IBM pSeries 690 SMP サーバー上での OSCAR マルチグレイン自動並列化コンパイラの性能評価

石坂 一久[†]　白子 準[†]　小幡 元樹[‡]　木村 啓二[†]　笠原 博徳[†]

[†]早稲田大学理工学部　[‡]日立製作所システム開発研究所

概要　SMP マシンの普及に伴い，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．OSCAR 自動並列化コンパイラでは，従来から商用コンパイラで利用されているループ並列性に加え，基本ブロック，ループ，サブルーチンなどの粗粒度タスク間の並列性およびステートメント間の近細粗粒度並列性等プログラム全域にわたる複数粗粒度の並列性を利用するマルチグレイン並列処理を実現している．マルチグレイン並列処理では，プログラム中のネストされた階層的な複数粗粒度の並列性を解析し，プログラム全体として最も効果的な並列処理が行えるように，各階層に必要なプロセッサを自動的に割り当てる．プロセッサとメモリの動作速度のギャップに伴う処理速度の停会を軽減するために，タスク間でのデータ転送にキャッシュを有効利用することで性能を向上させる．24 プロセッサ SMP サーバー IBM pSeries 690 上での OSCAR マルチグレイン自動並列化コンパイラの性能評価では，SPEC CFP95 の 10 プログラムに対して，IBM XL Fortran version 8.1 コンパイラの性能を平均 4.3 倍上回り，OSCAR コンパイラで実現されている並列化技術の有効性が示すことができた．

1　はじめに

現在マルチプロセッサシステムはハイパフォーマンスコンピュータをはじめエントリーレベルサーバー，デスクトップワークステーション，PC，ゲーム機等で幅広く利用されるようになっている．さらに今後はプロセッサのマルチコア化が進み多くの分野でマルチプロセッサシステムが使われるようになっていくと考えられている．そのようなマルチプロセッサシステムの実効性能を向上させ，ユーザーの使い易さや価格性能比を高めるには，逐次プログラムを自動的に並列化する高性能の自動並列化コンパイラが重要であると認識されている．従来より多くの自動並列化技術が研究されており，イリノイ大学の Polaris コンパイラ [1] 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などのような依存解析手法を用いたループ並列化を実現した．またスタンフォード大学の SUIF コンパイラ [2] はインタープロシージャ解析を用いたループ並列処理，ユニモジュラ変換およびアフィンパーティショニング等を用いたデータローカリティ最適化を考慮したループ並列処理を研究している．

しかし，マルチプロセッサシステムの最高性能と実効性能の差は拡大しており，今後さらに実効性能を向上させるためには，従来のループ並列化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重要である．カタルーニャ大学の NANOS コンパイラは，拡張した OpenMP API[3] 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 [4]．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 [5]．

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，プログラム中の複数レベルの並列性を利用するマルチグレイン並列処理を行っている．本論文では OSCAR コンパイラで用いられている並列化技術について述べる．

2　OSCAR 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マルチプラットフォーム対応の自動並列化コンパイラである．OSCAR コンパイラの構成を図 1 に示す．フロントエンドは逐次 FORTRAN プログラムおよび OpenMP FORTRAN プログラムを読み込み中間言語を生成する．ミドルパスでは無用式の削除，定数伝搬などの一般的な最適化に加え，マルチグレイン並列性を抽出するためのプログラムリストラクチャリング，マルチグレイン並列性解析，並列処理階層の自動選択，自動プロセッサ割り当て，キャッシュ最適化などの最先端の最適化を行う．OSCAR コンパイラは複数のバックエンドを持ち，OSCAR Chip Multiprocessor[6] 用マシンコード，SMP マシン用 OpenMP コード，分散メモリアーキテクチャ用の MPI コードなど複数のターゲット用に並列化コードを生成することができる．本論文では OpenMP Backend を用いて SMP 用 Open MP コードを生成する．

2.1　マルチグレイン並列処理

マルチグレイン並列処理は，プログラムを基本ブロック・サブルーチンブロック・繰り返しブロックの 3 種類のマクロタスクに分割し，これらのブロック間の並列性を利用する粗粒度タスク並列性，ループイタレーション間の並列性，基本ブロック内のステートメントや命令間の近細粒度並列性を併

Evaluation of OSCAR Multigrain Automatic Parallelizing Compiler on IBM pSeries 690

Kazuhisa Ishizaka[†], Jun Shirako[†], Motoki Obata[‡], Keiji Kimura[†], Hironori Kasahara[†]

[†]Waseda University, [‡]Hitachi Co.Ltd.

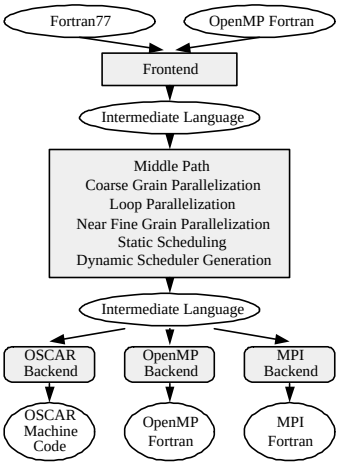


図 1: OSCAR コンパイラの構成

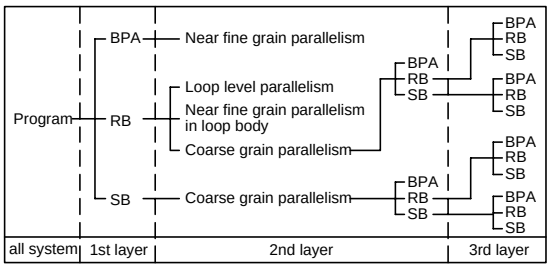
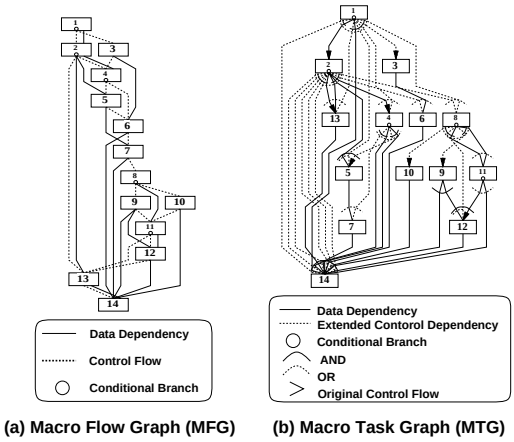


図 2: 階層的マクロタスク定義

用することにより，これまで利用されていなかったプログラム中の潜在的な並列性を利用する並列処理手法である．本論文では，評価を行うマルチプロセッサシステムとして，市販の共有メモリ型マルチプロセッサシステムを利用するため，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性は用いず，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．

2.1.1 階層的タスク生成

粗粒度タスク並列処理ではソースプログラムを，単一の基本ブロック，スケジューリングオーバーヘッドを考慮し複数の小基本ブロックを融合あるいは並列性向上のため単一の基本ブロックを分割して生成される疑似代入文ブロック（BPA），DO ループまたは IF 文による分岐によって生成されるループ，すなわち最外側ナチュラルループからなる繰り返しブロック（RB），サブルーチンブロック（SB）の三種類の粗粒度タスク（マクロタスク）に分割する．さらに，繰り返しブロック（RB）の内，データ依存等によりループ並列処理ができないシーケンシャルループのボディ部，またコード長などを考慮しインライン展開を行なわなかったサブルーチンの内部に対しては，図 2 に示すように階層的にその内部をサブマクロタスクに分割する．



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

図 3: マクロフローグラフとマクロタスクグラフ

2.1.2 最早実行可能条件解析

次に，生成された各階層のマクロタスク間の制御フロー，データフローを解析し，図 3(a) に示すようなマクロフローグラフを各階層ごとに生成する．マクロフローグラフ中，ノードはマクロタスク，ノード中の番号はマクロタスク番号，ノード内の小円は条件分岐ノード ノード間の実線はデータフロー，点線は制御フローをそれぞれ表す．

このマクロフローグラフは，マクロタスク間の制御フロー，データフローを表すだけであり，マクロタスク間の並列性を表してはいない．そこで，各マクロタスクに対して最早実行可能条件解析 [7] を適用することによって，マクロタスク間の並列性を抽出し図 3(b) に示すようなマクロタスクグラフを生成する．マクロタスクグラフでは，ノード，ノード中の番号，ノード中の小円はマクロフローグラフ同様，マクロタスクと条件分岐を表しているが，ノード間の実線はデータ依存，点線は拡張された制御依存を表している．拡張された制御依存とは，通常の制御依存の他に，マクロタスクの非実行確定条件を含んでいる．また，実線アーク（弧）は，アークが束ねるエッジが AND 関係にあり，点線アークは束ねるエッジが OR 関係にあることを表している．このマクロタスクグラフは，マクロタスク間の粗粒度並列性を直接表現したグラフとなっている．例えば，図 3 中のマクロタスク 6 の最早実行可能条件は,「条件分岐 MT2 が MT4 に分岐することが決定するか，MT3 の実行が終了」という条件になる．

2.1.3 タスクスケジューリング

粗粒度タスク並列処理では，マクロタスクのプロセッサへの割り当てには，スタティックスケジューリングまたはダイナミックスケジューリングが用いられる．

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適用され，コンパイル時にタスクのプロセッサへの割り当てを決定し，スケジューリング結果に従い各プロセッサ用に異なった並列化コードを生成する．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，データ転送，同期等のオーバーヘッドを最小化することができる．

一方，マクロタスクグラフが条件分岐などの実行時不確

定性を持つ場合は，実行時にマクロタスクを割り当てるダイナミックスケジューリングを用いる．コンパイラは実行時スケジューリングのためのスケジューリングルーチンを生成し，マクロタスクコードと共に出力する並列化コードに埋め込む．対象プログラム専用の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ユーザープログラムとして生成し，それを粒度の大きな粗粒度タスクのスケジューリングに用いることによって，実行時スケジューリングのオーバーヘッドを相対的に低くしている．

2.1.4 OpenMP によるコード生成

本論文では OpenMP バックエンドを用いて，マルチグレイン並列処理を実現した OpenMP FORTRAN コードを出力する．これによりポータビリティを高めることが可能で，OpenMP をサポートしている各種マルチプロセッサ上で実行することが可能となる．

OSCAR コンパイラによって生成された OpenMP コードでは，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．ディレクティブで作られる OpenMP セクションには，プログラム中の全階層での各プロセッサ用のコードが，選択されたスケジューリング方式に従って生成される．

本実現方式では，ネストされた階層的な並列処理を，各プロセッサ（スレッド）毎に別々のコードを生成する特殊な方式で実現しており，プログラム開始時の 1 回のスレッドの fork とプログラム最後の 1 回の join で，OpenMP の拡張を行うこと無く低オーバーヘッドで行えるという特徴がある．

2.2 自動プロセッサ割り当て

マルチグレイン並列処理では，プログラム中から階層的に複数レベルの並列性を抽出して利用する．したがって，各階層の並列性を解析してマルチプロセッサシステム上の利用可能な複数のプロセッサをどのように各階層の並列性に割り当てるかを判断することが重要である．しかし，適切な並列処理階層の決定と階層的プロセッサグルーピングを一般ユーザーが判断するのは非常に困難である

OSCAR コンパイラは，利用可能なプロセッサを自動的に効率良くプログラムの各所に割り当てる自動プロセッサ割り当てを実現している [8]．自動プロセッサ割り当て手法では，コンパイル時にプログラム中の各階層の並列度を推定した上で，より上位階層の並列性を優先するようにプロセッサを割り当てることで，同期やスケジューリングオーバヘッドを低減している．また並列性の小さい階層に対しては必要なプロセッサ数のみを利用し，利用しないプロセッサにはその個所の並列コードを生成せずに停止させることにより並列処理オーバヘッドの軽減を図っている．

2.3 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 [9] 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．

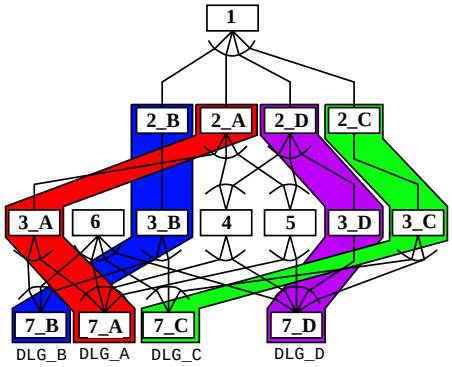


図 4: 粗粒度タスク間キャッシュ最適化

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手法 [10] を用いてデータ分割を行う．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 4 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がキャッシュ上の同一セットに割り当てられるかどうかを検出する．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，データが同一セットに割り当てれる場合は，ダイレクトマップキャッシュや連想度の小さいキャッシュでは，コンフリクトミスが生じてしまう可能性がある．そこで，OSCAR コンパイラは配列間パディングを用いることで，同一 DLG 内のタスクによってアクセスされる配列がキャッシュ上の異なるセットに割り当てられるように，データレイアウトを変換することでコンフリクトミスを削減する．

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．

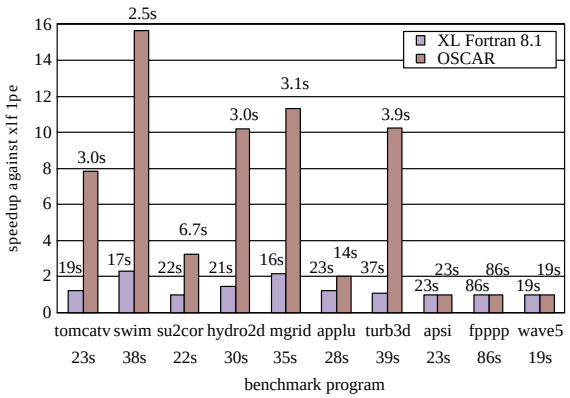


図 5: pSeries 690 上での性能評価

3 性能評価

OSCAR マルチグレイン自動並列化コンパイラの性能評価を 24 プロセッサ SMP サーバー IBM pSeries 690 上で行う．pSeries 690 の CPU は 1.1GHz の Power4 で OS は AIX 5.1，マシンネイティブコンパイラは IBM XL Fortran version 8.1 を用いた．性能評価では，OSCAR コンパイラの生成した OpenMP コードを XLF コンパイラでコンパイルして実行した．比較対象として XLF コンパイラの自動並列化を用いる．また性能評価には SPEC CFP95 の 10 プログラムを用いた．

図 5 に各コンパイラで並列化した場合の，XLF コンパイラのみを用いて逐次実行を行った場合に対する 24 プロセッサまでの最大の速度向上率を示す．図中のプログラム名の下の数値は逐次処理時間を，棒グラフ上の数値は並列処理処理時間を示す．例えば swim では逐次処理時間が 38 秒であるのに対し，XLF コンパイラで自動並列化を行った場合の 24 プロセッサまでの最小実行時間は 17 秒となり速度向上率は 2.3 倍，OSCAR コンパイラで並列化した場合の最小実行時間は 2.5 秒となり速度向上率は 15.6 倍である．したがって，OSCAR コンパイラを用いることにより XLF コンパイラに対して 6.8 倍（15.6/2.3）の性能向上が得られている．同様に tomcatv では XLF の最高性能が 19 秒，OSCAR が 3.0 秒で 6.5 倍の性能向上が得られ，su2cor では 22 秒から 6.7 秒へ 3.2 倍，hydro2d では 21 秒から 3.0 秒へと 7.1 倍，mgrid では 16 秒から 3.1 秒へ 5.3 倍，applu では 23 秒から 14 秒へ 1.7 倍，turb3d では 37 秒から 3.9 秒へ 10.3 倍の性能向上が得られた．全 10 プログラムの平均では，OSCAR コンパイラは XLF コンパイラに対して 4.3 倍の性能向上を示した．

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 IBM pSeries 690 SMP サーバー上での性能について述べた．本コンパイラは従来の並列化コンパイラで利用されているループ並列処理に加えて，ループ，サブルーチンなどの粗粒度タスク間の並列性，基本ブロック内のステートメント間の並列性を利用する近細粗粒度並性を抽出し，それらの複数階層的並列性を効果的に利用するプロセッサ割り当てを自動的に決定するマルチグレイン並列処理を実現している．また，データローカライゼーション技術及びデーレイアウト変換パディング技術を用いたキャッシュ最適化技術により，タスク間のデータ転送にキャッシュを有効利用することで，マルチグレイン並列処理の性能を向上させている．

24 プロセッサ SMP サーバー IBM pSeries 690 上での性能評価では，OSCAR コンパイラによる自動並列化は，IBM XL Fortran 8.1 の自動並列化の最高性能に対して，SPEC CFP95 ベンチマークの tomcatv で 6.5 倍，swim で 6.8 倍，hydro2d で 7.1 倍の性能向上を示したのをはじめ全 10 プログラムの平均で 4.3 倍上回る性能を示し，OSCAR コンパイラで用いられている並列化技術によりマルチプロセッサシステムの実効性能を大きく向上させることが示された．

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”[11] 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] R. Eigenmann, J. Hoeflinger, and D. Padua. On the automatic parallelization of the perfect benchmarks. *IEEE Trans. on parallel and distributed systems*, Vol. 9, No. 1, Jan. 1998.

[2] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the suif compiler. *IEEE Computer*, 1996.

[3] Openmp: Simple, portable scalable smp programing. http://www.openmp.org.

[4] M. Gonzalez, X. Martorell, J. Oliver, E. Ayguade, and J. Labarta. Code generation and run-time support for multi-level parallelism exploitation. In *Proc. of the 8st International Workshop on Compilers for Parallel Computing*, Jan. 2000.

[5] H. Saito, N. Stavakos, and C. Polychronopoulos. Multithreading runtime support for loop and functional parallelism. In *Proc. of the International Symposium on High Performdance Computing*, May 1999.

[6] K. Kimura, T. Kodaka, M. Obata, and H. Kasahara. Multigrain parallel processing on oscar cmp. In *Proc. of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, Jan. 2003.

[7] 笠原博徳. 並列処理技術. コロナ社, Jun. 1991.

[8] 小幡, 白子, 神長, 石坂, 笠原. 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, Apr. 2003.

[9] 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. Vol. 43, No. 4, Apr. 2002.

[10] 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, Sep. 1994.

[11] 笠原博徳. 最先端の自動並列化コンパイラ技術. 情報処理学会誌, Vol. 44, No. 4, pp. 384–392, Apr. 2003.