OpenMPを用いた粗粒度並列処理

石坂一久[†]　小幡元樹[†]　笠原博徳[†]

早稲田大学理工学部電気電子情報工学科[†]

〒169-8555 東京都新宿区大久保3-4-1 TEL:03-5286-3371
E-mail: {ishizaka,obata,kasahara}@oscar.elec.waseda.ac.jp

本論文では，商用SMP上での粗粒度タスク並列処理の実現手法とその性能評価について述べる．粗粒度並列処理は，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において，ループ並列性の限界を越えた性能を得るために重要である．本実現手法では，Fortranプログラムを粗粒度タスクに分割し，タスク間の制御・データ依存を考慮した並列性を解析した後，タスクをプロセッサに割り当てるダイナミックタスクスケジューリングルーチンを埋め込んだ，OpenMP並列プログラムを生成する．本コンパイラが自動的に生成したOpenMP Fortranプログラムでは，プログラム開始時に一度だけスレッドをforkし，終了時に一度だけスレッドをjoinするだけで，階層的な粗粒度タスク並列処理及びループ並列化が行えるため，スレッドのfork/joinオーバーヘッド等を最小化できる．本手法の性能は，新たに開発したOpenMPバックエンドを用いて，8プロセッサのSMPであるIBM RS6000 SP 604e High Node上で評価した．OSCARマルチグレインコンパイラはSPEC 95fpのSWIM, TomcatV, Hydro2d, Mgrid, Perfect BenchmarksのARC2Dに対してIBM XL Fortran Compiler Version 5.1自動並列化コンパイラより1.5〜3倍の速度向上が得られることが確かめられた．

Coarse Grain Task Parallel Processing with OpenMP API

Kazuhisa Ishizaka[†] , Motoki Obata[†] and Hironori Kasahara [†]

Department of Electrical, Electronics and Computer Engineering, Waseda University[†]

3-4-1 Ohkubo Shinjuku-ku, Tokyo 169-8555, Japan Tel: +81-3-5286-3371
E-mail: {ishizaka,obata,kasahara}@oscar.elec.waseda.ac.jp

This paper proposes a realization scheme of automatic coarse grain task parallel processing on a SMP machine using OpenMP. and its performance evaluation. 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 realization scheme decomposes a Fortran program into coarse grain tasks, analyzes parallelism among tasks by "Earliest Executable Condition Analysis" considering control and data dependencies, generates dynamic task scheduling codes to assign the tasks to processors and generates OpenMP Fortran source code. OpenMP Fortran code generated by OSCAR compiler forks threads only once at the beginning of the program and joins threads 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 scheme is evaluated on 8 processor SMP machine, IBM RS6000 SP 604e High Node, using a newly developed OpenMP backend. The evaluation shows that OSCAR compiler gives us 1.5 to 3 times larger speedup than IBM XL Fortran parallelizing compiler version 5.1 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)]．現在，GCDやBanerjeeのinexact and exact test，OMEGA test，シンボリック解析，セマンティック解析などの様々なデータ依存解析や，Array Privatizationや，ループ分割，ループ融合，ストリップマイニング，ループインターチェンジなど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ある．

例えば，Polarisコンパイラ[3)]は，サブルーチンのインライン展開，シンボリック伝搬，Array Privatization[4),5)]，run-timeデータ依存解析[6)]によってループ並列性を抽出する．PROMISコンパイラ[7)]は，HTGとシンボリック解析[8)]を用いるParafrase2コンパイラ[9)]と，細粒度並列処理を行うEVEコンパイラを組み合わせている．SUIFコンパイラ[10)]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[11),12)]に関する最適化などを用いループを並列処理する．データローカリティに関する研究は，プロセッサとメモリのアクセス速度差がますます大きくなっているため重要度が増しており，シングル，及びマルチプロセッサシステムにおけるBlocking, Tiling, Padding, Localizationなどのプログラムリストラクチャリングを用いたデータローカリティの利用に関する研究が行われている[13),14)]．

しかし，これらのコンパイラ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と並列化ができない．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，投機実行等の一層の高度最適化に加え，サブルーチン・ループ間などの粗粒度並列性を用いる必要がある．

Parafrase2をベースとしたNANOSコンパイラは，拡張したOpenMP API[15)]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．OSCARコ

ンパイラは，シングルチップマルチプロセッサから HPC マルチプロセッサにスケーラブルに適用できる粗粒度並列処理とループ並列処理を効果的に組合せたマルチグレイン並列処理 [16] を実現している．OSCAR コンパイラでは，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．PC に割り当てられた粗粒度タスクは，ループレベル並列処理，粗粒度並列処理，近細粒度並列処理を用いて，PC 内 PE によって階層的に並列処理される．

本論文では，粗粒度タスク並列処理手法を商用 SMP 上で実現する手法とその性能評価について述べる．SMP 上での粗粒度並列処理の実現にあたっては，SMP 上での標準並列化 API の OpenMP を用いる．OSCAR コンパイラを，逐次 Fortran プログラムを OpenMP Fortran に変換するプリプロセッサとして動作させ，粗粒度タスク並列処理のためのダイナミックスケジューリングコードを含めた OpenMP Fortran を自動的に出力する．本実現手法では，スレッドの fork をメインルーチンで一度だけ行い，プログラムの実行終了時に一度だけ join するだけでネストループ等に対する階層的な並列処理が実現でき，スレッドの fork/join オーバヘッドを最小に抑えることができる．

以下，2 章では OSCAR Fortran コンパイラでの粗粒度並列処理手法について，3 章では OpenMP を用いた粗粒度並列処理実現手法について，4 章では SMP である RS6000 上での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，ソースプログラムを基本ブロック (BB)，繰り返しブロック (RB)，サブルーチンブロック (SB) の 3 つのマクロタスク (MT) に分割し，その MT をプロセッサクラスタ (PC) やプロセッサエレメント (PE) 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，粗粒度タスク並列処理の手順は次のようになる．

1. ソースプログラムからマクロタスクを生成
2. マクロタスク間のコントロールフロー，データ依存を解析しマクロフローグラフ (MFG) を生成
3. 最早実行可能条件解析を行いマクロタスクグラフ (MTG) を生成
4. MTG がデータ依存エッジしか持たない場合は，MT はスタティックスケジューリングによって PC または PE に割り当てられる．一方，MTG がデータ依存エッジとコントロール依存エッジを持つ場合は，コンパイラによってユーザコード中に埋め込まれたダイナミックスケジューリングルーチンによって，MT を PC または PE に実行時に割り当てる．

2.1 マクロタスクの生成

粗粒度タスク並列処理では，まずソースプログラムを BB，RB，SB の 3 種類の MT に分割する．

生成された RB がループ並列処理可能な場合は，ループを 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，実行時間の大きなループ並列処理不可能な 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，その内部 (ボディ部) を階層的に粗粒度タスク (MT) に分割し，並列処理を行う．

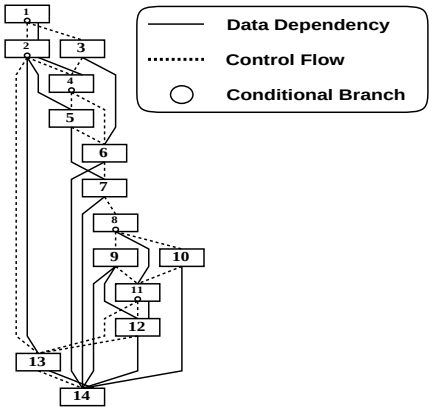

図 1: マクロフローグラフ

2.2 マクロフローグラフ (MFG) の生成

次に生成された各階層の MT に対して，MT 間のコントロールフローとデータ依存を解析する．解析された結果は図 1 に示すようなマクロフローグラフ (MFG) で表される．

図のノードは MT を表し，実線エッジはデータ依存を，点線エッジはコントロールフローを表す．また，ノード内の小円は条件分岐を表す．この MFG ではエッジの矢印は省略されているが，エッジの方向は下向を仮定している．

2.3 マクロタスクグラフ (MTG) の生成

MFG は MT 間のコントロールフローとデータ依存は表すが，並列性は表していない．並列性を抽出するためには，コントロールフローとデータ依存の両方を考慮した最早実行可能条件解析を各 MT に対して行う．最早実行可能条件とは，その MT が最も早い時点で実行可能になる条件であり，次のような実行条件から求められる．

1. MTi が MTj にデータ依存するならば，MTj の実行が終了するまで MTi は実行できない
2. MTj の条件分岐先が確定すれば，MTj の実行が終了しなくても，MTj にコントロール依存する MTi は実行できる．

したがって，最早実行可能条件の，一般形は次のようになる．

（MTi がコントロール依存する MTj が，
MTi に分岐する）
AND
（MTi がデータ依存する MTk（$0 \leq k \leq |N|$）が終了
OR MTk が実行されないことが決定する）

例えば図 2 の MFG の MT6 の最早実行可能条件は，次のようになる．

（MT1 が MT3 に分岐 OR MT2 が MT4 に分岐）
AND
（MT3 が終了する OR MT1 が MT2 に分岐）

MFG におけるコントロールフローを考えると，MT3 が終了するということは，MT1 は MT3 に分岐したということを含み，また MT2 が MT4 に分岐するということは，MT1 は MT2 に分岐しているので，この条件は簡略化され，次のようになる．

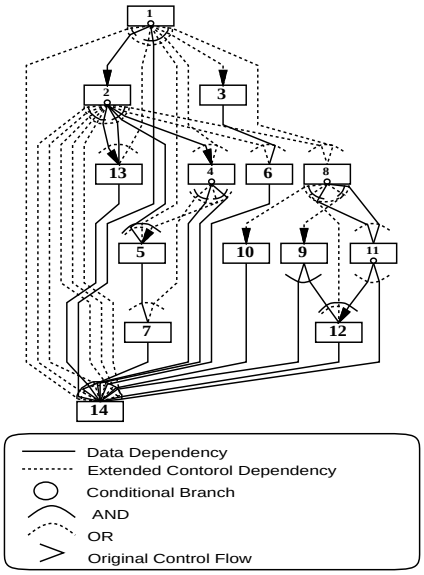

図 2: マクロタスクグラフの例

（MT3 が終了する OR MT2 が MT4 に分岐）

MT の最早実行可能条件は図 2 に示すようなマクロタスクグラフ (MTG) で表される．

MTG におけるノードは MT を表し，ノード内の小円は MT 内の条件分岐を表している．実線のエッジはデータ依存を表し，点線のエッジは拡張されたコントロール依存を表す．拡張されたコントロール依存とは，通常のコントロール依存だけでなく，MTi のデータ依存先行 MT が実行されない条件も表す．

また，エッジを束ねるアークには 2 つの意味があり，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，点線アークは束ねらエッジが OR 関係にあることを示している．

MTG においてはエッジの矢印は省略されているが，下向きが想定されている．また，矢印を持つエッジはオリジナルのコントロールフローを表す．

2.4 スケジューリングコードの生成

粗粒度タスク並列処理では，各階層で生成された MT は PC に割り当てられて実行される．どの PC に MT を割り当てるかを決定するスケジューリング手法として，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がある．

ダイナミックスケジューリングは，条件分岐などの実行時不確定性に対処するため，実行時に MT の PC への割り当てを決める方式である．コンパイラはこのダイナミックスケジューリングコードを生成し，並列化されたユーザプログラムの中に埋め込むことによって，スレッドスケジューリングのための OS コールを除去し，オーバーヘッドを軽減している．

一般にダイナミックスケジューリングはオーバーヘッドが大きいが，OSCAR コンパイラでは，粗粒度タスクの割り当てに適用しているので，相対的にオーバーヘッドを小さく押さえることができる．また，ダイナミックスケジューリングコード生成時には，一つの専用の PE がスケジューリングを行う集中スケジューリング方式と，スケジューリング機能を各プロセッサに分散した分散スケジューリング方式を，使用するプロセッサ台数，システムの同期オーバーヘッドを考慮して使い分けることができる．

また，スタティックスケジューリングは，MTG がデータ依存エッジのみを持つ場合に使用され，自動並列化コンパイラがコンパイル時に MT の PC または PE への割り当てを決める方式である．スタティックスケジューリングは，実行時スケジューリングオーバーヘッドを無くし，データ転送と同期のオーバーヘッドを最小化することが可能である．

3 OpenMP を用いた実現

本章では OpenMP を用いて粗粒度タスク並列処理を行う方法について述べる．

3.1 スレッドの生成

提案する OpenMP を用いた粗粒度タスク並列処理では，プログラムの開始直後に使用プロセッサ数と同数のスレッドを，PARALLEL SECTIONS ディレクティブによって生成する．

一般に，ネスト並列処理すなわち階層的な並列処理を行う場合は，一度生成されたスレッドがさらに子スレッドをフォークして，複数階層の並列処理を行うという手法が取られる．

しかし本手法では，PARALLEL SECTIONS と END PARALLEL SECTIONS 内の各セクションに，全階層にわたっての各スレッドの処理を記述しておくことで，スレッドの fork/join を一回に押さえることができる．

この実現手法により，1 回の fork/join で特別な言語拡張を用いることなく，階層的な粗粒度タスク並列処理を実現できる．

3.2 マクロタスクの実行

本節ではマクロタスクを階層的にスレッドグループに割り当てるためのコード生成法について述べる．

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方式では，スレッドグループへの MT の割り当てを，一つのスレッドが集中制御する．このスレッドを集中スケジューラー (CS) と呼ぶ．

CS となるスレッドの動作は次のようになる．これを実現するコードが CS 用スレッドの OpenMP ”セクション”に記述される．

step1 MT の終了情報を受け取る

step2 最早実行可能条件 (ECC) を調べて，ECC を満たす実行可能 MT を，レディキューに入れる．

step3 実行可能 MT を割り当てるスレッドを決める

step4 MT をスレッドに割り当てる．この MT が EMT の場合は，この階層のスケジューリングルーチンを終了する．

step5 1 に戻る

一方，CS 以外のスレッド (スレーブスレッド) は CS によって割り当てられた MT を実行する．スレーブスレッドの動作は

step1 CS からの MT 割り当て情報を待つ

step2 割り当てられた MT を実行する

step3 1 に戻る

のようになる．

したがって，集中スケジューリングが適用される階層の，スレーブスレッド用のコードの先頭には，MT の割り当てを待つ busy/wait コードが記述される．

ダイナミックスケジューリングでは，スレーブスレッドが実行する MT は実行時に決められるので，それぞれのスレーブスレッドのセクションには，その階層の MT のコードを全て記述しておき，前述の MT 割り当て待ちコードからジャンプによって実行できるようにしておく．

また，スレーブスレッドは MT の実行終了後は，再び CS からの MT の割り当てを待つために busy/wait コードにジャンプする．

図 3 に (k+1) 個のスレッド用に生成された OpenMP コードのイメージを示す．この図では最外側の四角で囲まれた第 1 階層は，ソースプログラムを分割して生成された 4 つのマクロタスク MT1, MT2, MT3, MT4 から構成される．この図の場合，第 1 階層にはスタティックスケジューリングが適用されており，後述するように，それぞれのスレッドが実行するべきコードが，Thread1 には MT1, MT2_1，Thread2 には MT2_2 のように，各セクションに記述されている．

第 1 階層の MT3 は，(k+1) 個のスレッドを 1 グループとしたスレッドグループに割り当てられており，MT3 の内部をサブ MT(MT3_1, MT3_2...) に分割して，第 2 階層の粗粒度タスクレベルで並列処理を行うことを示している．この図では第 2 階層には集中スケジューリングを選択した場合を示している．この第 2 階層では (k+1) 個のスレッドの内，k+1 番目のスレッドを CS とする場合の例を示している．図では残りの k 個のスレッドを二つのスレッドからなるグループに分割し，第 2 階層の MT3_1, MT3_2,... のタスクは，この k/2 個のスレッドグループに割り当てられ，MT 間で並列処理されると共に，各 MT はグループ内の 2 スレッド間でさらに並列処理される．

またこの図では，第 2 階層の MT3_2 をさらに内部をサブ MT (MT3_2_1, MT3_2_2,...) に分割して，第 3 階層の粗粒度並列処理をする場合を示している．この階層では，第 2 階層のスレッドグループ内の二つのスレッドを，さらに一つのスレッドからなる二つのスレッドグループに分割し，後述する分散スケジューリング方式で粗粒度タスク並列処理を行う場合を示している．

また，コンパイラは EndMT(EMT) という特殊な MT を生成する．この MT はその階層に属する全てのスレッドのセクションに記述されており，CS はその階層の処理が終わると，各スレッドグループにこの EMT を割り当てる．CS は全スレッドグループに EMT を割り当てると，スケジューリングルーチンを終了する．また各スレッドグループは図 3 に示すように EMT を実行するとジャンプ文によってその階層の処理を抜け，再び割り当て待ち部分に戻らずに，その階層の処理を終了する．この階層が最上位の階層ならばプログラムは終了し，上位の階層があればそちらに処理が戻る．

3.2.2 分散ダイナミックスケジューリング

分散スケジューリングが適用される階層では，各スレッドグループが自分が実行する MT を決定する．分散スケジューリングでは，各スレッドグループによって共有されるスケジューリング用の変数 (最早実行可能条件管理テーブル，レディタスクキュー) は共有メモリ上にとられ，排他的にアクセスされる．分散スケジューリング時のスレッドの動作は以下のようになる．

step1 スケジューリング変数に排他的にアクセスし，最早実行可能条件を満たした実行可能 MT を探し，

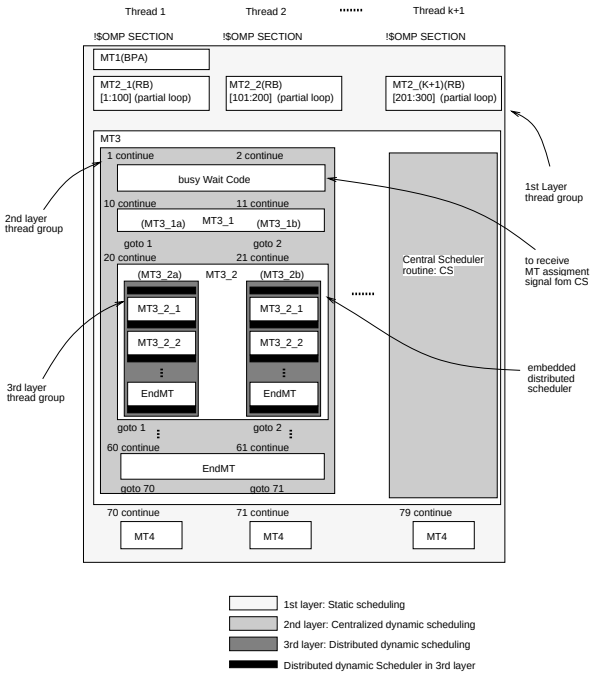

図 3: 生成される並列化コードイメージ

レディキューに入れる．

step2 レディキュー中の実行可能 MT の中から，自分が次に実行する MT を選ぶ．

step3 MT を実行する．

step4 MT 終了，分岐方向等のスケジューリング変数を排他的に更新する．

step5 1 に戻る．

図 3 では，第 2 階層の MT3_2 の内部を第 3 階層として，分散スケジューリングを適用した場合のコード生成イメージを示している．例では，それぞれ一つのスレッドからなる二つのスレッドグループが，MT3_2_1，MT3_2_2 等のタスクを分散スケジューリングでスレッドに割り当てる場合のコードを示している．

3.2.3 スタティックスケジューリング

対象階層の MTG がデータ依存エッジしか持たない場合は，データ転送，同期，スケジューリングのオーバーヘッドを最小化するために，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では，その階層での MT のスレッドグループへの割り当ては，コンパイル時に決められる．したがって，セクションには実行すべき MT が実行する順番に記述されている．すなわちコンパイラは，図 3 の第 1 階層のように，各スレッド用に異なったコードを生成する．

実行時には，各スレッドグループ間で同期をとり，MT 間のデータ依存を満たすように実行が行われる．

3.3 スレッドグループ内での処理

本節では，スレッドグループが MT を実行するときの処理とそのためのコード生成法を述べる．スレッドグループが MT を実行するときは，その MT の種類によって異なった処理が行わる．以下に MT の種類毎に述べる．

スレッドグループに割り当てられた MT が BB の場合は，グループ内の一つのスレッドがその BB を実行する．

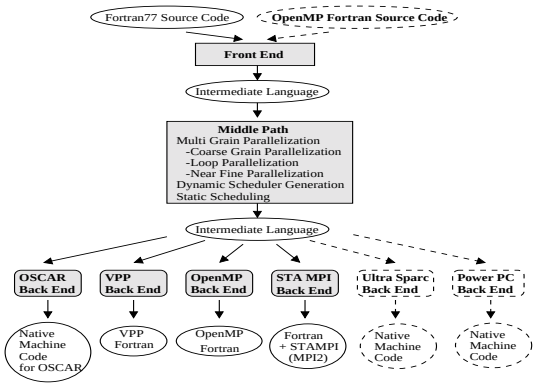

図 4: OSCAR Fortran コンパイラの構成

したがって，図 3 の第 1 階層の MT1(BPA) は，一つのスレッドのセクションにのみに記述され，その他のスレッドのセクションには MT1 のコードは記述されない．

MT がループ並列処理可能なループの場合は，コンパイル時にループをイタレーション方向に分割して，グループ内のスレッドにスタティックに割り当てる．図 3 の MT2(RB) は，(k+1) 個の小ループ (MT2_1,MT2_2, ..., MT2_(k+1)) に分割され，それぞれ，第 1 階層内の (k+1) 個のスレッドの用のセクションに記述される．

MT が実行時間の大きなシーケンシャルループやサブルーチンブロックの場合は，その内部を MT に分割して，グループ内のスレッドによって，階層的に粗粒度タスク並列処理を行う．

4 性能評価

本章では，本論文で述べる手法を実装した OSCAR Fortran Compiler について述べ，RS6000 SP 604e High Node 8 プロセッサ SMP 上での性能評価について述べる．

4.1 OSCAR Fortran Compiler

本コンパイラは図 4 に示すように，フロントエンド（FE），ミドルパス（MP），様々なマシン用のバックエンド（BE）から成る．OSCAR Fortran コンパイラには，OSCAR 分散/集中共有メモリマルチプロセッサシステム，富士通 VPP，MPI-2，UltraSparc，PowerPC，OpenMP のような様々なターゲット，並列処理言語，ライブラリ用のバックエンドがある．本論文で用いる OpenMP バックエンドは，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．したがって，OSCAR Fortran Compiler は逐次処理用 Fortran から OpenMP Fortran を出力するプリプロセッサとして動作する．

4.2 評価プログラム

今回の性能評価には，Perfect ベンチマークの ARC2D，SPEC 95fp の SWIM，TomcatV，Hydro2d，Mgrid を用いた．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あり，オイラー方程式を求解する．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．

4.3 IBM RS6000 SP のアーキテクチャ

本評価で用いた RS6000 SP 604e High Node は，200MHz 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，32KB の命令，データ L1

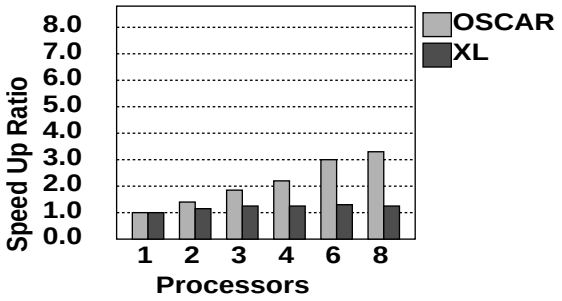

図 5: RS6000 上での ARC2D の速度向上率

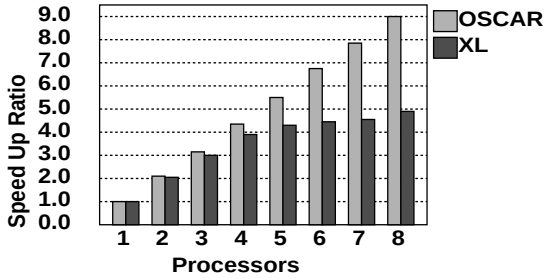

図 6: RS6000 上での SWIM の速度向上率

キャッシュと，1MB のユニファイド L2 キャッシュを持ち，共有主メモリは 1GB である．

4.4 SMP サーバ上での性能

本評価においては，OSCAR コンパイラによって自動的に生成された並列化プログラムを，IBM XL Fortran Compiler Version5.1 でコンパイルし，RS6000 SP 604e High Node の 1〜8 プロセッサを用いて実行し，IBM XL Fortran 自動並列化コンパイラのみを用いて実行した時との性能を比較した．この評価においては，XL Fortran 単体によるコンパイルオプションは，最大最適化オプションである “-qsmp=auto -O3 -qmaxmem=-1 -qhot” を用いた．

図 5 は，ARC2D を RS6000 上で 1 プロセッサから 8 プロセッサを用いて実行した際の 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の速度向上率を示している．OSCAR コンパイラでは 8 プロセッサでは 1 プロセッサに対して 3.3 倍の速度向上，同じ 8PE を用いた XL Fortran コンパイラに対しては 2.6 倍の速度向上を得られていることが分かる．

図 6 から図 9 にそれれぞれ，spec95 の swim, tomcatv, hydro2d, mgird における速度向上率を示している．

OSCAR コンパイラでは，逐次処理に対して 8PE でそれぞれ，9.0，4.5，3.1，8.1，6.8 倍の速度向上が得られている．ここで，プロセッサ数以上の速度向上率が得られたのは，タスクの分割によりキャッシュヒット率が上がったためである．

また，8PE での性能を XL Fortran コンパイラと比較すると，OSCAR コンパイラは各アプリケーションに対し，1.8, 3.1, 1.7, 1.6 倍の性能向上を達成することが確かめられる．

今回の評価に用いた全ての結果において，OSCAR コンパイラの速度向上率はプロセッサ数の増加に対しスケーラブルに増加している．これに対して，XL Fortran コンパイラの結果は，図 6, 8, 9 において，4 プロセッサまではスケーラブルに速度が向上するが，5 プロセッサ以上の

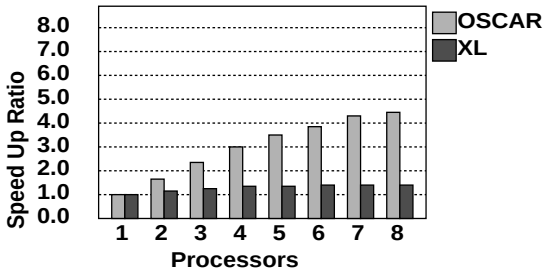

図 7: RS6000 上での TomcatV の速度向上率

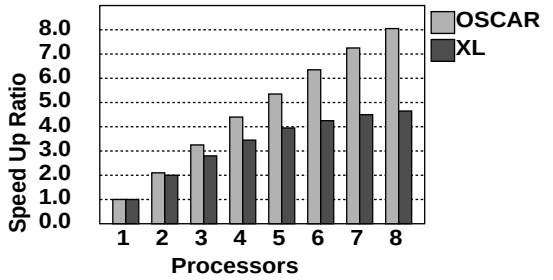

図 8: RS6000 上での Hydro2d の速度向上率

場合，速度向上が鈍くなっている [17]．

5 まとめ

本論文では，粗粒度タスク並列処理の OpenMP を用いた SMP 上での実現手法とその評価について述べた．OSCAR コンパイラでは，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 するだけで，階層的な粗粒度タスク並列処理を低オーバーヘッドで実現できる．本手法の性能を，8PE を搭載した IBM RS6000 SP 604e High Node 上の自動ループ並列化コンパイラである IBM XL Fortran Version 5.1 と比較した結果，同じ 8 プロセッサを用いた場合，XL Fortran コンパイラよりも Perfect ベンチマークの ARC2D で 2.6 倍，SPEC 95fp ベンチマークの SWIM で 1.8 倍，TomcatV で 3.1 倍 Hydro2d で 1.7 倍，Mgrid で 1.6 倍というように大幅な速度向上を得られることが確認できた．

今後は，他の SMP 上での評価を行いつつ，分散キャッシュに対する効果的なローカライゼーション手法の適用，ダイナミックスケジューリングルーチンの改良など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] Polaris: http://polaris.cs.uiuc.edu/polaris/.

[4] Tu, P. and Padua, D.: Automatic Array Privatization, *Proc. 6th Annual Workshop on Languages and Compilers for Parallel Computing* (1993).

[5] 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).

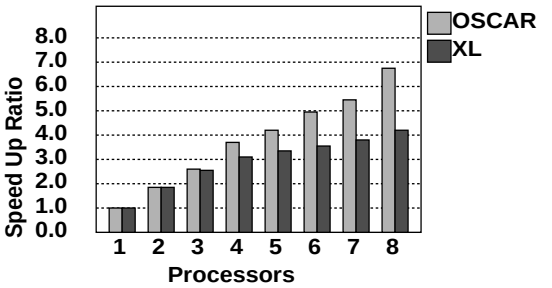

図 9: RS6000 上での Mgrid の速度向上率

[6] Rauchwerger, L., Amato, N. M. and Padua, D. A.: Run-Time Methods for Parallelizing Partially Parallel Loops, *Proceedings of the 9th ACM International Conference on Supercomputing, Barcelona, Spain*, pp. 137–146 (1995).

[7] PROMIS: http://www.csrd.uiuc.edu/promis/.

[8] Haghighat, M. R. and Polychronopoulos, C. D.: *Symbolic Analysis for Parallelizing Compliers*, Kluwer Academic Publishers (1995).

[9] Parafrase2: http://www.csrd.uiuc.edu/parafrase2/.

[10] 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).

[11] Lam, M. S.: Locallity Optimizations for Parallel Machines, *Third Joint International Conference on Vector and Parallel Processing* (1994).

[12] Anderson, J. M., Amarasinghe, S. P. and Lam, M. S.: Data and Computation Transformations for Multiprocessors, *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Processing* (1995).

[13] 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).

[14] Rivera, G. and Tseng, C.-W.: Locality Optimizations for Multi-Level Caches, *Super Computing '99* (1999).

[15] : OpenMP: Simple, Portable, Scalable SMP Programming http://www.openmp.org/.

[16] et al., H. K.: A Multi-grain Parallelizing Compilation Scheme on OSCAR, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).

[17] 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).