GPU를 이용한 효율적인 비압축성 자유표면유동 해석

홍 환 의,[1] 안 형 택,[*1] 명 훈 주[2]
[1]울산대학교 조선해양공학부
[2]한국과학기술정보연구원

AN EFFICIENT INCOMPRESSIBLE FREE SURFACE FLOW SIMULATION USING GPU

H.E. Hong,[1] H.T. Ahn[*1] and H.J. Myung[2]
[1]School of Naval Architecture and Ocean Engineering, University of Ulsan
[2]Super Computing Center, KISTI

This paper presents incompressible Navier-Stokes solution algorithm for 2D Free-surface flow problems on the Cartesian mesh, which was implemented to run on Graphics Processing Units(GPU). The INS solver utilizes the variable arrangement on the Cartesian mesh, Finite Volume discretization along Constrained Interpolation Profile-Conservative Semi-Lagrangian(CIP-CSL). Solution procedure of incompressible Navier-Stokes equations for free-surface flow takes considerable amount of computation time and memory space even in modern multi-core computing architecture based on Central Processing Units(CPUs). By the recent development of computer architecture technology, Graphics Processing Unit(GPU)'s scientific computing performance outperforms that of CPU's. This paper focus on the utilization of GPU's high performance computing capability, and presents an efficient solution algorithm for free surface flow simulation. The performance of the GPU implementations with double precision accuracy is compared to that of the CPU code using an representative free-surface flow problem, namely. dam-break problem.

Key Words : Free-Surface Flow(자유표면유동), GPU(그래픽처리장치), VOF(Volume-of-Fluid), Navier-Stokes Equation(나비어-스톡스 방정식), Incompressible flow(비압축성 유동)

1. 서 론

자유표면유동은 우리가 손쉽게 만날 수 있는 자연현상이다. 강 혹은 바다 표면유동은 물론이고, 선박과 해양구조물 주위에서 발생하는 유동, 화학분야의 다상유동 등에서 자유표면 유동을 찾아 볼 수 있다. 이러한 다양한 분야에서 보이는 자유표면유동에 대한 연구들은 지난 몇 십년간 이뤄져 왔고, 컴퓨터 프로세서 성능의 발전으로 수많은 과학자들이 이를 수치적으로 계산하기위해서 많은 연구들을 수행하였다. 특히 자유표면유동의 경계면을 질량보존법칙을 만족시키며, 보다 정확하게 예측하여 실제 자유표면의 움직임을 나타내기 위한 다양한 수치기법들이 개발되고 제안되고 있다.

본 연구에서는 지배방정식으로 단상 혹은 다상으로 존재하는 점성을 가진 비압축성 유동의 연속 방정식, Navier-Stokes 방정식을 고려한다. 현재 다상유동 및 자유표면이 존재하는 복잡한 유동현상을 해석하기 위한 수치기법으로 Volume of Fluidt법[1] 과 Level Set법[2]이 많이 사용된다. 이러한 유동의 수치해석법에서는 대류항의 수치적 확산과 수치 진동 등의 문제를 보여 왔다. 이러한 수치적 오류를 줄이기 위하여 3차 정도를 가지는 해법인 Constrained Interpolation Profile법(CIP법, [3])과 각기 유체의 고차 모멘트 정보를 Tracking함으로써 자유표면의 변화를 보다 정확히 표현 가능한 Moment of Fluid법(MOF법, [4]) 등이 개발 되었다. CIP법은 열유동 방정식을 풀 때 오차가 큰 대류항을 3차 스플라인 보간법을 이용하여 계산함으로써 밀도 차이가 큰 물질 사이의 거동이나 큰 밀도변화가 있는 상경계의 유동해석에서도 수치 확산을 크게 줄일

Received: December 5, 2011, Revised: June 8, 2012,
Accepted: June 11, 2012.
* Corresponding author, E-mail: htahn@ulsan.ac.kr
DOI http://dx.doi.org/10.6112/kscfe.2012.17.2.035

수 있다는 장점이 있다. CIP법 자체로는 비보존형의 형태이므로 이를 보존형 방정식으로 변화하여 보존성이 뛰어난 형태로 변형한 보존형 방법이 Constrained Interpolation Profile-Conservative Semi-Lagrangian법(CIP-CSL, [5])이다. 보존형 CIP법에도 다양한 변형이 존재하며 그중에서 CSL2은 보간 함수의 구성이 쉽고 메모리사용이 비교적 적기 때문에 보존형 해법으로 자주 사용되고 있다.

이러한 자유표면유동을 비롯하여 다른 CFD 문제를 해석하기 위해서는 많은 연산비용과 연산시간을 요구한다. 특히 유동해석을 위해서는 Navier-Stokes 방정식을 계산하게 되는데 포아송(Poisson equation)방정식을 푸는 반복 작업에 계산시간이 많이 걸리게 된다. 그래서 슈퍼컴퓨터와 CPU Cluster등을 사용해야하며 이러한 방정식의 해법에 많은 해석시간이 요구되는데, 본 논문에서는 이 문제를 신속하게 해결하기 위해 그래픽 프로세서(GPU)에 기반을 둔 병렬처리 기법을 도입하고자 한다.

최근 GPU의 수치연산능력이 병렬 아키텍처 발전에 힘입어 CPU의 수치연산능력을 따라잡았다고 알려져 있다. 또한 GPU 개발에 있어 선두를 업체라고 할 수 있는 nVIDIA사는 CUDA (Compute Unified Device Architecture)라는 소프트웨어 아키텍처를 개발하고, 이를 통하여 GPU환경에서의 병렬 연산을 가능하게 하는 수치해석 프로그램의 개발을 가능하게 하였다. 그러나 CPU에서의 일반적인 수치해석 프로그래밍과는 달리, GPU의 아키텍처상의 고유한 특성을 고려하지 않고 프로그래밍을 수행할 경우 성능이 저하될 수 있으므로 이를 고려한 프로그래밍이 필요하다.

본 연구에서는 비압축성 자유표면유동 시뮬레이션을 해보았고, CPU를 이용한 연산시간과 CUDA를 통해 GPU를 이용한 연산시간을 비교하였다.

2. 지배방정식 및 계산방법론

2.1 비압축성 유동해석을 위한 Navier-Stokes 방정식

본 연구에서는 지배방정식으로 다음과 같은 단상 혹은 다상으로 존재하는 점성을 가진 비압축성 유동의 연속 방정식, Navier-Stokes방정식을 고려한다.

$$\int_{\Gamma} \boldsymbol{u}\cdot\boldsymbol{n}dS = 0 \tag{1}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\boldsymbol{u}dV + \int_{\Gamma}\boldsymbol{u}(\boldsymbol{u}\cdot\boldsymbol{n})dS = -\frac{1}{\rho}\int_{\Gamma}p\boldsymbol{n}dS + \frac{1}{\rho}\int_{\Gamma}\boldsymbol{\tau}\cdot\boldsymbol{n}dS + \frac{\boldsymbol{F_{bf}}}{\rho} \tag{2}$$

여기서, u는 속도, p는 압력, ρ는 밀도, ν는 동점성계수, f_i는 중력과 같은 외력항을 뜻한다.

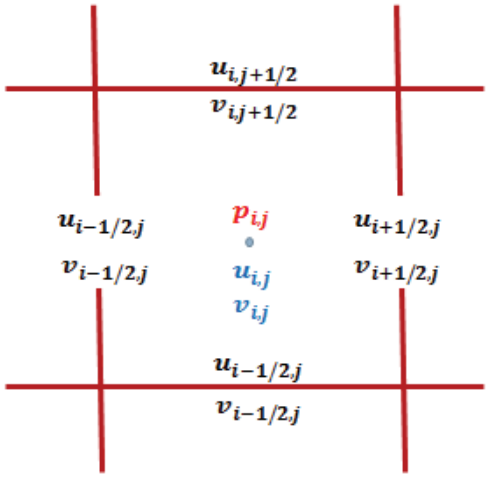


Fig. 1 Schematic grid figures in two dimensional case

위의 방정식들의 해를 구하기 위하여 Fractional step method[6]를 이용하여 식(2)을 다음과 같은 3단계로 나누어 계산한다.

- 대류 단계 :

$$\frac{\partial}{\partial t}\int_{\Omega}\boldsymbol{u}dV + \int_{\Gamma}\boldsymbol{u}(\boldsymbol{u}\cdot\boldsymbol{n})dS = 0 \tag{3}$$

- 대류 단계 1:

$$\frac{\partial}{\partial t}\int_{\Omega}\boldsymbol{u}dV = \frac{1}{\rho}\int_{\Gamma}\boldsymbol{\tau}\cdot\boldsymbol{n}dS + \frac{\boldsymbol{F}_{bf}}{\rho} \tag{4}$$

- 비대류 단계 2:

$$\int_{\Gamma}\boldsymbol{u}\cdot\boldsymbol{n}dS = 0 \tag{5}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\boldsymbol{u}dV = -\frac{1}{\rho}\int_{\Gamma}p\boldsymbol{n}dS \tag{6}$$

대류단계에서는 수치적 확산을 줄이기 위해 3차 스플라인 함수를 사용한 CIP-CSL법을 이용하여 계산하고, 비대류 단계에서는 cell averages 와 boundary values를 사용하는 Volume/Surface Integrated Average Method[7]를 이용하여 계산한다.

2.2 직교격자 상에서 변수의 배치

본 논문은 Fig. 1과 같이 직교격자의 cell averages 과 boundary values를 동시에 사용하는 grid method를 사용한다.

Cell averages 값 $u_{i,j,k}$, $v_{i,j,k}$, $w_{i,j,k}$, $p_{i,j,k}$은 직교격자의 중앙에 위치하고 boundary values값 $u_{i-1/2,j,k}$, $u_{i,j-1/2,k}$, $u_{i,j,k-1/2}$, $v_{i-1/2,j,k}$, $v_{i,j-1/2,k}$, $v_{i,j,k-1/2}$, $w_{i-1/2,j,k}$,

$w_{i,j-1/2,k}$, $w_{i,j,k-1/2}$은 직교격자의 경계에 위치한다. Cell average(or Volume Integrated Average) 와 boundary values(or Surface Integrated Average)는 변수로 사용되며 다음과 같이 정의한다.

$$u_{i,j,k} = \frac{1}{\triangle x \triangle y \triangle z}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}}\int_{z_{k-1/2}}^{z_{k+1/2}} u(x,y,z)dxdydz \quad (7)$$

$$u_{i-1/2,j,k} = \frac{1}{\triangle y \triangle z}\int_{y_{j-1/2}}^{y_{j+1/2}}\int_{z_{k-1/2}}^{z_{k+1/2}} u(x_{i-1/2},y,z)dydz \quad (8)$$

$$u_{i,j-1/2,k} = \frac{1}{\triangle x \triangle z}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{z_{k-1/2}}^{z_{k+1/2}} u(x,y_{j-1/2},z)dxdz \quad (9)$$

$$u_{i,j,k-1/2} = \frac{1}{\triangle x \triangle y}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}} u(x,y,z_{k-1/2})dxdy \quad (10)$$

2.3 CIP-CSL 방법

CIP-CSL법은 식(11)의 보존방정식을 계산하는데 사용된다.

$$\frac{\partial}{\partial t}\int_{\Omega}\phi dV + \int_{\Gamma}\phi(\boldsymbol{u}\cdot\boldsymbol{n})dS = 0 \quad (11)$$

여기서 ϕ는 임의의 물리량이다. CIP-CSL법에는 여러 가지 변주가 존재하는데, 본 논문에서는 CIP-CSL2[8]를 사용해 대류항을 계산하였다. 1차원 CIP-CSL2의 경우, 두 개의 점 $x_{i-1/2}$와 $x_{i+1/2}$에서의 물리량과 그 사이에 존재하는 하나의 평균값 ϕ_i을 고려한다. 두 개의 계산점사이의 프로파일을 식(12)의 2차 다항식으로 근사화 한다.

$$\Phi_i(x) = a_i(x-x_{i-1/2})^2 + b_i(x-x_{i-1/2}) + \phi_{i-1/2} \quad (12)$$

$$a_i = \frac{1}{\triangle x^2}(-6\phi_i + 3\phi_{i-1/2} + 3\phi_{i+1/2}) \quad (13)$$

$$b_i = \frac{1}{\triangle x}(6\phi_i - 4\phi_{i-1/2} - 2\phi_{i+1/2}) \quad (14)$$

중간항$\Phi_i(x)$를 사용하고 식(15)의 미분형 보존방정식을 이용해 boundary value $\phi_{i-1/2}$을 업데이트한다.

$$\frac{\partial\phi}{\partial t} + u\frac{\partial\phi}{\partial t} = -\phi\frac{\partial u}{\partial x} \quad (15)$$

식(15)에 semi-Lagrangian approach를 이용하여 접근하여 cell average ϕ_i는 finite volume formulation을 통해서 업데이트 된다.

$$\phi^*_{i-1/2} = \begin{cases}\Phi_{i-1}(x_{i-1/2} - u_{i-1/2}\triangle t) & \text{if } u_{i-1/2} \geq 0 \\ \Phi_i(x_{i-1/2} - u_{i-1/2}\triangle t) & \text{if } u_{i-1/2} \leq 0\end{cases} \quad (16)$$

$$\frac{\partial\phi}{\partial t} = -\phi^*\frac{\partial u}{\partial x} \quad (17)$$

$$\frac{\partial}{\partial t}\int_{x_{i-1/2}}^{x_{i+1/2}}\phi dx = -\frac{1}{\triangle x}(F_{i+1/2} - F_{i-1/2}) \quad (18)$$

여기서 $F_{i-1/2}$는 flux이다.

$$F_{i-1/2} = \begin{cases}-\int_{x_{i-1/2}}^{x_{i-1/2}-u_{i-1/2}\triangle t}\Phi_{i-1}(x)dx & \text{if } u_{i-1/2} \geq 0 \\ -\int_{x_{i-1/2}}^{x_{i-1/2}-u_{i-1/2}\triangle t}\Phi_i(x)dx & \text{if } u_{i-1/2} \geq 0\end{cases} \quad (19)$$

3차원으로 확장을 할 경우, x축 방향은 기존의 CIP-CSL법으로 구한 $\phi^n_{i,j,k}$, $\phi^n_{i-1/2,j,k}$을 통해서 $\phi^*_{i,j,k}$, $\phi^*_{i-1/2,j,k}$를 계산한다. $\phi^n_{i,j-1/2,k}$와 $\phi^n_{i,j,k-1/2}$은 식(20), 식(21)의 TEC (Time Evolution Converting)으로 업데이트 한다.

$$\phi^*_{i,j-1/2,k} = \phi^n_{i,j-1/2,k} + \frac{1}{2}(\phi^*_{i,j,k} - \phi^n_{i,j,k} + \phi^*_{i,j-1,k} - \phi^n_{i,j-1,k}) \quad (20)$$

$$\phi^*_{i,j,k-1/2} = \phi^n_{i,j,k-1/2} + \frac{1}{2}(\phi^*_{i,j,k} - \phi^n_{i,j,k} + \phi^*_{i,j,k-1} - \phi^n_{i,j,k-1}) \quad (21)$$

y, z축 방향으로도 이와 유사한 방식으로 접근하여서 계산하였다.

2.4 체적/면적 적분 평균법(Volume/Surface Integrated Average Method)

비대류항들은 multi-moment method를 기초로 한 Volume/Surface Integrated Average Method으로 계산을 수행한다. 점성응력항은 식(22)의 유한체적식 형태로 계산된다.

$$\frac{1}{\rho}\int_{\Gamma}\tau\cdot\boldsymbol{n}dS = \frac{1}{\rho_{i,j,k}}\left(\frac{\tau_{i+1/2,j,k} - \tau_{i-1/2,j,k}}{\triangle x} + \frac{\tau_{i,j+1/2,k} - \tau_{i,j-1/2,k}}{\triangle y} + \frac{\tau_{i,j,k+1/2} - \tau_{i,j,k-1/2}}{\triangle z}\right) \quad (22)$$

$u_{i-1/2,j,k}$와 같은 boundary values는 앞선 CIP-CSL부분에서

설명된 TEC를 통해서 업데이트 된다. 외력항 $\boldsymbol{F_{bf}}$도 이와 유사한 방식의 계산과정을 수행한다. 식 (6)과 식 (23)을 이용하면 식 (24)의 포아송 방정식을 얻게 된다.

$$\int_{\Gamma} \boldsymbol{u}^{n+1} \cdot \boldsymbol{n} dS = 0 \tag{23}$$

$$-\frac{1}{\rho}\int_{\Gamma} \nabla p^{n+1} \cdot \boldsymbol{n} dS = \frac{1}{\triangle t}\int_{\Gamma} \boldsymbol{u}^{*} \cdot \boldsymbol{n} dS \tag{24}$$

여기서 $\boldsymbol{u}^{*}$는 비대류 단계 1의 계산이 끝난 후 중간속도이다. 식 (24)을 이산화하면 식(25)와 같다.

$$\begin{aligned} &\frac{(\frac{1}{\rho^{n+1}}\partial_x P^{n+1})_{i+1/2,j,k} - (\frac{1}{\rho^{n+1}}\partial_x P^{n+1})_{i-1/2,j,k}}{\triangle x} \\ &+ \frac{(\frac{1}{\rho^{n+1}}\partial_y P^{n+1})_{i,j+1/2,k} - (\frac{1}{\rho^{n+1}}\partial_x P^{n+1})_{i,j-1/2,k}}{\triangle y} \\ &+ \frac{(\frac{1}{\rho^{n+1}}\partial_z P^{n+1})_{i,j,k+1/2} - (\frac{1}{\rho^{n+1}}\partial_x P^{n+1})_{i,j,k-1/2}}{\triangle z} \\ &= \frac{1}{\triangle t}(\frac{u^{*}_{i+1/2,j,k} - u^{*}_{i-1/2,j,k}}{\triangle x} + \frac{v^{*}_{i,j+1/2,k} - v^{*}_{i,j-1/2,k}}{\triangle y} \\ &\qquad + \frac{w^{*}_{i,j,k+1/2} - w^{*}_{i,j,k-1/2}}{\triangle z}) \end{aligned} \tag{25}$$

여기서,

$$(\frac{1}{\rho^{n+1}}\partial_x P^{n+1})_{i-1/2,j,k} \equiv \frac{2}{\rho^{n+1}_{i,j,k} + \rho^{n+1}_{i-1,j,k}} \frac{p^{n+1}_{i,j,k} - p^{n+1}_{i-1,j,k}}{\triangle x}$$

이다.

본 논문에서는 R-B Gauss-Seidel SOR을 이용하여 포아송 방정식을 풀고, p^{n+1}을 식 (26)에 적용하여 boundary values의 속도 $u_{i-1/2,j,k}$, $v_{i,j-1/2,k}$, $w_{i,j,k-1/2}$를 구하게 된다.

$$u^{n+1}_{i-1/2,j,k} = u^{*}_{i-1/2,j,k} - (\frac{1}{\rho^{n+1}}\partial_x p^{n+1})_{i-1/2,j,k} \tag{26}$$

그 외의 속도 $u_{i,j,k}$, $v_{i,j,k}$, $w_{i,j,k}$, $u_{i,j-1/2,k}$, $u_{i,j,k-1/2}$, $v_{i-1/2,j,k}$, $v_{i,j,k-1/2}$, $w_{i-1/2,j,k}$, $w_{i,j-1/2,k}$는 TEC를 이용해서 업데이트 한다.

3. GPU 병렬화

3.1 GPU활용을 위한 프로그래밍언어: CUDA

nVIDIA에서 개발한 CUDA는 GPU에서 수행하는 알고리즘을 코딩하는데 있어서 C프로그래밍 언어를 사용할 수 있도록

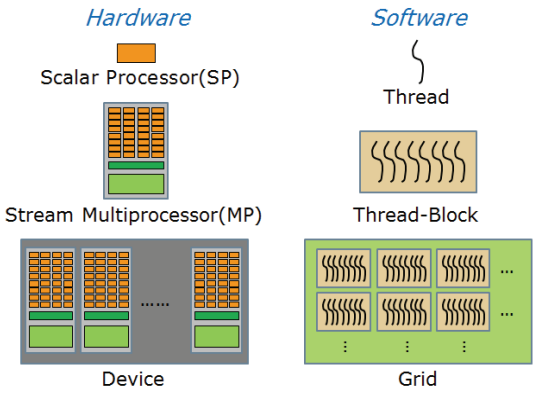


Fig. 2 GPU Physical, Logical Architecture

하는 기술이다. 이는 기존의 CPU와 달리 많은 수의 프로세서를 포함하고 있는 GPU를 이용하여 연산을 수행하게 해주는 GPU 프로그래밍 언어이다. 기존에 GPU를 이용하여 수학적인 연산을 하기 위해선 그래픽 함수를 이용한 쉐이딩 프로그래밍을 통하여 구현할 수 있었다. 그러나 쉐이딩 프로그래밍은 그래픽에 대한 사전지식을 많이 필요로 하기 때문에 접근이 쉽지 않았다. 하지만 CUDA는 C언어를 기반으로 하기 때문에 C를 알고 있는 프로그래머라면 복잡한 그래픽에 대한 사전지식없이 쉽게 GPU를 이용한 일반연산을 구현할 수 있다. 이는 일반적인 수학 연산자를 데이터를 병렬로 처리할 수 있는 GPU 상에서 쉽고 효과적으로 구현할 수 있다는 뜻이다. 또한 기존에 쉐이딩 언어를 사용함으로써 발생할 수 있는 오버헤드가 제거됨으로써 더욱 효율적인 연산수행을 가능하게 한다.

CUDA도 GPU를 이용하는 언어이기 때문에 하드웨어의 구조적인 원인에서 오는 문제점을 가지고 있다. 즉 GPU와 GPU간에 메모리를 공유하여 사용할 수 없으며, GPU에서 연산을 하기 위해선 주기억장치로부터 데이터를 전송 받아 연산을 수행해야 한다. 데이터 전송에 따른 지연을 줄이고 프로그램의 효율을 높이기 위해선 CPU와 GPU간의 데이터 교환은 최소가 되도록 한다[9].

3.2 병렬화 전략

본 논문에서는 자유표면유동 해석시 로드가 가장 많이 걸리는 포아송 방정식 해석단계를 CUDA를 통해 병렬처리를 하였다. 포아송 방정식 해석은 자유표면유동 해석시 로드가 가장 많이 걸리는 부분이다. 이런 포아송 방정식은 이중 혹은 삼중의 다중루프형식으로 계산이 된다. 이런 다중루프 형태는 병렬성을 극대화할 수 있는 구조라 병렬계산이 유리하다. GPU는 nVIDIA사의 Tesla C2050를 사용 하였다. 먼저 C2050의 Hardware 구조를 알아보면 Fig. 2와 같이 14개의 Streaming Multiprocessors (SMs)가 있으며 각각의 SM에는 32개의 Scalar

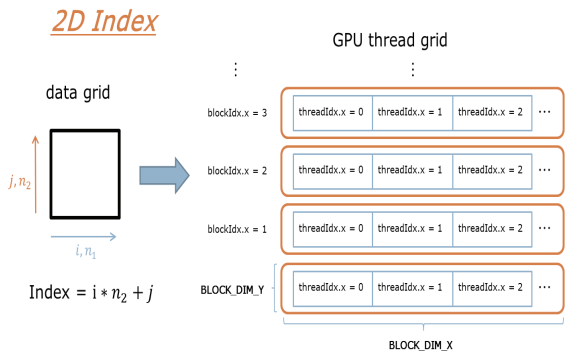


Fig. 3 2D Index in CUDA

Processors (SPs)가 포함된다. C2050을 사용해 수치해석을 하면 총 448개의 SP cores를 통한 병렬계산이 가능하다.

병렬계산을 위해서는 CUDA의 Kernel 함수 내 Index를 적합한 형태로 변환시켜야 한다. 이러한 특성을 고려하지 않고 CUDA프로그래밍을 수행할 경우 단일 코어 CPU를 사용할 경우보다 성능이 저하 될 수 있다. Kernel 내 Thread는 1개의 SP에서 작업을 수행하는데 Tread Index는 최대 3D Index로 표현할 수 있으나, TESLA C2050 GPU의 경우 최대 1024개까지만 표현 가능하므로 본 논문에서는 1D로만 제한해 사용하였다. Thread가 모인 형태를 Thread-Block 혹은 Block이라고 CUDA에서는 표현한다. Block의 경우 1D 또는 2D로 표현되는데, 이때 2D의 표현 가능한 최대 크기가 65535X655335이다. 해석시 필요한 격자수의 제한은 없을 것으로 판단하여 2D Index와 3D Index의 표현을 위해서 Block Index를 1D 혹은 2D로 사용하였다. 본 논문에서는 2D Case의 경우 Fig. 3과 같이 Thread를 x축으로 Block을 1D로 사용하여 y축으로 이용하는 형태로 CUDA내의 Index를 바꾸어 사용하였다.

3.3 병렬감소 연산

병렬 감소(Reduction) 연산은 병렬 연산 시스템에서도 부득이 하게 부분적으로 순차적으로 수행해야 하는 연산에 적용되는 방식이다. 병렬 감소 연산이 적용되는 연산은 여러 가지가 있지만 본 논문에서 제안하는 방식에서는 행렬의 원소중 최대값 (또는 최소값) 검색 연산에 사용된다. 이러한 병렬 감소 연산에 대한 연구는 이론적인 측면과 실제 CUDA 애플리케이션에서의 성능 최적화 등 실무적인 측면에서 오래전부터 활발하게 진행되어 왔다. Fig. 4은 병렬 감소 연산의 기본적인 구조를 설명하는 그림이다. 이 그림에서처럼 병렬 감소 연산은 상하가 뒤집어진 2진 트리 형태를 가진다. 각 사각형 노드는 연산의 대상이 되는 데이터 요소를 나타낸다. 예를 들어

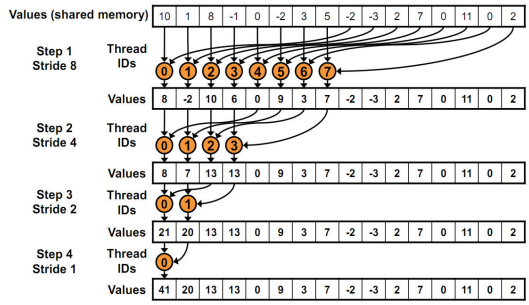


Fig. 4 Parallel Reduction Operation

N개의 실수의 합을 구하는 문제가 주어졌다고 할 경우, 가장 위쪽에 N 개의 실수가 입력되며 각각 2개씩 묶어 그 합을 다음 단계로 보낸다(Fig. 4에서 위에서 두 번째 줄)[9.10].

이렇게 $\log_2^N$단계를 거친 후 최종적으로 그 총합이 마지막 단계에 남게 된다. 앞서 언급한 대로 이 루틴은 병렬감소 뿐만 아니라, 최대/최소값 검색, 벡터 내적 연산에도 이용 가능하다. 일반적으로 OpenMP 등과 같이 성능이 높은 프로세서를 상대적으로 적은 개수를 적용하는 구조(coarse-grained parallel architecture)의 경우에는 제일 상위 단계에서 한 프로세서 내에 여러 개의 데이터 요소들을 처리하지만 CUDA와 같이 성능이 상대적으로 낮은 프로세스를 많이 사용하는 구조(dense-grained parallel architecture)에서는 프로세서 (또는 스레드) 하나 당 데이터 요소 하나를 할당하게 된다.

CUDA 기술 백서에서는 CUDA에서의 구현 시 메모리 액세스 패턴과 루프 풀기(loop unrolling) 기법을 적용함으로써 6단계에 걸친 최적화를 소개하고 있다. 그 결과로 가장 낮은 단계의 최적화 대비 가장 높은 단계의 최적화 성능이 약 20배 이상 증가함을 보이고 있다. 본 논문에서는 R-B Gauss Seidel 루틴 중에서 에러값의 최대요소를 찾는 루틴 ($P_{n+1} - P_n$의 최대값)에 병렬 감소 연산을 적용한다.

4. 결 과

4.1 2차원 댐붕괴 문제

개발된 코드의 검증을 위해서 대표적인 자유표면유동 문제인 댐붕괴 문제를 해석해 보았다. 초기설정은 Fig. 5와 같고 $w_l = 0.3$, $h_l = 0.6$, $w_t = h_t = 1.0$으로 설정하였다. 벽면에서의 속도와 압력의 경계조건으로 각각 no-slip조건과 Neumann조건을 사용하였다. 의 크기를 듬성하게(top, 32x32), 적당하게(meddle, 64x64), 조밀하게(bottom, 128x128) 나누어 해석을 수행하여 그 결과를 Fig. 6에 나타내었다.

격자의 수를 달리하여서 계산을 수행한 결과, 격자의 크기

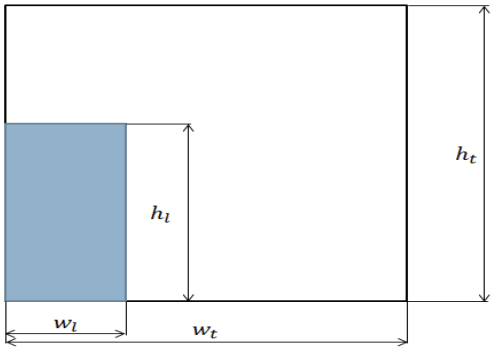


Fig. 5 Computational Domain

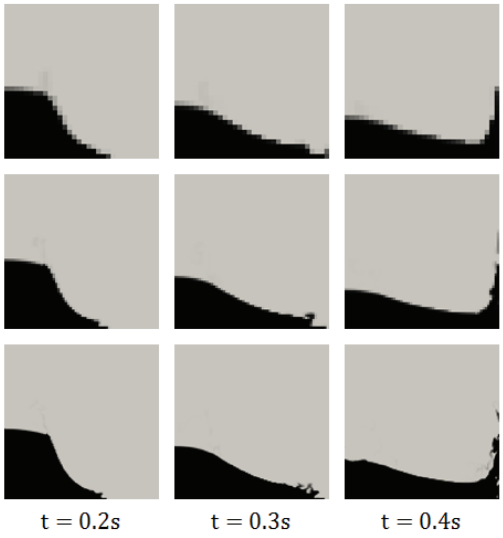


Fig. 6 Snapshots of the water-air interface for the dam-break problem. Top-32x32, Middle-64x64, Bottom–128x128

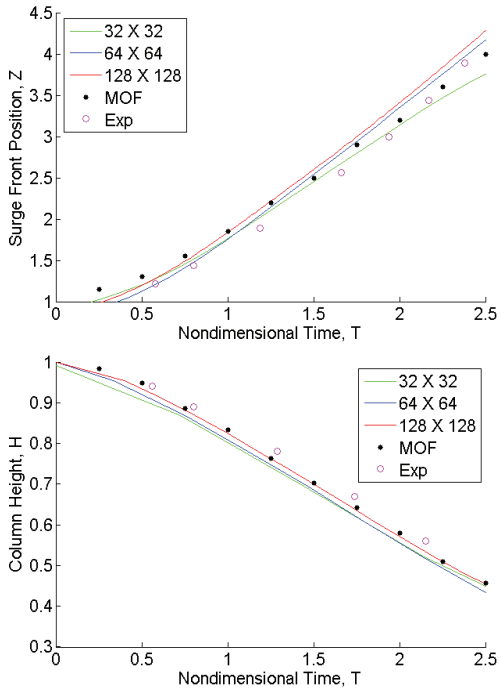
에 상관없이 일정한 경향을 보이는 것을 확인할 수 있었다. 또한 본 논문의 댐붕괴 문제 검증을 위해, Martin, J.C. et al.[11]의 실험값과 MOF법을 사용한 CFD계산결과와 비교해 보았다. 이 결과 비교에는 다음과 같은 무차원 값을 사용하였다.

$$\text{Surge Front Postion, } Z = \frac{z}{a} \tag{27}$$

$$\text{Column Height, } H = \frac{\eta}{an^2} \tag{28}$$

$$\text{Nondimensional time, } T = nt\sqrt{\frac{g}{a}} \tag{29}$$

여기서 a는 물기둥의 폭으로써 위에서 정의한 w_l과 같으며



Fig, 7 Surge front position(up) and column height(down) compassion with experimental and the other's computational data

무차원화의 기본이 되는 단위이고, n^2a는 물기둥의 높이, z는 붕괴하는 물기둥의 전진길이, η는 붕괴되는 물기둥의 높이, t는 시간, g는 중력을 뜻한다.

Fig. 7의 결과를 살펴보면 물기둥의 전진거리가 실험치보다 빠르게 진행됨을 알 수 있다. 또한 물기둥의 붕괴 또한 실험값과 비교시 붕괴가 이르게 되는 것을 확인 할 수 있다. 이런 점들은 실험값의 도메인 크기와 본 논문의 해석 도메인 크기의 차이에서 오는 레이놀즈수 차이를 감안한다면 비교적 잘 일치함을 알 수 있다.

4.2 GPU와 CPU계산의 성능비교

본 논문에서는 앞에서 계산을 수행해본 2차원 댐붕괴 문제를 CPU와 GPU를 이용해 해석하고, 서로의 해석시간을 비교해 보았다. CPU 연산은 Intel Xeon E5620@2.40GHz에서 수행하였으며, 1개의 코어만을 사용하였다. GPU연산은 위에서 언급한 CPU가 있는 컴퓨터에 nVIDIA Tesla C2050@1.15GHz를 장착하여 사용하였다. 식(30)를 이용하여 CPU에서의 해석시간과 GPU에서의 해석시간의 비율을 나타내 보았다.

$$Speed\text{-}Up = \frac{CPU\ time}{GPU\ time} \tag{30}$$

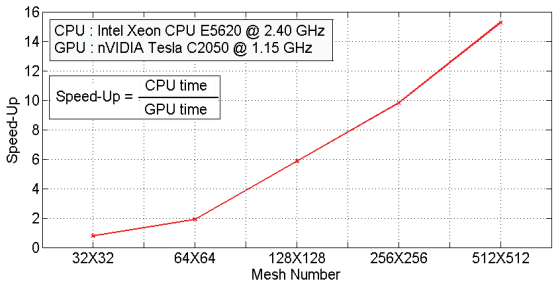


Fig. 8 Speed up as a function of mesh number (CPU,CUDA)

2차원 댐붕괴 문제의 해석시간 비교에 사용한 격자들은 Table 1에 나타내었다. Fig. 8에서 보여지는 바와 같이 격자수가 증가할수록 GPU연산 효율이 증가하는 것을 확인할 수 있다. CUDA를 이용하여 해석을 하였을 때 CPU만을 사용한 경우보다 약 2~15배정도 해석시간이 단축된 것을 알 수 있다.

V. 결 론

본 논문에서는 자유표면유동을 해석하는데 있어서 CUDA를 사용하여 GPU를 통해서 자유표면유동 문제를 계산하였다. 개발된 해석프로그램을 검증하기 위해서 댐붕괴 문제를 시뮬레이션 해보았다. 계산 결과는 검증된 실험과 잘 일치함을 보여 주었다. 또한 CPU와 GPU를 사용하여 연산을 수행 할 때의 해석시간에 대한 비교를 해 보았다. CUDA 프로그램 언어를 이용하여 GPU에서 자유표면유동 해석을 했을 때 CPU 1코어에서의 연산시간에 비해 2~15배 가까이 해석시간이 단축됨을 확인할 수 있었다. 이때 해석시간은 도메인의 격자수가 증가할수록 단축되었는데, 격자수가 증가할수록 GPU를 통한 병렬화의 효과가 증대된 것이라고 판단할 수 있었다.

본 연구에서는 CUDA를 이용하여 2차원 자유표면유동 문제를 효율적으로 해석하는 것에 초점을 두었다. 이번 연구에서는 댐붕괴 문제의 경우에 한해 그 효율성을 확인해 보았으며, 향후 다른 3차원 확장을 통하여 자유표면유동 문제해석에 GPU를 이용하여 그 효율성을 측정하고자 한다.

Table 1 Computation domain size and Speed-Up result of 2D Dam-break problem

	Mesh Size	Speed-Up
Case 1	32x32	0.78
Case 2	64x64	1.89
Case 3	128x128	5.85
Case 4	256x256	9.81
Case 5	512x512	15.27

후 기

이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(#2010-0004606).

참고문헌

[1] 1981, Hirt, C. W.. Nicholls, B. D.. "Volume of Fluid(VOF) Method for the Dynamics of Free Boundaries," *Journal of Computational Physics*, 39, pp.201-225.

[2] 1994, Sussman, M.. Smereka, P.. Osher, S.. "A Level Set Approach for Computing Solution to Incompressible Two-Phase Flows," *Journal of Computational Physics,* 114, pp.146-159.

[3] 1987, Takewaki, H.. Yabe, T.. "The Cubic-interpolated Pseudo Particle (CIP) Method Application to Nonlinear and Multi-dimensional Hyperbolic Equations," *Journal of Computational Physics*, 70, pp.355-372.

[4] 2009, Ahn, H.T.. Shashkov M.. Chiston M.A.. "The Moment-of-Fluid Method," *Communications in Numerical Methods in Engineering*, 25, pp.1009-1018.

[5] 2000, Yabe, T.. Tanaka, R.. Nakamura, T.. Xiao, F.. "An Exactly Conservative Semi-Lagrangian Scheme (CIP-CSL) in one dimension," *Monthly weather reiview*, 129, pp.332-344.

[6] 1985, Kim, J.. Moin, P.. "Applications of a Fractional-Sstep Method to Incompressible Navier-Stokes Equations," *Journal of Computational Physics*, 59, pp.308-323.

[7] 2005, Xiao, F.. Ikebata, A.. Hasegawa, T.. "Numerical Simulations of Free-interface Fluids by a Multi-integrated Moment Method,"*Computers & Structure*, 83, pp.409-423.

[8] 2011, Im, H.N.. "Interface-Tracking Simulation of Multi-phase Flow Using CIP-CSL2 Scheme," *Korean Society of Computational Fluids Engineering spring conference*, JEJU, Republic of Korea, 26-27 May 2011.

[9] http://www.nvidia.com/object/cuda_home_new.html.

[10] 2011, Park, T.J.. Woo, J.M.. Kim, C.H.. "CUDA-based Parallel Bi-Conjugate Gradient Matrix Solver for BioFET Simulation," .*Journal of IEEK*, 48-cl, pp.90-100.

[11] 1952, Martin, J.C.. Moyce, W.J.. "An Experimental Study of the Collapse of Liquid Columns on a Rigid Horizontal Plane," Philosophical Transactions of the Royal Society of London. Series A, *Mathematical and Physical Sciences*, Volume 244, Issue 882, pp.312-324.