計算機アーキテクチャ 141－7
（2001．1．26）

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実現手法の評価

石坂一久[†] 八木哲志[†] 小幡元樹[†,††]
吉田明正[†††] 笠原博徳[†,††]

早稲田大学理工学部電気電子情報工学科[†]
アドバンスト並列化コンパイラ研究体[††]
東邦大学理学部情報科学科[†††]

〒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

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までの幅広い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，粗粒度タスク並列処理の利用が重要となっている．また，プロセッサとメモリアクセス速度の差が大きくなっており，プロセッサに近接した共有メモリを有効に利用するためのデータローカリティの最適化も重要性を増している．本論文では，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，タスクをプロセッサに割り当てて並列実行させる粗粒度タスク並列処理手法，および粗粒度タスク並列処理の性能をさらに向上させるためのデータローカライゼーションを用いたキャッシュ最適化手法について述べる．本手法はOSCARマルチグレインコンパイラに実装されており，OpenMP Backendを用いることによって，本コンパイラは逐次FORTRANから，SMP上での標準APIであるOpenMPを用いて粗粒度タスク並列処理を実現する並列化FORTRANプログラムを自動生成する．本論文ではOSCAR FOTRAN Compilerを用いてOpenMP FORTRANプログラムを自動生成して，IBM RS6000 SP High Node上で，本手法の性能評価を行なった．性能評価ではSPEC 95fpのswim, mgirdにおいて，OSCARコンパイラにより，IBM XL FORTRANコンパイラ version 6.1の自動並列化に対して，最大約2倍の速度向上率が得られることが確かめられた．

Evaluation of coarse grain task parallel processing on the shared memory multiprocessor system

Kazuhisa Ishizaka[†] , Satoshi Yagi[†] , Motoki Obata[†] , Akimasa Yoshida[†††] and Hironori Kasahara [†]

Dept. of Electrical, Electronics and Computer Engineering, Waseda University[†]
Advanced Parallelizing Compiler Research Group[††]
Dept. of Information Science, Toho 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

Coarse Grain Task Parallel Processing is important to improve effective performance of wide range of multiprocessor systems from a single chip multiprocessor to a high performance computer beyond the limit of loop parallelism. In addition, it is important to optimize data locality because the speed gap between a processor and a memory is getting larger. This paper describes a realization scheme of coarse grain task parallel processing which decomposes a FORTARN program to coarse grain task and analyzes parallelism among tasks considering control and data dependencies, and schedules tasks to processors to execute these in parallel. This paper also describes a cache optimization scheme using data localization for coarse grain task parallel processing. The proposed scheme is implemented on OSCAR FORTRAN multi grain parallelizing Compiler. Using Open MP Backend, this compiler automatically generates a parallel FORTRAN program which realizes coarse grain task parallel processing by using OpenMP API. The proposed scheme is evaluated on IBM RS6000 SP High Node. The performance evaluation shows OSCAR Compiler gives up about 2 times larger performance up than the automatic parallelization of IBM XL FORTRAN Compiler version 6.1 for swim, mgrid in SPEC 95fp.

1 はじめに

小規模サーバからハイパフォーマンスコンピューティングシステムまで，幅広くマルチプロセッサシステムが普及してきている．これらのマルチプロセッサシステムの実行性能を上げ，ユーザーの使い易さを向上させるために，自動並列化コンパイラが重要である．

従来の自動並列化コンパイラは，強力なデータ依存解析

とプログラムリストラクチャリング手法を用いたループ並列化に重点がおかれている[1),2)]．イリノイ大学の Polaris コンパイラ[3)]は，シンボリック解析，Array Privatization[4),5)]，run-time データ依存解析[6)]などを用いてループ並列性を抽出する．スタンフォード大学の SUIF コンパイラ[7),8)]は，インタープロシージャ解析，ユニモジュラ変換[9),10)]，データローカリティに関する最適化などを用いたループ並列処理を行なっている．

データローカリティの最適化は，プロセッサとメモリのアクセス速度の差がますます大きくなる中，その重要度が高まっており，Blocking, 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に関する研究が行なわれている[11),12)]．

しかし，これらの手法を用いても，複雑な依存やループ外へ飛び出す条件分岐などのために並列化できないループは存在する．また，ループ並列化技術は成熟期に達しているため，今後大幅な性能向上は望めない．したがって，さらにマルチプロセッサシステムの実行性能を向上させるためには，従来のループ並列化に加えて，ループ間やサブルーチン間といった粗粒度レベルの並列性や，基本ブロック内での命令・ステートメント間の並列性を利用する必要がある．Parafrase2[13)]をベースとするカタルーニャ大学の NANOS コンパイラ[14)]は，拡張した OpenMP API 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．

また，OSCAR マルチグレイン並列化コンパイラは，ステートメント間の並列性を利用した近細粒度並列処理，基本ブロックやサブルーチン・ループ間の並列性を利用した粗粒度タスク並列処理を従来のループ並列化手法を組み合わせたマルチグレイン並列処理を実現している[15),16)]．

本論文では，粗粒度タスク並列処理および，データローカライゼーション手法[17)]を利用した粗粒度タスク間のキャッシュ最適化手法について述べる．本手法は OSCAR マルチグレインコンパイラに実装されており，今回の性能評価での共有メモリマシン上での実現には，共有メモリマシン上での標準 API である OpenMP[18)]を用いる．OSCAR コンパイラは，OpenMP FORTRAN を出力する Backend を持ち，シーケンシャル FORTRAN プログラムから，粗粒度タスク並列処理を実現する OpenMP FORTRAN プログラムを生成する．生成された OpenMP FORTRAN プログラムを用いて，本手法を商用 SMP マシンである IBM RS6000 SP High Node 上で性能評価する．

以下，2 章では OSCAR コンパイラでの粗粒度並列化手法について，3 章では粗粒度タスク並列処理に適用するキャッシュ最適化手法について，4 章で OSCAR コンパイラについて，5 章で本手法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，ソースプログラムを疑似代入文ブロック (BPA)，繰り返しブロック (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，そのマクロタスクを複数のプロセッサエレメント (PE) から構成される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，マクロタスク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，マクロタスク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，マクロタスクを PC または PE に実行時に割り当てる．

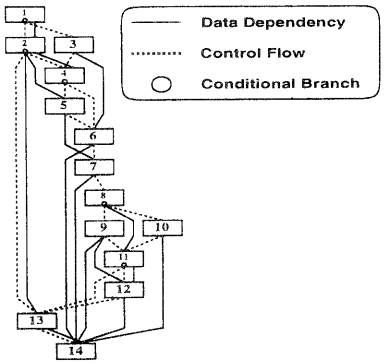

図 1: マクロフローグラフの例

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 BPA，RB，SB の 3 種類のマクロタスクに分割する．

また 3 章で述べるように，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し，ループイタレーション間の並列性の利用，マクロタスク間でのキャッシュ最適化を行う．

さらに，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に分割して，並列処理を行う．

2.2 マクロタスク間の並列性解析

次に生成された各階層のマクロタスクに対して，マクロタスク間の並列性を解析する．

2.2.1 マクロフローグラフ (MFG) の生成

まず生成された各階層のマクロタスクに対して，マクロタスク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．

図のノードはマクロタスクを表し，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．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．

2.2.2 マクロタスクグラフ (MTG) の生成

MFG はマクロタスク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マクロタスクに対

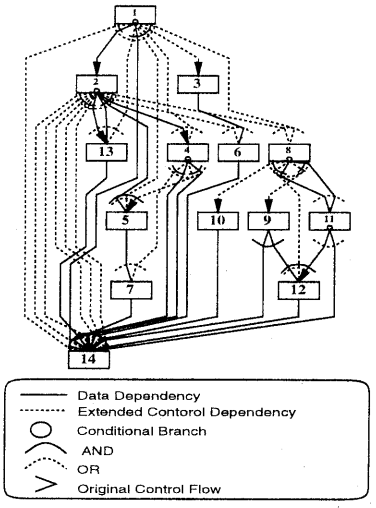

図 2: マクロタスクグラフの例

して行う．マクロタスクの最早実行可能条件とは，そのマクロタスクが最も早い時点で実行可能になる条件である．

マクロタスク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．

MFG と同様に，MTG におけるノードはマクロタスクを表し，ノード内の小円はマクロタスク内の条件分岐を表している．また，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，データ依存と制御依存を複合的に満足させるため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んでいる．

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れたエッジが 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．

2.3 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マクロタスクはプロセッサクラスタ (PC) に割り当てられて実行される．どの PC にマクロタスク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り，マクロタスクグラフの形状，実行時非決定性などを元に選択される．

2.3.1 ダイナミックスケジューリング

ダイナミックスケジューリング手法は，条件分岐などの実行時不確定性に対処するために，実行時にマクロタスクの割り当てを決める方式である．

実行時スケジューリングのための，スケジューリングコードはコンパイラによって生成され，コンパイラの生成する並列化されたユーザーコードの中に埋め込まれている．このため，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 FORTRAN コンパイラでは，粗粒度タスクの割り当てに適用すると共に，各ユーザープログラム毎に最適化されたダイナミックスケジューリングルーチンをコンパイラが自動生成しているため，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，一つの専用のプロセッサ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．

2.3.2 スタティックスケジューリング

一方，スタティックスケジューリングは，マクロタスクグラフ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マクロタスクの PC 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では，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 粗粒度タスク並列処理におけるデータローカライゼーション

本章では，粗粒度タスク並列処理に，データローカライゼーションを適用して，キャッシュを有効利用する手法について述べる．

3.1 マクロタスク間のキャッシュ最適化

データ共有量の多いマクロタスクを同じプロセッサで実行すれば，マクロタスク間での共有データの授受を，分散キャッシュ，分散共有メモリまたはローカルメモリを用いて高速に行うことができる．

今回の評価はローカルメモリを持たない主記憶共有型マルチプロセッサ上で行なったので，分散キャッシュを対象にデータローカライゼーション手法を適用し，キャッシュの有効利用により，粗粒度タスク並列処理の性能を向上させる．

図 3(b) に示すマクロタスクグラフは，図 3(a) のマクロタスクグラフに後述するループ整合分割を適用したものであるが，灰色の帯で結ばれたマクロタスク間のデータ共有量が多いことを表している．マクロタスクグラフでは，マクロタスク間のトランシティブなデータ依存エッジは省略されているが，図 3(b) のマクロタスク (2,13,23) 間，(3,14,24) 間，(4,15,25) 間，(5,16,26) 間にはデータ依存があり，特に共有データ量が大きい．また，マクロタスク 6-12,17-22,27-32 は使用データ量，実行時間が小さくキャッシュへの影響が小さいマクロタスクである．

オリジナルのプログラム上での実行順は，マクロタスクグラフのノードの番号順である．オリジナルの実行順にしたがった場合は，例えばマクロタスク 2 の実行後，使用データ量の大きなマクロタスク 3,4,5 を実行することによって，マクロタスク 2 の使用したデータが，マクロタスク 13 を実行する前にキャッシュから追い出されてしまう．したがって，マクロタスク間でのキャッシュの利用効率が悪い．

シングルプロセッサでダイナミックスケジューリングを適用した場合のトレース例を図 4 に示す．図は 1 行目から 4 行目へと時間的に接続されたものを表している．スケジューリング結果より，使用データ量の小さいマクロタスクを無視して見ると，マクロタスク (3,14),(4,15,25) が連続して実行されており，キャッシュが有効利用されていることが分かる．

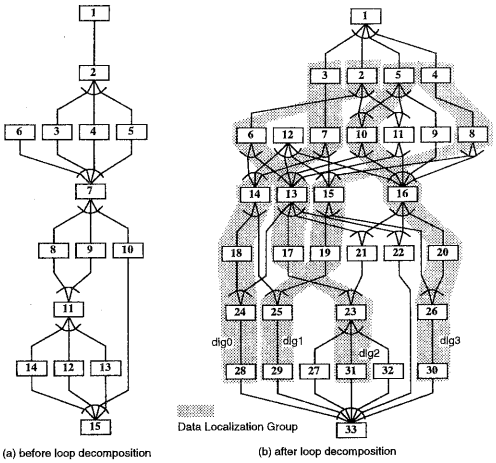

図 3: 分割前後のマクロタスクグラフ

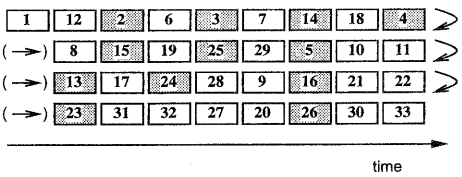

図 4: スケジューリング結果 (1PC)

3.2 ループ整合分割

本手法では，データ使用量が大きく，キャッシュに載りきらないループに対しては，データ量がキャッシュサイズに収まるようにイタレーション方向に分割し，別マクロタスクとして定義する．分割には，データを共有しているループ間でのデータ使用範囲が等しくなるようにループ整合分割 [19),20)] を用いる．

図 3(a) に示すマクロタスクグラフにおけるマクロタスク 2,3,7,8,11 それぞれを，ループ整合分割を用いて 4 分割したグラフが図 3(b) である．例えば分割前のマクロタスク 2 は，図 3(b) のマクロタスク 2,3,4,5 に分割されている．

3.3 グループ指定スケジューリング

キャッシュ利用効率を向上させるために，ループ整合分割によってキャッシュサイズに収まるように分割されたマクロタスク (部分ループ) のうち，データ共有量の多いものを同じ PC に割り当てる必要がある．これを実現するために，ループ整合分割によって分割されたマクロタスクに対して，図 3 中の灰色の帯に示すような Data Localization Group(dlg) というグループ指定を行う．これは，分割されたループが同じ PC に割り当てられることを保証するためのもので，同じグループ (dlg) に指定されたマクロタスクはダイナミックスケジューラーによって同じ PC にスケジューリングされる．

ダイナミックスケジューラーは，スケジューリング時に，このグループ指定を使って，同じグループのマクロタスクを同じ PC に割り当てるように拡張される．

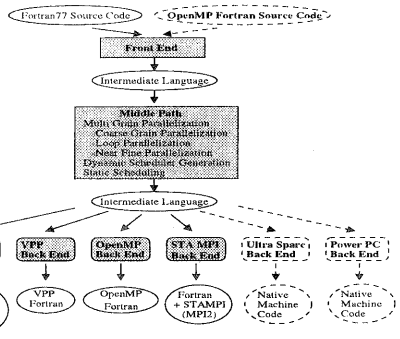

図 5: OSCAR FORTRAN Compiler の構成

4 OSCAR FORTRAN Compiler

本章では，本論文の手法を実装した OSCAR FORTRAN コンパイラについて述べる．

4.1 構成

OSCAR FORTRAN コンパイラは図 5 に示すように，フロントエンド，ミドルパス，複数のバックエンドから構成される．

フロントエンドは FOTRAN77 のソースコードを読み込み，シーケンシャルな中間言語を生成する．

ミドルパスは，制御フロー解析，データ依存解析を行い，プログラムのリストラクチャリング，マクロタスクの生成および，並列性の自動抽出を行ない，解析結果に基づき並列化された中間言語を生成する．

OSCAR FORTRAN コンパイラは，OSCAR マルチプロセッサシステム，UltraSparc，富士通 VPP，STAMPI，OpenMP のような様々なターゲット用のバックエンドを持ち，ミドルパスが出力した並列化中間言語から，各ターゲット用のアセンブリコード，もしくは並列化FORTRAN を生成する．

4.2 OpenMP Backend

OSCAR FORTRAN コンパイラの OpenMP Backend は，ミドルパスが生成した並列化中間言語から，OpenMP API で並列性を記述した並列化 FORTRAN プログラムを生成する．

生成された OpenMP FORTRAN では，プログラム開始時に，PARALLEL SECTIONS ディレクティブを用いて，一度だけプロセッサ台数分のスレッドを生成する．各 OpenMP セクションには，コンパイラによりタスクコードと，ダイナミックスケジューリングを行うときにはスケジューリングコードが生成され，本論文で述べる粗粒度タスク並列処理，およびキャッシュ最適化が実現される．OpenMP を用いた粗粒度タスク並列処理の実現 [21)] では，スレッドの fork/join は 1 度しか行なわず，スレッド生成のオーバーヘッドを抑えている．

図 6 に OpenMP Backend によって生成される OpenMP FORTRAN プログラムのコードイメージを示す．この例ではプログラム開始時に 8 スレッドが fork される．図の例では，第 1 階層 (プログラム全体) の内部に MT1, MT2, MT3 が定義されている．生成された 8 スレッドは第 1 階層では 4 スレッドずつにグループ化されている．このグループは PC に相当し，グループ単位でマクロタスクを実行する．この例では第 1 階層にはスタティックスケジューリングが適用され，スレッド 0-3 のグループに MT1,MT3 が，スレッド 4-7 のグループに MT2 が割

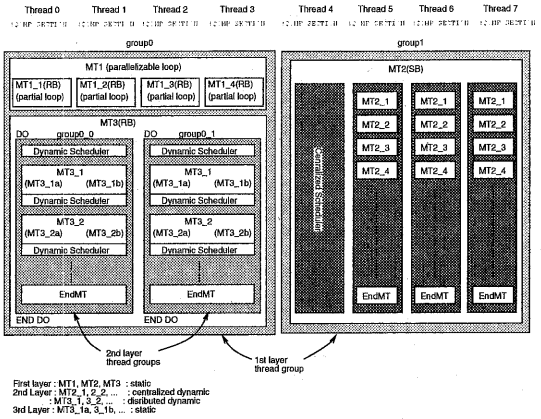

図 6: 生成される OpenMP コードイメージ

り当てられており，各セクション毎に割り当てられたマクロタスク用のコードがコンパイラによって生成される．

サブルーチンブロックである MT2 の内部には第 2 階層が定義され，内部をさらにマクロタスクに分割し，集中ダイナミックスケジューリングを適用し，グループ内の 4 スレッドによって粗粒度タスク並列処理が階層的に行なわれる．集中スケジューリングの場合，集中スケジューラとなるスレッドのセクションにはスケジューリングルーチンが生成され，それ以外のスレッドのセクションには，マクロタスクのコードとスケジューラからの通知を待ち，割り当てられたタスクの実行へ移るためのルーチンが生成される．

一方，繰り返しブロックである MT3 の内部にも第 2 階層が定義され，グループ内の 4 スレッドをさらに 2 スレッドずつにグループ化して，分散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が行なわれる．分散ダイナミックスケジューリングでは，各スレッドがスケジューリングとマクロタスクの実行を行うので，各セクションにはマクロタスクのコードとスケジューリングルーチンが図のように生成される．

並列処理可能ループである MT1 の処理は，グループ内の 4 スレッドに分割されている．

また，生成された OpenMP FORTRAN プログラムは，ターゲットマシン上のネイティブコンパイラによってコンパイルされ，実行される．したがって，OpenMP Backend を用いることによって，OSCAR FORTRAN コンパイラはシーケンシャルな FORTRAN プログラムから，OpenMP で並列化された FOTRAN プログラムを生成する並列化プリプロセッサとして動作する．

5 性能評価

本章では，本論文で提案する手法の共有メモリマルチプロセッサ上での性能評価について述べる．

5.1 評価環境

評価には IBM RS6000 SP 604e HighNode を使用した．この SMP サーバは，1 プロセッサ当たり 32KB の命令，データ L1 キャッシュと 1MB の L2 キャッシュ，1GB の共有主メモリを持つ．

また，評価プログラムには，SPEC 95fp の swim, mgrid を用いた．swim は shallow water equation の求解プログラムであり，mgrid は 3 次元の Multi-grid solver である．

表 1: swim の実行時間と速度向上率

PE	XLF Compiler		OSCAR Compiler	
	time [sec]	speedup	time [sec]	speedup
1	524.3	1.00	443.0	1.18
2	282.9	1.85	220.6	2.38
4	185.6	2.84	112.7	4.65

本評価では，OSCAR FORTRAN コンパイラによって生成された OpenMP FOTRAN プログラムを IBM XL FORTRAN コンパイラ version 6.1 によってコンパイルし実行させた．使用した XL FORTRAN コンパイラのコンパイルオプションは"-qsmp=noauto -O3 -qhot -qtune=auto -qarch=auto -qcache=auto -qundef -qmaxmem=-1 -qstrict" である．

また，比較対象として，IBM XL FORTRAN コンパイラのみを用いて，自動並列化を行なった．この時のコンパイルオプションは"-qsmp=auto -O3 -qhot -qtune=auto -qarch=auto -qcache=auto -qmaxmem=-1 -qstrict" である．

また，逐次実行の場合の実行時間を計測する際のコンパイルオプションは，"-O3 -qhot -qtune=auto -qarch=auto -qcache=auto -qmaxmem=-1 -qstrict" を用いた．

5.2 評価結果

5.2.1 swim

swim の実行時間のほとんどは，メインループから呼ばれる，calc1,calc2,calc3 の三つのサブルーチンで占められる．今回の評価では，これらのサブルーチンにキャッシュ最適化を組み込んだ粗粒度タスク並列処理を適用した．スケジューリング方式は，プロセッサ数が少ないことから，全プロセッサをタスク実行に使える分散ダイナミックスケジューリングを用いた

swim の実行時間と速度向上率を表 1 に示す．swim を XL FORTRAN コンパイラのみを用いて自動並列化を行なわずにコンパイルしたときの逐次実行時間は，524.3 秒であった．これに対し，OSCAR FORTRAN コンパイラを用いて本手法を適用し，キャッシュ最適化を行なった逐次 FORTRAN プログラムを生成し，XL FORTRAN コンパイラでコンパイル後実行したときの逐次実行時間は 443.0 秒であり，約 18%の速度向上を示した．

マルチプロセッサでの評価では，XL FORTRAN コンパイラで自動並列化をした際の実行時間は 2 プロセッサで 282.2 秒，4 プロセッサで 185.6 秒であり，逐次実行に対する速度向上率はそれぞれ 1.85 倍，2.84 倍であった．これに対して，OSCAR FORTRAN コンパイラを用いて粗粒度タスク並列処理とキャッシュ最適化を行うと，実行時間は 2 プロセッサで 220.6 秒，4 プロセッサで 112.7 秒となり，XL FORTRAN コンパイラのみを用いた逐次実行に対する速度向上率はそれぞれ 2.38 倍，4.65 倍と，プロセッサ台数倍を越える高い性能を得ることができた．

5.2.2 mgrid

mgrid の実行時間の約 70%はサブルーチン resid と psinv で占められる．この二つのサブルーチンにキャッシュ最適化を組み込んだ粗粒度タスク並列処理を適用した．スケジューリング方式は分散ダイナミックスケジューリングを適用した．

mgrid の実行時間と速度向上率を表 2 に示す．mgrid の逐次実行時間は，679.8 秒である．OSCAR Compiler

表 2: mgrid の実行時間と速度向上率

PE	XLF Compiler		OSCAR Compiler	
	time [sec]	speedup	time [sec]	speedup
1	679.9	1.00	638.8	1.06
2	359.9	1.89	326.4	2.08
4	336.7	2.02	168.7	4.03

を用いてキャッシュ最適化を行なったときの逐次実行時間は 638.8 秒となり，約 6.4%の速度向上が得られた．

2 プロセッサ使用時の，XL FORTRAN コンパイラの実行時間は 359.9 秒であり，4 プロセッサでは 336.7 秒であった．速度向上率はそれぞれ 1.89 倍と 2.02 倍であった．OSCAR FORTRAN コンパイラを使用したときは，実行時間は 2 プロセッサで 326.4 秒，4 プロセッサで 168.7 秒であり，速度向上率は 2.08 倍，4.03 倍となった．mgrid でも OSCAR FORTRAN コンパイラはプロセッサ台数を超える速度向上率を得ることができた．

6 まとめ

本論文では，粗粒度タスク並列処理およびさらに性能を向上させるキャッシュ最適化手法について述べた．また，本手法を実装した OSCAR マルチグレインコンパイラを使用して，OpenMP を用いて粗粒度タスク並列処理を実現し，IBM RS6000 SP High Node 上で性能評価を行なった結果について述べた．

本手法を用いたシングルプロセッサでの実行では，粗粒度タスク間のキャッシュ最適化の効果が確かめられ，IBM XL FORTRAN コンパイラのみを用いた逐次実行を swim で 18%上回る性能を示した．

複数プロセッサを使用した場合でも，OSCAR FORTRAN コンパイラは XL FORTRAN コンパイラの自動並列化を上回る性能を示し，4 プロセッサを用いた場合の mgrid で約 2 倍の性能向上が得られ，本手法の有効性が確かめられた．

今後は，データローカライゼーション手法の改良や，他の SMP マシン上での性能評価を行な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).

[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] SUIF: http://suif.stanford.edu/.

[8] 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).

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

[10] 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).

[11] 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).

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

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

[14] NANOS: http://www.cepba.upc.es/nanos/.

[15] KasaharaH, H., Honda, H., Aida, K., Okamoto, M. and Narita, S.: A Multi-grain Parallelizing Compilation Scheme on OSCAR, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).

[16] 岡本, 合田, 宮沢, 本多, 笠原: 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, 情処論, Vol. 35, No. 4, pp. 513–521 (1994).

[17] 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 (1999).

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

[19] 吉田, 前田, 尾形 and 笠原: Fortran マクロデータフロー処理におけるデータローカライゼーション手法, 情処論, Vol. 35, No. 9, pp. 1848–1994 (1994).

[20] 吉田, 八木, 笠原: SMP 上でのデータ依存マクロタスクグラフのデータローカライゼーション手法, 情処 ARC 研究会 (2001.1).

[21] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. 12th Workshop on Languages and Compilers for Parallel Computing* (2000).