대한기계학회논문집 B권, 제 27 권 제 6 호, pp. 753~765, 2003

이산화된 Navier-Stokes 방정식의 영역분할법을 위한 병렬 예조건화

강성우† · 최형권* · 유정열**

(2002 년 10 월 29 일 접수, 2003 년 3 월 27 일 심사완료)

Parallel Preconditioner for the Domain Decomposition Method of the Discretized Navier-Stokes Equation

Sungwoo Kang, Hyounggwon Choi and Jung Yul Yoo

Key Words : Domain Decomposition Method(영역분할법), Parallel Preconditioner(병렬 예조건인자), Navier-Stokes Equation(나비에 스톡스 방정식)

Abstract

A finite element code for the numerical solution of the Navier-Stokes equation is parallelized by vertex-oriented domain decomposition. To accelerate the convergence of iterative solvers like conjugate gradient method, parallel block ILU, iterative block ILU, and distributed ILU methods are tested as parallel preconditioners. The effectiveness of the algorithms has been investigated when P1P1 finite element discretization is used for the parallel solution of the Navier-Stokes equation. Two-dimensional and three-dimensional Laplace equations are calculated to estimate the speedup of the preconditioners. Calculation domain is partitioned by one- and multi-dimensional partitioning methods in structured grid and by METIS library in unstructured grid. For the domain-decomposed parallel computation of the Navier-Stokes equation, we have solved three-dimensional lid-driven cavity and natural convection problems in a cube as benchmark problems using a parallelized fractional 4-step finite element method. The speedup for each parallel preconditioning method is to be compared using upto 64 processors.

기호설명

C_p : 정압비열
g : 중력가속도
k : 열전도율
L : 특성 길이
Pr : 프란틀 수 ($\mathrm{Pr} = \nu/\alpha$)
Ra : 레일리 수
($Ra = g\beta C_p \rho^2 L^3 (T_H - T_C)/\mu k$)
Re : 레이놀즈 수 ($\mathrm{Re} = \rho UL/\mu$)
T_C : 냉각된 벽면 온도
T_H : 가열된 벽면 온도
U : 특성 속도
α : 열확산율
β : 열팽창율
ρ : 밀도
μ : 점성계수
ν : 동점성계수

† 책임저자, 회원, 서울대학교 대학원 기계항공공학부
E-mail : hyperon2@snu.ac.kr
TEL : (02)880-1654, FAX : (02)883-0179
* 회원, 서울산업대학교 기계공학과
** 회원, 서울대학교 기계항공공학부

1. 서 론

대형 병렬 컴퓨터의 빠른 개발 속도에 따라 대규모의 공학 문제를 병렬 컴퓨터를 이용하여 해석하고자 하는 노력이 꾸준하게 진행되고 있다. 병렬 컴퓨터는 각각의 기억장소(RAM)와 중앙처리장치(CPU)를 수십 개 혹은 수백 개씩 연결하여 연산을 수행하므로 큰 기억 용량과 많은 연산을 필요로 하는 대용량 문제를 해석하는데 효과적으

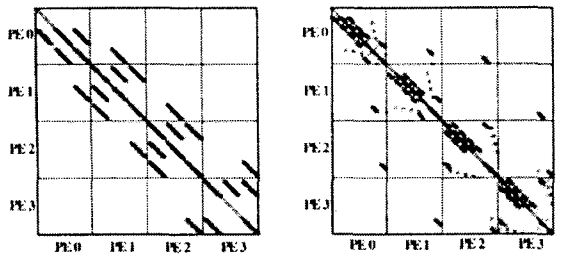


Fig. 1 Non-zero pattern of (a) 1-dimensional and (b) 2-dimensional domain-decomposed matrix obtained by the finite element discretization for the grids shown in Fig. 3(a) and 3(b).

로 쓰일 수 있다. 하지만 N 개의 중앙처리장치를 가지는 병렬 컴퓨터를 사용하여 해석을 수행할 때 N 배만큼 빠른 시간 안에 결과를 얻기는 대단히 어렵다. 이는 잘 알려진 Amdahl 의 법칙에 따라, 프로세서 수가 증가함에 따라 얻을 수 있는 속도향상(speedup)이 한계점에 도달하기 때문이다. 사용하고자 하는 알고리즘에 병렬화 되지 않는 부분이 존재하게 되면 속도향상은 프로세서 수에 따라 선형적으로 증가하지 않고 포화 상태에 이르게 되기 때문이다. 따라서 행렬해법을 비롯한 기존의 공학 문제 해석을 위한 알고리즘들을 병렬 컴퓨터 상에서 구현할 때 선형적인 속도향상을 보장하는 병렬 알고리즘들을 개발하는 작업은 매우 중요하다. 이는 병렬 컴퓨터공학에서 '범위성(scalability)'이라고 칭한다. 본 논문에서는 영역분할법을 이용하여 전산유체역학 문제 해석을 위한 범위성을 가지는 병렬 알고리즘을 개발하고 검증한다.

타원형 편미분방정식을 포함한 지배방정식들을 병렬 컴퓨터를 이용하여 해석하는 방법의 하나로 영역분할법은 분산메모리를 가지는 병렬 컴퓨터에서 매우 효과적으로 사용되어 왔다. 이 방법은 우선 풀고자 하는 전체 계산 영역을 통상 프로세서의 수로 나눈 후, 각각의 프로세서가 분할된 영역(subdomain)을 계산하는 방법이다. 각각의 분할된 영역은 국소행렬(local matrix)을 생성하며, 각 국소행렬은 Fig. 1 에서처럼 이웃하는 영역의 국소행렬과 내재적으로 연성되어 있다. 따라서 영역분할된 전체 영역의 해를 각각의 프로세서에서 생성된 국소행렬들을 이용하여 구하려면 각 영역(processor) 간의 통신이 필요하게 된다. 이는 통신 부하(communication overhead)라고 불린다. 또한, 국소행렬로부터 전체행렬의 해를 공액구배법(conjugate gradient, 이하 CG)과 같은 반복해법을 이용하여 구하면 영역을 분할할수록 반복해법의 반복계산 횟수가 증가한다. 반복계산 횟수가 증가하는 것은 통신 부하와 더불어 속도저하의 주요한 요인이 된다. 따라서 많은 연구자들은 통신 부하와 반복계산 횟수의 증가를 최소화하는 알고리즘을 개발하고자 하였다. 병렬화된 CG 알고리즘을 개발할 때, 범위성을 가지는 결과를 얻기 위해 가장 중요한 요소는 최적의 병렬화된 예조건인자(preconditioner)를 고안하는 것이다.

Basermann 등[1]은 "블록 ILU(0)" 형식의 예조건인자를 사용하여 여러 종류의 문제를 풀었다. 각각의 분할된 영역에서 구성된 국소행렬은 몇 개의 블록행렬로 분할되며 이웃하는 블록행렬과 내재적으로 연성된 부분을 무시하고 불완전 LU 분해법(incomplete LU factorization, 이하 ILU)을 적용한다. Issman 과 Degrez[2]는 부가적 Schwarz(additive Schwarz) 예조건인자를 이용하여 내재적으로 차분된 압축성 Navier-Stokes 방정식을 해석하였는데, 분할된 영역에 ILU(0)을 적용하였으므로 Basermann 등[1]이 사용한 블록 ILU(0)와 동일하다. 이들은 제시된 병렬 예조건화 기법과 GMRES 나 Bi-CGSTAB 의 반복해법을 사용하여 상당히 범위성을 가지는 결과를 얻었다.

한편, Wissink 등[3]은 천음속유동의 일종인 천음속미소교란(transonic small disturbance) 방정식을, 예조건인자로는 Di Brozolo 블록 ILU(0)를 사용하고 반복해법으로는 GMRES 또는 Orthomin 을 사용하여 병렬 계산하였다. 그들은 블록 ILU(0)에서 외부영역과 연성되는 부분을 무시해서 생기는 수렴성의 악화를 보상하기 위해 국소영역의 경계를 이웃 영역과 중복되게 한 후, 중복되는 영역에서 각 영역 예조건인자 값의 평균을 취하는 방법을 사용하였다. 국소블록행렬의 수가 작은 경우(프로세서 수 < 8) 좋은 결과를 얻을 수 있음을 보였지만, 프로세서 수가 더 증가함에 따라 범위성이 떨어지는 결함이 있었다. Chronopoulos 와 Wang[4]은 쌍곡선형의 교통흐름 문제를 해석하기 위하여 Di Brozolo 블록 ILU(0)와 Orthomin 을 이용하였다. Wissink 등[3]의 결과에서 추론할 수 있듯이 프로세서 수가 증가할수록 효율이 떨어지는 경향을 보였다.

Magolu monga Made 와 van der Vorst[5,6]는 기존의 블록 ILU(0)의 성능을 향상시키기 위해 국소행렬을 이웃 영역의 국소행렬과 중복되도록 하고 중복된 영역에서 ILU 시 고려하는 항(fill-in)을 증가시켜 예조건인자를 적용한 결과, 타원형 방정식에 대해 블록 ILU(0)와 비교하여 더 범위성을 가지는 결과를 얻었다. 이들은 중복되는 영역을 유지시킨 상태에서 ILU 시 고려하는 항을 늘리는 것보다 중

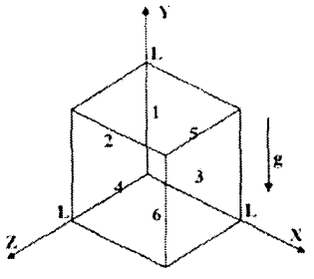


Fig. 2 Computational domain for 3-dimensional problems

복되는 영역을 늘림으로써 고려하는 항을 증가시키는 것이 더 효과적으로 블록 ILU(0)의 성능을 향상시킴을 보였다.

한편, Saad 와 Sosonkina[7]는 Schur 보행렬(Shur complement)을 활용한 병렬 예조건인자를 제안하였다. Schur 보행렬을 이용할 경우 각 국소영역의 경계에 해당하는 변수들로만 구성된 Schur 시스템이 얻어지며, 이 시스템을 엄밀하게 풀면 각 국소영역의 경계에 해당하는 변수들의 값이 구해진다. 일단 각 국소영역의 경계에서의 값이 얻어지면 이를 경계조건으로 하여 각 국소영역은 각 프로세서에 의해 완전히 병렬적으로 풀리게 된다. 하지만 Schur 시스템을 엄밀하게 푸는 데에는 많은 시간이 소요되어 기존의 블록 ILU(0)에 비해 비효율적인 병렬 알고리즘이 된다. 따라서 Saad 와 Sosonkina[7]는 Schur 시스템을 GMRES 를 이용한 병렬 알고리즘에 의해 근사적으로 풀어 국소영역 경계에서의 값을 구한 후, 내부영역에서의 값을 구하는 반복계산 방법을 제시하였다. 한편, 여기서 Schur 시스템의 예조건인자는 따로 구해지지 않고 국소행렬의 ILU 로부터 자동적으로 얻어짐을 주의해야 한다. Schur 보행렬을 이용한 방법을 블록 ILU(0) 예조건인자와 비교하였는데, 블록 ILU(0)의 경우에는 예조건화 단계에서 ILUT(incomplete LU threshold) 예조건인자를 사용한 GMRES 를 이용하여 각각의 국소행렬을 근사적으로 풀게 된다.

본 연구에서는 Choi 등[8]에 의해 개발된 비압축성 Navier-Stokes 방정식을 해석하기 위한 유한요소 프로그램을 영역분할법을 이용하여 병렬화하고자 한다. 개발된 프로그램은 분리 알고리즘에 기반을 두고 있으므로 영역분할법을 운동량방정식, 압력방정식, 속도보정방정식에 각각 적용함으로써 병렬화한다. 영역분할 병렬계산 방법을 위해 알려진 대표적 병렬 예조건인자들인 대각 예조건화(diagonal preconditioning, 이하 DIAG), 국소영역을 중복하지 않는 블록 ILU(0)(block ILU(0) without overlapping, 이하 BIWO)[9], 분산 ILU(0)(distributed ILU(0), 이하 DILU)[10], 반복 블록 ILU(0) (iterative block ILU(0), 이하 ITBI)들을 각각의 지배방정식에 적용하여 성능을 비교한다. 더 나아가 수정된 분산 ILU(0)(modified distributed ILU(0), 이하 MDLU)를 제안하고 그 성능을 기존의 병렬 예조건인자와 비교한다. 반복계산 해법으로는 CG 와 van der Vorst[11]가 제안한 Bi-CGSTAB 를 사용한다.

2. 해석 대상 및 지배방정식

계산 영역은 3 차원 문제의 경우는 Fig. 2 와 같으며 2 차원 문제의 경우에는 정사각형 계산 영역을 사용한다. 지배방정식과 경계조건은 문제에 따라 다르며 비정렬 격자에 대한 병렬성능을 측정하는 경우를 제외하면 각 방향으로 같은 수의 균일 격자 또는 비균일 격자로 구성된 정렬 격자를 사용한다. 수 변화량의 최대값이 10^{-5} 이하이면 수렴된 것으로 판정한다.

2.1 라플라스 방정식

$$\frac{\partial^2 u}{\partial x_j \partial x_j} = 0 \tag{1}$$

지배방정식은 식 (1)과 같으며 경계조건으로 Fig. 2 의 면 1 에는 u = 1, 면 2 – 6 에는 u = 0 을 준다.

2.2 입방체내의 공동 유동 문제

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}(\sigma_{ij}) + S_i$$
$$\sigma_{ij} = \nu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3}$$

Re = 100, 400, 1000 인 경우에 대해서 계산을 수행하며 지배방정식은 식 (2) 및 (3)과 같다. 지배방정식은 4 단계 분리 계산법을 적용한 유한요소법[8]으로 해석한다. 시간 적분 방법으로 2 차 정확도를 가지는 Crank-Nicolson 방법을 사용하며 공간 차분 방법으로 중심 차분 기법(central difference)인 Galerkin 방법을 사용한다. 경계조건은 Fig. 2 의 면 2 – 5 에서 $u_i = 0$ 이며 면 1 에서 $u_1 = U$ 이다.

2.3 입방체내의 자연 대류 문제

연속방정식은 식 (2)와 같고 운동량방정식은 식 (3)에 Boussinesq 가정을 사용하여 y 방향으로 밀도 차이에 의한 힘을 추가한다. 에너지방정식은 다음과 같이 표시된다.

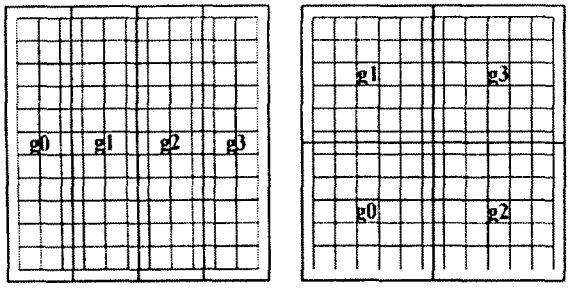


Fig. 3 Decomposition of computational domain: (a) 1-dimensional, (b) 2-dimensional

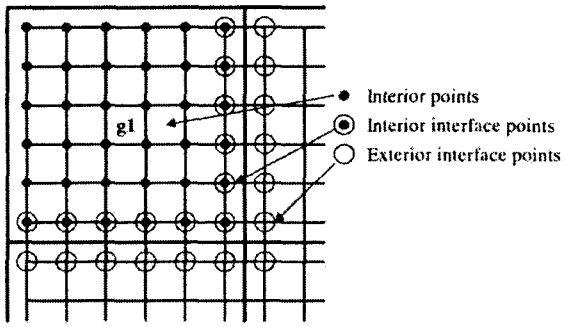


Fig. 4 Classification of nodes in a subdomain

$$\frac{\partial T}{\partial t}+\frac{\partial}{\partial x_j}(Tu_j)=\frac{\partial}{\partial x_j}(\alpha\frac{\partial T}{\partial x_j}) \tag{4}$$

$$\text{면}1,4\text{-}6: u_i=0, \frac{\partial T}{\partial n}=0$$

$$\text{면}2: u_i=0, T=T_C \tag{5}$$

$$\text{면}3: u_i=0, T=T_H$$

경계조건은 식 (5)와 같으며 Ra = 10^5, Pr = 0.71인 경우에 대해 계산을 하였다. 분리 알고리즘[8]을 사용하며 등온 상태의 유체가 양 측벽으로부터 가열과 냉각되면서 자연 대류가 일어나는 경우의 정상상태 해를 구한다.

3. 병렬화

병렬 컴퓨터를 사용하여 문제를 해석하기 위해 전체 격자를 구성한 후 계산 영역을 여러 개의 국소영역으로 나누는 영역분할법을 사용한다. 각 프로세서는 할당된 영역에서 독립적으로 국소행렬을 구성하며 구성된 각 국소행렬을 풀기 위해 영역의 경계에서 자료의 교환이 필요하다. 프로세서 간의 자료 교환은 MPI(message passing interface)[12]를 사용하며 예조건화된 공액구배법(preconditioned conjugate gradient, 이하 PCG) 알고리즘을 병렬화하여 전체 영역에서의 해를 구한다.

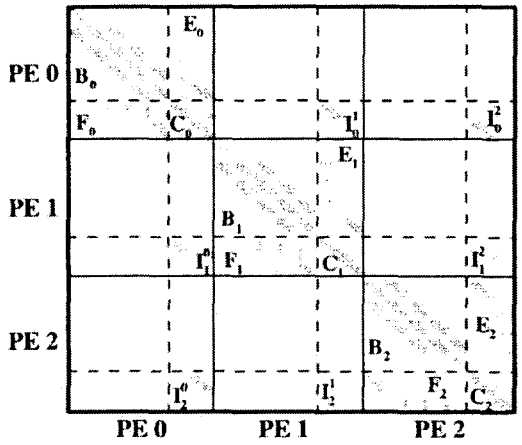


Fig. 5 Global non-zero pattern of matrix

3.1 영역분할법

효과적인 병렬화를 위해 프로세서간의 계산량을 균등하게 분배하고 자료 교환을 최적화하도록 영역을 분할하여야 한다. 본 연구에서는 계산 영역의 분할이 용이하도록 2 차원 문제의 경우 직사각형, 3 차원 문제의 경우 직육면체로 구성된 정렬 격자계를 주로 사용하여 병렬화된 알고리즘의 성능을 측정한다. 정렬 격자계를 사용한 경우에는 1 차원 분할과 다차원 분할을 통하여 영역을 분할한다. 비정렬 격자계의 경우에는 위의 영역분할을 적용하는 것이 쉽지 않으므로 METIS[13] 라이브러리를 사용하여 영역을 분할한다.

Fig. 3 은 2 차원 라플라스 방정식을 해석하기 위한 계산 영역을 각각 1 차원 및 2 차원 분할한 것이다. 1 차원 분할은 경계를 공유하는 프로세서가 2 개로 한정된다는 장점을 가지고 있지만 경계의 크기가 크므로 교환해야 할 경계치가 많다는 단점을 가지고 있다. 반면 다차원 분할은 프로세서 수가 증가할수록 각 프로세서의 경계의 크기가 작아지므로 각 프로세서 당 교환할 자료 교환량은 줄어들게 되나 1 차원 분할보다 상대적으로 경계를 공유하는 프로세서 수가 많다.

절점을 중심으로 계산 영역을 분할하며 각 영역은 각 프로세서의 계산량이 균등하도록 동일한 수의 절점으로 분할되었다. Fig. 4 와 같이 분할된 각 영역은 내부 절점, 내부 경계 절점, 외부 경계 절점으로 구성된다[14]. 전체 계산 영역이 분할된 후, 각 분할된 영역에서 절점은 내부 절점, 내부 경계 절점, 외부 경계 절점의 순으로 번호가 다시 매겨

진다. 내부 절점은 각 프로세서가 독립적으로 계산하며, 경계 절점들은 계산하기 전에 주위 프로세서와 값의 교환을 필요로 한다. 분할된 각 영역에서는 내부 절점과 내부 경계 절점만이 계산, 저장되므로 각 영역에서 국소행렬이 구성될 때, 외부 경계 절점에 대한 행은 무시된다. 내부 절점과 내부 경계 절점을 풀기 위한 국소행렬 $[A_n]$는 n 번 프로세서에서 다음과 같이 나타나며 그 형태는 Fig. 5 와 같다:

$$[A_n]\{x_n\} = \{f_n\} \tag{6}$$

$$[A_n] = \begin{pmatrix} B_n & E_n & 0 \\ F_n & C_n & I_n^m \end{pmatrix}, \{x_n\} = \begin{pmatrix} u_n \\ v_n \\ w_n^m \end{pmatrix}, \{f_n\} = \begin{pmatrix} s_n \\ t_n \end{pmatrix}$$

여기서 $\{u_n\}$와 $\{s_n\}$는 영역 내부 절점에 대한 값이고 $\{v_n\}$와 $\{t_n\}$는 영역 내부 경계에 대한 값이다. 이들은 영역 내부의 값으로 통신이 필요하지 않으나, $\{w_n^m\}$는 이웃하는 m 번 프로세서의 값이다. $[B_n]$는 영역 내부 절점간의 행렬이고 $[E_n]$와 $[F_n]$는 영역 내부와 영역 내부 경계 절점간의 행렬이다. 또 $[C_n]$는 영역 내부 경계 절점간의 행렬이며 $[I_n^m]$는 영역 내부 경계가 외부 경계 절점과 연성된 부분을 나타내는 항이다. 따라서 식 (6)의 $\{w_n\}$와 $[I_n^m]$는 다른 영역과 연성된 부분으로 이 부분을 풀기 위해서는 다른 영역과의 통신이 필요하게 된다.

3.2 예조건화된 공액구배법(PCG) 알고리즘

일반적인 PCG 알고리즘은 수렴하게 될 때까지 다음 식들에서와 같이 반복하게 되며 초기 잔여 벡터는 $\{r_0\}=\{f_n\}-[A]\{x_0\}$과 같이 정의된다. 이 식들에서 하첨자 j 는 반복 계산 횟수를 의미한다.

$$\alpha_j = \frac{\{z_j\}^T\{r_j\}}{\{d_j\}^T[A]\{d_j\}} \tag{7}$$

$$\{x_{j+1}\} = \{x_j\} + \alpha_j\{d_j\} \tag{8}$$

$$\{r_{j+1}\} = \{r_j\} - \alpha_j[A]\{d_j\} \tag{9}$$

$$\{z_{j+1}\} = [M]^{-1}\{r_{j+1}\} \tag{10}$$

$$\beta_j = \frac{\{z_{j+1}\}^T\{r_{j+1}\}}{\{z_j\}^T\{r_j\}} \tag{11}$$

$$\{d_{j+1}\} = \{z_{j+1}\} + \beta_j\{d_j\} \tag{12}$$

알고리즘의 병렬화시 국소행렬 및 국소벡터는 분할된 영역에서 독립적으로 각각 정의되므로 식 (8), (9), (12)는 각 프로세서에서 독립적으로 수행된다. 식 (7)과 (11)에서 α_j, β_j는 전체 계산 영역에서의 값이므로 분할된 영역에서 독립적으로 국소벡터의 내적을 한 후 모든 국소영역에서 그 값을 모아서 구한다. 식 (10)에서 예조건화 벡터를 구하는 방법은 병렬 예조건화 기법에 따라 그 계산 과정이 달라진다.

3.3 병렬 예조건화 기법

반복해법의 수렴속도는 행렬의 조건수(condition number)에 영향을 받으며 예조건화 기법은 조건수를 줄이는 방법으로 행렬의 수렴속도를 가속화한다. 일반적으로 ILU[15] 계열의 예조건화 기법들이 직렬 계산(프로세서를 1 개만 사용하는 계산)에서 좋은 수렴 속도를 보이지만 병렬 계산에 적용하여 범위성을 가지는 결과를 얻기 위해서는 알고리즘의 수정이 필요하다.

3.3.1 대각 예조건화(DIAG)

$[M_n]=\mathrm{Diag}[A_n]$이고 $[M_n]^{-1}$은 대각 성분들의 역수로 구성이 된다. 대각 성분은 각 국소영역 내부의 값이므로 식 (10)의 과정은 각 프로세서가 독립적으로 수행할 수 있다. 따라서 식 (10)의 과정에서 병렬화로 인한 추가적인 알고리즘의 수정이나 통신 부하는 없다.

3.3.2 국소영역이 중복되지 않는 블록 ILU(0) (BIWO)[9]

일반적인 ILU(0)[15] 예조건화 기법을 사용하면 예조건인자는 $[M]=[L][U]$의 형태로 나오므로 식 (10)은 $[L][U]\{z\}=\{r\}$이 되며 전진대입법과 후진대입법을 이용하여 예조건화된 벡터 $\{z\}$를 구한다. 이 방법은 계산 영역 전체의 행렬을 순차적으로 이용하여 예조건인자를 구성한 후 예조건화된 벡터 $\{z\}$를 구하므로 병렬화하는 데 어려움이 있다.

BIWO 는 ILU 예조건화를 전체의 계산 영역 대신 분할된 각 국소영역에 독립적으로 적용시키는 것이다. $[A_n]$에서 이웃하는 국소영역과 관련된 부분의 행렬인 $[I_n^m]$를 무시하고 나머지 부분만을 ILU 예조건화 한다. $[I_n^m]$를 무시함으로써 각각의 프로세서는 예조건인자 $[M_n]=[L_n][U_n]$를 구하는 과정과 $[L_n][U_n]\{z\}=\{r\}$의 과정을 독립적으로 수행할 수 있는 장점이 있다. 그러나 경계 부분의 행렬 $[I_n^m]$를 무시하므로 영역을 분할하면 할수록 행렬에서 무시되는 항들이 증가하여 반복계산 횟수가 증가하는 단점이 있다.

3.3.3 반복 블록 ILU(0)(ITBI)

ITBI 는 예조건인자 $[M_n]=[L_n][U_n]$를 구하는 과정에서는 BIWO 와 같이 경계 부분의 행렬 $[I_n^m]$를

무시하지만, 식 (10)에서 예조건화된 벡터 {z}를 구할 때는 $[I_n^m]$를 고려한다. 즉, Fig. 5 의 경우에는 $\{z_1\}$(프로세서 1 번에서 예조건화된 국소 벡터)을 구하기 위해서 프로세서 1 번은 다음의 식을 푼다.

$$[M_1]\{z_1\} = \{r_1\} - [I_1^m]\{z_m\}, (m \neq 1) \quad (13)$$

여기서 1 번 프로세서의 경우 $[I_1^m]\{z_m\}$를 계산하기 위해서 프로세서 0 번과 2 번에서 국소적으로 계산되는 $\{z_0\}$과 $\{z_2\}$가 필요하다. 따라서 이 값들을 프로세서 1 번으로 전달하기 위한 부가적인 통신이 필요하다. 또한 모든 프로세서가 식 (13)을 동시에 풀기 위해서는 다른 영역의 값들이 필요하므로 식 (13)의 과정을 식 (14)와 같이 반복적으로 해석한다.

$$[M_1]\{z_1^{p+1}\} = \{r_1\} - [I_1^m]\{z_m^p\}, (m \neq 1) \quad (14)$$

여기서, 상첨자 p 는 반복계산 횟수의 단계를 나타낸다. 전진대입법과 후진대입법을 반복적으로 사용하며 풀기 전에 매번 경계 부분의 $\{z_m^p\}$을 이웃하는 국소영역으로부터 받아 갱신해 주어야 한다. 본 연구에서는 이 추가적인 반복 계산을 1 번만 한다. ITBI 는 영역을 분할하더라도 CG 해법의 반복 계산 횟수가 거의 늘어나지는 않으나 일반적인 ILU(0)에 비해 반복계산 당 계산량이 많다는 단점이 있다.

3.3.4 분산 ILU(0)(DILU)[10]

DILU 에서는 각 국소영역에서 조립된 국소조립 행렬을 모아 전체행렬을 구성한 후 ILU(0) 예조건화를 수행한다. 따라서 블록 ILU(0) 계열과 달리 외부 경계 절점과 연관된 부분의 행렬 $[I_n^m]$에 대해서도 ILU(0) 연산을 수행한다. 그러므로 n 번 프로세서에서 이웃 프로세서와 연성되지 않는 $[B_n]$와 $[E_n]$ 부분에 대한 ILU(0) 연산은 각각의 프로세서들이 독립적으로 수행할 수 있지만 $[C_n]$, $[F_n]$, $[I_n^m]$를 포함하는 행의 경우 ILU(0) 연산을 수행하기 위해서는 이웃하는 k(k < n)번 프로세서로부터 ILU(0) 연산이 이미 수행된 경계 블록 [C]와 $[I^m]$의 값들을 전달 받아야 한다. 따라서 예조건화 행렬 [M]=[L][U]를 구성하기 위해서는 영역간의 통신과 대기 시간이 필요하므로 알고리즘의 특성상 범위성을 가지는 결과를 얻기 어렵다.

일단 병렬 예조건화 행렬이 구해진 후에는 식 (10)을 이용하여 예조건화 벡터 {z}를 구해야 한다. 벡터 {z}는 다른 영역과 연성된 $\{z^{ext}\}$와 연성되지 않은 $\{z^{int}\}$으로 구성된다. 이 경우에 블록 ILU(0)와 마찬가지로 전진, 후진대입법을 이용하여 [M]{z}={r} ([L][U]{z}={r})에서 {z}를 구한다. 각 국소영역에서 절점들은 내부 절점, 내부 경계 절점, 외부 경계 절점 순으로 그 번호가 매겨지며 전진, 후진대입법들은 계산이 순차적이므로 위의 절점 순서대로 계산이 진행된다. 즉, 전진대입법 과정에서는 내부 및 내부 경계 절점에 대해 계산을 먼저 한다. 외부 경계 절점의 경우 행렬 $[I_n^m]$가 무시되지 않으므로 n 번 프로세서는 k(k < n)번 프로세서의 경계 부분 행렬로부터 전진대입법을 적용한 값 $[U_k]\{z_k^{ext}\}$를 받아야 한다. $[U_n]\{z_n\}$를 구한 후에는 l(l > n)번 프로세서로 n 영역 경계 부분의 행렬에 해당되는 $[U_n]\{z_n^{ext}\}$를 보낸다. 즉 각 영역에서 전진대입법 계산 과정은 다음과 같다.

① 각 영역에서 독립적으로 $[U_n]\{z_n^{int}\}$를 구한다.

② 연성된 k(k < n)번 프로세서로부터 $[U_k]\{z_k^{ext}\}$를 받는다.

③ 연성된 벡터 $[U_n]\{z_n^{ext}\}$를 구한다.

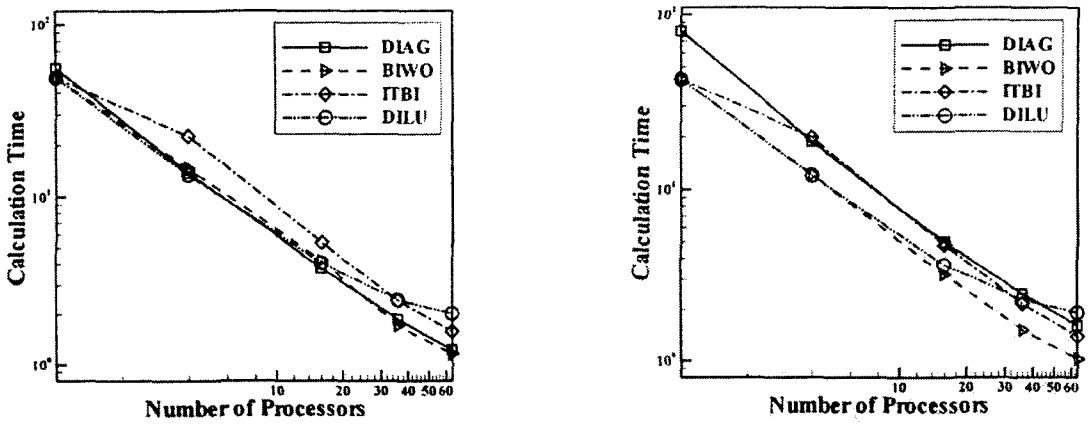
④ 연성된 l(l > n)번 프로세서로 $[U_n]\{z_n^{ext}\}$를 보낸다.

후진대입법의 경우에는 위의 과정을 역순으로 한다. 이 방법은 직렬 계산에서의 ILU(0) 예조건화 방법에 비해서 반복계산 횟수는 거의 증가하지 않지만 본질적으로 병렬화시킬 수 없는 부분이 많고, 대기 시간으로 인해 범위성을 가지는 결과를 얻기 어려운 단점이 있다.

3.3.5 수정된 분산 ILU(0)(MDLU)

내부 경계 절점에서의 예조건화된 벡터 값을 구하기 위한 이웃 영역과의 자료 교환 및 그에 수반하는 대기 시간은 DILU 에서 병렬 알고리즘의 효율을 떨어뜨린다. 즉, 전진대입법 과정의 경우에는 n 번째 영역의 내부 절점 값은 각 프로세서에서 독립적으로 구하지만 내부 경계 절점 값을 구하기 위해서는 먼저 n 번 프로세서의 내부 경계 절점 중 k(k < n)번 프로세서와 연성된 부분의 계산을 위해 k 번 프로세서로부터 필요한 값을 받고 내부 경계 절점 중 나머지 부분(l(l > n)번 프로세서와 연성된 부분)에 대해서 계산을 수행한 후 프로세서 l 에 필요한 자료를 보낸다. 따라서 본 연구에서는 DILU 의 대기 시간을 줄이기 위해 두 알고리즘을 제시하고 그 효율을 검증한다. 첫번째는 MDLU-I 알고리즘으로 다음과 같다.

(1) n 번 프로세서의 내부 경계 절점에서 전진대입법을 적용한 값([U]{z})을 구할 때, 대기 시간을 줄이기 위해 계산 순서를 바꾼다. k(k < n)번 프로세서와 연성된 부분의 계산은뒤로 미루고, 먼저 l(l > n)번 프로세서와 연성된 부분을 내부 영역에 속하는 부분과 함께 각각의 프로세서에서 독립적으로 계산한다. 여기서, l 번 프로세서와



(a) (b)

Fig. 6 Calculation time with 512 × 512 (a) uniform and (b) nonuniform structured grids

연성된 내부 경계 절점의 값을 구할 때 k 번 프로세서와 연성된 일부의 0 이 아닌 항(non-zero)은 무시한다.

(2) 각각의 n 번 프로세서는 k 번 프로세서와 연성된 나머지 값을 구하기 위해 1 번 프로세서로 $[U_n]\{z_{n-l}^{ext}\}$ 를 보내고 k 번 프로세서로부터 $[U_k]\{z_{n-k}^{ext}\}$ 을 받은 후 k 번 프로세서와 연성된 부분의 값을 구한다.

MDLU-II 는 대기 시간을 줄이기 위해 이웃하는 영역과의 경계 중 그 경계의 절점 수가 적은 영역과의 자료 교환을 무시한다. 정렬 격자를 사용한 경우에는 값을 교환해야 하는 이웃영역의 수가 2 차원의 경우 최대 8 개에서 4 개로, 3 차원은 최대 26 개에서 6 개로 줄어들게 된다. 그 결과 $[I_n^m]$ 해당하는 항 중 일부의 항들을 무시하게 되어 반복계산 횟수가 약간 늘어난다.

MDLU-III 는 MDLU-I 과 MDLU-II 를 결합한 알고리즘이다.

4. 결과 및 토론

병렬처리의 효율성을 판단하기 위한 척도로 다음과 같이 정의되는 속도향상(speedup)을 사용한다.

$$Speedup = \frac{T_1}{T_n} \tag{15}$$

T_1: 1 개의 프로세서를 사용했을 때 계산 시간

T_n: n 개의 프로세서를 사용했을 때 계산 시간

병렬화로 인한 부하가 없는 이상적인 경우에 속도향상은 프로세서 수와 같지만 실제에 있어서는 영역간의 자료 교환, 동기화를 위한 대기 시간과 병렬화로 인한 추가 계산 등으로 인해 그 값이 감소한다.

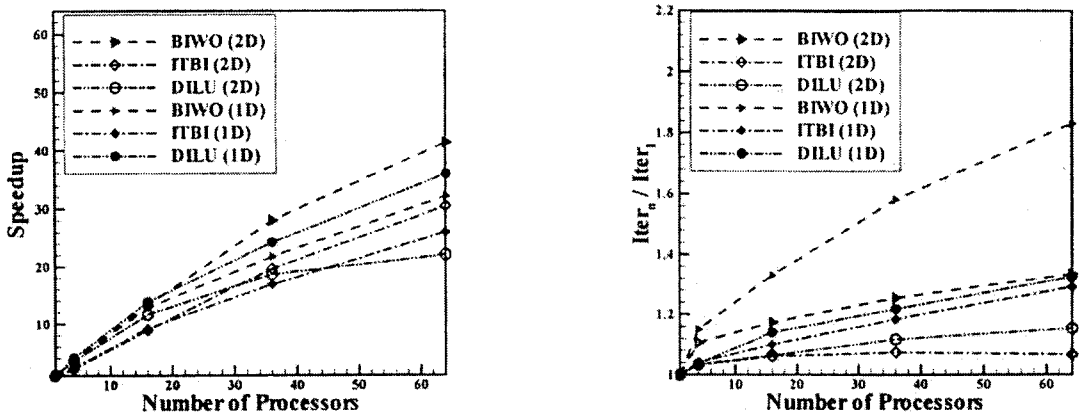
4.1 2 차원 라플라스 방정식에서 병렬 예조건화 기법의 성능

4.1.1 정렬 격자계에서의 성능

Fig. 6(a)와 6(b)는 512 × 512 균일 격자와 비균일 정렬 격자를 2 차원 영역분할하고 예조건화 기법들을 적용한 경우의 계산 시간을 비교한 것이다. DIAG 는 균일 격자보다 비균일 격자에서 그 성능이 떨어지지만 다른 예조건화 기법들은 균일 격자나 비균일 격자에서 거의 비슷한 성능을 보인다. 즉, DIAG 는 다른 기법들에 비해 행렬이 풀기 어려워질수록(조건수가 커질수록) 반복계산 횟수가 크게 증가한다. 대부분의 예조건화 기법들은 프로세서 수가 증가함에 따라 계산 시간도 이에 비례하여 줄어든다. 그러나 ITBI 는 다른 기법들과는 달리 프로세서 수가 1 개에서 4 개로 변할 때 계산 시간의 감소량이 작은데, 이것은 예조건화된 벡터 계산 과정의 추가적인 반복계산으로 인해 반복계산 한 번당 계산량이 순차계산에 비해 늘어나게 되기 때문이다. DILU 는 다른 기법들에 비하여 프로세서가 증가함에 따라 프로세서 대기 시간이 급격하게 증가한다. 이로 인해 DILU 는 프로세서 수가 증가할수록 점차 계산 시간의 감소량이 줄어든다.

4.1.2 영역분할의 영향

영역분할을 하는 방법에 따라서도 각 예조건화 기법의 속도향상이 차이를 보인다. 영역분할을 하는 방법에 따라서 프로세서간의 통신 부하, 전체 행렬의 형태와 예조건화시 무시되는 항이 변하기



(a) (b)

Fig. 7 Performance with 1-dimensional and 2-dimensional domain decompositions: (a) speedup, (b) $Iter_n / Iter_1$

때문이다. Fig. 7 은 512 × 512 비균일 정렬 격자에서 1 차원 영역분할과 2 차원 영역분할을 하였을 때의 속도향상과 반복계산 횟수 증가이다. DIAG 예조건화 기법은 Fig. 7 에 도시하지는 않았지만 예조건화시 무시하는 항이 없으며 통신 부하의 변화에 의한 영향이 거의 나타나지 않으므로 영역분할에 의한 차이가 거의 없다. 그 외의 예조건화 기법은 영역분할을 하는 방법에 따라 Fig. 1 과 같이 가상적으로 재구성되는 전체행렬의 변화와 예조건화시 무시되는 항들의 변화로 인해 반복계산 횟수가 변하게 된다. Fig. 7(b)에 의하면 같은 예조건화 기법을 사용하면 2 차원 영역분할을 한 경우가 1 차원 영역분할보다 반복계산 횟수의 증가가 적다. 이는 2 차원 영역분할이 1 차원 영역분할보다 경계의 절점 수가 적기 때문이다. 특히 BIWO 는 경계의 크기가 늘어나게 되면 이로 인해 예조건화시 무시되는 항들이 늘어나므로 1 차원 영역분할을 한 경우 반복계산 횟수가 더 크게 증가한다. ITBI 는 BIWO 와 같은 예조건화 행렬을 사용하지만 예조건화 행렬에서 무시되는 항을 추가적인 계산으로 고려함으로써 반복계산 횟수의 증가를 억제하므로 그 영향이 크게 나타나지 않는다. BIWO 와 ITBI 는 반복계산 횟수의 증가로 인해 2 차원 분할을 한 경우가 1 차원 분할에 비해 속도향상이 더 크다. 반면 DILU 의 경우에는 반복계산 횟수는 2 차원 분할이 더 적으나 통신 부하의 차이로 인해 2 차원 분할의 성능이 더 떨어진다. 통상 DILU 는 각 예조건화 행렬이 서로 연성되어 있고 예조건화된 벡터는 순차적으로 계산되어야 하므로 앞의 프로세서로부터 관련된 값을 받을 때까지 대기해야 한다. 그러나 1 차원 분할의 경우에는 이웃하는 두 경계가 서로 연성되어 있지 않으므로 앞 번호의 프로세서로부터 값을 받기 전에 뒷 번호의 프로세서와 연성된 부분을 먼저 계산하여 보낼 수 있다. 반면 2 차원 분할의 경우에는 앞 번호의 프로세서로부터 값을 받아 계산을 완료할 때까지 뒷 번호의 프로세서로 값을 보내는 것을 연기해야 한다. 따라서 2 차원 분할은 프로세서 수가 증가할수록 통신 부하가 급격히 증가한다.

4.1.3 문제 크기의 영향

속도향상은 문제의 크기에 의해서도 차이를 보인다. Fig. 8 은 비균일 격자를 2 차원 영역분할 하였을 때 문제의 크기에 따른 속도향상을 비교한 것이다. 모든 예조건화 기법들이 Fig. 8(a)보다 8(b)에서 속도향상의 기울기가 커진다. 절점 수가 많아질수록 각 영역의 총절점수에 비해 영역 경계의 절점 수가 적어지므로 각 프로세서에 할당되는 계산량에 비해 영역간의 통신으로 인한 부하가 상대적으로 줄어들기 때문이다. 대부분의 예조건화 기법들은 두 경우 속도향상의 크기는 다르지만 비슷한 경향을 보이는데 반해 DILU 는 512 × 512 격자에서 보다 256 × 256 격자에서 더 적은 프로세서 수에서 속도향상의 기울기가 급격히 감소되고 있다. 2 차원 분할을 한 DILU 에서 통신 부하는 속도향상 감소의 주요한 원인이며, 통신시간과 대기시간으로 구성된다. DILU 는 프로세서 수가 증가할수록 값을 교환해야 하는 절점 수가 줄어 통신시간은 줄지만 전체적인 절점 수가 늘어나서 대기시간은 증가한다. 즉, 문제의 크기가 커질수록 줄어드는 대기 시간의 영향이 급격한 속도향상 기울기 감소의 원인임을 추론할 수 있다. ITBI 예조건화는 병렬화에 의한 반복계산 한 번당 계산량의 증가로 인해 프로세서의 수가 1 개에서 4 개로 증가할 때 속도향상의 기울기가 다른 예조건화 기법

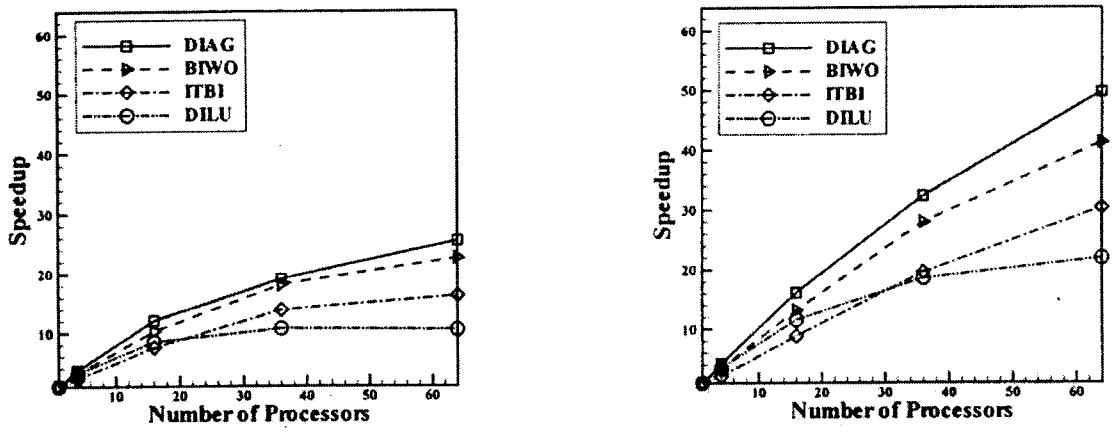


Fig. 8 Speedup of nonuniform structured grid: (a) 256 × 256, (b) 512 × 512

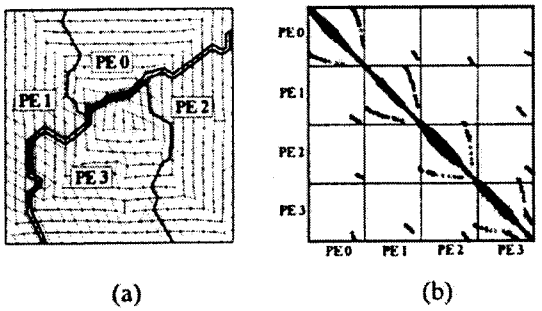


Fig. 9 Domain decomposition of unstructured grid: (a) partitioned grid, (b) global non-zero pattern of matrix

보다 작게 나타나지만 이후에는 다른 기법들과 거의 유사한 경향을 보인다.

4.1.4 비정렬 격자계에서의 성능

비정렬 격자계를 사용하여 계산을 수행하는 경우에서 정렬 격자계를 사용한 경우와 다른 점은 앞에서 적용한 1 차원 및 다차원 영역분할을 그대로 적용할 수 없다는 점이다. 그 이외의 과정은 어느 격자계를 사용하든지 차이가 없다.

본 연구에서는 비정렬 격자계에서 영역을 분할하기 위해 METIS[13] 라이브러리를 사용한다. 이 라이브러리는 통신 부하를 최소화하고 계산량을 균등하게 배분하도록 영역을 분할한다. Fig. 9(a)는 비정렬 격자를 사용한 정사각형의 계산영역을 METIS 라이브러리를 이용하여 분할한 것이다. 이 분할된 계산영역으로부터 생성되는 행렬에서 0 이 아닌 항의 형태는 Fig. 9(b)와 같으며 정렬 격자계를 2 차원 분할한 행렬의 형태인 Fig. 1(b)와 유사함을 확인할 수 있다. 본 연구에서는 전체 영역을 분할한 후, 각 국소영역에서 절점의 번호를 내부 절점, 내부 경계 절점, 외부 경계 절점의 순으로 다시 매긴다. 즉, 이로 인해 어느 격자계를 사용하든지 절점의 구성이 유사해지므로 2 차원 영역분할을 한 정렬 격자계와 METIS 를 사용한 비정렬 격자계의 행렬 형태가 유사하게 나온 것이다.

Fig. 10 은 2 차원 라플라스 방정식을 비정렬 격자계를 사용하여 해석하는 경우의 병렬 성능을 측정한 것이다. Fig. 10(a)의 비정렬 격자계에서 속도향상은 Fig. 8(a)의 정렬 격자계에서 속도향상과 그 경향이 유사하게 나타난다. 반면 Fig. 10(b)의 반복계산 횟수는 정렬 격자계에서의 반복횟수 증가인 Fig. 7(b)와 비교하면 그 경향이 다르게 나타난다. 정렬 격자계에서는 프로세서 수가 증가하더라도 분할된 각 국소영역의 모양이 유사하지만, METIS 를 사용한 비정렬 격자계의 경우에는 각 국소영역이 분할하는 개수나 격자에 따라 모두 다른 모양을 보이기 때문이다. 즉, 비정렬 격자계에서 병렬 성능은 반복계산 횟수의 증가를 제외하면 정렬 격자계와 유사함을 확인할 수 있다.

4.2 3 차원 유동문제에서 병렬 예조건화의 성능

Navier-Stokes 방정식의 병렬해를 구하는 알고리즘을 검증하기 위하여 공동 유동 문제와 자연 대류 문제를 풀어 기존의 결과와 비교하였다. 공동 유동 문제는 Re = 100, 400, 1000 인 경우를 32 × 32 × 32 비균일 정렬 격자를 사용하여 해석한다. Fig. 11 에서는 (x, z) = (0.5, 0.5)인 지점에서 Jiang 등[16]이 계산 영역의 절반을 50 × 52 × 25 격자를 사용하여 속도-압력-와도 공식화(velocity-pressure-vorticity formulation)와 최소자승 유한요소법으로 계산한 결과와 각 레이놀즈 수의 x 방향 속도 성분을 비교하여 잘 일치함을 확인할 수 있다.

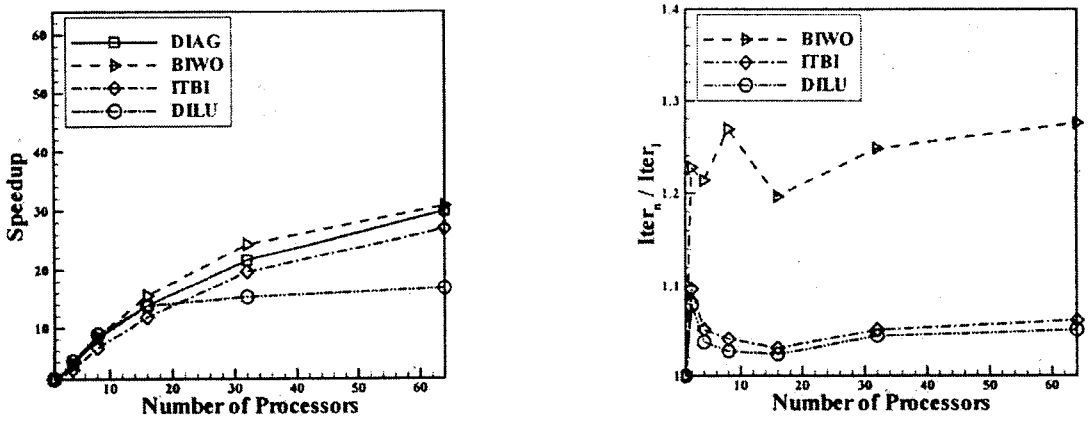


Fig. 10 Performance with unstructured grid: (a) speedup, (b) $Iter_n$ / $Iter_1$

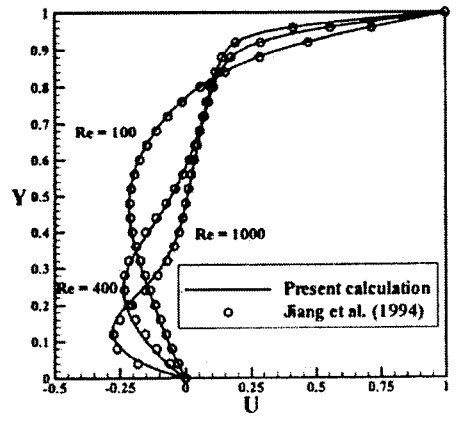
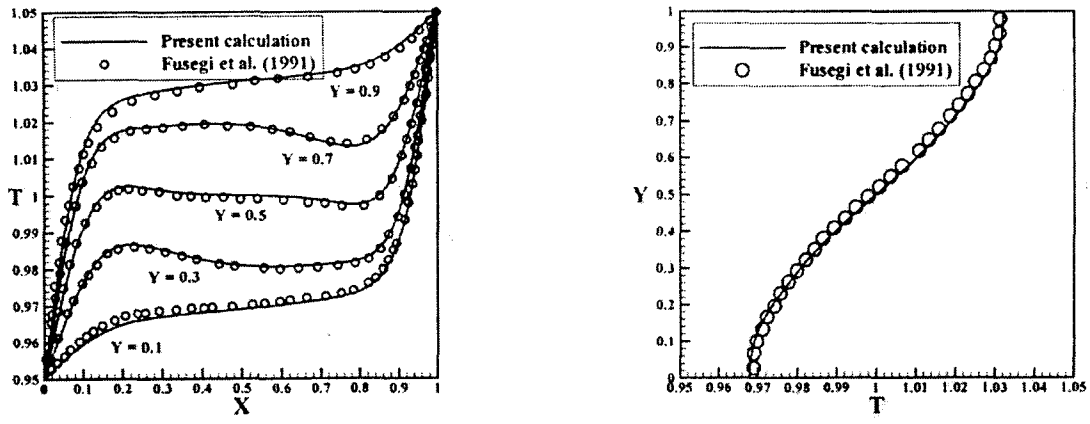


Fig. 11 Velocity U at (x, z) = (0.5, 0.5) for 3-dimensional driven cavity flow at Re = 100, 400 and 1000

자연 대류 문제는 62 × 62 × 62 비균일 정렬 격자를 사용하여 계산하며 Fusegi 등[17]이 62 × 62 × 62 격자를 사용하여 SIMPLE 과 유한차분법으로 계산한 결과와 비교한다. Fig. 12 와 13 에서는 z = 0.5 인 단면에서 각각 온도와 속도를 비교하며 기존의 결과와 잘 일치하는 값을 얻었다.

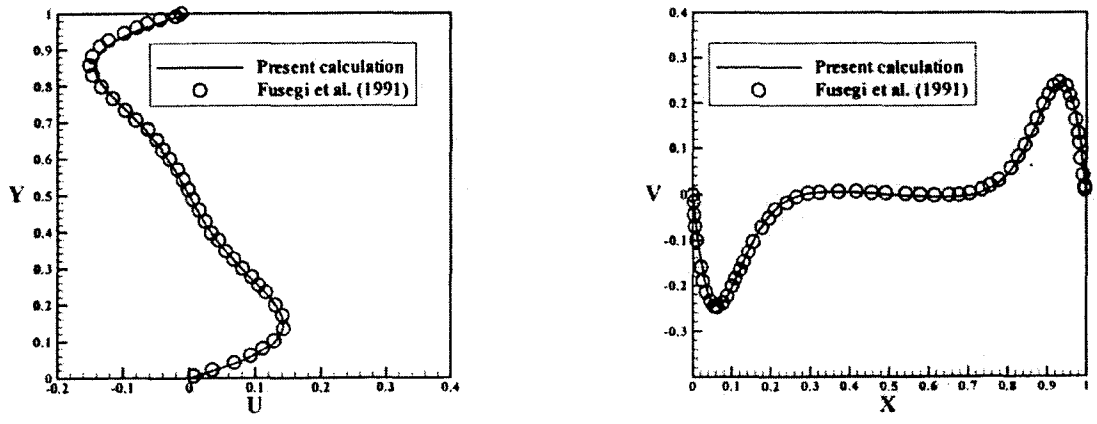
Fig. 14 는 3 차원 영역분할을 한 32 × 32 × 32 비균일 격자에서의 3 차원 라플라스 방정식과 자연 대류 문제의 속도향상 결과이다. 앞의 2 차원 라플라스 방정식에 적용한 결과와 비교하면 3 차원의 경우 속도향상이 매우 작다. 이것은 이 3 차원 문제가 앞의 2 차원 문제보다 통신 부하가 상대적으로 크기 때문이다. 2 차원 격자의 경우 2 차원 영역분할을 하면 각 국소영역이 N^2 의 절점을 가질 때 약 4N 정도의 경계 절점을 가진다. 3 차원 격자는 3 차원 영역분할을 할 경우 N^3 의 절점을 가질 때 약 $6N^2$ 의 경계 절점을 가진다. 즉 경계 절점과 내부 절점의 비율은 두 경우 모두 약 1/N 이지만 (4/N for 2-D, 6/N for 3-D) 3 차원의 경우 기억 용량의 문제로 인하여 2 차원 문제보다 더 N 이 작은 경우에서 속도향상을 측정하였으므로 속도향상이 작게 측정된 것이다. 2 차원 라플라스 방정식과 3 차원 라플라스 방정식의 속도향상은 정량적인 면에서는 많은 차이를 보이지만 정성적인 면에서는 유사한 경향을 보인다. 반면 Fig. 14(b)의 자연 대류 문제는 같은 격자를 사용한 라플라스 방정식보다 대체적으로 더 큰 속도향상 결과가 나타난다. 이는 같은 격자를 사용하더라도 자연 대류 문제가 라플라스 방정식보다는 통신량에 비해 병렬화가 가능한 계산량(예: 전체 행렬 계산)의 비율이 커지기 때문이다. 따라서, 문제의 크기가 커지면 통신 부하가 계산량에 비해 상대적으로 줄게 되어서 속도향상이 더 커질 것이다. 반면 DILU 예조건화 기법은 통신 부하의 영향으로 인하여 두 경우 성능의 차이가 거의 없다. 3 차원의 경우에는 2 차원의 경우보다 각각의 프로세서가 연성되어 프로세서 수가 많으며 DILU 예조건화는 앞에서 언급한 바와 같이 다른 예조건화 기법에 비해 프로세서 수가 증가할수록 통신 부하가 급격히 늘어나게 된다. 따라서, 두 경우 모두 급격히 증가한 통신 부하로 인해 DILU 예조건화 기법의 성능이 저하된 것이다.

Fig. 15 는 자연 대류 문제를 해석하기 위한 분리 알고리즘에서 운동량방정식과 압력방정식의 속도향상을 각각 도시한 것이다. 방정식을 푸는 알고리즘은 크게 행렬을 생성하는 독립적인 부분과 행렬을 해석하는 연성된 부분으로 나뉜다. 압력방정식의 경우에는 행렬을 해석할 때 많은 반복계산 횟수가 필요하므로 많은 부분의 계산이 연성된 부



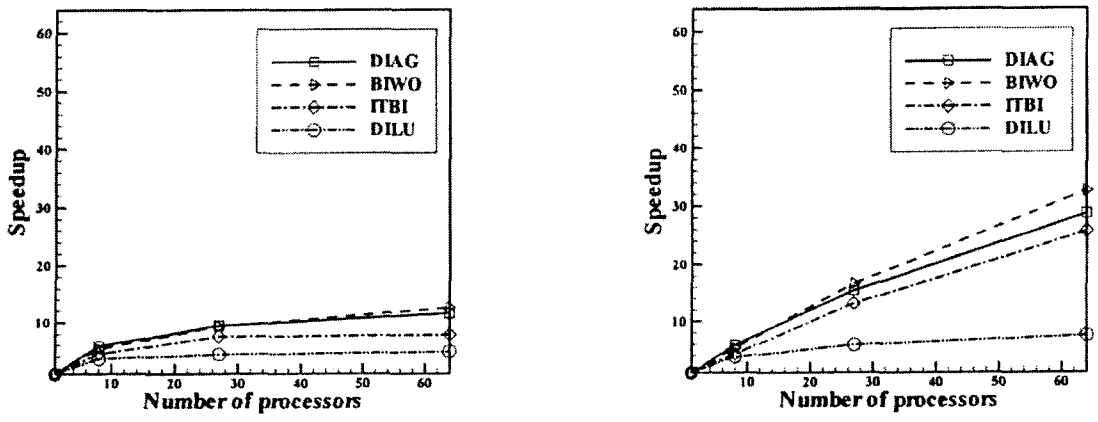
(a) (b)

Fig. 12 Temperature at z = 0.5 for natural convection: (a) at various heights, (b) at x = 0.5



(a) (b)

Fig. 13 Velocity at z = 0.5 for natural convection: (a) at x = 0.5, (b) at y = 0.5



(a) (b)

Fig. 14 Speedup of 3-dimensional problems: (a) Laplace equation, (b) natural convection

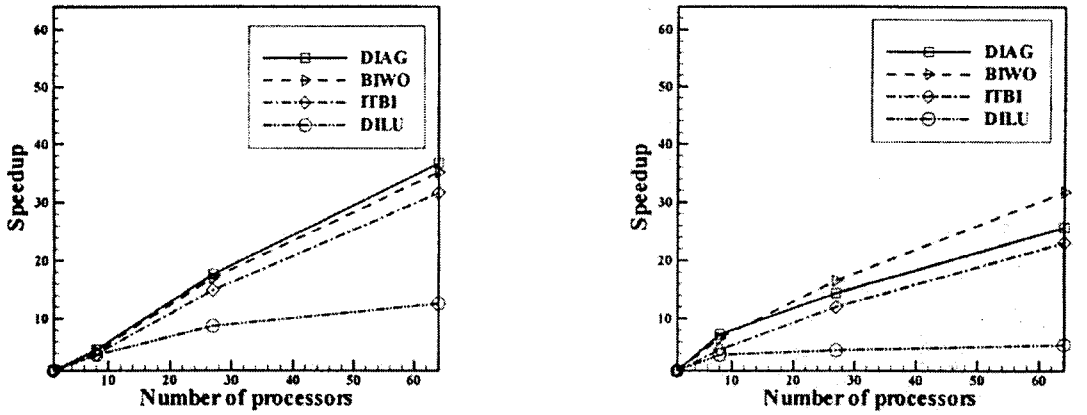


Fig. 15 Speedup of natural convection problem: (a) momentum equation, (b) pressure equation

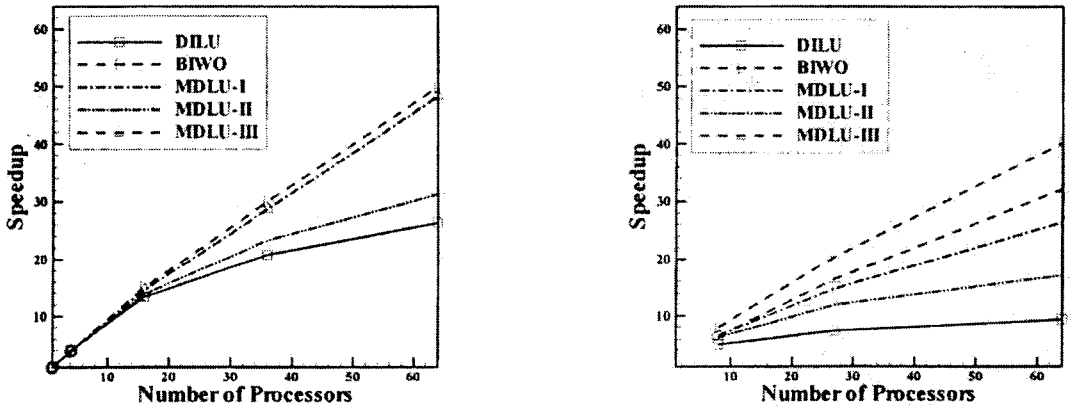


Fig. 16 Comparison of MDLU with DILU and BIWO: (a) 2-dimensional Laplace equation with 512 × 512 grid, (b) 3-dimensional Laplace equation with 64 × 64 × 64 grid

분에 해당된다. 반면, 운동량방정식은 적은 수의 반복계산을 통해 풀리므로 독립적인 부분의 비율이 운동량방정식에 비해 크다. 그 결과, 운동량방정식의 속도향상이 압력방정식보다 크게 나타난다.

4.3 MDLU 예조건화 기법의 성능

Fig. 16 은 MDLU 를 가장 좋은 속도향상을 보이는 BIWO 및 수정전의 DILU 와 2, 3 차원 라플라스 방정식에 적용하여 비교한 것이다. 2 차원 문제는 256 × 256 격자계를 사용하며 3 차원 문제는 64 × 64 × 64 격자계를 사용하였다. 본 연구에서는 128 개의 프로세서와 16 GB 의 메모리를 가지고 있는 Cray T3E 에서 연산을 수행하였다. 그로 인해 3 차원의 경우에는 기억용량 부족으로 인해 1 개의 프로세서에 의한 계산은 하지 않았으며 8 개의 프로세서를 사용한 결과 중 가장 빠른 BIWO 를 기준으로 속도향상을 측정한다. MDLU-II 보다 MDLU-I 의 속도향상이 더 크게 나오는 것으로부터 프로세서의 순차계산으로 인한 통신 부하가 DILU 성능 저하의 주원인이라는 것을 추정할 수 있다. 3 차원 문제의 경우에는 각 프로세서와 연성되는 이웃의 프로세서 수가 2 차원 문제보다 더 많기 때문에 그 수를 줄이는 것도 약간의 성능향상을 보여 준다. 2 차원 문제의 경우 MDLU 가 BIWO 과 비슷한 성능을 보이지만 3 차원 문제의 경우에는 그 성능이 떨어진다. 이는 본 연구에서 사용된 3 차원 문제의 경우 2 차원 문제 보다 반복계산 횟수가 적으며, 그 결과 BIWO 의 반복계산 횟수 증가가 크지 않았기 때문이다. 반면 MDLU 는 BIWO 에 비하면 반복계산 횟수의 증가가 상당히 적다. 즉, 반복계산 횟수가 큰(조건수가 큰 행렬) 문제에 MDLU 를 적용한다면 최소한 BIWO 보다 성능이 떨어지지는 않을 것이라는 점을 추정할

수 있다.

5. 결 론

본 연구에서는 분리 알고리즘을 적용한 동일 차수 유한요소법을 영역분할법과 MPI 를 이용하여 병렬화함으로써 Navier-Stokes 방정식의 해를 구하였다. 병렬 반복해법의 수렴속도를 가속화하기 위한 여러 병렬 예조건화 기법의 성능을 비교 검토하여 다음과 같은 결론을 얻었다.

(1) DIAG 예조건화 기법을 제외한 다른 기법들은 균일 격자와 비균일 격자의 반복계산 횟수의 차이가 크지 않았다.

(2) 다차원 영역분할을 하는 것이 1 차원 영역분할을 하는 것보다 반복계산 횟수의 증가가 적었으며 문제의 크기가 증가할수록 속도향상이 증가하였다.

(3) Navier-Stokes 방정식을 병렬 프로그램으로 해석할 때 반복계산 횟수가 클수록 통신 부하의 비율이 커지며 그로 인해 압력방정식의 속도향상이 운동량 방정식보다 적었다.

(4) 본 연구에서 실험한 문제들의 경우에는 BIWO 예조건화 기법이 가장 좋은 성능을 보였으며, 풀고자 하는 문제의 조건수가 아주 큰 경우에는 MDLU-III 도 좋은 대안이 되리라 여겨진다.

(5) BIWO, MDLU 와 같은 병렬 예조건화 기법을 적용한 병렬 유한요소법 프로그램은 막대한 기억용량과 계산시간을 요구하는 대용량 비압축성 유동 문제를 효율적으로 해석하는 도구가 될 것이다.

후 기

이 연구는 차세대자동차기술개발사업, 한국과학기술정보연구원(KISTI)의 제 4 차 슈퍼컴퓨팅 응용연구 전략지원프로그램과 마이크로 열시스템 연구센터의 지원으로 수행되었으며 이에 감사드립니다.

참고문헌

(1) Basermann, A., Reichel, B. and Schelthoff, C., 1997, "Preconditioned CG methods for Sparse Matrices on Massively Parallel Machines," *Parallel Computing*, Vol. 23, pp. 381~398.

(2) Issman, E. and Degrez, G., 1997, "Non-overlapping Preconditioners for a Parallel Implicit Navier-Stokes Solver," *Future Generation Comput. Syst.*, Vol. 13, pp. 303~313.

(3) Wissink, A. M., Lyrintzis, A. S. and Chronopoulos, A. T., 1996, "Efficient Iterative Methods Applied to the Solution of Transonic Flows," *J. Comput. Phys.*, Vol. 123, pp. 379~393.

(4) Chronopoulos, A. T. and Wang, G., 1997, "Parallel Solution of a Traffic Flow Simulation Problem," *Parallel Computing*, Vol. 22, pp. 1965~1983.

(5) Magolu monga Made, M. and van der Vorst, H. A., 2001, "A Generalized Domain Decomposition Paradigm for Parallel Incomplete LU Factorization Preconditionings," *Future Generation Comput. Syst.* Vol. 17, pp. 925~932.

(6) Magolu monga Made, M. and van der Vorst, H. A., 2001,"Parallel Incomplete Factorizations with Pseudo-overlapped Subdomains," *Parallel Computing*, Vol. 27, pp. 989~1008.

(7) Saad, Y. and Sosonkina, M., 1999, "Distributed Schur Complement Techniques for General Sparse Linear Systems," *SIAM J. Sci. Comput.*, Vol. 21, pp. 1337~1356.

(8) Choi, H. G., Choi, H. and Yoo, J. Y., 1997, "A Fractional Four-step Finite Element Formulation of the Unsteady Incompressible Navier-Stokes Equations Using SUPG and Linear Equal-order Element Methods," *Comput. Methods Appl. Mech. Eng.*, Vol. 143, pp. 333~348.

(9) Radicati di Brozolo, G. and Robert, Y., 1989, "Parallel Conjugate Gradient-like Algorithms for Solving Sparse Nonsymmetric Linear Systems on a Vector Multiprocessor ," *Parallel Computing*, Vol. 11, pp. 223~239.

(10) Saad, Y., 1996, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, pp. 374~376.

(11) Van der Vorst, H. A., 1992, "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-symmetric Linear Systems," *SIAM J. Sci. Statist. Comput.*, Vol. 12, pp. 631-634.

(12) Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J., 1996, *MPI: The complete reference*, The MIT Press, London.

(13) http://www-users.cs.umn.edu/~karypis/metis.

(14) Carey, G. F., Shen, Y. and McLay, R. T., 1998, "Parallel Conjuate Gradient Performance for Least-Squares Finite Elements and Transport Problems," *Int. J. Numer. Meth. Fluids*, Vol. 28, pp. 1421~1440.

(15) Kershaw, D. S., 1978, "The Incomplete Cholesky-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations," *J. Comput. Phys.*, Vol. 26, pp. 43~65.

(16) Jiang, B. N, Lin, T. L. and Povinelli, L. A., 1994, "Large-scale Computation of Incompressible Viscous Flow by Least-squares Finite Element Method," *Comput. Methods Appl. Mech. Eng.*, Vol. 114, pp. 213~231.

(17) Fusegi, T., Hyun, J. M., Kuwahara, K. and Farouk, B., 1991, "A Numerical Study of Three-dimensional Natural Convection in a Differentially Heated Cubical Enclosure," *Int. J. Heat Mass Transfer*, Vol. 34, No.6, pp. 1543~1557.