<학술논문> DOI https://doi.org/10.3795/KSME-B.2017.41.2.117 ISSN 1226-4881(Print)
2288-5324(Online)

Coloring이 적용된 Gauss-Seidel 해법을 통한 CPU와 GPU의 연산 효율에 관한 연구

윤종선* · 전병진** · 최형권***†
* 서울과학기술대학교 기계공학과, ** 연세대학교 의과대학 심혈관영상연구센터,
*** 서울과학기술대학교 기계·자동차공학과

An Investigation of the Performance of the Colored Gauss-Seidel Solver on CPU and GPU

Jong Seon Yoon*, Byoung Jin Jeon** and Hyoung Gwon Choi***†
* Dept. of Mechanical Engineering, Seoul Nat'l Univ. of Science and Technology,
** Integrative Cardiovascular Imaging Research Center, Yonsei Cardiovascular Center, College of Medicine, Yonsei Univ.,
*** Dept. of Mechanical and Automotive Engineering, Seoul Nat'l Univ. of Science and Technology.

(Received October 13, 2016 ; Revised November 3, 2016 ; Accepted November 8, 2016)

Key Words: GPU, CFD(전산유체역학), Finite Element Method(유한요소법), Finite Differential Method(유한차분법), Coloring Method(컬러링 기법), Gauss-Seidel Solver(가우스지달 해법)

초록: 본 연구에서는 Coloring 기법을 적용한 Gauss-Seidel 해법의 연산 성능을 분석하기 위해 2차원과 3차원 전도 열전달 문제를 다양한 격자 크기에서 해석하였다. 지배방정식의 이산화는 유한차분법과 유한요소법을 사용하였다. CPU의 경우에는 상대적으로 작은 격자계에서 연산 성능이 좋으며, 계산에 사용되는 메모리의 크기가 캐시메모리보다 크게 되면 연산 성능이 급격히 떨어진다. 반면에, GPU는 메모리 지연시간 숨김 특성으로 인하여 격자의 수가 충분히 많을 때 연산 성능이 좋다. GPU에 기반한 Colored Gauss-Seidel 해법은 단일 CPU를 이용한 연산에 비해서 각각 최대 7배의 속도 향상을 보인다. 또한, GPU 기반에서 Colored Gauss-Seidel 해법은 Jacobi 보다 약 2배 빠름을 확인하였다.

Abstract: The performance of the colored Gauss-Seidel solver on CPU and GPU was investigated for the two- and three-dimensional heat conduction problems by using different mesh sizes. The heat conduction equation was discretized by the finite difference method and finite element method. The CPU yielded good performance for small problems but deteriorated when the total memory required for computing was larger than the cache memory for large problems. In contrast, the GPU performed better as the mesh size increased because of the latency hiding technique. Further, GPU computation by the colored Gauss-Siedel solver was approximately 7 times that by the single CPU. Furthermore, the colored Gauss-Seidel solver was found to be approximately twice that of the Jacobi solver when parallel computing was conducted on the GPU.

1. 서 론

최근 이공계 분야에서는 실험을 하기 어려운 조건이거나 해석 대상의 스케일이 매우 큰 문제들의 해를 구하기 위해 컴퓨터를 이용하는 수치해석적인 연구가 많이 수행되고 있다. 이와 관련하여 하드웨어와 소프트웨어를 개발하는 회사들은 보다 정확하고 빠른 연산을 원하는 소비자들의 요구를 충족시키기 위해 새로운 메커니즘을 기반으로 하는 제품들을 출시하고 있다. 특히, 최근 하드웨어 시장에서는 연산을 수행하는 코어의 수가 수십 개 이상이고 기존의 클러스터 구조보

† Corresponding Author, hgchoi@seoultech.ac.kr

다 에너지를 절감할 수 있는 Intel Xeon-phi 또는 GPU 가 주목을 받고 있다.

컴퓨터 이용하여 미분방정식으로 표현된 유체 또는 고체 방정식을 풀기 위해서는 유한요소법 등의 수치해석 기법을 이용하여 이산화를 수행해야 한다. 이산화하여 얻은 행렬을 푸는 방법은 직접 또는 반복 해법이 있다. 일반적으로는 Jacobi, Gauss-Seidel, CG(Conjugate Gradient) 해법 등과 같은 반복 해법이 직접 해법보다 효율성이 좋기 때문에 많이 사용된다.

반복 해법 중에서 Jacobi 방식은 구하고자 하는 값과 이전 값이 완전히 독립적인 배열로 저장되기 때문에 GPU 등의 many-core 기반 시스템에서 효율적으로 동작할 수 있다. Wang 등[1]은 GPU 기반의 시스템에서 다양한 문제 크기에 대해 Jacobi 해법을 이용하여 해를 구하였으며, 단일 CPU 보다 약 3배 빠른 연산 속도를 얻을 수 있었다. Julien 등[2]은 2개의 GPU를 이용할 수 있는 시스템에서 약 300만개의 격자 크기를 가진 문제에 대해 Jacobi 해법을 사용하여 해를 얻었다. 그들은 압력 방정식을 풀 때 걸리는 시간을 단일 CPU 대비 20배 줄일 수 있었다. 또한, Dana 등[3]은 2개의 GPU 클러스터 구조에서 Jacobi 해법을 기반으로 CUDA, MPI, OpenMP를 결합하여 효율을 분석하였으며, 단일 CPU 대비 최대 26배의 성능향상을 얻었다. Dana 등[4]은 GPU의 개수에 따른 병렬 성능 특성을 파악하기 위해 128개의 GPU로 구성된 시스템에서 MPI와 CUDA 기반으로 개발한 Jacobi 알고리즘을 이용하여 Lid-driven cavity 문제를 해석하였다.

일반적으로, Gauss Seidel이나 SOR 해법은 Jacobi 해법보다 더 빠른 수렴성을 가지고 있다고 알려져 있다. 그러나, 전자의 경우에는 이전 값과 구하고자 하는 값의 사이에 데이터 의존성을 가지고 있기 때문에 병렬화에 제한이 있다. 이러한 문제를 극복하기 위해 많은 연구자들은 Coloring 알고리즘[5]을 제안하였다. Hsieh 등[6]은 다양한 2차원 쌍곡선형 문제들을 Red-Black Gauss-Seidel을 이용하여 GPU 기반에서 해석을 수행하였으며, 11배의 속도향상을 얻었다. Kuo 등[7]은 2차원 다상유동에 관하여 Red-Black SOR을 적용하였으며 GPU 환경에서 단일 CPU 대비 12배의 성능 향상을 확인하였다. 또한 Li 등[8]은 Jacobi 또는 Coloring이 적용된 Gauss Seidel을 예조건화로 사용하는 CG, GMRES 알고리즘에 대해서 예조건화에 따른 수렴성과 병렬 연산 성능을 분석하기 위해 다양한 표준 문제들을 풀었다.

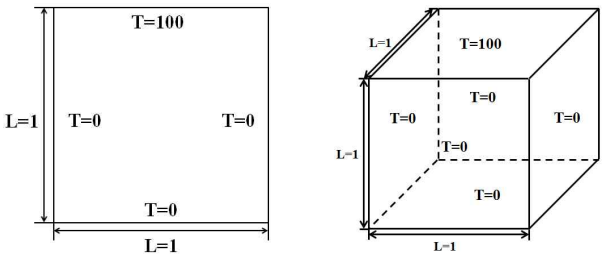
위와 같은 다양한 문제들에 대하여 수행한 선행 연구를 통해, GPU는 계산량이 많은 문제에서 캐쉬 메모리의 영향을 많이 받는 CPU보다 연산 시간 및 에너지 절약 측면에서 더 효율적인 것을 알 수 있다. 그러나, 동일한 문제를 다양한 격자 수로 구성하여 이산화 방식과 반복 해법 종류에 따른 GPU의 병렬 성능을 정량적으로 비교한 연구들은 미비한 실정이다.

따라서, 본 연구에서는 2차원 또는 3차원 전도 열전달 문제를 이산화 방법에 따라 CPU와 GPU를 이용하여 해석하였다. 이산화는 Jacobi와 Multi Colored Gauss-Seidel 해법을 이용하였으며, 수행된 결과를 이용하여 CPU와 GPU의 구조적 차이를 확인하고 격자 크기에 따른 CPU 혹은 GPU의 효율성을 정량적으로 분석하였다.

2. 연구 방법

Fig. 1은 2차원 혹은 3차원 전도 열전달 문제에 대한 간단한 개략도이며, 온도경계 조건은 위쪽 면에 100, 나머지 면에 대해서는 0을 주었다. 또한, 본 연구에서 사용한 CPU와 GPU는 E5-2620 v3 @2.4 GHz와 Nvidia Tesla K40이며, 컴파일러는 intel compiler 16.0과 CUDA toolkit 7.5를 사용하였다.

이번 장에서는 본 연구에서 사용된 Jacobi 해법과 Gauss Seidel 해법을 다루고자 한다. 그러나, 일반적인 Gauss-Seidel 해법은 인접한 노드에 대하여 의존성을 가지고 있기 때문에 다량의 코어를 동시에 다루는 GPU의 구조에 맞지 않다. 이러한 문제를 해결하기 위해서는 잘 알려져 있는 Coloring 기법을 이용해야 한다.



(a) 2-D problem (b) 3-D problem

Fig. 1 Schematic of heat conduction problem

2.1 Jacobi와 Gauss-Seidel 해법

차분법에 따른 성능 테스트를 수행하기 위해, 2차원, 3차원 전도 열전달 방정식에 대해서 유한차분법과 유한요소법을 이용하여 이산화를 수행하였다. 2차원 문제에 대해서 유한차분법과 유한요소법을 적용한 경우의 스텐실(stencil)은 5개 또는 9개이고, 3차원의 경우에는 7개 또는 21개가 된다.

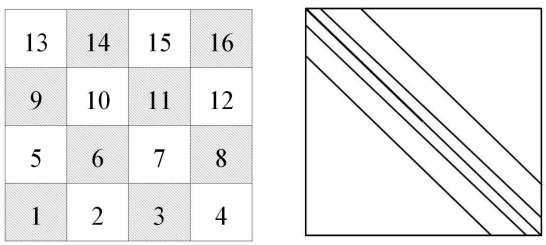
Jacobi 해법은 k번째 $x_i^{(k)}$를 얻기 위하여 다음과 같은 식을 사용한다.[9]

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\sum_{\substack{j=1 \\ j \neq i}}^{n} \left(-a_{ij} x_j^{(k-1)} \right) + b_i \right] \quad (1)$$

Gauss-Seidel 해법은 k번째 $x_i^{(k)}$를 얻기 위하여 다음과 같은 식을 사용한다.[9]

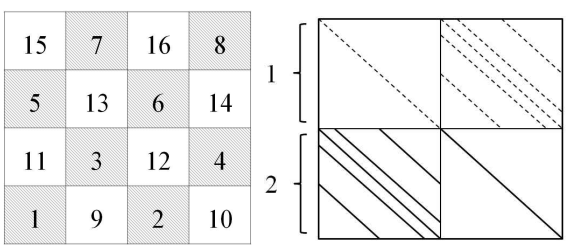
$$x_i^{(k)} = \frac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} (a_{ij} x_j^{(k)}) - \sum_{j=i+1}^{n} (a_{ij} x_j^{(k-1)}) + b_i \right] \quad (2)$$

Gauss-Seidel 해법은 $x_i^{(k)}$을 구하기 위하여 $x_i^{(k)}(j<i)$를 이용하게 되는데, 이것이 의존성 문제를 발생시켜 병렬성을 저하시키는 원인이다.



numbering of mesh pattern of matrix

(a) Before Red-Black coloring



numbering of mesh pattern of matrix

(b) After Red-Black coloring

Fig. 2 Schematic of coloring

2.2 Multi-Coloring 방법

본 연구에서는 Gauss-Seidel 해법의 의존성 문제를 해결하기 위하여 Coloring 방법을 사용한다. Fig. 2는 2차원 정렬 격자계에서 사용하는 유한차분법 기반의 Red-Black Coloring 방법을 보여준다. Coloring 방법은 인접한 노드에 기준 노드와 다른 색깔을 부여한 뒤에, 같은 색들을 기준으로 노드의 순서를 재배열하는 것이다. Fig. 2의 (a)는 Coloring을 적용하기 전의 노드 번호와 행렬 구조를 나타내고, (b)는 Coloring 적용하여 재배열 된 노드 번호와 행렬 구조를 보여준다. Coloring을 적용하면, 그림에 표시한 1번 영역을 먼저 계산 후에 2번 영역을 계산할 수 있는 의존성은 여전히 존재하지만 각 영역 내에서는 의존성이 사라지게 되어 병렬로 연산을 할 수 있다. 2차원 정렬 격자계에서 유한 차분법과 유한 요소법은 스텐실에 따라 각각 2개와 4개의 색을 가지며, 3차원 정렬 격자계에서는 2개와 8개의 색을 가지게 된다.

3. GPU 병렬화

본 연구에서는 GPU를 이용한 병렬 연산을 위하여 CUDA 언어를 사용하였다. GPU를 효율적으로 이용하기 위해서는 GPU 구조와 특징을 이해하는 것이 중요하다. 따라서 3장에서는 CUDA의 프로그래밍 모델과 메모리 구조, GPU를 효율적으로 이용할 수 있는 방법에 대하여 설명하였다.

3.1 GPU의 스레드 블록 모델

Fig. 3은 CUDA의 실행 모델을 보여준다. 커널(kernel)은 GPU에서 실행되는 함수를 지칭한다. GPU를 사용하기 위해 호스트에서 커널을 호출하면, 사용자가 지정한 만큼의 스레드(thread)와 블록(block)을 생성한다. 여기서, 스레드는 하나의 GPU 코어를 뜻하며, 블록은 스레드들의 집합을 의미한다. 블록 내의 스레드들은 연산을 수행할 때 워프(Warp) 단위의 개념으로 나눠진다. 워프는 GPU 내의 최소 작업 단위로 동시에 같은 명령어(instruction)를 수행하며, 32개의 스레드로 구성된다.

3.2 GPU 메모리 구조

Fig. 4는 GPU의 메모리 구조를 보여준다. GPU는 내부의 독립적인 메모리를 사용하기 때문에,

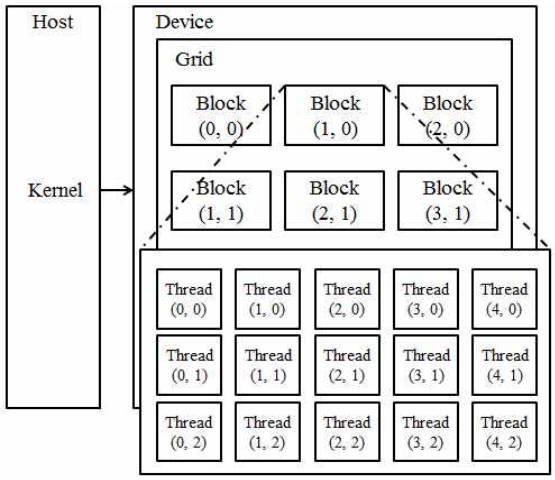


Fig. 3 Thread block model [10]

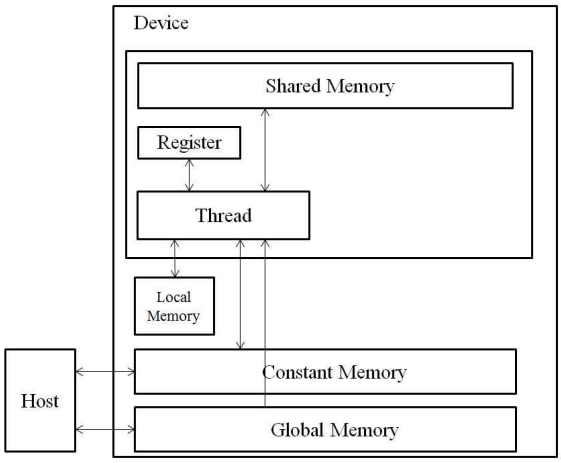


Fig. 4 CUDA Memory Model [10]

CPU에서 사용하는 메모리에 저장된 데이터를 공유할 수 없다. 따라서, GPU 연산을 수행하기 위해서는 CPU의 메모리로부터 필요한 데이터를 가지고 오는 과정이 필요하다. 다양한 GPU 메모리는 사용자가 직접 제어하여 사용할 수 있기 때문에 적절히 사용한다면 좋은 병렬 효과를 얻을 수 있다.

레지스터(Register)는 가장 빠르고 컴파일러에 의해 자동으로 할당되는 메모리다. 공유 메모리(Shared memory)는 GPU 1 사이클 이내를 소비하며, CPU의 L2 캐시와 같은 속도이고 사용자가 직접 제어할 수 있는 장점이 있다. 상수 메모리(Constant memory)는 읽기 전용 메모리이며 L2 캐시와 같은 속도로 이용할 수 있다. 전역 메모리(Global memory)라고 불리는 DRAM 메모리는 많은 데이터를 저장할 수 있지만 물리적으로 GPU 외부의 칩(Off chip)에 위치하기 때문에 GPU 400~600 사이클을 소비하는 매우 느린 메모리다. 지역메모리(Local memory)는 커널 내의 레지스터의 자원이 모자랄 때 컴파일러 스스로 할당시키는 메모리며, 전역 메모리와 같이 DRAM에 존재하기 때문에 매우 느린 속도를 가지고 있다.

3.3 지연시간 숨김(Latency hiding)

명령어가 실행되고 그 다음 명령어를 실행하기 전까지 기다리는 시간을 지연시간(Latency)이라고 한다. GPU는 매우 많은 코어를 동시에 다루도록 설계되어 있지만 코어의 개수가 CPU 보다 상대적으로 많기 때문에 지연시간이 증가할 수 있다. 이러한 단점을 극복하기 위하여 GPU는 지연시간

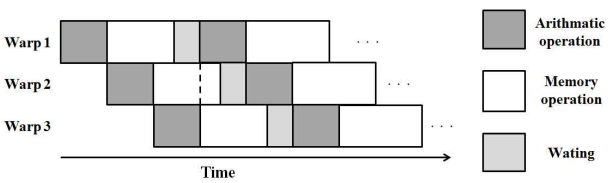


Fig. 5 Latency hiding

을 숨길 수 있도록 설계되어 있다. GPU는 문맥전환(Context switching)의 부하가 매우 작을 뿐만 아니라 연산 유닛과 메모리 유닛이 분리되어 있기 때문에, 메모리 작업 혹은 연산 작업의 지연이 발생하면 작업을 가져와 연산 혹은 메모리 작업을 실행할 수 있다. 이를 제어하는 유닛을 워프 스케줄러(Warp scheduler)라고 하며, Tesla K40은 56개의 워프 스케줄러를 가지고 있고 896개의 워프를 동시에 실행할 수 있다.

Fig. 5는 지연시간 숨김 효과의 간단한 예시를 보여준다. 예를 들어, 1개의 워프 스케줄러(Warp scheduler)에 실행 가능한 워프가 3개가 있다고 가정한다. warp 1에서 메모리 사용에 의한 지연시간이 발생하면 곧바로 warp 2를 할당하여 실행 시킨다. 마찬가지로 warp 2에서 지연시간이 발생하면 warp 3을 실행 시킨다. 이 때, warp 1은 warp 3의 작업이 완료될 때까지 기다리게 된다. 만약 실행 가능한 워프가 없다면 워프 스케줄러는 가용 워프가 생길 때까지 대기하게 된다. GPU 기반의 프로그래밍에서는 이런 지연시간을 숨기기 위하여 최대한 많은 워프를 만들어서 적절한 연산량 이상을 보장하는 것은 중요하다.

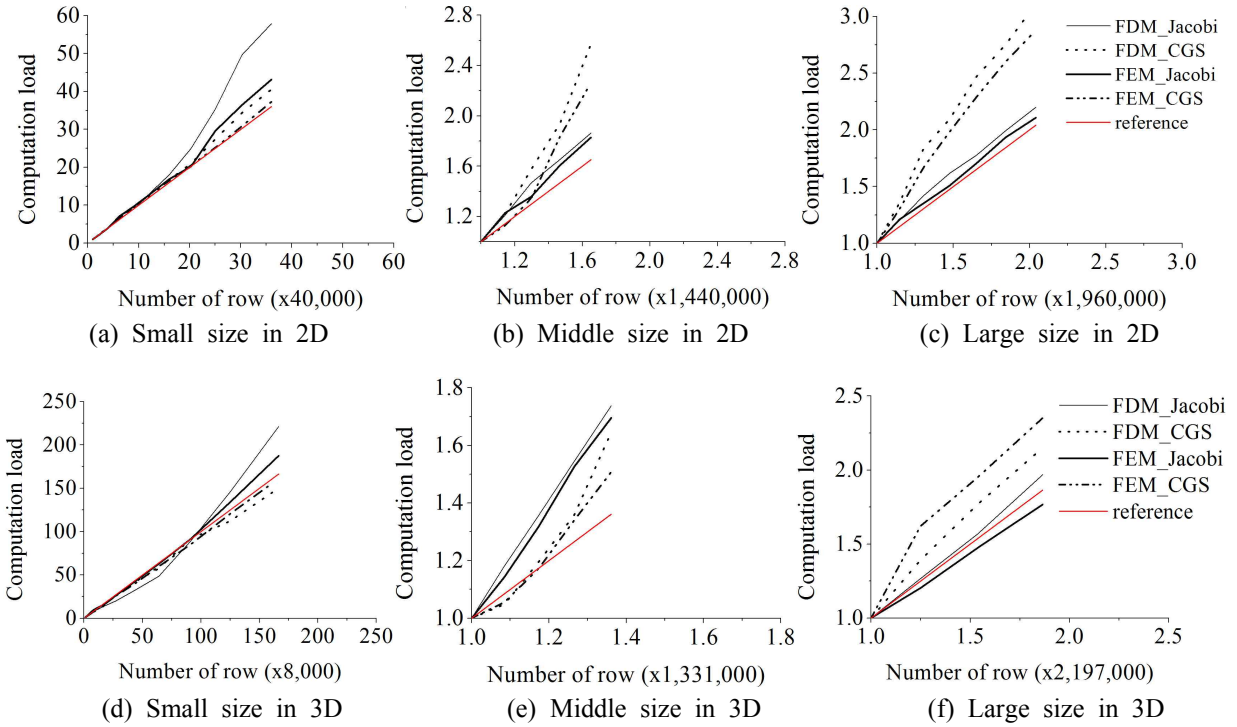


Fig. 6 Computation load of CPU with various size

4. 해석 결과 및 토의

본 연구에서는 C언어와 CUDA 언어로 작성된 Jacobi 해법과 Coloring이 적용된 Gauss-Seidel(이하, CGS)을 이용하여 전도 열전달 문제를 해석하였으며, 다양한 격자 크기의 대해서 CPU 대비 GPU의 속도 향상과 병렬 특성에 대해서 연구하였다. 2차원 또는 3차원 전도 열전달 문제는 정렬 격자계에서 유한차분법과 유한요소법으로 이산화하였다.

Fig. 6과 Fig. 7은 격자 크기에 따른 Jacobi 해법과 CGS의 연산 부하를 보여준다.

$$\text{Time}_{it} = \text{Elapsed time} \;/\; \text{Number of iteration} \quad (3)$$

Time_{it}는 전체 연산 시간을 반복 횟수로 나눈 값이며, Fig. 6과 Fig. 7의 x축은 격자의 크기를 나타내고 y축은 각 격자계의 Time_{it}를 기준 격자계의 Time_{it}로 나눈 상대적인 값을 나타낸다. 2차원 문제에 대한 기준 격자계는 40,000개, 1,440,000개, 1,960,000개이고, 3차원 문제에서는 8,000개, 1,331,000개, 2,197,000개이다. 기준선(reference line)은 예를 들어 계산량(격자의 크기)이 2배 증가할 때 연산시간이 2배 증가함을 나타내는 지표이다. 결과 값이 기준선보다 아래에 위치하게 되면 계산량이 적어 하드웨어의 자원을 충분히 사용하지 못하는 상태를 의미하며, 기준선보다 위쪽에 위치하면 하드웨어가 가진 자원보다 많은 일을 받아서 캐시 메모리 등을 효율적으로 사용하지 못하는 부하가 발생하는 것을 의미한다.

Fig. 6의 (a)와 (d)는 2차원과 3차원 문제에 대해서 상대적으로 작은 격자 크기의 병렬 성능을 확인한 결과이며, 각각 40,000개, 8,000개 격자의 연산 시간을 기준으로 무차원화 하였다. Jacobi 해법의 경우에는 선형적으로 증가하다가 행렬의 크기가 약 800,000개에서 그래프 기울기가 급격히 증가한다. 이것은 그 지점에서 캐시 메모리 용량(15MB) 초과로 연산에 필요한 데이터를 추가로 호출해야하기 때문에 급격히 느려지는 것이다. CGS 해법의 경우에는 비교적 일정하게 증가하는 것으로 확인된다.

Fig. 6의 (b)와 (e)는 격자크기가 약 1,400,000개에서 약 2,000,000개 사이에 있는 경우들에 대하여 확인한 결과이다. 이 구간에서 2차원과 3차원은 각각 1,440,000개와 1,331,000개의 연산시간을 기준으로 무차원화 하였다. 격자가 커짐에 따라

Table 1 Number of iteration with various case in 2-D problem

Matrix Size		100^2	200^2	400^2	800^2
FDM	Jacobi	26,997	97,961	349,079	1,220,449
	CGS	14,187	51,762	185,720	655,056
FEM	Jacobi	18,536	67,477	241,440	848,599
	CGS	9,727	35,593	128,174	454,188

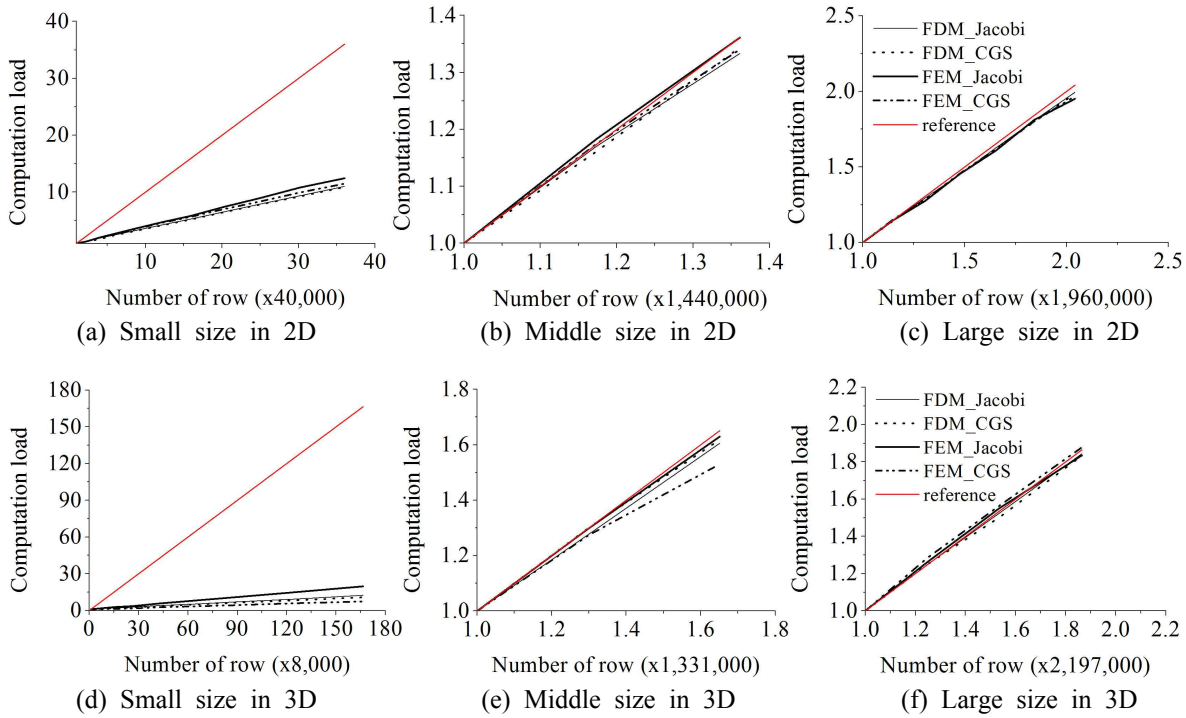


Fig. 7 Computation load of GPU with various size

유한차분법 혹은 유한요소법의 해법들은 모두 이상적인 기준선보다 위쪽에 위치하며, 선형적으로 증가하지 않는다. 이 구간에서 CGS의 2차원과 3차원 그래프 기울기가 약 1,500,000개의 격자 크기에서 급격하게 증가하는 것을 볼 수 있는데, 이것은 Jacobi 해법의 현상과 마찬가지로 캐시 메모리 초과로 인한 데이터 호출이 많아졌기 때문이다.

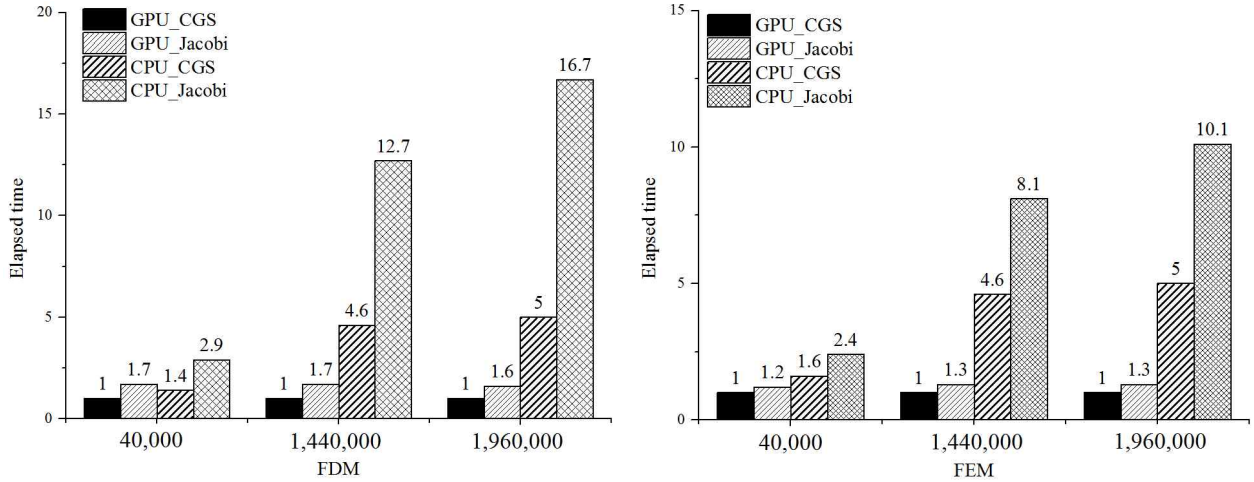
Fig. 6의 (c)와 (f)는 격자 크기가 약 2,000,000개일 때의 연산 시간을 기준으로 무차원화 하였다. 2차원과 3차원 문제의 결과가 모두 이상적인 기준선보다 위쪽에 위치하기 때문에, CPU는 큰 격자 구간에서 효율이 크게 안 좋은 것을 확인할 수 있다.

Fig. 7의 (a)와 (d)는 GPU에서 2차원과 3차원 문제에 대하여 40,000개와 8,000개의 연산 시간 기준으로 무차원 시킨 값이다. 선행연구들에서 이미 보고한 바와 같이, GPU는 작은 격자에 대하여 효율이 좋지 못함을 알 수 있다.

Fig. 7의 (b)와 (e)는 2차원과 3차원 문제에 대하여 각각 약 1,400,000개와 1,300,000개 격자 크기의 연산 시간 기준으로 무차원한 값으로 기준선에 근접해지는 것을 볼 수 있다.

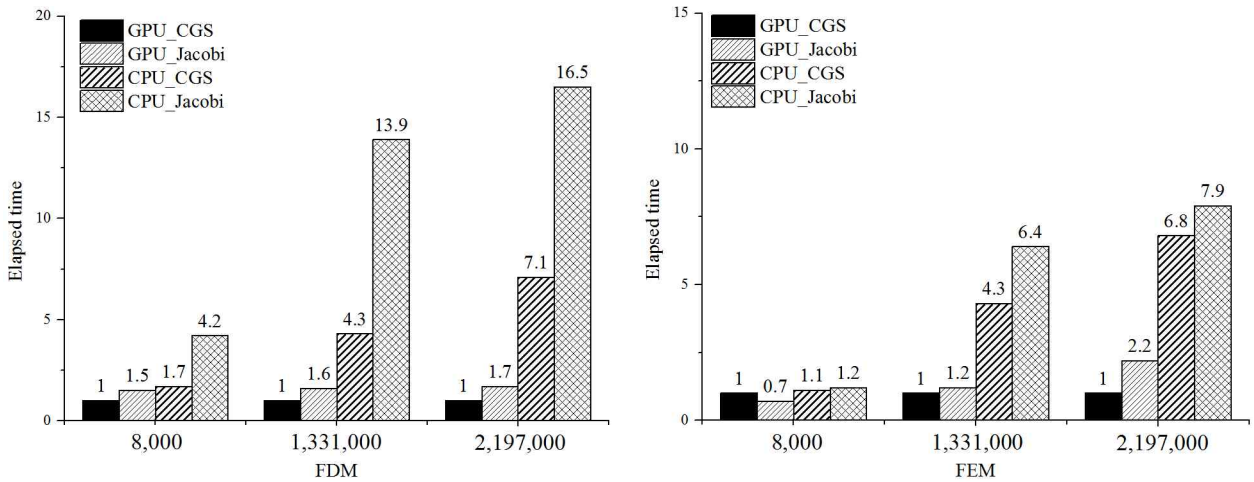
Fig. 7의 (c)와 (f)는 각 차원에 대하여 약 2,000,000개 격자 크기의 연산시간을 기준으로 무차원한 값으로 격자가 커짐에도 기준선과 비슷한 경향을 보이는 것을 확인할 수 있다. Fig. 7은 GPU가 CPU보다 큰 격자에 대하여 더 효율적인 것을 보여준다. 이것은 위에서 언급한 지연시간 숨김에 따른 결과로 연산량이 충분히 보장되었기 때문에 좋은 효율을 보이는 것이다.

Table 1은 2차원 문제에서 격자가 2^2배로 커질



(a) Case of FDM (b) Case of FEM

Fig. 8 Elapsed time in 2-D problem



(a) Case of FDM (b) Case of FEM

Fig. 9 Elapsed time in 3-D problem

때, 수렴된 결과를 얻을 때까지 걸리는 Jacobi 해법과 CGS 해법의 반복회수를 나타낸 것이다. 격자가 2^2배 커지면 반복회수가 약 2^2배정도 증가하며, CGS 해법은 Jacobi 해법보다 약 2배정도 적은 반복회수를 가지고 있음을 확인할 수 있었다.

Fig. 8과 Fig. 9는 각각 2차원과 3차원 문제에 대하여 유한차분법과 유한요소법의 전체 연산 시간을 비교한 것이다. 결과들은 각 구간의 대표 격자로 무차원한 상대 값으로 표현하였다. 본 연구에서는 GPU에서 유한요소법 기반의 CGS 방법을 이용하여 해석했을 때 가장 빠른 연산 시간을 얻을 수 있었으며 단일 CPU 연산보다 최대 7배의 성능 향상을 얻을 수 있었다. 또한, Jacobi 해법을 이용한 GPU 연산의 경우에는 단일 코어를 이용한 CPU 연산과 비교하여 최대 5배의 속도 향상을 보인다.

5. 결 론

본 연구에서는 2, 3차원 전도 열전달 문제를 다양한 크기의 정렬 격자계에서 CPU와 GPU를 이용하여 해석하고 GPU의 병렬 성능을 확인하였다. 또한, Gauss-Seidel 해법의 단점을 극복하기 위하여 도입된 Coloring 기법을 GPU에 적용하고 그 특성을 파악하였으며, Jacobi 해법과 비교하여 아래와 같은 결론을 도출하였다.

(1) 단일 코어를 이용한 CPU 연산에서는 Jacobi 해법을 적용하였을 때, 약 800,000개의 격자 이상

에서 캐시 메모리의 부족으로 인해 연산 성능이 급격히 안 좋아진다. Jacobi 해법보다 1/2배 적은 메모리를 사용하는 Colored Gauss-Seidel(CGS) 은 약 1,500,000개의 격자 이상에서 연산 성능이 안 좋아짐을 확인하였다.

(2) GPU는 매우 많은 코어를 사용하는 것과 메모리 지연 시간 숨김 특성으로 인하여, 연산 방법과 무관하게 약 1,500,000개의 이상의 격자에서 병렬 성능이 좋음을 알 수 있다.

(3) Jacobi 또는 CGS 방법에 따른 GPU 연산은 단일 코어를 이용한 CPU 연산과 비교하여 2차원과 3차원에서 각각 최대 7배의 속도향상을 보인다.

(4) CGS 해법의 계산 소요 시간은 Jacobi 해법보다, CPU 또는 GPU 계산에서 각각 최대 3배, 2배 빠름을 확인하였다.

후 기

본 연구는 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원(No.R0101-15-0171, 다중의료영상을 활용한 3차원 초정밀 시뮬레이션 기반 심·혈관 질환 진단-치료지원 통합 소프트웨어 시스템 개발)과 한국연구재단의 지원(No.2014R1A2A2A01004879, 표면장력이 지배적인 유동장의 해석을 위한 수정된 속도장과 표면장력의 완전내재적인 방법을 적용한 레벨셋 기법에 대한 연구)으로 수행된 연구임.

참고문헌
(References)

(1) Wang, T., Yao, Y., Han, L., Zhang, D. and Zhang, Y., 2009, "Implementation of Jacobi Iterative Method on Graphics Processor Unit," *Intelligent Computing and Intelligent Systems*, *IEEE International Conference*, Vol. 3, pp. 324~327.

(2) Thibault, J. C. and Senocak, I., 2009, “CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows,” *Proceedings of the 47th AIAA aerospace sciences meeting*, p. 758.

(3) Jacobsen, D. A. and Senocak, I., 2013, "Multi-level Parallelism for Incompressible Flow Computations on GPU Clusters," *Parallel Computing*, Vol. 39, No. 1, pp. 1~20.

(4) Jacobsen, D. A., Thibault, J. C. and Senocak, I., 2010, "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters," *In 48th AIAA aerospace sciences meeting and exhibit*, Vol. 16, p. 2.

(5) Kubale, M., 2004, *Graph Colorings*, American Mathematical Society, Rhode Island, pp. 1~20.

(6) Hsieh, C. W., Kuo, S. H., Kuo, F. A. and Chou, C. Y., 2010, "Solving Parabolic Problems Using Multithread and GPU," *In International Symposium on Parallel and Distributed Processing with Applications*, *IEEE*., pp. 75~80.

(7) Kuo, S. H., Chiu, P. H., Lin, R. K. and Lin, Y. T., 2010, "GPU Implementation for Solving Incompressible Two-phase Flows," *International Journal of Information and Mathematical Sciences*, Vol. 5, pp. 241~249.

(8) Li, R. and Saad, Y., 2013, "GPU-Accelerated Preconditioned Iterative Linear Solvers," *The Journal of Supercomputing*, Vol. 63, pp. 443~466.

(9) Burden, R. L. and Faires, J. D., 2011, *Numerical Analysis 9th Edition*, Cengage Learning, Boston, pp. 450~459.

(10) Cheng, J., Grossman, M. and McKercher, T., 2014, *Professional CUDA C Programming*, John Wiley&Sons, Inc., New York, p. 31.