NAL TR-1434

ISSN 0389-4010
UDC 681.31

航空宇宙技術研究所報告

TECHNICAL REPORT OF NATIONAL AEROSPACE LABORATORY

TR-1434

Cenju－3システムにおける宇宙推進系プログラムの並列処理

中 村 絹 代 ・ 高 橋 政 浩

2002年1月

独立行政法人 航空宇宙技術研究所
NATIONAL AEROSPACE LABORATORY OF JAPAN

目　次

Cenju－3システムにおける宇宙推進系プログラムの並列処理*

中　村　絹　代*1　高　橋　政　浩*2

Parallel Processing of Space Propulsion System Programs on Parallel Machine Cenju-3*

Kinuyo NAKAMURA*1　Masahiro TAKAHASHI*2

ABSTRACT

The Cenju-3 system is the distributed memory parallel machine adapted in a massively parallel processor. It is easy to add a processing element (PE) to the Cenju-3 compared with a shared memory parallel machine, and it is a low cost performance compared with a parallel vector machine. This report describes the parallel processing method of doing two programs on a space propulsion system and the performance evaluation of the parallel processing of the two programs on Cenju-3. As a result, we gained a 4.6 times speed increase using 16 PEs, and an 11.9 times speed increase using 16 PEs. In addition, we have described the parallel processing method and its effect.

Keyword: massively parallel processor, Fortran program, parallel processing

概　　要

Cenju-3システムは超並列計算機（MPP）で多く採用されている分散メモリ方式の並列計算機である。共有メモリ型並列計算機より多数の要素プロセッサ（PE）を容易に接続することが可能であり、従来のスーパーコンピュータに比較して低コストである。本報告は、このCenju-3システムの性能評価のために2本の宇宙推進系プログラムに適用した並列処理手法と並列処理効果について述べたものである。その結果として、プログラムはプロセッサ数16台で4.6倍から11.9倍の速度向上を得た。また、並列処理手法とその効果を記述した。

第1章　はじめに

Cenju-3システムは超並列計算機（MPP）で多く採用されている分散メモリ方式の並列計算機である。分散メモリ型の並列計算機では各プロセッサ毎にローカルメモリと呼ばれるメモリを接続し、それを1つの要素プロセッサ（PE)としてPE間を高速な内部ネットワークで接続することにより、共有メモリ型並列計算機より多数のプロセッサを接続することが可能である。共有メモリ型の並列計算機に比較してPEの増設が容易であり、従来のスーパーコンピュータに比較して低コストである。

Cenju-3システムの並列処理性能に関する報告に関しては、Cenju-3システムのアーキテクチャを主体として、その評価を行ったもの[1)]、流体解析の種々の計算法を主体としてCenju-3システムにおいて評価を行ったもの[2)]、有限要素法による非線形変形解析の並列性の観点からCenju-3システムにおいて評価を行ったもの[3)]、などがある。しかし、Cenju-3システム関係の論文において、評価を行う上で使用された数値計算プログラムに高速化のために適用された並列処理手法の観点を含む評価を行ったものは少ない。

また、それぞれの評価においては、並列処理による高速化倍率が高いもので、PE64台を用いて60倍の性能を得たものがあり、これは使用PE数に対して93%の稼働率を示すことである。また、低い方では、PE14台を用いて5倍の高速化倍率しか得られないものもあり、この場

＊　平成13年4月5日受付（received 5 April 2001）
＊1　CFD技術開発センター（CFD Technology Center）
＊2　角田宇宙推進技術研究所（Kakuda Space Propulsion Laboratory）

合には、使用PE数に対して35%の稼働率である。これらの評価においてはそれぞれ実行条件などの設定が異なるので、ひとまとめにして語るのは困難である。しかし、Cenjuシステムに限らず、計算機の性能評価例が多く世に出ることにより、計算機の評価をより精度の良いものにできると考える。

本報告では、Cenju-3システムにおける推進系プログラム2本の並列処理に適用した並列処理手法とその並列処理効果について述べる。以下、2章ではCenju-3システムの概要を、3章では推進系プログラムの概要を述べ、4章において2本のプログラムに適用した並列処理手法及び並列処理性能について述べる。高速化のために適用された並列処理手法についてはその内容と効果を詳しく記述した。

第2章　Cenju－3システムおよび推進系プログラムの概要

2.1 Cenju－3システムの概要[4)]

Cenhu-3は日本電気株式会社の分散メモリ型並列計算機である。ハードウェアの全体構成を図2.1に示す。

Cenju-3では、8台から256台までの要素プロセッサ（PE；Processor element）が多段接続網で結合されている。各々のPEは中央処理装置（CPU；central processing unit)、一次キャッシュ、二次キャッシュ、主記憶、PE間通信用チャネルで構成されている。各PEはハードウェア面では完全に等質であり、制御プロセッサのような特定用途のPEを持たない。またPE間には共有メモリは存在しない。PE間の通信には多段のクロスバスイッチによる接続網を用いる。

ファイルとの入出力やLAN (Local Area Net- work)との接続などはフロントエンドのワークステーションEWS4800が担当する。EWS4800はCenju-3用プログラムの開発作業などにも用いられる。

(1) 要素プロセッサ

①全体構成

要素プロセッサ（PE）の構成を図2.2に示す。各PEは基本的に、

・中央処理装置（VR4400）
・一次キャッシュ（32KB：CPU内に実装）
・二次キャッシュ（1MB：高速SRAM (static random access memory)）
・主記憶（32MBまたは64MB）
・PE間通信用DMA (direct memory access)チャネル（2本）

から構成される。

②中央処理装置

要素プロセッサの中央処理装置（CPU）として、マイクロプロセッサVR4400を採用している。VR4400は以下の特徴を持つ。

・RISCアーキテクチャを採用
　MIPS社R3000シリーズおよびR4000シリーズと互換性がある。
・64ビット長のレジスタおよび整数演算ユニット
・8段スーパーパイプライン処理
　外部クロックをCPU内部で2分周することにより、1クロックサイクルで2命令を実行することができる。また、8段のパイプラインを用いることで、8個の命令をオーバーラップして実行することができる。
・メモリ管理ユニットを内蔵
　仮想メモリ空間として最大16EB（1EB　10^{18}B)、物理メモリ空間として64GBのメモリ空間を管理することができる。
・浮動小数点ユニット（FPU)を内蔵

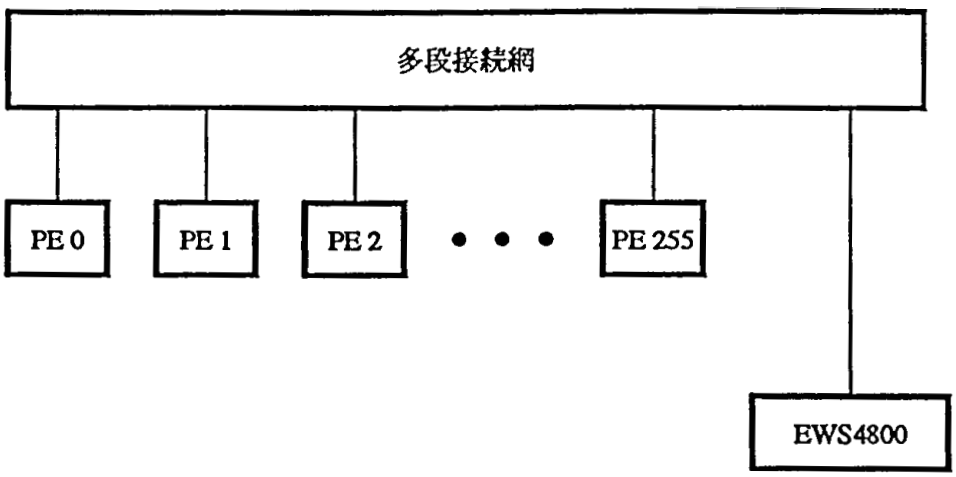


図2.1　Cenju-3システム構成図

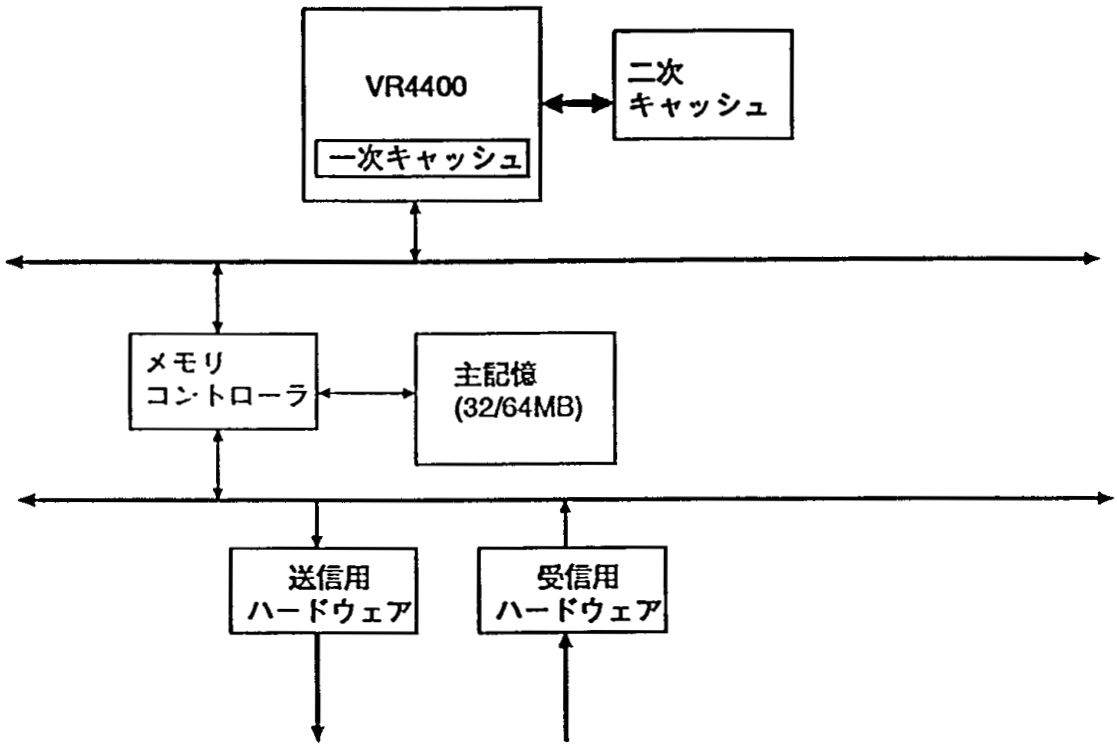


図2.2 要素プロセッサ

ANSI/IEEE標準規格754-1985「IEEE二進浮動小数点演算規格」に準拠した浮動小数点演算が可能である。FPU内部は加算器、乗算器、除算器の三つの独立した演算ユニットから構成されており、乗算器及び除算器は加算器と同時に作動させることができる。

・一次キャッシュメモリを内蔵

命令、データ用にそれぞれ16KBずつの一次キャッシュを内蔵している。

・二次キャッシュインタフェースを内蔵

（VR4400SC，VR4400MC）

③ キャッシュ

キャッシュは小容量・高速のメモリである。CPUが主記憶を参照する際、参照するメモリ番地の近傍のコピーをキャッシュ上に作っておくことで、すぐ後に再びその番地（及び近傍）を参照する必要が生じた場合に高速な参照が可能になる。通常のプログラムはこのようなメモリ参照の局所性が高いため、キャッシュはプログラム実行性能の向上に大きく寄与する。

Cenju-3の各PEはCPU内蔵の計32KBの一次キャッシュのほか、アクセス速度10ナノ秒の高速SRAMを使用した二次キャッシュを1MB備えている。

④ 主記憶

Cenju-3は各PEに32MBないし64MBの主記憶を装備している。主記憶は2バンク構成で、CPUから高速にアクセスすることが可能である。

⑤ PE間通信用チャネル

Cenju-3の各PEは、PE間通信用のチャネルを2チャネル備えており、送信専用に1チャネル、受信専用に1チャネルがそれぞれ用いられる。両チャネルは専用ハードウェアによって制御され、CPUを介さず主記憶との間で直接データを送受信するDMA (direct memory access)方式によって高速にデータの送受信を行う。両チャネルの通信速度はそれぞれ最大40MB／秒である。

(2) 多段接続網

Cenju-3ではPE間を結合する結合網としてクロスバスイッチを用いた多段接続網を採用している。図2.3に示すように、4入力4出力のスイッチングユニット（SU）を8個並べ16入力16出力の接続網を一枚の基板上に実現している。PE数が16を越える場合にはさらにこれを2段接続することにより、最大256入力256出力の接続網を構成することができる。

このような多段接続網は、他の接続網と比較して、

・すべてのPE間の距離が等しくなる。

・平均PE間距離が小さい。

・実現に必要なハードウェア量が少ない。

・デッドロックの回避が容易である。

などの特徴がある。

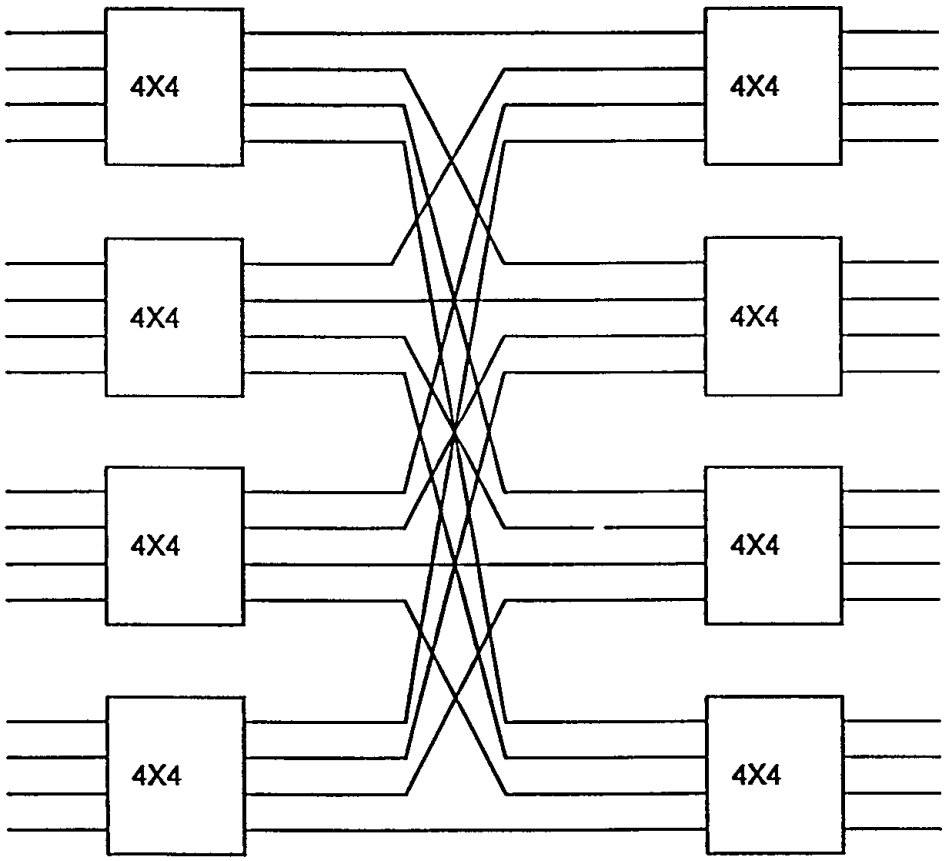


図2.3 16×16の接続網

(3) 評価に使用したシステム

Cenju-3にはPE台数およびクロック周波数などのシステム構成の違いにより表2.1に示すモデルがある。

本報告における評価に使用したモデルは「モデル16」であり、主記憶容量はPEあたり64MBのものを使用した。

(4) ソフトウェア

Cenju-3ではSPMD (single program/multiple data stream)に基づくプログラミングモデルを採用している。

表2.1 Cenju-3のモデル名とシステム諸元

		Cenju-3						
モデル		8S	16S	16	32	64	128	256
CPU		VR4400SC						
PE(CPU)台数		8	16	16	32	64	128	256
CPU外部クロック		50MHz		75MHz				
浮動小数点演算ピーク性能	PEあたり	33.3MFLOPS		50MFLOPS				
	総最大性能(GFLOPS)	0.266	0.533	0.8	1.6	3.2	6.4	12.8
命令実行ピーク性能	PEあたり	100MIPS		150MIPS				
	総最大性能(MIPS)	800	1,600	2,400	4,800	9,600	19,200	38,400
主記憶容量	PEあたり	32/64MB						
	総容量(GByte)	0.25/0.5	0.5/1.0	0.5/1.0	1.0/2.0	2.0/4.0	4.0/8.0	8.0/16.0
一次キャッシュ容量	PEあたり	32KB						
	総容量(MByte)	256	512	512	1,024	2,048	4,096	8,192
二次キャッシュ容量	PEあたり	1MB						
	総容量(MByte)	8	16	16	32	64	128	256
PE間結合網	結合形態	多段接続網						
	PEあたり転送速度	40MB/秒						
	総転送速度(MB/秒)	320	640	640	1,280	2,560	5,120	10,240

すなわち、全PEで同一のプログラムコードを持ち、各PEは自分の担当する範囲の演算及びデータ処理を行なう。

Cenju-3におけるユーザプログラムは以下の手順で実行される。

①PE0（以下、マスタPE）がユーザプログラムをロードする。
②マスタPEがユーザプログラムの実行を開始する。
③ユーザプログラムにおいて明示的にマルチプロセッサでの処理の開始が指示される。
④マスタPEが全スレーブPEにユーザプログラム等をコピーする。
⑤各PEによるユーザプログラムの並列実行が開始される。

Cenju-3上で動作する並列プログラムを開発するため、プロセッサ間の通信・同期・排他制御等を実現する並列処理記述用のライブラリPARALIB/CJが提供されている。ライブラリにはC言語用およびFortran言語用があるが、本推進系プログラムはFortran言語で記述されているのでFortran言語用のものを使用した。表2.2にPARALIB/CJの一覧表を示す。

このほか、システムソフトウェアとして以下のものを使用している。

・Cenju-3 OS，Ver1.4f
・Fortran Compiler，Ver8.00(COF2.20)

表2.2　Cenju用並列処理記述ライブラリ及びその機能

	ライブラリ名	機能
プログラム起動・終了の	ＣＪｆｏｒｋＦ	複数ＰＥでの並列処理の開始
	ＣＪａｂｏｒｔＦ	ＣＪｆｏｒｋＦで起動された全ＰＥを強制終了
	ＣＪｐｒｏｃｉｎｆｏＦ	ＰＥに関する情報取得
	ｅｘｉｔ	プログラムの終了
同期制御	ＣＪｂａｒｒｉｅｒＦ	バリアによる同期
リモートコピー	ＣＪｒｍｗｒｉｔｅＦ	リモートＰＥ上のローカルメモリへの書き込み
	ＣＪｒｍｗｒｉｔｅｍＦ	複数のリモートＰＥ上のローカルメモリへの書き込み
	ＣＪｒｍｒｅａｄＦ	リモートＰＥ上のローカルメモリからの読み出し
リモートジャプコール	ＣＪｒｐｃＦ	ブロック型リモートプロシージャコール
	ＣＪｎｂｒｐｃＦ	ノンブロック型リモートプロシージャコール
	ＣＪｎｂｒｐｃｍＦ	複数のＰＥに対するノンブロック型リモートプロシージャコール
その他の	ＣＪｇｅｔｔｍｒＦ	タイマからの読み取り

2.2 推進系プログラムの概要

(1) プログラムⅠの概要

プログラムⅠは、スクラムジェットエンジンの2次元燃焼器流れを解析するプログラムである。一般座標系ナビエ・ストークス方程式

$$\partial Q/\partial t+\partial F/\partial \xi+\partial G/\partial \eta=\partial F_V/\partial \xi+\partial G_V/\partial \eta$$

$Q,\ F,\ G,\ F_V,\ G_V$：4次元ベクトル

にKRCスキーム[5]を適用して差分方程式を求めIAF解法を用いて解いている。この方程式は次のように書くことができる。

$$A_\xi A_\eta \Delta Q^{n+1}=(L_\xi+L_\eta)Q^n$$

$L_\xi,\ L_\eta,\ A_\xi,\ A_\eta$：オペレータ

図2.4にプログラム構造及びサブルーチンの処理内容を示す。本プログラムでは、方程式の右辺を計算し、方程式を解き、ΔQ^{n+1}を求める操作を指定回数、繰り返す。

本プログラムの実行条件は格子点数601×101、反復回数2000回である。

(2) プログラムⅡの概要

プログラムⅡは、高温衝撃風洞の軸対称2次元化学反応流・非定常流れを解析するプログラムである。基礎方程式は以下の一般座標系ナビエ・ストークス方程式である。

$$\partial U/\partial t+\partial F/\partial \xi+\partial G/\partial \eta=\partial F_V/\partial \xi+\partial G_V/\partial \eta+S_r+S_s$$

$U,\ F,\ G,\ F_v,\ G_v,\ S_r,\ S_s$：11次元ベクトル

これを、Strung's Time Splittingにより、対流項＋粘性項（＋軸対称項）と化学反応生成項の式に時間分割し、対流項＋粘性項（＋軸対称項）に対しては陽的2次Runge-Kutta法を、そして化学反応生成項には陰的Crank-Nicolson法を適用し、時間2次のオペレータにより計算

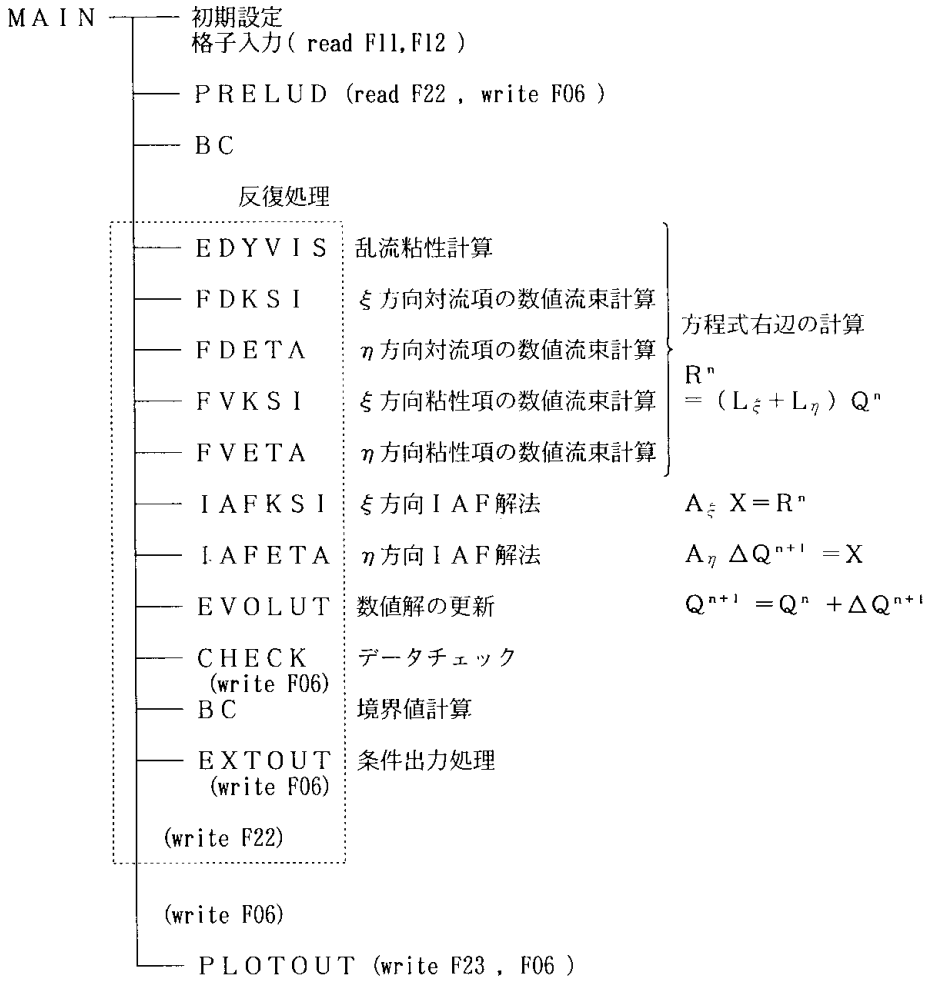


図2.4 プログラムⅠのプログラム構造

する。これによりUを求める。

L_f：　$\partial U/\partial t+\partial F/\partial \xi+\partial G/\partial \eta$
$=\partial F_v/\partial \xi+\partial G_v/\partial \eta+S_r$

L_s：　$\partial U/\partial t=S_s$

$U^{n+1}=L_f(\Delta t/2)L_s(\Delta t)L_f(\Delta t/2)U^n$

L_f：陽的2次Runge-Kutta法による
時間積分オペレータ
L_s：陰的Crank-Nicolson法

なお、空間差分については、対流項には2次精度KRCスキーム[5]、粘性項には中心差分法を用いている。

図2.5にプログラム構造およびサブルーチンの処理内容を示す。図において、MAINプログラムの前処理は初期設定及び格子生成のサブルーチン群である。後処理は本プログラムで求められた数値解や基本的な配列の磁気

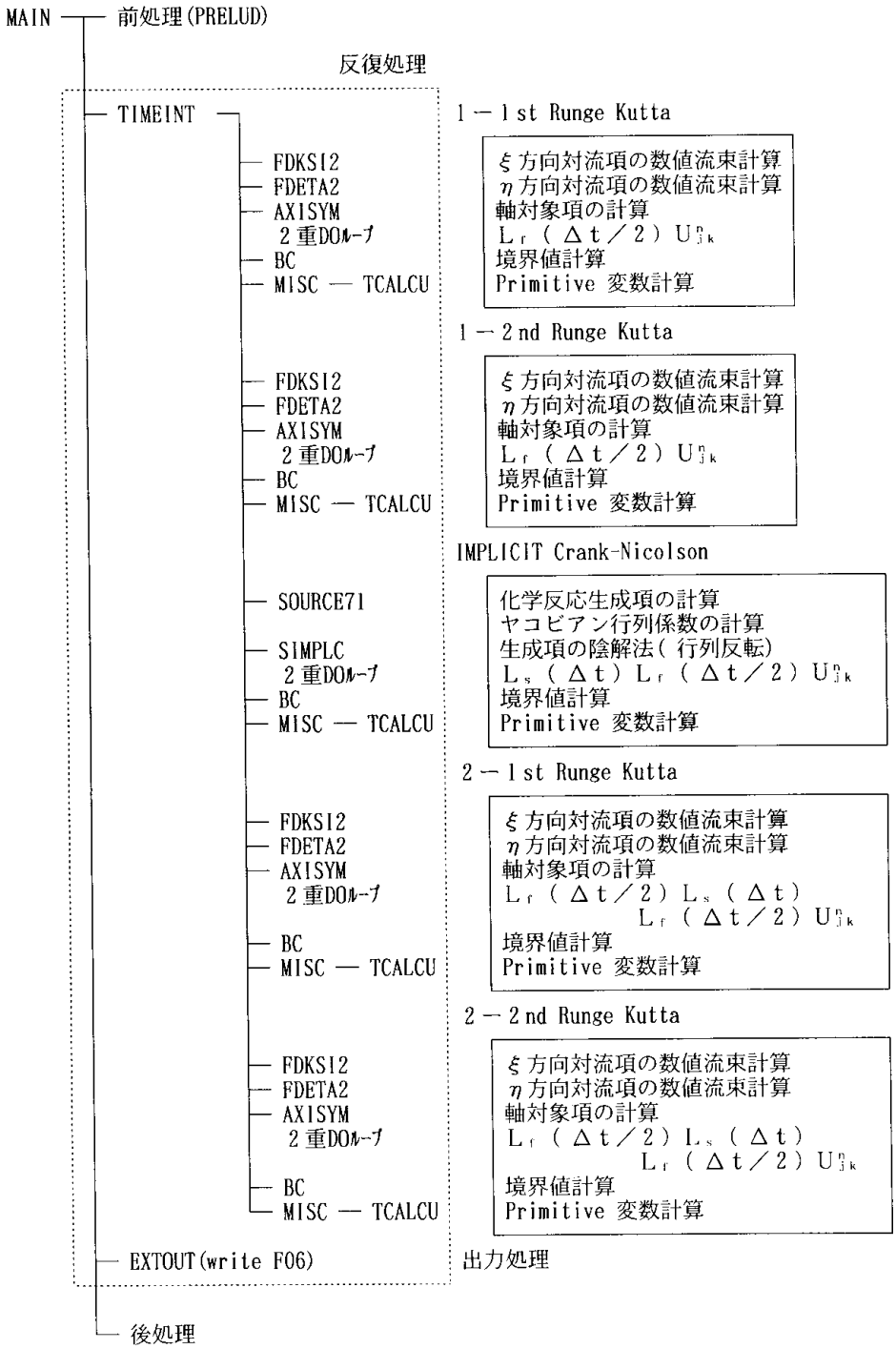


図2.5　プログラムIIのプログラム構造

ディスクへの出力処理である。煩雑のため、プログラム構造図への記載は省略する。

本プログラムの実行条件は格子点数801×51、反復回数1000回である。

第3章 推進系プログラムの並列処理手法及び並列処理性能

3.1 プログラムIの並列処理

(1) プログラムの構成

本プログラムは物理現象をシミュレートする二次元空間をメッシュで分割し、各標本点における物理量を配列データとして扱う。そして、物理現象を記述する差分方程式に基づいて、直前の状態の物理量から次の時刻の状態の物理量を次々と計算する構成になっている。

```
 1:      PROGRAM MAIN
 2:        ファイルのオープン
 3:        初期データ読み込み
 4:      CALL PRELUD
 5:      CALL BC
 6:        その他の初期化
 7:C
 8:    1 CONTINUE
 9:        TISTEP = TISTEP + 1
10:C
11:        最小値の計算
12:        CALL EDYVIS
13:        CALL FDKSI
14:        CALL FDETA
15:        CALL FVKSI
16:        CALL FVETA
17:        CALL IAFKSI
18:        CALL IAFETA
19:        CALL EVOLUT
20:        最小値・総和の計算
21:        CALL CHECK
22:C
23:        途中経過出力
24:        CALL EXTOUT
25:C
26:        CALL BC
27:      IF ( TISTEP.LT.TIEND ) GO TO 1
28:C
29:      最終結果出力
30:C
31:      END
```

図3.1 プログラムIのメインルーチンの構成

メインルーチンの構成の概略を図3.1に示す。メインルーチンは初期設定部、タイムマーチングを行なうメインループ（行番号1～GOTO1）及び最終結果出力部の3部構成となっている。

メインルーチンから呼び出されるサブルーチンのうち、PRELUDは種々の配列の値を初期化するルーチンであり、プログラム全体で一度しか呼び出されない。またEXTOUTは計算の途中経過をファイルに出力するルーチンであり、メインループ中に存在するが、実際にはループの何度かに一度ずつしか呼び出されない。本プログラムのパラメータ設定では全体で最後に一度だけ呼び出される。

各サブルーチンはほとんどDOループで構成されており、後述する「並列化の方針」に従ってDOループ単位で並列化している。サブルーチンFDKSI, FDETA, FVKSI, FVETA, IAFKSI, IAFETAは、配列の前回の値から新しい値を計算するルーチンであり、FDKSI及びFVKSIが配列の行方向の近傍による影響の計算、FDETA及びFVETAが配列の列方向の近傍による影響の計算を行なっている。このうち、FDKSI及びFVKSIは後述する「データ転送コードの挿入」の処理が必要なループを多数含んでいる。また、IAFKSIは後述する「並列化できないループ」をいくつか含んでいる。

メインルーチンのメインループの中は大部分が子ルーチンの呼び出しであるが、その他に最小値や総和を求める計算が含まれている。これらには後述の「特殊な計算の並列化」に述べる方法を適用した。

(2) 並列化の方針

プログラムは配列を多量に用いるため扱うデータの量は多いが、標本点である配列要素ごとに異なる計算を行うわけではなく、同じ計算をほとんどの配列要素に対して適用するものである。例外は境界の処理、すなわち配列の端の処理である。物理空間はサイズMK×MEのメッシュで標本化されているので、プログラムの主要部分は図3.2に示すDOループの形式である。

標本点の近傍の状態が次の時刻の状態に影響を及ぼす場合には図3.3の形式のDOループ計算になる。この例では標本点の前後各1近傍が次の時刻の状態に影響を及ぼしている。

Cenju-3は共有メモリを持っていないので図3.2および図3.3に示す形式のDOループを並列化するためには、次のような処理を行う。

- 配列を等分割し、複数の要素プロセッサ（PE）に均等に割り当てる。
- 各PEは、自分に割り当てられた配列部分に対する

```
1:    DO 10 J=1,MK
2:      DO 10 K=1,ME
3:        YYY(J,K) = f(XXX(J,K))          f ：関数
4:  10 CONTINUE
```

図3.2 簡単なDOループ

```
1:    DO 10 J=2,MK-1                      f , g ：関数
2:      DO 10 K=2,ME-1
3:        YYY(J,K) = f(XXX(J,K))
4:    &            + g(XXX(J,K+1)) + g(XXX(J,K-1))
5:  10 CONTINUE
```

図3.3 前後への配列参照を含むDOループ

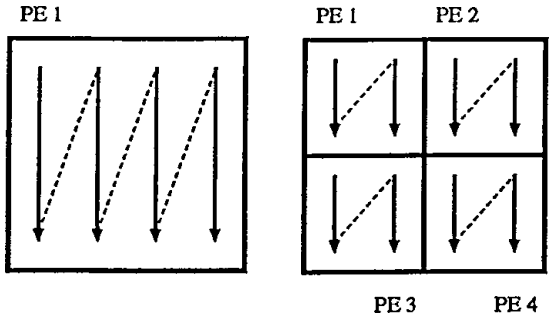
計算のみを行なう。

このような方法が一般的であり、この並列化をデータ並列に基づく並列化と呼ぶ。

図3.4にデータ並列に基づく並列化の概念図を示す。(a)の矩形は、オリジナルプログラムにおける一つの配列を示す。矢印は、PEがループによって配列を走査しながら配列の各要素の計算を行う様子を表す。(b)は(a)の配列を縦及び横に2等分ずつ、計4等分して4台のPEに割り当てる様子を示したものである。各々のPEは自分の持っている領域についてのみ計算を行なえばよいことになり、4台のPEは同時に動作することができるので、この計算にかかる時間は都合1/4になる。

配列の分割の仕方には様々な方法が考えられる。他の分割方法を図3.5に示す。本プログラムの並列化においては図3.5 (b)に示す行方向で等分割する方法をとる。

また、分割した配列の各PE上でのメモリへの割り当て方法についても2つの方法が考えられる。すなわち、割り当てられた配列部分を格納するのに必要最小限の大きさのメモリ領域をPEごとに確保する方法と、もとの配列全体が格納できるだけのメモリ領域を全PE上に確保しておき、そのうち各々に割り当てられた配列部分に相当する領域だけを実際に使用する方法である。これらの方法の相違を図3.6に示す。



(a) オリジナルプログラムの処理　(b) 並列化プログラムの処理

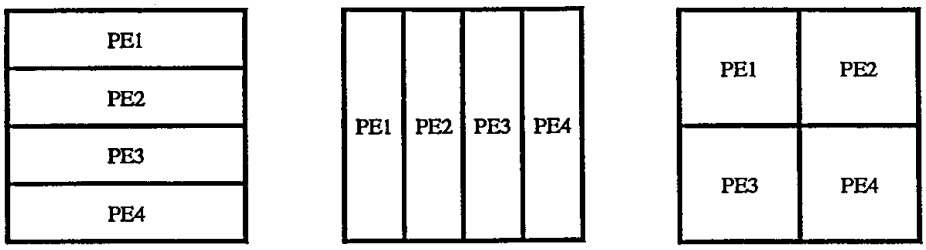
図3.4 データ並列に基づく並列化

前者は必要なメモリ領域が最小限で済むためメモリの使用効率が良く、より大きなサイズのシミュレーションが可能となる反面、配列を参照する際、もとのプログラムと並列化した後のプログラムで配列添字（インデックス）の値が変わってしまうため、並列化にあたってインデックスのつけ直しが必要となり、プログラムの書き換えの量が膨大になる。一方後者はメモリの使用効率は悪いが、プログラムの書き換え量が少なく、並列化作業が容易である。本プログラムでは後者の方法をとっている。

(3) ループの書き替え

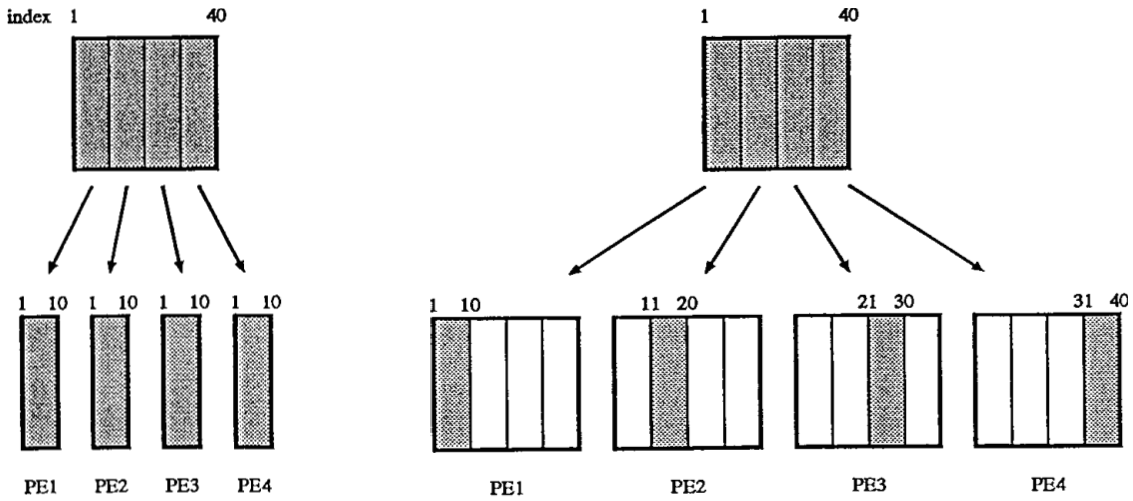
前述の並列化の方針に従って、実際にプログラムの並列化を行なう場合、プログラム中で配列を操作するループ、すなわち、図3.2及び図3.3に示す形式のDOループについて、各PEが自分の担当部分だけを実行するようにプログラムを書き換えなければならない。このような書き替え作業は本来並列化ツールによって自動的に行なわれるべきものであるが、現状ではまだそのようなツールが整備されていないため、手作業にてプログラムの書き換えを行なった。

例として、図3.2に示す配列の行方向で並列化する場合の書き替え方法を図3.7に示す。図中、下線表示したNUMPEは利用可能なPEの総数、MYPIDはそのうち自分が何番目のPEであるかを示すID番号（0, 1, …,



(a) 1次元目のみで分割　(b) 2次元目のみで分割　(c) 両方で分割

図3.5 データ並列に基づく並列化による分析方法



(a) 必要最小限の領域を確保する場合　　(b) 全領域を確保し必要な部分だけ使用する場合

図3.6 プロセッサにおけるメモリ分割方法

NUMPE－1) である。これらはプログラムの起動時にシステムから取得しておく。

配列の行方向に相当するのは2重ループのうちの内側ループである。元のプログラムでは配列XXXの行方向の大きさがMEであり、内側ループはそれに応じてME回まわるが、並列化プログラムではそれをNUMPE台のPEに均等に割り当てるため、1台あたりのサイズKBLOCKをまず求める (1行目)。KBLOCKを単純にME/NUMPEとしないのは、割り切れない場合の端数 (余り) の部分が捨てられるのを防ぐためである。次に、ループの開始インデックスKSTARTをKBLOCKとMYPIDより求め (2行目)、終了インデックスKENDをKSTARTより求める (3行目)。端数の関係でKENDがMEを越えてしまう場合のためにIF文で調整を行なう (4行目)。最後にDO文の制御範囲を書き換えて、自PEに割り当てられた部分だけが実行されるようにする (7行目)。

書換えにより、図中1行目～4行目までが加えられ、7行目のステーメントの変更が行われた。並列化対象ループについて、このような書き換えをプログラム全体にわたって行なった。

```
1:      KBLOCK = (ME + NUMPE - 1) / NUMPE
2:      KSTART = KBLOCK * MYPID + 1
3:      KEND = KSTART + KBLOCK - 1
4:      IF (KEND.GT.ME) KEND = ME
5: C
6:      DO 10 J=1,MK
7:        DO 10 K=KSTART,KEND
8:          YYY(J,K) = f(XXX(J,K))
9:   10 CONTINUE
```

図3.7 図3.2のDOループの並列化

(4) データ転送コードの挿入

サブルーチンFDKSI及びFVKSI中に多く現れる図3.3の形式のDOループを並列化するためには、図3.2に示す形式のDOループの並列化のように制御範囲を書き換えるだけでは適切に処理されない。これはDOループの計算処理に配列要素XXX (J, K－1)及びXXX (J, K＋1)の参照があり、PEの担当範囲の両端、すなわち、K=KSTART及びK=KENDではこれらの要素が自PE内に存在しないからである。この形式のDOループの並列処理のためには、ループの直前にデータ転送コードを挿入し、ループの実行に必要なデータを予め揃えておかなければならない。図3.8にこの概念図を示す。

図中、鎖線で示す柱は配列とそれを操作するループのインデックス範囲を表し、インデックスKは下へいくほど大きくなるものとする。太枠で囲んだ部分が自PEの担当範囲である。XXX (J, K＋1)の参照は、太枠の外側の網かけの部分が参照されることになるが、この部分は自PE内には存在せず、隣接PEが所有している。逆に、自分は隣接PEが必要とするデータ (斜線部分) を所有している。そこで、お互いに相手が必要としているデータを送りあうことで、必要なデータを揃える。

図3.3に示す形式のDOループの並列化プログラム例を、図3.9に示す。図中、1行目から6行目は各PEの計算範囲を求めるためのステートメントである。また、ループ開始インデックスが1ではなく2となっているので、KSTARTに関するチェックが余分に入っている (3行目)。

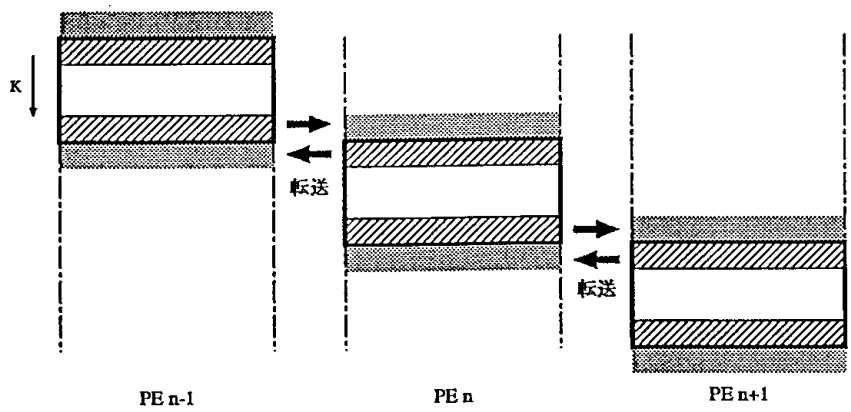


図3.8　プロセッサの処理に必要な配列の範囲

7行目から16行目までがデータ転送コードである。サブルーチンCJRMWRITEFは、Cenju-3の標準並列ライブラリであるPARALIB/CJに含まれるデータ転送プリミティブである。呼び出し形式は、

CALL CJRMWRITEF (FROM, LID, TO, NBYTES, ISTAT)

であり、データFROMをPE番号LIDであるPEのTOという変数の領域に転送（複写）する。NBYTESは転送するバイト数であり、単精度実数の場合にはデータ数の4倍、倍精度実数の場合にはデータ数の8倍である。ISTATには実際に転送に成功したバイト数が返る。

まず自分よりも若いインデックスの領域を担当するPEが存在するかどうかを調べ（7行目）、もしそのようなPEが存在すれば、自分の隣のPEに必要なデータを転送する（8～10行目）。同様に、自分よりも大きなインデックスの領域を担当するPEが存在するかどうかを調べ（12行目）、もしそのようなPEが存在すれば、自分の隣のPEに

```
 1:      KBLOCK = (ME + NUMPE - 1) / NUMPE
 2:      KSTART = KBLOCK * MYPID + 1
 3:      IF (KSTART.LT.2) KSTART = 2
 4:      KEND = KSTART + KBLOCK - 1
 5:      IF (KEND.GT.ME-1) KEND = ME-1
 6: C
 7:      IF (KSTART.GT.2) THEN
 8:        DO 5 J=1,MK
 9:      CALL CJRMWRITEF(XXX(J,KSTART),MYPID-1,XXX(J,KSTART),4,ISTAT)
10:    5   CONTINUE
11:      END IF
12:      IF (KEND.LT.ME-1) THEN
13:        DO 6 J=1,MK
14:       CALL CJRMWRITEF(XXX(J,KEND),MYPID+1,XXX(J,KEND),4,ISTAT)
15:    6   CONTINUE
16:      END IF
17:       CALL CJBARRIER(0)
18: C
19:      DO 10 J=2,MK-1
20:        DO 10 K=KSTART,KEND
21:          YYY(J,K) = f(XXX(J,K))
22:     &             + g(XXX(J,K+1)) + g(XXX(J,K-1))
23:   10 CONTINUE
```

図3.9　図3.3のDOループの並列化

必要なデータを転送する（13～15行目）。このように、隣のPEが必要としているであろうデータをお互いに転送しあうことで、必要なデータが揃うように協調動作する。

17行目はバリア同期を張るためのPARALIB/CJプリミティブの呼び出しである。全PEが転送処理を終了してこの地点に到達するまで、先に到達したPEはここで待たされる。ここでバリア同期を張らないと、必要なデータがまだ送られてきていないのにDOループの実行を始めてしまうという事態が発生する可能性があり、その場合には、プログラムの正当な実行が保証されなくなる。

なお、Fortranでは配列の列方向が主記憶上で連続したアドレスとなるよう記憶領域が割りつけられることを利用して、データ転送を一つ一つの要素ごとではなくまとめて行なうことができる場合がある。例えば、図3.9の8行目から10行目までは、以下のような1行のプログラムで置き換えることができる。

```
CALL CJRMWRITEF (XXX(J, KSTART), MYPID－1,
                 XXX (J, KSTART), 4＊MK, ISTAT)
```

データ転送処理の起動には一定のオーバーヘッドが存在するため、このような置き換えを行なってデータ転送を一括して行なうようにした方が転送効率が良い。本プログラムの並列化に対して、このような最適化をできる限り行なった。

(5) 特殊な計算の並列化

配列要素全体の総和の計算や、最小値・最大値の計算などは、今まで述べた方法だけでは効果的に並列化することができない。これらの計算が前述の方法で計算されるのは、それぞれのPEの担当する範囲内における部分和あるいは、部分最小値・部分最大値にすぎないからである。

このような特殊な計算が行なわれているループは、上述のような方法で並列化した後、部分和あるいは部分最小値・部分最大値を一箇所に集め、さらにそれらから最終的な総和や最小値・最大値を求めるという後処理が必要となる。図3.10に並列処理を行って最小値・最大値を求める方法の概念図を示す。プログラムの書き換え例については、繁雑であるため省略する。

(6) 並列化できないループ

サブルーチンIAFKSI中の3つのループは今までに述べた方法によっては並列化できない。PE間にまたがる漸化式の形になっているからである。図3.11に漸化式のDOループ例を示す。図中のDOループにおいて、XXX (J, K)の値を計算するのにXXX (J, K－1)の値を必要とするが、自PEの担当範囲の計算を始めるためには自PEよりもIDの小さいPEの担当範囲の計算がすべて終わっていなければならない。

このように漸化式計算のプログラムを処理するためには、1台目のPEの処理終了後、対象計算範囲の端の計算結果を2台目のPEに送り、2台目のPEの処理が終了したら、同様に端の計算結果を3台目のPEに送り、…という形で「バケツリレー」式にデータと制御を受け渡すようにループを書き替える。この様子を図3.12に示す。このようにして、並列実行できないループは分散データを分散させたままの状態で実行させるよう変換する。プロ

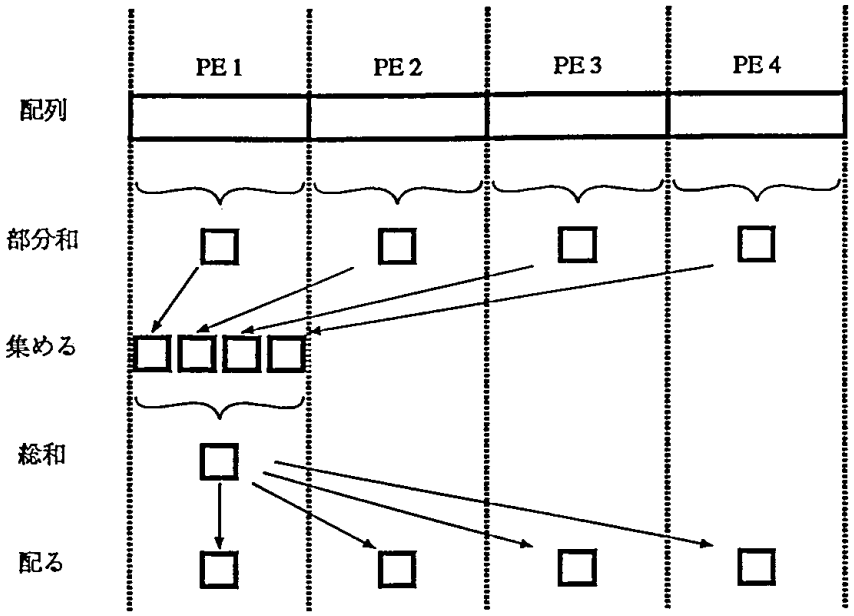


図3.10　総和・最小値・最大値の並列処理概念図

```
 1:      SUBROUTINE SUB1
 2:           :
 3:           :
 4:      DO 10 J=1,MK
 5:        DO 10 K=2,ME
 6:           XXX(J,K)=f(XXX(J,K-1))
 7:   10 CONTINUE
 8:           :
 9:           :
10:      DO 20 K=1,ME
11:        DO 20 J=2,MK
12:           XXX(J,K)=f(XXX(J-1,K))
13:   20 CONTINUE
14:           :
15:           :
16:      RETURN
17       END
```

図3.11　漸化式の形のDOループ

グラムの書き換え例は繁雑であるため省略する。

(7) その他の工夫

その他、並列化および性能向上のために次のような工夫を行なった。

・配列データの転置
・担当部分外の一部配列領域の計算
・ループの入れ換え
・配列サイズの変更

それぞれについて説明する。

① 配列データの転置

データを分割する際、一般にはサイズの大きな次元を分割する方が有利である。サイズの小さな次元で分割すると、PE間で分担するデータの量にアンバランスを生じやすく、またデータ転送は分割境界の両側で発生するので分割の境界線が短い方がデータ転送の総量を小さくできるためである。

一方、(4)「データ転送コードの挿入」の項で述べたように、データ転送はなるべく一括して行なう方が効率が良く、このためにはデータを行方向で分割する方が良い。

プログラムⅠでは主要データの二次元配列のサイズが601×101となっており、行方向の方がサイズが小さいため、このままでは上述の2つの条件を同時に満たすようにデータを分割することができない。

そこで、プログラム中に現れるすべての配列を「転置」し、101×601の配列として扱った。これは、プログラム中の配列参照および宣言に関して、添字の列と行をすべて取り替えることで実現できる。

② 担当部分外の一部配列領域の計算

サブルーチンFDKSI, FVKSI, IAFKSIでは、最終的に必要な配列要素の値をもとになるデータから数段階に分けて計算している。この様子を図3.13に示す。

各段階で作業配列に中間結果を生成するが、各々の段階で前段の中間結果配列の添字（J, K）と（J, K－1）または（J, K＋1）の値しか使用しないにもかかわらず、総体としてみるとRESULT (J, K)を計算するためにDATA (J, K－3)～DATA (J, K＋2)の値が必要になっている。

図3.13の矢印は前段の中間結果のデータが次の段でどのように使われるかを示している。例えばTMP1 (J, K)を計算するにはDATA (J, K)とDATA (J, K＋1)の値が使われ、TMP2 (J, K)を計算するにはTMP1 (J, K)とTMP1 (J, K－1)の値が使われていることを示す。このようにデータの流れを考えると、最終的に必要な値RESULT (J, K)を計算するためにDATA (J, K－3)～DATA (J, K＋2)の値が必要になっていることがわかる。

前述の方法による並列化では各段階毎にデータ転送が必要となるが、もとになるデータを格納する配列DATAをあらかじめ分割境界の前後±3の範囲で転送しておき、中間結果TMP1, TMP2, TMP3, TMP4はそれを使って計

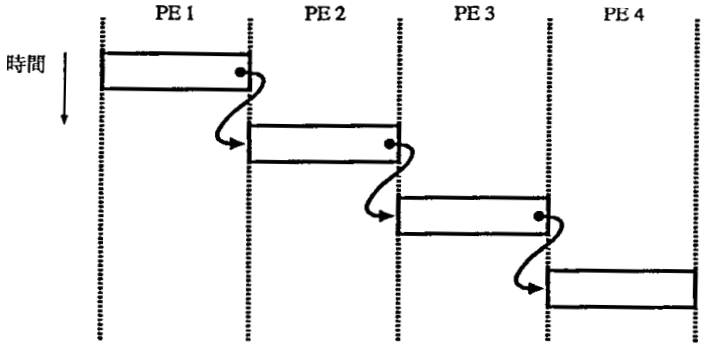


図3.12　漸化式計算の処理

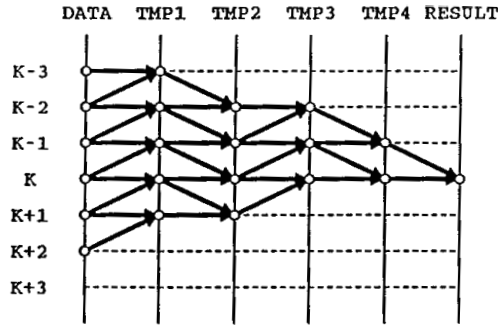


図3.13　使用する配列の範囲

算できる範囲一杯まで本来の受け持ち範囲よりも広く計算しておくことにより、データ転送を配列DATAだけに一括してまとめて行なうことができ、効率を上げることができる。

③ ループの入れ換え

Fortranプログラムの配列は列方向にメモリ割り付けにおいて連続したアドレスが割り当てられるため、多重ループにおいては列方向の添字を制御変数とするループを最内側ループとすれば、配列に対するアクセスが連続したアドレスに対して行なわれることになり、キャッシュメモリによる性能向上の効果を期待できる。

そこで、プログラム中の多重ループについて、列方向の添字に対応するループを最内側となるよう入れ換えてもプログラムの実行上結果に影響を及ぼさないループについては、ループの入れ換えを行なった。

④ 配列サイズの変更

プログラムⅠではいくつかの配列変数が実際には使用されない不要な部分を含めた少し大きめのサイズで宣言されている。これは、プログラムⅠをベクトル型スーパーコンピュータで実行する際、メモリバンク競合を回避するための措置と考えられるが、分散メモリ型並列計算機であるCenju-3においては無用であり、かえって配列のアドレスの連続性を損なってデータ転送の効率化等に支障を来たす場合があるため、このような不要なサイズの宣言は実際に必要なサイズのみの宣言に改めた。

(8) 評価結果

今までに述べた並列化を適用して得られたプログラムⅠの並列処理用プログラムを、Cenju-3モデル16にてPE数を1, 2, 4, 8, 16と変化させ、実行時間を測定した。測定条件は以下の通りである。

- コンパイラオプション「-0 -Zstatic」による最適化を行なった。このうち「-0」は標準的な最適化を指定するオプションであり、「-Zstatic」は初期化されない変数の初期値を0とするオプションである。
- 時間ステップ数100（オリジナルは2000）とした。これはオリジナルのステップ数のままでは時間がかかり過ぎ測定が困難なためである。実行時間を縮小することによる各サブルーチン実行時間の比率に支障はきたさない。

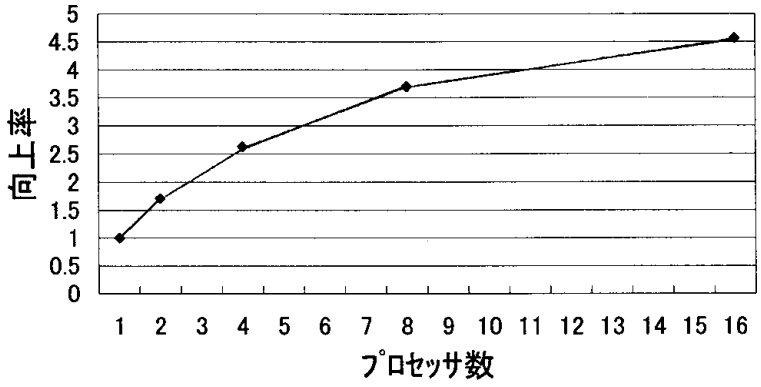


図3.14（a） プログラムⅠの速度向上率

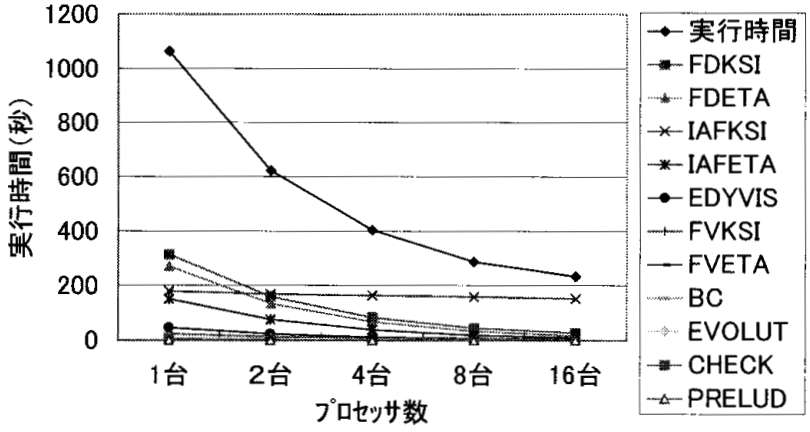


図3.14（b） プログラムⅠの実行時間

・物理空間を規定するサイズパラメータは、MK=601，ME=101とした。
・プログラム中のWRITE文は全てコメントにして非実行文とした。これは、Cenju-3の入出力性能がフロントエンドのEWS4800の負荷状態により大きく変化するため、WRITE文を含んだままでは並列化の効果を確認できないためである。

図3.14（a）にプログラムIの速度向上率を、(b)にPE毎の実行時間を示す。これらの元のデータを表3.1に示す。図3.14（a）からは速度向上率の向上率が徐々に減少していく様子がわかる。全体ではPE数16台で約4.6倍の加速が得られた。PE数16台に対し、速度向上率が4.6倍という値は、PE台数増大による効果があるとは言いがたい。言い換えれば、30%程度のPEの能力しか使用していないことを示す。

図3.14（b）を見ると、サブルーチンIAFKSIを除くサブルーチンはPE数の増加に比較的よく反比例して実行時間が短くなっている。しかし、IAFKSIの実行時間の短縮はあまりはかられていない。このままではPE数を更に増やして行ってもIAFKSIの実行時間はせいぜい140～150秒前後までしか減らせないと見られるので、それ以外の部分の実行時間がどれだけ小さくなっても全体では6～7倍前後の加速しか得られないと考えられる。

IAFKSIの速度が向上しないのは、IAFKSIが前節「並列化できないループ」の項で述べたPE間にまたがる漸化式の形になるループを含んでおり、この部分を並列実行

表3.1 プログラムIの実行時間（単位：秒）

プロセッサ数	1台	2台	4台	8台	16台
プログラム実行時間	1063	622	404	287	233
速度向上率	1.00	1.71	2.63	3.70	4.56
FDKSI	314	159	83	45	26
FDETA	270	134	67	33	16
IAFKSI	179	169	163	159	152
IAFETA	151	75	38	18	9
EDYVIS	46	23	11	6	3
FVETA	23	11	5	3	1
FVKSI	23	13	7	4	3
BC	22	11	6	4	2
EVOLUT	8	4	2	1	1
CHECK	2	1	0	0	0
PRELUD	1	0	0	0	0

できないためである。データの分割方向を変えて列方向での分割とすればIAFKSIの漸化式は単独PE内で処理できるようになるが、代わってIAFETAの漸化式がPE間にまたがるようになる。データを1つの次元で分割する限り、漸化式がPE間にまたがる方向は空間の次元数を大きくしても常に1つだけであるから、すなわち分割面に直交する方向1つだけであるので、より高次元の問題では漸化式の影響は相対的に小さくなると考えられる。しかし、分散メモリ型の並列計算ではこの問題を完全に避けることはできない。

行列の各行に沿った漸化式を処理するにあたり、漸化式の処理方式は、PE1が自分の担当する全行の計算を行ない、その結果をPE2に渡し、PE2は受けとった結果から自分の担当する全行の計算を行ない、その結果をPE3に渡し、…という方式である。すなわち、同時に動作しているPEは常に1つだけである。

しかし、結果の受け渡しを行ごとに行なうようにすることも考えられる。すなわち、PE1が1行分の計算を終えたらその結果をPE2に渡し、自分は次の行の計算を行なう。同時にPE2は受けとった結果により1行分の計算を始める。…という具合である。この方法によれば、漸化式であってもパイプライン的に並列化が可能であるため、ある程度の性能向上が見込まれる。ただし、パイプライン処理のためのデータ転送では要素の転送ごとにPE間で同期をとる必要がある。このパイプライン型並列制御は今後の課題としたい。

3.2 プログラムIIの並列処理

(1) プログラムの構成

プログラムIIのメインルーチンの概略を図3.15に示す。

```
      PROGRAM MAIN
        ファイルのオープン
        CALL PRELUD
        その他の初期化
C
    1   CONTINUE
          TISTEP = TISTEP + 1
          CALL TIMEINT
          途中経過出力
          CALL EXTOUT
        IF (TISTEP.LT.TIEND) GOTO 1
C
        最終結果出力
C
      END
```

図3.15 プログラムIIのメインルーチンの構成

プログラムの構成はプログラムIとほぼ同様である。

メインルーチンから直接呼び出されるサブルーチンは3個であり、それはサブルーチンPRELUD, TIMEINT, EXTOUTである。このうちPRELUDは種々の配列の値を初期化するルーチンであり、全体で一度しか呼び出されないので、サブルーチンPRELUDから呼び出される下位ルーチン、CHARAC, EQCOMP7, EXXP, LOWTEMP, QUARTIC, QUDRATC, GRIDRD, SETVISCも含め、並列化の対象としなかった。

TIMEINTとEXTOUTはメインループを構成する。繰り返し1回につき時間経過1ステップ分の計算を行なう。

```
      SUBROUTINE TIMEINT
        前処理
C
        CALL FDKSI2
        CALL FDETA2
        計算
        CALL BC
        CALL MISC
C
        CALL FDKSI2
        CALL FDETA2
        計算
        CALL BC
        CALL MISC
C
        CALL SOURCE71
        CALL SIMPLEC
        計算
        CALL BC
        CALL MISC
C
        CALL FDKSI2
        CALL FDETA2
        計算
        CALL BC
        CALL MISC
C
        CALL FDKSI2
        CALL FDETA2
        計算
        CALL BC
        CALL MISC
C
        RETURN
      END
```

図3.16 プログラムIIのTIMEINTルーチンの構成

ただしEXTOUTは中間結果を出力するルーチンであり、ループの繰り返しごとに毎回呼び出されるとは限らない。実行条件のパラメータ設定では全体で最後に一度呼び出されるだけである。

プログラムの主要部分はサブルーチンTIMEINT内に集約されている。サブルーチンTIMEINTの構成の概略を図3.16に示す。

(2) 並列化の方針

並列化の方針は基本的にはプログラムⅠと同様である。ただし配列の分割次元はプログラムⅠでは行方向としたが、プログラムⅡでは列方向とした。

TIMEINTに計5箇所ある「計算」の部分は、プログラムⅠの「並列化の方針」に従い並列化を行なった。「前処理」の部分にある最小値を求める計算に関しては、プログラムⅠの「特殊な計算の並列化」で述べた方法で並列化する。

TIMEINT以下のほとんどのサブルーチンはループで構成されており、これらは前述の「並列化の方針」に従い「ループの書き換え」を行なって並列化している。サブルーチンFDKSI2中には「データ転送コードの挿入」が必要なループがいくつか含まれる。なお、プログラムⅡ中には「並列化できないループ」に相当する漸化式の形の計算は存在しない。

プログラムⅠの「その他の工夫」で述べた工夫のうち、「担当部分外の一部配列領域の計算」についてはFDKSI2について行なっている。また「ループの入れ換え」についても、プログラム全体に渡って適宜行なっている。しかし、プログラムⅡでは配列の分割次元を列方向としたため配列の転置は行なっていない。配列サイズの変更についても、対象となる配列も存在しなかったため行なっていない。

その他、プログラムⅡのみで行なった工夫として、「配列形状の変更」がある。

・配列形状の変更

他のサブルーチンで2次元（MK行ME列）配列として使用されているＣＯＭＭＯＮ配列を、サブルーチンSOURCE71とSIMPLECでは1次元（MK×ME要素）配列として単一ループでアクセスしている。

配列を分割しない場合は連続アクセスとなるので問題とならないが、並列化のために列方向を分割した場合、各PEに割り当てられる領域はアドレスが連続しないので、単一ループでアクセスすることができない。これらのルーチン内でも2次元配列としてアクセスするよう配列の形状を1次元から2次元に変更した。

(3) 評価結果

前節に述べた並列化を適用して得られたプログラムⅡの並列処理プログラムを、Cenju-3モデル16にてPE数を1, 2, 4, 8, 16と変化させ、実行時間を測定した。測定条件は以下の通りである。

- コンパイラオプション「-0 -Zstatic」による最適化を行なった。
- 時間ステップ数1（オリジナルは1000）とした。本プログラムは演算例外多発により実行時間が通常より大きいため、オリジナルプログラムのステップ数のままでは時間がかかりすぎ、測定が困難なためである。ステップ数1の計測に変更しても、そのプログラムのCPU負荷を把握していれば、実行時間の推測において支障はきたさないと考えられる。
- サイズパラメータはMK=801，ME=51とした。
- プログラム中のWRITE文はすべてコメントにし非実行文とした。また、WRITE文を省略したので実質的に意味のない、結果の出力時に全データを1台のPEに集めるためのデータ転送処理を省略した。

測定結果を表3.2に示す。表には時間ステップ1の実行時間及び速度向上率を示す。表中、MAIN及びTIMEINTは子ルーチン呼び出しを除いた実行時間を示す。全体ではPE数16台で8.4倍の加速が得られた。また、図3.17は表3.2を図示したものである。

表3.2においてサブルーチンWINGFIXは3.1節(4)「データ転送の挿入」の項で述べたデータ転送処理をサブルーチンとして独立させたものである。PE1台ではデータ転送が発生しないので時間は0となる。PEが2台のときは転送相手となるPEが1台のみであるが、PEが4台以上のときは両隣の2台のPEにデータを転送するため2台のときのほぼ2倍の時間がかかっている。台数がそれ以上に増えても転送にかかる時間は理論上同じになるはずであるが、実際にはデータ転送時に発生する通信の競合などの発生頻度が高くなるため、台数の増加にともない若干性能が落ちる。

全体的な性能においては、PEが2台以上ではPE数の増加にともない、実行時間は減少し、2台以降においてはほぼ半減し、実行時間はPE数に反比例している。しかし、PE1台から2台にかけてはあまり並列化の効果が現れていない。これは、配列の前半と後半で性質の異なるデータが初期データとして与えられており、一方の半分の処理にかかる時間が他方の半分の処理にかかる時間よりも極端に長くなっているためであると考えられる。実際、PEごとの詳細な時間測定では、配列の前半部分を担当したPEと比べて後半部分を担当したPEの方が極端に多く

表3.2 プログラムIIの実行時間（単位：ミリ秒）

プロセッサ数	1台	2台	4台	8台	16台
プログラム実行時間	160418	140590	71293	36458	19111
速度向上率	1.000	1.141	2.250	4.400	8.394
MAIN*	115	153	202	234	270
PRELUD	560	559	558	559	559
TIMEINT *	5766	3766	1884	951	504
SIMPLC	82859	81605	41006	20607	10399
MISC	26575	21674	10899	5521	2859
SOURCE71	20110	16979	8531	4287	2164
FDETA2	12748	8124	4078	2041	1030
FDKSI2	11670	7571	3835	1948	1004
BC	15	10	7	6	6
EXTOUT	0	0	0	0	0
WINGFIX	0	149	293	304	316

＊　子ルーチン（呼び出しルーチン）の実行時間を除いた実行時間を表示している。

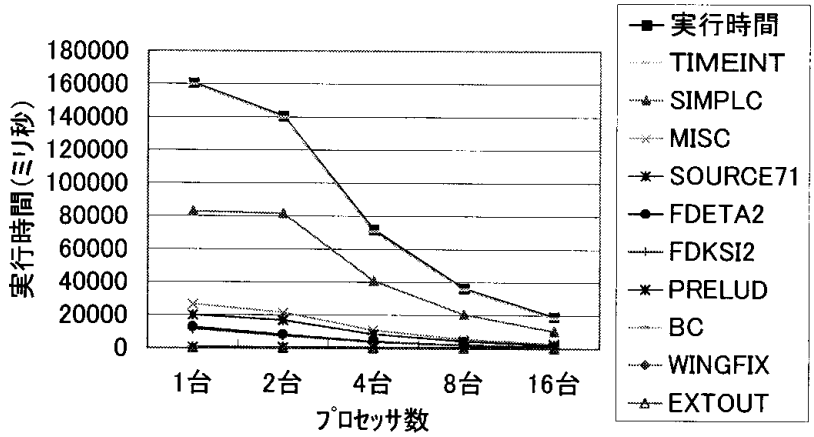


図3.17 プログラムIIの実行時間（1回）

表3.3 プログラムIIの推定実行時間（単位：ミリ秒）

プロセッサ数	1台	2台	4台	8台	16台
プログラム推定実行時間	42200	22560	12120	6880	4420
速度向上率	1.000	1.870	3.481	6.133	9.547
MAIN*	115	151	198	231	267
PRELUD	560	558	559	559	559
TIMEINT *	5770	3704	2099	1225	792
SIMPLC	2500	1254	630	316	158
MISC	9700	4870	2409	1226	691
SOURCE71	6200	3132	1567	787	398
FDETA2	9140	4616	2296	1154	582
FDKSI2	8200	4114	2040	1033	527
BC	15	12	9	8	8
EXTOUT	0	0	0	0	0
WINGFIX	0	150	307	333	340

＊ 子ルーチン（呼び出しルーチン）の実行時間を除いた実行時間を表示している。

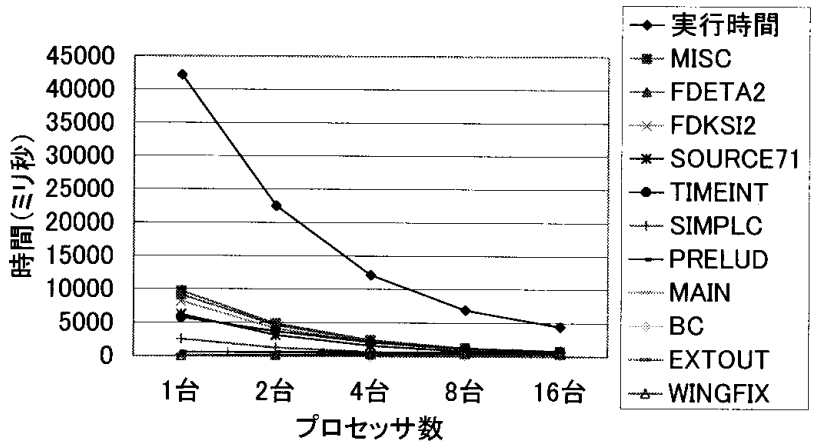


図3.18 プログラムIIの推定実行時間（1回）

表3.4 プログラムIIの推定実行時間（単位：秒）（1000ステップの場合）

プロセッサ数	1台	2台	4台	8台	16台
プログラム推定実行時間	41526	21853	11358	6083	3497
速度向上率	1.000	1.900	3.656	6.827	11.87
MAIN*	0.115	0.151	0.198	0.231	0.267
PRELUD	0.560	0.558	0.559	0.559	0.559
TIMEINT *	5770	3704	2099	1225	792
SIMPLC	2500	1254	630	316	158
MISC	9700	4870	2409	1226	691
SOURCE71	6200	3132	1567	787	398
FDETA2	9140	4616	2296	1154	582
FDKSI2	8200	4114	2040	1033	527
BC	15	12	9	8	8
EXTOUT	0	0	0	0	0
WINGFIX	0	150	307	333	340

＊ 子ルーチン（呼び出しルーチン）の実行時間を除いた実行時間を表示している。

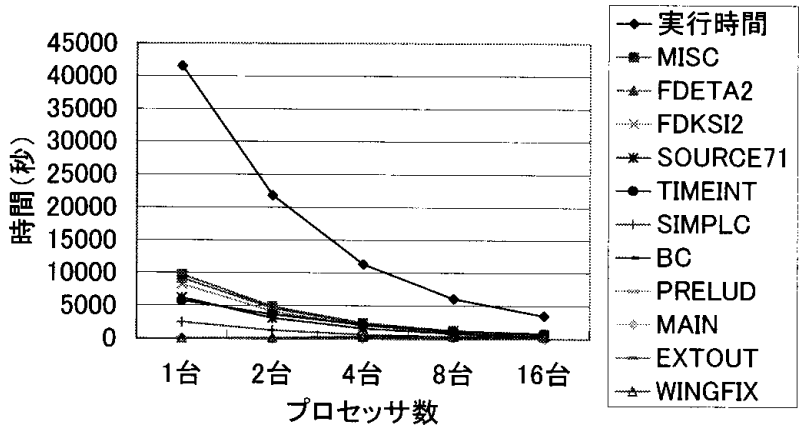


図3.19 プログラムIIの推定実行時間（1000回）

の時間を費やしている。後半部分を担当したPEでは、一種の演算例外が多発しており、このための処理に余分の時間がかかっているものと考えられる。このため、PE2台にデータを分割した場合には1台が処理時間のかからない部分、もう1台が処理時間のかかる部分を担当することになり、遅い方のPEの処理時間が全体の処理時間として見えてしまうが、4台以上の分割では処理時間のかかる部分についてもそれぞれ等分割されてPEに割り当てられるため、全体の処理時間が台数に応じて減少するように見えている。

測定においては時間ステップを1としているためこのような現象が発生しているが、ステップ数が増えれば時間の経過とともに配列中のデータの均一化が進み、演算例外の発生も少なくなるとみられる。このため、演算例外の発生がないと仮定した場合の処理時間を、実測したPEごとの処理時間から推定して求めた実行時間を表3.3に示し、また図3.18に図示する。

表3.3においてPE数2台以上のときの各値は、前述のプログラムでPE毎の所要時間を実測し、演算例外を起こさなかったPEのうち実行時間の最も大きいPEの実行時間である。PE1台のときの数値は、2台以上の各実行時間から推定される推定実行時間である。表より、オリジナルプログラムの1000分の1での実行時間で、PE数16台で9.5倍の加速となっていることがわかる。

この推定実行時間によれば、並列化しない場合には全体ではほぼ42秒かかっている。並列化した場合、約0.35秒のデータ転送（WINGFIX）の部分は常に必要であり、最終的に残る時間とすればプログラムⅡは並列化によって120倍程度までしか加速しない計算となるので、それ以上の要素計算機を並べても処理速度は向上しない。

一方、FDKSI2で行なっている「担当部分外の計算」は、本来担当する部分の±3の部分、合計6列分である。分割している次元のサイズは801であるから、PEが130台程度を越えると、本来担当である部分よりも担当外の部分の方が大きくなり、並列化の効果が期待できなくなる。

なお、表3.3に示す各サブルーチンの実行時間を基に、1000ステップ実行した場合の推定実行時間を表3.4に示し、図3.19に図示する。表3.4は表3.3のサブルーチンTIMEINT以下の時間を1000倍し、それにMAIN, PRELUDの実行時間を加えたものである。結果はPE数16台で約12倍の加速となることが推測できる。また表3.4及び図3.19から、PE数の増加にともない実行時間は減少し、並列処理の実行時間はPE数にほぼ反比例することが期待できる。

図3.20にプログラムⅡの時間ステップ1の実測実行時間の速度向上率、時間ステップ1の推定実行時間の速度向上率及び時間ステップ1000の推定実行時間の速度向上率をまとめた。

3.3 まとめ

プログラムⅠ及びプログラムⅡのCenju-3システムにおける並列処理において適用された並列処理手法を表3.5に示す。

これらの並列処理手法のうち、ループの書換え（1）は配列をどのように各PEに分割するかを示すものであり、並列処理を行うためには必ず必要となり、避けては通れない。このプログラム修正はCenjuシステムの自動並列化を考えれば、真先に対象となる。並列処理手法例について表示したプログラム例は本質的部分を抽出したものであり、簡単化されている。しかし、実際のプログラムではかなり複雑なものもあり、それらにPE分割に必要なステートメントを加えることは、更に煩雑なプログラムとなることが想像できよう。

また、配列データの転置（5）は並列計算機において効果的な手法であり、ループの入れ換え（7）はどの計算機においても効果のあるプログラミング手法である。

配列サイズの変更（8）及び配列形状の変更（9）は1PE処理又はベクトル計算機の場合には大いに効果のあるプ

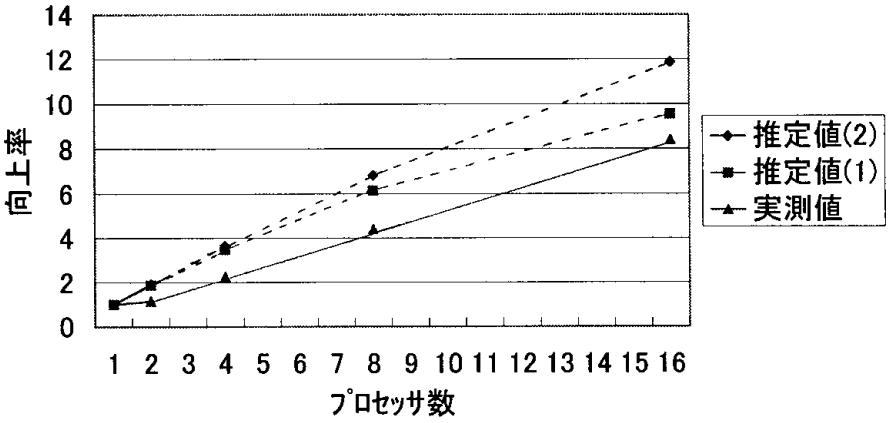


図3.20　プログラムⅡの速度向上率

表3.5 並列処理手法一覧表

NO	並列処理手法	プログラムⅠ	プログラムⅡ
1	ループの書換え	○	○
2	データ転送コードの挿入	○	○
3	特殊計算の並列化	○	○
4	漸化式計算	○	——
5	配列データの転置	○	——
6	担当部分外の一部配列領域の計算	○	○
7	ループの入れ換え	○	○
8	配列サイズの変更	○	——
9	配列形状の変更	——	○

ログラミングとなっている。超並列計算機システムでの並列処理においては、そのままでは高速化阻害となり修正を行なったものである。

その他のデータ転送コードの挿入 (2)、特殊計算の並列化 (3)、漸化式計算 (4) 及び担当部分外の一部配列領域の計算 (6) の処理手法については、対象とする問題のアルゴリズム等に起因する問題からでてくるプログラム表現であると考えられる。

図3.21にプログラムⅠ及びプログラムⅡのPE数と速度向上率を示す。PE数4を超えるとプログラムⅠの速度向上率はプログラムⅡのそれに比較して小さい。また、速度向上率の向上率もプログラムⅡがほぼ2倍近い勢いで伸びているのに対し、プログラムⅠの速度向上率のそれは低い。また、PE数16台ではプログラムⅠは4.6倍の

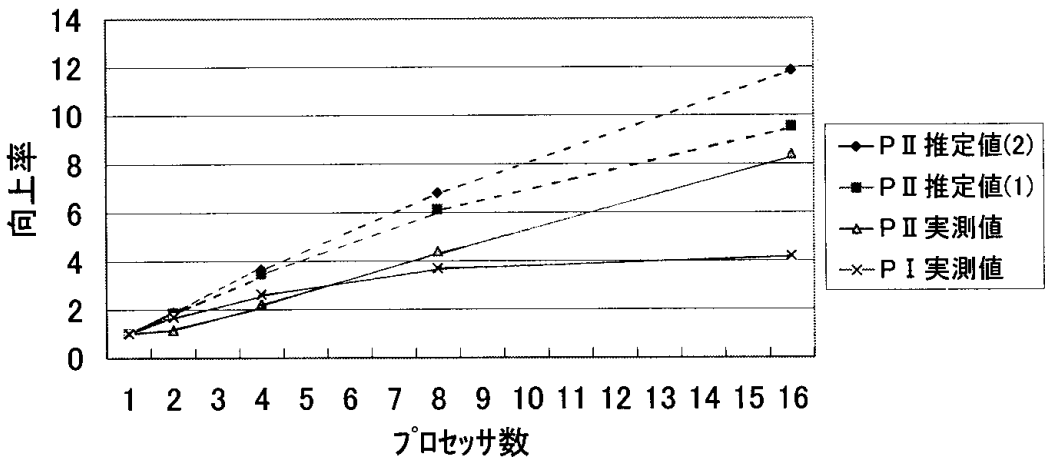


図3.21 プロセッサ数と速度向上率

速度向上率を示し、プログラムⅡは実測値で8.4倍、推定値で11.9倍と大きな差が出ている。更に、プログラムⅠの方が並列処理手法の適用数は多いにもかかわらず速度向上率はプログラムⅡのそれより小さいことがわかる。

プログラムⅡの速度向上率からは、PEを16台以上にしてもまだ速度向上率が増大することが期待できるが、プログラムⅠはPE数を増大してもPE数に見合う速度向上は得られないことが予想される。

第4章　おわりに

Cenju-3システムを使用して、宇宙推進系プログラム2本の並列処理を行った。プログラムⅠはPE数16台で4.6倍の速度向上を得た。プログラムⅡはPE数16台で11.9倍の速度向上を得た。プログラムの特徴によりその速度向上率に大きな差が出ることが得られた。

高速化のために適用された並列処理手法は種々ある。それらを使用する目的・原因もまた1つではない。対象とする計算機を意識すれば、並列処理手法を適用する必要のあった部分のコーディングは別のプログラムコーディングとなっていたであろうものもある。回避不可能なアルゴリズムに起因するものもある。このように、プログラム表現は解明すべき問題、方程式、解法、プログラムコーディングの様々の段階の複合的特徴を持つ。これらがプログラムの並列処理を行う上でどのように関わり、どのような影響を与えるかを研究することは、今後、使い易く効率的なソフトウェアの作成のために大きく役立つことと考えられる。また、これらのデータがCenjuシステムをはじめとするMPPシステムを活かすアルゴリズムの開発に結びつくことを期待したい。

最後に、本研究は航空宇宙技術研究所角田宇宙推進技術研究センターと日本電気株式会社との間で平成5年7月から平成6年12月までに行われた共同研究「宇宙推進系の研究開発に関する並列処理技術の研究」の一部であることを補足する。また、本共同研究及び報告書執筆にあたり、日本電気株式会社、松本寛氏並びに末広謙二氏のご協力に感謝の意を表する。

参考文献

(1) 山内　宗、中田登志之；並列LSIルーターPROTON2－並列マシンCenju2/Cenju3上での評価－、情報処理学会論文誌 Vol.36 No.7, 1995

(2) 村松一弘、鷲尾　巧、土肥　俊；並列マシンCenju2による非圧縮性流体解析、「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ、1994

(3) 加納　健、中田登志之、奥村秀人、大竹邦彦、中村孝、福田正大、小池誠彦；並列計算機のCenju上の有限要素法による非線形変形解析情報処理学会論文誌 Vol.34 No.4, 1993

(4) 丸山　勉、加納　健、広瀬哲也、中田登志之、村松一弘、浅野由裕、稲村　雄；並列コンピュータCenju-3のアーキテクチャとその評価、電子情報通信学会論文誌 Vol. J78-D-I No.2 pp.59-67, 1995

(5) 伊藤勝宏、高橋政浩、平岩徹夫；PointweiseNon-oscillatorスキームの開発と非定常・定常超音速流問題への応用、第6回数値流体力学シンポジウム、1992

航空宇宙技術研究所報告1434号

平成14年1月発行

発行所　独立行政法人　航空宇宙技術研究所
東京都調布市深大寺東町7-44-1
電話（0422）40-3935　　〒182-8522
印刷所　株式会社　実業公報社
東京都千代田区九段北1-7-8

Printed in Japan