計算機アーキテクチャ 130－8
（1998. 8. 5）

手続き間自動並列化コンパイラ WPP の試作 - 実機性能評価 -

青木 雄一郎† 佐藤 真琴† 飯塚 孝好† 佐藤 茂久‡ 菊池 純男†

†新情報日立研究室
‡新情報つくば研究センタ

プログラムからより多くの並列性を抽出するため、我々は手続き呼び出しを含むループを並列化可能な、共有メモリ型並列機向け手続き間並列化コンパイラ WPP (Whole Program Parallelizer) を開発中である。WPP では、これまでに手続きクローニングを伴う手続き間定数伝播、線形不等式系による手続き間配列参照領域解析、手続き間条件付き終値保証などを実現し、SPECfp ベンチマークで、米国 Stanford 大の SUIF コンパイラと同程度の並列化能力を実証した。実機評価の結果、条件付き終値保証を除けば実行時オーバーヘッドは小さく、SPEC92fp の mdljdp2、mdljsp2 では、プロセッサ 4 台で逐次実行の 1.75 倍、1.85 倍の並列実行性能を得た。

Prototyping of Interprocedural Parallelizing Compiler "WPP" - Performance Evaluation -

Yuichiro AOKI† Makoto SATOH† Takayoshi IITSUKA†
Shigehisa SATOH‡ Sumio KIKUCHI†

†RWCP Hitachi Laboratory
‡RWCP Tsukuba Research Center

To extract greater parallelism from programs, we are developing an interprocedural parallelizing compiler "WPP(Whole Program Parallelizer)" for shared-memory multiprocessors. We implemented interprocedural constant propagation with procedure cloning, interprocedural array region analysis using linear inequality system, interprocedural conditional last value assignment (ICvl) on WPP. Parallelizability of WPP for SPECfp benchmark suite is almost the same as that of SUIF compiler. Evaluation indicates that the execution overheads of parallelized program by WPP are small except that of ICvl. Parallelized programs of mdljdp2 and mdljsp2 of SPEC92fp on 4PE respectively run 1.75 times and 1.85 times faster than serialized ones.

1 はじめに

近年、大きな計算能力を容易に得る手段として、プロセッサを数台～数十台接続した共有メモリ型並列機が注目されている。データや処理の分散をユーザ自身が考えなければならない分散メモリ型並列機と異なり、共有メモリ型並列機では、逐次プログラムをコンパイラが自動並列化するので、既存の逐次プログラムから簡単に高性能を得ることができる。

自動並列化ではプログラムからより多くの並列性を抽出する技術が重要である。従来の並列化コンパイラは、手続き内の情報のみを用いて並列性解析を行なっていた。そのため、手続き呼び出しを含むループでは呼び出し先手続きでの変数参照関係等の情報がわからず、人間が見れば容易に並列化可能なループでも並列化できなかった。そこで我々は、手続き呼び出しを含むループも並列化可能な共有メモリ型並列機向け手続き間並列化コンパイラ WPP (Whole Program Parallelizer) を開発中である。

本稿では、WPP の機能、並列化能力、並列実行性能に関して述べる。

2 WPP の構成と機能

WPP は、既存の並列化支援システム [1] の手続き間スカラ・配列データフロー解析と別途開発中の手続き内並列化コンパイラを基に、手続きクローニングを伴った定数伝播、手続き間 DO ALL 並列化、手続き間リダクション並列化、手続き間プライベー

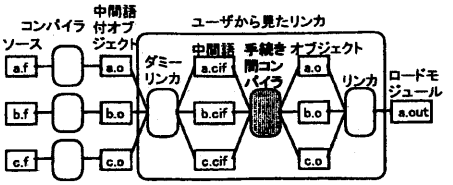

図 1: WPP の外部構成

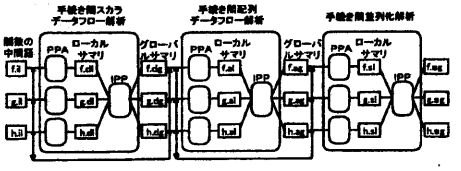

図 2: WPP 手続き間解析部の内部構成

ト化を組み込んだ、共有メモリ型並列機向け手続き間並列化コンパイラである。

図 1は WPP の外部構成である。WPP は、既存のプログラムを既存のユーザインターフェースで手続き間並列化できる外部構成を持つ。オブジェクトコードには各ソースプログラムに対応する中間語が埋め込まれており、プログラムの構成手続きが全て揃う「ユーザから見たリンク時」に手続き間コンパイラが起動する。「ユーザから見たリンカ」の内部では、手続き間コンパイラが、オブジェクトコード内の中間語を用いて手続き間解析・手続き間並列化を行ない、中間語を埋め込まないオブジェクトコードを作成し、リンカがそれらをリンクしてロードモジュールを作成する。

図 2は WPP の手続き間解析部の内部構成である。各手続き間処理では、PPA(Per-Procedure Analysis) で手続き毎の解析情報であるローカルサマリを収集し、IPP(InterProcedural Propagation) でローカルサマリを手続き間伝播し、プログラム全体の情報を反映したグローバルサマリを作成する。制御フロー依存な解析情報は、コールグラフに沿ってトップダウンまたはボトムアップに伝播してグローバルサマリを作成する。

以下で、手続き間スカラデータフロー解析、手続き間配列データフロー解析、手続き間並列化の機能を説明する。

2.1 手続き間スカラデータフロー解析

手続き間スカラデータフロー解析 [3] は、スカラ変数について、制御フロー依存なデータフロー解析である手続き間別名解析、手続き間参照変数解析を行なう。また、手続きクローニングを伴った手続き間定数伝播も行なう。本稿では、手続きクローニングを伴った手続き間定数伝播について述べる。

WPP での手続き間定数伝播の目的は、配列の寸法・添字、ループの上下限・増分などに使用されている変数の値を確定し、並列化を促進するのが目的である。手続きクローニングも、単に定数の数を増やすのではなく、並列化の促進が目的である。

手続き間定数伝播は以下のように実施する。まず、コールグラフ上のボトムアップ伝播によって、各手続きでの変数の値への作用を要約する。次に、コールグラフ上のトップダウン伝播によって、各手続きの入口と出口での変数値を求めていく。トップダウン伝播の過程では、コールサイトによって値の異なる変数がある場合、手続きクローニングを行ない、その値毎に異なる手続き呼び出しを呼び出すようプログラムを変換する。

2.2 手続き間配列データフロー解析

手続き間配列データフロー解析 [2] では、手続き間変数値シンボリック解析、配列についての連続矩形領域および線形不等式系による手続き間配列参照領域解析を行なう。本稿では、線形不等式系による手続き間配列参照領域解析について述べる。

多くのプログラムの並列化では、配列参照領域を連続矩形領域で近似すれば十分であった。しかし、SPECfp95 の turb3d のように、3 次元配列の特定の面を、呼出先手続きで 1 次元化した配列としてアクセスするプログラムでは、非連続な配列参照が発生する。turb3d の主要ループを並列化するには、この参照領域を正確に表現しなければならない。

線形不等式系による解析では、プログラム内に出現する変数についての線形な等式と不等式で配列参照領域を正確に表現し、それをコールグラフ上でボトムアップに伝播する。

2.3 手続き間並列化

手続き間並列化 [4] は、手続き間 DO ALL 並列化、手続き間リダクション並列化、手続き間プライベート化を行なう。手続き間プライベート化では、必要に応じて手続き間初期値・終値保証も行なう。

```
                    <変換前>
 program main                    subroutine sub(i)
 common /com/a(N)                common /com/a(N)
 do j = 1, M                     if(...) then
   do i = 1, N                     a(i) = i
     call sub(i)                 endif
   enddo                         return
 enddo                           end
 a(1:N) の使用点
 end

                    <変換後>
 program main                    subroutine sub(i)
 common /com/a(N)                common /com/a(N)
common /com/Pa(N,npe)            common /com/Pa(N,npe)
 integer mask                    integer mask
common /m/mask(N,npe)            common /m/mask(N,npe)
 mask(1:N,peN) = 0               if(...) then
 do j = 1, M/npe ! 並列ループ       a(i) = i
   do i = 1, N                     mask(i,npe) = 1
     call sub(i)                 endif
   enddo                         return
 enddo                           end
 バリア同期
 if(peN .eq. 0) then ! 条件付き終値保証
   do i = 1, N
     do j = npe, 1, -1
       if(mask(i,j) .eq. 1) then
         a(i) = Pa(i,j)
         goto 999
       endif
     enddo
   enddo
999 endif
 a(1:N) の使用点
 end
```

図 3: 手続き間条件付き終値保証

本稿では、手続き間初期値・終値保証について述べる。

プライベート化とは、並列ループ中の一時変数に、プロセッサ毎に別の領域を割り当てることである。

プライベート変数にループ中で使用だけされる部分がある場合、ループ後の最初の参照が使用の場合は、それぞれ、ループ直前でのプライベート変数への初期値設定 (初期値保証)、ループ直後でのプライベート変数から元の変数への値の書き戻し (終値保証) が必要である。

初期値保証は、露出使用領域を求め、露出使用が存在する変数に対して初期値保証コードを生成する。

終値保証は、定義領域、ライブ領域の積集合が空でない変数を対象とする。このうち、確定定義である部分はループの各繰り返しで必ず値が定義されるので、ループ最終回の値を終値とする確定終値保証コードを生成する。確定定義でない部分は最後に定義されるループ繰り返しが不明なので、これを実行時に検査する条件付き終値保証コードを生成する。条件付き終値保証は、対象変数にプロセッサ番号を表す次元を追加したマスク配列を、各ループ繰り返しでの定義の有無を記録するために用意する。マスク配列は対象変数の定義点直後で値を設定される。

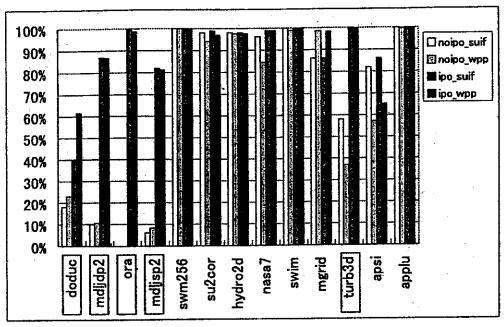

図 4: SPECfp92/SPECfp95 での並列化率

ループ終了後、マスク配列の値を設定した最大のループ繰り返しでのプライベート変数の値を終値として元の変数に書き戻す。

図 3は、配列 a に手続き間条件付き終値保証が必要なループの変換例である。npe はプロセッサ数、peN は自プロセッサ番号、Pa(N,npe) はプライベート化された配列 a、mask は a に対する条件付き終値保証用マスク配列である。

3 性能評価 (1) – 並列化率 –

本節では、WPP の並列化能力の指標とする並列化率の測定方法と評価結果について述べる。

並列化率は次の式で定義する。

$$\text{並列化率} = \frac{\text{並列実行部分の実行時間}}{\text{全実行時間}} \quad \text{(式 1)}$$

3.1 測定方法

longjmp と setjmp を利用して逐次プロセッサ上でマルチスレッドを順次実行するツールと日立 SR2201 のハードウェアカウンタを利用して、各スレッドの実行サイクル数を計測して並列化率を求めた。

3.2 評価結果

図 4は、WPP と SUIF コンパイラ [5] の並列化率である。noipo_* は手続き間並列化なしの、ipo_* は手続き間並列化ありの場合を表す。対象プログラムは、SPECfp92 から 8 本、SPECfp95 から 5 本を選んだ。除外したプログラムには手続き間並列化の効果はない。プログラム名が四角で囲まれたもの

表 1: 適用された手続き間並列化機能

プログラム	D+P	Red	Ivl	Fvl	Cvl
doduc	◎	◎	◎	◎	◎
mdljdp2	◎	◎	—	—	—
ora	◎	◎	—	◎	◎
mdljsp2	◎	◎	—	—	—
swm256	○	—	—	—	—
su2cor	○	○	○	—	○
hydro2d	○	○	○	—	○
nasa7	○	○	—	—	—
swim	○	—	—	—	—
turb3d	◎	○	—	—	—
apsi	◎	○	—	—	○
applu	○	○	○	○	—

D+P： DO ALL 並列化+プライベート化
Red：リダクション並列化、 Ivl：初期値保証
Fvl：終値保証、 Cvl：条件付き終値保証
◎：手続き間並列化で適用
○：手続き内並列化でのみ適用
—：適用例なし

は、手続き間並列化の効果が大きいプログラムである。並列化率については、 WPP と SUIF はほとんどのプログラムでほぼ同じ値となっている。

表 1は、適用された並列化機能である。プライベート化は DO ALL 並列化とまとめて扱った。手続き間 DO ALL 並列化、手続き間プライベート化、手続き間リダクション並列化は有効なプログラムが多い。手続き間初期値・終値保証は、 doduc, ora で並列化率に寄与している。

4 性能評価 (2) – 並列実行性能 –

本節では、 WPP で並列化したプログラムの並列実行方法、測定方法、評価結果について述べる。

4.1 並列実行方法

プロセッサ 4 台 (PE0 ～ 3) の場合を例にとって説明する。 WPP で並列化したプログラムは、各プロセッサに固定的に割り付けられたスレッドで並列実行される。スレッド生成・消去は、並列ループ毎に行なうのではなく、プログラム実行開始直後・終了直前にのみ PE0 が行ない、スレッド生成・消去のオーバーヘッドを小さくする。並列実行部の起動・終結は、バリア同期で実現する。

PE1 ～ 3 に割り付けられたスレッドは、実行開始直後にバリア同期を実行して同期成立待ちに入り、PE0 のスレッドのみ逐次実行部を実行する。

表 2: SR4300 の構成

CPU	PowerPC 604[2] (112MHz) × 4
キャッシュ	L1 32KB(I 16KB/D 16KB)
	L2 なし
メモリ	1GB

PE0 は並列実行部直前でバリア同期を実行して PE1 ～ 3 のスレッドの同期待ちを成立させ、全 PE が並列実行部を実行する。全 PE は並列実行部終了直前にバリア同期を実行し、その後、 PE1 ～ 3 は再びバリア同期を実行して同期成立待ちに入り、PE0 のみ直後の逐次実行部を実行する。

スレッドの生成・消去、バリア同期には POSIX[1] スレッドライブラリを用いた。

4.2 測定方法

並列実行性能評価は、共有メモリ型並列機 日立 SR4300 上で行なった。表 2は SR4300 の構成である。同じプログラムを 5 回測定し、平均値を実行時間とした。並列実行性能は、逐次実行時間に対する並列実行時間の比で評価し、この比をスピードアップと呼ぶ。適用した最適化は、手続き間並列化を除いて逐次と並列で同一である。

4.3 実行時オーバーヘッド

最初に基本機能であるバリア同期の実行時間を測定した。バリア同期は約 50μ sec かかる。また、並列実行部の起動終結は、バリア同期 2 つで構成されるため、 100μ sec 程度である。

手続き間リダクション並列化や手続き間初期値・終値保証は、総和演算、初期値設定、終値書き戻しといった逐次実行時にはない処理が必要であり、実行時オーバーヘッドとなっている。以下では、図 5のテストプログラムでこの実行時オーバーヘッドを評価する。

図 5の (a) ～ (d) は、 i ループを手続き間並列化する場合、配列 a に関してそれぞれ手続き間リダクション、手続き間初期値保証、手続き間確定終値保証、手続き間条件付き終値保証が必要である。

(a) ～ (d) それぞれについて、手続き間並列化したオブジェクトコード (コード 1) と、総和演算や初期値・終値保証コードがない以外は前者と同じオブ

[1]POSIX は、the Institute of Electrical and Electronics Engineers, Inc. (IEEE) で制定された標準仕様です。

[2]PowerPC 604 は、米国における米国 International Business Machines Corp. の商標です。

ループ長：N=10000, M=1000

(a) 手続き間リダクション並列化が必要なループ

```
program reduction          subroutine sub1(i,a)
real a(N)                  real a(N)
do j = 1, M                a(i) = a(i) + i
  do i = 1, N              return
    call sub1(i,a)         end
  enddo
enddo
end
```

(b) 手続き間初期値保証が必要なループ

```
program ival               subroutine sub2(i,a)
real a(2*N),b(M)           real a(2*N)
do j = 1, M                a(i) = i
  do i = 1, N              return
    b(j) = a(i+N)          end
    call sub2(i,a)
  enddo
enddo
end
```

(c) 手続き間確定終値保証が必要なループ

```
program fval               subroutine sub3(i,a)
real a(N),b(M)             real a(N)
do j = 1, M                a(i) = i
  do i = 1, N              return
    call sub3(i,a)         end
  enddo
enddo
a(1:N) の使用点
end
```

(d) 手続き間条件付き終値保証が必要なループ

```
program cfval              subroutine sub4(i,a)
real a(N),b(M)             real a(N)
do j = 1, M                if(...) a(i) = i
  do i = 1, N              return
    call sub4(i,a)         end
  enddo
enddo
a(1:N) の使用点
end
```

図 5: テストプログラム

ジェクトコード (コード 2) を作成して実行時間を測定し、手続き間並列化にかかるオーバーヘッドの割合を次式で評価した。

$$\text{オーバーヘッドの割合} = \frac{\text{コード 1 の実行時間} - \text{コード 2 の実行時間}}{\text{コード 1 の実行時間}} \quad \text{(式 2)}$$

表 3はその結果である。ループ長は M=1000, N=10000 である。

(a) ～ (c) はオーバーヘッドは最大でも 2.3% と十分小さい。

(d) は 15% 程度のオーバーヘッドがある。これは並列ループ中へのマスク変数代入文の挿入のためである。ここから、粒度が小さい場合はループを逐次実行した方がよいことがわかる。従って、手続き間条件付き終値保証は、粒度とオーバーヘッドを比較し、効果のある場合のみ適用する必要がある。

表 3: オーバーヘッドの割合

	プロセッサ数	2	4
(a)	手続き間リダクション	1.1%	1.1%
(b)	手続き間初期値保証	1.7%	1.1%
(c)	手続き間確定終値保証	1.2%	2.3%
(d)	手続き間条件付き終値保証	16.7%	14.4%

```
      SUBROUTINE FRCUSE
            :
      DO 20 I = 1, 500 - 1   ! ループA／手続き間並列化ループ
        JBEG = NBINDX(I)
        JEND = NBINDX(I+1) - 1
        IF (JBEG.GT.JEND) GO TO 20
        CALL JLOOPU (I,JBEG,JEND)
20    CONTINUE

      SUBROUTINE JLOOPU (I,JBEG,JEND)
      DO 20 JX = JBEG,JEND   ! ループB
            :
20    CONTINUE
      RETURN
      END
```

図 6: mdljdp2 の手続き間並列化ループ

4.4 SPEC ベンチマーク

手続き間リダクション並列化の実例として、SPEC92fp の mdljdp2, mdljsp2 を挙げる。これらは、精度 (mdljdp2：倍精度、 mdljsp2：単精度) を除いて基本的に同じ分子動力学プログラムである。以下では mdljdp2 の場合のみ説明する。

図 6は mdljdp2 の主要な手続き間並列化ループの概要である。以後区別の為、手続き FRCUSE のループ 20 をループ A、手続き JLOOPU のループ 20 をループ B と呼ぶ。ループ A は手続き間並列化可能であり、ループ中の 6 つのコモン変数に総和型の手続き間リダクション並列化が適用される。

表 4は、 mdljdp2 を並列化した場合のスピードアップである。

手続き内並列化を行なった場合は逐次より数 % 遅い。この原因として、手続き内並列化だけではループ A の 1/1000 以下の小粒度ループしか並列化できなかったことが挙げられる。

手続き間並列化のみを行なった場合は、プロセッサ 4 台で逐次の 1.51 倍 (mdljdp2)、 1.55 倍 (mdljsp2) にスピードアップした (表 4)。

図 6のループ B の範囲 JBEG, JEND は実行時に決まる。このプロファイルをとると、ループ A のループ制御変数 I の値が小さい場合はループ B のループ範囲が数百と大きく、大きい場合は 1 ～ 10 と小さいことがわかった。このため、ループ A のループ範囲を BLOCK 分割して手続き間並列化すると、

表 4: スピードアップ

名称	mdljdp2		mdljsp2		doduc	
PE 数	2	4	2	4	2	4
PAR	0.99†	0.96†	0.97†	0.90†	1.00	1.00
IPS	1.15	1.51	1.34	1.55	1.00	1.00
IPC	1.30	1.75	1.46	1.85	—	—
SUIF‡	—	1.8	—	1.8	—	1.0

†ループ B は並列化しない、‡文献 [6]
PAR：手続き内並列化、IPS：手続き間並列化
IPC：手続き間並列化＋ CYCLIC 分割

ループ B のループ繰り返し回数がプロセッサ間で不均一になり、負荷バランスが悪い。これを避けるため、ソースプログラムのループ A の直前に CYCLIC 分割指示文を人手で挿入して手続き間並列化を行ない、負荷バランスを改善した。また、CYCLIC 分割を行なった場合、BLOCK 分割ならばループ繰り返しにまたがって連続アクセスできる配列要素が連続アクセスできなくなり、キャッシュミスが増える可能性がある。そこで、CYCLIC 分割のブロックサイズをコンパイラ指示文で変えてテストを行ない、ループ A での最適なブロックサイズを得た。この結果、プロセッサ 4 台で逐次の 1.75 倍 (mdljdp2、ブロックサイズ 8)、1.85 倍 (mdljsp2、ブロックサイズ 2) のスピードアップを得た (表 4)。

次に、手続き間条件付終値保証の実例として、SPECfp92 の doduc を挙げる。スカラ変数 X1 に手続き間条件付終値保証が必要な手続き DESECO のループ 2500 の粒度を測定した。(式 2) でループ 2500 の手続き間条件付終値保証オーバーヘッドを評価すると、プロセッサ 2 台で 2.7 %、4 台で 2.6 % となり、doduc ではオーバーヘッドが小さいことが分かった。これはループ本体が大きいからである。

doduc で並列化可能なループは、いずれも起動終結オーバーヘッドに比べて粒度が小さい。そのため、粒度を考慮して並列化するとループは全て並列化されず、スピードアップは 1.00 倍となった (表 4)。

いずれのプログラムも、SUIF コンパイラとほぼ同等のスピードアップである [6]。

5 今後の課題

今回の結果から、条件付終値保証の適用に関してループ粒度を考慮する機能の追加や、最適ブロックサイズ判定も含めた CYCLIC 分散の自動適用の実現が今後の機能強化課題として挙げられる。

また、今回は WPP の並列化能力に重点を置いて性能評価を行なったが、今後はキャッシュ関係の性能評価も行ないたい。

6 まとめ

本研究では、手続き呼び出しを含むループを並列化可能な、共有メモリ型並列機向け手続き間並列化コンパイラ WPP を開発し、並列化能力、並列実行性能を評価した。

その結果、並列化率では、Stanford 大学の SUIF コンパイラとほぼ同程度の能力をもつことがわかった。実行時オーバーヘッドは、手続き間条件付き終値保証は、オーバーヘッドの大きい場合があるが、その他は十分小さいことが分かった。

SPEC92fp の mdljdp2、mdljsp2 に関しては、CYCLIC 分散指示文を人手で追加して手続き間並列化ループの負荷分散を図ることにより、プロセッサ 4 台で逐次実行の 1.75 倍、1.85 倍の性能と、SUIF と同程度の並列実行性能を得られた。

doduc では、ループ本体が大きいため手続き間条件付終値保証のオーバーヘッドが小さいことが分かった。また、起動終結オーバーヘッドに比べてループ粒度が小さいため、並列化されたループはなかった。

実用プログラムの自動並列化にはまだ多くの課題があり、今後も各種の性能評価を行ないつつ機能強化を進める予定である。

参考文献

[1] 菊池 他, 並列化システム「Parassist」の試作 – 機能と構成 –, 情報処理学会第 44 回全国大会, (1992)

[2] 飯塚 他, 手続き間並列化コンパイラ WPP の試作 – 現状と今後の課題 –, 情報処理学会第 56 回全国大会, (1998)

[3] 佐藤 茂 他, 手続き間並列化コンパイラ WPP の試作 – 定数伝播とクローニングの評価 –, 情報処理学会第 56 回全国大会, (1998)

[4] 佐藤 真琴 他, 手続き間並列化コンパイラ WPP の試作 – 変数プライベート化技術 –, 情報処理学会第 56 回全国大会, (1998)

[5] SUIF Compiler, http://suif.stanford.edu/

[6] Amarasinghe, S. P. et al. Multiprocessors from a Software Perspective, *IEEE Mirco*, 52-61, (1996-6)