九州大学学術情報リポジトリ
Kyushu University Institutional Repository

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

林, 徹生
九州大学大学院システム情報科学府/科学研究院

今里, 賢一
九州大学工学部

井上, 弘士
九州大学大学院システム情報科学府/科学研究院

村上, 和彰
九州大学大学院システム情報科学府/科学研究院

https://hdl.handle.net/2324/8827

出版情報：情報処理学会研究報告, 2007-ARC-176. 2008 (1), pp.59-64, 2008-01-15. 情報処理学会
バージョン：
権利関係：ここに掲載した著作物の利用に関する注意 本著作物の著作権は（社）情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。

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

林　徹生[†]　今里賢一[††]
井上弘士[†]　村上和彰[†]

チップマルチプロセッサでは並列処理によって性能向上を実現可能である．しかしながら，プロセッサコアの処理速度に比べ主記憶へのアクセス速度は非常に遅い．また，コア間での資源共有が必要であり，主記憶アクセスがプロセッサ性能抑制の主要因となっている．したがって，プロセッサシステム全体の性能向上のためには，各コアにおける演算の並列化効率とメモリ性能の両方を向上させる必要がある．そこで本稿では，メモリ貸与法に基づく SPM 型 CMP 向けコア協調実行方式を提案する．演算，メモリ性能の向上のため，それぞれにバランスよくコア資源を分配することでトータルでの性能向上を目指す．姫野ベンチマークを Cell プロセッサに実装して評価した結果，単純な並列処理に比べて最大で 13 %の性能向上を確認した．

Execution/Memory Performance Balancing: An On-chip Memory Management Technique for High-Performance CMP

Tetsuo Hayashi,[†] Kenichi Imazato,[††] Koji Inoue[†]
and Kazuaki Murakami[†]

This paper proposes performance balancing, that is core management technique focused on trade-off between calculation and memory performance. In CMPs, high-performance is achieved by exploiting TLP. However, resource sharing among the cores makes memory performance lower regardless of the already low performance compared with processor core's one. Thus, we have to consider not only scalability, but also the performance assumed ideal memory sub-systems. Our proposed technique attempts to select effective approach, exploit scalability or improve memory performance. We also focus on a software-controllable on-chip memory. By borrowing local memory of some cores to others, we achieve memory performance improvement, and try to improve processor performance. Our experimental results show 13% speed up in the best case, compared with conventional parallel processing on Cell Broadband Engine.

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．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(CBE)[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．今後，マルチコアの概念を発展させたメニーコア・プロセッサの実用化も現実となるであろう．

一般に，CMP ではスレッドレベル並列性を活用することによりチップ単体での高い演算性能を達成する．しかしながら，オフチップ・メモリバンド幅はコア数に比例して増加しないため，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) はより深刻となる．その結果，本来 CMP が有する高いピーク演算性能を十分に発揮できない状況が発生する．

このような問題を解決するため，これまでに様々なメモリ性能向上技術が提案された．例えば，新しいキャッシュ・メモリ構成法/制御法や効率的なプリフェッチ法などがある[4),11)]．しかしながら，これらの多くはメモリ性能向上のみに重きを置いており，必ずしもプロセッサシステム全体のトータル性能を考慮している訳ではない．

そこで本稿では，与えられたハードウェア資源制約下においてこれらを最大限に有効利用し，CMP のトータル性能を向上する新しいアプローチとして「**演算/メモリ性能バランシング**」を提案する．全てのコアをスレッド実行に活用する従来の単純なチップ内並列実行方式とは異なり，チップ内に搭載された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決定し，CMP のトータル性能向上を実現する．本研究では，ソフトウェア制御可能なオンチップ・メモリを有

† 九州大学大学院 システム情報科学府/科学研究院
Graduate school / Faculty of Information Science and Electrical Engineering, Kyushu University
†† 九州大学 工学部
Faculty of Engineering, Kyushu University

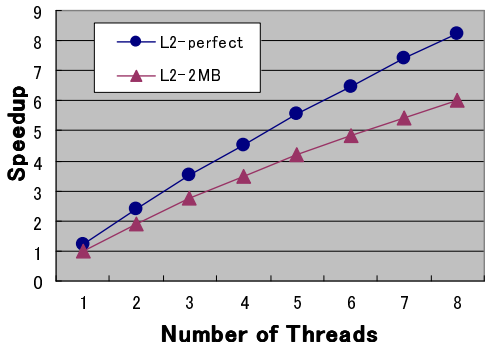

図 1　主記憶アクセスの影響：FMM

する CBE において演算/メモリ性能バランシングを適用し，性能評価を行った．その結果，最大で 13%の性能向上を達成することができた．

本稿は以下の構成である．第 2 節では提案手法である演算/メモリ性能バランシングの概要を説明する．第 3 節では，ソフトウェア制御可能オンチップ・メモリを搭載した CMP への適用を想定し，性能モデリングとそれに基づく定性的評価を行う．次に，第 4 節では提案方式の CBE への実装について説明し，第 5 節でベンチマーク・プログラムを用いた定量的評価の結果を示す．第 6 節で関連研究について述べ，最後に第 7 節でまとめる．

2. 演算/メモリ性能バランシング

第 1 節で述べたように，CMP ではメモリ・ウォール問題がより深刻となる．あるベンチマーク・プログラム (SPLASH2[5] の FMM) の実行において，メモリ性能が CMP での並列化効率に与える影響を図 1 に示す． 横軸は並列実行されるスレッド数 (コア数)，縦軸は 2MB 共有 L2 キャッシュを有する CMP(L2-2MB モデル) での逐次実行性能を基準としたスピードアップ値である．これらの実験結果はマルチプロセッサシミュレータ M5[6] を用いて採取した．全ての L2 キャッシュアクセスがヒットする L2-perfect モデルではスレッド数 (搭載コア数) に比例した性能向上を達成している．これに対し，共有 L2 キャッシュを 2MB と制限した場合には，8 個のコアを使用した場合でも性能向上は 6 倍程度に抑制されることが分かる．

ある 1 つのプログラムを並列処理する時，通常は並列化されたコードを全てのコアを用いて実行する．しかしながら，図 1 で示したように，演算とメモリの間に大きな性能差が存在する場合には，必ずしも高いトータル性能を達成できる訳ではない．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．例えば図 1 において，8 個のコアの内，7 個はスレッド実行用，1 個はメモリ性能向上用に利用したとする．もし，1 個のメモリ性能向上用コアにより全ての L2 ミスを回避できる場合，従来の 8 コアを用いた単純スレッド並列実行と比較して，高い性能向上を達成する事ができる (図 1 の L2-Perfect の 7 コア実行と同じになる)．すなわち，CMP でのプログラム実行において，

(1) 演算性能に関する並列処理効果とメモリ性能のインパクトを観測し，
(2) チップ内コアを「演算専用」と「メモリ性能向上専用」のいずれかに割り当てる．
(3) これにより，CMP の潜在能力を最大限引き出すよう演算とメモリ性能のバランスを調整する．

このように，単純な並列実行ではなく，実行対象プログラムの特徴や CMP 性能，メモリ・システムの構成などに応じてチップ内コアを適切に使い分ける事で演算/メモリ性能バランシングを実現し，CMP のトータル性能を向上する．

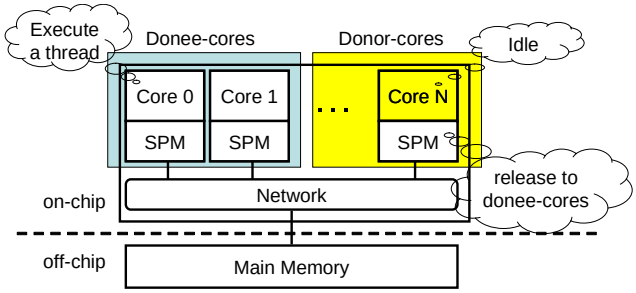

図 2　対象 CMP モデル

3. メモリ貸与に基づく演算/メモリ性能バランシング

3.1 SPM 型 CMP への適応

CMP でのオンチップ・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．本節では， SPM を搭載した CMP に着目し，メモリ貸与に基づく演算/メモリ性能バランシング法を提案する．

図 2 に提案手法において対象とする CMP のモデルを示す．各コアはソフトウェア制御可能なオンチップ・メモリを搭載しており，クロスバやリング型等のオンチップ・ネットワークで接続されている．また，各コアでの計算は，主記憶-SPM 間の DMA コントローラを用いたデータ転送と同時実行可能である．更に，前提条件として以下の 2 点を仮定する．

(1) 各コアは他コアの SPM(以降，リモート SPM) の物理アドレスを知ることが可能，かつ，アクセス可能．
(2) 計算で用いられる配列等のデータ領域の一部は，その先頭アドレスを静的に知ることが可能．

演算/メモリ性能バランシングにおいては，並列処理を担当するコアと並列処理をサポートするコアが存在する．そこで，提案手法におけるコアの振る舞いに応じて以下の 2 種類のコアを定義する．

- **Donee コア:** 並列化されたコード (スレッド) を実行する．
- **Donor コア:** スレッド実行は行わず，自コアの SPM を他コアのメモリ性能向上のために提供する．

図 2 に示すように，Donor コアは自身の SPM を donee コアに貸与する．この SPM は donee コアの SPM と主記憶間の階層メモリとして利用される．したがって，提案手法では，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オンチップの donor コアの SPM へとマッピングすることにより主記憶アクセスレイテンシの削減を期待できる．多くのコアを Donor コアとして利用することにより，スレッド実行当たりの利用可能な SPM 容量は大きくなる．しかしながら，その一方，これはスレッド・レベル並列性の活用を制限することになる．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 Donor コア数を決定する必要がある．

3.2 性能モデリングと定性的評価

本節では，SPM 型 CMP へ提案手法を適用した場合の性能モデリングを行い，その有効性を定性的に評価する．プログラムの実行時間 T_{exe} は以下の式で表される．

$$T_{exe} = T_{seq} + \frac{T_{para}}{N_{donee}} \quad (1)$$

$$= T_{seq} + \frac{T_{cal}}{N_{donee}} + T_{mem} \quad (2)$$

T_{seq}，T_{para} はそれぞれ，プログラム中の逐次処理部分，並列処理部分の実行時間である．N_{donee} は並列実行するコア数であり，式 (1) はアムダールの法則を表している．並列処理における理想的なメモリアクセスを考えると，実行時間は演算時間 T_{cal} とストール時間 T_{mem} を用いて式 (2) と表すことが出来る．従来の並列処理においては，N_{donee} は CMP の搭載コア数である．一方，提案手法の性能バランシングの性能 T_{exe_pb} は式 (3) で表現可能である．

$$T_{exe_pb} = \frac{T_{cal}}{N_{donee} - N_{donor}} + f(N_{donor}) \cdot T_{mem} \quad (3)$$

ここで，N_{donor} は donor コアの数であり，関数 f は主記憶アクセスレイテンシの削減率である．提案手法において全ての主記憶アクセスを削減できた場合 $f = 0$ となり，全く削減できなかった場合は $f = 1$ となる．式 (2,3) より，T_{mem} の削減量が並列処理における台数効果を上回った時，提案手法による性能向上が得られる事は自明である．

次に 2.1 節の仮定の下で主記憶アクセスレイテンシの削減について議論する．提案手法では，主記憶へのアクセス方法として DMA 転送を想定している．したがって，そのデータ転送時の読み込み時間 T_{read}，および，書き込み時間 T_{write} は式 (4,5) で表される．

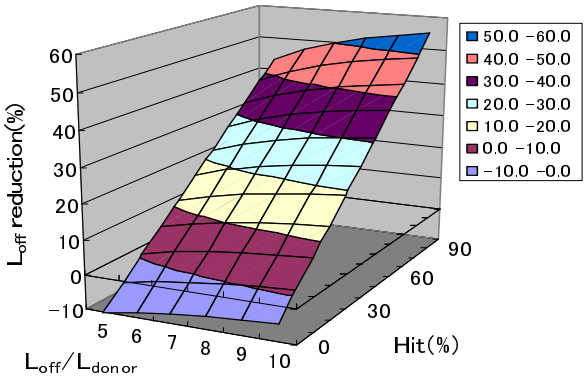

図 3 主記憶アクセスレイテンシ削減率

$$T_{read} = l_{DMA} + L_{ctrl} + L_{off} + Size_k/BW \quad (4)$$

$$T_{write} = L_{DMA} + L_{ctrl} + Size_k/BW \quad (5)$$

L_{DMA}，L_{ctrl}，L_{off} は順番に，DMA 転送の準備時間，メモリコントローラへのアクセス時間，主記憶へのアクセスレイテンシを示している．また，$Size_k$ は DMA 転送時のデータサイズであり，BW はオンチップ・バンド幅を意味する．ここで，提案手法の適用によって主記憶の代わりに donor コアの SPM にアクセスした場合，式 (4) で示した読み込み時間は式 (6) となる．

$$T'_{read} = L_{DMA} + L_{ctrl} + L_{donor} + Size_k/BW + OH_{RW} \quad (6)$$

L_{donor} は donor コアへのアクセスレイテンシであり，式ではコアの物理的な位置に関わらず一定と仮定している．また，OH_{RW} は主記憶へアクセスするか否かを判断するために必要なオーバーヘッドである．donor コアの SPM にアクセスした場合，L_{off} は L_{donor} に比べて十分大きいため，大幅なレイテンシの削減に対してこのオーバーヘッドは無視できる．一方，読み込みに関しては提案手法適用によって式 (5) は式 (7) で置き換えられる．

$$T'_{write} = L_{DMA} + L_{ctrl} + Size_k/BW + OH_{RW} \quad (7)$$

式 (7) は，DMA による書き込み時は性能改善が得られず，オーバーヘッドのみが発生することを意味する．この原因は，対象としている CMP モデルにおいては主記憶への書き込み完了を待つ必要が無く，L_{off} の項が存在しないからである．したがって，式 (4-7) より DMA 転送あたりの主記憶アクセスレイテンシの平均削減時間は式 (8) となる．

$$T_{reduce} = RW \cdot Hit(L_{off} - L_{donor}) - OH_{RW} \quad (8)$$

ここで，RW と Hit はそれぞれ，全ての DMA 転送に対する読み込みリクエストの割合，および，donor コアの SPM でデータがヒットする確率である．図 3 は RW=2/3，OH_{RW}=L_{donor}/2 と仮定した時の主記憶アクセスレイテンシの削減率を示している．L_{off}/L_{donor} の比率と donor コア上でのヒット率をパラメータとしている．例えば，関連研究[10] のワーストケース (L_{donor} が最大) でヒット率 80%を想定すると，図 3 より主記憶アクセスレイテンシを 40%削減可能である．

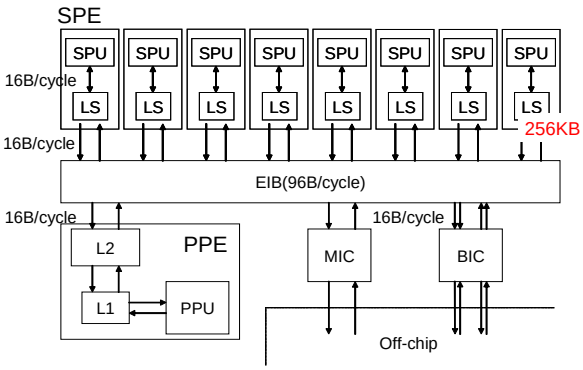

図 4 CBE のブロック図

```
for (i,j,k){
   calc();
   DMA_WRITE(A++, local); // save data
}

atomic();            // all core finish a function
for (i,j,k) {
  DMA_READ(A++ local);   // read saved data
  DMA_WRITE(B++ local); // update own area
}
atomic();            // wait for all core's finish
```

Our focusing instructions

図 5 着目する DMA 転送命令

4. CBE への実装

4.1 CBE アーキテクチャ

CBE はソフトウェア制御可能なオンチップ・メモリを採用した CMP として知られている[3] 図 4 に CBE のブロック図を示す．CBE では主に 8 個の SPE(実験で用いた実機は 7 個が動作) による並列処理によって性能向上を図っている．各 SPE は 256KB のスクラッチパッド・メモリを内部に持ち，提案手法の対象となる CMP モデルを満足する．本稿における並列処理方法は，機能分散ではなく処理分散によって実現している．また，提案手法の実現は SPE 主導で行う．

4.2 提案手法の実現方法

最初に，提案手法の実現と適用に向けて着目したプログラムコードについて図 5 を用いて説明する．CBE では計算と DMA 転送をオーバーラップして実行することが可能であるため，各コア間で同期を取る命令に着目した．同期を取った後の DMA 転送，特に読み込みの場合は主記憶アクセスレイテンシを隠蔽できないからである．したがって，提案手法の適用対象は同期後の読み込み，および，これらに関係する書き込みの DMA 転送命令とした．これらの命令のアクセス先のデータを donor コアの SPM にマッピングすることにより，主記憶アクセスレイテンシの削減を図る．

次に，コード拡張などの実装について説明する．CBE に限らず多重分岐命令は分岐予測ミスを頻発し，そのペナルティによりプロセッサ性能が低下する．したがって，提案手法の実装では，配列データを用いることで主記憶，或いは特定の donor コアの SPM へアクセスするかを判別する．また，配列データを用いることにより donee/donor コアの個数に関係なく一定の処理時間で判別可能である．図 6 は core-0 から core-3 までが donor コアである時の配列の状態を例示している． 図 6 が示すように，配列の要素 [0]-[2] に各 donor コアの SPM の物理アドレスが，[3] 以降には主記憶のアドレスが格納されている．配列の要素数は SPM サイズとデータを書き戻す領域のサイズによって決定される．また，SPM の全領域がマッピングされていないのは，CBE がフォン・ノイマン型アーキテクチャを採用しており，提案手法実現に向けて最低限のプログラムコード領域を確保する必要があるからである．したがって，配列データを用いることで donee コアは donor コアの SPM にアクセス可能となり，DMA 転送命令の offset を用いて以下のアドレスへアクセスする．

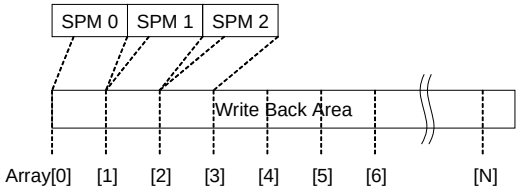

図 6 配列のイメージ

$$Address = Array[offset\%Size] + offset \qquad (9)$$

ゆえに，性能向上は図 3 の Hit，実装上では donor コア数と書き戻す領域サイズに大きく依存する．

最後に実際にプログラムを実行する時のフローを説明する．donee コアが提案手法の恩恵を得るまでには 4 つのステップがあり，以下の通りである．

(1) donor コアが起動してデータをプリフェッチする．プリフェッチが必要な理由は，初期参照や読み込みと書き込みデータサイズが違う時のデータ整合性を保証するためである．
(2) donee コアが起動して donor コアの SPM の物理アドレス情報を得る．
(3) donee コアが図 6 で示した SPM 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．この際，静的に donor コア数などのパラメータを設定する必要がある．
(4) donee コアは donor コアの SPM を活用しながら並列処理を行なう．

全ての donee コアの処理が終了した後，donor コアの SPM 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．

5. 性能評価

5.1 準　　備

CBE 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．評価には東芝の Cell Reference Set(CRS) を用いた．CRS では SPE

表 1 姫野ベンチマーク・プログラムにおける入力サイズ

size	$i \times j \times k$
SS	$33 \times 33 \times 65$
S	$65 \times 65 \times 129$
M	$129 \times 129 \times 257$

数は 7 個であるため，式 (1〜3) の N_{donee} は 7 である．以下に評価モデルを示す．

- **CONV:** 従来の，並列化されたコード部分を全コアを用いて並列処理するモデル．$N_{donee} = 7$
- **IDEAL:** 主記憶アクセスレイテンシを 0 と仮定した理想的なモデル．実装上では DMA 転送サイズを 0 として転送完了を待たずに次の処理に移る．このモデルでは理想的な性能を見積もることを目的とし，プログラム実行の正しさを保証しない．
- **PB-N:** donee コア数を N とした時の提案手法を示すモデル．したがって，残りのコア ($= 7 - N_{donee}$) は donor コアとなる．例えば，PB-5 というモデルでは 5 コアが並列処理を行ない，残りの 2 コアの SPM を階層メモリとして利用する．

評価対象プログラムとして，本稿では姫野ベンチマーク・プログラム[7] を用いた．このプログラムでは複数の 3 次元，4 次元の配列を用いて計算を行なう．主要な計算部分は反復回数を含めて 4 重ループの構成を持ち，最内ループにおいて一部の 3 次元データは各次元に対するデータ合計 19 個を用いる．CBE への実装では反復回数を一定とし，過去の報告[8] を参考に 2 番目のループレベルの粒度で並列化した．また，最内ループ部分でバッファリングを適応した以外に特別なコードチューニングは行なっていない．表 1 は評価に用いた入力サイズを示している．並列化したプログラムのコンパイルオプションには-O3 を用い，“time.h“を利用して実行時間を 10 回計測して最大/最小値を除いた平均値を代表値として選出した．

5.2 結　　果

実験結果を図 7 に示す．CBE での実行時間の逆数を性能とし，性能は CONV を基準とした正規化実行性能を示している．

最初に，入力データサイズ SS の時の PB-6 と CONV に着目する．図 7 が示すように，この場合は提案手法適用により 13%の性能向上を得ていることが分かる．また，この性能向上は IDEAL が示す上限に非常に近いことも分かる．このような良い結果が得られた理由としては，donor コアの SPM 上でのヒット率，つまりは図 3 や式 (8) の Hit の値が非常に高いことが挙げられる．入力データと評価モデル別の Hit の値を表 2 に示す．表 2 より，提案手法の対象となる DMA 転送命令で発生した主記憶アクセスの 70%が donor コアの SPM へとアクセス先を変更されていることが分かる．次に，提案手法のオーバーヘッドを議論するために PB-7 と CONV を比較する．PB-7 では CONV と同様に並列処理をする一方，donor コアを用いないので，式 (8) で示した DMA 転送のアドレスを判別するオーバーヘッド OH_{RW} のみが発生する．このオーバーヘッドによりプロセッサ性能が低下してしまう可能性がある．しかしながら，図 7 の結果が示すとおり，PB-7 と CONV のオーバーヘッドによる性能差は高々1%である．この事は OH_{RW} を無視できることを示している．

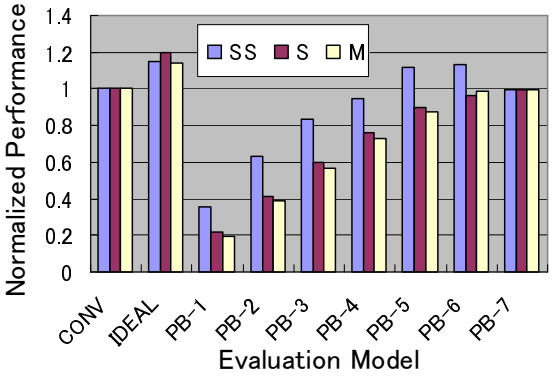

図 7 正規化実行性能

表 2 モデル別のヒット率 (%)

input	CONV	PB-1	PB-2	PB-3	PB-4	PB-5	PB-6	PB-7
SS	0	100	100	100	100	100	70	0
S	0	60	50	40	30	20	10	0
M	0	7.8	6.5	5.2	3.9	2.6	1.3	0

次に，入力データサイズ SS の時のその他のモデルについて提案手法の効果を議論する．表 2 により，PB-1, PB-2, PB-3, PB-4 では対象となる DMA 転送命令は全て donor コアの SPM へとアクセスしている．しかしながら，これらのモデルにおいては CONV に比べて性能が低下している．この原因としては，主記憶アクセスレイテンシを削減することによってメモリ性能は大幅に向上する一方，それ以上に並列処理能力が低下して全体としての性能が低下したと考えられる．PB-5 は性能向上が得られているが，ヒット率が低い PB-6 の方が高性能である．したがって，SS サイズにおいて最も効果的な性能バランシングは，演算に 6 コアを使用してメモリ性能向上に 1 コアを使用した時だと分かる．

最後に，提案手法適用による悪影響について述べる．入力データサイズが S, M の時，PB-N の各モデルで CONV に比べて性能が低下している．この主要因はヒット率の低さにある．書き戻すデータ領域のサイズに比べて donor コアの SPM にマッピング可能なデータ量が圧倒的に少ないため，表 2 が示すような低いヒット率になっている．例えば，SS サイズで最も高性能であった PB-6 に着目すると，S サイズで 10%，M サイズで 1.3%となり性能向上への貢献が小さい．したがって，同じアプリケーションプログラムであっても，提案手法の効果は入力データ等に大きく依存することが分かる．

6. 関連研究

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[9]．実ハードウェアに実装して評価している．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．

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[10)~13)]．Cooperative cache[11] 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[12]．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．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，donee/donor 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[13]．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．

7. 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，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を提案した．また， CBE に本手法を適用した結果，最大で 13%の性能向上を得る事ができた．

本研究では，演算/メモリ性能バランシングのための Donee コア数/Donor コア数決定法は議論していない．今後，静的 (コンパイル時など) または動的 (プログラム実行時) な Donee/Donor 数決定法を確立する予定である．また，MiBench 等の他のベンチマーク・プログラムを用いて提案手法の効果を評価・解析する．

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．なお，本研究は一部，科学研究費補助金若手研究 A(課題番号 17680005)、ならびに、文部科学省「次世代 IT 基盤構築のための研究開発」，研究開発領域「将来のスーパーコンピューティングのための要素技術の研究開発」(平成 17 年度〜19 年度) における研究開発課題「ペタスケール・システムインターコネクト技術の開発」による．

参 考 文 献

1) H.Q.Le et al. 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 et al. Introduction to the Cell multiprocessor. *IBM journal of Research and Development*, 49(4-5), Nov.2005.
4) M.K.Qureshi et al. Adaptive Insertion Policies for High-Performance Caching. *In Proc. of the 34th Intl. Symposium on Computer Architecture*, June.2007.
5) G.E.Suh et al. The SPLASH-2 programs: Characterization and methodological considerations. *Proc of the 22th Intl. Symposium on Computer Architecture*, pages24-36, June.1995.
6) N.L.Binkert et al. Network-oriented full-system simulation using M5. *In Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, Feb.2003.
7) Himeno Benchmark: http://accc.riken.jp/HPC/HimenoBMT/index_e.html
8) 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
9) 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.
10) M.Zhang and K.Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. *The 32nd Intl. Symposium on Computer Architecture*, June.2005.
11) J.Chang and G.S.Sohi. Cooperative Caching for Chip Multiprocessors. *The 33rd Intl. Symposium on Computer Architecture*, June.2006.
12) 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.
13) 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.