共有メモリ型マルチプロセッササーバー上における OSCAR マルチグレイン自動並列化コンパイラの性能評価

白子　準[†]　宮本孝道[†]　石坂一久[†]
小幡元樹[††]　木村啓二[†]　笠原博徳[†]

マルチプロセッサシステムの普及に伴い，実効性能，システム価格性能比，ソフトウェア生産性向上のため高性能な自動並列化コンパイラの重要性が高まっている．しかしながら並列処理技術において広く利用されているループ並列処理手法は既に成熟期に至り，今後の大幅な性能向上実現のためには従来とは異なる並列化手法の利用が必須である．本論文ではループ並列処理に加え，基本ブロック，ループ，サブルーチンといった粗粒度タスク間の並列性を利用する粗粒度タスク並列処理・基本ブロック内ステートメントレベルの並列性を用いる近細粒度並列処理によりプログラム全域にわたる並列化を行う OSCAR マルチグレイン自動並列化コンパイラの性能評価について述べる．OSCAR コンパイラではプログラムの形状や並列性に応じた適切な処理プロセッサ数や各並列処理粒度の決定，複数のループや粗粒度タスク間にまたがる広域的なキャッシュメモリ最適化技術が実現されている．SPEC95FP を用いた本性能評価において OSCAR コンパイラは，IBM pSeries690 Power4 24 プロセッササーバ上で IBM XL Fortran コンパイラ 8.1 の自動並列化性能に比べ平均 4.78 倍，SGI Altix3700 Itanium2 16 プロセッササーバ上において Intel Fortran Itanium Compiler 7.1 に比べ平均 2.40 倍，Sun Fire V880 Ultra SPARC III Cu 8 プロセッササーバ上において Sun Forte コンパイラ 7.1 に比べ平均 1.90 倍の性能向上が得られた．

Performance of OSCAR Multigrain Parallelizing Compiler on Shared Memory Multiprocessor Servers

Jun Shirako,[†] Takamichi Miyamoto,[†] Kazuhisa Ishizaka,[†]
Motoki Obata,[††] Keiji Kimura[†] and Hironori Kasahara [†]

The needs for automatic parallelizing compilers are getting larger with widly use of multiprocessor systems. However, the loop parallelization techniques are almost matured and new generation of parallelization methods like multi-grain parallelization are required to achieve higher effective performance. This paper describes the performance of OSCAR multigrain parallelizing compiler that uses the coarse grain task parallelization and the near fine grain parallelization in addition to the loop parallelization. OSCAR compiler realizes the following two important techniques. The first is the automatic determination scheme of parallelizing layer, which decides the number of processors and parallelizing technique for each part of the program. The other is global cache memory optimization among loops and coarse grain tasks. In the evaluation using SPEC95FP benchmarks, OSCAR compiler gave us 4.78 times speedup compared with IBM XL Fortran compiler 8.1 on IBM pSeries690 Power4 24 processors server, 2.40 times speedup compared with Intel Fortran Itanium Compiler 7.1 on SGI Altix3700 Itanium2 16 processors server, 1.90 times speedup compared with Sun Forte compiler 7.1 on Sun Fire V880 Ultra SPARC III Cu 8 processors server.

1. はじめに

主記憶共有型マルチプロセッサシステムはワークステーションやハイエンドサーバといった HPC 分野から携帯電話，PDA，ゲーム等で使われるチップマルチプロセッサまで幅広く利用されている．このようなマルチプロセッサシステムにおいて高い実効性能を実現するためには，プログラムからの適切なグレイン (粒度) での並列性抽出・キャッシュメモリの最適利用も含めプロセッサ間のデータ転送の最小化が必須であり，これを実現するための自動並列化コンパイラの研究・開発が行われている[1)~3)]．これら研究・開発の大部分はプログラム中のループ部分の並列化を対象としたものであり，現在までに様々なループ並列性解析手法やリストラクチャリング手法が開発されている．イリノイ大学の Polaris コンパイラ[2)] ではシンボリック解析，Array Privatization，実行時データ依存解析，レンジテスト，インタープロシージャ解析といった強力な依存解析手法を用いたループ並列化を実現している．またスタンフォード大学の SUIF コンパイラ[3)] では unimodular transformation や affine partitioning[4),5)] とった種々のループリストラクチャリングを内包した並列化手法を用いることでループレベルの並列性解析とデータローカリティ最適化を総合的に行っている．ループ並列化手法は大きな進歩を遂げたが，現在では既に成熟期に至り今後の大幅な性能向上は見込めないと考えられてい

† 早稲田大学理工学部 コンピュータ・ネットワーク工学科
Department of Computer Science, Waseda University

†† 株式会社日立製作所
Hitachi, Ltd.

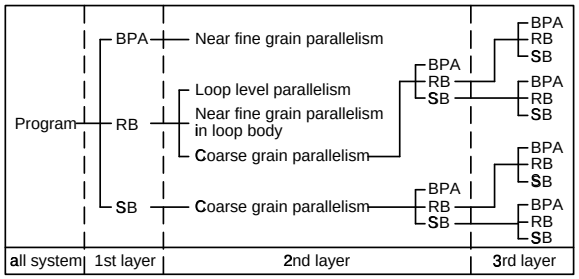

図 1 階層的マクロタスク定義

る．またその一方でマルチプロセッサシステムの大規模化が進み，システムのピーク性能と実際にプログラムを動作させた際の実効性能との性能差が今後さらに広まると予測されている．

これらの問題を克服しマルチプロセッサの更なる実効性能向上のためには，従来のループ並列性に加え，ループ間やサブルーチン間といった粗粒度タスクレベルの並列性や，基本ブロック内での命令・ステートメント間の近細粗粒度並列性などの複数レベルの並列性を利用することが必須である．このようなマルチレベルの並列性を利用するコンパイラには NANOS コンパイラ[6]，PROMIS コンパイラ[7]，そして OSCAR コンパイラ[8)~10)] が挙げられる．カタルーニャ大学の NANOS コンパイラでは，拡張した OpenMP API によって粗粒度タスク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 コンパイラはフロントエンドとバックエンドで共通の中間表現を用いてループ並列性と命令レベル並列性を統合しようとしている．早稲田大学で開発されている OSCAR マルチグレイン自動並列化コンパイラでは，プログラム中の粗粒度タスク並列処理，ループレベル並列処理，近細粒度並列処理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．また OSCAR コンパイラは抽出したマルチグレイン並列性に応じ，プログラムの各所に対する並列性に見合った適切な計算資源 (プロセッサ) の割当てや複数のループすなわち粗粒度タスク間にまたがる広域的なキャッシュメモリ最適化を行っている．本論文では OSCAR コンパイラにおいて用いられているこれら技術と性能評価について述べる．

2. マルチグレイン並列処理

本章では，OSCAR コンパイラで実現されているマルチグレイン並列処理について述べる．今回評価に用いたような商用マルチプロセッサシステムでは，高速なプロセッサ間データ通信機構が必要な近細粒度並列処理[11)] は効果が小さく，本論文では粗粒タスク度並列性とループ並列性を用いたマルチグレイン並列処理を行う．

粗粒度タスク並列処理とは逐次プログラムを階層的に粗粒度タスク分割し，生成された粗粒度タスクすなわちマクロタスクをプロセッサエレメント (PE)[12)]，もしくはプロセッサグループ (PG)[12)] 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並列処理手法である．

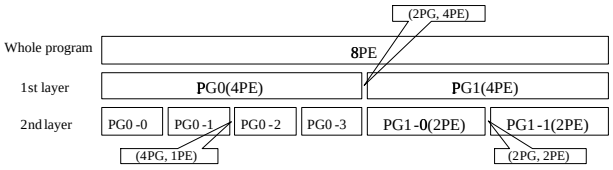

図 2 プロセッサグループ・プロセッサエレメントの階層的定義

2.1 粗粒度タスク生成

粗粒度タスク並列処理では，プログラムは基本ブロックまたはその融合ブロックで構成される疑似代入文ブロック BPA[10)]，DO ループや後方分岐により生じるナチュラルループで構成される繰り返しブロック RB[10)]，サブルーチンブロック SB[10)] の 3 種類のマクロタスク MT[10)] すなわち粗粒度タスクに分割される．繰り返しブロック RB やサブルーチンブロック SB に対しては，その内部をさらにマクロタスク分割し階層的なマクロタスク構造を生成する (図 1)．

2.2 粗粒度タスク並列性抽出

マクロタスク生成後，各階層においてマクロタスク間のデータ依存と制御フローを解析し，マクロタスク間のデータと制御のフローを表すマクロフローグラフ[8),10)] を生成する．

次に，階層的に生成されたマクロフローグラフに対し最早実行可能条件解析[8),10)] を適用し，階層的なマクロタスクグラフ MTG[8),10)] を生成する．ここで最早実行可能条件とは，制御依存とデータ依存を考慮したマクロタスクの最も早く実行を開始してよい条件でありマクロタスクグラフが粗粒度タスク並列性を表す．

2.3 プロセッサグループとプロセッサエレメント

ネストした構造であるマクロタスクグラフを効果的に処理するためにはプロセッサもネストした集合形態をとる必要がある．本手法では，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たプロセッサグループ PG を定義し，このプロセッサグループにマクロタスクグラフ上のマクロタスクを割り当てる．割り当てられたマクロタスクがサブルーチンや繰り返しブロックであり，内部に更にマクロタスクグラフが定義されている場合は，内部マクロタスクグラフに対してプロセッサグループ内のプロセッサを更にグルーピングする．これを繰り返すことでプログラム全域にわたる階層的粗粒度タスク並列性を利用することができる．

図 2 にプロセッサグループ・プロセッサエレメントの階層的な定義の例を示す．図中，第 1 階層では (2PG, 4PE) の構成をとり 2 並列で粗粒度タスク並列処理を行ない，4 PE を割り当てられた第 2 階層ではそれぞれ (4PG,1PE) で 4 並列，(2PG, 2PE) で 2 並列という粗粒度タスク並列処理を行っている．このように階層的なプロセッサ構成を定義することでそれぞれのマクロタスクグラフの並列性に適したグルーピングを行なうことが可能となる．

2.4 プロセッサグループへのマクロタスク割り当て

上述のように各マクロタスクグラフに対して生成されたプロセッサグループがそのマクロタスクグラフ上のマクロタスクを処理する単位となり，次にスケジューラがマ

クロタスクを各プロセッサグループに割り当てる．マクロタスクグラフ上に条件分岐が無い場合はコンパイル時に静的にスケジューリングが行われ各プロセッサグループの処理するマクロタスクが決定される．マクロタスクグラフが条件分岐を含む場合は実行時にスケジューリングを行なう動的スケジューリングルーチンをコンパイラが自動生成し，並列プログラム中に埋め込まれたこのスケジューリングルーチンが実行時にマクロタスクをプロセッサグループに割り当てる．生成されたスタティックスケジューリングコード及び実行時スケジューラはユーザコードであり，OS のシステムコールによるスケジューラに比べ極めて低オーバーヘッドなスケジューリングが可能である．

以下の章においては，MT はマクロタスク，MTG はマクロタスクグラフ，PG はプロセッサグループ，PE はプロセッサエレメントを表わす．また疑似代入文ブロックを BPA，繰り返しブロックを RB，サブルーチンブロックを SB と表記する．

2.5 並列処理階層自動決定手法

抽出された粗粒度タスク，ループレベルといったマルチレベルの並列性をマルチプロセッサシステム上の利用可能なプロセッサに効率よく割当てることが，マルチグレイン並列処理の性能向上において重要である．OSCAR コンパイラでは並列処理階層自動決定手法[12),13)] により，算出した MTG の並列度を用い PG 数・PE 数を適切に決めることで抽出されたマルチグレイン並列性の効果的な利用を実現している．本手法ではまず以下のような並列度を算出する．

- $Para$：粗粒度タスク並列度
- $Para\_ALD$：粗粒度タスク並列性とループ並列性の総合的な並列度
- $Hierarchical\_Para\_max$：当該 MT の内部ネスト構造を考慮した最大並列度

$Para$，$Para\_ALD$ は MTG に対して定義され，$Para$ は対象 MTG の純粋な粗粒度タスク間の並列度，$Para\_ALD$ は並列化可能ループを並列処理の効果がある最小コストとなるよう粗粒度タスク分割した場合の粗粒度タスク間並列度を表す．$\lceil Para\_ALD \rceil$ が総合的な粗粒度タスク並列性 $Para\_ALD$ を利用するのに必要と考えられる PG 数である．また $Hierarchical\_Para\_max$ は MT 内にネストされているマルチグレイン並列性も積算した値で，MT に割当てて効果のある最大のプロセッサ数を表すものとする．

階層的に定義された MTG において上位階層 (ネストレベルの浅い階層) では各 MT の処理コストは下位階層に比べ大きく，並列処理による同期・プロセッサ間通信などのオーバーヘッドは相対的に小さくなる．このため上位階層における並列処理は並列処理オーバーヘッドの小さい効果的な並列化が可能であり，本手法では上位階層の並列性を優先して利用する．一方，並列性がほとんど見られない箇所の並列化は並列処理オーバーヘッドのため，逐次処理時よりも性能が低下する場合がある．このため並列性の小さな箇所では必要なプロセッサ数のみを利用し，不要となるプロセッサには処理を与えずオーバーヘッドの軽減を計る．

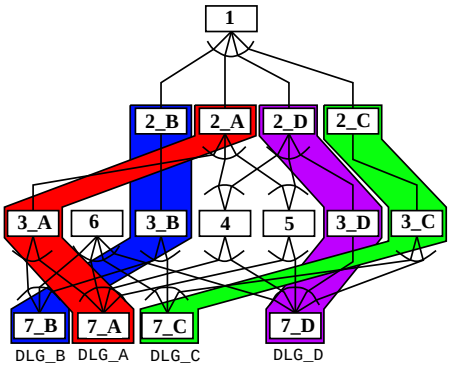

図 3 粗粒度タスク間キャッシュ最適化

注目する MTG_i 上で利用可能な総プロセッサ数を $N_{Avail\_PE_i}$ とし，MTG_i の PG 数 N_{PG_i}，PE 数 N_{PE_i} を以下の手順で決定する．

step.1 並列度を用いた N_{PG_i}・N_{PE_i} の決定

$$\lfloor Para_i + 0.5 \rfloor \leq N_{PG_i} \leq \lfloor Para\_ALD_i + 0.5 \rfloor \quad (1)$$

$$N_{PG_i} \times N_{PE_i} = N_{Avail\_PE_i} \quad (2)$$

(1)，(2) 式を満たすうち N_{PG_i} が最大となる組合わせを N_{PG_i}，N_{PE_i} の解として選ぶ．

step.1 により MTG_i の PG 数 N_{PG_i}，PE 数 N_{PE_i} が決定され，MTG_i 上に存在するマクロタスク MT_{i_j} では N_{PE_i} が利用可能な総プロセッサ数 $N_{Avail\_PE_{i\_j}}$ の上限となる．以下のステップにより，最大の並列度 $Hierarchical\_Para\_max_{i\_j}$ を超えるプロセッサ数が MT_{i_j} に割当てられた場合，超過分のプロセッサには処理を与えず，不要なオーバーヘッドを削除する．

step.2 オーバーヘッド削減のための使用プロセッサ数 $N_{Avail\_PE_{i\_j}}$ 補正

- $Hierarchical\_Para\_max_{i\_j} \geq N_{PE_i}$ の場合
  $N_{Avail\_PE_{i\_j}} = N_{PE_i}$
- $Hierarchical\_Para\_max_{i\_j} < N_{PE_i}$ の場合
  $N_{Avail\_PE_{i\_j}} = Hierarchical\_Para\_max_{i\_j}$

step.1, step2 を全 MTG に再帰的に適用することでプログラム全体の MTG に対して PG 数・PE 数決定を行う．

2.6 粗粒度タスク間キャッシュ最適化

プロセッサとメモリの速度差の拡大により，キャッシュを有効利用することがマルチプロセッサシステムの性能向上に重要となっている．OSCAR コンパイラでは，データローカライゼーション手法を用いた粗粒度タスク間キャッシュ最適化[14)] を行うことで，データを分割し複数粗粒度タスク間でキャッシュ上のデータを効果的に用いることにより，マルチグレイン並列処理の性能を向上させている．

データローカライゼーション手法では，まず複数ループ間のデータ依存を解析し，各ループでアクセスするデータサイズがキャッシュサイズにフィットし，データ依存する分割後の小ループ間でのデータ授受がキャッシュを介して行われるように，それらのループを整合して分割する整合分割[15)] を行う．次に分割されたループのうち同一データにアクセスする複数のループは，データローカラ

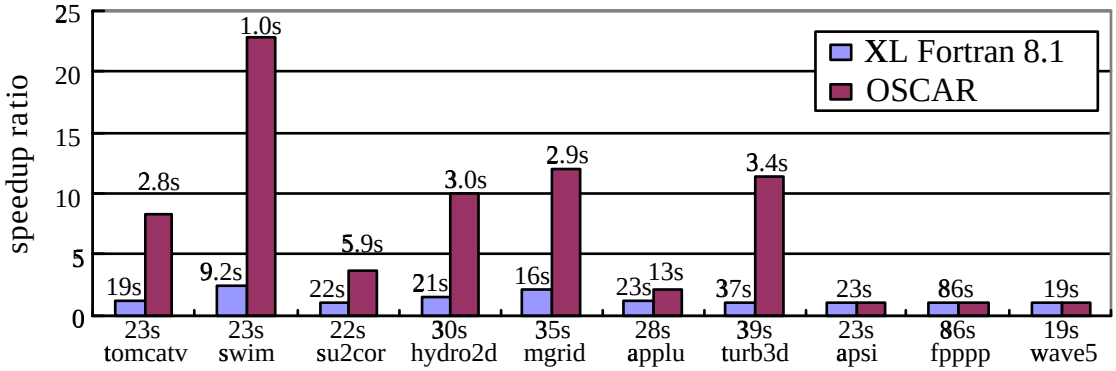

図 4 pSeries690 上での速度向上率

イザブルグループ（DLG）と呼ぶタスク集合にグループ化される．図 3 にループ整合分割を適用したマクロタスクグラフを示す．図中の網掛けで結ばれたマクロタスクが DLG に属するマクロタスクで，この例では DLG_A から DLG_D の 4 つの DLG が定義されている．DLG はキャッシュ上のデータを用いて処理を行えるタスクのグループを表す．グループ内の各タスクは，各グループでアクセスされる総データ量がキャッシュサイズ以下となるように，オリジナル粗粒度タスクから分割され生成されている．

粗粒度タスクスケジューリングでは，粗粒度タスク間の並列性を考慮しながら，同一 DLG に属するループが可能な限り同一プロセッサ上で連続的に実行されるようにスケジューリングを行う．このようにループ分割と連続実行を組み合わせることにより，複数のループにわたりデータをキャッシュから追い出される前に再利用することを可能とすることでキャッシュミスを削減し，タスク間のデータ授受をキャッシュを用いて高速に行うことが可能とする．

さらにコンパイル時にキャッシュ上でのデータレイアウトを推定し，DLG 内ループによってアクセスされる配列間でのキャッシュラインコンフリクトの発生可能性を検出する．データローカライゼーションによって一 DLG 内タスク集合がアクセスするデータ（以下 DLG 内データ）のサイズがキャッシュサイズに収まるように分割されているにも関わらず，ラインコンフリクトが生じる場合は，ダイレクトマップやセットアソシアティブキャッシュの way 数，way サイズを考慮して配列間パディングを行う．このデータレイアウト変換によりコンフリクトミスを削減する[16]．従来のコンパイラによるキャッシュ最適化手法は主に単一ループや融合されたループを対象としているのに対して，本手法ではオリジナルプログラム中で離れた位置にある異なる複数のループ間でのキャッシュ最適化も可能となる．

3. 性 能 評 価

本章では SPEC95FP ベンチマークを用いた OSCAR マルチグレイン自動並列化コンパイラの性能評価について述べる．OSCAR コンパイラの OpenMP バックエンドを用い OpenMP API によって並列化されたプログラムを出力，各マシン用のネイティブコンパイラでコンパイルし実行した．OpenMP バックエンドでは PARALLEL SECTION ディレクティブを用いることでプログラムの開始時にプロセッサ台数分の並列スレッドを生成し，各スレッド内タスクの実行をコンパイラがスタティックに制御するか，生成したスケジューラコードによりダイナミックに制御する．

3.1 pSeries690 上の性能

図 4 に 1.1 GHz の Power4 24 way ハイエンドサーバ IBM pSeries690 上での評価結果を示す．図中，横軸が実行したベンチマークを表し，縦軸が IBM XL コンパイラ ver.8.1 の逐次処理に対する速度向上率を表す．各ベンチマークにおいて左側のバーが XL コンパイラのループ並列化で 24 プロセッサまで用いたうちの最高性能，右側のバーが OSCAR コンパイラの 24 プロセッサまでの最高性能を表す．各ベンチマーク名上の数字は XL コンパイラによる逐次処理時間であり，バー上の数字は並列処理時間である．

tomcatv, swim, hydro2d, mgrid はループ並列性の高いベンチマークであるが，図 4 のように OSCAR コンパイラの並列化性能は XL コンパイラを大きく上回っている．OSCAR コンパイラでは並列処理階層自動決定手法によりプログラム中の各部分に対し，常に全プロセッサを割当てるのではなく並列性に合った処理プロセッサ数を割当てることで，多過ぎるプロセッサの利用により不要な並列処理オーバーヘッドが増え処理速度が低下するといった状況を回避している．また各並列スレッドをコンパイラが生成したスケジューリングコードで制御するため，OS のシステムコールによる制御に比べスケジューリングオーバヘッドを抑制している．このため並列処理時のスレッド管理オーバーヘッドが大きい XL コンパイラに対して，大幅な性能向上が得られた．更に su2cor での処理時間の大きい階層における粗粒度タスク並列性，turb3d の処理時間の大部分を占めるループのループ並列性など XL コンパイラでは利用できなかった並列性が OSCAR コンパイラでは利用可能であった．

OSCAR コンパイラによる並列処理の速度向上率は tomcatv では逐次処理に比べ 8.15 倍 (XL コンパイラの並列処理に比べ 6.76 倍)，swim では 22.8 倍 (9.13 倍)，su2cor では 3.66 倍 (3.66 倍)，hydro2d では 10.0 倍

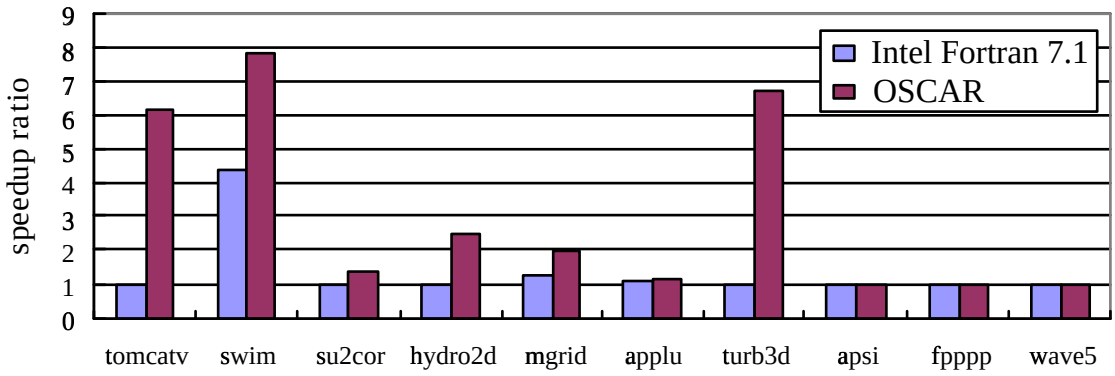

図 5 Altix 上での速度向上率

(6.94 倍)，mgrid では 12.1 倍 (5.65 倍)，applu では 2.12 倍 (1.78 倍)，turb3d では 11.5 倍 (10.9 倍) となった．apsi，fpppp，wave5 では並列化可能ループがないあるいはループサイズが小さいため市販サーバ上での並列化が困難であり，OSCAR コンパイラ，XL コンパイラともに並列処理による速度向上は得られなかった．ただしこれらのアプリケーションも SMP サーバではなくチップマルチプロセッサのようにプロセッサ間データ転送オーバーヘッドが小さくより細かい粒度の並列性利用が可能なシステム上では OSCAR コンパイラによる性能向上が望める[11]．また SPEC95FP 10 本の平均では OSCAR コンパイラの速度向上率は XL コンパイラに比べて 4.78 倍となった．

3.2 Altix3700 上の性能

図 5 に 1.3 GHz の Itanium2 16 way サーバ SGI Altix3700 上での評価結果を示す．図の見方は図 4 と同様であり，Intel Fortran Itanium Compiler 7.1 と OSCAR コンパイラの 16 プロセッサまでの最高性能を表す．この Altix 上での評価においては Altix 用にチューニングしたコードは生成せず，pSeries690 用に生成した OpenMP コードをそのまま実行して評価を行った．

OSCAR コンパイラによる並列処理の速度向上率は tomcatv では逐次処理に比べ 6.13 倍 (Intel コンパイラの並列処理に比べ 6.13 倍)，swim では 7.81 倍 (1.78 倍)，su2cor では 1.37 倍 (1.37 倍)，hydro2d では 2.46 倍 (2.46 倍)，mgrid では 1.94 倍 (1.55 倍)，applu では 1.16 倍 (1.07 倍)，turb3d では 6.73 倍 (6.66 倍) となった．apsi，fpppp，wave5 では pSeries690 と同様な理由で並列処理による速度向上は得られなかった．SPEC95FP 10 本の平均の速度向上率は OSCAR コンパイラが Intel コンパイラに比べて 2.40 倍の性能向上を示した．

これから OSCAR コンパイラが生成する並列化 OpenMP コードは十分なポータビリティがあることが確かめられた．Altix 上での性能に関しては今後 Altix の CC-NUMA アーキテクチャを意識したチューニング行うことでより高い性能が得られると考えられる．

3.3 V880 上の性能

図 6 に Sun Fire V880 Ultra SPARC III Cu 8 way サーバ上での Sun Forte コンパイラ ver.7.1 と OSCAR コンパイラの 8 プロセッサまで用いた場合の最高性能を示す．V880 ではハードウェア，ソフトウェア両方のプリフェッチが使用可能であり，このプリフェッチの有効性を示すため Forte コンパイラのプリフェッチを用いない場合の逐次処理性能に対する速度向上率を表している．左側のバーが Forte コンパイラのプリフェッチを用いた場合の逐次処理性能，中央のバーが Forte コンパイラのプリフェッチを用いた場合の並列処理性能．右側のバーが OSCAR コンパイラのプリフェッチを用いた場合の並列処理性能を表している．

V880 は L2 キャッシュが 2 way セットアソシアティブであり，配列間パディングを含むキャッシュメモリ最適化が性能向上のために重要な要素となる．OSCAR コンパイラにはこれらデータローカライゼーション技術が実装されており，tomcatv, swim, hydro2d, mgrid, turb3d といったループ並列性の高いアプリケーションで Forte コンパイラの並列化性能を大きく上回った．特に swim では 8 プロセッサ使用時に，プリフェッチを用いない逐次処理に対して 32.9 倍 (プリフェッチを用いた逐次処理に対して 18.2 倍) という非常に大きなスーパーリニアスピードアップが得られた．

プリフェッチを用いた OSCAR コンパイラによる並列処理の速度向上率は tomcatv では逐次処理に比べ 12.3 倍 (プリフェッチを用いた Forte コンパイラの並列処理に比べ 2.04 倍)，swim では 32.9 倍 (2.48 倍)，su2cor では 4.71 倍 (1.14 倍)，hydro2d では 7.15 倍 (1.59 倍)，mgrid では 6.94 倍 (1.52 倍)，applu では 1.52 倍 (1.05 倍)，turb3d では 6.72 倍 (6.13 倍) となった．SPEC95FP 10 本の平均では OSCAR コンパイラの速度向上率は Forte コンパイラに比べて 1.90 倍となった．

4. ま と め

本論文では OSCAR マルチグレイン自動並列化コンパイラの共有メモリサーバ上での性能について述べた．本コンパイラはループ並列処理に加え，粗粒度タスク並列処理，近細粗粒度並列処理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．また各粒度の並列性に応じた適切な計算資源割当て及びデータレイアウト変換パディング技術を用いたデータローカライゼーション技術により，マルチグレイン並列処理の性能を向上させている．

各共有メモリサーバ上での SPEC95FP を用いた評

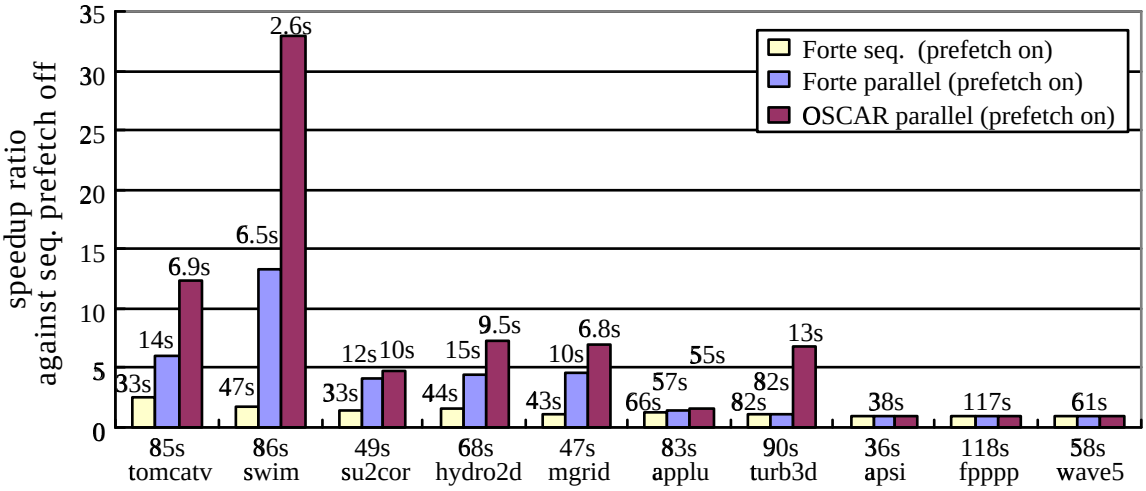

図 6 v880 上での速度向上率

価では，OSCAR コンパイラによる自動並列化は IBM pSeries690 Power4 24 プロセッササーバ上で IBM XL Fortran コンパイラ 8.1 の自動並列化性能に比べ平均 4.78 倍，IBM 用のコードをそのままチューニングなしに SGI Altix3700 Itanium2 16 プロセッササーバ 上で動作させた場合 Intel Fortran Itanium Compiler 7.1 に対し平均 2.40 倍，Sun Fire V880 Ultra SPARC III Cu 8 プロセッササーバ上 Sun キャッシュプリフェッチ ON の状態で Sun Forte コンパイラ 7.1 に対し平均 1.90 倍の性能向上が得られ，また OSCAR マルチグレイン並列化 + Sun プリフェッチにより 8 プロセッサ時に 1 プロセッサプリフェッチ OFF の場合に対して 32.9 倍という大きなスピードアップが得られることが確かめられた．これにより OSCAR コンパイラで用いられている並列化技術の SMP サーバ上での実用性，ポータビリティ，商用コンパイラによるプリフェッチとの相互協調可能性等が確かめられた．

5. 謝　　辞

本研究の一部は METI/NEDO ミレニアムプロジェクト IT21 “Advanced Parallelizing Compiler”，NEDO 先進ヘテロジニアスマルチプロセッサ研究開発，及び STARC（半導体理工学研究センター）の支援により行われた．

参 考 文 献

1) M.Wolfe: High Performance Compilers for Parallel Computing, *Addison-Wesley Publishing Company* (1996).
2) 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).
3) Hall, M. W., Anderson, J. M., Amarasinghe, S. P., Murphy, B. R., Liao, S., Bugnion, E. and Lam, M. S.: Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer* (1996).
4) : .
5) Lim, A. W., Cheong, G. I. and Lam, M. S.: An Affine Partitoning Algorithm to Maximize Parallelism and Minimize Communication, *Proc. of the 13th ACM SIGARCH International Conference on Supercomputing* (1999).
6) Gonzalez, M., Martorell, X., Oliver, J., Ayguade, E. and Labarta, J.: Code Generation and Runtime Support for Multi-level Parallelism Exploitation, *Proc. of the 8st International Workshop on Compilers for Parallel Computing* (2000).
7) Saito, H., Stavakos, N. and Polychronopoulos, C.: Multithreading Runtime Support for Loop and Functional Parallelism, *Proc. of the International Symposium on High Performdance Computing* (1999).
8) 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).
9) H.Kasahara and et al: A Multi-grain Parallizing Compilation Scheme on OSCAR, *Proc. 4th Workshop on Language and Compilers for Parallel Computing* (1991).
10) 笠原博徳: 最先端の自動並列化コンパイラ技術, 情報処理, Vol.44 No. 4(通巻 458 号), pp.384-392 (2003).
11) 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: 木村 啓二 and 加藤 孝幸 and 笠原 博徳, 情報処理学会論文誌, Vol. 42, No. 4 (2001).
12) 小幡元樹, 白子準, 神長浩気, 石坂一久, 笠原博徳: マルチグレイン並列処理のための階層的並列処理制御手法, 情報処理学会論文誌, Vol. 44, No. 4 (2003).
13) 白子準, 長澤耕平, 石坂一久, 小幡元樹, 笠原博徳: マルチグレイン並列性向上のための選択的インライン展開手法, 情報処理学会論文誌, Vol. 45, No. 5 (2004).
14) 石坂, 中野, 八木, 小幡, 笠原: 共有メモリマルチプロセッサ上でのキャッシュ最適化を考慮した粗粒度タスク並列処理, 情報処理学会論文誌, Vol. 43, No. 4 (2002).
15) 吉田, 前田, 尾形, 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, 情報処理学会論文誌, Vol. 35, No. 9, pp. 1848–1994 (1994).
16) 石坂一久, 小幡元樹, 笠原博徳: 配列間パディングを用いた粗粒度タスク間キャッシュ最適化, 情報処理学会論文誌, Vol. 45, No. 4 (2004).