ISSN 0389-4010
UDC 681.31

航空宇宙技術研究所報告

TECHNICAL REPORT OF NATIONAL AEROSPACE LABORATORY

TR-1069

CRAY X-MPシステムの マルチタスキングによる並列処理

中村絹代・福田正大

1990年6月

航空宇宙技術研究所
NATIONAL AEROSPACE LABORATORY

CRAY X-MPシステムの マルチタスキングによる並列処理*

中村絹代** 福田正大**

Parallel Processing Using Multitasking on CRAY X-MP System

Kinuyo NAKAMURA and Masahiro FUKUDA

ABSTRACT

We investigated parallel processing using multitasking on the CRAY X-MP/216 system, which is a supercomputer with multiple vector processors. Multitasking is a multiprocessor computer operation mode that executes two or more parts of a single program in parallel. It is one method of high-speed processing.

The FORTRAN programs under investigation are those for a subsonic flow over an airfoil, an unsteady transonic flow over a wing, and an inviscid compressible flow through a two-dimensional cascade. These programs were rewritten so that they could be effectively executed on two CPUs. The computing times of the original programs with one CPU, multitasked programs with one CPU, and multitasked programs with two CPUs are presented in this paper, and the ratio of the computing times of the multitasked programs to those of the original program is discussed.

1. まえがき

計算科学の隆盛と共に計算機の高速処理能力への要求は増大の一途であり，この要求圧力のもとにスーパーコンピュータといわれるものが次々に開発され，成功を収めている。そしてその成功がさらに計算科学の可能性を大きく拓くものとなり，より一層高速な計算機を求めている。

航技研が先導的に示した，流体力学分野における粘性流数値シミュレーション手法の技術的および自然認識ツールとしての有効性も，スーパーコンピュータのこの発展を除いては考えられないものである。この結果は，数値流体力学における新しい領域の開拓，乱流構造の直接シミュレーションへの展望，技術現場での実利用等に向けて，さらに高速のスーパーコンピュータを要求している。

一方，これまで順調に開発されてきたスーパーコンピュータ，特に日本のスーパーコンピュータメーカーが進めてきた単一計算機による処理の高速化は，その性能において飽和に達しつつある。このため，より一層の高速処理を実現するには，計算機内部では日本のメーカーもすでに実現しているベクトル演算方式を，そして，計算機間では米国メーカーがすでに実現している並列処理方式の採用が不可避となっている。

本報告は，計算機間での並列処理方式を採用している単一パイプライン多重計算機方式スーパーコンピュータであるCRAY X-MPを対象に，航技研の計算空力コード3種類に並列処理を適用した結果について報告するものである。以下，2章ではスーパーコンピュータにおける高速処理の概略を，3章ではCRAY X-MPの多重計算機システム及び並列処理の概略を，4章では計算空力コードへの並列処理適用とその結果を記す。

* 平成2年4月20日受付

** 数理解析部

2. スーパーコンピュータにおける高速処理の概略[1)]

高速演算処理を実現するには，計算機の基本実行単位時間であるマシンサイクルタイムの短縮と単位時間内の処理量増大を図ることが必要である[2)]。前者は，高速素子(ガリウム・砒素，HEMT (High Electron Mobility Transistor)，ジョセフソン等)，コンパクトな実装技術，冷却技術等ハードウェア的観点から為されるものである。

一方後者は，計算機の構成方式(アーキテクチャ)の工夫というソフトウェア的観点から為されるもので，処理の並列性を如何にして増大させるかにある。この手段としては，パイプライン処理方式，並列処理方式(現在，多重ベクトル計算機と呼ばれているものも並列処理方式の範疇に入る)等があるが，それぞれの方式の概念は必ずしも排他的ではなく，互いに他の方式を取り入れてより一層の高速化を実現している。

現在最もよく実用化されているスーパーコンピュータはパイプライン処理方式の演算器(以後パイプライン演算器という)を持つベクトル計算機と呼ばれるものである。この方式のスーパーコンピュータでは，ベクトル演算をパイプライン処理するための基本処理時間をパイプラインピッチ時間(以後，パイプラインピッチと言う)と呼び，マシンサイクルタイムと区別している。マシンサイクルタイムとパイプラインピッチが同じスーパーコンピュータもあれば異なっているスーパーコンピュータもある。例えば，富士通の FACOM VP200ではマシンサイクルタイムが14ナノ秒，パイプラインピッチが7ナノ秒，日本電気のSX-2ではマシンサイクルタイム，パイプラインピッチとも6ナノ秒である。

ベクトル演算の処理能力はパイプラインピッチで決まる。或るパイプライン演算器が，1パイプラインピッチ(pナノ秒とする)毎に1つの演算結果を出力するとすれば，そのパイプライン演算器の理論性能－ピーク性能ともいわれる－は，

1000/p−MFLOPS

となる。ここで MFLOPS は，Mega Floating Operations Per Second の略で，1秒間に100万回の浮動小数点演算を実行することを意味し，スーパーコンピュータの処理速度を表す単位として用いられる。

前述したのは最も単純なパイプライン演算器の場合であって，パイプラインピッチを2ナノ秒としても500 MFLOPS にしかならず，このままでは速度向上が小さい。このため，さらに次のような演算器間での並列化，演算器内での並列化の工夫がなされている。

前者の演算器間での並列化は，論理的にも物理的にも複数の演算器が独立に動作するものである。独立に動作するパイプライン演算器がm本あれば，1パイプラインピッチ毎にm個の演算結果が得られることになる。従って理論性能もm倍になる。この機能を持つスーパーコンピュータには富士通の FACOM VP400，日本電気の SX−2 等があり，乗算パイプラインと加算パイプラインが独立に動作し，異種演算を並列に行う。また，日本電気の SX−3，日立の S−810，S−820 のように独立に動作する2つ以上の例えば加算器といったように，同一の演算に対して複数の演算器を備えているものもある。

後者の演算器内での並列化は，論理的には1本のパイプラインであるが，物理的には複数のパイプラインより成っているものである。論理的に1本にまとめられている複数のパイプライン演算器は独立には動作しない。このようなパイプラインのことを多重パイプラインといい，物理的にはn本より成っている1本のパイプラインをn重パイプラインという。多重化されていないパイプラインのことを単一パイプラインという。n重パイプラインでは，1パイプラインピッチ毎にn個の演算結果が得られることになる。従って理論性能もn倍になる。この場合には同一の演算対象，例えば，a＝b＊cの a, b, c は mod(n)で配分され実行される。富士通の FACOM VP400 は4重の加算，乗算パイプラインを備えている。

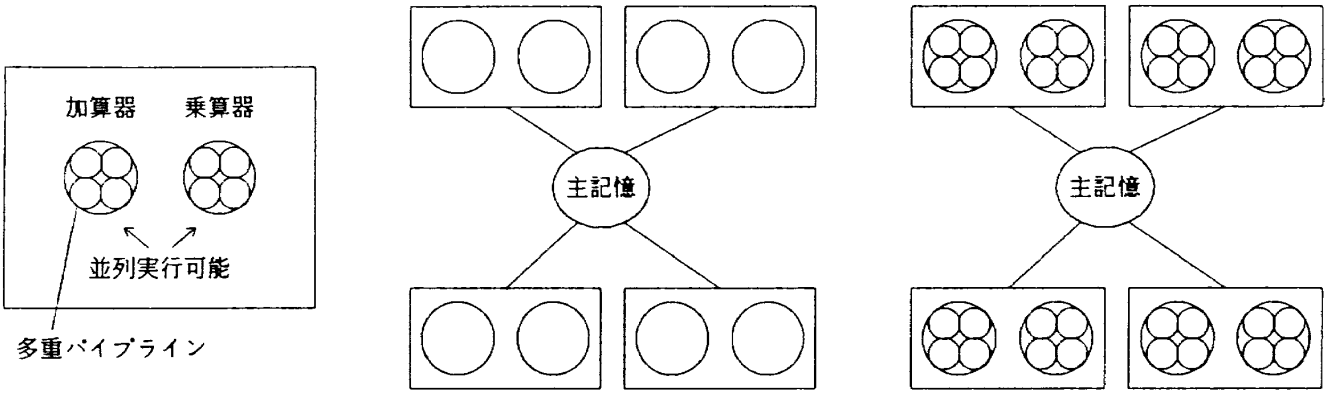
これら二つの工夫を同時に採用すると，並列動作可能なm本のn重パイプライン演算器を持つスーパーコンピュータとなり，1パイプラインピッ

チ毎にm×n個の演算結果が得られることになる。例えば，FACOM VP400は，並列動作可能な4重化された加算と乗算パイプラインを1本ずつ持っている。この結果，VP400では1パイプラインピッチ毎に最大8個の演算結果（4個の加算結果と4個の乗算結果）が得られる。VP400の1パイプラインピッチは7ナノ秒なので，その理論性能は8×1000/7=1143 MFLOPS となる。SX-3は，並列動作可能な4重化された加算と乗算パイプラインを2本ずつ持っており，1パイプラインピッチ毎に最大16個の演算結果（8個の加算結果と8個の乗算結果）が得られ，2.9ナノ秒のパイプラインピッチのもとで，理論性能は16×1000/2.9=5517 MFLOPS となる。

さらに，1台の計算機の内部での並列化に止まらず，計算機システムとして計算機間での並列化を図って高速処理を実現しようとする方法もある。これが通常「並列計算機」といわれているものである。台数が少ない場合には「多重計算機」といわれることもある。これに対して，1台の計算機より成る計算機システムを特に区別して呼ぶ場合には単一計算機という。単一ベクトル計算機方式のスーパーコンピュータとしては，富士通の VP200, VP400，日本電気の SX-1, SX-2 等がある。また，多重ベクトル計算機方式のスーパーコンピュータとしては，CRAY の X-MP, Y-MP, ETA のETA 10の外，ミニスーパーコンピュータといわれているものが数多くある。

単一，多重計算機方式を問わず，演算器を加算器，乗算器，ロード/ストア器，論理演算器等と機能別に有しており，前述したような1計算機内での並列処理方式を併せて採用している。パイプラインの多重化と計算機間での並列処理方式の組み合わせにより，単一パイプライン単一計算機方式，単一パイプライン多重計算機方式，多重パイプライン単一計算機方式，多重パイプライン多重計算機方式の4つが考えられる。これらの構成様式を模式的に示すと図2.1のようになる。多重パイプライン単一ベクトル計算機方式のスーパーコンピュータとしては FACOM VP, NEC SX-1, 2が，単一パイプライン多重ベクトル計算機方式のスーパーコンピュータとしては CRAY X-MP, Y-MP が，多重パイプライン多重ベクトル計算機方式のスーパーコンピュータとしては ETA 10, NEC SX-3 がある。

並列処理方式の計算機の開発計画は数多くあるが，市販されているものは，使い易さ，実行性能向上の容易さの点でベクトル計算機に劣り，また最大処理速度でも劣るため，大規模数値シミュレーションへの利用度は少ない。しかし素子技術による劇的な処理速度向上は期待し難く，また現状素子による単一ベクトル計算機の最大処理速度は数G～10GFLOPS 程度と考えられるので，現在のスーパーコンピュータの数十倍の処理速度達成のためには複数台の構成によるベクトル計算機の多重化が必須となる。商業的に販売されている最



多重パイプライン
単一計算機方式
(a)

単一パイプライン
多重計算機方式
(b)

多重パイプライン
多重計算機方式
(c)

図2.1　パイプライン方式の構成様式

高速のスーパーコンピュータはすでにこの方向にあり(ETA 10, NEC SX-3), これから発表されるものも同様である。ETA社は89年4月にスーパーコンピュータ事業から撤退した。米国では並列処理技術の研究が進んでおり, CRAY Y-MPの並列化処理能力は高いといわれている。

3. CRAY X-MPのシステム及び並列処理の概要

3.1 ハードウェア構成について[3)]

今回利用したシステムは, IBM3090-200をフロントエンド計算機, CRAY X-MP/216システム(以後X-MP/216と呼ぶ)をバックエンド計算機とするシステムである。X-MP/216はパイプライン方式のベクトル計算機を2台, メモリ16MWをもつ2多重ベクトル計算機システムである。ベクトル計算機のマシンサイクルは8.5nsであり, 2台がメモリを共有している。ジョブはIBMシステムを通ってX-MP/216に送られる。図3.1にCRAY X-MP多重計算機システム構成図を示す。

X-MP/216ではCPU(計算機)間通信制御装置により, 2台のCPU間や共有している主記憶の処理調整を行い, 異なるジョブを2台のCPUで別々に並列処理することも, 単一ジョブを2台のCPUで並列に処理することも可能である。前者の使用方法は, ジョブのスループット(システム内滞在時間)を短縮する利点を有しているが1つのジョブの処理に要する時間は変わらない。後者の使用方法は1つのジョブの処理時間の短縮に役立ち, これが本報告で対象とする並列処理である。表3.1にX-MP/216のハードウェア主要諸元を示す。最大性能は2台合計で470 MFLOPSであり[4)], 各CPUには,

(i) 固定小数点加算演算
(ii) 論理演算(2)
(iii) シフト演算

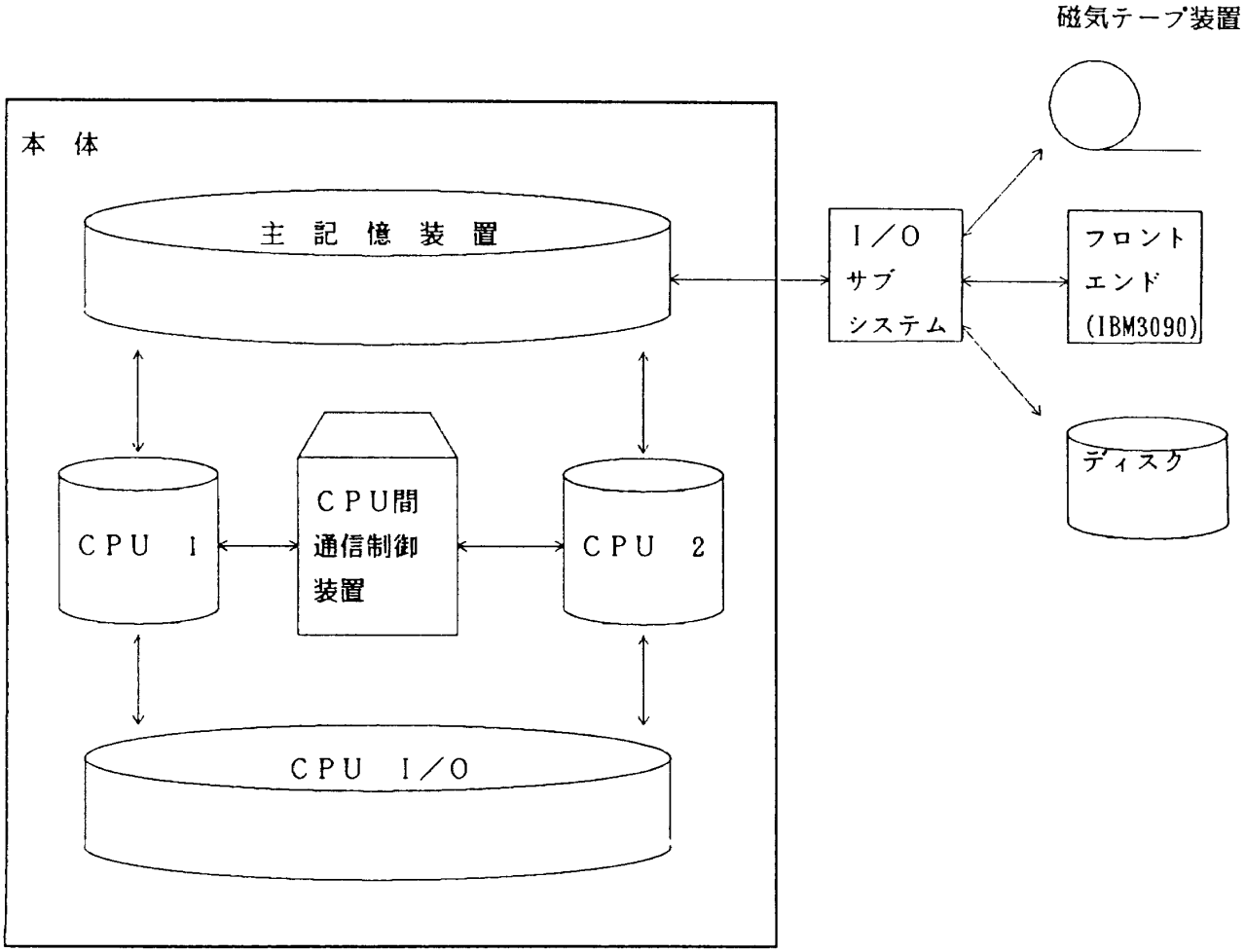


図3.1 CRAY X-MP多重計算機システム

表3.1 CRAY X-MP/216システムのハードウェア主要諸元

項目		性能
最大性能		470*)MFLOPS
CPU数		2
マシンサイクル		8.5 ns
スカラ命令数		111
ベクトル命令数		42
データ形式	論理データ	64 bit
	固定小数点データ	64 bit
	浮動小数点データ	64 bit
処理装置	ベクトルレジスタ	64 word ×8 個
	マスクレジスタ	1
	スカラレジスタ	64 word ×8 個
ベクトル演算器		8
バッファストレージ		4.8 MW
主記憶	容量	16 MW
	インターリーブ	32

*) 参考文献4)

(iv) 計数演算
(v) 浮動小数点加算演算
(vi) 浮動小数点乗算演算
(vii) 浮動小数点逆演算

のベクトル演算器が用意されている。

3.2 並列処理の概要

(1) 並列化について

計算機に与えられた或る単位をもつ仕事(タスク)は，通常逐次的に処理される。2つ以上のタスクが並列に実行可能であるということは，それらのタスクが正確に同時実行可能であることである。タスクのレベルとしては，

(i) ジョブレベル
(ii) ジョブステップレベル
(iii) サブルーチンレベル
(iv) ループレベル
(v) ステートメントレベル
(vi) オペレーションレベル

が考えられる。ジョブレベルからオペレーションレベルへとレベルが高くなるほどタスクの規模は小さくなる。(i)及び(ii)のレベルの並列処理は，通常多重計算機システムでは行われており，ユーザーが特に意識する必要はない。単一ジョブ(ジョ

ブステップ）内で同時実行可能な部分があれば，多重計算機システムでは並列に処理可能であり，(III)～(V)のレベルの並列処理が適用できる。通常ユーザーが並列処理を考えるというのはこのレベルのものである。(VI)のレベルの並列処理は計算機のハードウェア命令レベルのものであり，汎用計算機でもこの技術が取り入れられている。ベクトル計算機にあっても加算機と乗算器の同時実行，チェーニング処理はこれに応じるものである。

並列処理の対象となるタスクの規模は上に述べたように幾種類か考えられるが，そのタスクが並列に実行可能であるというのは次のようなことである。下記のような FORTRAN の実行文を考える。

A = G (B)

C = H (D)

G, H：ある処理単位を代表する文の集合

A=G (B) というのはBというデータの集合にGという手続きを施してAというデータの集合になることを意味する。ここでHの処理を行うに当たってはGに含まれるデータを必要としないものとする。即ち，$A \cap D = \phi$。これらの実行文は通常，図3.2のように逐次的に実行される。しかし，これらの実行は仮定により同時に実行しても演算結果に影響しない。したがって，図3.3のような命令スケジュールを組むことにより並列に実行可能である。

また，Hの処理を行うに当たってGの処理結果Aを利用する場合 ($A \cap D \neq \phi$) には並列に実行すると正常な演算結果が得られず，この場合には逐次実行を行わねばならない。

このように単一ジョブが計算機上で実行される場合，逐次的に処理しなければならない部分とそうでない部分，すなわち，同時実行可能な部分とに分けられる。ジョブ全体の実行時間に占める同

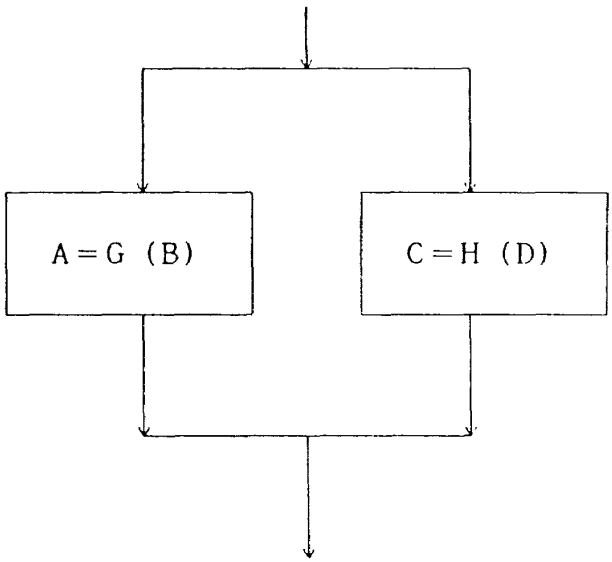


図3.3　並列処理構造図

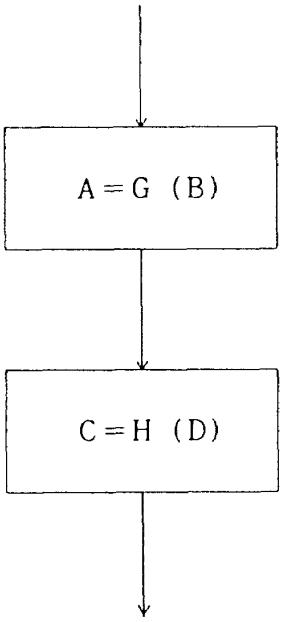


図3.2　逐次処理構造図

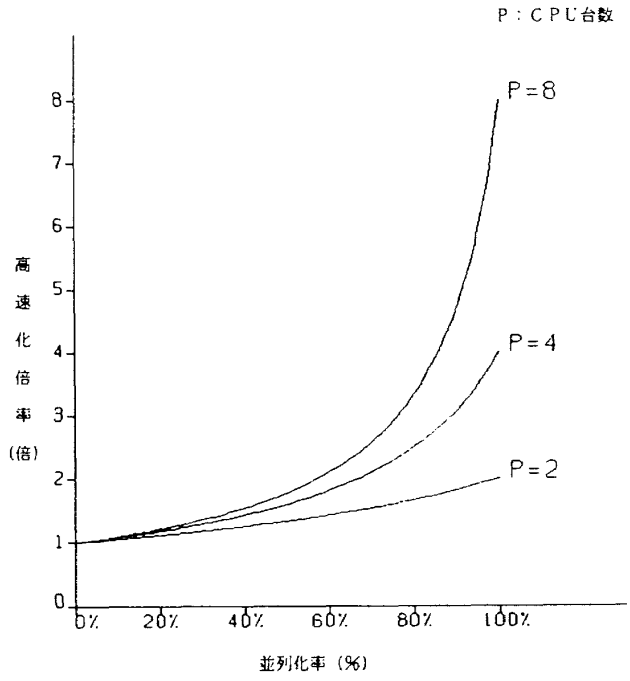


図3.4　並列化率と高速化倍率

時実行可能な部分の実行時間の割合を並列化率と呼ぶ。また同時実行可能部分でも，各処理装置で処理される実行時間には長短のバラツキがあり，このバラツキの程度をワークロードバランスと呼ぶ。従って完全にワークロードのバランスがとれていれば，同時に実行開始になったタスクは同時に実行終了となる。一方の極端として，同時実行可能ではあるが片方のタスクは殆ど無視できる程度の処理時間しかかからない場合が考えられる。この場合には並列処理の効果が殆どないことになる。完全にワークロードのバランスがとれた並列化率80%のジョブを2台の CPU を用いて並列処理すると，並列化のためのオーバーヘッド等を無視する単純計算では，1台の CPU で処理する場合の60% (100−80+80/2) の時間で実行でき，約1.7倍の高速化を達成することができる。

多重計算機システムにより単一ジョブを並列処理した場合，オーバーヘッドがなく完全にワーク

表3.2 マルチタスキング・ライブラリ

ルーチン名		機　　能
タスク制御ルーチン	CALL TSKSTART	タスクを生成する。
	CALL TSKWAIT	他のタスクの実行終了を待つ。
	CALL TSKVALUE	特定のユーザIDを復活させる。
	CALL TSKLIST	実行タスク状態を記録する。
イベント制御ルーチン	CALL EVASGN	イベントとして使用する整変数を示す。
	CALL EVWAIT	特定のイベントがポストされるまで待つ。
	CALL EVPOST	イベントを発生させる。
	CALL EVCLEAR	イベントをクリアする。
	CALL EVREL	イベントとして使用する整変数を解放する。
ロック制御ルーチン	CALL LOCKASGN	ロックとして使用する整変数を示す。
	CALL LOCKON	メモリアクセスをロックする。
	CALL LOCKOFF	メモリアクセスへのロックを解除する。
	CALL LOCKREL	ロックとして使用する整変数を解放する。
チューニングルーチン	CALL TSKTUNE	ライブラリスケジューラ内のチューニングパラメータを修正する。

ロードバランスがとれていると仮定して得られる高速化倍率を図3.4に示す。横軸は並列化率を，縦軸は高速化倍率を示し，PはCPUの台数を示す。当然のことながら並列化率が0%であればCPUを何台並べてもCPU 1台と同じ実行時間であるし，並列化率が100%であればワークロードバランスが完全にとれているとしたので，利用したCPUの台数倍の高速化倍率を得る。しかし実際には，並列処理を行うための準備時間の増大，ワークロードバランスの度合やメモリアクセス競合によるオーバーヘッドがかかり，この図よりは小さい高速化倍率となる。

(2) マルチタスキングについて[5)]

CRAY X-MPシステムにおける単一ジョブの並列処理はマルチタスキングと呼ばれ，それはサブルーチン単位にサポートされるマクロタスキングとサブルーチン，ファンクション，ループ，ステートメントの集合に対してサポートされるマイクロタスキングの2種類に分けられる。

マクロタスキングはサブルーチン相互間レベルでタスクが定義され，並列処理が行われる。タスクの発生やタスク間同期などはすべてFORTRANプログラムからライブラリを呼ぶことにより行われ，タスクの管理はユーザーが行わねばならない。マクロタスキングは複数のサブルーチンにまたがる並列処理を行う場合に向いている。表3.2にマルチタスキングライブラリの一覧表を示す。

図3.5はマクロタスキングを用いて2CPUで並列処理を行う場合の方法を示したもので，(a)は通常のプログラム，(b)，(c)は並列化したプログラ

```
      S=0.0
      DO 20 J=1,N
         DO 10 I=1,N
            A(I,J)=B(I)*C(J)
10       CONTINUE
         S=S+ABS(A(1,J))
20    CONTINUE
```

(a)

```
Task 0
COMMON/MT/A,B,C,N,S,LOCKS
CALL LOCKASGN(LOCKS)
S=0.0
CALL TSKSTART(IDT,T)
DO 20 J=1,N/2
  DO 10 I=1,N
    A(I,J)=B(I)*C(J)
10   CONTINUE
  CALL LOCKON(LOCKS)
  S=S+ABS(A(1,J))
  CALL LOCKOFF(LOCKS)
20 CONTINUE
  CALL TSKWAIT
```

```
Task 1

SUBROUTINE T
COMMON/MT/A,B,C,N,S,LOCKS
DO 20 J=N/2+1,N
  DO 10 I=1,N
    A(I,J)=B(I)*C(J)
10   CONTINUE
  CALL LOCKON(LOCKS)
  S=S+ABS(A(1,J))
  CALL LOCKOFF(LOCKS)
20 CONTINUE
  RETURN
  END
```

(b)

```
Task 0
COMMON/MT/A,B,C,N,S1
S0=0.0
S1=0.0
CALL TSKSTART(IDT,T)
DO 20 J=1,N/2
  DO 10 I=1,N
    A(I,J)=B(I)*C(J)
10   CONTINUE
  S0=S0+ABS(A(1,J))
20 CONTINUE
  CALL TSKWAIT
  S=S0+S1
```

```
Task 1

SUBROUTINE T
COMMON/MT/A,B,C,N,S1
DO 20 J=N/2+1,N
  DO 10 I=1,N
    A(I,J)=B(I)*C(J)
10   CONTINUE
  S1=S1+ABS(A(1,J))
20 CONTINUE
  RETURN
  END
```

(c)

図3.5 マクロタスキング利用例

ムである。(a)のプログラムの DO 20のループを

1　≦J≦N/2

N/2+1≦J≦N

のように，J の値により2つの部分に分けて一方をサブルーチン化し，並列化を行う。(b)の方法では変数Cが共通に使われるため，TASK0でCALL LOCKASGN (LOCKS)でLOCKSという変数をロックとして使用することを宣言する。引き続きCALL TSKSTART (IDT, T)により，このタスクのコントロールアレイとしてIDTを使用し，タスクの入口点としてのサブルーチン名Tを指定してタスクTの生成を行う。この後TASK0とTASK1のDO 10を並列に実行し，またタスク0およびタスク1に共通する実行文〔S=S+ABS(A(1,J))〕の変数の値を保証するため，メモリアクセスのロックおよび解除(LOCKON, LOCKOFF) を行いながら，Sの値を求める。最後にTSKWAITによりサブルーチンTの実行終了を待つ。この例の場合では(c)のように表現すれば，元のプログラムで共通に使われている変数Sを用いなくてもよくなり，オーバーヘッドの軽減が図れる。

```
MAIN PROGRAM
      :
   CALL SUB
      :
STOP
END
```

fray

```
SUBROUTINE SUB
                          control structure
    (process1)
    (process2)
    (process3)

RETURN
END
```

図3.6　並列処理単位構成図

マクロタスキングに対し，マイクロタスキングではタスク管理はシステムが行い，ユーザーはプロセッサの数と並列処理部の指定を補助文を用いて行う。マイクロタスキングは一つのサブルーチン内での並列処理に向いている。

マイクロタスク化されるサブルーチンをfrayといい，同時実行させたい仕事の最小単位をprocessという。processの集合をcontrol structureという。frayは多くのprocessやcontrol structureから成る。その関係を図3.6に示す。

マイクロタスキングのための補助文はCMIC$

```
CMIC$ MICRO
      SUBROUTINE SUB1(A,B,N,M)
      DIMENSION A(M,N),B(M,N)
CMIC$ DO GLOBAL
      DO 100 I=1,N
        DO 50 J=1,M
          A(J,I)=A(J,I)*B(J,I)
   50   CONTINUE
  100 CONTINUE
      RETURN
      END
```

図3.7　CMIC$ DO GLOBAL 補助文の使用例

```
CMIC$ MICRO
      SUBROUTINE SUB2
      COMMON A(100),B(100),C(100),M,
     *       D(100),E(100),F(100),N
CMIC$ PROCESS
      DO 100 I=1,M
      A(I)=B(I)*C(I)
  100 CONTINUE
CMIC$ ALSO PROCESS
      DO 200 I=1,N
      D(I)=E(I)*F(I)
  200 CONTINUE
CMIC$ END PROCESS
      RETURN
      END
```

図3.8　CMIC$ PROCESS 補助文の使用例

で始まる。その種類および機能を表3.3に，またその利用例を図3.7～図3.12に示す。

図3.7は，MICROおよびDO GLOBAL補助文の使用例である。下線のついた補助文 CMIC$ MICRO はサブルーチン SUB1 がマイクロタスク化されるサブルーチンであることを示す（以下の例でも同じ）。同じく CMIC$ DO GLOBAL は DO 100のループが並列処理されることを示す。例えば4 CPU で並列処理されると仮定すれば，

CPU0　I=1, 5, 9, . . .
CPU1　I=2, 6, 10, . . .
CPU2　I=3, 7, 11, . . .
CPU3　I=4, 8, 12, . . .

のように4個おきのIに対して各 CPU は DO 50

表3.3　マイクロタスキングのための補助文

補助文	機能
CMIC$ GETCPUS n	利用するＣＰＵ台数を指定する。ｎ：利用最大ＣＰＵ台数
CMIC$ RELCPUS	マイクロタスキングを行うために獲得したＣＰＵをシステムに戻すことを示す。
CMIC$ MICRO	マイクロタスク化されるサブルーチンであることを示す。
CMIC$ PROCESS	コントロールストラクチュアの始まりを示す。
CMIC$ ALSO PROCESS	コントロールストラクチュア内の第1プロセス以外のプロセスの始まりを示す。かつ、前のプロセスの終了を示す。
CMIC$ END PROCESS	プロセスの終了とコントロールストラクチュアの終了を示す
CMIC$ DO GLOBAL	プロセスのすべてがＤＯループの反復内にあることを意味する。コントロールストラクチュアを意味する。
CMIC$ STOP ALL PROCESS	プロセス又は反復のすべてを実行することなしにプロセスと DO GLOBAL コントロールストラクチュアの両方から外に出る方法を与える。
CMIC$ GUARD n	同時実行から保護されるべきコードのセクションの始まりを示す。コントロールストラクチュア内又はコントロールストラクチュア内で呼ばれるサブルーチン内で使用される。 ｎ：整数０～６３（ｎ＞６３⇨ｎ－ｎ／６４＊６４）
CMIC$ END GUARD n	同時実行から保護されるコードのセクションの終了を示す。 ｎ：整数０～６３（ｎ＞６３⇨ｎ－ｎ／６４＊６４）
CMIC$ CONTINUE	別のサブルーチン内でマイクロタスキングを続行することを指示する。

```
CMIC$ MICRO
      SUBROUTINE SUB1(A,B,N,M)
      DIMENSION A(M,N),B(M,N)
CMIC$ DO GLOBAL
      DO 100 I=1,N
        DO 50 J=1,M
          A(J,I)=A(J,I)*B(J,I)
   50   CONTINUE
  100 CONTINUE
      RETURN
      END
```

(a)

```
CMIC$ MICRO
      SUBROUTINE SUB1(A,B,N,M)
      DIMENSION A(M,N),B(M,N)
CMIC$ PROCESS

        DO 50 J=1,M
          A(J,1)=A(J,1)*B(J,1)
   50   CONTINUE

CMIC$ ALSO PROCESS

        DO 51 J=1,M
          A(J,2)=A(J,2)*B(J,2)
   51   CONTINUE

CMIC$ ALSO PROCESS

        DO 52 J=1,M
          A(J,3)=A(J,3)*B(J,3)
   52   CONTINUE

CMIC$ ALSO PROCESS
            :
            :
            :
CMIC$ ALSO PROCESS

        DO nn J=1,N
          A(J,N)=A(J,N)*B(J,N)
   nn   CONTINUE

CMIC$ END PROCESS

      RETURN
      END
```

(b)

図3.9 CMIC$ DO GLOBAL 補助文と CMIC$ PROCESS 補助文の関係

のループを実行する。

図3.8は，PROCESSおよびALSO PROCESS, END PROCESS 補助文の使用例である。これらは，PROCESS或いはALSO PROCESS文とEND PROCESS文に挟まれた部分が並列に実行されることを意味する。本例では，下線のついた補助文CMIC$ PROCESS と同じく CMIC$ ALSO PROCESS に挟まれた DO 100のループが1つのCPUで，CMIC$ ALSO PROCESS と CMIC$ END PROCESS に挟まれた DO 200のループがもう1つのCPUで，というように2台のCPUで並列に処理される。

```
CMIC$ MICRO
      SUBROUTINE SUB1(IA,IB,A)
      COMMON B(1000),C(1000),M(1000),N
CMIC$ PROCESS
      A=0.0
CMIC$ END PROCESS
CMIC$ DO GLOBAL
      DO 100 I=1,N
      IF(IA.LT.0) THEN
        DO 50 J=1,M(I)
        B(J)=C(J)*C(J)
   50   CONTINUE
      ENDIF
      WORKA=B(M(I))
      IF(IB.LT.0) THEN
CMIC$ GUARD
        A=A+WORKA
        GO TO 100
CMIC$ END GUARD
      ELSE
CMIC$ GUARD
        A=A+WORKA*0.5
        DO 30 K=1,M(I)
        C(K)=C(K)*0.5
   30   CONTINUE
CMIC$ END GUARD
      ENDIF
        :
  100 CONTINUE
      RETURN
      END
```

図3.10 CMIC$ GUARD 補助文の使用例

図3.9はDO GLOBAL補助文とPROCESS補助文により処理されるタスクの関係を示す。これまでの説明からもわかる通り，DO GLOBAL文で繰り返される添字を分解して，その一つずつをPROCESSあるいはALSO PROCESS文とEND PROCESS文で挟んだものが同等となる。即ち，図3.9(a)はDO GLOBAL補助文を用いたプログラム，図3.9(b)はPROCESS補助文を用いたプログラムで，この二つは同等である。DO GLOBAL文は同じ手続を繰り返す（DO型処理）場合に向いており，PROCESS文は異なる手続あるいは一重ループの並列化に向いている。

図3.10はGUARD補助文の例である。この例では下線のついた実行文A＝A＋...はDO GLOBAL補助文により並列処理される各Iに対して共通に使用される配列である。このため，図に示すようにCMIC$ GUARDとCMIC$ END GUARDで挟んで保護しなければならない。

図3.11は並列処理の途中ですべての並列実行を終了させるSTOP ALL PROCESS 補助文の使用例を示したものである。本例では，設定条件を満

```
CMIC$ MICRO
      SUBROUTINE SUB1(A,B,N,M)
      DIMENSION A(M,N),B(M,N)
CMIC$ DO GLOBAL
      DO 100 I=1,N
        DO 100 J=1,M
          A(J,I)=A(J,I)*B(J,I)
            IF(A(J,I).LE.10) THEN
CMIC$ STOP ALL PROCESS
              GO TO 200
              ENDIF
  100 CONTINUE
  200 CONTINUE
      RETURN
      END
```

図3.11 CMIC$ STOP ALL PROCESS 補助文の使用例

オリジナルプログラム

```
C     MAIN PROGRAM

      DO 10 N=1,NEND
        CALL SUB1
10    CONTINUE

      STOP
      END



      SUBROUTINE SUB1

      DO 10 K=1,KL
       DO 10 J=1,JL
        DO 10 I=1,IL
          . . . . .
10    CONTINUE

      CALL SUB2
      RETURN
      END



      SUBROUTINE SUB2

      DO 10 K=1,KL
       DO 10 J=1,JL
        DO 10 I=1,IL
          . . . . .
10    CONTINUE
      RETURN
      END
```

マイクロタスク化プログラム

```
C     MAIN PROGRAM
CMIC$ GETCPUS
      DO 10 N=1,NEND
        CALL SUB1
10    CONTINUE
CMIC$ RELCPUS
      STOP
      END



CMIC$ MICRO
      SUBROUTINE SUB1
CMIC$ DO GLOBAL
      DO 10 K=1,KL
       DO 10 J=1,JL
        DO 10 I=1,IL
          . . . . .
10    CONTINUE
CMIC$ CONTINUE
      CALL SUB2
      RETURN
      END



CMIC$ MICRO
      SUBROUTINE SUB2
CMIC$ DO GLOBAL
      DO 10 K=1,KL
       DO 10 J=1,JL
        DO 10 I=1,IL
          . . . . .
10    CONTINUE
      RETURN
      END
```

図 3.12　他の補助文の使用例

足するDO変数に達したなら，DO GLOBALで指定されたループの外（ラベル200の行）へ実行が移動する。このため，この時実行状態にある他のCPUを停止させる必要があり，このような時にSTOP ALL PROCESS補助文を用いる。

図3.12は上記以外の補助文の使用例である。使用するCPUの獲得と解放は，GETCPUS補助文およびRELCPUS補助文により行われる。マイクロタスク化されたサブルーチン(SUB1)の中から呼び出されるサブルーチン(SUB2)でも並列処理を行いたい時には，CONTINUE補助文により並列処理の続行を指示する。

以上に述べただけで実際のプログラミングが出来るわけではないが，ジョブの並列処理を記述する一つの方法について若干の知識を持って頂けたものと思う。

4. 計算空力コードの並列処理

本章では3章で述べた並列処理を3種類の計算空力コードに適用した結果について述べる。

プログラムAは三次元翼回りの亜音速流れ解析，プログラムBは三次元翼回りの遷音速流れ解析，プログラムCは二次元翼列回りの流れ解析プログラムである。表4.1に適用プログラムの処理目的，次元，方程式，手法および計算法を示す。これらのプログラムを表4.2に示す実行条件でマルチタスク化し，実行時間の計測を行った。

4.1 マルチタスク化の方法

3章で述べたようにマルチタスキングにはマクロタスキングおよびマイクロタスキングの二つの方法がある。今回は，システムがタスク管理を行い，ユーザーにとって比較的容易に利用可能な3種類のマイクロタスキングの適用を試みた。

以下に単純化したプログラムによって説明する。

処理例 1（1重ループの分割）

```
C       MAIN PROGRAM
        COMMON A(100), B(100), C(100), N
        N=100
        . . . . .
```

表4.1 調査対象プログラムの概要

プログラム	処理目的	次元・方程式	手法	計算法
A	亜音速翼回りの流れ解析	三次元、ラプラス	境界要素法	直接法
B	遷音速翼解析	三次元、ナビエストークス	差分法	LU－ADI法
C	翼列解析	二次元、オイラー	有限面積法	陽解法

表4.2 調査対象プログラムの実行条件

プログラム	処理目的	格子	反復回数
A	亜音速翼回りの流れ解析	510パネル	
B	遷音速翼解析	121×31×31	10
C	翼列解析	256×41	2000

```
      . . . . .
      CALL SUB1
      . . . . .
      . . . . .
      STOP
      END

      SUBROUTINE SUB1
      COMMON A(100), B(100), C(100),N
      DO 100 I=1, N
          A(I)=B(I)＊C(I)
  100 CONTINUE
      RETURN
      END
```

前章の説明から，この場合にはDO GLOBAL文，あるいはPROCESS文が使用可能なことが理解できるであろう。本例では，サブルーチンSUB1のDO 100のループが1重であるため，DO GLOBAL文を使用すると，それぞれの添字Iに応じたスカラ処理の繰り返しになってしまう。X−MP/216システムはパイプライン演算器を持っているので，このベクトル処理能力を利用するためには，個々のCPUでもベクトル演算となるように並列化を図る必要がある。そのためには，PROCESS文を用いて，

```
C     MAIN PROGRAM
      COMMON A(100), B(100), C(100),N,
      N=100
      . . . . .
      . . . . .
CMIC$ GETCPUS 2
      CALL SUB1
CMIC$ RELCPUS
      . . . . .
      . . . . .
      STOP
      END

CMIC$ MICRO
      SUBROUTINE SUB1
      COMMON A(100), B(100), C(100), N
CMIC$ PROCESS
      DO 100 I=1, N/2
          A(I)=B(I)＊C(I)
  100 CONTINUE
CMIC$ ALSO PROCESS
      DO 101 I=N/2+1, N
          A(I)=B(I)＊C(I)
  101 CONTINUE
CMIC$ END PROCESS
      RETURN
      END
```

のように，原プログラムのサブルーチン内のDO 100ループを，Iが1～N/2とN/2+1～Nとなる二つの部分に分けて別個のCPUで実行させれば，長さN/2のベクトルとして処理できる。下線を引いた行はマイクロタスキングのための補助文である。

二番目の並列処理方法は次のようなものである。

処理例　2（2重ループのDO GLOBAL文）

```
C     MAIN PROGRAM
      COMMON A(100, 100), B(100, 100),
     1       C(100, 100), N
      N=100
          . . . . .
          . . . . .
      CALL SUB2
          . . . . .
          . . . . .
      STOP
      END

      SUBROUTINE SUB2
      COMMON A(100, 100), B(100, 100),
     1       C(100, 100), N
      DO 100 J=1, N
      DO 100 I=1, N
        A(I,J)=B(I, J)＊C(I, J)
  100 CONTINUE
      RETURN
      END
```

本例でも，前例に述べたようにDO GLOBAL文かPROCESS文の利用が考えられ，この場合には

並列化対象ループが2重ループであるのでDO GLOBAL文の利用が適切となる。並列処理を施したプログラムは,

```
C       MAIN PROGRAM
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
        N=100
            .....
            .....
CMIC$ GETCPUS 2
        CALL SUB2
CMIC$ RELCPUS
            .....
            .....
        STOP
        END

CMIC$ MICRO
        SUBROUTINE SUB2
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
CMIC$ DO GLOBAL
        DO 100 J=1, N
        DO 100 I=1, N
          A(I, J)=B(I, J)*C(I, J)
  100   CONTINUE
        RETURN
        END
```

となる。この場合には個々のCPUの中でのベクトル処理は添字Iによって行われ，ベクトル長Nのベクトル演算となる。並列処理は添字Jによって行われ，Jが奇数(1, 3, 5, ・・・・)の場合はCPU1で，Jが偶数(2, 4, 6, ・・・・)の場合にはCPU2でというように並列処理される。Jの繰り返し回数が予め定数でわかっている場合にはPROCESS文で表現することは可能であるが，特別の目的がない限り適切でない。

三番目の並列処理方法は次のようなものである。

処理例　3（異なるプロセスの並列処理）

```
C       MAIN PROGRAM
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
        N=100
            .....
            .....
        CALL SUB3
            .....
            .....
        STOP
        END

        SUBROUTINE SUB3
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
        DO 100 I=1, N
          A(I, 1)=B(I, 1)*C(I, 1)
  100   CONTINUE
        DO 110 I=1, N
          A(I, N)=B(I, N)*C(I, N)
  110   CONTINUE
        RETURN
        END
```

本例では，DO 100とDO 110が同時実行可能であるからPROCESS文を用いて,

```
C       MAIN PROGRAM
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
        N=100
            .....
            .....
CMIC$ GETCPUS 2
        CALL SUB3
CMIC$ RELCPUS
            .....
            .....
        STOP
        END

CMIC$ MICRO
        SUBROUTINE SUB3
        COMMON A(100, 100), B(100, 100),
       1            C(100, 100), N
```

```
CMIC$ PROCESS
      DO 100 I=1, N
        A(I, 1)=B(I, 1)*C(I, 1)
  100 CONTINUE
CMIC$ ALSO PROCESS
      DO 110 I=1, N
        A(I, N)=B(I, N)*C(I, N)
  110 CONTINUE
CMIC$ END PROCESS
      RETURN
      END
```

とする。この場合にはDO 100はCPU1で，DO 110はCPU2で実行される。

用いた3種類の計算空力コードそれぞれに対して，その全実行時間に対して最も実行時間を費やすサブルーチンに，以上のような並列処理を施した。以下に各計算空力コードに適用した処理例とCPUの獲得および解放の補助文設定位置を示す。

プログラムA

処理例1

CMIC$ GETCPUSはメインプログラムで，マイクロタスク化されるサブルーチンを含む3重ループの直前にある。

CMIC$ RELCPUSはメインプログラムの同じ3重ループの直後にある。

プログラムB

処理例2

CMIC$ GETCPUSはメインプログラムの全体の繰り返しループの直前にある。

CMIC$ RELCPUSはメインプログラムの同じ繰り返しループの直後にある。

プログラムC

処理例2，処理例3

CMIC$ GETCPUSはメインプログラムの最初にある。

CMIC$ RELCPUSはメインプログラムの最後にある。

4.2 並列化率および高速化倍率

2台の要素計算機よりなる並列計算機を考えた場合，並列化率，および並列化による高速化倍率は次のように考えればよい。主タスクを実行するCPUをCPU1とする。

1台の計算機でプログラム全体をスカラ処理したときのCPU使用時間をT，並列処理できない部分をスカラ処理したときのCPU使用時間をT_S，並列処理指定部分をスカラ処理したときのCPU使用時間をT_Pとする($T=T_S+T_P$)。このT_PのうちCPU1で処理した時間をT_{P1}，CPU2で処理した時間をT_{P2}とする($T_P=T_{P1}+T_{P2}$)。このときオーバーヘッドを無視すると，並列処理による高速化倍率a_0は，

$$a_0=T/(T_S+\max(T_{P1}, T_{P2}))$$
$$=1/(1-f_P+\max(b_{P1}, b_{P2})) \qquad (4.1)$$

となる。ここで$b_{P1}=T_{P1}/T$, $b_{P2}=T_{P2}/T$であり，それぞれCPU1およびCPU2の並列処理部分の処理に要した時間のTに対する割合である。$f_P=T_P/T$は並列化率と呼ばれるものであり，$f_P=b_{P1}+b_{P2}$である。並列処理部分でのCPU1とCPU2の負荷が同じであるときには，$T_{P1}=T_{P2}=T_P/2$, $b_{P1}=b_{P2}=f_P/2$となる。

オーバーヘッドを考慮に入れると，並列処理のためのすべてのオーバーヘッド時間の合計をH_Pとすれば，並列処理による高速化倍率a_hは，

$$a_h=T/(T_S+\max(T_{P1}, T_{P2})+H_P)$$
$$=1/(1-f_P+\max(b_{P1}, b_{P2})+h_P) \qquad (4.2)$$

となる。ここで$h_P=H_P/T$である。

X-MP/216システムの要素計算機はベクトル計算機であるから，これにベクトル化の効果が加わることになる。オーバーヘッドを無視した場合には次のように考えれば良い。逐次処理部分(T_S)のベクトル化による高速化倍率をα_{vn}，並列処理部分(T_P)のそれをα_{vp}とすると，ベクトル化を考慮した並列処理による高速化倍率a_{V0}は，

$$a_{V0}=(T_S/\alpha_{vn}+T_P/\alpha_{vp})/(T_S/\alpha_{vn}$$
$$+\max(T_{P1}/\alpha_{vp1}, T_{P2}/\alpha_{vp2}))$$
$$=(r_{pn}\times(1-f_P)+f_P)$$
$$/(r_{pn}\times(1-f_P)$$
$$+\max(r_{P1}\,b_{P1}, r_{P2}\,b_{P2})) \qquad (4.3)$$

となる。ここで$r_{pn}=\alpha_{vp}/\alpha_{vn}$, $r_{p1}=\alpha_{vp}/\alpha_{vp1}$, $r_{p2}=\alpha_{vp}/\alpha_{vp2}$である。オーバーヘッドを考慮した場合も同様に考えることができる。

4.3 結果と検討

以下の説明においては，マルチタスク化される前のプログラムをシングルタスクプログラムと呼び，マルチタスク化されたプログラムをマルチタスクプログラムと呼ぶことにする。

実行時間の測定は，X-MP/216システムとして一つのジョブしか実行しない一多重処理条件下で，シングルタスクプログラムを1CPUだけを使って実行した場合（Sと呼ぶ），同じ条件下でマルチタスクプログラムを1CPUだけを使って実行した場合（M1と呼ぶ）と2CPUを使った場合（M2と呼ぶ），の3通りの条件で行った。それぞれの条件下で実行に要した合計CPU時間を例えばM1(CPU)と表す。またCPU1, CPU2の実行時間を例えばM2(CPU1)と表す。S(CPU)=S(CPU1)，M1(CPU)=M1(CPU1)である。

表4.3は各CPUの実行時間，経過時間，実行待ち時間及びI/O待ち時間の一覧表である。実行待ち時間，I/O待ち時間は一多重処理下であってもフロントエンド計算機との関係などからシステムの状態に左右される。

まず始めに，並列記述したことによる実行時間の増加倍率はM1(CPU)/S(CPU)により与えられる。この増加は，並列記述を行った事により生じたものであり，たとえば，DOループを半分にしたためにベクトル長が短くなり効率が悪くなったこと，及びマルチタスク化のためのオーバーヘッドであると考えられる。この値から1を引いた百分率はプログラムAで14%，Bで4%，Cで11%である。

並列計算機にあっては，この並列化のために要するオーバーヘッドを如何に少なくするかは重要な課題である。例えばn多重計算機の場合，10%のオーバーヘッドがあれば，プログラムの始めから終わりまでを各要素計算機の負荷が等しくなるように完全に並列化できたとしても，その高速化

表4.3 シングルタスクプログラムとマルチタスクプログラムの実行時間測定結果
GOステップ実行時間，単位：秒

プログラム		CPU使用時間			経過時間	実行待ち時間		I／O待ち時間	
		CPU1	CPU2	TOTAL		CPU1	CPU2	CPU1	CPU2
A	S	4.2480	-	4.2480	6.3576	0.0462	-	2.2431	-
	M1	4.8296	-	4.8296	4.9362	0.0502	-	0.1964	-
	M2	3.1147	2.3415	5.4562	3.2615	0.0979	0.0095	0.2145	0
B	S	25.7835	-	25.7835	32.2109	0.1191	-	6.5688	-
	M1	26.7907	-	26.7907	33.0074	0.1247	-	6.3326	-
	M2	23.2224	22.1909	45.4133	30.1082	0.2768	0.0212	6.6445	0
C	S	41.4325	-	41.4325	42.0461	0.0476	-	0.6854	-
	M1	45.8679	-	45.8679	46.5347	0.3782	-	0.3575	-
	M2	29.0369	28.8155	57.8524	29.7681	0.3478	0.3719	0.3931	0

S ：シングルタスクプログラム実行時間（一多重処理下）
M1：マルチタスクプログラム実行時間（一多重処理下、1CPU）
M2：マルチタスクプログラム実行時間（一多重処理下、2CPU）

倍率は，

$1/(1/n+0.1)$

にしかならない。n=2のときこの値は1.67であり，nが無限大になっても10にしかならない。即ち，並列に動作可能な計算機が無限大台数であっても10倍以上の処理速度は得られない。オーバーヘッドの少ないハードウェアを如何に開発するか，またそれを生かすソフトウェアを如何に開発するかの重要性は，この数字からだけでも明らかである。

次に並列処理することによりどれだけ処理が高速化されたかを見る。並列処理すればオーバーヘッドなどのために合計したCPU時間は増加する。従って，並列処理したことによる高速化の効果を見るには合計CPUでの比較ではなく，S(CPU)/M2(max(CPU1, CPU2))の値を使用しなければならない。この値はそれぞれプログラムAでは1.36，プログラムBでは1.11，プログラムCでは1.43である。プログラムBの高速化倍率が小さいのは次に見るように並列化率が小さいことに原因がある。

ここで得られた高速化倍率の値と前述のオーバーヘッドの値を使えば(4.3)式から並列化率f_Pの値が求まる。実測データを使うためr_{pn}，r_{p1}，r_{p2}の効果が含まれており，式としては(4.2)式の形になる。また，今回の調査においてプログラムAおよびBに対して適用された並列化手法では$b_{p1}=b_{p2}=f_P/2$としてよい。この結果プログラムAの並列化率は80.9%，プログラムBは27.8%である。プログラムCには処理例3を適用したため$b_{p1}\neq b_{p2}$である。高速化倍率が同じであっても並列処理部分のロードバランスが悪い場合には，その並列化率は並列処理部分のロードバランスが良い場合に比べて大きくなる。従って，プログラムCの並列化率は$b_{p1}=b_{p2}$の場合の並列化率82.1%より大きい値である。プログラムBの並列化率が小さいのは今回の調査で並列化した部分が小さいのであって，プログラムそのものとしてはまだ並列化可能である。プログラムA，Cについてももう少し並列化する余地はある。またプログラムBのオーバーヘッドが小さいのが，並列化率が小さいことによるものなのか，プログラムの性質によるものなのかは今回の調査では特定できなかった。

実際に複数台のCPUで並列処理させれば，オーバーヘッドのためにユーザープログラムの実行CPU時間の合計は増加する。この増加倍率は，2台のCPUより成る並列計算機では

(CPU1のユーザープログラム実行CPU時間+CPU2のユーザープログラム実行CPU時間)/S(CPU)

の値で与えられ，今回の調査ではM2(CPU)=M2(CPU1)+M2(CPU2)が分子に相当すると考えられる。しかし並列化率の小さいプログラムBではM2(CPU2)の値はM2(CPU1)の値よりかなり小さいと予想されるのに対し，実測値はほぼ同じであることやCPUの確保を示す補助文の挿入位置から考えて，M2(CPU2)の値にはCPUの使用に関係なくCPUを確保している時間も含まれる可能性があると考えられる。もしM2(CPU2)の値としてユーザープログラム実行CPU時間のみのデータが得られれば，並列化によるCPU時間の増加倍率は，

(M2(CPU1)+M2(CPU2))/S(CPU)

表4.4 オーバーヘッド，高速化倍率および推定並列化率

プログラム	オーバーヘッド	高速化倍率	推定並列化率
	(M1(CPU)/S(CPU)-1)X100	S(CPU)/M2(CPU1)	(4.2)式
A	14 %	1.36	80.9 %
B	4 %	1.11	27.8 %
C	11 %	1.43	>82.1 %

として求めることが可能である。

最後に，本節で求められたオーバーヘッド，高速化倍率および推定並列化率を表4.4にまとめておく。

5. まとめ

ベクトル計算機を要素計算機とする多重計算機システムCRAY X-MPを対象に，計算空力コードを用いて並列処理の調査を行った。この計算機システムでは，ベクトル化と同時に並列化できるプログラムとすることが重要である。今回は並列化の調査を目的とし，ベクトル化の分析に必要なデータの取得は行わなかった。この結果，以下の結論が得られた。

i) 今回の調査プログラムでは，並列化のためのオーバーヘッドは4～14%であった。

ii) 一多重実行条件下での高速化倍率は1.1～1.4であった。小さい数字は並列化率が小さいプログラムBによるものであり，これも含めてさらに並列処理を施すことにより倍率を大きくすることが可能である。

iii) CRAY X-MPは所謂TCMP（密結合主記憶共有型多重計算機）であるため，調査対象とした計算空力コードでは，それほど複雑な並列化手法を使うことなく相当程度の並列化が達成できる見込みがついた。

iv) 並列処理モードでエラーが起きたとき，それがプログラムの論理的間違いによるものなのか，或いは並列化の間違いによるものなのか，また後者の場合どのような間違いを起こしたのかなどディバッグツールがベクトル計算機以上の機能を具えている必要がある。

6. むすび

汎用計算機でも採用されている多重計算機システムによる並列処理は，システムとしてのCPU使用効率の向上，全体としての経過時間の短縮を図ることに重点がある。多重処理を行うことにより，遊んでいるCPUを活かすようなシステム環境となっている場合，他のジョブとマルチタスクジョブが混在したときには複数のCPUを独占することは不可能である。その結果，マルチタスクジョブにとって最高のターンアラウンドタイムは保証されず，並列化による高速化が十分には達成されない。NASA Ames研究所ではCRAY-2システム，CRAY Y-MPシステムが稼動しており，単一ジョブの並列処理を行うことは可能であるが，実際には積極的には行われていないようである[6]。

これは運用者の政策でもあるが，CPUを遊ばせてでも並列処理を行って1ジョブのCPU経過時間を短縮するよりは，複数のCPUを別々に使用してシステム効率を上げる方が良いと判断したものと考えられる。

このような運用をする場合には，多重計算機システムが持っている主記憶を複数ジョブで分割しなければならず，航技研が実行しているような計算空力のためには相当大きな主記憶が必要となる。例えば4多重計算機で8多重処理をし，一つのジョブに50MBの主記憶を割り当てようとすれば，400MBのユーザー使用可能主記憶が必要となる。ちなみに航技研のジョブクラスで最大使用可能な主記憶は968MBであり，50MBのジョブは主記憶としては小規模ジョブである。この観点からするとCRAY X-MPの主記憶容量は小さすぎる。

1台のベクトル計算機による処理速度向上は限界に近づいており，いずれにしろ何らかの多重化（並列化）が不可避である。すでに新聞紙上などでも公表されているが，国産次期スーパーコンピュータも多重計算機の方向にある。これをどう運用するかはそれぞれの計算センタの政策であるが，何のために並列化して高速処理を実現しようとしているのかという原点を見失ってはならない。

参考文献

1) スーパーコンピューター製品・技術・応用；日経BP社，1989年6月

2) 渡辺　貞；スーパーコンピュータの高速化技術，第3回航空機計算空気力学シンポジウム論文集，1985年11月

3) The CRAY X-MP Series of Computer Systems，日本クレイ株式会社パンフレット，1987年10月

4) SUPERCOMPUTER HARDWARE : A REPORT BY THE IEEE-USA SCIENTIFIC SUPERCOMPUTER SUBCOMMITTEE, 1989

5) CRAY X-MP MULTITASKING PROGRAMMER'S REFERENCE MANUAL, SN-0222

6) 藤井孝蔵；動き出したNASシステム，日本航空宇宙学会誌 Vol.36, 1988年7月

航空宇宙技術研究所報告1069号

平成2年6月発行

発行所 航空宇宙技術研究所
東京都調布市深大寺東町7丁目44番地1
電話三鷹(0422)47-5911(大代表)〒182

印刷所 株式会社三興印刷
東京都新宿区西早稲田2-1-18

Printed in Japan