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

笠原博徳† 小幡元樹† 石坂一久†

本論文では，共有メモリ型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のワンタイム・シングルレベルスレッド生成を用いた実現方式について提案する．粗粒度タスク並列処理は，現在のループ並列性の限界を超え，シングルチップマルチプロセッサからハイパフォーマンスコンピュータに至る広範囲のマルチプロセッサシステムの性能改善のために，重要な技術である．提案する粗粒度タスク並列処理実現手法では，まずFortranプログラムを粗粒度タスクに分割し，最早実行可能条件解析を用いてタスク間の並列性を解析した後，スタティックに粗粒度タスクをプロセッサに割り当てるか，実行時に粗粒度タスクをプロセッサに割り当てるダイナミックスケジューリングコードを埋め込んだOpenMP並列化Fortranプログラムを生成する．生成されるOpenMP並列化プログラムでは，階層的に粗粒度タスク並列処理を，プログラム開始時の一度だけのスレッドforkと，終了時の一度だけのjoinで低オーバヘッドで実現できる．本論文では，提案手法の有効性を8プロセッサからなる共有メモリマルチプロセッサIBM RS6000 SP 604e High Node上で評価する．本評価では，Perfect Club BenchmarksにおけるARC2D，SPEC 95fpのSWIM，TOMCATV，HYDRO2D，MGRIDを用い，提案する粗粒度タスク並列処理方式で生成したOpenMPコードをIBM XL Fortran compilerでコンパイルする．評価の結果，8プロセッサを用いた場合，提案する粗粒度並列処理手法は，XL Fortran単独によるループ自動並列化性能を1.5から3倍改善できることが確認できた．

Coarse Grain Task Parallel Processing on a Shared Memory Multiprocessor System

Hironori Kasahara,† Motoki Obata† and Kazuhisa Ishizaka†

This paper proposes an implementation method named "one-time single level thread generation" for a coarse grain task parallel processing scheme on an off the shelf shared memory multiprocessor system. The coarse grain task parallel processing is important to improve the effective performance of wide range of multiprocessor systems from a single chip multiprocessor to a high performance computer beyond the limit of the loop parallelism. The proposed scheme decomposes a Fortran program into coarse grain tasks, analyzes parallelism among tasks by "Earliest Executable Condition Analysis" considering control and data dependencies, statically schedules the coarse grain tasks to processors or generates dynamic task scheduling codes to assign the tasks to processors and generates OpenMP Fortran source code for a shared memory multiprocessor system machine. The thread parallel code using OpenMP generated by OSCAR compiler forks threads only once at the beginning of the program and joins only once at the end even though the program is processed in parallel based on hierarchical coarse grain task parallel processing concept. The performance of the proposed scheme is evaluated on a 8-processor shared memory multiprocessor system machine, IBM RS6000 SP 604e High Node, using a newly developed OpenMP backend of OSCAR multigrain compiler. The evaluation shows that OSCAR compiler with IBM XL Fortran compiler gives us 1.5 to 3 times larger speedup than native IBM XL Fortran compiler for SPEC 95fp SWIM, TOMCATV, HYDRO2D, MGRID and Perfect Benchmarks ARC2D.

1. はじめに

Doall，Doacrossのようなループ並列化技術は，マルチプロセッサシステム用の並列化コンパイラで広く用いられてきた[1),2)]．GCDやBanerjeeのinexactおよびexact test[1),2)]，OMEGA test[3)]，シンボリック解析[4)]，セマンティック解析などの様々なデータ依存解析[5),6)]や，アレイプライベタイゼーション[7)]，ループ分割，ループ融合，ストリップマイニング，ループインタチェンジなど[8),9)]のプログラムリストラクチャリング技術により，多くのDoループが並列化可能で

† 早稲田大学
Waseda University

ある．

たとえば，Polaris コンパイラ[10)~12)]は，サブルーチンのインライン展開，シンボリック伝搬，アレイプライベタイゼーション[7),11)]，実行時データ依存解析[12)]によってループ並列性を抽出する．SUIF コンパイラ[13)~15)]は，インタプロシージャ解析，ユニモジュラ変換，データローカリティ[16),17)]に関する最適化などを用いてループを並列処理する．しかし，従来のループ並列化技術では，複雑なループキャリド依存や，ループ外へ飛び出す条件分岐などが存在する場合，効率的な並列処理ができないという問題点が依然存在する．

また最近，メモリとプロセッサの速度差が徐々に大きくなっており，ループ並列化に加え，Blocking，Tiling，Padding，Localization などのプログラムリストラクチャリング技術を用いたデータローカリティ最適化に関する研究が重要になっている．[16),18)~21)]．

したがって，今後のマルチプロセッサシステムの性能改善のためには，データ依存解析，投機実行などのいっそうの高度最適化に加え，サブルーチン・ループ間などのこれまで自動並列化コンパイラでは抽出されていなかった粗粒度並列性を，キャッシュの有効利用も考慮しつつ用いる必要がある．

Parafrase2 をベースとした NANOS コンパイラ[22),23)]は，拡張した OpenMP API[24),25)]によっ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．このマルチレベル並列性を利用するための OpenMP API の独自拡張としては，生成された並列スレッドをグルーピングする GROUPS クローズや，グルーピングされたスレッドをプログラム中の指定した部分に割り当てる ONTO クローズなどが検討されている[26)]．PROMIS コンパイラ[27),28)]では，HTG とシンボリック解析[4)]を用いる Parafrase2 コンパイラ[29)]と細粒度並列処理を行う EVE コンパイラを組み合わせて，プロトタイプ版の開発が共同で行われたが，現在はイリノイ大学で実用レベルのコンパイラの開発が行われている．OSCAR コンパイラは，ループ並列化に加え，粗粒度並列処理[30)~35)]，近細粒度並列処理を効果的に組み合わせたマルチグレイン並列処理[31)~33)]を実現している．OSCAR コンパイラでは，スタティックスケジューリングに加え，条件分岐による実行時不確定性に対処するため，粗粒度タスクはプロセッサ（PE），もしくはプロセッサクラスタ（PC）に実行時にスケジューリングされる．コンパイラにより実行プログラム中に組み込まれる粗粒度ダイナミックタスクスケジューラとしては，集中スケジューリング手法[32),33)]と分散スケジューリング手法[36)]のいずれかを，各階層の並列性および使用可能なプロセッサ数に応じて選択的に用いることができる．PC にスケジューリングされた各粗粒度タスクは，タスクの種類およびタスク内の並列性を考慮して，ループレベル並列処理，粗粒度並列処理，近細粒度並列処理などの方式により PC 内 PE 上で階層的に並列処理される．

本論文では，このような粗粒度タスク並列処理を市販の共有メモリマルチプロセッサシステム上で実現するための手法とその性能評価について述べる．本手法では，逐次処理用 Fortran プログラムは OSCAR コンパイラによって自動的に並列化され，OpenMP API を用いて記述された並列化 Fortran プログラムを生成する．すなわち本研究では，OSCAR Fortran コンパイラは，逐次処理用 Fortran プログラムをプログラムの並列性やターゲットマシンの各種性能パラメータを考慮し，スタティックスケジューリングや集中・分散ダイナミックスケジューリングを用いた OpenMP 並列化 Fortran に変換する並列化プリプロセッサとして用いられる．本手法では，並列スレッドの fork をメインルーチンの最初で一度だけ行い，プログラムの実行終了時に一度だけ join するワンタイム・シングルレベルスレッド生成法で階層的な粗粒度並列処理を実現することにより，スレッド生成オーバヘッドの最小化を行っている．

本論文 2 章では OSCAR コンパイラでの粗粒度並列処理手法について，3 章では提案する粗粒度並列処理実現手法について，4 章では 8 プロセッサの共有メモリマルチプロセッサシステムである IBM RS6000 SP 604e High Node 上での性能評価について述べる．

2. 粗粒度タスク並列処理

粗粒度タスク並列処理とは，プログラムを基本ブロック（BB），繰返しブロック（RB），サブルーチンブロック（SB）の 3 種類のマクロタスク（MT）に分割し，その MT をプロセッサエレメント（PE）や複数 PE をグループ化したプロセッサクラスタ（PC）に割り当てて実行することにより，MT 間の並列性を利用する方式である．

OSCAR マルチグレイン自動並列化コンパイラにおける粗粒度並列処理の手順は次のようになる．

(1) 逐次プログラムの MT への分割．
(2) MT 間の制御フロー，データ依存解析によるマクロフローグラフ（MFG）の生成．
(3) 最早実行可能条件解析によるマクロタスクグラフ（MTG）の生成[30),31),34)]．
(4) MTG がデータ依存エッジのみで構成される場

合は，スタティックスケジューリングによるMTのPCまたはPEへの割当て．MTGがデータ依存エッジと制御依存エッジを持つ場合は，コンパイラによるダイナミックスケジューリングルーチンの自動生成．

(5) OpenMPを用いた並列化プログラムの生成．

以下，各ステップの概略を示す．

2.1 マクロタスクの生成

粗粒度タスク並列処理では，ソースプログラムは，基本ブロック（BB），繰返しブロック（RB），サブルーチンブロック（SB）の3種類のマクロタスク（MT）に分割される．生成されたRBが並列化可能ループの場合は，PC数やキャッシュサイズを考慮した数の粗粒度タスクにループを分割し，それぞれ異なった粗粒度タスクとして定義する．

また，ループ並列化不可能で実行時間の長いRBやインライン展開を効果的に適用できないSBに対しては，そのボディ部を階層的に粗粒度タスクに分割し，並列処理を行う．

2.2 マクロフローグラフの生成

次に，各階層（ネストレベル）において，生成されたMT間のデータ依存と制御フローを解析する．解析結果は，**図1** (a) に示すようなマクロフローグラフ（MFG）として表される．

図中，ノードはMTを表し，実線エッジはデータ依存，点線エッジは制御フローを表す．また，ノード内の小円は条件分岐を表す．図中のエッジの矢印は省略されているが，エッジの向きは下向きを仮定している．

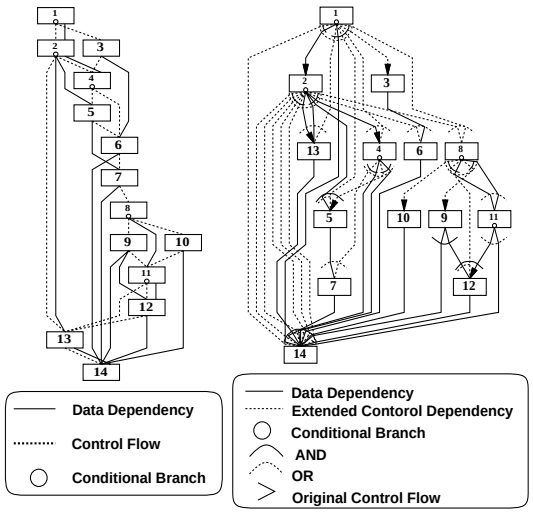

図1 マクロフローグラフ（MFG）とマクロタスクグラフ（MTG）の例

Fig. 1 Sample macro-flow graph and macro-task graph.

2.3 マクロタスクグラフの生成

MFGを基にMT間の並列性を抽出するために，データ依存と制御依存を考慮し，各MTの最早実行可能条件を解析する．最早実行可能条件とは，各MTが最も早い時点で実行可能になる条件を表し，以下の条件を前提として解析される．

(1) MTiがMTjにデータ依存するならば，MTiはMTjの実行が終了するまで実行できない．

(2) もしMTjの分岐方向が確定すれば，MTjの実行が終了しなくても，MTjに制御依存するMTiは実行できる．

よって，最早実行可能条件の一般形は次のようになる．

（MTiが制御依存するMTjが，MTiの実行を確定する方向に分岐する）

AND

（MTiがデータ依存するMTk（$0 \leq k \leq |N|$）が終了するOR MTkが実行されないことが確定する）

ここで N はMTiがデータ依存する先行MTの数である．

たとえば，図1 (a) におけるMT6の最早実行可能条件の一般形は次のようになる．

（MT1がMT3に分岐 OR MT2がMT4に分岐）

AND

（MT3が終了 OR MT1がMT2に分岐）

しかし，MT3の終了はMT1がすでにMT3に分岐していることを意味し，MT2がMT4に分岐するということは，すでにMT1がMT2に分岐していることを意味する．よって，この条件は簡略化されて，次のようになる．

（MT3が終了 OR MT2がMT4に分岐）

各MTの最早実行可能条件は，図1 (b) に示すようなマクロタスクグラフ（MTG）として表される．

MTGにおいても，ノードはMTを，ノード内の小円は条件分岐を表す．また，実線エッジはデータ依存を表し，点線エッジは拡張された制御依存を表す．ただし，拡張された制御依存とは，通常の制御依存だけでなく，MTiのデータ依存先行MTが実行されない条件も含むものである．

図1 (b) 中のエッジを束ねる実線アークは，アークによって束ねられたエッジがAND関係にあることを示し，点線アークはOR関係にあることを表す．

MTGにおいても，エッジの矢印は省略されているが，下向きを仮定している．また，矢印のあるエッジはオリジナルの制御フローを表している．

2.4 粗粒度タスクスケジューリング

粗粒度タスク並列処理においては，ダイナミックスケジューリング，あるいはスタティックスケジューリングを用いて，MT を PC もしくは PE に割り当てる．ダイナミックスケジューリングにおいては，条件分岐のような実行時不確定性に対応するために，MT は実行時に PC もしくは PE に割り当てられる．ダイナミックスケジューリングルーチンは，OS コールによるスレッドスケジューリングオーバヘッドを避けるために，コンパイラによって各プログラムに合わせて生成され，出力される並列化プログラム内に埋め込まれる．

一般的に，ダイナミックスケジューリングオーバヘッドは大きいが，OSCAR コンパイラにおいては実行コストの大きい粗粒度タスクのスケジューリングに用いられるため，相対的にオーバヘッドは小さく抑えられる．ダイナミックスケジューリングには，1 つのプロセッサがスケジューリングルーチンを実行する集中ダイナミックスケジューリングと，すべてのプロセッサがスケジューリングを行う分散ダイナミックスケジューリングの 2 種類が使用可能であり，プログラムの並列性，使用可能なプロセッサ数，同期性能などにより使い分けられる．

また，MTG にデータ依存エッジのみが存在する場合には，スタティックスケジューリングを用いて，PC もしくは PE への MT の割当てをコンパイル時に決定する．スタティックスケジューリングを用いる場合には，実行時スケジューリングオーバヘッドをなくすとともに，データ転送，および同期オーバヘッドを最小化することができる．

3. 共有メモリマルチプロセッサ上での粗粒度タスク並列処理の実現手法

本章では，提案する共有メモリマルチプロセッサ上での粗粒度タスク並列処理の実現手法について述べる．

OSCAR コンパイラにおける粗粒度タスク並列処理では，前述のようにマクロタスク（MT）はプロセッサクラスタ（PC）もしくはプロセッサエレメント（PE）に割り当てられる．2 章で述べた階層的粗粒度タスク並列処理を，NANOS コンパイラのような特殊な OpenMP 拡張を用いてネストされたスレッド生成を行う必要なしに，ワンタイム・シングルレベルスレッド生成で低オーバヘッドで実現できるところに提案手法の特徴がある．この提案手法は，OpenMP をはじめ，種々のスレッド生成手法でも実現可能あるが，本論文では，高いポータビリティを有する OpenMP API を用いる場合の例について述べる．

3.1 並列スレッドの生成

提案する粗粒度タスク並列処理実現手法では，OpenMP ディレクティブ “PARALLEL SECTIONS” によってプログラムの実行開始時に一度だけ並列スレッドを生成する．

一般的に，ネスト並列処理，すなわち階層的な並列処理は，NANOS コンパイラ[22),23)]のように，上位スレッドが子スレッドを生成することによって実現される．しかし，提案手法では，“PARALLEL SECTIONS” ～“END PARALLEL SECTIONS” ディレクティブ間の “SECTION” ディレクティブ間に，生成される全 MTG 階層での処理やダイナミックスケジューリングを適用する全 MTG 階層のスケジューリングルーチンを記述することによって，シングルレベルスレッド生成のみで階層的並列処理を実現する．この手法により，スレッドの fork/join オーバヘッドの最小化，および既存の言語仕様のみで階層的粗粒度並列処理を実現することができる．以下，このスケジューリング実現法について述べる．

3.2 マクロタスクスケジューリング

本節では，スレッドまたはスレッドグループへの MT 割当てを行うコード生成手法について述べる．

コンパイラは，プログラム中の各階層において，データ依存エッジから構成され，各 MT の処理時間をコンパイル時に推定できる MTG に対してはスタティックスケジューリングを選択し，制御依存エッジなどの実行時不確定性を含む MTG に対してはダイナミックスケジューリングを適用する．

OSCAR コンパイラは，プログラムの並列性や，使用可能プロセッサ数，同期オーバヘッドなどのマシンパラメータを考慮し，各 MTG ごとに 3 種類の MT スケジューリング手法を使い分ける．

図 2 は，MT1～3 の 3 つの MT が階層的に粗粒度並列処理される場合に生成される並列コードイメージを表しており，第 1 階層に属する MT1～3 のスレッドグループへの割当てにスタティックスケジューリング，第 1 階層 MT2 内部の第 2 階層には集中ダイナミックスケジューリング，第 1 階層 MT3 内部の第 2 階層には分散ダイナミックスケジューリングを使用する例である．また，第 1 階層においては各 4 スレッドから成る 2 スレッドグループ，すなわちスレッド 0～3 とスレッド 4～7 から成る 2 つのスレッドグループに対して MT を割り当てている．MT2 内部の第 2 階層においては，各グループ 1 スレッドから成る 3 スレッドグループを定義し，残りのスレッド 7 を集中スケジュー

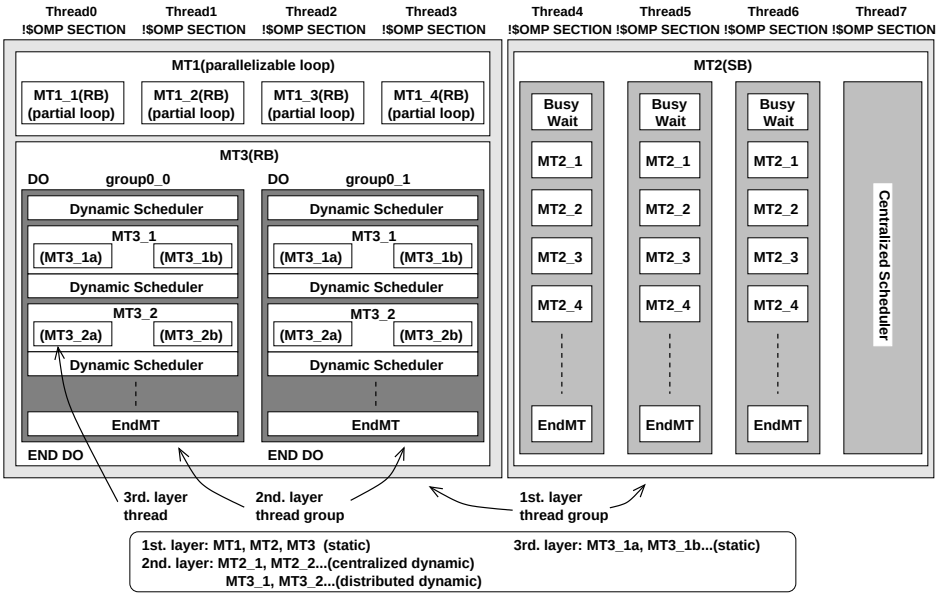

図 2 並列化コードイメージ
Fig. 2 Image of parallelized code.

ラとしている．MT3 内部の第 2 階層においては，各グループ 2 スレッドから成る 2 スレッドグループを定義し，分散スケジューリングにより並列処理を行う場合を示している．

3.2.1 集中ダイナミックスケジューリング

集中スケジューリング手法が適用される階層では，MT を割り当てるべきスレッドグループ中の 1 スレッドを集中スケジューラとして使用する．

集中スケジューラとなるスレッドの動作は以下のようになる．

step1 各 MT から終了通知もしくは分岐情報を受け取る．

step2 最早実行可能条件を調べ，実行可能な MT をレディ MT キューに入れる．

step3 ダイナミック CP 法により割り当てるべき PC もしくはスレッドグループを決定する[31]．

step4 MT 実行スレッドグループに MT を割り当てる．割り当てられた MT が “EndMT”（EMT）である場合，集中スケジューラはその階層におけるスケジューリングを終了する．

step5 step1 に戻る．

集中スケジューリングを適用する MTG の実行開始時には，初期レディ MT が MT 実行スレッドグループに事前に割り当てられている．MT の実行が終了，もしくは分岐方向が確定した際には，集中スケジューラにその情報を送る．

集中スケジューラがこれらの情報を受け取った際には，最早実行可能条件を満たし新しく実行可能になるレディ MT があるかを調べる．レディ MT がある場合にはレディ MT キューに投入し，ダイナミック CP 法によって高い優先度を持つ MT を割り当てるべきスレッドもしくはスレッドグループを決定し，それらのスレッドあるいはスレッドグループに実行すべき MT を通知する．MT を割り当てた後，集中スケジューラは再び Busy Wait 状態に戻る．スケジューラと実行スレッドの間の情報伝達にはスレッド間共有変数を用いており，正しく動作するためには，メモリストアに対する順序が生成されたプログラムどおりである必要がある．したがって，この操作は基本的に OpenMP における Flush ディレクティブで実現可能である．今回の性能評価時に OSCAR コンパイラが出力した OpenMP コードをコンパイルするために用いたネイティブコンパイラ IBM XL Fortran Ver.5.1 は本ディレクティブをサポートしていないが，IBM RS6000 SP 604e High Node はシーケンシャルコンシステンシが守られているため，実現上問題ない．

ダイナミックスケジューリングにおいては，すべてのスレッドあるいはスレッドグループは，実行時の MT 割当ての結果によってはすべての MT を実行する可能性があるため，各スレッドあるいはスレッドグループには全 MT コードが記述されており，スケジューラからの割当てによって，対応する MT コードにジャンプし実行する．

また，コンパイラは各階層のマクロタスクグラフの

実行終了を検出できるように “End MT”（EMT）という制御用 MT を生成する．図 2 における第 1 階層の MT2 の内部に示すように，EMT は当該階層の最後に記述され，その階層を実行している全スレッドを終了する際に，スケジューラはその階層を実行している全スレッドに EMT を割り当ててスケジューリングを終了し，MT 実行スレーブスレッドとともに上位階層に制御を移す．もし，対象 MTG の階層が最上位階層であれば，プログラムは実行を終了する．

図 2 における第 1 階層の MT2 のサブルーチン内部に定義された第 2 階層は，スレッド 4〜6 を並列実行用スレッドとし，スレッド 7 を集中スケジューラとする例を示している．第 2 階層のマクロタスク MT2_1, MT2_2 などは実行時までどのスレッドで実行されるかが確定しないため，スレッド 4〜6 の全スレッドに対して，MT2 内のすべての MT コードを持つスレッドコードを生成する．

3.2.2 分散ダイナミックスケジューリング

分散ダイナミックスケジューリングが選択される階層においては，集中スケジューリングのようにスケジューラ用スレッドは確保せず，各スレッドあるいはスレッドグループは，MT の実行とスケジューリングの両方を行う．

分散ダイナミックスケジューリングでは，スケジューリングのための全共有データは共有メモリ上にあり，排他的にアクセスされる．共有データに対するアクセスに関しては，並列スレッドからのメモリストア順序が守られることが必要である．

分散スケジューラの動作は以下のようになる．

step1 レディキューより，ダイナミック CP 法の優先順位の高い MT を，次に実行すべき MT として選択する．選択された MT が EMT の場合は，その階層の実行を終了する．

step2 MT を実行する．

step3 各 MT の前後に埋め込まれるコードによって，その MT の実行終了あるいは分岐方向を最早実行可能条件テーブルに排他的に登録し，最早実行可能条件を満足する MT を調べ，実行可能 MT をレディ MT キューに入れる．

step4 step1 に戻る．

図 2 における第 1 階層の MT3 の内部に定義された第 2 階層は，分散ダイナミックスケジューリングを用いるコード出力例である．第 2 階層においては，それぞれ 2 スレッドから成る 2 つのスレッドグループ group0_0 および group0_1 が定義され，実行時には各スレッドグループでの MT の実行が終了するたびに，次に実行すべき MT をスケジューリングによって得る．

MT2 内部と同様に MT3 内部においても，第 2 階層 MT である MT3_1，MT3_2 がスレッドグループ group0_0 と group0_1 のどちらのグループで実行されるのかは実行時まで分からないため，図 2 に示すように両方のスレッドグループに同一の第 2 階層の全 MT コードが記述される．

3.2.3 スタティックスケジューリング

注目階層における MTG がデータ依存エッジのみで構成される場合，データ転送，同期，スケジューリングオーバヘッドを軽減するために，スタティックスケジューリングが適用される．

スタティックスケジューリング手法では，PC に対応するスレッドグループ，もしくは PE に対応するスレッドへの MT 割当てはコンパイル時に決定される．したがって，各 OpenMP “SECTION” には，CP/DT/MISF，DT/CP，ETF/CP 法のようなスタティックスケジューリングアルゴリズムによって決定された順序で，そのスレッドで実行すべき MT のコードのみが記述される．すなわち，図 2 の第 1 階層の MT1, MT2 などに示すように，コンパイラは各スレッドあるいはスレッドグループに対して異なるプログラムを生成する．スタティックスケジューリングの結果，MT 間の同期やデータ転送が必要になった場合は，データ転送用スレッド間共有変数にデータを書き込んだ後，同期用スレッド間共有変数による Busy Wait コードが自動挿入される．転送データと同期用データのメモリ操作は，コードどおりに行われなければならないため，提案手法を実現するためには前述のようにストア順序が守られることが必要となる．

4. 性能評価

本章では，提案手法を実装した OSCAR Fortran コンパイラについて述べ，RS6000 SP 604e High Node 8 プロセッサ上での性能評価について述べる．

4.1 OSCAR Fortran コンパイラ

OSCAR コンパイラはフロントエンド，ミドルパス，バックエンドから構成される．本論文では，今回開発した，OpenMP ディレクティブを含む並列化 Fortran ソースコードを自動的に生成する OpenMP バックエンドを用いる．すなわち，OSCAR コンパイラは逐次 Fortran を OpenMP Fortran に変換するプリプロセッサとして動作する．

4.2 IBM RS6000 SP のアーキテクチャ

本評価で用いた RS6000 SP 604e High Node は，200 MHz の PowerPC 604e を 8 プロセッサ搭載した

SMP サーバである．1 プロセッサあたり，32 KB の命令，データ L1 キャッシュと，1 MB のユニファイド L2 キャッシュを持ち，共有主メモリは 1 GB であり，メモリへのストア順序は IBM XL Fortran Compiler を用いることで守られる．

4.3 RS6000 上での性能

評価に用いたプログラムは，Perfect ベンチマークの ARC2D，SPEC 95fp の SWIM，TOMCATV，HYDRO2D，MGRID である．ARC2D は流体問題を解析するための有限差分陰解法のコードであり，オイラー方程式の求解を行う．SWIM は shallow water equation の求解プログラム，TOMCATV はベクトルメッシュ生成プログラム，HYDRO2D は流体力学 Navie Stokes 方程式の求解プログラム，そして MGRID は 3 次元空間の Multi-grid solver である．

本評価においては，OSCAR コンパイラによって自動生成された並列化プログラムを，IBM XL Fortran Compiler Version5.1 [37]でコンパイルし，RS6000 SP 604e High Node1〜8 プロセッサを用いて実行することにより，提案するシングルレベル並列スレッド生成による粗粒度タスク並列処理の性能と，IBM XL Fortran 自動ループ並列化コンパイラの性能を比較する．各ベンチマークのオリジナルのソースプログラムに対して，XL Fortran コンパイラにおける逐次処理用コンパイルオプションとして最高の最適化を行う “-O3 -qmaxmem=-1 -qhot” を用いた際の実行時間を逐次実行時間とし，OSCAR コンパイラ，XL Fortran コンパイラを用いて並列処理を行った際の速度向上率を求めた．OSCAR コンパイラの速度向上率の測定においては，生成するスレッド数をコンパイル時に指定し，OpenMP Fortran プログラムを生成した．また，XL Fortran コンパイラでの自動ループ並列化では，自動並列化用コンパイルオプションとして，最高の最適化オプションである “-qsmp=auto -O3 -qmaxmem=-1 -qhot” を用い，使用スレッド数は実行時に環境変数によって指定した．

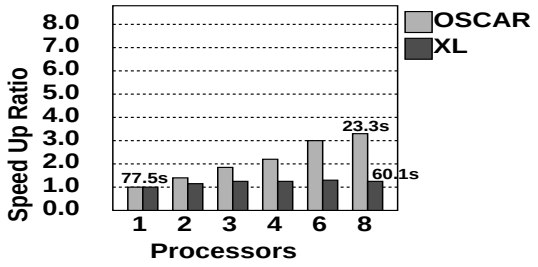

図 3 ARC2D の速度向上率
Fig. 3 Speed up ratio of ARC2D.

図 3 は，ARC2D を OSCAR コンパイラによる粗粒度タスク並列処理と XL Fortran による自動ループ並列処理により RS6000 上で並列処理した場合の逐次処理時間に対する速度向上率を示している．図 3 中，1 プロセッサおよび 8 プロセッサに対応する棒グラフ上の数字は実行時間を示しており，単位は秒である．ARC2D の逐次処理時間は 77.5 秒であり，XL Fortran Version5.1 自動並列化コンパイラを用いた 8 プロセッサでの並列処理時間は 60.1 秒であった．一方，OSCAR Fortran コンパイラによる粗粒度並列処理手法の実行時間は，8 プロセッサで 23.3 秒であった．すなわち，OSCAR コンパイラでは，8 プロセッサでは逐次処理時間に対して 3.3 倍，XL Fortran コンパイラに対しては 2.6 倍の速度向上が得られることが分かる．この ARC2D は，Main ルーチン内の逐次ループから呼ばれるサブルーチン INTEGR が実行時間の 95%以上を占めるアプリケーションであり，**図 4** は Main ルーチンとサブルーチン INTEGR のマクロタスクグラフ（MTG）を示している．図 4 において，Main ルーチン内の逐次ループはマクロタスク（MT）6 として表されている．ARC2D では，サブルーチン INTEGR は分散ダイナミックスケジューリング，Main ルーチンを含むその他の部分はスタティックスケジューリングを用いて階層的並列化が行われた．図 4 におけるサブルーチン INTEGR の MTG は，インライン展開，ループアンローリング，ループ分割，定数伝搬などの最適化後のマクロタスクグラフである．このプログラムでは，サブルーチン INTEGR 内から呼び出される実行時間の大きなサブルーチン内のループの回転数が 3，もしくは 4 と小さいため，ループ並列処理では，プロセッサ数が 5 以上の場合アイドルプロセッサが生じ，並列処理効率が悪化する．しかし，OSCAR コンパイラによる粗粒度タスク並列処理では，図 4 のマクロタスクグラフに示されるようなサブルーチンやループ内部の粗粒度並列性を総合的に利用するため，上述のような性能向上が可能となっている．**図 5** に，3 スレッドを用いて分散ダイナミックスケジューリングを行った際の，Main ルーチンにおける逐次ループ 1 イタレーション分のサブルーチン INTEGR の実行トレースデータを示す．図 5 における MT 番号は，図 4 のサブルーチン INTEGR の MT 番号に対応しており，白い部分は MT の実行時間，灰色の部分はアイドル状態を表す．各 MT の分散スケジューリングオーバヘッドは，図 5 におけるタスク間の線上に表されており，低オーバヘッドでスケジューリングされていることが分かる．また，処理コストの大きな MT100，

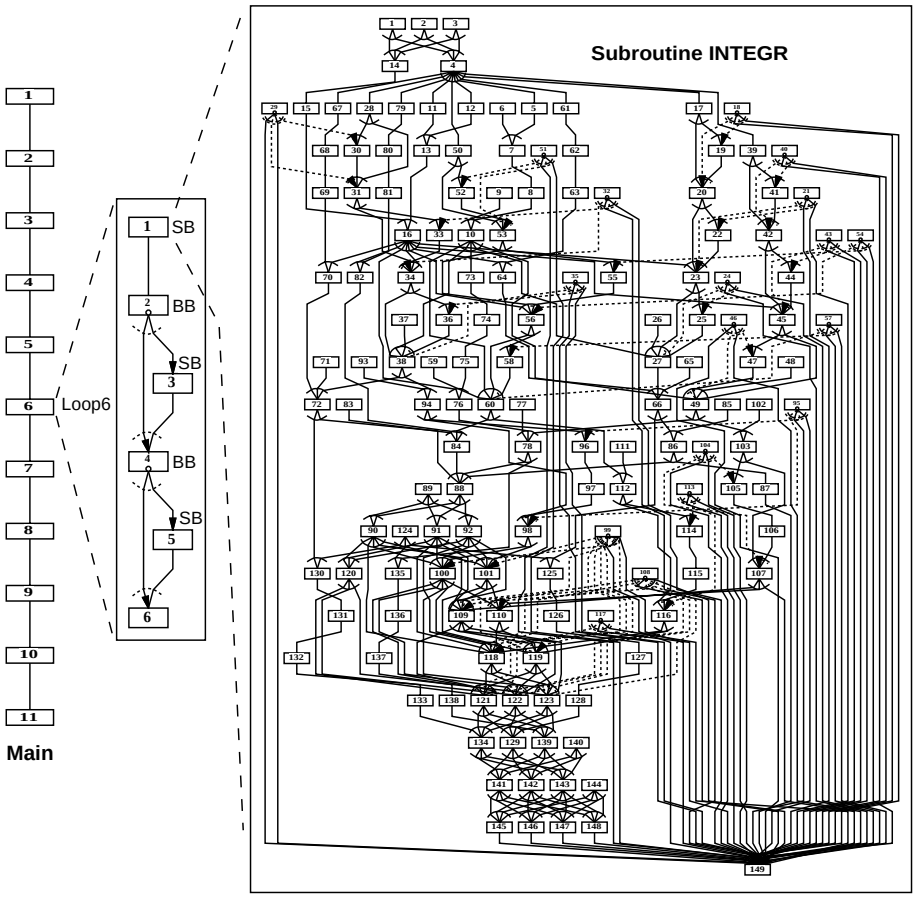

図 4 最適化後のサブルーチン INTEGR の MTG
Fig. 4 Optimized Macro-Task Graph of subroutine INTEGR in ARC2D.

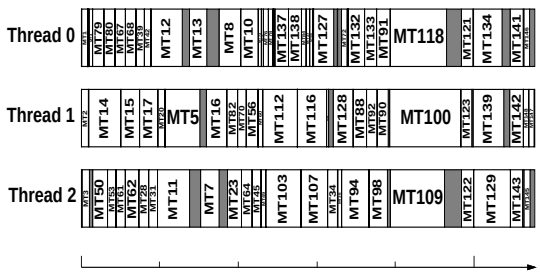

図 5 3 PE 使用時のサブルーチン INTEGR の実行トレース
Fig. 5 Execution trace data of INTEGR in ARC2D.

109，118 の 3 つのサブルーチンが並列に実行されており，粗粒度並列性が有効に利用されていることが分かる．

図 6 は SWIM の速度向上率を示しており，逐次実行時間は図中 1 プロセッサの棒グラフ上に示すように 551 秒であった．XL Fortran による 8 プロセッサを用いた自動ループ並列化による処理時間は 112.7 秒であり，OSCAR コンパイラによるスタティックスケジューリングを用いた粗粒度タスク並列化による処理時間は 61.1 秒であった．すなわち，OSCAR コンパイラでは逐次処理時間に対し 9.0 倍の速度向上，XL Fortran に対しては 1.8 倍の速度向上が得られている．このように，OSCAR コンパイラが XL Fortran による逐次処理時間に対しスーパリニアスピードアップとなっているのは，8 プロセッサにより使用可能なキャッシュ量が増加したため，各プロセッサに分散されたタスクによってアクセスされるデータがキャッシュに収まるようになったためと考えられる．

図 7 に TOMCATV の速度向上率を示す．TOMCATV の逐次処理時間は 691 秒であったのに対し，XL Fortran による 8 プロセッサを用いた並列処理時間は 484 秒であった．これに対し，OSCAR コンパイラによる 8 プロセッサを用いた粗粒度タスク並列処理時間は 154 秒となり，逐次実行に対し 4.5 倍の速度向上が得られ，同数のプロセッサを用いた XL Fortran コンパイラによるループ並列化と比較して 3.1 倍の速度

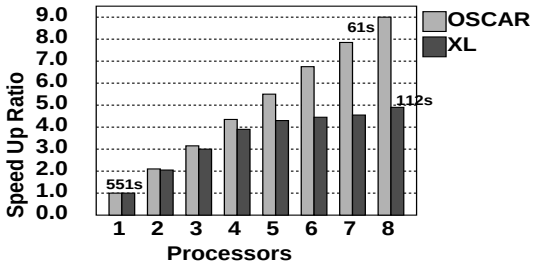

図 6 SWIM の速度向上率
Fig. 6 Speed up ratio of SWIM.

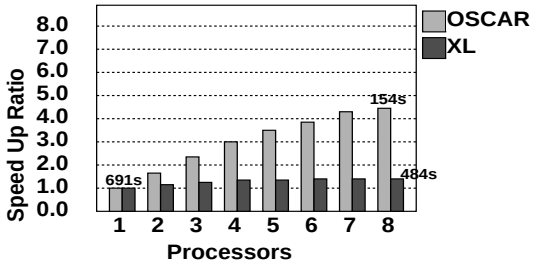

図 7 TOMCATV の速度向上率
Fig. 7 Speed up ratio of TOMCATV.

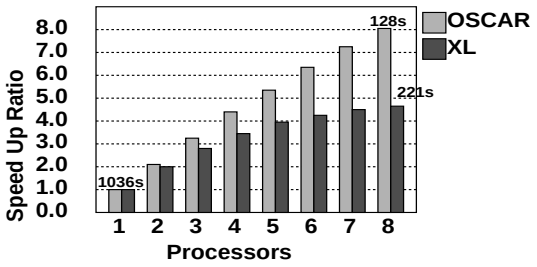

図 8 HYDRO2D の速度向上率
Fig. 8 Speed up ratio of HYDRO2D.

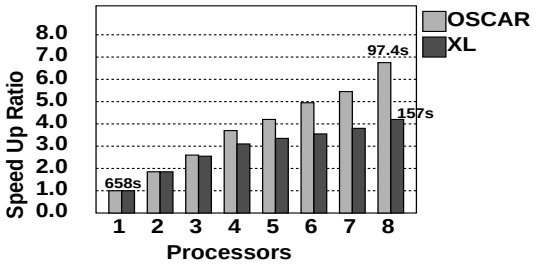

図 9 MGRID の速度向上率
Fig. 9 Speed up ratio of MGRID.

向上が得られた．本来，このアプリケーションはループ並列性が高く，ほとんどのネストループが並列化可能だが，最外側ループで並列化を行うと，メモリアクセスパターンが複雑となり高速化が難しい．今回の評価では，XL Fortran コンパイラも OSCAR コンパイラも自動並列化においては外側ループの並列性を利用しており，どちらのコンパイラも並列性としては同じレベルのループ並列性を利用している．したがって，上述の性能差は，OSCAR コンパイラにおけるスタティックスケジューリングを用いたワンタイム・シングルレベルスレッド生成が，スレッド生成，スレッドスケジューリングのオーバヘッドを軽減したために得られたと思われる．

図 8 は HYDRO2D における速度向上率を示している．HYDRO2D の逐次実行時間は 1036 秒であり，XL Fortran コンパイラの 8 プロセッサの処理時間は 221 秒，すなわち 4.7 倍（221 秒）の速度向上であった．これに対し，OSCAR コンパイラでは 128 秒の並列処理時間が得られ，8.1 倍の速度向上が得られた．すなわち OSCAR コンパイラでは，同じ 8 プロセッサを用いた XL Fortran コンパイラと比較して 1.7 倍の速度向上が得られた．

最後に，**図 9** は MGRID の速度向上率を示している．MGRID の逐次処理時間は 658 秒であった．このプログラムに関しては，逐次処理に対する 8 プロセッサを用いた XL Fortran コンパイラによる速度向上率は 4.2 倍であり，実行時間は 157 秒であった．一方，OSCAR コンパイラでは 6.8 倍の速度向上が得られ，実行時間は 97.4 秒であった．すなわち，OSCAR コンパイラでは，同数のプロセッサを用いた XL Fortran コンパイラと比較して 1.6 倍の速度向上が得られている．

HYDRO2D，MGRID の両プログラムもループ並列性が高く，自動抽出された並列性自体はどちらのコンパイラでも大きな差はないが，OSCAR コンパイラにおけるダイナミックスケジューリングをともなうワンタイム・シングルレベルスレッド生成による粗粒度タスク並列処理がオーバヘッドを低く抑えていることが分かる．

なお，XL Fortran コンパイラの結果に関しては，文献 38) において本評価と同様の結果であることが述べられている．

5. ま と め

本論文では，共有メモリマルチプロセッサシステム上でのワンタイム・シングルレベルスレッド生成を用いた粗粒度タスク並列処理の実現手法と，それを実装した OSCAR マルチグレインコンパイラを用いた評価について述べた．

ワンタイム・シングルレベルスレッド生成では，プログラム開始時にスレッドを一度だけ fork し，終了時に一度だけ join する並列化 Fortran プログラムコー

ドを OpenMP などの並列化 API を用いて作成するだけで，階層的粗粒度タスク並列処理を，ネストスレッド生成などの特別な拡張を行うことなくシングルレベルスレッド生成のみで実現できるシンプルな手法である．したがって本手法は，P-thread などを含めた種々のスレッド生成法に対して適用可能であるが，本論文ではポータビリティという点から OpenMP API を用いた．

本コンパイル手法の性能を，IBM RS6000 SP 604e High Node 上で 8 プロセッサを用いて性能評価した結果，IBM XL Fortran Version 5.1 の逐次処理時間に対して，SPEC 95fp ベンチマークの SWIM では 9.0 倍，TomcatV では 4.5 倍，Hydro2d では 8.1 倍，Mgrid では 6.8 倍，Perfect ベンチマークの ARC2D では 3.3 倍の速度向上が得られ，各ベンチマークにおいてスケーラブルな性能向上が得られた．同じ 8 プロセッサを用いた場合，IBM XL Fortran コンパイラによる自動ループ並列処理に対して，SPEC 95fp ベンチマークの SWIM では 1.8 倍，TomcatV では 3.1 倍，Hydro2d では 1.7 倍，Mgrid では 1.6 倍，Perfect ベンチマークの ARC2D では 2.6 倍の速度向上を得られた．

以上の結果より，提案するワンタイム・シングルレベルスレッド生成による粗粒度タスク並列処理の実現法の有効性が確認できた．今後は，他の共有メモリマルチプロセッサ上での性能評価を行うとともに，分散キャッシュに対するローカライゼーション手法の効果的な適用法などに関する研究を行っていく予定である．

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) Pugh, W.: The OMEGA Test: A Fast and Practical Integer Programming Algorithm for Dependence Alysis, *Proc. Supercomputing '91* (1991).
4) Haghighat, M.R. and Polychronopoulos, C.D.: *Symbolic Analysis for Parallelizing Compliers*, Kluwer Academic Publishers (1995).
5) Barnerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Pub. (1989).
6) Petersen, P. and Padua, D.: Static and Dynamic Evaluation of Data Dependence Analysis, *Proc. Int'l Conf. on Supercomputing* (1993).
7) Tu, P. and Padua, D.: Automatic Array Privatization, *Proc. 6th Annual Workshop on Languages and Compilers for Parallel Computing* (1993).
8) Wolfe, M.: *Optimizing Supercompilers for Supercomputers*, MIT Press (1989).
9) Padua, D. and Wolfe, M.: Advanced Compiler Optimizations for Supercomputers, *Comm. ACM*, Vol.29, No.12, pp.1184–1201 (1986).
10) Polaris: http://polaris.cs.uiuc.edu/polaris/.
11) Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.1 (1998).
12) Rauchwerger, L., Amato, N.M. and Padua, D.A.: Run-Time Methods for Parallelizing Partially Parallel Loops, *Proc. 9th ACM International Conference on Supercomputing, Barcelona, Spain*, pp.137–146 (1995).
13) Hall, M.W., Murphy, B.R., Amarasinghe, S.P., Liao, S. and Lam, M.S.: Interprocedural Parallelization Analysis: A Case Study, *Proc. 8th International Workshop on Languages and Compilers for Parallel Computing* (*LCPC95*) (1995).
14) 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 Comput.* (1996).
15) 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* (1995).
16) Lam, M.S.: Locallity Optimizations for Parallel Machines, *3rd Joint International Conference on Vector and Parallel Processing* (1994).
17) Anderson, J.M., Amarasinghe, S.P. and Lam, M.S.: Data and Computation Transformations for Multiprocessors, *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Processing* (1995).
18) Han, H., Rivera, G. and Tseng, C.-W.: Software Support for Improving Locality in Scientific Codes, *8th Workshop on Compilers for Parallel Computers* (*CPC '2000*) (2000).
19) Rivera, G. and Tseng, C.-W.: Locality Optimizations for Multi-Level Caches, *Super Computing '99* (1999).
20) Kasahara, H. and Yoshida, A.: A Data-Localization Compilation Scheme Using Partial Static Task Assignment for Fortran Coarse Grain Parallel Processing, *Journal Of Parallel Computing Special Issue On Languages And Compilers For Parallel Computers* (1998).
21) 吉田明正，越塚健一，岡本雅巳，笠原博徳：階

層型粗粒度並列処理における同一階層内ループ間データローカライゼーション手法，情報処理学会論文誌，Vol.40, No.5, pp.2064–2071 (1999).
22) Martorell, X., Ayguade, E., Navarro, N., Corbalan, J., Gozalez, M. and Labarta, J.: Thread Fork/Join Techniques for Multi-level Parllelism Exploitation in NUMA Multiprocessors, *ICS'99 Rhodes Greece* (1999).
23) Ayguade, E., Martorell, X., Labarta, J., Gonzalez, M. and Navarro, N.: Exploiting Multiple Levels of Parallelism in OpenMP: A Case Study, *ICPP'99* (1999).
24) OpenMP: Simple, Portable, Scalable SMP Programming
http://www.openmp.org/.
25) Dagum, L. and Menon, R.: OpenMP: An Industry Standard API for Shared Memory Programming, *IEEE Computational Science & Engineering* (1998).
26) Ayguade, E., Gonzalez, M., Labarta, J., Martorell, X., Navarro, N. and Oliver, J.: NanosCompiler: A Research Platform for OpenMP Extensions, *Proc. 1st Europian Workshop on OpenMP* (1999).
27) PROMIS: http://www.csrd.uiuc.edu/promis/.
28) Brownhill, C.J., Nicolau, A., Novack, S. and Polychronopoulos, C.D.: Achieving Multi-level Parallelization, *Proc. ISHPC'97* (1997).
29) Parafrase2:
http: //www.csrd.uiuc.edu/parafrase2.
30) Kasahara, H., Honda, H., Iwata, M. and Hirota, M.: A Macro-dataflow Compilation Scheme for Hierarchical Multiprocessor Systems, *Proc. Int'l. Conf. on Parallel Processing* (1990).
31) Kasahara, H., et al.: A Multi-grain Parallelizing Compilation Scheme on OSCAR, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).
32) 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).
33) Kasahara, H., Okamoto, M., Yoshida, A., Ogata, W., Kimura, K., Matsui, G., Matsuzaki, H. and Honda, H.: OSCAR Multi-grain Architecture and Its Evaluation, *Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (1997).
34) 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-1, No.12, pp.951–960 (1990).
35) 笠原博徳：並列処理技術，コロナ社 (1991).
36) Moreira, J.E. and Polychronopoulos, C.D.: Autoscheduling in a Shared Memory Multiprocessor, *CSRD Report No.1337* (1994).
37) IBM: *XL Fortran for AIX Language Reference.*
38) Kulkarni, D.H., Tandri, S., Martin, L., Copty, N., Silvera, R., Tian, X.-M., Xue, X. and Wang, J.: XL Fortran Compiler for IBM SMP Systems, *AIXpert Magazine* (1997).

(平成 12 年 9 月 14 日受付)
(平成 13 年 3 月 9 日採録)

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，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 1 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 等会員．

小幡　元樹（学生会員）
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 10 年同大学大学院博士後期課程進学．平成 12 年同大学同学部助手，現在に至る．

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，現在に至る．