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

도시기상모델 CFD_NIMR의 GP-GPU 실행을 위한 병렬 프로그램의 구현

GP-GPU based Parallelization for Urban Terrain Atmospheric Model CFD_NIMR

김 영 태[1*] 박 혜 자[2] 최 영 진[3]
Youngtae Kim Hyeja Park Young-Jeen Choi

요 약

본 논문은 도시기상모델인 전산유체역학모델(CFD_NIMR)을 GP-GPU에서 실행시키기 위해 CUDA Fortran 병렬프로그램을 구현하였다. GP-GPU는 원래 PCI 카드 형태의 그래픽 처리 장치이지만 저비용, 저전력으로 대량의 계산을 초고속으로 수행할 수 있는 일반 계산 가속기이다. 모델을 단일 Intel XEON 2.0 GHz CPU에서 실행한 결과와 Nvidia Tesla C1060 GPU에서 실행한 성능을 비교하였을 때 GP-GPU에서 15배 정도의 빠른 속도를 보였다. 또한 다중 CPU를 사용한 MPI 병렬프로그램과 비교한 경우에도 GP-GPU에서 보다 더 효율적인 성능을 보였다. 본 논문에서 제시한 프로그램 방식은 유사한 구조를 가진 수치모델을 GP-GPU 병렬 프로그램으로 구현하는데 쉽게 적용할 수 있을 것으로 기대한다.

☞ 주제어 : CFD_NIMR, GP-GPU, CUDA Fortran, 병렬 프로그램

ABSTRACT

In this paper, we implemented a CUDA Fortran parallel program to run the CFD_NIMR model on GP-GPU's, which simulates air diffusion on urban terrains. A GP-GPU is graphic processing unit in the form of a PCI card, and a general calculation accelerator to perform a large amount of high speed calculations with low cost and electric power. The GP-GPU gives performance enhancement of speed by 15 times to compare the Nvidia Tesla C1060 GPU with Intel XEON 2.0 GHz CPU. In addition, the program on a GP-GPU shows efficient performance compared to an MPI parallel program on multiple CPU's. It is expected that a proposed programming method on the GP-GPU parallel program can be used for numerical models with a similar structure.

☞ keyword : CFD_NIMR, GP-GPU, CUDA Fortran, Parallel Program

1. 서 론

1990년대 초반부터 본격적으로 사용되기 시작한 슈퍼컴퓨터는 11년 주기로 최고 속도가 1,000배씩 빨라지는 비약적인 발달을 거듭하여 왔으며, 2019년경 슈퍼컴퓨터의 최고 성능은 1초에 10^{18}번 연산을 할 수 있는 ExaFlops 급으로 예측되고 있다[1]. 슈퍼컴퓨터는 2000년대 까지는 공유메모리형이 많은 수를 차지하다가 이후에는 MPP (Massively Parallel Processing) 방식의 분산메모리형이 우위를 차지하고 있다[2]. 하지만 MPP 방식의 슈퍼컴퓨터는 많은 계산 장치로 인한 전력 소모, 발열 및 공간 등의 문제 때문에 프로세서 수의 확장만으로는 과거만큼 쉽게 속도 개선을 할 수 없는 단계에 와 있다[3]. 이러한 문제의 해결책으로서 초고속 계산을 저전력으로 할 수 있는 계산 가속기인 GP-GPU(General Purposed Graphic Processing Unit)가 주목을 받고 있다. GP-GPU는 원래 컴퓨터 화면의 그래픽 처리를 하기 위한 PCI(Peripheral Component Interconnect) 카드 형태의 장치인 GPU를 일반 계산에 사용하는 가속 계산 장치로서 내부에는 수십 개의 프로세서들이 격자형으로 배치가 되어 있어 이를 이용하여 저전력과 저비용으로 초고속의 계산을 할 수 있다. 또한 GP-GPU를 계산에 쉽게 사용할 수 있도록 C와 Fortran 등의 고급 언어의 컴파일러가 일반화 되면서 일반 수치 계

1 Department of Computer Science & Engineering, Gangneung-Wonju National University, 150 Heungup Namwonno Wonju Gangwon, 220-711, Korea
2 Forecast Research Division, National Institute of Meteological Research, 33 Seohobuk-ro Seogwipo-si Jeju-do, 697-845, Korea
3 Weather Information Service Engine Foundation, 434 World Cup buk-ro Mapo-gu Seoul, 121-835, Korea
* Corresponding author (ykim@gwnu.ac.kr)
[Received 11 January 2014, Reviewed 17 January 2014, Accepted 10 March 2014]

산에서도 효율적으로 사용이 되고 있다[4,5].

본 논문에서는 전산유체역학모델을 기본으로 하는 도시 지형에서의 미기상모델 CFD_NIMR(Computational Fluid Dynamics, NIMR은 국립기상연구소의 영문 약자임) [6]을 GP-GPU에서 실행하기 위하여 병렬프로그램을 구현하였다.

CFD_NIMR 모델은 유체의 흐름을 나타내는 역학방정식은 2차 미분 방정식이 기본으로 구성이 되어 있는데 복잡한 도시 지형에서의 흐름을 모의하기 위하여 최대한 작은 크기의 격자의 간격을 사용하기 때문에 모의를 하기 위해서는 많은 계산을 필요로 한다. 따라서 효율적으로 도시 기상을 모의하기 위해서는 병렬프로그램이 필수적이다. GP-GPU에서 실행되는 병렬프로그램은 Fortran을 사용하여 구현된 기존의 프로그램의 변경을 최소화하기 위하여 CUDA(Computer United Device Architecture) Fortran을 사용하여 구현하였다.

본 논문은 서론에 이어 2장에서 GP-GPU 병렬프로그램의 구현 방식에 대하여 설명하고, 3장에서 성능 분석을 통하여 프로그램의 성능을 확인하며, 4장에서 결론을 맺는다.

2. CFD_NIMR 모델의 GP-GPU 프로그램 구현

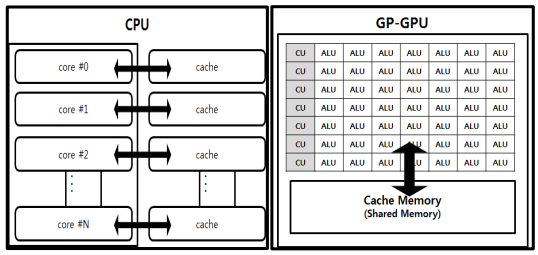
이 장에서는 도시기상모델 CFD_NIMR를 GP-GPU에서 실행하기 위하여 구현된 프로그램의 병렬화 과정에 대하여 설명한다.

2.1 도시기상모델 CFD_NIMR

CFD_NIMR 모델은 2005년에 국립기상연구소와 서울대학교가 국지 기상을 모의하기 위하여 개발한 유체역학 모델이다[6]. CFD_NIMR 모델에서 역학방정식은 먼저 2차 미분 방정식의 각 항의 값을 계산하고 수치적분법을 활용하여 2차 미분방정식의 해를 계산한다. 이 때 계산된 값은 수렴 과정을 거쳐서 재계산될 수 있다. C-grid의 엇갈림 격자 구조로서, 유체의 운동을 나타내는 변수 u, v, w는 각 운동 방향에 따라 격자의 면 중심에서 계산이 되며 격자의 중심에는 온도, 압력 물질의 농도 등의 변수가 계산이 된다. 이 구조의 계산은 규칙적으로 진행이 되며 격자를 병렬프로그램의 격자 프로세서에 적용하여 병렬계산을 하면 효율적인 계산을 할 수 있다.

2.2 GP-GPU와 CUDA

GP-GPU는 원래 그래픽 처리를 위한 GPU를 일반 계산에 사용하기 위해 개발된 장치이다. GP-GPU에서는 수십 개의 코어들이 격자 구조로 배치되어 할당된 데이터를 다중 스레드를 사용하여 동시에 실행한다. (그림 1)은 GP-GPU의 구조를 CPU와 비교하여 보여준다.



(a) CPU의 구조 (b) GP-GPU의 구조

(그림 1) CPU와 GP-GPU의 구조 비교

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

GP-GPU는 CPU와의 데이터를 PCI-E(Peripheral Component Interconnect Express)를 통하여 송수신한다[4]. PCI는 시스템과 별도의 장치를 관리하기 위한 장치로서 PCI-Express 2.0의 경우에 500MB/s의 전송 속도를 가지는데 이는 CPU의 데이터 전송 속도에 비해 현저하게 떨어진다[7]. 따라서 CPU와 GP-GPU를 혼용하여 계산을 하게 되면 많은 통신 부하가 발생하게 된다. 또한, 프로그램의 일부분을 병렬계산을 통하여 속도를 높이는 경우에는 Amdahl's의 법칙에 의하여 병렬 계산 분분이 일정 비율 이하를 차지할 경우 전체 프로그램의 성능은 일정 수준 이상으로 좋아지지 않는다[8]. 따라서 본 연구에서는 CPU와 GP-GPU간의 통신 오버헤드를 없애고 Amdahl's 법칙에 적용받지 않기 위해서 CFD_NIMR의 전체 프로그램을 GP-GPU에서 실행하도록 하였다.

GP-GPU를 처리할 수 있는 프로그램 언어로는 OpenCL, OpenGL, CUDA 등이 있다[4]. 이중 Nvidia에서 개발한 CUDA는 고급 프로그램 언어인 C와 Fortran 등에서 사용하는 API를 제공하는데, 본 연구에서는 CUDA를 사용할 수 있는 Portland Group의 상용 Fortran 언어를 사용하여 구현하였다.

2.3 CFD_NIMR CUDA 프로그램의 정형

GP-GPU를 사용하는 CFD_NIMR의 CUDA 프로그램은 CUDA Fortran을 사용하여 기존의 Fortran 코드와 최대한 동일하게 코드를 사용할 수 있도록 구현하였다. CUDA 프로그램의 정형화의 예는 다음과 같다.

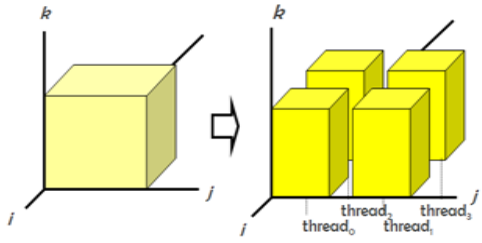
서브루틴 coefu()를 호출하면 코드와 같이 호출이 되는데 CUDA 프로그램으로의 컴파일을 위하여 컴파일 시 전달된 CUDA 지시문에 의해서 CUDA 코드를 선택하게 된다. 이 때 지시문이 사용되지 않으면 기존의 CPU 코드로 컴파일이 되어 CFD_NIMR 프로그램에서 CUDA 프로그램의 이식성을 제공한다. CUDA 코드로 설정이 되면 coefu() 대신에 Fortran90의 주요 기능인 *module*을 통하여 cuda_coefu() 루틴이 호출 되고, 이 루틴은 GP-GPU를 실행하기 위한 제반 설정과 함께 GP-GPU에서의 실행 서브루틴인 coefu_kernel()을 호출한다. GP-GPU에서 계산되는 커널 함수는 *attribute(global)*로 표현이 되는데 이 서브루틴의 호출에 대한 자세한 내용은 다음 절에서 설명한다.

```
subroutine coefu
...
#ifndef CUDA
(original coefu code)
#else
call cuda_coefu
#endif
return
end subroutine coefu

module cuda_coefu_mod
attributes(global) subroutine coefu_kernel
(coefu CUDA code)
end subroutine coefu_kernel

subroutine cuda_coefu
...
call coefu_kernel<<<dimGrid,dimBlock>>>(...)
...
end subroutine cuda_coefu
end module cuda_coefu
```

2.4 GP-GPU 초기화 및 데이터의 동적 할당

GP-GPU의 2차원 형태의 다중 스레드(스레드 블록) 구조를 활용하기 위하여 (그림 2)와 같이 3차원 데이터 도메인을 수평 방향으로 2차원으로 분할하여 서브 도메인을 GP-GPU의 각 스레드로 할당하게 하였다. 이러한 도메인 분할은 2차원 그리드 형태의 프로세서에서 3차원 데이터 도메인을 분할하는 가장 일반적인 방식이다 [9,10].

GP-GPU를 사용하기 위한 초기화의 설정은 다음과 같

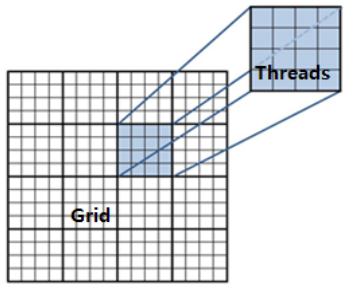


(그림 2) 스레드 블록의 도메인 분할
(Figure 2) Domain decomposition on a thread block

다. 먼저 주어진 스레드의 수 n을 사용하여 수평 2차원 도메인 $M \times N$을 효율적으로 분할하기 위한 GP-GPU의 2차원 스레드 블록 $gpu_nt_x \times gpu_nt_y$를 결정한다. 이 크기는 도메인을 효율적으로 분할하여 계산하기 위하여 도메인의 크기와 가장 비례하는 값을 구하는데 알고리즘은 다음과 같다.

$$gpu\_nt\_x = int\ \sqrt{\frac{M}{N} \times n}\,;$$
$$q = int\left(\frac{n}{gpu\_nt\_x}\right);$$
$$while\,(gpu\_nt\_x \times gpu\_nt\_y \neq n)\,\{$$
$$\quad gpu\_nt\_x = gpu\_nt\_x + 1;$$
$$\quad gpu\_nt\_y = int\left(\frac{n}{gpu\_nt\_x}\right);$$
$$\}$$

CUDA 프로그램에서는 계산 단위가 스레드의 블록인데 스레드 블록들의 구조를 나타내는 그리드는 전체 도메인(*M,N*)을 스레드 블록으로 나누어서 선언한다. (그림 3)은 GP-GPU의 그리드와 스레드 블록의 구조를 보여 준다. 그림에서 그리드는 계산하고자 하는 전체 도메인이고 스레드 블록은 병렬로 동시에 계산하는 서브 도메인을 말한다.



(그림 3) GP-GPU의 그리드와 스레드 블록

(Figure 3) A grid and a thread block on a GP-GPU

스레드 블록의 형태와 이를 기반으로 하는 그리드를 선언하고 이 값들을 GP-GPU 커널 함수에 전달하는 CUDA 프로그램의 예는 다음과 같다.

```
dimBlock = dim3(gpu_nt_x, gpu_nt_y, 1)
dimGrid =  dim3((M+dimBlock%x-1)/dimBlock%x, &
           (N+dimBlock%y-1)/dimBlock%y, 1)
...
call coefu_kernel<<<dimGrid,dimBlock>>>(...)
```

GP-GPU에서는 CPU 프로그램과는 달리 모든 배열 변수를 동적으로 할당하고 호스트로부터 데이터를 복사하여 사용한다. 따라서 아래 프로그램과 같이 먼저 동적으로 할당하는 GP-GPU 배열의 포인터로 정의한 후에 GP-GPU kernel 함수를 호출 전에 GP-GPU의 메모리를 동적으로 할당한 후 호스트의 값을 복사한다. 프로그램에서 *cu_uu*는 호스트 프로그램에서 사용하는 변수명인 *uu*를 *cu_*를 사용하여 호스트의 변수명과 구분하였다.

```
real, allocatable, device :: cu_uu(:,:,:)
…
allocate(cu_uu(M, N, kmax),stat=istate)
cu_uu = uu
```

2.5 매개변수의 전달을 이용한 기존 CPU 프로그램의 보존

CUDA 프로그램은 최대한 기존의 CPU 프로그램과 동일한 프로그램을 사용하도록 구현하였다. 이를 위하여 CUDA 프로그램은 CPU 프로그램과 동일한 변수명을 사용하여야 한다. 하지만 위에서 설명한 바와 같이 GP-GPU의 변수는 다른 변수명으로 선언되어야 하기 때문에, 이를 해결하기 위하여 형식 매개변수를 통하여 GP-GPU 함수로 전달하여 함수 내에서는 동일한 변수를 사용할 수 있게 하였다. 예를 들어, 다음 코드에서 *uu*는 호스트의 변수이고 *cu_uu*는 GP-GPU에서의 변수인데 CUDA 함수를 호출할 때 매개변수로 *cu_uu*를 사용하고 CUDA 함수에서 형식 변수로 *uu*를 사용하면 CUDA 프로그램에서 기존의 변수명을 사용할 수가 있게 된다.

```
call oefu_kernel<<<gpu%dimGrid,gpu%dimBlock>>>
.       (…, cu_uu,…)

attributes(global) subroutine coefu_kernel(…,uu,…)
real :: …,uu(limax2,ljmax2,gkmax2),…
```

2.6 GP-GPU 커널 함수에서의 인덱스 전환

GP-GPU 병렬프로그램은 스레드 블록이 동시에 계산되기 때문에 i-방향과 j-방향의 반복문은 존재하지 않는다. 대신에 도메인에서 반복문의 시작과 끝이 도메인 전체를 계산하지 않는 경우에는 도메인의 경계를 결정하는 조건문으로 변경되어야 한다. 한편 2.4절에서 설명한 바와 같이 3차원 도메인을 수평으로 분할하여 계산하기 때문에 k-방향의 반복문은 남아 있게 된다.

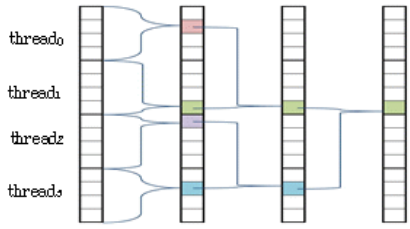
```
do k=2,kmax1                    do k=2,kmax1
do j=2,jmax1                    if((2<=j).and.(j<=(ljmax-1)) then
do i=2,imax                     if((2<=i).and.(i<=(limax-1)) then
…                          ⇨    …
enddo                           endif
enddo                           endif
enddo                           enddo
```

2.7 병렬 reduction

GP-GPU에서는 각 원소의 값이 스레드에 분산 되어 있기 때문에 분산된 행렬의 값을 하나의 스칼라 값으로 합하는 프로그램의 경우에는 병렬 reduction 방식을 사용하였다. 이 프로그램은 CFD_NIMR 모델의 *sor-* 코드에서 동일하게 나타난다. 병렬 reduction에서 사용한 알고리즘은 다음과 같다. 먼저 각 스레드에서 합계를 구하고, 다음 단계에서 앞 단계에서 계산된 값만을 이용하여 스레드들이 계산을 하게 되는데 이렇게 계산을 하게 되면 다음 단계로 갈수록 사용되는 스레드의 수를 나타내는 변수인 *dist*는 1/2로 줄어들어 최종적으로 하나의 스레드에서 합이 계산된다.

```
dist = blockDim%x*blockDim%y/2
do while(dist.ge.1)
  if (idx(tid_x,tid_y).le.dist) then
    smem(idx(tid_x,tid_y)) = smem(idx(tid_x,tid_y))
.                          + smem(idx(tid_x,tid_y)+dist)
  endif
  dist = dist/2
enddo
```

(그림 4)는 위 프로그램을 4개의 스레드를 사용한 예를 보여 준다. 최초에 4개의 스레드가 2개로 줄어들면서 각 원소의 합을 계산하고 최종적으로 1개의 스레드가 전체 원소의 합을 저장하게 된다.



(그림 4) 병렬 reduction 알고리즘
(Figure 4) Parallel reduction algorithm

3. CFD_NIMR CUDA 프로그램의 성능 분석

이 장에서는 CFD_NIMR CUDA 프로그램을 GP-GPU에서 실행하여 그 실행 성능을 CPU 프로그램과 비교 분석하였다.

3.1 시스템 사양

성능 분석에 사용한 시스템의 사양은 (표 1)과 같다. 성능 분석에 사용된 GP-GPU는 Nvidia Tesla C1060으로 1.3 GHz의 속도를 가진 프로세서로서 최대 240개의 코어까지 사용이 가능하다.

(표 1) 시스템 사양
(Table 1) System Specification

CPU	Intel Xeon E5405 (2.0GHz)
GP-GPU	Tesla C1060 (1.3.GHz)
Global memory	4G Bytes
#Multiprocessors	30
#Cores	240

3.2 성능 비교 분석

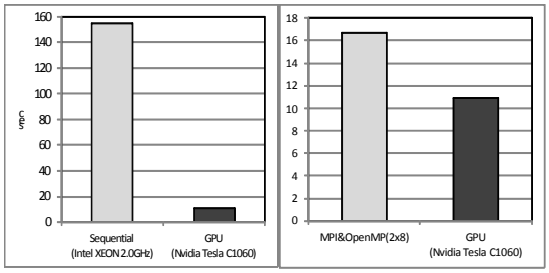
GP-GPU 코드의 결과에 대한 정확성을 알아보기 위하여 다음과 같이 두 프로그램의 결과값의 절대 오차를 계산하였다.

$$mean\ of\ errors = \frac{\sum|x-\overline{x}|}{n}$$
$$where\ x: Output\ from\ CPU\ program$$
$$\overline{x}: Output\ from\ the\ GPGPU\ program$$
$$n: number\ of\ grid\ points$$

GP-GPU 코드의 실행 결과값과 CPU 코드의 실행 결과값을 비교한 결과 결과값의 차이는 수치 오차의 허용범위(10^{-4}) 내에 있었으므로 GP-GPU 프로그램의 정확성에는 문제가 없음을 확인하였다.

CPU와 GP-GPU는 전혀 다른 형태의 계산 장치이기 때문에 속도의 비교는 큰 의미는 없지만 GP-GPU에서 실행된 병렬프로그램의 대략적인 성능을 알아보기 위하여 두 장치에서의 실행 성능을 비교하였다. (그림 5)는 GP-GPU에서의 계산 성능을 다양한 CPU의 계산 환경에서의 성능과 비교하여 보여 준다. 사용된 도메인의 크기는 100×100×60이며, 성능은 초기 100번 반복 시간의 평균을 초로 나타내었다. 그림 5(a)는 단일 CPU와 GP-GPU에서 실행한 결과를 비교한다. GP-GPU에서 사용한 스레드의 수는 256개이며 단일 Intel XEON 2.0GHz CPU에서 실행한 결과와 비교하여 15배 정도의 성능 개선을 보여주는 것을 알 수 있다.

한편, CFD_NIMR 모델은 분산메모리형 병렬컴퓨터의 프로그램 방식인 MPI(Message Passing Interface)와 공유메모리형 병렬컴퓨터의 프로그램 방식인 OpenMP를 사용한 병렬프로그램이 구현이 되어 있는데[11], 여기서는 각 노드가 8개의 코어로 구성된 2대의 노드를 사용한 병렬프로그램과 비교하였다. GP-GPU에서의 성능은 다중 CPU들을 사용한 CFD_NIMR의 병렬 프로그램과 비교한 경우에도(그림 5 (b)) 1.5배 이상 성능의 효율적인 결과를 보여 준다.



(a) CPU프로그램과의 비교 (b) 병렬프로그램과의 비교
(그림 5) CFD_NIMR 프로그램의 성능 비교
(Figure 5) Performance comparison of CFD_NIMR

programs

4. 결 론

CFD_NIMR 모델은 전산유체역학을 기반으로 하여 도시 지형을 모의하는 수치 모델로서 응용 기상 분야에서 활용도가 높은 프로그램 중의 하나이다. 이 프로그램은 계산 집약적인 프로그램으로서 방대한 계산량을 처리해야 하며, 효율적인 실시간 예측 및 현업 적용을 위해 보다 빠른 계산이 가능한 프로그램으로의 개선이 필요하다. 따라서 본 논문에서는 차세대의 초고속 계산 가속기로서 주목 받고 있는 Nvidia GP-GPU를 활용하였다.

본 논문에서는 GP-GPU에서 CFD_NIMR 모델을 실행할 수 있도록 CUDA Fortran과 여러 알고리즘, 기법들을 사용하여 CUDA 병렬프로그램을 구현하였다. 구현된 CUDA CFD_NIMR 프로그램은 CPU에서 실행한 프로그램과의 결과 비교를 통하여 검증하였고, 기존의 CPU를 사용한 순차프로그램과는 15배 이상, 16개의 CPU를 사용한 병렬프로그램과는 1.5배 이상의 뛰어난 계산속도의 개선을 보였다.

본 논문에서의 CUDA CFD_NIMR 프로그램의 구현 방식은 CUDA Fortran을 사용한 GP-GPU 프로그램의 정형 제공함으로써, 다른 수치모델의 프로그램에도 쉽게 적용이 가능하여 GP-GPU 계산 프로그램의 구현 기간이 단축될 것으로 기대한다.

참 고 문 헌(Reference)

[1] Meuer, H., The future of HPC, Scientific Computing World: June/July 2009.

[2] The Top Trends in High Performance Computing, The Top 500 Report, Top 500 Supercomputer Sites, 23 June 2009.

[3] NVIDIA CUDA Compute United Device Architecture Reference Manual Version 5.0, NVIDIA Corporation, 2012.

[4] Nvidia, CUDA Programming Guide 4.2.

[5] CUDA Fortran Programming Guide and Reference, The Portland Group, 2012.

[6] National Institute of Meteorological Research, Diagnosis of characteristics of local meteorology and development of techniques for the meteorological environmental impact assessment (I), 218 pp, 2006.

[7] Budruk, R., D. Anderson, T. Shanley, and J. Winkles, PCI Express System Architecture, Mind share PC system architecture, Addison-Wesley, pp 1120, 2003.

[8] Amdahl, G., Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings (30): 483 – 485, 1967. (2013)

[9] Youngtea Kim, Yong Hee Lee, and Kwan-Young Chung, WRF Physics Models Using GP-GPUs with CUDA Fortran, Atmosphere. Korean Meteorological Society, Vol. 23, No. 2, 231-235, 2013.

[10] Molnár Jr., T. Szakály, R. Mészáros, and I. Lagzi, Air pollution modeling using a graphics processing unit with CUDA, Comput. Geosci. 36(5), 105-112., 2010.

[11] Min-Wook Kim,Young-Jean Choi, and Youngtae Kim, Hybrid Parallelization for High Performance of CFD_NIMR Model, Atmosphere. Korean Meteorological Society, Vol. 22, No. 1, 109-115, 2012.

◐ 저 자 소 개 ◑

김 영 태

1986년 연세대학교 수학과 졸업(학사)
2001년 아이오와주립대학교(미국) 대학원 컴퓨터과학과 졸업(석사)
2006년 아이오와주립대학교(미국) 대학원 컴퓨터과학과 졸업(박사)
1998년~현재 강릉원주대학교 컴퓨터공학과 교수
관심분야 : 병렬처리, 초고속컴퓨팅
E-mail : ykim@gwnu.ac.kr

박 혜 자

2001년 강릉대학교 컴퓨터공학과 졸업(학사)
2006년 강릉원주대학교 교육대학원 교육학과 컴퓨터교육전공 졸업(석사)
2012년 강릉원주대학교 대학원 컴퓨터공학과 졸업(박사)
2012년~2014년 강릉원주대학교 컴퓨터공학과 초빙교수
2014년~현재 국립기상연구소 예보연구과 연구원
관심분야 : 소프트웨어공학, 수치모델링
E-mail azuregem.hp@gmail.com

최 영 진

1981년 연세대학교 대기과학과 졸업(학사)
1984년 연세대학교 대학원 대기과학과 졸업(석사)
2003년 연세대학교 대학원 대기과학과 졸업(박사)
1987년~2013년 국립기상연구소
2013년~현재 차세대 도시기상 사업단 단장
관심분야 : 기상관측, 수치모델
E-mail : junokma@gmail.com