU 분해 프로그램과 실행 시간을 비교하였다. 성능의 비교를 위하여 1024×1024, 2048×2048, 3072×3072, 4096×4096 크기의 행렬에 대해 각각 쓰레드의 수를 다르게 설정하여 실행 시간을 측정하였다. 계산에는 모두 부분 피벗팅을 포함하였다.

그림 5는 Tesla C1060에서의 병렬 블록 LU 분해 프로그램의 성능을 블록을 사용하지 않는 병렬 LU 분해 프로그램과 비교하였다. 그림에서 (a)는 64(=8x8) 개의 쓰레드를 사용한 경우이며 (b)는 256(=16x16) 개의 쓰레드를 사용한 경우이다.



(a) 64(=8x8) 쓰레드를 사용한 계산 시간의 비교



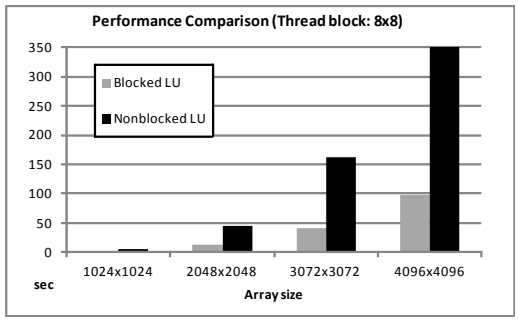
(b) 256(=16x16) 쓰레드를 사용한 계산 시간의 비교

(그림 5) 블록 LU 분해 프로그램과 일반 LU 분해 프로그램의 성능 비교(Tesla C1060)

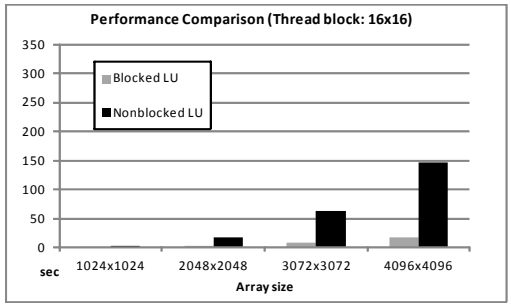
(Figure 5) Performance comparison of the blocked LU decomposition and the non-blocked LU decomposition programs(Tesla C1060)

그림에서 볼 수 있듯이 쓰레드 블록이 클수록 성능의 개선은 향상되며, 256개의 쓰레드를 사용할 경우(그림 (b) 참조)에는 최대 9배 정도의 향상된 성능을 보인다.

그림 6은 Tesla C2075를 사용하여 병렬 LU 분해 프로그램들의 성능을 비교하였다. 그림5에서와 유사하게 병렬 블록 LU 분해 프로그램의 향상된 성능을 보였으며 최대 8배 정도의 성능을 보인다.



(a) 64(=8x8) 쓰레드를 사용한 계산 시간의 비교



(b) 256(=16x16) 쓰레드를 사용한 계산 시간의 비교

(그림 6) 블록 LU 분해 프로그램과 일반 LU 분해 프로그램의 성능 비교(Tesla C2075)

(Figure 6) Performance comparison of the blocked LU decomposition and the non-blocked LU decomposition programs(Tesla C2075)

그림 5와 6에서 나타나듯이 캐시메모리를 활용한 블록 LU 분해 프로그램의 성능이 기존의 LU 분해 프로그램에 비하여 뛰어 나게 향상된 것을 볼 수 있으며, 쓰레드 블록의 크기가 클수록 캐시메모리의 사용이 증가하여 성능의 개선은 향상되었다. 두 GP-GPU를 비교하면 Tesla C2075가 C1060보다 계산 속도는 빠르지만 성능 개선의

효과는 비슷하게 나타난다.

4. 결 론

GP-GPU는 저비용 및 저전력의 고성능 병렬 계산 장치로서 많은 활용이 기대되는 차세대 계산 장치이다. 본 연구에서는 Nvidia GP-GPU에서 캐시메모리를 사용하여 성능을 높이기 위하여 블록 LU 분해 프로그램을 구현하였다. 프로그래밍 언어는 CUDA C를 사용하였으며 구현된 병렬 블록 LU 분해 프로그램은 동일한 계산 환경에서 블록을 사용하지 않은 LU 분해 프로그램과 비교하여 7~8배 이상의 성능 향상을 보였다.

일반적으로 캐시메모리는 응용프로그램에서 직접 사용하기가 어렵기 때문에 성능의 향상을 위한 프로그램의 최적화는 제한적이다. 하지만 Nvidia GP-GPU에서는 응용프로그램이 직접 캐시메모리(공유메모리)를 사용하는 것이 가능하며 이는 성능 향상을 위해서는 절대적으로 필요하다. 따라서 본 연구에서 제안한 프로그램 방식은 뛰어난 성능의 향상을 보였으며, 이 방식은 향 후 유사한 행렬의 수치 프로그램에서 활용할 수 있을 것으로 기대한다.

참 고 문 헌(Reference)

[1] Nvidia, CUDA Programming Guide 4.2.

[2] J. Nickolls, "Scalable Parallel Programming with CUDA", ACM Queue, vol. 6, no. 2, pp.40 -53 2008.

[3] E. Lindholm, "NVIDIA Tesla: A Unified Graphics and Computing Architecture", IEEE Micro, vol. 28, no. 2, pp.39-55 2008.

[4] Nico et al., "LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware", SC '05 Proceedings of the 2005 ACM/IEEE conference on Supercomputing, pp. 3.

[5] John L. Hennessy, David A. Patterson. "Computer Architecture: A Quantitative Approach". 2011.

[6] Golub, Gene H. Van Loan, Charles F. (1996), Matrix Computations (3rd ed.), Baltimore: Johns Hopkins.

[7] Shin, B., Y. Kim, Implementation of high performance parallel LU factorization program for multi-threads on GPGPUs, Journal of Korean Society for Internet Information, Vol. 12, No. 3, pp. 131-137, 2011.

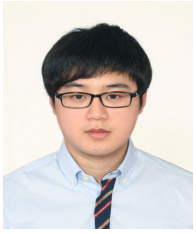
[8] Kim, Y., Performance Comparison of Two Parallel LU Decomposition Algorithms on MasPar Machines, Journal of IEEE Korea Council, Vol. 2, No. 2, pp. 247-255, 1999.

[9] 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.

[10] Gallivan et al., "Parallel Algorithms for Matrix Computations", SIAM, Philadelphia, 1991.

◐ 저 자 소 개 ◑

김 영 태(Youngtae Kim)
1986년 연세대학교 수학과(이학사)
1991년 Iowa State Unniv. 대학원 Department of Computer Science(석사)
1996년 Iowa State Unniv. 대학원 Department of Computer Science(Ph.D.)
1998년~현재 강릉원주대학교 컴퓨터공학과 교수
관심분야 : 병렬처리
E-mail : ykim@gwnu.ac.kr



김 두 한(Doo-han Kim)
2013년 강릉원주대학교 컴퓨터공학과(공학사)
2013년~현재 강릉원주대학교 컴퓨터공학과 대학원생
관심분야 : 병렬처리
E-mail : mkonjo@naver.com

유 명 한(Myoung-han Yu)
2009년 강릉대학교 컴퓨터공학과(공학사)
2011년 강원대학교 전자공학과(공학석사)
1998년~현재 강릉원주대학교 컴퓨터공학과 대학원생
관심분야 : 무선통신, 컴퓨터네트워크
E-mail : greatymh@gmail.com