ハイパフォーマンス
コンピューティング 81－9
（2000．6．2）

キャッシュサイズループ並列化とその評価

青木 雄一郎　　佐藤 真琴
㈱ 日立製作所システム開発研究所

高性能を容易に得る手段として共有メモリ型並列機（SMP）が注目されている。SMP向け自動並列化コンパイラにおいて高性能プログラムを得るには、ループ並列化に加えてキャッシュ向け最適化の適用も必要である。本研究では、連続する複数の並列ループに渡ってキャッシュ上のデータ再利用を促進するキャッシュサイズループ並列化の実現方法を検討し、性能評価を行った。人手で並列化したプログラムによる実機評価では、SGI Origin2000上でSPECfp95／tomcatvのカーネルループ中の2ループに対し、1次データキャッシュに対する本並列化を適用した場合、この2ループが13.7～19.5%(1～16PE)の高速化を、また2次キャッシュに対して適用した場合、4.9～17.0%(1～16PE)の高速化を達成した。

The Cache-Size Loop Parallelization and Its Evaluation

Yuichiro Aoki　and　Makoto Satoh
Systems Development Laboratory, Hitachi, Ltd.

Shared-memory multiprocessors(SMPs) are getting popular because users can easily obtain high performance on SMPs. To generate more efficient code for SMPs, automatic parallelizing compilers should apply some cache optimizations to parallel loops. The cache-size loop parallelization transforms a set of successive loops into a new loop that includes the parallelized successive loops each of which reuses data on cache. In this study we examine its implementation and evaluate its performance. We applied its transformation to two loops in SPECfp95/tomcatv by hand and evaluate their performance on SGI Origin2000. The resulting codes targetting level 1 and level 2 cache run faster than the conventional one by 13.7-19.5% (1-16PE) and 4.9-17.0% (1-16PE).

1．はじめに

本研究の目的は、連続する複数の並列ループに渡ってキャッシュ上のデータ再利用を促進するキャッシュサイズループ並列化の実現方法を検討し、その性能評価を行うことである。

近年、高い計算能力を容易に得る手段として、共有メモリ型並列機(SMPs:Shared-memory multiprocessors)が注目されている。SMPを有効利用するにはSMP向け並列プログラムが必要だが、既存の逐次プログラムを人手で並列化するには依存解析や並列化変換に大きな手間がかかるため、自動並列化コンパイラへの期待が大きい。

この期待に応えるため、我々は手続き間自動並列化コンパイラWPP(Whole Program Parallelizer)を開発している。これまでに、ループ並列化では、手続き間doall並列化、手続き間リダクション並列化、手続き間変数プライベート化を[1)2)]、ループ並列性解析の強化では、手続きクローニングを伴った手続き間定数伝播[3)]、連立線形不等式表現による手続き間配列参照領域解析[4)]を実現している。

またWPPでは、各プロセッサの参照間のデータ依存をなくすことによるバリア同期削減を目的に、手続き間静的アフィニティスケジューリングを開発している[5)]。手続き間静的アフィニティスケジューリングは、先行並列ループであるプロセッサが参照したデータを、後続並列ループでも同じプロセッサが参照するようにループ範囲の分割を行なって、並列ループ間のバリア同期を削除する機能である。

この機能により、各プロセッサが参照するデータ量がキャッシュサイズ以下だった場合には、(1)バリア同期の削除、(2)キャッシュ間競合の解消、(3)連続する複数の並列ループに渡ったキャッシュ上のデータの再利用促進、が起こるため、大幅な性能向上が見込める。一方で、各プロセッサが参照するデータがキャッシュサイズに納まらない場合には、連続する複数の並列ループに渡ったキャッシュ上のデータの再利用は困難なため、こうした性能向上は期待できなかった。

連続する複数の並列ループに渡ってキャッシュ上のデータの再利用を促進する従来のキャッシュ向け最適化技術としては、ループ融合がある[6)]。しかし、ループ融合はループ上下限値が同じ並列ループ同士でなければならないため、

適用範囲に制限があった。

この問題を解決するため、我々はキャッシュサイズループ並列化の実現方法を検討した。本並列化は、あるプロセッサが先行並列ループで参照したデータを、後続並列ループで同じプロセッサが参照する場合に、後続並列ループの参照時でもそのデータがキャッシュ上に残っているように、並列化された両ループのイタレーションを分割し、先行並列ループの分割されたイタレーションが実行された直後に、同じデータを参照する後続並列ループの分割されたイタレーションが実行されるように、両ループの外側に制御ループを設ける機能である。これにより、上下限値が異なる連続する並列ループに対し、複数の並列ループに渡ってキャッシュ上のデータ再利用を促進するようなループ並列化が実現できる。

以下、第2節ではコンパイラの構成を、第3節では処理方法を、第4節では評価結果を、第5節では関連研究を、第6節では結論を述べる。

2．コンパイラの構成

図1は手続き間自動並列化コンパイラWPPの構成である。

以下では並列化部各処理を説明する。並列化部は、手続き間ループ並列化解析部、キャッシュサイズループ並列化解析部、ループ並列化変換部から構成される。

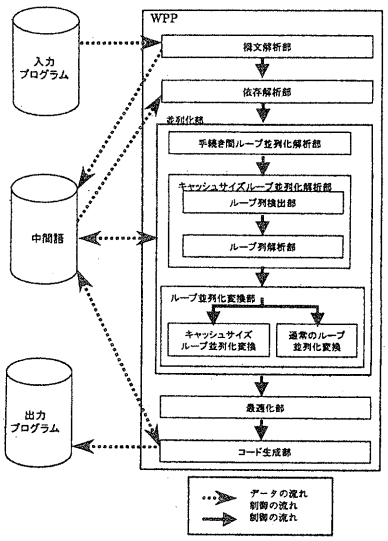

図1 手続き間自動並列化コンパイラWPPの構成

手続き間ループ並列化解析部は、中間語を入力し、どのループが並列化可能か解析する。

キャッシュサイズループ並列化解析部は、ループ列検出部、ループ列解析部から構成される。ループ列検出部は、本並列化適用候補の並列ループを検出する。ループ列解析部は、検出された候補並列ループについて、本並列化が適用可能かを詳しく解析して適用並列ループを決定する。

ループ並列化変換部は、本並列化適用ループならばキャッシュサイズループ並列化変換を行い、そうでなければ通常のループ並列化変換を行い、変換後の中間語を出力する。

3．処理方法

本節では、キャッシュサイズループ並列化の処理方法を説明する。まず最初に本並列化の適用条件を述べる。第1節で述べたように、本並列化では、先行並列ループの分割されたイタレーションが実行された直後に、同じデータを参照する後続並列ループの分割されたイタレーションが実行されるように、両ループの外側に制御ループが設けられる。そのため、本並列化が適用可能なのは、以下の条件(A)～(D)を満たす連続する複数の並列ループである。これらの条件については後で詳述する。

条件(A) 手続き間静的アフィニティスケジューリングが適用可能である。

条件(B) 各ループの並列化によりループ間にバリア同期が生じない。

条件(C) 同じ共有配列を参照する並列ループが2つ以上ある。

条件(D) 配列参照量がキャッシュサイズより大きい並列ループが存在する。

条件(A)は、並列ループ間にバリア同期が存在しない条件である。しかし、リダクション並列化の総和計算やプライベート変数の終値保証のような並列化変換に伴って現れる追加処理によるバリア同期は未考慮である。

条件(B)は、条件(A)で未考慮の並列化変換に伴ってバリア同期が発生しないための条件である。バリア同期が発生すると、先行並列ループの全イタレーション終了後でないと後続並列ループのイタレーションが実行開始できない。そのため、先行並列ループの分割されたイタレーションが実行された直後に、同じデータを参照する後続並列ループの分割されたイタレーションを実行できず、本並列化が適用できない。

条件(C)、(D)は、複数の並列ループに渡ってキャッシュ上のデータ再利用が促進されるための必要条件である。

以下の各項では、これらの条件の解析方法および本並列化変換方法について述べる。

3．1 ループ列検出部

本処理は、プログラム中の並列ループが条件(A)を満たすか解析し、手続き間アフィニティスケジューリングが適用可能な連続する複数の並列ループを検出する。

手続き間静的アフィニティスケジューリングは、条件(A-1)～(A-7)を満たすループに適用されるので[5]、本処理ではこれらの条件を満たす並列ループを検出する。これらの条件は同類条件と呼び、これを満たす複数のループをループ列と呼ぶ。

条件(A-1) 先行ループと後続ループのストライドは同じ。

条件(A-2) 先行ループと後続ループの上下限値の差をそれぞれCu、Cl とすると、Cu、Cl は定数。

条件(A-3) |Cu|、|Cl|＜一定値。

条件(A-4) 先行ループと後続ループが共通に参照する配列の分割次元（添字にループ制御変数が現れる次元）は等しい。

条件(A-5) 先行ループと後続ループが共通に参照する配列の同じ分割次元の分割種別（BLOCK 分割、CYCLIC 分割、...）は等しい。

条件(A-6) 先行ループと後続ループが共通に参照する配列の同じ分割次元に現れる添字は、全て差が条件(A-3)を満たす定数である。

条件(A-7) 先行ループと後続ループは連続するループであり、かつそれぞれが、最外側ループから見て密多重なループの１つである。

ただし、ループ並列化変換部の実現を容易にするために、条件(A-1)のストライドは 1 に、条件(A-5)の分割種別はBLOCK 分割に限定する。

3．2 ループ列解析部

ループ列解析部は、ループ列検出部で見つかったループ列に対し、ループ間解析処理、参照量解析処理を行い、本並列化適用ループ列を決定する。以下では各処理を説明する。

3．2．1 ループ間解析処理

本処理は、ループ列中の並列ループが条件(B)、(C)を満たすか解析する。条件(B)を具体化したのが条件(B-1)である。

条件(B-1) ループ列中の並列ループは、以下の(a)～(d)を全て満足する。ここで、変数:ループ制御変数を除くループ中で参照されるスカラ変数または配列変数、C:ループ列中の並列ループ数、とする。

(a) i 番目の並列ループで、初期値保証が必要なプライベート変数に対し、1～i-1 番目の並列ループで定義がない（2≦i≦C）。

(b) j 番目の並列ループで、終値保証が必要なプライベート変数に対し、j+1～C 番目の並列ループで参照がない（1≦j≦C-1）。

(c) k 番目の並列ループでリダクション並列化を行う変数に対し、1～k-1 番目の並列ループで参照がない（2≦k≦C）。

(d) m 番目の並列ループでリダクション並列化を行う変数に対し、m+1～C 番目の並列ループで参照がない（1≦m≦C-1）。

条件(B-1)を設けた理由は以下の通りである。条件(B-1)の(a)～(d)のいずれかを満たさない変数が現れた場合、先行並列ループと後続並列ループの間にバリア同期が必要となり、本並列化を適用できない。

図2の並列ループL1、L2は、条件(B-1)の(d)を満たさない例であり、先行並列ループL1のリダクション変数aが定義した値を後続並列ループL2で使用するため、L1の全イタレーションと、L1の直後で実行されるリダクション変数a総和計算が終了しないとL2を実行開始できず、本並列化が適用できない。

```
L1:     do i = 1, N
          a = a + b(i)   ! aはリダクション変数
        enddo
L2:     do j = 1, N
          c(j) = b(j) + a
        enddo
```

図2 適用不能ループ

3．2．2 参照量解析処理

本処理は、ループ列中の並列ループが条件(D)を満たすか解析し、満たす場合に本並列化適用ループ列と判定し、元の並列ループのイタレーションをいくつに分割するか計算し、その値をイタレーション分割数とする。

まず、ループ列中の各並列ループでのループ内配列参照量を、各配列について（全ループ長）×（配列要素サイズ）

の和をとることで計算する。ここで全ループ長とは、並列ループ及び並列ループがネストする全内側ループのループ長の積である。ループボディ中に条件分岐があった場合は、条件分岐はないものとして計算する。次に、ループ列中での最大のループ内配列参照量をプロセッサ数とキャッシュサイズで割った結果が1より大きい場合に、その結果を切り上げた値をイタレーション分割数とする。切り上げは、並列ループ内での配列参照量をキャッシュサイズより小さくするために行う。結果が1以下の場合は本並列化の効果がないので、本並列化を適用しない。

3．3 ループ並列化変換部

本処理は、本並列化適用並列ループにはキャッシュサイズループ並列化変換を、そうでない並列ループには通常のループ並列化変換を行う。ここでは、キャッシュサイズループ並列化変換についてのみ説明する。

まず、本並列化が適用されるループ列に対し、ループ列の外側に、下限値が1、上限値がイタレーション分割数、増分値が1の制御ループを設ける。次に、ループ列中の並列ループのイタレーション上下限値を次の式で与える。次の式でblk、lb、ub、new_blkは、各々手続き間静的アフィニティスケジューリングのみを適用した場合の並列ループの並列化後のループ長、ループ下限値、ループ上限値、本並列化適用後のループ長を表し、kは制御ループのループ制御変数、Ndivはイタレーション分割数、Cl、Cuは条件(A-2)に現れる定数である。

$$\text{new\_blk} = \lceil \text{blk} / \text{Ndiv} \rceil$$

$$(\text{ループ下限}) = \text{lb} + \text{new\_blk} \times (k-1) - Q1$$

$$\text{ただし}\quad Q1 = \begin{cases} Cl & (k=1) \\ 0 & (k \neq 1) \end{cases}$$

$$(\text{ループ上限}) = \min(\text{ub}, ((\text{ループ下限}) + \text{new\_blk} - 1)) + Q2$$

$$\text{ただし}\quad Q2 = \begin{cases} Cu & (k=\text{Ndiv}) \\ 0 & (k \neq \text{Ndiv}) \end{cases}$$

3．4 変換例

図3は、キャッシュサイズループ並列化に対する入力プログラム例であり、WPPの処理過程における本並列化解析部への入力となる中間語をOpenMPプログラムで表現したものである。OpenMP指示文より、ループL1、L2は並列ループ、ループ間のバリア同期は不要であることがわかる。

図3のプログラムを、プロセッサ8台、1次データキャッシュサイズ128KB、2次データキャッシュなしのSMPに対し従来法で並列化した場合を考える。各プロセッサはL1、L2のイタレーションを1024／8PE＝128ずつ担当するので、L1は配列aをプロセッサ当り1MB参照することになる。これはキャッシュサイズより大きいので、L2で配列aを参照する時点では配列aのデータはキャッシュから追い出されており、L1、L2に渡ってはキャッシュ上のデータを再利用できない。

図3のプログラムに本並列化のアルゴリズムを適用した場合は、次のようになる。

まず、図3のプログラムに対しループ列検出部が実行され、L1、L2が条件(A-1)～(A-7)の同類条件を満たすループ列として検出される。

次にループ列解析部の各処理が実行される。まず最初にループ間解析処理が実行され、L1、L2が条件(B-1)、(C)を満たし、配列aが両ループで参照されることが解析される。続いて参照量解析処理が実行され、L1、L2のループ内配列参照量がそれぞれ8MB、16MBであることから、(ループ列中の最大ループ内配列参照量16MB)／8PE／(キャッシュサイズ1MB)＝16より、イタレーション分割数が16と計算される。その結果、並列ループのイタレーションは、1024／8PE／16＝8ずつ実行される。

最後にループ並列化変換部が実行される。図3のプログラムに本並列化を適用した変換結果をOpenMPプログラムで表現したのが図4である。まずL1、L2の外側に、ループ下限値が1、ループ上限値がイタレーション分割数である16、ループ増分値が1の制御ループL3を作成する。次に、L1のイタレーション範囲を変更する。まず、元の並列ループ下限値文S1、上限値文S2を用いて、本並列化の下限値文S1'、上限値文S2'を作成し、それらをL1の新たなイタレーション上下限値として与える(ループL1')。L2も同様にして、元の上下限値文S3、S4から本並列化の上下限値文S3'、S4'を作成し、L2の新たなイタレーション上下限値として与える(ループL2')。

変換後のプログラムでは、制御ループL3がループL1'、L2'のイタレーションを8ずつ16回実行させる。そのため、L1'は配列aをプロセッサ当り64KBとキャッシュサイズ以下で参照する。そのため、L2'ではL1'が参照した配列aのデータをキャッシュ上で参照することができる。

```
      program explin
      integer i, j, N
      parameter (N=1024)
      real*8 a(N+1,N), b(N+1,N)
!$omp parallel
!$omp do schedule(static)
L1:     do j = 1, N
          do i = 1, N
            a(i,j) = DBLE(i) + DBLE(j)
          enddo
        enddo
!$omp end do nowait
!$omp do schedule(static)
L2:     do j = 1, N
          do i = 1, N
            b(i,j) = a(i,j)
          enddo
        enddo
!$omp end do nowait
!$omp end parallel
        write(*,*) b
        stop
        end
```

図3 入力プログラム例

4．評価結果

本節ではSGI Origin2000上での性能評価について述べる。

4．1 評価環境

表1は分散共有メモリ型並列機SGI Origin2000の構成である。Origin2000は、2PEが704MBのメモリを物理的に共有するノードを8個持っている。各ノードのメモリは論理的に共有されており、704MB×8ノード=5.5GBの共有メモリとしてどのプロセッサからでもアクセスできる。キャッシュは、1次、2次ともプロセッサ毎に持っている。

表1 SGI Origin2000の構成

プロセッサ	MIPS R10000 (195MHz)	
PE数	16	
キャッシュ	1次(データ)	32KB
	1次(命令)	32KB
	2次	4MB
メモリ	5.5 GB	
OS	IRIX 6.5.4	

4．2 評価方法

評価プログラムはSPECfp95/tomcatvを用いた。tomcatvのカーネルループはループ飛び出しを持つループであり、7つのループを内側に含む。そのうちの最初の2ループに本並列化が適用される。

そこで、本並列化を適用した場合としない場合の2通りについて、両ループのみの実行時間を処理系が提供するTIMEF手続きで測定した。更に、本並列化を適用した場合

```
      program explout
      integer i, j, lb1, ub1, lb2, ub2, lblm
      integer ublm, lb2m, ub2m, peN, N
      parameter (N=1024)
      real*8 a(N+1,N), b(N+1,N)
!$omp parallel private(peN,k,lb1,ub1,
!$omp+ lblm,ublm,lb2,ub2,lb2m,ub2m,i,j)
      peN = omp_get_thread_num()
L3:   do k = 1, 16
S1:     lb1 = 1 + 256 * (peN - 1)
S2:     ub1 = lb1 + 255
S1':    lblm = lb1 + 8 * (k - 1)
S2':    ublm = min(ub1, lblm + 7)
L1':    do j = lblm, ublm
          do i = 1, N
            a(i,j) = DBLE(i) + DBLE(j)
          enddo
        enddo
S3:     lb2 = 1 + 256 * (peN - 1)
S4:     ub2 = lb2 + 255
S3':    lb2m = lb2 + 8 * (k - 1)
S4':    ub2m = min(ub2, lb2m + 7)
L2':    do j = lb2m, ub2m
          do i = 1, N
            b(i,j) = a(i,j)
          enddo
        enddo
      enddo
!$omp end parallel
      write(*,*) b
      stop
      end
```

図4 変換後プログラム例

表2 測定結果

プロセッサ数	1	2	4	8	16
(a)	68.6s	40.6s	22.1s	11.8s	5.9s
(b)	59.2s	32.7s	18.4s	9.9s	4.8s
(c)	63.4s	38.1s	21.0s	10.6s	4.9s
((a)-(b))/(a)*100 (%)	13.7%	19.5%	16.6%	15.8%	18.0%
((a)-(c))/(a)*100 (%)	7.5%	6.2%	4.9%	9.8%	17.0%

(a) 従来法、(b) 1次データキャッシュに対する本並列化

(c) 2次キャッシュに対する本並列化

は、1次データキャッシュに対して適用した場合と、2次キャッシュに対して適用した場合を測定した。両ループで参照される配列は2個あり、ともに513×513個の要素を持つ倍精度2次元配列である。入力データはrefサイズを用いた。refサイズを入力とした場合、この2ループは、逐次実行時にプログラム全実行時間の38.6%を占める。

実測用プログラムはtomcatvに本並列化を人手で適用したOpenMPプログラムであり、MIPSpro Fortran90 Ver7.30でコンパイルした。オプションは-mp -Ofast=ip27 -OPT:IEEE_arithmetic=3である。

4．3 評価結果

表2は測定結果である。1次データキャッシュに対する本並列化は、1～16PEで従来法より13.7～19.5%の高速化を達成した。また2次キャッシュに対する本並列化は、1～16PEで4.9%～17.0%の高速化を達成した。結果より、1次データキャッシュをターゲットにした方が、2次キャッシュをターゲットにした場合よりも高速化されていることがわかる。

5．関連研究

稲石ら[7]は、タスク内で参照するデータ量がキャッシュサイズ以下であり、かつタスク間の参照データの重なりが大きくなるようにプログラムをタスクに分割し、直前に実行されたタスクと参照データの重なりが最大でありかつタスク間依存関係を満たすタスクが、次に実行されるようスケジュールする逐次実行コードを生成する最適化を発表している。一方本研究は、適用対象はループのみだが、並列実行に対応している。

吉田ら[8]は、ローカルメモリと集中共有メモリ(CSM)を持つ並列計算機に対し、CSM経由のデータ授受を削減するために、タスク間で参照するデータができるだけ重なるように、兄弟ループのイタレーションをタスクに分割し、重なりの大きなタスクが同じプロセッサで実行されるようスケジュールする並列実行コードを生成する最適化を発表している。しかし、ループ中で参照される配列の分割次元の添字が、並列ループのループ制御変数の1次式に限定される。本研究では、同類条件より非1次式でも取り扱える。

6．まとめ

本研究では、連続する複数の並列ループに渡ってキャッシュ上でのデータ再利用を促進するために、あるプロセッサが先行並列ループで参照したデータを、後続並列ループで同じプロセッサが参照する場合、そのデータがキャッシュ上に残っているように、並列化された両ループのイタレーションを分割するキャッシュサイズループ並列化の実現方法を検討し、性能評価を行った。その結果、SGI Origin2000上で、SPECfp95/tomcatvのカーネルループ中の2ループに対し、1次データキャッシュに対して本並列化を適用した場合、1～16PEで従来の並列化方法より13.7～19.5%の高速化を達成した。また、2次キャッシュに対して適用した場合は4.9～17.0%の高速化を達成した。

今後は、本並列化をWPP上に実現し、様々なプログラムに適用して性能評価を行う予定である。

謝 辞

本研究の一部は通産省Real World Computing Partnershipの中で行いました。本研究を始めるにあたり、早稲田大学理工学部電気電子情報工学科笠原研究室で行われている階層メモリ向け最適化の研究が大変参考になりました。研究内容を紹介して下さった笠原博徳教授に感謝致します。

参考文献

1) 佐藤(真) 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)

2) 青木 他, 手続き間自動並列化コンパイラ WPP の試作 － 手続き間並列化技術の実機評価 － , 98-ARC-130 (SWoPP長岡'98), pp. 43-48 (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) 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)

5) M. Satoh, SPMD region extension and automatic parallelization for SMP, LCPC'99: Paper and Poster Abstracts (1999)

6) M.Wolfe, High Performance Compilers for Parallel Computing, Addison-Wesley Publishing Company, pp.315-323 (1996)

7) 稲石 他, 最早実行可能条件解析を用いたキャッシュ利用の最適化, 98-ARC-130 (SWoPP長岡'98), pp. 31-36 (1998)

8) 吉田 他, 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, pp.2054-2063 (1999)

Origin2000は、Silicon Graphics, Inc.の商標です。