社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－ARC－178 (20)
2008／5／14

演算/メモリ性能バランスを考慮した Cell/B.E. 向けオンチップ・メモリ活用法とその評価

林 徹生† 福本 尚人† 今里 賢一† 井上 弘士† 村上 和彰†

† 九州大学大学院システム情報科学府 〒819-0395 福岡市西区元岡 744 番地
E-mail: {hayashi,fukumoto,imazato}@c.csce.kyushu-u.ac.jp, {inoue,murakami}@i.kyushu-u.ac.jp

あらまし 現在我々は，チップマルチプロセッサの高性能化を目的とした演算/メモリ性能バランシング技術を提案している．本技術では，チップ内に搭載された複数コアを必要に応じて「演算用」もしくは「メモリ性能向上用」として使い分ける．本方式では，如何にして適切なコア配分を実現するかが極めて重要となる．そこで本稿では，性能モデリングに基づくコア分配法を提案する．また，本方式を Cell/B.E. プロセッサに実装し，その有効性を評価する．ベンチマークプログラムを用いた定量的評価を行った結果，単純な並列処理に比べて最大で 14.5%の性能向上を達成できた．
キーワード チップマルチプロセッサ，並列処理，Cell Broadband Engine，演算/メモリ性能バランシング

Performance Balancing: An Implementation of Efficient On-chip Memory Hierarchy on Cell/B.E.

Tetsuo HAYASHI†, Naoto FUKUMOTO†, Kenichi IMAZATO†, Koji INOUE†, and Kazuaki MURAKAMI†

† Department of Information Science and Electrical Engineering, Kyushu University
744 Motooka Nishi-ku Fukuoka 819-0395 JAPAN
E-mail: {hayashi,fukumoto,imazato}@c.csce.kyushu-u.ac.jp, {inoue,murakami}@i.kyushu-u.ac.jp

Abstract We have proposed the concept of *Performance Balancing* to improve the CMP performance. This approach attempts to exploit the on-chip cores not only for executing the parallelized threads, but also for improving the memory performance. In this technique, it is very important to decide an appropriate number of cores dedicated to memory performance improvements. In this paper, we propose an algorithm to solve this problem and implement it on a Cell/B.E. processor. In our evaluation, it is observed that our approach can achieve 14% performance improvement in the best case compared to a conventional CMP model.
Key words CMP, parallel processing, Cell Broadband Engine, Performance Balancing

1. はじめに

現在，複数のプロセッサコアを 1 チップに搭載したチップマルチプロセッサ (CMP) が主流となっている．チップ内スレッドレベル並列処理によりチップ単体での高い演算性能を達成することができるためである．例えば，Intel の Xeon や IBM の Power6 [1] では 2 つのコアを 1 個のプロセッサチップに搭載している．また，半導体の微細加工技術の進展に伴い搭載コア数は増加傾向にあり，Niagara [2] や Cell Broadband Engine(Cell/B.E.) [3] のように 8〜9 個のコアを搭載した商用 CMP も登場している．

しかしながら，パッケージ制約による I/O ピンの制限等によりオフチップ・メモリバンド幅はコア数に対してスケールしない．また，主記憶で使用される DRAM の動作速度はプロセッサ速度と比較して極めて遅い．その結果，CMP ではプロセッサーメモリ間の性能差拡大 (いわゆる，メモリ・ウォール問題) がより深刻となる．

この問題を解決するため，現在我々は演算/メモリ性能バランシング技術を提案している [4]．従来の単純な並列処理では，チップに搭載された全てのコアをスレッド実行に活用する．これに対し，提案方式では，コアを必要に応じて「演算用」もしくは「メモリ性能向上用」として使い分ける．これにより，演算性能とメモリ性能の適切なバランスを実現し CMP のトータル性能を向上する．本方式では，如何にして適切なコア配分を実現するかが極めて重要となる．しかしながら，文献 [4] では演算用/メモリ性能向上用コアの最適な配分は既知であると仮

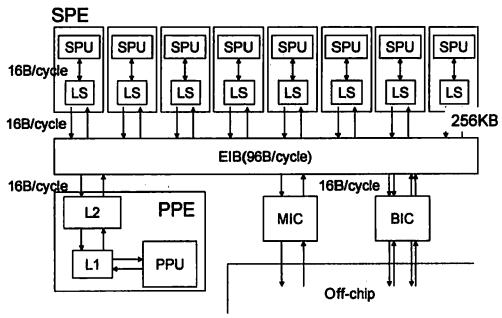

図 1 Cell/B.E. のブロック図

定していた.

そこで本稿では，性能モデリングに基づくコア分配アルゴリズムを提案する．また，提案方式を Cell/B.E. プロセッサに実装し，その有効性を評価する．本稿で提案するコア分配決定法では，実行対象アプリケーション・プログラムに対して小規模入力サンプルを用いた事前実行を行う．これによりプロファイル情報を取得し，性能モデル式を構築することで適切なコア配分を決定する．

本稿は以下の構成である．第 2 節では，演算/メモリ性能バランシングの概要を説明する．次に，第 3 節では，演算用とメモリ性能向上用のコア配分を決定するアルゴリズムを提案する．第 4 節ではベンチマーク・プログラムを用いた定量的評価の結果を行い，提案方式の有効性を示す．そして，第 5 節で関連研究について述べ，第 6 節で簡単にまとめる．

2. Cell/B.E. を対象とした演算/メモリ性能バランシング

2.1 Cell/B.E. アーキテクチャ

Cell/B.E. は，1 個の PPE (PowerPC Processor Element) と 8 個の SPE (Synergistic Processor Element) が搭載されたヘテロジニアス型 CMP である [3]．また，各 SPE は 256KB のソフトウェア制御オンチップ・メモリ (以下，Local Store:LS と略す) を搭載している．Cell/B.E. の内部構成を図 1 に示す．PPE は汎用プロセッサ・コアであり，SPE へのスレッド分配や標準 I/O 制御等を司る．一方，SPE は SIMD 演算器を搭載した高速エンジンであり，これを複数個活用することで高い演算性能を達成することができる．

2.2 メモリ貸与に基づく性能バランシング

一般に，CMP では実行対象となるスレッドを全てのコアに割当てる．しかしながら，メモリ・ボトルネックが顕著に表れる場合には，使用コア数に見合った性能向上効果を得ることが難しくなる．このような場合，幾つかのコアをメモリ性能の向上を目的に利用することで，より高い性能を実現できる [4]. Cell/B.E. を対象とした演算/メモリ性能バランシングにおいては，各 SPE を以下のように使い分ける

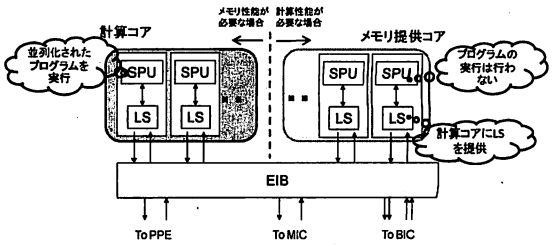

図 2 演算/メモリ性能バランシングの概念図

- **計算コア:** 並列化された実行対象プログラムのスレッドを実行.
- **メモリ提供コア:** LS を他の計算コアに提供 (スレッド実行は行わない).

つまり，図 2 に示すように，メモリ提供コアは自身の LS を計算コアに貸与する．そして，提供された LS は，計算コアにおける LS －主記憶間の階層メモリとして利用される．つまり，演算/メモリ性能バランシングでは，従来手法において主記憶アクセスが発生する場面において，そのアクセス先の一部をメモリ提供コアの LS へとマッピングする．

演算/メモリ性能バランシングにおいては，メモリ提供用コア数を増加させることにより，計算コアが利用可能なオンチップメモリ容量が増大する．しかしながら，その反面，メモリ提供コアは対象プログラムのスレッドを実行しないため，CMP 全体の演算性能が低下する．この悪影響に対し，メモリ提供コアによるメモリ性能向上効果が十分大きい場合には，CMP のトータル性能の向上することができる．したがって，演算性能とメモリ性能のバランスを考慮し，適切な計算/メモリ提供コア数を決定する必要がある．

2.3 メモリ提供コアへの主記憶領域マッピング対象データ

演算/メモリ性能バランシングによる性能向上効果を最大限発揮するためには，実行性能に対して影響の大きな主記憶データをメモリ提供コアの LS へ割当てる必要がある．一般に，LS のようなオンチップ・メモリを搭載する場合，プログラム開発者は，DMA 転送命令やバッファリング技術を活用することでオフチップ・アクセス・レイテンシを隠蔽する．しかしながら，以下のような場合には十分にオフチップ・メモリアクセス・レイテンシ（以降，主記憶レイテンシと略す）を隠蔽できない．メモリ提供コアに対する主記憶領域マッピングを行う際には，これらを考慮する必要がある．

(1) オーバラップ対象演算の不足：多くの場合，演算対象データが必要となる前に DMA 転送を行うことでプリフェッチ効果を得ることができる．しかしながら，最初の演算において必要となるデータへのアクセスや，DMA 転送の発行を十分に前倒しできない場合には主記憶レイテンシを十分に隠蔽することができない．

(2) 頻繁な DMA 転送要求：ループ実行に着目した場合，バッファリングによりイタレーション跨いだ最適化が可能となり，主記憶レイテンシを隠蔽できる（例えば，次イタレー

```
/* メモリ提供コア数の配列を確保 */
Step.1   unsigned long array[ Num_Aid ] ;

/* メモリ提供コアのLSの開始アドレスを取得 */
Step.2   for( i=0; i<Num_Aid; i++)
                array[ i ] = Aid[ i ] +0x6000 ;

/* メモリ提供コアのLS、もしくは、主記憶へアクセス */
Step.3   if ( offset > Size*Num_Aid)   //主記憶へアクセス
                Address = Base + offset ;
         else                          // メモリ提供コアのLSへアクセス
                Address = array[offset/Size] + offset %Size;
```

図 3　プログラム中のアクセス先判別部分

ションで必要となるデータを事前に読込む等)．しかしながら，DMA 転送時間に対して各イタレーションでの演算時間が短い場合には，主記憶レイテンシを十分隠蔽することができない．

(3) 同期処理の実行：並列プログラムの実行においては，その多くの場合で同期操作を伴う．早期に同期ポイントへ到着したコアは，他の全てのコアが同期ポイントへ到着するまで待ち合わせる必要がある．そのため，同期直後の DMA 転送は同期処理完了後まで発行できないため，主記憶レイテンシを隠蔽することができない．

2.4 Cell/B.E. におけるメモリ貸与法の実装

本手法では，あらかじめ主記憶からメモリ提供コアの LS へマッピングするメモリ領域について決定する．より高い性能向上を実現するには，メモリ提供コアの LS に，前節で議論したような主記憶アクセスを多くマッピングする必要がある．そこで我々は，性能へ与える影響が大きく，静的にアドレスを予測することが容易な (2)，(3) の主記憶アクセスの削減を狙い，マッピングするメモリ領域を決定する．具体的な指定方法としては，「ループ内で利用される共有データの中で最も小さい (0x0000 に近い) アドレスから，利用可能な LS 領域分(注1)× メモリ提供コア数の容量」をマッピングする．このマッピングにより，同期処理後の主記憶アクセスとバッファリングの効果が低い主記憶アクセスを効果的に削減できる．

次に，メモリ提供コアの LS と主記憶を区別してアクセスする方法を説明する．図 3 にプログラム中のアクセス先判別部分を示す．計算コアは，Step1 でメモリ提供コア数 (*Num_Aid*) 分の配列を確保する．Step2 において，各メモリ提供コアの LS のアドレス (*Aid*[*i*]) を取得し，LS 中のプログラム領域 (0x0000-0x5FFF) を除くために 0x6000 を加算する．そして，Step3 においてアクセス先を判定する．*Size* は 1 つあたりの LS で使用できる領域を，*Base* はメモリ提供コアのマッピング開始アドレスを，また，*offset* は *Base* からのアドレス距離を表わす．したがって，*offset* がメモリ供給コアの LS の範囲 (*Size*Num_Aid*) を超えると従来どおり主記憶にアクセスする．逆の場合は，メモリ提供コアの LS にアクセスする．

最後に実際プログラム実行フローを説明する．計算コアが提案手法の恩恵を得るまでには以下の 4 つのステップがある．

(1) メモリ提供コアが起動して，主記憶から自身の LS にデータを読みこむ．

(2) 計算コアが起動してメモリ提供コアの LS の物理アドレス情報を得る．

(3) 計算コアが LS と主記憶のマッピング情報を格納する配列データを作成する．

(4) 計算コアはメモリ提供コアの LS を活用しながら並列処理を行なう．

全ての計算コアの処理が終了した後，メモリ提供コアの LS 上のデータを主記憶に書き戻すことで並列処理を終了する．

3. プロファイルに基づくコア配分決定方法

第 2 節で提案したメモリ/演算性能バランシングで性能向上を得るには，**適切な計算/メモリ提供コア数**で実行しなければならない．適切な計算/メモリ提供コア数は，実行するプログラムの特性やハードウェア構成により変化する．そこで本稿では，事前実行で取得したプロファイルに基づく計算/メモリ提供コア数決定方法を提案する．

最適な計算/メモリ提供コア数は，実行対象となるプログラムと実際の入力データを用いて，計算/メモリ提供コア数のすべての組合せについて事前実行を実施することで得られる．しかしながら，実際には同一入力データを使用することは非現実的である．また，全ての組合せについて事前実行するため多くの時間を費やす．そこで本稿では，精度をある程度犠牲にしつつも，高速に計算/メモリ提供コア数を求める方法を提案する．具体的には，小サイズの入力データを使用することで 1 回の試行時間を短縮する．また，全パタンではなく，いくつかの計算/メモリ提供コア配分パタンについてのみ事前実行する．これにより得たプロファイルに基づき性能モデリングを実施し，最適な計算/メモリ提供コア数を予測する．

コア配分決定方法に使用する性能モデル式の導出を行う．理想的な環境における性能は，コア数を N，プログラム中の並列化可能な割合を p とすると，次式で表すことができる．

$$Speed\ Up = 1/((1-p) + p/N) \quad (1)$$

理想的な環境では，式 (1) のように並列処理性能は極限値に向かって単調増加を示す．しかしながら，実環境の並列処理においてメモリ性能の低下などの理由により，一定コア数を超えると性能は単調減少に転じる．極限値が一意に定まり，かつ，式 (1) と似た曲線を描く関数として，2 次関数を採用する．従来実行方式により正規化した性能を y，計算コア数を x として $y = ax^2 + bx + c$ を解くことで，a, b, c の値を導出する．

(注1)：プログラムとスタック領域を除いた 224KB

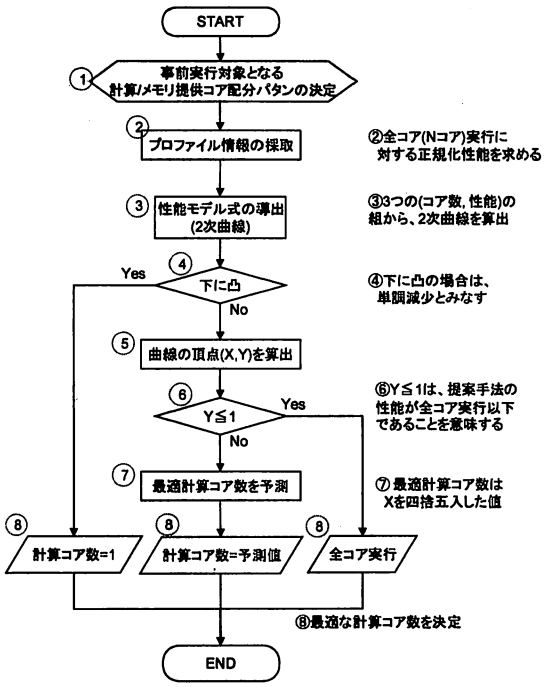

図 4　計算/メモリ提供コア数決定フロー

計算/メモリ提供コア数決定フローを図 4 に示す．例えば、①②③で性能モデリングを行った結果，$y = -0.05 \cdot (x-4.75)^2 + 1.25$ である 2 次曲線を得た場合，④⑤⑥と経由して⑦で計算コア数=5 を得る．ここで，Cell/B.E. での動作を想定すると，コア数 N=7 であるので計算コア数=5，メモリ提供コア数=2 を最適と予測する．

4. 性能評価

4.1 準　　備

性能評価には，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ドウェア・プラットフォームである．CRS では，内蔵する Cell/B.E. は，SPE が 7 個動作する．この 7 個の SPE を計算コア，または，メモリ提供コアとして用いて性能評価を行う．実行時間は 10 回計測したもののうち最大/最小値を除いた平均値を使用する．また，性能は実行時間の逆数とする．

評価対象プログラムとして，姫野ベンチマーク・プログラム [5] と Mibench [6] の Susan-Edge を用いた．姫野ベンチマーク・プログラムのコード修正は過去の報告 [7] を参考に，3 重ループの 2 番目のループレベルの粒度 (反復回数 →i→j→k の多重ループ構造における i) で並列化した．一方，Susan-Edge は，(1) 初期化，(2) エッジ強調，(3) エッジ補正，(4) 原画像重ね，の 4 ステップのうち，住吉らの報告 [8] を参考に (2)(4) を並列化した．姫野ベンチマーク・プログラムの入力は表 1 を，Susan-Edge の入力は large input を使用した．

表 1　姫野ベンチマーク・プログラムの入力サイズ

size	$i \times j \times k$
SSS(自作)	17 × 17 × 33
SS	33 × 33 × 65
S	65 × 65 × 129
M	129 × 129 × 257

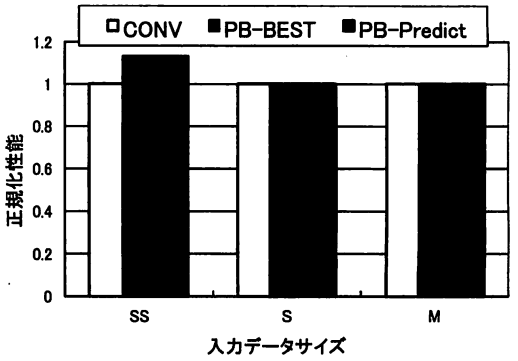

図 5　姫野ベンチマークの正規化実効性能

Cell/B.E. での性能評価に対して提案手法による性能への影響を議論するため，評価モデルを構築する．以下に評価モデルを示す．

- **CONV:** 従来の単純並列実行モデル．全ての SPE コアを利用して並列化されたコードを実行する．
- **PB-BEST:** 理想的な演算/メモリ性能バランシングを前提としたモデル．同一入力を用い，計算/メモリ提供コア配分に関する全ての組み合わせについて事前実行する．これにより，最も高い性能を実現する計算/メモリ提供コア分配を実施する．
- **PB-Predict:** 現実的な演算/メモリ性能バランシングを前提としたモデル．第 3 節で提案した方式に基づき計算/メモリ提供コア配分を決定する．事前実行にはそれぞれのベンチマーク・プログラムより 1 ランク小規模な入力データを使用する．例えば，姫野ベンチマークの入力サイズ S のコア配分予測には入力サイズ SS を，Susan-Edge の large input のコア配分を予測する場合には small input を使用する．そして，計算コア数 2，4，6 のプロファイル情報をもとに性能モデリングを行い最適コア配分を予測する．

4.2 結果 (姫野ベンチマーク)

姫野ベンチマークの実験結果を図 5 に示す．横軸に入力データサイズを，また，縦軸に Conv モデルで正規化した性能を表す．凡例は前節の評価モデルである．入力データ SS では，理想的なコア配分の演算/性能バランシングを行うことで，13%性能が向上している．これは，計算コアの主記憶アクセス回数が効果的に削減された結果である．また，PB-Predict モデルでも同様の結果が得られており，第 3 節で提案したコア配分予測手法が有効であることがわかる．入力データサイズ S と M では，従来方式と理想的なバランシングを行った場合の結果が変わらない．これは，全てのコアで通常実行する場合が，最も効

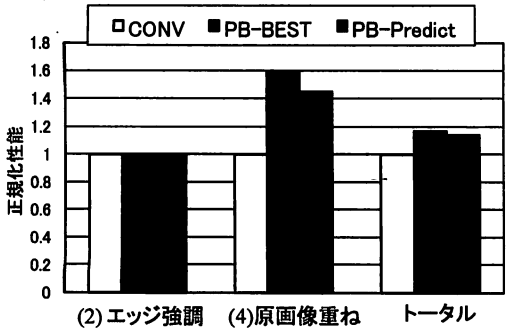

図 6 Susan-Edge の正規化実効性能

果的な計算/メモリ提供コア配分であることを示す．またこの場合も，コア配分予測手法により理想的なコア配分を正確に予測できている．

4.3 結果 (Susan-Edge)

Susan-Edge における並列処理部分の性能のみを抽出した評価結果を図 6 に示す．横軸の数字は，第 4.1 節で説明したプログラム中のステップを意味し，トータルはステップ (2)，(4) を合わせた結果である．トータルでは，理想的なバランシングを行うことで 17%，コア配分予測手法の場合で 14.5%の性能向上が得られている．以下，プログラムのそれぞれのステップについて詳しく議論する．(2) エッジ強調では，従来方式と提案方式の性能が等しい．これは，プログラムの並列処理効率が高いため，メモリ貸与によるメモリ性能向上よりも計算性能向上の効果が高いことによる．一方，(4) 原画像重ねでは，理想的なバランシングにより 60%の性能向上が得られている．これは，計算コア数低下による並列処理性能の低下を最小限に抑えて，メモリ性能の向上を実現したためである．並列処理性能の低下を抑えられた理由は，(4) 原画像重ねでは計算コア数増加による性能向上率が低いためである [8]．また，メモリ性能が大幅に向上した理由は，バッファリングの効果が薄いため主記憶アクセスが性能へ与える影響が大きく，その主記憶アクセスを提案手法により効果的に削減できたためである．計算/メモリ提供コア配分の予測は，(2) エッジ強調では正確に予測できているが，(4) 原画像重ねでは理想的なコア配分を予想できていない．しかしながら，(4) 原画像重ねでも，理想的ではないが 45%の性能向上を得ることができている．

4.4 コア配分決定に関する詳細評価

本節では，コア配分決定方法のパラメータ (使用する入力のサイズ，事前実行対象の選定方法) を変化させて評価を行う．初めに，姫野ベンチマークにおいて入力 SSS を用いて，SS，S，M サイズにおける最適な計算/メモリ提供コア配分を予測する場合を考える．この方法ならば予測に必要な情報を極めて短い時間で採取できる．しかしながら，全ての入力に対して同じデータをもとに予測するため，予測精度が低下する恐れがある．表 2 に，事前実行に使用した計算コアの構成，予測結果の計算コア数，最適な計算コア数を示す．ただし，メモリ提供コアの個数は，(7 − 計算コア数) である．全てのパタンの計算コアを

表 2 入力 SSS における姫野ベンチマークのコア配分予測情報

事前実行対象 計算コア数	PB-Predict 計算コア数	PB-BEST 計算コア数 (SS)	(S)	(M)
1〜7	5			
1,3,5	6	6	7	7
2,4,6	6			

表 3 1 ランク小規模な入力を用いた姫野ベンチマークのコア配分予測情報

事前実行対象 計算コア数	予測対象 サイズ	PB-Predict 計算コア数	PB-BEST 計算コア数
1〜7	SS	5	6
	S	6	7
	M	7	7
1,3,5	SS	6	6
	S	7	7
	M	7	7
2,4,6	SS	6	6
	S	7	7
	M	7	7

表 4 small input を使用した Susan-Edge のコア配分予測情報

事前実行対象 計算コア数	対象ステップ	PB-Predict 計算コア数	PB-BEST 計算コア数
1〜7	ステップ (2)	7	7
	ステップ (4)	2	2
1,3,5	ステップ (2)	7	7
	ステップ (4)	1	2
2,4,6	ステップ (2)	7	7
	ステップ (4)	1	2

事前実行する場合，性能モデリングを行わず，使用した入力の最適な計算コア数を採用している．表 2 において，各入力における PB-Predict と PB-BEST には大きな差異がある．これは，姫野ベンチマークの場合，入力サイズによって提案手法の効果が大きく変化するためである．したがって，入力 SSS を用いた最適な計算/メモリ提供コア配分の見積もりは困難である．

次に，事前実行を行う対象が予測精度に与える影響について解析する．表 3 に，1 ランク小規模な入力を使用した場合のコア配分予測結果を示す．性能モデリングにより計算コア数を予測する場合，最適なコア配分を正確に予測できている．一方，すべての計算/メモリ提供コア配分を試す場合，計算コア数をやや小さく見積る傾向にある．この原因は，入力データサイズが小さい場合，プログラム全体で使用するデータ量に対するメモリ提供コアの LS が大きくなり，メモリ性能改善効果が高くなるためである．以上より，プログラムの振る舞いが入力データによって極端に変化しない場合，1 ランク小規模な入力データを用いることで適切な予測ができることがわかった．

次に，Susan-Edge において，最適コア配分を予測する場合について詳細に評価する．small input を使用して事前実行し，その情報をもとに large input の最適な計算/メモリ提供コア数を予測した場合の結果を表 4 に示す．Susan-Edge においては，全ての計算コア数を事前実行して最適計算/メモリ提供コア配分を予測した場合，正しい予測結果が得られている．一方，性能モデリングにより計算/メモリ提供コア数を予測した場合，ス

テップ (4) の最適計算コア数を予測できていない．しかしながら，計算コア数 1 は計算コア数 2 の次に性能が高く，提案手法により性能向上が得られている．したがって，Susan-Edge では，計算/メモリ提供コア配分予測方法は有効に働くといえる．

5. 関連研究

キャッシュメモリをオンチップ・メモリとして持つ CMP においては，主記憶アクセスの削減を目的とした手法が数多く提案されている [9]〜[11]．Cooperative cache [9] では，あるコアのキャッシュにのみ存在するデータが追い出される時，そのデータをリモートコアのキャッシュにコピーすることでキャッシュミス回数の削減を狙っている．しかしながら，1 つのプログラムを並列処理する時には効果が薄いと考えられる．Ganusov らは CMP においてメモリ性能を向上させるヘルパースレッドを提案している [10]．彼らもメモリ性能に着目しているが，アイドルコアの存在を前提としている．我々の提案手法では全コアを使用することを前提としており，また，計算/メモリ提供コアの数をパラメータ化しているため，アイドルコアの存在も許容する．Cho らは OS によるキャッシュへのページ単位のデータ配置を提案している [11]．隣接するコアにデータを集めるこの手法は，配置するデータの粒度が大きいことが提案手法と似ている．しかしながら，静的なプロファイル情報を元に実現している一方，ネットワーク上での衝突を考慮していない．

複数のプロセッサコアで並列処理を行う場合，プログラムを実行するコア数の増加により性能が悪化する場合がある．この現象に注目して，あえて並列性を落とすことで高性能化を狙う手法が提案されている [12]．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し，予測されたコア数でプログラムを実行する．彼らの手法は実行コア数の増加により性能が悪くなる場合のみ性能向上が得られるが，我々の手法は，さらにメモリを貸与するため，実行コア数の増加により性能が良くなる場合も性能向上を得ることができる．

ScratchPad Memory(SPM) の使用法という観点では，Miller らはソフトウェアベースの命令キャッシュを実現している [13]．彼らの目的は，アクセスあたりの消費電力が低い SPM 環境において，プログラムの容易性と過去のソフトウェア資産の移植性を実現することであり，高性能化を目的とした本稿の提案手法とは目的が異なる．

6. おわりに

数十から数百のコアを搭載するメニーコア時代においては，「如何にコアを協調動作させ，プロセッサチップが本来有する潜在能力を最大限に引き出す事ができるか？」が重要となる．現在，このような大規模 CMP を前提とし，性能だけでなく信頼性や消費電力といった様々な要求を満足できるコア間協調実行技術に関する研究を進めている．本稿では，その一環として，以前提案したメモリ貸与に基づく SPM 型 CMP 向け演算/メモリ性能バランシング法の計算/メモリ提供コア数決定法について議論・評価した．評価の結果，いくつかのパタンにより，適切なコア数を見積もることで，短時間で適切な計算/メモリ提供コア数を決定することができた．今後は，さらに精度が高く見積もり時間が短い計算/メモリ提供コア数決定方法について研究を行う予定である．

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 研究センターの諸氏に感謝します．また，実験にあたり協力を頂いた株式会社東芝セミコンダクター社に感謝します．

文　献

[1] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz and M. T. Vaden. IBM Power6 microarchitecture. *IBM Journal of Research and Development*, vol.51, No.6, Nov.2007.

[2] P. Kongetira, K. Aingarn, and K. Olukotun. Niagara: A 32-way multithreaded spark processor. *IEEE Micro*, 25(2):21-29, 2005.

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy. Introduction to the Cell multiprocessor. *IBM journal of Research and Development*, 49(4-5), Nov.2005.

[4] 林徹生，今里賢一，井上弘士，村上和彰，演算/メモリ性能バランスを考慮した CMP 向けオンチップ・メモリ貸与法の提案，情報処理学会研究報告，ARC-176，pp.59-64，2008 年 1 月

[5] Himeno Benchmark: http://accc.riken.jp/HPC/HimenoBMT/index_e.html

[6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proc. of the IEEE 4th Annual Workshop on Workload Caracterization*, pp.3-14, Dec.2001.

[7] 西川由里，鯉渕道紘，吉見真聡，天野英晴，Clear Speed 製コプロセッサの並列ベンチマークによる性能評価と性能向上手法，情報処理学会研究報告，ARC-172/HPC-109，pp257-262，2007 年 3 月

[8] 住吉正人，田辺靖貴，天野英晴，MiBench の並列化およびオンチップマルチプロセッサの評価，情報処理学会研究報告，ARC-165，pp.45-50，2005 年 12 月

[9] J. Chang and G. S. Sohi. Cooperative Caching for Chip Multiprocessors. *The 33rd Intl. Symposium on Computer Architecture*, June.2006.

[10] I. Ganusov and M. Burtscher. Efficient Emulation of Hardware Prefetchers via Event-Driven Helper Threading. *The 15th Intl. Conference on Parallel Architectures and Compilation Techniques*, Sep.2006.

[11] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. *The 39th Annual IEEE/ACM Intl. Symposium on Microarchitecture*, Dec.2006.

[12] 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. *13th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, March.2008

[13] J. E. Miller and A. Agarwal. Software-based Instruction Caching for Embedded Processors. *12th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, Oct.2006.