고차 정확도 수치기법의 GPU 계산을 통한 효율적인 압축성 유동 해석

장 태 규,[1] 박 진 석,[2] 김 종 암*[1,2]
[1]서울대학교 기계항공공학부
[2]서울대학교 항공우주신기술연구소

EFFICIENT COMPUTATION OF COMPRESSIBLE FLOW BY HIGHER-ORDER METHOD ACCELERATED USING GPU

T.K. Chang,[1] J.S. Park[2] and C. Kim*[1,2]
[1]Department of Aerospace Engineering, Seoul National University
[2]Institute of Advanced Aerospace Technology, Seoul National University

The present paper deals with the efficient computation of higher-order CFD methods for compressible flow using graphics processing units (GPU). The higher-order CFD methods, such as discontinuous Galerkin (DG) methods and correction procedure via reconstruction (CPR) methods, can realize arbitrary higher-order accuracy with compact stencil on unstructured mesh. However, they require much more computational costs compared to the widely used finite volume methods (FVM). Graphics processing unit, consisting of hundreds or thousands small cores, is apt to massive parallel computations of compressible flow based on the higher-order CFD methods and can reduce computational time greatly. Higher-order multi-dimensional limiting process (MLP) is applied for the robust control of numerical oscillations around shock discontinuity and implemented efficiently on GPU. The program is written and optimized in CUDA library offered from NVIDIA. The whole algorithms are implemented to guarantee accurate and efficient computations for parallel programming on shared-memory model of GPU. The extensive numerical experiments validates that the GPU successfully accelerates computing compressible flow using higher-order method.

Key Words : 전산유체역학(CFD), 압축성유동(Compressible Flow), 고차 정확도 수치기법(Higher-order Method), 불연속 갤러킨 방법(Discontinuous Galerkin Method), CPR 방법(Correction Procedure via Reconstruction Method), GPU 계산(GPU computing), CUDA 프로그래밍(CUDA Programming)

1. 서 론

최근 충격파와 와류의 상호작용과 같은 복잡한 유동을 비정렬 격자계에서 장시간 정밀하게 해석할 수 있는 고차 정확도 CFD 방법이 많은 관심을 받고 있다. 대표적으로는 불연속 갤러킨(Discontinuous Galerkin) 방법[1]과 correction procedure via reconstruction(CPR) 방법[2,3]이 있다. 이들은 유한체적법(FVM)의 보존성과 유한요소법(FEM)의 내삽 개념을 결합하여, 복잡한 형상에 유연하게 적용 가능하면서도 좁은 스텐실을 통해 임의의 높은 정확도를 얻을 수 있다.

하지만 충격파가 존재하는 유동을 해석하기 위해서는 수치 안정성을 위해 추가적인 공간 제한기법이 필요하고, 이를 위해 기존의 유한체적법에서 사용되던 제한기법들을 적용하려는 시도들이 있었다. 최근에는 다차원 유동의 특성이 반영된 다차원 공간 제한기법, MLP(Multi-dimensional Limiting Process)를 DG와 CPR 방법에 적용하여 이런 유동을 강건하고 정밀하게 해석하는 연구가 진행되어 성과를 거두었다[4-8].

이같이 고차 정확도 CFD 방법은 유동 해석에 있어 다양한 장점을 가지고 있지만, 정확도를 높이기 위해 격자 내부의 자유도(DOF)가 증가하고 수치 안정성을 보장할 수 있는 시간 전진이 제한되어, 결과적으로 계산량이 크게 늘어난다. 따라서 이를 해결하기 위한 효율적인 계산 방법에 대한 연구가 필요하다. 그리고 대용량의 계산량을 처리하기 위한 계산 가

Received: June 25, 2014, Revised: September 11, 2014, Accepted: September 11, 2014.
* Corresponding author, E-mail: chongam@snu.ac.kr
DOI http://dx.doi.org/10.6112/kscfe.2014.19.3.052

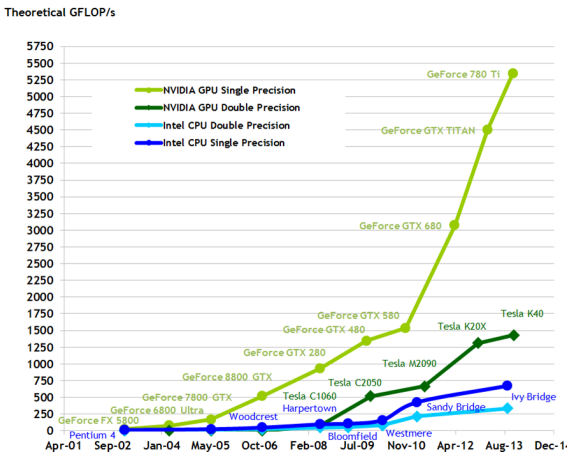


Fig. 1 Comparison of theoretical FLOPS between CPU and GPU[9]

속 방법 중에서 기존의 CPU 대신 GPU(Graphics Processing Units)를 이용하는 계산이 각광 받고 있다. GPU는 컴퓨터 화면의 많은 픽셀을 표시하기 위한 장치로 내부가 수많은 작은 코어(core)들로 이루어져 있어서 대규모 병렬 계산에 적합하다고 알려져 있다. 실제 GPU의 이론적인 계산 성능은 Fig. 1에서 볼 수 있듯이 CPU에 비해 훨씬 크다[9]. 이에 따라 NVIDIA사의 CUDA(Compute Unified Device Architecture)와 크로노스 그룹의 OpenCL(Open Computing Language)이 개발되어 이를 이용한 GPU 계산이 과학적 계산에서 점점 많이 활용되고 있고, 전산유체역학 분야에서도 이를 응용하기 위한 연구가 진행되어 왔다[10-12].

이러한 점들에 주목하여 본 연구에서도 충격파가 있는 압축성 유동을 해석하기 위한 MLP가 적용된 고차 정확도 수치기법을 CUDA를 이용하여 효율적으로 계산하려고 한다. 또한 실제 GPU 계산의 계산 효율성을 확인해보고, 어떤 요소들이 영향을 미치는지 알아보고자 한다.

2. 지배방정식 및 수치기법

2.1 Euler 방정식

본 연구의 대상인 비점성 압축성 유동장의 지배방정식인 Euler 방정식은 다음과 같다.

$$\frac{\partial \mathbf{Q}}{\partial t}+\nabla\cdot\mathbf{F}=\mathbf{0}, \tag{1}$$

여기서 Q는 보존벡터(conservative vector)를 나타내고, F는 비점성 플럭스 텐서(inviscid flux tensor)이며 3차원인 경우에는 다음과 같다.

$$\mathbf{Q}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho E\end{bmatrix},\ \mathbf{F}=\begin{bmatrix}\rho u & \rho v & \rho w\\ \rho u^2+p & \rho uv & \rho uw\\ \rho vu & \rho v^2+p & \rho vw\\ \rho wu & \rho wv & \rho w^2+p\\ \rho uH & \rho vH & \rho wH\end{bmatrix}. \tag{2}$$

ρ, p, E 와 (u, v, w)는 각각 밀도, 압력, 전에너지(total energy)와 속도 벡터이다. 이상 기체 상태방정식을 이용하여 압력과 전엔탈피(total enthalpy) H는 다음과 같이 구해진다.

$$p=(\gamma-1)\rho\left(E-\frac{1}{2}(u^2+v^2+w^2)\right). \tag{3}$$

$$H=E+\frac{p}{\rho}. \tag{4}$$

비열비 γ는 1.4이다. 이후 부분에서는 Euler 방정식에 대하여 수치기법을 적용하여 전개하도록 하겠다.

2.2 불연속 갤러킨(DG) 방법

불연속 갤러킨 방법은 고차 정확도를 얻기 위해 격자 내부의 분포를 형상 함수의 선형 결합으로 나타낸다. 이 형상 함수는 n차 다항함수(P_n) 공간 V^n에서 정의된다.

$$\mathbf{Q}_i^h(\mathbf{x},t)=\sum_{l=1}^{n}\mathbf{Q}_i^{(l)}(t)b_i^{(l)}(\mathbf{x}). \tag{5}$$

$\mathbf{Q}_i^h$는 격자(cell) T_i에서 근사(approximate)된 보존벡터이고, $b_i^{(l)}$는 형상 함수이다. 형상 함수는 다항함수 공간 V^n의 기저함수들은 모두 가능하나, 본 연구에서는 계산 효율성을 위해 직교 형상 함수(orthogonal shape function)의 일종인 Jacobi 다항식의 텐서곱[13]을 사용하였다.

다차원 비정렬 격자계에 대해서 Euler 방정식 (1)을 DG 방법에 따라 표현하면 다음과 같다.

$$\int_{T_i}\frac{\partial \mathbf{Q}}{\partial t}\mathbf{W}dV+\int_{\partial T_i}\mathbf{F}\cdot\mathbf{n}\mathbf{W}dS -\int_{T_i}\mathbf{F}(\mathbf{Q})\cdot\nabla\mathbf{W}dV=\mathbf{0}, \tag{6}$$

여기서 n은 법선 방향 벡터이고, W는 시험함수 벡터로 다항함수 공간 V^n에서 표시된다. 격자 경계면(face)에서의 불연속을 표현하기 위하여 유한체적법에서 사용하는 단조성의 수치

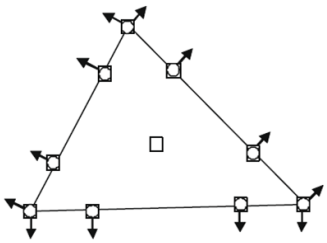


Fig. 2 Solution points(squares) and flux points(circles) for *P*3 reconstruction of CPR method

플럭스 함수를 적용하면 다음의 근사화된 지배방정식이 얻어진다.

$$\int_{T_i} \frac{\partial \mathbf{Q}_i^h}{\partial t} \mathbf{B}_i dV + \int_{\partial T_i} \mathbf{H}(\mathbf{Q}_{i-}^h, \mathbf{Q}_{i+}^h, \mathbf{n}) \cdot \mathbf{n} \mathbf{B}_i dS - \int_{T_i} \mathbf{F}(\mathbf{Q}_i^h) \cdot \nabla \mathbf{B}_i dV = 0. \tag{7}$$

여기서 Q_{i-}^h는 안쪽 격자 T_i에서부터 근사된 격자 경계면 ∂T_i에서의 보존벡터를 의미하며, Q_{i+}^h는 격자 T_i와 인접한 격자에서부터 근사된 격자 경계면에서의 보존벡터를 의미한다. 수치 플럭스 함수로는 정밀하고 강건한 충격파의 포착이 가능하도록 Roe's FDS(Flux Difference Splitting) 기법을 개선한 RoeM 기법을 사용하였다[14]. 경계 및 영역에서의 적분은 Gauss 구분구적법을 적용하여 각각 *2n*, *2n*+1의 필요한 정확도를 가지도록 하였다.

2.3 Correction procedure via reconstruction(CPR) 방법

CPR 방법은 식 (6)에서 수치 플럭스를 대입하고 마지막 항을 부분 적분하여 얻어지는 다음의 strong form에서 출발한다.

$$\int_{T_i} \frac{\partial \mathbf{Q}}{\partial t} \mathbf{W} dV + \int_{\partial T_i} (\mathbf{H} - \mathbf{F}) \cdot \mathbf{n} \mathbf{W} dS - \int_{T_i} \nabla \cdot \mathbf{F}(\mathbf{Q}) \mathbf{W} dV = 0, \tag{8}$$

또한 격자 내부는 Lagrange 다항식을 이용하여, 다음과 같이 정의된다.

$$\mathbf{Q}_i^h(\mathbf{x},t) = \sum_{j=1}^{n} \mathbf{Q}_{i,j}^h(t) L_{i,j}^h(\mathbf{x}). \tag{9}$$

$\mathbf{Q}_{i,j}^h$는 격자 T_i의 solution point $\mathbf{x}_j$에서의 보존벡터이다. CPR 방법은 계산의 효율성을 위해서 Fig. 2처럼 Lagrange 다항식을 구성하는 solution point들을 수치 플럭스를 계산하는 격자 경계면의 flux point와 일치시킨다. 식 (8)을 해가 존재하는 함수공간 V^n으로 투영하고, 적절한 시험함수를 적용하여 정리하면, 각 solution point에 대해 다음 지배방정식을 얻을 수 있다.

$$\frac{\partial \mathbf{Q}_{i,j}^h}{\partial t} + \Pi\left(\nabla \cdot \mathbf{F}\left(\mathbf{Q}_{i,j}^h\right)\right) + \delta_{i,j} = 0, \tag{10}$$

여기서 Π는 투영 연산자(projection operator)이고 δ_i는 lifting operator로 다음과 같이 정의 된다.

$$\int_{T_i} \delta_i \mathbf{W} dV = \int_{\partial T_i} \left(\mathbf{H}(\mathbf{Q}_{i-}^h, \mathbf{Q}_{i+}^h) - \mathbf{F}(\mathbf{Q}_{i-}^h)\right) \cdot \mathbf{n} \mathbf{W} dS. \tag{11}$$

시험함수의 선택에 따라서 CPR 방법은 DG 방법이나 spectral volume(SV), spectral difference(SD) 방법으로 환원된다. 본 연구에서는 Lagrange 다항식을 사용하여 얻어지는 CPR-DG 방법을 이용하였다[4]. 또한 CPR 방법은 보존성에 대한 문제가 있었는데, 최근에 이를 해결한 conservative CPR 방법[15]을 이용하여 연구를 진행하였다.

2.4 시간 적분법

DG와 CPR 방법의 시간 차분에 대해서도 고차 정확도를 얻을 수 있는 시간 적분법을 사용하였다. *P*2 근사 이하의 3차 이하의 공간 정확도의 해석에는 3차 정확도 TVD Runge-Kutta 방법을 적용하여 계산하였고, *P*3 근사 이상의 4차 이상의 공간 정확도의 해석에 대해서는 4차 정확도 SSP Runge-Kutta 방법을 사용했다[16].

2.5 다차원 공간 제한기법(MLP)

충격파와 같은 불연속 영역의 수치진동을 제어하기 위해 다차원 공간 제한기법을 적용하였다. MLP는 다차원 유동의 특성을 반영하여 DG와 CPR 방법에 대해서 충격파를 정밀하게 포착 가능한 공간 제한기법으로 크게 세 부분으로 이루어져 있다[8].

2.5.1. Troubled-cell marker

먼저 공간 제한이 필요한 셀(troubled-cell)을 찾는 과정이다. 불연속이 있는 구간에서 셀 내부 분포를 살펴보면 수치 진동이 linear polynomial로 전파되어 급격한 기울기를 만들고 단조성을 위반하게 된다. 이를 이용하여 불연속 구간인지를 판단

하고 troubled-cell을 찾기 위해 다음과 같이 다차원으로 확장한 조건을 이용하였다. (*P*1-projected MLP condition)

$$\bar{q}_{v_j}^{\min} \le L(\mathbf{x}_{v_j}) = \Pi^1 q_i^{h,Pn}(\mathbf{x}_{v_j}) \le \bar{q}_{v_j}^{\max}, \tag{12}$$

여기서 $L(\mathbf{x}_{v_j}) = \Pi^1 q_i^{h,Pn}(\mathbf{x}_{v_j})$는 꼭지점(vertex) $\mathbf{x}_{v_j}$에서 Pn으로 근사한 벡터 $q_i^{h,Pn}(\mathbf{x}_{v_j})$를 P1 다항함수 공간으로 투영한 값이며, $\bar{q}_{v_j}^{\min/\max}$는 꼭지점 $\mathbf{x}_{v_j}$을 포함하는 모든 셀들의 평균값 $\bar{q}$ 중 최소값과 최대값을 의미한다.

2.5.2. Extrema detector

이와 같이 troubled-cell을 찾게 되면 수치 진동부근 뿐만 아니라 완만한(smooth) 영역의 극점들도 troubled-cell로 판단하게 된다. 따라서 이 영역의 국소 극점(smooth local extrema)을 찾는 단계가 필요하다.

꼭지점에서 smooth extrema가 존재하는 경우, 이 꼭지점으로 근사한 값에 대해서 평균값과 선형 다항식에 의한 값(*Pn*-projected slope), 고차항에 의한 값(*P*1-filtered *Pn*)으로 나눌 수 있다.

$$\begin{aligned} q_i^{h,Pn}(\mathbf{x}_{v_j}) = {} & \bar{q}_i && : (\text{averaged value}) \\ & + \left(L(\mathbf{x}_{v_j}) - \bar{q}_i\right) && : (Pn-\text{projected slope}) \\ & + \left(q_i^{h,Pn}(\mathbf{x}_{v_j}) - L(\mathbf{x}_{v_j})\right) && : (P1-\text{filtered } Pn), \end{aligned} \tag{13}$$

이 때 smooth extrema에서는 선형 다항식 값과 고차항 값이 서로 반대의 부호를 가진다는 것을 확인하였고 이를 이용하여 extrema를 찾는다.

2.5.3. Limiter of troubled-cell

이와 같은 과정을 통해 찾아낸 troubled-cell은 계층적인 제한기법을 적용하였다. 셀 내부가 다음과 같이 표현될 때,

$$q_i^{h,Pn}(\mathbf{x}) = \bar{q}_i + P1(\mathbf{x}) + P2(\mathbf{x}) .. + Pn(\mathbf{x}), \tag{14}$$

고차 다항함수 공간에 존재하는 항부터 차례대로 제거하면서 값을 제한하였다. 여기서,

$$Pn(\mathbf{x}) = \Pi^m q_i^{h,Pn}(\mathbf{x}) - \Pi^{m-1} q_i^{h,Pn}(\mathbf{x}), \tag{15}$$

는 m차 다항함수 공간에 존재하는 항을 나타낸다. 2차 다항함수 공간의 항까지 제거되어 선형 함수로 표현되는 경우에는 각 꼭지점의 값을 인접 셀의 평균값으로 제한하여 단조성을 만족할 수 있도록 하였다.

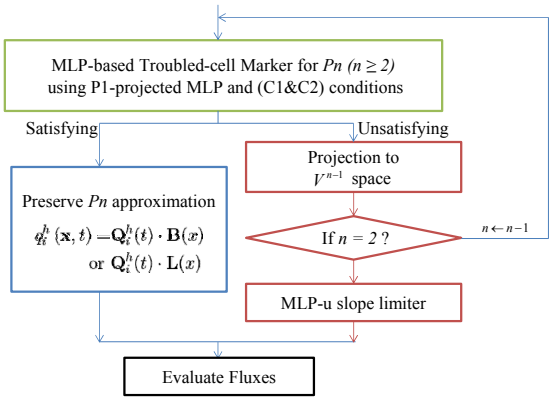


Fig. 3 Flowchart of MLP limiting strategy

MLP의 전 과정을 순서도로 나타내면 Fig. 3와 같다.

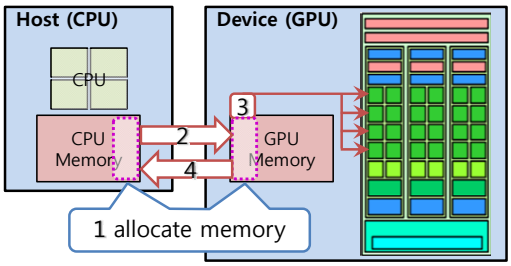
3. CUDA 프로그래밍을 이용한 GPU 계산

3.1 CUDA 프로그램 구조

CUDA 라이브러리는 GPU의 많은 코어를 이용하여 병렬계산을 수행하는 프로그램을 작성하도록 도와준다. CUDA 프로그램은 C 언어를 기반으로 하며, CPU영역인 호스트(host)와 GPU영역인 디바이스(device)를 모두 이용하여 실행하도록 되어 있다. 데이터 병렬성이 적은 부분은 호스트 코드로 구현되고 데이터 병렬성이 큰 부분을 디바이스 코드로 구현하게 되며, NVIDIA C 컴파일러(nvcc)는 이 둘을 분리하여 컴파일한다. 전체적인 코드는 순차적인 ANSI C 코드로 되어 있으며, 디바이스 코드의 핵심적인 부분은 커널(kernel) 함수 또는 커널이라고 불리는 데이터 병렬 함수이다.

커널은 일반적으로 데이터를 병렬로 처리하기 위해 이용하는 사용자 정의 함수로써 호출되면 GPU에 주어진 수의 쓰레드(thread)와 블럭(block)을 생성한다. 쓰레드는 하나의 GPU 코어에 대응되며, 블럭은 SM(Streaming Multiprocessor)이라고 불리는 쓰레드 배열에 대응된다. 생성된 쓰레드들은 커널 함수 내부의 같은 부분을 동시에 실행하게 된다. 따라서 일반적으로 반복문의 하나의 반복 실행을 하나의 쓰레드에서 수행하는 방식으로 커널 코드를 작성한다. 조건문의 경우는 특정 조건에 따라 그 코드를 쓰레드가 수행하느냐 아니냐의 분기가 일어나서 계산의 병렬성을 떨어뜨리기 때문에 커널에는 가능하면 피하는 것이 좋다.

GPU 메모리는 CPU 메모리와는 물리적으로 다른 칩에 존재하기 때문에 GPU에서 계산하기 위해서는 CPU 메모리에




```
main() {
1 allocate(host_memory,device_memory)
2 copy(host_memory to device_memory)
3 kernel<<<nblock,nthread>>>(device_memory)
4 copy(device_memory to host_memory)
5 deallocate(host_memory,device_memory)
}
```

Fig. 4 Basic structure of CUDA program

저장된 데이터를 GPU 메모리에 복사하는 것이 필요하다. CUDA 라이브러리는 이런 메모리 할당, 복사, 해제 등의 기능을 가지는 함수를 제공한다. 따라서 기본적인 CUDA 프로그램 구조는 Fig. 4와 같이 호스트와 디바이스에 메모리를 할당하고 GPU 계산에 필요한 데이터를 디바이스에 복사, 커널 함수를 호출하여 병렬 계산 수행, 계산 결과를 다시 호스트로 복사해와 후처리하고 할당했던 호스트와 디바이스 메모리를 해제하는 순서로 이루어진다.

3.2 GPU 계산 성능 최적화를 위한 고려사항

GPU의 계산 성능을 최대한으로 이끌어 내기 위해서는 GPU 장치의 내부 구조의 특성에 따른 몇 가지 사항들을 고려해야한다.

3.2.1 데이터 구조와 알고리즘(algorithm) 측면

첫 번째로 GPU 메모리의 접근은 통합적(coalesced memory access)이어야 하고 메모리 전달과정은 최소화되어야 한다. 통합적 접근이란 각 쓰레드가 연속된 메모리의 값에 접근할 때 한꺼번에 메모리의 데이터에 접근하는 것을 말한다. 따라서 데이터 구조를 통합적 접근이 가능하게 구성하면 메모리 접근의 비용이 줄어들어 성능을 향상시킬 수 있다. 그리고 앞서 3.1절의 CUDA 프로그램 구조에서 CPU와 GPU 사이의 데이터를 복사하는 속도가 느리기 때문에, 이 과정을 최소화할 수 있도록 GPU 계산이 수행되는 알고리즘을 구성하여야 한다. 본 연구의 CUDA 기반 유동 해석자에서는 한번 CPU 데이터를 전송하면 후처리를 하기 전까지는 GPU에서만 계산이 가능하도록 변수와 커널을 구성하였다.

두 번째로 데이터 경쟁(data race)이 일어나지 않으면서 데이터 병렬성(data locality)이 보장되는 알고리즘이 필요하다. 이 부분은 계산의 정확도와도 관계되어 있다. 데이터 경쟁은 서로 다른 쓰레드에서 동일한 메모리의 데이터에 쓰는 작업을 하는 경우에 발생하는 충돌하는 현상을 말한다. 경우에 따라서는 전혀 의도하지 않았던 결과를 불러올 수 있다. 데이터 병렬성의 보장이란 각 쓰레드의 작업이 서로 독립적으로 수행 가능하게 하는 것을 의미한다. 예를 들어, 0번 쓰레드의 결과가 1번 쓰레드의 계산에 인자로 이용되는 작업이 있다고 하자. 정확한 계산을 위해서는 0번 쓰레드의 결과를 기다린 후에, 1번 쓰레드의 작업이 수행되어야 하지만 이런 계산 순서는 동시에 여러 작업이 진행되지 못하기 때문에 병렬 계산의 효율성을 크게 떨어뜨릴 수밖에 없다. 따라서 각 쓰레드에서 독립적으로 동시에 작업이 진행되면서 데이터를 쓰는 데 있어서 충돌이 발생하지 않는 데이터 구조와 알고리즘이 요구된다. 이 부분 역시 CUDA 기반 유동 해석자에 반영하였다.

3.2.2 GPU 메모리 측면

GPU 계산에서 가장 중요한 요소 중 하나는 지연시간 은닉 기술(latency hiding technique)이다. 메모리에 접근하는 데는 일정 시간이 걸리는데 GPU에서는 이 접근으로 인한 지연시간 동안 다음 워프(warp)의 작업을 수행한다. 워프는 동시에 실행되는 쓰레드의 단위를 의미하는 것으로 32개의 쓰레드로 고정되어 있다. 이런 방법을 통해 메모리에 접근하는 시간으로 인한 지연효과를 최소화 하는 것이 GPU의 중요한 특징 중 하나이다. 따라서 쓰레드를 생성할 때 그 개수는 32의 배수로 배정해주는 것이 좋다고 알려져 있고, 실제 최적의 개수는 실험적으로 구해져야 한다. 본 연구에서도 반복적으로 최적의 개수를 찾는 과정을 수행하여 커널 호출시의 생성할 쓰레드 개수를 결정하였다.

GPU는 내부에 서로 다른 특성을 가지는 여러 종류의 메모리를 가지고 있다(Table 1). 각 메모리의 특성을 고려하여 사용하면 계산 성능을 높일 수 있다. 특히 메모리의 접근 속도(access latency)가 중요하게 작용한다. GPU에 복사되는 데이터는 일반적으로 전역 메모리(global memory)에 저장된다. 그렇지만 Fig. 5와 같이 이 메모리의 접근 속도가 느리기 때문에

Table 1 Features of device(GPU) memory

Memory	Location on/off chip	Access	Scope	Lifetime
Register	On	R/W	Thread	Thread
Local	Off	R/W	Thread	Thread
Shared	On	R/W	Block	Block
Global	Off	R/W	Application	Host allocation
Constant	Off	R	Application	Host allocation
Texture	Off	R	Application	Host allocation

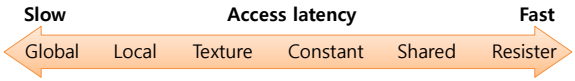


Fig. 5 Access latency of device memory

텍스쳐 메모리(texture memory)를 이용하면 접근하는데 걸리는 시간을 줄이고 계산 성능을 높일 수 있다. 텍스쳐 메모리는 읽는 작업만 가능하지만, 비통합적 메모리 접근(non-coalesced memory access)을 허용하기 때문에 복잡한 알고리즘에 대해서도 통합적 접근을 고려하지 않고 이용 가능하다. 따라서 CUDA 기반 유동 해석자에서 비통합적 접근이 일어나게 될 가능성이 높은 보존벡터와 격자에 대한 정보의 접근은 텍스쳐 메모리를 이용할 수 있도록 구성하였다.

블럭 단위로 공유가 가능한 공유 메모리(shared memory)를 활용하게 되면 전역 메모리 접근 횟수를 줄이고, 공유 메모리의 빠른 접근 속도를 이용하여 계산 성능을 높일 수 있다. 하지만 공유 메모리를 올바르게 활용하기 위해서는, 워프 개념을 고려하여 전역 메모리의 데이터를 공유 메모리로 가져와야 하고 쓰레드 간의 동기화도 신경써주어야 한다. 알고리즘에 따라서 공유 메모리의 사용이 가능한 부분에 대해 활용하도록 고려하였다.

커널 함수 내의 변수들은 기본으로 접근 속도가 가장 빠른 레지스터(register)에 저장된다. 그렇지만 GPU 전체에 대해서 사용가능한 레지스터의 양이 정해져 있기 때문에 쓰레드의 개수가 커져서 변수의 사용이 많아져 제한된 양을 넘기면 이 변수들을 접근 속도가 느린 지역 메모리(local memory)에 저장한다. 이는 계산 성능을 저하시키는 작용을 하며 이를 반영하여 역시 쓰레드 개수를 실험적으로 조절하게 되었다.

3.3 MLP가 적용된 고차 정확도 수치기법의 CUDA 기반 유동 해석자 개발

고차 정확도 수치기법으로 압축성 유동을 해석할 때, 계산 시간을 가장 많이 차지하는 핫스팟(hot-spot)은 공간 차분을 수행하는 부분이다. 격자 내부와 face에서의 구분구적법 계산을 포함하며, 모든 face에 대해 수치 플럭스 계산을 해야 하기 때문에 그 계산 시간이 길 수밖에 없다. 그래서 이 부분을 계산하는데 있어, 데이터를 병렬성이 보장되도록 하여 GPU의 쓰레드에 분배, 병렬 계산할 수 있는 디바이스 코드를 작성하여 CUDA 기반 유동 해석자를 완성하였다.

3.3.1 DG-MLP 기법

DG-MLP 기법의 지배방정식인 식 (7)에서 시간 적분 항과 divergence 항은 각 격자에서 계산되지만, 수치 플럭스는 격자의 각 face에서 계산되어야 한다. 격자 내부에서 계산되는 항

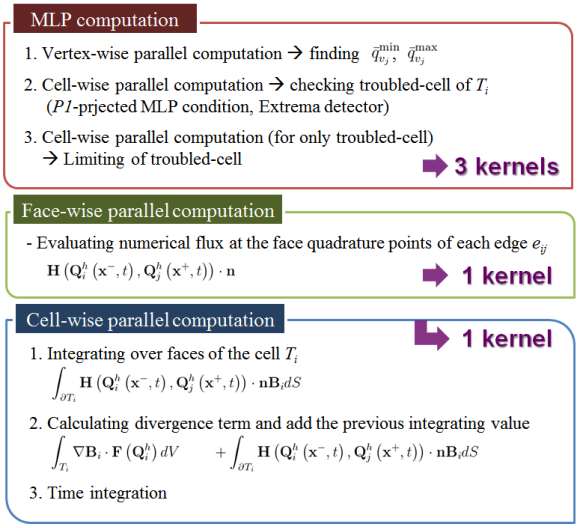


Fig. 6 Procedure of MLP-DG scheme on CUDA program

들은 하나의 쓰레드에 하나의 격자를 대응시켜 격자가 가지고 있는 보존벡터, 부피 등의 정보를 읽고 각 격자에 대해 병렬로 계산을 수행할 수 있다(cell 단위 계산으로 커널 작성).

수치 플럭스 항을 위와 같이 격자 단위로 병렬 계산하려면, 격자의 face 각각에 대해 인접 격자의 보존벡터와 법선방향 벡터의 정보를 읽어서 수치 플럭스를 계산하고, 이를 적분하여 수치 플럭스 항을 구하게 된다. 이는 결국에 한 face에 대하여 양쪽의 격자에서 중복된 계산을 하는 결과를 가져온다. 이런 중복된 계산을 피하기 위해 face에서 바로 인접한 격자에 대해서 더해주는 방식을 많이 사용하는데, 이 경우에 3.2.1항에서 언급한 데이터 충돌이 발생할 수 있다.

본 연구에서는 이런 점들을 고려해 하나의 face를 하나의 쓰레드에 할당, 수치 플럭스 항을 먼저 병렬 계산하여 저장하였다(face 단위 계산으로 커널 작성). 이후에 격자 단위 병렬 계산에서 다시 수치 플럭스 항을 적분하여 divergence 항에 합하고 최종적으로 시간 적분을 계산을 수행하도록 알고리즘을 구성하였다(Fig. 6의 Cell-wise parallel computation).

MLP 기법에서 troubled-cell을 찾기 위해서는 $\bar{q}_{v_j}^{\min}$, $\bar{q}_{v_j}^{\max}$ 두 값이 필요하다(식 (12)). 이 값들은 각 꼭지점 vj의 인접한 격자의 물성치 중에서 최소와 최대인 값을 찾는 형태로, 꼭지점 단위로 쓰레드에 나누어서 가속된 계산을 수행할 수 있다(vertex 단위 계산으로 커널 작성). 이후에는 식 (12), (13)을 이용하여 격자 단위 병렬 계산을 통해 troubled-cell인지를 확인하고 이 정보를 저장한다. 다시 격자 단위 병렬 계산을 통해 troubled-cell에 대해서만 공간 제한 과정을 수행하게 된다(Fig. 6의 MLP computation).

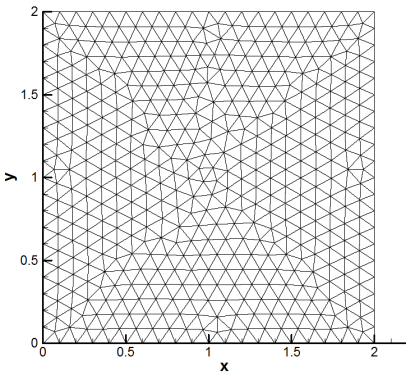


Fig. 7 Unstructured triangular mesh for convergence study

3.3.2 CPR-MLP 기법

CPR-MLP 기법의 구현도 DG-MLP 기법과 거의 비슷하게 진행된다. 식 (11)에서 lifting operator를 구하기 위해서는 수치 플럭스를 먼저 구해서 적분을 수행해야 한다. 따라서 앞서 DG-MLP 기법과 동일하게 수치 플럭스를 먼저 face 단위로 병렬 계산하고 지배방정식인 식 (10)을 격자 단위로 병렬 계산하는 방식으로 CUDA 기반 유동 해석자를 개발하였다.

4. 수치 실험 결과

완성된 CUDA 기반 유동 해석자를 이용하여 2496개의 코어를 가진 NVIDIA Tesla K20에서 해석을 수행하였다. GPU 계산 결과와 비교할 CPU 계산 결과는 2.40GHz Intel Xeon CPU를 사용하여 얻었다(순차 해석). CPU 병렬 해석은 CUDA와 비슷한 공유 메모리 프로그래밍 모델인 OpenMP 프로그래밍을 이용하였고, 동일 Xeon CPU 2개를 SMP(Symmetric Multi-Processing)로 구성한 총 12개의 코어에서 수행하였다.

4.1 Convergence study

먼저 고차 정확도 수치기법의 정확도를 확인하기 위하여 불연속 구간이 없는 다음과 같은 밀도 분포를 가지는 유동이 일정한 속도 $(u, v) = (1, 1)$로 전파되는 문제를 해석하였다.

$$\rho_0 = \begin{cases} 1+0.2\sin(\pi(x+y)) & : 2\text{차원} \\ 1+0.2\sin(\pi(x+y+z)) & : 3\text{차원} \end{cases}. \quad (16)$$

Euler 방정식(식 (1))에 대한 이 문제의 엄밀해는 다음과 같다.

$$\rho_{exact} = \begin{cases} 1+0.2\sin(\pi(x+y-t)) & : 2\text{차원} \\ 1+0.2\sin(\pi(x+y+z-t)) & : 3\text{차원} \end{cases}. \quad (17)$$

Fig. 7과 같이 균일한 크기의 삼각형으로 이루어진 비정렬 격자계에서 해석하였으며, 격자의 개수를 바꾸어 계산 자유도를 늘려 가면서 L^p-error를 계산하여 정확도를 확인할 수 있었다. $p = 1, 2$에 대해서는

$$L^p = \frac{1}{\Sigma_i |T_i|}\left(\sum_i \int_{T_i} \left| q_i^h(\mathbf{x}) - q_{exact}(\mathbf{x}) \right|^p dV\right)^{1/p}, \quad (18)$$

로 계산하였고, L^∞는 최대 차이를 이용하여 구하였다.

Table 2와 Table 3는 각각 2차원과 3차원 해석 결과이다. 먼저 Table 2에서 DG와 CPR 방법 모두 3차($P2$)와 4차($P3$) 정확도의 결과를 얻을 수 있었다. 또한, CPU 순차해석에서 계산한 값과 GPU에서 계산한 값의 차이가 machine-error 수준으

Table 2 L^2-error of 2-dimensional convergence study

2-D	DOF	L^2			
		CPU	GPU	Deviation	Order
DG $P2$ TVD-RK3	5,388	1.7618E-05	1.7618E-05	1.4718E-13	
	21,564	2.0922E-06	2.0922E-06	2.5251E-16	3.07
	86,472	2.5418E-07	2.5418E-07	4.3896E-14	3.04
	344,184	3.1270E-08	3.1270E-08	5.3137E-16	3.03
DG $P3$ SSP-RK (4,5)	8,980	4.5783E-07	4.5783E-07	2.5556E-16	
	35,940	2.6414E-08	2.6414E-08	3.7100E-16	4.11
	144,120	1.6140E-09	1.6140E-09	6.5519E-16	4.03
	573,640	1.0049E-10	1.0050E-10	1.5076E-14	4.02
CPR $P2$ TVD-RK3	5,388	5.1504E-05	5.1504E-05	2.9847E-13	
	21,564	6.1691E-06	6.1691E-06	2.6681E-15	3.06
	86,472	7.5423E-07	7.5423E-07	9.5874E-14	3.03
	344,184	9.3268E-08	9.3268E-08	4.4811E-15	3.03
CPR $P3$ SSP-RK (4,5)	8,980	1.2747E-06	1.2747E-06	1.8389E-13	
	35,940	7.3741E-08	7.3741E-08	4.7397E-15	4.11
	144,120	4.5179E-09	4.5179E-09	2.6492E-15	4.02
	573,640	2.8280E-10	2.8280E-10	2.5730E-15	4.01

Table 3 L^∞ and L^1-error of 3-dimensional convergence study

3-D (GPU)	DOF	L^∞		L^1	
		Error	Order	Error	Order
DG $P2$ TVD-RK3	103,680	3.2050E-03		1.8527E-04	
	245,760	1.5303E-03	2.57	7.9750E-05	2.93
	480,000	8.1469E-04	2.83	4.0711E-05	3.01
	829,440	4.8211E-04	2.88	2.2938E-05	3.15
DG $P3$ SSP-RK (4,5)	207,360	4.8391E-04		1.2556E-05	
	491,520	1.6408E-04	3.76	3.8417E-06	4.12
	960,000	6.6254E-05	4.06	1.5577E-06	4.05
	1,658,880	3.2427E-05	3.92	7.3962E-07	4.09
CPR $P2$ TVD-RK3	103,680	1.5008E-02		4.8089E-04	
	245,760	6.3655E-03	2.98	1.3928E-04	4.31
	480,000	3.1726E-03	3.12	6.8145E-05	3.20
	829,440	1.7712E-03	3.20	3.9252E-05	3.03
CPR $P3$ SSP-RK (4,5)	207,360	1.5955E-03		3.0265E-05	
	491,520	5.1932E-04	3.90	9.0501E-06	4.20
	960,000	2.1765E-04	3.90	3.5697E-06	4.17
	1,658,880	1.0550E-04	3.97	1.6731E-06	4.16

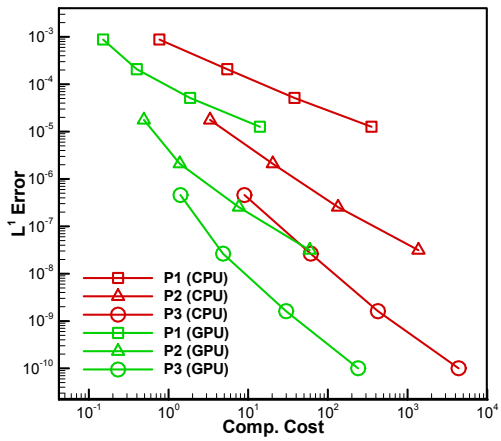


Fig. 8 Plot of computational cost-accuracy for DG method

로 작다는 것을 확인할 수 있다. L^{∞}, L^{1}에 대해서도 비슷한 결과를 얻을 수 있었고, 고차 정확도 수치기법이 CUDA 기반 유동 해석자에서 정확하게 구현되었다는 것을 확인할 수 있었다. Table 3는 GPU로 계산한 결과만 나타내었다. 3차원으로 확장한 유동 해석자도 고차의 정확도가 얻어져, 수치기법이 2차원과 3차원에 대해 모두 제대로 구현되었음을 확인하였다.

Fig. 8은 계산 비용 대비 L^{1}-error를 나타낸 그래프로, 수치기법의 정확도가 높으면 같은 계산 비용으로도 높은 정확도를 얻을 수 있다. 또한 GPU를 이용하여 계산을 가속하게 되면 역시 적은 계산 비용으로 높은 정확도를 얻을 수 있다.

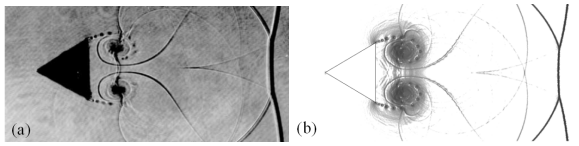
4.2 Schardin 문제

다음으로 충격파가 존재하는 2차원 문제를 해석한 결과이다. Schardin 문제는 마하 1.34의 이동충격파가 60°의 쐐기와 부딪히면서 생기는 유동 현상이다. 쐐기의 끝단에서 발생하는 와류(primary vortex)와 반사 충격파가 상호작용을 하면서 복잡한 유동 구조가 생긴다.

Fig. 9은 이 문제의 유동 결과를 보여준다. (a)는 실험에서 얻어진 결과이고[17], (b)는 약 39만 개의 균일한 삼각형으로 이루어진 비정렬 격자계에 대해서 DG-MLP 기법($P3$)으로 해석한 결과다. 두 결과 모두에서 충격파와의 상호작용에 의한 유동의 불안정성으로 나타나는 작은 원형의 vortexlet이 포착

Table 4 Computational efficiency for Schardin's problem

Computational cost	CPU (12 cores)	GPU (K20)	Speedup
DG $P2$	1.0	0.41	x 2.44
DG $P3$	2.84	1.61	x 1.57
CPR $P2$	1.0	0.39	x 2.53
CPR $P3$	2.22	1.13	x 1.96



(a) (b)

Fig. 9 Comparison of shadowgraphs from experiments(a)[16] with numerical Schileren image of computations by DG method(b)

된 것을 확인할 수 있다.

Table 4는 이 문제의 해석에 소요된 계산 시간을 바탕으로 CPU와 GPU의 계산 효율성을 비교한 결과이며, DG와 CPR 방법의 $P2$ 근사를 기준(1.0)으로 삼아 상대적인 계산 비용을 나타내었다. Speedup은 CPU 병렬 계산(12개 코어) 시간 대비 GPU 계산 시간으로 구하였다. 이는 GPU의 구조에 맞게 병렬화하여(알고리즘 변경 및 최적화) 수행한 해석이 CPU를 사용한 병렬화 해석에 비해 얼마나 빠른가를 나타낸다. $P2$ 근사에서 $P3$ 근사보다 큰 speedup을 가지는 결과가 얻어졌다.

GPU 병렬 계산 시에 주로 격자 단위로 나누는 방법으로 계산을 나누어서 수행한다. 따라서 격자 내부를 근사하는 정확도가 증가하면 각 격자를 담당하는 쓰레드에서 접근해야 하는 데이터가 늘어난다. 결국 사용하는 변수의 양도 증가하면서 3.2.2항에서 설명하였던 레지스터의 사용이 제한되어 접근 속도가 느린 지역 메모리로의 접근이 늘어나고 계산 성능을 감소시켜서 $P3$ 근사의 speedup이 감소한 것으로 보인다. $P4$ 이상의 높은 정확도의 근사에서는 격자당 계산량이 더욱 증가하여 speedup에 영향을 줄 것이다. 따라서 GPU 특성을 고려하여, 쓰레드를 격자 단위로 나누는 병렬 계산 대신에 solution point같은 더 작은 단위로 나누어 병렬화한다면 GPU의 이론적인 성능에 가까운 speedup을 얻을 수 있을 것이다.

4.3 3차원 폭발 문제

3차원 문제는 간단한 폭발 문제에 대하여 해석을 수행하였다. 다음과 같이 구면을 따라서 불연속의 밀도 차이를 가짐에 따라서 발생하는 폭발 현상을 해석하였다.

$$(\rho_0, u_0, v_0, w_0, p_0) = \begin{cases} (1, 0, 0, 0, 1) & \text{if } r < 0.4 \\ (0.125, 0, 0, 0, 1) & \text{otherwise} \end{cases} \quad (19)$$

1/8 단위 구에 대해서 약 46만 개의 균일한 크기의 사면체로 이루어진 비정렬 격자계에 대해서 해석한 결과가 Fig. 10의 안쪽에 나타나있다. Fig. 10의 그래프는 대각선을 따라 밀도 분포를 나타낸 것으로 "Reference"로 표기된 결과는 폭발 문제를 1차원으로 변환하여 8,000개의 격자로 해석한 결과이다. 고차 정확도를 가지는 CPR 방법(빨간색과 초록색 선)이 유한

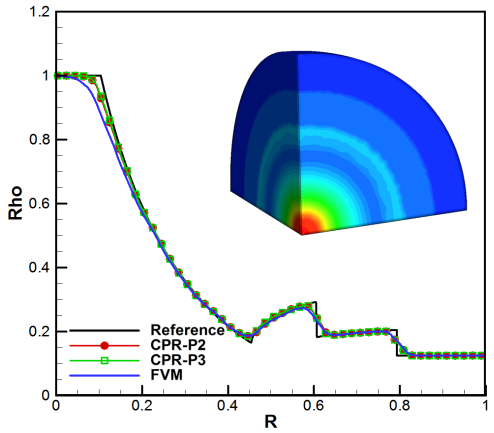


Fig. 10 Result of 3-D explosion problem(inside) and density distribution along diagonal line

체적법(파란색 선)에 비해 팽창이 시작되는 부근($x=0.1$)과 contact 부근($x=0.6$)에서 높은 해상도를 보여준다.

이 문제에 대해서도 CPU와 GPU의 계산 효율성을 비교하여, 앞서 Schardin 문제와 동일한 방식으로 Table 5로 정리하였다. DG와 CPR 방법 모두 3차원 경우에 speedup이 2차원 경우보다 감소되었다. 이 역시 앞 절에서 설명한 내용과 비슷하게 2차원에서 삼각형인 격자가 3차원에서 사면체로 바뀌면서 격자당 계산량이 크게 증가되었기 때문이다. 또한 사면체의 face 적분은 삼각형에 대해 이루어지기 때문에 실제 계산 시간에서 CPR 방법이 DG 방법에 비해 최대 50% 이상 빠른 결과가 얻어졌다(Fig. 11의 오른쪽 그래프).

5. 결 론

고차 정확도 CFD 방법으로 충격파가 존재하는 압축성 유동을 정밀하게 해석하였다. 이 때 불연속 구간에서의 수치 안정성을 보장하기 위하여 MLP를 적용하였고, 방대한 계산량을 효율적으로 처리하기 위하여 NVIDIA Tesla K20 GPU를 이용한 병렬 계산을 수행하였다. GPU에서 정확하고 효율적인 계산을 하기 위해서 데이터 병렬성이 보장되면서 데이터 경쟁이 없도록 GPU 내부 구조를 고려하여 데이터 구조와 알고리

Table 5 Computational efficiency for 3-D explosion problem

Computational cost	CPU (12 cores)	GPU (K20)	Speedup
DG *P2*	1.0	0.69	x 1.45
DG *P3*	4.82	3.55	x 1.38
CPR *P2*	1.0	0.60	x 1.67
CPR *P3*	3.20	2.61	x 1.23

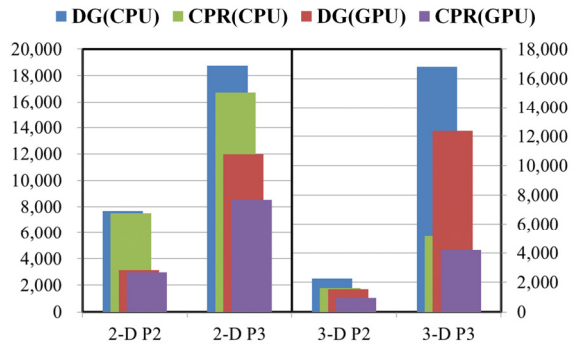


Fig. 11 Comparison of real computational time(sec.) of CPU serial and GPU parallel computations for 2-D (Schardin's problem) and 3-D (explosion problem) cases

즘을 조정하여 CUDA 기반 유동 해석자를 개발하였다. 이를 이용해 실제 2차원, 3차원 유동 문제들을 해석하여 CPU 계산과 동일한 결과를 얻었으며 충격파가 있는 유동 현상을 정확하게 포착 가능하였다.

계산 효율성 측면에서는 GPU 계산이 CPU 12개 코어를 사용한 것보다 대략 1.5~2.5배 정도 계산 시간이 단축되는 것을 확인하였다. 연구에 사용한 장비들의 Fig. 1과 같은 이론적인 계산 성능은 Tesla K20이 1.17GFlops, Intel Xeon CPU가 0.44GFlops로 이론적으로는 약 2.65배 정도의 성능차이를 가진다. 이와 비교하여, 논문에서 수행한 GPU 계산이 어느 정도 병렬화가 잘 이루어진 것으로 보인다. 향후에 GPU 병렬 계산에 대한 추가적인 최적화 과정도 가능할 것으로 예상된다.

후 기

본 연구는 교육과학기술부의 우주기초원천기술개발 사업(NSL, National Space Lab, NRF-2009-0091724), 첨단사이언스 교육허브개발 사업(EDISON, NRF-2011-0020559), 2014년도 BK21 플러스 사업 및 서울대학교 항공우주신기술연구소의 지원을 받아 수행되었습니다.

References

[1] 1998, Cockburn, B. and Shu, C.-W., "The Runge-Kutta Discontinuous Galerkin Method for Conservation Laws V: Multidimensional Systems," *J. of Comp. Physics*, Vol.141, pp.199-224.

[2] 2007, Huynh, H.T., "A Flux Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin

Methods," *18th AIAA Computational Fluid Dynamics Conference*, AIAA2007-4079.

[3] 2009, Wang, Z.J. and Gao, H., "A Unifying Lifting Collocation Penalty Formulation Including the Discontinuous Galerkin, Spectral Volume/Difference Methods for Conservation Laws on Mixed Grids," *J. of Comp. Physics*, Vol.228, pp.8161-8186.

[4] 2008, Yoon, S.-H., Kim, C. and Kim, K.-H., "Multi-dimensional Limiting process for Three Dimensional Flow Physics Analyses," *J. Comp. Physics*, Vol227, pp6001-6043.

[5] 2010, Park, J.S., Yoon, S.-H. and Kim, C., "Multi-dimensional Limiting Process for Hyperbolic Conservation Laws on Unstructured Grids," *J. of Comp. Physics*, Vol.229, pp.788-812.

[6] 2012, Park, J.S. and Kim, C., "Multi-dimensional limiting process for finite volume methods on unstructured grids," *Computers and Fluids*, Vol.65, pp.8-24.

[7] 2014, Park, J.S. and Kim, C., "Higher-order Multi-dimensional Limiting Strategy for Discontinuous Galerkin Methods in Compressible Inviscid and Viscous Flows," *Computers & Fluids*, Vol.96, pp377-396.

[8] 2014, Park, J.S., Chang. T.K., and Kim, C., "Higher-order Multi-dimensional Limiting Strategy for Correction Procedure via Reconstruction," *AIAA 52nd Aerospace Sciences Meeting*, AIAA2014-0772.

[9] 2012, NVIDIA, *CUDA C Programming Guide*, Ver.5.0.

[10] 2009, Corrigan, A., Camelli, F. and Löhner, R., "Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware," *19th AIAA Computational Fluid Dynamics Conference*, AIAA2009-4001.

[11] 2009, Klöockner, A., Warburton, T. and Hesthaven, J.S., "Nodal Discontinuous Galerkin Methods on Graphics Processors," *J. of Comp. Physics*, Vol.228, pp.7863-7882.

[12] 2013, Zimmerman, B.J. and Wang, Z.J., "The Efficient Implementation of Correction Procedure via Reconstruction with GPU Computing," *21st AIAA Computational Fluid Dynamics Conference*, AIAA2013-2692.

[13] 1999, Karniadakis, G.E. and Sherwin, S.J., "Spectral/hp element methods for CFD," *Oxford University Press*, USA.

[14] 2003, Kim, S.-s., Kim, C., Rho, O.-H. and Hong, S.K., "Cures for the Shock Instability: Development of a Shock-stable Roe Scheme," *J. of Comp. Physics*, Vol.185, pp.342-374.

[15] 2013, Wang, Z.J. and Gao, H., "A Conservative Correction Procedure via Reconstruction Formulation with the Chain-Rule Divergence Evaluation," *J. of Comp. Physics*, Vol.232, pp.7-13.

[16] 2001, Spiteri, R.J. and Ruuth, S.J., "A New Class of Optimal High-Order Strong Stability-Preserving Time Discretization Methods," *SIAM J. on Num. Analysis*, Vol.40, No.2, pp.469-491.

[17] 2000, Chang, S.-M. and Chang, K.-S., "On the Shock-Vortex Interaction in Schardin's Problem," *Shock Waves*, Vol.10, pp.333-343.