計算機アーキテクチャ 149－20
（２００２．８．２３）

SMPシステム上でのOSCARマルチグレイン並列化コンパイラの性能

小幡　元樹[†,††]　　石坂　一久[†,††]
白　子　準[†]　　笠原　博徳[†,††]

早稲田大学[†]　アドバンスト並列化コンパイラ共同体[††]

本論文では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」の一環として開発している OSCAR マルチグレイン並列化コンパイラについて述べ，SMP マシン上での性能を評価する．OSCAR マルチグレイン並列化コンパイラは，オンチップマルチプロセッサからハイエンドサーバに至る様々なシステム上において，従来から利用されてきたループ並列性に加え，ループ・サブルーチン・基本ブロック間の粗粒度並列性と基本ブロック内のステートメント間の近細粒度並列性を階層的に用いることを可能とする．また，メモリアクセスオーバヘッドを軽減するためのデータローカライゼーション手法を用いた，異なるループ間，すなわち粗粒度タスク間にわたるキャッシュ最適化も行うことができる．性能評価では，複数の異なる SMP 上において SPEC CPU95 FP ベンチマークを用いて OSCAR コンパイラの性能を評価した．その結果，16 プロセッササーバ IBM RegattaH では MGRID で逐次処理に対して 10.6 倍の速度向上，8 プロセッササーバ IBM RS6000 604e High Node 上では HYDRO2D で 8.5 倍，また Sun V880 サーバ上で 4 プロセッサを用いた場合，SWIM で 6.0 倍の速度向上を得ることができた．

Performance of OSCAR Multigrain Parallelizing Compiler on SMPs

MOTOKI OBATA[†,††], KAZUHISA ISHIZAKA[†,††], JUN SHIRAKO[†]
and HIRONORI KASAHARA[†,††]

Waseda University[†]　Advanced Parallelizing Compiler Project[††]

This paper describes OSCAR multigrain parallelizing compiler which has been developed in Japanese Millennium Project IT21 ”Advanced Parallelizing Compiler” and its performance on SMP machines. The compiler realizes multigrain parallelization for chip-multiprocessors to high-end servers to hierarchically exploit coarse grain task parallelism among loops, subroutines and basic blocks and near fine grain parallelism among statements inside a basic block in addition to loop parallelism. Also, it globally optimizes cache use over different loops, or coarse grain tasks, based on data localization technique to reduce memory access overhead. Performance of OSCAR compiler for SPEC95fp is evaluated on different SMPs. For example, it gives us 10.6 times for MGRID on 16 processor IBM RegattaH, 8.5 times speedup for HYDRO2D on 8 processor IBM RS6000 604e High Node against sequential processing and 6.0 times speedup for TOMCATV using 4 processors on Sun Fire V880 server.

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をはじめとする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，ハイエンドサーバで広く用いられている．しかし，マルチプロセッサシステムにおけるピーク性能と実効性能の差は，プロセッサ数が増加するつれて広がる一方である．その上，効果的な並列化プログラムを開発するためには，並列処理に関する知識や経験が必要であり，そのための長い習得期間が必要である．したがって，マルチプロセッサシステムにおける実効性能，コストパフォーマンス，使いやすさを向上させ，チップマルチプロセッサを用いる携帯電話・PDA・ゲーム機を含めたシステムオンチップ市場を発達させるためには，強力な自動並列化コンパイラが必要となっている．

従来より，マルチプロセッサシステム用自動並列化コンパイラにおいては，Do-all，Do-across のようなループ並列処理技術が用いられており，研究用コンパイラとしては Polaris コンパイラ [1]，SUIF コンパイラ [2] などの優れた性能を示すコンパイラが開発された．しかし，これらの優れた研究成果によって，ループ並列化技術は飽和状態にあると言われている．したがって，マルチグレイン並列処理のような次世代並列化技術が，ループ並列化の限界を打ち破るために期待されている．

従来より開発を続けている OSCAR コンパイラ [3] におけるマルチグレイン並列処理では，イタレーション間ループ並列性に加え，ループ・サブルーチン・基本ブロック間の粗粒度並列性 [4] と基本ブロック内のステートメント間の近細粒度並列性 [5] を用いている．また，NANOS コンパイラ [6]，PROMIS コンパイラ [7] のようにマルチレベルの並列性を利用しようとしているコンパイラも研究が進んでいる．

共有メモリ型マルチプロセッサシステムの実効性能，使いやすさ，コストパフォーマンスを改善するため，ミレニアムプロジェクト IT21 の一部として，OSCAR コンパイラをベースとした経済産業省／ NEDO アドバンスト並列化コンパイラ（APC）プロジェクト [8] が 2000 年度にスタートした．

本論文では，この OSCAR コンパイラについて述べ，16 プロセッササーバ IBM pSeries690 Regatta H，8 プロセッササーバ IBM RS6000 SP 604e High Node，SUN Fire V880 のような商用 SMP システム上において，OSCAR コンパイラの OpenMP バックエンドを用いた際の性能評価を行う．

本評価で使用される OSCAR コンパイラの OpenMP バックエンドは，階層的粗粒度タスク並列処理を実現するワンタイム・シングルレベルスレッド生成手法及びデータローカライゼーションによるグローバルキャッシュ最適化技術を用い，OpenMP コードを出力する．この OpenMP

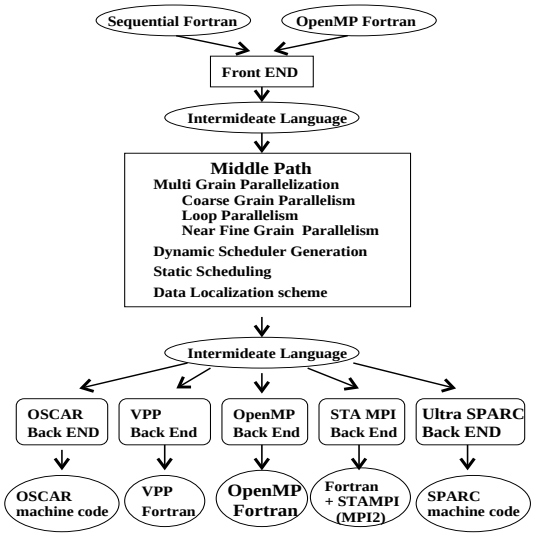


図 1: OSCAR 並列化コンパイラの構成

コードは各マシン上のネイティブコンパイラでオブジェクトに変換され，実行される．

2 OSCAR マルチグレイン並列化コンパイラ

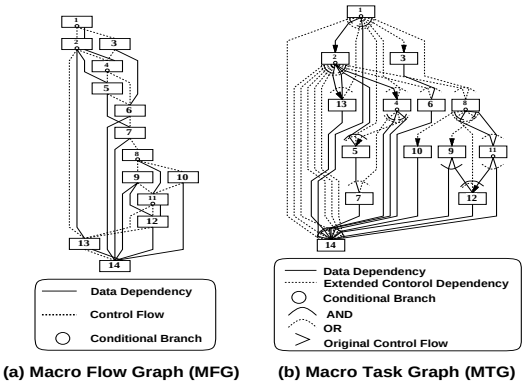
OSCAR コンパイラは，プログラム全域にわたるマルチグレイン並列性，すなわち粗粒度並列性，ループ並列性，近細粒度並列性を利用する．図 1 に示すように，OSCAR コンパイラは Fortran フロントエンド，マルチグレイン並列性を抽出するミドルパス，バックエンドから構成される．バックエンドとしては，UltraSPARC チップマルチプロセッサ，OSCAR チップマルチプロセッサ [9]，OpenMP API をサポートする SMP システム用バックエンド [10] や MPI をサポートするクラスタシステム用の各バックエンドを研究・開発している．

本論文で扱う SMP システム用マルチグレイン並列処理では，まずコンパイラはループ，サブルーチン，基本ブロックの 3 種類の粗粒度タスク，すなわちマクロタスクを生成し，コントロール，データ依存解析結果を元にした最早実行可能条件解析 [4] によってマクロタスク間の粗粒度並列性を解析する．

また，キャッシュや分散共有メモリ型マルチプロセッサシステムに対する最適化を行う場合は，ループ整合分割 [11] を用いてマクロタスクとデータを分割する．生成されたマクロタスクは，スタティックもしくはダイナミックにプロセッサやプロセッサをグループ化したプロセッサクラスタにスケジューリング [12] される．これらのスタティック及びダイナミックスケジューリングを実現する並列プログラムは，ワンタイム・シングルレベルスレッド生成を用いた OpenMP として生成される [10]．

2.1 マクロタスク生成

まず OSCAR コンパイラは，入力プログラムから擬似代入文ブロック 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を生成する．生成されたブロックが逐次処理繰り返しブロックやサブルーチンブロックの場合は，そのボディ部を階層的に粗粒度タスク分割する．もし繰り返しブロック（RB）が並列化可能ループである場合は，利用可能プロセッサ数やキャッシュサイズなどを考慮して，ループをイタレーション方向に分割した部分ループを生成する．これらの部分ループは異なるマクロタスクとして定義され，並列に実行される．



(a) Macro Flow Graph (MFG)　(b) Macro Task Graph (MTG)

図 2: マクロタスクグラフとマクロフローグラフ

2.2 マクロフローグラフとマクロタスクグラフ

マクロタスク生成後，各階層におけるマクロタスク間のデータ依存とコントロールフローを階層的に解析し，図 2（a）に示すようなマクロフローグラフ（MFG）として表現する．図 2（a）においては，ノードはマクロタスク，実線エッジはマクロタスク間のデータ依存，点線エッジはコントロールフローをそれぞれ表す．ノード内の小円はそのマクロタスクが条件分岐ブロックであることを表す．マクロフローグラフ内の各エッジの矢印は省略しているが，全て下向きであると仮定している．

マクロフローグラフからマクロタスク間の並列性を抽出するため，コンパイラは各マクロタスクの最早実行可能条件を解析する．最早実行可能条件とは，マクロタスクが最も早く実行可能になるための条件を表し，全マクロタスクの最早実行可能条件は図 2（b）に示すマクロタスクグラフ（MTG）として表される．マクロタスクグラフにおいても，ノードはマクロタスク，ノード内の小円は条件分岐をそれぞれ表している．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コントロール依存を表している．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の他に，データ依存している先行マクロタスクが実行されない条件を含むことを意味している．また，実線アークは，アークで束ねられたエッジが AND 関係にあること，点線アークは OR 関係を表す．マクロタスクグラフにおいても，各エッジの矢印は省略するが，矢印を持つエッジは，マクロフローグラフにおけるオリジナルの条件分岐方向を表している．

2.3 並列処理階層の自動決定

マクロタスクグラフ生成後，抽出された粗粒度並列性を効果的に利用するため，各マクロタスクのコストと MTG のクリティカルパス長から，各階層のマクロタスクグラフの並列度を算出する [13]．この際，並列処理可能ループを粗粒度タスク分割することにより，ループ並列性を粗粒度並列性に変換した並列度を算出し，プログラム全域の粗粒度並列性とループ並列性を考慮した総合的な粗粒度並列性を算出する．

マクロタスクのコストを求めるために，OSCAR コンパイラはコンパイル時に，基本ブロック内ステートメントのコストを推定し，その合計をマクロタスクの処理コス

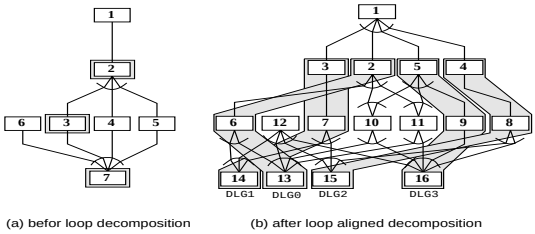


図 3: ローカライゼーションの例

トとする．マクロタスクがコンパイル時に繰り返し回数を決定できないループである場合は，マクロタスク内の配列宣言サイズよりループ回転数を推定する．また，条件分岐マクロタスクがある場合は，分岐確率を 50%としてマクロタスクグラフの処理コストを推定する．

こうして得られた並列度をもとに，各階層のマクロタスクグラフに適したプロセッサクラスタと，その内部のプロセッサを割り当てることによって，並列性を考慮した階層的自動マルチグレイン並列処理が行われる [13]．

2.4 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

もし同じデータにアクセスするマクロタスクが同じプロセッサ上でできるだけ連続に実行されるなら，キャッシュ，分散共有メモリ，ローカルメモリといったプロセッサに近い高速なメモリを用いて，マクロタスク間でデータを転送することができる．そのようなタスク割り当てを実現するため，データローカライゼーション手法 [11] が提案されており，コンパイラが生成するパーシャルスタティックタスクスケジューラによって同じデータにアクセスするマクロタスクの同じプロセッサでの連続実行を可能とする．

キャッシュミスを最小化するため，大きなデータを扱うループに対してループ整合分割（LAD）[11] を適用する．各ループの使用データサイズをキャッシュサイズよりも小さくするため，ループ整合分割によってループ回転数が小さい部分粗粒度ループブロックを生成する．次に，マクロタスクグラフ上においてデータ依存関係にある部分ループが，データローカライゼーショングループ（DLG）[11] としてグループ化され，スタティックもしくはダイナミックスケジューラによって同一プロセッサに割り当てられる．

図 3（a）では，マクロタスク 2，3，7 が同じデータを扱う並列処理可能ループであり，データサイズはキャッシュサイズよりも大きいと仮定している．この例では，各ループは LAD によって 4 つの並列処理可能ループに分割される．例えば，図 3（a）におけるマクロタスク 2 は，図 3（b）におけるマクロタスク 2~5 に分割される．この場合，共有データを持つマクロタスク集合であるデータローカライゼーショングループは，図 3（b）のマクロタスク（2,6,13），(3,7,14），(4,8,15），(5,9,16）の組であり，これらの組中のマクロタスクを可能な限り連続実行することにより，複数ループにわたるキャッシュ最適化が実現できる．

2.5 マクロ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，プロセッサもしくはプロセッサクラスタにマクロタスクを割り当てるために，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手法が用いられる．マクロタスクグラフや，同期オーバヘッド，データ転送オーバヘッドなどのターゲットマシンパラメータを考慮して適切なスケジューリング手法が選択される．

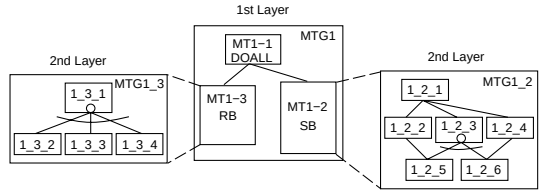


図 4: サンプルマクロタスクグラフ

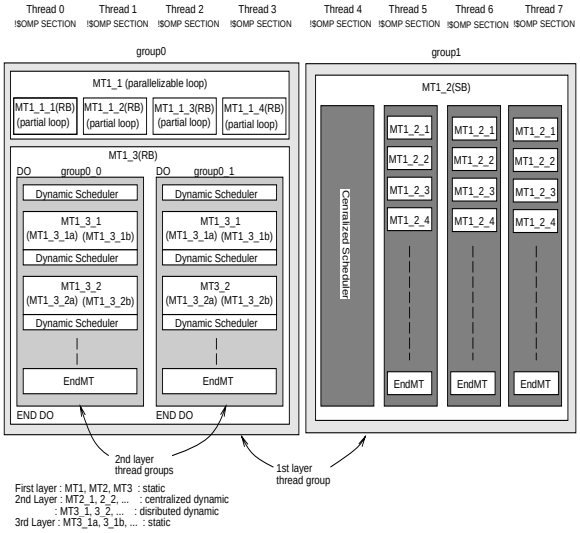


図 5: 8 スレッド用 OpenMP コード生成イメージ

一方，マクロタスクグラフに条件分岐などの実行時不確定性がある場合は，実行時にプロセッサへのタスク割り当てを決定するダイナミックスケジューリング手法が選択される．ダイナミックスケジューリングルーチンは，コンパイラによって並列化マクロタスクコード中に埋め込まれる．

OSCAR コンパイラで用いられるダイナミックスケジューリング手法は，集中スケジューリングと分散スケジューリングの 2 種類である．集中ダイナミックスケジューリングでは，1 つのプロセッサがスケジューリングルーチンを実行し，他のプロセッサはスケジューラから割り当てられたマクロタスクを実行する．分散ダイナミックスケジューリングでは，スケジューリングルーチンは全プロセッサに分散しており，レディタスクキュー，最早実行可能条件などのスケジューリング情報に対して排他制御を用いてアクセスする．

2.6 OpenMP コード生成

次に，OpenMP コードが生成される．図 4 におけるマクロタスクグラフを 8 スレッドで実行するための生成コードイメージを図 5 に示す．図 5 中では，OpenMP PARALLEL SECTIONS ディレクティブによって 8 スレッドが生成されている．「ワンタイム・シングルレベルスレッド生成」に基づき，プログラムの最初に fork されたスレッドは，プログラムの最後に一度だけ join する．この例では，第 1 階

層はスタティックスケジューリングされており，8 スレッドがそれぞれ 4 スレッド（プロセッサに対応）ずつの 2 スレッドグループ（プロセッサクラスタに対応）になっている．マクロタスク 1_1 と 1_3 はスレッドグループ 0，マクロタスク 1_2 はスレッドグループ 1 に割り当てられている．スタティックスケジューリング適用時は，図 5 のようにスタティックスケジューリング結果にしたがって，コンパイラは各 OpenMP SECTION に異なるコードを生成する．スレッドグループに割り当てられたマクロタスクは，スレッドグループ内のスレッドによって階層的に並列処理される．図 4 のマクロタスク 1_2 はスレッドグループ 1 に割り当てられ，集中ダイナミックスケジューリング手法を用いて並列処理される例を示している．この例では，スレッド 4 が集中スケジューラとなり，スレッド 5~7 は，集中スケジューラにより割り当てられた，マクロタスク 1_2 内のサブマクロタスク 1_2_1，1_2_2 などを実行する．一方，マクロタスク 1_3 は分散ダイナミックスケジューリングの例である．ここでは，マクロタスク 1_3 はサブマクロタスク 1_3_1~1_3_4 に分割され，4 スレッドを利用可能なスレッドグループ 0 をそれぞれ 2 スレッドとなるよう階層的に定義されたスレッドグループ 0_0 と 0_1 に割り当てられる．分散ダイナミックスケジューリングルーチンは図 5 に示すように各マクロタスクに挿入される．これらのスケジューリング方式は，使用されるマシンやプログラムの並列度などを考慮してコンパイラが選択することができる．

3 性能評価

本章では，SPEC95fp ベンチマーク中のプログラムを用いて，IBM Regatta H（16 プロセッササーバ），IBM RS6000 SP 604e High Node（8 プロセッササーバ），Sun Fire V880（8 プロセッササーバ，本評価では 4 プロセッサを使用）上での OSCAR マルチグレイン並列化コンパイラの性能評価を行う．ここでは SPEC95fp から TOMCATV, SWIM, SU2COR, HYDRO2D, MGRID, APPLU, TURB3D の 7 本のプログラムを用いる．本評価では，SU2COR と TURB3D に対しては，インライン展開や変数リネーミングといった簡単なリストラクチャリングを手動で適用したコードを，OSCAR コンパイラの評価とネイティブコンパイラの評価の双方に用いた．V880 における SWIM の評価においては，インライン展開を適用したコードを評価に用いた．

また本評価では，各 SMP 上で native コンパイラが最小の逐次処理時間，並列処理時間を与えるコンパイルオプションを選択した．また，OS あるいはランタイムライブラリのチューニングなどのコンパイルオプション以外のパラメータチューニングは行っていない．

3.1 IBM Regatta H での評価

まず，IBM Regatta H 上での評価結果について述べる．IBM Regatta H は 8 つの Power4 プロセッサ，つまり 16 プロセッサ利用可能なハイパフォーマンス SMP サーバである．Regatta H での評価においては，ネイティブ自動ループ並列化コンパイラとして IBM XL Fortran for AIX Version 7.1 を用いた．OSCAR コンパイラの評価においては，OSCAR コンパイラが出力した OpenMP コードを XL Fortran コンパイラでコンパイルした．XL Fortran の逐次コンパイルオプションとして “-O5 -qhot -qarch=pwr4” を，自動ループ並列化オプションとして “-O5 -qsmp=auto -qhot -qarch=pwr4” を用いた．また，OSCAR コンパイラ

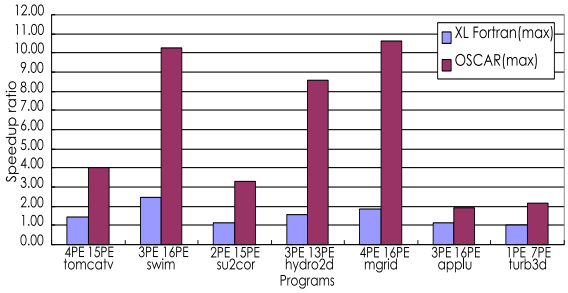


図 6: Regatta 上での性能（最大 16PE 使用時）

によって生成された OpenMP コードをコンパイルする際のオプションは “-O5 -qsmp=noauto -qhot -qarch=pwr4” である．ただし，SU2COR と TURB3D については最適化レベルとして-O4 を用いている．

図 6 に，16 プロセッサまで用いた際の RegattaH における OSCAR コンパイラ，XL Fortran コンパイラの最大性能値を示す．横軸はアプリケーション名，縦軸は XL Fortran による逐次処理に対する速度向上率である．各プログラムにおいては，左側のグラフが XL Fortran の最大性能，右側のグラフは OSCAR コンパイラの性能である．各グラフ下部の PE 数は，その性能を計測した PE 数である．

Regatta 上で OSCAR コンパイラをプリプロセッサとして使うことにより，XL Fortran コンパイラの性能を最も大きく向上させることができたプログラムは MGRID であり，OSCAR コンパイラは逐次性能に対して 16 プロセッサで 10.6 倍，XL Fortran の最大性能に対して 5.7 倍の性能向上を示した．また，TOMCATV では 15 プロセッサで逐次性能の 4.0 倍，SWIM では 16 プロセッサで逐次性能の 10.3 倍，SU2COR では 15 プロセッサで逐次性能の 3.3 倍，HYDRO2D では 13 プロセッサで逐次性能の 8.6 倍，APPLU では 16 プロセッサで逐次性能の 1.9 倍，TURB3D では 7 プロセッサで逐次性能の 2.2 倍の性能向上を得ることができた．また，OSCAR コンパイラをプリプロセッサとして使うことにより，SWIM における XL Fortran コンパイラの性能を 16 プロセッサで 5.7 倍向上させることができた他，TOMCATV では 15 プロセッサで 2.8 倍，SWIM では 16 プロセッサで 4.1 倍，SU2COR では 15 プロセッサで 2.9 倍，HYDRO2D では 13 プロセッサで 5.4 倍，APPLU では 16 プロセッサで 1.7 倍，TURB3D では 7 プロセッサで 2.2 倍，XL Fortran の性能を向上させることができた．

TOMCATV，SWIM，HYDRO2D，MGRID はループ並列性に優れたプログラムであり，並列性の抽出状況は OSCAR コンパイラ，XL Fortran 共に同様であった．これらのプログラムにおける OSCAR コンパイラの性能向上の要因は，Regatta 上でのスレッド管理オーバヘッドが大きいため，ワンタイム・シングルレベルスレッド生成手法を用いたスレッドスケジューリングオーバヘッド軽減効果が大きいことによる．例えば，XL Fortran による自動ループ並列化コンパイルを行った TOMCATV では，最小の実行時間が得られる 4 スレッド（プロセッサ）使用時の各スレッド上での処理時間は，マスタースレッドが全 CPU 時間の 58%を占め，残りの 3 スレーブスレッドがそれぞれ 14%というアンバランスな負荷分散が行われていることから，ロードバランスが悪く，スレッドスケジューリングが適切に行われていないことがわかる．他のプログラムについても同様の傾向が示されている．

SU2COR，TURB3D では，並列処理階層の自動決定手

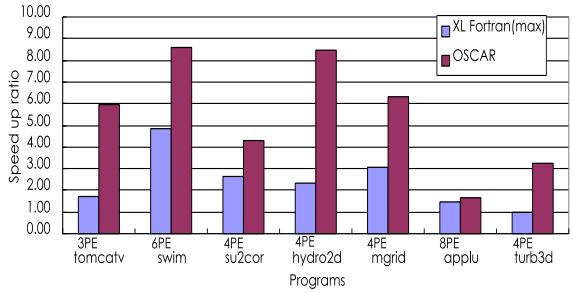


図 7: RS6000 604e 上での性能（最大 8PE 使用時）

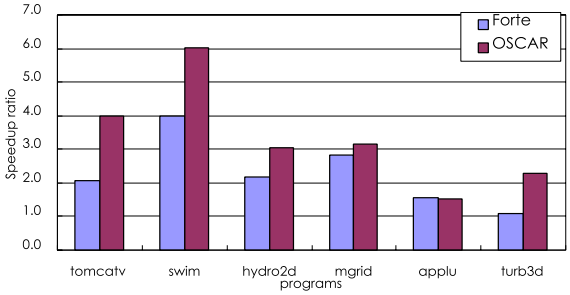


図 8: V880 上での性能（4 プロセッサ）

法により粗粒度並列性を抽出でき，複数 PC 構成を用いた粗粒度並列処理により性能向上が得られている．OSCAR コンパイラは，SU2COR では，サブルーチン LOOPS 内のループ DO400 の最内側を分散ダイナミックスケジューリングによって粗粒度並列処理している．また，TURB3D はサブルーチン TURB3D のナチュラルループ内の粗粒度並列性を利用している．ただし，この TURB3D では，並列処理階層決定手法の改善やダイナミックスケジューリングオーバヘッドの軽減により，今後のより大きな性能向上が期待できる．

また APPLU では，サブルーチン BUTS，BLTS のパイプライン並列性を用いないと大きな性能向上が期待できないが，現在の OSCAR コンパイラ，XL Fortran コンパイラの双方とも，この並列性を利用できていないため，逐次処理に対しては，ほとんど性能が向上しない．ただし，OSCAR コンパイラでは，並列処理階層自動決定によって，並列性が全くない箇所と少ないながらも並列処理可能な箇所を適切に判断できるため，16 プロセッサ使用時の性能で 1.7 倍の性能向上を得ることができた．

3.2 RS6000 SP 604e High Node での評価

次に，IBM RS6000 SP 604e High Node は 8 プロセッサ SMP サーバでの評価について述べる．RS6000 上での評価においても，ネイティブ自動ループ並列化コンパイラとして IBM XL Fortran for AIX Version 7.1 を用いた．XL Fortran による逐次処理コンパイルでは，オプションとして “-O5 -qhot -qarch=ppc -qtune=auto -qcache=auto” を用い，XL Fortran による自動ループ並列化オプションとしては “-O5 -qsmp=auto -qhot -qarch=ppc -qtune=auto -qcache=auto” を用いた．また，OSCAR コンパイラによって生成された OpenMP コードをコンパイルする際には，“-O3 -qsmp=noauto -qhot -qarch=ppc -qtune=auto -qcache=auto” を用いた．

図 7 に RS6000 SP 上での速度向上率を示す．図 6 同様，各グラフ下部の数値は使用プロセッサ数であるが，図 7 では，OSCAR コンパイラの性能は全て 8 プロセッサを用いた際の性能である．これは，XL Fortran コンパイラは 8 プロセッサよりも少ないプロセッサで最小実行時間，すなわち最大性能を示すことが多いが，OSCAR コンパイラは常に 8 プロセッサでの実行時間が最小であったためである．

図 7 において OSCAR コンパイラと XL Fortran で最も性能差があったのは HYDRO2D であり，OSCAR コンパイラをプリプロセッサとして使うことにより，逐次性能に対して 8.5 倍の性能向上を示し，XL Fortran の性能を 3.7 倍向上させることができた．また，TOMCATV では逐次性能の 5.9 倍，XL Fortran の 3.5 倍，SWIM では逐次性能の 8.6 倍，XL Fortran の性能を 1.8 倍，SU2COR では逐次性能の 4.3 倍，XL Fortran を 1.7 倍，MGRID では逐次性能の 6.3 倍，XL Fortran を 2.1 倍，APPLU では逐次性能の 1.7 倍，XL Fortran を 1.2 倍，TURB3D では逐次性能の 3.3 倍，XL Fortran を 3.3 倍性能向上させることができた．

表 1: V880 における L2 キャッシュミス率（単位：%）

program	Forte	OSCAR
tomcatv	20.9	3.8
swim	11.3	2.2
hydro2d	4.5	1.3
mgrid	3.9	3.0
applu	17.9	17.2
turb3d	10.6	15.9

全体的な傾向は，同じコンパイラを用いた Regatta 上での評価と同様である．OSCAR コンパイラと XL Fortran の性能差の主な要因は，低オーバヘッドで適切なタスクスケジューリングを行うことができるワンタイム・シングルレベルスレッド生成手法の効果，並列処理階層の自動決定による適切な並列処理階層の選択による効果である．

3.3 V880 上での評価

Sun V880 は UltraSPARCIII プロセッサを 8 プロセッサ搭載した SMP サーバである．ネイティブコンパイラである Forte 6 Update 2 による逐次処理コンパイルには “-fast” を用いた．また，Forte による自動ループ並列化オプションとしては，“-fast -parallel -reduction -stackvar” を用い，OSCAR コンパイラによる OpenMP コードを Forte でコンパイルする際のオプションとしては，“-fast -mp=openmp -explicitpar -stackvar” を用いた．V880 では，評価プログラム中 TOMCATV と SWIM を用いたキャッシュ最適化の予備評価も同時に行ったが，V880 用のアジャスタメントがまだ実現できていないため，用いたコードは V880 のキャッシュ構成に最適化されたものではなく，Sun Ultra80 用に最適化されたコードである．

図 8 に Sun V880 における 4 プロセッサ使用時の性能を示す．TOMCATV では，OSCAR コンパイラは逐次性能に対して 4.0 倍の性能向上を示し，同じ 4 プロセッサを用いた Forte コンパイラの性能を 1.9 倍向上させることができた．SWIM では逐次処理に対し 6.0 倍，HYDRO2D では 3.1 倍，MGRID では 3.2 倍，APPLU では 1.5 倍，TURB3D では 2.3 倍の性能向上が得られた．また，V880 上でも OSCAR コンパイラをプロプロセッサとして使うことにより，Forte コンパイラの性能を，SWIM では 1.5 倍，HYDRO2D では 1.4 倍，MGRID では 1.1 倍，APPLU では 1.0 倍，TURB3D では 2.1 倍，それぞれ向上させる

ことができた．

V880 上での評価結果は，プロセッサを 4 台しか用いなかったこともあり，Regatta，RS6000 と比べ性能向上が小さくなっているが，TOMCATV，SWIM，TURB3D の 3 プログラムでは大幅な性能向上が見られる．TOMCATV，SWIM における性能差は，用いたコードは V880 用ではなく 4 プロセッサワークステーション Sun Ultra80 のキャッシュ用に最適化されたものであるが，データローカライゼーションによるキャッシュ最適化により得られたものである．表 1 に各プログラムごとの L2 キャッシュミス率を示す．例えば，TOMCATV では，Forte コンパイラでの L2 キャッシュミス率が 20.9%であるのに対して，OSCAR コンパイラでを用いた場合，L2 キャッシュミス率は 3.8%に低下している．また，SWIM においても同様の傾向を示しており，Forte では 11.3%であった L2 キャッシュミス率が OSCAR コンパイラでは 2.2%となっている．また，図 8 において，Forte と OSCAR であまり性能差のない HYDRO2D，MGRID，APPLU においては，キャッシュミス率においてもほとんど差がない結果となっている．また TURB3D では，表 1 では Forte の L2 ミス率の方が低いが，3.1 節で述べた並列処理階層の自動決定手法による粗粒度並列性の抽出による効果によって，全体性能では OSCAR コンパイラが 2 倍の性能向上を達成している．

これらの結果より，OSCAR コンパイラを用いることによって，3 種の SMP サーバにおいて，データローカライゼーションによるキャッシュ最適化やワンタイム・シングルレベルスレッド生成によるスレッド管理オーバヘッド軽減を含めたマルチグレイン並列処理によって大きな性能向上を得られることが確認できた．また，OpenMP コードを用いた並列化コードを出力するためポータビリティに優れており，容易に他の SMP 上で優れた性能が得られることが確認された．

4 まとめ

本論文では，ミレニアムプロジェクト IT21 の一部である経済産業省／ NEDO アドバンスト並列化コンパイラプロジェクト（APC）の一環として開発中の OSCAR マルチグレイン並列化コンパイラの性能について述べた．OSCAR コンパイラは粗粒度，ループ，近細粒度並列性を抽出し，チップマルチプロセッサ，SMP ワークステーション，SMP ハイエンドサーバ上で，ワンタイム・シングルレベルスレッド生成やデータローカライゼーションによるグローバルキャッシュ最適化を実現する．

SPEC95fp 中の 7 プログラムを用いた性能評価では，16 プロセッサ SMP サーバ IBM pSeries690 Regatta H，8 プロセッサ SMP サーバ IBM RS6000 SP 604e High Node，8 プロセッサ SMP サーバ Sun V880 の 3 種類の SMP を用いて，評価を行った．

Regatta では，MGRID による評価でネイティブコンパイラ XL Fortran の性能を 5.7 倍，RS6000 では HYDRO2D を用いた評価で XL Fortran の性能を 3.7 倍，V880 では TOMCATV による評価でネイティブコンパイラ Forte 6 Update 2 の性能を 1.9 倍向上させることができた．

OSCAR コンパイラを OpenMP コード生成用プリプロセッサとして用いることにより，IBM の 2 種の新旧 SMP サーバ，Sun の最新サーバ上では，マルチグレイン並列性の抽出，キャッシュ最適化，ワンタイムシングルレベルスレッド生成における低オーバヘッドスレッド並列処理により優れた性能向上を得られることが確かめられた．また，OpenMP コード出力によるプラットフォームフリー性（高ポータビリティ）も確認された．

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 の一部であるアドバンスト並列化コンパイラプロジェクト（http://www.apc.waseda. ac.jp）および早稲田大学特定課題 No.2000A-154 により，行われた．

参考文献

[1] 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).

[2] 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).

[3] 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).

[4] 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).

[5] 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).

[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] : http://www.apc.waseda.ac.jp/.

[9] Kimura, K. and Kasahara, H.: Near Fine Grain Parallel Processing Using Static Scheduling on Single Chip Multiprocessors, *Proc. IWIA'99, IEEE press* (1999).

[10] 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).

[11] Yoshida, A., Yagi, S. and Kasahara, H.: A Data Localization Scheme for Coarse Grain Task Parallel Processing on Shared Memory Multiprocessors, *Proc. of IEEE International Workshop on Advanced Compiler Technology for High Performance and Embedded Systems*, pp. 111–118 (2001).

[12] Ishizaka, K., Obata, M. and Kasahara, H.: Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor, *Proc. 14th LCPC2001* (2001).

[13] 白子準, 神長浩気, 近藤巧章, 石坂一久, 小幡元樹, 笠原博徳: 並列処理階層自動決定手法を用いた粗粒度タスク並列処理, 情報処理学会研究報告 ARC2002-148-4 (2002).