計算機アーキテクチャ　141－1
（２００１．１．２６）

乱流シミュレーションの並列化と評価

高田　雅美 *　山本　義暢 †　功刀　資彰 †　城　和貴 *

takata@ics.nara-wu.ac.jp

* 奈良女子大学大学院　人間文化研究科　情報科学専攻
† 京都大学大学院　工学研究科　原子核工学専攻

概要

本稿では，自由表面乱流場における直接数値シミュレーションの逐次プログラムを，並列化ライブラリ Message Passing Interface (MPI) を用いて並列プログラムに変換するための並列化手法について報告する．４ならびに８プロセッサ対応の並列プログラムを実装し，それらの実行時間が，逐次プログラムと比較して，それぞれ約 1/4, 1/8 になることを，実験によって示す．

Parallelization and Evaluation of a Turbulent Flow Simulation

Masami Takata *　Yoshinobu Yamamoto †　Tomoaki Kunugi †　Kazuki Joe *

* Graduate School of Human Culture, Nara Women's University
† Department of Nuclear Engineering, Kyoto University

Abstract

In this paper, we present parallelization methods for a direct numerical simulation using Message Passing Interface. The parallelized programs for 4 processors and 8 processors have been implemented, and we show that execution time of the parallelized programs are about 1/4 and 1/8 compared with the original sequential program respectively by experimental results.

1　はじめに

原子炉，核融合炉，化学プラント等の工業装置内，さらには海洋，河川の環境中と，自由界面を有する乱流場は，広範囲な工学分野で頻繁に出現する．それと同時に自由表面上での熱物質移動が界面での乱流構造を通して行われる．この乱流の直接数値シミュレーション（*DNS*: Direct Numerical Simulation）[1] は，近年，スーパーコンピュータの発達とともに，乱流解析における有力な解析手法となった．しかし，*DNS* を計算するためには，計算機資源を長時間にわたって使用しなければならない．このことを回避するために，*DNS* 計算のためのプログラムを並列化し，高速化を行う必要がある．

本稿では，分散メモリ型並列計算機を対象とする並列化ライブラリとして Message Passing Interface (MPI) を利用し，*DNS* を並列プログラムに変換するための並列化手法について述べる．実験設備の関係上，今回は，２の累乗で表される４台及び８台のプロセッサを用いる並列プログラムを実装することを目的とした．

以下，第２章において自由表面乱流場における熱物質輸送の *DNS* について説明し，そのプログラムソースに対する並列化手法を第３章に記す．第４章では，実際に開発された逐次プログラムと並列プログラムの実行時間を比較する．

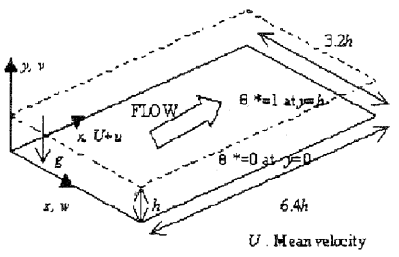

図 1: 解析対象と座標系

2 自由表面乱流場における熱物質輸送の直接数値シミュレーション

自由表面を有する乱流場は広範囲な工学分野で頻繁に出現するので，その乱流構造及びそれに付随する熱物質輸送を解析することは工学上重要である．しかし乱流は非線形の非定常現象で，大小様々な時間・空間スケールを有し，極めて高い自由度を持つ散逸力学系である．そのため乱流の完全数値シミュレーション (*FTS* : Full Turbulence Simulation) においては，レイノルズ数 : $Re = \mathrm{U}\,L/\nu$ (U : 代表速度，L : 代表長さ，ν : 流体の動粘性係数) の 9/4 乗以上の格子点数を要した，非定常の 3 次元計算を強いられる．従って，比較的低いレイノルズ数 1000 の計算においても，1000^3 以上の格子点数が必要となり現在の計算機能力の限界に近い値となる．さらに，流体の拡散係数 ν よりも熱の拡散係数 α が小さい高プラントル流体における乱流熱輸送の解析においては，3 次元の各方向にプラントル数 : $Pr = \nu/\alpha$ の $-1/2$ 乗倍の格子解像度を要することから，高プラントル自由表面乱流場の *DNS* においては，大規模容量と高速演算に対処した並列化計算が不可欠となっている．

図 1 に本研究における解析対象 (2 次元開水路乱流場) と座標系を示す．計算条件は，流れ場の平均断面流速 U_m と水深 h によるレイノルズ数を約 2270，プラントル数を 1 とし，x, y, z 方向にそれぞれ $64, 82, 64$ の格子点を用いた．数値解析手法は，基礎方程式である Navier-Stokes 方程式，連続式において，時間進行は，対流項及び粘性項に 2 次精度の Adams-Bashforth 法，圧力項に Euler 陰解法を用いこれらを Fractional step 法により解いた． 空間に対する離散化は 2 次精度の中心差分 [2][3] を使用した．パッシブスカラーを仮定した温度の輸送方程式に対しては同様に，2 次精度の Adams-Bashforth 法，及び 2 次精度の中心差分を用いた．境界条件は，流体運動に対しては x,z 方向に周期境界条件，壁面においては no-slip 条件，水面においては鉛直方向の流速を 0，その他に対称条件を用いた．温度に対しては，水面及び壁面で温度一定とし， x,z 方向には同様に周期境界条件を課した．図 2 はその計算例で，上流側から仮想的な粒子を注入して流れを可視化したものである．流れは，左から右に向かって流れている．壁面付近の低速流体が乱流の組織運動により，水面付近まで達し，表面更新渦となり大規模な水平渦を構成している様子が観察できる．

基礎方程式は，Navier-Stokes 方程式 (1)，連続式 (2)，そしてエネルギー方程式 (3) であり，それぞれ以下のように表せる．

$$\frac{\partial u^*_i}{\partial t} + u^*_j \frac{\partial u^*_i}{\partial x_j} = -\frac{\partial}{\partial x_i}\left(\frac{P^*}{\rho}\right) + \nu \frac{\partial^2 u^*_i}{\partial x_j \partial x_j} \quad (1)$$

$$\frac{\partial u^*_i}{\partial x_i} = 0 \quad (2)$$

$$\frac{\partial \theta^*}{\partial t} + u^*_j \frac{\partial \theta^*}{\partial x_j} = \alpha \frac{\partial^2 \theta^*}{\partial x_j \partial x_j} \quad (3)$$

ここで，u^*_i は i 方向の速度成分 $(i = 1, 2, 3)$，$x_1(x)$ は流下方向，$x_2(y)$ は鉛直方向，$x_3(z)$ は横断方向，上付き添え字 $*$ は瞬間値，P^* は圧力，ρ は密度，ν は動粘性係数，θ^* は，$= (T^* - T_b)/(T_s - T_b)$ で表される水面及び壁面における温度差により規格化した温度，T^* は温度，T_s,T_b は水面及び壁面における温度 (ただし，$\Delta T = T_s - T_b > 0$)，α は熱拡散係数をそれぞれ示し，繰り返し添え字は縮約規約に従うものとする．

3 並列化手法

DNS の並列化を MPI を利用して行うのに必要な手法等について，以下のように説明する．3.1 節では，データ通信の方法に関して，3.2 節では，データ分割に関して述べる．1 イタレーション内でもっとも計算量の多いポアソン方程式である共役残差法に関する *subroutine press* についての並列手法を 3.3 節で説明する．最後に，出力部に関して 3.4 節で述べる．

3.1 プロセッサ間のデータ通信

分散メモリ型並列計算機を対象とした並列プログラムに対して，ある程度効果的なデータ分割がされ

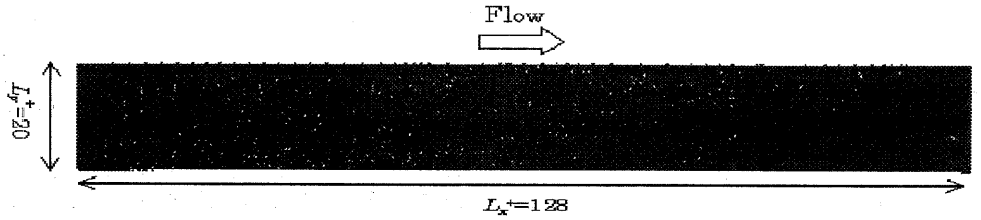

図 2: 流れの可視化

ていても，最低限のデータ通信が必要となる．

MPI のデータ通信方式として，同期式通信と非同期式通信の 2 種類が存在する．

同期式通信の場合，送信側プロセッサ S による送信用関数（*MPI_SEND* 等）の呼び出しと，受信側プロセッサ R による受信用関数（*MPI_RECV* 等）の呼び出しが，同時に行われる．そのため，非同期式通信に代表されるような，通信によるデータの書き換えミスは生じない．しかし，同期式通信をするために，プロセッサ S, R のどちらかが，計算を一時中断して待たなければならない．このことにより，通信遅延が生じる．

一方，非同期式通信の場合，送信用関数（*MPI_ISEND* 等）の呼び出しと受信用関数（*MPI_IRECV* 等）の呼び出しは，別々のタイミングに行われる．そのため，データ通信開始のタイミングによっては，間違ったタイミングにデータを書き換えてしまう危険性がある．この危険を回避するためには，プログラマが，データ通信のタイミングを考慮する必要がある．それゆえ，プログラムの開発が困難となり，プログラマへの負担が大きくなる．しかし，通信を同時に開始する必要がないため，非同期式の送受信用関数を呼び出した後，実際に通信が行われるまでの間，各プロセッサは，計算を中断することなく実行を続ける事が可能となる．そのため，通信によるオーバーヘッドを減少させる事ができる．

今回，我々は，全体の計算時間を減らすために，非同期式通信を適用する事とする．ただし，MPI において，以下に述べる動作を行うための関数は，同期式通信しか存在しないので，括弧内の関数を採用するものとする．

- 各プロセッサのデータ x の中から，最大値，最小値や総和を求め，その結果を全プロセッサに送信するための関数（*MPI_ALLREDUCE*）
- ブロードキャスト関数（*MPI_BCAST*）

3.2 データ分割

ループ文において，異なるイタレーション内に依存関係が存在しない場合，その *do* 文を並列実行文の *doall* に変換する事ができる．

一般に，メモリ分散型並列計算機において，*doall* は，各プロセッサ用に strip mining [4] されることが理想的とされている．

```
<例>
      do 10  i = 1,100
  10      x(i) = i
      min = y(1)
      do 20  j = 2,100
           if(min .gt. y(j)) then
                min = y(j)
           endif
  20      continue
      do 30  k = 1,100
  30      z(k) = z(k)*k
      do 40  i = 1,200
  40      w(i) = x(I(i) )*min
```

しかし，上記の例の *do* 10，*do* 20，*do* 30 のように依存関係を持たない場合，この方法によって分割されることは，好ましくない．その理由は 2 つある．

1 つ目の理由は，全プロセッサに strip mining した場合，例の *do* 20 において，同期式通信の *MPI_ALLREDUCE* が必要となり，また，*do* 10 に関して，各プロセッサで計算された配列 x の値が配列 I の値に従って *do* 40 において用いられるため，同期式通信の *MPI_BCAST* が必要となるからである．前項でも述べたように，同期式通信が必要となる状況は，あまり好ましくない．

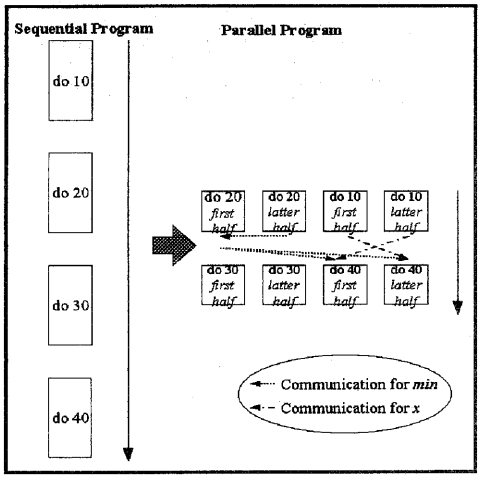

図 3: 例に対する並列化

2 つ目の理由は，通信回数について考慮されていないからである．通信にはオーバーヘッドが伴うので，これを回避するために，通信回数を少なくする必要がある．例えば，例の *do* 10 と *do* 40 の間には，配列 x に関するデータ依存が存在するため，送信，受信をそれぞれ 1 回と数えると，非同期式通信が 1 プロセッサ当り $2*$（全プロセッサ台数 -1）回必要となる．また，仮に同期式通信を用いた場合でも，全プロセッサ台数 回必要となる．この回数を減らすためには，配列 x に関する *do* 10, 40 の計算を行うプロセッサ数を減らせばよい．

そこで，我々は，各ループ文を全プロセッサに strip mining するのではなく，いくつかのプロセッサに strip mining する手法を用いる事にする．ただし，各プロセッサのローカルメモリ容量は，充分あるものと考える．

例えば，4 つのプロセッサを用いて例の逐次プログラムを並列化する場合，各プロセッサは，図？で示されている次の手順をふむ．

プロセッサ *id* 0 及び *id* 1 では，まず，*do* 20 を半分ずつ計算する．その後，プロセッサ *id* 1 からプロセッサ *id* 0 へ変数 min を非同期通信し，プロセッサ *id* 0 において，非同期式受信が終了した事を確認した後自分の変数 min と比較し，小さい方をプロセッサ *id* 2 及び *id* 3 に非同期式送信する．その後，*do* 30 の計算を行う．

プロセッサ *id* 2 及び *id* 3 では，プロセッサ *id* 0 から変数 min を非同期式受信し，*do* 10 を半分ずつ計算した後，お互いに計算した配列 x を相互に非同期式通信する．その後，全ての非同期式受信が終了した事を確認し，*do* 40 の計算を行う．このことにより，単純に全プロセッサに strip mining した場合と比べ，同期式通信が不要となり，通信回数も，変数 min に関する通信は，6 回から最大 3 回に減り，配列 x に関しても，非同期式通信で 6 回，同期式通信ならば 4 回必要な通信が最大 2 回に減少する．

自由表面乱流場における熱物質輸送の *DNS* 用プログラムにおいて，主な配列として，3 次元方向の格子間隔と座標の位置を表す配列 x, y, z，水面及び壁面における温度を表す配列 $dist_x, dist_y, dist_z$，x, y, z 方向の各流速を表す配列 u, v, w，対流項に粘性項を足した値を表す配列 $fu, fuo, fv, fvo, fw, fwo$，温度を表す配列 t，温度のエネルギー方程式における対流項に粘性項を足した値を表す配列 ft, fto，圧力を密度で割った値をあらわす配列 p がある．

3 つの配列 x，y，z は，お互いに依存関係がないので，同タイミングに計算可能である．同様に，配列 $dist_x$，$dist_y$，$dist_z$ についても，同タイミングに計算可能である．

4 つの配列 u，v，w，t は，同一イタレーション内において，それぞれ 1 イタレーション前の各配列を利用して計算されるので，同タイミングに計算しても構わない．同様に，配列 fu, fuo，fv, fvo，fw, fwo，ft, fto は，同タイミングに計算可能である．また，配列 p は，*subroutine press* によって計算される場合（3.3 節参照）を除いて，同一イタレーション内において，他の配列と同タイミングに変更可能である．よって，今回，我々は，各配列に関するループのイタレーション数を考慮し，8 プロセッサを対象とする並列プログラムを開発するために，2 プロセッサ毎に，配列 t, ft, fto, p，配列 $x, dist_x, u, fu, fuo$，配列 $y, dist_y, v, fv, fvo$，配列 $z, dist_z, w, fw, fwo$ を計算させるものとする．

また，2 プロセッサ毎に計算される配列をそれぞれ 1 プロセッサ毎に計算されるように変更することにより，4 プロセッサ用の並列プログラムへと拡張する事が可能となる．つまり，この分割方法は，プロセッサ数が 4 の倍数の時，最も有効となる．

3.3 ポアソン方程式の共役残差法に関する並列化手法

ポアソン方程式の共役残差法の関数 *subroutine press* では，配列 p と他の計算部分と異なる配列が用いられる．

この関数を計算する際，同配列内での依存は，境界依存があるのみである．それに対して，異配列同士での依存は，他の計算部分と異なり極めて強い．ゆえに，3.2 節のような分割方法では，非同期式通信が終了するまで計算を停止させる関数（*MPI_WAIT* 等）によって長い待ち時間が生じ，非同期式通信の長所をいかしきれない．

そこで，*MPI_WAIT* による時間遅延を解消するために，計算量を考慮した次の 2 つの分割方法が考えられる．

まず一つ目は，8 プロセッサを対象とする並列プログラムを実装する場合，配列の最大値を求めるための *subroutine norm* に関する計算を 1 つのプロセッサで行い，残り 7 つのプロセッサで，データに関する計算を行うという分割方法である．演算数を静的に数えた場合 *subroutine norm* に関する計算量の 7 倍が残りの計算量にほぼ等しくなるからである．この分割方法を用いると，プロセッサ数が 8 の倍数である時の並列化が最も有効となる．しかし，この方法では，目的の配列の要素数が 2 の倍数で実装される事が多いにもかかわらず，各配列を 7 等分しなければならないので，あまり好ましくない．また，通信に関する情報が増え，プログラマの負荷が非常に大きくなることが予想されることも問題である．

二つ目は，各配列を全てのプロセッサで等分に分割する方法である．この方法では，ある配列内の最大値を求めるための関数 *subroutine norm* と，総合計を求めるための関数 *subroutine inprod* において同期式通信 *MPI_ALLREDUCE* が必要となる．

subroutine press には，*doall* 以外の計算は，数命令しか存在しないという特徴がある．そのため，各配列を全プロセッサ数で分割し，各 *doall* を全プロセッサで strip mining したとしても，プロセッサの性能の差が大きくない限り，計算時間の差はほとんど生じない．よって，同期式通信による通信遅延が生じる危険性は小さい．ただし，皆無となるわけではない．

以上より，*subroutine press* に関する並列化手法として，データ分割は後者を用い，通信方式は，*subroutine norm* と *subroutine inprod* において同期式通信 *MPI_ALLREDUCE* を用いる場合以外，非同期式通信を用いる方法が適していると考える．

この分割方法を用いる場合，*subroutine press* の計算終了後，各プロセッサが計算した配列 p のデータをブロードキャストしなければならない．しかし，単純に各プロセッサがブロードキャストしたのでは，送受信を通信 1 回と数えると，各プロセッサの通信回数がプロセッサ台数回になる．もし，相互結合網のトポロジーが例えばハイパーキューブであるならば，この回数を減らすための通信手段として，次の方法を提案するができる．

まずプロセッサ $id\,i$（i:偶数）とプロセッサ $id\,i+1$ に関するデータを，お互いに送信しあい，次にプロセッサ $id\,j$（$j:0,1,4,5$）とプロセッサ $id\,j+2$ に関する新たなデータを，送信しあい，最後にプロセッサ $id\,k$（$k \leqq 3$）とプロセッサ $id\,k+4$ に関する新たなデータを，送信しあう．この方法によって，各プロセッサの通信回数は 3 回に減少する．(ただし，プロセッサ数 8 の場合)

今回は，イサーネットを利用しているため，この通信手法では通信効率を悪くするのみであるので，採用しなかった．

3.4 出力部に関する並列化手法

標準出力の場合，逐次プログラムを実行した場合同様，プログラム実行コマンドが打ちこまれたウィンドウへ，全て出力される．この時，ウィンドウへの出力表示は，プロセッサ $id\,0$ に関する出力が最初に行われる．その他のプロセッサに関しては，プログラム実行終了後，終了したプロセッサから順に，出力する．プログラムの実行終了順序は，任意順序なので，標準出力の順番に意味を持たせている場合，どれか一つのプロセッサのみで，標準出力関数を呼び出すように組まなければならない．

今回，我々は，元の逐次プログラムに関する標準出力の順序を変えない事とした．そのため，標準出力関数の呼び出しは，プロセッサ $id\,0$ でのみ行うものとした．よって，計算量の関係上，大部分のファイル出力は，プロセッサ $id\,0$ 以外のプロセッサで行うものとした．

4 実験

逐次プログラムと MPI で 4 プロセッサ用と 8 プロセッサ用にそれぞれ実装された並列プログラムの実行時間を図 4 に表す．

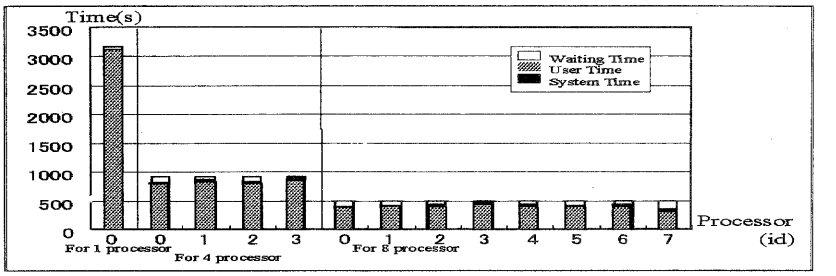

図 4: 実行時間

実験環境として，計算機は Sparc SUN Workstation Ultra-2 (SunOS 5.6)，メモリ容量 512MB を利用し，通信方法として，100base-tx を用いた LAN を利用した．

逐次プログラムの実行時間は 3179 秒であった．

それに対して，4 プロセッサを用いた並列プログラムの実行時間は 913 秒，8 プロセッサを用いた並列プログラムの実行時間は 502 秒であった．

図 4 が表すように，逐次プログラムを実行した場合も，並列プログラムを実行した場合同様，若干待ち時間が生じている．この待ち時間は，実行結果をファイル出力するために生じたものと思われる．

また，実行時間の短縮に関して，単純に逐次プログラムがプロセッサ台数で等分に並列化されたのであれば，並列プログラムの実行時間が，(逐次プログラムの実行時間)/(プロセッサ使用数) となる．しかし，並列化によって生じた通信のために，図 4 で表されるように，待ち時間やシステム時間の割合がより大きくなっている．そのため，ユーザ時間のみに着目すれば，理想時間であるにもかかわらず，実行時間に関しては，理想時間よりも遅くなっているものと思われる．よって，逐次プログラムと比較して，4, 8 プロセッサを用いた並列プログラムの実行時間は，それぞれ，0.287 倍，0.1579 倍に短縮され，これはほぼスケーラブルな性能向上であるといえる．ゆえに，自由表面乱流場における熱物質輸送の *DNS* に関して，今回，我々の採用した並列手法は極めて有効である．

5 結論

本稿では，逐次プログラムで開発された自由表面乱流場における熱物質輸送の直接数値シミュレーションの特徴を考慮し，分散メモリ型並列計算機用の並列プログラムライブラリ Message Passing Interface (*Ver.*1.2.0) を利用して 4 ならびに 8 プロセッサ用の並列プログラムをそれぞれ開発し直した．その結果，逐次プログラムの実行時間に対して，並列プログラムの実行時間を，ほぼ 1/4, 1/8 に短縮することができた．

今後の課題としては，我々の手法がより多くのプロセッサ数に対し有効であるかどうか検証することである．また，物理メモリ容量が計算データ総量よりも少ない場合を想定した並列化手法も考えていく必要がある．

参考文献

[1] Yoshinobu Yamamoto, Tomoaki Kunugi, Akimi Serizawa, " Turbulence Statistic and Scalar Transport in an Open-ChannelFlow" *Advances in Turbulence VIII edited by C. Dopazo* , pp.231-234, 2000.

[2] Hiroshi Kawamura, " Direct numerical simulation of turbulence by finite difference scheme" *The recent developments in turbulence research, International Academic Publishers, Beijing* , pp.54-60, 1994.

[3] 梶島　岳夫, "対流項の差分形式とその保存性" 日本機械学会論文集 *B* 編，vol.60, No.574, pp.2058-2063, 1994.

[4] 中田　育男
コンパイラの構成と最適化，朝倉書店 (1999)