Vol.2017-ARC-228 No.10
2017/11/8

# 階層アジャスタブルブロックを用いた 自動マルチコア・ローカルメモリ管理とその性能評価

白川 智也[1]　阿部 佑人[1]　大木 吉健[1]　吉田 明正[1,2]　木村 啓二[1]　笠原 博徳[1]

**概要**：自動車におけるハードリアルタイム制御のようにデッドラインを確実に守らなければならないプログラムでは，キャッシュメモリの使用は困難であり，各コアがローカルメモリを持つマルチコアが利用されている．ローカルメモリは小容量であるため，その有効利用のためには配置するデータの選択，データ再利用のためのループ分割，ローカルメモリ上におけるデータの割り当て領域の決定，データ転送の挿入など複雑な管理が必要となり，これをプログラマが手動で管理することは極めて困難である．特に，ローカルメモリ管理対象となる配列はアプリケーションによってさまざまな次元やサイズを持つため，これらのデータをローカルメモリに割り当てる際には共有メモリとローカルメモリ間のデータ転送単位であるブロックサイズを適切に選び，ローカルメモリの効率的な利用が出来るようにする必要がある．そのため本稿ではコンパイラを用いた自動的なローカルメモリ管理，およびその中でもアプリケーションに応じてブロックサイズがソフトウェアコンパイル時に決定され，さらに各ブロックを整数分の 1 に分割して使用できる 階層アジャスタブルブロックにローカルメモリを区切り割り当て領域を決定する手法を提案する．本手法の性能評価を 32KB のローカルメモリを搭載した SH4A を 8 コア集積したマルチコアプロセッサ RP2 上で行ったところ，Nas Parallel Benchmarks のプログラム BT において共有メモリを用いた逐次実行に比べて 4PE 時に 5.76 倍の性能向上を得ることに成功した．

# Automatic Local Memory Management Using Hierarchical Adjustable Block for Multicores and Its Performance Evaluation

Tomoya Shirakawa[1]　Yuto Abe[1]　Yoshitake Oki[1]　Akimasa Yoshida[1,2]　Keiji Kimura[1]
Hironori Kasahara[1]

## 1. はじめに

近年，組込みシステムにおいてプログラムの高速化，および低消費電力化を目指すためマルチコアプロセッサの利用が普及している．これらのシステムにおいて，特にデッドラインを厳守する必要がある自動車のエンジン制御等のリアルタイム処理においては．キャッシュミス時のメモリアクセスペナルティによる実行時間が予測困難であるキャッシュメモリではなく，マルチコアの各コアが持つローカルメモリを利用する必要がある．その際にはプログラマが明示的にローカルメモリの利用を考慮したプログラムの記述を行う必要がある．しかし，ローカルメモリの有効活用のためにはローカルメモリへ配置するデータの選択，データを効率的に再利用するためのループ分割，データを割り当てるメモリ上の領域の決定，データ転送の挿入など複雑なローカルメモリ管理が必要となる．これをプログラマが手動で行うとソフトウェアの生産性の低下につながるため，コンパイラにおける自動的なローカルメモリ管理が必要となる．

従来の研究ではコンパイラによる静的なローカルメモリ

[1] 早稲田大学 理工学術院 基幹理工学部 情報理工学科
Waseda University Department of Computer Science and Engineering.
[2] 明治大学 先端数理科学研究科
Meiji University Graduate School of Advanced Mathematical Sciences.

割り当て [1,2] や，単一プロセッサにおけるローカルメモリ配置変更 [3,4]，マルチコアプロセッサを対象としたループのデータリユース方法 [5,6] 等が提案されている．
一方，筆者らは粗粒度タスク並列処理により，データローカリティの最適化を行う動的なローカルメモリ管理手法を提案している．また，このローカルメモリ管理手法におけるループ整合分割手法を多次元に拡張し，より小さなサイズまでデータサイズを分割して管理を行うデータ多次元整合手法を提案している．[7–12] しかし，これらの従来手法はいずれもプログラム中のデータをローカルメモリを 2 の冪乗分の 1 に分割したサイズの領域に割り当てて管理を行うものであり，メモリ使用効率の低下やメモリ配置不可能なデータが生じることが問題になっていた．
本稿では，これらのローカルメモリ管理手法のデータ割り当て手法を改良し，プログラムのデータをローカルメモリ上の整数分の 1 に分割したサイズの領域に割り当てる，階層アジャスタブルブロックを用いたローカルメモリ管理手法を提案する．更に，提案手法を OSCAR マルチグレイン自動並列化コンパイラに実装し，SH4A ベースのマルチコアプロセッサである RP2 上で評価した結果についても報告する．

## 2. ローカルメモリ管理のためのデータ多次元整合分割

本稿で提案する階層アジャスタブルブロックを用いたローカルメモリ管理を行う際には，まずプログラム中の並列性，データローカリティを解析しコード変形を行う必要が有る．そのために，まず 2.1 節で述べるようにプログラムを粗粒度のタスクに分割し，それらの間の並列性を抽出する．その後，2.2 節で述べるようにタスクの集合で共通して使用される配列データをローカルメモリ上にロードしたままプログラムの実行が出来るように，データ多次元整合分割手法を用いてプログラム中のループを多次元にわたって分割する．

### 2.1 粗粒度タスク並列処理

粗粒度のタスクの分割においてはソースプログラムを基本ブロック（BB），繰り返しブロック（RB），サブルーチンブロック（SB）の 3 種類のマクロタスク（MT）に分割する．また RB や SB の内部にさらに粗粒度タスク性が存在する場合は階層的な分割を行う．MT の生成後，MT のコントロールフロー依存およびデータ依存を解析し，その結果をマクロフローグラフ（MFG）として表現する．MFG に最早実行可能条件解析を行うことで各タスクの並列性を抽出したマクロタスクグラフ（MTG）が生成され，プログラム中の並列性が抽出される．

### 2.2 データ多次元分割

ローカルメモリは小容量であるため，タスク間で共有するデータをメモリに乗せたままタスクを実行するにはデータサイズを分割する必要がある．そのため，データの効率的な配置とリプレース管理のために，イタレーション間依存を考慮したループ分割を行うループ整合分割手法が提案されている [9,10]．しかし，この従来手法の分割対象となるループはプログラム中のループの最外側のループに限られる．プログラムの中に多重ループが存在し，その内側ループの回転数が多い場合にはこの最外側ループの分割のみでは分割後のデータがローカルメモリに乗り切らない場合が存在する．このような場合に多重ループの複数ネストレベルにわたって，つまり多次元配列の複数次元にわたって分割を行うことで更なるデータサイズ縮小を図るデータ多次元分割手法が提案されている [11,12]．この手法を用いたループ分割のコードイメージは図 1 のようになる．こ

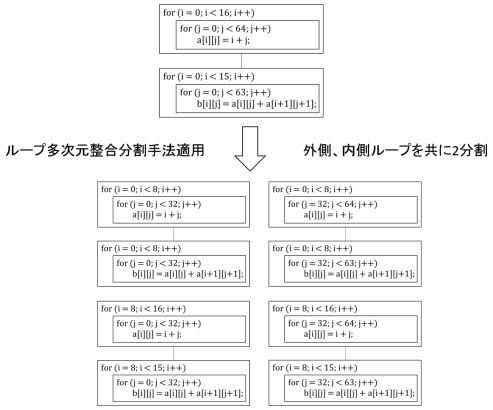


**図 1** データ多次元整合分割適用のコードイメージ

の例では，2 重ネストループの内側と外側をそれぞれ 2 分割することでデータ多次元整合分割を行っている．このように，複数ネストレベルにわたってループを分割することによってデータサイズをより縮小することが可能になり，大きなデータサイズを持つプログラムにおいても有効的なローカルメモリの活用を可能にしている．本データ多次元分割手法は，以下の手順から構成される：まず，ローカルメモリ管理の対象となるループ群を集めてこれをターゲットループグループ（TLG）とする．次に，TLG 中のループに対して多次元に拡張した ILD 解析を行い整合分割を行う．この際コードコンパクション手法を適用する．以下，多次元に拡張した ILD 解析，分割数決定，コードコンパクション手法について説明する．

#### 2.2.1 多次元に拡張した ILD 解析

まず，TLG 中のループに対してループ間データ依存解析（ILD 解析）を適用する．TLG 中の最も推定処理コストの大きなループを標準ループとして選択し，TLG 中のその他のループから標準ループのあるイタレーションに対しての依存を解析する．多次元整合分割の場合，ループの各

次元において依存するイタレーションの上限値，下限値を算出する．ILD 解析を行うことでその後の分割数の決定やコード変形をローカリティを考慮して行うことが出来る．図 1 上部に示した 2 つ目のループが標準ループとなる．配列変数 a の添え字に関して解析すると，標準ループのイタレーション（i,j）は 1 つ目のループのイタレーション（i,j）と (i+1,j+1) に依存しているため，各次元において上限 1，下限 0 のイタレーション間依存が存在することがわかる．このような解析を TLG 中でデータを共有する配列全てにおいて行う．

#### 2.2.2 分割数決定

分割後にループ間で共有されるデータサイズがローカルメモリサイズ以下になるように分割数の決定を行う．この時 TLG 内のループの中で最も扱うデータサイズが大きいループのデータサイズを基準とし，それがローカルメモリサイズ以下になるように分割数を決定する．この時，2.2.3 節で述べるように分割後のループが境目となるイタレーションを重複して実行することも考慮してサイズを決定する．基準となるデータサイズとローカルメモリサイズから単純な除算を行い必要な分割数を決定した後，外側ループの分割だけでは不十分な場合に順次内側に分割数を割り振り，各ネストレベルにおける分割数を決定する．

#### 2.2.3 コードコンパクション手法

2.2 節で提案したデータ多次元整合分割においては，ループの分割がコードコピーによって実現される．そのため，同じループボディを持ちループの上下限値のみ分割がなされているループが分割数分生じることになる．特に分割数が大きくなるデータ多次元整合分割においては出力されるコードサイズが元コードのサイズに比べて非常に大きくなってしまう．そのため，本手法においてはループブロッキングを利用したリストラクチャリングを行うことでコードコピーを行うことなく分割コードと等価なコードの生成を行っている．特に今回適用するコードコンパクション手法では TLG 中の複数の多重ループに対して同一ブロックサイズでループブロッキングを適用し，その後ブロッキンググループのフュージョンを行っている．この手法を図 1 の上部ループ群に適用すると図 2 のようになる．図 1 の上部

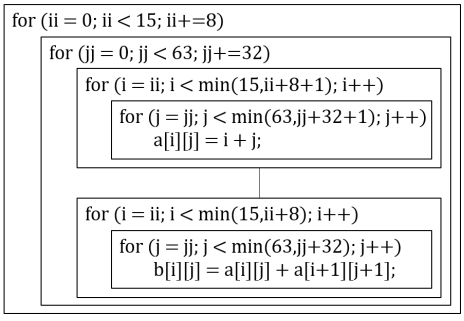


**図 2** コードコンパクション手法適用後のコードイメージ [11]

ループ群においてはそれぞれのループの上下限値が異なっている．本手法はブロッキング後にループフュージョンを行うためフュージョン後のループの上下限値は標準ループのものに統一される．従って，この例においては一つ目のループの i=15 および j=63 のイタレーションにおいてピーリングを行うことで等価なコードの生成を実現している．ピーリング部分のループに関しては図 2 では省略している．今回は図 1 と同様の分割数を想定し各ループネストを 2 分割するため，外側 8, 内側 32 のブロックサイズでループブロッキングを行う．この際，ILD 解析の結果を用いてインナーループの上下限値を調整しデータ転送を最小限におさえるようにしている．特に今回の例では境目となるイタレーションを重複して実行することにより余分なデータ転送の挿入を回避している．

## 3. 提案する階層アジャスタブルブロックを用いたローカルメモリ管理

本章では先に提案した階層アジャスタブルブロックを用いたローカルメモリへのデータ割り当て方法，およびテンプレート配列を用いたローカルメモリ管理手法とその問題点について述べ，これを改良する提案手法を説明する．

### 3.1 アジャスタブルブロック

プログラム中のデータを配置する際に可変長サイズのブロック単位で割り当てを行うとフラグメンテーションを引き起こす可能性がある．しかし，一種類の固定長サイズのブロック単位で割り当てを行うとメモリ利用効率が低下してしまう．そのため，先に提案した手法ではアジャスタブルブロックと呼ばれるアプリケーションに応じてブロックサイズを変えることのできるブロック単位でのメモリ割り当て手法を行っている [13]．アジャスタブルブロックではまずプログラム中で最もローカルメモリを最適に利用できる固定のブロックサイズを選択し，その後そのブロックを 1/2 単位で分割したサブブロックを定義する．プログラム中に存在するサイズの異なる配列をそのサイズに応じてこれらのブロック，サブブロックに割り当てることで効率的なメモリ割り当てを実現している．上記のブロックによるローカルメモリの管理例を図 3 に示す．各ブロックにはレベルとブロック番号が割り振られる．この時最大のブロックサイズを $full\_block\_size$ と呼び，level=0 として定義する．またブロック中の任意のレベル l のブロックに対し，そのサイズを 1/2 に分割したサブブロックをレベル l+1 のブロックとして定義する．図 3 に示すようにレベル l+1 のブロックはレベル l のブロックを 2 つに分ける形で同じ領域のローカルメモリ上にマッピングされる．また各レベルのブロックに対し最もアドレスが低いものをブロック番号 0 として，順にブロック番号を割り振る．このブロックを利用してローカルメモリを管理することでローカルメモリ

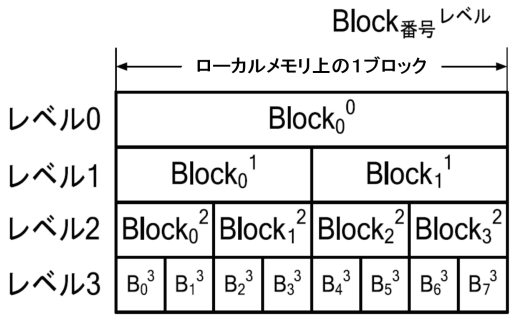


**図 3** アジャスタブルブロックのイメージ図 [11]

上の同じアドレスに対して，プログラム中のデータアクセス状況に応じて様々なサイズの配列を割り当てることが可能になり，フラグメンテーションを抑えたうえで効率的なローカルメモリ管理を行うことが出来る．

### 3.2 従来手法におけるテンプレート配列

ローカルメモリは出力コード上で 1 次元の配列として表現される．しかし特に多次元配列をローカルメモリへ割り当てる際にこの 1 次元配列を介してアクセスをするとその添え字の計算が複雑になり出力コードの可読性が低下してしまう．そこで，アジャスタブルブロックと同じサイズ，次元，型のテンプレート配列と呼ばれる配列を用意し，1 次元配列であるローカルメモリ上に割り当てられた多次元配列をテンプレート配列を介してアクセスすることで，可読性を維持したままアジャスタブルブロックへのアクセスを容易にする．テンプレート配列のイメージを図 4 に示す．アジャスタブルブロックにおける各サブブロックが親

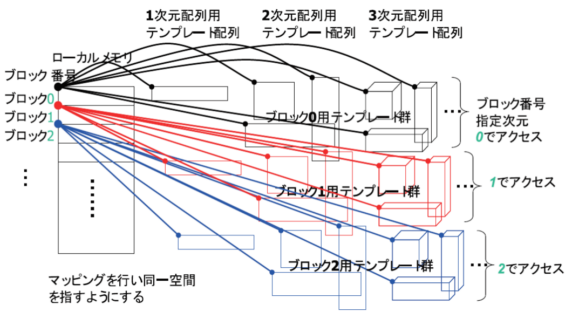


**図 4** テンプレート配列利用イメージ図 [11]

ブロックの 1/2 単位で分割されることを考慮すると，配列の各次元のサイズのそれより大きい 2 の冪乗サイズに補正されたサイズのテンプレート配列に各配列が割り当てられることとなる．そのため，配列の次元 $k$ のサイズを $S_k$ とするとテンプレート配列の各次元のサイズ $T_k$ は次のように定義される．

$$2^n - 1 < S_k \leq 2^n$$

$$T_k = 2^n$$

また，定義されたテンプレート配列の各次元のサイズ $T_k$ と元配列の次元数 $d$，型を $type$，同一レベル上のブロックの個数を $block\_num$ とするとテンプレート配列は以下のような $TemplArray$ と定義される．

$$typeTemplArray[block\_num][T_1][T_2]...[T_d];$$

この時，この配列が配置されるブロックのサイズ $S$ は型のサイズを $type\_size$ とすると以下のように計算できる

$$S = type\_size \prod_{k=1}^{d} T_k$$

そのため，このブロックが配置されるブロックのレベル $l$ は

$$S = \frac{full\_block\_size}{2^l}$$

を満たす l として求まる．

しかし，配列が各次元においてそのサイズより大きい 2 の冪乗サイズのテンプレートに割り当てられる関係上，配列サイズによってはメモリ利用効率が低下する可能性がある．例えば 3 次元の配列 $A[3][5][5]$ はテンプレート配列 $Temp[4][8][8]$ を用いてメモリ上に割り当てられる．この時，配列サイズ $A$ とテンプレート配列サイズ $S$ は以下のように計算される．

$$A = 3 \times 5 \times 5 = 75$$

$$S = 4 \times 8 \times 8 = 256$$

そのため，メモリ上の割り当て領域に対するデータが実際に割り当てられている領域の割合をメモリ利用効率 $U$ と定義すると以下のように計算される．

$$U = \frac{A}{S} \simeq 0.29$$

このように従来のアジャスタブルブロックを利用したメモリ配置では配列のサイズによってメモリ利用効率が大きく低下する可能性がある．

### 3.3 整数分の１分割可能なアジャスタブルブロック

前述の通り従来のアジャスタブルブロックを用いたメモリ配置手法では配列のサイズによってはメモリの利用効率が著しく低下する可能性がある．そのためアプリケーションで使用する配列の種類によってはデータ多次元整合分割を用いてデータサイズを縮小しても必要な配列がローカルメモリに乗り切らない可能性がある．本稿ではこのようなメモリ利用効率の低下を防ぐため，アジャスタブルブロックにおけるデータ分割の単位を整数分の 1 とし，より効率よく配列をローカルメモリに割り当てる手法を提案する．本稿で提案するアジャスタブルブロックを用いたメモリ管理においては，まず 3.3.1 節で述べるようにプログラム中の各配列を割り当てるブロックのサイズを決定する．その後，3.3.2 節で述べるように各配列をローカルメモリ上の空いている領域に割り当てられる様，割り当てる先のブロック番号，レベルを決定していく．

#### 3.3.1 ブロックサイズの決定

従来の手法ではブロック中の全てのサブブロックは親ブロックの 1/2 の単位で分割されていたためローカルメモリに載せる最大サイズの配列が決定した時点ですべてのブロックのサイズを決定することが出来た．提案手法ではまずローカルメモリに載せる最大サイズの配列が決定した際に，そのサブブロックをどの分割数で定義していくか決定するというブロックサイズの決定を行う必要がある．特に，管理対象の配列がすべて倍数関係になっているとは限らないため，以下の手順で各サブブロックのサイズを決定する．

(1) TLG 中のローカルメモリ管理対象配列を総サイズの大きい順にソートし，n=0 から非減少順に番号を割り当てる
(2) 総サイズが最も大きい配列を基準にし，サブブロックサイズを以下の手順で再帰的に決定する
- n+1 番目の配列サイズが n 番目の配列サイズの整数分の 1 である場合，n+1 番目の配列サイズをサブブロックのサイズとして採用し，親ブロックに対しサブブロックが何分割されるかを計算する

この時，n+1 番目の配列が n 番目の配列の整数分の 1 のサイズではない場合には n+1 番目の配列は n 番目の配列と同様のレベルのブロックに配置することとする．図 5 で示す総サイズ順にソート済みの配列定義群を例に以上の手順を説明する．まず，配列 A が最もサイズが大きいため，

```
n=0 : double A[3][5][5];
n=1 : double B[5][5];
n=2 : double C[5][3];
n=3 : double D[5];
```

**図 5** ローカルメモリ管理対象の配列定義群の例

この配列を最大サイズの配列としレベル 0 のブロックサイズ $3 \times 5 \times 5 = 75$ のブロックを定義する．配列 B は配列 A の 1/3 のサイズであるためレベル 1 のサブブロックとして，ブロックサイズ $5 \times 5 = 25$ が定義される．この時，分割数である $25 \div 75 = 1/3$ がレベル 0 とレベル 1 間の倍数関係として保持される．配列 C は配列 B の整数分の 1 の関係になっていないため，配列 C は配列 B と同レベルのブロックとして配置することが決定される．配列 D は配列 B，C のブロックサイズの 1/5 のサイズであるため，レベル 2，ブロックサイズ 5 のサブブロックが定義され，レベル 1 に対するレベル 2 の分割数は $5 \div 25 = 1/5$ として保持される．この例では，レベル 0 のブロックに対しレベル 1 のブロックが 1/3，レベル 1 のブロックに対しレベル 2 のブロックが 1/5 のサイズにそれぞれ分割される．このように各レベルのブロック間の分割数関係を保持することで，1/2 単位で分割していた従来のアジャスタブルブロックと同様のメモリ管理を行うことが出来るように，ブロックサイズを決定する．

#### 3.3.2 ブロックへの割り当て

各配列を配置するブロックサイズが決定されたのち，各ループが実行される順番にそれぞれの配列を割り当てるブロック番号を決定していく．そのブロック番号を利用して配列を各ブロック上のテンプレート配列へ割り当てることでローカルメモリへの配置を実現する．図 5 で示したすべての配列がローカルメモリ上へ割り当てられている際のアジャスタブルブロックへの配列の割り当てイメージを図 6 に示す．実際には TLG 中のループの配列使用状況をもと


**図 6** 図 5 における配列の整数分の 1 アジャスタブルブロックへの割り当て

にメモリ配置決定し，必要となる共有メモリとローカルメモリ間の DMA コントローラ（DTU）を用いたデータ転送命令を挿入する事で動的なローカルメモリ管理を実現する．

## 4. 性能評価

本章では階層アジャスタブルブロックを用いたローカルメモリ管理手法を OSCAR マルチグレイン自動並列化コンパイラに実装し，情報家電用マルチコア RP2 上での性能評価を行った結果について述べる．

### 4.1 評価に用いるマルチコアプロセッサ RP2

RP2 は NEDO 半導体アプリケーションチップ「リアルタイム情報家電用マルチコア」プロジェクトにおいてルネサステクノロジー，日立製作所，早稲田大学によって開発がなされてた，OSCAR マルチコアアーキテクチャを持つコンパイラ協調型マルチコアである．RP2 のアーキテクチャ図を図 7 に示し，メモリに関連する仕様を表 1 に示す．図 7 に示すように RP2 は SH4A プロセッサコアを 8 コア

**表 1** RP2 仕様

| | |
|---|---|
| LDM レイテンシ | 1 クロック |
| LDM サイズ | 32KB |
| DSM レイテンシ | 2 クロック |
| DSM サイズ | 64KB |
| オフチップ CSM レイテンシ | 約 55 クロック |

集積しており，各プロセッサコアはローカルデータメモリ

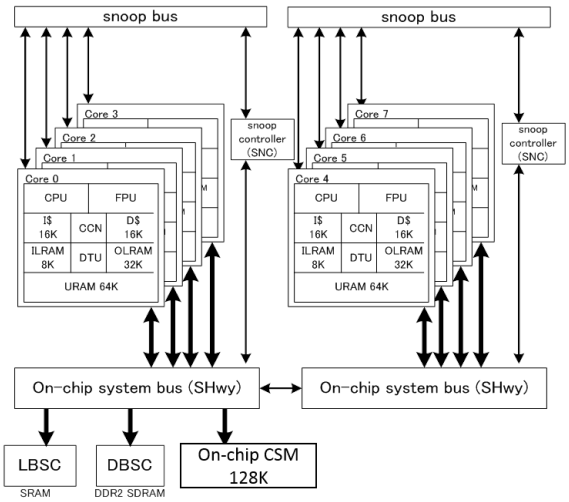


**図 7** RP2 のアーキテクチャ図

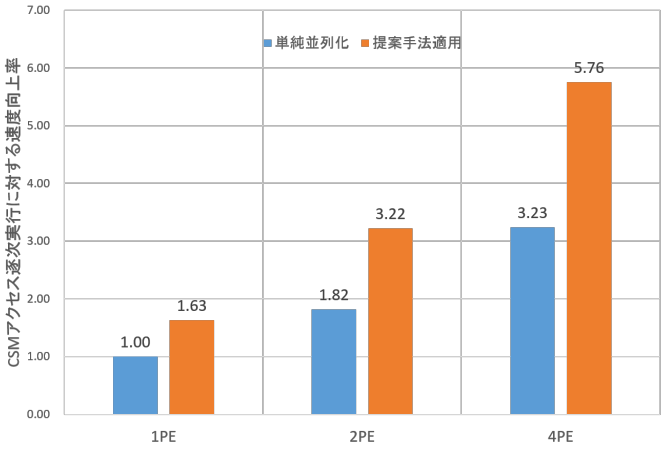


**図 8** BT における CSM アクセス単純並列化時と提案手法適用時の性能比較結果

(LDM) と分散共有メモリ (DSM) といったグローバルアドレス空間にマップされたメモリを持っているが，今回はより小さいメモリ領域を利用した評価を行うためにデータは LDM に置き，同期で用いる共有変数の利用のみ DSM を使用している．データ転送を行う DTU，そしてコア間で共有されるオフチップの集中共有メモリ (CSM) を持っている．

### 4.2 評価プログラム

評価対象プログラムとして，Nas Parallel Benchmarks のプログラム BT を利用した．評価に際してはオリジナルのソースコードに対して本手法の適用が容易になるようにループディストリビューションとループ内の関数のインライン展開をあらかじめ手動で施し，さらに評価時間短縮のため通常 200 回繰り返し演算を行うものを短縮して用いている．BT は最大で 6 次元もの要素を持つ単精度浮動小数点型の配列群に対する演算であり，従来のアジャスタブルブロックを用いたメモリ割り当てでは必要なすべての配列がローカルメモリに乗り切らない形となっている．

### 4.3 BT における性能評価結果

BT に対する評価結果を図 8 に示す．

横軸はプロセッサ数を示し，縦軸はオフチップ CSM にデータを割り当てる従来の逐次実行に対する速度向上率を示す．CSM 上にデータを配置した従来の逐次実行に対し，提案手法を適用した逐次実行では 1.63 倍の速度向上が得られた．また，従来の並列化では 4PE 時の速度向上が 3.23 倍にとどまるのに対し，提案手法では 4PE 時の速度向上が 5.76 倍に達し，同じ 4PE では従来の共有メモリにデータを配置する方式に比べ約 1.78 倍の速度向上率を達成している．

## 5. まとめ

本稿では，マルチコア上でローカルメモリを有効活用するために必要な，データ多次元を用いたローカルメモリ管理手法において階層アジャスタブルブロックを用いてより効率的にローカルメモリ上の領域を利用する手法を提案した．本手法ではローカルメモリ上の領域を整数分の 1 に分割した階層的な固定サイズの領域にデータを割り当てることで，サイズ・次元の異なるブロックをより効率よくローカルメモリに配置することを可能にした．本手法を各プロセッサコアが 32KB のローカルメモリデータを搭載した SH4A を 8 個集積したマルチコアプロセッサである RP2 上で評価を行ったところ，ローカルメモリを利用しない従来方式の 1PE の場合に対して Nas Parallel Benchmarks BT で 4PE で 5.76 倍の速度向上が得られることが確かめられた．

### 参考文献

[1] Avissar, O., Barua, R. and Stewart, D.: An Optimal Memory Allocation Scheme for Scratch-pad-based Embedded Systems, *ACM Trans. Embed. Comput. Syst.*, Vol. 1, No. 1, pp. 6–26 (2002).

[2] Panda, P. R., Nicolau, A. and Dutt, N.: *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*, Kluwer Academic Publishers, Norwell, MA, USA (1998).

[3] Udayakumaran, S., Dominguez, A. and Barua, R.: Dynamic Allocation for Scratch-pad Memory Using Compile-time Decisions, *ACM Trans. Embed. Comput. Syst.*, Vol. 5, No. 2, pp. 472–511 (2006).

[4] Li, L., Nguyen, Q. H. and Xue, J.: Scratchpad allocation for data aggregates in superperfect graphs, *Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'07), San Diego, California, USA, June 13-15, 2007*, pp. 207–216 (2007).

[5] Kandemir, M. and Choudhary, A.: Compiler-directed Scratch Pad Memory Hierarchy Design and Manage-

ment, *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, New York, NY, USA, ACM, pp. 628–633 (2002).
[6] Issenin, I., Brockmeyer, E., Durinck, B. and Dutt, N.: Multiprocessor System-on-chip Data Reuse Analysis for Exploring Customized Memory Hierarchies, *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, New York, NY, USA, ACM, pp. 49–52 (2006).
[7] 桃園　拓，中野啓史，間瀬正啓，木村啓二，笠原博徳：マルチコアのためのコンパイラにおけるローカルメモリ管理手法，研究報告組込みシステム（EMB），Vol. 2009, No. 1, pp. 69–74 (2009).
[8] 中野啓史，桃園　拓，間瀬正啓，木村啓二，笠原博徳：マルチコアプロセッサ上での粗粒度タスク並列処理のためのコンパイラによるローカルメモリ管理手法，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 2, No. 2, pp. 63–74 (2009).
[9] 吉田明正，前田誠司，尾形　航，笠原博徳：Fortran マクロデータフロー処理における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. 35, No. 9, pp. 1848–1860 (1994).
[10] 吉田明正，越塚健一，岡本雅巳，笠原博徳：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).
[11] 山本康平，白川智也，吉田明正，木村啓二，笠原博徳：データ多次元整合分割によるマルチコア・ローカルメモリ管理手法，研究報告システム・アーキテクチャ（ARC），No. 10 (2016).
[12] Kouhei Yamamoto, Tomoya Shirakawa, Y. O. A. Y. K. K. a. H. K.: Automatic Local Memory Management for Multicores Having Global Address Space, *Languages and Compilers for Parallel Computing*, pp. 282–296 (2016).
[13] 笠原博徳，木村啓二，中野啓史，仁藤拓実，丸山貴紀，三浦　剛，田川友博：メモリ管理方法、情報処理装置、プログラムの作成方法及びプログラム，日本国特許第 5224498 号 (2007).