Doacross ループの sandglass 型並列化方法とその評価

高畠志泰[†] 本多弘樹[†]
大澤範高[††] 弓場敏嗣[†]

本論文では，1重の Doacross 型ループを対象とした新たな並列化方法を提案する．本方法は，従来のイタレーション単位で分割されたタスクを並列実行する方法と，パイプライン並列処理を用いて並列実行する方法の両方の特徴を持つ並列化方法である．すでに提案されている Doacross 型ループの並列化手法について LogP モデルを用いて並列プログラムの実行時間を解析し，実機による評価結果と比較する．その結果，提案する手法が他の方法に比べ，より効率的に実行できることを示す．

A Sandglass Type Parallelization Technique for a Doacross Loop

MOTOYASU TAKABATAKE,[†] HIROKI HONDA,[†] NORITAKA OSAWA[††]
and TOSHITSUGU YUBA[†]

In this paper, we propose the sandglass-type parallelization technique for a doacross loop which has the characteristics of iteration-based parallelizing and software pipelining. We prove its effectiveness by comparing the sandglass-type to well-known parallelization techniques on the basis of the LogP model and on a real parallel machine. We conclude that the sandglass-type parallelization technique is the most effective among the techniques.

1. はじめに

並列プログラムを効率的に実行するには，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（PE）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．本論文では，1重の Doacross 型のループ[2] 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する．

Doacross ループとは，イタレーション間に依存関係が存在するために，Doall 型の並列処理が不可能なものである．Doacross ループの並列実行方法には，ループの1つのイタレーションを1つのタスクとし，各 PE に割り当て並列に実行する方法[2),6)] が一般的である．ループボディをいくつかに分割し，パイプラインを構成して並列に実行する方法[10)] も提案されている．これらの方法は，共有記憶型の並列計算機を前提としている．また，ループ分割を行い，Doall 型ループと逐次ループに分けて，実行する方法も知られている．さらに，ループ間 Doacross 方式[9)] と呼ばれる手法も提案されており，これは，従来のイタレーションで分割した方法を分散記憶型の並列計算機向けに改良したものである．

本論文では，上記のイタレーションで分割する方法とパイプラインを構成する方法の特徴を合わせ持った新たな並列化方法を提案する．本方法は，共有記憶型と分散記憶型の両方を対象計算機とする．LogP モデル[1)] を使って，すでに提案されている並列実行方法の実行時間と通信回数，使用 PE 数を比較することで提案する方法の有効性を示し，実機を用いた評価を行う．

2. Doacross ループのモデル

2.1 Doacross ループ

本論文で扱う Doacross ループ[2)] のモデル化を示す．ループボディ内のタスクの依存関係を有向グラフで表現する．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．また，

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† メディア教育開発センター
National Institute of Multimedia Education

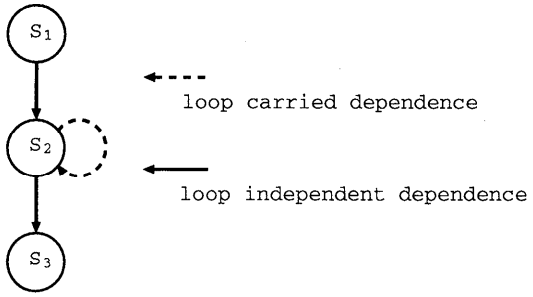


図 1 Doacross loop の依存グラフ
Fig. 1 Dependence graph of a doacross loop.

イタレーション間の依存関係には，以下の 2 種類がある[4]．

(1) ループ運搬依存（loop carried dependence）：同一ループ内の異なる 2 つのイタレーション間に存在する依存関係．

(2) ループ独立依存（loop independent dependence）：同一ループ内の 1 つのイタレーション内に存在する依存関係．

ループ運搬依存において，あるイタレーション内のタスクが n 個前のイタレーション内のタスクに依存する場合，これをループ依存距離 n という．

ループにおける依存グラフは，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 1 つのタスクとし，タスクサイズを大きくすることにより，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．

本論文では，以下の条件を満たすループを扱う．

(1) 1 重の Doacross ループである．

(2) ループ依存距離は 1 とする．

(3) ループ運搬依存の数は 1 以上とする．

(4) 条件分岐がループ内に存在しない．

ループ運搬依存タスク S_2 では，ループ運搬依存は複数存在してもかまわない．ループ依存距離は 2 以上の場合もあるが，本論文では簡単化のために 1 とする．

2.2 LogP モデル

並列計算機上での並列プログラムの実行時間を評価するために LogP モデル[1] を用いる．同モデルにおいては，並列計算機の PE 数と PE 間の通信時間を以下の 4 つのパラメータを使って表す．

- L：メッセージを送るのに要する時間の遅れの上限（通信遅延）．
- o：PE においてメッセージの送受信に要する時間（送受信オーバヘッド）．
- g：メッセージを連続して送受信するときの最小時間間隔（バンド幅の逆数）．
- P：PE 数．

対象とする計算機にはブロック転送機能があり，計算と通信を独立に行えるものを想定したものとする．ブロック転送により複数の送受信オーバヘッドの回数を 1 回に減らすことができる．また，通信遅延はメッセージ長が変わっても一定と仮定する．これらのパラメータを用いて，Doacross ループの実行時間と最適 PE 数を求める．

3. 並列実行の方法

3.1 実行方法の比較

図 2 はループ繰返し回数（イタレーション数）を 8 とした場合における Doacross ループの各方法の実行例である．図 2 (a) では，図 1 の各タスクの 1 イタレーション分の実行時間と依存関係を示している．

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，並列実行する方法がある．これを本論文では，Do-across 法と呼び，図 2 (b) に示す．

Do-across 法は，1 つのイタレーションを 1 つのタスクにするため，タスク生成や並列実行が容易である．しかし，ループ運搬依存による PE 間通信が実行時間のクリティカルパス上に現れるので実行時間が長くなる．

2 つ目の方法として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がある．これを Do-pipeline 法と呼ぶことにする（図 2 (c)）．Do-pipeline 法では，ループ運搬依存タスクを 1 つのステージとし，それを 1 つの PE 内で処理するので，ループ運搬依存による PE 間通信がなく，実行効率が良い．また，通信のブロック化により各ステージ間の通信回数を減らすことも可能である．パイプラインのステージの数を増やすことで並列度を上げることもできる．図 2 (c) においては，S_1 を S_{11} と S_{12}，S_3 を S_{31} と S_{32} に分割し，5 台の PE で実行している．この方法の問題点は，データ依存関係によりパイプラインピッチを揃えることができない場合，最も長いパイプラインピッチを持つステージによって実行効率が下がってしまうという点である．図 2 (c) では，S_{11} が最もパイプラインピッチが長い場合となっている．

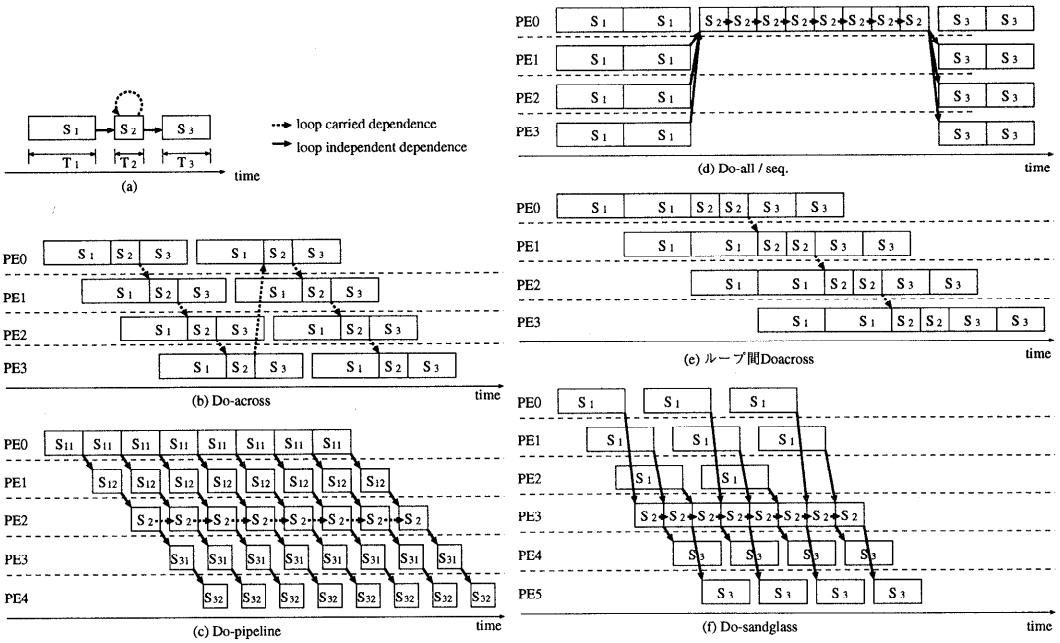


図 2 並列実行方法
Fig. 2 Gantt chart of five parallelization techniques for a doacross loop.

最も一般的な方法としては，並列に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（図 2 (d)）．この方法では，ループ運搬依存の通信回数を減らし，通信オーバヘッドを大きく軽減することができる．しかし，逐次実行する部分が並列処理のボトルネックになる．

別の方法としてループ間 Doacross 方式[9] 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ごとに部分ループ化し，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．この方法の実行過程を図 2 (e) に示す．

ここで，新しい実行方法を提案する．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が粒度が大きい場合，非ループ運搬依存タスクをイタレーションごとに異なる 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 (f)）．この方法を Do-sandglass 法と呼ぶことにする（**図 3**）．

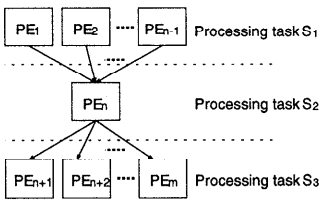


図 3 Do-sandglass における PE 間通信
Fig. 3 Communications among PEs on do-sandglass.

この方法により，Do-across 法で問題になる異なるイタレーションのループ運搬依存タスク間の通信遅延がクリティカルパス上からなくなる．また，Do-pipeline 法で問題になるパイプラインピッチを揃えることが必要なくなる．この方法では，通信のブロック化により通信回数を減らすことも可能である．

この方法が有効である条件は，ループ運搬依存タスクが非ループ運搬依存タスクより小さいことがあげられる．これは，ループ運搬依存タスクが非ループ運搬依存タスクより大きい場合，パイプラインでループ運搬依存タスクが最もパイプラインピッチの大きいステージになる．そのため，非ループ運搬依存タスクをイタレーションごとに並列に実行しても，ボトルネックになるループ運搬依存タスクによって高速化がされない．

3.2 各実行方法の実行時間

Doacross ループにおいて，ループ運搬依存が複数

あり，1つのイタレーションで複数のデータが送受信される場合，通信における送受信処理のオーバヘッドを考慮し，ブロック転送を行い1回の通信で送受信すると仮定する．各々の方法におけるループの実行時間をLogPモデルを用いて表す．LogPモデルにおいて，送受信オーバヘッド o を送信オーバヘッド o_s と受信オーバヘッド o_r に分けて考える．また，本論文では通信のブロック化によりメッセージ長を長くできるとする．一般的には，N ワードのメッセージの送受信には，N 回の通信を行う必要があり，送受信オーバヘッド，レイテンシは N の影響を受ける．ここでは，通信のブロック化により N 回の通信を1回にすることで，送受信オーバヘッド，レイテンシは N に影響を受けなくなると仮定する．

ループの繰返し回数を N，図2(a)における S_i $(i = 1 \sim 3)$ の実行時間を各々T_i とする．Do-across法で実行した場合のループの実行時間 T_{across} は，以下の式で与えられる．

$$T_{across} = T_1 + NT_2 + T_3 + (N-1)(o_s + L + o_r) \quad (1)$$

第1，2，3項は，各 S_1，S_2，S_3 の実行時間である．第4項は，通信にかかる時間である．

Do-pipeline法で実行した場合を考える．パイプラインのステージ数を S，各ステージの実行時間を T_{Si} $(i = 1 \sim S)$ とする．この方法では，通信のブロック化が可能なので，ブロックサイズを k とする．Do-pipeline法でのループの実行時間 T_{pipe} は，以下の式で与えられる．

$$T_{pipe} = (N-k)\max_{i=1,S} T_{Si} + k\sum_{i=1}^{S} T_{Si} + \left(\left\lceil \frac{N}{k} \right\rceil + S - 2\right)(o_s + o_r) + (S-1)L \quad (2)$$

第1，2項は，最も大きいステージと各ステージの実行時間である．第3，4項は通信にかかる時間である．

Do-all/seq.法で実行した場合を考える．この方法で実行した場合のループの実行時間 T_{all} は，以下の式で表される．

$$T_{all} = \frac{N}{P}T_1 + NT_2 + \frac{N}{P}T_3 + (P+1)(o_s + o_r) + 2L \quad (3)$$

第1，2，3項は，各 S_1，S_2，S_3 の実行時間である．第4，5項は，通信にかかる時間である．

ループ間Doacross法で実行した場合を考える．この方法で実行した場合のループの実行時間 T_{loop} は，以下の式で表される．

表1 通信オーバヘッドの比較
Table 1 Comparison of communication overheads.

実行方法	L の回数	o_s，o_r の回数
Do-across	$N-1$	$N-1$
Do-pipeline	$S-1$	$\lceil N/k \rceil + S - 2$
Do-all/seq.	2	$P+1$
ループ間 Doacross	$\lceil N/k \rceil - 1$	$\lceil N/k \rceil - 1$
Do-sandglass	2	$\lceil N/k \rceil$

$$T_{loop} = kT_1 + NT_2 + kT_3 + \left(\left\lceil \frac{N}{k} \right\rceil - 1\right)(o_s + L + o_r) \quad (4)$$

ほとんどDo-across法と同じであるが，ブロック化を行っているため式に k が付加されている．

Do-sandglass 法で実行した場合を考える．Do-sandglass法で実行した場合のループの実行時間 T_{sand} は，以下の式で表される．

$$T_{sand} = kT_1 + NT_2 + kT_3 + \left\lceil \frac{N}{k} \right\rceil (o_r + o_s) + 2L \quad (5)$$

この方法でもブロック化が可能であるため式に k が付加されている．第1，2，3項は，各 S_1，S_2，S_3 の実行時間である．第4，5項は通信にかかる時間である．

3.3 通信オーバヘッドの比較

各方法における通信オーバヘッドの比較を**表1**に示す．それぞれの方法について，クリティカルパス上に現れる通信オーバヘッドの通信遅延（L）と送受信オーバヘッド（o_s，o_r）の回数を比較する．

Do-across法では，ループを並列実行する場合，すべての通信遅延と送受信オーバヘッドがクリティカルパス上に現れるために実行に時間がかかる．

Do-pipeline法では，ステージ数によって通信回数が変わる．この方法では，通信のブロック化（ブロックサイズ k）により送受信オーバヘッドの回数を減らすことができる．

Do-all/seq.法では，通信遅延の回数と送受信の回数がループ回数に依存しないため，他の方法と比べ，通信遅延の回数と送受信の回数が最も少ない．しかし，1度に通信するデータ量は，他の方法と比べると最も多くなり，また，通信のタイミングが重なるため通信網の飽和状態が起きやすくなる．

ループ間Doacross法では，複数のイタレーションを1つのPE上に割り当ててブロック化（ブロックサイズ k）し，通信遅延の回数を減らしている．Do-across法と比べ $1/k$ 倍の回数に減らすことができる．

Do-sandglass 法では，通信遅延の回数は，Do-all/seq.法と同じ2回である．送受信回数は，Do-

all/seq. ほど減らすことができないが，通信のブロック化（ブロックサイズ k）により減らすことが可能である．しかし，S_2 を処理する PE へ通信が集中するために，通信データの衝突が起きる可能性がある．

3.4 各実行方法の使用 PE 数

Do-across 法での使用 PE 数 P_a は，以下の式で表される．

$$P_{across} = \left\lceil \frac{T_1 + o_r + T_2 + o_s + T_3}{o_r + T_2 + o_s + L} \right\rceil \quad (6)$$

この式で求まる PE 数は，Do-across 法で実行する場合において無駄な時間を生じない場合の PE 数の上限である．求まる PE 数より少ない PE 数で実行することも可能である．

Do-pipeline 法での使用 PE 数 P_{pipe} はパイプラインのステージ数と等しい．

$$P_{pipe} = S \quad (7)$$

この式は，ステージ数 S によって任意の PE 数を使うことができることを意味する．

Do-all/seq. 法での使用 PE 数 P_{all} は，任意の PE 数を使うことができる．

$$P_{all} = P \quad (8)$$

ループ間 Doacross 法での使用 PE 数 P_{loop} は，文献 9) では，実機のすべての PE 数を使っている．

$$P_{loop} = P \quad (9)$$

Do-sandglass 法における使用 PE 数 P_{sand} では，以下の式で与えられる．

$$P_{sand} = \left\lceil \frac{kT_1 + o_s}{o_r + kT_2 + o_s} \right\rceil + 1 + \left\lceil \frac{o_r + kT_3}{o_r + kT_2 + o_s} \right\rceil \quad (10)$$

この式で求まる PE 数は，Do-sandglass 法で実行する場合において無駄な時間を生じない場合の PE 数の上限である．第 1 項は，S_1 を実行するための PE 数，第 2 項は，S_2 を実行するための PE 数，第 3 項は，S_3 を実行するための PE 数である．

4. 評価実験

4.1 各方法における実験結果

評価実験には，電子技術総合研究所の EM-X 上[3] を用いた．対象とするループは，NAS Kernel Benchmark Program の一部分（**図 4**）を抜き出し並列化した．このループには，cp(k,l) と cp(k-1,l) および cpm どうしのループ依存距離が 1 の 2 つのループ運搬依存が存在する．

ループのタスク分割は，図 4 (a) のプログラムから図 4 (b) のように分割した．この例では，図 1 における S_3 に相当するタスクは存在しない．

```
      do 9 k = 2, nwall(l)
        cp(k,l) = cp(k-1,l)
     &          + (3. * (dpds(k,l) + dpdp(k-1,l))
     &          + dpds(1+mod(k,nwall(l)), l)
     &          + dpds(1+mod(k+nwall(l)-3,nall(l)),l)
     &        / (4. * DELT * cupst)
        cpm = max(cpm, cp(k,l))
9     continue
```

(a)

```
      do 91 k = 2, nwall(l)
        tmp(k) = (3. * (dpds(k,l) + dpdp(k-1,l))
     &          + dpds(1+mod(k,nwall(l)), l)
     &          + dpds(1+mod(k+nwall(l)-3,nall(l)),l)
     &        / (4. * DELT * cupst)
91    continue

      do 92 k = 2, nwall(l)
        cp(k,l) = cp(k-1,l) + tmp(k)
        cpm = max(cpm, cp(k,l))
92    continue
```

(b)

図 4 評価プログラム（NAS Kernel Benchmark Program の一部）
Fig. 4 Evaluation program (part of NAS Kernel Benchmark Program).

表 2 EM-X における LogP モデルパラメータの値
Table 2 Values of LogP model parameters.

パラメータ	値（clock cycle）
o_s	8
o_r	23
L	8
T_1	207
T_2	49
T_3	0

EM-X における LogP モデルのパラメータは，**表 2** のようになる．この値は，ある 1 つの PE で送信し，別の 1 つの PE で受信することを実機上で繰り返し行ったときの平均値である．各々のタスクの実行時間は，逐次実行のときのアセンブラの命令数である．ループ回数 N は 10000 とする．

それぞれの方法においてブロックサイズを 1 としたときの実行結果を**図 5** に示す．ループ間 Doacross 法では，ブロックサイズを 1 としたとき，Do-across 法と同じになるので省略する．

データの初期配置は，各 PE が同じデータを持った状態とした．分散記憶型並列計算機における一般なプログラムでは，ループが始まる前にデータの再配置を行い，ループの実行が終わった後で次の処理に合わせたデータの再配置が必要である．しかし，データの分割方法やデータの再配置の方法を考えると問題が複雑

表 3 実験結果と理論値の比較
Table 3 Comparison between extermination results and ideal values.

	実験結果		理論値	
実行方法	PE 数	実行時間 (msec.)	PE 数	実行時間 (msec.)
Do-across	4	39.1	4	40.5
Do-pipeline	4	58.5	4	47.5
Do-all/seq.	10	38.3	10	34.9
Do-sandglass	4	36.6	4	39.5

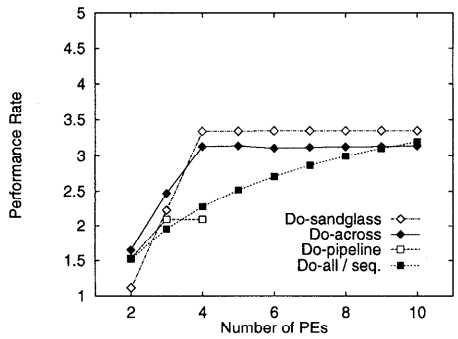


図 5 ブロックサイズ 1 での実行結果
Fig. 5 Execution results of block size 1.

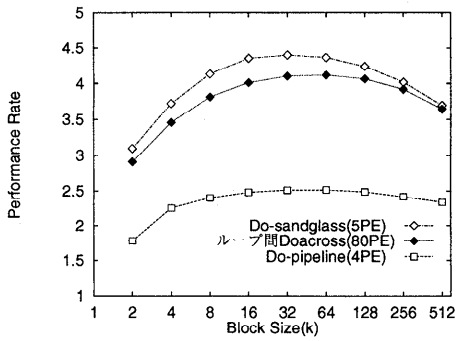


図 6 ブロック化による効果
Fig. 6 Effects of blocking.

になる．本論文では，問題の簡単化，また，予備的な評価としてそのようなデータの配置を考えず，並列化されたループの実行時間のみの実行時間を比較する．また，共有記憶型の並列計算機での評価とみることもできる．

Do-pipeline 法において，実験はパイプラインのステージ分割を 5 以上にすることが，このプログラムでは不可能になったために 4 台までしか行えなかった．

各方法の中で最も高速に実行できたのは，提案する Do-sandglass 法である．逐次との比較で約 3.3 倍であった．

式 (1)，(2)，(3)，(5)，(6)，(7)，(8)，(10) の値をそれぞれ求め，各方法での実験による最適な実行結果と比較を**表 3** に示す．

理論値では，Do-all/seq. 法が最も良い結果になっているが，実験では，Do-sandglass 法より悪い結果になっている．これは，Do-all/seq. 法では通信を一括処理するため，通信網が飽和状態になり通信遅延が大きくなったためだと考えられる．

Do-pipeline 法が理論値の実行時間が実験値の実行時間より短くなっている理由は，パイプラインの分割を理想的に分割した場合を想定したためである．Do-across 法と Do-sandglass 法では，理論値で実行時間を近似できている．

使用 PE 数では，Do-across 法と Do-sandglass 法は，式 (6)，(10) から得られた値を示す．他の方法は，任意に PE 数を決めることができるが，実験した PE 数で実行時間の理論式から得られる最短の実行時間のときの PE 数を理論値として示す．どの方法においても実験結果と理論値が一致した．LogP モデルを用いて表現した理論式が正しいことが分かる．

4.2 ブロック化による効果

Do-pipeline 法と Do-sandglass 法は，通信のブロック化が可能である．ループ間 Doacross 法では，ループを部分的にブロック化している．両者のブロック化は，通信処理が終わってからブロック数のイタレーションの処理を行い次の通信処理を行うと考えると同じものとして扱える．

ブロック化した結果を**図 6** に示す．使用 PE 数は，Do-pipeline 法では式 (2) より 4 台，ループ間 Doacross 法では文献 9) より 80 台とした．Do-sandglass 法は，式 (5) から 5 台とした．どの方法でもブロック化を行うことで実行効率を上げることができる．最も良い結果を得たのは，Do-sandglass 法であった．ブロックサイズ 32 で，逐次との比較で約 4.4 倍である．提案する手法がブロック化することでさらに有効になることが分かる．

5. ま と め

Doacross ループの新しい並列化手法を提案した．また，既存の並列化方法と実行時間，通信回数，使用 PE 数の 3 つの点で比較し，実機を用いて有効性を検証し

た．EM-X 上で逐次実行と比べ約 3 倍の速度向上がみられた．さらに，通信のブロック化を行うことで逐次実行と比べ約 4.4 倍の速度向上が得られた．

本手法をコンパイラへ組み込む場合には，LogP モデルを使った式を利用することで，容易に最適化を行うことができる．

今後の課題としてループ依存距離が 1 より大きい場合について考えることがあげられる．ループ運搬距離が長いと依存関係のあるタスクどうしの間にある処理は，そのタスクとは依存関係がないので，並列に処理することが可能である．また，データの初期配置とループが終わった後のデータの再配置まで考慮した実行時間の比較が必要である．今回の評価実験では，すべてのデータが全 PE 上に存在するとし，データの再配置を行っていない．一般的な分散記憶型用の並列プログラムではデータは分割されていて，ループを実行する前後，あるい実行中にデータの再配置が必要である．データの再配置は，データの分割方法によって通信回数や通信量が異なり処理時間も異なる．プログラム全体の実行時間を考えると，データの再配置の処理時間を考慮する必要がある．

謝辞 本研究の一部は文部省科学研究費（基盤研究(B)(2) 課題番号 10480057「並列処理粒度の調整機能をもつ並列化コンパイラの研究」）による．EM-X の利用環境をご提供いただ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いたします．有益なコメントをいただいた査読者の方々に感謝いたします．

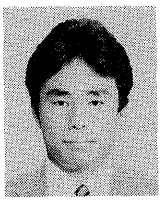
参 考 文 献

1) Culler, D.E., Karp, R., Patterson, D., Sahay, A., Schauser., K., Santos, E., Subramonian, R. and Eicken, T.V.: LogP: Towards a Realistic Model of Parallel Computation, *Proc. 4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp.1–12 (1993).
2) Cytron, R.: Doacross: Beyond Vectorization for Multiprocessors, *Proc. Int. Conf. on Parallel Processing*, pp.836–844 (1986).
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) Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, Addison-Wesley (1991).
5) 中西恒夫，城 和貴，Polychronopoulos, C.D.，荒木啓二郎，福田 晃：ループ最小並列実行時間を算出する一手法，並列処理シンポジウム JSPP'96, pp.57–64 (1996).
6) 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).
7) 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).
8) 高畠志泰，本多弘樹，弓場敏嗣：Doacross ループの sandglass 型並列化手法の有効性について，情報処理学会研究報告，98-ARC-128/98-HPC-70, Vol.98, No.18, pp.133–138 (1998).
9) 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).
10) 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，並列処理シンポジウム JSPP'97, pp.197–204 (1997).

(平成 10 年 8 月 31 日受付)
(平成 11 年 3 月 5 日採録)

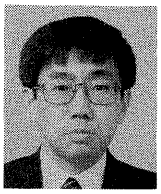
高畠 志泰（学生会員）

昭和 47 年生．平成 7 年電気通信大学電気通信学部情報工学科卒業．平成 9 年同大学大学院情報システム学研究科情報ネットワーク学専攻博士前期課程修了．現在，同大学院博士後期課程在学．並列化コンパイラ，とくにスケジューリング法の研究に従事．



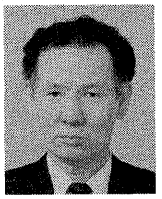
本多 弘樹（正会員）

昭和 36 年生．昭和 59 年早稲田大学理工学部電気工学科卒業．平成 3 年同大学大学院博士課程修了．昭和 62 年から平成 3 年同大学情報科学研究教育センター助手．平成 3 年山梨大学工学部電子情報工学科専任講師，平成 4 年同大学助教授．平成 9 年電気通信大学大学院情報システム学研究科助教授，現在に至る．工学博士．並列処理方式，並列化コンパイラ，マルチプロセッサアーキテクチャ，マルチプロセッサ OS 等の研究に従事．電子情報通信学会，IEEE，ACM 各会員．



大澤 範高（正会員）

昭和58年東京大学理学部情報科学科卒業．昭和60年同大学大学院理学系研究科情報科学専攻修士課程修了．昭和63年同博士課程修了．ソフトウェア開発会社を経て，平成5年電気通信大学大学院情報システム学研究科助手．平成10年文部省メディア教育開発センター助教授．理学博士．並列分散システムソフトウェアに興味を持つ．電子情報通信学会，ACM，IEEE-CS各会員．



弓場 敏嗣（正会員）

昭和16年9月22日生．昭和41年3月神戸大学大学院工学研究科修士課程修了．野村総合研究所を経て，昭和42年通商産業省工業技術院電気試験所（現，電子技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機等の研究に従事．その間，計算機方式研究室長，情報アーキテクチャ部長等を歴任．平成5年4月より，電気通信大学大学院情報システム学研究科教授．並列処理一般に興味を持つ．工学博士．電子情報通信学会，日本ロボット学会，日本ソフトウェア科学会各会員．