計算機アーキテクチャ　147－33
ハイパフォーマンスコンピューティング　89－33
（2002．3．8）

粗粒度並列性抽出のための解析時インライニングとフレキシブルクローニング

熊澤慎也[†]　石坂一久[†,††]
小幡元樹[†,††]　笠原博徳[†,††]

本論文では，粗粒度タスク並列性の更なる抽出を目的として，解析時のインライン展開，並列性解析，フレキシブルクローニングを組み合わせたプロシージャ間並列性解析手法を提案する．本手法は，解析を目的としてコンパイラ内部でインライン展開を施し並列性解析を行った後，解析された並列性を失わないように並列性のない部分を“フレキシブルクローニング”すなわちオリジナルソースあるいは別な形のサブルーチンに変換し，コード量の増加を抑える．この並列性解析手法により，生成コードの過度な増加を抑えつつグローバルな粗粒度タスク並列性を有効に引き出すことができる．SUN Ultra80 4プロセッサSMPワークステーション上でPerfect Club BenchmarkのARC2Dを用いて性能評価を行った結果，提案手法により，SUN Forteコンパイラの自動並列処理に比べて，本手法を適用しマルチグレイン並列化を行うことで4プロセッサ上で約15%のスピードアップが得られ，またコードサイズでもインライン展開のみを用いるとソースプログラムに対しオブジェクトコードで26.8%のコード量増大になるのに対し，フレキシブルクローニング手法を用いることで，コード増加量を14.8%に抑えられていることが確認された．

An Analysis-time Procedure Inlining and Flexible Cloning Scheme for Coarse-grain Automatic Parallelizing Compilation

Shinya Kumazawa,[†] Kazuhisa Ishizaka,[†,††] Motoki Obata[†,††]
and Hironori Kasahara [†,††]

This paper proposes an interprocedural parallelism analysis scheme which combines analysis-time inline expansion and flexible cloning for coarse-grain parallelization. The analysis-time inlining is applied to selected subroutines. After the analysis of global parallelism over procedures, compiler generates inlined code for program part having global parallelism or applies “flexible cloning” to program parts without global parallelism into the original shape or different shape of subroutine. With this scheme, the compiler can exploit global coarse-grain with minimum increase in the code size. Performance evaluation using benchmark program ARC2D on SUN Ultra80 shows the proposed scheme gives us maximum 15% speedup than automatic parallelization of SUN Forte compiler. And by using flexible cloning, increase of code size has reduced by 14.8% from the case which doesn't use it.

1. はじめに

マルチプロセッサシステム用Fortran自動並列化コンパイラにおいては，従来よりループ並列化手法が広く用いられている．しかし，ループ並列化手法では従来までの多くの研究によりすでに成熟期に入っており今後大幅な性能向上が望めない．また現時点でもプロセッサの有効利用が困難ということを考えると，今後のプロセッサ数の増加と共にスケーラブルな実効性能向上を達成することは困難であると考えられる．この問題に対処するために，基本ブロック，ループ，およびサブルーチン間の粗粒度並列性を利用するためのアプローチとして粗粒度タスク並列処理[1)]，ループイタレーション間の中粒度並列処理，ステートメント間の近細粒度並列処理を階層的に組み合わせて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提案されている[2)]．この粗粒度タスク並列処理においては，サブルーチン内外の粗粒度タスク間並列性を抽出するためにインタープロシージャ解析が必須となる．インタープロシージャ解析は，ループ並列性を抽出するためにも従来から多くの手法が提案されており，配

† 早稲田大学 理工学部 電気電子情報工学科
Department of Electrical, Electronics and Computer Engineering, School of Science and Engineering, Waseda University
†† Advanced Parallelizing Compiler Project

列添字の詳細とループ範囲をコールサイトへ伝搬するアトムイメージ法[3]や，ループ境界とループストライドからサブルーチン内での参照配列部分を決定しコールサイトへ伝搬する *Bounded Regular Section* 解析法[4]，コールサイトコンテキストの相違による解析精度の低下を防ぐためにサブルーチンを選択的にクローニングする手法[5][6]などが提案されている．しかし，アトムイメージ法では実引数及び仮引数整合配列の次元が異なる場合サブルーチン側での情報をコールサイト側での情報に変換できず解析できない．また，*Bounded Regular Section* 解析法では大きくコールサイトコンテキストの異なるものが存在する場合に解析精度が低下し，選択的クローニング手法ではコードサイズが増大するなどの問題がある．これらの問題を解決するためにアトムイメージ法と配列の一次元化を組み合わせた手法[7][8]も提案されているが，粗粒度タスク間並列性の抽出は考慮されていない．そこで本論文では，インタープロシージャ解析の精度をあげつつ，サブルーチン内外の粗粒度タスクの並列性の解析を行うために，コンパイラにおける解析時にのみインライン展開し並列性抽出を行った後，並列性及びコードサイズを考慮し元のサブルーチンあるいは他の形のサブルーチンにフレキシブルに戻し，解析精度を向上させつつコードサイズの過度な増大を抑える手法を提案する．

以下，2章で，OSCAR コンパイラで行っているマルチグレイン処理の概要を述べ，本論文で扱っている解析時インライニングとフレキシブルクローニングについて述べる．3章で，Perfect Club Benchmark の ARC2D を用いて本手法の適用例を説明し，4章で SUN Ultra80 上での性能評価を述べる．

2. 本手法の概要

2.1 OSCAR コンパイラにおけるマルチグレイン並列処理

OSCAR マルチグレイン自動並列化コンパイラ[2]では，プログラムを次の 3 種類のマクロタスク (MT) に分割する．

- BPA　基本ブロック及びこれを複数融合したブロック
- RB　最外側ナチュラルループ
- SB　サブルーチン

OSCAR コンパイラにおける粗粒度タスク並列処理はこれらの MT 間の並列性を用いた処理手法である．また中粒度並列処理はループイタレーション間の並列性を用いた処理であり，近細粒度並列処理は BPA 内の各ステートメントをタスクと定義してそのタスク間の並列性を用いた並列処理である．そして，この 3 つの粒度の並列処理を階層的に組み合わせたものがマルチグレイン並列処理である．

プログラム中のネスト構造に対応し MT は階層的に定義され，コンパイラによりソフト的に構成されるプロセッサクラスタ (PC) 上で並列処理される．また各階層で PC に割り当てられた MT は，さらに PC 内のプロセッサエレメント (PE) あるいは下位階層の PC により階層的に並列処理される．例えば MT が RB で DOALL ループならば中粒度並列処理を，ループボディが大規模で基本ブロックから成る逐次ループの場合は近細粒度並列処理を，また RB がボディ部にループ，サブルーチンコール等を含む大規模逐次ループである場合にはループボディ部に対し階層的に粗粒度タスク並列処理を適用する[1][2]．

以下では，サブルーチン内外で効果的な粗粒度タスク並列処理を行うためのインタープロシージャ解析法として，高精度で行うための解析時インライニング[9]とコード増大を抑えるためのフレキシブルクローニングについて述べる．

2.2 粗粒度並列性抽出のための解析時インライニング およびフレキシブルクローニング

2.2.1 適用する SB の選定とインライニング

解析時インライニングはサブルーチン内外のグローバルな並列性が抽出できる可能性のあるサブルーチンに対して適用される．解析時インライニングを適用する SB の選定は以下のように行う．

1) まず最初にメインルーチンをターゲットの階層に選ぶ
2) ターゲットの階層中，最大処理時間を持つ SB を選ぶ
3) 選ばれた SB が，ターゲット階層の処理時間の 90%以上の処理時間を占める場合は，SB 内部と外部との粗粒度並列性を効果的に抽出することは期待できないと考え，SB の内部を次のターゲットの階層に選んで，再び 2) の手順から行う
4) 選ばれた SB が，3) に合致せず，アトムイメージ法で解析が行えない場合，その SB をインライニング対象の SB に選ぶ
5) マクロタスクグラフ (MTG) 上で，4) で選ばれた SB とデータ依存エッジで接続されている SB をすべてインライニング対象の SB に加える

5) でインライニング対象の SB とデータ依存のある SB を選ぶ理由は，4) で選ばれたサブルーチンの内部の MT とデータ依存を持つ SB から呼ばれるサブルーチン内部の MT との間でグローバルな粗粒度並列性を抽出できる可能性があるためである．この条件を満たした SB は並列性解析のために中間語レベルで展開される．

2.2.2 並列性解析

この解析時インライニングの使用により，引数情報の明確化，コールサイトコンテキストを用いた定数，シンボリック値の伝搬，無用コードの削除などを行うことが容易となり，インライン展開を用いない従来の

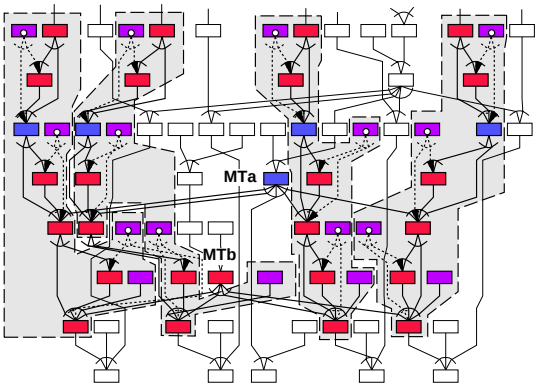
インタープロシージャ解析手法に比べ，より多くの粗粒度タスク間並列性を抽出することが可能となる．また粗粒度並列性の解析時には，OSCAR マルチグレイン自動並列化コンパイラの最早実行可能条件解析 [10)1)] を使用する．

2.2.3 フレキシブルクローニング

並列性解析の終了後，このインライニングによってより多くの並列性を抽出できたかどうかを検査する．この検査においては，まず最早実行可能条件解析の結果から作成された当該階層の MTG に対してその MTG の並列度 Para を下式にて計算し，展開前の Para と比較することにより並列性が増加したかを判断する．並列度 Para は以下の式で表される．

$$\text{Para} = \frac{\text{1PC での逐次処理時間}}{\text{MTG のクリティカルパス長}}$$

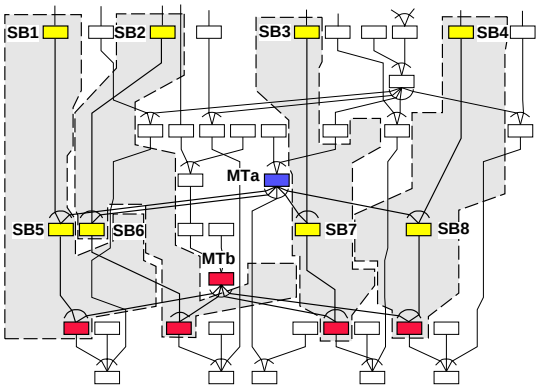
次に，展開によりできた各 MT について並列性の向上に寄与しているかどうかを検査する．これは，Para が当該階層 MTG 全体における並列度を示すものであるため，全体として Para が増加していたとしても個々の SB の展開でできた MT の中には Para の増加に寄与していないものが存在する可能性があるからである．この検査によって，並列性増加に寄与していないと判断された MT 集合はフレキシブルクローニングにより元のサブルーチンあるいは異なる形のサブルーチンへ変換される．



内部のマクロタスクは解析時インライニングにより生成されたマクロタスク群

図 1 解析時インライニング直後の MTG の例

ここでは，フレキシブルクローニングのための部分並列性解析とクローニング例を図 1と図 2を用いて説明する．図 1は Perfect Club Benchmark 中のプログラム ARC2D におけるサブルーチン INTEGR の MTG の一部であり，サブルーチン FILERX を解析時インライニングした後 4 回転のループをアンローリングした場合を表している．図中点線で囲まれた 4ヶ所の部分は上記のアンローリングされた 1 イタレーション内の



部はフレキシブルクローニングにより生成されたサブルーチンブロック

図 2 図 1に対しフレキシブルクローニングを適用した後の MTG

MT 群を表している．これらの MT 群は，1RB 中のループボディ部の MT であったが，解析時インライニングとアンローリングにより，以前はサブルーチン外にあった多くの MT との間に並列性が引き出されまたサブルーチン外部にあった MTa, MTb とのデータ依存が明確に解析できていることがわかる．

次にコード量の増大を避けるためのフレキシブルクローニングを適用する．そのために解析時インライニング後に生成された MTG を出口ノードから辿り，解析時インライニングを施した結果生まれた MT 群の中で以前はサブルーチン外であった MT との間でデータ依存が生じているものを捜す．前述のように，図 1の例では各網掛け部分には外部の 2 つの MT（MTa と MTb）からの依存が生まれている．そこで新しく依存が生じた MT を入口ノードとして，再び外部から依存がある MT まで上方向に辿っていくことで MT グループを生成し，そのグループの MTG（部分 MTG）に対して部分的な Para の値を計算し，この部分 MTG が全体の MTG の Para の増加に寄与しているかどうかを調べる．図 1の場合は，その部分が元々ほぼ一直線の部分 MTG であり Para が約 1.00 と並列性がないので，全体の MTG の Para の増加に寄与していないと判断される．従って，この部分 MTG はフレキシブルクローニングの対象になりサブルーチンへと変換される（SB1〜SB8）．このフレキシブルクローニングを適用した結果は図 2の様になり，全体の並列性を維持したまま MTG が簡素化されたことがわかる．クローニングの際，クローニングする部分が同じコードを共有しているのであればクローニングの結果生まれるサブルーチンは一つだけとなり，インライン展開の副作用であるコード量の増大を最小限に抑えることができる．

このフレキシブルクローニングにより，サブルーチン内部の MT を上位階層 MT へ変換することにより，サブルーチン内部に隠されているグローバルな並列性

を引き出すことができる．

3. 解析時インライニングおよびフレキシブルクローニングの適用例

ここでは提案手法を Perfect Club Benchmark 中の ARC2D への適用例を通して具体的に説明する．

3.1 解析適用 SB の決定とインライニング

まず， 2.2.1節の 1) から 5) で解析時インライニングを適用する場所を決定する．

まず， 2.2.1節の 2) に沿って，メインルーチン中で，最も大きな推定コストの SB である INTEGR が選ばれるが，この SB はメイン階層の実行時間の約 95%を占め， 2.2.1節の 3) に合致する．したがって INTEGR はインライニング対象の SB には選ばず，次に INTEGR の内部の階層に入って再び調査する．図 3に INTEGR の内部の MTG を示す．

再び 2.2.1節の 2) から，INTEGR の内部の階層で最も大きな推定コストの SB である STEPFX が選ばれる．この SB は推定コストが INTEGR 全体の 50%ほどであり，今度は 2.2.1節の 4) に合致し，インライニング対象 SB として選ばれる．次に 2.2.1節の 5) を行い，INTEGR 内のデータ依存を調べると，図 3 から，すべての SB が STEPFX へ直接あるいは間接的にデータ依存を持つことが分かる．したがって，INTEGR 内のすべての SB がインライニング対象の SB として選ばれる．

インライニング後の MTG の対して再び 2.2.1節の 5) を行う．サブルーチン STEPFY のインライン展開により INTEGR 内に現れた SB が 2.2.1 節の 5) に合致して選ばれ，再度インライン展開が行われる．この 2 回目の展開後再度選定を行うと，既にこの階層にはサブルーチンコールは 1 つもなく選定及び解析時のインライン展開は終了する．

3.2 並列性解析とフレキシブルクローニング

インライン展開後，コールサイトコンテキストを使ったシンボリック値（定数を含む）の伝搬や無用コードの削除など並列性増加のための各種最適化を行った後，データ依存解析及び並列性の解析を行う．

INTEGR の場合，図 4に示す STEPFY から YPENTA の呼び出しが重要である．インライン展開を用いない場合，STEPFY 内の配列 S の次元は 3 次元であり，YPENTA の配列は 2 次元であるので，YPENTA 内の S に関する参照情報を STEPFY 側に伝搬することができず依存解析ができない．

これに対し解析時インライニングを用いる本手法では，繰り返し解析時インライニングを適用することにより STEPFY はインライン展開され，また YPENTA 及び YPENT2 もインライン展開されている．この結果，インライン展開により一次元化された配列添字を検査することができ，結果，配列 S には依存がないことが分かる．

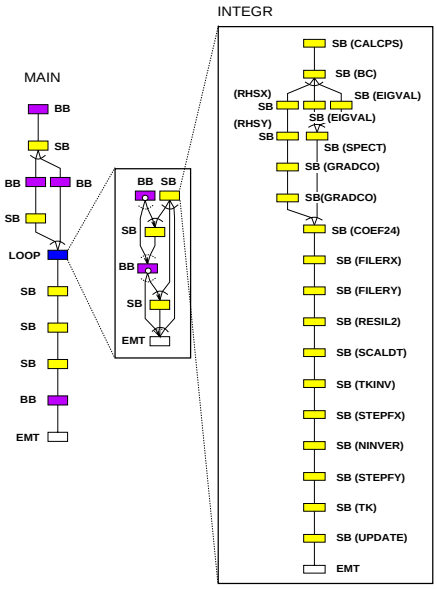


図 3 サブルーチン INTEGR の MTG

```
SUBROUTINE STEPFY(.....)
DIMENSION S(JDIM,KDIM,4)
DO N=2,4
 .....
 IF(N.EQ.2)THEN
  CALL YPENT2(..,S(1,1,N),..)
 ELSE
  CALL YPENTA(..,S(1,1,N),..)
 ENDIF
ENDDO

SUBROUTINE YPENTA(..,F,..)
DIMENSION F(JDIM,KDIM)
```

図 4 サブルーチン STEPFY における YPENTA の呼び出し

次に 2.2.3 節で述べたフレキシブルクローニングを適用する．図 5(a) は，STEPFY をインライン展開した後，ループアンローリングした図である．図 5を見ると，MT5, MT10, MT15 に外部から新しく依存ができていることが分かる．そこで，MT5 を出発点として，MTG を上方向に辿っていくと，MT1 から MT4 までがデータ依存で一直線上につながれており並列性が存在しないので，フレキシブルクローニングが適用される．同様に図 5(a) の MT6 から MT9，そして MT11 から MT14 にも適用され，図 5(b) のようになる．このクローニングの際，MT1 から MT4，MT6 から MT9，そして MT11 から MT14 は図 4の N のループをアンローリングしてできたものでありコード自体に違いがないため，クローニングによって新しくできた 3 つの SB(MT1', MT6', MT11') は同じサブルーチンを呼ぶようになり，コードサイズの増加が抑止できる．

本手法を適用した後の MTG を図 6に示す．この

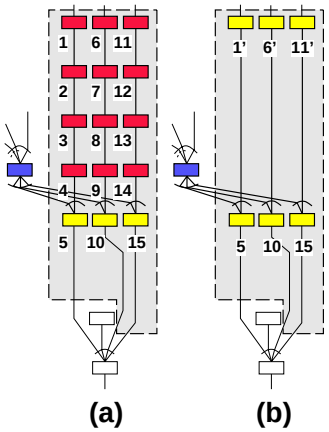


図 5 STEPFY 相当部分に対するフレキシブルクローニングの適用

表 1 SUN Ultra80 4 プロセッサ SMP

	SUN Ultra80
CPU	UltraSparcII 450MHz x4
L1 cache(I/D)	16Kbyte/16Kbyte
L2 cache	4Mbyte
Main Memory	1Gbyte
Native Compiler	SUN Forte version 6 update 2

MTG を見ると，インライニング前に比べて多くの粗粒度並列性が抽出できていることが分かる．解析時インライニングを行った INTEGR 内の para を計算したところ約 3 あり，粗粒度並列性が十分に引き出せたことがわかる．

4. 性能評価

本節では，3 節で本手法の説明に使用した ARC2D を使い，SUN 4 プロセッサ SMP ワークステーション Ultra80 上で手法の評価を行った．ARC2D はオイラー法を用いた 2 次元流体解析プログラムであり，コード量は約 4500 行サブルーチン数 40 である．なお，本手法の評価では，OSCAR コンパイラを通して本手法の適用を行った後，OpenMP ディレクティブを用いて並列化した Fortran コードを出力し，その Fortran コードを SUN Forte コンパイラ version 6 update 2 でオブジェクトに落とし実行した．SUN Ultra80 ハードウェアの詳細を 表 1に示す．

まず，表 2に Forte の自動並列化および本手法を用いて実行した結果を示す．なお，OSCAR コンパイラを用いたときのプロセッサ構成としては，ループ並列処理の評価として pc 数を 1 にして pe 数を 1 から 4 にしたとき，粗粒度並列処理の評価として pe 数を 1 にして pc 数を 1 から 4 にしたときに加え，そしてマルチグレイン処理の評価として，2pc2pe のときの評価を行った．Forte のコンパイルオプションは，Forte の逐次実行には -fast を，Forte の自動並列処理の場合に -fast -parallel -reduction -stackvar を，OSCAR コンパイラを通して本手法を用いた場合は，-fast -mp=openmp -explicitpar -stackvar を利用した．

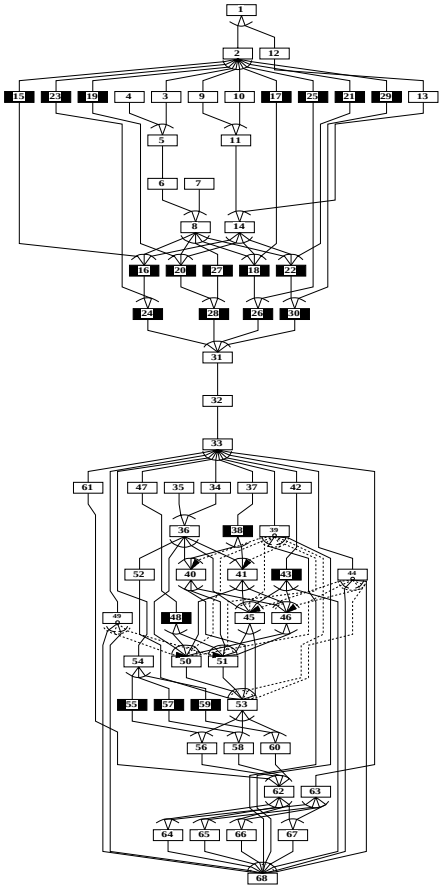


図 6 本手法を用いた後の INTEGR の MTG
(黒く塗られている MT がフレキシブルクローニングにより生成された SB)

表 2 ARC2D の実行時間

pe	実行時間 sec	
	forte	oscar (階層構成)
1	12.457	12.553 (1pc1pe)
2	9.632	9.528 (1pc2pe) 8.982 (2pc1pe)
3	8.702	8.965 (1pc3pe) 8.493 (3pc1pe)
4	8.763	8.624 (1pc4pe) 8.339 (4pc1pe) 7.560 (2pc2pe)

まず，粗粒度並列性を利用して並列実行した場合 (nPC1PE) と，ループ並列性を利用して並列実行した場合 (1PCnPE) を比べてみると，例えば 1PC4PE が 8.624 秒，4PC1PE が 8.339 秒というように各プロセッサ台数の時に粗粒度並列性を利用した場合の方が速くなっていることが分かる．ARC2D は，ループ並列性に富んだベンチマークであるが，各ループ処理コ

ストが小さいので，並列処理オーバヘッドが相対的に大きく性能が向上していない．一方，粗粒度並列性を用いるとループ並列性に比べて大きい粒度で並列性を利用できるので，並列処理オーバヘッドが軽減し，その結果粗粒度並列性を利用した方が性能が向上しているのである．

また，2PC2PE でマルチグレイン並列化を行うと，7.560 秒と一番高い性能向上を示しており，今回の DOALL ループの粗粒度タスクへの変換を行わない場合の評価では INTEGR の para が 3 程度となったため，プロセッサ 4 台を使って粗粒度並列化を行うよりも，2PC で粗粒度並列化を行い PC 内でループ並列化を行うマルチグレイン並列化により効率よくタスクが処理できることを示している．このとき，Forte の自動並列化による処理時間は 8.763 秒で，マルチグレイン並列化により 1.15 倍の速度向上率が得られている．

また，サブルーチンの境界を越えた粗粒度並列性抽出と並ぶもう一つの大きな目標であるコードサイズ増加の抑制についてであるが，インライン展開を適用しない場合，インライン展開を適用してそのまま出力した場合と，本フレキシブルクローニングを適用した場合について，1PE 用のオブジェクトコードを出力させそのサイズを比較した．解析時インライン展開を用いない場合が約 473Kbyte に対し，解析時インライン展開のみを適用し，フレキシブルクローニングを適用しない場合が約 600Kbyte，本フレキシブルクローニング手法を用いた場合が約 543Kbyte となり，本フレキシブルクローニング手法を用いた場合は，インライン展開を用いない場合に比べると約 14.8%コード量が増大しているものの，インライン展開だけを用いた場合に比べると約 10%コード量が削減できることが確かめられた．

5. ま と め

本論文では，マルチグレイン自動並列化コンパイラにおける粗粒度タスク間の並列性抽出のためのインタープロシージャ解析手法として，推定処理コストが大きく，サブルーチン内外で粗粒度並列性が効果的に抽出できるサブルーチンコールに対して中間コードレベルでインライン展開を施した上で並列性の解析を行い，粗粒度並列性を失わないように，オリジナルと同一あるいは別形状のサブルーチンにフレキシブルクローニングを行う，解析時インライニング及びフレキシブルクローニング手法を提案した．Perfect Club Benchmark 中の ARC2D に本手法を適用し，SUN Ultra80 4 プロセッサワークステーション 上で性能評価を行った結果，SUN Forte コンパイラの自動並列化に比べ，本手法を用いたマルチグレイン並列処理化により 4 プロセッサで約 15%の性能向上が得られた．また，フレキシブルクローニングにより，フレキシブルクローニングを適用しなかった場合よりも，オブジェクトコードで約 10%のコードが削減されることが確かめられた．今後，今回適用しなかった並列ループ分割による粗粒度タスク化，ならびにデータローカライゼーションによるキャッシュ最適化と組み合わせ，より多くのアプリケーションに対する性能を評価していく予定である．

本研究の一部は，経済産業省の 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) 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, pp. 910–920 (2001).
2) Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K. and Narita, S.: Multi-Grain Parallelizing Compilation Scheme for OSCAR, *Proceedings of 4th Workshop on Language and Compiler for Parallel Computing* (1991).
3) Li, Z. and Yew, P.-C.: Program Parallelization With Interprocedural Analysis, CSRD Technical Report 775, Center for Supercomputing Research & Development, University of Illinois (1988).
4) Havlak, P. and Kennedy, K.: An Implementation of Interprocedural Bounded Regular Section Analysis, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 3, pp. 350–360 (1991).
5) Hall, M., Amarasinghe, S., Murphy, B., Liao, S. and Lam, M.: Detecting Coarse-Grain Parallelism Using an Interprocedural Parallelizing Compiler, *Proceedings of Supercomputing '95* (1995).
6) 佐藤真琴, 青木雄一郎, 和田清美, 太田寛, 飯塚孝好: 手続き間自動並列化コンパイラ WPP の評価, 情報処理学会 ARC 研究報告, Vol. 141, No. 4, pp. 17–22 (2001).
7) Hind, M., Burke, M., Carini, P. and Midkiff, S.: An Empirical Study of Precise Interprocedural Array Analysis, *Scientific Programming*, Vol. 3, No. 3, pp. 255–271 (1994).
8) Hoeflinger, J. and Paek, Y.: Unified Interprocedural Parallelism Detection, *International Journal of Parallel Processing* (2000).
9) 吉井謙一郎, 松井巌徹, 小幡元樹, 熊澤慎也, 笠原博徳: マルチグレイン自動並列化のための解析時インライニング, 情報処理学会 ARC/HPC 研究報告 (2000).
10) 笠原博徳: 並列処理技術, コロナ社 (1991).