Vol. 44 No. 4 情報処理学会論文誌 Apr. 2003

マルチグレイン並列処理のための階層的並列性制御手法

小幡元樹[†,††] 白子準[†] 神長浩気[†]
石坂一久[†,††] 笠原博徳[†,††]

従来，チップマルチプロセッサから HPC まで幅広く使われている共有メモリ型マルチプロセッサシステム上での自動並列化コンパイラではループレベル並列処理が主に用いられてきたが，その並列化技術の成熟により，ループ並列化では今後大幅な性能向上は難しいと言われている．このループ並列性の限界を越えるために，現在ループ・サブルーチン・基本ブロック間の粗粒度タスク並列性，ステートメント間の近細粒度並列性を従来のループ並列処理に加えて利用するマルチグレイン並列処理が有望視されている．マルチグレイン並列処理において各種粒度の並列性を階層的に抽出し，効率よい並列実行を実現するためには，各々の階層（ネストレベル）の並列性に応じて，何台のプロセッサ，あるいはプロセッサのグループ（プロセッサクラスタ）を割り当てるかを決定する必要がある．本論文ではプログラム中の各階層の並列性を効果的に用いるための階層的並列性制御手法を提案し，本手法を実装した OSCAR マルチグレイン並列化コンパイラによる階層的並列処理の評価では，SMP サーバ IBM pSeries690 Regatta 16 プロセッサシステム上にて SPEC95FP ベンチマークを用いた結果，逐次処理に対して 1.9〜10.6 倍の性能向上が得られることが確かめられた．

Hierarchical Parallelism Control Scheme for Multigrain Parallelization

Motoki Obata,[†,††] Jun shirako,[†] Hiroki Kaminaga,[†]
Kazuhisa Ishizaka[†,††] and Hironori Kasahara [†,††]

A multigrain parallel processing is very important to improve effective performance beyond the limit of the loop parallelism on a shared memory multiprocessor system. In the multi-grain parallelization, coarse grain parallelism among loops, subroutines and basic blocks, and near fine grain parallelism among statements inside a basic block are exploited in addition to the conventional loop parallelism. In order to efficiently use hierarchical parallelism of each nest level, or layer, in multigrain parallel processing, it is required to determine how many processors or groups of processors should be assigned to each layer, according to the parallelism of the layer. This paper proposes a hierarchical parallelism control scheme for multigrain parallel processing so that the parallelism of each hierarchy can be used efficiently. Performance of the hierarchical parallelization using the proposed scheme implemented on OSCAR multigrain parallelizing compiler is evaluated on IBM pSeries690 Regatta SMP server with 16 processors using SPEC95FP benchmarks and the hierarchical parallelization using the proposed scheme gave us 1.9 to 10.6 times speed up against sequential processing.

1. はじめに

現在，共有メモリ型マルチプロセッサアーキテクチャは，チップマルチプロセッサからワークステーション，ミッドレンジサーバ，ハイパフォーマンスサーバに至る多くのシステムで採用されている．このような共有メモリ型マルチプロセッサ用自動並列化コンパイラでは，従来よりループレベル並列化技術[1),2)] が用いられ，様々なデータ依存解析技術，プログラムリストラクチャリング技術が開発されてきた．

例えば，Parafrase，Cedar Fortran コンパイラによってループ並列化技術[1)~3)] を発展させてきた Polaris[3)] は，イリノイ大学とパデュー大学で現在開発中であり，サブルーチンのインライン展開，シンボリック解析，アレイプライベタイゼーション，実行時データ依存解析を用いたループ並列処理を行っている．また，スタンフォード大学の SUIF[4)] は，インタープロシージャ解析，ユニモジュラ変換，データローカリティ最適化[5)] などの技術を用いている．

これまで，これらの優れたループ並列化コンパイル

† 早稲田大学
Waseda University

†† ミレニアムプロジェクト IT21 アドバンスト並列化コンパイラ
Millennium Project IT21 Advanced Parallelizing Compiler

技術を用いた様々なプログラムの解析・評価が行われてきたが，ループ並列化手法は既に成熟期に入っており，今後大幅な性能向上は見込めないと言われている．したがって， 今後の並列処理の性能向上のためには，ループ並列性に加え，粗粒度タスク並列性，近細粒度並列性等を用いるマルチグレイン並列化が重要となる．

粗粒度タスク並列性を利用するコンパイラとしては，NANOS コンパイラ[6] と PROMIS コンパイラ[7]，そして OSCAR マルチグレイン並列化コンパイラが挙げられる．NANOS コンパイラでは，階層並列化実現のための拡張 OpenMP API[8] を用いて，粗粒度並列性を含むマルチレベル並列性を抽出しようとしている．また，PROMIS コンパイラは，HTG とシンボリック解析を用いる Parafrase2 コンパイラをベースとして，実用レベルのコンパイラを開発する努力が行われている．日本では，ミレニアムプロジェクト IT21 において，官民連携プロジェクト METI/NEDO アドバンスト並列化コンパイラプロジェクト（APC）[9] が 2000 年秋より 3 年間プロジェクトとして開始された．このプロジェクトではマルチグレイン並列化をキーワードに，共有メモリマルチプロセッサシステム上での実効性能，使いやすさ，コストパフォーマンスを改善することを目標としている．

APC におけるコアコンパイラの 1 つである OSCAR マルチグレイン並列化コンパイラ[10),11)] では，ループ並列性に加え，サブルーチン・ループ・基本ブロックを粗粒度タスクとし，これらのブロック間の並列性を利用する粗粒度タスク並列処理[11)~13)] や，ステートメント間の並列性を利用する近細粒度並列処理[14),15)] を実現している．APC プロジェクトではターゲットマシンが SMP であるため，主に粗粒度タスク並列化が研究対象となっている．

この OSCAR コンパイラにおける粗粒度タスク並列化では，ソースプログラムを 3 種類の粗粒度タスクに分解後，最早実行可能条件解析[11),12)] によって，各粗粒度タスク間の並列性を抽出し，マクロタスクグラフ（MTG）を生成する．生成された MTG の並列性が少なくプロセッサが有効利用できない場合，その MTG 上の粗粒度タスクがサブルーチンブロック，ループブロックである場合は，階層的にその内部を粗粒度タスクに分割して階層的 MTG を生成し，プログラム全域の階層的な並列性を抽出する．

しかし，このような階層的な（ネストされた）粗粒度タスク並列性を持つプログラムを効果的に並列処理するためには，どの階層の MTG に何台のプロセッサを割り当てるかを的確に判断する必要がある．この問題に対処するため，本論文では，階層的 MTG から自動計算した粗粒度並列性を効率的に利用する階層的並列性制御手法を提案し，本手法を実装した OSCAR マルチグレイン並列化コンパイラを用いて，階層的並列処理の評価を行う．本手法では，粗粒度タスク並列性とループ並列性を総合的に考慮した並列度を算出し，割り当てるべきプロセッサ数を決定する．

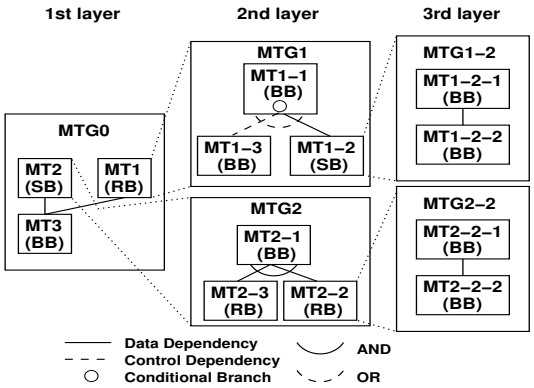


図 1 階層的マクロタスクグラフ
Fig. 1 Hierarchical macro-task graph

2. 粗粒度タスク並列処理

本章では，逐次プログラムの階層的な粗粒度タスク分割，粗粒度タスク間並列性解析手法，階層的マクロタスクグラフ生成について述べる．

2.1 粗粒度タスク生成

粗粒度タスク並列処理では，逐次処理用プログラムを基本ブロックまたはその融合ブロックである BPA（Block of Pseudo Assignments），繰り返し（ループ）ブロック RB（Repetition Block），サブルーチンブロック SB（Subroutine Block）の 3 種類の粗粒度マクロタスク（MT）に分割する．RB や SB に対しては，必要に応じボディ部を階層的に粗粒度タスク分割し，プログラム全域の階層的な並列性をを抽出する．

2.2 粗粒度並列性抽出

各階層のマクロタスク（MT）生成後，MT 間のデータ依存と制御フローを表した階層的マクロフローグラフ[10)] を生成する．

次に，制御依存とデータ依存を考慮し MT 間並列性を最大限に引き出すために，各 MT の最早実行可能条件を解析する．各 MT の最早実行可能条件は，**図 1** に示すような階層型マクロタスクグラフ（MTG）[10)] で表される．

2.3 プロセッサクラスタとプロセッサエレメント

OSCAR コンパイラにおける粗粒度タスク並列処理

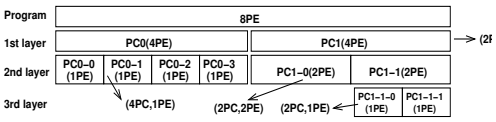


図 2 プロセッサクラスタとプロセッサエレメントの階層的定義
Fig. 2 Hierarchical definition of processor clusters and processor elements

では，複数のプロセッサエレメント（PE）をソフトウェア的にグループ化して 1 つのプロセッサクラスタ（PC）と定義し，この PC に MTG 内の MT を割り当てる．割り当てられた MT 内で更に MTG が定義されている場合は，プログラム中の階層的 MTG の並列性を有効に利用するため，内部 MTG の並列性に応じて PC 内の PE を階層的にグループ化する．

もし，プロセッサを階層的にグループ化しなければ，図 1 に示すような階層的 MTG の場合，第 1 階層（図中，1st layer）の粗粒度タスク並列性は利用できるが，第 2 階層（2nd layer）に示すような下位階層の粗粒度タスク並列性は利用できない．図 1 の MTG_2 内の MT2-2，MT2-3 をループ並列処理不可能な RB と仮定すると，下位階層である MTG_2 内の粗粒度タスク並列性を利用しない場合，MT2-1，MT2-2，MT2-3 の順に 1PC 上で実行されるが，MT2-2，MT2-3 間の粗粒度タスク並列性を利用する場合，MT2-1 を実行後，MT2-2 と MT2-3 を 2PC を用いて同時に実行することができる．したがって，プロセッサに階層構造を持たせた階層的粗粒度タスク並列処理では，単階層のみの粗粒度タスク並列処理よりも多くのプロセッサを有効に利用することができ，プログラムの処理時間も短縮することができる．

例えば図 1 に示す階層的 MTG の場合，プロセッサは**図 2** に示すようにソフトウェア的に階層的 PC にグループ化され，割り当てられた MT を実行する．図 2 の第 1 階層（図中，1st layer）は，利用可能な 8 プロセッサを，4PE を持つ PC0，PC1 の 2PC にグループ化した場合を表している．また第 2 階層（2nd layer）は，PC0 内の 4PE はそれぞれ 1PE を持つ PC0-0~PC0-3 の 4PC へ，PC1 内の 4PE はそれぞれ 2PE を持つ PC1-0，PC1-1 の 2PC に階層的にグループ化される場合を表している．また PC1-1 は，さらにそれぞれ 1PE を持つ PC1-1-0，PC1-1-1 の 2PC に階層的に分割された場合を示している．

以上のように階層的粗粒度タスク並列処理は，プロセッサを効率的に利用でき，プログラムの処理時間の短縮のために有効だが，プログラムの階層的並列性を解析し，適切な階層的プロセッサ利用を一般ユーザが判断するのは非常に困難であるため，利用可能なプロセッサを自動的に効率よくプログラムの各所に割り当てる手法が必要となっている．この問題を解決するため，次章では階層的並列性制御手法を提案し，自動的にプロセッサを割り当てる手法について述べる．

3. 並列処理階層決定手法を用いたプロセッサ割り当て

本章では，生成された階層的マクロタスクグラフ（MTG）に対する階層的並列性制御手法を提案する．一般に，上位階層のマクロタスク（MT）の方がプログラムの処理コストが大きいため，本手法ではより上位の階層に多くのプロセッサを割り当てて並列処理し，同期やスケジューリングによるオーバーヘッドを軽減する方式をとる．

3.1 MT の実行コスト計算

提案手法においては，各 MTG の逐次処理コストを求める必要があるため，まずコンパイラが MTG 中の各 MT の逐次処理コストを算出する．各 MTG の逐次処理コストは，MTG 中の全 MT の逐次処理コストをコントロールフローにしたがって総和した値となる．

処理コスト算出の対象となる MT が繰り返しブロック（RB）の場合は，ループ回転数を内部ブロックコストの合計に乗じた値を RB の逐次処理コストとする．もし対象 RB がループインデックスの初期値および終値が静的に定まらない DO ループであり，かつ添字にループインデックスが含まれる配列が内部ブロックで使用されている場合には，その配列の宣言サイズを DO ループの回転数とするヒューリスティックを利用している．しかし，ループインデックスと配列添字の関係が特定できず宣言サイズが利用できない場合等には，ループ回転数を固定回転数，例えば 100 回転としてコストを算出する．

また，MTG に条件分岐が含まれる場合は，分岐確率を用いて実行コストを求める．本論文では，実行プロファイルなどを用いず，初見のプログラムのコンパイルを行うことを前提としているため，分岐確率は全分岐方向に対して等確率としてコスト算出を行っている．しかし，プロファイル等の分岐確率を用いることができる場合には，より正確な逐次処理コストを求めることが可能であり，後述する提案手法の精度も高まると考えている．

3.2 各 MTG の並列度の計算

提案手法では MT_i 内の MTG_i の逐次処理コストとクリティカルパス長を用い，MTG_i の粗粒度並列度を求める．計算に際して，提案手法はプログラムの下位

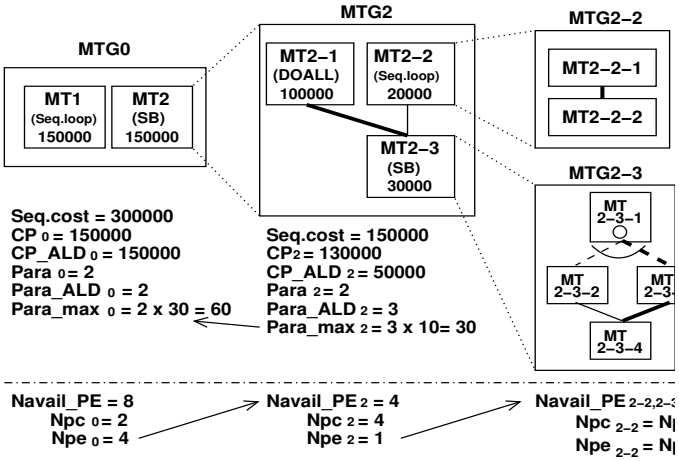


図 3 並列度の計算とプロセッサの割り当て
Fig. 3 Calculation of parallelism and assignment of processors

階層の粗粒度タスク並列性を考慮するため，プログラムの最も深い階層から計算を進める．

MT_i の逐次処理コスト，すなわち MTG_i の逐次処理コストを Seq_i，クリティカルパス長（入口ノードから出口ノードまでの最長のパス長）を CP_i とし，MTG_i の粗粒度並列度 $Para_i$ を以下のように定義する．

$$Para_i = \lceil Seq_i / CP_i \rceil$$

したがって，$Para_i$ は MTG_i を CP_i で処理するための PC 数の下限値となる．

次に粗粒度タスク並列性とループ並列性を総合した並列度 $Para\_ALD_i$（Para After Loop Division）を定義する．MTG_i における粗粒度タスク並列性とループ並列性を総合的に考えるため，提案手法では MTG_i 内のループ並列処理可能 RB に対して，並列タスク駆動オーバヘッド，タスクスケジューリングオーバヘッドを考慮した，並列処理の効果がある最小のコスト T_{min} を用い，並列処理可能 RB がこの T_{min} を超えるコストになるようにタスク分割した場合の粗粒度タスク並列性を，その並列ループの等価粗粒度タスク並列性と考える．ただし，対象 RB の 1 回転分の処理コストが T_{min} を超える場合は，RB の回転数を等価粗粒度タスク並列性とする．ここで等価粗粒度タスク並列性という語を使うのは，この段階では粗粒度並列度の計算上，仮想的なループ分割を考えるだけであり，実際のループ分割は行わないためである．この等価粗粒度タスク並列性を考慮した MTG_i の CP 長を CP_ALD_i とし，MTG_i の粗粒度タスク並列性，ループ並列性を総合的に考慮した $Para\_ALD_i$ を以下のように定義する．

$$Para\_ALD_i = \lceil Seq_i / CP\_ALD_i \rceil$$

$Para\_ALD_i$ は，MTG_i を CP_ALD_i で処理する際に必要な総プロセッサ数であり，MTG_i の粗粒度タスク並列性，ループ並列性を総合して用いる際に十分な PC 数であると言える．よって，MTG_i に $Para\_ALD_i$ を超える PC 数を割り当てた場合，超えた分の PC がアイドル状態になる可能性が高くなる．

次に，MTG_i と，その全下位階層の並列性を考慮する最大粗粒度並列度
$Para\_max_i$ を以下のように定義する．

$$Para\_max_i = Para\_ALD_i \times max\{Para\_max_{insideMTGi}\}$$

$Para\_max_{insideMTGi}$ は MTG_i 内の MT が持つ全ての $Para\_max$ を指し，その最大値は MTG_i の下位階層の最大粗粒度並列度を意味する．MTG_i の全粗粒度並列度 $Para\_ALD_i$ と MTG_i の下位階層の最大粗粒度並列度の積である $Para\_max_i$ は，MTG_i とその下位階層の並列性を最大限に用いる際のプロセッサ数と考えることができる．並列処理可能 RB を含む MTG_i の $Para\_max_i$ を求める際には，RB 自体のループ並列性を $Para\_max_i$ に反映させるため，この RB を処理コストが T_{min} 以上となるよう最大限にタスク分割した際の分割数を，並列処理可能 RB 自体の最大粗粒度並列度とし，$max\{Para\_max_{insideMTGi}\}$ を求める際に利用している．コンパイラにおける後の手順では，PC 数が決定された後，MTG_i 内の並列処理可能 RB は適切な実行コストを持つようタスク分割され，並列コードが生成されるが，潜在的な並列度を求めるこの段階では，並列処理可能 RB のループ並列性を考慮した計算のみを行う．

例として**図 3** における各 MTG の並列度を求める．

図 3 は，第 1 階層の MTG_0，第 2 階層の MTG_2，第 3 階層の MTG_{2-2}，MTG_{2-3} の 4 つの階層的 MTG を表している．図 3 中，DOALL は並列実行可能な繰り返しブロック（RB），Seq.loop は並列実行不可能な RB，すなわち逐次ループを意味する．また，MTG_0，MTG_2，MTG_{2-2} 及び MTG_{2-3} 内の実線エッジはデータ依存，点線エッジは制御依存を表し，太線はクリティカルパス，ノード内の数字は逐次処理コスト，小円は条件分岐を表している．ここでは，ループにおいて並列処理の効果がある最小のコスト T_{min} を 10000 とする．説明を簡単にするため，第 1 階層 MTG_0 内の MT1 における第 2 階層 MTG_1，第 2 階層 MTG_2 内の MT2-1, MT2-2, MT2-3 における第 3 階層 MTG_{2-1}，MTG_{2-2}，MTG_{2-3} 内には並列性がない場合を考える．また図 3 では，MTG_1 と MTG_{2-1} は，スペースの都合で省略している．

まず，最も深い階層から Seq，CP，CP_ALD，$Para$，$Para\_ALD$，$Para\_max$ をそれぞれ求める．MT2-1，MT2-2，MT2-3 の内部 MTG である MTG_{2-1}，MTG_{2-2}，MTG_{2-3} には並列性がないと仮定しているので，これらの MTG では $Para = Para\_ALD = Para\_max = 1$ である．次に MTG_2 の並列度を求める．MTG_2 の逐次処理コストは $Seq_2 = 100000 + 20000 + 30000 = 150000$，$CP_2 = 100000 + 30000 = 130000$ となる．また，MTG_2 の並列処理可能ループ MT2-1 が $T_{min} = 10000$ となるように 10 分割された場合の MTG_2 のクリティカルパス長 CP_ALD_2 は，CP が MT2-1 と MT2-3 から MT2-2 と MT2-3 のパスに変わるため，$CP\_ALD_2 = 20000 + 30000 = 50000$ となる．したがって $Para_2 = \lceil 150000/130000 \rceil = 2$，$Para\_ALD_2 = \lceil 150000/50000 \rceil = 3$ となる．ここで，MTG_2 の最大並列度 $Para\_max_2$ を求めるため，MTG_2 における各マクロタスク内 MTG の最大並列度 $Para\_max_{insideMTG2}$ を求める．並列処理可能 RB である MT2-1 においては，$T_{min} = 10000$ になるように最大限にタスク分割を行う場合の分割数 10 を，MT2-1 自体の最大粗粒度並列度と見なし，$Para\_max_{2-1} = 10$ とする．仮定より MT2-2，MT2-3 の最大粗粒度並列度は 1 であるので，MTG_2 の下位階層の最大粗粒度並列度は $Para\_max_{insideMTG2} = 10$ となる．よって，$Para\_max_2 = 3 \times 10 = 30$ となり，MTG_2 には最大で 30 プロセッサ割り当て可能であることがわかる．また，MTG_2 と同一階層である MT1 内部の MTG_1 には並列性はないと仮定しているため，$Para_1 = Para\_ALD_1 = Para\_max_1 = 1$ である．

次に上位階層である MTG_0 の並列度を求める．MTG_0 においては，$Seq_0 = 150000 + 150000 = 300000$，$CP_0$ は 150000 である．また，MTG_0 には並列処理可能ループがないため，$CP\_ALD_0 = 150000$ でもある．よって，$Para_0 = Para\_ALD_0 = \lceil 300000/150000 \rceil = 2$ となる．最大粗粒度並列度 $Para\_max_0$ は，$Para\_ALD_0 = 2$ と MTG_0 内の MT1 と MT2 内の MTG の最大粗粒度並列度のうち，大きい方である $Para\_max_2 = 30$ を用いて，$Para\_max_0 = 2 \times 30 = 60$ と計算される．

3.3 各階層の PC，PE 構成決定手法

本節では 3.2 節で得られた並列度を用いた，各階層へのプロセッサ自動割り当て手法について述べ，図 3 に示す階層的 MTG を用い，プログラム全体の利用可能プロセッサ数を 8 とした際の自動プロセッサ割り当ての具体例を示す．

3.3.1 MTG の並列度による PC 数，PE 数の仮決定

一般に上位階層の方がタスクあたりの処理コストが大きいため，スケジューリングオーバヘッド，同期オーバーヘッドは相対的に小さくなる．よって提案手法では上位階層の並列性を優先して決定する．現在注目している階層（MTG_i）で利用可能な総プロセッサ数を N_{Avail_PEi} とし，MTG_i の PC 数を N_{PCi}，PC 内 PE 数を N_{PEi} とする．

MTG_i の並列性は $Para_i$，$Para\_ALD_i$ によって示され，$Para_i$ で示される粗粒度タスク並列性を十分に用いるには $Para_i \leq N_{PCi}$ でなければならない．また，粗粒度タスク並列性と並列処理可能 RB の等価粗粒度タスク並列性を総合的に考慮した $Para\_ALD_i$ を超える PC 数を MTG_i に割り当てると，アイドル状態となる PC が増加する可能性が高い．したがって，MTG_i の PC，PE の組み合わせの候補を $[N_{PCi}, N_{PEi}]$ としたとき，粗粒度タスク並列性を最大限利用しつつ，利用可能プロセッサをできるだけ全て利用するため，以下の式（1），(2）を満たす N_{PCi}，N_{PEi} のうち，最大の N_{PCi} を持つ組み合わせを PC 数，PE 数として仮決定とする．ここで仮決定とするのは，後の手順で下位階層の並列性を考慮した補正を行うからである．

$$Para_i \leq N_{PCi} \leq Para\_ALD_i \quad (1)$$

$$N_{PCi} \times N_{PEi} = N_{Avail\_PEi} \quad (2)$$

ただし $Para_i = Para\_ALD_i$ のように N_{PCi} のとり得る値に幅がない場合は，MTG_i の粗粒度タスク並列性をできるだけ用いるため，$Para_i \leq N_{PCi}$ かつ $N_{PCi} \times N_{PEi} \leq N_{Avail\_PEi}$ を満たす最小の N_{PCi} を MTG_i の仮 PC 数とする．

$Para_i \geq N_{Avail\_PEi}$ である場合は，粗粒度タスク並列性を最大限に用いるため，$N_{PCi} = N_{Avail\_PEi}$，$N_{PEi} = 1$ を MTG_i の N_{PCi}，N_{PEi} の組み合わせとする．

例えば，図 3 の MTG_0 の PC 数，PE 数の仮決定を考える．プログラム全体で利用可能なプロセッサ数は 8 であるため，$N_{Avail\_PE0} = 8$ である．MTG_0 では $Para_0 = 2 \leq N_{PC0} \leq Para\_ALD_0 = 2$ 及び式（2）より，$N_{PC0} = 2$，$N_{PE0} = 4$ を PC 数，PE 数と仮決定する．

3.3.2 下位階層の並列性を考慮した PE 数の補正

提案手法では，MTG_i に割り当てる N_{PCi}，N_{PEi} の決定に際して，並列処理可能 RB 以外の下位階層を持つ MT 内の並列性を考慮する．並列処理可能 RB のループ並列性は MTG_i での並列処理に用いられるため，その下位階層は考慮しない．ここで，MTG_i における並列処理可能 RB 以外の MT のうち最大の $Para\_max$ を $MaxN_{PEi}$ とする．$MaxN_{PEi}$ は MTG_i の下位階層に割り当てるプロセッサ数の上限，すなわち N_{PEi} の上限を表す．

もし $MaxN_{PEi} \geq N_{PEi}$ である場合は，仮決定している N_{PEi} を MTG_i に割り当てる PE 数として決定する．

もし $MaxN_{PEi} < N_{PEi}$ である場合は，下位階層へ余分に PE を割り当てることになり，無駄な同期やスケジューリングオーバーヘッドを増加させてしまう可能性が高い．これを避けるため，$N_{PEi} = MaxN_{PEi}$ を PE 数と決定する．

図 3 における MTG_0 では，並列処理可能 RB 以外の MT の $Para\_max$ である $MaxN_{PE0}$ は $MaxN_{PE0} = Para\_max_2 = 30$ である．よって，先の手順で仮決定した $[N_{PC0}, N_{PE0}]=[2,4]$ のうち，$N_{PE0} = 4$ は $N_{PE0} < MaxN_{PE0} = 30$ であるので，そのまま PE 数と決定する．

3.3.3 MTG_i と下位階層の並列性を考慮した PC 数の補正

本節では，3.3.1 節で仮決定された MTG_i の PC 数 N_{PCi} を，MTG_i 自体の並列性と下位階層の並列性を考慮して補正する．

- MTG_i 内に並列処理可能 RB がない場合や，$N_{PCi} \times N_{PEi} = N_{Avail\_PEi}$ である場合

  仮決定している N_{PCi} を割り当て PC 数と決定する．
- MTG_i 内に並列処理可能 RB があり，$N_{PCi} \times N_{PEi} < N_{Avail\_PEi}$ である場合

残りのプロセッサ（$N_{Avail\_PEi} - N_{PCi} \times N_{PEi}$）は，この RB に割り当てることが可能であるにも関わらずアイドル状態となってしまう．よって，MTG_i とその下位階層を考慮した最大粗粒度並列度，すなわち MTG_i に割り当てるプロセッサ数の上限値を意味する $Para\_max_i$ を用いて，$N_{PCi} \times N_{PEi} \geq Para\_max_i$ となる最小の N_{PCi} を捜す．ただし $N_{PCi} \times N_{PEi} > N_{Avail\_PEi}$ となる場合，$N_{PCi} \times N_{PEi} \leq N_{Avail\_PEi}$ となる最大の N_{PCi} を求める．

MTG_i の N_{PCi}，N_{PEi} は以上のように決定され，N_{PEi} を MTG_i の下位階層の N_{Avail_PE} としてこれらの手順を繰り返し，$N_{Avail\_PE} = 1$ となった時点で手順を終了する．

例として，図 3 における MTG_0 の PE 数の補正，及び MTG_2，MTG_{2-2}，
MTG_{2-3} の PC 数，PE 数を考える．3.3.1 節で仮決定した PC 数について考えると，MTG_0 には並列処理可能ループがないため，現在の組み合わせ $[N_{PC0}, N_{PE0}]=[2,4]$ がそのまま PC 数，PE 数と決定される．

MTG_2 について考えると，$N_{Avail\_PE2} = N_{PE0} = 4$ であり，式（1）より $Para_2 = 2 \leq N_{PC2} \leq Para\_ALD_2 = 3$ である．ここで，全プロセッサをできるだけ利用するため式（2）を満たす N_{PC2} を求めると，$[N_{PC2}, N_{PE2}] = [2,2]$ となる．この $N_{PE2} = 2$ であるが，MTG_{2-2}，MTG_{2-3} に割り当てられるプロセッサ数の上限値は図 3 における $Para\_max_{2-2} = 1$ 及び $Para\_max_{2-3} = 1$ より，$MaxN_{PE2} = 1$ であるため，$N_{PE2} = 1$ と補正される．次に MTG_2 には並列処理可能 RB があるため，現在の組み合わせ $[N_{PC2}, N_{PE2}] = [2,1]$ の PC 数を補正する．図 3 より $Para\_max_2 = 30$ であるため，$N_{PC2} \times MaxN_{PE2} \geq Para\_max_2 = 30$ を満たす N_{PC2} を求めると，$N_{PC2} = 30$ となる．しかし，$N_{Avail\_PE2} = 4$ であるため，$N_{PC2} \times MaxN_{PE2} \leq N_{Avail\_PE2}$ を満たす最大の PC 数は $N_{PC2} = 4$ となる．よって，MTG_2 の PC 数，PE 数は $[N_{PC2}, N_{PE2}] = [4,1]$ と決定される．ここで $N_{Avail\_PElower_layer} = 1$ となったので，自動プロセッサ割り当てを終了する．

この結果，提案手法によって，$[N_{PC0},\ N_{PE0}] = [2,4]$，$[N_{PC2},\ N_{PE2}] = [4,1]$，$[N_{PC2-2},\ N_{PE2-2}] = [1,1]$，$[N_{PC2-3},\ N_{PE2-3}] = [1,1]$ と決定される．

4. 性能評価

本章では， 提案する階層的並列性制御手法を OSCAR マルチグレイン並列化コンパイラに実装し，その性能を 16 プロセッサ サーバ IBM pSeries690 Regatta（以下，Regatta）上で評価する．

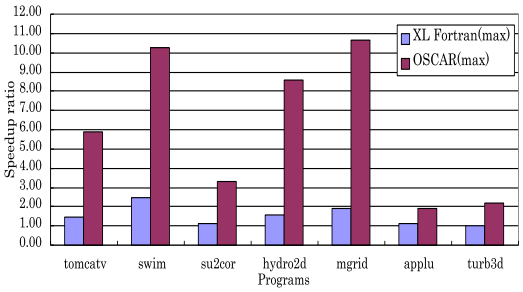


図 4 SPEC95FP の最大性能
Fig. 4 maximum performance for SPEC95FP

4.1 評価環境

本評価では，提案手法を組み込んだ OSCAR コンパイラを並列化プリプロセッサとして用い，OpenMP を用いた粗粒度タスク並列化プログラムを出力した．この OpenMP プログラムは一回のみ単一レベルのスレッド生成を行い，階層的並列処理を行うのに必要なスケジューリングコードなどを各スレッドに埋め込むワンタイム・シングルレベルスレッド生成手法を用い，スレッド生成及びタスクスケジューリングオーバヘッドを最小化することを可能としている[13]．

生成された OpenMP プログラムは IBM XL Fortran for AIX Version 7.1 によってコンパイルされ，Regatta 上で実行される．評価に用いた Regatta は 1.1GHz のチップマルチプロセッサ Power4 を 8 チップ，すなわち 16 プロセッサ利用可能な SMP サーバであり，1 プロセッサあたり命令 L1 キャッシュ64KB，データ L1 キャッシュ32KB，2 プロセッサ共有 L2 キャッシュ1.5MB，全プロセッサ共有 L3 キャッシュ256MB を持ち，共有主メモリは 8GB である．

本評価では，XL Fortran コンパイラでの逐次処理，並列処理の双方で全体的に最小処理時間を得られるコンパイルオプションを用いた．ただし，OS やランタイムライブラリなどの他のパラメータチューニングは行わない．

4.2 SPEC95FP による評価

SPEC95FP のうち，tomcatv，swim，su2cor，hydro2d，mgrid，applu，turb3d の 7 プログラムを用いて提案手法を用いた階層的並列処理の評価を行った．OSCAR コンパイラの出力した OpenMP プログラムを XL Fortran を用いてコンパイルする際のオプションとしては，“-O5 -qsmp=noauto -qhot -qarch=pwr4” を用いた．ただし，su2cor と turb3d の評価においては “-O5” の動作が不安定なため，OSCAR コンパイラには不利であるが最適化レベルの低い “-O4” を用いた．また，今回の評価の目的が階層的並列性制御であるため，OSCAR コンパイラで実現されているデータローカライゼーション手法を用いたキャッシュ最適化は適用していない．したがって，本評価においては，各階層内の並列処理可能 RB を割り当てられた PC 数によって分割し，粗粒度並列処理を行う．また，分割したサブ RB を実行する PE 数が 2 以上の場合は，割り当て PE をそれぞれ 1PE からなる PC として，この並列処理可能サブ RB をイタレーション方向に PC 数で分割した内部 MTG を生成し，スタティックに処理を割り当てる．ただし，並列処理可能 RB の分割数が RB 自体の等価粗粒度並列度よりも大きい場合，すなわち $N_{pc} > Para\_ALD_{RB}$ である場合は，割り当てプロセッサ数ではなく $Para\_ALD_{RB}$ によって分割を行っている．なお， 参考として XL Fortran の自動ループ並列化性能も示す．XL Fortran による逐次処理の際のコンパイルオプションは “-O5 -qhot -qarch=pwr4” を用い，自動ループ並列化用オプションとしては “-O5 -qsmp=auto -qhot -qarch=pwr4” を用いた．また，評価に用いたプログラムのうち，su2cor にはインライン展開，配列リネーミング，turb3d にはループディストリビューションを手動で適用したプログラムを，OSCAR コンパイラによる提案手法，XL Fortran による逐次及び自動ループ並列処理の評価に用いた．

提案手法を用いて並列処理した SPEC95FP の各プログラムの逐次処理時間及び最小処理時間を**表 1** に，各プログラムの最小処理時間の，逐次処理時間に対する速度向上率，すなわち各プログラムの最大性能を**図 4** に示す．表 1 においては，上からプログラム名，XL Fortran の逐次処理時間，XL Fortran のループ並列化で 16 プロセッサを用いた際の処理時間，XL Fortran で 1 プロセッサから 16 プロセッサを用いた際の最小処理時間，提案手法を用いた OSCAR コンパイラで 1 プロセッサから 16 プロセッサまで用いた場合の最小処理時間を示している．図 4 は，横軸がプログラム名，縦軸は各プログラムにおける最大性能を逐次処理時間に対する最小処理時間の速度向上率として表している．また，**表 2** には，提案手法による PC 数，PE 数決定の例を示す．左からプログラム名，プログラム中の場所及び Main ルーチンの最上位階層を第 1 階層とした際の階層名，並列度 $Para$，$Para\_ALD$，その場所で利用可能プロセッサ数 N_{Avail_PE}，割り当て PC 数 N_{PC}，PE 数 N_{PE}，選択したマクロタスクスケジューリング手法を示している．プログラム中の場所については，例えば “MAIN-DO140” はメインルーチンのループ DO140 であり，ルーチン名のみの

表 1 16 プロセッササーバ IBM Regatta 上での SPEC95FP の実行時間（秒）
Table 1 Execution time(seconds) of SPEC95FP on 16 processors IBM Regatta

Programs	tomcatv	swim	su2cor	hydro2d	mgrid	applu	turb3d
Sequential	26.7	22.5	23.0	31.0	36.7	27.0	40.2
XLF 16PEs	29.3	23.2	79.3	116.8	76.7	37.2	41.3
XLF minimum (PE)	18.4(4)	9.1(3)	20.1(2)	19.5(3)	19.5(4)	23.5(3)	40.2(1)
OFC minimum (PE)	4.5(15)	2.2(16)	7.0(15)	3.6(13)	4.6(16)	14.0(16)	18.5(7)

*OFC: OSCAR コンパイラ, XLF: XL Fortran, ( ): 使用プロセッサ数

表 2 主な部分のプロセッサ割り当て結果
Table 2 Processor assignment result by the proposed scheme

Program	場所	$Para$	$Para\_ALD$	N_{Avail_PE}	N_{PC}	N_{PE}	scheduling
tomcatv	MAIN-DO140 第 2 階層	1	494	15	15	1	DYNAMIC
	MAIN-DO140-DO50 第 2 階層内 MT	1	15	15 分割			-
	MAIN-DO140-DO90 第 2 階層内 MT	1	1	分割なし			-
	MAIN-DO140-DO110 第 2 階層内 MT	1	6	6 分割			-
su2cor	LOOPS 第 4 階層または第 5 階層	1	1	15	1	15	DYNAMIC
	LOOPS-DO900-DO400 第 8 階層または第 9 階層	2	3	15	3	5	DYNAMIC
applu	SSOR-DO ISTEP 第 3 階層	1	1	16	1	16	STATIC
	JACLD 第 4 階層	1	1089	16	16	1	STATIC
	BUTS 第 4 階層	1	1	16	1	1	STATIC
turb3d	TURB3D 第 2 階層	1	1	7	1	7	STATIC
	TURB3D-1000 第 3 階層	6	6	7	7	1	DYNAMIC

場合は，そのルーチン全体を指す．マクロタスクスケジューリング手法は，DYNAMIC の場合は実行時のダイナミックスケジューリング，STATIC の場合はコンパイル時にマクロタスクを PC に割り当てることを意味する．

表 1 より tomcatv では，逐次処理時間 26.7 秒，XL Fortran での最小処理時間 18.4 秒に対して，提案手法を用いた OSCAR コンパイラの自動並列処理による実行時間は 15 プロセッサで 4.5 秒となり，図 4 に示すように逐次処理に対して 5.9 倍の性能向上が得られていることがわかる．また，OSCAR コンパイラは XL Fortran の最高性能を 4.1 倍向上させていることがわかる．tomcatv はループ並列性が非常に高いプログラムであり，プログラム中最も時間を要するのが逐次処理ループ DO140 である．提案手法においては，第 2 階層にあたるこの DO140 内部をダイナミックスケジューリングを用いる並列処理階層として選択し，15 プロセッサを用いた並列処理を自動的に行っている．この際，DO140 のループボディにある並列処理可能 RB にはループ分割が適用され，ダイナミックスケジューラによって PC に割り当てられている．ただし，DO140 内の 7 つの並列処理可能 RB の処理コスト算出の結果，表 2 の tomcatv の部分に示す MAIN-DO140-DO90 と MAIN-DO140-DO110 は，処理コストが非常に小さいと判断されるため $Para\_ALD < N_{Avail\_PE} = 15$ となり，全プロセッサを用いた並列処理ではなく，等価粗粒度並列度と等しいプロセッサ数で並列処理を行っている．MAIN-DO140-DO90 は，表 2 に示すように $Para\_ALD = 1$ であるのでループ分割が適用されず 1PC で逐次処理され，MAIN-DO140-DO110 は，$Para\_ALD = 6$ よりループが 6 分割され，6PC で粗粒度並列処理されている．表 2 における MAIN-DO140-DO50 をはじめとする残りの 5 ループは 15 分割され，15PC で粗粒度並列処理されている．

swim, hydro2d も，tomcatv と同様にループ並列性が高いプログラムであるため，階層的マクロタスクグラフの等価粗粒度並列性と利用可能プロセッサ数に応じたプロセッサ割り当てが行われた．swim では，表 1 に示すように，逐次処理時間は 22.5 秒，XL Fortran による最小処理時間は 9.1 秒である．これに対し，OSCAR コンパイラの自動並列化による最小処理時間は 16 プロセッサを用いた際の 2.2 秒であり，図 4 に示す通り逐次処理に対して 10.3 倍の性能向上が得られ，XL Fortran の性能を 4.1 倍向上させることができている．hydro2d では，逐次処理時間 31.0 秒に対して，OSCAR コンパイラの自動並列処理では 13 プロセッサで 3.6 秒となり，逐次処理に対して 8.6 倍の速度向上が得られている．また，XL Fortran の最大性能を 5.4 倍向上させることができている．

mgrid では，表 1 より 36.7 秒の逐次処理時間に対して，XL Fortran の最小処理時間は 4 プロセッサ使用時の 19.5 秒，OSCAR コンパイラの最小処理時間は 16 プロセッサ使用時の 4.6 秒であった．mgrid においては，整合配列やサブルーチン呼び出しによる配列次元数の変更によってループ回転数がほとんど算出できない．したがって，提案手法を適用しても，上位階層・下位階層の並列度を考慮した PC，PE の組み合わせを適切に求められないため，より上位階層の並列処理可能ループに対して，利用可能な全プロセッサを割り当てて並列処理を行っている．

次に su2cor については，表 1 に示すように，逐次処理時間が 23.0 秒，XL Fortran の最小処理時間は 20.1 秒，OSCAR コンパイラの自動並列化による最小処理時間は 15 プロセッサで 7.0 秒であり，図 4 にも示すように OSCAR コンパイラは XL Fortran の最高性能を 2.9 倍向上させていることがわかる．su2cor において，処理時間が全実行時間に対して占める割合が大きいプログラム部分は，**図 5** に示すサブルーチン SWEEP と，第 3 階層及び第 4 階層から呼ばれるサブルーチン LOOPS 内の 3 重ネストループ DO400 であり，提案手法はこのループの最内側をダイナミックスケジューリングによる粗粒度タスク並列処理階層として自動的に選択している．3 重ネストループの DO400 最内側では，表 2 に示す通り $Para = 2$, $Para\_ALD = 3$ となるため，15 プロセッサ使用時は，図 5 に示すように，この階層で $[N_{PC}, N_{PE}]$=[3, 5] を選択し，DO400 のループボディから呼ばれるサブルーチン INT4V を，$[N_{PC}, N_{PE}]$=[5, 1] で実行している．

これに対して，XL Fortran は相対的に小さい処理コストを持つ下位階層のループ並列性のみを利用している．OSCAR コンパイラと XL Fortran の差は，提案手法を適用した OSCAR コンパイラが，より上位の階層を並列処理階層として自動的に選択できたためであると考えられる．

表 1 より，turb3d の逐次処理時間及び最小処理時間は 40.2 秒，OSCAR コンパイラの最小処理時間は 18.5 秒である．turb3d では，サブルーチン TURB3D 内の RB が全実行時間の大部分を占めると判断される．従来研究によって，Main ルーチンから呼び出されるサブルーチン TURB3D 内の第 3 階層にあたる RB からサブルーチン XYFFT，ZFFT を呼び出すループにはループ並列性があることがわかっているが，現在の OSCAR コンパイラはこれらのループ並列性を解析できないため，提案手法はこの並列処理可能 RB の等価粗粒度並列性は用いていない．しかし，提案手法による粗粒度タスク並列性解析の結果，表 2 に示すように $Para = 6$ となっており，表 1 の結果はこの粗粒度タスク並列性のみを用いた際の処理時間である．今後の OSCAR コンパイラの改良によって，これらのループ並列性を解析できれば，さらに効率的なマルチグレイン並列処理が適用可能である．

applu では，表 1 に示すように，逐次処理時間 27.0 秒に対して，XL Fortran の最小処理時間は 23.5 秒，OSCAR コンパイラの自動並列処理による最小処理時間は 16 プロセッサで 14.0 秒となり，図 4 に示すように，逐次処理に対して 1.9 倍の性能向上が得られている．applu においては，表 2 に示す第 4 階層にあたるサブルーチン JACLD の他，サブルーチン JACU，RHS は高い並列性を持つと判断され，利用可能プロセッサ数 16 を [PC, PE] = [16, 1] とし，16 プロセッサを用いて粗粒度並列処理されているが，同じく第 4 階層であるサブルーチン BUTS などの他のサブルーチンには並列性がないと判断し，逐次処理を選択している．

全評価にわたって，XL Fortran の最小実行時間が少ないプロセッサ数で得られ，その実行時間が OSCAR コンパイラに比べて長いのは，XL Fortran ではスレッド管理オーバヘッドが極めて大きいことが要因としてあげられる．例えば，tomcatv では最小実行時間が得られる 4 プロセッサ使用時の全 CPU 時間に対してマスタスレッド上での処理時間が 58%，残りの 3 スレーブスレッドが各 14%と，スレッド間の負荷の不均衡が生じており，より多くのプロセッサ使用時はこの差がさらに拡大する．マスタスレッドの CPU 時間には，逐次処理時間と並列処理時間とスレッド管理オーバヘッドが含まれるが，並列処理可能ループの実行時間が大

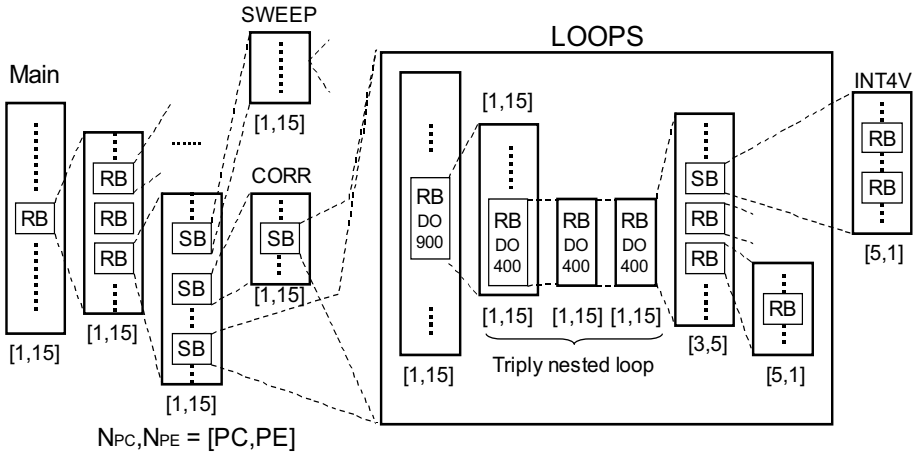


図 5 su2cor の階層構造と 15 プロセッサの自動割り当ての例
Fig. 5 Structure of su2cor and the processor assignment using 15PE

部分を占める tomcatv において，マスタスレッドとスレーブスレッドの CPU 時間の差はあまりに大きいと考えられる．これに対して，OSCAR コンパイラでは，ワンタイムシングルレベルスレッド生成手法と提案する階層的並列性制御手法を用いた自動粗粒度並列処理によって，XL Fortran の自動ループ並列化性能を大きく凌駕する結果を得られている．他のアプリケーションについても同様の傾向であり，XL Fortran において 16 プロセッサを用いた場合の各プログラムの実行時間は，表 1 に示すように，逐次処理と同等か，それよりも悪化してしまっている．

以上の結果より，提案するマルチグレイン並列処理のための階層的並列性制御によるプロセッサ自動割り当て手法によって，プログラムの全階層の並列性に応じた適切なプロセッサ数を自動的に割り当てることが可能であり，効果的な並列処理の実現のために非常に有効であることが確認された．

また，本論文で扱う階層的プロセッサ数決定問題は，最適なプロセッサ割り当てを求めようとした場合は，巨大な探索空間を持つ組み合わせ問題となるため，本論文で提案した階層的並列性制御手法によって得られた結果が最適であるかどうかの判断は極めて難しい．手動による最適化結果との比較を試みたが，現時点ではコンパイラによる自動決定結果を上回る手動最適化は実現できていない．このことからも，プロセッサ割り当ての組み合わせを探索することなく，階層的並列性を考慮した各階層のプロセッサ割り当てを決定できる本手法の有効性は極めて高いと考えられる．

5. ま と め

本論文では，マルチグレイン並列処理における階層的並列性制御手法を提案した．提案手法は，プログラムの階層的マクロタスクグラフの粗粒度並列性とループ並列性を総合的に考慮した並列性を推定し，その並列性に応じてプログラムの上位階層よりプロセッサを割り当てることによって，並列性の高い階層には多くのプロセッサを，並列性の低い階層では並列性に応じた適切なプロセッサ数を自動的に判断して割り当てる．

16 プロセッサ SMP サーバ IBM pSeries690 Regatta において，SPEC95FP を用いて提案手法を用いた自動階層的並列処理の評価を行った結果，逐次処理に対して tomcatv では 5.9 倍，swim では 10.3 倍，su2cor では 3.3 倍，hydro2d では 8.6 倍，mgrid では 10.6 倍，applu では 1.9 倍，turb3d では 2.2 倍の性能向上を得ることができた．また，Regatta 上の自動ループ並列化コンパイラである IBM XL Fortran for AIX Version7.1 の最大性能を tomcatv で 4.1 倍，swim で 4.1 倍，su2cor で 2.9 倍，hydro2d で 5.4 倍，mgrid で 5.7 倍，applu で 1.7 倍，turb3d で 2.2 倍向上させることができ，ワンタイムシングルレベルスレッド生成を用いた OSCAR コンパイラに組み込んだ提案する階層的並列性制御による自動プロセッサ割り当て手法は極めて有効に機能することが確認された．

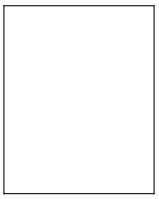
本論文では，SMP 上での評価を行ったが，将来的には，現在研究が進んでいるシングルチップマルチプロセッサ[15] などの階層的並列処理をハードウェアでサポートできるシステム上での評価や，より多くのプロセッサを用いた評価を行いたいと考えている．

謝辞 なお本研究の一部は，経済産業省/NEDO ミレニアムプロジェクト IT21 アドバンスト並列化コンパイラにより行われた．

参 考 文 献

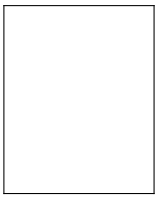
1) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley (1996).
2) Banerjee, U.: Loop Parallelization, *Kluwer Academic Pub.* (1994).
3) Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, *IEEE Trans. on parallel and distributed systems*, Vol. 9, No. 1 (1998).
4) Hall, M. W., Anderson, J. M., Amarasinghe, S. P., Murphy, B. R., Liao, S.-W., Bugnion, E. and Lam, M. S.: Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer* (1996).
5) Lim, A. W. and Lam., M. S.: Cache Optimizations With Affine Partitioning, *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing* (2001).
6) Ayguade, E., Martorell, X., Labarta, J., Gonzalez, M. and Navarro, N.: Exploiting Multiple Levels of Parallelism in OpenMP: A Case Study, *ICPP'99* (1999).
7) Brownhill, C. J., Nicolau, A., Novack, S. and Polychronopoulos, C. D.: Achieving Multi-level Parallelization, *Proc. of ISHPC'97* (1997).
8) Ayguade, E., Gonzalez, M., Labarta, J., Martorell, X., Navarro, N. and Oliver, J.: NanosCompiler: A Research Platform for OpenMP Extensions, *Proc. of the 1st Europian Workshop on OpenMP* (1999).
9) : http://www.apc.waseda.ac.jp/.
10) 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手法, 情報処理学会論文誌, Vol. 35, No. 4, pp. 513–521 (1994).
11) Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K. and Narita, S.: A Multi-grain Parallelizing Compilation Scheme on OSCAR, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).
12) 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, 電子情報通信学会論文誌, Vol. J73-D-I, No. 12, pp. 951–960 (1990).
13) 笠原博徳, 小幡元樹, 石坂一久: 共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理, 情報処理学会論文誌, Vol. 42, No. 4 (2001).
14) Kasahara, H., Honda, H. and Narita, S.: Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR, *Proc. IEEE ACM Supercomputing'90* (1990).
15) 木村啓二, 加藤孝幸, 笠原博徳: 近細粒度並列処理用シングルチップマルチプロセッサにおけるプロセッサコアの評価, 情報処理学会論文誌, Vol. 42, No. 4 (2001).

(平成 0 年 0 月 0 日受付)
(平成 0 年 0 月 0 日採録)



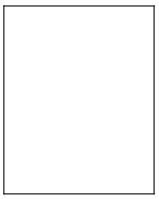
小幡 元樹（正会員）

昭和 48 年生．平成 8 年早稲田大学理工学部電気工学科卒業．平成 10 年同大学大学院修士課程修了．平成 11 年同大学同学部助手．平成 13 年同大学理工学研究科博士課程修了．平成 14 年同大学理工学総合研究センター助手．工学博士．現在は株式会社日立製作所に勤務．在学中はマルチグレイン自動並列化コンパイラに関する研究に従事．



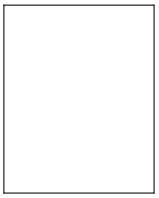
白子 準

昭和 54 年生．平成 14 年早稲田大学理工学部電気電子情報工学科卒業．平成 14 年同大学大学院修士課程進学，現在に至る．



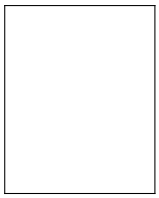
神長 浩気

昭和 52 年生．平成 13 年早稲田大学理工学部電気電子情報工学科卒業．平成 15 年同大学大学院修士課程修了．現在，ソニー株式会社に勤務．在学中はマルチグレイン自動並列化コンパイラに関する研究に従事．



石坂 一久（学生会員）

昭和 51 年生．平成 11 年早稲田大学理工学部電気電子情報工学科卒業．平成 13 年同大学大学院修士課程修了．平成 13 年同大学大学院博士課程進学．平成 14 年同大学理工学部電気電子情報工学科助手，現在に至る．



笠原　博徳（正会員）

昭和 32 年生．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 年同大学同学部助手．昭和 60 年学術振興会特別研究員．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 年同大学電気電子情報工学科教授．平成 15 年同大学コンピュータ・ネットワーク工学科教授，現在に至る．平成元年〜2 年イリノイ大学 Center for Supercomputing Research & Development 客員研究員．昭和 62 年 IFAC World Congress 第一回 Young Author Prize．平成 9 年度情報処理学会坂井記念特別賞受賞．著書「並列処理技術」(コロナ社)．情報処理学会，電子情報通信学会，IEEE などの会員．