修 士 論 文

分離型解法による非圧縮性流体解析に適した並列アルゴリズムの開発

指導教官　松澤照男 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

原田 隆

1997 年 2 月 14 日

目 次

第 1 章

はじめに

現在において、より現実的な問題を解くために大規模な数値解析が求められている。これまでの大規模な流体解析は差分法解析で行なわれてきた。しかし、海岸や湖沼などの潮流現象や、気象などの大気の循環、飛行機まわりの流れの解析、建築物などの問題は一般的に境界の形状が複雑なものが多い。一方、有限要素法による流体解析は、任意の有限要素（三角形、四角形、四面体、六面体など）による形状近似や構造解析との整合性がよいなどの利点がある。しかし、数値的、安定性、解析精度、計算速度などの点から、課題が残されている。最近、超並列コンピュータの発達により、大規模解析、複雑な現象の解析などが可能になってきている。そのため、有限要素法解析による大規模な流体解析の並列化の研究が進められている。

かつて鎌形 [1] は、流れ関数渦度法を用いた有限要素法の並列化を超並列計算機ＣＭ－５上でＳＩＭＤ型アルゴリズムで実現した。しかし、流れ関数渦度法を用いた解析法は３次元に拡張できない。さらに境界条件の取り扱いが流速と圧力を未知変数とする原始変数法の解析方法とくらべて、直接的ではなく複雑な与え方をするので最近の研究例が少ないのが現状である。

そこで、原始変数法のうち比較的に簡単に高い精度が得られる分離型解法を用いた。分離型解法は時間段階Δt を１ステップ進めるためにさらに３段階の計算過程に分解して計算する方法である。さらにこの解法は、非線形である移流項と非圧縮の連続の式を分けて解くため、比較的数値が安定して求められるなど利点がある。しかし、大規模な連立１次

方程式を３度解くことになるため、計算負荷が非常に高い。これを解消するために分離型解法に基づく有限要素解析法の並列化を行なう。基本的な有限要素解析法の並列化は鎌形が開発したＳＩＭＤ型のアルゴリズムを踏襲する。

現在、並列計算機上での連立方程式の解法は、反復法やブロック化したガウスの消去法など数多く議論されていて、ＣＭ－５上でもその成果として計算ライブラリなどに搭載されている。しかし、有限要素解析法で作られる大規模行列は、非対称疎行列である。そのうえ、要素の数値の大部分が０なためにインデックスを用いて行列を小さくすることになり、そのままでは計算ライブラリが使えないことが多い。そのため M.Behr[3] らや denĕk Johan[4] らによる有限要素法による非圧縮性粘性流体解析を並列計算機でおこなった研究ではＳＩＭＤ型プログラミングで連立方程式の解法を作成しているのが現状である。連立方程式の解法である直接法や前処理つき反復法をＳＩＭＤ型プログラミングで行なおうとすると、どうしても逐次処理の部分が残ってしまい、思ったほど効率が上がらないことが多い。

そこで、超並列計算機ＣＭ－５のプログラミングモデルの一つであるグローバル／ローカル・プログラミング [7] を採用する。グローバル／ローカル・プログラミングはＳＩＭＤ型プログラミング言語 CM-Fortran で作ったグローバルプログラムをメインルーチンとして動かし、ノードごとに制御でき、メッセージパッシングを行なうＭＩＭＤ型プログラムをサブルーチンとして呼び出すことで実現されている

本研究では、分離型解法による有限要素法解析の並列化をほどこし、グローバル／ローカル・プログラミングを検討することで有限要素法解析の高速化と効率化を試み、その有効性を確認する。

第 2 章

解析方法

非圧縮性粘性流体の２次元解析を有限要素法によって行なう。数値計算法としては、時間に関する分離型解法を採用し、この章でくわしく述べることする。

2.1 ２次元非圧縮性粘性流体の基本方程式

流れの基礎方程式は、次式のような Navier-Stokes 方程式と連続の式となる。

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0 \tag{2.1}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_i u_j) + \frac{\partial p}{\partial x_i} - \frac{\partial}{\partial x_i}\tau_{ij} = \rho \kappa_i \qquad (j = 1, 2, 3) \tag{2.2}$$

u_iは流体の速度、p は圧力、ρは流体の密度、κ_iは流体の単位質量当りに働く i 方向の外力である。τ_{ij}は粘性応力テンソルで次式によって定義される。

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k} \right) \tag{2.3}$$

ただし、μは粘性係数、δ_{ij}はクロネッカーのデルタ関数とする。

対象とする流体は、粘性係数 μが一定、外力は作用しないとし、さらに非圧縮性流体条件から、密度ρも一定とする。

これらより、式 (2.1) と式 (2.2) は次式になる。

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.4}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} - \frac{1}{Re} \frac{\partial}{\partial x_j} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) = 0 \qquad (j = 1, 2, 3) \tag{2.5}$$

ここでの、Re は無次元パラメータの Reynolds 数で、

$$Re = \frac{\rho L U}{\mu} \tag{2.6}$$

と表される。ただし、L、Uはそれぞれ代表長さと代表流速である。

また、基礎方程式に対する境界条件は、

$$u_i = \hat{u}_i \qquad on \Gamma_0 \tag{2.7}$$

$$\left\{ -p\delta_{ij} + \frac{1}{Re} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right\} \cdot n_j = \hat{t_i} \qquad on \Gamma_1 \tag{2.8}$$

で与えられる。

ここでの、n_iは境界上の外向きの法線ベクトルで、t_iは表面力である。本研究では、$\hat{t_i} = 0$ とする。

2.2 分離型解法

非定常非圧縮粘性流体の数値解析についての分離型解法はいくつか提案されている。ここで説明する分離型解法は、時間方向にΔt 進める際に３つの時間段階に分けることで、非線形項と非圧縮の連続の式とを分けて解く分離型解法の一つである。

2.2.1 θ-scheme による３段階時間分離法

次の微分方程式について考える。

$$\frac{d\varphi}{dt} + A(\varphi) = 0 \tag{2.9}$$

$$\varphi(0) = \varphi_0 \tag{2.10}$$

ただし、φ はスカラー関数とし、A は n 次元ベクトル空間の演算子で、

$$A = A_1 + A_2 \tag{2.11}$$

がなりたつとする。（ただし、A_1, A_2はそれぞれ A より簡単な演算子とする。）

ある$\theta(0 < \theta \leq \frac{1}{3}))$ について、φ^n を既知と仮定し、$\varphi^{n+\theta}$,$\varphi^{n+1-\theta}$,φ^{n+1}を時間進行法を一般化台形公式にのっとって計算すると、次式のようになる。

$$\frac{\varphi^{n+\theta} - \varphi^n}{\theta \Delta t} + A_1(\varphi^{n+\theta}) + A_2(\varphi^n) = 0 \tag{2.12}$$

$$\frac{\varphi^{n+1-\theta} - \varphi^{n+\theta}}{(1-2\theta)\Delta t} + A_1(\varphi^{n+\theta}) + A_2(\varphi^{n+1-\theta}) = 0 \tag{2.13}$$

$$\frac{\varphi^{n+1} - \varphi^{n+1-\theta}}{\theta \Delta t} + A_1(\varphi^{n+1}) + A_2(\varphi^{n+1-\theta}) = 0 \tag{2.14}$$

Edward J.Dean[5] によると、演算子 A を

$$A = \alpha A + \beta A, \tag{2.15}$$

$$\alpha + \beta = 1, \qquad 0 \leq \alpha \leq 1, 0 \leq \beta \leq 1 \tag{2.16}$$

としたとき、θ-scheme が二次の精度を持つならば、

$$\theta = 1 - \frac{1}{\sqrt{2}} \tag{2.17}$$

$$\alpha = 2 - \sqrt{2} \tag{2.18}$$

$$\beta = \sqrt{2} - 1 \tag{2.19}$$

を満たす。

これを Navier-Stokes 方程式と連続の式に応用する。

速度を陽的に、圧力を陰的に扱い、演算子 A に粘性項を代入すると次の式のようになる。

$$\begin{cases} \dfrac{\mathbf{u}^{n+\theta} - \mathbf{u}^n}{\theta \Delta t} - \dfrac{\alpha}{Re}\Delta \mathbf{u}^{n+\theta} + \nabla p^{n+\theta} = \dfrac{\beta}{Re}\Delta \mathbf{u}^n - (\mathbf{u}^n \cdot \nabla)\mathbf{u}^n \\ \\ \nabla \cdot \mathbf{u}^{n+\theta} = 0 \end{cases} \tag{2.20}$$

$$\frac{\mathbf{u}^{n+1-\theta} - \mathbf{u}^{n+\theta}}{(1-2\theta)\Delta t} - \frac{\beta}{Re}\Delta \mathbf{u}^{n+1-\theta} + (\mathbf{u}^{n+1-\theta} \cdot \nabla)\mathbf{u}^{n+1-\theta} = \frac{\alpha}{Re}\Delta \mathbf{u}^{n+\theta} - \nabla p^{n+\theta} \tag{2.21}$$

$$\begin{cases} \dfrac{\mathbf{u}^{n+1}-\mathbf{u}^{n+1-\theta}}{\theta\Delta t}-\dfrac{\alpha}{Re}\Delta\mathbf{u}^{n+1}+\nabla p^{n+1}=\dfrac{\beta}{Re}\Delta\mathbf{u}^{n+1-\theta}-(\mathbf{u}^{n+1-\theta}\cdot\nabla)\mathbf{u}^{n+1-\theta} \\ \\ \nabla\cdot\mathbf{u}^{n+1}=0 \end{cases} \tag{2.22}$$

ただし、$n \geq 0$ とし、

$$\mathbf{u}^0 = \mathbf{u}_0 \tag{2.23}$$

を満たす。

式（2.20）、式（2.22）を解くことは、結局ストークス流れの計算をすることと同じである。また、式（2.21）で非線形項である移流項を使ってを計算し時間を進める。このように連続の式と移流項を別々に計算できることでき、数値が比較的に安定して求められるのが、分離型解法の利点である。本来は、$(\mathbf{u}^{n+1-\theta}\cdot\nabla)\mathbf{u}^{n+1-\theta}$をニュートン・ラプソン法などで近似するのが通例であるが、本研究では、定常流れを解析するので、$(\mathbf{u}^{n+\theta}\cdot\nabla)\mathbf{u}^{n+1-\theta}$として計算する。

第 3 章

有限要素法に基づく離散化

先の解法のアルゴリズムに示したとおり、解かなければならない方程式は３つある。そこで、これらの式の空間微分項を有限要素近似する。

3.1 重み付き残差方程式

まず、重み付き残差方程式を求める。

$u_i^*(x,y)$ を領域Ωの境界$\partial\Omega$で 0 であるような任意の滑らかな関数とする。同様に $p^*(x,y)$ も任意の滑らかな関数とする。これらを重み関数という。式（2.20）ー式（2.22)、にそれぞれに重み関数を掛けて領域Ωで積分すると：

$$\frac{1}{\theta\Delta t}\int_\Omega u_i^*(u_i^{n+\theta}-u_i^n)d\Omega-\frac{\alpha}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+\theta}}{\partial x_j}+\frac{\partial u_j^{n+\theta}}{\partial x_i}\right)d\Omega+\int_\Omega u_i^*\frac{\partial p}{\partial x_i}d\Omega$$
$$=\frac{\beta}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^n}{\partial x_j}+\frac{\partial u_j^n}{\partial x_i}\right)d\Omega-\int_\Omega u_i^*u_j^n\frac{\partial u_i^n}{\partial x_j}d\Omega\qquad(j=1,2,3)\qquad(3.1)$$

$$\int_\Omega p^*\frac{\partial u_i^{n+\theta}}{\partial x_i}d\Omega=0\qquad(3.2)$$

$$\frac{1}{(1-2\theta)\Delta t}\int_\Omega u_i^*(u_i^{n+1-\theta}-u_i^{n+\theta})d\Omega-\frac{\beta}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+1-\theta}}{\partial x_j}+\frac{\partial u_j^{n+1-\theta}}{\partial x_i}\right)d\Omega$$
$$+\int_\Omega u_i^*u_j^{n+1-\theta}\frac{\partial u_i^{n+1-\theta}}{\partial x_j}d\Omega$$
$$=\frac{\beta}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^n}{\partial x_j}+\frac{\partial u_j^n}{\partial x_i}\right)d\Omega-\int_\Omega u_i^*\frac{\partial p}{\partial x_i}d\Omega\qquad(j=1,2,3)\qquad(3.3)$$

$$\frac{1}{\theta\Delta t}\int_\Omega u_i^*(u_i^{n+1}-u_i^{n+1-\theta})d\Omega-\frac{\alpha}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+1}}{\partial x_j}+\frac{\partial u_j^{n+1}}{\partial x_i}\right)d\Omega+\int_\Omega u_i^*\frac{\partial p}{\partial x_i}d\Omega$$
$$=\frac{\beta}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+1-\theta}}{\partial x_j}+\frac{\partial u_j^{n+1-\theta}}{\partial x_i}\right)d\Omega-\int_\Omega u_i^*u_j^{n+1-\theta}\frac{\partial u_i^{n+1-\theta}}{\partial x_j}d\Omega \qquad (3.4)$$

$$\int_\Omega p^*\frac{\partial u_i^{n+1}}{\partial x_i}d\Omega=0 \qquad (3.5)$$

次に、式（3.1）についてのみ考える。

ガウス-グリーンの定理より、式（3.1）の左辺第二項、左辺第三項、右辺第一項はそれぞれ、

$$\begin{aligned}\frac{\alpha}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+\theta}}{\partial x_j}+\frac{\partial u_j^{n+\theta}}{\partial x_i}\right)d\Omega &= \frac{\alpha}{Re}\oint_{\partial\Omega}(\frac{\partial u_i^{n+\theta}}{\partial x_k}m_ku^*)dS-\frac{\alpha}{Re}\int_\Omega(\frac{\partial u_i}{\partial x_k}\frac{\partial u^*}{\partial x_k})d\Omega\\ &= -\frac{\alpha}{Re}\int_\Omega(\frac{\partial u_i}{\partial x_k}\frac{\partial u^*}{\partial x_k})d\Omega \qquad (3.6)\end{aligned}$$

$$\begin{aligned}\int_\Omega\frac{\partial p}{\partial x_i}u^*d\Omega &= \oint_{\partial\Omega}pmu^*dS-\int_\Omega p\frac{\partial u^*}{\partial x_i}d\Omega\\ &= -\int_\Omega p\frac{\partial u^*}{\partial x_i}d\Omega \qquad (3.7)\end{aligned}$$

$$\begin{aligned}\frac{\beta}{Re}\int_\Omega u_i^*\frac{\partial}{\partial x_j}\left(\frac{\partial u_i^{n+\theta}}{\partial x_j}+\frac{\partial u_j^{n+\theta}}{\partial x_i}\right)d\Omega &= \frac{\beta}{Re}\oint_{\partial\Omega}(\frac{\partial u_i^{n+\theta}}{\partial x_k}m_ku^*)dS-\frac{\beta}{Re}\int_\Omega(\frac{\partial u_i}{\partial x_k}\frac{\partial u^*}{\partial x_k})d\Omega\\ &= -\frac{\beta}{Re}\int_\Omega(\frac{\partial u_i}{\partial x_k}\frac{\partial u^*}{\partial x_k})d\Omega \qquad (3.8)\end{aligned}$$

となる。

m_iは境界上の x_i方向の外向きの単位法線ベクトル$\vec{n}$の方向余弦で $cos(\vec{n},x_i)$ とする。上式の境界上での積分は 境界$\partial\Omega$で区分的になめらかであるという条件から、0となる。

次に関数を基底関数に置き換えて、離散化を行なう。流速 u_i圧力 p、重み関数 u^*をそれぞれ次のように置き換える。

$$u \approx \sum_{\mu=1}^{N_u}u_\beta\phi_\beta \qquad (3.9)$$

$$p \approx \sum_{\mu=1}^{N_p} p_\mu \psi_\mu \tag{3.10}$$

$$u^* \approx \sum_{\alpha=1}^{N_u} u^*_\alpha \phi_\alpha \tag{3.11}$$

ただし、N_u, N_pは、x 方向の流速と圧力の自由度の数を表し、それぞれの基底関数$\psi_\mu, \phi_\alpha, \phi_\alpha$は、以下についてみたす。

$$\begin{cases} \psi_i = 1 \ （番号\, i \,の節点で) \\ \psi_i = 0 \ （他の節点で) \end{cases}$$

よって式（3.1）は、次式のようになる。

$$\begin{aligned} u^{n+\theta}_{\beta i} \int_{\Omega_e} N_\alpha N_\beta d\Omega + \theta \Delta t \left(u^{n+\theta}_{\beta i} \frac{\alpha}{Re} \int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_i} \frac{\partial N_\beta}{\partial x_i} d\Omega \cdot + \delta_{ij} \int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_j} \frac{\partial N_\beta}{\partial x_i} d\Omega \right) - \theta \Delta t p_\mu \int_\Omega \psi_\mu \frac{\partial \phi_\alpha}{\partial x} d\Omega \\ = u_{\beta i} \int_{\Omega_e} N_\alpha N_\beta d\Omega - \theta \Delta t \frac{\beta}{Re} \left(\int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_i} \frac{\partial N_\beta}{\partial x_i} d\Omega \cdot + \delta_{ij} \int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_j} - \frac{\partial N_\gamma}{\partial x_j} d\Omega u^n_{\gamma i} \right) \\ - \theta \Delta t u^{n+1}_{\beta j} \int_{\Omega_e} N_\alpha N_\beta \frac{\partial N_\gamma}{\partial x_j} d\Omega u^n_{\gamma i} \end{aligned} \tag{3.12}$$

行列表示すると、

$$\begin{pmatrix} M_{\alpha\beta} + S_{,x} & 0 & -H_{\alpha\beta,x} \\ 0 & M_{\alpha\beta} + S_{,y} & -H_{\alpha\beta,y} \\ H_{\alpha\beta,x} & H_{\alpha\beta,y} & 0 \end{pmatrix} \begin{pmatrix} u^{n+\theta}_\beta \\ v^{n+\theta}_\beta \\ p^{n+\theta}_\mu \end{pmatrix} = \begin{pmatrix} u^n_\beta (M_{\alpha\beta} - S_{,x}) - u^{n+1}_{\beta j} K_{\alpha\beta\gamma j} \\ u^n_\beta (M_{\alpha\beta} - S_{,y}) - u^{n+1}_{\beta j} K_{\alpha\beta\gamma j} \\ 0 \end{pmatrix} \tag{3.13}$$

ただし、

$$M_{\alpha\beta} = \int_{\Omega_e} N_\alpha N_\beta d\Omega, \quad H_{\gamma\beta,i} = \int_{\Omega_e} N_\gamma \frac{\partial N_\beta}{\partial x_i} d\Omega \tag{3.14}$$

$$S_{\alpha\beta,i} = \int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_i} \frac{\partial N_\beta}{\partial x_i} d\Omega \cdot + \delta_{ij} \int_{\Omega_e} \frac{\partial N_\alpha}{\partial x_j} \frac{\partial N_\beta}{\partial x_i} d\Omega \tag{3.15}$$

$$K_{\alpha\beta\gamma j} = \int_{\Omega_e} N_\alpha N_\beta \frac{\partial N_\gamma}{\partial x_j} d\Omega u^n_{\gamma i} \tag{3.16}$$

よって、基礎方程式から有限要素方程式が得られた。

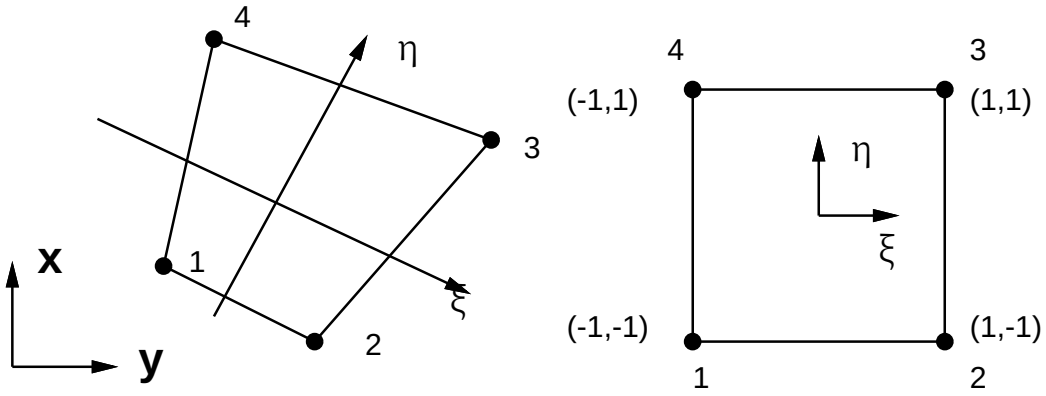


図 3.1:

3.2 アイソパラメトリック要素

使用する有限要素はアイソパラメトリック要素とする。

図（3.1）の左図に示す任意の４節点四角形を考える。

まず、内部の変数を４節点の変数により１次の多項式で近似すると、任意の点でのϕは次のように表現することができる。

$$\phi = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 xy \tag{3.17}$$

座標変換することで図（3.1）の右図の正方形に変換される。

座標変換したことで、式（3.17）は次のように表される

$$\phi = \alpha_0 + \alpha_1 \xi + \alpha_2 \psi + \alpha_3 \xi\psi \tag{3.18}$$

式（3.18）に図（3.1）の右図がしめす四角形の４節点の条件を代入すると

$$\phi = N_1(\xi,\psi)\phi_1 + N_2(\xi,\psi)\phi_2 + N_3(\xi,\psi)\phi_3 + N_4(\xi,\psi)\phi_4 \tag{3.19}$$

ただし、上式での係数は次式のようになる。

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$
$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$
$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$
$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$

よって、積分計算はガウスの求積法でおこなう。

$$\inf_A (N_\alpha N_\beta) dA = \sum_{i=1}^{n}\sum_{j=1}^{n} N_\alpha(\xi_i,\psi_j) \cdot N_\beta(\xi_i,\psi_j) det[\mathbf{J}] W_i W_j \tag{3.20}$$

ここで、(ξ_i, ψ_j) は積分点、n は積分点の数、$W_i W_j$は重み関数を表す。

次に、変数変換のための準備をする。x, y座標を $x = x(\xi,\eta), y = y(\xi,\eta)$ とすると、関数ψに対する一般的な変換は次のようになる。

$$\frac{\partial\psi}{\partial x} = \frac{\partial\psi}{\partial\xi}\frac{\partial\xi}{\partial x} + \frac{\partial\psi}{\partial\eta}\frac{\partial\eta}{\partial y} \tag{3.21}$$
$$\frac{\partial\psi}{\partial y} = \frac{\partial\psi}{\partial\xi}\frac{\partial\xi}{\partial y} + \frac{\partial\psi}{\partial\eta}\frac{\partial\eta}{\partial y} \tag{3.22}$$

通常、ξ, ηを x, yで表す式は簡単に得られないので、

$$\frac{\partial\psi}{\partial x} = \frac{1}{|\mathbf{J}|}\left(\frac{\partial y}{\partial\eta}\frac{\partial\psi}{\partial\xi} - \frac{\partial y}{\partial\xi}\frac{\partial\psi}{\partial\eta}\right) \tag{3.23}$$
$$\frac{\partial\psi}{\partial y} = \frac{1}{|\mathbf{J}|}\left(\frac{\partial x}{\partial\xi}\frac{\partial\psi}{\partial\eta} - \frac{\partial x}{\partial\eta}\frac{\partial\psi}{\partial\xi}\right) \tag{3.24}$$

ただし、$|\mathbf{J}|$ は

$$|\mathbf{J}| = \frac{\partial x}{\partial\xi}\frac{\partial y}{\partial\eta} - \frac{\partial x}{\partial\eta}\frac{\partial y}{p}\partial\xi$$

アイソパラメトリック要素の定義から、形状関数は次の条件を満足する。

$$\sum_{j=1}^{4} N_i(\xi,\eta) = 1 \tag{3.25}$$
$$\sum_{j=1}^{4} N_i(\xi,\eta) x_i = x \tag{3.26}$$
$$\sum_{j=1}^{4} N_i(\xi,\eta) y_i = y \tag{3.27}$$

よって、

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \cdots & \frac{\partial N_n}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \cdots & \frac{\partial N_n}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \cdots & \cdots \\ x_n & y_n \end{bmatrix} \tag{3.28}$$

以上で、全ての有限要素方程式が計算できる。

第 4 章

グローバル／ローカル・プログラミング

グローバル・ローカルプログラミング [7] は超並列計算機CM－５のプログラミングモデルの一つである。ＳＩMD型プログラミングはＭＩMD型プログラミングに比べ、プログラミングが比較的容易である一方、プロセッサ単位でのプログラミングやプロセッサ間のメッセージパッシングなどの緻密な制御が困難である。そこで、ＳＩMD型のメインプログラムからサブルーチンとしてＭＩMD型プログラムを呼び出し実行する方法で、ハイブリットな並列プログラミングが実現されている。

本研究では、このプログラミングモデルを有限要素法に応用する。またプログラミング言語は、ＳＩMD型プログラミング言語CM Fortran とメッセージ・パッシング・ライブラリCMMDの組合せて使用する。

4.1　グローバル／ローカル・プログラミングの概要

CM－５などの超並列計算機でのＳＩMD型プログラミングは、多くのプロセッサから構成されている並列計算機であっても１プロセッサの計算機を扱うような「グローバル」な考えで、プログラミングを行なう。一方、ＭＩMD型プログラミングは個々のプロセッサに対する「ローカル」なプログラムを作成し、個々のプログラムを別々に実行する考えで、プログラミングを行なう。これらを組み合わせてＳＩMD型プログラミングにメッセージ・パッシングのプログラムテクニックを合わせたハイブリットなプログラミングモデルをグローバル／ローカル・プログラミングとここでは呼ぶものとする。

具体的に、図（4.1）に示したようにグローバル／ローカル・プログラムは、必ずＳＩ

MD型プログラミング言語 CM-Fortran で作ったグローバルプログラムをメインルーチンとして動かす。そして、あるいくつかの場所で、ノードごとに制御するためのメッセージパッシング・ライブラリ CMMD を使用したローカルプログラムをサブルーチンとして呼び出し、ローカルプログラムを実行する。ローカルプログラム終了のあと、再びグローバルプログラムの戻ることで実現されている。

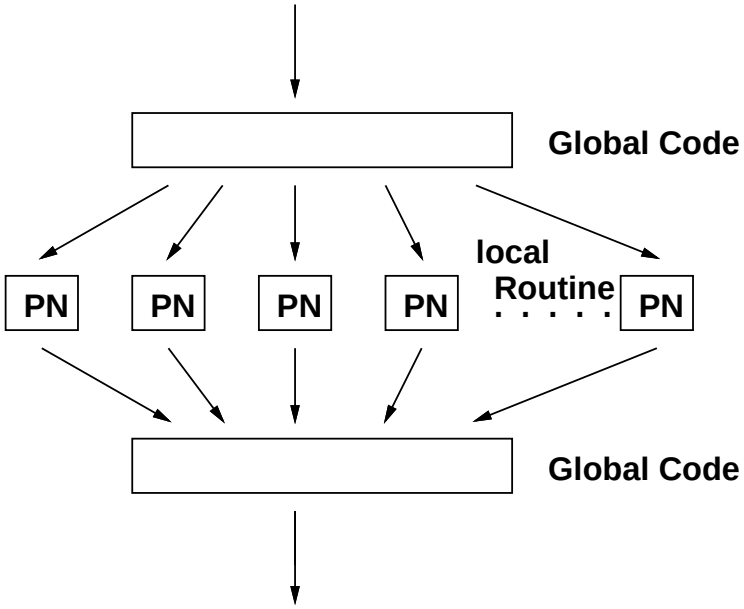


図 4.1:

4.2 グローバル／ローカル・プログラミングでの配列

具体的なプログラムの話の前にグローバル／ローカル・プログラミングでの配列について説明する。グローバル／ローカル・プログラミングでは、４種類の配列がある。

1. グローバル parallel 配列：この配列はグローバルプログラムで宣言される parallel 配列
2. グローバル serial 配列：この配列はグローバルプログラムで宣言される serial 配列
3. ローカル parallel 配列：この配列はローカルプログラムで宣言される parallel 配列

4. ローカル serial 配列：この配列はローカルプログラムで宣言される serial 配列。

1.2. の配列は、いわゆる通常のＳＩＭD型プログラムで使われる配列である。一方、3.4 配列と呼ばれるものはＭＩＭD型プログラムで使われる配列である。グローバル／ローカル・プログラミングでは、グローバルプログラムから、ローカルプログラムに移る時、グローバル＊＊配列の扱いが変わる。

グローバル parallel 配列の場合は、ローカルプログラムに移った時、もともと割り振られていたノードごとに分割され、そのノードプロセッサが扱っている一部分の配列のみに対して、命令が実行される。

例えば、配列の値をずらす CSHIFT 命令をグローバルプログラムで実行した場合、図（4.2）の真中の図で示したように配列要素が一つずつずれ、一番最後の配列要素が最初にくる。しかし、ローカルプログラムで最後の図のようには、各ノードごとの一番最後の配列要素が最初にくることになる。

グローバル parallel 配列のどの要素がどこのノードに配置するかは、LAYOUT 命令で、決めることができる[1] また、ローカルプログラム下で、グローバル parallel 配列のどの要素がどこのノードに配置されているかを知るライブラリも用意されている。

一方、グローバル serial 配列は、グローバルプログラムの時 front-end processor におかれている。この配列をローカルプログラムで使う場合、あらかじめライブラリで front-end processor から、各ノードへメッセージ・パッシングする必要がある。通常、グローバルプログラムからローカルプログラムの切替えは数マイクロ秒以内ですむが、グローバル serial 配列を転送する場合、serial 配列の大きさに比例して時間がかかってしまうので注意が必要である。また、グローバル serial 配列はベクトル演算ユニット上で演算されない。

4.3 グローバル／ローカル･プログラミングのパフォーマンス

グローバルローカル・プログラミングの例として、比較的簡単で数値計算上も重要なものとして、行列・ベクトル積と行列の乗算を取り上げる。

4.3.1 行列・ベクトル積

行列・ベクトル積には以下の方法がある。

[1] 各ノードあたり、$n \times m$ の行列を作る命令である

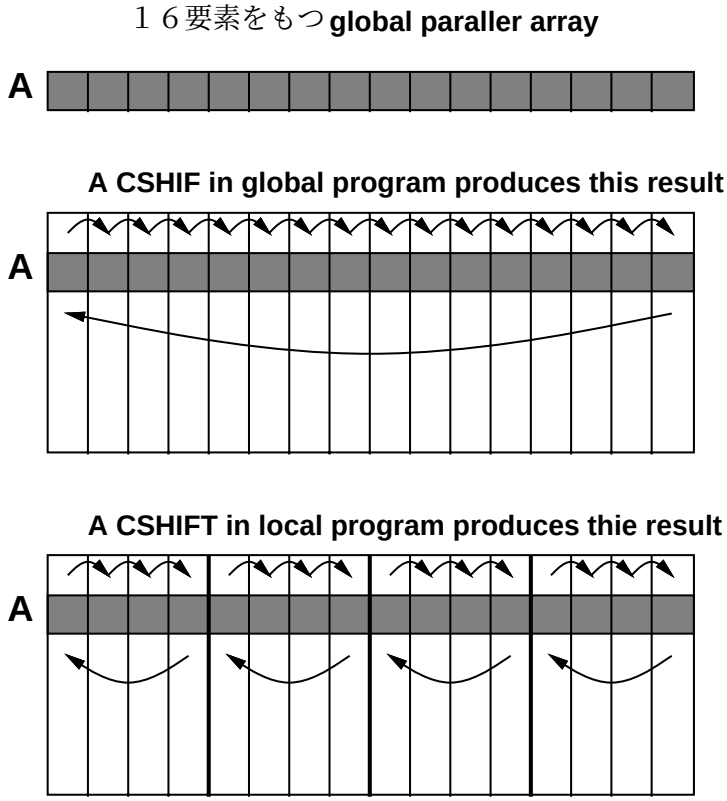


図 4.2:

- CM-Fortran の MATMUL 関数を使う。Aが２次、B, Cが１次配列だとすると、C＝MATMUL（A, B）となる。
- 並列化構文を使用して、行列・ベクトル積のコードを書く。
- CMMD を利用したローカルプログラム作成し、サブルーチンとして使用する。

今回は、倍精度の行列・ベクトル積演算プログラムを、並列化構文を使用して作ったＳＩＭＤ型プログラムとグローバル／ローカル・プログラムで、作成した。グローバル／ローカルプログラムでは、あらかじめグローバル・プログラムで行列Ａとベクトルｃをグローバル parallel 配列として宣言し、配列Ａは図（4.3）のように配置した。一方、ベクトルＢはグローバル serial 配列として宣言し、グローバルプログラムからローカルプログラムに切替える前にメッセージパッシングで全ノードにデータを転送し、ベクトル演算ユニットで演算されるように、ベクトルＢを parallel 配列に変換した後、行列・ベクトル積

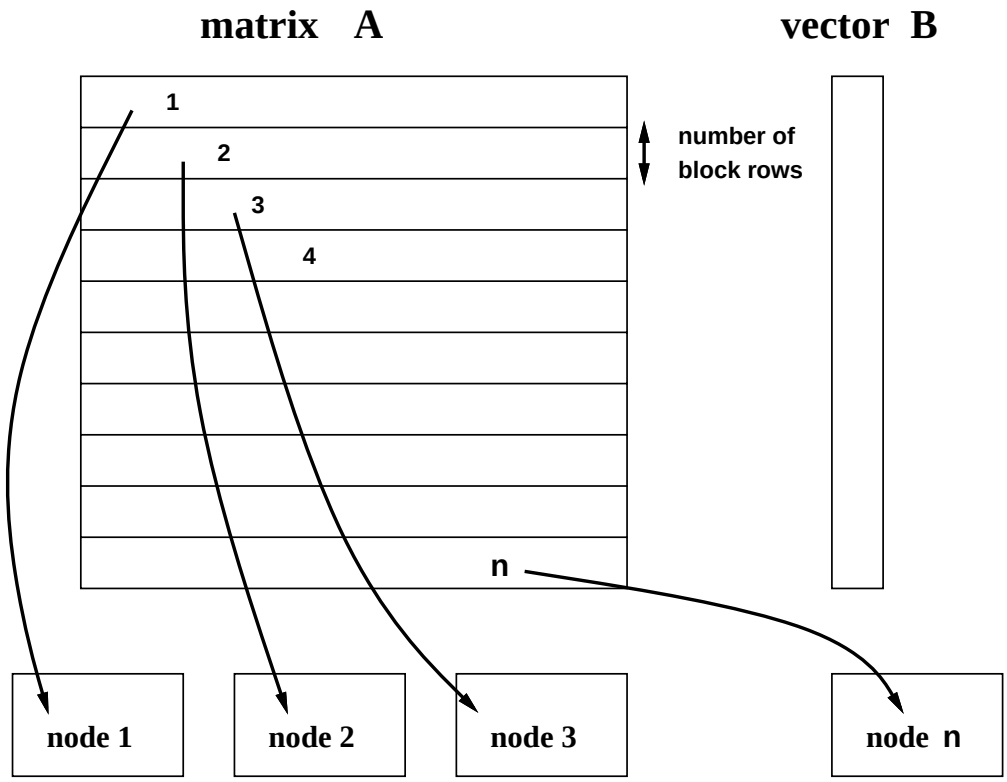


図 4.3: 全体行列分割

を実行した。また、具体的な数値はグローバルプログラム上で予め与えておいた。実行時間の計測は切り替わる直前、つまりベクトルBのデータを全ノードにメッセージパッシングする前から行なうことする。

また、配列Aの配置はＳＩMD型プログラムとグローバル／ローカル・プログラムでは、共に同じとする。ベクトルBについては、ＳＩMD型プログラムでは parallel 配列、グローバル／ローカル・プログラムでは、グローバル serial 配列とした。

結果は図（4.3.1）で示した。これを見ると、行列の次元数が増えてくるにつれてグローバル／ローカル・プログラムの実行時間は一次に比例し、ＳＩMD型プログラムとの実行時間は２次に比例していることが分かる。図（4.3.1）には示していないが、メッセージパッシングしないＭＩMD型プログラムもＳＩMD型プログラムの実行時間と同様な比例をし、それほど差がないことが分かっている。これらのことから、ＳＩMD型とグローバル／ローカルの実行時間の差はベクトルBを各ノードへメッセージ・パッシングの処理時間と parallel 配列への変換時間と考えれる。また、グローバル／ローカル・プログラムの

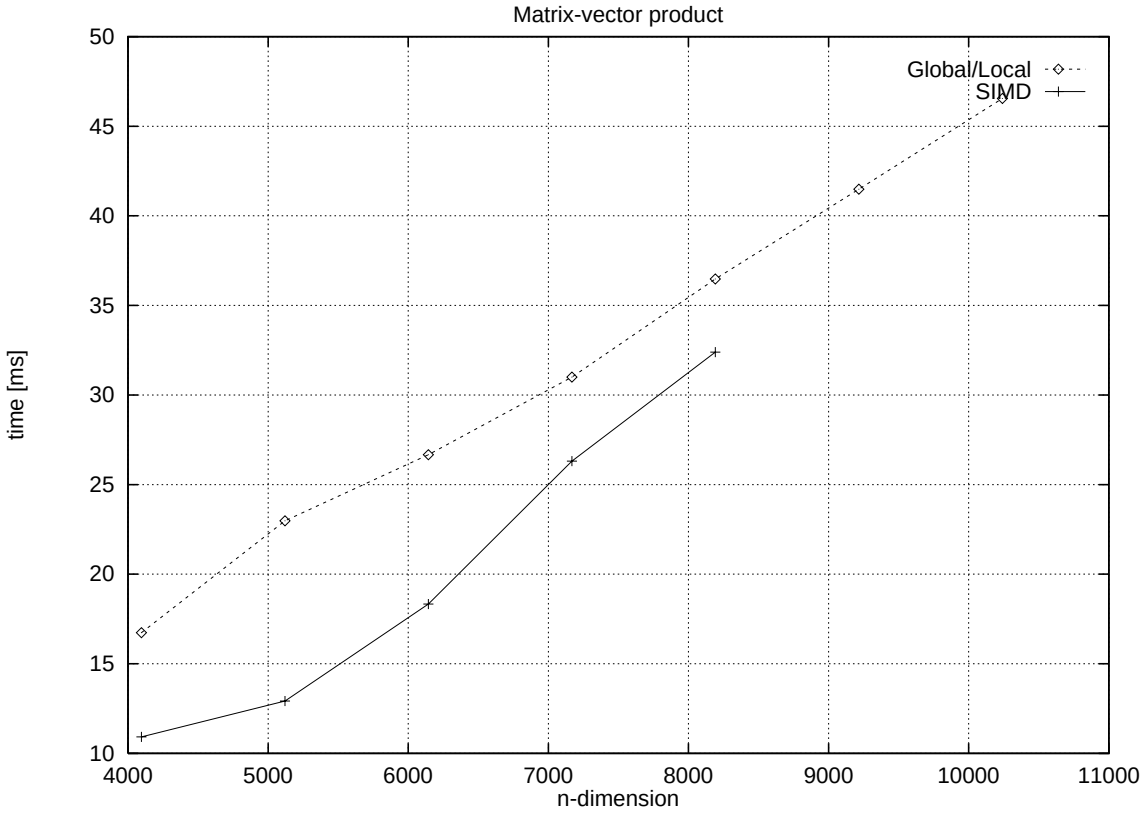


図 4.4: 行列・ベクトル積の実行時間

方がより大きい行列・ベクトル積演算ができることがわかった。

4.3.2 行列の乗算

行列の乗算には以下の方法がある。

- CM-Fortran の MATMUL 関数を使う。A, B, Cが２次配列だとすると、C＝MATMUL（A, B）となる。
- 並列化構文を使用して、行列の乗算のコードを書く。
- CMMD を利用したローカルプログラム作成し、サブルーチンとして使用する。配列A, B, Cは各ノードに分割配置し、規則的なメッセージ・パッシングを行なうことで、行列の乗算を実行する。

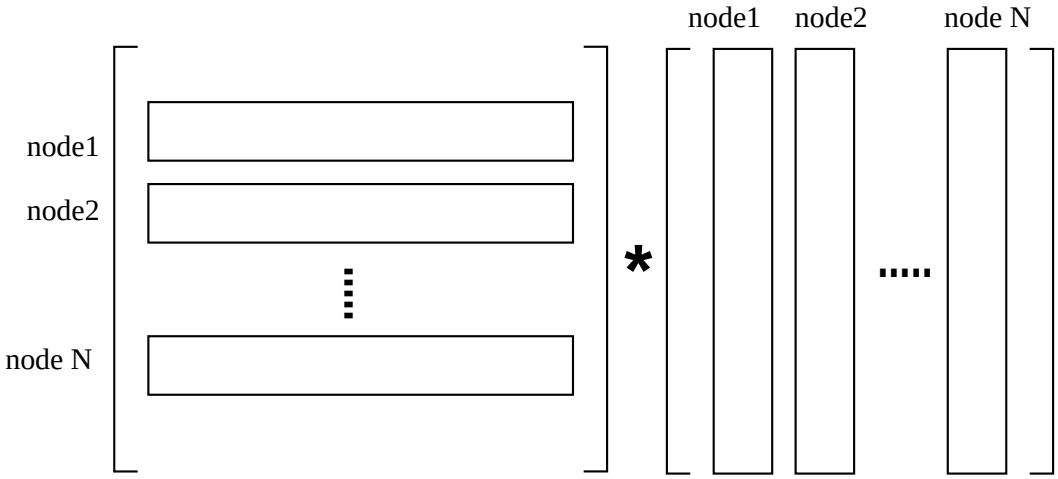


図 4.5: 行列分割

今回は、MATMUL 関数とグローバル／ローカル・プログラムで倍精度の行列乗算を作成した。グローバル／ローカル・プログラムでは、配列A, B, Cをあらかじめグローバルプログラム上で、グローバル parallel 配列として宣言し、配列A、配列Bをそれぞれ図（4.5）のように各ノードへ分割して配置し、具体的な数値も代入しておく。そして、ローカルプログラム上で、各ノードごとで小さい帯状ブロック行列の乗算と合計値への加算が実行され、そのあと配列Bを分割分ごとに、となりのノードプロセッサにメッセージ・パッシングを行ない、それをくりかえすことで行列の乗算を実現させた。また、配列の配置はＳＩMD型プログラムとグローバル／ローカル・プログラムとで同じにした。結果は、図（4.6）に示した。

具体的に演算速度を計算してみると、n=4096 の場合、MATMUL 関数は 834Mflops、グローバル／ローカル・プログラムでは、1621.2Mflops となって、MATMUL 関数より高速になった。（CM－５の理論演算速度は、163Gflops）この実験にかぎり、事実上グローバル／ローカル・プログラムとＭＩMD型プログラムと同じものだと考えられる。配列への数値代入はグローバルプログラムで行なったが、そのあとの処理はローカルプログラムとＭＩMD型プログラムの処理は変わらない。実際、実行時間は図には示していないがほとんど差はない。この実験でもちいたグローバル／ローカル・プログラムでの行列の乗算アルゴリズムは志田 [10] がおこなった CSHIFT 命令をつかった行列の乗算アルゴリズムとほぼ同じと思われる。また、志田のと比べると、まだ 1 割ほど速い。これはノードごと

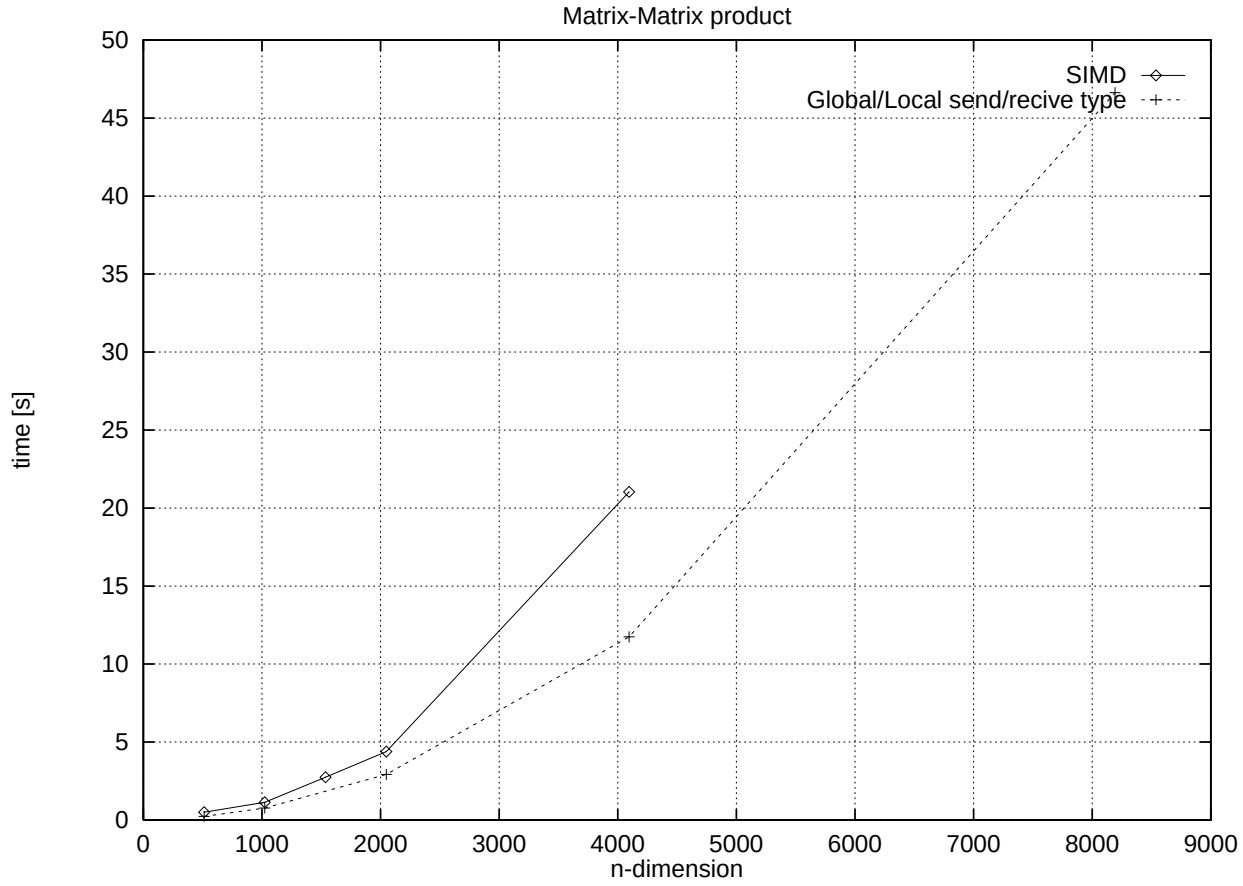


図 4.6: 実行時間

の待ち時間がほとんどなく一括してメッセージ・パッシングがおこなわれたためと思われる。これは、メッセージ・パッシングが仮想プロセッサ[2]ごとに行なわれのではなく、実プロセッサごとに行なわれることで、ノードごとの待ち時間が、削減できたと考える。ちなみに、グローバル／ローカルプログラムにした場合、n=8192 まで計算でき、メモリの有効利用ができるようになった。

[2] CM－５では配列要素一つに対して、プロセッサを割り当てることでＳＩＭＤを実現している。実プロセッサ 1 個あたりに仮想プロセッサ $4n$ 個を模擬している。配列の大きさ＝仮想プロセッサ数が半端な値のときでも、ＳＩＭＤプログラムならば、NOP にしたり待ち時間にすることで解決されていると思われる。しかし、グローバル／ローカル・プログラム上でのグローバル parallel 配列は、半端な値がゆるされず、配列要素の数は必ず $8n$ 個でなくてはならない。これは、ベクトル・ユニットに配列が配置され、小型並列マシンとしてあつかわれているためである。

以上、実験からグローバルローカル・プログラミングはＳＩMD型プログラミングに比べると次の特徴が上げられる。

- ＳＩMD型で難しかった並列化が、ローカルなプログラムを作成することで、より並列化が可能になる。
- メッセージ・パッシングをユーザーが制御できるため、ノードの待ち時間を減らせることが可能になる。
- 大きなグローバル serial 配列をローカルプログラムで使おうとすると、データを転送するための時間かかる。
- ＳＩMD型プログラムと比べるとメモリ使用量が少なくなる。

特にCM－５の場合、メモリの使用量はプログラム実行時に自動的に取得される作業領域の存在など種々の避けられない事情から配列の大きさから単純に割り出すことができない。

特に配列の場合、配列の各要素ごとに「仮想プロセッサ」が用意され、要素ごとに並列に操作される。さらに配列に対する操作がかならずしも並列にできないので、待ちなどの処理のために仮想プロセッサの処理内容が大きくなるため、大規模配列が取れにくくなったり、ベクトルユニットが効果的に働かなったりする[3]。有限要素解析を行なう場合、その点からも、グローバル／ローカル・プログラミングは有理になると考えられる。

[3]CM-5 では、各ノードに４つのベクトルユニットがあり、その働きによって実行時間が著しく変わる

第 5 章

有限要素法の並列化とグローバル／ローカル・プログラミングへの応用

全体の計算の流れは、式（2.20）－式（2.22）の３つの連立方程式を順番に解くことにより時間を進める。計算ステップの流れ図は次に示す。（定常流れを準非定常流れとして解析する場合）

1. 初期設定
2. 要素行列の計算
3. 要素行列の全体行列への重ね合わせ
4. 全体行列へ境界条件の設定
5. 時間進行ステップ
   (a) 前時間ステップの結果から、一段階めの右辺ベクトルの作成
   (b) 一段階めの連立方程式の解法
   (c) 一段階めの結果から、２段階めの全体行列と右辺ベクトルの作成
   (d) 二段階めの連立方程式の解法
   (e) 二段階めの結果から、三段階めの右辺ベクトルの作成
   (f) 三段階めの連立方程式の解法

(g) 収束条件の判定

6. 終了処理

今回の実験モデル（図5）では、有限要素の計算格子の時間的に変化がない。そこで、式（2.20）と式（2.22）についての全体行列Aは有限要素の計算格子と初期条件で決定されるので、一度だけの計算で済む。

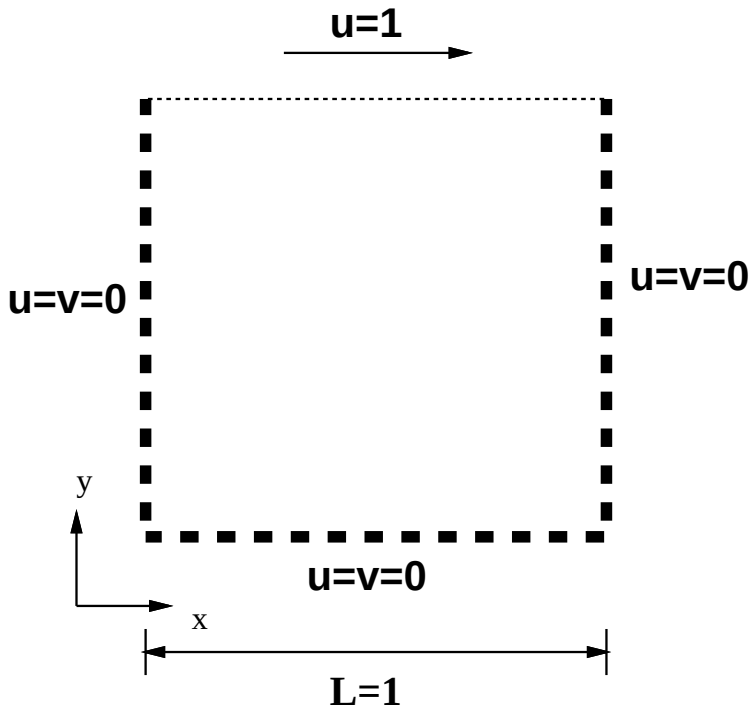


図 5.1: 実験モデル

本研究は、分離型解法の並列化を試み、特に、

- 要素行列の全体行列への重ね合わせのＳＩMD型プログラミングによる並列化と、グローバル／ローカルプログラムによる並列化
- 連立方程式の解法である SOR 法のＳＩMD型プログラミングによる並列化と、グローバル／ローカルプログラムによる並列化

を試みた。ＳＩMD型プログラムとグローバル／ローカルプログラムとの違いを考察し、分離型解法による有限要素法でのグローバル／ローカルプログラムの効果を検討する。

5.1 要素行列の全体行列への重ね合わせのＳＩＭＤ型プログラミングによる並列化と、グローバル／ローカルプログラムによる並列化について

第３章で求めた有限要素方程式は、有限要素内における方程式であり、これをこのまま解いても全体の解析領域を解いたことにならない。各要素の有限要素方程式から解析領域全体への連立方程式への作りかえを「全体行列への重ね合わせ」と呼ばれている。

5.1.1 ＳＩＭＤ型プログラミングでの全体行列重ね合わせ

全体行列への重ね合わせは、基本的に gather/scater 処理である。この処理の並列化は工夫をしないと通信衝突がおこり、正しい結果が得られない。具体的に三節点三角形要素での右辺ベクトルの重ね合わせを図（5.1.1）のような簡単な格子を例にあげて説明する。

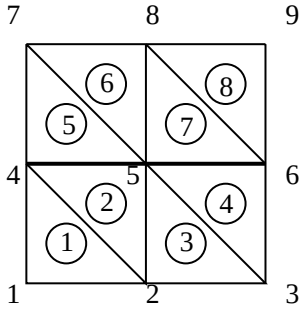


図 5.2: 有限要素の例

次の逐次プログラムの場合、

```
      do 100 j=1,node
         do 200 k=1,3
            ip=i(j,k)
            y(ip) = y(ip) + x(k)
200      continue
```

要素番号	配列 I		
k	i(1,k)	i(2,k)	i(3,k)
1	1	2	4
2	5	4	2
3	2	3	5
4	6	5	3
5	4	5	7
6	8	7	5
7	5	6	8
8	9	8	6

表 5.1: 要素番号と節点番号の関係を示す配列 I

```
100  continue
```

ここで、node は要素数で、i(j,k) は要素 j の節点 k の節点番号を入っている配列、x は節点 k での数値が収められた配列とする。

これを単純に CM-Fortan の Forall 文で並列化をすると、次のようになる。

```
      do 100 k=1,3
         forall(j=1:node)b(i(j,k))=b(i(j,k))+s(j,k)
100   continue
```

上記のプログラムでの forall 文による並列化には問題がある。表のような要素と節点の関係があるとすると、 i(1,k) についての forall 文の処理を図（5.3）にしめした。図をみてわかるように Y(5) に通信衝突がおこり、値が保証されないことが分かる。[1]

[1]実際には、２０か７０のどちらかの値が入る。実行時にエラーが出てもよさそうだが、でない

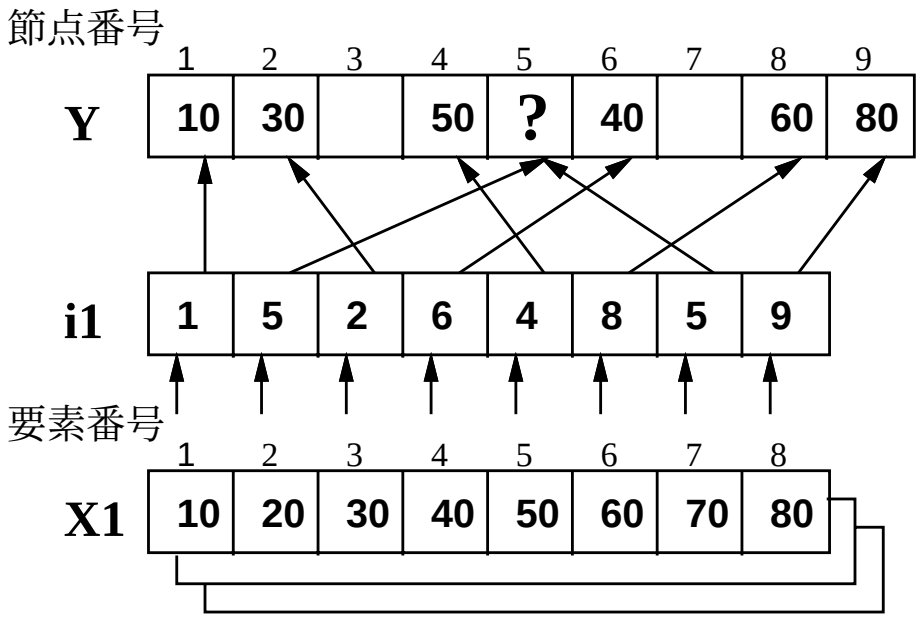


図 5.3: 通信衝突例

以上のような問題をさけるために図（5.1.1）のように２色に分類する。これを元に重ね合わせを２段階にすることで図（5.1.1）のように通信衝突をさけることができる。

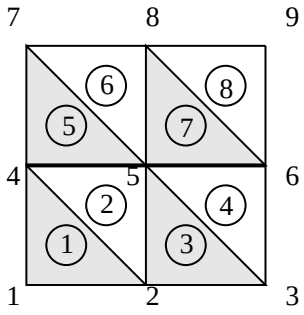


図 5.4: 要素を２色に分類した例

つまり、 下の例のようにプログラムを組む。

```
      do 100 k=1,3
         forall(j=1:node:2)b(i(j,k))=b(i(j,k))+s(j,k)
         forall(j=2:node:2)b(i(j,k))=b(i(j,k))+s(j,k)
100   continue
```

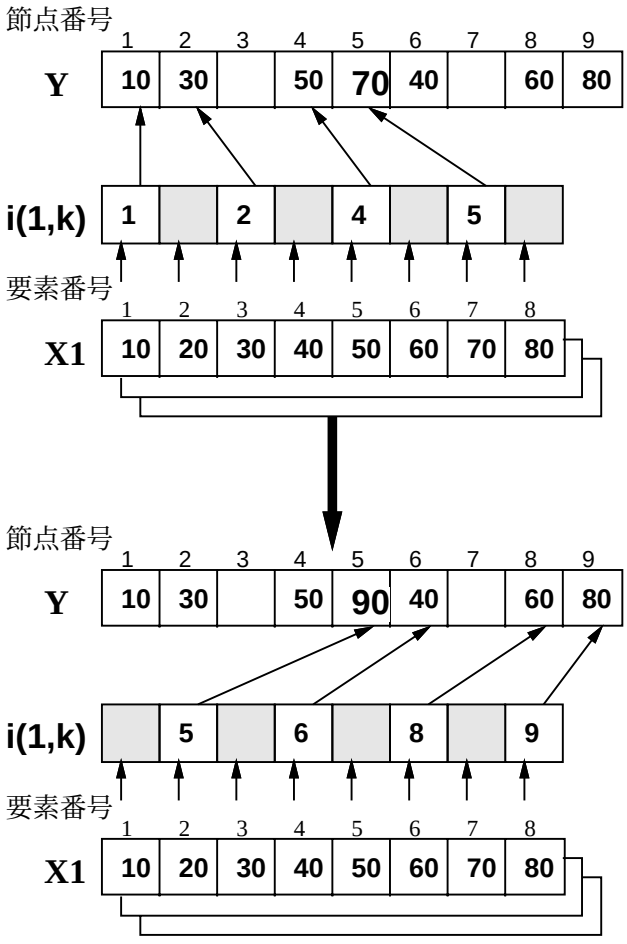


図 5.5:

この場合、変更前のプログラムと比べるとほぼ2 倍の実行時間がかかる。これは、２つ飛ばしでも実行時間は変わらないためで、仮想プロセッサの仕様と思われる。

本研究において、２次元解析の場合には８節点四角形要素を用いた。四角形要素の場合、一つの節点にかならず４つの要素しか接しないため、きちんと重ならないように節点番号と要素番号をつければ、上のような変更は必要ない。

付則：プログラムはさらに、

```
forall(j=1:node,k=3)b(i(j,k))=b(i(j,k))+s(j,k)
```

と並列化することできるが、３節点三角形要素ではそれほどの速度向上にはつながらなかった。しかし、３次元化をおこなった際に２０節点六面体要素を使用した時、重ね合わせの実行時間が大幅に増えることがわかった。上のような重ね合わせプログラムが使用できる工夫を考える必要があると思われる。

5.1.2 グローバル／ローカルプログラミングでの全体行列への重ね合わせ

重ね合わせプログラムをグローバル／ローカルプログラミングで作成した。全体行列Aと要素行列をグローバル・プログラムでグローバル parallel 配列として宣言し、全体行列Aを図（5.6）のようにノードごとに分割し、要素行列の配置を図（5.7）のようにノードごと等しく分配した。まず、グローバル・プログラムで要素行列計算を実行した。

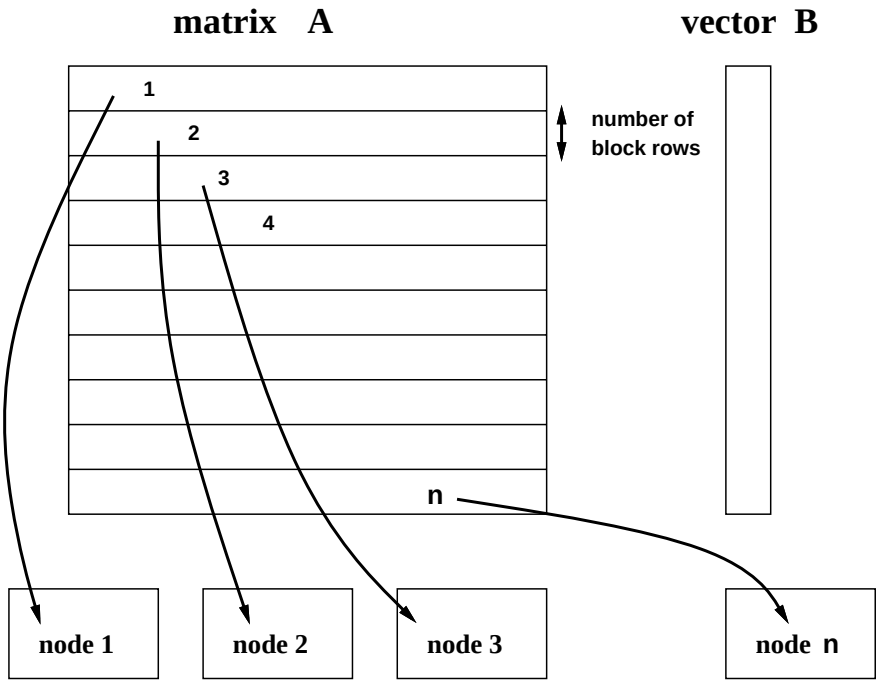


図 5.6: 全体行列の分割

次に、ローカル・プログラムで、要素行列を全体行列へ重ね合わせを実行した。
ローカル・プログラムでの全体行列へ重ね合わせの具体的なアルゴリズムは、

1. まず、各ノードごとで受けとるべきデータを、どこのノードから受けとるかを検索する。次にデータを受けとる準備をし、データを受けとると所定の配列にデータを収め、再び次のデータを受けとる準備をする。

2. 受けとるべきデータをすべて受けとったあと、そのノードがもっている要素行列のなかで別のノードへ渡すべきデータがあれば、相手先を検索し、渡す。

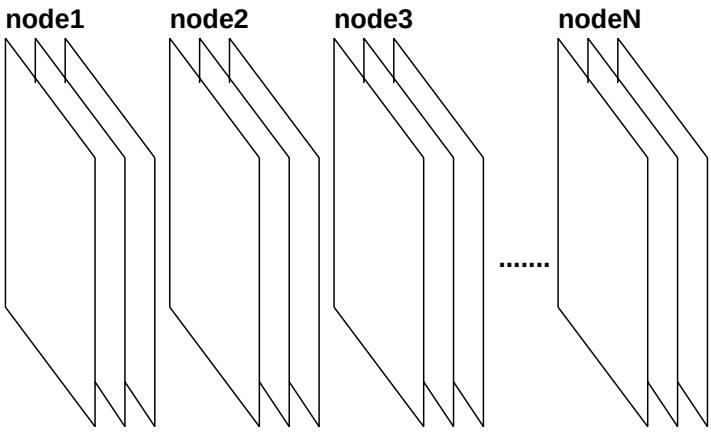


図 5.7: 要素行列の配置

これによって、通信衝突をすることなく、グローバル／ローカルプログラミングでの重ね合わせが実行できた。

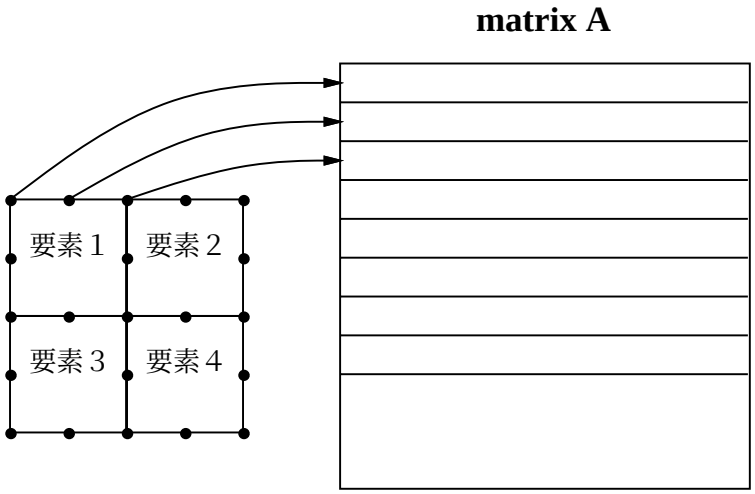


図 5.8: 要素行列の全体行列への重ね合わせ

5.1.3 実験結果

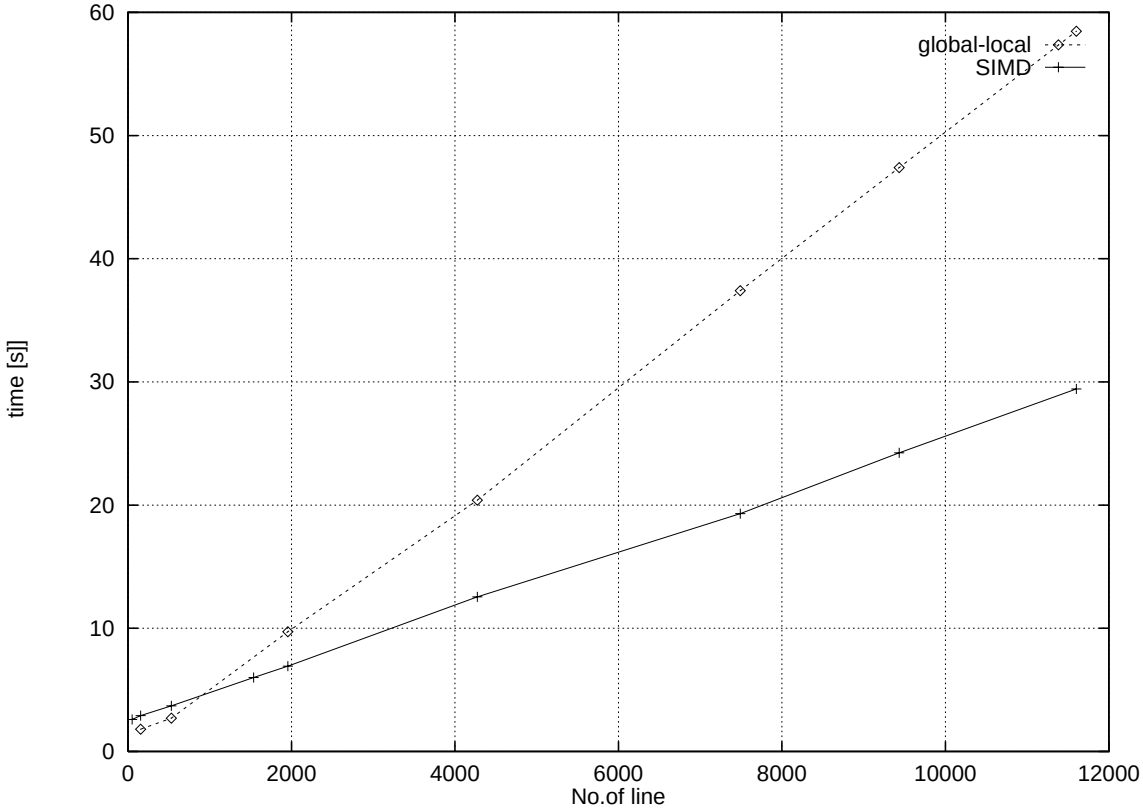


図 5.9: 重ね合わせの実行時間

結果は、図（5.1.3）でしめした。ローカルプログラムでは通信衝突をさけるために、別のノードからデータをすべて受けとった後に自分が持つ要素行列のデータを他のノードに転送する方法をとったために、（データを受けとるまでの）ノードの待ち時間が行列の次元が大きくなるにつれて多くなったと考えれる。待ち時間以外の実質的な処理は、ＳＩＭＤ型プログラム, グローバル／ローカルプログラムともほぼ同じなので、最適化を行って待ち時間を減らせれば、ＳＩＭＤ型プログラムに近い実行時間が得られると思われる。しかし、その作業は複雑でかなりの労力が必要であり、割に合わない作業と考えれる。

5.2 連立方程式の解法であるSOR法のＳＩＭＤ型プログラミングによる並列化と、グローバル／ローカルプログラムによる並列化について

SOR法は、現在では汎用性にかけ、一般には使用しない方がいいと言われている。

しかし、係数行列Aや右辺$\mathbf{b}$を修正しながら何回も解きたい場合には、直前の解を初期値とするSOR法のほうが、結果的に前処理付きＣＧ系の解法を使うよりは、反復回数が少なくなることがある。今回の分離型解法の場合２段階めの式（2.21）は対角要素に０がない粗行列であること、時間反復を繰り返すたびに２段階めの全体行列が１段階めの結果から変更されることから採用した。本研究での図（5）の実験モデルの場合、実際に準定常状態近くになると反復回数がわずか数回とかなり少なくなる。

5.2.1 ＳＩＭＤ型プログラミングでのSOR法の並列化

SOR法は、以下の式を繰り返し反復することで解を求める方法である。

$$\mathbf{x^{(k)}} = (1-\omega)\mathbf{x^{(k-1)}} + \omega D(-L\mathbf{x^{(k)}} - R\mathbf{x^{(k-1)}} + \mathbf{b}) \tag{5.1}$$

ただし、x_kを反復過程の第k段階の解、D, R, Lはそれぞれ配列Aの対角行列、下三角行列、上三角行列とする。

SOR法で一番実行時間がかかる処理は下三角行列とベクトル$x^{(k)}$との積, 上三角行列とベクトル$x^{(k-1)}$との積である。逐次型プログラムでは、それぞれの処理は２重のdoループになっている。それらをＳＩＭD型の並列化を行なった。下三角行列とベクトル$x^{(k)}$の積は計算済みの$x_1^{(k)},,,x_{(i-1)}^{(k)}$を使用するので、DOループが残ってしまった。

```
      DO 70 I=2,N
         sum2=0.0d0
         forall(J=1:I-1) sum2(j) = A(I,J) * X(J)
         ssum = sum(sum2,1)
         X(I) = X(I) - SSUM * pa
 70   continue
```

5.2.2 グローバル／ローカルプログラミングでの SOR 法の並列化

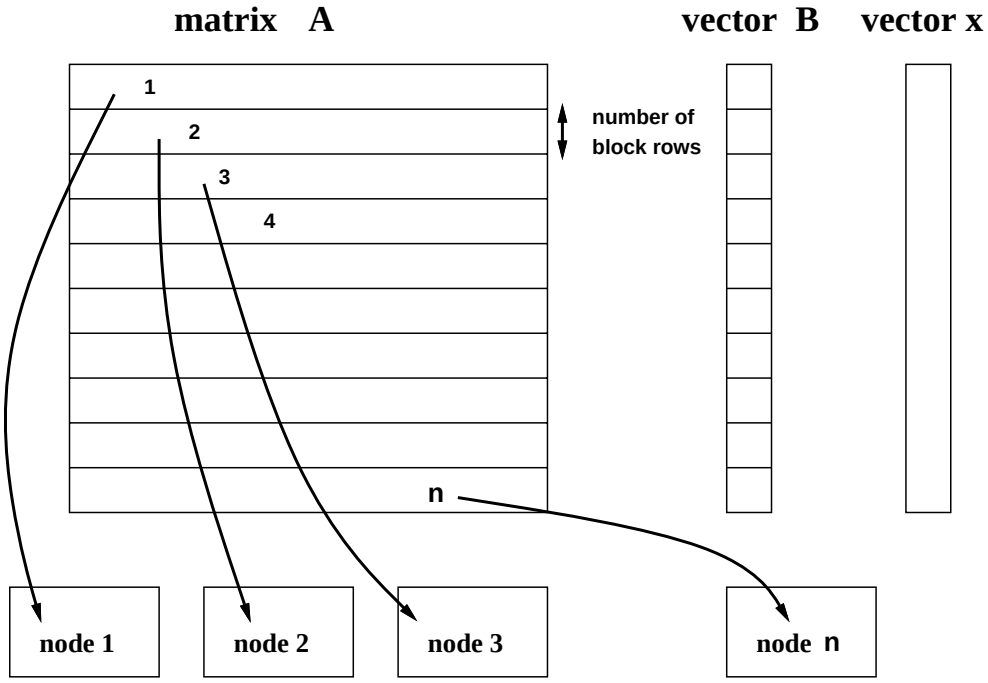


図 5.10: 全体行列の分割

SOR 法をグローバル／ローカル・プログラミングで作成した。解くべき全体行列Aと右辺ベクトル b と初期解 x_0はグローバル・プログラムでグローバル parallel 配列と宣言し、解くべき全体行列Aと右辺ベクトルBを図（5.10）のように行ブロックごとにノードごとに分割した。初期解 x_0はすべてのノードにメッセージパッシングした。特に、ＳＩＭＤ型では、残してしまった三角行列とベクトル $x^{(k)}$の積の DO ループ部分について並列化を行なった。問題の部分をローカル・プログラムでは、ブロックごとに下三角行列とベクトル $x^{(k)}$との積計算をおこなうことにした。これによってブロックごといくつかの計算済みの j 個の $x_i^{(k)},,,x_{(i+j)}^{(k)}$ができる。それをメッセージパッシングを行ない、すべてのノードに計算済みの $x_i^{(k)},,,x_{(i+j)}^{(k)}$が行き渡せる。データを受けとった各ブロックは先ほどの計算済みの $x_i^{(k)}$を使って、下三角行列とベクトル $x^{(k)}$との積計算であらかじめ可能な部分を先行して行なうことで、スピードアップを計ることにした。

5.2.3 結果

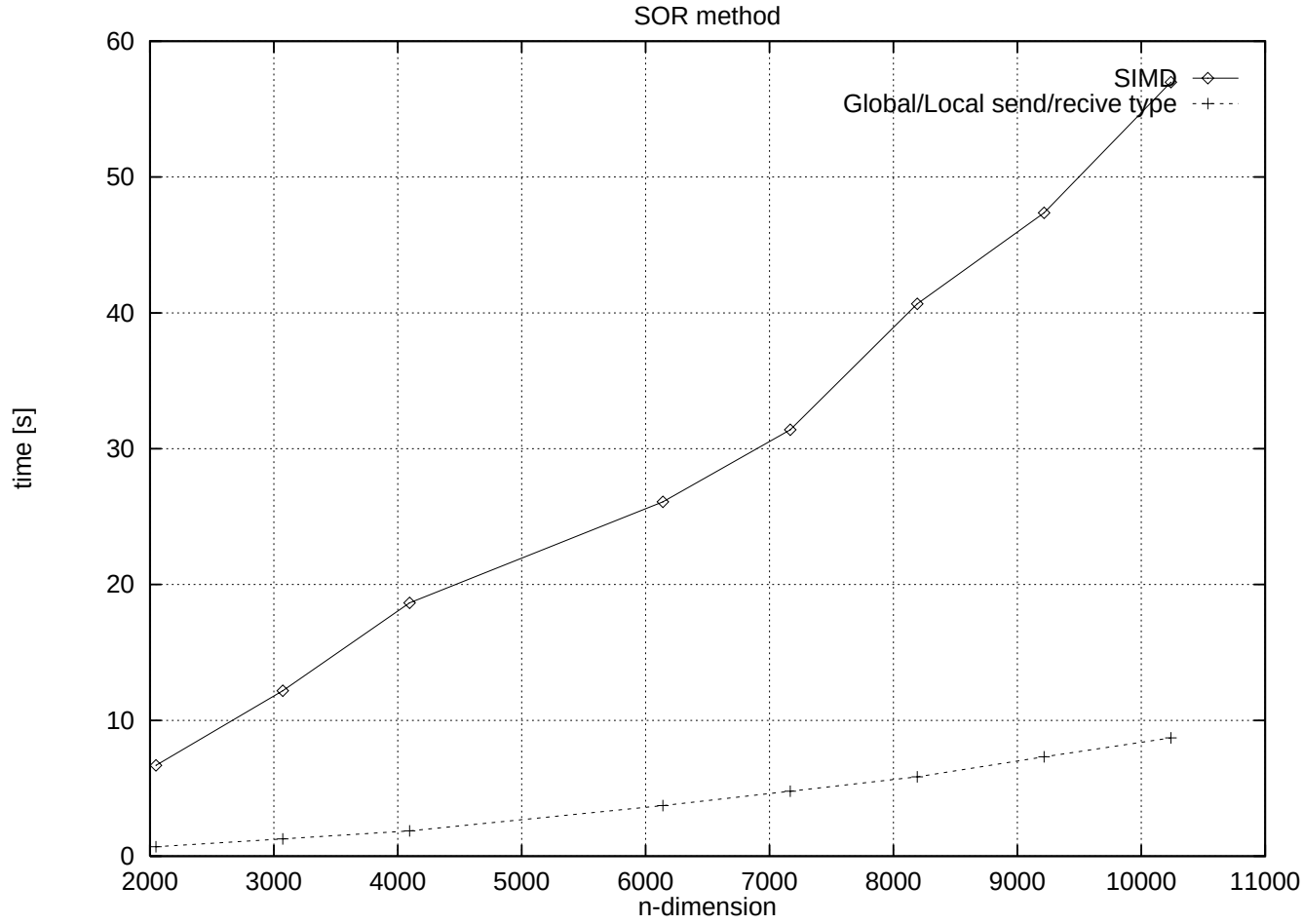


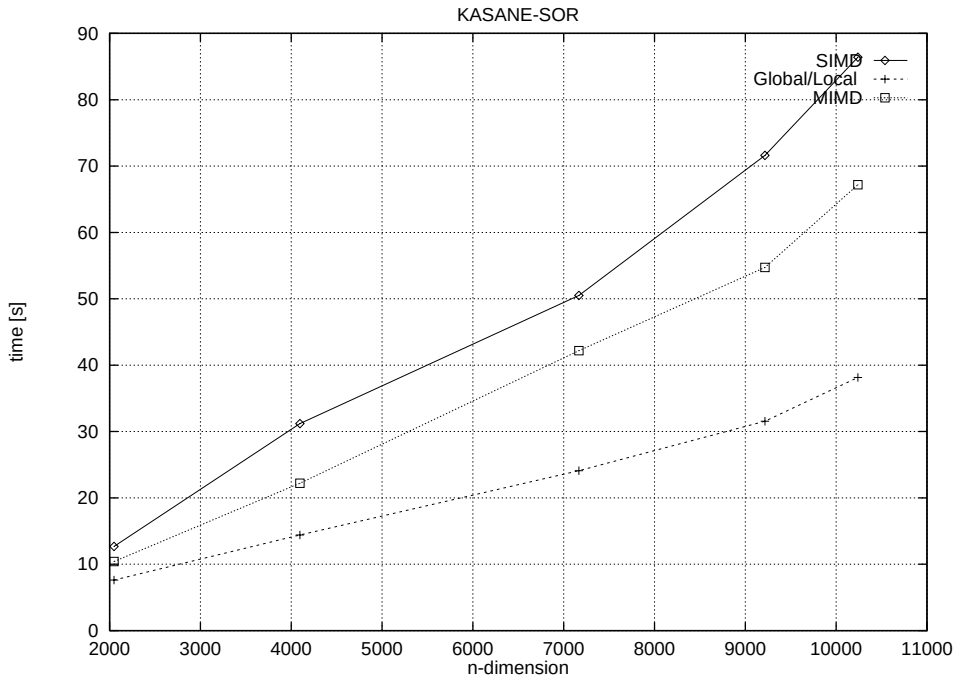
図 5.11: SOR 法の実験結果

結果は、図（5.2.3）でしめした。グローバル／ローカル・プログラムとＳＩMD型プログラムでは、最大で５倍の実行時間比が得られた。グローバル／ローカル・プログラミングによって、下三角行列とベクトル $x^{(k)}$ との積の並列化の効果が出ている。また、細かく調べてみると、上三角行列とベクトル $x^{(k-1)}$ との積さらに並列化を施した部分があるためと思われる。が、少しＳＩMD型プログラムが遅いような思われる。もしかすると別の部分で実行時間に差ができてしまう処理の部分がある可能性があると思われる。

5.3 有限要素法でのグローバル／ローカル・プログラムの効果

有限要素法でのグローバル／ローカルプログラムの効果を推し量るために、全体行列への重ね合わせとＳＯＲ法のプログラムをそれぞれ以下のような場合についてのプログラムを実行してみた。行列の配置などは、前実験を踏襲することとする。

- 重ねあわせプログラムとＳＯＲ法プログラムともＳＩＭＤ型プログラムする場合
- 重ねあわせプログラムをグローバルプログラムし、ＳＯＲ法プログラムをローカルプログラムとする場合
- 重ねあわせプログラムとＳＯＲ法プログラムともＭＩＭＤ型プログラムとする場合



5.3.1 結果

結果は、図（5.3）で示した。ＭＩＭＤ型については、あまり最適化していない等で公平ではないが、ＳＩＭＤ型で不利な部分をＭＩＭＤ型プログラムにかえるというグローバ

ル／ローカル・プログラムの有効性を示せたと考える。

第 6 章

まとめ

6.1 グローバル／ローカルプログラミングについて

第四章での実験結果から、グローバル／ローカルプログラミングにはいくつかの特徴があることがわかる。

グローバル／ローカルプログラミングでの行列・ベクトル積は行列をグローバル parallel 配列、ベクトルをグローバル serial 配列にした。このため、一度、ベクトルを全てのノードにデータをメッセージパッシングをしてしまうと、２度とすることなく処理が終了してしまう。ＳＩMD型プログラムで、行列・ベクトルとも parallel 配列に宣言した場合と実行時間をくらべるとあきらかに、ＳＩMD型プログラムが速い。メッセージパッシングしないＭＩMD型の行列・ベクトル積の実行時間とＳＩMD型プログラムと比べてみると、時間差はほとんどない。グラフに現れた時間差はメッセージパッシングと para llel 配列への変換時間と考えれる。

行列の乗算の実験から、ＳＩMD型プログラムの場合、実行が進むたびにメッセージパッシング等の乗算以外の処理が発生すると考えられる。一方、グローバル／ローカルプログラミングの場合、そのメッセージパッシングが規則的に行なわれ、ノードプロセッサの待ち時間なしに効率的に行なわれる。このことが、ベクトルユニットが効果的に働かされるために処理が速くなったと考えれる。これは効率的な通信を行なうことで結果的に処理が速くなる好例だと思われる。

6.2　グローバル／ローカルプログラミングの有限要素法への応用について

２つの実験からローカルプログラム化していい部分とわるい部分があることがわかった。

メッセージ・パッシングをともなわないプログラムにおいて、ＳＩMD型プログラミング、グローバル／ローカルプログラミングとでは、さほど処理時間は変わらない。むしろ、グローバル／ローカルプログラミングするには確実により並列化が出来る部分があるか、メッセージ・パッシングをコントロールしてノードプロセッサの待ち時間を減らしたり、よりベクトルユニットをうまく動かせるという工夫ができる等がなければ、速くならないことわかった。また、CM－５の今の命令群では、どこのノードにどのように配列要素を分配するための制御が大雑把にしかできず、かえってローカルプログラムに新たに処理を加えることになり、遅くなってしまうことが多分にあることが実験過程でわかった。

第 7 章

あとがき

7.1　本研究で得られた成果

本研究で得られた成果としては、グローバル／ローカルプログラミングによって新たな並列処理が可能になることを示したでことである。得られた知見として、以下があげられる。

- ＳＩMD型プログラムではむずかしかった並列化がグローバル／ローカルプログラミングを採用することで可能になること
- グローバル／ローカルプログラミングでは、メッセージ・パッシングをユーザー側でコントロールできることから、ノードプロセッサの待ち時間を制御できる。
- 大きなグローバル serial 配列をローカルプログラムで使おうとすると、データを転送するための時間かかるため、注意が必要。
- ＳＩMD型プログラムと比べるとメモリ使用量が少なくなる。

7.2　今後の課題

今回、プログラムが間に合わなかった前処理付きＣＧ法についてのグローバル／ローカルプログラミングについて評価をおこなう。また、ＳＩMD型プログラムとでしか比べなく、結局、ＭＩMD型プログラムの優位な点でしか調べていないと可能性がある。すくな

くとも領域分割法（もしくは、全体行列の配列を、物理空間に合わせてノードごとに分割して処理するが、あくまで一つの行列として扱う）について、ＭＩMD型プログラムによるものとグローバル／ローカルプログラミングによるものとを比べてみる必要がある。

第 8 章

付録

8.1 計算結果と収束判定

今回使用したスキームが正しい解をしめすか確かめるために、正方キャビティ流れを用いた。このモデルは水をいっぱいに満たした水槽を考え、この水槽の表面を水平方向から風が一様にふくとき、風によって水槽の表面が一様な流速U_0でながれ始めるとする。だだし水面は波たつとこなく、重力方向に対してはいつも垂直な面を保っていると仮定する。今回の場合、$U_0 = Vs = 1$ とし、$\Delta t = 0.02, Re$ 数は 2000 とした。

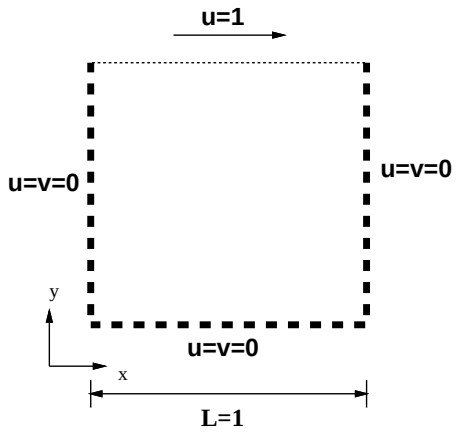


図 8.1: 実験モデル

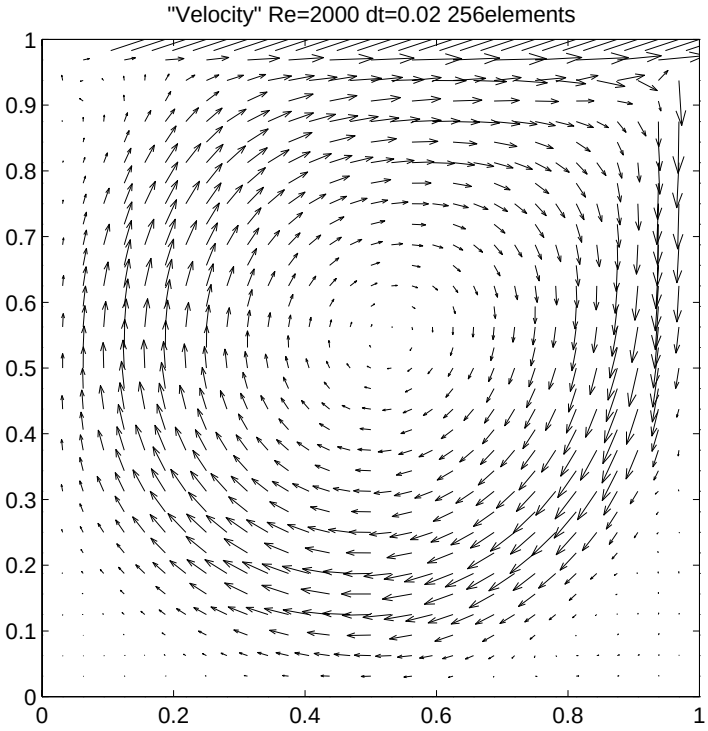

"Velocity" Re=2000 dt=0.02 256elements
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

図 8.2:

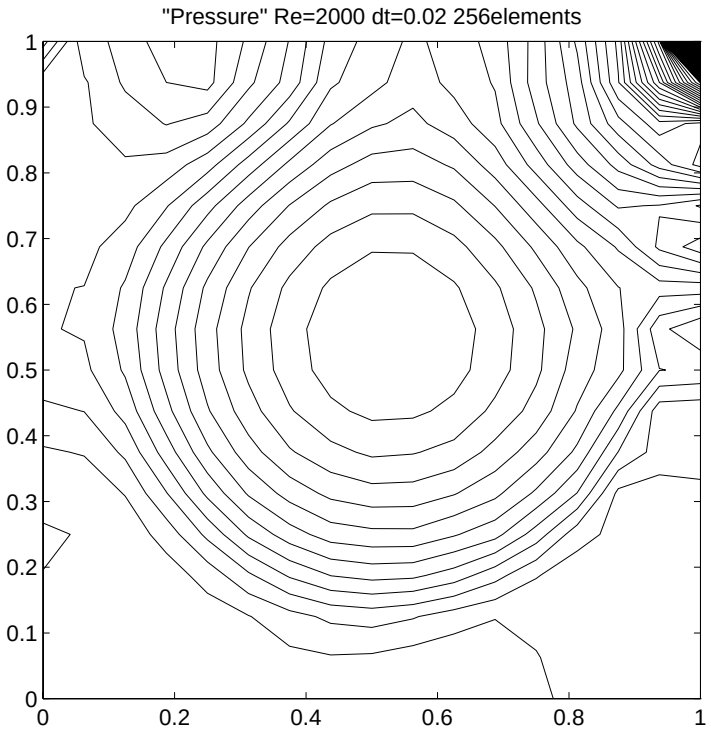

"Pressure" Re=2000 dt=0.02 256elements
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

図 8.3:

謝辞

本研究を行なうあたり、日頃御指導を頂いた松澤照男教授に深く感謝いたします。また、有益な助言や討論等をいただいた古山彰一君、杉山晃一君に感謝致します。さらに3年間の生活を共にした松澤研究室の皆様に感謝致します。

参考文献

[1] 原田 隆, 鎌形昌宏, 松澤照男:
“有限要素法による非圧縮性粘性流れに関する並列アルゴリズムの開発” 第８回計算力学講演会講演論文集 pp301-302,1995

[2] 原田 隆, 松澤照男:
“グローバル／ローカル・プログラミングによる有限要素法アルゴリズムの開発” 第９回計算力学講演会講演論文集 pp475-476,1996

[3] M. Behr and A.Johnson :
Computation of incompressible flows with implicit finite element implementations on the Connection Machine, Computer Methods in Applied Mechanics and Engineering, 108, 99-118, 1993

[4] deněk Johan,Kapil K. Mathur, S. Lennart Johnsson and Thomas J. R.Hug hes :
A Data Paraller Finite Element Method for Computational Fluid Dynamics on the Connection Machine System, Computer Methods in Applied Mechanics and Engineering, 1994

[5] Edward J.Dean and Roland Glowinski:
On Some Finite Element Methods for the Numerical Simulation of Incompressible Viscous Flow *Incompressible computational fluid dynamics*2,pp17-66

[6] CMSSL for CM Fortran:
CM-5 Edition, Thinking Maxhines Corporation,Cambridge,1994

[7] CM Fortran Programming Guide:
CM-5 Edition, Thinking Maxhines Corporation,Cambridge,1994

[8] CM Fortran Performance Guide:
CM-5 Edition, Thinking Maxhines Corporation,Cambridge,1994

[9] 数値流体力学編集委員会 編:
数値流体力学シリーズ１非圧縮性流体解析, 東京大学出版会, 1995

[10] Frontier 利用者の会 編:
北陸先端科学技術大学院大学 情報科学センター 利用の手引