Atmosphere. Korean Meteorological Society
Vol. 23, No. 2 (2013) pp. 231-235
http://dx.doi.org/10.14191/Atmos.2013.23.2.231

기 술 노 트

WRF 물리 과정의 GP-GPU 계산을 위한 CUDA Fortran 프로그램 구현

김영태[1),*] · 이용희[2)] · 정관영[2)]
[1)]강릉원주대학교 컴퓨터공학과
[2)]국립기상연구소 예보연구과

(접수: 2012년 12월 17일, 수정: 2013년 1월 25일, 게재확정일: 2013년 3월 25일)

WRF Physics Models Using GP-GPUs with CUDA Fortran

Youngtae Kim[1),*], Yong Hee Lee[2)], and Kwan-Young Chung[2)]
[1)]Department of Computer Science & Engineering, Gangneung-Wonju National University
[2)]Forecast Research Division, National Institute of Meteorological Research

(Received: 17 December 2012, Revised: 25 January 2013, Accepted: 25 March 2013)

Abstract We parallelized WRF major physics routines for Nvidia GP-GPUs with CUDA Fortran. GP-GPUs are originally designed for graphic processing, but show high performance with low electricity for calculating numerical models. In the CUDA environment, a data domain is allocated into thread blocks and threads in each thread block are computing in parallel. We parallelized the WRF program to use of thread blocks efficiently. We validated the GP-GPU program with the original CPU program, and the WRF model using GP-GPUs shows efficient speedup.

Key words: WRF, GP-GPU, CUDA Fortran

1. 서 론

기상 및 기후 모의를 위한 수치모델은 대부분 많은 계산식을 포함하기 때문에 컴퓨터의 단일 프로세서를 사용할 경우에는 방대한 계산 시간을 요구하여 계산을 통한 모의 예측 시간의 지연이 발생한다. 따라서 많은 프로그램들이 초고속 계산을 할 수 있는 병렬컴퓨터에서 실행할 수 있도록 병렬프로그램으로 구현이 되어 있다. 그러나 수백 내지는 수천 개 이상의 프로세서로 구성된 MPP (Massively Parallel Processors) 형태의 병렬컴퓨터에서 프로세서 수를 확장하여 성능을 높이는데, 이 방식은 데이터 처리 및 전력제어 등의 문제점을 가지고 있다. 이를 해결해 줄 수 있는 방편으로 격자 구조의 프로세서들을 작은 공간에 적재하여 효율적으로 계산할 수 있는 그래픽 처리 장치인 GPU (Graphic Processing Unit)를 일반 계산에 활용할 수 있는 GP-GPU (General Purpose GPU)가 미래형 초고속 계산 장치로 주목을 받고 있다 (Top 500 supercomputer sites, 2009). 대표적 그래픽 카드 제조사 중의 하나인 Nvidia에서는 병렬 계산 환경인 CUDA (Computer Unified Device Architecture)를 제공하고 있으며 프로그래밍 언어로는 CUDA C, CUDA Fortran, OpenCL 등이 있다 (NVIDIA Corporation, 2012; The Portland Group, 2012). CUDA 환경에서는 코드를 GPU에 배치하여 SIMT (Single Instruction Multiple Threads) 연산을 수행하는 방식으로서 성능이 뛰어나지만 프로세서의 기반이 다르기 때문에 기존의 프로그램을 GP-GPU 프로그램으로 다시 구현해야 한다.

WRF (Weather Research Forecast) 모델은 세계적으로 가장 많이 사용되는 범용기상모델 중의 하나이며, 다양한 연구 및 신속하고 정확한 예보를 위하여 보다 빠른 계산을 필요로 한다 (Michalakes *et al*., 2004; Michalakes *et al*., 2008). WRF 프로그램은 분산 및 공유메모리 병렬처리환경에서 실행이 가능하며, 실험용으로 물리과정 WSM5가 CUDA C언어를 사용하여

*Corresponding Author: Youngtae Kim, Department of Computer Science & Engineering, Gangneung-Wonju National University, 150 Heungup Namwonno, Wonju 220-711, Korea.
Phone : +82-33-760-8667, Fax : +82-33-760-8661
E-mail : ykim@gwnu.ac.kr

구현이 된 바 있다. 본 연구에서는 WRF 모델의 주요 물리과정 중 미세물리과정 (WSM6, WDM6), 경계물리과정 (YSUPBL), 복사물리과정 (RRTMG)과 지표물리과정 (SFCLAY) 등을 CUDA Fortran 언어를 사용하여 구현하였으며, 이 병렬프로그램은 GP-GPU의 계산을 CPU와 혼합형으로 계산할 수 있는 방식으로서 모델의 모의 시간을 효율적으로 단축시켜 준다. 본 연구에서 사용하는 Fortran 언어는 Portland Group에서 개발한 CUDA Fortran을 사용하였으며 컴파일러 옵션을 통하여 기존의 프로그램과 호환을 가지고 Nvidia GP-GPU에서 실행할 수 있도록 하였으며 향후 이 방식은 다른 수치 모델의 구현에도 활용될 수 있다.

본 논문의 2장에서는 GP-GPU에서 실행 가능한 WRF 물리 과정의 CUDA Fortran 프로그램의 구현에 대하여 상세히 설명한다. 그리고 3장에서는 CUDA 프로그램을 GP-GPU 계산 환경에서의 성능을 검증하고 4장에서는 결론을 제시한다.

2. GP-GPU 병렬프로그램의 구현

이 장에서는 GP-GPU의 구조 및 CUDA Fortran 프로그램의 기본 형태를 알아보고, WRF 물리과정 프로그램의 GP-GPU 병렬화에 대하여 설명한다.

2.1. GP-GPU의 구조 및 CUDA Fortran 프로그램의 기본 구조

GP-GPU는 계산 도메인(그리드)을 기본 계산장치인 쓰레드들의 격자(쓰레드 블록)로 나누고 나누어진 그리드의 해당 부분을 하나의 쓰레드 블록이 동시에 병렬로 계산하는 형태로서 SIMT (Single Instruction Multiple Thread)라고 불린다 (NVIDIA Corporation, 2012). Fig. 1은 그리드와 쓰레드 블록의 관계를 보여 주는데 쓰레드 블록은 나누어진 서브도메인을 순차적으로 계산하며 이 반복은 묵시적으로 이루어진다 (Shin *et al*., 2011).

Nvidia GP-GPU에서의 프로그램 환경을 제공하는 CUDA는 쓰레드 블록의 크기와 인덱스, 그리고 쓰레드의 인덱스값 등을 시스템 변수로서 제공한다. 이 변수들을 사용하여 전체 데이터 도메인에서 쓰레드 블록이 병렬로 동시에 계산하기 위한 서브도메인을 지정한다. 또한 전체 도메인을 계산하기 위한 쓰레드 블록들의 순차적인 반복은 묵시적으로 이루어지기 때문에 이 반복문은 삭제된다. 코드 (1)은 2차원 구조의 쓰레드 블록을 사용한 GP-GPU 커널 코드의 기본적인 형태이다. 프로그램에서 *blockDim*, *blockIdx*와 *threadIdx*는 CUDA에서 제공하는 시스템 변수이며 이 변수들로 계산되는 인덱스 *i*와 *j*는 도메인에서의 격자의 위치를 표현하지만 이를 사용한 반복문은 존재하지 않으며, 필요 시에 쓰레드 계산을 제외한 반복문은 존재하게 된다.

```
subroutine wdm6_gpu_kernel(..)
...
i = (blockIdx%x-1)*blockDim%x + threadIdx%x
j = (blockIdx%y-1)*blockDim%y + threadIdx%y
if (((i.ge.its).and.(i.le.ite)).and.((j.ge.jts).and.(j.le.jte))) then
do k=kts,kte
   data(i,k,j)=...
enddo
endif
end subroutine wsd6_gpu_kernel
```
(1)

본 연구에서는 WRF 모델의 3차원 데이터 도메인을 수직 방향으로 분할하여 수평 방향 2차원으로 배열된 각 쓰레드 블록이 분할된 3차원의 서브도메인을 동시에 계산하도록 하였다. 이 방식은 WRF의 병렬 MPI (Message Passing Interface) 프로그램뿐 (Michalakes, 2000) 아니라 많은 3차원 도메인을 기반으로 하는 수치 모델에서 사용하는 방식이다 (Nichols *et al*., 2008; Molnár *el al*., 2010). 따라서 프로그램에서는 수직 방향의 반복만만 남게 되고 수평 방향의 반복문은 생략이 된다 (코드 (1) 참조). Fig. 2는 2차원의 쓰레드 블록에 분할된 3차원 도메인을 보여준다.

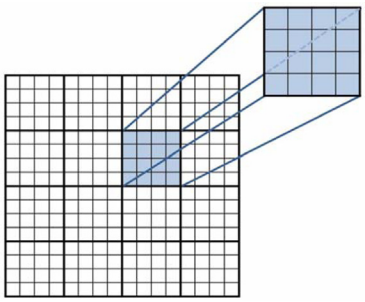

Fig. 1. A grid and a thread block on a GP-GPU.

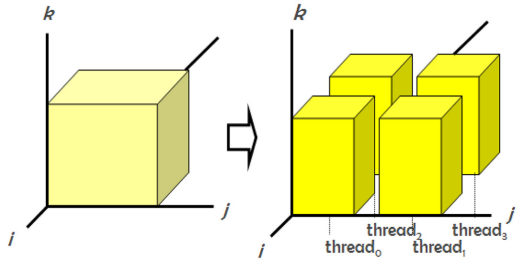

Fig. 2. Domain decomposition on a thread block.

2.2. GP-GPU 메모리의 사용

이 절에서는 호스트 컴퓨터의 메모리와는 별도로 운용되는GP-GPU의 메모리에 관하여 설명한다.

2.2.1. GP-GPU 변수의 할당

도메인에서의 변수들을 계산하기 위한 배열 변수는 크게 전역 변수와 지역 변수로 구분된다. CUDA Fortran에서는 함수 내에서 가변적인 크기를 지정할 수 있는 지역 변수 (automatic array)를 지원하지 않기 때문에 전역 변수와 지역 변수를 모두 프로그램의 초기에 동적으로 할당하여 사용한다.

2.2.2. Fortran 모듈 데이터의 사용

WRF의 일부 물리과정의 프로그램에서는 Fortan90의 주요 기능의 하나인 모듈을 사용하여 최초에 변수들을 초기화하고 이후에 이들을 저장하고 있는 모듈을 데이터베이스로서 사용한다 (The Portland Group, 2012). CUDA Fortran에서는 이러한 방식을 지원하지 않기 때문에 *use* 문을 사용하여 CPU 코드에서 초기에 작성한 모듈 데이터를 GP-GPU의 메모리로 복사하여 사용한다. 코드 (2)는 CUDA 코드의 예로서, 모듈 *rrlw_wvn*에 있는 변수 *ng*와 *nspa*를 동일한 이름의 GP-GPU 변수 *ng*, *nspa*와 구분하기 위하여 *ng_h*와 *nspa_h*로 지정한 다음 이 변수들을 GP-GPU 변수로 복사한다.

CUDA Fortran에서는 또한 배열 변수를 초기화시키는 *data* 문을 사용할 수 없기 때문에 CPU 코드에서 배열 변수를 초기화한 다음에 위와 같은 방법으로 GP-GPU의 변수로 복사하여 사용한다.

2.2.3. CPU 변수의 전달

GP-GPU에서는 모든 쓰레드가 동시에 공유하는 메모리인 global 메모리에 도메인 변수들을 저장하여 사용한다 (NVIDIA Corporation, 2012). 따라서 본 연구에서 구현한 WRF와 같은 혼합형 프로그램에서는 먼저 CPU에서 사용하는 변수들은 global 메모리에 있는 전역 변수로 복사한 후 GP-GPU에서 계산하고 그 결과를 다시 CPU의 변수로 복사한다. GP-GPU 메모리의 전역 변수는 프로그램의 초기에 동적으로 할당하며 프로그램에서 같은 이름의 변수명이 CPU와 GP-GPU에 동시에 존재하기 때문에 GP-GPU의 전역 변수명은 CPU 변수명에 *_d*를 사용하여 구분하였다. 이 변수들은 GP-GPU의 kernel 함수의 매개 변수로 전달할 때 해당 함수 내에서 원래의 변수명으로 전달됨으로써 기존의 프로그램에서의 변수명을 그대로 사용할 수 있도록 하여 CUDA 프로그램에서 수정을 최소화하였다. 코드 (3)은 변수를 전달하는 코드의 예로서 코드에서 *emiss*와 *p8w*는 CPU에서의 변수명이고 *emiss_d*와 *p8w_d*는 GP-GPU에서의 변수명이며 GP-GPU kerkel 함수의 호출 전후에 CPU와 GP-GPU 간에 각각 전달(복사)을 한다. 이 GP-GPU 변수들은 형식 매개변수를 통하여 GP-GPU kernel 함수에 전달함으로써 함수 내에서는 원래의 변수명인 *emiss*와 *p8w*를 사용할 수 있게 된다.

```
emiss_d = emiss
p8w_d = p8w
...
call rrtmg_lwrad_gpu_kernel<<<dimGrid, dimBlock>>> &
(emiss_d, p8w_d, ...)
emiss = emiss_d
p8w = p8w_d
....
attributes(global) subroutine rrtmg_lwrad_gpu_kernel(emiss, p8w, ...)
...
end subroutine rrtmg_lwrad_gpu_kernel
```
(3)

2.2.4. 지역 변수의 사용

CUDA kernel 함수 내에서 사용하는 지역변수는 CPU프로그램의 변수들과 구분할 필요가 없기 때문에 같은 이름을 사용하여 초기에 할당한다. CUDA kernel 함수에서 각 쓰레드 상에서만 실행되도록 호출하는 device 함수에서는 수평 도메인을 표현할 필요가 없기 때문에 도메인을 나타내는 인덱스는 없지만 kernel 함수에서는 인덱스가 표현이 된다. 코드 (4)는 GP-GPU에서 지역변수를 사용하는 코드의 예이다. 아래 코드에서 device 함수 *inatm_sw_gpu()*에서 원래 2차원으로 선언된 변수 *play*와 *plev*는 초기에 4차원 전역 변수로서 할당을 하고 kernel 함수에서 매개변수로 전달할 때 각 격자에서 수평 2차원을 제외한 나머지 2차원 도메인인 *play(1,1,i,j)*, *plev(1,1,i,j)*로 전달하여 사용한다.

```
real, dimension(:,:,:,:), ALLOCATABLE, DEVICE :: play, plev
...
call inatm_sw_gpu(..., play(1,1,i,j), plev(1,1,i,j),...)
...
attributes(device) subroutine inatm_sw_gpu(..., play, plev,...)
real(kind=rb), intent(in) :: play(ncol,nlay)
real(kind=rb), intent(in) :: plev(ncol,nlay+1)
...
```
(4)

2.3. CUDA Fortran프로그램의 구현

이 절에서는 기존의 WRF 물리과정 프로그램의 구조를 CUDA Fortran 프로그램의 구조로 전환하는 방식에 대하여 설명한다.

2.3.1. 3차원 구조로의 변경

WRF 모델의 물리과정 루틴들의 일반적인 형태는

3차원 도메인을 i,j 수평 방향 중 j-방향만을 따라서 2차원으로 계산을 하며 전체 j-방향의 크기만큼 반복한다. 코드 (5)의 왼쪽 코드는 이러한 계산의 형태를 보여준다. 이 방식은 과거의 컴퓨터 메모리 부족 등의 문제를 극복하기 위한 방식으로서 병렬프로그램에서는 비효율적인 형태이다. 따라서 GP-GPU의 2차원의 쓰레드 블록을 사용하여 3차원의 도메인을 계산하기 위해서는 오른쪽의 코드와 같이 3차원의 코드로 전환하여야 한다.

```
do k = kts, kte                          do k = kts, kte
do i = its, ite                          do j = jts, jte
cpm(i,k) = cpmcal(q(i,k))                do i = its, ite
xl(i,k) = xlcal(t(i,k))                  cpm(i,k,j) = cpmcal(q(i,k,j))
enddo                                    xl(i,k,j) = xlcal(t(i,k,j))
enddo                                    enddo
do k = kts, kte                          enddo
do i = its, ite                          enddo
delz_tmp(i,k) = delz(i,k)                do k = kts, kte
den_tmp(i,k) = den(i,k)                  do j = jts, jte
enddo                                    do i = its, ite
enddo                                    delz_tmp(i,k,j) = delz(i,k,j)
                                         den_tmp(i,k,j) = den(i,k,j)
                                         enddo
                                         enddo
                                         enddo
```

(5)

2.3.2. 수평 반복문의 삭제

2.1.절에서 설명한 바와 같이 GP-GPU에서는 쓰레드 블록이 계산하는 반복문은 불필요하기 때문에 프로그램에서는 수평 방향의 반복문은 삭제를 하게 된다. 따라서 GP-GPU 프로그램에서는 수평 방향의 i_loop와 j_loop를 제외한 반복문만 존재하게 된다. 코드 (6)의 오른쪽 코드는 GP-GPU 커널 코드의 일반적인 형태이다.

```
do k = kts, kte                          do k = kts, kte
do i = its, ite                          cpm(i,k,j) = cpmcal(q(i,k,j))
cpm(i,k,j) = cpmcal(q(i,k,j))            xl(i,k,j) = xlcal(t(i,k,j))
xl(i,k,j) = xlcal(t(i,k,j))              enddo
enddo
enddo                                    do k = kts, kte
do k = kts, kte                          delz_tmp(i,k,j) = delz(i,k,j)
do i = its, ite                          den_tmp(i,k,j) = den(i,k,j)
delz_tmp(i,k,j) = delz(i,k,j)            enddo
den_tmp(i,k,j) = den(i,k,j)              enddo
enddo
enddo
```

(6)

3. 병렬프로그램의 성능분석

GP-GPU 코드의 결과에 대한 정확성을 알아보기 위하여 별도의 비교 프로그램을 사용하여 GP-GPU 코드의 실행 결과값과 CPU 코드의 실행 결과값을 비교하였다. 결과값의 차이는 오차의 허용 범위 내에 있었으며 GP-GPU 병렬화에 있어서 정확성에는 문제가 없음을 확인하였다. 본 장에서는 WRF 물리과정 프로그램들을 GP-GPU에서 실행하여 그 성능을 분석하였다.

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

3.1. 실험 시스템의 사양

성능 분석에 사용한 시스템의 사양은 Table 1과 같다. GP-GPU는 4G bytes의 메모리 용량을 가진 Tesla C1060을 사용하였으며 이 GP-GPU는 PCI express 슬롯을 사용하여 시스템에 장착이 되어 있다.

3.2. GP-GPU에서 WRF 물리과정 프로그램들의 성능

CPU와 GP-GPU는 전혀 다른 계산장치이기 때문에 속도의 비교는 큰 의미는 없지만 GP-GPU의 성능을 알아보기 위하여 두 장치의 성능을 비교하였다. Fig. 3는 각 물리 과정의 GP-GPU 실행시간을 CPU 코드의 실행시간과 비교하였다. 사용된 도메인의 크기는 71×61×27 이며 실행 시간은 milli-seconds를 사용하였다. 그림에서는 모든 GP-GPU 코드가 CPU 코드에 비하여 4~5배 이상의 뛰어난 성능을 보이는 것을 알 수 있다. 이 성능은 이미 CUDA C 언어로 구현된 WRF의 물리과정 루틴인 WSM5에 비교하여 유사한 성능을 보임으로써 GPU를 사용하기 위한 코드의 최적화가 되어 있다고 보여진다 (Michalakes *et al*., 2008).

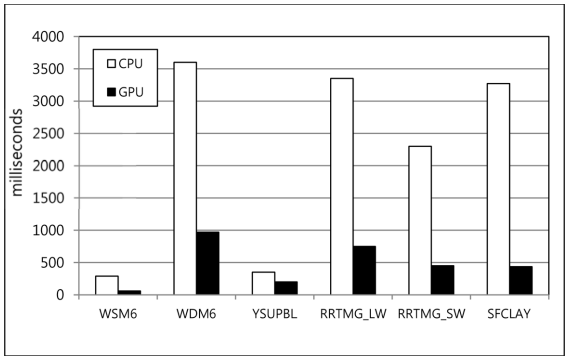

Fig. 3. Performance of WRF physics routines.

4. 결 론

본 연구에서는 WRF 모델의 주요 물리과정인 WSM6, WDM6, YSUPBL, RRTMG와 SFCLAY 루틴들을 실행 시에 GP-GPU와 CPU를 동시에 활용할 수 있도록 혼합형으로 프로그램을 구현하였다. GP-GPU에서 실행되는 물리과정의 프로그램들은 계산 시간을 획기적으로 단축하여 전체 WRF도 효율적으로 실행될 수 있도록 하였다. 또한 병렬화에 따른 성능을 분석하였다.

GP-GPU는 저전력, 저비용, 고성능의 미래지향형 계산 장치이다. 본 연구에서 구현된 물리과정의 GP-GPU 프로그램은 CPU 프로그램에 비하여 뛰어난 성능의 개선을 보여 주었으며, 향후 WRF 모델에 있어서 많은 활용을 기대한다. 본 연구에서의 GP-GPU를 사용하기 위한 병렬 방식은 Fortran 언어를 사용하기 때문에 코드의 재사용이 용이하며, WRF 모델뿐 아니라 Fortran 언어로 구현된 많은 기상 모델에서도 이러한 병렬 방식이 적용될 수 있다.

감사의 글

본 연구는 국립기상연구소 주요사업 NIMR-2012-B1의 일환으로 수행되었다.

REFERENCES

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

Michalakes, J., 2000: RSL: A parallel runtime system library for regional atmospheric model with nesting, in Structured Adaptive Mesh Refinement (SAMR) Grid Methods. *IMA Volumes in Mathematics and Its Applications*, **117**, 59-66.

_____, J. Dudhia, D. Grill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, 2004: The Weather Research and Forecast Model: Software Architecture and Performance. *Proceeding of the 11th ECMWF Workshop on the Use of high Performance Computing in Meteorology*, 25-29.

_____, and M. Vachharajani, 2008: Acceleration of Numerical Weather Prediction, *Parallel Processing Letters Vol. 18 No. 4. World Scientific*., pp. 531-548.

Molnár Jr., T. Szakly, R. Mszros, and I. Lagzi, 2010: Air pollution modeling using a graphics processing unit with CUDA. *Comput. Geosci*., **36**(5), 105-112.

Nichols, J., I. Buck, M. Garland, and K. Skadron, 2008: Scalable parallel programming with CUDA, *ACM Queue* **6**(2), 40-53.

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

Shin, B. and Y. Kim, 2011: Parallel LU decomposition using multiple threads on GPUs, *Korean Society For Internet Information*, **12**, 131-137.

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