<학술논문> DOI http://dx.doi.org/10.3795/KSME-B.2016.40.9.597 ISSN 1226-4881(Print)
2288-5324(Online)

# 유한요소 비압축성 유동장 해석을 위한 이중공액구배법의 GPU 기반 연산에 대한 연구

윤종선* · 전병진** · 정혜동*** · 최형권****†
* 서울과학기술대학교 기계공학과, ** 연세대학교 의과대학 심혈관영상연구센터,
*** 전자부품연구원 임베디드&소프트웨어 연구센터, **** 서울과학기술대학교 기계·자동차공학과

# A Study on GPU Computing of Bi-conjugate Gradient Method for Finite Element Analysis of the Incompressible Navier-Stokes Equations

Jong Seon Yoon*, Byoung Jin Jeon**, Hye Dong Jung*** 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.,
*** Embedded & Software Research Center, Korea Electronics Technology Institute,
**** Dept. of Mechanical and Automotive Engineering, Seoul Nat'l Univ. of Science and Technology.

(Received April 13, 2016 ; Revised June 7, 2016 ; Accepted June 8, 2016)

**Key Words**: GPU, Bi-conjugate Gradient Method(이중공액구배법), CFD(전산유체역학), Finite Element Method (유한요소법)

**초록**: 본 연구에서는 GPU를 이용한 비압축성 유동장의 병렬연산을 위하여, P2P1 유한요소를 이용한 분리 알고리즘 내의 행렬 해법인 이중공액구배법(Bi-Conjugate Gradient)의 CUDA 기반 알고리즘을 개발하였다. 개발된 알고리즘을 이용해 비대칭 협착관 유동을 해석하고, 단일 CPU와의 계산시간을 비교하여 GPU 병렬 연산의 성능 향상을 측정하였다. 또한, 비대칭 협착관 유동 문제와 다른 행렬 패턴을 가지는 유체구조 상호작용 문제에 대하여 이중공액구배법 내의 희소 행렬과 벡터의 곱에 대한 GPU의 병렬성능을 확인하였다. 개발된 코드는 희소 행렬의 1개의 행과 벡터의 내적을 병렬 연산하는 커널(Kernel)로 구성되며, 최적화는 병렬 감소 연산(Parallel Reduction), 메모리 코얼레싱(Coalescing) 효과를 이용하여 구현하였다. 또한, 커널 생성 시 워프(Warp)의 크기에 따른 성능 차이를 확인하였다. 표준예제들에 대한 GPU 병렬연산 속도는 CPU 대비 약 7배 이상 향상됨을 확인하였다.

**Abstract**: A parallel algorithm of bi-conjugate gradient method was developed based on CUDA for parallel computation of the incompressible Navier-Stokes equations. The governing equations were discretized using splitting P2P1 finite element method. Asymmetric stenotic flow problem was solved to validate the proposed algorithm, and then the parallel performance of the GPU was examined by measuring the elapsed times. Further, the GPU performance for sparse matrix-vector multiplication was also investigated with a matrix of fluid-structure interaction problem. A kernel was generated to simultaneously compute the inner product of each row of sparse matrix and a vector. In addition, the kernel was optimized to improve the performance by using both parallel reduction and memory coalescing. In the kernel construction, the effect of warp on the parallel performance of the present CUDA was also examined. The present GPU computation was more than 7 times faster than the single CPU by double precision.

## 1. 서 론

최근에는 설계, 의학, 금융 등 다양한 분야에서 수치해석을 이용한 연구가 진행되고 있다. 수치해석에서 정확한 결과를 얻기 위해서는 상당한 시간이 소요된다. 따라서 빠른 계산 결과를 얻으려는 소비자들의 필요에 따라 컴퓨터 성능과 소프트웨어 발전이 동시에 이루어지고 있다.

† Corresponding Author, hgchoi@seoultech.ac.kr

수치해석을 위한 시스템 환경은 단일 프로세서에서 시작하여 병렬 컴퓨팅으로 발전했다. 병렬 컴퓨팅은 크게 분산 메모리와 공유 메모리 구조로 나눠진다. 분산 메모리 구조는 2개 이상의 노드로 구성된 클러스터와 같으며, 공유 메모리 구조는 Intel Xeon-Phi, GPU 등과 같이 수십 개의 코어가 메모리를 공유하는 시스템을 말한다.

유체 또는 고체 방정식을 풀기 위해서는 유한요소법 등의 수치해석 기법을 이용하여 이산화한 행렬에 대해 이중공액구배법(BiCG, Bi-Conjugate Gradient) 등을 이용하는데, 이와 같은 과정은 시스템 환경에 따라 연산 성능이 크게 차이난다.

기존 CPU를 이용한 연산의 경우, 희소 행렬과 벡터 곱(SpMV, Sparse Matrix Vector Multiplication)을 이용할 때의 메모리 접근은 불규칙하거나 간접적이기 때문에 높은 캐시 미스(cache miss) 발생 확률과 제한된 메모리 대역폭을 보인다. Temam 등[1]은 희소 행렬과 벡터의 곱 연산의 메모리 효율에 대해서 분석하고, 지역성(locality)을 활용하여 캐시 메모리의 이용 빈도를 증가시켰다. Vuduc 등[2]은 CPU 환경에서 캐시 메모리의 최대 효율을 이끌어내는 캐시 블록킹(Cache Blocking) 방법을 이용해 40개 이상의 행렬에 적용하여 분석하였으며, 특정 행렬에 대해서 최대 2배의 속도 향상을 이뤄냈다.

CPU를 이용한 병렬 연산은 OpenMP 또는 영역분할을 이용한 MPI를 적용하여 수행되었다. 전병진 등[3]은 CPU 기반의 클러스터 구조에서 MPI와 하이브리드(OpenMP+MPI) 병렬 방식을 이용해 이중공액구배법의 병렬 성능을 측정하였다.

최근에는 CPU 성능 개발의 정체와 에너지 효율과 관련하여 GPU에 대한 관심이 높아지고 있다. 특히 NVIDIA사에서는 CUDA(Computed Unified Device Architecture) 라이브러리를 제공하여 사용자가 GPU를 쉽게 이용할 수 있도록 했다. 이외에도 Chronos Group은 GPU 프로그래밍을 위한 OpenCL 등[4]을 제공한다.

GPU는 기존 CPU 보다 월등히 많은 코어 수와 높은 메모리 접근 효율 등의 장점을 가지고 있으며, CPU 기반의 프로세서들과 다른 구조로 구성되어 있다. 따라서, 몇몇 연구자들은 선형화 된 희소행렬을 계산하기 위하여 효율적으로 데이터를 저장하고 이용하는 방법을 연구하였다. Bell 등[5]은 COO, CSR, ELL-PACK 등의 저장형식을 비교하고 병렬에 적합한 커널 코드를 제시하였다. Feng 등[6]은 메모리 접근을 효율적으로 하기위해서 padding 방법을 이용한 SIC 저장 형식을 개발하여 최적화를 진행했다. Ashari 등[7]은 NVIDIA kepler 구조의 동적 병렬화(dynamic parallelism)와 A-CSR(Adaptive CSR) 구조를 이용해 희소행렬과 벡터 곱의 연산 효율을 향상시켰다.

이와 같은 여러 연구들을 바탕으로 CUDA 프로그래밍은 과학·공학에 적용되었고 다양한 분야에서 성능을 개선시키려는 노력이 이루어지고 있다. 박태정 등[8]은 GPU 기반의 BioFET 시뮬레이션 연구를 수행하여 최대 33배의 속도향상을 보였다. 장태규 등[9]은 GPU 환경의 다차원 공간 제한기법을 통해 압축성 유동해석을 진행하였으며 1.5~ 2.5배의 속도향상을 얻었다.

본 연구에서는 선행연구들에서 제안한 알고리즘을 이용하여 비대칭 협착관(stenosis) 유동 문제를 빠르게 해석하고자 한다. 따라서, C언어로 작성한 P2P1 유한요소를 이용한 분리 알고리즘 내의 이중공액구배법을 CUDA 기반의 알고리즘으로 수정하고, 다양한 격자계를 통해서 단일 CPU 대비 GPU의 병렬 연산 성능을 측정하고자 한다. 추가적으로, 본 행렬과 다른 패턴을 가지는 유체구조 상호작용 문제(FSI, Fluid Structure Interaction)를 계산하였다. 유체구조 상호작용 문제는 비대칭 협착관 유동 문제보다 복잡한 예조건화를 필요로 하므로 이중공액구배법 내의 상당한 시간을 소모하는 희소 행렬과 벡터 곱에 대해서만 병렬화를 진행하였다.

## 2. 3단계 분리 유한요소 알고리즘

비압축성 Navier-Stokes 방정식을 풀기 위해 3단계 분리 유한요소 알고리즘을 사용하였다. 이전 시간에서의 속도와 압력이 주어질 때, 분리 계산법의 1단계에서는 아래의 식을 풀게 된다.

$$\frac{\hat{u}_i - u_i^n}{\triangle t} + \frac{1}{2}\left(\hat{u}_j \hat{u}_{i,j} + u_j^n u_{i,j}^n\right) = -\frac{1}{\rho} p_{,i}^n + \frac{1}{2}\left(\hat{\sigma}_{ij} + \sigma_{ij}^n\right)_{,j} \quad (1)$$

여기서, 위첨자 n은 이전 시간을 의미하며 이 식으로부터 $\hat{u}_i$을 구한다. 아래의 2단계 식에서는 1

단계에서 구한 $\hat{u}_i$와 이전 시간에서의 $p_{,i}^n$를 이용하여 중간 단계의 속도인 $u_i^*$를 구한다.

$$\frac{u_i^* - \hat{u}_i}{\triangle t} = \frac{1}{\rho} p_{,i}^n \tag{2}$$

다음 3단계에서는 연속방정식으로부터 유도된 압력 방정식을 이용하여 $p_{,i}^{n+1}$를 구하고, 갱신된 압력장으로부터 $u_i^{n+1}$를 얻는다. 본 연구에서는 압력 방정식을 풀 때 병렬화에 장점을 갖고 있는 Uzawa 알고리즘을 이용하였다.

$$\begin{cases} u_{i,i} = 0 \\ \dfrac{u_i^{n+1} - u_i^*}{\triangle t} = -\dfrac{1}{\rho} p_{,i}^{n+1} \end{cases} \tag{3}$$

## 3. GPU 병렬화

본 연구에서는 GPU를 이용한 병렬 연산을 위하여 CUDA 언어를 사용하였다. GPU 연산의 성능을 높이기 위해서는 GPU와 CUDA 언어의 구조적 특징을 이해하는 것이 중요하다. 따라서 3장에서는 GPU의 구조적 특징과 본 연구에 사용된 최적화 방법에 대하여 설명하였다.

### 3.1 GPU 프로그래밍 구조

CUDA 프로그램은 C언어를 기반으로 만들어졌으며, 호스트 영역인 CPU와 디바이스 영역인 GPU를 다룰 수 있다. 데이터 병렬성이 큰 부분은 디바이스 영역에서 구현되며 이 영역을 커널(kernel)이라고 칭한다. 호스트에서 커널을 호출할 때, GPU에서는 사용자가 지정한 만큼의 스레드(thread)와 블록(block)을 생성한다. 여기서, 스레드는 하나의 GPU 코어를 뜻하며, 블록은 스레드들의 집합을 의미한다. 블록 내의 스레드들은 동시에 같은 명령어(instruction)를 수행한다.

### 3.2 GPU 메모리 구조

Fig. 1은 GPU의 메모리 구조를 보여준다. GPU는 CPU와 메모리는 분리되어 있다. 따라서 데이터를 공유할 수 없으며 CPU로부터 필요한 메모리를 가져와야한다. 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

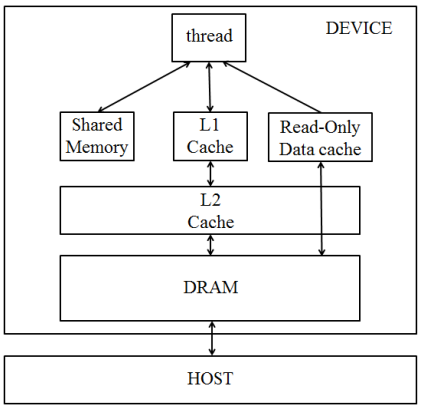



**Fig. 1** GPU memory architecture

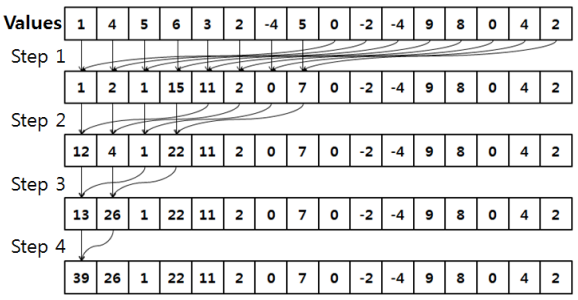



**Fig. 2** Parallel reduction operation[11]

소비하는 매우 느린 메모리다. 공유 메모리(Shared memory)는 GPU 1 사이클 이내를 소비하며, CPU의 L2 캐시와 같은 속도이고 사용자가 직접 제어할 수 있는 장점이 있다. Kepler 기반의 GPU[10]에서 새롭게 도입된 읽기 전용 캐시는 일반적인 불러오기의 경우에 이 캐시에 직접 접근할 수 있고 데이터 정렬 유무와 관계없이 효율적으로 사용할 수 있다.

### 3.3 병렬 축소(Parallel Reduction)

희소행렬과 벡터의 곱은 내적 연산이 발생한다. 그러나, 벡터 원소들의 덧셈 과정에서는 순차적인 연산 특성이 내포되어 있어 병렬 처리에 어려운 부분이 있다. 이런 문제의 해결책으로는 병렬 축소(Parallel Reduction) 방법이 있다. 병렬 축소 방법은 덧셈 과정에서 O(log2N)의 연산을 보장한다.

Fig. 2는 병렬 축소 방법[11]을 보여준다. 각 스레드는 독립적으로 계산을 수행하고 그 결과 값을 병합하는 여러 단계를 거쳐서 최종 값을 얻는다. CPU 환경에서는 4~12개의 코어가 작동하므로 병렬 축소가 큰 영향을 미치지 않지만, 다수

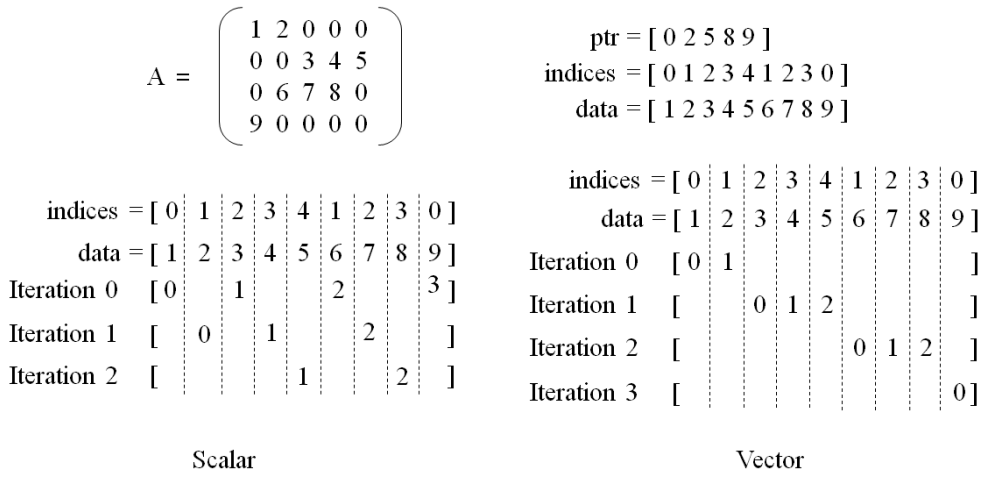



**Fig. 3** The memory access pattern of Scalar and Vector SpMV kernel[5]

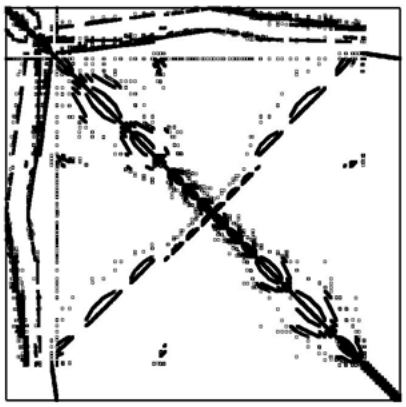


**Fig. 4** Matrix pattern of the eccentric stenosis problem

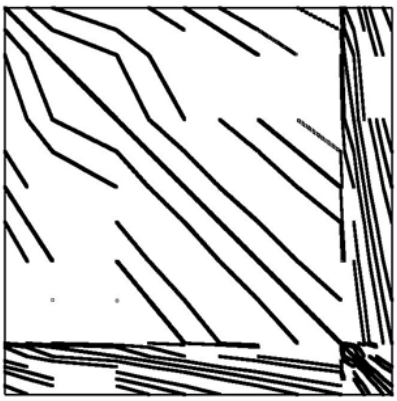


**Fig. 5** Matrix pattern of the Fluid Structured Interation problem

의 코어를 사용하는 GPU의 경우에는 효율적으로 병렬 축소를 구현해야 한다.

### 3.4 Coalescing 효과

Coalescing 효과란 GPU 내부의 스레드들이 같은 지역의 메모리에 접근할 경우에 적은 비용으로 데이터를 읽는 것을 의미한다.

Bell 등[5]은 희소행렬과 벡터 곱을 연산하기 위한 스칼라 또는 벡터 형태의 커널을 제안하였다. Fig. 3은 스칼라와 벡터 커널의 스레드 접근 방식에 대한 차이를 보여준다. Fig. 3의 Iteration에 나타낸 괄호 안의 번호는 스레드들의 인덱스를 나타낸다. 스칼라 커널의 경우에는 1개의 스레드가 1개의 행을 담당하므로, 반복될 때마다 새로운 행의 첫 번째 원소에 접근한다. 이런 경우에는 Iteration 0에서 스레드 0, 1, 2, 3번이 data 배열의 1, 3, 6, 9에 접근한다. 이와 같이, 각 스레드들은 연속적인 메모리 접근이 어렵게 된다. 그러나 벡터 커널의 경우에는 다수의 스레드들이 1개의 행을 담당하게 된다. 따라서, 각 Iteration에서 스레드들은 data 배열에 차례대로 접근할 수 있으므로 Coalescing 효과를 얻을 수 있다.

## 4. 해석 결과 및 토의

본 연구에서는 비대칭 협착관 유동 문제를 풀기 위하여, C언어로 작성된 P2P1 유한요소를 이용하는 분리 알고리즘 내의 이중공액구배법을 CUDA 기반 알고리즘으로 개발하였다. 또한 다른 행렬 패턴을 가지는 문제에 대하여 검토하기 위하여 유체구조 상호작용 문제에 대하여 해석을 수행하였다. Table 1은 본 연구에서 이용한 행렬의 정보를 보여주며, NNZ는 희소 행렬 내에 0이 아닌 원소의 개수를 말한다. Fig. 4와 Fig. 5는 비대칭 협착관 또는 유체구조 상호작용 문제의 희

**Table 1** Matrices in experiment

(a) Stenosis matrix

| | Row | NNZ | NNZ/Row |
|---|---|---|---|
| M1 | 496,555 | 13,271,952 | 26.73 |
| M2 | 654,497 | 17,679,924 | 27.01 |
| M3 | 1,088,031 | 29,702,596 | 27.30 |
| M4 | 1,312,695 | 35,916,784 | 27.36 |

(b) FSI matrix

| | Row | NNZ | NNZ/Row |
|---|---|---|---|
| M5 | 714,555 | 60,329,688 | 84.43 |
| M6 | 1,230,723 | 104,426,504 | 84.85 |
| M7 | 1,766,187 | 150,173,928 | 85.03 |
| M8 | 3,126,555 | 266,532,344 | 85.25 |

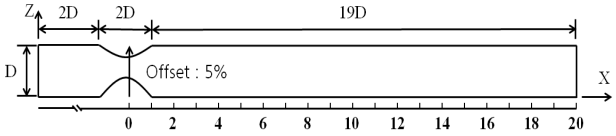



**Fig. 6** Schematic of the eccentric stenosis problem[12]

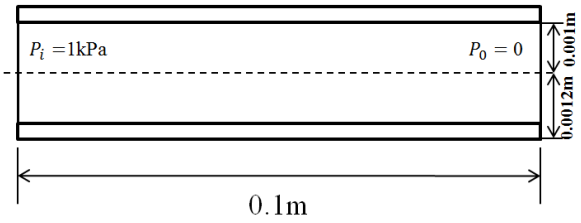
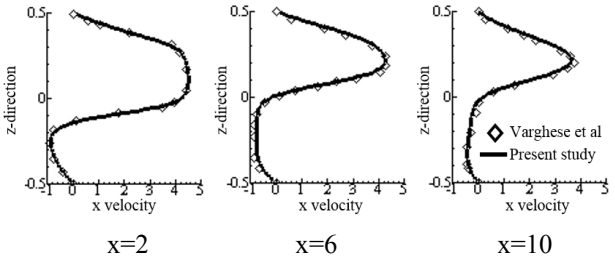


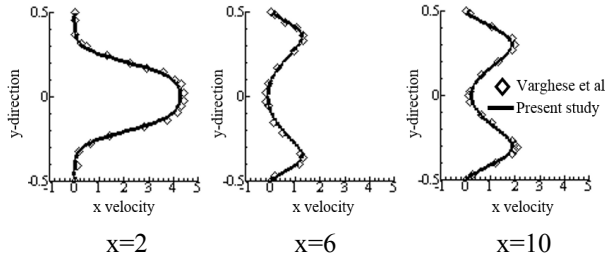

**Fig. 7** Schematic of the Fluid Structured Interaction problem[13]

소 행렬 패턴을 보여준다. 유체구조 상호작용 문제의 경우에는 이중공액구배법 내의 희소행렬과 벡터 곱에 대해서만 병렬화하였다. 본 연구에서 사용한 CPU와 GPU는 E5-2637 @3.50GHz와 Nvidia Tesla_K20X이며, 컴파일러는 Intel C Compiler 15.0과 CUDA toolkit 6.5를 사용하였다.

Fig. 6와 Fig. 7는 각각 비대칭 협착관 유동 문제[12]와 유체구조 상호작용 문제[13]에 대한 개략도를 보여준다. 비대칭 협착관 유동 문제의 경우, 입구의 지름은 D이고, 협착 면적은 입구 기준으로 75% 축소되었다. 또한, 협착 영역의 중심은 z 방향으로 입구의 지름 대비 5%가 편심되었다. 유체구조 상호작용 문제의 형상은 관의 지름이 0.001m이며 두께는 0.0002m이다. 입구와 출구에




(a) Asymmetric plane (x-z plane)




(b) Symmetric plane (x-y plane)

**Fig. 8** Time-averaged axial velocity($u/u_i$) profiles at the three locations[12]

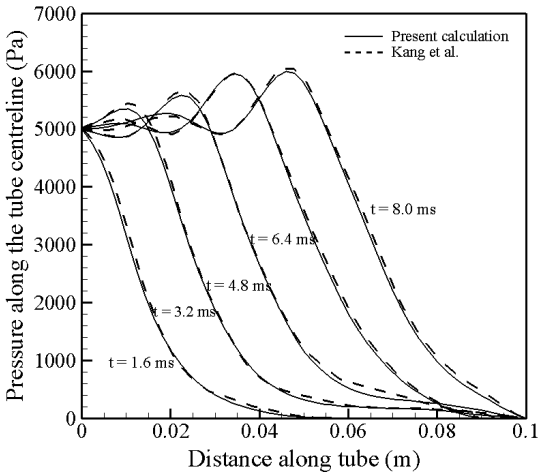
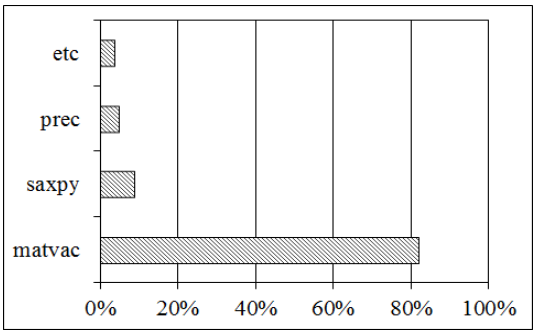



**Fig. 9** Pressure along the centerline with various time[13]

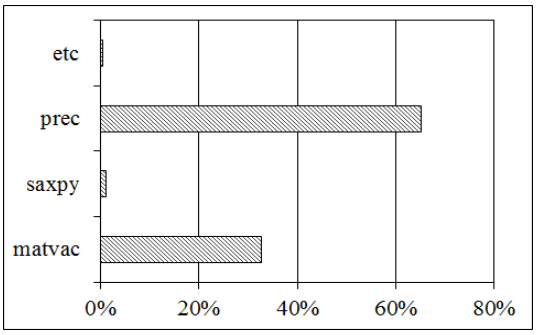
각각 1kPa과 0Pa의 압력 경계조건을 주었다.

Fig. 8는 입구 지름 기준의 레이놀즈 수가 500인 비대칭 협착관 유동에 대해서 대칭 또는 비대칭 평면에서의 주유동 방향 속도 분포를 Varghese 등[12]의 선행연구와 비교한 결과이다. 각 위치에서의 속도 분포 결과는 선행연구의 결과와 일치함을 확인하였다. 또한 Fig. 9은 유체구조 상호작용 문제에 대해 특정 시간에서의 튜브 중심 압력 분포를 Kang 등[13]의 선행연구와 비교한 결과이며 일치하는 것을 확인하였다.

Fig. 10은 이중공액구배법 알고리즘 내의 부분




(a) Case of stenosis

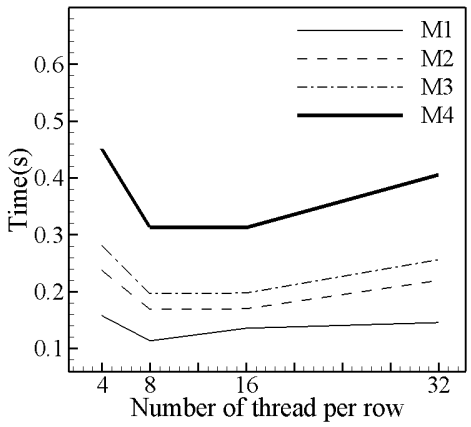



(b) Case of FSI

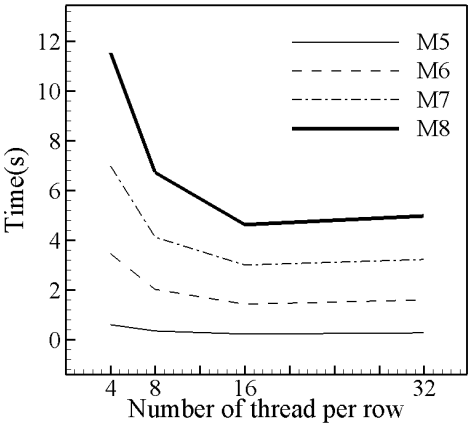
**Fig. 10** Profiling result in Bi-CG

별 연산 시간을 분석하여, 총 연산시간 대비 각 부분의 연산 시간을 비율로 나타낸 것이다. matvac은 희소행렬과 벡터의 곱이고 saxpy는 두 벡터의 합 연산 과정이다. Prec는 각 문제별 예조건화(Preconditioner)를 뜻하며 etc는 이외의 나머지 연산 과정을 뜻한다. 본 연구에서, 비대칭 협착관 유동 문제는 대각 예조건화(Digonal preconditioner) 기법을 사용하였으며 유체구조 상호작용 문제는 LU 예조건화를 사용하였다. 위의 두 가지 문제들은 희소행렬과 벡터 곱에서 전체 수행시간의 80% 또는 30% 이상의 시간을 소요한다. 따라서, 병렬화에 적합한 희소 행렬과 벡터의 곱을 GPU로 연산하게 되면 상당한 계산 시간의 절약을 예상할 수 있다.

비정렬 격자 기반의 희소 행렬에서 각 행은 이웃하는 절점들의 수가 다르므로 일정하지 않은 원소의 개수를 가지고 있다. Bell 등[5]이 제시한 벡터 커널은 32개의 스레드들이 하나의 행을 담당하는 알고리즘이다. 그러나, 행의 원소 개수가 32개 이하이면 비활성 스레드가 발생하게 되어 비효율적이다. 그러므로, 문제의 크기에 따라서 적절한 워프(Warp) 크기의 조정 작업이 필요하다. Fig. 11는 비대칭 협착관 유동 문제와 유체구조



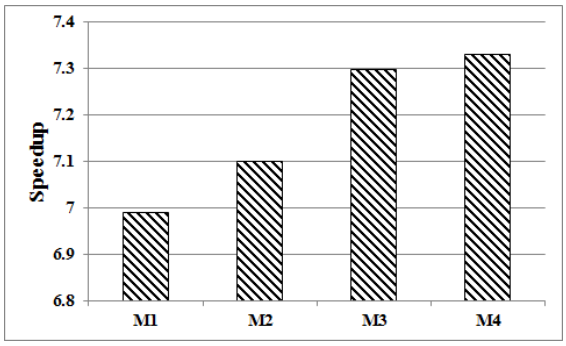

(a) Case of stenosis



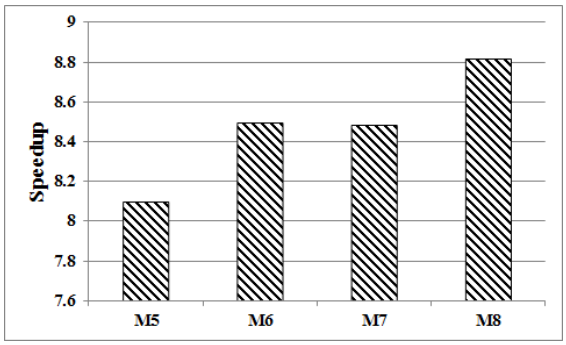

(b) Case of FSI

**Fig. 11** Execution time with various warp size in Bi-CG

상호작용 문제의 격자계에 대해서 하나의 행을 담당하는 스레드의 개수에 따른 GPU 연산 속도를 보여준다. 비대칭 협착관 유동의 경우에는 대체적으로 8개가 하나의 행을 맡았을 때 가장 빠르고 4개를 이용하였을 때 가장 느린 것을 확인하였다. 반면 유체구조 상호작용 문제의 경우에는 16개일 때 가장 빠르고, 4개를 이용하였을 때 가장 느린 결과를 보였다. 이와 같은 차이는 Table 1에 보여지는 것과 같이, 각 문제의 행이 가지고 있는 평균 원소의 개수 차이에 의해서 발생한다. 1개의 스레드가 적절한 양의 일을 담당하게 되면, 메모리 접근에 대한 지연 시간 숨김(Latency hiding)을 잘 이용하게 되어 효율적인 연산을 할 수 있기 때문이다. 본 연구의 문제들에서는 평균적인 행의 원소 개수가 작업하는 스레드 개수의 2~4배가 되었을 때 효율적인 연산 성능을 보였다. 격자의 특성에 따라 가장 빠른 연산 속도를 보이는 스레드 개수를 선택할 경우에




(a) Case of stenosis




(b) Case of FSI

**Fig. 12** Performance improvement for the various grid size

는 Bell 등[5]이 제시한 32개의 스레드 개수의 속도보다, 작은 격자는 약 10%, 큰 격자는 약 20%의 성능 차이를 보인다.

Fig. 12는 격자수에 따른 CPU의 연산 속도 대비 GPU의 병렬 연산 속도 향상을 보여준다. 비대칭 협착관 유동 문제와 유체구조 상호작용 문제의 행을 담당하는 스레드의 개수는 각각의 결과에서 가장 빠른 연산 속도를 보여준 8개와 16로 하였다. 격자의 행의 수 또는 행렬의 요소의 수가 많아질수록 GPU의 연산 속도가 전반적으로 증가하는 것을 볼 수 있으며 작은 격자에서는 약 7배, 큰 격자에서는 약 8배로 속도가 증가하였다.

## 5. 결 론

본 연구에서는 P2P1 유한요소를 이용한 전산유체역학 문제의 해석에서 GPU 연산의 성능을 평가하기 위하여, 반복행렬해법인 이중공액구배법의 CUDA 기반 알고리즘을 개발하였다. 개발된 코드를 이용한 수치실험을 통하여 아래와 같은 결론을 도출하였다.

(1) 개발된 CUDA 기반의 병렬 알고리즘의 검증을 위하여 비대칭 협착관 유동 문제와 유체 구조 상호작용 문제인 유연한 튜브 내의 유동을 CUDA 기반의 연산을 수행하여 선행 연구 결과들과 잘 일치함을 확인하였다.

(2) GPU와 CPU의 다른 메모리 접근 특성에 근거하여, 본 연구에서는 병렬 감소 연산(Parallel Reduction)과 메모리 코얼레싱(Memory Coalescing) 등을 이용하여 GPU 병렬 알고리즘의 최적화를 수행하였다. 비대칭 협착관 유동 문제의 이중공액구배법 알고리즘에서는 약 7배, 유체구조 상호작용 문제의 희소행렬과 벡터의 곱 연산에서는 약 8배의 성능 향상을 얻었다.

(3) 비정렬 격자계를 이용한 연산의 특성에 의해서 희소행렬과 벡터의 곱에 대한 연산은 하나의 행을 담당하는 스레드 개수에 따라 병렬 성능 차이가 발생하는 것을 확인하였다.

(4) GPU는 보드와 메모리가 분리되어 있기 때문에 메모리 규격 변경이 어려운 CPU와 다르게 내부의 메모리 규격과 버스 규격을 자유롭게 변경할 수 있다. 따라서, 추후에 동일한 문제에 대해서 향상된 성능을 가지는 Tesla K40, Tesla K80 등의 GPU를 이용하여 성능 분석을 수행하고자 한다.

## 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) Temem, O. and Jalby, W., 1992, "Charact- erizing the Behavior of Sparse Algorithms on Caches," *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pp. 578~587.

(2) Richard, V., Demmel, J. W., Yelick, K. A., Kamil, S., Nishtala, R. and Lee, B. J., 2002, "Performance Optimizations and Bounds for Sparse Matrix-Vector

Multiply," *Proceedings of the IEEE/ACM SC2002 Conference*.

(3) Jeon, B. J. and Choi, H. G., 2014, "Comparison of Message Passing Interface and Hybrid Programming Models to Solve Pressure Equation in Distributed Memory System," *Trans. Korean Soc. Mech. Eng. B*, Vol. 39, No. 2, pp. 191~197.

(4) https://www.khronos.org/opencl/

(5) Bell, N. and Garlandy, M., 2008, “Ecient Sparse Matrix-Vector Multiplication on CUDA,” NVIDIA Technical Report NVR-2008-004.

(6) Feng, X., Hai Jin, Zheng, R., Hu, K., Zeng, J. and Shao, Z., 2011, “Optimization of Sparse Matrix-Vector Multiplication with Variant CSR on GPUs,” 22 2011 IEEE 17th International Conference on Parallel and Distributed Systems.

(7) Ashari, A., Sedaghati, N., Eisenlohr, J., Paqrthasarathy, S. and Sadayappan, P., 2014, "Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 781~792.

(8) Park, T. J., Woo, J. M. and Kim, C. H., 2011, "CUDA-based Parallel Bi-Conjugate Gradient Matrix Solver for BioFET Simulation," *Journal of the Institute of Electronics Engineers of Korea*, Vol. 48, No. 1, pp. 90~100.

(9) Chang, T. K., Park, J. S. and Kim, C., 2014, "Efficient Computation of Compressible Flow By Higher-order Method Accelerated Using GPU," *J. Comput. Fluids Eng*, Vol. 19, No. 3, pp. 52~61.

(10) https://www.nvidia.com/content/PDF/kepler/ NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

(11) http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

(12) Varghese, S. S., Frankel, S. H. and Fischer, P. F., 2007, "Direct Numerical Simulation of Stenotic Flow: Part 1. Steady Flow," *J. Fluid Mech*., Vol. 582, pp. 253~280.

(13) Kang, K., Choi, H. G. and Yoo, J. Y., 2012. “Investigation of Fluid-structure Interactions using a Velocity-linked P2/P1 Finite Element Method and the Generalized-method,” *Int. J. Numer. Meth. Engng*. pp. 1539~1547.