数 値 解 析 37－15
（１９９１．７．１７）

階層メモリ型スーパーコンピュータにおける並列化技法

安藤 憲行、中村 絹代[1]、西 直樹

日本電気C&Cシステム研究所、[1]航空宇宙技術研究所

計算空気力学(CFD)コードを効率良く並列ベクトル実行するためのアルゴリズムを示し、ローカルメモリを有するベクトルプロセッサが共有メモリに結合されたスーパーコンピュータを計算機モデルとして評価を行なった。本プログラムではスイープ方向が3軸方向に対し順次変化していくため、データのローカルメモリへの割り当て方を最適化しないと、共有データの入れ替えのため多量のデータ転送が発生してしまう。そこで、並列実行において最大限の参照の局所性が得られるように空間の切り方を変え、プロセッサと共有メモリ間とのデータ転送を低減し高速化をはかっている。また、最大値計算の並列化も行なった。これにより、128プロセッサを用いた場合の並列化加速率が、オリジナルプログラムを単純に並列化した場合の30.5倍から、81.3倍に向上した。

Parallelization Techniques for a Supercomputer with Hierarchical Memory

This paper describes CFD (Computational Fluid Dynamics) algorithm on a highly parallel supercomputer with hierarchical (common / local) memory system. Our original CFD code uses IAF (Implicit Approximate Factrization) scheme and is fully vectorized. We developed several kinds of sub-space mapping methods to find the best parallelization. Localizing memory accesss is the most important point of our optimization. Minimizing serial portion of the program and enlarging process granularity are general techniques for parallelism. Simulation results for 128 processor system estimate that this algorithm has a speed-up factor of 81.3 compared to the speed-up of 30.5 for the original algorithm.

1．はじめに

近年、さまざまな分野で大規模かつ高速な科学技術計算に対する要求が高まって来ている。この高速化要求に答えるため、現在のスーパーコンピュータではデバイス技術による単体プロセッサの高性能化に加え、マルチプロセッサ構成(図1.1[B])を採用するものが増えつつある。しかし、プロセッサ台数を単純に増やしても、共有メモリのみの密結合型ではメモリアクセス競合が頻繁に発生し台数効果が得られなくなる。そこで、数十台以上のマルチプロセッサ構成においては、メモリアクセス競合を低減させるため各プロセッサにローカルメモリを持たせることが必須と考えられる(図1.1[C])。ローカルメモリを持たせることにより、計算空気力学（CFD）等の並列性が良く、データの局所性が高いアルゴリズムに対して十分な台数効果を得ることができる。

このメモリ階層型の特徴をいかし、並列型スーパーコンピュータの性能を十分に引き出すためには、並列化／ベクトル化部分の切り分け、各変数領域の割り当て位置、さらには、ローカル／共有メモリの容量やスループット等に注意を払ったプログラミングをする必要がある。

本稿では、CFDコードを評価対象にし、並列ベクトル化実行方法について述べている。また、階層メモリ型に適するようにプログラムの書き換えを行ない、その効果を求めている。

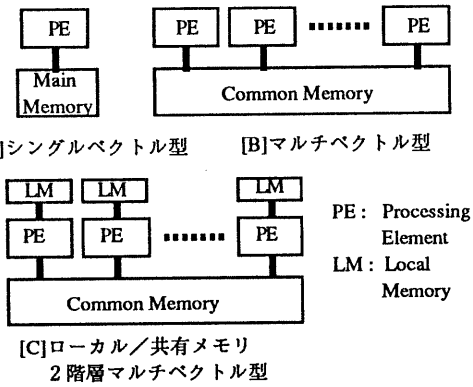

図1.1 スーパーコンピュータ分類

2. NALTESTプログラム

2.1 プログラムの概要

評価対象プログラムは三次元薄層近似ナビエストークス方程式をTVD差分を用いて差分化し、その差分方程式をIAF法で解くものである。

基礎方程式（２方向薄層近似）は以下の通りになる[1]。ここで、$Q_\tau, E_\xi, F_\eta, G_\zeta, S_{1\eta}, S_{2\zeta}$は5次元のベクトルである。

$$Q_\tau + E_\xi + F_\eta + G_\zeta = (S_{1\eta} + S_{2\zeta}) / Re$$

この基礎方程式にIAF法（基本的にはADI法であるが高次の項を付け加えて因数分解する）を適用する。Jは物理空間から計算空間に座標変換をするヤコビ行列である。

$$\begin{aligned}
&(I + h\delta_\xi A^n - \varepsilon_1 J^{-1} \nabla_\xi \Delta_\xi J) \\
&(I + h\delta_\eta B^n - \varepsilon_1 J^{-1} \nabla_\eta \Delta_\eta J) \\
&(I + h\delta_\zeta C^n - \varepsilon_1 J^{-1} \nabla_\zeta \Delta_\zeta J)(q^{n+1} - q^n) \\
&= -\Delta t \Big(\underline{\delta_\xi E^n + \delta_\eta F^n + \delta_\zeta G^n} \\
&\qquad \underline{- Re^{-1}\delta_\eta S_1^n - Re^{-1}\delta_\eta S_2^n} \Big) \\
&\quad - \varepsilon_E J^{-1}((\nabla_\xi \Delta_\xi)^2 + (\nabla_\eta \Delta_\eta)^2 + (\nabla_\zeta \Delta_\zeta)^2) J q^n
\end{aligned}$$

δ: 中心差分オペレータ
Δ: 前進差分オペレータ
∇: 後進差分オペレータ
A, B, C : E, F, Gのヤコビ行列

TVD法[2]のため上式の下線部は、この部分にさらに補正項を加えて変形されている。

この差分方程式をプログラム化して解く場合、

$$A_\xi A_\eta A_\zeta \Delta Q^{n+1} = R^n$$

と考えると分かり易い。

右辺のR^nは、

$$R^n = (L_\xi + L_\eta + L_\zeta) Q^n$$

で表され、Q^nにそれぞれL_ξ、L_η、L_ζの作用素を施したものの和であり、これを右辺として、この方程式を、

$$A_\xi x = R^n$$
$$A_\eta y = x$$
$$A_\zeta \Delta Q^{n+1} = y$$

として、ΔQ^{n+1}を求めることになる。そして、

$$Q^{n+1} = Q^n + \Delta Q^{n+1}$$

として、新しいQ^{n+1}を求めることができる。この演算を繰り返すことになる。

プログラム構造を図2.1 に示す。プログラムはデータの初期設定を行なう前処理部、繰り返し部、データの出力を行なう後処理部の３つの部分に分かれ、繰り返し部はSUB1～SUB6サブルーチンの６つのサブルーチン、処理１および処理２からなる。

SUB1～SUB3サブルーチンは差分方程式の右辺にあたるR^nを求めるサブルーチンである。処理１では求められたR^nを図2.1に示すように変換する。（処理過程は簡略化されている）。SUB4～SUB6サブルーチンはブロック３重対角行列を解くサブルーチンである。処理２では求められた解をQ^nに加算してQの更新を行ない、解の和をプリントアウトした後、収束判定を行なう。

格子点数は1024×512×256であり、現在解かれている問題規模の数100倍程度の大きさである。

(0) 前処理　（変数の初期化）
(1) SUB1　$R^n = L_\xi Q^n$
(2) SUB2　$R^n = R^n + L_\eta Q^n$
(3) SUB3　$R^n = R^n + L_\zeta Q^n$
(4) MAIN : 処理1

```
A = MAX( EGNMAX(I) )
B = MAX( C*RHS( I,J,K,1) / Q( I,J,K,1 ) )
DTIME = F( A,B ),  F:function
RHS( I,J,K,N ) = DTIME * RHS( I,J,K,N )
```

(5) SUB4　$A_\xi x = R^n$
(6) SUB5　$A_\eta y = x$
(7) SUB6　$A_\zeta \Delta Q^{n+1} = y$
(8) MAIN : 処理2

```
Q( I,J,K,N ) : INTERNAL POINT UPDATE
             : BOUNDARY POINT UPDATE
RHS( I,J,K,N ) : SUNMATION
収束判定
```

(9) 後処理　(WRITE DISK)

図2.1 NALテストプログラムの概要

2.2 プログラムの並列化

オリジナルプログラムを並列ベクトル実行する場合、まず並列化の対象になるのは、サブルーチン部の多重DOループである。各サブルーチンは３重DOループで構成される。最外、最内DOループはデータ依存関係が無いため並列化が可能である。２重めのDOループはスイープ方向であり、SUB4～SUB6ではデータ依存関係があるため、この方向での並列化はできない。

SUB1～SUB6のDOループ構成を表2.1に示す。本プログラムを並列化する場合、その構造を考慮すると３重DOループの最外DOループの指標により分割した平面で各APが処理する方法となる。こうすると、SUB1、SUB2、SUB4、SUB5はK方向を適当に切って各APに数枚の(I,J)平面の計算を担当させることになる。さらに、SUB3、SUB6はJ方向に適当に切って数枚の(I,K)平面の計算を担当させることになる。また、ベクトル化は最内DOループに対して行うことにする。

表2.1　各３重DOループ構成

DOループ	SUB1	SUB2	SUB3	SUB4	SUB5	SUB6
最外	K	K	J	K	K	J
中	I	J	K	I	J	K
最内	J	I	I	J	I	I

配列変数からみたサブルーチンの機能を表2.2 に示す。配列Q は前述の Q^nに対応し、配列RHSはR^nに対応する。配列RMETRはメトリックJ に対応する。SUB1サブルーチンは、配列Q,RMETRを入力し、SUB2～SUB6サブルーチンは、配列Q, RMETR, RHSを入力し、各々配列RHSを出力する。プログラム的には配列RHSを各サブルーチンで引き継いでいるが、SUB1～SUB3については各サブルーチンで独立に求めて最終的にその和を取ってもよい。最終的には処理2で配列Qの更新を行ない、次のステップに進む。

表2.2　配列からみたサブルーチンの機能

N = I ×J ×K

	入力		出力	
	配列名	量	配列名	量
SUB1	Q(I,J,K,5)	5N	RHS(I,J,K,5)	5N
	RMETR(I,J,K,9)	9N	EGNMAX(I)	
SUB2	Q(I,J,K,5)	5N	RHS(I,J,K,5)	5N
～	RMETR(I,J,K,9)	9N	EGNMAX(I)	
SUB3	RHS(I,J,K,5)	5N		
	EGNMAX(I)			
処理1	Q(I,J,K,5)	5N	DTIME	
	RHS(I,J,K,5)	5N		
	EGNMAX(I)			
SUB4	Q(I,J,K,5)	5N	RHS(I,J,K,5)	5N
～	RMETR(I,J,K,9)	9N		
SUB6	RHS(I,J,K,5)	5N		
処理2	Q(I,J,K,5)	5N	Q(I,J,K,5)	5N
	RHS(I,J,K,5)	5N	RESMX	
	RMETJ(I,J,K)	N		

2.3 プログラムチューニング

高速化のために、2種のプログラムチューニングを行なった。一つは、ローカルメモリにマッピングされる共有変数のデータ領域を最適化し、PEと共有メモリ間のデータ転送を削減させるもの、もう一つは、MAINルーチン中で逐次処理で実行していた部分を並列化するものである。

SUB3はJ方向の並列であり、前後のルーチンがK方向の並列化であるため、SUB3の前後に行列の転置の様なデータの切り替えが発生し、多量のデータ転送が必要となる。そこで、SUB3をK方向で並列化すれば、このデータ転送は行なわなくても済む。従って、サブルーチンSUB1～SUB5までの処理をK方向で分割して(I,J)平面の実行を各PEが担当する。この分割割り付けを行なった後、SUB1～SUB5までを実行させる。ここで注意を要することはSUB3においてはK方向がシリアル処理となるため、Kの両端では計算が他のKとは異なる。従って、その処理を行なうため、サブルーチンを３つ用意することになる。

SUB6では、スイープ方向にデータ依存関係があるため、SUB3のようなスイープ軸方向の並列化はできない。そのため、SUB6ではJ方向分割を行ない、(I,K)平面の実行を各PEが担当することになる。

3　並列計算機モデル

今回CFDコード評価のための並列計算機モデルとして、ローカルメモリを持つベクトルプロセッサと、共有メモリから構成される２階層メモリ型マルチプロセッサ構成を用いる（図1.1[C]）。プロセッサ台数は１６、３２、６４、１２８台とし、各々について性能向上率を比較検討している。プロセッサの性能はＳＸ－２相当と見なし、単体の実行時間は、本CFDコードの小規模版をSX-2で実行することにより算出している。プロセッサと共有メモリ間のスループットは12.8GB/secと仮定している。共有データの転送時間はシミュレーションにより求めている。

メモリ構成はシステム全体の性能を決定する要因の１つであり、高いスループットと短いアクセス時間を実現する必要がある。しかし、高並列型の共有メモリにおいては、多くのプロセッサ間との接続を実現するために実装等の制約から、単体プロセッサあたりのスループットは低く抑えられてしまう。また、アクセスタイムも、多数のメモリアクセスの競合チェック等により長くなってしまう。競合が発生すれば、さらにアクセスタイムは伸びる。そこで、前述の条件を満足するローカルメモリを各プロセッサに持たせることにより、性能劣化を避けることができる。しかし、各プロセッサで持ち回る変数に対しては、共有メモリアクセスが必要になる。

ここで、２種の変数の定義をする。自プロセッサのみで閉じて使用され、他プロセッサから参照定義されない変数はローカルメモリに割り付ける。この変数をローカル変数と言う。また、複数個のプロセッサ間で持ち回る変数は共有メモリに置く必要がある。この変数を共有変数と言う。

4　評価

4．1　CFDコード実行方法

オリジナルプログラムであるプログラム0では、各サブルーチンで最内ループをベクトル実行、最外ループをPE毎の並列化実行で処理する。このとき、PE台数をｎとすると、SUB１、SUB２、SUB４、SUB５では、各PEは(256/n)個の(I,J)平面を担当し並列実行を行うことになる。また、SUB３、SUB６では、各PEは(512/n)個の(I,K)平面を担当することになる。MAINルーチン内の処理においては、１台のPEがシングル実行で処理する。ただし、各サブルーチンの前後に共有メモリとのデータ転送が行なう必要がある。各PEはサブルーチン内の計算の前に、参照する必要がある共有変数を共有メモリより転送し、ローカルメモリ内にそのコピーを作る。サブルーチン内での計算が終了すると、更新された共有変数のみを共有メモリに書き戻す。

SUB1、SUB2は、どちらも(I,J)平面での並列化なので、同一PEが同一平面を担当すれば、このサブルーチン間の共有データ転送は不要になる。このことはSUB4、SUB5間でも言える。プログラム0'は、同じPEが同じ平面を割り付けられるように、データ領域の割り当てを最適化したプログラムである。

プログラム１は、SUB３の並列処理単位を(I,J)平面に変え、SUB1からSUB5までを各PEが同一平面を担当し並列実行を行なうように書き換えられたプログラムである。従って、SUB1～SUB5間の共有変数のデータ転送が不要となる。しかし、SUB3とSUB4間で処理1用の共有データ転送は依然残って

いる。以下に、プログラム1の繰り返し部における、各PEと共有メモリ間での共有変数のデータ転送方法を簡単に図示する。但し、PE台数を64台とする。プログラムは(1)～(8) の順に実行される。

(1) Q,RMETR変数の共有→ローカルメモリ転送
（図4.1: 白抜きの参照平面を含む6平面）
(2) SUB1,2,3S,3K,3Eの実行
(3) 処理1の実行
(4) SUB4,5の実行
(5) RHS変数のローカル→共有メモリ転送
（図4.1：網掛けの4平面）
(6) RHS、Q、RMETR、RMETJ変数の
共有→ローカルメモリ転送（図4.2：8平面）
(7) SUB6の実行
(8) RHS変数のローカル→共有メモリ転送
（図4.2：8平面）
(9) 処理2の実行

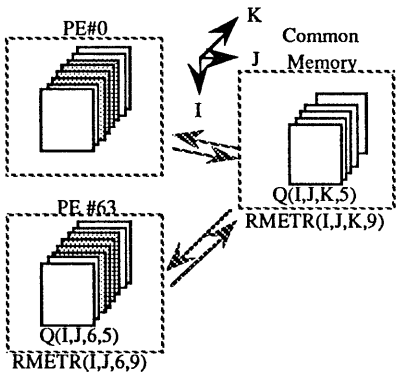

図4.1　SUB1前、SUB5後の共有データ転送

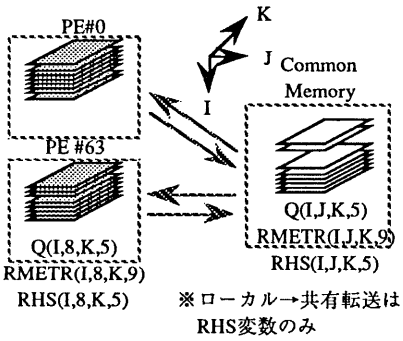

図4.2　SUB6前後の共有データ転送

プログラム１までは並列化の対象は各サブルーチンに限られていたが、MAINルーチン内の処理1と処理2の２箇所についても並列化が可能である。プログラム２は、この並列化ための書き換えを行なったプログラムである。書き換え例として処理1のMAX値計算部分の並列化について説明する。プログラム１の場合、MAX計算は以下のように実行される。ここでは配列QとRHSからMAX計算を行ない、この結果から配列RHS変数に対する更新を行なう。従って、シングル実行を行なう１台のプロセッサに対して配列Q、RHSの全空間配列の膨大な量のデータ転送が発生し、大きなオーバヘッドが生じてしまう。

```
(MAX値計算)
   RATE = 0.0D0
   DO 3510  K = 1 , KK
   DO 3510  J = 1 , JJ
   DO 3510  I = 1 , II
    RATE = MAX ( RATE ,
       C * ABS( RHS(I,J,K,1) / Q(I,J,K,1) ) )
3510 CONTINUE
   IF( RATE.NE.0 )
      DTIME = MIN( CFL/EMAX, CFL2/RATE )
        |
(配列RHS の更新)
   DO 3600  M = 1 , 5
   DO 3600  K = 1 , KK
   DO 3600  J = 1, JJ
   DO 3600  I = 1 , II
    RHS(I,J,K,M) = -DTIME * RHS(I,J,K,M)
3600 CONTINUE
```

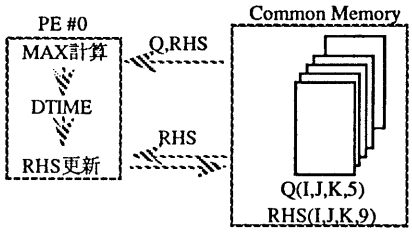

図4.3　処理1のシングル実行

プログラム２では、この部分の並列化の書き換えを行なっており、以下のように実行される。SUB3サブルーチン実行後、各プロセッサは、ローカルメモリ中に配列RHS、Qを各々持っている。そ

こで、プロセッサが部分MAX値を計算し、この結果を共有メモリに書き込む。部分MAX値から全体のMAX値の計算は1台のプロセッサによるシングル実行になるが、データ転送量、並びにMAX値計算量はプロセッサ台数の要素を持つ配列RATEMXの転送と計算で済む。得られた結果DTIMEは各PEに通知され、各PEはこの値を元に、ローカルメモリ内にある配列RHSの更新を実行する。従って、共有メモリ間のデータ転送量は、サイズがプロセッサ台数である配列RATEMXとMAX値計算結果のスカラ変数DTIMEのみに減らせる。

```
(各プロセッサ毎実行)
(RATEの部分最大値(RATEK)を求める)
      RATE = 0.0D0
      DO 3510 K = 1 , KN - 2
      DO 3510 J = 1 , JJ
      DO 3510 I = 1 , II
      RATEK= MAX( RATE ,
            C * ABS( RHS(I,J,K,1) / Q(I,J,K,1) )
3510 CONTINUE
(RATEKを共有メモリに書き戻す)
      RATEMX( KP ) = RATEK

(シングル実行によるMAX値計算)
      RATE = 0.0
      DO 3500 K = 1 , KPEN
3500 RATE = MAX( RATE , RATEMX( K ) )
      IF( RATE.NE.0 )
          DTIME = MIN( CFL/EMAX, CFL2/RATE )
          |
(DTIME を各PEのローカルメモリに転送)
      DTIMEK = DTIME

(各プロセッサ毎実行)
(RATEの部分最大値(RATEK)を求める)
```

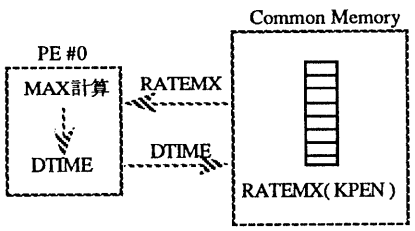

図4.4 部分MAX値によるMAX値計算

4.2 評価結果

4.2.1 必要メモリ量

始めに、本CFDコードを実行するのに必要となる共有メモリ容量と、ローカルメモリ容量を求めてみる。各変数は倍精度(8Byte)とする。主な共有変数はQ(I,J,K,5)、RMETR(I,J,K,3,3)、RHS(I,J,K,5)であり、これらの変数の全空間分確保のためには19GBの共有メモリ領域が必要になる。

ローカルメモリには、共有変数のバッファ用とローカル変数用の2種の領域が必要となる。各プログラムについて必要となるローカルメモリ容量を表4.1に示す。明らかに、各PEが担当する分担部分空間が多ければ、この空間分の共有変数のバッファ領域は大きくなるが、適当にこの空間を区切って順次実行すればバッファ領域を小さくすることができる。また、プログラム0,0' とプログラム1,2との比較では、SUB3サブルーチンで4枚の参照(I,J)平面が必要となるため、プログラム1,2のバッファ用領域は大きくなってしまう。

表4.1 必要ローカルメモリ容量

(単位 MB)

	PE32台			PE128台		
	C	L	合計	C	L	合計
prog. 0,0' SUB1,2,4,5	608	100	708	152	100	252
prog. 0,0' SUB3,6	304	50	354	76	50	126
prog. 1,2 SUB1,2,3,4,5	912	100	1012	456	100	556
prog. 1,2 SUB6	304	50	354	76	50	126

C(ommon)：共有変数のバッファ用
L(ocal) ：ローカル変数用

4.2.3 共有データ転送量

次に、共有データ転送量を比較して見る。表4.2は各プロセッサが共有メモリ間で転送しなければならないデータ転送量を示している。Nkは全(I,J)平面数を示し(Nk = 256)、Njは全(I,K)平面数を示す(Nj = 512)。また、nはPE台数とする。例えば、SUB1サブルーチン実行前の共有データ転送では(Nk / n)分の(I,J)平面を各PEに転送する必要がある

ことを示す。

プログラム0は各サブルーチンの前後で共有データ転送が発生するが、プログラム0'はSUB1 - SUB2、SUB4 - SUB5間の転送は行なう必要が無い。さらに、プログラム2は、処理1,2部での配列Q、RHSの全空間分のロード、ストアを行なう必要が無いため、かなりのデータ転送オーバヘッドが削減できる。

表4.2　共有変数データ転送量比較

	Prog.0	Prog.0'	Prog.1	Prog.2
Load	(K/n)Nij	(K/n)Nij	(K/n+4)Nij	(K/n+4)Nij
SUB 1				
Store	(K/n)Nij	-	-	-
Load	(K/n)Nij	-	-	-
SUB 2				
Store	(K/n)Nij	(K/n)Nij	-	-
Load	(J/n)Nik	(J/n)Nik	-	-
SUB 3				
Store	(J/n)Nik	(J/n)Nik	-	-
処理1	4N	4N	4N	-
Load	(K/n)Nij	-	-	
SUB 4				
Store	(K/n)Nij	-	-	-
Load	(K/n)Nij	-	-	-
SUB 5				
Store	(K/n)Nij	(K/n)Nij	(K/n)Nij	(K/n)Nij
Load	(J/n)Nik	(J/n)Nik	(J/n)Nik	(J/n)Nik
SUB6				
Store	(J/n)Nik	(J/n)Nik	(J/)Nik	(J/n)Nik
処理2	3N	3N	3N	-

N=I×J×K

Nij=I×J, Nik=I×K

プログラム1、2は、SUB3サブルーチンを(I,J)平面で並列化することによりSUB1 - SUB5間で共有変数のデータ転送が発生しないが、4枚の(I,J)平面の参照平面を余分にロードする必要がある。PE台数が多くなり並列度が高くなると担当する計算平面が少なくなるため、この参照平面ロードの転送オーバヘッドが大きくなる。表4.3は各サブルーチン前後に発生する共有変数転送に注目し、PE台数に対する転送量比較を示している。PE台数が多くなるとプログラム0'、1間で転送量の差が小さくなるのが分かる。

表4.3 PE台数による転送量比較

PE台数	プログラム1	プログラム2
128台 2平面／PE	14Nij+16Nik 1.63GB	14Nij+32Nik 2.23GB
64台 4平面／PE	20Nij+32Nik 2.67GB	28Nij+64Nik 4.45GB
32台 8平面／PE	32Nij+64Nik 4.75GB	56Nij+128Nik 8.91GB

Nij=I×J, Nik=I×K

4.2.3 性能向上率

各プログラムでの、PE台数16台～128台に対する性能向上率を表4.4と図4.5に示す。プログラム0ではPE台数が多くなると、共有データ転送ネックにより、性能向上率が飽和し並列実行の効果が薄れるのが分かる。

プログラム2で行なったMAINルーチン部の並列化の効果を見てみる。並列度が高くなると、全体の実行時間に対する逐次実行部の比率が高くなり並列化の効果が薄れてくる。さらに、逐次実行部での共有データ転送時間がかなり大きいことも影響している。これはシステム全体の高いスループットを生かさず、1台のPEと共有メモリ間のバス間のみで膨大なデータ転送を行なうことが、かなりのオーバヘッドになるからである。プログラム2ではかなり改善され、性能がPE台数に比例して向上率が上がることが分かる。

表4．4　性能向上率（単体性能比）

PE台数	16	32	64	128
プログラム0	11.63	17.97	24.43	30.54
プログラム0'	11.98	18.94	26.47	33.64
プログラム1	12.08	19.29	27.05	33.44
プログラム2	14.77	27.93	49.70	81.30

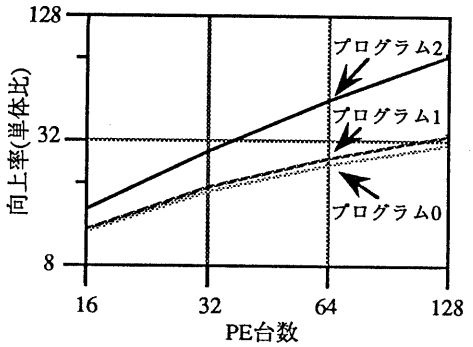

図4.5 性能向上率（単体性能比）

次に、全体の実行時間に占める、共有メモリ間のデータ転送部の割合を表4.5 に示す。プログラム０、１は共有メモリ間のデータ転送の比率が非常に大きく、これが性能向上のネックになっていることが分かる。特に、PE台数が多く並列化が十分に行なわれているときに顕著になる。

表4.5 共有メモリ間データ転送部比率（%）

PE台数	16	32	64	128
プログラム0	19.29	31.40	44.89	54.99
プログラム0'	16.80	27.70	40.30	50.41
プログラム1	14.42	24.66	36.45	47.83
プログラム2	3.54	8.25	15.94	28.07

図4.6 に、PE 64台の場合におけるプログラム書き換えの効果を図示する。共有データ転送を低減することによる性能向上が良く得られているのが分かる。

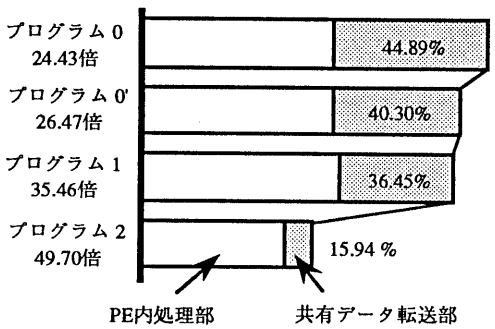

図4.6 PE 64台時のプログラム比較

5 おわりに

今回のCFDコードの評価結果では、仮定した並列型スーパーコンピュータのプロセッサ台数が128台の場合で、単体実行に対してオリジナルプログラムで30.54倍、チューニング後で81.30倍の性能向上が得られた。CFDコードを含む流体系プログラムにおいても、空間分割による並列化が容易に行なえるものが多いので、並列ベクトル処理を行なうことにより十分な高速性能が得られる可能性があると考えられる。

しかし、多数のプロセッサによる並列実行を行なう場合には、プログラムを書き換えを行なわなければ、階層メモリ型の特徴を十分に生かせず、高い性能向上率が得られないことも分かった。このとき、留意すべき点は、

（１）逐次処理部の割り合いををできるだけ小さくする。

（２）並列化方法を工夫し、ローカルメモリに割り当てるデータ領域を最適化することにより、共有メモリアクセスを低減させる。

の2点である。並列化によるチューニングというと（１）に特に注意が払われるが、（２）の最適化も重要であることが分かった。

参考文献

1) Implicit Finite-Difference Simulations of Three-Dimensional Compressible Flow; Thomas H.Pulliam,Joseph L.Steger, AIAA JOURNAL Vol.18 NO.2

2) Implicit Total Variation Diminishing(TVD) Schemes for Steady-State Calculations ; H.C.Yee, R.F.Warming, A.Harten, NASA Technical Memorandum 84342

3) 安藤、西 ; 2階層メモリを持つ密結合型スーパーコンピュータにおける並列化技法、第8回航空機計算空気力学シンポジウム論文集、SP-13