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

桃園 拓† 中野 啓史† 間瀬 正啓† 木村 啓二† 笠原 博徳†

† 早稲田大学 理工学術院 基幹理工学部 情報理工学科 〒169-8555 東京都新宿区大久保 3-4-1

あらまし 従来容量制限のあるローカルメモリ利用の最適化はプログラマにより手動で行われており，これは長時間を要する非常に困難な作業であった．そこで，本稿ではマルチコア上で，プロセッサに近接した高速小容量のローカルメモリを自動並列化コンパイラにより自動的に有効活用する手法を提案する．本手法では，データローカリティと並列性を考慮してループ整合分割とタスクスケジューリングを行った後，スケジューリング結果を利用してローカルメモリ上のデータを長時間に渡り再利用できるようにデータ配置，DMA コントローラを用いたリプレースを行う．本自動ローカルメモリ管理手法の性能評価を，32KB のローカルデータメモリと 64KB の分散共有メモリを搭載した SH4A を 8 コア集積した情報家電用マルチコアである RP2 上で行ったところ，逐次実行に比べ，8PE 時に MPEG2 エンコーダで約 6.20 倍，AAC エンコーダで約 7.25 倍，MiBench susan で約 7.64 倍の速度向上を自動で得ることに成功した．

キーワード マルチコア，ローカルメモリ，自動並列化コンパイラ，ローカルメモリ管理

Local Memory Management Scheme by a Compiler for Multicore Processor

Taku MOMOZONO†, Hirofumi NAKANO†, Masayoshi MASE†, Keiji KIMURA†, and Hironori KASAHARA†

† Department of Computer Science, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

Abstract This paper proposes a local memory management scheme for an automatic parallelizing compiler to realize effective use of a limited size of local memory. After the loop aligned decomposition and task scheduling considering data locality and parallelism, the compiler allocates data to the local memory effectively using the task scheduling result. This paper evaluates the proposed scheme on RP2 multicore for consumer electronics which has 8 SH4A processor cores. Each core integrates 32KB of local data memory and 64KB of distributed shared memory. As the results, the proposed scheme using 8 processors gives us about 6.20 times speedup for MPEG2 encoding program, 7.25 times speedup for AAC encoding program and 7.64 times speedup for susan against the sequential execution.

Key words Multicore, Local Memory, Automatic Parallelizing Compiler, Local Memory Management

1. はじめに

情報家電，ロボット，自動車等の組み込み分野では低消費電力，高性能，リアルタイム処理のためにローカルメモリを搭載したマルチコアが研究開発されている．

各プロセッサコア上のローカルメモリを利用するためには，ローカルメモリに配置するデータの選択や分割，プログラムのデータアクセス状況に応じたメモリ配置，及びオフチップメモリとローカルメモリ間のデータ転送等のローカルメモリの管理が必要となる．このローカルメモリの利用は，キャッシュミスヒット等の実行時不確定性によるデッドライン管理の難しさを避け，ハードリアルタイム制約を満たすことを可能とする．しかしながらプログラマは，ローカルメモリサイズを考慮しつつ，データローカリティとデータ転送の最適化を行う前述のローカルメモリ管理という困難な作業を手動で行う必要があった．このようなローカルメモリ管理を行いつつ，ソフトウェアの生産性を向上させるために，コンパイラによるマルチコアでのローカルメモリ管理の実現が望まれている．

コンパイラによる初期のローカルメモリ管理手法として，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，実行終了まで同じアドレス上に保持する静的なローカルメモリ管理 [1] [2] がある．しかし，小容量のローカルメモリを有効利用するためには，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上に割り当てるデータを更新する動的なローカ

ルメモリ管理が必要である．動的なローカルメモリ管理手法として，ループやサブルーチンなどの粗粒度タスク間でデータをローカルメモリに割り当てる手法 [3] [4] が提案されているが，いずれも単一プロセッサを対象としていた．マルチプロセッサを対象としたローカルメモリ管理手法としては，ループを対象としてデータを動的にローカルメモリに割り当てる手法 [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リティを利用することができない．

そこで，本稿では，コンパイラが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を考え整合分割，配置，リプレース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，NEDO「リアルタイム情報家電用マルチコア技術の研究開発」プロジェクトにおいて，日立，ルネサステクノロジ，早稲田大学で開発された「OSCAR マルチコアメモリアーキテクチャ」に準拠したマルチコアである RP2 上で評価したので，その結果についても報告する．

本稿の構成を以下に示す．第 2 章で提案するローカルメモリ管理手法について述べる．第 3 章で本手法の性能評価結果について述べる．第 4 章で本稿のまとめを述べる．

2. 提案するローカルメモリ管理手法

本稿で提案するローカルメモリ管理手法は，C あるいは Fortran プログラムを入力ファイルとし，OSCAR API [7] によりメモリ配置，データ転送を含めて並列化した C あるいは Fortran プログラムファイルを出力するコンパイレーション手法である．

本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ループやサブルーチンコール等の粗粒度タスクに分割し，これらの粗粒度タスク間の並列性を最早実行可能条件の形で抽出する．次に，2.2.1 節で述べるように粗粒度タスクをタスク間のデータローカリティと並列性を考慮してループの整合分割 [8] を行い，2.2.2 節で述べるように分割されたタスクをプロセッサ間のデータ転送を含む実行時間が最小となるようにプロセッサにスケジューリングする．これらのデータローカライゼーション技術を適用後，2.3 節で提案する，各粗粒度タスクのデータアクセス状況を考慮した動的なローカルメモリ管理を適用する．

2.1 粗粒度タスク並列処理

粗粒度タスク並列処理とは，ソースプログラムを擬似代入文ブロック（BPA），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．SB や RB の内部に粗粒度タスク並列性が存在する場合は，その内部をさらに MT に分割し，階層的に MT を生成する．

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，マクロフローグラフ（MFG）を生成する．さらに，MFG に最早実行可能条件解析を適用して MT 間の並列性を解析し，マクロタスクグラフ（MTG）を生成する．MT 構造が階層的に生成されていた場合，MFG と MTG も階層的に生成される．

粗粒度タスク並列処理では，MT を複数のプロセッサエレメント（PE）から構成されるプロセッサグループ（PG）に割り当てて実行する．PG に MT を割り当てるタスクスケジューリング手法としてコンパイル時に割り当てを決めるスタティックスケジューリングと，実行時に割り当てを決めるダイナミックスケジューリングがあり，MTG の形状及び実行時不確定性を元に決定される．本手法は，スタティックスケジューリングを行うことができる MTG を適用対象とする．

2.2 データローカリティと並列性を考慮したループ分割とタスクスケジューリング

効率的なローカルメモリへのデータ配置，リプレース管理を行うために，データローカリティを考慮したループ整合分割とタスクスケジューリングを行う．

2.2.1 ループ整合分割

ループ整合分割では，粗粒度タスク間のデータローカリティと並列性を考慮しつつ，タスク集合がアクセスするデータサイズがローカルメモリサイズ以下に収まるようにタスクとデータを分割する手法である．ループ整合分割では，まず，MTG 上で同一の配列にアクセスし，データ依存関係のあるループ（MT）を探す．その際，MT 同士が異なるサブルーチンに存在した場合は，必要に応じインライン展開を行い，それらのループ同士を同一の階層に展開した後にグループ化する．グループ化されたループ群をターゲットループグループ（Target Loop Group: TLG）と呼ぶ．その後，TLG を分割し，分割された部分ループでアクセスされる部分配列をローカルメモリサイズ以下に収める．

この時，本手法では部分配列をブロックへ割り当てることを考慮し，部分配列のサイズがなるべく等しくなるように分割を行い，全ての部分配列をローカルメモリへ割り当てる際に必要となる総ブロックサイズがローカルメモリサイズ以下に収まるようにする．

このループ整合分割により，実行時に必要とされるデータサイズを限られたローカルメモリサイズ以下に抑えた上で，プログラム全域のデータローカリティを有効活用する．

2.2.2 データローカリティを考慮したマクロタスクスケジューリング

ループ整合分割後，本手法ではスタティックスケジューリングを用いて，マクロタスクをプロセッサグループに静的に割り当てる．この段階では配列データの具体的なローカルメモリ上の特定位置への割り当てまでは行わないが，異なるプロセッサに割り当てられたマクロタスク間ではデータ依存する配列範囲のデータ転送が発生することを考慮し，ETF/CP/MISF [9]，CP/ETF/MISF [9] 等の複数種類のヒューリスティックアルゴリズムを用いてスケジューリングを行い，スケジューリング長が最も短くなるものを採用する．

2.3 動的なローカルメモリ管理

本ローカルメモリ管理手法の特徴は以下の通りである．

- アジャスタブルブロックと呼ぶ，アプリケーション中の多様な形状・サイズの配列に適した固定サイズのブロック（ローカルメモリ上のデータ管理単位で，各ブロックは階層的に整数

分の1のサブブロックに分割可能）にローカルメモリ領域を分割し，このブロック単位でローカルメモリへのデータの配置，リプレースを行う

- ブロック上に配列を割り当てる際に，テンプレート配列と呼ぶ，元ソースの配列が2次元であれば2次元のテンプレート，3次元であれば3次元のテンプレートを用意し，元ソースプログラム中の配列添え字のままローカルメモリ上のブロックにアクセスできるようにすることで，プログラムの可読性を維持しつつ，データのブロック割り当てを容易にする
- ブロックのデータ割り当て，リプレースすべきブロックの決定は，タスクスケジューリング結果を利用してデータの再利用性を高めるように行う

2.3.1 アジャスタブルブロック

アプリケーションによって，定義・参照される配列の数，次元，サイズは多様である．データをローカルメモリ領域に割り当てる際に可変長ブロックを用いるとフラグメンテーションが生じ，ローカルメモリ利用効率が低下する．これを避けるためには固定ブロックサイズが望ましいが，本手法ではソフト管理のブロック割り当てを行うため，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ができるアジャスタブルブロックの概念を提案する．このアジャスタブルブロックにおいては，そのアプリケーション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すると共に，ブロックを整数分の1のサイズに分割したサブブロックを定義できるようにして，階層的にブロックを定義することで，異なる配列等をより効率良くローカルメモリに割り当てたり，リプレースしたりすることを可能としている．なお，今回のインプリメントではブロックのサイズは2の冪乗のサイズとし，各階層ブロック間で下位レベルのサブブロックは上位レベルのブロックの1/2のサイズとすることで，サブブロックを階層的に定義しやすくしている．

上記のブロックによるローカルメモリの管理例を図1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．最大のブロックサイズを $full_block_size$ と呼び，レベルが $l+1$ のブロックはレベルが l のブロックの1/2のサイズとなるようにサブブロックを定義する．図1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，各レベル毎にアドレスの低いほうを0とし，順にブロック番号を割り振る．

部分配列の次元 k のサイズを S_k，次元数を d，型のサイズを $type_size$ とし，C_k を S_k より大きい2の冪乗の数値とすると，部分配列に対応するブロックのサイズ S は

$$2^{n-1} < S_k \leqq 2^n$$
$$C_k = 2^n$$
$$S = type_size \prod_{k=1}^{d} C_k$$

と求まる．従って，各部分配列に対応するブロックのレベル l は，

$$S = \frac{full_block_size}{2^l}$$

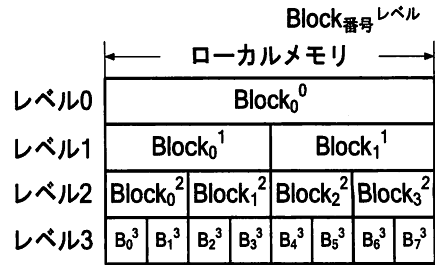

図1 ブロックによるローカルメモリ管理

を満たす l として求まる．

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多様なサイズの部分配列を割り当てることが可能となり，フラグメンテーションを抑えた上でローカルメモリの効率的な利用が可能になる．

2.3.2 テンプレート配列を用いたブロックへのアクセス記述

ローカルメモリへのアクセスをコンパイラの出力する，Cあるいは Fortran で記述する際に，ローカルメモリ上のブロックを指定するのに一次元の配列を用いることを考える．この時，例えばローカルメモリ上のブロックに割り当てられた部分配列が多次元配列であった場合，この部分配列を上述の一次元配列を介してアクセスすると，配列の添え字に複雑な計算が必要となってしまう．そのようなことが起きると，ローカルメモリ管理後のプログラムの可読性は著しく劣化する．そこで，ブロックへ割り当てられた配列データへのアクセスをプログラムで表現する際に，テンプレート配列を用いる．

まず，テンプレートは，ソースプログラム中の部分配列と同じ次元数，同じ型を持つように定義する．部分配列の次元 k のサイズを S_k，次元数を d，型を $type$ とし，テンプレートの次元 k のサイズを T_k とすると，この部分配列に対応するテンプレート $templ$ は次のように定義される．

$$2^{n-1} < S_k \leqq 2^n$$
$$T_k = 2^n$$

$type\ Templ[T_d][T_{d-1}]...[T_1];$

テンプレートのサイズは，同じレベルのブロックのサイズと等しくし，テンプレートをブロックにマッピングする．

各ブロックには，図2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，ソースプログラム中の配列を分割した部分配列は，次元数の同じテンプレートに割り当てる．このテンプレートに，ブロック番号を指定する次元を持たせ，何番目のブロックを利用するか指示することができるようにし，これをテンプレート配列と定義する．テンプレート配列は，テンプレートの型を $type$，次元数を d，次元 k のサイズを T_k とし，同一レベル上のブロックの個数を block_num とすると，以下のような $TemplArray$ と定義される．

$type\ TemplArray[block_num][T_d][T_{d-1}]...[T_1];$

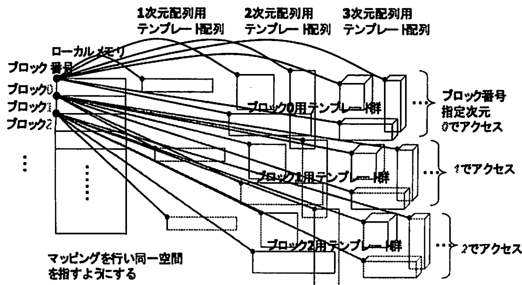

図 2 ローカルメモリとテンプレート配列の関係

```
for (i=0; i<2; i++)
 for (j=0; j<16; j++)
  for (k=0; k<4; k++)
   A[i][j][k]=i+j+k;
```

図 3 サンプルプログラム

それぞれの部分配列を対応するテンプレート配列にマッピングすることにより，部分配列をローカルメモリ上のブロックへマッピングする．テンプレート配列とローカルメモリ上のブロックへのマッピングの例を図 2 に示す．なお，図中ではブロックのレベルは省略した．図 2 では，部分配列に対応する 1 次元，2 次元，3 次元用のテンプレート配列が各ブロックにマッピングされている様子を表す．これにより，データ（部分配列）のブロックへの割り当ては，各ブロック上の対応するテンプレートへの割り当ての形で実現できるようになる．

以上の手順を図 3 のサンプルプログラムを用いて説明する．図 3 のような部分配列 A[2][16][4] を，サイズが 1024bytes のローカルメモリ上のブロックへ割り当てるとする．なお，配列 A は char 型とし，1 要素 1byte とする．部分配列 A を割り当てるブロックのサイズは 128bytes と求まる．従って，部分配列 A に対応するブロックの個数は 8 個となる．部分配列 A に対応するテンプレートは Templ[2][16][4] のようになり，テンプレート配列は TemplArray[8][2][16][4] となる．このときのローカルメモリ上のブロックとテンプレート配列との対応関係を図 4 に示す．なお，図中のブロックのレベルは省略した．この図で示されるように，ブロックと，この次元のテンプレートは一対一で対応し，テンプレート配列のブロック番号指定次元の添え字でブロック番号を指定できる．従って，ブロック番号が 2 であるブロックに部分配列 A が割り当てられたとすると，テンプレート配列を利用することで，図 5 の $TemplArray[2][i][j][k]$ に示すように，簡潔にブロックへのアクセスを記述できる．

このように，オフチップメモリをアクセスしていた部分配列を，テンプレート配列を用いたローカルメモリアクセスに変更しても，配列のアクセス添え字は元のプログラムの次元の情報を保ったままとなり，ローカルメモリ管理後のプログラムの可読性を維持する．

2.3.3 ブロックを利用したローカルメモリ配置

2.2.2 節で述べたタスクのプロセッサ上へのスケジューリン

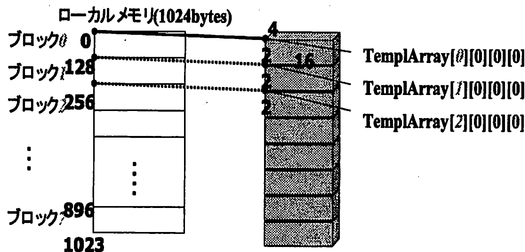

図 4 ローカルメモリ上のブロックとテンプレート配列の対応例

```
for (i=0; i<2; i++)
 for (j=0; j<16; j++)
  for (k=0; k<4; k++)
   TemplArray[2][i][j][k]=i+j+k;
```

図 5 テンプレート配列を利用したサンプルプログラム

グ結果を基に，時刻順に，各プロセッサ上のマクロタスクで扱うそれぞれの部分配列を割り当てるブロック番号を決定する．そのブロック番号を利用して部分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．

マクロタスクで定義・参照される部分配列総サイズがローカルメモリサイズより大きい場合は，ローカルメモリ配置の利得が小さいものをローカルメモリ管理対象からはずす．ローカルメモリ配置の利得は，部分配列の推定アクセス回数を n，推定データ転送時間を D_t とし，ローカルメモリのレイテンシを $lm_latency$，オフチップメモリのレイテンシを $csm_latency$ として，以下のようなローカルメモリ配置の利得 lm_gain と定義する．

$$lm_gain = n(csm_latency - lm_latency) - D_t$$

割り当てるべき部分配列のサイズが複数存在する場合にはレベルの小さい順，すなわちサイズの大きい順に割り当てるブロック番号の決定を行う．マクロタスク間でローカルメモリ上のデータを長時間再利用できるようにするために，以下の手順により割り当てるブロック番号の決定を行う．

1. 割り当て対象となる部分配列が既に割り当てられているローカルメモリ上のブロックが存在する場合，そのブロックを割り当てる
2. 未割り当てで，空いているブロックがあれば，そのブロックを割り当てる
3. 空いているブロックがなければ，スケジューリング時刻上，将来最も先の時刻まで参照されないデータが割り当てられているブロックに対して，以下で述べるブロックのリプレースを行い，空いたブロックに割り当て対象部分配列を割り当てる

ブロックのリプレースとは，部分配列が割り当てられているあるブロックに別の部分配列を割り当てる際に，ブロックに既に割り当てられている部分配列が今後参照されない死んでいるデータである場合には，当然データを共有メモリにストアす

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
データキャッシュレイテンシ	1 クロック
データキャッシュサイズ	16KB
オフチップ CSM レイテンシ	約 55 クロック

ることなく別データを割り当て，再度参照される場合には，その部分配列をオフチップメモリにストアするデータ転送を挿入し，ブロックを空け，新データを割り当てる処理である．このブロックのリプレースは，スケジューリング結果を用いて，将来最も参照される回数の少ないブロックを選択して行う．具体的には，ブロックをリプレースすることにより発生するデータ転送時間を，再度参照されるまでの時間で割った値を再参照度と定義し，再参照度が最も小さいものを将来最も参照されないブロックとしてリプレース対象とする．なお，将来参照されることのない死んでいるデータを保持するブロックの再参照度は 0（分母が無限大のため）とする．

2.3.4 データ転送の挿入

ここでは，発生するデータ転送は，CPU と非同期にバースト転送が可能なデータ転送ユニット（DTU）を利用して効率的に行う．今回のインプリメントでは，配列の定義により発生するローカルメモリからオフチップメモリへの転送はその定義を行うマクロタスクの直後，配列の参照により発生するオフチップメモリからローカルメモリへの転送はその参照を行うマクロタスクの直前に挿入するものとし，CPU によるタスク処理と DTU によるデータ転送のオーバラップは行わないバージョンで実現している．データ転送オーバヘッドを隠蔽できるオーバラップ実行も現在実装中である．

3. 性能評価

本章では，本手法を OSCAR マルチグレイン自動並列化コンパイラに実装し，8 コアの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，OSCAR マルチコアメモリ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 は SH4A プロセッサコアを 8 コア集積している．RP2 のアーキテクチャ図を図 6 に示し，プロセッサの仕様を表 1 に示す．図 6 に示されるように，各プロセッサコアにはローカルデータメモリ（LDM）と複数のプロセッサコアからアクセス可能な分散共有メモリ（DSM），データ転送を行う DTU をそれぞれ持ち，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．本稿の評価では，各プロセッサコア上の LDM と DSM を本ローカルメモリ管理手法を用いて利用する．

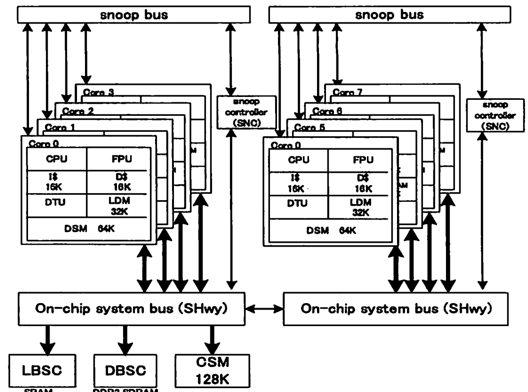

図 6 RP2 のアーキテクチャ図

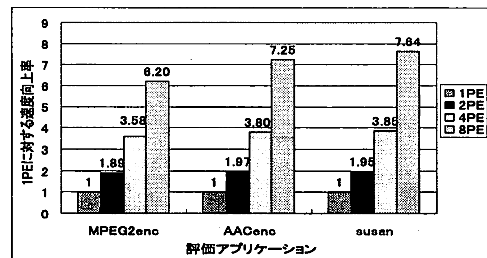

図 7 提案するローカルメモリ管理を用いた場合の逐次実行に対する速度向上

3.2 評価アプリケーション

MediaBench に収録されている MPEG2 エンコーダと，ルネサステクノロジ提供の AAC エンコーダ，MiBench の susan を用いて本手法の性能評価を行った．これらのアプリケーションは制約付き C で書き直されたものを用いた．また，入力データがオフチップ CSM 上に配置された状態から出力結果をオフチップ CSM に書き戻すまでの時間を評価の対象とした．

MPEG2 エンコーダの入力には 352x256 ピクセルの画像 30 フレームを用いた．AAC エンコーダの入力には 30 秒の音源を入力とし，出力データのビットレートは 128Kbps とした．susan は smoothing モードを用いた．

3.3 性能評価結果

本手法の逐次実行に対する，並列処理性能の評価結果を図 7 に示す．図 7 より，逐次実行に比べ 8PE 時に MPEG2 エンコーダで約 6.20 倍，AAC エンコーダで約 7.25 倍，susan で約 7.64 倍とスケーラブルな速度向上が得られることが分かる．

次に，本手法未適用時として全ての配列データをオフチップ CSM に配置したものと，本手法を適用したものとの比較を行った結果を図 8 に示す．8PE 時で比較すると，本手法適用により，未適用の CSM にデータを配置する場合に対して MPEG2enc で約 8.29 倍，AACenc で約 17.00 倍，susan で約 14.50 倍の速度向上を得ることができた．このように大きな速度向上が得られた理由として，ローカルメモリサイズを考慮した分割と配列の適切なローカルメモリ配置により，表 2 に示すように

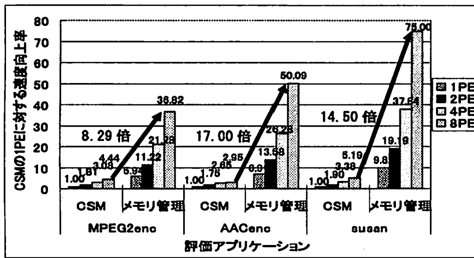

図 8 ローカルメモリ管理を用いた場合のローカルメモリ管理なし (CSM) に対する速度向上

表 2 評価アプリケーションの配列ローカルメモリアクセス率

MPEG2enc	AACenc	susan
99.7%	99.8%	99.4%

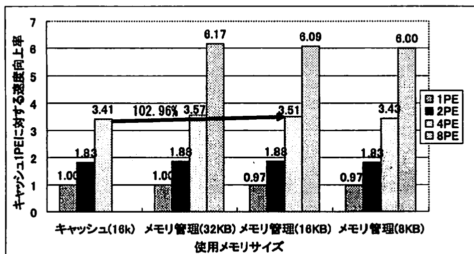

図 9 MPEG2 エンコーダにおけるローカルメモリサイズを変化させた際のキャッシュに対する速度向上

100%に近いローカルメモリアクセス率が達成できたことが挙げられる．ここで配列のローカルメモリアクセス率は，総配列アクセスに対するローカルメモリアクセスの割合として定義している．

次に，本手法の異なるローカルメモリサイズに対する有効性を評価するため，MPEG2enc において本手法で用いるローカルメモリサイズを変化させて計測を行った．また，参考として 4 コアまでのコヒーレントキャッシュを用いた場合と処理時間の比較を行った結果を合わせ図 9 に示す．図 9 に示すキャッシュは，ループ整合分割によるデータローカリティの最適化を行ったものである．またこの評価では，本手法で用いるローカルメモリは LDM のみとした．図 9 のように，ローカルメモリサイズが 32KB の時には，4PE のキャッシュに対して約 4.57%の性能向上，キャッシュと同サイズのメモリ量である 16KB を利用した時には，4PE のキャッシュに対して約 2.96%の速度向上を得た．これらの理由として，本手法によるスケジューリング結果に基づいたブロックの適切なリプレースが，キャッシュの LRU によるリプレース以上にデータローカリティを最適化できたことや，キャッシュではハードウェアによるデータコヒーレンス制御のオーバヘッドが存在することが挙げられる．また，キャッシュサイズの 1/2 の 8KB までローカルメモリサイズを抑えた場合でも，キャッシュと同等の性能を得ることができた．

4. まとめ

本稿では，マルチコア上で，高速小容量のローカルメモリを有効に利用するコンパイラによる動的なローカルメモリ管理手法を提案した．本手法では，データローカリティと並列性を利用するためのループ分割とスケジューリングを行った後，少量のローカルメモリにおいてもデータローカリティを最適化するため，スケジューリング結果を利用したデータ配置，リプレースを行った．

本手法を，32KB のローカルデータメモリと 64KB の分散共有メモリを搭載した SH4A を 8 コア集積したマルチコアである RP2 上で評価を行ったところ，逐次実行に対して，MPEG エンコーダで約 6.20 倍，AAC エンコーダで約 7.25 倍，susan で約 7.73 倍とスケーラブルな速度向上が得られた．また，MPEG2 エンコーダにおいて，使用するローカルメモリサイズをコヒーレントキャッシュサイズと同じ 16KB とした時に，キャッシュに比べ 4PE 時に約 2.96%の速度向上を得ることができた．さらに，MPEG2 エンコーダにおいて使用するローカルメモリサイズをキャッシュサイズの 1/2 の 8KB と変更した場合にも，キャッシュと同等の性能を得ることができた．

謝辞 本研究の一部は NEDO「リアルタイム情報家電用マルチコア技術」,「情報家電用ヘテロジニアスマルチコア」プロジェクト，早稲田大学グローバル COE「アンビエント SoC」の支援により行なわれた．

文献

[1] Avissar, O., Barua, R. and Stewart, D.: An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems, ACM Transactions on Embedded Computing Systems, Vol. 1, No. 1, pp. 6-26(2002).

[2] Panda, P. R., Dutt, N. and Nicolau, A.: Memory issues in embedded systems-on-chip, Kluwer Academic Publishers (1999).

[3] Udayakumaran, S., Dominguez, A. and Barua, R.: Dynamic Allocation for scratch-pad-memory using compile-time decisions, Trans. on Embedded Computing Sys., 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, LCTES '07: Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages compilers, and tools, New York, NY, USA, ACM, pp. 207-216 (2007).

[5] Kandemir, M., Kadayif, I., Choudhary, A., Ramanujam, J. and Kolcu, I.: Compiler-directed scratch pad memory optimization for embedded multiprocessors, Very Large Scale Integration (VLSI) Systems, IEEE Transac-tions on, Vol. 12, No. 3, pp. 281-287 (2004).

[6] Issenin, I., Brockmeyer, E., Durinck, B. and Dutt, N.: Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies, DAC '06: Proceedings of the 43rd annual conference on Design automation New York, NY, USA, ACM, pp. 49-52 (2006).

[7] OSCAR API 仕様書, http://www.kasahara.cs.waseda.ac.jp/api/re

[8] 吉田，越塚，岡本，笠原: 階層的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).

[9] 笠原：並列処理技術，コロナ社 (1991).