プログラミング－言語・基礎・実践－研究報告　9－10
（1992．　10．　30）

OSCAR Fortran マルチグレインコンパイラ

吉田明正†　岡本雅巳†　合田憲人†　尾形航†　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，ループ・サブルーチン等のような粗粒度タスクの並列処理を行うマクロデータフロー処理，Doall，Doacross等を用いたイタレーションレベルの中粒度並列処理を行うループ並列化，基本ブロック・シーケンシャルループをステートメントレベルの細粒度タスクを用い処理する細粒度並列処理を階層的に組み合わせ、プログラム全体の並列性を効果的に使用するマルチグレイン並列化コンパイラについて述べる．特に，本論文では，マクロデータフロー処理を中心に，マクロタスク分割・融合，階層的マクロデータフロー処理手法について述べる．本マルチグレイン並列化コンパイラは，マルチプロセッサシステムＯＳＣＡＲ上でインプリメントされており，その性能評価についても述べる．

OSCAR Fortran Multi Grain Parallelizing Compiler

Akimasa YOSHIDA†　Masami OKAMOTO†　Kento AIDA†　Wataru OGATA†
Hiroki HONDA‡　Hironori KASAHARA†

†Waseda University　‡Yamanashi University

This paper describes multi grain parallelizing compiler, which exploits parallelism in a Fortran program effectively by macro-dataflow computation, loop concurrentization, and fine grain parallelization. In this compiler, coarse grain tasks such as loops, subroutines and basic blocks are processed on processor clusters by using macro-dataflow technique. A macrotask assigned to a processor cluster is processed on processors inside the cluster by using loop concurrentization techniques like do-all and do-across, near fine grain parallelization technique which exploits parallelism among statements inside a basic block, or hierarchical macro-dataflow technique.

1 はじめに

マルチプロセッサシステム上における Fortran プログラムの並列処理では，Doall，Doacross 等のループ並列化（中粒度タスクの並列処理）[1][2] が従来より広く用いられている．最近のデータ依存解析 [3] とプログラムリストラクチャリング技術 [2] の進歩により，多くのタイプの Do ループが並列化できるようになっている．しかし，イタレーション間にまたがる複雑なデータ依存 (Loop carried dependence) や，ループ外への条件分岐に起因し，ループ並列化が適用できないシーケンシャルループも依然存在する．また，ループ以外の部分の並列性，例えば基本ブロック [4] 内部の並列性や，サブルーチン，ループ，及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．

筆者らは，すでに，マクロデータフロー処理と呼ぶ粗粒度並列処理手法を実現するために，Fortran プログラムからのマクロタスク定義手法，及び最早実行可能条件を用いた粗粒度並列性の自動的抽出手法 [5] を提案してきた．また，細粒度並列処理に関しても，ステートメント程度の細粒度タスクを，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を用いて，プロセッサへ割り当て処理を行う細粒度並列処理手法 [6][14] を提案している．

本論文では，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）および細粒度並列処理を階層的に組み合わせたマルチグレイン並列処理について述べる．このマルチグレインコンパイラは，まず，Fortran プログラムをマクロタスクと呼ぶ粗粒度タスクに分割し，それらの間の並列性を条件分岐（制御依存）とデータ依存を考慮して最早実行可能条件の形で抽出する．次に，マクロタスクをプロセッサあるいはプロセッサクラスタに実行時に割り当てるためのダイナミックスケジューリングコードを生成する．このコンパイラにより生成されたダイナミックスケジューリングコードの利用により，従来問題となっていたＯＳコールなどによる大オーバーヘッドは生じなくなる．また，プロセッサクラスタに割り当てられたマクロタスクは，そのプロセッサクラスタを構成する複数プロセッサで，ループ並列化や細粒度並列処理手法を用いて階層的に並列処理される．

第２章では，マルチグレイン並列処理手法，特に，マクロデータフロー処理におけるマクロタスクの分割・融合手法 [7]，ループやサブルーチン内部でマクロデータフロー処理を階層的に利用する階層的マクロデータフロー処理 [8] について述べる．また，第３章では，本手法をインプリメントしたマルチプロセッサシステムＯＳＣＡＲ [9] のアーキテクチャを紹介し，第４章では，ＯＳＣＡＲ上で行ったマルチグレイン処理手法の性能評価について述べる．

2 マルチグレイン並列処理

マルチグレインの並列化コンパイレーションでは，マクロデータフロー処理，ループ並列化，細粒度並列処理の３つの技術が重要になる．ただし，ループ並列化に関しては，従来から多く研究されており，すでに市販マルチプロセッサシステム上でも一般的なものとなっているので，本章では，主に，マクロデータフロー処理，および細粒度並列処理に関するコンパイレーション手法について述べる．

2.1 マクロデータフロー処理

本節では，Fortran プログラムのマクロデータフロー処理について概説する．

2.1.1 マクロタスク生成

本マクロデータフローコンパイレーション手法では，Fortran プログラムをマクロタスクと呼ぶ並列処理単位に分割する．この時マクロタスクの粒度は，並列実行時の各マクロタスクの処理時間，プロセッサ間データ転送オーバーヘッド，同期オーバーヘッド，スケジューリングオーバーヘッドを考慮して適切に決められねばならない．ここでは，マルチプロセッサシステムＯＳＣＡＲ上での上記オーバーヘッドの大きさを考慮して，擬似代入文ブロック（Block of Pseudo Assign Statements：ＢＰＡ），繰り返しブロック (Repetition Block：ＲＢ)，サブルーチンブロック (Subroutine Block：ＳＢ) の３種類のマクロタスクを生成する．ここで，ＢＰＡは基本的には通常の基本ブロックであるが，基本ブロックの処理時間が短い場合は，複数の基本ブロックを融合し，ＢＰＡを生成する．ＲＢは最外側ナチュラルループ [4] である．また，サブルーチンに関しては，基本的に可能な限りインライン展開を適用するが，コード長が長くなり過ぎ，効果的にインライン展開できない場合には，そのサブルーチンをマクロタスク（ＳＢ）として定義する．

ただし，上述のマクロタスク定義では，十分に並列性が引き出せない場合あるいは粒度が細かすぎてスケジューリングやデータ転送オーバーヘッドが相対的に大きくなってしまう場合がありえる．そこで，本手法ではマクロタスクを再分割し，マクロタスク間並列性を向上させたり，複数マクロタスクを融合することにより [7] 粒度を高め相対的オーバーヘッドを低減する最適化を行う．

例えば，上述のＲＢの定義では，Doall ループが１つのマクロタスクとして１つのプロセッサクラスタに割り当てられてしまうため，プロセッサクラスタ内のプロセッサ台数分の並列性しか利用できない．この問題を解決するために，Doall ループは，分割後の各ループの処理時間がスケジューリング時間より大きい場合に限り，複数（現在はクラスタ台数個）の別々の Doall ループに自動分割される．この分割によりマクロデータフローにおける Doall 処理を，通常のループ並列化と同様，システム中の全プロセッサを用いて行なうことができる．ＯＳＣＡ

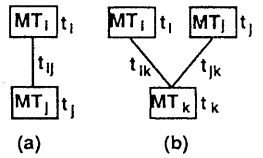

図 1: マクロタスク融合パターン

Rコンパイラでは，n イタレーションを持つ Doall ループがある場合，このループを ⌈n/m⌉ (n:イタレーション数，m:プロセッサクラスタ数) イタレーションからなる m 個の Doall ループに分割する方式をとっている．

また，上述のマクロタスク再分割が行われた後，マクロタスク間のデータ転送オーバーヘッド・スケジューリングオーバーヘッドを軽減するため，ヒューリスティック・マクロタスク融合法を適用する．マクロタスクの実行は条件分岐により実行時に決定されるため，全てのマクロタスクが実行されるとは限らない．そこで，本ヒューリスティック・マクロタスク融合法では，マクロタスクグラフ（MTG）上に，図1に示すパターンのサブグラフが存在する場合，マクロタスク融合を試みる．本手法では図1のようなパターンが発見された場合，その部分グラフを並列処理する際のデータ転送及びスケジューリングオーバーヘッドを考慮した最小並列処理時間と，単一のプロセッサクラスタ上で処理する場合の処理時間を比較し，並列処理時間の方が大きい場合に，それらのマクロタスクを１つに融合する．

このようにマクロタスク間のデータ転送量が多いマクロタスクが融合されると，融合されたマクロタスクは同一プロセッサクラスタに割り当てられるため，プロセッサクラスタ間のデータ転送よるオーバーヘッド及びダイナミックスケジューリングオーバーヘッドが軽減される．

2.1.2 マクロフローグラフ（MFG）の生成

BPA，RB，SBなどのマクロタスク生成後，コンパイラはマクロタスク間の制御フロー，データフローを解析する．解析されたマクロタスク間の制御フロー，データフローは，図2に示すようなマクロフローグラフで表現される．図2において，各ノードはマクロタスクを表し，ノード中の小円は，条件分岐を表している．またノード間の点線，実線はそれぞれマクロタスク間の制御フローおよびデータ依存を表している．図中で矢印は省略されているが，エッジの方向は全て下向きであることを仮定している．また，ループを構成する後方へのエッジ（バックエッジ）はRBの定義によりRB内部に含まれるため，MFGは一般に無サイクル有向グラフとなる．

2.1.3 マクロタスクグラフ（MTG）の生成

マクロフローグラフは，マクロタスク間のコントロールフローとデータ依存関係を陽に表わしたものであり，

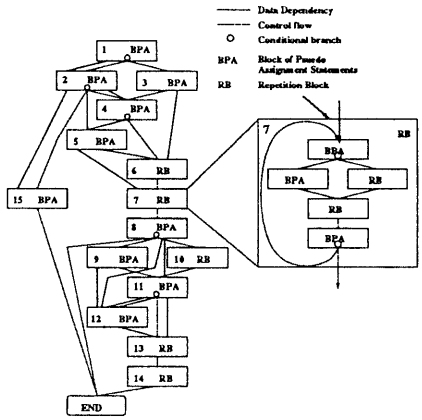

図 2: マクロフローグラフ

マクロタスク間の並列性を表現していない．マクロタスク間にデータ依存関係が存在しない場合には，コントロール依存グラフあるいはプログラム依存グラフ [10] と呼ばれるグラフにより最大の並列性を表現できる．しかしマクロタスク間には，普通，データ依存関係も存在する．従って，マクロフローグラフからマクロタスク間の並列性を効果的に抽出するためには，コントロール依存とデータ依存を同時に解析しなければならない．

本コンパイレーション手法では，コントロール依存とデータ依存を考慮したマクロタスク間の最大の並列性を表わすものとして，各マクロタスクの最早実行可能条件 [2][5] を用いる．マクロタスクｉ（MTｉ）の最早実行可能条件とは，MTｉが最も早い時点で実行可能となるための条件である．ただし，このマクロデータフロー処理における実行可能条件は，次のような実行条件を仮定して求められる．

1) マクロタスクｉ（MTi）がマクロタスクｊ（MTｊ）にデータ依存するならば，MTｊの実行が終了するまではMTｉは実行を開始できない．

2) MTｊの条件分岐先が確定すれば，MTｊの実行が終了しなくても，MTｊにコントロール依存するMTｉは実行を開始することができる．

MTｉの最早実行可能条件の一般形は次の通りである．

[(MTｉがコントロール依存するMTｊがMTｉに分岐する)
ＡＮＤ((MTｉがデータ依存する全てのマクロタスクMTk($0 \leq k < |N|$) の実行が終了する) ＯＲ (MTｋが実行されないことが確定する))]

この条件において，ＡＮＤの前の最初の条件がコントロール依存に起因する実行確定条件である．ＡＮＤの

表1：最早実行可能条件の論理式表現

ブロック番号	実行開始条件
1	
2	1_2
3	$①_3$
4	$2_4 \vee ①_3$
5	$④_6 \wedge (2_4 \vee ①_3)$
6	$3 \vee ②_4$
7	$5 \vee ④_6$
8	$②_4 \vee ①_3$
9	$⑧_9$
10	$⑧_{10}$
11	$8_9 \vee 8_{10}$
12	$11_{12} \wedge (9 \vee ⑧_{10})$
13	$11_{13} \vee 11_{12}$
14	$⑧_9 \vee ⑧_{10}$
15	2_{15}

ｉ：Ｔｉの実行が終了する
①ｊ：Ｔｉが Ｔｊに分岐する
ｉｊ：Ｔｉが Ｔｊに分岐し Ｔｉの実行が終了する

後の条件がデータ依存に起因するデータアクセス可能条件である．しかし，この条件式は冗長な項を含んでいるので，ＯＳＣＡＲ Fortran コンパイラでは，冗長な条件を排除した最早実行可能条件を自動的に生成している．最早実行可能条件における冗長な条件を排除することは，ダイナミックスケジューリングによるオーバーヘッドを減らすために重要である．図2のマクロフローグラフの各マクロタスクの最早実行可能条件を表1に示す．Girkar と Polychronopoulos は，この実行可能条件の研究結果[2][5][15]を利用し，若干の変更を加えた同様のアルゴリズムを発表している[11]．

マクロタスクの最早実行可能条件は図3に示すようなマクロタスクグラフ（ＭＴＧ）と呼ばれる無サイクル有向グラフで表すことができる．ＭＴＧにおいて各ノードはマクロタスクを表す．点線のエッジは拡張されたコントロール依存を示し，実線エッジはデータ依存を表す．この拡張コントロール依存エッジは，通常のコントロール依存だけでなく，ＭＴｉのデータ依存先行タスクが実行されないための条件も表わしている．ＭＴＧ中のノード内の小円を起点とするデータ依存エッジつまり実線のエッジは，コントロール依存とデータ依存の２つを同時に表している（表１中のｉｊに対応）．図中のエッジを束ねている実線のアークは，そのアークによって束ねられたエッジが互いにＡＮＤの関係にあることを示す．点線のアークは，そのアークで束ねられたエッジが互いにＯＲの関係にあることを示す．ノードの内の小円は，ＭＦＧと同様条件分岐を表している．このＭＴＧにおいてもエッジの向きは下向きと仮定しており，ほとんどの矢印は省略されている．矢印がついているエッジは，ＭＦＧ上のオリジナルな分岐方向を表すエッジである．

2.1.4 ダイナミックスケジューリングコードの生成

マクロデータフロー処理では，条件分岐やマクロタスクの実行時間の変動のような，実行時不確定性の問題に対処するため，マクロタスクを実行時にプロセッサクラスタ（ＰＣ）あるいはプロセッサ（ＰＥ）に割り当て

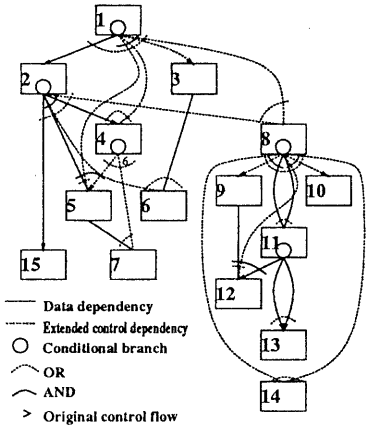

図 3: マクロタスクグラフ

る方式をとる．このダイナミックスケジューリングは，粗粒度タスクに対して適用されるため，スケジューリングオーバーヘッドは相対的に小さく抑えられる．さらに本手法では，ＯＳコール等を用いず，コンパイラにより生成されたダイナミックスケジューリングルーチンを使用するため，さらにオーバーヘッドを小さく抑えることができる．このスケジューリングルーチンは任意のプロセッサ（ＰＥ）上で実行可能である．

ダイナミックスケジューリングアルゴリズムは，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[2]をダイナミックスケジューリング用に拡張したＤｙｎａｍｉｃ－ＣＰ法を用いる．このＤｙｎａｍｉｃ－ＣＰ法は，ＭＴＧ上の各マクロタスクから出口ノードまでの最長パスに基づいて作成されるプライオリティに従って，スケジューリングルーチンが実行時にマクロタスクをＰＣへ割り当てる手法である．

2.2 中粒度並列処理

前節で述べたように，マクロタスクは実行時にＰＣに割り当てられる．ＰＣに割り当てられたマクロタスクがDoall ループで構成されている場合，このマクロタスクは，ＰＣの内部のＰＥによって中粒度すなわちイタレーションレベルの粒度で並列処理される．Doall に関しては，セルフスケジューリング，チャンクスケジューリング，ガイデッドセルフスケジューリングなどといったいくつかのダイナミックスケジューリングの手法が提案されている[2][12]．しかし，ＯＳＣＡＲが，イタレーションをダイナミックスケジューリングするためのハードウェアをもっていないということと，データのローカリティを高めるという理由で，スタティックスケジューリングを用いて，同数のイタレーションを各プロセッサに割り当てている．

ＰＣに割り当てられたマクロタスクが，イタレーション間にデータ依存のあるループの場合，コンパイラは同期オーバーヘッドを短縮するリストラクチャリング [1][13] を適用し、その時の Doacross 処理時間を推定する．次に，コンパイラは Doacross の推定処理時間と，2.3 で述べるループボディの細粒度並列処理の場合の推定処理時間を比較する．Doacross 処理時間が細粒度並列処理時間より短い場合は Doacross のコードを生成する．

2.3 細粒度並列処理

ＰＣに割り当てられたマクロタスクがＢＰＡである場合，マクロタスクは１ステートメントからなる細粒度タスク [6] に分割され，ＰＣ内部のＰＥで並列処理される．

2.3.1 タスク生成とタスクグラフ生成

ＢＰＡを効率よく並列処理するためには，並列性が十分得られ，なおかつデータ転送や同期によるオーバーヘッドをできるだけ少なくなるように，ＢＰＡをタスクに分割することが必要である．本論文では，ＯＳＣＡＲの処理能力やデータ転送能力を考慮して，細粒度タスクの粒度としてステートメントレベルの粒度を用いている．

```
<< LU Decomposition >>
 1) u12 = a12 /l11
 2) u24 = a24 /l22
 3) u34 = a34 /l33
 4) l54 = -l52 * u24
 5) u45 = a45 / l44
 6) l55 = a55 - l54 * u45
<< Forward Substitution >>
 7) y1 = b1 / l11
 8) y2 = b2 / l22
 9) b5 = b5 - l52 * y2
10) y3 = b3 / l33
11) y4 = b4 / l44
12) b5 = b5 - l54 * y4
13) y5 = b5 / l55
<< Backward Substitution >>
14) x4 = y4 - u45 * y5
15) x3 = y3 - u34 * x4
16) x2 = y2 - u24 * x4
17) x1 = y1 - u12 * x2
```

図 4: 細粒度タスクの例

図4はクラウト法によるスパース行列の求解を、シンボリックジェネレーション法を用いてループフリーコードに展開して行うプログラムである．この基本ブロック内のステートメントをタスクと定義すると，タスク間にはデータ依存 [1][3] が存在する．データ依存すなわちタスク間の先行制約は図5に示すようなタスクグラフと呼ばれる無サイクル有向グラフで表される．図中，各タスクは各ノードに対応している．図5においてノードの内の数字はタスク番号ｉを表し，ノードの脇の数字はＰＥ上でのタスク処理時間ｔｉを表す．ＮｉからＮｊに向かって引かれたエッジはタスクＴｉがＴｊに先行するという部分的な順序制約を表す．タスク間のデータ転送も考慮す

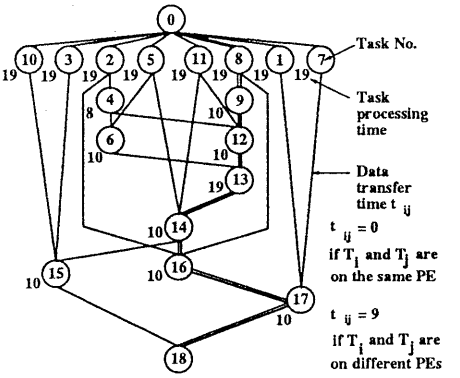

図 5: タスクグラフ

る場合には各々のエッジは通常，可変の重みを持つ．この重みｔijは，タスクＴｉとＴｊが異なるＰＥに割り当てられた場合には，２つのタスク間のデータ転送時間となる．また，２つのタスクが同じＰＥに割り当てられた場合は，重みｔijは０またはレジスタあるいはローカルメモリへのアクセス時間となる．

また，ＯＳＣＡＲでは、後述のように命令を１クロックで実行するRISCプロセッサを採用しているので，コンパイル時にタスク処理時間を正確に求めることができる．

2.3.2 スタティック・マルチプロセッサ・スケジューリング・アルゴリズム

タスク集合をマルチプロセッサ上で効率よく処理するためには，タスクのＰＥへの最適な割当て，および同一プロセッサでのタスクの最適な実行順序の決定を行わなければならない．タスクの最適な割当て，及び最適な実行順序の決定問題は，実行時間最小マルチプロセッサスケジューリング問題として扱うことができる．ＯＳＣＡＲコンパイラでは，スケジューリングに要する時間と生成されるスケジュールの質の双方を考えて，データ転送時間を考慮したヒューリスティックアルゴリズムであるＣＰ/ＤＴ/ＭＩＳＦ法（Critical Path / Data Transfer / Most Immedeate First）[2][14] を採用している．

2.3.3 マシンコード生成

実際のマルチプロセッサ上で細粒度並列処理を効率良く行うためには，スタティックスケジューリング結果を用いて最適な並列マシンコードを生成する必要がある．本コンパイル手法では、スケジューリング結果から次のような情報を使用する．１）各ＰＥでどのタスクが実行されるか．２）同一のＰＥで実行されるタスクの実行順序．３）いつ，どのＰＥ上のどのタスク間でデータ転送，及び同期が必要か．

従って，ＰＥに割り当てられたタスクの命令列を順番に並べ，データ転送命令や同期命令を必要な箇所に挿入することにより，各ＰＥのマシンコードを生成することができる．ＯＳＣＡＲコンパイラは，タスク間で同期をとるためにバージョンナンバー法[14]を用い，また，各ＢＰＡの最後の部分で同期をとるために，ＯＳＣＡＲのハードウェアでサポートしているバリア同期命令を各ＰＥのプログラムに挿入する．

2.4 階層的マクロデータフロー処理

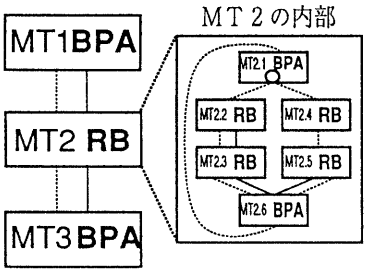

図 6: 階層的なマクロフローグラフ

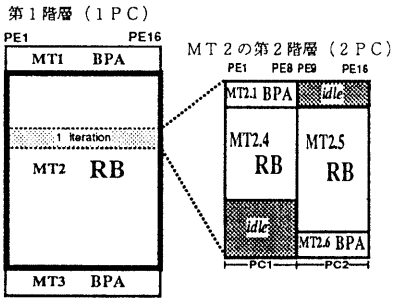

図 7: 階層的マクロデータフロー処理

2.1.1で述べたマクロタスク生成法により生成されたマクロタスク数が非常に少ない場合，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く，かつ，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に利用できない場合には，上述のマルチグレイン並列処理手法では効率よい並列処理が行えない．そこで，各マクロタスク内部でマクロタスク（サブマクロタスク）を定義し，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）を利用することを考える．すなわち，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適用する方式をとる．

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，ＢＰＡ，ＲＢ，ＳＢ等の（サブ）マクロタスクを生成する．その際，プロセッサに関しても，プロセッサクラスタ（ＰＣ）内でさらにプロセッサをグループ化し（サブ）ＰＣを定義し，階層的なマクロデータフロー処理を可能とする．

例えば，図6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実行イメージを図7に示す．図7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(図7左側)に割り当てる．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分割した第２階層プロセッサクラスタ（図7右側：図中では２クラスタ）に割り当て，マクロデータフロー処理を行う．

3 ＯＳＣＡＲのアーキテクチャ

本章では，ＯＳＣＡＲ Fortran コンパイラの性能評価のために使用したマルチプロセッサシステムＯＳＣＡＲのアーキテクチャ[9]（図8）を簡単に説明する．

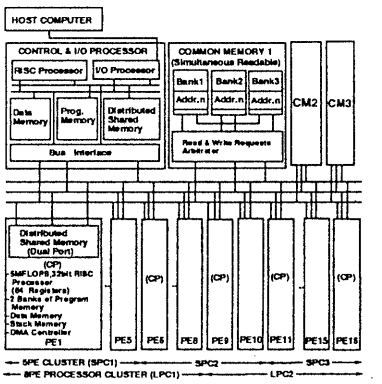

図 8: ＯＳＣＡＲのアーキテクチャ

ＯＳＣＡＲは，ローカルメモリと分散共有メモリを持つプロセッサエレメント１６台を，集中型の共有メモリ（CM）に３本のバスで接続した共有メモリ型マルチプロセッサシステムである．ＯＳＣＡＲの各ＰＥは，３２ビットRISCプロセッサで，ローカルデータメモリ（DM）とローカルプログラムメモリ（PM），分散共有メモリ（DPM），DMA コントローラから成り立っている．各集中型共有メモリモジュールの内部は３バンクに分けられており，３ＰＥが同一アドレスを同時に読み出すことを可能としている．また各バスはバリア同期用のハード

ウェアを持っており，同時に3つまでのバリア同期を数クロックでとることができる．またDMAコントローラは，CMからＰＥへのプレロードとＰＥからCMへのポストストアを可能にしている．

ＯＳＣＡＲは，ＰＥを共有メモリに平等結合したアーキテクチャとなっているが，複数のプロセッサエレメントをグループ化し，1本のバスを割り当てることにより，3クラスタまでのマルチプロセッサクラスタシステムを効率よくシミュレートできる．ＯＳＣＡＲでは，各プログラムの粗粒度タスク間の並列性に応じ，全プロセッサを2または3プロセッサクラスタに分け，マクロデータフロー処理を行うことができる．その際，各クラスタでは，内部のプロセッサエレメントを用いて，各マクロタスクを中粒度（Doallループにおけるイタレーションレベル），細粒度（基本ブロック及びシーケンシャルループのステートメントレベル），または粗粒度（マクロタスク内のＲＢ，ＢＰＡ，ＳＢレベル）で階層的に並列処理することができる．

4　ＯＳＣＡＲ上での性能評価

本章，ＯＳＣＡＲ上でのマルチグレイン並列処理の性能評価について述べる．

4.1　マルチグレイン並列処理の性能評価

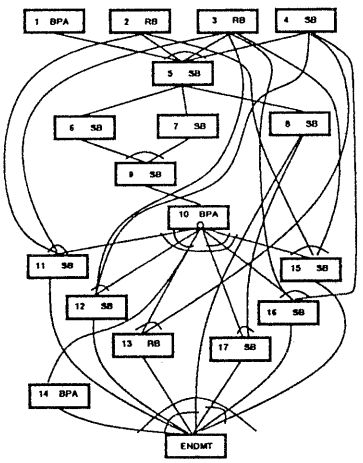

図9: マルチグレイン並列処理評価プログラムのMTG

図9は，ＲＢ，ＳＢ，ＢＰＡの１７個のマクロタスクからなるFortranサンプルプログラムから生成したマクロタスクグラフである．

図9で示されるＭＴＧをＯＳＣＡＲ上で並列実行したときの実行トレースを図10に示す．図10（ａ）は，１ＰＥでプログラムをシーケンシャルに実行した時の実行トレースである．この場合の実行時間は，9.63[s]である．図10（ｂ）は，各々が１ＰＥからなる３つのＰＣで，マクロデータフロー処理だけを行って実行した場合の実行トレースであり，実行時間は，3.32[s]である．さらに，図10（ｃ）は，各々が２ＰＥを持つ３つのＰＣ（つまり６ＰＥ）を使用して，マルチグレイン並列処理を行った実行トレースであり，実行時間は1.83[s]である．このトレースを見ると，マクロタスクとマクロタスク間のダイナミックスケジューリングのオーバーヘッドが極めて小さいことが分かる．また，この実行トレースから，本論文で提案したマルチグレイン並列処理によって，マクロタスク間とマクロタスク内部の並列性を効果的に引き出せていることが確認できる．

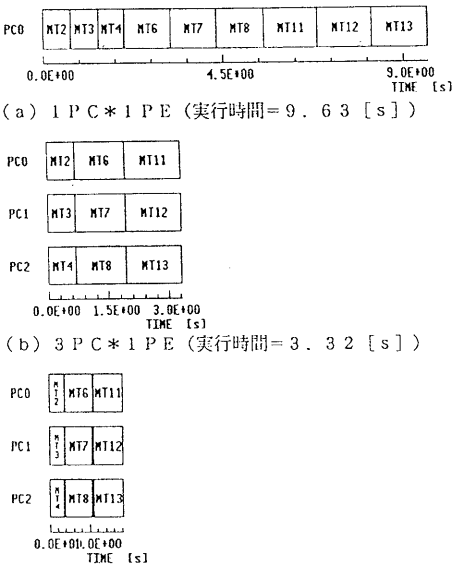

図10: マルチグレイン並列処理の実行トレース

4.2　階層的マクロデータフロー処理における性能評価

ここで，使用するプログラムは連立１次方程式間接解法の一つであるＣＧ法のプログラムである．図11に，ＣＧ法プログラムのマクロタスクグラフを示す．このＣＧ法のプログラムでは，第１階層マクロタスクＭＴ１からＭＴ１３までは小規模ループまたは基本ブロックであり、ＭＴ１４のみが大規模ループで，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階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，階層化マクロデータフロー処理手法により，ＭＴ１４をサブマクロタスクに分割し，その内部で図11

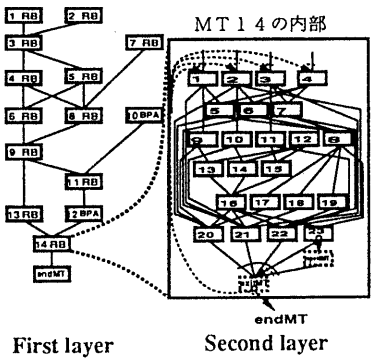

図 11: CG 法のマクロタスクグラフ

右側のように第２層のマクロタスクグラフを作成する．この第２階層マクロタスクは，全てが小規模ループまたは基本ブロックからなり，マクロタスク間にも比較的多くの並列性が存在する．このため，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階層のマクロタスクを第１階層ＰＣ（２ＰＣ）で処理することにより，並列処理をより効率良く行うことができる．

このプログラムをＯＳＣＡＲ上の１プロセッサを用いて実行したときの実行時間は，0.431[s] であった．次に，各々が３ＰＥからなる２ＰＣ（すなわち計６ＰＥ）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，0.147[s] であり，１プロセッサで実行した場合の 2.93 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，0.086[s] であり，１プロセッサで実行した場合の 5.02 倍のスピードアップを得ることができた．

5 むすび

本論文では，Fortran プログラムのマルチグレインコンパイレーション手法について述べた．本マルチグレイン並列化コンパイラは，ＯＳＣＡＲ上で行った性能評価の結果からもわかるように，粗粒度，中粒度，細粒度タスクの並列性を自動抽出し，効率のよい並列処理を可能とすることが確かめられた．さらに，マクロタスク分割・融合，階層的マクロデータフロー処理の実現により，より高い並列処理効果が得られることも確認された．融合されたマクロタスク内でのデータローカライゼーションの実現，また，階層的マクロデータフロー処理における，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の自動決定手法の開発が今後の研究課題である．

参考文献

[1] D.A.Padua, M.J.Wolfe: "Advanced Compiler Optimizations for Super Computers", C.ACM, **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, Pub. (1988).

[4] A.V.Aho,R.Sethi and J.D.Ullman: "Compilers(Principles, Techniques, and Tools)", Addison Wesley (1988).

[5] 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 (D-I), **J73-D-I**, 12, pp951-960 (1990-12).

[6] 本多, 水野, 笠原, 成田: "ＯＳＣＡＲ上での Fortran プログラム基本ブロックの並列処理手法", 信学論 (D-I), **J73-D-I**, 9, pp756-766 (1990-09).

[7] 笠原, 合田, 吉田, 岡本, 本多: "Fortran マクロデータフロー処理のマクロタスク生成手法", 信学論 (D-I), **J75-D-I**, 8, pp.511-525 (1992-08).

[8] 岡本, 合田, 尾形, 吉田, 本多, 笠原: "Fortran プログラムの階層的マクロデータフロー処理", 情処研報, **92-ARC-95**, pp.105-112 (1992-08).

[9] 笠原, 成田, 橋本: "ＯＳＣＡＲのアーキテクチャ", 信学論 (D), **J71-D**, 8 (1988-08).

[10] J.Ferrante, K.J.Ottenstein, J.D.Warren :"The Program Dependence Graph and Its Use in optimization", ACM Trans. on Prog. Lang. and Syst, 9, 3, pp.319-349 (Jul.1987).

[11] M.Girkar and C.D.Polychronopoulos: "Optimization of Data/Control Conditions in Task Graphs", Proc. 4th Workshop on Languages and Compilers for Parallel Computing (Aug.1991).

[12] C.D.Polychronopoulos: "Paralle Programming and Compilers", Kluwer Academic Pub. (1988).

[13] M.Wolfe: "Optimizing Supercompilers for Supercomputers", MIT press (1989).

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

[15] H.Kasahara,H.Honda,M.Iwata and M.Hirota:"A Macro Data- flow Compilation Scheme for Hierarchical Multiprocessor System",Proc.Int'l Conf.on Parallel Processing(Aug.1990).