Vol.2016-ARC-218 No.10
Vol.2016-SLDM174 No.10
2016/1/19

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

山本 康平[1]　白川 智也[1]　吉田 明正[1,2]　木村 啓二[1]　笠原 博徳[1]

概要：自動車制御のようなリアルタイム制御システムの開発では，デッドラインを厳密に守るために各コアがローカルメモリを持つマルチコアの利用が必要となる．容量制限のあるローカルメモリ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ことから，コンパイラによって自動的にローカルメモリの管理を行う手法の開発が望まれている．従来の並列化コンパイラではローカルメモリの自動管理ができるものはなかった．各コアのローカルメモリにデータを転送し，そのデータを可能な限り再利用して高速化を目指すことが効率的な並列化では重要である．このためにはプログラムで使用される配列データをローカルメモリに載るサイズに分割し，プログラム各部でのワーキングセットがローカルメモリに配置され，さらにローカルメモリ上のデータを複数のループ等で再利用するという処理が望まれる．このデータの分割においては多次元配列の場合最外側のループを分割する方法が考えられるが，ローカルメモリサイズが小さい場合さらに内側のネストループにわたって分割することでよりデータサイズを縮小することが必要となる．本稿ではこのようなデータ多次元整合分割と分割したデータをローカルメモリの領域に割り当てる方式，さらに分割数が増大した時に発生しうるコードサイズ増大の問題を解消するコードコンパクションのためのストリップマイニング手法を提案する．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，SPEC95swim において逐次実行に比べて 8PE 時に 11.3 倍の性能向上を得ることに成功した．

1. はじめに

情報家電や自動車等の組み込み分野では低消費電力，高性能化を目的としてマルチコアプロセッサが利用されており，特にハードリアルタイム制約のためのマルチコアでは各コアがローカルメモリを持つアーキテクチャが望まれている．キャッシュメモリではミスヒット時のメモリアクセスペナルティが大きく，デッドライン管理を厳密に守ることが難しいので，ローカルメモリを利用することによりメモリアクセスデータ転送を明示的に行ってハードリアルタイム制約を満たすことが重要になる．また，高性能マシンの分野においては高性能化と低消費電力化の両立のためにベクトルアクセラレータの有効利用が期待されている．そのためには，DMA コントローラを用いてローカルメモリにデータをオーバラップ転送し，一度ローカルメモリに配置したデータをアクセラレータで繰り返し利用することが重要となる．しかし，マルチコアプロセッサ上で限られたサイズのローカルメモリを有効に利用するためには，配置するデータの選択，ローカルメモリにフィットするサイズへのデータの分割，共有メモリとのデータの転送といった複雑なローカルメモリ管理が必要となる．このようなローカルメモリ管理を手動で行うのは非常に困難であり，ローカルメモリを利用しつつソフトウェアの生産性を向上させるためにコンパイラによるローカルメモリ管理の実現が望まれる．

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割り当て [1], [2] があるが，これはプログラム開始時にローカルメモリに割り当てたデータを実行終了時まで保持し続けるものであるため，ローカルメモリに配置できるデータはローカルメモリサイズ以下に限られてしまう．そこで，プログラム中のデータアクセス状況に応じてローカルメモリに割り当てるデータを変更する動的な手法が提案されている．動的なローカルメモリへのデータの割り当て手法では単一プロセッサにおいてループやサブルーチンといった粗粒度タスク間でデータのローカルメモリ配置を変更する手法 [3], [4] が提案されている．マルチコアプロセッサを対象としたローカルメモリを有効利用する方法としてはループを対象としたデータリユースの解析法 [5], [6] が提案されているが，粗粒度タスク間の並列性とデータローカ

[1] 早稲田大学
Waseda University.
[2] 明治大学
Meiji University.

リティを最適化するマルチコアローカルメモリの管理技術は開発されていない．筆者らはプログラム全域の並列性を抽出しつつデータローカリティを最適化できるよう，データをローカルメモリサイズに応じて分割，アクセス状況に応じてローカルメモリや共有メモリにデータを転送するマルチコアのための動的なローカルメモリ管理手法 [7], [8] を提案している．このローカルメモリ管理手法においてはループ整合分割 [9], [10] によって複数ループ間のデータローカリティを抽出することを前提にしているが，このループ整合分割は最外側ループのみを分割の対象としているため十分な分割が行えずにデータサイズを縮小しきれずローカルメモリへの配置が不可能な場合が存在した．本稿では上記マルチコアのための動的なローカルメモリ管理手法におけるループ整合分割を改良し，データサイズをより小さなサイズまで分割可能にするデータ多次元整合分割を用いたローカルメモリ管理手法を提案する．

さらに提案手法を 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. 提案するデータ多次元整合分割によるローカルメモリ管理手法

本稿で提案するデータて次元整合分割によるローカルメモリ管理手法ではまず 2.1 節で述べるように入力ソースプログラムを粗粒度タスクへ分割して最早実行可能条件解析を用いてタスク間の並列性を抽出する．次に 2.3.1 節で述べるように，対象となる粗粒度タスクの集合でアクセスされるデータを，ローカルメモリにフィットしてタスク間で再利用できるようにデータ多次元整合分割を行い，分割されたデータ（当該タスク集合のワーキングセット）をローカルメモリ上のブロックにテンプレート配列を用いて配置しスタティックスケジューリングする．最後に，DMA コントローラを用いたデータ転送命令の挿入を行う．

2.1 粗粒度タスク並列処理

粗粒度並列タスク処理で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）に分割し，SB や RB の内部に粗粒度タスク並列性が存在する場合にはさらに MT へと階層的に分割する．MT 生成後，MT 間のコントロールフローとデータ依存関係を解析してマクロフローグラフ（MFG）を生成し，MFG に最早実行可能条件解析を適用して MT 間の並列性を抽出し，マクロタスクグラフ（MTG）を生成する．

2.2 ループ整合分割

容量の限られたローカルメモリへデータを効率的に配置するためにはタスク集合がアクセスするデータサイズをローカルメモリに載るサイズに分割する必要がある．ループ整合分割では効率的なデータ配置とリプレース管理を行う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 [9], [10]．まず，MTG 上で同一の配列にアクセスするデータ依存のあるループをグループ化し，こうして集められたループ群をターゲットループグループ（TLG）と呼ぶ．次に，TLG 中で最も推定コストの大きいループを標準ループとし，標準ループのあるイタレーションを基準とした他のループのイタレーションとの依存を解析するループ間データ依存（ILD）解析を行う．ローカルメモリのサイズとタスク集合中の配列サイズから必要な分割数を決定し，ILD 解析の結果とあわせて適切にループを分割する．従来提案のループ整合分割は最外側ループのみを分割対象としており，最外側ループの分割のみで配列サイズを充分に分割できない場合が存在したため，本稿ではこれを多重ループに拡張子，多次元配列の複数次元にわたって分割するデータ多次元整合分割を提案する．

2.3 提案するデータ多次元分割手法

前述のとおり従来のループ整合分割手法では多重ループにおいてその最外側ループのみを分割している．内側ループの回転数が多い多重ループにおいては最外側の分割のみでは十分なデータサイズ縮小とならず，配列データのローカルメモリへの配置が不可能な場合が存在する．配列データ側からいえば，2 以上の次元を持つ配列において 1 次元目のみのデータ分割では不十分ということである．本稿ではこのような場合でもローカルメモリ配置を可能にするため，多重ループの複数のネストレベルにわたって，すなわち多次元配列データの複数次元にわたって分割することにより更なるデータサイズ縮小を図るデータ多次元整合分割を提案する．

2.3.1 データ多次元整合分割手法

従来のループ整合分割手法では図 1 に示すようにネストされたループの最外側ループのみを対象としてループ分割を行っている．図 1 の場合，外側ループを最大限分割した

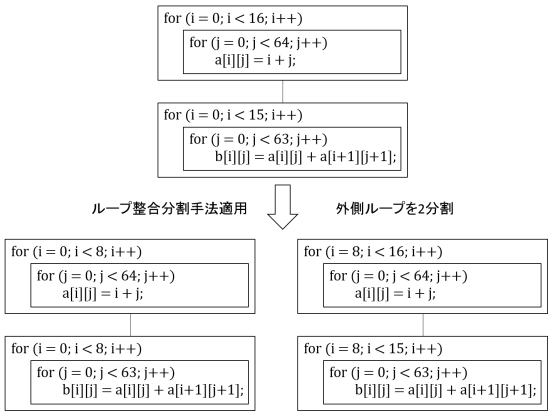

図 1 従来手法適用のコードイメージ

としても内側の 64 回転ループでアクセスされる配列データが 2 ループの間で共有されるデータとなる．この配列データサイズがローカルメモリサイズを超過する場合には従来手法においてはローカルメモリへの配置を諦めるほかなかった．そこで，図 2 のように内側のループまで分割することを考える．これは外側，内側ループをそれぞれ 2 分割した例である．複数ネストレベルにわたってループを分

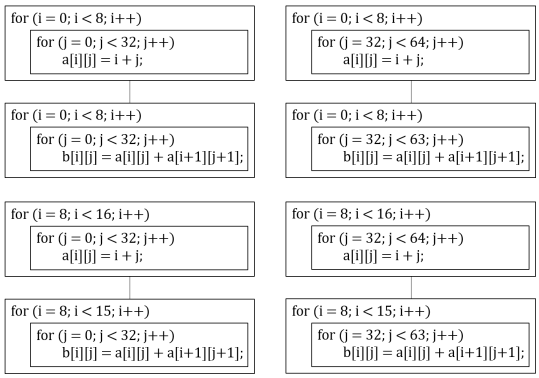

図 2 提案手法適用後のコードイメージ

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有するプログラムにおいても限られたローカルメモリを最大限利用することが可能になる．今回提案するデータ多次元整合分割では完全ネストの多重ループのみを対象としている．

2.3.2 TLG 生成

まず対象となるループ群を集めてターゲットループグループ（TLG）を生成する．提案手法を適用可能な TLG は以下の条件を満たさなければならない．

- TLG に含まれるループの完全ネストレベルが 2 以上である
- ループの誘導変数が配列添字に現れる次元がループネスト順と対応している

図 1 上部に示した手法適用前の 2 つのループが TLG に相当する．

2.3.3 ILD 解析

続いて 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．ILD 解析では TLG 中で最も推定コストの大きいループを標準ループとして，標準ループのイタレーションが依存する他ループのイタレーションを特定する．ILD 解析を行っておくことにより，ループの分割やローカルメモリ管理に際するローカリティの考慮が容易になる．提案手法では各次元において依存するイタレーションの上限値と下限値を算出し，これを分割時やローカルメモリ管理時に利用する．図 1 に示した TLG においては 2 つ目のループが標準ループとなる．配列変数 a の添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，i,j それぞれの次元に関して下限 0，上限 1 のイタレーション間依存があることがわかる．こういった解析が TLG 中でデータを共有する配列変数全てに対して行われる．

2.3.4 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は多くともローカルメモリサイズ以下になる必要がある．理想的にはローカルメモリに同時に存在すべきデータサイズを考慮して分割数を決定すべきであるが，今回は解析と分割数決定のアルゴリズムが複雑化することを避け，TLG の中の全ての配列が同時にローカルメモリに存在することが可能なサイズとなるように分割数を決定することで，確実にローカルメモリ配置が可能であることを保証するという方針を用いる．TLG 中の総配列サイズとローカルメモリサイズから単純な除算により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は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．

2.4 スケジューリング

分割後の各ループはそのイタレーション範囲が同一のものを同一 PE で連続実行する必要がある．各次元のイタレーション範囲の組み合わせに応じて分割後ループにタグ番号を割り振り，同一のタグ番号を持つループ群をデータローカライザブルグループ（DLG）として同一 PE 上で連続実行するようにスケジューリングすることで，多次元に渡った分割においてもデータローカライゼーションが実現される．

2.5 ローカルメモリ管理

ここでは，プログラムに合わせてメモリ管理・データ転送用のブロックサイズを変えることができるアジャスタブルブロック，このブロックに 1 次元から n 次元の配列を割り当てるためのテンプレート配列に関して説明し，最後にデータ転送の挿入について述べる．

2.5.1 アジャスタブルブロック

ローカルメモリの管理対象となる配列は，アプリケーションによって様々である．多様な次元やサイズを持つ配列を可変サイズブロックを用いてローカルメモリ上の空き領域に割り当てると，フラグメンテーションが生じて効率的な利用ができなくなってしまう．当手法では図 3 に示すようにローカルメモリをアジャスタブルブロック [11] と呼ぶ固定サイズの領域に区切って扱う．アプリケーションに応じて適切なブロックの固定サイズを選択し，さらにブロックの整数分の 1 のサイズをもつサブブロックに分割できるようにする．このように階層的なブロックを定義することにより，サイズ・次元の異なるブロックを効率よくローカルメモリに配置することが可能になる．今回の実装ではブロックサイズを 2 の冪乗に限定し，サブブロックの

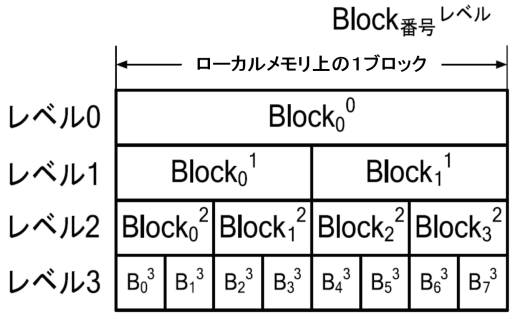

図 3　アジャスタブルブロックのイメージ図

サイズは上位のブロックに対して 1/2 のサイズに限定することにより簡略化を図っている．

2.5.2　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が，多次元配列をローカルメモリに配置した際にこの 1 次元配列を介してアクセスすると添字の計算が複雑となるため出力コードの可読性が低下する．そこで，テンプレート配列 [11] と呼ばれる元配列と同じ次元・型の配列を用意し，1 次元のブロック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．各

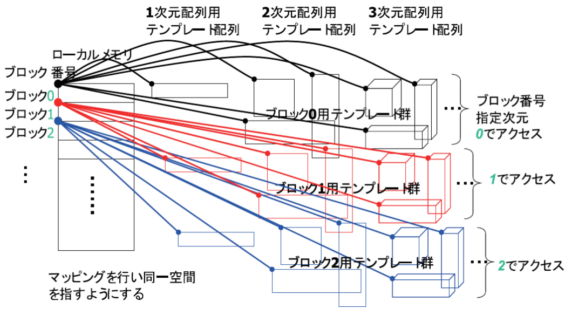

図 4　テンプレート配列利用イメージ図

ブロックには図 4 に示すように次元の異なる複数のテンプレートをマッピングしておく．ローカルメモリ管理対象になる分割された部分配列は，それと同じ次元数のテンプレートに割り当てられる．各テンプレートにはブロック番号を指定するための次元を新たに用意し，この次元を用いることでローカルメモリ中の何番のブロックを利用するかを指定できる．

2.5.3　ブロックの掃き出し管理

新たにデータをローカルメモリに配置しようとする際，空いているブロックが存在しない場合はローカルメモリから追い出しても影響の少ないブロック上のデータを共有メモリに掃き出してから使用する必要がある．データ転送を最小限に抑えローカルメモリを効率的に利用するためには，再利用されるデータをローカルメモリ上に残す必要がある．本ローカルメモリ管理手法においては以下の様に掃き出しの優先度を定義することによってブロックの掃き出しの効率化を図っている．

(1) 死んでいる変数（それ以降アクセスされない変数）
(2) 他プロセッサでのみアクセスされる変数
(3) 再度自プロセッサでアクセスされるが直後ではない
(4) 直後に自プロセッサでアクセスされる

2.5.4　データ転送の挿入

第 2.4 節でのスケジューリングの結果を利用して，アジャスタブルブロックを用いたローカルメモリ配置を決定し，最後に必要となる共有メモリとローカルメモリ間のデータ転送命令を挿入する．データ転送ユニット (自律型 DMA コントローラ) は CPU と非同期にバースト転送が可能なものを想定しているが，今回はデータ転送が必要となる MT に対して MT の直前にロードの転送，MT の直後にストアの転送を挿入するものとし，オーバラップ転送は実装を容易にするために実装していない．このオーバラップ転送は今後実現していく計画である．

2.6　コードコンパクション手法

従来手法及び第 2.3.1 節で提案したデータ多次元整合手法において，ループの分割はコードコピーによって実現されているため，ループボディ部分が同一で上下限値が異なるループが分割数に応じて生成されることになる．出力コード上においても分割された分だけ生成されたループが記述されるため，特に分割数の多くなるデータて時限制号分割手法では出力コードサイズが元コードのサイズに対して非常に大きくなってしまう．この問題を回避するためにはコードコピーによらないでループ分割と同等の効果を得られる処理が必要であり，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によってこれを実現した．この手法を用いる場合はマクロタスクとしてループが分割されるわけではないため，最外側のブロッキングループにおいて中粒度並列化を行えば第 2.4 節で示したようなスケジューリングを行わずとも自ずとローカリティを考慮した PE 割り当て・実行順が保証できる．なお，本手法においては TLG のループをまとめてブロッキングループで包むような構造になる関係上，各ループと依存関係のある BB がループの間に存在する場合は適用することができない．

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，さらにブロッキングループでフュージョンを行う．この手法を図 1 のコードに対して適用すると図 5 のようになる．図 1 における TLG では 1 つ目のループと 2 つ目のループでイタレーション範囲が異なるため，範囲の狭い 2 つ目のループに合わせて 1 つ目のループの i=15 及び j=63 のイタレーションのピーリングを行う．これは，ブロッキング後のループフュージョンを可能にす

```
for (ii = 0; ii < 15; ii+=8)
  for (jj = 0; jj < 63; jj+=32)
    for (i = ii; i < min(15,ii+8+1); i++)
      for (j = jj; j < min(63,jj+32+1); j++)
        a[i][j] = i + j;
    for (i = ii; i < min(15,ii+8); i++)
      for (j = jj; j < min(63,jj+32); j++)
        b[i][j] = a[i][j] + a[i+1][j+1];
```

図 5　コードコンパクション手法適用後のコードイメージ

るために必要な処理である．ピーリング部分に関しては図 5 では割愛している．続いて決められた分割数となるように各ループにおいてブロッキングを行う．今回は図 2 と同様の分割数を想定し，各ループネストを 2 分割することを考えると，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えば良い．このとき，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することにより，従来のループ整合分割と同様にデータ転送を最小限に抑えることができる．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データ転送を回避している．また，インナーループの上下限値が元のループの上下限値を超過しないように調整している．

3. 性能評価

本章ではデータ多次元整合分割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．

3.1 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，日立製作所，早稲田大学により開発された．RP2 は 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り，SH4A プロセッサコアを 8 コア集積した構成となっている．RP2 のアーキテクチャ図を図 6 に示し，メモリに関連する仕様を表 1 に示す．図 6 に示すように各プロセッサコアはローカルデータメモリ（LDM）と分散共有メモリ（DSM），データ転送ユニット（DTU）を持ち，コア間で共有されるオフチップの集中共有メモリ（CSM）を持つ．なお，今回の評価においては DSM は使用されていない．

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

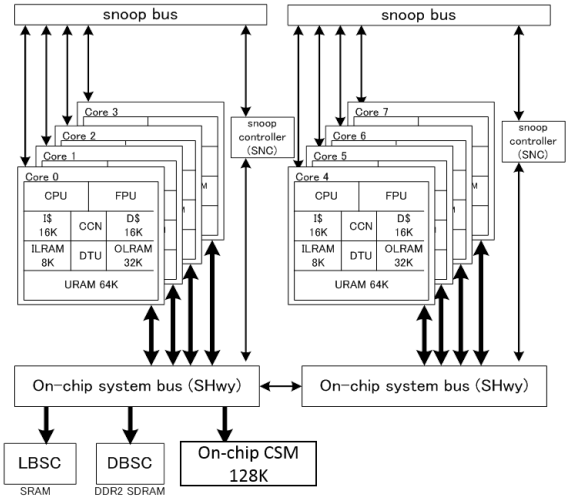

図 6　RP2 のアーキテクチャ図

3.2 評価プログラム

図 1 上部に示した方式説明用プログラムと SPEC95 102.swim を用いて本手法の性能評価を行った．方式説明用プログラムに関しては従来手法と提案手法の比較が容易になるように，表 1 に示す RP2 の仕様を考慮して外側ループを 128 回転，内側ループを 8192 回転とした．

swim に関してはオリジナルのソースコードを C 言語で書き直したうえで，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ピーリングを予め施したものを用いた．また，評価時間短縮の都合上通常 900 回演算を繰り返すものを短縮して用いている．swim は各次元 512 要素をもつ単精度浮動小数点型の 2 次元配列群に対する演算であり，外側ループのみではデータサイズを充分に分割することができないものとなっている．

3.3 方式説明用プログラムにおける性能評価結果

方式説明用プログラムにおける本手法の性能評価結果を図 7 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

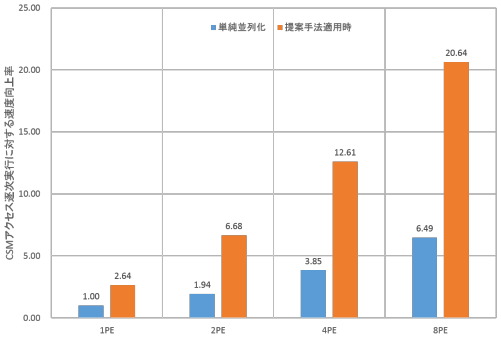

図 7　方式説明用プログラムにおける CSM アクセス単純並列化時と提案手法適用時の性能比較結果

る速度向上率を示す．CSM 上にデータを配置した従来の並列化では 8PE 時の速度向上が 6.49 倍にとどまるのに対し，提案手法では 8PE 時の速度向上が 20.6 倍に達し，同じ 8PE 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 3.18 倍の速度向上率を達成している．提案手法では全ての配列をローカルメモリに配置することが可能になるため，CSM と LDM のレイテンシの差から図 7 に示すような大幅な性能向上が得られたと考えられる．

3.4 swim における性能評価結果

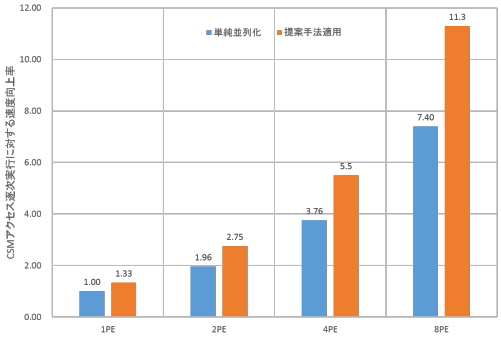

図 8 swim における CSM アクセス単純並列化時と提案手法適用時の性能比較結果

swim の評価結果を図 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．図 8 より従来の逐次実行に比べて 8PE 時に 11.3 倍の速度向上が得られたことがわかる．今回の実装では第 1 ステップの評価として DTU を用いたデータ転送を安全サイドに倒して実現しているため本来不要な冗長なデータ転送が入ってしまっている．CPU と非同期にバースト転送が可能な DTU を考慮してプレロードやポストストアといったオーバラップ転送を実装していないため今後これらの実現によりさらなる速度向上が得られると考えられる．

4. まとめ

本稿では，マルチコア上で高速小容量なローカルメモリを誰でも簡単に有効利用することを可能とする自動ローカルメモリ管理の手法において，処理データサイズの大きなプログラムに対しても有効な多次元データの分割手法を提案した．本手法では多次元配列を用いる演算を小容量のローカルメモリ上で実行するためにデータを多次元にわたって分割し，分割データの再利用を最適化するようにローカルメモリ上にアジャスタブルブロック及びテンプレート配列を用いて割り当て，DTU を用いたオフチップ共有メモリとローカルメモリ間のデータ転送を自動生成する．また，分割によるプログラムサイズの増大を回避するためストリップマイニングを利用したコードコンパクション手法を用いている．本手法を，各プロセッサコアが 32KB のローカルデータメモリを搭載した 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 1 コア場合に対して，SPEC95swim で 11.3 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 Management, *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] 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).