計算機アーキテクチャ　95－14
(1992. 8. 19)

Fortranプログラムの階層的マクロデータフロー処理手法

岡本 雅巳+　合田 憲人+　尾形 航+　吉田 明正+　本多 弘樹++　笠原 博徳+
+早稲田大学理工学部　++山梨大学工学部

本論文ではFortranプログラムにおける，ループ間・サブルーチン間の粗粒度並列性を階層的に利用する階層的マクロデータフロー処理手法について述べる．筆者らはすでに粗粒度タスク間の並列性を最早実行可能条件解析を用いて抽出する単階層マクロデータフロー処理手法を実現している．階層的マクロデータフロー処理は，従来の単階層マクロデータフロー処理では利用されていなかったループやサブルーチン等のマクロタスク内部の粗粒度並列性も抽出することを可能にする．特に，本論文ではこの階層的マクロデータフロー処理手法における粗粒度タスク（マクロタスク）の階層的定義手法，マクロタスク間の階層的並列性抽出手法，および階層的に定義されたマクロタスクの階層的なプロセッサクラスタへのスケジューリング方式について述べる．また，本手法のＯＳＣＡＲ上での性能評価についても述べる．

A HIERARCHICAL MACRO-DATAFLOW COMPUTATION SCHEME OF FORTRAN PROGRAMS

Masami Okamoto+　Kento Aida+　Wataru Ogata+　Akimasa Yochida+
Hiroki Honda++　Hironori Kasahara+

+Waseda University　3-4-1 Ohkubo Shinjuku-ku, Tokyo, 169, Japan
++Yamanashi University　4-37 Takeda,Koufu-shi, Yamanashi

This paper proposes a hierarchical macro-dataflow computation scheme which hierarchically exploits the coarse grain parallelism among loops and subroutines in a Fortran program. The authors have already implemented a single layer macro-dataflow processing compiler using the earliest executable condition analysis among the highest level macrotasks. The hierarchical macro-dataflow computation allows us to exploit the coarse grain parallelism inside a macrotask like a subroutine or a loop hierarchically. This paper especially describes a hierarchical definition of macrotasks, a parallelism extaction scheme among macrotasks defined inside an upper layer macrotask and a scheduling scheme which assigns hierarchical macrotasks on hierarchical processor clusters. Also, a performance of the hierarchical macro-dataflow computation is ecaluated on a multiprocessor system OSCAR.

1．はじめに

マルチプロセッサシステム上でのFortranプログラムの並列処理では，従来よりDo-allやDo-across等を用いたループ並列化[1][2]の研究が中心に行われている．市販のマルチプロセッサシステムの多くもこのループ並列化技術を使用している．これらのマシンで使用されているＫＡＰあるいはＶＡＳＴ等の自動並列化コンパイラでは，強力なデータ依存解析[1][3][4]とプログラムリストラクチャリング[1][5]手法を用いて，さまざまな形状のDoloopを並列化できるようになっている．しかし，複雑なループ-キャリド-ディペンデンスや，ループ外への条件分岐等により，並列化できないシーケンシャルループも依然存在する．また，これらのコンパイラでは，ループ以外の並列性，つまり基本ブロック内部での並列性や，ループ，サブルーチンおよび基本ブロック間の並列性を効果的に抽出することができなかった．従って，シーケンシャルループや基本ブロック内部の細粒度並列性や，ループ，サブルーチン及び基本ブロック間の粗粒度並列性を抽出することが今後のマルチプロセッサシステム用自動並列化コンパイラの重要な課題である．

マルチプロセッサシステム上での粗粒度並列処理手法はマクロデータフロー処理[2][6][7][8]と呼ばれる．しかし，このマクロデータフロー処理をFortranプログラムに適用するには，粗粒度タスク，すなわちマクロタスクの定義[9]，マクロタスク間の並列性の自動抽出手法の開発が必要である．筆者らは従来より，このFortranプログラムからののマクロタスク定義手法，及び最早実行可能条件[10]を用いた粗粒度並列性の自動的抽出手法を提案してきた[2][12]．

また，マルチプロセッサシステムにおける細粒度並列処理に関しては，ステートメント程度の粒度をタスクとし，データ転送オーバーヘッドを考慮したスタティックスケジューリングアルゴリズム[2]を用いてプロセッサへのタスク割当てを行う細粒度並列処理手法が提案されている[2][11]．

また筆者らは，粗粒度並列処理（マクロデータフロー処理），中粒度並列処理（ループ並列化），基本ブロックやシーケンシャルループ内での細粒度並列処理を階層的に利用したマルチグレイン並列処理手法[9]もすでに提案し，マルチプロセッサシステムＯＳＣＡＲ上でその性能評価を行っている．

このマルチグレイン並列処理におけるマクロデータフロー処理では，条件分岐やマクロタスク実行時間などの実行時不確定条件に対処するため，マクロタスクはダイナミックスケジューリング[13][14]によってプロセッサクラスタ（ＰＣ）に割当てる方式がとられている．また，プロセッサクラスタに割り当てられた各マクロタスクはDo-allループであれば中粒度，シーケンシャルループあるいは基本ブロックであれば細粒度タスクに分割され，クラスタ内のプロセッサで階層的に並列処理される．

しかし，これまでにインプリメントされているマクロデータフロー処理ではループ内あるいはサブルーチン内で階層的にマクロタスク生成を行うことができなかった．本論文では，ループやサブルーチンなどのマクロタスク内部でマクロデータフロー処理を階層的に適用する階層的マクロデータフロー処理手法のインプリメント法を提案し，その性能について述べる．

2．マクロデータフロー並列処理

本章ではマルチグレイン並列処理でインプリメントされている単階層粗粒度並列処理手法，すなわち単階層マクロデータフロー処理手法について述べる．

2.1.単階層マクロデータフロー処理

ＯＳＣＡＲ Fortranコンパイラでは，以下の1)から5)に示す手順でプログラムを自動並列化する．

1)マクロタスク（ＭＴ）の生成
2)マクロフローグラフの生成
3)マクロタスクグラフの生成
4)ダイナミックスケジューリングルーチンの生成
5)マクロタスクのコードの生成

この単階層のマクロデータフロー処理では，以下に示す３種類のマクロタスク（ＭＴ）を定義する．

1)ＢＰＡ　(Block of Pseudo Assignment)
2)ＲＢ　(Repetetion Block)
3)ＳＢ　(Subroutine Block)

ＢＰＡは単一の基本ブロック，あるいは複数の小基本ブロックを融合したブロック，または基本ブロックを複数に分割したブロックとして定義される．ＲＢはDoループまたはIF文の後方分岐などにより生成されるループ，すなわち最外側ナチュラルループとして定義される．ＳＢは，コード長等を考慮しインライン展開が有効に適用できないサブルーチンである．

これらのマクロタスクは複数のプロセッサクラスタ上で並列処理される．さらにプロセッサクラスタに割り当てられた各マクロタスクはクラスタ内の複数ＰＥ上で並列処理される．ＢＰＡ内部では１ステートメントを１タスクとする細粒度並列処理が適用される．ＲＢ内部では，ループ並列化が有効な場合はループ並列化が適用される．また，ループ並列化が有効でないＲＢ内部では従来は細粒度並列処理が適用されていた．ＳＢ内部ではプログラムをマクロタスクに分割した後，後述するマクロフローグラフを生成し，そのグラフからマクロタスク間の並列性抽出を行わずに，コントロールフローに沿って，基本ブロック内部では細粒度並列処理，ループの内部ではループ並列化もしくは細粒度並列処理を適用するという方法がとられていた．

また，マクロデータフロー処理ではマクロタスク間のコントロールフロー及びデータ依存をマクロフローグラフで表し，このグラフより，マクロタスクの最早実行可能条件[2][12]を求める．このマクロタスクの最早実行可能条件は，マクロタスク間のコントロール依存・データ依存を考慮した最大の並列性を表わしている．さらに，マクロタスクの最早実行可能条件をマクロタスクグラフで表現する．

次にマクロタスクグラフ上のマクロタスクは実行時にダイナミックスケジューリング[13][14]によってプロセッサクラスタに割り当てられる．このスケジューリングではＯＳコール等を使用せずに，コンパイラが各Fortranプログラム毎に専用のダイナミックスケジューリングルーチンを生成してスケジューリングすることで，ダイナミックスケジューリングオーバーヘッドを抑えている[9]．

２．２　階層的マクロデータフロー処理

従来の単階層のマクロデータフロー処理は，ＲＢやＳＢの内側では中粒度並列性および細粒度並列性しか利用していなかった．したがって，生成されるマクロタスク数が非常に少ない場合（図１参照），例えばプログラム全体が大規模ループであるような場合や，マクロタスク間に複雑なデータ依存が存在してマクロタスク間の並列性が少ないような場合には，各マクロタスク内で中粒度並列性あるいは，細粒度並列性が有効利用できない場合には並列性を得られなかった．

例えば図１に示すようなマクロフローグラフを，２プロセッサクラスタ上で単階層マクロデータフロー処理を行なうと，図２に示すようにマクロタスクは一方のクラスタのみに割り当てられ，もう一方のプロセッサクラスタは常にアイドル状態になってしまう．

しかし，ループ（ＭＴ２：ＲＢ）の内部で図１右側のＭＴ２．１～ＭＴ２．６のように複数のサブマクロタスクが定義できるならば，それらのサブマクロタスク間の並列性を利用することにより並列処理効果の向上が期待できる．ここでは，こ

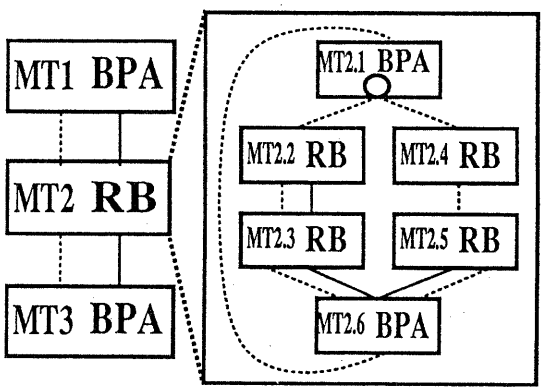

図１　階層的なマクロフローグラフ

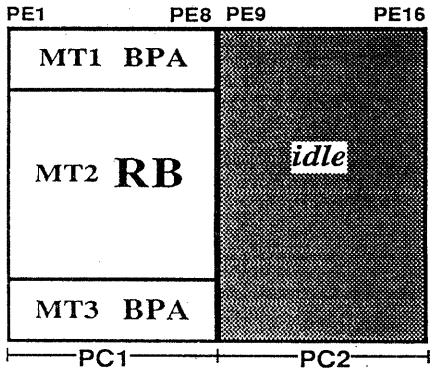

図２　単階層マクロデータフロー処理

のようなサブマクロタスク間並列性（階層的マクロタスク間並列性）を利用するための方法として，従来の単階層マクロデータフロー処理手法を拡張し，マクロタスク内部に対して，マクロデータフロー処理を階層的に階層的に適用する方式をとる．

この方式では，各階層毎にダイナミックスケジューリングルーチンを生成するか，スタティックスケジューリングを利用するかを各階層のマクロタスクグラフの形状に合わせて選択することできるの．

図１のマクロフローグラフに対して，階層的マクロデータフロー処理を適用した場合の割当て例を図３に示す．図３では全体のプログラム（第０階層マクロタスク）を分割して生成したマクロタスクを第１階層プロセッサクラスタに割り当てる（図３左側）．そしてさらに，ＭＴ２の大規模ループ内（ループボディ）を（サブ）マクロタスクに分割し，それらのマクロタスクを，第１階層のプロセッサクラスタを複数の（サブ）プロセッサクラスタ（図中では２クラスタ）に分割した第２階層プロセッサクラスタ割当て，マクロデータフロー処理を行なう（図３右側）．

以下ではこの階層的マクロデータフロー処理コンパイレーション手法について説明する．

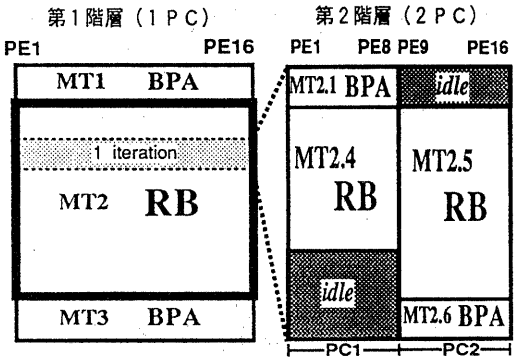

図３　階層的マクロデータフロー処理

2.2.1　階層的マクロタスク生成手法

前述のように階層的マクロデータフロー処理では，ループ並列化が不可能なＲＢのループボディやＳＢの内部では（サブ）マクロタスクを生成する．これらのＲＢやＳＢ内部でも，

ＢＰＡ：基本ブロック
ＲＢ　：各層の最外側ナチュラルループ
ＳＢ　：サブルーチンブロック

の３種類のマクロタスクを生成することが可能である．さらにマクロタスク内で生成された（サブ）ＲＢや，（サブ）ＳＢ内部でも同様にその内部で（サブ-サブ）ＢＰＡ，（サブ-サブ）ＲＢ，（サブ-サブ）ＳＢを生成することができる．また，各階層で生成されたマクロタスクはプロセッサクラスタ内で階層的に定義されるプロセッサクラスタに割り当てられて実行される．

次に，『階層』の概念を具体的に定義する．まずプログラム全体を第０層マクロタスクと定義し，その内部で生成されるマクロタスクを第１層マクロタスクとする．また，第１層マクロタスクの内部で生成される細粒度，中粒度タスク，および粗粒度タスクを第２層マクロタスクとする．さらに第２層のＲＢ，ＳＢ内部で第３層マクロタスクを生成可能であり，さらにその内部でも同様にマクロタスクを定義することができる．この定義によって階層的にマクロタスクを定義していくと，最内側層（最下位層）マクロタスクは細粒度タスクあるいは，中粒度タスクになる．

一方プロセッサクラスタ（ＰＣ）も図５のように階層的に定義できる．まず，マルチプロセッサシステム全体を第０層ＰＣとし，第０層ＰＣ内のＰＥより作られるＰＣを第１層ＰＣとする．第１層ＰＣ内部のＰＥより作られるＰＣを第２層ＰＣとする．さらに，その内側でも同様に第３層ＰＣを生成できる．最内側層（最下位層）ＰＣは１台のＰＥとなる．ただし，プログラムのマクロタスク階層とプロセッサのＰＣ階層はプログラム中の並列性により対応付けられ，必ずしも第ｎ階層の

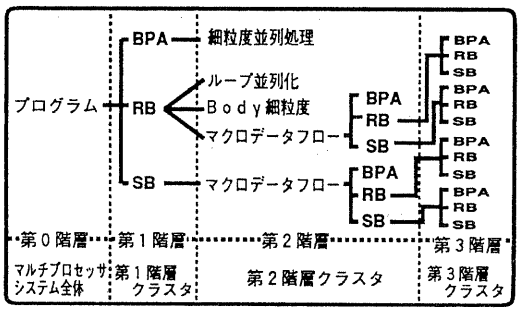

図４　マクロタスクの階層

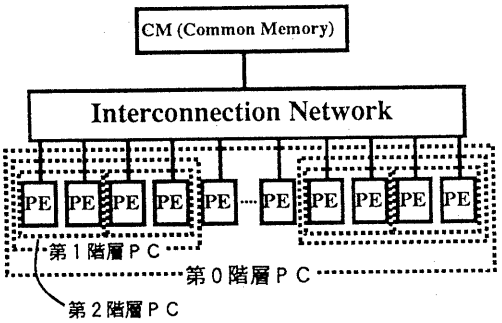

図５　プロセッサクラスタの階層

マクロタスクを第ｎ階層のに割り当てる必要はない．

2.2.2　階層的マクロフローグラフ

各階層でマクロタスクが生成されると，コンパイラはマクロタスク間のコントロールフロー解析とデータ依存解析を行い，図６のように各層でマクロフローグラフを生成する．第１層マクロタスクのマクロフローグラフは従来の単一階層マクロフローグラフと同一である．第１層ＲＢ内部の第２層マクロフローグラフではＲＢが各階層における最外側ナチュラルループであるという定義から，出口ノードから入口ノードへのバックエッジが存在する．しかしループボディ中のバックエッジは次階層（例では第３層）のＲＢ内部に含まれてしまうため，第２層マクロフローグラフでは出口ノードから入口ノードへのバックエッジだけが現れる．このようなバックエッジは，マクロフローグラフはＤＡＧ（無サイクル有向グラフ）であるという性質を満たさず，従来の最早実行可能条件解析が適用できないため，マクロフローグラフの変換が必要になる．図６右側のサブマクロフローグラフをダミーノードの導入によりＤＡＧに変換したマクロフローグラフを図７に示す．ここでは，repeatＭＴというダミーノードを生成し，次イタレーションへの分岐が発生したときに，このノードへ制御が移行するようにする．さらに，exitＭＴというダミーノードをマクロフローグラフの出口に配置し，この階層のマクロタスクの処理が終了したかをチェックするために他の全てのノードから仮想的なデータ依存がこのexitＭＴに対してあるようにグラフを作成する．また，このexitＭＴの出口からは，繰り返しを示す信号や，ループ外への分岐先を示す信号を出力させる．この変換により，バックエッジを除去することができ，マクロフローグラフをＤＡＧとして扱うことができる．

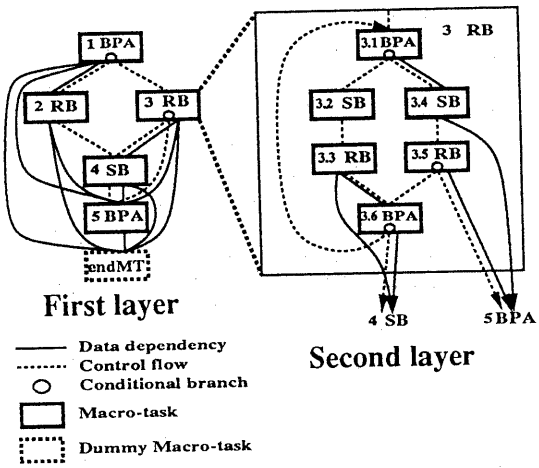

図６　階層的マクロフローグラフ

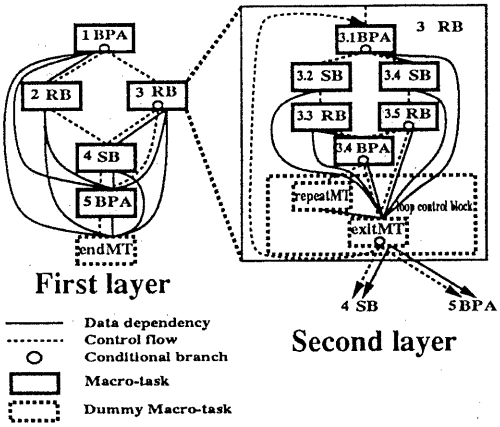

図７　ループ内のマクロフローグラフの変換

2.2.3　階層的マクロタスクグラフ

階層的にマクロフローグラフを生成した後は各階層のマクロタスク最早実行可能条件を求める．ＲＢに対して上述の変換を施した後は各マクロフローグラフに対して従来の最早実行可能条件解析を適用できるので各階層の並列性を抽出しマクロタスクグラフを生成できる．

図７の階層的マクロフローグラフから求めた最

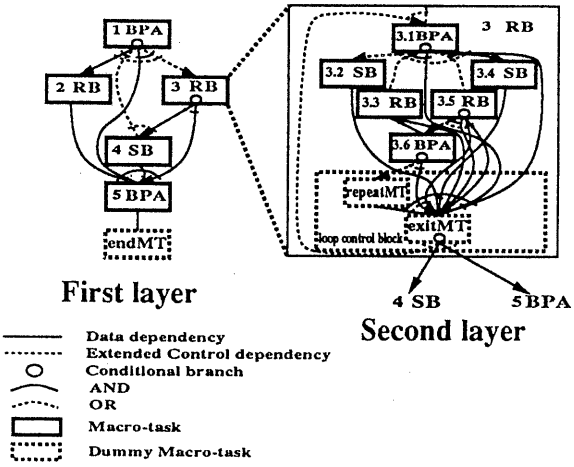

図8　階層的マクロタスクグラフ

早実行可能条件を階層的マクロタスクグラフで表したものを図8に示す.

2.2.4 ダイナミックスケジューリングルーチンの生成

第i階層のマクロタスクは第j階層のプロセッサクラスタにダイナミックに割り当てられる．その際，スケジューラは集中型と分散型の両方の方式をとることが可能である．集中型スケジューラの場合には，コンパイラにより生成されたスケジューリングルーチンが第j階層ＰＣ中の単一ＰＥで実行され，分散型スケジューラの場合にはスケジューリングルーチンが各マクロタスクの前後に挿入され共有メモリ上のスケジューリング用管理テーブルに排他アクセスをしながらスケジューリングを行なう．分散型スケジューラは共有メモリへの排他アクセスが低オーバーヘッドで実現できる場合，あるいはプロセッサ数が少ない場合に有利である．また，集中型スケジューラはＰＥ数が多い場合に有利である．ここでは今回使用するマルチプロセッサシステムＯＳＣＡＲが共有メモリ排他アクセスのための高速ハードウェアを持っていないので集中型スケジューラ方式を用いる．

また，ダイナミックスケジューリングアルゴリズムとしてはＤｙｎａｍｉｃ－ＣＰ法を使用いる．これはスタティックスケジューリングアルゴリズムであるＣＰ法をダイナミックスケジューリング用に拡張したものである．Ｄｙｎａｍｉｃ－ＣＰ法では，コンパイル時に各マクロタスクからマクロタスクグラフの出口ノードまでの最長パスを計算することにより得られる優先度に基づいて，スケジューリングルーチンが実行可能なマクロタスクのＰＣへの割当をおこなう．

3.　ＯＳＣＡＲのアーキテクチャ

本章では今回性能評価に使用したＯＳＣＡＲ[15] (Optimally ScheduledAdvanced Multiprocessor)について述べる．図9にＯＳＣＡＲのアーキテクチャを示す．ＯＳＣＡＲは，１６台のプロセッサエレメント（ＰＥ）と1台のコントロール　＆　Ｉ／Ｏプロセッサ（ＣＩＯＰ）とが，3つの集中型コモンメモリ（ＣＭ）と各ＰＥ上の分散共有メモリに3本のバスを介して接続された，メモリ共有型のマルチプロセッサシステムである．各ＰＥは5MFLOPSの処理速度を持つＲＩＳＣプロセッサであり，６４個の３２ビット汎用レジスタ，整数演算ユニット，浮動小数点演算ユニット，データメモリ，２バンクのプログラムメモリ，分散共有メモリとして使用されるデュアルポートメモリ（ＤＰＭ），スタックメモリ，ＤＭＡコントローラで構成されている．各ＰＥ内のＤＰＭは，そのＰＥと他のＰＥから同時にアクセスすることができる．

また，ＯＳＣＡＲでは，ＤＰＭとＣＭを用いることで次に示すような3種類のデータ転送モードが可能である．

1) ＤＰＭを使用した１ＰＥ対１ＰＥの直接データ転送
2) ＤＰＭを使用した１ＰＥ対全ＰＥのブロードキャスト転送．

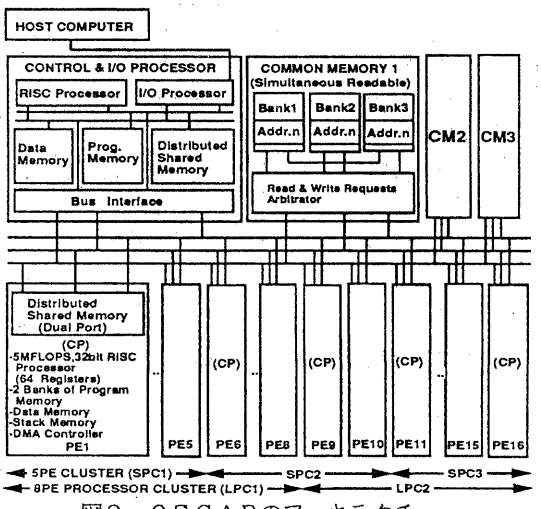

図9　ＯＳＣＡＲのアーキテクチャ

3) ＣＭを使用した１ＰＥ対複数ＰＥの間接データ転送

ＣＭへのアクセスはアドレスが同一であっても，３台のＰＥまでが，同時にリード可能である．

3.1 マクロデータフロー処理のための アーキテクチャサポート

ＯＳＣＡＲの，３つのバスには高速なバリア同期を実現するためのコントロールラインが存在するので，２ＰＣあるいは３ＰＣをシミュレートする場合，各ＰＣに，このコントロールラインを割り当ててマルチプロセッサクラスタシステムをシミュレートすることができる．ただし，効率は低下するがソフトウェアによるバリア同期を用いて４ＰＣ以上のクラスタをシミュレートすることもできる．このようにＯＳＣＡＲではソフトウェアによりマルチクラスタをシミュレートしているため，実行時であっても，ＰＣ数やＰＣ内のＰＥ台数を，対象プログラムの並列性に応じて変更することが出来る．

ＯＳＣＡＲ上でのマクロデータフロー処理においては，ＤＰＭを使用した１ＰＥ対１ＰＥの直接データ転送やブロードキャスト転送は，おもにダイナミックスケジューリングのために使用される．また，共有データはＣＭに割り当てられるため，複数ＰＣ間での共有データの受授は，通常，ＣＭを介して行なわれる．

4． 性能評価

本章ではＯＳＣＡＲ上での階層的マクロデータフロー処理手法の性能評価について述べる．
今回使用したプログラムは連立一次方程式間接解法の１つであるＣＧ法のプログラムである．図１０はＣＧ法プログラムのマクロタスクグラフである．

このＣＧ法プログラムでは，第１層マクロタスクＭＴ１からＭＴ１３までは小規模ループまたは基本ブロックである．しかし，ＭＴ１４が大規模ループであり，実行時間のほとんどがこのループで消費される．この第１層マクロタスクグラフを２ＰＣ上でマクロデータフロー処理すると，ＭＴ１４は単一ＰＣに割り当てられるため，他方のＰＣはアイドル状態になってしまう．そこで，ＭＴ１４内部で図１０の右のように第２層マクロタスクを生成する．第２層マクロタスクは全てが小規模ループまたは基本ブロックになり，マクロタスク間の並列性も得られる．そこで，第１階層マクロタスクを第０階層ＰＣ（すなわち全ＰＥから構成される１ＰＣ）で処理し，第２層のマクロタスクを第１層ＰＣ（２ＰＣ）で処理することにより並列処理効果を得ることができる．

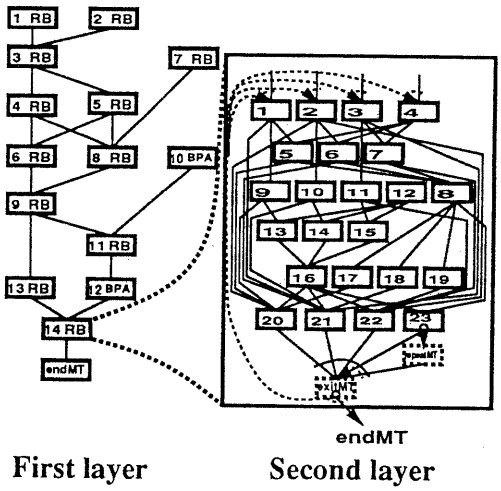

図１０ ＣＧ法プログラムの マクロタスクグラフ

それでは，このプログラムをＯＳＣＡＲ上で実行をした場合の実行時間を述べる．プログラムを１プロセッサで実行した時の実行時間は０．４３１０[ｓ]であった．

このプログラムを，各々が３ＰＥからなる２ＰＣ上で，単階層マクロデータフロー処理を行い並列処理した場合の実行時間は，０．２６９４[ｓ]であり，１プロセッサで実行した場合の１．６０倍のスピードアップであった．

これに対して，階層的マクロデータフロー処理を適用した場合は，０．１６７７[ｓ]であり，１プロセッサで実行した場合の２．５７倍のスピードアップを得ることができた．

5． まとめ

本論文では階層的マクロデータフロー処理手法について提案した．提案手法では，大規模シーケンシャルループや大規模サブルーチン内部の粗粒度並列性を階層的に利用することができる．また，階層的マクロデータフロー処理をインプリメントし，ＯＳＣＡＲ上で性能評価を行った結果，従来

は利用できなかった各マクロタスク内部の階層的な粗粒度並列性が有効に利用できることが確認された.

今後，プロセッサの最適クラスタリング，および，マクロデータフロー処理を適用するマクロタスク階層を，コンパイラが自動的に決定する手法を開発する予定である．またＯＳＣＡＲではクラスタ数を実行時に変更することができるので，更に発展して同一階層のマクロタスクを複数のグループに分割し，各グループの処理のための最適なクラスタ数を求める手法の開発を行う予定である.

参考文献

[1]D.A.Padua , M.JWolfe, "Advanced Compiler Optimizations for Super Computers", C.ACM, Vol.29,12,pp.1184-1201,Dec.1986

[2]笠原,並列処理技術,コロナ社,1991-06

[3]U.Banerjee , " Dependence Analysis for Supercomputing,"Kluwer Academic, Publisher,1988.

[4]D.A.Padua, D.J.Kuck and D.H.Lawrie,"High-speed multiprocessor and compilation techniques,"IEEE Trans.Comput. ,Vol.C-29, No.9,pp763-776,Sep.1980

[5]M.Wolfe, "Optimizing Supercompilers for Supercomputers,"MIT Press,1989.

[6]D.D.Gajiski , D.J.Kuck, D.Lawrie and A.Sameh,"CEDAR,"Report UIUCDCS-R-83-1123, Dept, of Computer Sci., Univ.Illinois at Urbana-Champaign, Feb.1983.

[7]D.D.Gajiski, D.J.Kuck,D.A.Padua, "Dependence Driven Computation,"Proc. of COMPCON 81 Spring Computer Conf.,pp.168-172, Feb. 1981.

[8]H.E.Husmann , D.J.Kuck and D.A.Padua, "Automatic Compound Funstion Definition for Multiprocessors , "Proc. 1988 Int'l Conf. on Parallel Processing, Aug. 1988.

[9]H.Kasahara , H.Honda, S.Narita, "A Multi-Grain Compilation Scheme for OSCAR,"Proc. 4th Workshop on Languages and Compilers for Parallel Computing, Aug.1991.

[10]本多,岩田,笠原,Fortranプログラム粗粒度タスク間の並列性検出手法, 信学論, J73-D-I (12),12 1990

[11]H.Kasahara, H.Honda, S.Narita, "Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR,"in Proc. IEEE ACM Supercomputing'90, Nov. 1990.

[12]H.Kasahara,H.Honda,M.Iwata and M.Hirota, "A Macro-dataflow Compilation Scheme for Hierarchical Multiprocessor System," Proc.Int'l Conf. on Parallel Processing, Aug.1990.

[13]C.D.Polychronopoulos,"Parallel Programming and Compilers,"Kluwer Academic Pub., 1988.

[14]V.Sarkar , Partitioning and Scheduling Parallel Programs for Multiprocessors, MIT Press,1989.

[15]笠原,成田,橋本,ＯＳＣＡＲのアーキテクチャ,信学論D,J71D(8),1991-8