商用SMP上での粗粒度タスク並列処理

小幡 元樹[†,†††] 石坂 一久[†,†††] 神長 浩気[†]
中野 啓史[†] 吉田 明正[††,†††] 笠原 博徳[†,†††]
早稲田大学[†] 東邦大学[††] アドバンスト並列化コンパイラ共同体[†††]

本論文では，SPEC95 ベンチマークと Perfect Club ベンチマークの 5 本のプログラムを用い，商用 SMP 上における OSCAR マルチグレイン並列化コンパイラを用いた粗粒度並列処理の評価を行う．現在，サーバアーキテクチャの主流である SMP 上での自動並列化コンパイラを用いた並列処理では，ループレベル並列処理の性能が飽和状態に達しており，その限界を越えるため粗粒度タスク並列処理が研究されている．OSCAR FORTRAN コンパイラにおける粗粒度並列処理手法では，ソースプログラム中のサブルーチン・ループ・基本ブロック間の並列性を抽出し，各種 SMP 上で粗粒度タスク並列化を実現するために，OpenMP を用いたワンタイムシングルレベルスレッド生成手法を用いている．さらに SMP で問題になる共有メモリアクセスオーバヘッドを軽減するため，複数タスク間での共有データの授受にキャッシュを最大限利用しようとするデータローカライゼーション手法を併用することで，さらに性能を向上させることができる．これらの技術を用いて，本論文では商用 SMP サーバ IBM RS6000 SP 604e High Node，SMP ワークステーション SUN Ultra80 での粗粒度並列化の性能評価を行った．その結果，粗粒度並列処理は，スレッド管理オーバヘッド，メモリアクセスオーバヘッドの軽減により既存のループ自動並列化コンパイラの性能を 5 つのアプリケーションにおいて 60%から 430%上回ることが確認された．

Coarse Grain Task Parallel Processing on Commercial SMPs

Motoki Obata[†,†††], Kazuhisa Ishizaka[†,†††], Hiroki Kaminaga[†], Hirofumi Nakano[†], Akimasa Yoshida[††,†††] and Hironori Kasahara[†,†††]
Waseda University[†] Toho University[††] Advanced Parallelizing Compiler Project[†††]

This paper evaluates performance of coarse grain task parallel processing using OSCAR Multigrain Parallelizing Compiler for five applications from SPEC95FP and Perfect Club benchmarks on commercial SMP machines. The coarse grain task parallel processing is important to improve the effective performance of SMP machines beyond the limit of the loop parallelism. In this OSCAR compiler, One-time Single Level Thread Generation scheme using OpenMP API and a data localization scheme are used to realize coarse grain task parallelization efficiently on various SMP machines. The evaluation shows that the coarse grain parallel processing gives us 60-430% larger speed up than the automatic loop parallelizing compiler for the five applications by the reduction of overheads of thread management and shared memory access on SMP server IBM RS6000 SP 604e High Node and SMP workstation SUN Ultra80.

1 はじめに

現在，SMP アーキテクチャはシングルチップマルチプロセッサからワークステーション，各種サーバマシンまで多くのシステムで採用されている．このような SMP マシン上では従来よりループレベル並列処理技術が用いられ，様々なプログラムリストラクチャリング技術を用いたループ並列化コンパイラが開発されてきた．例えば，Polaris コンパイラ [1),2)] は，サブルーチンのインライン展開，シンボリック伝搬，アレイプライベタイゼーション [1),3)]，実行時データ依存解析 [2)] によってループ並列性を抽出する．また，SUIF コンパイラ [4)~6)] は，インタープロシージャ解析，ユニモジュラ変換，アフィンパーティショニングを用いたループ内とループ間のデータローカリティ [7)~9)] に関する最適化などを用いてループを並列処理する．

これらをはじめとする優れたループ並列化コンパイラにより，様々なプログラムの解析・評価が行われてきたが，ループレベル並列性抽出技術は既に成熟期にあり，今後の画期的な性能向上は見込めないと言われている．したがって，今後の並列処理の性能向上のためには，これまでの並列化コンパイラでは抽出できなかった粗粒度並列性の利用が重要となる．

粗粒度並列性を利用するコンパイラとしては，NANOS コンパイラ [10),11)] と PROMIS コンパイラ [12),13)]，そして OSCAR マルチグレイン並列化コンパイラが挙げられる．NANOS コンパイラでは，階層並列化実現のための拡張 OpenMP API[14)~16)] を用いて，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，PROMIS コンパイラは，HTG とシンボリック解析 [17)] を用いる Parafrase2 コンパイラ [18)] をベースとして，実用レベルのコンパイラを開発する努力が行われている．

筆者らが開発中の OSCAR FORTRAN マルチグレイン並列化コンパイラ [19),20)] は，ループ並列化に

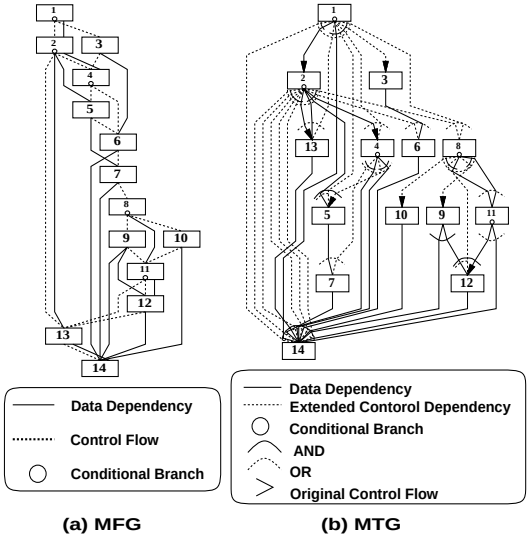

図 1: マクロフローグラフ（MFG）とマクロタスクグラフ（MTG）の例

加え，ループ・サブルーチン・基本ブロック間の並列性を利用するマルチグレイン並列処理 [19)~21)] を実現し，OpenMP を用いたワンタイムシングルレベルスレッド生成手法 [22)] によって，NANOS コンパイラのような言語拡張を行うこと無く，低オーバヘッドでの粗粒度並列処理を可能としている．さらに SMP マシンで問題となる大きな共有メモリアクセスオーバヘッドを軽減するため，粗粒度タスク間共有データをキャッシュメモリ上で授受できるようにするデータローカライゼーション手法 [23)] の開発も行っている．

本論文では，この OSCAR マルチグレイン並列化コンパイラを用いて，SPEC95FP ベンチマークの 101.tomcatv，102.swim，107.mgrid，103.su2cor，Perfect Club ベンチマークの ARC2D について，SMP サーバ IBM RS6000，SMP ワークステーション SUN Ultra80 の 2 機種のマシン上で粗粒度並列処理性能の評価を行う．

2 粗粒度タスク並列処理

本章では，逐次プログラムを粗粒度タスクに分割し粗粒度タスク間並列性解析を行う手法，及び並列性抽出後の OpenMP を用いた粗粒度並列化プログラム生成法について述べる．

粗粒度タスク並列処理とは，プログラムを基本ブロック，あるいはその融合ブロック（BPA），繰返しブロック（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 MT をプロセッサエレメント（PE）や複数 PE をグループ化した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，MT 間の並列性を利用する方式である．

OSCAR マルチグレイン並列化コンパイラにおける粗粒度並列処理の手順は次のようになる．

1. 逐次プログラムの MT への分割．
2. MT 間の制御フロー，データ依存解析によるマクロフローグラフ（MFG）の生成．
3. 最早実行可能条件解析によるマクロタスクグラフ（MTG）の生成 [21),24)]．
4. MTG がデータ依存エッジのみで構成される場合は，スタティックスケジューリングによる MT の PC または PE への割り当て．MTG がデータ依存エッジと制御依存エッジを持つ場合は，コンパイラによるダイナミックスケジューリングルーチンの自動生成．
5. OpenMP による粗粒度並列化プログラム生成．

以下，各ステップの概略を示す．

2.1 粗粒度タスク生成

粗粒度タスク並列処理では，ソースプログラムは，基本ブロック，あるいはその融合ブロック（BPA），繰返しブロック（RB），サブルーチンブロック（SB）の 3 種類のマクロタスク（MT）に分割される．生成された RB が並列化可能ループ，すなわち Doall ループの場合は，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ループを分割し，それぞれ異なった粗粒度タスクとして定義する．また，ループ並列化不可能で実行時間の長い 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，そのボディ部を階層的に粗粒度タスクに分割し並列処理を行う．

2.2 粗粒度並列性抽出

次に，各階層（ネストレベル）において，生成された MT 間のデータ依存と制御フローを解析する．解析結果は，図 1（a）に示すようなマクロフローグラフ（MFG）として表される．図中，ノードは MT を表し，実線エッジはデータ依存，点線エッジは制御フローを表す．また，ノード内の小円は条件分岐を表す．図中のエッジの矢印は省略されているが，エッジの向きは下向きを仮定している．

MFG 生成後，MT 間の並列性を抽出するために，データ依存と制御依存を考慮し，各 MT の最早実行可能条件を解析する．最早実行可能条件とは，各 MT が最も早い時点で実行可能になる条件を表し，例えば，図 1（a）における MT6 の最早実行可能条件は，MT3 が終了するか MT2 が MT4 に分岐，となる．各 MT の最早実行可能条件は，図 1（b）に示すようなマクロタスクグラフ（MTG）として表される．

MTG においても，ノードは MT を，ノード内の小円は条件分岐を表す．また，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制御依存を表す．ただし，拡張された制御依存とは，通常の制御

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含むものである．図 1（b）中のエッジを束ねる実線アークは，アークによって束ねられたエッジが AND 関係にあることを示し，点線アークは OR 関係にあることを表す．MTG においても，エッジの矢印は省略されているが，下向きを仮定している．また，矢印のあるエッジはオリジナルの制御フローを表している．

2.3 SMP 用粗粒度並列化コード生成

SMP 上での粗粒度タスク並列処理では，OpenMP ディレクティブ “PARALLEL SECTIONS” によってプログラムの実行開始時に一度だけ並列スレッドを生成するワンタイムシングルレベルスレッド生成手法 [22] を用いる．OSCAR コンパイラによって生成された粗粒度タスクは，ダイナミックスケジューリング，あるいはスタティックスケジューリングを用いて，PE に対応するスレッド，あるいは PC に対応するスレッドグループに割り当てられる．スレッドグループとは，MT の割り当て単位として，任意の数のスレッドをプログラム的にグループ化したものである．MTG 内に条件分岐のような実行時不確定性が存在する場合はダイナミックスケジューリングが適用され，MT は実行時にスレッドグループもしくはスレッドに割り当てられる．ダイナミックスケジューリングルーチンは，OS コールによるスレッドスケジューリングオーバヘッドを避けるために，コンパイラによって各プログラムに合わせて生成され，出力される並列化プログラム内に埋め込まれる．一方，MTG がデータ依存のみで構成される場合には，コンパイル時にスタティックスケジューリングを適用し，実行時スケジューリングオーバヘッドの最小化を図っている．

階層的な MTG が生成されている場合は階層的粗粒度並列処理を行うが，一般的に，階層的並列処理は，上位スレッドが子スレッドを生成することによって実現される．しかし，本論文で用いる OSCAR FORTRAN コンパイラでは，“PARALLEL SECTIONS”～“END PARALLEL SECTIONS” ディレクティブ間の “SECTION” ディレクティブ間に，生成される全 MTG 階層での処理やダイナミックスケジューリングを適用する全 MTG 階層のスケジューリングルーチンを記述することによって，シングルレベルスレッド生成のみで階層的並列処理を実現する．これにより，スレッドの fork/join オーバヘッドの最小化，及び既存の言語仕様のみで階層的粗粒度並列処理を実現することができる．

また，スタティックスケジューリング，ダイナミックスケジューリングにおいて，MT 間でキャッシュメモリを用いて共有データを受け渡すためのデータローカライゼーション手法 [23] を適用することも可能である．データローカライゼーションを適用する際は，各 MT の使用データ量がキャッシュサイズ以内に収まるように粗粒度タスク分割し，同一データ領域を使用する MT ができるだけ同一プロセッサ上で連続実行されるようにスケジューリングを行う．

3 SMP 上での性能評価

本章では，OSCAR FORTRAN コンパイラを用いて，SPEC95 ベンチマークの tomcatv，swim，mgrid，su2cor，Perfect Club ベンチマークの ARC2D プログラムを粗粒度タスク並列化し，IBM RS6000 SP 604e High Node，SUN Ultra80 上で性能評価を行った結果を述べる．

3.1 OSCAR FORTRAN Compiler

マルチグレイン並列化コンパイラ “OSCAR FORTRAN 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．本論文では，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 OpenMP バックエンドを用いる．すなわち，OSCAR コンパイラは逐次 Fortran を OpenMP Fortran に変換するプリプロセッサとして利用される．

3.2 RS6000 上での評価

本節では，RS6000 上での粗粒度並列処理結果について述べる．

評価に用いた 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 ずつの L1 命令，データキャッシュと，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．使用したコンパイラは XL Fortran Compiler Version 7 であり，OSCAR コンパイラの出力をコンパイルする際のオプションは，“-O3 -qsmp=noauto -qhot -qarch=ppc -qtune=auto -qcache=auto” である．XL Fortran Ver.7 単独での評価においては，逐次性能評価でのオプションとして “-O5 -qhot -qarch=ppc -qtune=auto -qcache=auto” を用い，自動並列化では “-O5 -qsmp=auto -qhot -qarch=ppc -qtune=auto -qcache=auto” を用いた．

図 2 に tomcatv，swim，mgrid，su2cor，arc2d の評価結果を示す．横軸はプログラム名を示し，プログラム名の下の数字はオリジナルプログラムを XL Fortran で逐次処理コンパイルした際の実行時間を表す．縦軸は，XL Fortran Ver.7 での逐次処理時間を 1.00 とした場合の速度向上率を示す．各プログラムについては，図中左からオリジナルソースプログラムを XL Fortran によって自動並列化した際の 8 スレッドでの性能，XL Fortran による自動並列化において 8 スレッドまで用いて並列化を行った場合の最高性能が得られたスレッド数とその性能，OSCAR コンパイラの OpenMP 出力を XL Fortran でコンパイルした際に最高性能を与えたスレッド数

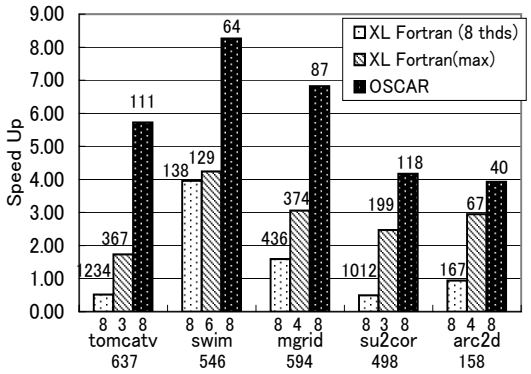

図 2: RS6000での評価

とその際の性能を表している．また，棒グラフ上部の数字はそれぞれの並列処理時間である．

図 2 中，tomcatv では，XL Fortran による自動並列化においては 3 スレッドで最高性能が得られ，1.7 倍の速度向上となったのに対して，OSCAR コンパイラは 8 スレッドで 5.7 倍の速度向上が得られた．swim では，逐次処理に対する XL Fortran は 6 スレッドで 4.2 倍だが，OSCAR コンパイラの性能は 8 スレッドで 8.3 倍となった．また mgrid においては，XL Fortran では 4 スレッドで 3 倍の速度向上であったが，OSCAR コンパイラでは 8 スレッド時で 6.8 倍の最高性能が得られた．tomcatv，swim，mgrid では，スタティックスケジューリングを用いて粗粒度並列処理を行っている．これらのアプリケーションはループ並列性が大きいため，XL Fortran も OSCAR コンパイラもループ並列性を利用しているが，OSCAR コンパイラのスレッド数に対するリニアな性能向上に対して，XL Fortran ではスレッド数の増加に対してスケーラブルな性能向上が見られない．この差の理由として，time コマンドによる計測結果における System time の差が使用スレッド数が増えるにつれて非常に大きくなることが挙げられる．例として，表 1 に，tomcatv と swim における各スレッド数ごとの System time を示す．表 1 の tomcatv では，XL Fortran の自動並列化で最高性能を示す 3 スレッドでの System time が 110 秒であるのに対して，OSCAR コンパイラでは最高性能が得られる 8 スレッドで 3.1 秒であり，差が非常に大きい．また，RS6000 上で各スレッドやプロセスの処理時間などを計測できる PPROF コマンドの出力結果より，XL Fortran による自動並列化の際の各スレッドでの処理時間は，マスタースレッドの処理時間とスレーブスレッドでの処理時間に非常に大きな差が出ることを確認しており，負荷の不均衡が生じている．

一方，図 2 における su2cor，arc2d は，分散ダイナミックスケジューリングによる粗粒度並列処理を行った結果である．ただし，OSCAR コンパイラによる評価では，su2cor ではインライン展開，配列リネーミング，arc2d ではインライン展開，ループアンローリングをオリジナルの逐次プログラムに対して適用したプログラムを逐次プログラムとして OSCAR コンパイラに入力した．これは，研究用コンパイラである OSCAR コンパイラは既存のコンパイラで行われていない機能の実現・実装に主眼を置いているため，多くの市販コンパイラで実現されているリストラクチャリング技術のいくつかは未実装，あるいはデバッグ作業を遅らせているため，今回の評価では手動でリストラクチャリングを行った後、OSCAR コンパイラを用いた粗粒度並列化を行った。

表 1: RS6000 での tomcatv，swim の System time

	tomcatv		swim	
Thd	OSCAR	XLF	OSCAR	XLF
1	0.8	0.8	0.5	0.5
2	0.9	50.9	0.7	2.1
3	1.2	110	0.9	8.5
4	1.5	195	1.3	11.8
8	3.1	1407	3.1	44.9

単位：秒

逐次処理に対する XL Fortran による自動並列化での最高性能は，su2cor では 2.5 倍，arc2d では 3.0 倍の速度向上であったが，OSCAR コンパイラにおける最高性能は，su2cor では 8 スレッドで 4.2 倍，arc2d では 8 スレッドで 3.9 倍となった．su2cor，arc2d においては，OSCAR コンパイラによる評価では，分散ダイナミックスケジューリングを適用することによって，図 2 に示すような性能が得られたのに対して，XL Fortran の性能は tomcatv，swim，mgrid と同様，OSCAR コンパイラよりも低い性能を示している．また，使用スレッド数と性能で比較した場合，OSCAR コンパイラでは全てにおいて 8 スレッドを用いた場合が最高性能を示すが，XL Fortran での 8 スレッド使用時の性能は，swim と mgrid を除いて，逐次処理よりも性能が低下している．

したがって，XL Fortran では OS および OpenMP ランタイムルーチンによるスレッドスケジューリングを含めたスレッド管理のためのオーバヘッドが大きいため，性能が向上しにくいことがわかった．同時に，OSCAR コンパイラでのワンタイムシングルレベルスレッド生成手法を用いた粗粒度並列処理手法は，このスレッド管理オーバヘッドを低く抑えられるため，有効であることが確認された．

3.3 Ultra80 上での評価

次に，OpenMP を用いた粗粒度並列処理手法のポータビリティを示すため，tomcatv，swim の Ultra80 上での粗粒度並列処理結果について述べる．

本論文で用いた SUN Ultra80 は，450MHz の Ultra SPARCII を 4 つ搭載しており，1 プロセッサあ

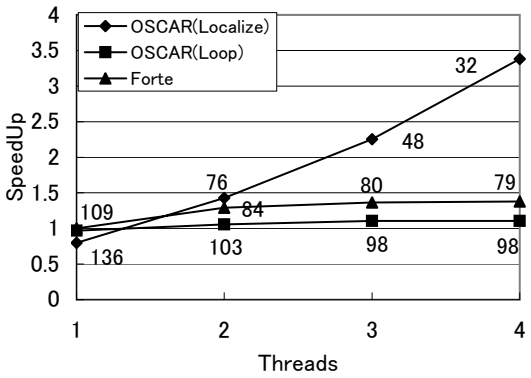

図 3: Ultra80 での tomcatv の評価結果

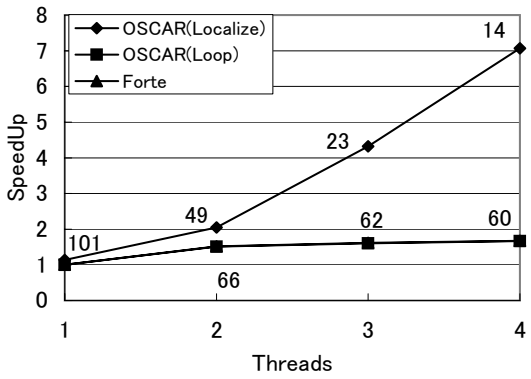

図 4: RS6000 での swim の評価結果

たり，16KB ずつの L1 命令，データキャッシュと 4MB 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．使用したネイティブコンパイラは Forte Developer 6 Update 1 であり，OSCAR コンパイラの出力した粗粒度並列化 Fortran をコンパイルする際のオプションは “-fast -mp=openmp -explicitpar -stackvar” である．Forte 自体の性能評価においては，逐次処理では “-fast”，自動並列化では “-fast -parallel -reduction -stackvar” を用いた．Ultra80 は RS6000 と比較して共有メモリアクセスオーバヘッドが非常に大きいため，今回の tomcatv の評価においてはローカライゼーション手法を適用した．その際，前述のように OSCAR コンパイラの未実装部分、およびバグを避けるため，ループ終値変数の定数化，収束ループ内のループインターチェンジ，配列の次元入れ替えといったリストラクチャリングを手動で行った．今回は，Forte による逐次処理，自動並列化，OSCAR コンパイラによる評価全てに対してこれらの改変後のプログラムを用いた．また，swim についても，サブルーチンのインライン展開とクローニング，ループ終値変数の定数化を行い，tomcatv 同様の評価を行った．これらの変換についても、RS6000 での評価と同様に，既存のコンパイラによって実現されている変換技術であり、現在 OSCAR コンパイラへ組み込み中である。

図 3，4 に Ultar80 上での tomcatv，swim の評価結果をそれぞれ示す．グラフの横軸は使用スレッド数，縦軸は Forte による逐次処理時間を 1.00 とした速度向上率を表し，“OSCAR(Localize)” は OSCAR コンパイラによるローカライゼーション手法，“OSCAR(Loop)” は OSCAR コンパイラでループ並列性のみの利用，そして “Forte” は Forte 6 による自動並列化の結果を表す．また，各プロット横の数字は実行時間（単位は秒）を表す．

tomcatv，swim ともに，OSCAR コンパイラによるデータローカライゼーション手法を適用した結果は優れた性能を示している．Forte による自動並列化では性能はほとんど向上しないが，OSCAR コンパイラの評価においては，tomcatv では，ローカライゼーションの適用により 4 スレッドで 3.4 倍の速度向上が得られた．swim では，4 スレッドで 7.1 倍の速度向上を達成している．

図 3 の 1 スレッド部分を見ると，OSCAR コンパイラによってローカライゼーション手法を適用した結果は Forte による逐次処理の性能よりも低い性能を示しているが，スレッド数が増えるにつれて高効率を示す．この結果は，Forte ではオリジナルソースプログラムに対するキャッシュタイリング技術を中心とした逐次処理最適化は非常に強力だが，ローカライゼーション手法を適用した OSCAR コンパイラによる逐次処理用出力への最適化には向かないことを示している．また図 3，4 より，並列処理時における OSCAR コンパイラでのループ並列性利用結果は，Forte の性能とほぼ同等かそれよりも若干低下していることが分かる．この結果は，Ultra80 ではキャッシュミスヒットした際の主メモリへのアクセスが非常に高コストであるため，OSCAR コンパイラによるデータローカライゼーション手法が有効であることを顕著に示している．

4 まとめ

本論文では，SPEC95FP ベンチマーク，Perfect Club ベンチマークから，101.tomcatv，102.swim，103.su2cor，107.mgrid，ARC2D について，SMP サーバ IBM RS6000 SP 604e High Node と SMP ワークステーション SUN Ultra80 上での粗粒度タスク並列処理の性能評価を行った．その結果，RS6000，Ultra80 用のネイティブコンパイラ XL Fortran Version 7，Forte Developer 6 Update 1 の自動ループ並列化性能を 60%から 430%上回る性能を得ることができた．この評価より，本論文で適用した粗粒度並列処理スレッド生成手法であるワンタイムシングルレベルスレッド生成手法，およびキャッシュ最適化のためのデータローカライゼーション手法が，スレッド管理オーバヘッド，共有メモリアクセスオーバヘッドを軽減するために有効な手段であることが確認された．今後は，より多くのプログラムを用い

た粗粒度並列性解析を行いつつ，その他の商用SMP上でも用いた評価を行う予定である．

なお本研究の一部は，経済産業省／NEDOミレニアムプロジェクトアドバンスト並列化コンパイラにより行われた．

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] 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).

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

[4] Hall, M. W., Murphy, B. R., Amarasinghe, S. P., Liao, S., and Lam, M. S.: Interprocedural Parallelization Analysis: A Case Study, *Proceedings of the 8th International Workshop on Languages and Compilers for Parallel Computing (LCPC95)* (1995).

[5] 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).

[6] Amarasinghe, S., Anderson, J., Lam, M. and Tseng, C.: The SUIF Compiler for Scalable Parallel Machines, *Proc. of the 7th SIAM conference on parallel processing for scientific computing* (1995).

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

[8] 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).

[9] Lim, A. W. and Lam., M. S.: Cache Optimizations With Affine Partitioning, *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing* (2001).

[10] Martorell, X., Ayguade, E., Navarro, N., Corbalan, J., Gozalez, M. and Labarta, J.: Thread Fork/Join Techniques for Multi-level Parllelism Exploitation in NUMA Multiprocessors, *ICS'99 Rhodes Greece* (1999).

[11] Ayguade, E., Martorell, X., Labarta, J., Gonzalez, M. and Navarro, N.: Exploiting Multiple Levels of Parallelism in OpenMP: A Case Study, *ICPP'99* (1999).

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

[13] Brownhill, C. J., Nicolau, A., Novack, S. and Polychronopoulos, C. D.: Achieving Multi-level Parallelization, *Proc. of ISHPC'97* (1997).

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

[15] Dagum, L. and Menon, R.: OpenMP: An Industry Standard API for Shared Memory Programming, *IEEE Computational Science & Engineering* (1998).

[16] Ayguade, E., Gonzalez, M., Labarta, J., Martorell, X., Navarro, N. and Oliver, J.: NanosCompiler: A Research Platform for OpenMP Extensions, *Proc. of the 1st Europian Workshop on OpenMP* (1999).

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

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

[19] 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).

[20] Kasahara, H., Okamoto, M., Yoshida, A., Ogata, W., Kimura, K., Matsui, G., Matsuzaki, H. and H.Honda: OSCAR Multi-grain Architecture and Its Evaluation, *Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems* (1997).

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

[22] 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).

[23] 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).

[24] 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).