計算機アーキテクチャ 132−24
システムソフトウェアとオペレーティング・システム 80−24
ハイパフォーマンスコンピューティング 75−24
（1999. 3. 5）

# データの分割配置を考慮した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

{takabatake, honda, yuba}@yuba.is.uec.ac.jp

**概要**

分散記憶型並列計算機上で並列プログラムを実行する場合，配列データの分割配置の方法と，プログラムのタスクへの分割方法が重要となる．本稿では，これまで提案したDoacrossループのsandglass型並列化手法が，データの分割配置にどのような影響を受けるかを調べる．そして，並列計算機EM-Xを用いて，sandglass型並列化手法によって並列化されたループの実行時間がブロック分割やサイクリック分割のデータの分割配置に大きく影響されないことを述べる．

# Performance Measurements of Sandglass-Type Doacross Parallelization Considering Data Distribution

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

**Abstract**

When a parallel program is executed on a distributed memory type parallel computer, the performance significantly depends on its data distribution among processing element. In this paper, We investigate how data distribution affects the sandglass-type parallelization technique for executing doacross loops. As an experiment environment, we use the EM-X parallel computer with a fine-grain parallel architecture which was developed at Electrotechnical Laboratory. We consider that the performance of the sandglass type parallelization technique is not seriously affected by data distribution.

## 1　はじめに

分散記憶型並列計算機上で並列プログラムの効率的な実行を行うには，プログラムに合わせたデータの分割配置の方法が重要である．また，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も重要となる．我々は，Doacrossループ[1]を対象とした，sandglass型並列化手法という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]．本手法は，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．

論文[5]では，他の並列化手法との比較を行ない，提案した手法の有効性を示した．しかし，論文[5]では，データは，全てのPEにコピーされているという前提で評価を行なってい

たため，初期データが分割配置された場合に提案手法がどのように影響を受けるか明らかではなかった．そこで，本稿では，データの分割配置も考慮に入れ，sandglass 型並列化手法により並列化されたループの実行時間がどのような影響を受けるかを示す．

## 2 sandglass 型並列化手法

本稿は Doacross ループ [1] を依存グラフとしてモデル化する．グラフにおける節はタスク，枝は依存関係に対応する．

タスクは，ループボディ内の 1 つの命令，文，基本ブロックのいずれかに相当する．データ依存関係には，フロー依存，逆依存，出力依存の 3 種類がある．さらに，イタレーション間の依存関係には，ループ運搬依存 (loop carried dependence) とループ独立依存 (loop independent dependence) に分類される [3]．

ループにおける依存グラフでは，ループ運搬依存によって循環が生じる部分とそうでない部分の 2 つの種類に分けることができる．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ことにより，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．

sandglass 型並列化手法では，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 上で実行する．ループ運搬依存タスクより非ループ運搬依存タスクの粒度が大きい場合，非ループ運搬依存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に実行する (図 2)．

実行過程を図 3(a) に模式的に示す．この手法により，クリティカルパス上の通信レイテンシの回数は，最初 (PE0 → PE3) と最後 (PE3 → PE5) の 2 回のみとなる．また，分割したイタレーションのブロック化により通信のブロック化が可能で，通信回数を減らし，送受信の準備のオーバヘッドを減らすことも可能である (図 3(b))．

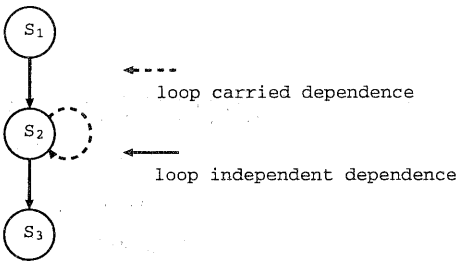


図 1: Doacross loop の依存グラフ

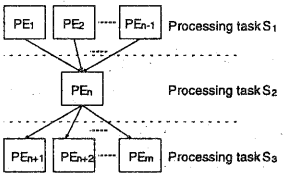


図 2: sandglass 型並列化手法のタスク割り当て

## 3 分割配置されたデータへのアクセス方法

図 2の矢印で示す PE 間通信は，ループ独立依存な依存関係により必要な通信である．

各 PE 内で参照，定義されるデータが，ローカルメモリ上にない場合には，他 PE からデータの転送が必要となる．

$S_2$ で参照される配列データが分割配置されている場合，通信により読み出す方法として次の 3 つが考えられる．

**読み出し：**

1. イタレーションの実行中に必要となった時点でリモートメモリリードでデータを読み出す方法．
2. ループ開始前に全ての配列データをあらかじめ一括して集めておく (Gather) 方法．

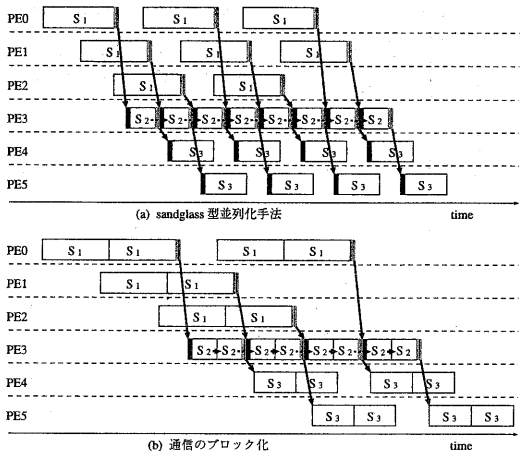


図 3: 並列実行の過程と通信のブロック化

3. イタレーションの実行中に，$S_1$で前もってデータを読み出し，$S_2$を処理する PE へデータを転送を行なう方法.

また，$S_2$で定義された配列データを他 PE に書き込む方法として 3 つある.

**書き込み：**

1. 定義されたデータの計算が終った時点でデータを書き込む方法.

2. ループの終了後に $S_2$を処理した PE が，データを一括して分散させる (Scatter) 方法.

3. 定義されたデータを通信により $S_3$を処理する PE へ転送し，$S_3$を処理する PE が書き込む方法.

なお，$S_2$以外の $S_1$, $S_3$で参照，定義される配列データは，クリティカルパスの処理時間に大きな影響を与えないと考え，全て必要となった時点で読み書きすることとした.

# 4 評価環境

評価実験には，電子技術総合研究所で開発された EM-X 上 [2] を用い，プログラミングには，EM-X 用の記述言語 EM-C[4] を用いた.

```
do i = 2, N
      a[i] = a[i] + a[i-1]
      b[i] = a[i] + c[i]
continue
```

図 4: ループプログラム

対象プログラムとして図 4を用いた．ループボディの 1 行目の左辺の a[i] と 右辺の a[i-1] にループ運搬依存が存在する．使用した PE 数は 64 台で，ループ回数は，8190 とした．sandglass 型並列化手法では，ループ運搬依存の存在する 1 行目 ($S_2$に相当) を同一の PE で計算するので，ループ運搬依存のあるデータを通信で送受信する必要がない．しかし，1 行目で定義された a[i] から 2 行目 ($S_3$に相当) の a[i] の参照へループ独立依存があり，そのため通信が必要となるので，EM-X の I_structure を使い，データ転送と同時に同期をとる．通信のブロック化を行なう場合は，(送るデータの数−1) 個のデータをリモートメモリライトで書き込み，最後の 1 個を I_structure を使って同期とデータ転送を同時に行なう.

2 行目の c[i] の参照は，クリティカルパス上にはないので，リモートメモリリードで行なうことにする．b[i] の書き込みもクリティカルパス上にはないので，リモートメモリライトとする.

また，比較対象として，ループ内でのデータ受信以外のデータ参照による通信以外を行なわない場合を考える．つまり，ループの開始時点では，全ての PE が同じ配列データのコピーを持ち，ループの終了後では，各 PE で定義されたデータは計算した PE のローカルにしか存在しないと仮定した場合である.

## 4.1 分割配置による評価実験

## 4.2 ブロック分割

図 4の配列変数 a，c の初期配置をブロック分割とした場合の sandglass 型並列化手法で並列化したループの実行時間を図 5に示す．計算結果が代入される配列 a と配列 b はアライン

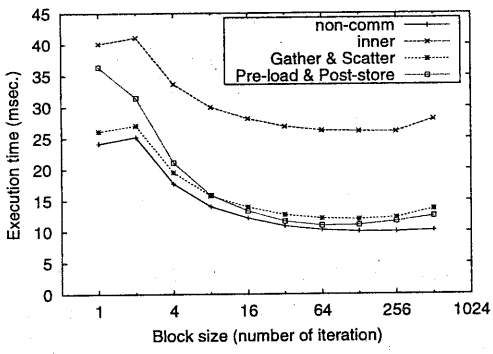


図 5: ブロック分割のときの実行時間とブロックサイズ

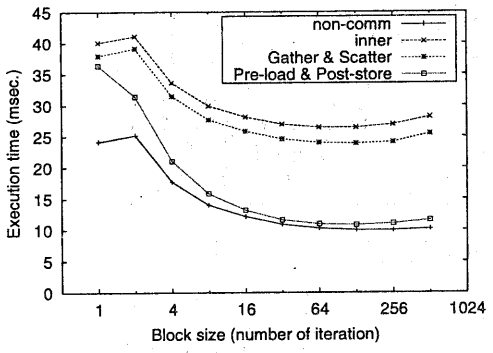


図 6: サイクリック分割のときの実行時間とブロックサイズ

されたブロック分割とする．縦軸はループの実行時間を示し，横軸は，通信をブロック化したときのブロックサイズを示す．

inner では，3 章における読み出しが 1 と書き込みが 1 の組合せで行なった場合である．

Gather & Scatter では，3 章における読み出しが 2 と書き込みが 2 の組合せで行なった場合である．

Pre-load & Post-store では，3 章における読み出しが 3 と書き込みが 3 の組合せで行なった場合である．

図中の non-comm では，全ての PE が配列データのコピーを持っている場合を示す．

ブロックサイズが 1 の場合に比べ 2 の場合に実行時間が長くなっている理由は，通信のブロック化によって生じるブロック化の処理オーバヘッドのためと考えられる．

inner では，リモートメモリリードで PE 間の通信時間が必要になり，それがオーバヘッドとなるため，ループの実行時間が遅くなっている．その他のものは，ほぼ等しい値であるが，Pre-load & Post-store を使った方が最もよい結果であった．ブロックサイズが 64 のとき，ループの実行時間が 10.9msec. であった．配列データが全ての PE にコピーされているとした場合，ブロックサイズが 256 のとき，実行時間が 10.0msec. であった．Pre-load & Post-store を使った場合は，全ての PE が配列データのコピーを持つ non-comm の場合に近付くことがわかった．Pre-load & Post-store を用いることで，ループ運搬依存以外のデータの通信に影響を受けにくい，つまり，sandglass 型並列化手法ではデータの分割配置の影響が少ないことがわかった．

## 4.3 サイクリック分割

配列変数 a，c の初期配置をサイクリック分割とした場合の sandglass 型並列化手法で並列化したループの実行時間を図 6に示す．計算結果が代入される配列 a と配列 b はアラインされたサイクリック分割とする．

inner の実行時間が長い理由は，ブロック分割の場合と同じである．Gather & Scatter で，実行時間が長くなる理由は，ブロック分割の場合では，受け取るデータの配列が連続しているので，受け取るデータを連結すればよいが，サイクリック分割の場合で Gather & Scatter を使った場合には，配列が連続していないので並べ変える必要があり，その処理時間の分だけ全体の実行時間が長くなるためである．

Pre-load & Post-store を使った場合は，ブロックサイズが 128 台のとき，ループの実行時間は 10.8msec. であった．全ての PE で配列データのコピーを持つ non-comm の場合に最も近い方法である．また，ブロック分割の場合のループの実行時間を比べると 1%の差しかなく，データの分割配置に影響が少なく実行でき

ることがわかる．

# 5 他の並列化方法との比較

## 5.1 並列化方法

従来，よく用いられる Doacross ループの並列実行方法として，1つのイタレーションを単位として PE に割り当て，並列実行する方法 [1] がある．これを Do-across 法と呼ぶこととする．

実験では，イタレーションの割り当ては，owner computes rule で行なわれる．

ループボディをいくつかの実行段階に分割し，パイプラインを構成し並列実行する方法 [7] がある．これを Do-pipeline 法と呼ぶことにする．

配列データはループの開始前に集め (Gather)，ループの終了後に分散させた (Scatter)．

最も一般的な方法としては，並列で実行可能な非ループ運搬依存タスクを Do-all ループで実行し，ループ運搬依存タスクを逐次ループで実行する方法がある．この方法を Do-all/seq. 法と呼ぶことにする．

実験では，逐次ループの開始前に配列データを集め (Gather)，逐次ループの終了後に分散させた (Scatter)．Do-all ループでは，イタレーションの割り当ては，owner computes rule で行なわれる．

また，別の方法としてループ間 Doacross 方式 [6] が提案されている．この方法は，Do-all/seq. 法において，Do-all ループと逐次ループを融合し，$k$個のイタレーション毎に部分ループ化 (ブロック化) し，部分ループを PE に割り当てる．この方法より PE 間の通信回数を Do-across 法と比べ $1/k$倍にすることができる．配列データの参照と定義には，Pre-load と Post-store を使った．

また，sandglass 型並列化手法で行なう場合を Do-sandglass と呼ぶことにする．配列データの参照と定義には，前述の評価実験で最もよかった Pre-load と Post-store を使った．

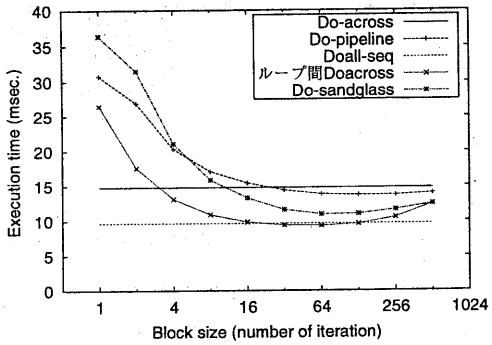


図 7: ブロック分割における各並列実行方法の実行時間

## 5.2 ブロック分割

ブロック分割の場合で，各並列化方法の実行時間を比較する．Do-across と Do-all/seq. は，ブロック化には影響を受けない．その理由は，Do-across では，イタレーションの PE への配置を静的に決めているからで，Do-all/seq. では，既に Gather と Scatter で一括転送を行なっているからである．

最もよい結果は，ループ間 Doacross で，ブロックサイズが 64 のときに実行時間が 9.3 msec. であった．Do-sandglass のときの実行時間がループ間 Doacross のときより遅い理由は，ループ間 Doacorss では，ブロックサイズを変えても各 PE 間の通信量は常に 1 個であるが，Do-sandglass では，ブロックサイズを大きくすると 1 回の通信での通信量が増えるためである．

Do-all/seq. の方が Do-sandglass よりもよい結果となっている理由は，ブロック分割では，配列データが連続しているのでの収集 (Gather)，分割 (Scatter) が簡単に行なえるために，データの再配置が高速に行なわれるためである．

## 5.3 サイクリック分割

サイクリック分割の場合で，各並列化方法の実行時間を比較する．ブロック分割同様，最もよい結果は，ループ間 Doacross であった．ループ間 Doacross で，ブロックサイズが 64 のと

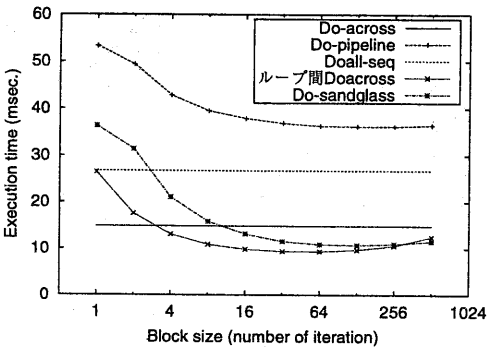


図 8: サイクリック分割における各並列実行方法の実行時間

き，ループの実行時間が 9.4msec. であった．

今回実験に用いたループは，ループ運搬依存が1つしかないが，対象とするループのループ運搬依存の数が多いプログラムの場合，ループ間 Doacross では，ループ運搬依存のデータを全て通信しなければならないために通信量が増える．Do-sandglass では，ループプログラムでループ運搬依存のあるデータが増えても，同じ PE 内で処理するので通信が必要ない．そのため，ループ間 Doacross よりも高速に実行できる可能性もある．

## 6 まとめ

sandglass 型並列化手法により並列化されたループの実行時間がブロック分割およびサイクリック分割によりほとんど影響を受けないことがわかった．

また，それぞれの並列化手法で並列化されたループの実行時間の比較で，sandglass が常によいとは限らないこともわかった．これは，対象とするループプログラムにより並列化の方法を変えることで，ループの実行時間が最短のものを選ぶようにすればよい．

今後の課題として，ブロックサイクリック分割の場合について，評価を行なう必要がある．また，対象とするループプログラムに対して，最適な並列化手法を決める方法を考える必要がある．

## 謝辞

EM-X の利用環境をご提供頂いた電子技術総合研究所アーキテクチャラボの方々に感謝致します．

## 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] 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).

[3] Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, Addison-Wesley Publishing Company, Inc. (1991).

[4] 佐藤三久, 児玉祐悦, 坂井修一, 山口喜教: 並列計算機 EM-4 の並列プログラミング言語 EM-C, 並列処理シンポジウム JSPP'93, pp. 183–190 (1993).

[5] 高畠志泰, 本多弘樹, 大澤範高, 弓場敏嗣: Doacross ループの並列化手法とその評価, 並列処理シンポジウム JSPP'98, pp. 367–374 (1998).

[6] 山名早人, 佐藤三久, 児玉祐悦, 坂根広史, 坂井修一, 山口喜教: 並列計算機 EM-X におけるループ間 Doacrosss 方式の自動最適化, 情報処理学会研究報告書, No. 106, pp. 17–24 (1994).

[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).