マルチグレイン並列化FORTRANコンパイラ

岡本雅巳[†] 小幡元樹[††] 松井巖徹[†††]
松崎秀則[†] 笠原博徳[††] 成田誠之助[††]

本論文では今後のシングルチップマルチプロセッサからスーパーコンピュータまでの幅広いマルチプロセッサシステムで，システムの実効性能および使いやすさの向上のために有用なFORTRANマルチグレイン並列化コンパイラについて述べる．マルチグレイン並列化は従来のマルチプロセッサ用自動並列化コンパイラで用いられていたループ並列化に加え，サブルーチン，ループ，基本ブロック間粗粒度並列化，ステートメント/命令間（近）細粒度並列化を階層的に適用することによりプログラム全域の並列処理を可能とする．

Multi-grain Parallelizing FORTRAN Compiler

MASAMI OKAMOTO,[†] MOTOKI OBATA,[††] GANTETSU MATSUI,[†††]
HIDENORI MATSUZAKI,[†] HIRONORI KASAHARA[††]
and SEINOSUKE NARITA[††]

This paper describes a FORTRAN multi-grain parallelizing compiler. The multi-grain parallelizing compiler improves effective performance and ease of use of multiprocessor systems from single-chip multiprocessors to supercomputers. Multi-grain parallelizing scheme realizes effective parallel processing over the whole program by hierarchically appling coarse grain parallelization among subroutines, loops and basic blocks, and fine grain parallelization among statements or instructions in addition to conventional loop parallelization.

1. はじめに

21世紀初頭に主要なアーキテクチャとなると予想されるシングルチップのマルチプロセッサから，数千台規模のプロセッサを用いた数十TFLOPSを目指すスーパーコンピュータに至るまで，マルチプロセッサシステムの実効性能および使いやすさの向上を目指すためには自動並列化コンパイラが重要である．しかし，現在の商用コンパイラで自動並列化が適用可能なのはループだけに限定されており，ループの形状が複雑であると並列化できない場合も多い．これを改善するためにイリノイ大Polaris[1)~3)]，Parafrase-2[4)]，スタンフォード大SUIF[5)~7)]グループなどはランダム依存解析[8)~10)]，シンボリック解析[3),5),11)]，レンジテスト[11)]，配列プライベート化[1),3)]，ユニモジュラ変換[12)]，インタープロシージャ解析[3),7)]，データローカリティ最適化[3),5)]などを用い，より多くのループの効率的な並列化の研究を行っている．

しかしこれらの手法を駆使しても，複雑なループキャリド依存やループ外部への条件分岐のために並列化できないシーケンシャルループは依然存在する．またループの回転数がプロセッサ数に対して小さいプログラムなども存在する．一方ではループ外部の並列性，たとえば基本ブロック内部の細粒度並列性や，基本ブロック・サブルーチン・ループ間の粗粒度並列性も，いまだに有効利用できないという問題がある．

筆者らはこれらの問題を解決するために，基本ブロック内のステートメント間の近細粒度並列性の並列処理手法[13),14)]や，基本ブロック・サブルーチン・ループ間の粗粒度並列性の自動抽出手法および並列処理手法[15),16)]を提案してきた．さらに，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を提案している．

本論文ではマルチグレイン並列処理手法を実装した

† 株式会社東芝
TOSHIBA Corporation
†† 早稲田大学理工学部
School of Science and Engineering, Waseda University
††† 松下電器産業株式会社
Matsushita Electric Industrial

FORTRANコンパイラのコンパイレーション手法について述べる．また，実アプリケーションを用いた性能評価についても述べる．

2. マルチグレイン並列処理手法

本章では，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ト方式について述べる．

2.1 マクロデータフロー処理手法

マクロデータフロー処理手法では生成された粗粒度タスク〔MT: Macro Task（マクロタスク）〕は，1プロセッサエレメント（PE: Processor Element）以上からなるプロセッサクラスタ（PC: Processor Cluster）に割り当てられる．

2.1.1 マクロタスク生成

本コンパイラにおけるマクロデータフロー処理手法では以下の3種類のマクロタスクを生成する．

- BPA（Block of Pseudo Assignments）
  単一の基本ブロック，スケジューリングおよびデータ転送オーバヘッドを考慮し複数の小基本ブロックを融合したブロック，または並列性向上のため単一の基本ブロックを分割して得られるブロックより生成されるMT．
- RB（Repetition Block）
  DOループまたはIF文による分岐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[17]より生成されるMT．
- SB（Subroutine Block）
  コード長などを考慮してインライン展開が有効に適用できないサブルーチンをMTとして生成したもの．

BPA内部では1ステートメントまたは数ステートメントからなるタスクを用いた近細粒度並列処理手法が適用される．RB内部ではDOALLによる中粒度並列処理，あるいはループボディにおける近細粒度並列処理またはマクロデータフロー処理の各手法のうち有効な手法が適用され，階層的に並列処理される．SB内部ではサブルーチンがサブMTに分割され，階層的にマクロデータフロー処理が適用される．マクロデータフロー処理が階層的に適用される場合はMT内部で生成されるサブMTは，親階層MTが割り当てられるプロセッサクラスタ（PC: Processor Cluster）を分割して得られるサブPCに割り当てられる．マクロデータフロー処理を適用する階層の深さはプログラム，サブルーチン，ループの形状や大きさを考慮して，コンパイラが決定できる．MTは図1のように階層的に定義することが可能である．また，PCもその並列性を考慮し，図2のように階層的に定義することが可能である．

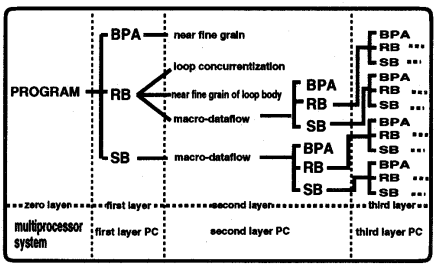


図1 マクロタスクの階層的な定義
Fig. 1 The hierarchical definition of macrotask.

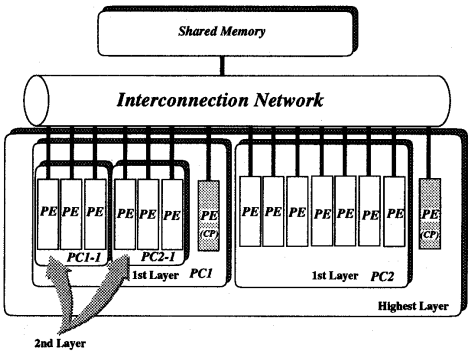


図2 プロセッサクラスタの階層的な定義
Fig. 2 The hierarchical definition of processor cluster.

2.1.2 マクロフローグラフ生成

MTの生成後，(サブ) MT間の制御フローやデータ依存が解析され，その結果はマクロフローグラフと呼ばれるDAG（Directed Acyclic Graph）[12),17)]で表現される．マクロフローグラフは次項で述べる最早実行可能条件解析の際に使用される．最早実行可能条件解析のアルゴリズムはマクロフローグラフがDAGであるという特徴を使用している．しかし，RB内部でマクロデータフロー処理手法を適用する場合，RB内部のサブマクロフローグラフの出口ノードから入口ノードに至る後方制御フローエッジが存在するため，そのバックエッジを除去するための変換が必要となる．マルチグレインコンパイラではこのマクロフローグラフ変換として，Repeat-MTというダミーノードを配置し，バックエッジの始点ノードからRepeat-MTへの制御フローエッジを生成し，バックエッジを除去する．さらに，Exit-MTというダミーノードを配置し，

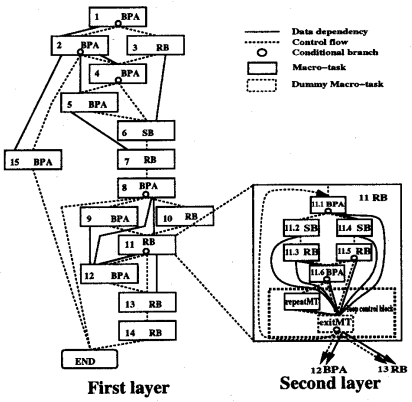


図3 サブマクロフローグラフ
Fig. 3 Sub-macro flowgraph.

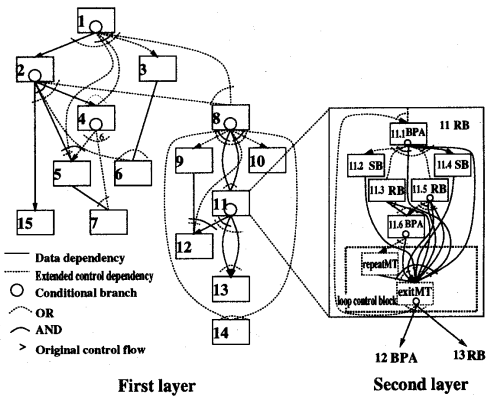


図4 マクロタスクグラフ
Fig. 4 Macrotask graph.

全ノードから Exit-MT へ至るデータ依存エッジを導入する．次イタレーションの実行確定は Repeat-MT の実行可能性と等価となり，実行中イタレーションの終了はこの Exit-MT が実行可能性と等価となる．以上の変換を行った結果，得られるマクロフローグラフを**図3**に示す．図3中の第2階層マクロフローグラフでは，*exitMT* には第2階層のすべての MT からのデータ依存エッジが入力されており，*exitMT* の実行可能条件としては，11.1_*BPA* が 11.2_*SB* に分岐した場合は分岐方向のすべてのマクロタスク，つまり 11.2_*SB*，11.3_*RB* および 11.6_*BPA* のすべてが終了することが必要条件であり，11.1_*BPA* が 11.4_*SB* に分岐した場合は分岐方向のすべてのマクロタスク，つまり 11.4_*SB*，11.5_*RB* および 11.6_*BPA* のすべてが終了することが必要条件である．すなわち *exitMT* の実行可能条件はこのループのイタレーションの終了条件となる．また，*repeatMT* が実行可能となった場合，イタレーション終了後，再びこのマクロフローグラフ内をスケジューリングすることによってループの制御を行うことができる．

2.1.3 マクロタスクグラフ生成

各 MT の最早実行可能条件はマクロフローグラフを入力として解析される．この最早実行可能条件は制御依存，データ依存を同時に考慮した MT 間の最大の並列性を表現している[15]．Girkar と Polychronopoulos は，筆者らの提案している最早実行可能条件解析アルゴリズムを基に，条件分岐の実行が必ず MT の最後にあると仮定して最早実行可能条件を求めるアルゴリズムを提案している[18]．全 MT の最早実行可能条件は，**図4**に示すようなマクロタスクグラフと呼ばれるデータ構造で表現される．図4において各ノードは MT を表し，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エッジを表している．ここで述べた拡張された制御依存エッジは，一般的な制御依存先行ノードとの関係を表すほか，先行ノードが実行されないことを確定する条件分岐を含む依存エッジである．各矢印はオリジナルの分岐方向を表している．ノードの入口や出口付近でエッジと交差する各実線弧はその弧に交差する全エッジが同時に成立する，つまり 'AND' 関係にあることを表している．ノードの入口や出口付近でエッジと交差する各点線弧はその弧に交差する全エッジのいずれか1つが成立する，つまり 'OR' 関係にあることを表している．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．たとえば，図4左の *SB*6 の最早実行可能条件は

[(*BPA*2 から *BPA*4 へ分岐が確定する)
OR
(*RB*3 が終了する)]

であることが分かる．

2.1.4 マクロタスクのスケジューリング手法

本コンパイラでは実行コードの形状や使用可能プロセッサ数に応じた効率の良い並列処理を行うため，PC 数や PC 内の PE 数を変更した場合も最適なスケジューリング結果を得られるよう，コンパイルし直す方法を採用している．コンパイル時に各階層でスタティックスケジューリングとダイナミックスケジューリングのいずれを適用するかが選択される．

実行時不確定性，つまり条件分岐などが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[19]を適用することが可能である．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MTをPC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ヘッドを削減するため，コンパイラが各階層の専用のダイナミックスケジューリングルーチンを生成する．ダイナミックスケジューリングルーチンはMTの終了や分岐方向決定の通知に応じて，タスクキューやMT実行管理テーブルを操作し，各MTの最早実行可能条件を検査する．MTが実行可能であればタスクキューにMTが投入される．タスクキュー内の優先順位に従ってソートされ，優先順位の高いMTがアイドル状態のPCに割り当てられる．スケジューリングアルゴリズムとしてはDynamic-CP（Critical Path）法と呼ばれるものを採用した．Dynamic-CP法では出口ノードからのクリティカルパス長に基づく優先順位に従いMTがPCに割り当てられる[15),16)]．

また，本コンパイラにおけるダイナミックスケジューリング手法には，集中スケジューラ方式と分散スケジューラ方式がある[20)]．

PCに割り当てられたMTは複数のサブMTに分割され，PCを分割した複数のサブPC上での階層的なマクロデータフロー処理を適用する．本コンパイラではPC内のサブPCを使用するマクロデータフロー処理を適用する場合においても，上位階層のMTがダイナミックスケジューリング，スタティックスケジューリングのいずれの手法によってPCに割り当てられたかにかかわらず，そのMT内のサブMT間に実行時不確定性が存在するかどうかによってダイナミックスケジューリングとスタティックスケジューリングの選択を行うことができる．

ダイナミックスケジューリングを選択した場合，集中スケジューラ方式，分散スケジューラ方式のいずれの場合も，各階層ごとに固有のダイナミックスケジューリングルーチンが生成される．つまり，生成されたダイナミックスケジューリングルーチンが特定の階層内のスケジューリングのみを行うことにより，MT数が爆発することを抑制できる．

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手法が適用される．この近細粒度並列処理手法では単一ステートメント，あるいは数ステートメントからなるタスクが生成される．近細粒度タスク間には条件分岐は存在せず，タスクのサイズが小さく構造も単純なためそのコスト計算が容易であり．スタティックスケジューリング手法を効果的に適用することができる．

マルチグレインコンパイラは，データ転送オーバヘッドを考慮したヒューリスティックアルゴリズムとして，CP/DT/MISF（Critical Path/Data Transfer/Most Immediate Successors First）法[13),14),19)]，DT/CP（Data Transfer/Critical Path）法[19)]，およびETF/CP（Earliest Task First/Critical Path）法[21)]の中で最も良好なスケジュールが得られるスタティックスケジューリングアルゴリズムを選択する．

2.3 ループ並列化手法

MTがDOALL可能なループで構成される場合，同数のイタレーションが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各プロセッサに割り当てられる．DOALLループの外側階層で，マクロデータフロー処理が適用される場合，マルチグレインコンパイラはMT間の制御依存，データ依存あるいはマクロタスクのコストなどを考慮してDOALLループを複数に分割し，マクロデータフロー処理の効果を向上させることが可能である．この手法を用いることによって，DOALL内部の並列性を犠牲にすることなく，マクロデータフロー処理を適用することが可能となる．

3. コンパイラの概要

本コンパイラは，シーケンシャルなFORTRANプログラムのプログラム全域にわたる並列性の自動的抽出手法およびスケジューリング手法の研究開発と，提案手法の有効性実証を目的として開発された．提案手法の検証用のプラットホームという使用目的を考慮し，本コンパイラを3つのフェーズに分解し，今回はそれぞれを独立したプログラムとした．各フェーズは以下の3種類である．

- FE（Front End）
  FEはFORTRAN77のパーザでシーケンシャルなFORTRANソースプログラムをシーケンシャルな中間言語に変換し出力する．
- MP（Middle Path）
  MPはシーケンシャルな中間言語を入力し，制御フロー解析，データ依存解析を行う．この結果に基づいてプログラムリストラクチャリング，タスク生成，タスク間の並列性抽出を行い，並列化された中間言語を出力する．
- BE（Back End）
  BEは並列化された中間言語を入力とし，ターゲットマシン用のネイティブコードまた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出力する．

FEはターゲットアーキテクチャや並列処理手法に依存しない手続きのみから構成されるため，ターゲットアーキテクチャの変更や，新たな並列処理手法の導

入時の修正が不要である．最適化や並列化の大半の処理はMPで行われる．MPはタスクのコスト算出などの一部の処理を除き，可能な限りターゲットアーキテクチャに依存しないよう設計されている．BEは最適なマシンコード生成が可能なように各ターゲットアーキテクチャごとに独立したプログラムとして提供する．現在，早稲田大学のマルチプロセッサシステムOSCAR用のバックエンドはすでに開発されており，富士通のVPPシリーズ，KSR1，NEC Cenju-3用のバックエンドを開発中である．

4. 中間言語構造

本マルチグレインコンパイラの中間言語はプログラムのデータ依存，制御依存などの解析結果データの管理の容易性，リストラクチャリングの容易性を考慮し，サブプログラム/マクロタスク/ステートメント/命令語などを階層的に表現したデータ構造とした．同じ階層で隣接するマクロタスクどうしやステートメントどうし，あるいはマクロタスクとその内部のサブマクロタスクやステートメントなどの隣接するデータ構造間やデータ構造の階層間は双方向のポインタで相互結合される．これにより，マクロタスクやステートメントなどの検索・追加・削除や複製の生成・データ構造の移動などを容易にしている．

4.1 構成要素

コンパイラ内部では以下に示すように，サブプログラム単位，ループ/基本ブロック/サブルーチンコールなどのマクロタスク単位，ステートメント単位，中間語命令単位に情報を管理するための構造体を定義している．

- **サブプログラム構造体**
  メインプログラム，サブルーチンサブプログラム，関数サブプログラム，BLOCK DATA単位ごとに生成されるデータ構造である．内部に後述の命令ブロック構造体や，変数テーブル，一時データテーブル，ジャンプラベル識別子テーブルなどのデータ構造へのポインタを持ち，サブプログラム内のプログラムやサブプログラムローカルな情報を一括して管理する．
  また，本構造体には呼出先のサブプログラム構造体のリストと，呼出元のサブプログラム構造体のリストへのポインタも備わっており，コールグラフのノードとして使用される．
- **命令ブロック構造体**
  ループや基本ブロック，単一のサブルーチンコールなどのマクロタスク単位に生成されるデータ構造である．命令ブロック構造体は内部に，前処理および本体ステートメント構造体のリスト，内側のサブブロックの命令ブロック構造体のリスト，後処理のステートメント構造体のリストへのポインタを持つ．基本ブロックやサブルーチンコールではサブブロックへのポインタをNULLポインタとし，マクロタスク内のステートメントは前処理ステートメントの構造体リストに付加される．また，複数の命令ブロック構造体のリストへのポインタを持つ命令ブロック構造体を生成することにより，複数のループ，基本ブロック，サブルーチンコールなどを融合したマクロタスクを表現することが可能である．
  命令ブロック構造体は内部にマクロタスク間の制御フローやデータフロー・データ依存情報あるいは最早実行可能条件式の各データ構造を持っており，MPは内部で解析したマクロタスク間依存解析結果や並列性などのあらゆる情報を本データ構造上に構築する．
- **ステートメント構造体**
  FORTRANプログラムソース中のステートメント単位に生成されるデータ構造．本構造体には，後述の中間語命令構造体のリストへのポインタが備わっている．近細粒度並列処理手法では本データ構造を単一の近細粒度タスクとして使用する．
- **中間語命令構造体**
  中間語命令構造体は，四つ組ライクな命令語を表現する．オペランドには変数，定数，ジャンプラベル識別子，マクロタスク識別子，サブルーチン識別子，同期リソース識別子，関数種別などを持つことが可能である．

4.2 ミドルパスによる中間言語操作

本コンパイラでは，フロントエンド（FE）により変換されたソースプログラムは中間言語ファイルとして一時的に保存され，ミドルパス（MP）に渡される．MPで，中間言語は最適化・並列化され出力される．MPにおける最適化・並列化は**図5**に示す各フェーズにより行われる．図5に示す最適化フェーズは主に並列化フェーズの準備段階で並列性を向上するための一般的な逐次最適化，および並列性向上のためのコードの変換を意味している．具体的には，データ依存解析を容易にする定数伝搬，データのプライベート化などもここで行われる．Illinois大の並列化コンパイラPolarisにおいても，これらの最適化手法の下でループの並列化を行い，高い並列処理性能を実現している[1)~3)]．

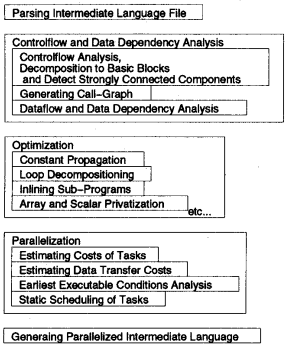


図 5 ミドルパスの内部処理
Fig. 5 Processing inside middle path.

```
@plain {                %% block structure includes all statements
 $call {                %% "call" subroutine statement
  !call (S2(A1,V1));    %% instruction(CALL S2(A1,V1)
 }
 $assign {              %% "assign" statement
  !assign (A1(V2),C1); %% instruction(A1(V2)=C1
 }
 $jf {                  %% "jmp-if-false" statement
  !eq (T1,V1,C2);       %% instruction(T1 = V1.EQ.C2)
  !jf (L1,T1);          %% instruction(IF .NOT.T1 GO TO !label(L2))
 }
 $jmp {                 %% "jmp" statement
  !jmp (L2);            %% instruction(GO TO !label(L2))
 }
 $label {               %% "label" pseudo statement
  !label (L1);          %% instruction(label)
 }
 $call {                %% "call" subroutine statement
  !call (S2(A1,V1));    %% instruction(CALL S2(A1,V1))
 }
 $label {               %% "label" pseudo statement
  !label (L2);          %% instruction(label)
 }
 $return{               %% "return" statement
  !return ();           %% instruction(RETURN)
 }
}
```

図 6 FE の出力例
Fig. 6 Intermediate language file generated from FE.

出力形式は MP の入力形式と上位互換性のある中間言語ファイル形式である．この中間言語ファイルは MP 中のあらゆるフェーズで出力可能であり，また MP は中間言語ファイルパーザを持っているが，MP 自身が出力する中間言語ファイルはすべて解析可能であり，中間言語データ構造を構築することが可能となっている．この仕組みにより，MP は自身の解析結果をフィードバックし，高度な最適化・並列化を実現可能にしている．たとえば，MP の処理中に解析された MT 間のデータ転送コストや MT の処理コストなどの情報をフィードバックし，マクロタスクの再分割を行ったり，複数 MT の融合 MT を生成したりすることも可能である[22),23)]．

FE により出力される中間言語では各サブプログラム構造体ごとに唯一の命令ブロック構造体が生成される．この命令ブロック構造体は，サブプログラム内の全ステートメント構造体のリストを保有する．**図 6** に FE の出力形式の中間言語ファイル例を示す．図 6 の **@plain{...}** はプログラム全体からなるブロックであり，FE の出力においてのみ使用される．**$call{...}** や **$assign{...}** など，'$' 文字で始まる記号はソースプログラムのステートメントに対応する．また，**!eq (...)** や **!jf (...)** など，'!' 文字で始まる記号は中間言語の命令語である．

```
@block1(block){ %% block-structure includes all sub-blocks
 @sb1(block){    %% SB(Subroutine Block) block-structure
  $call{
   !call(S2(A1,V1));
  }
 } % @sb1
 @bb2(block){    %% BB(Basic Block) block-structure
  $assign{
   !assign(A1(V2),C1);
  }
  $jf{
   !eq(T1,V1,C2);
   !jf(L1,T1);
  }
  $jmp{
   !jmp(L2);
  }
 } % @bb2
 @sb3(block){    %% SB(Subroutine Block) block-structure
  $label{
   !label(L1);
  }
  $call{
   !call(S2(A1,V1));
  }
 } % @sb3
 @bb4(block){    %% BB(Basic Block) block-structure
  $label{
   !label(L2);
  }
  $return{
   !return();
  }
 } % @bb4
} % @block1
```

図 7 マクロタスク分割後の MP 出力例
Fig. 7 Intermediate language file generated after decomposing MT.

MP の内部では FE から入力されたステートメント構造体のリストが基本ブロックに分割され，強連結部分[12),17)]の検出，サブルーチンコール検出などが行われた後，マクロタスクが生成される．マクロタスク分割後に出力された中間言語ファイルを，MP の中間言語パーザに入力した場合は，MP の中間言語パーザを通した時点でマクロタスク分割済みの中間言語データ構造となり，その後の不要な解析は省略することができる．マクロタスク分割後に出力された中間言語ファイルを**図 7** に示す．図 6 中の **@plain{...}** は，図 7 では **@block1(block){...}** に置き換わり，さらにその内側は 1〜数ステートメントからなる **@sb1(block){...}** や **@bb2(block){...}** などのブロックに分割される．このブロックがすなわちマクロタスクである．中間言語ファイルパーザは図 6 や図 7 に示される **@plain{...}** や **@block1(block){...}**, **@sb1(block){...}** などの構文解析の結果より，4.1 節で述べた命令ブロック構造体を構築し，**$call{...}** や **$assign{...}** などの構文解析の結果よりステートメント構造体を構築し，そして **!eq (...)** や **!jf (...)** などの構文解析の結果より中間語命令構造体を構築する．

```
$jf{
 !eq(T1,V1,C2);%% T1 = V1.EQ.C2
 !jf(L1, T1);  %% IF .NOT.T1 GO TO !label(L1)
$jmp{
 !jmp(L2);     %% GO TO !label(L2)
}
   :
   :
$label{
 !label(L2);   %% jump destination
}
   :
   :
$label{
 !label(L1);   %% jump destination
}
   :
```

図8 分岐命令とラベル
Fig. 8 Jmp statement and label pseudo statement.

4.3 マルチグレイン並列化用の制御フロー情報の表現

基本ブロック・ループ・サブルーチンコール間の制御フローは中間言語ファイル上では分岐命令の形で表現される．MPは分岐命令より，各マクロタスクの制御フロー先行ノード/後続ノードを解析し，先行ノードリスト・後続ノードリストを命令ブロック構造体内部に生成する．

本コンパイラの中間言語ではジャンプラベル命令と呼ばれる疑似命令を導入している（図8）．この疑似命令は単一のラベル識別子をオペランドとして持つ，一方，分岐命令のオペランドにもラベル識別子が存在し，このジャンプラベル疑似命令とジャンプ命令のラベル識別子が一致する場合，この分岐命令の分岐先がこのジャンプラベル疑似命令であることを表している．たとえば，図8のジャンプ命令`!jmp(L2);`はラベル疑似命令`!label(L2);`の位置に分岐し，条件ジャンプ命令`!jf(L1,T1);`は条件（論理値）`T1`が不成立のときにラベル疑似命令`!label(L1);`に分岐する．ここで述べたラベル疑似命令への分岐命令は一般的な用途で使用され，BEではジャンプラベル命令は分岐先アドレスの計算に使用され，分岐命令はその分岐先アドレスへの分岐命令コードに変換される．

本コンパイラのDOループでは後方分岐は明示せず，DOループブロック構造体の前処理ステートメントリスト中に図9のように`!do`命令という中間言語命令を挿入し，内部のループボディ出口から暗黙的に`!do`命令にジャンプする．コンパイラが並列化や最適化に費やすコストを削減するため，このように省略可能な命令は排除し，ループボディをシンプルにすることでループボディの各PE用の複製の生成などの処理を容易にするよう考慮した．

マルチグレイン並列化コンパイラではマクロタスク間のデータ依存・制御依存を考慮した粗粒度並列性を抽出する．ここではマクロタスク間の並列性を有効に利用するため，分岐方向の確定を可能な限り早期のタイミングでダイナミックスケジューリングルーチンが

```
@loop1(block){
  $do{                    %% do-statement
   !do(V2,C1,V1,C1);       %%  DO V2=C1,V1,C1
  }
 @bb1(block){              %% a basic block of a loop body
   $assign{                  %% assign-statement
    !add(T1,A1(V2),C1);
    !add(T2,V2,C1);
    !assign(A1(T2),T1);   %% A1(V2+C1) = A1(V2) + 1
   }
 } %% NO JUMP STATEMENT AT BOTTOM OF THE DO-LOOP
}
```

図9 DOループの制御命令
Fig. 9 Do statement.

```
@loop1(mt5){ %% MT5
      :
      :
  $mjmp{      %% "jump-toward-MT" statement
   !jmp(M6); %% this MT(MT5) branches to MT6
  }
      :
      :
}
@loop2(mt6){ %% MT6
    :
    :
}
```

図10 MT間の分岐
Fig. 10 Jmp statement among MTs.

検知するようにしている．これにより，相互間に制御依存が存在しデータ依存が存在しない場合には，先行マクロタスクの終了を待たずに後続マクロタスクを実行可能状態とすることができる．これらのマクロタスク間にデータ依存が存在する場合でも分岐方向を早期に検知することで，分岐先の確定から後続マクロタスク実行開始までの間を利用し必要なデータを転送する[19]ことも可能である．

このようにマクロタスク間の分岐では，分岐方向の確定を早期にダイナミックスケジューリングルーチンに通知する手段が必要である．そこで，並列性が抽出されたマクロタスク間分岐命令表現は，上述のラベルへの分岐命令の内部表現とは明確に区別している．

並列性抽出後のマクロタスク間分岐では，図10中の`!jmp(M6);`の`M6`のようにマクロタスク番号を識別子とし，分岐先のマクロタスクを特定する分岐命令を使用する．図10で示した`!jmp(M6);`は，後続マクロタスク`@loop2(mt6){...}`への分岐を表現している．この分岐命令は，処理される時点で実際に制御を分岐先に移すような命令ではなく，この命令の通過時点で分岐方向が確定することを表している．したがって，この命令が挿入されるのはマクロタスクの末尾である必要はなく，分岐方向を確定させる評価式直後に挿入することが可能である．BEでは，このマクロタスク間分岐命令はダイナミックスケジューラへの分岐方向通知処理に変換される．

5. OSCARのアーキテクチャ

今回の評価では，マルチグレイン並列処理を支援するアーキテクチャ[24]の1つである図11に示すマル

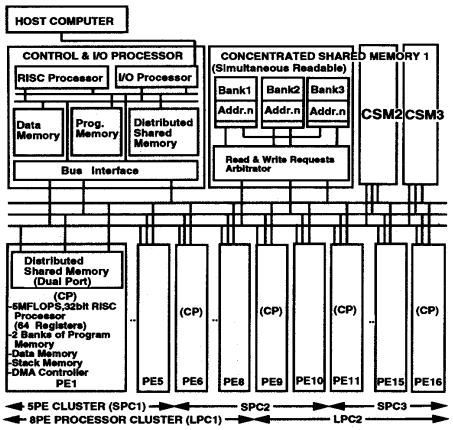


図 11 OSCAR のアーキテクチャ
Fig. 11 OSCAR's architecture.

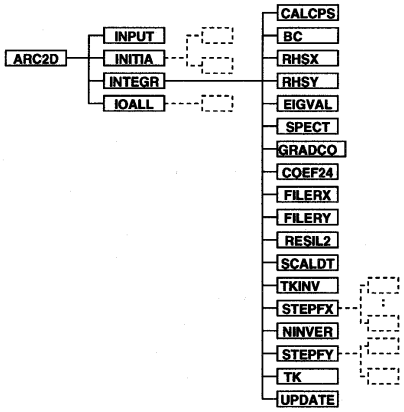


図 12 ARC2D のコールグラフ
Fig. 12 The call graph on main program of ARC2D.

チプロセッサシステム OSCAR を性能評価対象アーキテクチャとして用いる．OSCAR は早稲田大学で開発された共有メモリマルチプロセッサシステムで，16 台の PE とコントロール I/O プロセッサ（CIOP）が 3 バンクの集中共有メモリ（CSM）と 3 本のバスを介して結合されている．各 PE 上にはカスタムメードの 32 bit RISC プロセッサ，64 個の 32 bit 汎用レジスタ，整数演算装置，浮動小数点演算装置，2 k ワードのデュアルポートメモリを使用した分散共有メモリ（DSM），256 k ワードのローカルデータメモリ（LDM），2 バンクのプログラムメモリ（PM），4 k ワードのスタックメモリ（SM），DMA コントローラがそれぞれ装備されている．

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 1 PE 対 1 PE のデータ転送．
- 分散共有メモリを介した 1 PE 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 1 PE 対複数 PE のデータ転送．

各 PE 上の DSM へはその PE 自身と他の 1 PE からの同時アクセスが可能である．各 CSM の同一または異なるアドレスに対しては 3 台以内の PE から同時読み出し可能である．各 PE は PE 内部の LDM や DSM に対して 1 クロックで読み書き可能である．CSM や他 PE 上の DSM に対しては最短で 4 クロックでの読み書きが可能である．

OSCAR では全 PE を複数のグループに分割し共有メモリ型の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．この場合，実行時であっても PC 数，PC 内の PE 数は変更可能である．3 本のバスにはバリアコントロールラインが備わっており，PC 内でこのハードウェアを使用することにより，数クロックで PE 間のバリア同期を行うことが可能である．

OSCAR のプロセッサでは全インストラクションが固定クロックで実行可能であるため，正確にスケジュールされた並列マシンコードを生成することが可能である[14]．

6. 性能評価

本章ではマルチグレイン並列処理手法のマルチプロセッサシステム OSCAR 上での性能評価について述べる．

6.1 実アプリケーションを用いた性能評価

本節では，Perfect Club ベンチマークに収録されている ARC2D を使用した性能評価について述べる．ARC2D は，有限差分法による明解法の二次元流体問題でオイラー方程式による，ロバストで汎用目的の数値計算プログラムである．ソースプログラムは約 4000 行の FORTRAN77 のコードである．

ARC2D のコールグラフを図 **12** に示す．メインプログラムからは **INPUT**，**INITIA**，**INTEGR** および **IOALL** の 4 つのサブルーチンが呼ばれる．単一プロセッサを使用した場合では 90 パーセント以上の実行時間が **INTEGR** で費やされる．サブルーチン **INTEGR** 内部の最適化前のマクロフローグラフを図 **13** に示す．MT21 は出口ノード（Exit-MT）である．MT21 以外のすべての MT は SB，すなわちサブルーチンである．

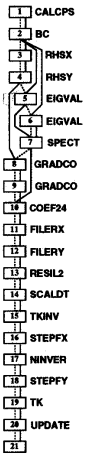


図 13 サブルーチン INTEGR 内のオリジナルのマクロフローグラフ
Fig. 13 The original sub-macro-flowgraph inside INTEGR of ARC2D.

図 13 のマクロフローグラフ中の MT3 と MT9 の間に並列性が存在する．しかしこれらの MT のコストは比較的小さく，この部分のみの並列処理では効果がほとんど期待できない．一方 MT11（サブルーチン **FILERX**），MT12（サブルーチン **FILERY**），MT16（サブルーチン **STEPFX**），MT18（サブルーチン **STEPFY**）のコストが大きい．特に MT11 や MT16 の実行コストは他の MT に比べかなり大きい．しかし，これらの MT 間には並列性が存在しない．したがって，これらの MT 間で高い並列性を抽出するためには，おもに MT11，MT12，MT16，MT18 に対してインライン展開を適用する必要がある．

まずサブルーチン **FILERX** のオリジナルのサブマクロタスクグラフを**図 14** に示す．サブルーチン **FILERX** の 4 イタレーションの最外側ループにはループアンローリングが適用され，ループ制御変数の初期値，終値に関して定数伝搬が適用される．サブルーチン **FILERX** の最適化されたマクロタスクグラフを**図 15** に示す．

FILERY 内部に対しても FILERX と同様の最適化が行われる．サブルーチン **STEPFX** は **INTEGR** 内部で最も大きいサブルーチンである．サブルーチン **STEPFX** のオリジナルのサブマクロタスクグラフを**図 16** に示す．**STEPFX** は 1 つの基本ブロックと 3 イタレーションの最外側ループから構成される．このループは配列のリネーミングを適用することによって DOALL 可能となる．さらにループアンローリングを適用することによって得られる最適化されたマクロタスクグラフを**図 17** に示す．図 17 から，インタープロシージャ解析により MT8，MT9，MT17，MT18，MT26，MT27 の 6 つのサブルーチン間の並列性が得られていることが分かる．今回の評価では，インライン展開，ループアンローリング，リネーミング，およびスカラ/配列プライベート化の適用箇所をユーザがコンパイラディレクティブを用いて指定する方式をとった．

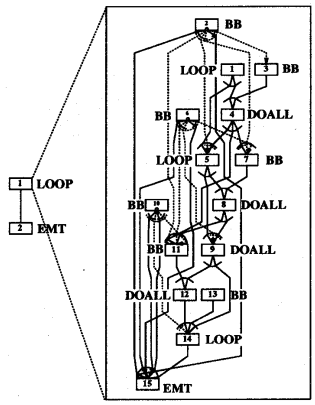


図 14 FILERX 内部のオリジナルのマクロタスクグラフ
Fig. 14 The original sub-macrotask graph inside FILERX.

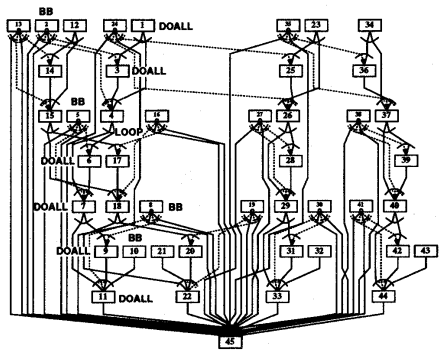


図 15 最適化された FILERX 内部のマクロタスクグラフ
Fig. 15 The optimized sub-macrotask graph inside FILERX.

サブルーチン **STEPFY** は単一の基本ブロックと，最外側ループから構成される．ループ内部には，**図 18** に示すように，それぞれが条件分岐を持った 3 つの BPA が存在する．各条件分岐は **STEPFY** の最外側ループの制御変数の値により分岐方向が確定する．したがって，ループアンローリングを適用することにより，これらの条件分岐はすべて削除することができる．さらに，変数伝搬やインタープロシージャ解析を適用

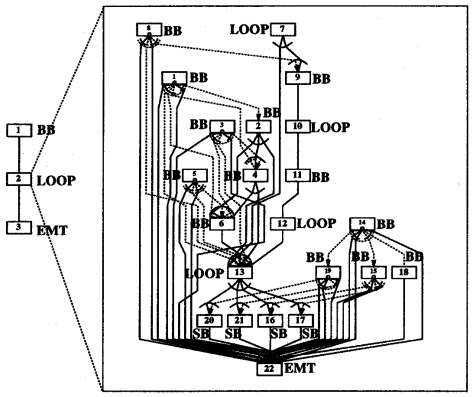


図 16 STEPFX 内部のオリジナルのマクロタスクグラフ
Fig. 16 The original sub-macrotask graph inside STEPFX.

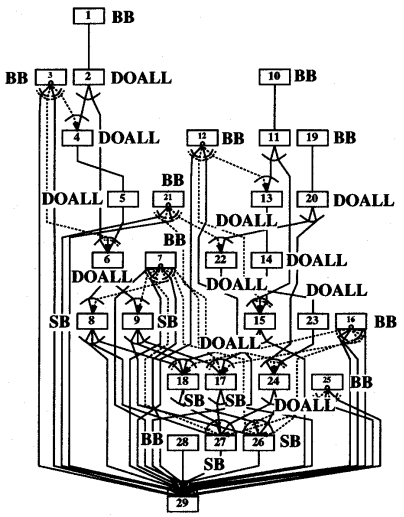


図 17 最適化された STEPFX 内部のマクロタスクグラフ
Fig. 17 The optimized sub-macrotask graph inside STEPFX.

し，得られる最適化されたマクロタスクグラフは図 **19** のようになる．

INTEGR より呼ばれるサブルーチンで **BC** と **GRADCO** を除くすべてのものは **INTEGR** 上にインライン展開する．さらに配列プライベート化と変数伝搬を適用し，得られる **INTEGR** 内のマクロタスクグラフは図 **20** に示すようになる．

表 1 はマルチプロセッサシステム OSCAR 上で ARC2D を実行した場合の実行時間である．表 1 の “1 PE” を除くすべての実行で，インライン展開，配列・スカラプライベート化，定数伝搬など最適化が適

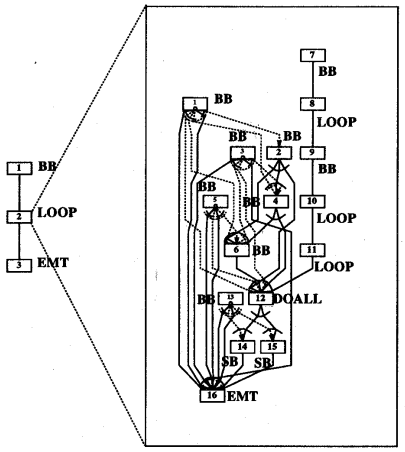


図 18 STEPFY 内部のオリジナルのマクロタスクグラフ
Fig. 18 The original sub-macrotask graph inside STEPFY.

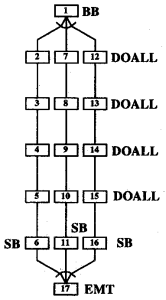


図 19 最適化された STEPFX 内部のマクロタスクグラフ
Fig. 19 The optimized sub-macrotask graph inside STEPFY.

用されている．表 1 の中で複数 PC を使用した値は，**INTEGR** 内部でマクロデータフロー処理を適用した場合の実行時間である．その他のサブルーチンは全 PE から構成される単一 PC で実行した．スピードアップは単一 PE を使用した場合の実行時間を基準に計算している．

表のように，8 PE を使用してループ並列化のみを適用した場合は 1 PE の 5.81 倍のスピードアップが得られた．これに対し 4 PE からなる PC を 2 つ使用してマルチグレイン並列処理を適用した場合は 6.81 倍のスピードアップが得られた．つまり，8 プロセッサを使用した場合では，マルチグレイン並列処理手法はループ並列化の場合より 17 パーセントのスピードアップが得られた．また，15 PE を使用してループ並列化のみを適用した場合は 7.47 倍のスピードアップが得

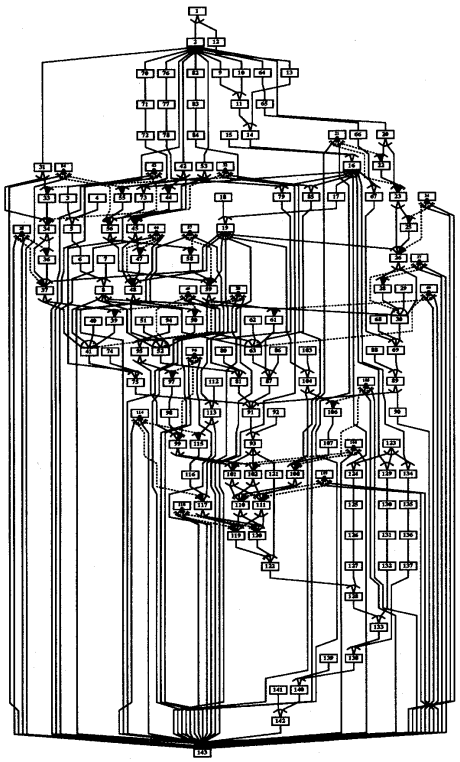


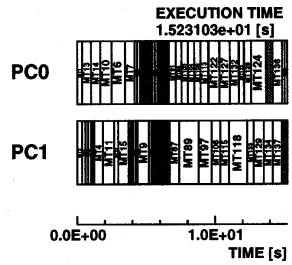
図20 最適化されたINTEGR内部のマクロタスクグラフ
Fig. 20 The optimized macrotask graph inside INTEGR.

表1 ARC2Dを使用したマルチグレイン並列処理の性能評価
Table 1 Performance evaluation of multi-grain parallel processing.

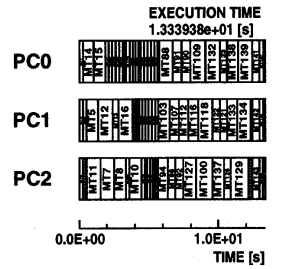
Num.PEs/PCs		Exec. Time [s]	Speedup
1PE		14802.31	1.00
4PE	1PC×4PE	3970.95	3.37
	2PC×2PE (MDF inside INTEGR)	3823.91	3.87
8PE	1PC×8PE	2546.27	5.81
	2PC×4PE (MDF inside INTEGR)	2172.30	6.81
15PE	1PC×15PE	1982.57	7.47
	3PC×5PE (MDF inside INTEGR)	1494.22	9.91

られた．5PEのPCを3つ使用してマルチグレイン並列処理を適用した場合は9.91倍のスピードアップが得られた．つまり，15プロセッサを使用した場合ではマルチグレイン並列処理はループ並列化の場合より33パーセントのスピードアップが得られた．

INTEGR内の1イタレーション分のGanttチャートを**図21**に示す．図21 (a)は2PC（各6PE）の場合，(b)は3PC（各5PE）の場合を示している．この図から3PCレベルで粗粒度並列性が有効に使われており，コンパイラが生成したスケジューリングルーチンを用いたダイナミックスケジューリングのオーバヘッドが非常に小さいことが分かる．



(a) Gannt chart for 2PCs (each PC has 6PEs)



(b) Gannt chart for 3PCs (each PC has 5PEs)

図21 INTEGRの1イタレーションのGanttチャート
Fig. 21 Gantt chart for a loop iteration inside a subroutine INTEGR.

7. おわりに

本論文では開発したOSCAR FORTRANマルチグレイン並列化コンパイラと実アプリケーションプログラムを用いた評価について述べた．マルチグレイン並列化手法は近細粒度並列処理手法とループ並列化手法とマクロデータフロー処理手法を効果的に組み合わせた手法であり，従来のループ並列化手法では利用できなかったループ外の並列性を効果的に抽出することを可能とする．これにより，今回のARC2Dのようにループ回転数がプロセッサ数に比べ少なくループ並列性が十分でないようなプログラムに対しても効果的な並列処理を行うことが可能となることが確かめられた．

現在，マルチグレイン並列処理手法におけるデータローカライゼーション[23]，データプレロード/ポストストア手法[19]を用いた自動最適化手法を開発している．今後それらの手法を駆使し，より多くの実アプリケーションプログラムを用いた性能評価を通してマルチグレイン並列化手法の性能向上を図る予定である．

最後に本研究の一部は通産省次世代情報処理基盤技

術開発事業並列分散コンピューティング技術分野マルチプロセッサコンピューティング領域の一環として行われた.

参考文献

1) 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).
2) Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lawrence, T., Lee, J., Padua, D., Paek, Y., Pottenger, B., Rauchwerger, L. and Tu, P.: Parallel Programming with Polaris, *IEEE Computer*, Vol.29, No.12, pp.78–82 (1996).
3) Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks (1997). *IEEE Trans. on Parallel and Distributed Systems.* (to appear)
4) Polychronopoulos, C., Girkar, M., Haghighat, M., Lee, C., Bruce, L. and Schouten, D.: Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessors, *Proc. Intl. Conf. Parallel Processing*, St. Charles IL, pp.39–48 (1989).
5) Amarasinghe, S., Anderson, J., Lam, M. and Tseng, C.: The SUIF Compiler for Scalable Parallel Machines, *Proc. 7th SIAM conference on parallel processing for scientific computing*, SIAM (1995).
6) Hall, M., Anderson, J., Amarasinghe, S., Murphy, B., Liao, S., Bugnion, E. and Lam, M.: Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer*, Vol.29, No.12, pp.84–89 (1996).
7) Hall, M., Amarasinghe, S., Murphy, B., Liao, S. and Lam, M.: Detecting Coarse-Grain Parallelism Using an Interprocedural Parallelizing Compiler, *Proc. Supercomputing '95* (1995).
8) 笠原：並列処理技術，コロナ社 (1991).
9) Pugh, W.: The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependency Analysis, *Proc. Supercomputing '91* (1991).
10) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic Pub. (1988).
11) 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).
12) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley (1996).
13) Kasahara, H., Honda, H. and Narita, S.: Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR, *IEEE ACM Supercomputing'90*, pp.856–864, IEEE (1990).
14) Ogata, W., Yoshida, A., Okamoto, M., Kimura, K. and Kasahara, H.: Near Fine Grain Parallel Processing without Explicit Synchronization on a Multiprocessor System, *Proc. 6th Workshop on Compilers for Parallel Computers* (1996).
15) Honda, H., Aida, K., Okamoto, M., Yoshida, A., Ogata, W. and Kasahara, H.: FORTRAN Macro-Dataflow Compiler, *Proc. 4th Internatioanl Workshop on Compilers for Parallel Computers*, pp.265–286 (1993).
16) 本多，合田，岡本，笠原：Fortranプログラム粗粒度タスクのOSCARにおける並列実行方式，信学論，Vol.J75-D-I, No.8, pp.511–525 (1992).
17) Aho, V. A., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley (1986).
18) Girkar, M. and Polychronopolous, C.: Optimization of Data/Control Conditions in Task Graphs, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).
19) 藤原，白鳥，鈴木，笠原：データプレロード及びポストストアを考慮したマルチプロセッサスケジューリングアルゴリズム，信学論，Vol.J75-D-I, No.8, pp.495–503 (1992).
20) 合田，岩崎，岡本，笠原，成田：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，情報処理学会論文誌，Vol.37, No.3, pp.418–429 (1996).
21) Hwang, J., Chow, Y., Anger, F. and Lee, C.: Scheduling precedence graphs in systems with interprocessor communication times, *SIAM J. Compute.*, Vol.18, No.2, pp.244–257 (1989).
22) 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，Vol.J75-D-I, No.8, pp.526–535 (1992).
23) Yoshida, A. and Kasahara, H.: Data Localization Using Loop Aligned Decomposition for Macro-Dataflow Processing, *Proc. 9th Workshop on Languages and Compilers for Parallel Computers* (1996).
24) 笠原，尾形，木村，小幡，飛田，稲石：マルチグレイン並列化コンパイラとそのアーキテクチャ支援，信学技報，Vol.CPSY98-10, pp.71–76 (1998).

(平成10年 8 月10日受付)
(平成11年10月 7 日採録)

岡本 雅巳（正会員）

平成2年早稲田大学理工学部電気工学科卒業．平成4年同大学大学院理工学研究科電気工学専攻修士課程修了．平成9年同大学大学院理工学研究科電気工学専攻博士後期課程単位取得退学．在学中に並列化コンパイラ，並列実行方式に関する研究を行う．同年（株）東芝入社．現在東芝府中電力システム工場勤務．発電監視制御システムの開発に従事．電子情報通信学会会員．

小幡 元樹

平成8年早稲田大学理工学部電気工学科卒業．平成10年同大学大学院理工学研究科電気工学専攻修士課程修了．同年同大学大学院理工学研究科電気工学専攻博士後期課程に進学．平成11年早稲田大学電気電子情報工学科助手，現在に至る．現在，マルチグレイン自動並列化コンパイラとデータローカライゼーションに関する研究に従事．

松井 巌徹

平成8年早稲田大学理工学部電気工学科卒業．平成10年同大学大学院理工学研究科電気工学専攻修士課程修了．同年松下電器産業（株）入社．現在松下電器本社マルチメディア開発センター画像・情報グループ所属．

松崎 秀則

平成8年早稲田大学理工学部電気工学科卒業．平成10年同大学大学院理工学研究科電気工学専攻修士課程修了．同年（株）東芝入社．現在東芝研究開発センター コンピュータ・ネットワークラボラトリー所属．

笠原 博徳（正会員）

昭和55年早稲田大学理工学部電気工学科卒業．昭和60年同大学大学院博士課程修了．工学博士．昭和58～60年同大学理工学部助手．昭和60年カリフォルニア大バークレー短期客員研究員，日本学術振興会第1回特別研究員．昭和61年早稲田大学理工学部電気工学科専任講師．昭和63年同助教授．平成9年同大学理工学部電気電子情報工学科教授，現在に至る．平成元年～2年イリノイ大学 Center for Supercomputing R & D 客員研究員．昭和62年 IFAC World Congress 第1回 Young Author Prize，平成9年情報処理学会坂井記念特別賞受賞．主な著書「並列処理技術」（コロナ社）．電子情報通信学会，電気学会，シミュレーション学会，ロボット学会，IEEE，ACM 等会員．

成田誠之助

昭和37年早稲田大学大学院修士課程修了，同年アメリカ・パデュー大学大学院留学（フルブライト留学生）．昭和38年早稲田大学理工学部助手，以後講師・助教授を経て，昭和48年教授，現在に至る．工学博士．分散計算機制御システム，並列処理，産業用ロボット制御，ディジタル制御理論，CIM 等の研究に従事．計測自動制御学会，ロボット学会，IEEE 各会員．