지구물리와 물리탐사 Geophysics and Geophysical Exploration
Vol. 19, No. 3, 2016, p. 145~152 http://dx.doi.org/10.7582/GGE.2016.19.3.145

ISSN 1229-1064 (Print)
ISSN 2384-051X (Online)

그래픽 프로세서를 이용한 시간 영역 3차원 파동 전파 모델링과 메모리 관리

김아름 · 류동현 · 하완수*

부경대학교 에너지자원공학과

Time-domain 3D Wave Propagation Modeling and Memory Management Using Graphics Processing Units

Ahreum Kim, Donghyun Ryu, and Wansoo Ha*

Department of Energy Resources Engineering, Pukyong National University

요 약: 효율적인 시간 영역 3차원 파동 전파 모델링을 위해 그래픽 프로세서를 사용하였다. 그래픽 프로세서는 대규모 병렬 연산을 위한 프로세서로, 그래픽 프로세서를 효율적으로 이용하기 위해서는 계산 과정과 메모리 복사 과정을 최적화할 필요가 있다. 본 연구에서는 메모리 관리에 초점을 맞추고 메모리 관리 방법에 따라 그래픽 프로세서를 이용한 프로그램의 성능이 어떻게 달라지는지 확인하였다. 또한 유한 차분법 차수와 속도 모델의 크기를 변화시켜가며 메모리 복사가 프로그램 성능에 미치는 영향을 시험하였다. 그 결과 3차원 파동장 전체를 복사하는 프로그램에서 메모리 관리가 유한 차분법 계산보다 큰 비중을 차지함을 알 수 있었다.

주요어: 파동 전파, 그래픽 프로세서, 메모리

Abstracts: We used graphics processing units for an efficient time-domain 3D wave propagation modeling. Since graphics processing units are designed for massively parallel processes, we need to optimize the calculation and memory management to fully exploit graphics processing units. We focused on the memory management and examined the performance of programs with respect to the memory management methods. We also tested the effects of memory transfer on the performance of the program by varying the order of finite difference equation and the size of velocity models. The results show that the memory transfer takes a larger portion of the running time than that of the finite difference calculation in programs transferring whole 3D wavefield.

Keywords: Wave propagation, Graphics Processing Units, Memory

서 론

석유 가스 탐사 분야에서 3차원 및 4차원 자료 처리 기술과 컴퓨팅 기술 발전에 따라 고해상도 지하 정보를 얻기 위한 계산량이 갈수록 증가하고 있다. 대량의 자료를 처리하기 위해서는 컴퓨터 계산 자원이 상당히 많이 필요한데, 계산 문제를 해결하기 위한 방안으로 최근 그래픽 처리장치를 이용한 연산 기술(General Purpose computing on Graphics Processing Units, GPGPU)이 주목을 받고 있다. GPGPU는 그래픽 프로세서(Graphics Processing Units, GPU)의 그래픽 병렬 연산 처리 능력을 일반 수치 연산에 활용할 수 있도록 하는 기술이다. 이 기술은 기존의 계산 서버에 GPGPU 프로그래밍이 가능한 그래픽 카드와 관련 컴파일러를 추가로 설치하는 것으로 사용 가능하며 중앙 처리 장치(Central Processing Unit, CPU) 대비 전력 효율과 연산 능력이 뛰어나 다양한 분야에서 도입하고 있다(Kirk and Hwu, 2013; Cheng *et al.*, 2014). 탄성파 탐사 분야에서는 키르히호프 참 반사 보정(Kirchhoff migration), 거꿀 참 반사 보정(Reverse-time migration), 완전 파형 역산 등의 분야에서 연구하고 있고, 앞으로 더욱 다양한 자료 처리 단계에서 활용될 것으로 예상된다(Wang *et al.*, 2010; Shi *et al.*, 2010; Suh and Wang, 2011; Liu *et al.*, 2013; Kim *et al.*, 2013; Shin *et al.*, 2014; Yang *et al.*, 2014, 2015).

본 연구에서는 그래픽 프로세서를 이용하여 시간 영역 3차원 음향파 파동 방정식 모델링을 구현하였다. 파동 방정식 모델링은 거꿀 참 반사 보정이나 완전 파형 역산과 같은 고해상

Received: 2 August 2016; Revised: 18 August 2016;
Accepted: 18 August 2016
*Corresponding author
E-mail: wansooha@pknu.ac.kr
Address: Department of Energy Resources Engineering, Pukyong National University, 45 Yongso-Ro, Nam-Gu, Busan, Republic of Korea (48513)

도 자료 처리의 기초가 되는 기술이다. 기존 연구에서도 3차원 파동 방정식 모델링을 구현한 사례들이 있으나, 주로 연산을 위한 공유 메모리 사용이나 영역 분해를 위한 통신 등에만 초점을 맞추고 CPU 메모리와 GPU 메모리 사이의 효율적인 통신에 관한 언급은 거의 없었다(Micikevicius, 2009; Komatitsch *et al.*, 2009; Michéa and Komatitsch, 2010; Komatitsch *et al.*, 2010a, b; Mu *et al.*, 2013; Weiss and Shragge, 2013). GPU 기술 서적들에서도 이러한 내용을 다루는 경우가 많지 않은데, 효율적인 메모리 복사와 관련된 내용은 관련 기술 사용 설명서에서 찾을 수 있다(CUDA Toolkit Documentation, 2016). 본 연구에서는 단일 GPU 카드 사용시 CPU와 GPU 메모리 사이의 파동장 복사 과정 최적화에 초점을 맞추고 예제들을 통해 메모리 관리가 연산 최적화 못지 않게 중요함을 보인다.

파동 전파 모델링 알고리즘

본 연구에서는 다음의 시간 영역 3차원 음향파 파동 방정식을 유한 차분식을 이용해 풀었다.

$$\frac{1}{v^2(\mathbf{x})}\frac{\partial^2 u(\mathbf{x};t)}{\partial t^2} = \frac{\partial^2 u(\mathbf{x};t)}{\partial x^2} + \frac{\partial^2 u(\mathbf{x};t)}{\partial y^2} + \frac{\partial^2 u(\mathbf{x};t)}{\partial z^2} + f(\mathbf{x};t) \quad (1)$$

위 식에서 v는 P파 전파 속도, u는 압력 파동장, f는 송신원이다. 위 식을 유한 차분법을 이용해 CPU 상에서 풀기 위한 유사 알고리즘은 Table 1과 같다. 초기화 과정에는 배열 메모리 할당과 속도 모델 및 송신원 정보를 읽어들이는 작업이 포함된다. 시간 반복문 내에서 유한 차분법은 시간에 대해 2차, 공간에 대해 8차 중앙 차분식을 사용하였다. 시간 전진은 효율적인 연산을 위해 파동장을 복사하지 않고 C언어 포인터를 교환하는 방식으로 구현하였다. 경계 조건은 연산 능력과 메모리 복사 성능을 정확히 비교하기 위해 고정단 경계조건(Dirichlet boundary condition, Kreyszig, 2011)을 사용하였다. 파일 출력 시간이 전체 모델링 시간에 미치는 영향을 줄이기 위해 파동장을 파일에 쓸 때에는 3차원 파동장 전체를 쓰지 않고 지표 송신원 위치에서의 x 방향 측선만 출력하였다.

파동 방정식을 GPU에서 풀기 위한 유사 알고리즘은 Table 2에 제시하였다. CPU의 초기화 과정은 앞에서와 같지만, GPU 초기화 단계에서는 GPU 메모리 상에 배열을 할당하는 과정이

Table 1. A pseudo algorithm for a time-domain wave propagation modeling. The initializing process includes allocating arrays and loading velocity and source information.

```
Initialize
FOR each time DO
   Solve finite-difference equation
   Inject source
   Time march
   Write output
END FOR
```

Table 2. A pseudo algorithm using GPU for a time-domain wave propagation modeling. The initializing process of CPU includes allocating arrays and loading velocity and source information. The initializing process of GPU includes allocating arrays.

```
CPU, GPU: Initialize
CPU to GPU: Copy velocity, source information
FOR each time DO
   GPU: Solve finite-difference equation
   GPU: Inject source
   GPU: Time march
   GPU to CPU: Copy wavefield
   CPU: Write output
END FOR
```

주가 된다. GPU에서는 직접 디스크 입출력이 안 되기 때문에 모델링을 위해서는 속도 모델과 송신원 정보를 CPU 메모리에서 GPU 메모리로 복사해주는 과정이 필요하다. 시간 반복문 내에서 유한 차분식과 송신원 추가, 시간 전진 과정은 GPU에서 수행한다. GPU 연산 결과인 파동장은 매 시간 반복시 GPU 메모리에서 CPU 메모리로 복사한다. 3차원 파동장 전체를 복사하였고, 파일로 쓸 때에는 앞에서와 같이 지표 송신원 위치에서의 x 방향 측선만 출력하였다. 이때 시간을 가장 많이 차지하는 부분은 유한 차분식 계산 과정과 파동장 복사 과정이다. 3차원 파동장 전체를 복사하지 않고 일부만 복사하면 메모리 복사 시간을 줄일 수 있지만 거꿀 참 반사 보정이나 완전 파형 역산 과정에서 일반적으로 3차원 파동장 전체가 필요하므로 본 연구에서도 파동장 전체를 복사하였다.

실험 구성

CPU 알고리즘(Table 1)에서 유한 차분식은 메모리 접근 순서를 고려한 3개의 중첩 반복문으로 구성하였다. GPU에서는 CPU와 동일하게 x, y, z 방향 3개의 중첩 반복문으로 구성한 경우와 GPU 공유 메모리를 사용하여 GPU 쓰레드와 블록으로 x, y 반복문을 대신하고 깊이 방향에 대해서만 반복문으로 구현한 Micikevicius (2009)의 알고리즘을 적용한 경우를 비교하였다.

메모리 복사를 위해서는 CPU에서 GPU와 통신을 위해 일반적인 메모리를 할당하여 사용하는 경우, 페이지 잠금 메모리(page-locked memory)를 사용하는 경우, 그리고 GPU에서 연산과 메모리 복사 중첩을 위해 스트림(stream)을 사용한 경우를 비교하였다. 일반적으로 CPU에서 사용하는 메모리는 페이지 가능한 메모리(pageable memory)로, 가상 메모리 구현을 위해 운영체제에서 필요에 따라 메모리의 내용을 디스크에 옮기기도 한다. 그러나 페이지 잠금 메모리를 사용하면 운영체제의 페이징 과정 없이 메모리 복사를 실행하게 한다(부록A). 스트림을 사용할 경우 깊이 방향으로 영역을 여러 개로 분리한 후 각 영역별로 연산 후 메모리 복사를 실행하는 방식으로 구현하였다. 이 경우 하나의 영역에서 메모리 복사를 실행하는

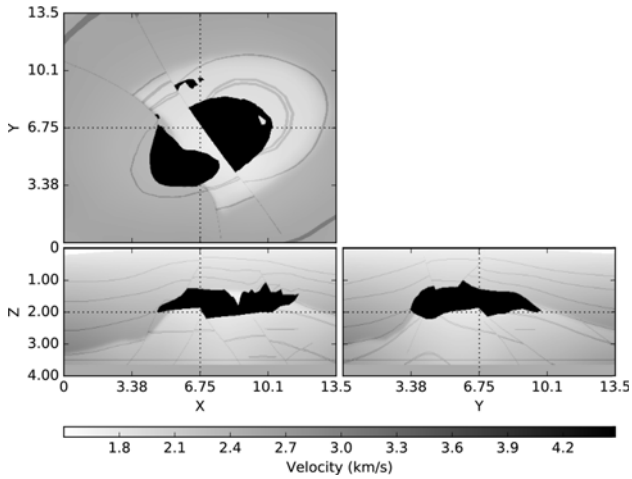

Fig. 1. SEG/EAGE 3D salt velocity model (Aminzadeh *et al.*, 1994).

동안 다른 영역에서는 유한 차분식 계산을 수행할 수 있게 된다(부록B). GPU 메모리 구조와 특성에 대해서는 사용 설명서 또는 관련 서적들에서 확인할 수 있다(Kirk and Hwu, 2013; Cheng *et al.*, 2014; CUDA Toolkit Documentation, 2016).

CPU 연산을 위해서는 공개된 GNU C 컴파일러의 2단계 최적화 옵션을 이용해 컴파일한 C 프로그램을 이용하였고, GPU 연산을 위해서는 엔비디아 사의 컴파일러를 이용해 컴파일한 CUDA (Compute Unified Device Architecture) 프로그램을 이용하였다. 하드웨어는 인텔 사의 Xeon E5-2680 2.50 GHz CPU와 엔비디아 사의 K40c GPU를 이용하였다.

CPU 연산과 GPU 연산 정확도 및 속도 비교를 위해 3차원 SEG/EAGE 암염돔 속도모델(Aminzadeh *et al.*, 1994)을 이용하였다(Fig. 1). 속도모델의 크기는 676 × 676 × 201, 격자 간격은 20 m이다. 지표면 6 km, 6 km 지점에 송신원을 두고 2 ms 간격으로 총 4초간 모델링을 수행하였다. 연산에 걸리는 시간을 측정할 때에는 초기화 부분은 제외하고 시간 반복문의 실행 시간만 비교하였다. 최적의 메모리 관리 방법을 확인한 후에는 공간에 대한 유한 차분식을 2차, 4차, 8차로 변화시켜 가며 성능을 비교하였고, 이후 모델 크기에 따른 연산 속도를 비교할 때에는 상속도 모델과 8차 유한 차분식을 이용하였다.

정확도 비교

GPU를 이용한 계산은 병렬 연산으로, 순차적으로 계산하는 CPU 프로그램의 경우와 연산 순서가 달라지게 된다. 컴퓨터에서의 부동 소수점 연산은 엄밀히 말해서 교환법칙이 성립하지 않는 연산이므로 GPU에서의 계산 결과는 CPU에서의 계산 결과와 달라질 수 있다(Kirk and Hwu, 2013). 따라서 GPU 프로그램의 결과가 충분히 정확한지 확인하는 과정이 필요하다. 연산 결과의 정확도를 확인하기 위해 CPU 프로그램에서 얻은 파동장과 GPU 프로그램에서 얻은 파동장을 비교하였다.

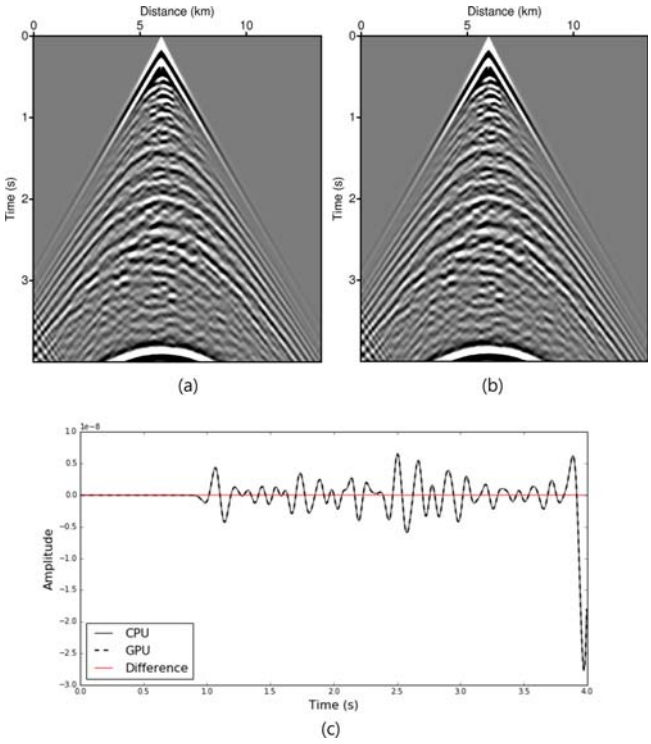

Fig. 2. Surface seismograms (y = 6 km) from (a) CPU and (b) GPU programs. (c) Traces extracted from a surface receiver 2 km apart from the source. 'CPU' and 'GPU curves show the traces from the CPU and GPU programs. 'Difference' curve shows the difference between the results from the CPU and GPU programs.

Fig. 2에 모델링 결과로 얻은 파동장, CPU 프로그램에서 얻은 트레이스와 GPU 프로그램에서 얻은 트레이스, 그리고 그 차이를 그렸다. 사용한 GPU 프로그램은 뒤에 나오는 최적화된 프로그램이다. 트레이스는 송신원으로부터 2 km 떨어진 지점에서 추출하였다. 실제 두 파동장의 차이 값이 0은 아니지만 무시할 수 있을 만큼 작음을 알 수 있다.

메모리 관리 최적화에 따른 실행 시간 비교

다음으로는 모델링 프로그램의 실행 시간을 이용해 CPU와 GPU의 성능을 비교하였다. CPU 프로그램의 실행 시간은 유한 차분식 계산 과정에서 대부분이 소모된다. GPU 프로그램의 실행 시간은 크게 유한 차분식 계산 시간과 파동장 복사 시간으로 나눌 수 있다(Fig. 3). 나머지 송신원을 추가하는 시간과 시간 전진 기법을 적용하는데 걸린 시간은 총 시간의 1% 미만으로 무시할 수 있다. GPU 프로그램의 총 실행 시간은 프로파일링 도구를 사용하지 않은 경우의 실행 시간이고, 계산시간과 메모리 복사 시간은 프로파일링 도구를 이용해 실행한 후 얻은 값이다.

그 결과 CPU에서와 동일한 알고리즘으로 유한 차분식을 계산할 때에는 유한 차분식 계산 부분이 전체 실행시간에서 차지하는 비중이 파동장 복사 부분보다 크지만 공유 메모리를

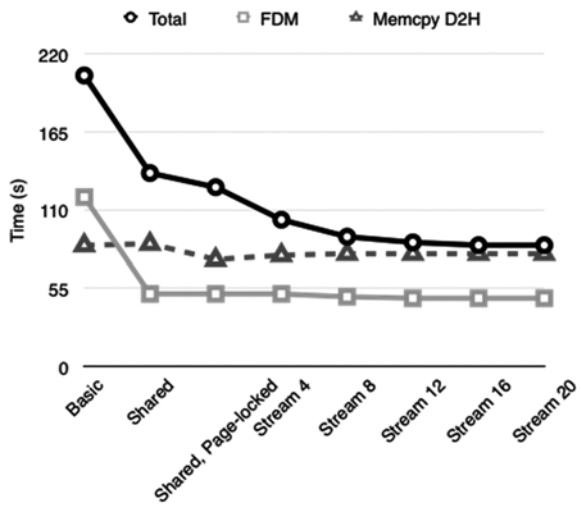

Fig. 3. Execution times of GPU programs. 'Basic' used the same finite difference algorithm to that of the CPU program. 'Shared' used shared memory to calculate the finite difference algorithm. 'Page-locked' used the page-locked memory for memory copy. 'Stream' used CUDA stream to overlap calculation and memory copy. 'Total' curve shows the total execution time, 'FDM' curve shows the calculation time of the finite difference kernel, and 'Memcpy D2H (Device to Host)' curve shows the total memory copy time of the 3D wavefield.

이용해 계산 과정을 GPU에 맞도록 최적화한 후에는 파동장 복사 비중이 유한 차분식 계산 비중보다 큼을 알 수 있다(Fig. 3). 페이지 잠금 메모리를 사용하면 파동장 복사 시간이 10% 이상 빨라지는 반면, 스트림을 사용할 경우 유한 차분식 계산 시간과 파동장 복사 시간은 큰 차이가 없었다. 대신 깊이 방향으로 영역을 분해해서 각 스트림에 할당할 경우 한 영역에서 유한 차분식 계산 후 파동장 복사를 수행하는 동안 다른 영역에서 유한 차분식 계산을 중첩하여 수행할 수 있기 때문에 전체 실행 시간은 줄어들게 된다. Fig. 4에 12개의 스트림을 사용한 경우 한 번의 시간 반복 내에서 GPU 프로그램을 분석한 프로파일러 창을 보였다. 창에서 x축은 시간을 의미하고, 녹색 상자는 유한 차분식 계산 함수의 실행 시간, 노란색 상자는 파동장 복사 시간을 의미한다. 하나의 스트림에서 유한 차분식 계산이 끝나고 파동장을 복사하기 시작하면 다른 스트림에서 유한 차분식 계산을 시작하는 것을 알 수 있다. 이런 방식으로 계산과 파동장 복사 과정을 중첩하게 되면 전체 실행 시간을

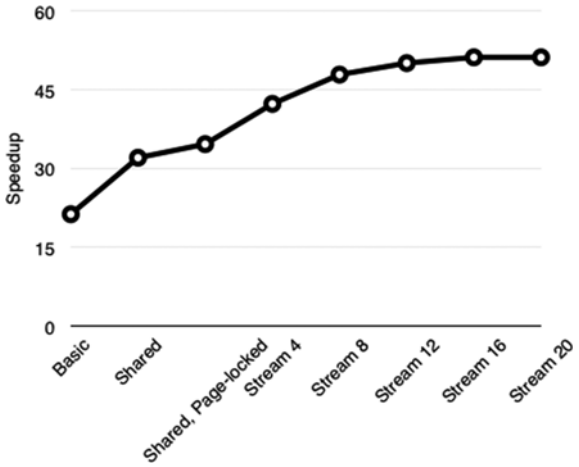

Fig. 5. Relative speedups of the GPU programs with respect to the CPU program.

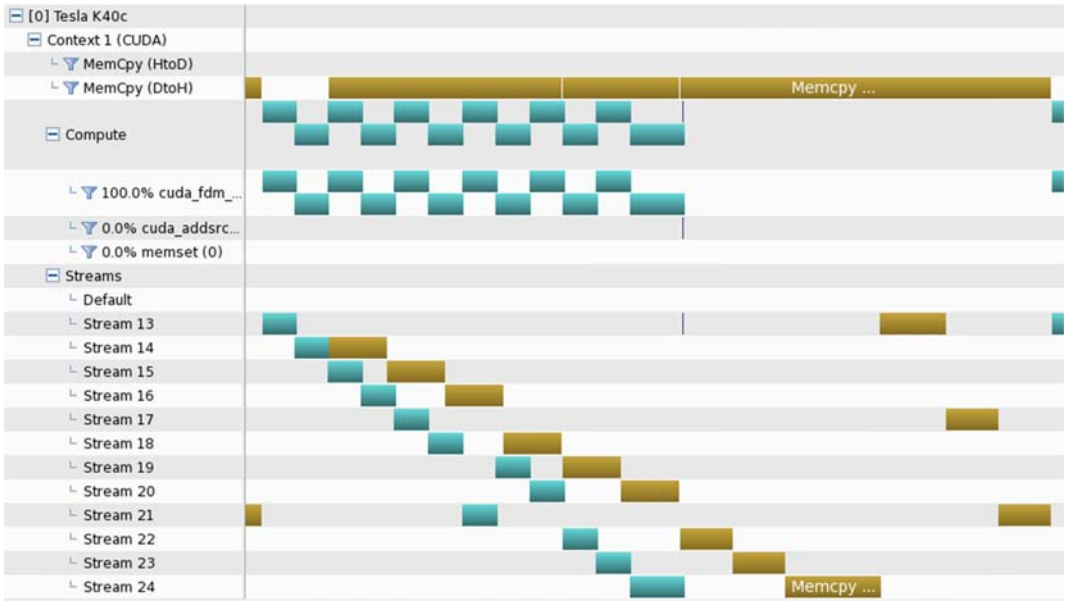

Fig. 4. A screenshot of a GPU profiling tool showing one time iteration. Green boxes show the finite difference calculation processes and yellow boxes show the memory copy processes. The width of a box shows duration of that process. There are 12 streams and a calculation process in one stream can be overlapped with a memory copy process of another stream. Note that the time of the memory copy process takes most of the execution time.

줄일 수 있다. 결과로부터 전체 프로그램 실행 시간은 유한 차분식 계산 시간보다 메모리 복사 과정에 크게 영향 받음을 알 수 있다. 단, 분해한 영역 수가 많아지면 GPU에 그에 따른 부하가 발생하기 때문에 전체 실행 시간이 계속해서 빨라지지는 않는다.

CPU 프로그램 대비 속도 향상 정도를 살펴보면 단순히 CPU에서 사용한 알고리즘을 GPU로 옮겼을 때 약 21배, 공유 메모리를 사용하여 GPU에 맞도록 유한 차분식 계산 알고리즘을 수정했을 경우 32배, 페이지 잠금 메모리와 스트림을 사용하여 메모리 관리를 최적화했을 때 약 51배 실행 속도가 향상되었다(Fig. 5). 물론 성능 향상 정도는 유한 차분식 차수나 모델 크기, 컴파일러, 장비 사양 등에 따라 달라지게 된다.

유한 차분식 차수에 따른 비교

이번에는 공간에 대한 유한 차분식의 차수를 2차, 4차, 8차로 변화시켜가면서 CPU와 GPU 프로그램의 성능을 비교하였다. GPU 프로그램은 앞의 예제에서 최고 성능을 보였던 20개의 스트림을 활용하는 프로그램을 사용하였다. 유한 차분식 차수를 2, 4, 8차로 변화시킬 경우 유한 차분식 계산에 사용하는 격자 수가 7개, 13개, 25개로 증가하고 그에 따라 계산량과 메모리 접근 시간도 증가하게 된다. 반면에 메모리 복사량은 큰 차이가 없다. 3차원 파동장의 크기는 유한 차분식 차수와 무관하지만, 실제 구현시 효율적인 메모리 복사를 위해 유한 차분식 계산에 필요한 후광 영역(halo area)까지 복사하기 때문에 차수가 올라갈수록 메모리 복사량이 약간씩 증가한다. 8차 유한 차분법과 단정도 배열을 사용할 경우 파동장 복사 크기는 매 시간 반복마다 4(676+8)(676+8)(201+8) = 373.0 MB, 총 복사 크기는 2000회의 시간 반복 동안 728.5 GB에 달한다.

실험 결과 CPU에서는 유한 차분식 차수 증가에 따라 계산

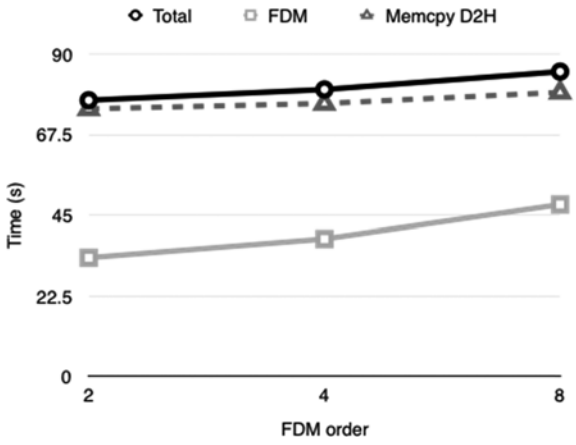

Fig. 6. Execution times of GPU programs according to the order of the finite difference equation. 'Total' curve shows the total execution time, 'FDM' curve shows the calculation time of the finite difference kernel, and 'Memcpy D2H (Device to Host)' curve shows the total memory copy time of the 3D wavefield.

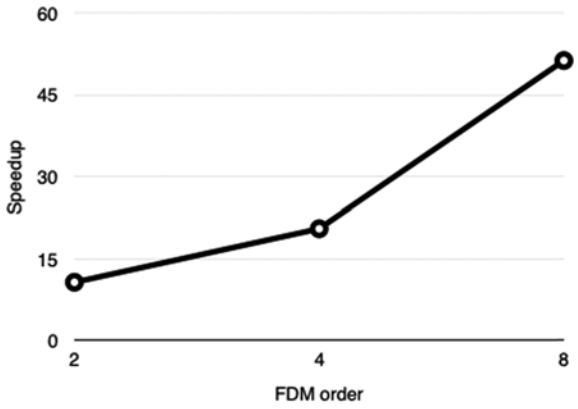

Fig. 7. Relative speedups of the GPU programs with respect to the CPU programs using the same order.

시간도 크게 증가하였다. 그러나 GPU에서는 차수가 증가해도 유한 차분식 계산 시간이 크게 증가하지 않았다(Fig. 6). 이는 공유 메모리를 이용해 격자점 접근 시간을 단축하여 계산 시간을 향상시키는 Micikevicius (2009)의 알고리즘을 적용했기 때문으로, 계산량이 많아질수록 GPU를 사용하는 것이 유리함을 알 수 있다. 파동장 복사의 경우 복사하는 배열 크기가 유사하므로 복사 시간은 예상한 것처럼 유한 차분식 차수가 증가해도 크게 달라지지 않았다. 이에 더하여 스트림을 이용해 유한 차분식 계산과 파동장 복사 과정을 중첩했기 때문에 총 시간 증가량 또한 크지 않았다. 따라서 유한 차분식 차수가 올라갈수록 GPU 프로그램의 실행 속도는 CPU 프로그램의 실행 속도와 비교하여 더 빨라지게 된다(Fig. 7).

모델 크기에 따른 비교

이번 예제에서는 모델 크기를 200 × 200 × 200, 400 × 400

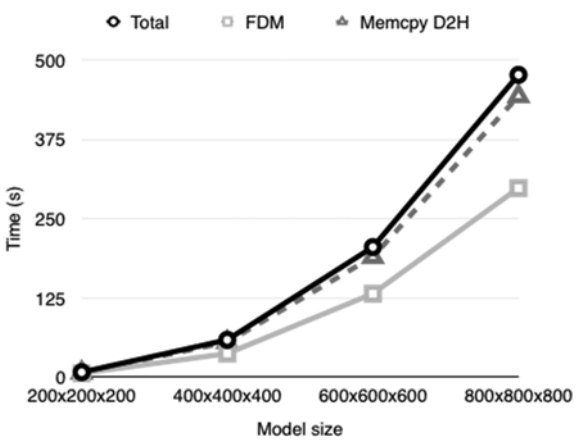

Fig. 8. Execution times of GPU programs according to the model size. 'Total' curve shows the total execution time, 'FDM' curve shows the calculation time of finite difference kernel, and 'Memcpy D2H (Device to Host)' curve shows the total memory copy time of the 3D wavefield.

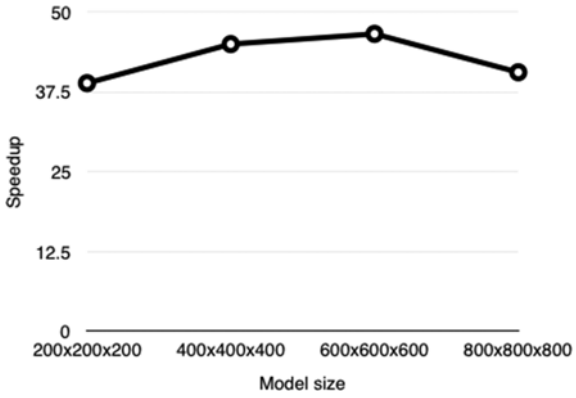

Fig. 9. Relative speedups of the GPU programs with respect to the CPU programs using the same model.

× 400, 600 × 600 × 600, 800 × 800 × 800으로 변화시켜가며 CPU와 GPU 프로그램의 성능을 비교하였다. 속도모델은 3 km/s의 상속도 모델을 사용하였다. 각각 공간에 대해 8차 유한 차분식을 사용하였고, GPU 프로그램은 모델 크기와 무관하게 20개의 스트림을 사용하였다. 모델 크기가 증가하면 유한 차분식 계산량과 GPU 메모리로부터 CPU 메모리로의 파동장 복사량이 함께 증가하게 된다.

실험 결과 모델 크기 증가에 따라 총 연산 시간 또한 비례해서 증가함을 알 수 있다(Fig. 8). 그러나 앞의 실험에서와 달리 모델 크기에 따른 성능 향상 정도는 크게 달라지지 않았다(Fig. 9). 총 연산 시간 중 메모리 복사가 차지하는 비율(Fig. 8)과 모델 크기에 따른 시간 비율(Fig. 10)을 살펴봤을 때 성능 향상 정도가 일정한 이유는 메모리 복사 때문임을 알 수 있다. Fig. 10에서 모델 크기가 증가함에 따라 총 격자수는 8배, 27배, 64배로 증가하고 CPU 프로그램의 실행 시간도 그에 비례하여 증가한다. GPU 프로그램의 경우 유한 차분식 계산 시간은 위의 비율보다 작게 증가하지만 파동장 복사 시간이 격자

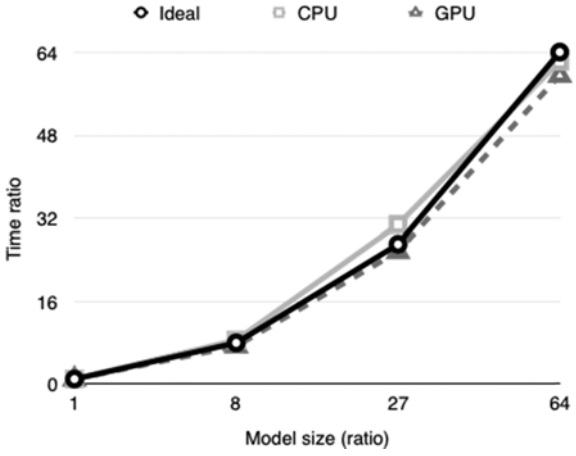

Fig. 10. Ratios of execution times to that using the 200 × 200 × 200 model. 'Ideal' curve shows the same value to the model size ratio, 'CPU' curve shows the execution time of the CPU programs, and 'GPU' curve shows the execution time of the GPU programs.

수에 비례하여 증가한다(Fig. 8). 따라서 전체 GPU 프로그램의 실행 시간도 격자 수에 비례하여 증가하고 CPU 프로그램 대비 GPU 프로그램의 속도 증가 정도도 크게 달라지지 않는다.

토 의

앞의 예제들에서의 성능 비교는 CPU의 단일 코어를 사용한 실행 속도와 비교한 것으로, 다중 코어를 이용하는 병렬 CPU 프로그램과 성능을 비교하면 GPU 프로그램의 속도 향상 정도가 떨어지게 된다. 또한 성능 최적화를 위해 공유 메모리와 스트림을 사용하도록 알고리즘을 수정한 GPU 프로그램과 달리 CPU 프로그램은 메모리 상의 배열 접근 순서를 고려해 반복문을 구성한 것 외에는 특별한 최적화를 수행하지 않았다.

모든 예제에서 단정도 부동 소수점 실수 연산을 사용하였다. GPU는 상대적으로 단정도 부동 소수점 연산 능력이 우수하므로 배정도 연산을 사용하게 되면 성능 향상 정도가 떨어지게 된다(Cheng *et al.*, 2014). 배정도 연산을 사용할 경우 메모리 복사량 또한 두 배로 증가하므로 GPU 프로그램의 실행 시간이 증가한다.

3차원 SEG/EAGE 암염돔 예제에서 상용 컴파일러인 인텔 컴파일러로 생성한 프로그램의 실행 속도는 GNU 컴파일러를 사용했을 때보다 1.9배 빨랐다. 따라서 인텔 컴파일러를 이용한 경우와 비교할 경우 GPU의 성능 향상 정도도 그만큼 떨어지게 된다.

앞에서 살펴본 결과들에서 GPU 프로그램 실행 시 그래픽 프로세서를 이용하는 계산 시간보다 CPU와 GPU 사이의 메모리 복사 시간이 더 큰 부분을 차지하는 것을 볼 수 있다. 스트림을 이용하면 계산 과정은 상당 부분 메모리 복사 과정과 중첩될 수 있기 때문에 전체 실행 시간은 메모리 복사 시간에 따라 달라진다(Fig. 4). 따라서 GPU를 사용한 거꿀 참 반사 보정이나 완전 파형 역산의 경우 알고리즘을 수정하여 계산량을 늘리는 대신 메모리 복사를 없애는 연구들도 진행되고 있다(Yang *et al.*, 2014, 2015).

결 론

시간 영역 3차원 파동 전파 모델링을 GPU로 수행할 경우 메모리 복사가 성능에 미치는 영향에 대해 살펴보았다. 단순히 CPU 프로그램을 GPU를 사용하도록 변환해도 어느 정도 향상된 성능을 얻을 수 있다. 그러나 계산 과정 최적화와 더불어 메모리 관리 또한 최적화했을 때 더욱 우수한 성능을 얻을 수 있었다. 효율적인 GPU 메모리 관리를 위해 페이지 고정 메모리를 이용해 메모리 복사 시간을 단축하고 스트림을 이용하여 계산과 메모리 복사 과정을 중첩하였다. 시간 영역 3차원 파동 전파 모델링에서는 메모리 복사량이 많기 때문에 메모리 복사가 프로그램 실행 시간의 대부분을 차지했고, 따라서 효율

적인 메모리 관리가 중요함을 확인할 수 있었다.

감사의 글

본 연구는 한국지질자원연구원 주요사업인 '고성능 석유해저 탄성파탐사 자료처리 및 다성분 탐사기술 개발' 과제(16-3313)와 산업통상자원부 자원개발특성화대학사업의 일환으로 수행되었습니다.

References

Aminzadeh, F., Burkhard, N., Nicoletis, L., Rocca, F., and Wyatt, K., 1994, SEG/EAEG 3-D modeling project: 2nd update, *The Leading Edge*, **13**, 949-952.

Cheng, J., Grossman, M., and McKercher, T., 2014, Professional CUDA C programming, Wrox.

CUDA Toolkit Documentation, 2016.8.1., http://docs.nvidia.com/cuda/

Kim, Y., Y. Cho, U. Jang, and C. Shin, 2013, Acceleration of stable TTI P-wave reverse-time migration with GPUs, *Computers and Geosciences*, **52**, 204-217.

Kirk, D.B., and Hwu, W.W., 2013, Programming massively parallel processors, 2nd ed., Morgan Kaufmann.

Komatitsch, D., D. Michéa, and G. Erlebacher, 2009, Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, *J. Parallel Distrib. Comput.*, **69**, 451-460.

Komatitsch, D., G. Erlebacher, D. Göddeke, and D. Michéa, 2010a, High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, *Journal of Computational Physics*, **229**, 7692-7714.

Komatitsch, D., D. Göddeke, G. Erlebacher, and D. Michéa, 2010b, Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs, *Computer Science - Research and Development*, **25**, 75-82.

Kreyszig, E., 2011, Advanced engineering mathematics, 10th Ed., John Wiley & Sons, Inc.

Liu, G., Y. Liu, L. Ren, and X. Meng, 2013, 3D seismic reverse time migration on GPGPU, *Computers and Geosciences*, **59**, 17-23.

Michéa, D., and D. Komatitsch, 2010, Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards, *Geophysical Journal International*, **182**, 389-402.

Micikevicius, P., 2009, 3D finite difference computation on GPUs using CUDA, Proceedings of 2nd Workshop on General Purpose GPU.

Mu, D., P. Chen, and L. Wang, 2013, Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using multiple GPUs with CUDA and MPI, *Earthquake Science*, **26**, 377-393.

Shi, X., C. Li, S. Wang, and X. Wang, 2010, Computing prestack Kirchhoff time migration on general purpose GPU, *Computers and Geosciences*, **37**, 1702-1710.

Shin, J., W. Ha, H. Jun, D.-J. Min, and C. Shin, 2014, 3D Laplace-domain full waveform inversion using a single GPU card, *Computers and Geosciences*, **67**, 1-13.

Suh, S., and B. Wang, 2011, Expanding domain methods in GPU based TTI reverse time migration, *SEG Expanded Abstract Technical Program*, 3460-3464.

Wang, S.-Q., X. Gao, and Z.-X. Yao, 2010, Accelerating POCS interpolation of 3D irregular seismic data with Graphics Processing Units, *Computers and Geosciences*, **36**, 1292-1300.

Weiss, R. M., and J. Shragge, 2013, Solving 3D anisotropic elastic wave equations on parallel GPU devices, *Geophysics*, **78**, F7-F15.

Yang, P., J. Gao, and B. Wang, 2014, RTM using effective boundary saving: A staggered grid GPU implementation, *Computers and Geosciences*, **68**, 64-72.

Yang, P., J. Gao, and B. Wang, 2015, A graphics processing unit implementation of time-domain full-waveform inversion, *Geophysics*, **80**, F31-F39.

부록 A: 페이지 고정 메모리

일반적인 운영체제는 물리적 메모리 공간보다 더 큰 메모리 공간을 사용할 수 있도록 가상 메모리(virtual memory)를 사용한다. 메모리의 내용 중 덜 사용하는 부분을 디스크의 스왑 공간(swap space)에 저장해 두었다가 필요할 때 다시 메모리로 복사하는 방식인데, 이를 위해 운영체제는 응용 프로그램에 실제 물리적 메모리 주소가 아닌 가상 주소(virtual address)를 제공한다. 응용 프로그램에서는 가상 주소를 이용해 공간 제약이 없는 것처럼 메모리를 사용하고 운영체제에서 가상 주소를 실제 물리적 주소로 바꿔주는데 이때 주소들을 저장해 놓은 페이지 표(page table)을 사용한다. 이렇게 운영체제가 관리하는 메모리를 페이지 가능 메모리라 한다.

현재 CPU 메모리와 GPU 메모리 사이의 통신을 위해서는 PCI-E (Peripheral Component Interconnect Express) 통신을 이용하는데, 이를 위해서 CPU 메모리는 페이지 고정 메모리를 사용해야 한다. 따라서 일반적인 CPU 메모리의 내용을 GPU 메모리로 복사할 경우 페이지 가능 메모리에 있는 내용을 페이지 고정 메모리에 복사한 후 GPU로 복사하게 된다. 반면에 페이지 고정 메모리를 사용하게 되면 CPU 메모리 내에서의 복사 과정 없이 바로 GPU 메모리에 복사하게 되므로 통신 속도가 향상된다.

Table A1에 페이지 가능 메모리와 페이지 고정 메모리를 할당하고 해제하는 예를 보였다. 페이지 고정 메모리를 사용하기 위해서는 코드에서 볼 수 있는 것처럼 cudaMallocHost 또는 cudaHostAlloc 함수를 사용하여 배열의 메모리 공간을 할당한다. 일단 배열을 할당하면 두 경우의 배열 사용법은 동일하다. 페이지 고정 메모리의 경우 메모리 용량에 따라 할당이 실패

Table A1. A code snippet showing how to use the page-locked memory.

```
float *u1, *u2
// Pageable memory
u1 = (float*) malloc(nbytes);
free(u1);

// Page-locked (pinned) memory
cudaError_t status = cudaMallocHost((void**) &u2, nbytes);
if(status != cudaSuccess)
  printf("Error allocating pinned host memory!\n");
cudaFreeHost(u2);
```

할 수 있으므로 예시에 보인 것과 같이 오류를 확인하는 것이 바람직하다.

부록 B: CUDA 스트림

최근의 GPU 카드들은 GPU 계산 과정과 메모리 복사 과정을 중첩시키는 기능을 제공한다. 카드 사양 및 사용하는 함수에 따라 계산 과정들끼리의 중첩, CPU 메모리 읽기와 쓰기 과정의 중첩 또한 가능하다. 이러한 중첩 기능을 이용하기 위해서는 CUDA에서 스트림을 이용해야 한다. 동일한 스트림 내에서는 계산과 메모리 복사를 중첩할 수 없지만 다른 스트림끼리는 계산과 메모리 복사를 중첩할 수 있다.

Table B1에 스트림을 사용하는 코드를 일부 제시하였다. 스트림을 사용하기 위해서는 먼저 몇 개의 스트림(nstream)을 만들지 결정한 후 스트림을 생성한다(cudaStreamCreate). 스트림 생성 후에는 각 스트림별로 필요한 함수를 실행하고 실행 결과를 복사한다. 본 연구에서는 깊이 방향으로 영역을 분해하여 각 스트림에 할당하였기 때문에 유한 차분식 구현 함수의 인자로 영역의 시작과 끝 정보를 전달하였다. 실행 결과를 복사할 때에는 비동기 복사 함수(cudaMemcpyAsync)를 이용해야 다른 스트림의 계산 과정과 중첩이 가능하다. 이런 방식으로 실행하면 Fig. 4에 보인 것과 같이 계산과 메모리 복사 과정을 중첩하게 된다. 스트림끼리 동기화가 필요할 경우 동기화 함수(cudaStreamSynchronize)를 이용하고 스트림이 더이상 필요 없으면 생성했던 스트림을 제거한다(cudaStreamDestroy).

Table B1. A code snippet showing how to use the CUDA stream.

```
// Initialize
cudaStream_t streams[num_streams];
for(istream = 0; istream < nstream; istream++)
 cudaStreamCreate(&streams[istream]);

// Launch CUDA kernel
for(istream = 0; istream < nstream; istream++)
{
  cuda_kernel_function<<<dimGrid, dimBlock, 0,
  streams[istream]>>>(…);
  cudaMemcpyAsync(…, cudaMemcpyDeviceToHost,
  streams[istream]);
}

// Synchronize
for(istream = 0; istream < nstream; istream++)
  cudaStreamSynchronize(streams[istream]);

// Finalize
for(istream = 0; istream < nstream; istream++)
  cudaStreamDestroy(streams[istream]);
```