計算機アーキテクチャ　128－23
ハイパフォーマンス　70－23
コンピューティング
（１９９８．３．６）

Doacross ループの sandglass 型並列化手法の有効性について

高畠 志泰　本多 弘樹　弓場 敏嗣
電気通信大学大学院 情報システム学研究科

概要

本稿では、Doacross ループを対象とした新たに提案した sandglass 型並列化手法について検証する。sandglass 型並列化手法は、従来から使われるイタレーション単位でタスク分割する方法とソフトウェアパイプラインを使った並列化を利用した方法を融合させた特徴を持つ。提案する方法と上記の 2 つの方法、ループ分割を行い doall 型並列化と逐次ループを組み合わせた方法を実機の上で比較評価を行う。評価には、3 種類のベンチマークプログラムを用いる。その結果、sandglass 型並列化手法の効果が得られるのは、要素プロセッサ数が少ない状況で、ループ分割により、ループ運搬依存が存在するタスクより存在しないタスクループが大きい Doacross ループであることがわかった。

Performance Measurements on Sandglass-Type Parallelization of Doacross Loops

Motoyasu Takabatake　Hiroki Honda　Toshitsugu Yuba
Graduate School of Information Systems, The University of Electro-Communications

Abstract

In this paper, we evaluate a new sandglass type parallelization technique for doacross loops, which has the characteristics of iteration-based parallelizing and software pipelining. We compare four parallelization techniques; iteration-based, pipelining, combining doall and sequential, and sandglass-type. combining doall and sequential. As a result, sandglass-type parallelization technique is most effective among them, when a task existed loop-carried dependence is smaller than a task not existed in few processing elements.

1　はじめに

並列プログラムの効率的な実行を行うには、プログラムを並列実行可能な部分プログラム (タスク) に分割する方法と、分割された部分プログラムをどの要素プロセッサ (以下、PE と略す) に割り当てるかというスケジューリング方法が重要である。我々は、Doacross ループ [1] を対象とした、新たなループの分割方法と、分割した部分プログラム (タスク) のスケジューリング方法を提案した [5]。

従来から使われている方法として、ループ運搬依存 (loop-carried dependence) のある Doacross ループは、1 つのイタレーションを 1 つのタスクとし、各 PE に割り当てる方法がある [2][4][6]。この方法では、イタレーション間に存在するループ運搬依存により、各イタレーションごとに PE 間でデータの送受信が必要になる。そのため、通信オーバヘッドの大きい並列計算機では、並列化の効果が得られない。

Doacross ループをいくつかのステージに分割し、ソフトウェアパイプラインを構成することで並列化する方法もある [7]。ステージ間の通信が必要だが、通信のブロック化により、通

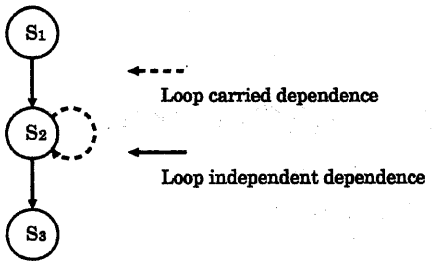

図 1: Doacross loop の依存グラフ

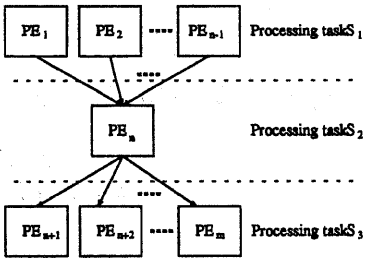

図 2: sandglass 型並列化手法

信回数を減らすことが可能である。しかし、パイプラインを用いた並列化は、一般的にパイプラインピッチを揃えることが困難である。

また、一般的な方法として、ループ分割を行ない、ループ運搬依存のあるループとないループに分ける方法が使われている。ループ運搬依存のあるループは逐次実行を行ない、ループ運搬依存のないループは、doall 型の並列化により並列実行させる。

本稿では、イタレーション単位で分割し並列化する方法と、ソフトウェアパイプラインを使った並列化の方法を組み合わせた特徴を持つ sandglass 型並列化手法の有効性を検証する。それぞれの方法を実機上で実行し、比較評価し、提案する方法の有効範囲について考察する。

2 sandglass 型並列化手法

2.1 依存グラフのモデル化

ループにおける依存グラフは、ループ運搬依存 (Loop-carried dependence) によって循環が生じる部分とそうでない部分の 2 つの種類に分けられる。各々の部分においてループ独立依存の枝にそって依存元タスクと依存先タスクを統合し一つのタスクとし、タスクサイズを大きくする。その結果、一般的な Doacross ループの依存グラフは、図 1のようにモデル化することができる。ループ運搬依存により循環する部分 S_2をループ運搬依存タスク、循環しない部分 S_1、S_3を非ループ運搬依存タスクと呼ぶことにする。

ループ運搬依存タスク S_2では、ループ運搬依存が複数存在しても構わない。

Doacross ループのループ運搬依存は、データの定義と参照の順番で前向き (LFD: Lexically-forward dependence) と後向き (LBD: Lexically-backward dependence) に分類される [1]。

2.2 並列化手法

並列計算機において、ループ運搬依存タスクと非ループ運搬依存タスクは、異なる PE 上で実行させる。ループ運搬依存タスクより、非ループ運搬依存タスクの方の粒度が大きい場合、そのタスクをイタレーション毎に異なる PE に割り当てる。つまり、ループ運搬依存タスクと非ループ運搬依存タスクの間ではパイプラインを形成し、非ループ運搬依存タスクは、異なるイタレーション間で並列で実行する [5]。この方法を sandglass 型並列化手法と呼ぶことにする (図 2)。

3 評価用 Doacross ループ

NAS Kernel Benchmark Program の一部分 (図 3(a))、spline 補間プログラムの一部分 (図 4(a))、Livermore Fortran Kernel の 23 番 (図 5(a)) を Doacross ループの評価用プログラムとして用いる。

Sandglass 型並列化手法は、ループ運搬依存タスクより非ループ運搬依存タスクが大きい場合に有効なので、その大きさの比により分類する。

NAS Kernel Benchmark プログラムの一部分 (以下、NAS と略す) は、1 重のループで、

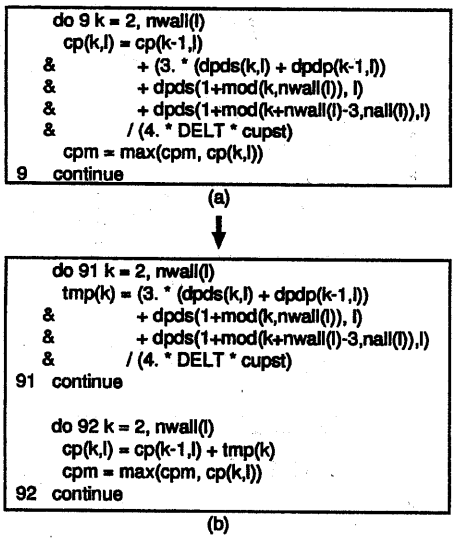

図 3: NAS kernel Benchmark の一部分のループ分割

ループ運搬依存は LBD である。また、その依存距離は 1 である。このプログラムは、図 3(b) のように分割することができる。図 3(b) において、前半 (S_1に相当) と後半 (S_2に相当) の計算量の比は、加減乗除をそれぞれ 1 とし、MAX の計算を 2 とすると、約 3 : 1 となる。

spline 補間のプログラムの一部分 (以下、spline と略す) は、1 重のループで、ループ運搬依存は LFD である。依存距離は、1 である。このプログラムは、図 4(b) のように分割することができる。図 4(b) において、前半 (S_1) と後半 (S_2) の計算量の比は、約 2 : 3 となる。

Livermore Fortran Kernel 23 番のプログラム (以下、Livermore と略す) は、2 重のループで、ループ運搬依存は LFD である。依存距離は、j、k ともに 1 である。このプログラムは、内側のループだけを分割し、図 5(b) に、または、二重ループ全てを分割し、図 5(c) に、することができる。図 5(b) において、前半 (S_1) と後半 (S_2) の計算量の比は、約 3 : 2 となる。図 5(c) において、前半 (S_1) と後半 (S_2) の計算量の比は、約 2 : 3 となる。

本稿で評価に用いるループは、表 1のようになる。$S_1 : S_2 : S_3$は、それぞれのタスクの大きさの比を表している。全てのループは、S_3 に相当する部分が存在しない。

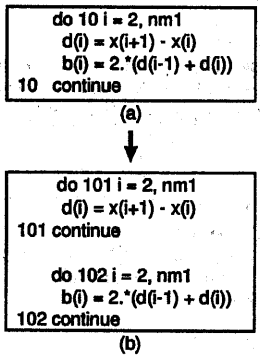

図 4: スプライン補間プログラムの一部分のループ分割

表 1: 評価用ループ

	依存方向	$S_1 : S_2 : S_3$	ネスト
NAS	LBD	3 : 1 : 0	1 重
spline	LFD	2 : 3 : 0	1 重
Livermore	LBD	(b) 3 : 2 : 0 (c) 2 : 3 : 0	2 重

4 評価

従来から使われている 1 つのイタレーションを 1 つのタスクとして、各 PE に割り当てる方法を Do-across と呼ぶ。ソフトウェアパイプラインを構成し並列実行する方法を Do-pipeline と呼ぶ。ループ分割を行ない、Doall と逐次を組み合わせたものを、Do-all & seq と呼ぶ。また、提案する sandglass 型並列化手法を Do-sandglass と呼ぶ。

それぞれの評価用ループのループ回数は、10,000 回とする。Livermore は、二重ループなので、外側を 10 回、内側を 1,000 回とした。

評価には、EM-X、SPARCserver1000、Cenju-3 の 3 機種を用いる。EM-X[3] は、電子技術総合研究所で開発された分散記憶型並列計算機で、細粒度の通信が可能である。SPARC-server1000 (以下、S1000 と略す) は、共有記

```
      do 23  j= 2,6
      do 23  k= 2,n
         qa= za(k,j+1)*zr(k,j) +za(k,j-1)*zb(k,j) +
     .       za(k+1,j)*zu(k,j) +za(k-1,j)*zv(k,j) +zz(k,j)
23    za(k,j)= za(k,j) +.175*(qa -za(k,j))
```

(a)

```
      do 232  j= 2,6
      do 231  k= 2,n
         tmp(k,j) = za(k,j+1)*zr(k,j)+ za(k,j-1)*zb(k,j)
     .       + za(k+1,j)*zu(k,j) +zz(k,j)
231   continue

      do 232 k = 2,n
         qa = tmp(k,j)
              + za(k-1,j)*zv(k,j)
         za(k,j)= za(k,j) +.175*(qa -za(k,j))
232   continue
```

(b)

```
      do 231  j= 2,6
      do 231  k= 2,n
         tmp(k,j) = za(k,j+1)*zr(k,j)
     .       + za(k+1,j)*zu(k,j) +zz(k,j)
231   continue

      do 232 j = 2,6
      do 232 k = 2,n
         qa = tmp(k,j)
              + za(k,j-1)*zb(k,j) + za(k-1,j)*zv(k,j)
         za(k,j)= za(k,j) +.175*(qa -za(k,j))
232   continue
```

(c)

図 5: Livermore Fortran Kernel 23 番のループ分割

憶型並列計算機で、メモリへの write と read でデータの送受信が可能である。Cenju-3 は、NEC で開発された分散記憶型並列計算機である。EM-X 上でそれぞれの評価用プログラムのループの特徴を調べ、sandglass 型並列化が有効な場合について、他の並列計算機上での有効性について調べる。

4.1 EM-X 上での評価

通信は、EM-X で実装されている直接遠隔記憶書き込み命令 (SYSWR) で送信データを書き込む。ハードウェア化された待ち合わせ機構を持つ I-structure を使って同期をとる。送信データが 1 ワードの時は、I-structure のみを用いて、データの送信と同期をとることにする。

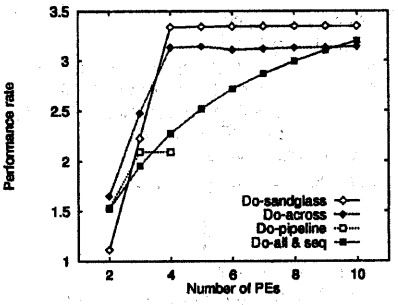

図 6: NAS on EM-X

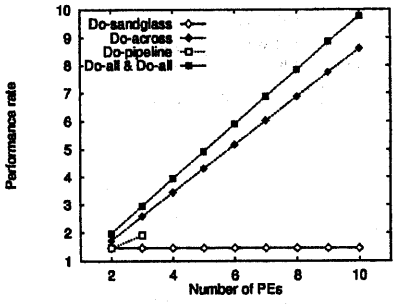

図 7: spline on EM-X

NAS を EM-X 上で実行した結果を図 6に示す。もっとも有効な方法は、Do-sandglass で、PE 数が 4 台のとき対逐次性能が約 3.4 倍であった。Do-sandglass は、S_1 : S_2が約 3 : 1 なので、PE を 5 台以上使用しても効果が得られない。EM-X は、細粒度通信が可能なため、Do-across が対逐次性能で約 3 倍である。Do-pipeline は、ステージ分割の方法が困難なため、4 台までしか測定できていない。Do-all & seq は、PE 数が 10 台で対逐次性能が約 3.2 倍であり、do-sandglass より性能が悪いが、台数を増やした 80 台のときは、対逐次性能が約 4.2 倍であった。PE 数が多ければ、Do-all & seq は、do-sandglass より性能がよくなる。しかし、PE 数が少なく 4 台の場合では、Do-sandglass が有効である。

spline を EM-X 上で実行した結果を図 7に示す。もっとも有効な方法は、Do-all & Do-all であった。spline は、ループ運搬依存が LFD で

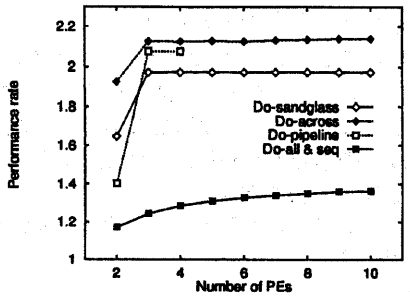

図 8: Livermore 23 on EM-X

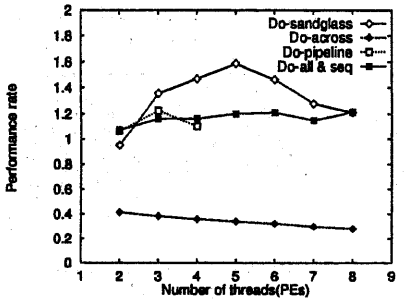

図 9: NAS on S1000

あるため、ループ分割を行なうと前後半ともにdoall 型並列化が可能である。そのため、Do-all & Do-all という型になる。また、Do-across がPE 数に比例して速度向上をしている。これは、ループ運搬依存がLFD であるため、各イタレーションの実行にズレが生じずに、同時に実行を開始できるためである。

spline で Do-sandglass が有効でない原因は2 つある。1 つは、ループ運搬依存がLFD であるために、他の方法が台数に比例して速度向上するためである。もう 1 つは、$S_1 : S_2$が約2 : 3 であるため、S_1をイタレーション毎に並列化しても、S_2がボトルネックになり、S_2の処理時間より短くならないためである。

Do-pipeline は、ステージ分割が困難なためPE 数で 3 台までしか測定出来なかった。

Livermore を EM-X 上で実行した結果を図 8に示す。Do-sandglass は、S_1よりS_2の方が小さい方がよいので、図 5(b) のようにループを分割した。Do-all & seq は、図 5(c) のようにループ分割した。

もっとも有効な方法は、Do-across であった。Do-across は、内側のループにをイタレーション毎に分割するため、内側のループが終った時点で、外側のループ運搬依存によって運ばれるデータが、既に自分自身のPE 内にある。そのため、外側のループ運搬依存についてデータの送受信の必要がない。Do-sandglass では、内側のループ運搬依存タスクを 1 つの PE 上で動かすため、その PE 上で定義されたデータを外側のループ運搬依存により、各PE に分配しなければならない。この差が実行結果の差となった。

Do-sandglass で、$S_1 : S_2$が約 3 : 2 であるため、PE 数が 3 台でもっともよくなり、それ以上では、効果が得られていない。

Do-all & seq で効果が小さい理由は、図 5(b)では、前半の並列実行可能な部分が後半の部分より小さいため、全体の実行時間のほとんどが後半の実行時間となるためである。

4.2 SPARCserver1000 上での評価

pthread ライブラリを用いて並列化し、同期にはスピンロックを用いた。

NAS を S1000 上で実行した結果を図 9に示す。スレッドの数は、使用するPE 数と同じである。S1000 上でもっとも有効な方法は、Do-sandglass であった。PE 数が 5 台のとき対逐次性能が約 1.6 倍であった。EM-X 上で実行したした場合と異なり、5 台を超えたところで性能が低下している。これは、スレッドの数が増えたため、メモリへのアクセス競合が増えたためと考えられる。

Do-all & seq は、スレッドの数を増やすに従い速度向上するが、do-all の処理が終ったときのスレッドの同期のオーバヘッドにより、並列化の効果が小さい。

4.3 Cenju-3 上での評価

Cenju-3 上で NAS を実行した結果を図 10に示す。通信には、MPI ライブラリを用いた。Cenju-3 は、通信のオーバヘッドが大きいためDo-all & seq 以外の方法では、対逐次性能は約

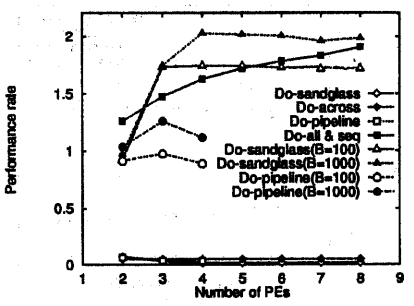

図 10: NAS on Cenju-3

0倍となり、並列化の効果が得られない。そのため、通信のブロック化を行なった。Do-pipeline と Do-sandglass は、通信のブロック化が可能である。Do-across は、1つのイタレーションを1つのタスクとしているため、通信のブロック化が不可能である。ブロックサイズ (B) を 100、1000 とした場合を図 10の中に示す。

通信のブロック化により並列化の効果が得られる。もっとも効果が得られる方法は、do-sandglass であり、ブロックサイズが 1000 のとき、PE 数が 4 台で対逐次性能は約 2 倍となった。

5 結論

NAS のようなループ運搬依存タスクより非ループ運搬依存タスクが大きい場合には、どの計算機上でも sandglass 型並列化手法が有効であることがわかった。ただし、PE 数を多くすれば、Do-all と逐次実行を組み合わせた方がよくなる。

spline のようなループ運搬依存タスクより非ループ運搬依存タスクが小さい場合には、提案する手法が有効でない。また、ループ運搬依存が LFD である場合、提案する手法以外のものが有効である。

Livermore において、do-all と逐次実行を組み合わせた方法が必ずしもよいとは限らないことがわかった。

また、Cenju-3 のような通信オーバヘッドの大きい並列計算機でも、通信をブロック化することで、sandglass 型並列化手法の効果を得ることができることがわかった。

今後の課題としては、2重ループような場合、ループリストラクチャリングを行なうことで依存距離を長くすることが可能である。依存距離が 2 以上の場合で sandglass 型並列化手法の有効性を考える必要がある。

また、S_3が存在するループを使って評価する必要がある。

参考文献

[1] Cytron, R.: Doacross: Beyond Vectorization for Multiprocessors, *Proc. of the Int. Conf. on Parallel Processing*, pp. 836–844 (1986).

[2] Cytron, R.: Limited Processor Scheduling of Doacross Loops, *Proc. of Int. Conf. on Parallel Processing*, pp. 226–234 (1987).

[3] Kodama, Y., Sakane, H., Sato, M., Yamana, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 22nd Annual Int. Symp. on Computer Architecture*, pp. 14–23 (1995).

[4] 福田晃: イタレーション実行時間が異なる1重 Doacross ループの並列処理における最適プロセッサ数, 信学論 D-I, Vol. J75-D-I, No. 7, pp. 450–458 (1992).

[5] 高畠志泰, 大澤範高, 弓場敏嗣: Doacross ループにおける粒度調整方法の検討, 情報処理学会研究報告,97-PRO-14, Vol. 97, No. 78, pp. 1–6 (1997).

[6] 山名早人, 安江俊明, 岡村洋一, 山口喜教: 分散共有メモリ型並列計算機における1重 Doacross 型ルー プの実行時間算出法, 信学論 D-I, Vol. J78-D-I, No. 2, pp. 170–178 (1995).

[7] 金子智一, 古関聰, 小松秀昭, 深澤良彰: ループステージング: 共有メモリ型並列計算機を対象とし たループ並列化技法とその評価, *JSPP'97*, pp. 197–204 (1997).