研究速報

流体シミュレーションにおける並列化技法の研究

The Study of the Parallelized Capacity in a Fluid Simulation

橋本明義*・谷口伸行**・小林敏雄*
Akiyoshi HASHIMOTO, Nobuyuki TANIGUCHI and Toshio KOBAYASHI

1. 緒　　言

LES乱流解析のための流体計算コードでは，流体の運動を記述する基本的な非線形偏微分方程式であるナヴィエ・ストークス（Navier-Stokes）方程式をコンピュータで数値的に解いている.新しい環境での計算が必要な際に，流体シミュレーションコードがどの程度のパフォーマンスで実行できるかを予め評価できることは有用である.そこでパフォーマンスデータがデータベース化されている，既存のベンチマークテストである姫野ベンチマークテスト[1]のデータを用いる．本研究では流体コードを並列化し，その中で実行時間の大部分を占める行列計算サブルーチンについてSR 8000/mpp（以下SR 8000），Origin2000といった並列計算機で並列化効率の検証を行う．プログラミング言語としてはFORTRANを対象とし，並列ライブラリとしてはMPIを対象とする．

2. 計算条件と対象並列計算機

Table 1に計算条件を，Table 2に対象となる並列計算機のスペックを記す．

Table 1　計算条件

格子点数（i, j, k）	121×21×41 or 241×41×81
SGSモデル	Smagorinskyモデル
圧力方程式解法	Bi-CGStab法
空間離散化	2次精度中心差分
時間進行法（対流項）	2次精度Adams-Bashforth法
時間進行法（拡散項）	Crank-Nicolson法

また，レイノルズ数は64500，時間ステップは1.0×10^{-6}，収束判定残差は1.0×10^{-8}である．

Table 2　対象並列計算機

	SGI Origin2000	HITACHI SR8000/MPP
OS	IRIX6. 5	HI-UX/MPP
最大PE数	512 (32)	1152 (144node)
理論性能 (FLOPS)	800M / PE	1. 8G / PE
メモリ構成	分散共有メモリ	集中共有メモリ（1node）分散メモリ
並列化	自動並列化MPI, OpenMP	MPI, PVM, OpenMP

3. MPIによる逐次コードの並列化

並列計算機において，大規模な流体解析などの数値計算を行うためには，MPI（Message Passing Interface）やPVM（Parallel Virtual Machine）などの並列ライブラリを用いて，従来の逐次処理プログラムを並列化することが必要になる．その中で，MIMDマシンで最も汎用性が高くかつ将来有望である並列ライブラリはMPIである．それを踏まえ本研究では逐次コードをMPIにより3次元方向に計算領域分割することで並列化した．部分領域では逐次計算を行い，接続境界では派生データタイプを用いて通信によるデータ交換を行う．また，メモリ使用量削減並びにキャッシュミスによる実行速度低下対策として配列のアロケートも行った．

逐次コード並列化の方針として以下の点を挙げる．

・ソースコードの変更量を少なくする
・使用するMPIライブラリ関数を少なくする
・データ転送における通信回数を少なくする

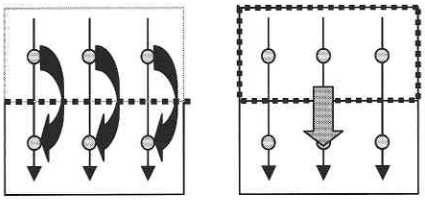
ソースの変更量は少なくし，できるだけ機械的な変更ですむようにする．使用するMPIライブラリ関数は10種類程度のみとすることで，並列コードの単純化を図る．

*東京大学生産技術研究所　情報・システム部門
**東京大学生産技術研究所　人間・社会部門

4. 接続境界でのデータ送受信

4.1 派生データタイプ

メモリ上でデータはi方向に連続である．i方向に分割を行った場合，i方向同士でデータ通信を行う必要がある．通信にはMPI_SEND，MPI_RECVを利用するが，連続データしか通信できない．i方向同士でデータ通信を行う場合，通信データがメモリ上で不連続であるためデータごとに通信を行う必要が生じ，結果として通信回数が多くなる．通信回数増大により通信立ち上がり時間が増大し，通信スループットに大きく影響を与える．不連続データをいったん新しく定義した送信バッファに圧縮して入れた後送信し，受信側でその圧縮されたデータを元の不連続データに戻す方法もある．この場合通信回数は1回ですむが，不連続データを圧縮し元に戻す作業のためパフォーマンスが若干低下する．また，ソースもわかりづらくなる．そこで派生データという機能を用いる．ユーティリティー・サブルーチンとしてPARA_TYPE_BLOCK 3（3次元）が用意されている（ref. AOYAMA）．それによって不連続なデータを1つのまとまった新しいデータタイプとして扱うことが可能となり，不連続データを圧縮し元に戻す作業によってソースが複雑になることなく1回の通信で全データを送受信可能となる（Fig. 1）．



通信回数3回　　通信回数1回

(a) 派生データタイプなし　　(b) 派生データタイプあり

Fig. 1　派生データタイプ

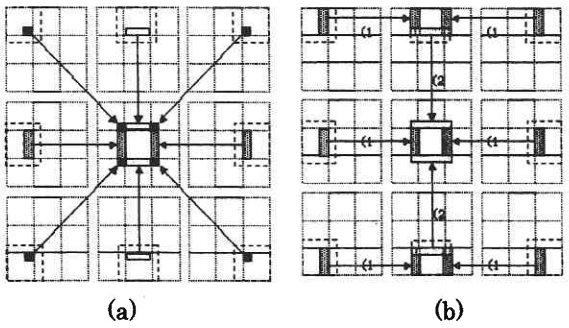


Fig. 2　接続境界でのデータ通信

4.2 通信回数を減らすデータ送受信方法

本研究で使用するコードの特徴として，あるセルについて計算を行う際斜めの要素を参照する．ここでは簡単のため説明は2次元で行う．（Fig. 2（a）(b)）中心の計算領域において計算を行う場合，必要なデータを周囲の計算領域から転送してもらう必要が生じる．(a) の方法は直感的に分かりやすいが，通信回数が多くなってしまい，その分並列化効率が悪くなる．それを改良した方法が (b) である．この場合データ転送の順序が重要であり，j方向，i方向の順番でデータ転送を行う．（3次元の場合k方向，j方向，i方向の順番）それによって最終的に少ない通信回数で斜め成分も得ることが可能となる．

5. コードの性能評価

コードの性能をメモリ使用量，FLOPS値で評価する．ここではメモリ使用量とFLOPS値について定式化を行う．メモリ使用量を定式化することで実行前にどれだけメモリを使用するか予測可能である．また，FLOPS値の定式化については姫野ベンチとの相関を考える．もし関係式が得られればPE数から姫野ベンチでのFLOPS値が分かり，そのFLOPS値から対象コードのFLOPS値が予測可能である．対象コード（行列の反復解法コード）のFlop数は反復回数さえ分かれば予め計算可能であり，Flop値とFLOPS値よりコードの実行時間が予測可能である．

5.1 メモリ使用量

SR 8000におけるPE数とメモリ使用量の相関をFig. 3に示す．全ての配列をアロケートしているためPE数に対してリニアにメモリ使用量が増加しない．これにより大規模シミュレーションが可能となる．

総メモリ使用量は以下の様に定式化可能である．

（総メモリ使用量）＝（Staticデータ）×PE数
＋（アロケートした配列のメモリ使用量）……(1)

ここでStaticデータとはsizeコマンドで得られた値であ

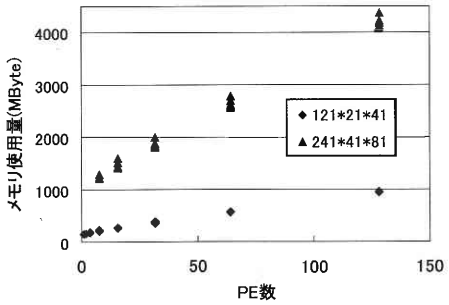


Fig. 3　メモリ使用量（SR8000）

る．

メモリ使用量はコードの実行前に格子数と分割方法のパラメータから予測可能となる．

5.2 FLOPS値

対象コードの特徴として，実行時間の約96％が行列計算サブルーチン（Bi-CGStab法）である（ホットスポット）．Bi-CGStab法において，接続境界通信は1反復につき6回行う．対象行列は17重対角対称帯行列である．一方流体計算の性能を予測するベンチマークテストである姫野ベンチは，ヤコビ法による行列解法でありアルゴリズムは異なる．しかし対象行列は対象コードと同じ17重対角対称帯行列であり，LinpackやSPEDCfpと比較して，より対象コードの比較対象として適していると考える．

対象コードのFLOPS値を姫野ベンチのFLOPS値と比較するために，対象コード中の行列計算サブルーチンと格子数を本研究対象の格子数と同じにした姫野ベンチについて，FLOPS値をOrigin2000，SR 8000上で計測した．Origin2000は格子数約10万点（121×21×41），SR 8000は格子数約10万点，約80万点（241×41×81）についてPE数1～128で計測した．ただし，SR 8000において擬似ベクトルオプションである-pvecオプションはつけていない．同PE数については様々な分割方法で計測した．例えばPE数が8の場合，Fig. 4にある通り，10通りの分割方法が存在する．

Origin2000についての結果をFig. 5に，SR 8000についての結果をFig. 6に示す．またFig. 6におけるPE数と，分割方法の違いによるFLOPS値相違率の関係をTable. 3に示す．Fig. 5より，Origin2000では対象コードについてのFLOPS値は姫野ベンチのFLOPS値と比較して約10～20％ほど値が大きいことが分かる．（姫野ベンチのFLOPS値）×1.15（MFLOPS）と計算可能である．

Fig. 6において，(a)，(b) 両ケースとも対象コードと姫野ベンチのFLOPS値の関係はほぼ同じ傾向であるので，格子数約80万点について検証をFig. 7（PE数8～64）で行う．

Fig. 7における姫野ベンチ（241×41×81，256×128×128），対象コードピーク値の関係式を以下に示す．

［姫野ベンチ（241×41×81）］＝
［姫野ベンチ（256×128×128）］×（0.85~0.90）‥(2)

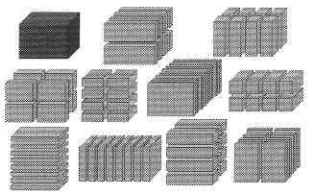


Fig. 4　PE数8における分割方法

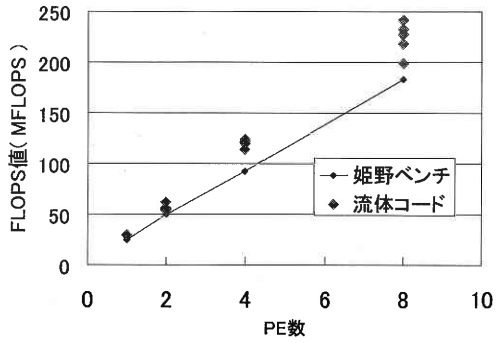
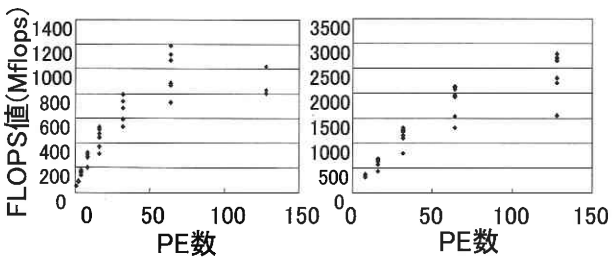


Fig. 5　FLOPS値（Origin2000）



(a)約10万点(121×21×41)　(b)約80万点(241×41×81)

Fig. 6　FLOPS値（SR8000）

Table 3　FLOPS値相違率と最良・最悪分割方法（i, j, k）

格子数	2	4	8	16	32	64	128
121×21×41	8. 94%	22. 4%	38%	40. 5%	32. 6%	38. 4%	
最良分割方法	(1,1,2)	(1,2,2)	(1,2,4)	(1,2,8)	(2,4,4)	(2,4,8)	
最悪分割方法	(2,1,1)	(4,1,1)	(8,1,1)	(8,2,1)	(8,4,1)	(8,4,2)	
241×41×81			16. 7%	37. 4%	39. 8%	39. 1%	44. 8%
最良分割方法			(1,1,8)	(2,2,4)	(2,4,4)	(1,4,16)	(2,4,16)
最悪分割方法			(8,1,1)	(16,1,1)	(16,2,1)	(16,4,1)	(16,8,1)

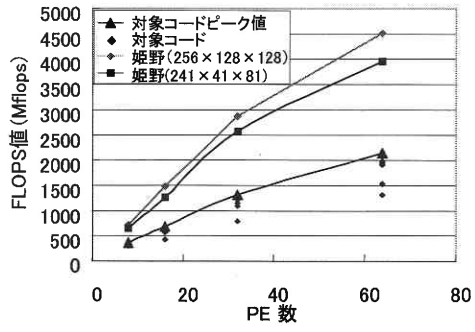


Fig. 7　FLOPS値（SR8000約80万点）

［対象コードピーク値］＝
［姫野ベンチ（241×41×81）］×（0.51~0.57）･･･(3)

ここでPE数に関わらず各比例定数がほぼ同じ値に収まることは意味があると考える．すなわちPE数が64以上でも上式の比例定数がほぼ同じ値をとると予測する．

Origin2000において対象コードのFLOPS値が姫野ベンチのFLOPS値より良い結果が得られた主な原因として，Origin2000のメモリアクセス能力が挙げられる．姫野ベンチの行列解法であるヤコビ法はメモリアクセスが非常に多いアルゴリズムである．SR 8000ではメモリアクセス能力が優れているので，通信時間の影響が大きい対象コードのFLOPS値が悪くなる．一方Origin2000はメモリアクセス能力がSR 8000より低く，姫野ベンチにおいてメモリアクセス時間の影響がより大きい．それは対象コードにかかる通信時間より影響が大きく，姫野ベンチのFLOPS値が悪くなると考える．

また，PE数増加によってFLOPS値がサチュレートするが，原因はPE数増加による通信時間の増大である．さらにTable 3より，同PE数でも分割方法によってFLOPS値に大きな開きがあることが分かる．主な原因は通信時間の差にあると考えられる．16 PE以上で分割方法による相違率が大きくなっているのは，16 PE以上ではノード間通信が発生し，ノード内通信のみだった8 PEまでの場合に比べ通信時間に開きが生じやすくなるためである．

以上より，Flop値が分かっていれば，未知のマシンで対象コードを実行させる時，姫野ベンチの結果から予めコード実行時間が予測できる可能性を示した．Flop値を知るには，反復回数が必要である．反復回数は格子数，タイムステップ，格子形状，対象流れ場などに依存する．

5.3　SR8000 1024PEsによる計算

次に格子数約80万点について，PE数8～1024でのFLOPS値計測結果をFig. 8に示す．各PE数において計測したFLOPS値のピーク値をつないでいる．1024においてプロットが1つしかないが，東京大学情報基盤センターにおいて1024 PEs使用可能な時間が制限されているためであり，そのためそれまでの計測経験から明らかに最もFLOPS値がよいと予測される分割方法で計測している．1024 PEsでは分割方法の制限により，(i, j, k) = {(16, 8, 8), (8, 16, 8), (8, 8, 16)} の3つの分割パターンしかない．k方向の分割数が多い方がFLOPS値がよいため，(8, 8, 16) で領域を分割しFLOPS値を計測した．図から明らかなように，PE数増加に伴いFLOPS値はサチュレートしている．主な原因として粒度が小さくなることによる通信時間割合の増大，コードの性質上生じる分割方法の制限が挙げられる．そのため，PE数小では，

［対象コードピーク値］＝
［姫野ベンチ（241×41×81）］×（0.51~0.57）･･･(3)

であった関係式が，PE数大（PE数1024近辺）において，

［対象コードピーク値］＝
［姫野ベンチ（241×41×81）］×（約0.25）･････(4)

となる．この問題については格子数を増大させることによって，0.25の値は（0.51～0.57）の値に近づくと考えられる．

Bi-CGStab法以外のメインルーチンにおける，PE数と実行時間（1 step）の関係についてTable 4に示す．通信時間の影響によりある程度サチュレートしているものの，PE数増加に伴い速度向上していることが分かる．

これまで-pvecオプション無しについてFLOPS値の検証を行ってきた．Table 5に-pvecオプションの有無によるFLOPS値の比較結果を示す．PE数増加により，-pvecオプション有無によるFLOPS値増加率低下の様子が分かる．また式（3）は-pvecオプション無し条件下で得られたものであり，-pvecオプションあり条件下で式（3）がどのようになるか考える必要がある．姫野ベンチ，対象コード共に同PE数に対して同じようなFLOPS値増加率であるため結果は式（3）とほぼ同じ式となる．

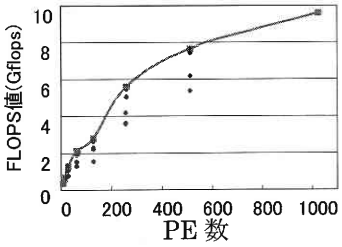


Fig. 8　FLOPS値（SR8000 1024 PEs）

Table 4　Bi-CGStab法以外における実行時間

PE数	Time(sec)
8	6.467455
64	0.977977
256	0.302398
512	0.188391
1024	0.132547

Table 5　-pvecオプション有無によるFLOPS値

（上）姫野ベンチ（下）対象コード

分割方法 (i, j, k)	-pvec無し (Mflops)	-pvec有り (Mflops)	-pvec無し /-pvec有り
(1,1,8)	508.48	2383.22	4.69
(1,2,8)	1481.06	4628.11	3.12
(1,4,8)	2866.43	8684.05	3.03

分割方法 (i, j, k)	-pvec無し (Mflops)	-pvec有り (Mflops)	-pvec無し /-pvec有り
(1,1,8)	368.33	1493.99	4.06
(2,2,4)	682.26	2129.33	3.12
(2,4,4)	1315.41	3562.25	2.71
(1,4,16)	2140.75	5353.147	2.50
(2,4,16)	2795.32	6744.036	2.41

6. 通信時間

分割方法の違いによってFLOPS値に開きが生じてしまう主な原因は通信時間の差にあると考えた．すなわち同PE数で最も通信時間が少なくなる分割方法が最も適した分割方法である．もし通信バイト数と通信時間の関係が定式化できれば，格子数と分割方法のパラメータから通信時間が計算可能である．それによって，最適分割方法を見つけることが可能となる．

SR 8000上とOrigin2000上で通信バイト数と通信時間の関係を調べた．対象コードにおいて通信する配列のインデックスは100～1000のオーダであるため，通信バイト数（0～8000 [Byte]）について測定した．結果をFig. 9，10，11に示す．Fig. 9において曲線は実測データであり，直線は曲線の近似直線である．Fig. 9の近似直線は以下のように定式化できる．（x: 通信バイト数，y: 通信時間）

$$y = 10^{-8}x + 0.0001 \cdots\cdots (5)$$

（y切片: T latency，傾き: 1/（バンド幅 [Byte/s]））

SR 8000について，ノード内通信はFig. 10より，

$$y = 3 \times 10^{-9}x + 0.00002 \ (0 < x < 1024) \cdots\cdots (6)$$

$$y = 1 \times 10^{-9}x + 0.00003 \ (1024 < x < 4096) \cdots\cdots (7)$$

$$y = 1 \times 10^{-9}x + 0.00004 \ (4096 < x < 8000) \cdots\cdots (8)$$

またノード間通信はFig. 11より，

$$y = 2 \times 10^{-8}x + 0.0001 \ (0 < x < 1024) \cdots\cdots (9)$$

$$y = 2 \times 10^{-8}x + 0.0003 \ (1024 < x < 4096) \cdots\cdots (10)$$

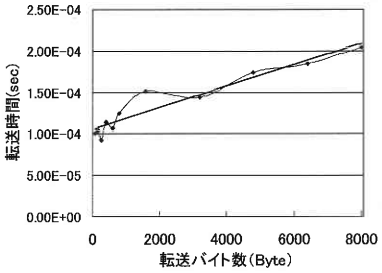


Fig. 9　通信時間（Origin2000）

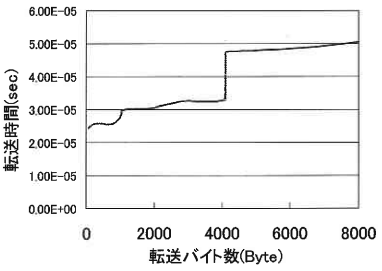


Fig. 10　通信時間（SR8000ノード内）

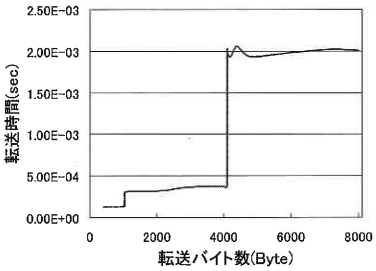


Fig. 11　通信時間（SR8000ノード間）

$$y = 8 \times 10^{-9}x + 0.0019 \ (4096 < x < 8000) \cdots\cdots (11)$$

である．この場合，ノード間通信・ノード内通信にかかわらず，通信バイト数1024と，4096でステップ状に通信時間が増大した．また，通信速度はノード内通信の方が数十倍速いことが分かる．

以上の定式化により，あらかじめ格子数とプロセッサ数，分割方法といったパラメータから通信回数，各通信におけるデータ量が計算可能であり，通信時間が予測できる．これによりコードコンパイル前に，通信時間の減少に特化した場合の最適分割方法が予測可能となる．

7. 結言と今後の課題

実用LES流体シミュレーションコードの並列化ニーズを受け，MPIによる並列化を行い，その評価を行った．

今後の課題として，通信時間の削減と並列入出力の2点

が挙げられる．

通信時間の削減について，格子数約10万点で性能モニター機能（-pmfunc）を用いてBi-CGStab法の通信時間割合を計測した結果，約68％であった．格子数nに対して計算時間はO（n），通信時間はO（$n^{\frac{2}{3}}$）のオーダであり，通信時間の占める割合は格子数増大により減少することが容易に想像できるが，それでも非常に大きい．SR 8000に限定した場合，LightMPIが用意されている．これはMPI-1機能のうち，主要な機能のみを使用することで実行時間の速度向上を狙ったものであり，通信においては立ち上がり時間がMPI-1の通信関数の立ち上がり時間と比較して半分になることが報告されている．今後LightMPIの使用を検討する必要がある．ただし，LightMPIを使用することはSR 8000に特化したプログラミングであり，コードの汎用性に欠ける．

並列入出力について本研究では，MPI-1の機能しか用いていない．例えば入力についてnode0に全データをreadした後，全てのnodeに各node担当データを送信する方法を取っている．MPI-2の並列入出力関数を用いることで飛躍的に性能が向上する報告がなされており[4]，今後入出力部分についてMPI-2で用意される関数の使用を検討する必要がある．

8. 謝　　辞

本研究の一部は文部科学省ITプログラム「戦略的基盤ソフトウェアの開発」プロジェクトの一環として行われた．東京大学情報基盤センターSR 8000のプログラム開発と計算実行には本センター黒田久泰氏のご助言を頂いた．ここに謝意を表する．

（2002年12月9日受理）

9. 参 考 文 献

1) http://w3 cic.riken.go.jp/HPC/HimenoBMT/
2) スーパーコンピューティングニュース Vol. 1 No. 3 1999. 9 東京大学情報基盤センター（スーパーコンピューティング部門）
3) PCクラスタと流体のための並列化プログラミング（2002. 5）社団法人 日本機械学会
4) 実践MPI-2メッセージパッシング・インタフェースの上級者向け機能 ウイリアム・グロップ，ユーイング・ラスク，ラジーブ・タークク著　畑崎隆雄訳