計算機アーキテクチャ 120－8
（1996．10．31）

実用レベルのマルチグレインFORTRANコンパイラの開発

岡本雅巳† 合田憲人† 吉田明正†
笠原博徳† 成田誠之助†

マルチプロセッサシステムが普及する中，現在市販されている並列化コンパイラを用いても十分なスピードアップを得るのは困難な場合も多く，さらなる並列性の抽出が望まれる．そのためには，従来利用できていなかったループ以外の並列性を自動抽出する並列化コンパイラの開発が重要である．本稿は，従来のループ並列手法のみならず，粗粒度並列性や細粒度並列性などの新しい並列性抽出手法を実装した，筆者等が開発したコンパイラの構造とコンパイレーション手法ついて述べる．実アプリケーションの階層的な並列性の自動抽出の結果も報告する．

Development of A Practical Level Multi-grain FORTRAN Compiler

Masami Okamoto,† Kento Aida,† Akimasa Yoshida,†
Hironori Kasahara† and Seinosuke Narita †

Recently multiprocessor systems are increasingly more popular. However, use of commercial parallelizing compilers have often given enough speed-up, therefore more parallelism is expected to be extracted. In order to exploit more parallelism, it is important to develop a compiler extracting parallelism that was unable to be used before. This paper discribes structure and compilation scheme of the parallelizing compiler, developed by authors, which impletents not only loop parallelization scheme usually used but also coarse grain parallelization scheme and near fine grain parallelization scheme is implemented. The paper also reports results of auto-parallelizing a practical application using the parallelizing compiler.

1. はじめに

近年，市販のスーパーコンピュータはマルチプロセッサ型が一般的になった．また，KAPなどのように自動並列化コンパイラも市販されている例もある．このような並列化コンパイラの研究では古くからループ内部の並列化[1),2)]に重点がおかれており，イリノイ大学のPolaris[3)]やスタンフォード大学のSUIF[4)]などのような先端の並列化コンパイラでは強力なデータ依存解析[1),5),6)]とループリストラクチャリング手法[2),6)~8)]を組み合わせることでさまざまな形状のループが並列化可能になっている．

しかし，複雑なデータ依存やループ外への条件分岐が原因で並列化できないループは存在するため，さらなるスピードアップを得るには，従来のループ並列化手法以外の新しい並列処理手法の開発が重要である．

筆者等は以前より，従来は抽出できなかった基本ブロック内部におけるステートメントレベルの粒度のタスクの近細粒度並列性抽出手法[9)]，および基本ブロック・ループ・サブルーチン間の粗粒度並列性抽出手法[10),11)]を提案している．また，近細粒度並列処理手法・粗粒度並列処理手法(マクロデータフロー処理手法)と従来のループ並列化手法を階層的かつ効果的に組み合わせたマルチグレイン並列処理手法[12),13)]を提案し，コンパイラに実装してきた．この筆者等が開発したマルチグレインFORTRANコンパイラを使用することにより，実アプリケーションの階層的な並列性自動抽出が可能となった．本稿ではマルチグレインコンパイラのコンパレーション手法と，実例として大気の流体プログラムの階層的な並列性自動抽出の解析結果について述べる．

2. マルチグレイン並列処理手法

本節では，本コンパイラで実装されているマクロデータフロー処理手法・近細粒度並列処理手法・およびループ並列化手法のインプリメンテ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手法

マルチグレイン並列処理手法におけるマクロデータフロー処理では，生成した粗粒度タスク(マクロタスク(MT))を，全プロセッサを複数グループに分割して得られるプロセッサクラスタに割り当てる．プロセッ

† 早稲田大学理工学部
School of Sience and Engineering, Waseda University

サクラスタ内部では，近細粒度並列処理やループ並列化を適用するか，あるいはマクロタスクをサブマクロタスクに分割，またクラスタをサブクラスタに分割して階層的にマクロデータフロー処理を適用し[12]、マクロタスクを並列処理する．

2.1.1 マクロタスク生成

マクロデータフロー処理手法では以下の三種類のマクロタスク (MT) を定義する[14]．

- BPA(Block of Peudo Assignments)
  単一の基本ブロック，複数の小基本ブロックを融合したもの，または基本ブロックを分割して得られるブロック．
- RB(Repetition Block)
  DO ループおよび後方への分岐命令によって生成されるループ，すなわち最外側ナチュラルループ．
- SB(Subroutine Block)
  インライン展開が適用されないサブルーチンから生成されるマクロタスク．

BPA 内部では 1 ステートメントや数ステートメントからなる疑似代入文を 1 タスクとして近細粒度並列処理を適用する．RB 内部が DOALL 可能であればループ並列化によって中粒度の並列処理を適用する．DOALL が不可能なシーケンシャルループの場合はループボディに近細粒度並列処理を適用するか，もしくはループボディをサブマクロタスクに分割し階層的にマクロデータフロー処理を適用する．SB 内部は，サブマクロタスクに分割し階層的にマクロデータフロー処理を適用する．階層的にマクロデータフロー処理を適用する場合，マクロタスク内部で生成されるサブマクロタスクはプロセッサクラスタ内部を分割して得られるサブプロセッサクラスタに割り当てられる．マクロデータフローを適用する階層の深さはプログラムの規模や形状などによって任意に決定することができる．マクロタスクは図 1のように，プロセッサクラスタは図 2のように階層的に定義することができる．

マクロタスク間の並列性やデータ転送量を考慮して，複数のマクロタスクを融合もしくは再分割することによって，新たにマクロタスクを生成することも可能である[14),15)]．

2.1.2 マクロフローグラフ生成

マクロタスクの生成後，(サブ) マクロタスク間の制御フロー解析，およびデータ依存解析を行ない図 3のようなマクロフローグラフ[10),12)]という有効グラフを生成する．各ノードはマクロタスク，各実線エッジはデータ依存，各点線エッジは制御フローを表している．またノード内の小円はノードが条件分岐ノードであることを表している．図 3の右側は，最外側 (図 3の左側) の MT3 番内部で生成されたサブマクロフローグラフである．マクロフローグラフは次節で述べる最早実行可能条件解析[10)]に用いられる．最早実行可能条件解析はマクロフローグラフが DAG である性質を利用するが，RB 内部に階層的にマクロデータフロー処理を適用する場合は，図 3の右側に示すサブマクロフローグラフのようにマクロフローグラフの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[12)]ためこのままでは最早実行可能条件解析は適用できない．そこで，RepeatMT というダミーノードを配置し，バックエッジの出発ノードからこの RepeatMT に制御フローエッジを導入し，バックエッジを除去する．次イタレーションが実行確定の判定は RepeatMT が実行可能かどうかチェックすることで行なうことができる．さらに，ExitMT というダミーノードを配置し，全マクロタスクから ExitMT にデータ依存エッジを導入する．イタレーションの実行終了判定は，ExitMT が実行可能かどうかチェックすることで行なう．以上に述べたマクロフローグラフ変換の結果を図 4に示す．

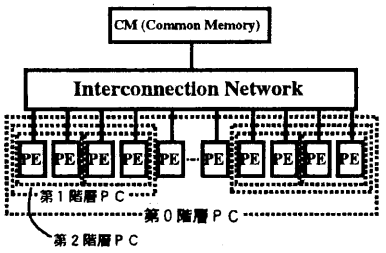

図 1　マクロタスクの階層的な定義

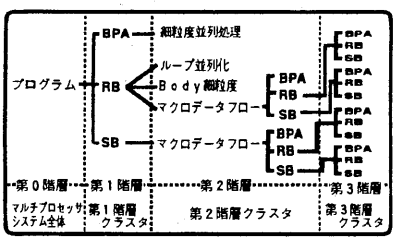

図 2　プロセッサクラスタの階層的な定義

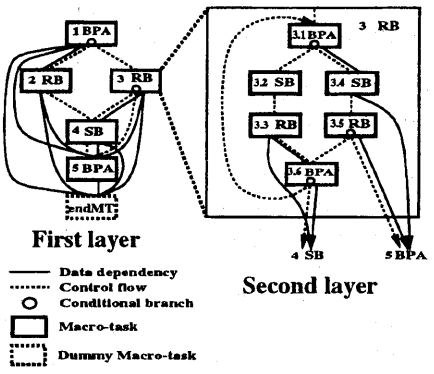

図 3　マクロフローグラフ

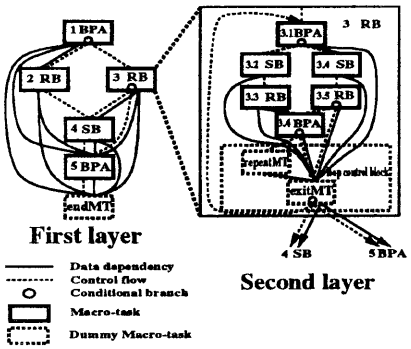

図4　DAG に変換されたサブマクロフローグラフ

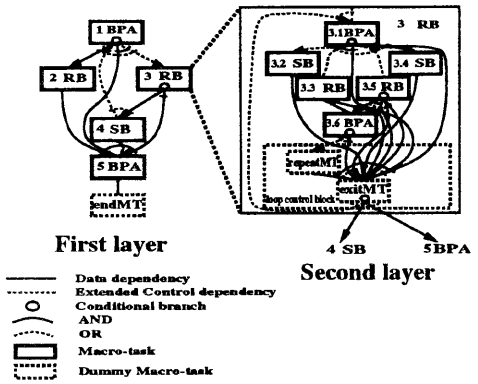

図5　マクロタスクグラフ

2.1.3　マクロタスクグラフ生成

各マクロタスクの最早実行可能条件解析はマクロフローグラフを用いて行なう．最早実行可能条件は制御依存・データ依存を考慮したマクロタスク間の最大限の並列性を表している．ダイナミックスケジューリングによって実行時にマクロタスクをプロセッサクラスタに割り当てる場合，ダイナミックスケジューリングルーチンは最早実行可能条件が成立したかどうかを検査しマクロタスクの実行可能判定を行なう[11]．各マクロタスクの最早実行可能条件は図5のようなマクロタスクグラフ[10),12)]というデータ構造で表現される．各ノードはマクロタスクを表している．各実線エッジはデータ依存，各点線エッジは拡張された制御依存を表している．ここで述べた拡張された制御依存とは一般の制御依存に加え，データ依存先行ノードが非実行を確定する条件分岐に起因する依存を含めたものである．矢印はオリジナルの分岐方向を表す．各エッジに交差する実線弧はその各エッジが互いに AND の関係にあることを表し，各エッジに交差する点線弧はその各エッジが互いに OR の関係にあることを表し．また，ノード内の小円はそのノードが条件分岐ノードであることを表している．例えば図5左の SB4 の最早実行可能条件は

[BPA1 から RB2 へ分岐が確定する
OR
RB3 が終了する]

となる．

2.1.4　マクロタスクのスケジューリング手法

条件分岐などの実行時不確定性の存在しないマクロタスクグラフに関してはコンパイル時にスタティックスケジューリングを適用することが可能である．

条件分岐などの実行時不確定性が存在する場合にはダイナミックスケジューリングによって実行時にマクロタスクをプロセッサクラスタに割り当てる．本手法では，ダイナミックスケジューリングオーバーヘッドを削減するためプログラム全体やその他の各階層専用のダイナミックスケジューリングルーチンをコンパイラが生成する[11]．ダイナミックスケジューリングルーチンはマクロタスクの終了や分岐方向確定に応じてスケジューラ内部のマクロタスク管理テーブルの内容を更新し，それによって実行可能になるマクロタスクの最早実行可能条件を検査する．このマクロタスクが実行可能ならばレディキューに投入し，レディキューの中身を優先順位に基づいてソートし，プロセッサクラスタにマクロタスクを割り当てる．ダイナミックスケジューリングアルゴリズムとしては実行時に出口ノードからのクリティカルパスの長いマクロタスクを優先的にプロセッサに割り当てる Dynamic-CP 法を採用した．

ダイナミックスケジューリング手法は集中スケジューラ手法と分散スケジューラ手法の 2 種類から選択する[16]．集中スケジューラ手法では，マクロタスクを実行するプロセッサクラスタの以外の 1PE にダイナミックスケジューリングルーチンを割り当てる．ダイナミックスケジューリングルーチンの実行はマクロタスクの実行とオーバーラップできるため，スケジューリングオーバーヘッドを極めて低く押えることが可能である．マクロタスク内部で階層的にマクロデータフロー処理を行なう時に，集中スケジューラ方式を適用する場合はプロセッサクラスタ内部の全 PE を複数のサブプロセッサクラスタとスケジューラ用の PE とに分割する．分散スケジューラ方式ではダイナミックスケジューリングルーチンは各クラスタ内 PE の各マクロタスクコード間に挿入される．ダイナミックスケジューリングルーチンはマクロタスク管理テーブルにスケジューラが排他アクセスすることよって実行される．スケジューラ専用に PE が不要なので，使用できるプロセッサ数が少ない場合には有利な手法である．

2.2　近細粒度並列処理手法

BPA 内部では近細粒度並列処理が適用される[9),12),13)]．近細粒度並列処理では FORTRAN の 1 ステートメントもしくは数ステートメントで構成され

る疑似代入文からなるタスク生成する．各タスクはコンパイル時にスタティックスケジューリングによってクラスタ内の各 PE に割当てられる．スケジューリングアルゴリズムとしてはデータ転送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 と DT/CP の双方のスケジューリング結果を比較し良好な解が得られた方を採用するようにした．

2.3 ループ並列化手法

DOALL が可能なループに関しては DOALL で処理する．各イタレーションはコンパイル時にスタティックスケジューリングによってクラスタ内の各 PE に同数のイタレーションが割り当てられる．

3. コンパイラの構成

本マルチグレイン並列化コンパイラは大きく分けて以下の 3 つの部分で構成される．

- FE(フロントエンド)
  FORTRAN77 のソースプログラムからシーケンシャルな中間言語への変換を行なう．
- MP(ミドルパス)
  制御フロー解析・データ依存解析を行ない，プログラムのリストラクチャリング，タスクの生成および並列性の自動抽出を行なう．解析結果に基づいて，シーケンシャルな中間言語を並列化された中間言語に変換する．
- BE(バックエンド)
  並列中間言語からターゲットマシン用のマシンコード，もしくは並列処理用に拡張された FORTRAN や C 言語のソースコードを生成する．

並列化の全ての処理は MP が行なっている．MP の処理の流れを図 6に示す．FP 全体や MP のタスクのコスト算出などの一部の処理を除くの殆んどの部分はターゲットアーキクチャを変更してもそのまま利用することができる．BE は，最適なコードが生成できるように各アーキテクチャ毎に用意される．現在，OSCAR の他，Fujitsu VPP-500，Kendall Square Research KSR1，NEC Cenju-3 などの BE が開発されている．

4. ターゲットアーキテクチャ

ターゲットアーキテクチャとしては図 2に示したような平等型共有メモリマルチプロセッサを仮定している．ダイナミックスケジューリング手法を用いたマクロデータフロー処理ではマクロタスクの割り当てが実行時に確定するため，どの変数がどのプロセッサから参照されるかが実行時まで確定することができない．そのためマクロタスク間の共有変数は共有メモリに配置される．近細粒度並列処理手法を適用する場合，データ転送オーバーヘッド・同期オーバーヘッドを以下に削減するかが重要で，データ転送命令は一般の演算命令に比べ高速である必要がある．この解決法としては各 PE 上に高速な分散共有メモリを配置し，転送元の PE が転送先の PE 上のメモリに直接書き込めるようにすることなどが考えられる．

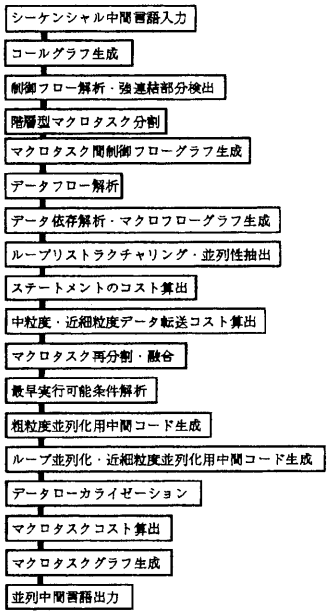

図 6　ミドルパスの処理の流れ

マクロデータフロー処理手法と近細粒度並列処理手法を階層的に効果的に実装可能なアーキテクチャとして以下に述べるマルチプロセッサシステム OSCAR[17] が挙げられる．

4.1 OSCAR のアーキテクチャ

OSCAR は 16PE が 3 本のバスを介して集中共有メモリ (CSM) に接続された平等型の共有メモリ型マルチプロセッサシステムである．各 PE はローカルメモリ (LM)，分散共有メモリ (DSM) をもっている．

OSCAR では次の 3 種類のデータ転送モードが用意されている．

- 分散共有メモリを介した 1PE 対 1PE のデータ転送．
- 分散共有メモリを介した 1PE 対全 PE のデータのブロードキャスト転送．
- 集中共有メモリを介した 1PE 対複数 PE のデータ転送．

ローカルメモリや自 PE 上の分散共有メモリは 1 クロックでリードライト可能である．また集中共有メモリや他 PE 上の分散共有メモリなどバスを介したメモリアクセスでは最短で 4 クロックのリードライトが可能である．

4.1.1 OSCAR におけるマクロデータフロー処理のためのハードウェアサポート

ダイナミックスケジューリングを適用する場合，実

行時まで割当が確定できないのでマクロタスク間の共有データは一般には集中共有メモリもしくは分散共有メモリに配置される．ただし，データローカライゼーション手法[13]をもちいることによって，データをローカルメモリに配置し必要な時に一括して転送することが可能になる．ダイナミックスケジューラと各プロセッサクラスタ間の通信は分散共有メモリを介して行ない，スケジューリングオーバーヘッドを削減している．

4.1.2 OSCARにおける近細粒度並列処理のためのハードウェアサポート

OSCARのプロセッサは浮動小数点の加減乗算を含む全ての命令を1クロックで実行可能であるため，タスクの実行時間を正確に算出可能である．これによってスタティックスケジューリングで良質な解を得ることができる．また，タスクの実行タイミングを正確に予測することで，すべての同期コードを削除しNOPコードを挿入することにより無同期近細粒度並列処理[9]も実現できる．OSCARでは1PE対1PEデータ転送モードやブロードキャスト転送を用いた1PE対全PEデータ転送モードが最短で4+1=5クロックと高速であり、近細粒度並列処理の際のデータ転送オーバーヘッドを低く押えている．

5. 解析結果

本稿では，マルチグレインコンパイラによって実アプリケーションを階層的に並列性を自動抽出した結果について述べる．ここではPerfect benchmarksに収録されているADMを用いた解析結果について述べる．ADMは大気汚染の流体解析のプログラムであり，アルゴリズムとして高速フーリエ変換を用いている．図7はADMのメインプログラムより得られる第1階層マクロタスクグラフである．図8はADMのサブルーチンPSET内の第2階層マクロタスクグラフで，メインプログラム(図7)のMT35よりCALLされる．図9はADMのサブルーチンRUN内の第3階層マクロタスクグラフで，図8のMT3よりCALLされる．図10はADMのサブルーチンRUN内の第4階層マクロタスクグラフで，図9のRB(MT16)の内側で生成される階層である．図11はADMのサブルーチンDCTDY内の第5階層マクロタスクグラフで，図10のRB(MT5)の内側で生成される階層である．図12はADMのサブルーチンDCTDY内の第6階層マクロタスクグラフで，図11のMT4からCALLされる．図9の第4階層や図12の第6階層などで並列性が得られることがわかる．また，マクロタスクのコストとダイナミックスケジューリングオーバーヘッドを考慮した場合，複数のDOループやサブルーチン間で粗粒度並列性が得られている図12の第6階層での粗粒度並列処理によるスピードアップが期待できる．

6. おわりに

本稿では，マルチグレイン並列化コンパイラの構造，コンパイレーション手法について述べた．本コンパイラでさまざまなアプリケーションの粗粒度・近細粒度並列性を自動抽出が可能になっており，性能評価を現在行っている．

今後，インタープロシージャ解析を用いて粗粒度並列性をさらに利用可能にしてゆく予定である．また，OSCAR以外のターゲットアーキテクチャのBEの開発を進め，マルチプラットフォームコンパイラとして発展させる予定である．

参考文献

1) 笠原博徳: 並列処理技術, コロナ社 (1991).
2) Banerjee, U.: *Loop Parallelization*, Kluwer Academic Pub. (1994).
3) Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lee, J. and Padua, D.: Advanced Program Restructuring for High-Performance Computers with Polaris, Technical Report 1473, CSRD, University of Illinois, Urbana-Champaign (1996).
4) Amarasinghe, S., Anderson, J., Lam, M. and Tseng, C.: The SUIF Compiler for Scalable Parallel Machines, Proceedings of the seventh siam conference on parallel processing for scientific computing (1995).
5) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic Pub. (1988).
6) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley Pub. Co., Inc. (1996).
7) Pugh, W.: The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependency Analysis, *Proc. Supercomputing '91* (1991).
8) Blume, W., Eigenmann, R., Hoeflinger, J., Petersen, P., Rauchwerger, L. and Tu, P.: Automatic Detection of Parallelism, *IEEE Parallel & Distributed Technology*, Vol. 2, No. 3, pp. 37–47 (1994).
9) 尾形航, 吉田明正, 合田憲人, 岡本雅巳, 笠原博徳: スタティックスケジューリングを用いたマルチプロセッサシステム上の無同期近細粒度並列処理, 情処論, Vol. 35, No. 4, pp. 522–531 (1994).
10) 本多弘樹, 岩田雅彦, 笠原博徳: Fortranプログラム粗粒度タスク間の並列性検出手法, 信学論, Vol. J73-D-I, No. 12, pp. 951–960 (1990).
11) 本多弘樹, 合田憲人, 岡本雅巳, 笠原博徳: Fortranプログラムの粗粒度タスクのOSCARにおける並列実行方式, 信学論, Vol. J75-D-I, No. 8, pp. 511–525 (1992).

12) 岡本雅巳, 合田憲人, 宮沢稔, 本多弘樹, 笠原博徳: OSCAR マルチグレインコンパイラにおける階層型マクロデータフロー処理, 情処論, Vol. 35, No. 4, pp. 513–521 (1994).
13) 吉田明正, 前田誠司, 尾形航, 笠原博徳: Fortran マルチグレイン並列処理におけるデータローカライゼーション手法, 情処論, Vol. 36, No. 7, pp. 1551–1559 (1995).
14) 笠原博徳, 合田憲人, 吉田明正, 岡本雅巳, 本多弘樹: 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, Vol. J75-D-I, No. 8, pp. 526–535 (1992).
15) Yoshida, A. and Kasahara, H.: Data Localization Using Loop Aligned Decomposition for Macro-Dataflow Processing, *Proc. of 9th Workshop on Languages and Compilers for Parallel Computers* (1996).
16) 合田憲人, 岩崎清, 岡本雅巳, 笠原博徳, 成田誠之助: 共有メモリ型マルチプロセッサシステム上での Fortran 粗粒度タスク並列処理の性能評価, 情処論, Vol. 37, No. 3, pp. 418–429 (1996).
17) 笠原博徳, 成田誠之助, 橋本親: OSCAR のアーキテクチャ, 信学論, Vol. J71-D, No. 8, pp. 1440–1445 (1988).

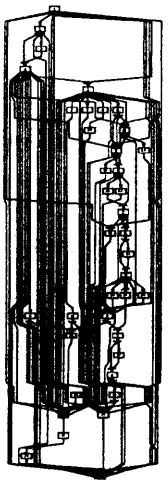

図 7 メインルーチンのマクロタスクグラフ (第 1 階層マクロタスクグラフ)

図 8 サブルーチン PSET のマクロタスクグラフ (第 2 階層マクロタスクグラフ)

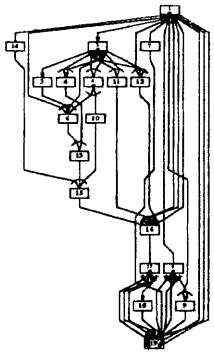

図 9 サブルーチン RUN のマクロタスクグラフ (第 3 階層マクロタスクグラフ)

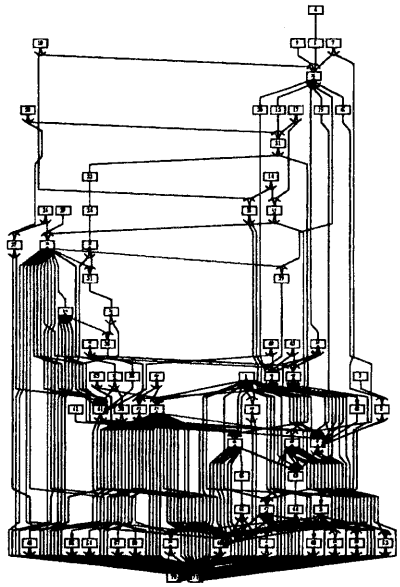

図 10 図 9 の MT16 内部より生成されたマクロタスクグラフ (第 4 階層マクロタスクグラフ)

図 11 図 10 の MT5 内側より生成されたマクロタスクグラフ (第 5 階層マクロタスクグラフ)

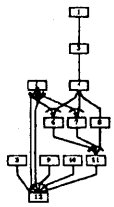

図 12 サブルーチン DCTDY のマクロタスクグラフ (第 6 階層マクロタスクグラフ)