MPI 기반의 병렬 성층·회전 난류 시뮬레이션

김 병 욱† · 양 성 봉††

요 약

본 논문에서는 MPI 기반이 성층·회전 난류 시뮬레이션을 위한 LES코드의 병렬화 기법에 대해 연구하며 그 결과를 실험한다. 본 논문에서는 병렬화 기법을 위해 순차 LES코드에 내재되어 있는 Tridiagonal solver의 제거를 통한 병렬화의 성능 향상과 포아선 방정식의 병렬화를 위한 영역 분할 방법을 소개한다. 또한 본 논문에서 연구되어진 병렬 LES 코드를 슈퍼컴퓨터에서 다양한 영역 분할에 대한 실험을 수행하며 그 결과에 대해 나타낸다. 실험 환경은 CRAY-T3E에서 수행하였으며, 다양한 영역 분할에 대해 프로세서의 개수를 변화시키며 수행속도와 그에 따른 속도의 향상을 측정하였다. 그 결과 단일 프로세서에서 순차 LES를 수행하는 것보다 병렬 LES코드에서 최고 16배에 해당되는 속도의 향상의 결과를 얻을 수 있었다.

Parallel Stratified and Rotating Turbulence Simulation based on MPI

Byung-Uck Kim† · Sung-Bong Yang††

ABSTRACT

We describe a parallel implementation for the large-eddy simulation(LES) of stratified and rotating turbulence based on MPI. The parallelization strategy is specified by eliminating the tridiagonal solver with explicit method and by domain decompositions for solving the poisson equation. In this simulation we have run on CRAY-T3E under the message passing platform MPI with a various domain decomposition and the scalability of this parallel code of LES are also presented. The result shows that we can gain up to 16 times faster speed up on 64 processors with xyz-directional domain decomposition and scalable up to 128×128×128 which processing time is almost similar to that of 40×40×40 on a single processor machine with a sequential code.

1. 연구 소개

1.1 연구개발의 필요성

난류(turbulence) 현상은 대기와 해양의 경계층, 구름, 오염 확산 등을 비롯하여 많은 대기와 해양 현상을 연구하기 위해서는 반드시 이해되어야 하는 중요한 유체역학 현상이다. 이 밖에도 난류는 기계공학에서 항공기의 날개 및 엔진, 핵융합로의 설계 등을 위해, 화학공학에서 최적의 혼합 상태를 얻기 위해, 토목공학에서 하천과 연안의 제어를 위해 중요한 역할을 하는 등 공학 여러 분야에서도 매우 중요한 연구 대상일 뿐만 아니라, 물리학, 천문학 분야에서도 프리즈마와 태양, 성간 물질 등의 운동을 이해하기 위한 매우 중요한 위치를 차지하고 있다.

Deardorff, Smagorinsky, Lilly 등의 일단의 기상학자들에 의해 큰 난류와동은 있는 그대로 시뮬레이션하고, 일반적인 특성을 가지는 작은 규모의 난류와동은 난류 이론을 바탕으로 모수화하는 LES(Large Eddy Si-

※ 이 논문은 1998년 한국과학기술평가원(KISTEP)의 지원에 의하여 이루어진 것임.
† 준 회 원 : 연세대학교 대학원 컴퓨터과학과
†† 정 회 원 : 연세대학교 컴퓨터과학과 교수
논문접수 : 1999년 8월 4일, 심사완료 : 1999년 11월 15일

mulation : 큰 에디모사)가 60, 70년대 걸쳐 제시되었으며, 80년대 들어서 이를 뒷받침할 수 있는 계산 능력의 급속한 발달과 함께 지구과학과 공학 분야에 걸친 난류 시뮬레이션에 있어서 획기적인 성과를 올리고 있다.

1.2 연구목표

본 연구에서는 회전과 성층의 영향을 받는 난류에 대한 LES를 개발하며 이를 통해 난류현상을 효과적이고 유용한 시간 안에 재현할 수 있으며, 난류 내에 부유입자가 있을 경우 부유입자의 운동에 대한 예측을 할 수 있게 해주는 LES의 병렬 최적화 연구이다. 특히 이러한 병렬 LES 코드의 개발은 수치시뮬레이션을 통하여 회전 및 성층 난류에서의 부유입자의 운동에 대한 유체역학적인 현상 원리를 규명할 수 있게 해주는 중요한 역할을 한다.

개발된 LES가 일정한 시간제약(time constraint) 안에 원하는 결과를 산출할 수 있도록 LES의 성능향상을 이루어내는 연구가 있다. 즉 개발된 LES를 순차적으로 수행하기에는 계산량이 너무 많고 또 복잡하기 때문에 원하는 정도의 정확성을 가지도록 하기 위해서 mesh수를 늘린다면 처리 속도가 더욱 느려져 원하는 결과의 얻어낼 수 없기 때문이다. 이처럼 빠른 연산처리를 요구하는 과학 계산에 연산처리 속도를 지속적으로 증가시키기 위한 방법을 이용하여 LES를 병렬화 시키는 것은 개발된 LES의 활용도를 증대시킬 수 있다.

병렬화는 현재까지 자연과학 현상을 규명할 수 있는 효과적인 컴퓨팅 기술의 하나로서 각광받고 있다. 슈퍼컴퓨터를 이용하여도 원하는 시간 안에 결과를 얻어낼 수 없는 Grand Challenge 문제들에 대해서는 병렬 초고속 컴퓨터를 사용하는 것이 바람직하다.

순차적으로 수행되도록 연구된 문제들을 병렬 초고속 컴퓨터에서 수행하기 위해서는 가용한 병렬성을 모두 찾아내어 최대한으로 병렬프로세서들을 이용할 수 있도록 병렬화 하는 연구가 가장 중요하다. 이러한 병렬화는 문제의 특성에 따라 그 방법이 달라지게 됨으로 일반적인 병렬화 기법보다는 대상 문제의 특성에 적합한 병렬화 연구 개발의 필요하다.

따라서 본 연구에서는 대상 LES에 대한 병렬화를 수행하면서 LES가 속한 유체역학의 특성을 고려한 병렬화 방법을 연구하며 그 결과를 실험한다.

또한, MPI(Message Passing Interface)와 같이 표준화된 메시지 전달 방법을 제공하는 라이브러리를 이용함으로서 효과적이고, 서로 다른 플랫폼간의 이식성을 높일 수 있는 병렬화에 대한 고려도 중요하다[4].

2. 연구 배경

본 논문에서 병렬화를 시도한 LES 프로그램의 복잡도(complexity)가 상당히 크고, CRAY-T3E에서 사용자에게 부여된 CPU time limit를 초과하는 문제들이 발생하였다. 또한 LES 프로그램의 Tridiagonal solver에서 병목현상이 발생됨을 발견하였다. 그리하여, Tridiagonal solver를 재구성하여 순차 LES 프로그램을 병렬화 시키는데 따른 문제점들을 파악하고 이를 바탕으로 LES 코드를 병렬화에 보다 적합하도록 재구성하였다. 재구성된 순차 LES 코드를 MPI를 이용하여 병렬화를 하였다. MPI를 이용하면 다양한 형태의 컴퓨터에서 병렬 프로그램을 수행시킬 수 있으며, CRAY-T3E 컴퓨터도 병렬 프로그래밍의 도구로 MPI를 제공하고 있다.

프로그램의 병렬화는 우선 영역 분할(domain decomposition)을 수행하고, 이 분할된 영역 분할을 바탕으로 연산을 분할하는 두 단계로 이루어진다. High Performance Fortran(HPF)과 같은 언어를 이용하면, 프로그래머가 사용된 데이터 배열에 대한 영역 분할 정보를 directive 형태로 기술하면 컴파일러가 자동적으로 병렬화된 코드를 생성한다. 하지만 MPI를 이용하는 경우는 프로그래머가 어떠한 방법으로 영역을 분할할 것인지를 결정하고, 결정된 영역 분할 방법에 따라 프로그램을 직접 작성하여야 한다.

영역 분할 방법이 결정되면 프로그래머는 각 프로세서가 다룰 수 있는 데이터 영역을 고려하여 연산이 수행되어야 하는 범위를 결정하여야 한다. 특히, 경계조건이 어느 프로세서에 속하는지 정확히 파악하고, 각 프로세서의 ghost cell에 주어져야 하는 값들을 정확히 파악하여 루프 연산의 범위를 결정하여야 한다. 실질적인 병렬 코드에는 루프 연산시 참조되는 각 프로세서의 ghost cell을 채우는 통신 부분이 필요하다. 그러므로, 일반적으로 루프 연산을 수행하기 전에 각 프로세서간의 통신이 이루어진다. 이 통신 부분은 순차코드의 각 루프 및 데이터 통신에 필요한 부분을 MPI를 이용하여 직접 삽입하는 병렬화 프로그래밍을 수행하였다.

3. 연구 수행 모델

다음은 본 논문에서 개발 연구되어진 성층·회전 난류 시뮬레이션의 기반이 되는 수학적 모델이다.

3.1 모형

모형에서는 3차원의 비정수, Boussinesq 근사를 적용한 방정식을 사용하였다.

$$\frac{\partial \tilde{u}_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial \tilde{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\tilde{u}_i \tilde{u}_j) = -\frac{1}{\rho}\frac{\partial \tilde{p}}{\partial x_i} + \nu\frac{\partial^2 \tilde{u}_i}{\partial x_j \partial x_j} + \tilde{b}\delta_{i3} \tag{2}$$

$$\frac{\partial \tilde{b}}{\partial t} + \frac{\partial}{\partial x_j}(\tilde{u}_i \tilde{b}) = \kappa\frac{\partial^2 \tilde{b}}{\partial x_j \partial x_j} \tag{3}$$

여기서 $(\tilde{u}_1, \tilde{u}_2, \tilde{u}_3)$ 또는 $(\tilde{u}, \tilde{v}, \tilde{w})$는 (x_1, x_2, x_3) 또는 (x, y, z)방향의 속도 성분이고, $\tilde{p}$는 압력, ρ는 참고 밀도, $\tilde{b}(=-g\Delta\rho/\rho)$는 부력, ν는 점성계수(kinematic viscosity coefficient), κ는 확산계수(diffusion coefficient)이다. x와 z축은 각각 수평과 연직 방향을 나타내고, y 축은 평균적으로 균일한 세 번째 방향을 나타낸다.

LES를 위해서는 실제 유동장에 대해 filtering을 하여야 한다. 부피-평균 box filter는 다음과 같이 정의된다.

$$u_i(x,t) = \frac{1}{\Delta x_1 \Delta x_2 \Delta x_3} \int_{x_1-\Delta x_1/2}^{x_1+\Delta x_1/2}\int_{x_2-\Delta x_2/2}^{x_2+\Delta x_2/2}\int_{x_3-\Delta x_3/2}^{x_3+\Delta x_3/2} \tilde{u}_i(x', t)dx' \tag{4}$$

filtering을 가하면, 이류 항은 다음과 같이 다시 쓰여질 수 있다.

$$\frac{\partial}{\partial x_j}(\overline{\tilde{u}_i \tilde{u}_j}) = \frac{\partial}{\partial x_j}(u_i u_j + L_{ij} + C_{ij} + R_{ij}) \tag{5}$$

여기서

$$L_{ij} = \overline{u_i u_j} - u_i u_j \tag{6}$$

$$C_{ij} = \overline{u_i u'_j} + \overline{u'_i u_j} \tag{7}$$

$$R_{ij} = \overline{u'_i u'_j} \tag{8}$$

는 각각 Leonard stress, cross term 그리고 subgrid-scale Reynolds stress를 나타낸다. 2차 정밀도(second-order accuracy)의 유한 차분 방안이 사용될 때, Leonard stress는 truncation error와 비슷한 크기로 무시할 수 있다[6]. 다른 항들은 모수화시켰다.

그러면, LES에 대한 지배 방정식은 다음과 같이 나타난다.

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{9}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial P}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} + \nu\frac{\partial^2 u_i}{\partial x_j \partial x_j} + b\delta_{i3} \tag{10}$$

$$\frac{\partial b}{\partial t} + \frac{\partial}{\partial x_j}(u_i b) = (\kappa_T + \kappa)\frac{\partial^2 b}{\partial x_j \partial x_j} \tag{11}$$

여기서 식 (10)의 아격자전단 (τ_{ij})은 다음과 같이 정의된다.

$$\tau_{ij} = Q_{ij} - \frac{1}{3}Q_{kk}\delta_{ij} \tag{12}$$

$$P = \frac{p}{\rho} + \frac{1}{3}Q_{kk} \tag{13}$$

$$Q_{ij} = R_{ij} + C_{ij} \tag{14}$$

아격자모수화를 위해서는 Smagorinsky의 방법이 사용되었다[5]. 즉

$$\tau_{ij} = -2\nu_T S_{ij} \tag{15}$$

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{16}$$

$$\nu_T = \ell_0^2\sqrt{2S_{ij}S_{ij}} \tag{17}$$

부력 방정식의 경우에 대해서도 $\kappa_T = \nu_T$로 가정하여 유사한 아격자모수화를 시켰다.

식 (17)에서의 길이규모 l은 다음과 같이 주어진다.

$$\ell_0 = C_s(\Delta x \Delta y \Delta z)^{1/3} \tag{18}$$

filter의 특성 규모를 결정하는 계수 C_s는 Cs=0.2가 쓰였다[2].

그러나, 실제 해양과 같이 성층화된 유체의 경우 에디의 운동은 성층화에 의해서 크게 영향을 받으므로 이에 대한 아격자모수화를 수행하는 것이 필요하다. 따라서 Mason의 경우와 유사하게 다음과 같이 성층화의

영향을 모수화하였다[1]. 즉,

$$For\ Ri < 0,\ \nu_T = \ell^2\sqrt{2S_{ij}S_{ij}} \tag{19}$$

$$For\ A^{-1} > Ri > 0, \tag{20}$$

$$\nu_T = \ell_0^2\sqrt{2S_{ij}S_{ij}}\,(1 - ARi)^{1/2}$$

$$For\ Ri > A^{-1},\ \nu_T = 0 \tag{21}$$

여기서 Richardson수 Ri는 다음과 같이 정의된다.

$$Ri = \frac{\partial b/\partial z}{2S_{ij}S_{ij}} \tag{22}$$

한편 성층화에 의해 보정된 길이규모 l은 다음과 같이 주어진다.

$$\ell = \min[\ell_0, k(z + z_0)\phi]0 \tag{23}$$

여기서

$$\phi = (1 - 4\gamma Ri)^{1/4} \tag{24}$$

를 의미하고, γ = 3.0, z_0 = 0.0으로 주어지고, k(= 0.4)는 von Karman 상수이다.

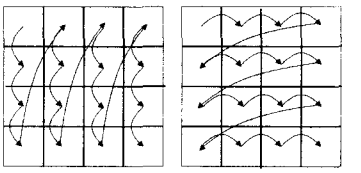
4. 병렬화 방법

본 논문은 효율적이고 수치적으로 정확하며 다양한 기종의 슈퍼컴퓨터에서도 실행될 수 있도록 호환성을 가지는 병렬 LES 코드의 연구 및 실험을 목표로 한다[3]. 또한 성층·회전 난류 시뮬레이션을 위해 순차 LES 코드를 병렬화 할 때의 문제점을 파악하여 병렬 최적화를 위한 방법을 연구하며 실험한다.

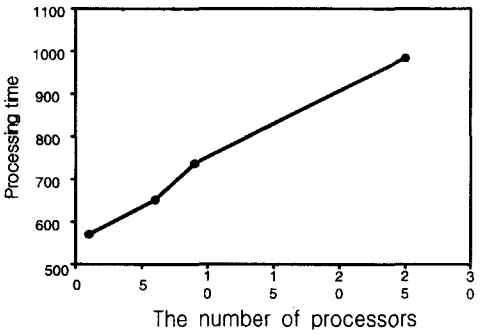
4.1 Tridiagonal Solver

효과적이고 최적화된 병렬 LES코드를 개발하기 위해서 먼저 1차원의 순차코드를 분석하여 병렬화에 장애가 되는 부분을 찾아내야 한다. 그래서 먼저 LES 코드의 주요 부분인 Tridiagonal solver 부분을 HPF(High Performance FORTRAN)와 MPI를 이용하여 병렬화하였다. 하지만 (그림 1(a))에서 보는 바와 같이 x축과 y축으로의 데이타 의존도(data dependency)가 존재하여 병렬화의 효과를 크게 기대 할 수 없다.

(그림 1(b))는 각 프로세서의 수에 따른 병렬화된 Tridiagonal solver의 수행 시간을 나타낸다. 그 결과를 살펴보면 프로세서의 수가 증가될수록 오히려 전체적인 수행 시간이 늘어남을 알 수 있다. 이것은 Tridiagonal solver가 가지는 데이타 의존도에 의해 오히려 병렬화에 사용되어진 프로세서 수의 증가와 더불어 프로세서간의 통신 부담의 증가에 따른 전체적인 수행시간이 증가되는 것이다.



(a) x축과 y축 방향의 데이타 의존도



(b) 병렬 Tridiagonal Solver의 수행속도

(그림 1) 데이타 의존도와 병렬 Tridiagonal Solver의 수행 결과

Tridiagonal solver는 식 (25)와 같은 방정식을 풀기 위하여 사용되어지며, 순차 LES 코드의 많은 부분을 차지한다.

$$\frac{\partial U}{\partial t} = \frac{\partial U}{\partial x} \tag{25}$$

이 식은 시간에 대한 속도의 순간 변화와 공간에 대한 속도의 순간 변화 값이다. 이식을 풀기 위하여 다음과 같이 implicit 방법과 explicit 방법을 사용할 수 있으며, 각각은 식 (26)과 식 (27)에 나타나 있다.

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = \frac{U_{i+1}^{n+1} - U_i^{n+1}}{\Delta x} \tag{26}$$

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} = \frac{U_{i+1}^{n} - U_i^{n}}{\Delta x} \tag{27}$$

여기에서, U는 속도, i는 공간 도메인, n은 시간 도메인, Δt는 시간 변화량, Δx는 공간 변화량을 나타낸다.

식 (25)를 풀기 위하여 식 (26)과 같은 implicit방법을 사용하게 되면, Tridiagonal solver에 의해 데이타 의존도가 증가하여 병렬화의 효과를 얻을 수 없지만, 식 (27)과 같은 방법으로 Tridiagonal solver를 제거하면, 데이타 의존도를 줄일 수 있어 병렬화의 큰 효과를 얻을 수 있게 된다.

즉, LES코드의 병렬화를 위하여 순차 프로그램이 1차원 배열을 3차원 배열로 바꾸고, 경계조건들을 바꾸는 방법 정도의 코드 재구성으로는 병렬화의 큰 이득을 얻을 수 없다. Tridiagonal solver와 같이 데이타 의존도가 높은 부분, 즉 병렬화에 방해가 되는 병목현상을 제거해야만 한다.

4.2 영역 분할(Domain Decomposition)

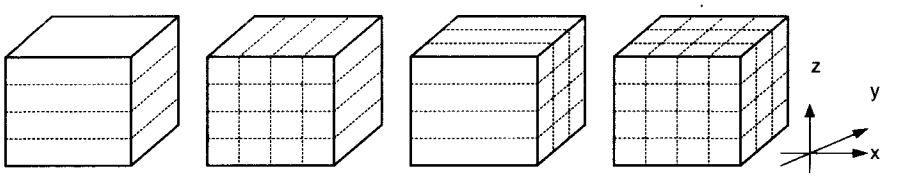
순차 LES코드에서 전체 수행시간의 대부분을 차지하는 연산은 포아선 방정식에서 GAUSEI 함수 값을 연산하는 부분이다. 즉, 이 부분을 병렬 연산 처리하여 병렬화에 따른 전체적인 수행 속도의 개선이 본 논문에서의 성능향상을 위한 중요한 부분이다.

위와 같은 프로그램을 병렬화 하려면 영역 분할을 수행하고 이 분할된 영역을 바탕으로 연산을 분할하는 단계를 거쳐야 한다. HPF와 같은 언어를 이용하면, 프로그래머가 사용된 데이타 배열에 대한 영역 분할 정보를 지시적(directive) 형태로 기술하면 컴파일러가 자동으로 병렬화 된 코드를 생성한다. 하지만 MPI를 이용하는 경우는 프로그래머가 어떠한 방법으로 영역을 분할할 것인지를 결정하고, 결정된 영역분할 방법에 따라 직접 병렬프로그램을 기술하여야 한다.

영역분할은 다양하게 수행할 수 있다. 포트란의 경우 배열의 열(column)을 위주로 메모리 주소를 생성하는 특징을 가지고 있다. 즉 3차원 배열 A(i, j, k)와 A(i+1, j, k)는 바로 인접한 메모리 상에 위치하게 된다. 그러므로 가능한 배열의 바깥쪽 차원부터 분할하는 것이 바람직하다. 이러한 특징을 감안하여 본 논문에서는 LES에서 사용되는 3차원 배열들을 z방향으로 일차원 영역분할을 수행하였다. 또한 효과적인 영역분할 방법을 얻기 위하여 y와 z방향, x와 z방향의 이차원 영역분할을 수행하였으며, 이때 프로세서도 이차원 위상(topology)으로 구성하였다. 가장 일반적인 것은 모든 방향으로 영역분할을 수행하는 것이다. 그러므로, x,y,z 모든 방향으로 영역분할을 수행하였고 또한 이때 프로세서도 분할방법에 맞추어 3차원 위상으로 구성하였다. (그림 3)은 각각의 영역분할 방법을 묘사한다.

(그림 2(a))는 1차원 프로세서 위상을 보여주고 있다. (그림 2(b))와 (그림 2(c))의 경우는 동일한 2차원 프로세서 그리드(grid)를 가지지만 배열을 분할하는 방향에 의하여 각 프로세서에 속하는 데이타의 내용이 달라지게 된다. 또한 (그림 2(d))는 각 방향 즉 x, y 그리고 z방향으로 데이타를 분할하여 연산을 수행하게 된다.

이렇게 분할 영역이 결정되면, 프로그래머는 각 프로세서가 다룰 수 있는 데이타 영역을 고려하여 연산이 수행되어야 하는 범위를 결정하여야 한다. 특히, 경계조건이 어느 프로세서에 속하는지 정확히 파악하고, 각 프로세서의 ghost cell에 주어져야 하는 값들을 정확히 파악하여 루프 연산의 범위를 결정해야 한다. 실질적인 병렬 코드에는 루프 연산시 참조되는 각 프로세서의 ghost cell을 채우는 통신 부분이 필요하다. 그러므로, 일반적으로 루프 연산을 수행하기 전에 각 프로세서간의 통신이 이루어진다. 이 통신 부분은 순차 코드의 각 루프 및 데이타 통신이 필요한 부분에 MPI



(a) 1차원 영역분할(z-방향) (b) 2차원 영역분할(xz-방향)
(c) 2차원 영역분할(yz-방향) (d) 3차원 영역분할(xyz-방향)

(그림 2) 영역 분할

를 이용하여 직접 삽입되었다. 이러한 방법으로 다음 (그림 4)와 같이 LES코드에서 가장 많은 수행시간을 소비하는 포아선 방정식의 GAUSEI함수를 병렬화 하여 재구성한 pseudo 코드의 한 예제이다. 즉, 순차 LES코드의 GAUSEI함수를 병렬 LES코드에서는 각각의 프로세서가 영역 분할 방법에 따라 할당되어진 부분을 계산하고, 경계 조건들을 서로 교환함으로서 포아선 방정식을 병렬로 수행하게 된다. 따라서, 영역 분할을 통한 계산량의 분산은 큰 병렬효과를 가져오게 된다.

```
Subroutine GAUSEI()
   While (.NOT.Converge)
      Calculate Phi(i, j, k)
      Calculate Boundary Condition.
      Check Convergence
   End While
End Subroutine
```

(a) 순차 코드(Sequential Code)

```
Subroutine GAUSEI()
   While (.NOT.Converge)
      Calculate Phi(i, j, k)
      Calculate Boundary Condition.
      Excahnge Boundarys
      Check Convergence
   End While
End Subroutine
```

(b) 병렬 코드(Parallel Code)

(그림 3) GAUSEI 함수의 Pseudo 병렬 코드

5. 실험 결과

본 논문에서는 4장에서 언급한 것과 같이 다양한 영역분할 방법에 따라 병렬 코드를 작성하였다. 먼저 문제의 크기를 40×40×40으로 고정하고 각 영역 분할 방법에 따라 프로세서의 수를 변화시켜가면서 수행시간과 그에 따른 전체 수행 속도의 향상이 어떻게 이루어지는가를 실험하였다. 또한 프로그램의 확장성을 살펴보기 위해 프로세서 위상을 고정시키고 데이타 크기를 변화하며 그 성능을 실험한 결과도 나타나 있다.

본 논문에서 제시한 병렬 LES코드의 실험 결과 값은 CRAY-T3E컴퓨터에서 MPI를 이용하여 Fotran90으로 프로그래밍 하였다.

5.1 영역 분할 방법에 따른 실험 결과

<표 1>은 각각의 영역 분할 방법에 따라 프로세서의 개수를 증가시키면서 해당 프로세서의 위상에 따른 수행시간과 그에 따른 속도 향상을 실험결과를 나타낸다. 여기에서 속도 향상은 (수행시간/순차 LES 코드 수행시간(1430 sec))의 비율이며, 소수점 셋째 자리에서 반올림한 값이다. 또한 <표 1>의 실험 결과에서 굵은 수치 값은 해당 프로세서에서 가장 성능이 좋은 경우를 나타낸다.

<표 1>의 실험결과를 분석해 보면 첫 번째, z 방향으로 영역을 분할하였을 경우, 즉 프로세서 위상을 1차원으로 구성했을 경우, 각각 2, 4, 8개의 프로세서를 사용하여 실험을 수행하였다. <표 1>에서 보는 것과 같이 프로세서의 수가 증가함에 수행시간의 단축과 속

〈표 1〉 실험 결과

		z축 영역분할		xz축 영역분할		yz축 영역분할		xyz축 영역분할	
프로세서 수	프로세서위상	수행시간	속도향상	수행시간	속도향상	수행시간	속도향상	수행시간	속도향상
2	2	**816**	**1.75**	-	-	-	-	-	-
4	4	496	2.88	-	-	-	-	-	-
	2×2	-	-	494	2.89	**308**	**4.64**	-	-
8	8	456	3.13	-	-	-	-	-	-
	2×4	-	-	328	4.36	211	6.78	-	-
	4×2	-	-	427	3.35	**165**	**8.67**	-	-
	2×2×2	-	-	-	-	-	-	240	5.96
16	4×4	-	-	258	5.54	99	14.44	-	-
	2×2×4	-	-	-	-	-	-	120	11.91
	2×4×2	-	-	-	-	-	-	**96**	**14.90**
	4×2×4	-	-	-	-	-	-	166	8.61
64	8×8	-	-	242	5.91	92	15.54	-	-
	4×4×4	-	-	-	-	-	-	**61**	**16**

도의 향상이 이루어진다. 하지만 4개 이상의 프로세서가 쓰이는 경우에는 그 성능 향상의 폭이 급격히 감소하는 것을 알 수 있다. 프로세서 4개를 사용했을 경우와 프로세서 8개를 사용했을 경우 그 성능의 차는 거의 없음을 알 수 있다.

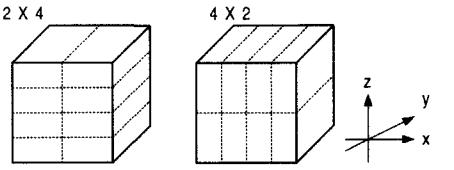
프로세서의 수가 어느 정도의 수준을 넘게 되면 전체적인 성능 향상 정도가 줄어드는 이유는 프로세서의 수가 많아질수록 각각의 프로세서간 통신비용이 많이 들기 때문이다. 예를 들어 프로세서가 2개인 경우 각 프로세서는 40×40×20의 컴퓨팅 영역과 40×40의 통신 영역을 가지게 된다. 일반적으로 (컴퓨팅영역/통신영역)의 값이 클수록 통신비용이 적게든다. 따라서 프로세서가 두 개인 경우 (컴퓨팅영역/통신영역)의 값이 20이지만, 프로세서가 4개인 경우는 컴퓨팅 영역이 40×40×10이 되므로 10으로 통신비용이 증가하게 된다. 즉 프로세서의 수가 증가하게 되면 연산을 분할하여 개별적으로 빠른 수행속도를 보장하지만 전체적으로는 각각의 프로세서간의 통신비용 부담이 증가하게 되므로 프로세서의 수와 그에 따른 통신비용의 증가사이에는 적당한 합의선(Trade-off)이 존재하여 전체적인 성능 향상에 영향을 끼친다.

두 번째, xz 영역 분할과 yz방향 영역 분할의 2차원 프로세서 구성에 따른 결과를 살펴보자. 먼저, <표 1>의 결과에서 나타나듯이 전체적인 성능은 yz방향으로 영역 분할을 시도한 경우에 그 성능이 훨씬 뛰어남을 알 수 있다. 그 이유는 x축으로의 데이타 지역성(locality)이 존재하기 때문에 이러한 데이타 지역성을 잘 이용할 수 있는 영역 분할인 yx축 영역 분할이 그 성능이 훨씬 뛰어남을 알 수 있다.

그리고, 프로세서 8개인 경우 각각 프로세서 위상을 2×4와 4×2의 두 가지 방식으로 구성할 수 있다. 두 경우를 살펴보면, x축의 데이타 지역성을 잘 활용할 수 있는 영역 분할과 프로세서 위상의 경우가 성능이 우수함을 알 수 있다.

또한 (그림 4)와 <표 1>에서 알 수 있듯이 y축 방향의 데이타 지역성이 존재함을 확인 할 수 있다. xz축 분할일 경우 프로세서 위상을 4×2로 구성하였을 때, y축 방면으로의 데이타 지역성에 영향을 받아 그 성능이 우수함을 알 수 있다.

마지막으로 영역 분할의 가장 일반적인 경우인 x축, y축 그리고 z축의 3방향에 대한 영역 분할을 수행하고 각각의 프로세서 개수에 따른 그 실험 결과를 살펴보자. 프로세서가 8개인 경우엔 yz방면 영역 분할(4×2)의 경우보다는 낮은 속도 향상을 보이지만, 나머지 경우에서는 더 우수한 성능 향상이 있음을 알 수 있다.



(그림 4) 프로세서 8개를 사용한 xz방면 영역 분할

전체적인 실험 결과에서 나타나듯이 데이타의 규모가 작으므로 8~16개의 프로세서에서 병렬화의 효과를 잘 나타남을 알 수 있으며, 프로세서가 증가함에 따라 통신비용도 증가하여 전체적으로 병렬화의 효과가 감소하는 현상을 볼 수 있었다. 즉 프로세서의 개수가 많아짐으로서 하나의 프로세서가 담당하게 되는 영역의 크기는 줄어들어 프로세서 하나 당 처리하는 수행시간은 줄어들지만 각각의 프로세서가 서로 통신하는 비용이 그 만큼 증가하여 전체적으로는 병렬화의 효과를 기대하기가 어렵게 됨을 알 수 있다.

5.2 데이타 크기에 따른 실험 결과

프로그램의 확장성을 평가하기 위하여 프로세서를 64개를 사용하여 xyz방향으로 영역 분할을 수행하여 그 위상을 4×4×4로 고정하고 데이타의 크기를 변화시키면서 프로그램의 수행 성능을 측정하였다.

<표 2>는 데이타의 크기를 작은 사이즈에서부터 시작하여 충분히 큰 사이즈까지 변화시키면서 수행한 프로그램의 수행시간을 나타낸다. 그 결과를 살펴보면 데이타의 크기가 128×128×128인 경우 하나의 프로세서가 수행하는 데이터의 크기가 32×32×32로 40×40×40의 데이타 크기를 단일 프로세서에서 수행했을 때 걸리는 시간인 1430(sec)와 비슷한 1510(sec)만에 수행할 수 있음을 나타낸다. 즉, 기존의 순차 LES 코드보다 약 3×3×3배 더 커진 영역에 대해 거의 비슷한 시간 안에 병렬 LES코드를 수행할 수 있는 성능 향상이 있음을 알 수 있다.

이러한 결과는 단일 프로세서에서 작은 데이타 크기를 가지는 문제를 수행할 수 있는 시간에 병렬 프로그램을 이용하면 훨씬 더 큰 데이타 크기를 가지는 문제를 같은 시간 안에 해결 할 수 있음을 보여준다.

〈표 2〉 데이타 크기 변화에 따른 수행시간

데이타 크기	수행시간 (sec)
32 × 32 ×32	39
64 × 64 ×64	185
128 × 128 × 128	1510

6. 결 론

본 논문에서는 성층·회전 난류 시뮬레이션을 위한 MPI기반의 병렬 LES 코드 연구 개발을 위한 방법과 그 실험 결과에 대해 분석하였다.

순차 LES코드는 단일 프로세서에서 수행되므로 문제의 데이타 영역의 크기가 어느 정도 커지면 유용한 시간 안에 그 해답을 내놓을 수 없게 된다. 따라서 순차 LES코드를 병렬 LES코드로 재구성하게 되면, 데이타 크기를 더욱 더 크게 잡을 수 있으므로 시뮬레이션의 영역을 좀 더 넓힐 수 있고 또한 빠른 시간 안에 그 결과를 얻을 수 있게 된다. 본 논문에서도 이러한 결과를 얻을 수 있음을 나타낸다.

참 고 문 헌

[1] Mason, P. J., "Large eddy simulation of the convective boundary layer," J. Atmos. Sci. 46, pp. 1492-1516, 1988.

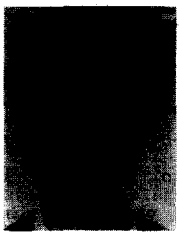
[2] Mason, P. J., "Large eddy simulation : A critical review of the technique," Quart. J. Roy Meteor. Soc., 120, pp.1-26, 1994.

[3] Martin, S. "Parallel Turbulence Simulation based on MPI," The proceedings of the HPCN'96 in Brussels, 1996.

[4] Message Passing Interface Forum. MPI : A Message-Passing Interface Standard, June, 1995.

[5] Smagorinsky, J., "General circulation experiments with the primitive equations," I. the basic experiment. Mon. Wea. Rev., 91, pp.99-164, 1963.

[6] Shaanan, S., J. Feziger and W. Reynolds, "Numerical Simulation of Turbulence in the Presence of Shear.," Thermoscience Division, Dept Mechanical Engineering, Stanford Univ., Report TF-6, Stanford, CA, 1975.



김 병 욱

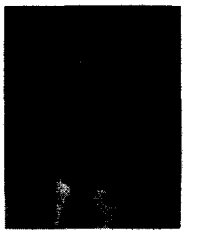
e-mail : kimbu@mythos.yonsei.ac.kr

1996년 연세대학교 컴퓨터과학과 (학사)

1998년 연세대학교 대학원 컴퓨터과학과(석사)

1998년 9월~현재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 병렬처리, 3D 컴퓨터 그래픽 랜더링 알고리즘



양 성 봉

e-mail : yang@mythos.yonsei.ac.kr

1981년 연세대학교 공과 대학(학사)

1984년 University of Oklahoma 컴퓨터과학과(석사)

1992년 University of Oklahoma 컴퓨터과학(박사)

1993년~1994년 전주대학교 전자계산학과 전임강사

1994년~현재 연세대학교 공과대학 컴퓨터과학과 부교수

관심분야 : 병렬처리, Computational geometry, Spatial data processing