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

2008－ARC－176 (10)
2008－EMB－ 7 (10)
2008／1／15

多機能メニーコアにおけるデータ供給を支援するキャッシュコアの提案

森谷 章[†1] 藤枝直輝[†1]
佐藤真平[†1] 吉瀬謙二[†1]

チップ・マルチプロセッサにおけるコアの数が増加する傾向にある．しかし，チップ内のコアが増加するとメインメモリへのアクセス集中などによる通信オーバーヘッドへの悪影響が懸念される．本稿ではメニーコア・プロセッサにおけるコアの利用法として，データ供給を支援する方式を提案する．また，その重要な機能の１つとしてソフトウェア実装によるキャッシュを持つキャッシュコアを提案する．Cell/B.E. の SPE にキャッシュコアを実装し，いくつかのプログラムを用いて性能を評価する．さらに，Cell/B.E. 機能レベルシミュレータ SimCell を用いてコア間の通信レイテンシサイクルを変化させた場合の評価を行い，キャッシュコアの可能性を検討する．

The proposal of Cache Core which supports data supply on multifunctional many-core processors

Akira Moriya,[†1] Naoki Fujieda,[†1] Shimpei Sato[†1]
and Kenji Kise[†1]

The number of cores in chip multi-processors tends to increase. However, when the number of cores increase we suffer from the negative impact on communication latancy by centralized access to the main memory. In this article, we propose new usage of a core supports data supply in many-core processors. As one of the important feature, we propose the Cache Core which is software-implemented cache. We implement the Cache Core on SPE of Cell/B.E. and evaluate its performance with some programs. In addition, We evaluate performance of the Cache Core by changing communication latancy on Cell/B.E. Functional Simulator SimCell.

1. はじめに

プロセッサの性能向上かつ低消費電力化を目指して，チップに複数のプロセッサコアを集積して，スループットの向上やスレッドレベルの並列性を利用するチップ・マルチプロセッサが注目されている．チップ・マルチプロセッサでは，アプリケーションの並列性を高め１チップ上に複数配置したコアにうまくタスクを分配することで処理の高速化が見込める．従って，チップ・プロセッサ上のコアの数は現在主流である 2, 4 などから数 10 以上であるメニーコアへと増えていくと考えられる．

しかしこの方式による高速化では，コアの数の増加と共にアプリケーションのさらなる並列化の必要が伴うことになる．プログラム内にコア数に見合った並列性が見出せなければ，期待される速度で処理を行うことはできない．また，現段階ではコア数の増加と共に線形に性能が向上せず，途中で性能の上昇が止まってしまう傾向にある．これはメインメモリや他コアとの通信がボトルネックになってしまうためである．

そこで，メニーコア・プロセッサにおけるコアの利用法[1] として他のコアへのデータ供給を支援するコアを提案する．そして，その機能の１つとして**キャッシュコア**の可能性を評価する．キャッシュコアとは，ハードウェアのキャッシュと同じ動作を行うソフトウェアキャッシュプログラムを実行し，メニーコア・プロセッサにおいて他コアのデータ・キャッシュとして働くコアである．キャッシュコアは自分の持つ資源を他コアのキャッシュとして提供する．実際のプログラムを実行するコア（**計算コア**）はキャッシュコアに対してデータアクセスを行い，通信レイテンシの大きいメインメモリへのアクセスを減少させ処理速度の増加を目指す．

本稿では，ハードウェアのキャッシュをもたない Cell/B.E. の SPE に計算コアとキャッシュコアを実装する．機能レベルの Cell/B.E. プロセッサシミュレータ **SimCell**[2] を用いてコア間のレイテンシを変化させてアプリケーションの処理速度を評価し，キャッシュコアの可能性を検討する．

本稿の構成を述べる．2 章ではメニーコア・プロセッ

†1 東京工業大学
Tokyo Institute of Technology

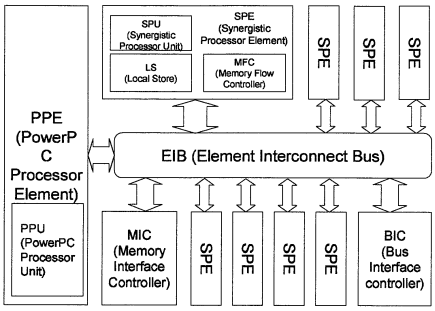

図 1 チップ・マルチプロセッサ Cell/B.E. の構成.

サにおけるデータ供給支援コアを提案する．3 章ではキャッシュコアの設計，実装について述べる．4 章では Cell/B.E.(PLAYSTATION 3) を用いたキャッシュコアの評価を行う．5 章では SimCell においてコア間の通信レイテンシを変化させた場合の実行について評価する．6 章で本稿をまとめる．

2. データ供給支援コアの提案

本研究ではメニーコア・プロセッサモデルとしてハードウェアのキャッシュを持たないチップ・マルチプロセッサ Cell/B.E. を使用する．

図 1 に示すように，Cell/B.E. はヘテロジニアスなチップ・マルチプロセッサである．Cell/B.E. には PPE と呼ばれる PowerPC アーキテクチャの汎用プロセッサ，SIMD 型の独自のアーキテクチャを採用する 8 個の Synergistic Processor Element(SPE) と合計 9 個のコアが搭載されている．SPE はローカルストア (LS) と呼ばれる 256KB の専用メモリを持つ．SPE がデータを処理するためには LS に予め DMA 転送によりロードしておく必要がある．

メインメモリへのアクセスは大きなレイテンシを要し，SPE にとっては大きなオーバーヘッドとなる．また，複数のコアからメインメモリへアクセスが集中した場合にはさらに大きなレイテンシが生じ，各コアの実行に多大な悪影響を及ぼす恐れがある．

本研究では，メニーコア・プロセッサにおけるコアの利用法として，チップ上の他コアへのデータ供給を支援するコアを提案する．そして，その機能の 1 つとしてキャッシュコアの可能性を検討する．メニーコア・プロセッサのようにコア数が数 10 ともなると 1 つのプログラム内にコア数に見合った並列性を実現することは困難である．従って，チップ上に存在するコア全てに処理をさせるのではなく計算コアへデータ供給を支援するコアを形成する．

一般に，チップマルチ・プロセッサにおいてコア間の通信レイテンシはメインメモリの通信レイテンシより小さい．計算コアがキャッシュコアへアクセスすることでデータアクセス時のレイテンシの軽減を目指す．加えて，キャッシュのヒット率を向上させることでメインメモリへのアクセスの回数を減少させる．

データ供給支援コアによる利点を以下に示す．

- ソフトウェアによってキャッシュを実装し，データアクセスのレイテンシを削減し，メインメモリへのアクセスを減少させる．
- プリフェッチによりソフトウェアキャッシュの性能を向上させる．
- 明示的に割り当てられたバッファを搭載し，ソフトウェアキャッシュを超える高速化を図る．ソフトウェアキャッシュではタグ計算やラインのリプレースなどが必要なためオーバーヘッドが大きい．この処理を省略して高速化を図る．
- 複数のコアを統合して利用することによる機能の強化を実現する．例として N 個のコアを用いて N-Way セットアソシアティブ方式のソフトウェアキャッシュを実現する．
- これらの機能はソフトウェアによって実現されるため，実行するアプリケーションに適した構成へと動的に変更することで，効率化を達成する．

本稿では，データ供給支援コアの機能の初期検討として，データキャッシュとして動作するキャッシュコアの性能を評価する．SPE の 1 つを計算コア，もう 1 つをキャッシュコアとして実装・評価する．PPE には SPE スレッドの呼び出し以外の処理は割り当てない．

3. キャッシュコアの設計と実装

3.1 キャッシュコアの設計方針

本研究ではプラットフォームに Cell/B.E. を用いているが，キャッシュコアは汎用的なメニーコア・プロセッサ上での使用を目指して設計している．従って他のプロセッサでは搭載されていない Cell/B.E. 特有の命令やテクニックなどは用いない．また，キャッシング方式は処理オーバーヘッドをなるべく少なくするため，ダイレクトマップ方式を採用する．計算コアに搭載する L1 キャッシュ，キャッシュコアに搭載する L2 キャッシュはそれぞれエントリ数とラインサイズを変更することが可能である．アプリケーションはデータアクセスにおいて自分のコア内の L1 キャッシュを参照し，L1 キャッシュはキャッシュコアの L2 キャッシュを参照する．従って，L1 キャッシュと L2 キャッシュのライン間で DMA 転送が生じるので，DMA 転送の簡単化のため L1 キャッシュと L2 キャッシュのラインサイズは等しいものとする．

リプレース方式は L1 キャッシュではライトバック，L2 キャッシュではライトスルー方式を採用する．これはそれぞれの実装の実行速度の測定結果による．

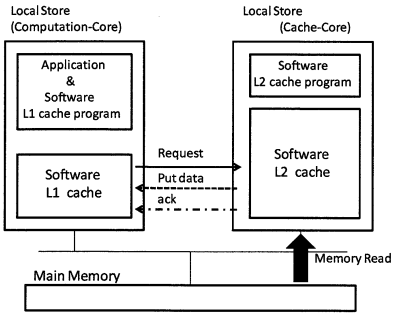

図 2 Hand-shake 方式における読み出し時の処理の流れ.

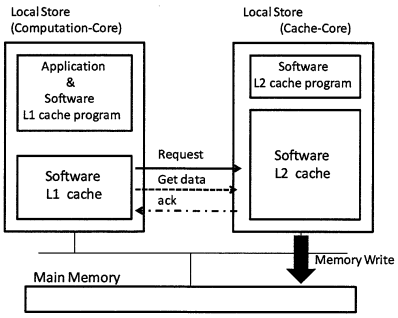

図 3 Hand-shake 方式における書き込み時の処理の流れ.

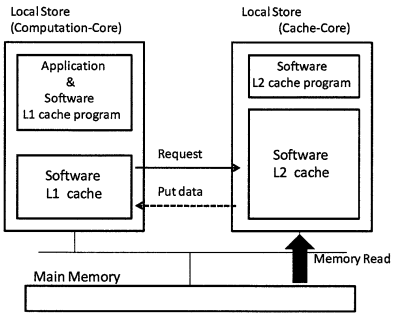

図 4 valid-packing 方式における読み出し時の処理の流れ.

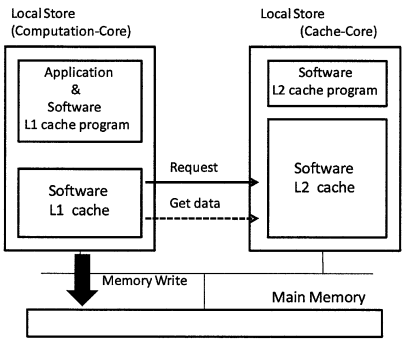

図 5 valid-packing 方式における書き込み時の処理の流れ.

3.2 キャッシュコアの設計

計算コアには，処理するアプリケーションと同時に L1 キャッシュとして動作するソフトウェアキャッシュのプログラムを実行させる．キャッシュコアにはソフトウェアキャッシュのプログラムのみを実行させ，L2 キャッシュとして使用する．

計算コアに加えてキャッシュコアを使用してプログラム処理をさせる場合，オーバーヘッドと LS の利用量のトレードオフから，コア間のデータ転送と同期の方法に 2 つの方式が考えられる．以下にそれぞれの特徴を述べる．

Hand-shake 方式　本方式におけるデータの読み出し時，書き込み時の処理をそれぞれ**図 2**, **図 3** に示す．計算コアからキャッシュコアへ要求が送られるとキャッシュコアはデータ転送した後に，ack を転送をする．計算コア，キャッシュコアはそれぞれ L1 キャッシュ，L2 キャッシュに加え，データ要求や ack 転送用の構造体を持つ．

読み出しではまず，計算コアが読み出し要求をキャッシュコアに DMA 転送する (Request)．キャッシュコアは要求を受け取り，L2 キャッシュにヒットするとデータを L1 キャッシュに DMA 転送する (Put data)．ミスした場合はメインメモリを参照する．データ転送が終了するとキャッシュコアは計算コアの構造体に ack を DMA 転送する (ack)．計算コアは ack を受け取ると処理を再開する．

書き込み時は，計算コアが書き込み要求をキャッシュコアに DMA 転送する．キャッシュコアは要求を受け，計算コアからデータを受け取る．次に受け取ったデータをメインメモリへ書き戻す．処理が終わると計算コアに ack を DMA 転送する．計算コアは ack を受け取ると処理を再開する．

Valid-packing 方式　データ要求用の構造体を持つが，キャッシュラインの最後に valid ビットを付加する方式である．計算コアからの要求は Hand-shake 方式と同じくキャッシュコアの構造体に対する DMA 転送が必要だが，キャッシュコアからは 1 回の DMA 転送で済む．

図 4, **図 5** にそれぞれ本方式におけるデータの読み出し時，書き込み時の処理を示す．読み出しでは，計算コアが読み出し要求をキャッシュコアに DMA 転送する．このとき，L1 キャッシュの valid ビットを無効にしておく．キャッシュコアは要求を受け取り，L2 キャッシュにヒットすると該当ラインの最後に valid ビットを付加して L1 キャッシュに DMA 転送する．ミスした場合はメインメモリを参照する．L2 キャッシュの valid ビットが L1 キャッシュの valid ビットの領域

に最後に転送されるので，Hand-shake 方式における ack の DMA 転送が不要となる．計算コアは L1 キャッシュの valid ビットが有効になると処理を再開する．

書き込み時は，計算コアが書き込み要求をキャッシュコアに DMA 転送する．それと同時に，メインメモリへの書き込みも行う．計算コアはキャッシュコアの処理とは無関係に処理を再開する．キャッシュコアは要求を受け取り，次に L1 からデータを受け取り，処理を完了する．

Cell/B.E. アーキテクチャにおける DMA 転送は 16 バイト以上の転送では，16 バイトの定数倍の単位でしか転送できない．従って，ラインサイズを 128 バイトにした場合は次に指定できる転送サイズは 144 バイトとなる．しかし，Cell/B.E. の DMA 転送は 128 バイト単位で転送を行うのが最も速い．従って，144 バイトを転送するのであれば 256 バイト転送する方が高速になる．このため，valid ビットに使用する 1 ビットのために 128 バイト使用する実装とした．ラインサイズの約半分は使用しない無駄な領域となってしまう．

Hand-shake 方式では，1 回のキャッシュ処理における DMA 転送の回数が多いので通信のオーバーヘッドが大きい．しかし，ラインサイズが倍必要である valid-packing 方式に比べて無駄なメモリ領域が少ないという利点がある．Valid-packing 方式では，DMA 転送の回数を削減することで Hand-shake 方式の高速化を実現できるという利点がある．

3.3 アプリケーションのプログラミングモデル

本研究において用いる Cell/B.E. のアプリケーションプログラムは，汎用のプロセッサとできるだけ同様の記述ができることを目指す．メインメモリ上のデータを 1 つの配列と見なし，1 要素のデータアクセスに対して 1 回の DMA 転送を行う．但し，ソフトウェアキャッシュを使用する場合には，DMA 転送を省略できる場合がある．このような実装のため，Cell/B.E. 用に実装されたプログラムと比較すると極端に性能が低下する．

3.4 キャッシュコアの実装

本稿では実行速度を考慮して Valid-packing 方式を採用する．**図 6** にキャッシュコアにおけるソフトウェアキャッシュのプログラムの主要部の疑似コードを示す．

1 行目から 7 行目の cache_main() がキャッシュコアのメインループである．構造体のフラグが READ になれば read() を呼び出し，WRITE になれば write() を呼び出す．計算コアはアプリケーションの処理が完了すると最後にキャッシュコアに対して FINISH のフラグを転送する．これを検出するとキャッシュコアの処理を終了する．

read() では計算コアからフラグと共に転送されるデータのアドレスからタグとインデックスを計算し，ヒットかどうかを調べる．14 行目から 18 行目でキャッシュミスの場合の処理を行う．20 行目から 22 行目でデータを計算コアへ送り，フラグを VALID に戻し処理を完了する．

```
1  cache_main()
2  {
3    while (cache_str.flag != FINISH) {
4         if (cache_str.flag == READ)   read();
5         if (cache_str.flag == WRITE) write();
6    }
7  }
8
9  read()
10 {
11   tag = (cache_str.addr >> TAG_SHIFT);
12   idx = (cache_str.addr >> IDX_SHIFT) & I_MASK;
13
14   if (cache_tag[idx] != tag) {
15        cache_tag[idx] = tag;
16        DMA(cache[idx], main_mem_addr, GET);
17        wait_dma();
18   }
19
20   DMA(cache[idx], L1cache_addr, PUT);
21   cache_str.flag = VALID;
22   wait_dma();
23 }
24
25 write()
26 {
27   tag = (cache_str.addr >> TAG_SHIFT);
28   idx = (cache_str.addr >> IDX_SHIFT) & I_MASK;
29
30   DMA(cache[idx], L1cache_addr, GET);
31   cache_str.flag = VALID;
32   cache_tag[idx] = tag;
33   wait_dma();
34 }
```

図 6 キャッシュコアの主要部の疑似コード．DMA,wait_dma() の部分は実際には DMA 転送のコードが記述してある．

write() では read() と同様にタグとインデックスを計算し，計算コアから転送されるキャッシュアドレスを用いて L1 キャッシュのデータを受け取る．

4. Cell/B.E. における評価[*1]

4.1 検証プログラム

キャッシュの効果を検証するプログラムは，計算コア上に L1 のソフトウェアキャッシュを実装する版 (利用するコアは 1 個) と，上の版に加えて，キャッシュコアを L2 キャッシュとして利用する版 (利用するコアは 2 個) の 2 種類を用いる．L1 キャッシュのみを使用する版ではデータアクセスはコア内の L1 キャッシュを参照し，ミスが発生した場合はメインメモリを参照する．L1 キャッシュの容量は共通して 2KB である．L2 キャッシュの容量は 64KB である．

3.3 節におけるプログラミングモデルのデータアクセスに対してソフトウェアキャッシュを導入する．また，L1，L2 それぞれのキャッシュプログラムは API として実装するため，各アプリケーションのプログラマはデータアクセスに対して意識する必要がない．

なお，アプリケーションにはクイックソート，行列

[*1] 実機として PLAYSTATION 3 を使用

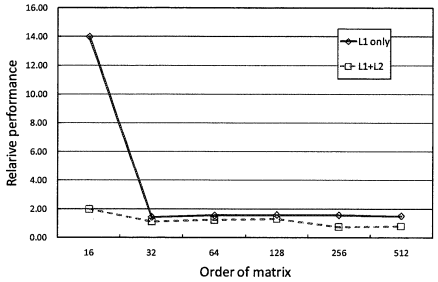

図 7 実機における行列積演算のキャッシュを用いない版に対する相対性能.

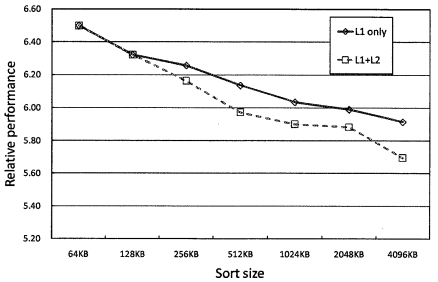

図 9 実機におけるクイックソートのキャッシュを用いない版に対する相対性能.

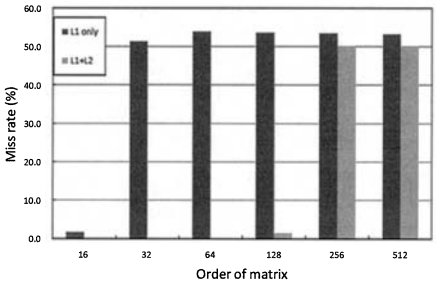

図 8 実機における行列積演算のキャッシュミス率.

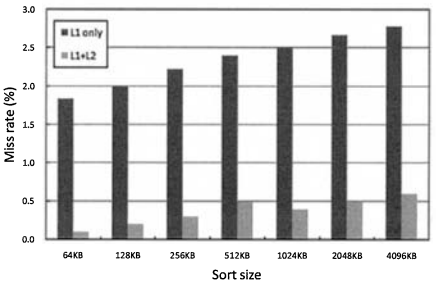

図 10 実機におけるクイックソートのキャッシュミス率.

積演算を用い，計算コア，キャッシュコアの数はそれぞれ 1 とする．

4.2 ソフトウェアキャッシュの効果

図 7〜図 10 に実機における各アプリケーションの実行結果を示す． 各図の横軸は各アプリケーションプログラムにおいて処理する要素の大きさを示す．図 7, 図 9 における縦軸はキャッシュを用いない版を基準とする相対性能を示す．図 8, 図 10 における縦軸はキャッシュミス率を示す．L1 キャッシュのみを用いる版と，L1 と L2 キャッシュを用いる版の性能を示す．

各アプリケーションにおいて，L1 キャッシュによる速度向上の効果は大きい．また，キャッシュコアを使用することで，キャッシュヒット率は向上する．すなわち，キャッシュコアの利用によりメインメモリのアクセスが減少し，アクセス集中によるレイテンシの増大防止が見込める．

キャッシュコアを用いる場合，L1 キャッシュのみを用いる版に比べ，性能は低下してしまう．しかしながら，クイックソートにおいてはキャッシュを使用しない版と比べ約 5.7 倍の速度向上を得る．

キャッシュコアの使用による速度低下には次の 2 つが原因として挙げられる．1 つ目は，キャッシュ処理のオーバーヘッドである．つまり，キャッシュヒットによる恩恵よりもソフトウェアキャッシュの構造体などのデータを操作するオーバーヘッドが大きい．2 つ目は，コア間における通信レイテンシである．Cell/B.E. ではコア間のレイテンシは数百サイクルとそれほど小さくない．このため，数回の通信を必要とするキャッシュコアのオーバーヘッドは大きくなる．

5. SimCell を用いた評価

先の章で述べた通り，通信のレイテンシが大きいことがキャッシュコアによる性能向上の妨げとなっている．従って，本章の実験では機能レベルシミュレータ SimCell を用いて通信レイテンシを変化させてキャッシュコアの可能性を検討する．SimCell では DMA 転送によるメインメモリのリードとライト，コア間の通信レイテンシ，IPC などがそれぞれパラメータとして指定できる．設定できるレイテンシとは，DMA 転送命令を発行してからデータの転送が完了するまでのサイクル数である．

5.1 SimCell の IPC を求める実験

SimCell の想定する構成と実際の構成は異なるため，アプリケーションによって実行サイクル数に差が生じる．しかし，IPC を調節することである程度まで，差を縮めることができる．

L1 キャッシュのみを用いる版を使って適切な IPC を調べる．SimCell におけるメモリアクセスのレイテンシは文献 3) の値を参考に，リードを 500，ライトを 300 と設定する．SimCell が算出する実行時間は，IPC と処理に要したサイクル数を用いて計算したもの

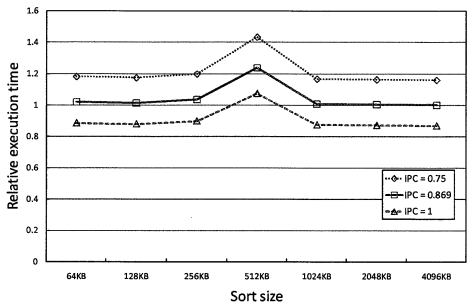

図 11 Cell/B.E. 及び SimCell によるクイックソートの IPC による相対実行時間.

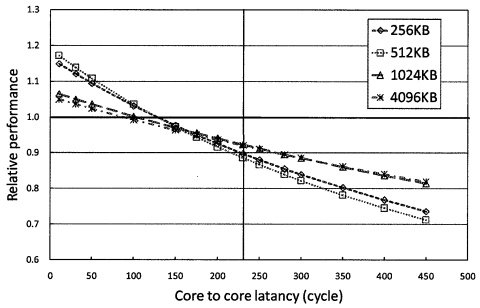

図 12 ソフトウェア L1 キャッシュのみの性能に対するキャッシュコアを追加した場合の相対性能.

である.

図 11 に，実機で実行した場合を基準とする，SimCell で IPC のパラメータを変更して実行する場合の相対性能を示す．横軸はソートするデータのサイズを示す．この図に示した IPC は経験則により求めたものである．IPC を 0.869 とした時に実機とほぼ同様の実行時間が得られる．従って，以降の実験において IPC のパラメータは 0.869 とする.

5.2 キャッシュコアの可能性の検討

キャッシュコアを利用する版を用い，キャッシュコアの効果を検証する．検証するアプリケーションはクイックソートを用いる．**図 12** に，ソートサイズ毎に SimCell においてコア間レイテンシを変更して実行した結果を示す．SimCell における IPC 等のパラメータは前節と同様である．横軸はコア間の通信レイテンシを表し，縦軸はキャッシュコアを使用しない版を基準とする相対性能を表す．なお，図中の縦線は実機におけるコア間レイテンシを表す．文献 3) によると，Cell/B.E. のコア間レイテンシは約 230 サイクルである.

図 12 より，コア間のレイテンシが約 100 サイクル以下の場合にキャッシュコアによる速度向上を得る．また，コア間のレイテンシが 10 と小さい場合には約 20 %の速度向上を達成する．このように，コア間レイテンシサイクルが 100 であるメニーコア・プロセッサではキャッシュコアの利用によりメインメモリへのアクセスの減少という利点と共に速度向上が期待できる.

図 12 より，ソートサイズが小さいほどコア間レイテンシの影響を大きく受けることがわかる．これは，計算コアにおいてクイックソートの処理よりもキャッシュコアとの通信の処理が支配的であると考えられる.

6. おわりに

本稿ではメニーコア・プロセッサにおけるコアの使用法として，他コアへのデータ供給を支援する方式を提案した．また，ソフトウェアキャッシュを実現するキャッシュコアを提案した．キャッシュコアを搭載したプログラムの Cell/B.E. における実行の評価と，Cell/B.E. 機能レベルシミュレータ SimCell を用いてコア間レイテンシを変化させた場合の実行の評価を行った．実験結果より，キャッシュコアの可能性が確認できた．クイックソートの評価では，コア間の通信レイテンシが約 100 サイクル以下で速度向上が見られる.

本稿ではデータを示していないが，キャッシュコアを使用しキャッシュのヒット率を向上させることで，メインメモリへのアクセスの大幅な削減が見込める．今後，データ供給支援コアとしての様々な機能を実装し，多くのアプリケーションにおける速度向上を図る.

加えてキャッシュコアの方式をアプリケーションに適した構成に動的に変更する，キャッシュコアを複数使用する場合の効果などの検討も行う.

謝辞 本研究の一部は，科学研究費補助金若手研究 (B) 課題番号 18700042「投機技術を積極的に利用するチップマルチプロセッサに関する研究」の助成による.

参考文献

1) 小林良太郎, 吉瀬謙二: 多機能メニーコアを実現するアーキテクチャ技術 Feature-Packing の構想, 情報処理学会研究報告 2007-ARC-175, pp. 11-15(November 2007)
2) 佐藤真平, 藤枝直輝, 田原慎也, 吉瀬謙二: 実用的かつコードのシンプルさを追求した Cell BE の機能レベルシミュレータ SimCell の設計と実装, コンピュータシステム・シンポジウム ComSys2007(2007).
3) Kistler, M., Perrone, M. and Petrini, F.: Cell Multiprocessor Communication Network: Built for Speed, *IEEE micro*, Vol.26, No.3, pp.10-23(2006).
4) 佐藤芳紀, 神酒 勤: Cell Broadband Engine への SPE ソフトウェアデータキャッシュの実装, 情報処理学会研究報告, HPC-110, pp.13-18(2007).