計算機アーキテクチャ　148－4
（2002.　5.　13）

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

白子　準[†]　神長浩気[†]　近藤巧章[†]
石坂一久[†,††]　小幡元樹[†,††]　笠原博徳[†,††]

チップマルチプロセッサから HPC まで幅広く使われているマルチプロセッサシステムの実効性能の向上, 使い易さの向上のため, 基本ブロック, ループ、サブルーチン間の粗粒度並列処理・ループイタレーション間の中粒度並列処理・基本ブロック内ステートメント間の近細粒度並列処理を階層的に組合せ, プログラム全域の並列性を利用するマルチグレイン並列処理が重要となっている. マルチグレイン並列処理において階層的に並列性を抽出し, 効率よい並列実行を実現するためには, 各々の階層 (ネストレベル) の並列性に応じて, 何台のプロセッサ, あるいはプロセッサのグループ (プロセッサクラスタ) を割り当てるかを決定する必要がある. 本稿ではプログラム中の各階層の並列性を効果的に用いるための, 各階層へ割り当てるべきプロセッサ数の決定手法を提案する。本手法の有効性を SMP サーバ IBM RS6000 PowerPC 604e High Node 8 プロセッサシステム上にて, SPEC95FP ベンチマーク中 8 本を用いて評価を行った結果について述べる.

Coarse Grain Task Parallel Processing with Automatic Determination Scheme of Parallel Processing Layer

Jun shirako,[†] Hiroki Kaminaga,[†] Noriaki Kondo,[†]
Kazuhisa Ishizaka,[†,††] Motoki Obata[†,††]
and Kasahara Hironori [†,††]

For improvement performance and usablity of multiprocessor systems used from a chip multiprocessor to high performance computer, a multi-grain compilation scheme, which exploits coarse grain parallelism among loops, subroutines and basic blocks, conventional medium grain parallelism among loop-iterations in a Doall loop and near fine grain parallelism among statements inside a basic block, is important. In order to extract the parallelism of each layer(nest level) hierarchically and achieve a better performance in multi-grain parallel processing, it is necessary to determine how much processors or groups of processors(,or processor clusters) should be assigned to the layers, according to the parallelism of the target program layers. This paper proposes an automatic determination scheme of the number of processors to be assigned to each layer, to use the parallelism of each hierarchy in a program efficiently. Effectiveness of the proposed scheme is evaluated on IBM RS6000 SMP server with 8 processors using 8 programs of SPEC95FP.

1. はじめに

マルチプロセッサシステム上での自動並列化コンパイラを用いた並列処理では従来よりループ並列化手法[1)]が用いられている．例えば，イリノイ大学の Polaris[3)] やスタンフォード大学の SUIF[2)] などのような最先端の並列化コンパイラでは，強力なデータ依存解析手法とループリストラクチャリング手法を組み合わせることでさまざまな形状のループが並列化可能になっている．しかしながらこれらのループ並列化手法は既に成熟期に入っており, 今後大幅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.

筆者らが開発中の OSCAR FORTRAN コンパイラ[4),5)] ではループ並列性に加えサブルーチン, ループ, 基本ブロックを粗粒度タスクとし, これらの間の並列性を利用する粗粒度タスク並列処理を実現している. OSCAR コンパイラではプログラムを粗粒度タスクに分割し, 最早実行可能条件解析[6),9)] を用いて各粗粒度タスク間の並列性を抽出してマクロタスクグラフ (MTG) を生成する. 生成された粗粒度タスクがサブルーチン, ループブロックの場合は, 階層的にその内部を粗粒度タスクに分割し, 階層的な MTG を生成することによりプログラムの各ネストレベルに対して MTG を定義し, プログラム全域の並列性を抽出する.

階層的に抽出された並列性を効率よく利用するために

† 早稲田大学理工学部電気電子情報工学科
Dept. of Electrical, Electronics and Computer Engineering, Waseda University

†† アドバンスト並列化コンパイラ研究体
Advanced Parallelizing Compiler Reserch Group
http://www.apc.waseda.ac.jp/

はどの階層の MTG を何台のプロセッサで実行するかを適切に判断する必要がある.[7] 本論文で提案する並列処理階層自動決定手法では, 得られた階層的マクロタスクグラフの並列度を, 静的に計算されたプログラム中の各 MT の実行コスト (逐次処理時間) とクリティカルパス長を用いることにより算出し, この並列度に応じて割り当てるべきプロセッサ数を決定する.

2. 粗粒度タスク並列処理

本章では, 逐次プログラムを階層的に粗粒度タスク分割して階層的マクロタスクグラフ生成し, 粗粒度タスク間並列性解析を行う手法について述べる.

粗粒度タスク並列処理とは生成された MT をプロセッサ (PE), もしくは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.

2.1 粗粒度タスク生成

粗粒度タスク並列処理では, Fortran ソースプログラムは基本ブロックまたはその融合ブロックである BPA, 繰り返し (ループ) 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(粗粒度タスク) に分割される. RB や SB に対しては, そのボディ部を階層的に粗粒度タスク分割し, この処理をプログラム全域に適用する.

2.2 粗粒度並列性抽出

各階層のマクロタスク生成後, 各階層 (ネストレベル) において, MT 間のデータ依存と制御フローの解析により階層的マクロフローグラフ[4] を生成する. 次に，コントロールフローとデータ依存を考慮しマクロタスク間並列性を最大限に引き出すために，各マクロタスクの最早実行可能条件を解析する．この最早実行可能条件は，コントロール依存とデータ依存を考慮したマクロタスク間の並列性を表す. 各マクロタスクの最早実行可能条件は，階層型マクロタスクグラフ (MTG)[4] で表すことができる.

2.3 プロセッサクラスタとプロセッサエレメント

階層的に定義された MTG を効率よく処理するためにはプロセッサも階層的な集合形態をとらねばならない. OSCAR コンパイラでは, 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,1 つのプロセッサクラスタ (PC) と定義し, この PC に MTG 内のマクロタスク (MT) を割り当てる. 割り当てられた MT 内で更に MTG が定義されている場合は, 内部 MTG に応じて PC 内の PE を階層的に PC 化する. これを階層的に行うことでプログラム全域にわたる粗粒度タスク並列性を利用することができる.

3. 並列処理階層決定手法を用いたプロセッサ割り当て

本章では生成された階層的マクロタスクグラフ (MTG) に対して, ネストレベルの浅い上位階層の並列性を優先したプロセッサの割り当て方法を述べる. これは, 一般に上位の階層の方が相対的にプログラムの逐次処理コストが大きく, なるべく外側の階層で粗粒度並列処理する方が同期やスケジューリングによるオーバーヘッドが少なくてすむためである. 本手法では各 MTG の純粋な粗粒度タスク並列性だけでなく, 並列処理可能ループを粗粒度タスク分割することで, ループ並列性を内包する総合的な粗粒度タスク並列性を考慮することが可能である. また, ループを分割することで全 PE になるべく均等に負荷を分散させ, 粗粒度タスク並列処理で問題となる, 負荷のアンバランスを解消している.[8]

3.1 MT の実行コスト推定

提案手法においてはターゲットマシンごとに四則演算, 内部関数, バリア同期などの実行コストを測定し, 各ブロック中の実行コストの合計により MT の逐次処理コストを推定している. これらの MT の逐次処理コストをコントールフローに従って総和した値が MTG の逐次処理コストとなる. 処理コスト推定の対象となる MT が, ループインデックスの初期値および終値が静的に定まらない DO ループであり, かつ内部ブロックで配列が使用されている場合には, その配列の宣言サイズを DO ループの回転数としている. しかし宣言サイズが特定できない場合等には, ループ回転数を事前に設定した値の回転数として推定している. このように算出された回転数を内部ブロックコストの合計に乗じた値が RB の逐次処理コストとなる. また, MTG に条件分岐が含まれる場合は, 分岐確率を用いて実行コストを求める. 今回は実行プロファイルなどを用いず, コンパイル時にコスト推定を行っているため分岐確率を 50% と推定している. しかし, これはプロファイル等を用いることができる場合にはその分岐確率を用い, より正確なタスクコストを求めることが可能である. 以上のように, MTG の逐次処理コストを階層的に積算することで各 MT の逐次処理における実行コストを推測している.

3.2 並列度の計算

次に, 提案手法では各 MTG の逐次処理コストとクリティカルパス長を用い, MTG の粗粒度タスク並列性を表す粗粒度並列度を求める.[8] MTG_i の CP 長 (クリティカルパス長:入り口ノードから出口ノードまでの最短のパス長) を CP_i, 逐次処理コストを Seq_i とし, MTG_i の粗粒度並列度 $Para_i$ を $Para_i = Seq_i/CP_i$ と表す. したがって $\lceil Para_i \rceil$ は MTG_i を CP_i で処理するための PC 数の下限値となる. 次にループ並列性と粗粒度タスク並列性を統合した並列度 ($Para\ after\ Loop\ division$) を $Para\_ALD_i$ と定義する. RB におけるループ並列性と, MTG_i の純粋な粗粒度タスク並列性を総合的に考えるため, 提案手法ではループ並列処理可能 RB に対して, 並列タスク駆動オーバーヘッド, スケジューリングオーバーヘッドを考慮した, 並列処理の効果がある最小のコスト T_{min} を定め, MTG_i 内の並列処理可能 RB がこの T_{min}

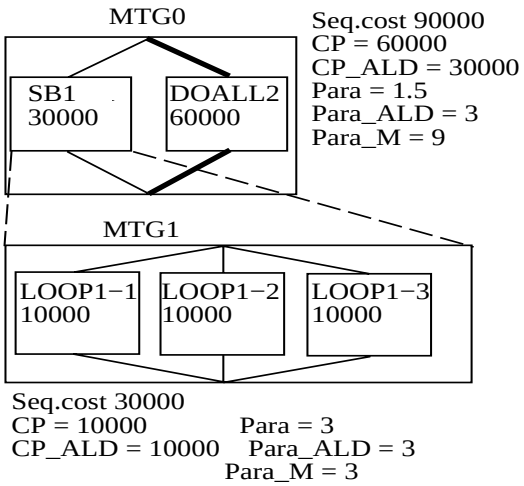

図 1 $Para, Para\_ALDPara\_M$ の例

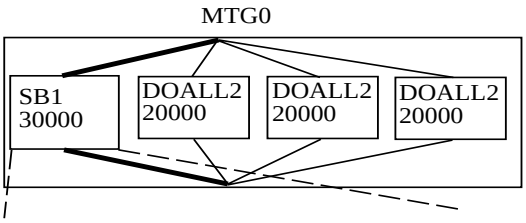

図 2 DOALL2 を 3 分割した場合の MTG_0

を超えるコストになるようにタスク分割する場合を想定する. ただしその RB の 1 回転分の処理コストが T_{min} を超える場合は, RB の回転数をタスク分割数とする. ここで想定するという語を使うのは, この段階では, 並列性の計算上ループを分割した場合を考えるのであり, 実際のループ分割は行わないためである. このようなタスク分割を想定した際に得られる CP 長を CP_ALD_i とする. これを用いて $Para\_ALD_i = Seq_i/CP\_ALD_i$ と定義する. ここで MTG_i をループボディとして持つマクロタスク MT_i が並列処理可能ループであるとき, この MT_i 自体のループ並列性を $Para\_ALD_i$ に反映すべきである. このため, 上記の並列処理の効果が得られる最小コスト T_{min} を超える回転数になるように, MT_i をタスク分割した際の分割数を, MTG_i の内部の $Para\_ALD$ に乗じた値を $Para\_ALD_i$ とする. $\lceil Para\_ALD_i \rceil$ は, MTG_i を CP_ALD_i で処理する際に必要な総プロセッサ数であり, 処理コストを各プロセッサクラスタ (PC) に均等に分散させるために適した PC 数であると言える. よってこれを超えるプロセッサ数を割り当てた場合, その分だけプロセッサがアイドル状態になる可能性が高くなる.

最後に, MTG_i と, その全下位階層の並列性を用いるのに十分なプロセッサ数 ($Max\ Para$) を表わすものとして $Para\_M_i$ を定義する. $Para\_M_i = \lceil Para\_ALD_i \rceil \times \lceil Para\_M_{inner} \rceil$ であり, $Para\_M_{inner}$ は MTG_i 内にある MT のうち最大の $Para\_M$ である. ただし MTG_i 内の並列処理可能なループについては, $\lceil Para\_ALD_i \rceil$ でそのループをタスク分割したものとして $Para\_M$ を考える. これは提案手法が並列処理可能ループの分割を考慮しているためである. 実際には PC 数の決定後, 分割可能ループは属する階層の PC 数, またはプロセッサのキャッシュサイズに応じて, 分割数が決定されるが, ここでは最大の並列度を求めているのであり, MTG_i に必要な PC 数は $\lceil Para\_ALD_i \rceil$ であるので, MTG_i 内の並列処理可能ループは $\lceil Para\_ALD_i \rceil$ で分割されたと考える.

例として図 1について考える. ここで DOALL とは並列実行可能なループ, LOOP は並列実行不可能なループ, すなわち逐次ループであり, 太線はクリティカルパスである. ノード内の数字は逐次処理コストを表している. ループ分割の際の, 並列処理の効果がある最小のコスト T_{min} を 10000 とする. 簡単のため MTG_0 の DOALL2 と SB1 内部のマクロタスクグラフである MTG_1 上の LOOP1-1,LOOP1-2, LOOP-3 の内部には並列性がないものとする. 実際の解析では, これらの MT の内部で MTG が階層的に生成されるが, ここでは LOOP1-1, LOOP1-2, LOOP1-3 および DOALL2 内部の MTG は省略している. ネストレベルの深い階層から各 CP, CP_ALD, $Para$, $Para\_ALD$, $Para\_M$ を求める. 上記のように LOOP1-1, LOOP1-2, LOOP1-3 の内部の MTG には並列性がないのでこれらの LOOP に関しては $Para = Para\_ALD = Para\_M = 1$ である.

MTG_1 の逐次処理コスト Seq_1 は 30000, CP_1, CP_ALD_1 共に 10000 となる. したがって $Para_1 = 30000/10000 = 3$, $Para\_ALD_1 = 30000/10000 = 3$ となる. また MTG_1 内部の $Para\_M$ は, LOOP1-1, LOOP1-2, LOOP1-3 ともに 1 であり, $Para\_M_1 = \lceil Para\_ALD_1 \rceil \times Para\_M_{1\_1} = 3 \times 1 = 3$ となる. よって MTG_1 には最大で 3PE を割り当てればよいと分かる.

また, DOALL2 内部には並列性がないが, DOALL2 自体は, 並列処理可能最小コストを 10000 としたので, 60000 / 10000 = 6 より 6 分割できる. よって DOALL2 においては, $Para_2 = 1$, $Para\_ALD_2 = 6$ である. 次に MTG_0 について, $Seq_0 = 90000$, CP_0 は DOALL2 の Seq で 60000 である. DOALL2 が並列処理可能の最低コスト 10000 となるように分割された際の CP 長 CP_ALD_0 は SB1 のコストで 30000 となる. これより $Para_0 = 90000/60000 = 1.5$, $Para\_ALD_0 = 90000/30000 = 3$ となる. $Para\_M_0$ について, 内側 MT は SB1, DOALL2 であり, 計算上, DOALL2 を $Para\_ALD_0 = 3$ で分割した図 2を想定する. SB1 は $Para\_M_1 = 3$ である. 分割前の DOALL2 は $Para\_M_2 = 6$ であり, $Para\_ALD_0 = 3$ で分割されたものとするので DOALL2 のループ分割された MT のひとつは $Para\_M_2 = 6/3 = 2$ となる. 以上より MTG_0 の内部で $Para\_M$ が最大となるのは SB1 であり, $Para\_M_0 = \lceil Para\_ALD_0 \rceil \times Para\_M_1 = 9$ と

計算される.

3.3 並列度に基づく各階層の PC, PE 構成決定手法

本章では 3.2で得られた並列度を用いた, 各階層へのプロセッサ自動割り当て手法について述べる.

手順 1) 一般に上位階層の方が処理コストは大きく, スケジューリングオーバーヘッド, 同期オーバーヘッド も相対的に小さくなる. よって上位階層の並列性を優先して決定する. 現在注目している階層 (MTG_i) に既に割り当てられた総プロセッサ数を $Avail\_PE_i$ とし, MTG_i の PC 数を PC_i, PE 数を PE_i とする. MTG_i のみの並列性は $Para_i$, $Para\_ALD_i$ によって示され, $Para_i$ で示される粗粒度並列性を十分に生かすには $Para_i \leq PC_i$ でなければならない. また, MTG_i に $Para\_ALD_i$ を超える PC を割り当てると, 余分な PC 分, アイドル状態となる. すなわち, MTG_i の PC, PE の組み合わせの候補を $[PC_i,\ PE_i]$ とすると, これは $PC_i \times PE_i = Avail\_PE_i$ かつ $Para_i \leq PC_i \leq Para\_ALD_i$ となる最大の PC_i を持つ組合わせを割り当てればよい. ただし $Para_i = Para\_ALD_i$ のように PC_i のとり得る値に幅がない場合は, 純粋な粗粒度タスク並列性を保証するため, $Para_i \leq PC_i$ かつ $PC_i \times PE_i = Avail\_PE_i$ を満たす最小の PC_i を MTG_i の PC 数とする. $Para_i \geq Avail\_PE_i$ である場合は, 粗粒度タスク並列性を最大限に生かすため, $PC_i = Avail\_PE_i, PE_i = 1$ を MTG_i の PC, PE の組み合わせとする.

手順 2)MTG_i の内部で, 並列処理可能ループ以外の MT のうち最大の $Para\_M$ を $MaxPE_i$ とする. これは下位階層に割り当てるプロセッサの上限, すなわち PE_i の上限を表す. また, $MaxPCPE_i = Para\_M_i$ とする. これは MTG_i に割り当てる総プロセッサ数の上限である. ただし, $MaxPCPE_i$ が MTG_i 内部で使用する総プロセッサ数の上限を表すようにするため, MT_i 自体が分割可能ブロックの時はその分割数で $Para\_M_i$ を割ったものとする. $MaxPE_i < PE_i$ である場合は, 下位階層へ余分に PE 数を割り当てることになり, 無駄な同期やスケジューリングオーバーヘッドを増加させてしまう可能性がある. これを避けるため,$PE_i = MaxPE_i$ と決定し, 以下の処理に移る.

手順 3.a)MTG_i 内の MT が全て並列処理可能ブロックでないとき, 現在の候補 PC_i を割り当て PC と決定する.

手順 3.b)MTG_i 内に並列処理可能ループがある場合, 安易に MTG_i で使用できる総プロセッサ数を減らすと, このループを並列処理するために十分な数のプロセッサを確保できなくなる恐れがある. このため, MTG_i に割り当てる総プロセッサ数の上限である $MaxPCPE_i$ を上回るよう PC_i を増やす. つまり, $PC_i \times MaxPE_i \geq MaxPCPE$ となる最小の PC_i を捜す. ただし $PC_i \times MaxPE_i > Avail\_PE_i$ となる場合, $PC_i \times MaxPE_i \leq Avail\_PE_i$ となる最大の PC_i を求める.

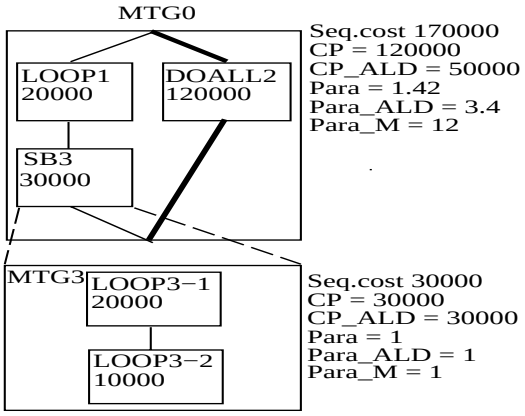

図 3 $Para, Para\_M$ を用いた PC, PE の決定

以上の手順を上位階層から順に繰り返し, $Avail\_PE_i = 1$ となるまで繰り返す.

実際に図 3の例を用いてプロセッサ数の割り当てを行う. 各 MTG の逐次処理コスト, CP, Para 等は図に示す通りである. MTG_0 は $Para_0 = 1.42 \leq PC_0 \leq Para\_ALD = 3.4$ の範囲で PC の候補を考えると $PC_0 = 2$ となり, この階層の [PC, PE] の組み合わせ候補は [2PC, 2PE] となる. MTG_0 で, 並列処理可能ループ以外の MT は LOOP1, SB3 であり, ともに $Para\_M = 1$ なので $MaxPE_0 = 1$ となる. これより MTG_0 の下位階層には 1PE を超えるプロセッサ数を割り当てる必要はないと判断され, $PE_0 = MaxPE_0 = 1$ と決まる. また, この階層に必要な総 PE 数は $MaxPCPE_0 = Para\_M_0 = 12$ となる. MT_0 内には DOALL2 があり, この階層で使用する総プロセッサ数を減らした場合, DOALL2 に十分なプロセッサ数が割り当てられなくなる可能性がある. ここで, $PC_0 \times MaxPE_0 \geq MaxPCPE$ を満たす PC 数は $PC_0 = 12$ であるが, この階層に割り当てられた総 PE 数は 4 なので最終的に [4PC, 1PE] がこの階層の構成となる. LOOP1, SB3 には 1PE が割り当てられ, これより下の階層はすべて 1PE で実行される. MTG_0 を [4PC, 1PE], MTG_3 を [1PC, 1PE] で実行した場合, LOOP1, SB3 を 1PE で処理している間に残り 3PE で DOALL2 を処理するので, 処理時間は (20000 + 30000) / 1 = 50000 となる. よって全体を処理する時間は 50000 となる. これに対して MTG_0 以下をループ並列処理した場合, DOALL2 を 4PE で実行するのにかかる時間は 130000 / 4 = 32500 であり, LOOP1, SB3 を実行する処理時間は (20000 + 30000) / 1 = 50000 となる. よって合計で 32500 + 50000 = 82500 となり, 本手法を用いた場合の処理時間 50000 より約 60% 大きな値をとる.

4. 性 能 評 価

本章では, 提案する並列処理階層自動決定手法を OSCAR マルチグレイン並列化コンパイラに実装し, その性能を 8 プロセッサ サーバ IBM RS6000 PowerPC 604e-

表 1　SPEC95FP 各ベンチマークの実行時間

benchmark	tomcatv	swim	su2cor	hydro2d	mgrid	applu	turb3d	fpppp
逐次処理	636.8	549.1	517.2	987.7	592.0	707.4	649.0	505.9
XLF 8PE	1180.5	130.6	941.9	620.7	344.8	489.9	2071.9	506.3
XLF 最高性能	373.0(3)	112.6(6)	197.9(4)	426.2(4)	193.0(4)	489.9(8)	649.0(1)	505.9(1)
OFC 8PE	118.5	64.0	138.9	127.0	93.6	12245	9874	16326
OFC(提案手法, 8PE)	107.2	64.4	120.1	116.7	192.2	423.7	197.9	506.0

＊OFC = OSCAR FORTRAN COMPILER, XLF = XL Fortran,( ) 内は用いた PE 数, 単位は秒である

High Node 上で評価する.

4.1 評価環境

本評価では, 提案手法を組み込んだ OSCAR コンパイラを並列化プリプロセッサとして用い, OpenMP を用いた粗粒度並列化プログラムを出力した.

出力されたプログラムを IBM RS6000 上での IBM XL Fortran Version 7 によるコンパイル後, 実行する. このマシンは 200MHz の Power PC 604e を 8 プロセッサ 搭載した SMP サーバであり, 1 プロセッサあたり, 32KB ずつの命令及びデータ L1 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.

4.2 SPEC95FP の評価結果

SPEC95FP のうち, swim, tomcatv, mgrid, hydro2d, applu, turb3d, su2cor, fpppp の 8 本を用いて提案手法の評価を行った. 評価対象として, 提案手法を用いない場合の OSCAR コンパイラで, ループ並列性をプロセッサ台数と等しい数の粗粒度タスクに変換する場合の自動並列化性能を挙げる. なお, 参考として XL Fortran Compiler Version 7 の自動ループ並列化性能も示す. Open MP ディレクティブを用いた OSCAR コンパイラの出力を, XL Fortran でコンパイルするときのオプションは, 提案手法を用いた場合と用いない場合ともに “-O3 -qsmp=noauto -qhot -qarch=ppc -qtune=auto -qcache=auto -qstrict” とし, XL Fortran のループ自動並列化の際のオプションは”-O5 -qsmp=auto -qhot -qarch=ppc -qtune=auto -qcache=auto” とした. また逐次性能評価のオプションは”-O5 -qhot -qarch=ppc -qtune=auto -qcache=auto” とした. OSCAR コンパイラ, XL Fortran コンパイラ共に, su2cor ではインライン展開と配列リネーミング,turb3d ではループディストリビューションを行ったソースを入力とした. また, 一部のプログラム中では, 通常のコンパイラがループ並列性を判断できるが, OSCAR コンパイラのバグのため並列性が検出できていない, いくつかのループに関しては, 並列化可能であることを示す OSCAR コンパイラ用ディレクティブを挿入した。

評価を行った SPEC95FP の各プログラムの実行時間を表 1 に示す. 上から XL Fortran の逐次処理, XL Fortran のループ並列化で 8PE を用いた場合, XL Fortran で 1PE から 8PE を用いた場合の最高性能, 提案手法を用いない場合の OSCAR コンパイラ (8PE), 提案手法を用いた OSCAR コンパイラ (8PE) である.

表 1 より swim に関しては XL Fortran の最高性能に比べ 1.75 倍の速度向上が得られているものの, 提案手法を用いた場合, 用いない場合ではほとんど性能差は見られない. これは swim のプログラムの大部分が並列処理可能ループ で構成されており, どちらの方法でも全プロセッサを各ループに割り当てれば, 十分に並列性を利用することができるためである. なお XL Fortran では, スレッド管理オーバーヘッドが大きいため, ループを並列実行する際に与えられたプロセッサ全てを効果的に用いることができず, 6PE が最高性能となっている.

tomcatv, hydro2d も swim 同様, 処理コストの大きな箇所の大部分は並列処理可能ループであるが, ループ並列処理出来ない箇所が一部存在する. 提案手法を用いた場合, 各階層の並列性を算出し, 並列性のない階層に対しては, その階層で利用可能な PE 数が 2 以上であった場合でも 1PE で逐次処理を行う. これにより無駄なオーバーヘッドが削除されるため, 提案手法を用いない場合に比べ実行速度が向上している. また, XL Fortran はスレッド管理オーバーヘッドが大きいため, OSCAR コンパイラで提案手法を用いた場合, XL Fortran に対して tomcatv では 2.15 倍, hydro2d では 3.65 倍の性能を得られた.

mgrid では表 1 より, 提案手法を用いない OSCAR コンパイラが 93.6 秒であるのに対し, 用いた場合は 192.2 秒となり, 性能が悪化している. mgrid において実行時間に大きく影響するサブルーチンは INTERP, RESID, PSINV であり, これらは並列処理可能なループ とサブルーチン COMM3 で構成されている. 提案手法を用いた場合に性能が悪化した原因は, INTERP, RESID, PSINV の階層で [8PC, 1PE] となり, COMM3 に 1PE のみを割り当てているからである. これは OSCAR コンパイラでのコスト推定の結果, COMM3 の処理コストを小さく見積もったためであり, 実際は COMM3 を並列処理することにより大幅な実行時間の短縮が見られることが確認されている. XL Fortran に関してはこれもプロセッサに見合うほどの性能は得られず, 最高性能は 4PE 時にとどまっている.

su2cor では提案手法を用いた場合が 120.1 秒, 用いない場合が 138.9 秒である. この差は, 提案手法を用いない OSCAR コンパイラでは, ネストの深い階層に存在する比較的処理コストの小さい並列処理可能ループによるループ並列性のみを用いるが, 提案手法ではサブルーチン LOOPS の, DO 400 のループ内の階層に存在する, 規模の大きな粗粒度タスク並列性を用いたためである. この階層は $Para = 1.90$, $Para\_ALD = 3.00$ であり, 提

案手法を用いた場合, この階層で [2PC, 4PE] とし, この並列性を有効に利用できることが確認された.

turb3d ではサブルーチン TURB3D 内の逐次ループが実行時間の大部分を占める. この RB はループ並列性がほとんどないが, 粗粒度並列性は高く $Para = 5.98$ であり, 提案手法を用いると $Para \leq PC$ の範囲で組合わせを選び, [8PC, 1PE] が選択される. ループ並列処理のみで実行する場合, この部分の粗粒度並列性を用いることができず, さらに下位階層のループ並列性を用いようとする. しかしこれ以下の階層にもプログラムの全体の実行時間に大きく影響する並列処理可能ループは存在せず, 提案手法を用いない場合の OSCAR コンパイラでは, 実装上のバグのため 9874[s] と極めて大きい処理時間となり, XL Fortran においても複数のプロセッサを用いた方が逐次処理よりも遅くなるという結果となった.

applu においてはサブルーチン JACLD, JACU, RHS は高い並列性を持つと判断されたが, これ以外のサブルーチンには並列性が全く見られず, 提案手法を用いることで, これらのサブルーチンを並列処理し, 他を逐次処理することによって 1 プロセッサで全体を逐次処理した場合に対して, 1.67 倍の速度向上が得られた. 手動による解析の結果, サブルーチン blts, buts にはパイプライン並列性が存在することが確認されているが, 現在の OSCAR コンパイラでは自動判定できなかったためこの並列性は利用していない. 提案手法を用いない OSCAR コンパイラはバグにより冗長なバリア同期が挿入されてしまうためオーバーヘッドが大きく, 実用的な時間内では終了しなかった.

fpppp は main 階層で $Para\_M = 1$ となりこの階層以下に並列処理の出来るブロックは全くないと判断された. よって第 1 階層で (1PC, 1PE) で実行するという判断が下された. fpppp は文献 10) より, 近細粒度並列処理を用いることで性能向上することが確認されているが, 今回用いたような SMP においては逐次処理が最適であると考えられる. 表 1, XL Fortran の逐次処理と 8PE を用いた場合はほぼ同等の性能となった. これは XL Fortran では 8PE を 割り当ててもループ並列処理を行う箇所がないためである. 提案手法を用いた OSCAR コンパイラでは main 階層 で 1PE しか使わない (逐次処理をする) と判断したため, XL Fortran とほぼ同等の結果となっている. 提案手法を用いない場合, applu と同様に多大なバリアオーバーヘッド等ため実行時間が逐次処理に比べ大幅に遅くなるという結果となった.

5. ま と め

本論文では, 階層的粗粒度タスク並列処理における並列処理階層および各階層での使用プロセッサ数自動決定手法を提案した. SPEC95FP ベンチマーク中 7 本のプログラムを用いて, SMP サーバ IBM RS6000 SP 604e High Node 上で提案手法を評価した結果, 提案手法を用いない OSCAR コンパイラに対して tomcatv で 1.11 倍, swim で 0.99 倍, su2cor で 1.16 倍, hydro2d で 1.09 倍, mgrid で 0.49 倍, の速度向上となった. また, OSCAR コンパイラがバグ等の影響もあり, 逐次処理より遅くなったプログラムに関しても, 逐次処理に対して applu で 1.67 倍, turb3d で 3.28 倍, fpppp で 1.00 倍と, 並列処理すべき階層, すべきでない階層を適切に判断し, 並列性に見合ったプロセッサを割り当てることができた.

また, 今回は粗粒度タスク並列性のみの処理の評価であったが, 開発中のデータローカライゼーションによるキャッシュ最適化手法と組み合わせ, より効率的な並列処理を実現することが今後の課題である.

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 IT21 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) Wolfe, M. : “High Performance Compilers for Parallel Computing”, Addison-Wesley Publishing Company. 1996.
2) M.W.Hall, el al.: “Maximizing Multiprocessor Performance with the SUIF Compiler”, IEEE Computer, Dec. 1996.
3) R.Eigenmann, el al.: “On the Automatic Parallelization of the Perfect Benchmarks”, IEEE Transaction on parallel and distributed systems,Vol.9, Jan. 1998.
4) 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”, 情報処理学会誌, Vol.35, No. 4,pp. 513-521. 1994.
5) H. Kasahara and M. Okamoto and A. Yoshida and W. Ogata and K. Kimura and G. Matsui and H. Matsuzaki 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.
6) Kasahara, H. et al.: “A Multi-grain Parallizing Compilation Scheme on OSCAR” Proc. 4th Workshop on Language and Compilers for Parallel Computing. 1991.
7) 山本 晃正, 他: OSCAR マルチグレイン並列化コンパイラにおける階層的並列処理手法, 情報処理学会第 58 回全国大会, 2D-04 Mar. 1999.
8) 山本 正行, 他: マルチグレイン並列処理における階層的並列処理のためのプロセッサクラスタリング決定手法, 情報処理学会第 60 回全国大会, 4J-05 Mar. 2000.
9) 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.
10) 木村 啓二, 間中 邦之, 尾形 航, 岡本 雅巳, 笠原 博徳: シングルチップマルチプロセッサ上での近細粒度並列処理の性能評価, 情報処理学会研究報告 ARC134-4, pp19-24, Aug. 1999.