演算/メモリ性能バランスを考慮したマルチコア向けオンチップメモリ貸与法

福本尚人[†] 井上弘士[††] 村上和彰[††]

本稿では，プログラムの特徴に応じてメモリ性能を改善するマルチコア・プロセッサ向けオンチップメモリ貸与法を提案し，評価を行った．マルチコア・プロセッサの性能向上阻害要因として，メモリウォール問題の顕著化がある．これに対して，一部のプロセッサコアを「演算用」だけでなく「メモリ性能向上用」に活用することで，性能向上ならびに消費エネルギー削減を目指す．メモリ性能向上用のコアは，自身が持つオンチップメモリを演算用のコアへ貸与し，プログラム実行を行わない．プログラムの特徴に応じてメモリ性能向上用のコア数を変更することで，演算性能とメモリ性能の間の適切なバランスを取る．これにより，主記憶アクセス回数の削減による高性能化，ならびに，一部コアの動作停止による低消費エネルギー化を同時に狙う．提案手法をサポートするコンパイル手法を開発し，実機を用いた定量的な評価を行った結果，最大で 46%の実行時間の削減と 32%の消費エネルギーの削減を達成した．

Performance Balancing: Software-based On-chip Memory Management for Multicore Processors

Naoto Fukumoto,[†] Koji Inoue[††] and Kazuaki Murakami[††]

This paper proposes the concept of performance balancing, and reports its performance and energy impact on a multicore processor. Integrating multiple processor cores into a single chip, can achieve higher peak performance by means of exploiting thread level parallelism. However, the off-chip memory bandwidth which does not scale with the number of cores tends to limit the potential of multicore processors. To solve this issue, the technique proposed in this paper attempts to make a good balance between computation and memorization. Unlike conventional parallel executions, this approach exploits some cores to improve the memory performance. These cores devote the on-chip memory hardware resources to the remaining cores executing the parallelized threads. In our evaluation, it is observed that our approach can achieve 46% of reducing execution time and 32% of reducing energy compared to a conventional parallel execution model.

1. はじめに

複数のプロセッサコア（以下，コアと略す）を 1 チップに搭載したマルチコア・プロセッサが主流となっている．複数のシンプルなコアで並列処理を行うことで，高性能化と低消費エネルギー化を両立できる．微細化技術の進歩とともに，チップに搭載されるコア数は増加する傾向にある．例えば，2006 年にソニー/東芝/IBM から発売された Cell/B.E.[3] では 9 個，2009 年に Sun が発表した Rainbow falls では 16 個のコアが搭載されている．しかしながら，必ずしも，コア数に見合う性能が常に得られるわけではない．プロセッサ−メモリ間の性能差の拡大（いわゆる，メモリウォール問題）が深刻化するためである．低速な主記憶アクセスは，プロセッサ性能を悪化させる．また，I/O ピンボトルネックにより，コア当たりのオフチップメモリバンド幅が低下する．したがって，プロセッサ性能を高めるには，コア数の増加による演算性能の向上のみならず，メモリ性能の改善も極めて重要となる．

そこで本稿では，演算性能とメモリ性能のバランシング（調整）を可能にするマルチコア向けオンチップメモリ貸与法を提案し，それを実現するコンパイル手法を示す☆．従来のマルチコア実行では，スレッドレベル並列性を最大限活用するために，チップ上の全てのコアで並列プログラムを実行する．これに対し，提案方式では，一部のコアをメモリ性能向上用に活用する．具体的には，当該コアでのプログラム実行を強制

† 九州大学大学院 システム情報科学府 情報知能工学専攻
Department of Advanced Information Technology, Kyushu University
†† 九州大学大学院 システム情報科学研究院 情報知能工学部門
Department of Advanced Information Technology, Kyushu University

☆ 本研究の概要は文献 6) で示している．これに対し，本稿では，ヘルパーコア数決定に用いる性能モデルの改善，オンチップメモリ間データ通信の高速化，ならびに，消費エネルギーの評価を行った．

的に禁止し，当該コアが持つオンチップメモリ資源をプログラム実行用のコアに貸出す．プログラムの特徴に応じてメモリ性能向上用コアの数を変更することで，演算性能とメモリ性能のバランスをとる．これにより，主記憶アクセス回数削減による高性能化，ならびに，一部のコアの動作停止による低消費エネルギー化を実現できる．実機を用いた定量的な評価を行った結果，最大 46%の実行時間の削減と 32%の消費エネルギーの削減を達成した．

本稿の構成は以下の通りである．まず，第 2 節で関連研究の紹介を行う．次に，第 3 節で提案方式の概要を述べ，第 4 節で提案方式をサポートするソースコード生成法を説明する．その後，第 5 節で実機を用いた定量的な評価を行う．そして，第 6 節では本稿が前提とするメモリアーキテクチャとは異なる場合を想定し，提案方式の適用可能性を議論する．最後に，第 7 節でまとめる．

2. 関 連 研 究

マルチコア・プロセッサにおいて，未使用コア（またはアイドル状態のコア）を用いてメモリ性能を改善する方式が提案されている．代表的な例としては，未使用コアにてプリフェッチ用ヘルパースレッドを実行することでメモリアクセス・レイテンシを隠蔽する方式[7),13)] や，ソフトウェアキャッシュを実装しオンチップキャッシュのヒット率を向上する方式[15)] などが挙げられる．未使用コアは，コア数より少ない数の逐次プログラムを実行する場合や，並列プログラムの逐次部分を実行する場合にのみ存在する．そのため，これらの状況においては有効である反面，並列プログラムを実行する際には適用することはできない．

複数コアでの並列処理において，実行コア数の増加により性能が悪化する場合がある．この現象に着目して，あえて実行コア数を減らすことで高性能化を狙う手法が提案されている[4),11)]．この手法では，プログラム実行時にハードウェアカウンタから得られた情報をもとに最適な実行コア数を予測する．これらの手法では，演算性能を低下させることでメモリ性能との差を埋める．

我々の提案するマルチコア向けオンチップメモリ貸与法では，未使用コアを使用するのではなく，従来の並列プログラムを実行するコアをメモリ性能向上用のコアに置き換える．そのため，未使用コアが存在しない並列実行部においても適用可能である．また，メモリ性能と演算性能の差を埋めるために，コアを停止させるだけでなくメモリ性能改善用に活用する．これにより，実行コア数を減らす手法より高い性能を達成できる．第 5.3 節において，単純な実行コア数削減方式との比較を行っており，提案方式の有効性を確認している．

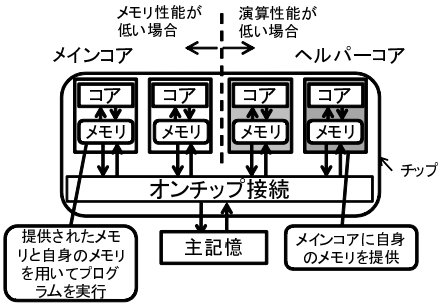


図 1 提案手法の概要
Fig. 1 Basic idea

3. マルチコア向けオンチップメモリ貸与法

3.1 対象プロセッサ

マルチコア・プロセッサのオンチップメモリ構成法としては，ハードウェア制御に基づくキャッシュメモリの搭載や，ソフトウェア制御に基づくスクラッチパッドメモリ (以降，SPM と略す）の活用が挙げられる．特に後者に関しては，メモリ性能の予測容易性や，単純な回路構造に伴う高速動作と低消費電力といった利点があり，多くの組込みシステムや高性能アプリケーションにおいて活用されている．代表的なプロセッサとしては，Cell/B.E. がある．以降，本節では，SPM を搭載したマルチコア・プロセッサに着目し，オンチップメモリ貸与法を説明する．なお，キャッシュメモリを有するより一般的なマルチコア・アーキテクチャへの提案方式の適用に関しては，第 6 節で議論を行う．

前提とするマルチコア・プロセッサでは，各コアは SPM を搭載しており，それぞれはオンチップネットワークにより接続される．格納するデータの入替えは DMA (Direct Memory Access) 転送によって行われる．コアでの命令実行と DMA 転送は同時に行うことができ，アドレス指定により，SPM －主記憶間ならびに SPM － SPM 間の DMA 転送が可能である．

3.2 基本アイディア

従来のマルチコア・プロセッサにおける並列処理ではチップに搭載されている全コアで並列プログラムを実行する．しかしながら，これにより，必ずしも高性能を達成できるわけではない．メモリ性能がボトルネックとなる場合，実行コア数に見合った性能向上が得られない．このような場合，いくつかのコアをメモリ性能向上用に活用することで，より高い性能を実現できる可能性が高い．そこで，マルチコア向けオンチップメモリ貸与法では，図 1 のように，各コアを以下のように使い分ける．

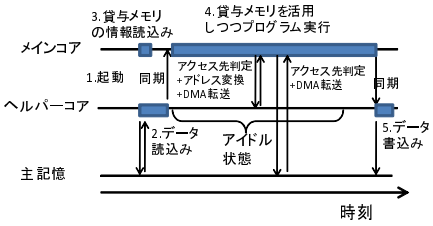


図 2 プログラム実行の流れ
Fig. 2 Execution flow

- メインコア: 並列プログラムを実行するコア．従来のマルチコア実行では，全てのコアがメインコアとして動作する．
- ヘルパーコア: メモリ性能改善を目的として自身のオンチップメモリを他のメインコアに貸与する．並列プログラムの実行は行わない．貸与したオンチップメモリ (以降，貸与メモリと呼ぶ) は，メインコアのオンチップメモリとして利用される．

メモリ貸与法の適用により，メインコアが利用できるオンチップメモリ容量が増加し，オンチップメモリのヒット率が向上する．さらに，メインコア数の減少により，共有資源におけるアクセス競合の発生率が減る．これらの効果で，ヘルパーコアを増加させることにより，メモリ性能が向上する．しかしながら，その反面，ヘルパーコアは並列プログラムを実行しないため，演算性能が低下してしまう．ヘルパーコア追加によるメモリ性能向上が，メインコア減少による演算性能低下を上回ることができれば，性能向上が達成できる．したがって，適切なコア配分を選択することが重要となる．メモリ貸与法では，演算性能とメモリ性能のバランスを考慮し，最も高い性能を発揮するコア配分を求める．

3.3 プログラム実行の流れ

メモリ貸与法を適用後のプログラム実行は図 2 のようになる．

(1) メインコアとヘルパーコアを起動する．
(2) ヘルパーコアは主記憶から自身の SPM にデータを読込む．
(3) メインコアはヘルパーコアの貸与メモリに格納されたデータに関するアドレス情報を取得する．
(4) ヘルパーコアのデータ読込み完了後，メインコアは貸与メモリを活用しながらプログラム実行を行う．
(5) メインコアのプログラム実行終了後に，ヘルパーコアは貸与メモリのデータを主記憶へ書き戻す．

(4) においてメインコアが DMA 転送を行うとき，参照データが貸与メモリ内に存在するか否か判定する．

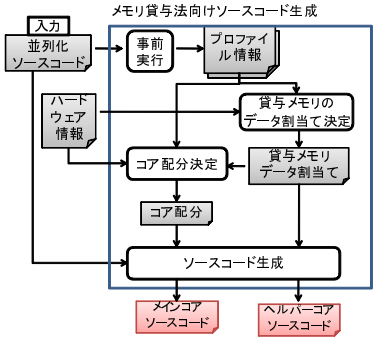


図 3 コンパイルフロー
Fig. 3 Compile flow

各メインコアは，貸与メモリに格納されているデータのアドレス (以降，タグと呼ぶ) を保持している．このタグを参照することで，DMA 転送対象データの格納されたメモリを特定する．もし，貸与メモリに参照データが格納されている場合は，ヘルパーコアから直接当該データを得るためにアドレス変換を行い DMA 転送を発行する．

4. メモリ貸与法向けソースコード生成

4.1 コンパイルフロー

メモリ貸与法向けコンパイラでは，図 3 で示すように，並列プログラムコードならびにハードウェア情報を入力として，メモリ貸与法適用後のメインコア用コードならびにヘルパーコア用コードを出力する．これを実現するために，以下の手順でソースコードを生成する．

(1) プロファイル情報取得：コード生成に必要な情報を事前実行により取得する．
(2) 貸与メモリのデータ割当ての決定：取得したプロファイル情報などを元に割当てデータを決定する．
(3) コア配分決定：メインコアとヘルパーコアの適切な配分を決定する．現実時間内に事前実行が終わる場合，メインコアとヘルパーコアの全ての組合わせの実行時間を元に決定する．そうでない状況では，メインコア数 1 での事前実行で得た情報を元に性能を予測することで配分を決定する．
(4) コード生成：メモリ貸与法を適用したメインコアのコードとヘルパーコアのコードを生成する．入力である並列プログラムコードに対して，メインコア用コードではプログラム実行に使用するコア数 (メインコア数) の変更，アドレス変換に使用するタグの挿入が行われる．ヘルパーコア用コードは，貸与メモリ読込みと書出しが

記述される．

4.2 プロファイル情報の取得

本節では，第 4.3 節，第 4.4 節で使用するプロファイル情報の取得方法について説明する．メインコア数 1 で事前実行することでプロファイル情報を得る．必要なプロファイル情報と取得のためのコード修正箇所は以下の通りである．

- DMA 転送対象アドレスとその回数：DMA 転送のトレースを取得する．DMA 転送関数に出力用関数を挿入する．
- DMA 転送によるストール時間：DMA 転送の完了を待つ関数の前後に時間取得用関数を挿入することで取得する．
- 実行時間：時間観測用の関数を自動で挿入することで取得する．
- 並列処理に要する時間：並列実行部の前後に時間観測用の関数を挿入することで取得する．並列実行部の前後にはプログラマが目印を置くこととする．

4.3 貸与メモリの割当てデータの決定法

メモリ貸与法では，メインコアは貸与メモリを自身のオンチップメモリとして活用する．貸与メモリがメインコアの要求するデータを保有している場合，メインコアは高速なオンチップメモリ間転送によりデータを取得できる．そのため，貸与メモリに有用なデータを読込むことは非常に重要である．

コンパイル時に自動で SPM のデータ割当てを決定する手法として，Static allocation[1),10)] がある．Static allocation は，プログラム実行中に SPM にロードしたデータを入替えない前提でデータ割当てを決定する．そこで，本方式では Static allocation を採用する．ただし，文献 1) とは異なり，一定のデータ単位（以降，データ・ブロックと呼ぶ）で割当てを決定する．これは，第 3.3 節で説明したアドレス変換処理の簡略化のためである．

割当てデータは以下の手順で決定される．まず，事前実行で得られた DMA 転送のトレースをもとに，データブロック単位で DMA 転送回数を集計する．次に貸与メモリのアクセス回数最大を目的関数として，ナップサック問題を解くことで，割当てデータを決定する．

4.4 コア配分決定法

メモリ貸与法で性能向上を達成するには，適切なヘルパーコア数でプログラム実行することが重要である．本稿のコア配分決定法では，メインコアとヘルパーコアの全ての組合わせで事前実行することで配分を決定する方法と，メインコア数 1 での事前実行で得られた情報を元に配分を決定する方法がある．それぞれ，事前実行に要する時間と最適なコア配分の予測精度において一長一短ある．前者は事前実行を複数回行い，実行時間を計測することで適切な配分を求める単純な方式であるため，詳細な説明を省略する．以降，後者の方式について説明を行う．なお，本節では，入力データによって実行の振る舞いが大きく変化しない場合を前提として．コア配分の予測を行う．

コア配分決定に使用する性能モデルの導出を行う．あるメインコアの実行時間 $T(m,n)$ は，演算実行に要する実行時間 $T_{exe}(m,n)$，DMA 転送に要する時間 $T_{mem}(m,n)$，演算と DMA 転送の同時実行時間 $T_{over}(m,n)$ を用いて，

$$T(m,n) = T_{exe}(m,n) + T_{mem}(m,n) - T_{over}(m,n)$$

と表すことができる．m,n はそれぞれヘルパーコア数とメインコア数である．

$T_{exe}(m,n)$ は，DMA 転送先アドレス変換に要する時間 $T_{addrtrans}$ とメインコア数 1 におけるプログラム中の逐次実行部の実行時間が全体の実行時間に占める割合 F を用いて，

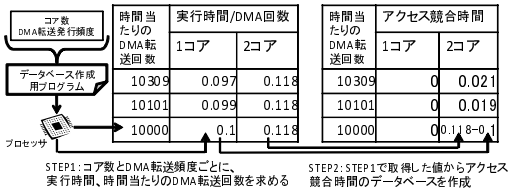
$$T_{exe}(m,n) = T_{exe}(0,1) \times \left(\frac{F}{n} + 1 - F\right) + T_{addrtrans}(m,n) \quad (1)$$

のように表す．アドレス変換処理では，貸与メモリに格納されているデータのアドレスを検索し，DMA 転送対象データが貸与メモリに存在する場合のみ DMA 転送先アドレスを変換する．よって，アドレス変換時間は，DMA 転送先アドレスによって変化する．アドレス変換時間はアドレス変換回数と 1 回あたりの変換時間の積となるため，$T_{addrtrans}$ は，以下のように表すことができる．

$$T_{addrtrans} = \sum_{i=0}^{2l} (AC_i \times AT_i) \quad (2)$$

l は，各コアが持つタグ数（割当て領域数），AC_i はアドレス区間 i に対する DMA 転送回数を示す．ここでアドレス区間 i は，貸与メモリ領域によって区切られる i 番目の区間を指す．たとえば，貸与メモリ領域が一つの場合，貸与メモリ領域よりアドレスの小さい区間をアドレス区間 0，貸与メモリ領域をアドレス区間 1，それ以外の区間をアドレス区間 2 と呼ぶ．AT_i はアドレス区間 i が DMA 転送転送先である場合の 1 回あたりのアドレス変換時間を表す．

$T_{mem}(m,n)$ は，メインコア当たりの DMA 転送回数 AC，貸与メモリアクセスに要する時間 AT_{SPM2}，主記憶アクセスに要する時間 AT_{main}，貸与メモリのヒット率 HR_{SPM2}，共有資源における DMA 転送競合の解決に要する時間 T_{cont} を用いて，以下のように表すことができる．



時間当たりのDMA転送回数	実行時間/DMA回数	
	1コア	2コア
10309	0.097	0.118
10101	0.099	0.118
10000	0.1	0.118

時間当たりのDMA転送回数	アクセス競合時間	
	1コア	2コア
10309	0	0.021
10101	0	0.019
10000	0	0.118-0.1

図 4 アクセス競合時間データベースの作成方法
Fig. 4 Data base on contention time

$$T_{mem}(m,n) = AC(n) \times (HR_{SPM2}(m) \times AT_{SPM2} + (1 - HR_{SPM2}(m)) \times AT_{main} + T_{cont}(m,n))) \quad (3)$$

上記の項のうちメインコア数およびヘルパーコア数によって変化する項は，$T_{cont}(m,n), AC(n), HR_{SPM2}(m)$ である．まず，T_{cont} のモデル化を行う．DMA 転送時のアクセス競合解決に要する時間は，各コアの DMA 転送の発行頻度によって決まる．そこで，アクセス競合時間は，各コアの DMA 転送がプログラム実行中に均等に発行されるとして，コア一つ当たりの DMA 転送実行頻度とメインコア数でアクセス競合時間を近似する．

アクセス競合時間は，DMA 転送の実行頻度とメインコア数を入力として，データベースを参照することで求める．データベースの作成方法の概要を図 4 に示す．一定頻度で DMA 転送を行うデータベース作成用プログラムを用意する．これは DMA 転送の実行頻度と実行コア数を自由に変更できる．STEP1 では，コア数や DMA 転送の実行頻度を変更し，実行時間，DMA 転送回数を求める．次に STEP2 において，STEP1 で得た値を元にアクセス競合時間のデータベースを作成する．各アクセス競合時間の値は，STEP1 で作成したデータベースの同一行において，(求めたいコア数の実行時間/*DMA* 回数の値 - 1 コアでの実行時間/*DMA* 回数の値) で求める．なお，測定した結果，貸与メモリとのデータ通信はアクセス競合へ影響を与えなかったため，メインコアの主記憶に対する DMA 転送回数を元にデータベースを作成する．アクセス競合時間は，メインコア数，ならびに，(総主記憶アクセス回数/メインコア数 1 の実行時間) を用いてデータベースを参照することで求める．

次に $AC(n)$ と HR_{SPM2} について考える．$AC(n)$ は並列化部分と逐次部分を考慮して以下のように求める．

$$AC(n) = AC(1) \times \left(\frac{F}{n} + 1 - F\right)$$

一方，$HR_{SPM2}(m)$ は，貸与メモリのデータ割当ては実行前に決まっているため，DMA 転送のトレースと比較することで求めることができる．

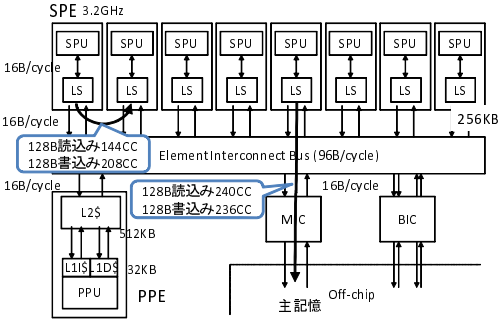


図 5 Cell/B.E. のモデルとパラメータ
Fig. 5 Cell/B.E. model and parameters

最後に，T_{over} についてモデル化を行う．演算と DMA 転送の同時実行時間は，DMA 転送と演算を行うタイミング，ならびに，演算時間とデータ転送時間の比で決まる．前者は，実行するコードが同一であるため，コア配分によって大きく変化しない．後者は，各々のメインコアが処理する演算回数と DMA 転送回数の比はコア配分によらないため，コア配分による変化量は小さい．そこで，DMA 転送時間に占める同時実行時間の割合は，コアの配分によらないとすると，以下のように T_{over} を表すことができる．

$$T_{over}(m,n) = T_{mem}(m,n) \times \frac{T_{over}(0,1)}{T_{mem}(0,1)}$$

ただし，$T_{over}(m,n)$ は $T_{exe}(m,n)$ より大きくならない．

メインコア数 1 における事前実行により得たプロファイル情報をもとに，上記のモデル式を用いて $T(m,n)$ を求める．その後，最も $T(m,n)$ の小さい m,n の組合せを算出する．式中の $F, AC(1), HR_{SPM2}(m), AC_i$ は事前実行により抽出したプロファイル情報をもとに算出する．F は，並列実行部の実行時間とプログラム全体の実行時間との比により求める．$AC(1)$ はトレースより導出する．$T_{exe}(0,1)$ は，$T(0,1)$ と DMA 転送によるストール時間の差分により求める．$T_{over}(0,1)$ は，$T_{mem}(0,1)$ と DMA 転送によるストール時間の差し引きにより算出する．ここで，$T_{mem}(0,1)$ は式 (3) より求める．$AT_{main}, AT_{SPM2}, AT_i$, ならびにアクセス競合時間データベースは，事前に値を実測しておく．

5. 評　　価

5.1 評価環境

提案手法の評価には，株式会社 東芝の Cell Reference Set (CRS) を用いた．CRS は Cell/B.E. とそのチップセット，I/O インターフェースを実装したハー

表 1 ベンチマーク・プログラムの入力とメモリ使用量
Table 1 Input of benchmark programs

プログラム名	入力名	入力サイズ	使用メモリ量
HIMENO	SS	$33 \times 33 \times 65$	3,871 KB
	S	$65 \times 65 \times 129$	29,806 KB
LU	256	256×256	512 KB
	512	512×512	2,048 KB
	1024	1024×1024	8,192 KB
FFT		8,388,608 point	262,144 KB
Matrix_mul		512×512	3,072 KB

ドウェア・プラットフォームである．Cell/B.E. は図 5 のように演算に特化した 8 個のコア (SPE) と 1 個の汎用コア (PPE) で構成される．各 SPE は 256KB の LS と呼ばれる SPM と演算を行う SPU で構成される．本評価で使用した CRS で動作する SPE は 7 個であり，これらをメインコアまたはヘルパーコアとして使用する．第 4.4 節で示した性能モデルに使用する AT_i, アクセス競合データベースの値，AT_{SPM2}，AT_{main} は，Cell/B.E. において実測した値を用いる．代表して，図 5 には転送サイズ 128B での DMA 転送時間に要するクロックサイクル数を記載している．

貸与メモリへ格納するデータは，主記憶-SPM 間の DMA 転送時間と SPM-SPM 間の DMA 転送時間を比較した結果，Cell/B.E. ではどの DMA 転送サイズにおいても，書込み対象データを貸与メモリに格納するより，読込み対象データを格納した方が，アクセス時間の削減量が大きいことが分かった．そこで，主記憶からの読込み回数が最小となるようヘルパーコアのデータ割当てを決定する．貸与メモリのデータブロックのサイズは，アドレス変換時間とメモリ性能改善効果のバランスを考え，SPM 容量とした．実行時間は各 SPE の実行時間のうち最も長いものを選択する．

評価には，以下の 4 つの並列プログラムを用いた．

- *HIMENO*[12]：公開されているソースコードに対し並列化を行うとともにダブルバッファリングを適用した．
- *LU*：並列ベンチマークプログラム集の SPLASH-2[14] よりソースコードを入手し，これにダブルバッファリングを新たに適用した．
- *FFT*：IBM Cell SDK 3.1 のサンプルコードからソースコードを取得し，コア数を 2 のべき乗以外で実行できるように修正した．すでに Cell/B.E. 向けにカスタマイズされているため，新たなチューニングは行っていない．
- *Matrix_mul*：IBM Cell SDK 3.1 のサンプルコードからソースコードを取得した．すでに Cell/B.E. 向けにカスタマイズされているため，新たなチューニングは行っていない．

評価に使用したプログラムの入力とメモリ使用量を表 1 に示す．使用メモリ量は各プログラムにおける主要な配列の容量から算出した．第 4.4 節で述べたように本方式では，入力データによって実行の振る舞いが大きく変化しない場合での利用を前提としている．そこで，評価においては，事前実行に評価時と同一の入力を用いる．

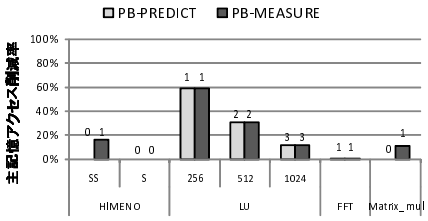


図 6 主記憶アクセスの削減率
Fig. 6 Hit rate of on-chip memory

Cell/B.E. における提案手法の効果を議論するため，以下のような評価モデルを定義する．

- **BASE：**従来の単純並列実行モデル．全てのコアを利用して並列プログラムを実行する (つまり，メインコア数 7，ヘルパーコア数 0 としてプログラムを実行)．
- **PB-PREDICT:** 性能予測に基づくコア分配を前提としてメモリ貸与法を適用するモデル．第 4.4 節で提案した方式に基づきメイン/ヘルパーコア配分を決定する．ヘルパーコアの SPM のデータ割当ては第 4.3 節で提案した方式を用いる．
- **PB-MEASURE：** 全探索に基づくコア分配を前提としてメモリ貸与法を適用するモデル．全てのメインコア数とヘルパーコア数の組合せについて事前実行を行うことで最適な配分を求める．貸与メモリのデータ割当て方法は *PB-PREDICT* と同一である．
- **THROTTLE:**第 2 節で紹介したプログラムの特徴に応じて実行コア数を適切に選択するモデル．最も高い性能を達成できるコア数は既知であり，さらに，実行コア数決定によるオーバヘッドが発生しない理想的な状況を想定する．

5.2 主記憶アクセス削減効果

図 6 に各ベンチマークプログラムにおけるメモリ貸与法の適用による主記憶アクセスの削減率を示す．バーの上にある数字はヘルパーコア数を表す（*1* と記述されている場合は，残りの 6 コアがメインコアとなる）．*PB-MEASURE* では，*HIMENO (SS)*，*LU*，*Matrix_mul* において，約 10%から 60%の主記憶アクセス回数を削減できている．*LU* において，入力サイズが小さいほうがヘルパーコア当たりの主記憶アクセスの削減率が高い．これは，入力サイズが小さいと

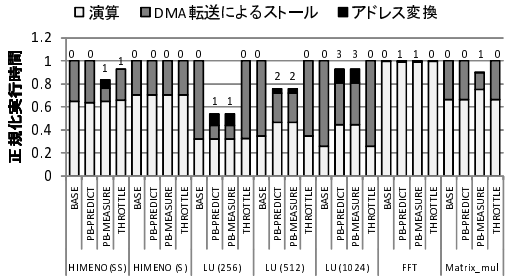


図 7 正規化実行時間
Fig. 7 Execution time normalized by BASE

プログラム中で参照するデータ容量が小さくなり，貸与メモリへ効率よくデータを割当てることができたためである．これらのプログラムではヘルパーコア追加により主記憶アクセスが効果的に削減できている．*FFT* ではヘルパーコア数は 1 であるが，主記憶アクセスの削減効果は非常に低い．その他のプログラムでは，主記憶アクセス削減効果がない．これは，従来の全コア実行が最も性能が高いためである．この場合ヘルパーコアは必要ない．一方，*PB-PREDICT* では，ほぼ全てのプログラムにおいて *PB-MEASURED* と同等の主記憶削減を実現できている．ただし，*HIMENO(SS), Matrix_mul* において主記憶アクセスの削減効果が *PB-MEASURE* と異なる．これは従来の全コア実行が最も性能が高いと予測したためである．

5.3 性　　能

図 7 に実機において実測した実行時間とその内訳を示す．縦軸は *BASE* により正規化した実行時間である．横軸はベンチマーク・プログラム名と入力を示す．バーの上に記した数字はヘルパーコア数を示す．

PB-MEASURE による実行時間の削減効果について議論する．メモリ貸与法を適用することで，*LU*, *HIMENO (SS)*, *Matrix_mul* で 11%から 47%の実行時間削減を達成している．これは主記憶アクセス削減による DMA 転送ストール時間削減効果が，メインコア減少による演算時間の増加とアドレス変換によるオーバヘッドを上回ったためである．これらのプログラムでは，ヘルパーコア追加による主記憶アクセス削減率が高いためメモリ性能の改善効果が大きい．特に *LU* では，オフチップバスでのアクセス競合によるメモリ性能低下が大きいため，メモリ性能改善効果が高い．さらに，実行コア数増加による性能向上効果が低いため，メインコア数減少による性能低下が小さい．このような特性を持つプログラムでは，メモリ貸与法による性能改善が期待できる．その他のプログラムでは，従来実行方式である *BASE* と同じ性能である．これは従来の全コア実行が最も性能が高いメインコアとヘルパーコアの配分であるためである．これらのプログラムでは，使用メモリ容量が貸与メモリと比較して非常に大きいため，メモリ性能改善効果が低い．このようなプログラムにおいてもコアの配分を適切に決定することで，性能低下は発生しない．

次に，*PB-PREDICT* による実行時間の削減効果について議論する．*PB-PREDICT* では，第 4.4 節で構築した性能モデルを用いて，メインコア数とヘルパーコア数を決定する．メモリ貸与法を適用することで，ほぼ全てのプログラムにおいて *PB-MEASURE* と同等の性能向上効果が得られている．ただし，*HIMENO(SS)* と *Matrix_mul* において，*PB-PREDICT* と *PB-MEASURE* の結果が異なる．正しく予測できていない理由は，*BASE* における DMA 転送によるストール時間を過小評価しているためである．アクセス競合の発生率は各コアの DMA 転送の発行頻度によって変わる．発行間隔は一定であるとしてモデル化しているため，発行頻度に偏りのあるプログラムでは過小評価することになる．しかしながら，これらのプログラムにおいても適切なコア配分を予測することで，従来の全コア実行と同等の性能が得られている．以上の結果より，性能予測手法により適切にコア配分が決定できていることが分かる．

最後に，第 2 節で紹介したコア数決定方式である *THROTTLE* について議論を行う．ここでは，最も性能の高いコア数を決定できるとして評価を行う．*HIMENO(SS)* において *BASE* と比較して，8%実行時間を削減した．その他のプログラムでは，実行時間の削減効果を得ることができていない．これに対して，*PB-MEASURE* では，*HIMENO(SS)* において，*THROTTLE* より大きな実行時間の削減を達成している．また，*THROTTLE* で実行時間の削減効果が得られていない *LU*，*Matrix_mul* においても，8%から 47%程度の実行時間削減を達成している．この結果より，本稿での実験環境では，メモリ貸与法は既存のコア数決定法より，高い性能向上を達成できることが分かった．

5.4 消費エネルギー

5.4.1 消費エネルギーモデル

評価に使用する消費エネルギーモデルを構築する．本評価では実行時間の大部分を占める SPE での並列実行に着目する．この場合，Cell/B.E. の消費エネルギー E は以下の式で近似できる．

$$E = E_{SPU} + E_{LS} + E_{main}$$

ここで，E_{SPU}, E_{LS}, E_{main} は SPU，LS，主記憶の消費エネルギーを表す．E_{SPU} は，SPU の消費電力 P_{SPU}，ストール時に消費する電力 P_{stall}，メインコアのストール時以外の実行時間 $T_{actmain}$，ヘルパーコア

のストール時以外の実行時間 $T_{acthelper}$，メインコア DMA 転送によるストール時間 $T_{stallmain}$，ヘルパーコアの DMA 転送によるストール時間 $T_{stallhelper}$ を用いて，以下のように表す．

$$E_{SPU} = n \times E_{SPUmain} + m \times E_{SPUhelper} \quad (4)$$

$$E_{SPUmain} = P_{SPU} \times T_{actmain} + P_{stall} \times T_{stallmain} \quad (5)$$

$$E_{SPUhelper} = P_{SPU} \times T_{acthelepr} + P_{stall} \times T_{stallhelper} \quad (6)$$

ここで，n はメインコア数，または，従来実行時の実行コア数，m はヘルパーコア数を表す．

E_{LS} は，ロード/ストア回数 AC_{16B}，ロード/ストア 1 回あたりの消費エネルギー E_{16B} DMA 転送または命令フェッチによる LS アクセス回数 AC_{128B}，DMA 転送または命令フェッチによる LS アクセス当たりの消費エネルギー E_{128B} を用いて以下のように近似する．

$$E_{LS} = AC_{16B} \times E_{16B} + AC_{128B} \times E_{128B} \quad (7)$$

Cell/B.E. では命令フェッチおよび DMA 転送は 128B 幅で行い，ロード/ストア命令は 16B 幅で行う．それぞれのアクセス回数とアクセス当たりの消費エネルギーの積を取ることで LS の消費エネルギーを求める．

E_{main} は，主記憶アクセス回数とアクセス当たりの消費エネルギーの積で表される．

$$E_{main} = AC_{mainread} \times E_{mainread} + AC_{mainwrite} \times E_{mainwrite} \quad (8)$$

ここで，$AC_{mainread}$, $AC_{mainwrite}$, $E_{mainread}$, $E_{mainwrite}$ は，それぞれ主記憶からの読込み回数，主記憶への書込み回数，主記憶読込みに要する消費エネルギー，主記憶書込みに要する消費エネルギーを示す．

このモデルを用いて，消費エネルギーを算出する．消費エネルギーモデルで使用する SPU の消費電力ならびにアクセス当たりの消費エネルギーを表 2 に示す．表 2 に記載されている以外の項は，プログラムごとに計測した値を用いる．$T_{mainactive}$ ならびに $T_{mainstall}$ はメインコア (または従来実行のコア) のうち最も実行時間が長いコアの値を用いる．また，ヘルパーコアでは，データ読込み後は電源遮断により電力を消費しないとして，データ読込み完了までを実行時間とする．P_{stall} は，$P_{SPU} \times \alpha$ とおき，α の値を 0〜1 に変更して評価する．AC_{128B} と AC_{16B} の算出に必要となる命令数ならびにロード/ ストア回数は IBM が開発した Cell/B.E. シミュレータ[2] を用いて求める．$AC_{mainread}$，$AC_{mainwrite}$，AC_{128B} の算出に必要な DMA 転送回数に関する情報は，当該情報を取得できるように各プログラムを修正することで求める．

5.4.2 消費エネルギー評価

メモリ貸与法による消費エネルギー削減効果は以下の 3 つの要因によって変わる．

表 2 SPU の消費電力，アクセス当たりの消費エネルギー
Table 2 SPU power and energy per access

パラメータ	電力/エネルギー	算出方法
SPU		
P_{SPU}	3.334[W]	SPU と LS の平均消費電力 4W[5] から，LS の平均消費電力を差し引いて求めた．LS 平均消費電力は，CPI=1，全命令の 35%がロード/ストア命令，5%が DMA 転送命令として算出した．P_{stall} の値は，$P_{SPU} \times \alpha$ とおき，実験時に α の値を 0〜1 に変更して評価する．
LS		
E_{16B}	0.3389[nJ]	CACTI 6.5[9] を用いて，プロセステクノロジー 90nm の Cell/B.E. における LS の構成を入力して求めた．
E_{128B}	1.1015[nJ]	
主記憶		
$E_{mainwrite}$	17.97[nJ]	DMA 転送サイズが 64B であるとして文献 9) を参考に求めた．転送サイズが 64B より小さい場合においても 64B 転送と同じ消費エネルギーを消費する．64B より大きいサイズの DMA 転送の場合は，表記の消費エネルギー×「転送サイズ/64B」を要する．
$E_{mainread}$	12.83[nJ]	

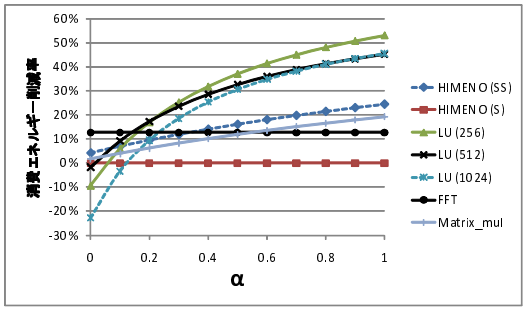


図 8 消費エネルギー削減率
Fig. 8 Energy reduction varying α

- ストール時間削減による SPU の消費エネルギーの削減：ヘルパーコア追加によるストール時間の削減量が大きいほど，この削減効果は大きくなる．また，ストール時の消費エネルギーの割合 α が大きいほど，この削減効果は大きい．
- 主記憶アクセス削減による消費エネルギーの削減：主記憶アクセス削減効果が高いほど，削減効果は高い．
- メモリ貸与法適用によるアドレス変換処理やヘルパーコア実行に要する消費エネルギー：ヘルパーコア数が多く DMA 転送の実行頻度が高いほど，アドレス変換処理の消費エネルギーは増加する．ヘルパーコア実行に要する消費エネルギーは，実行するプログラムによる変化量は小さい．

これら 3 つの効果の多寡により消費エネルギーの削減効果は決まる．

ストール時の消費エネルギーの割合 α を変更した

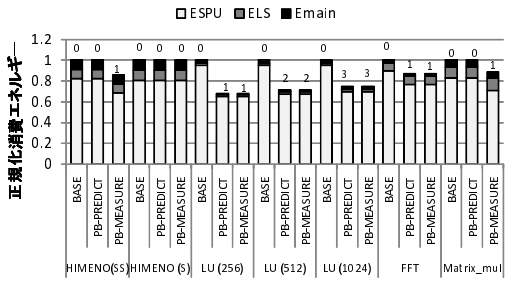


図 9 正規化消費エネルギー
Fig. 9 Energy normalized by BASE

場合の消費エネルギー削減率を図 8 に示す．これは前節の消費エネルギーモデルを用いて計算している．縦軸は *PB-MEASURE* の消費エネルギーの削減率 $(\frac{(BASE\ の\ E)-(PB-MEASURE\ の\ E)}{BASE\ の\ E} \times 100)$ を表す．消費エネルギー削減率が負の場合，メモリ貸与法適用により消費エネルギーが増加していることを表す．*HIMENO (S)* を除いた全てのベンチマーク・プログラムにおいて，α が大きいほど，消費エネルギーの削減率は増加する．DMA 転送によるストール時間の削減量の大きい *LU* では，α に対する消費エネルギーの削減率の変化が大きい．α の割合が大きいプロセッサにおいて，メモリ貸与法は効果的である．

プロセッサストール時に，理想的なクロックゲーティングができていると仮定した場合，動的消費電力はクロックの消費電力で決まる．ここで，クロックで消費する電力の値は，文献 8) のプロセッサ消費電力の 40%を採用する．$\alpha = 0.4$ の場合の各評価モデルにおける消費エネルギー (E) を図 9 に示す．縦軸は *BASE* により正規化した消費エネルギーである．*PB-PREDICT* では，*LU (256)*, *LU (512)*, *LU (1024)*, ならびに *FFT* において，13%から 32%の消費エネルギーの削減を達成している．これは主に，SPU の消費エネルギーの削減による．ヘルパーコアは稼働時間がメインコアと比較して非常に短いため，ヘルパーコアの追加により，SPU の消費電力は削減される．さらに，メモリ貸与法適用により実行時間が削減されるため，SPU の消費エネルギーは減る．主記憶の消費エネルギーは減少する．この原因は，メインコアの主記憶アクセス回数が削減されるためである．特に主記憶アクセス削減効果の高い *LU (256)* では，主記憶の消費エネルギーが大きく削減されている．これらの効果によりメモリ貸与法では効果的に消費エネルギーを削減することができる．

PB-MEASURE では，*PB-PREDICT* で消費エネルギー削減効果のあったプログラムに加えて，*HIMENO (SS)*，*Matrix_mul* において消費エネルギー削減を達成している．

6. キャッシュ搭載型マルチコアモデルに対する適用可能性

本稿では，SPM 搭載型マルチコア・プロセッサを前提として，メモリ貸与法について議論した．本節ではキャッシュ搭載型マルチコア・プロセッサモデルに対してメモリ貸与法の適用について議論する．

メモリ貸与法はキャッシュ搭載型のマルチコア・プロセッサに適用することが可能である．第 4.1 節と同様に，プロファイル情報を取得し，貸与メモリへの割当てデータの決定，および，適切なコア配分の決定を行う．ヘルパーコアは，指定されたデータをロード命令で自身のキャッシュメモリに読込んだ後，停止する．メインコアは，通常通りメモリアクセスを行うことで貸与メモリを使用できる．これは，コヒーレンス制御をハードウェアが行っているためである．ただし，メインコアによって書込みが発生する場合，Write Invalidate 方式を採用していると，貸与メモリ内のデータが無効化される．したがって，ヘルパーコアは貸与メモリ内のデータを定期的に更新するなどの対策を行う必要がある．

また，手法適用の効果はプロセッサの構成によって異なる．各コア占有のオンチップメモリ領域（つまり，貸与可能なメモリ領域）が大きく，オンチップメモリ間データ通信が高速であるほど，性能改善効果は大きい．したがって，貸与できるメモリ領域が小さい共有キャッシュ搭載型マルチコア・プロセッサでは，メモリ貸与法適用による性能向上効果は低いと考えられる．一方，非共有キャッシュ搭載型マルチコア・プロセッサでは，メモリ貸与法適用による性能改善が期待できる．

7. おわりに

本稿では，プログラムの特徴に応じて適応的にプロセッサを動作させるマルチコア向けオンチップメモリ貸与法の提案し，Cell/B.E. プロセッサにより有効性を示した．本手法では，プログラムの特徴を分析し，性能向上が得られると予測した場合のみ，プロセッサコアをメモリ性能改善に使用する．提案手法を適用するソースコード変換手法を実装し，評価した結果，全てのコアで並列プログラムを実行する従来方式と比較して，最大 46%の実行時間の削減を達成した．さらに消費エネルギーモデルにより評価した結果，最大 32%の消費エネルギー削減効果が得られた．今後は，事前実行に小規模入力を用いた場合における最適なコア配分決定法を研究する予定である．

謝辞 日頃から御討論頂いております九州大学安浦・村上・松永・井上研究室ならびにシステム LSI 究セン

ターの諸氏に感謝いたします．なお，本研究は一部，半導体理工学研究センター（STARC）との共同研究ならびに科学研究費補助金（課題番号：21680005）による．

参 考 文 献

1) O. Avissar, R. Barua, and D. Stewart. An Optimal Memory Allocation Scheme for Scratchpad-based Embedded Systems. *ACM Transction Embeded Computing Systems*, Vol. 1, No. 1, pp. 6–26, 2002.
2) P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson, et al. Mambo: A Full System Simulator for the PowerPC Architecture. *ACM SIGMETRICS Performance Evaluation Review*, Vol. 31, No. 4, pp. 8–12, 2004.
3) T.Chen, R.Raghavan, J.N. Dale, and E.Iwata. Cell Broadband Engine Architecture and its First Implementation - A Performance View. *IBM Journal of Research and Development*, Vol. 51, No. 5, pp. 559 –572, 2007.
4) M. Curtis-Maury, K. Singh, S.A. McKee, F. Blagojevic, D.S. Nikolopoulos, B.R. DeSupinski, and M.Schulz. Identifying Energy-Efficient Concurrency Levels using Machine Learning. In *Proceedings of the Internatinal Workshop on Green Computing*, pp. 488–495, 2007.
5) B. Flachs, S. Asano, S.H. Dhong, P. Hotstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H. Oh, S.M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, and N. Yano. A Streaming Processing Unit for a CELL Processor. In *Digest of Technical Papers of International Solid-State Circuits Conference*, pp. 134 –135, 2005.
6) N.Fukumoto, K.Imazato, K.Inoue, and K.Murakami. Performance Balancing: Software-based On-chip Memory Management for Effective CMP Executions. In *Proceedings of the 10th MEDEA Workshop on Memory Performance: Dealing with Applications, Systems and Architecture*, pp. 28–34, 2009.
7) I. Ganusov and M. Burtscher. Efficient Emulation of Hardware Prefetchers via Event-Driven Helper Threading. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, pp. 144–153, 2006.
8) P.E. Gronowski, W.J. Bowhill, R.P. Preston, M.K. Gowan, and R.L. Allmon. High-Performance Microprocessor Design. *IEEE Journal of Solid-State Circuits*, Vol. 33, No. 5, pp. 676–686, 1998.
9) N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. In *School of Computing, University of Utah, Technology Report*, 2007.
10) S. Steinke, L. Wehmeyer, Bo-Sik Lee, and P. Marwedel. Assigning Program and Data Objects to Scratchpad for Energy Reduction. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 409 –415, 2002.
11) M. A. Suleman, M. K. Qureshi, and Y. N. Patt. Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-Threaded Workloads on CMPs. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 277–286, 2008.
12) Himeno benchmark: http://accc.riken.jp/hpc_e.html.
13) D. H. Woo and H. S. Lee. COMPASS: A Programmable Data Prefetcher using Idle GPU Shaders. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 297–310, 2010.
14) S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of 22nd International Symposium on Computer Architecture*, pp. 24–36, 1995.
15) 森洋介, 森谷章, 吉瀬謙二. マルチコアプロセッサの高速化を目指したキャッシュコアの最適化. 先進的計算基盤システムシンポジウム SACSIS2009 論文集, pp. 389–398, 2009.