Vol.2011-ARC-196 No.3
2011/7/27

DMA ベースメニーコアにおける通信オーバーヘッド削減手法

高前田（山崎） 伸也[†1,†2]　吉　瀬　謙　二[†1]

Cell/B.E のように，小規模なローカルメモリを持つアーキテクチャが提案されている．このようなアーキテクチャにおいては，ソフトウェアが明示的にコア間・コア-メモリ間の DMA 転送命令を用いてデータ共有を行うことにより，データ共有を行う．しかしながら，データ転送と演算の両方を考慮したの効率的なプログラム並列化はプログラマにとって大きな負担となる．加えて，並列化ができたとしても予測不可能なデータ通信レイテンシにより，並列化の恩恵を十分に受けることができない懸念がある．本稿では，DMA 命令により明示的にコア間およびコア-メモリ間のデータ転送を行うメニーコアアーキテクチャを対象に，DMA 転送のオーバーヘッドを削減する手法，特に DMA 転送元における通信と処理のオーバーラッピングをサポートするハードウェアの追加を提案する．各コアのローカルメモリとロードストアユニットの間にストアバッファを挿入し，ゼロコピー DMA 転送とロードストアの間のデータハザードを回避しながら，DMA 転送元の積極的な命令実行を支援する．また，ストアバッファが飽和することによる命令実行の停止頻度を削減するために，DMA 転送対象のローカルメモリ範囲を監視する機構を追加することで，DMA 転送を完了した範囲に対するストア結果を格納するエントリーを早期に解放する．ソフトウェアシミュレータとマイクロベンチマークを用いた評価では，64 ノード構成で最大約 19%の性能向上を確認したが，ほとんど性能向上が見られないものもあった．

†1 東京工業大学 大学院情報理工学研究科
†2 日本学術振興会 特別研究員 (DC1)

1. はじめに

1 チップに複数のコアを搭載するマルチコアプロセッサはスーパーコンピュータから組み込み機器まで，幅広い範囲で用いられている．更なる半導体プロセスに進歩および三次元実装技術に発展により，さらに多くののコアを搭載するメニーコアプロセッサの登場が期待されている．しかしメニーコアプロセッサにおいて高い性能を維持するには，メモリウォールに対する解決策が必要である．メモリウォールに対抗する一つの手段として，Cell/B.E のように，各コアが従来のキャッシュではなく，小規模なローカルメモリを持つアーキテクチャが提案されている．このようなアーキテクチャにおいては，ソフトウェアが明示的にコア間・コア-メモリ間の DMA 転送命令を用いてデータ共有を行うことにより，メモリバンド幅およびオンチップネットワークのバンド幅を効率的に利用することが可能となる．しかしながら，データ転送と演算の両方を考慮したの効率的なプログラム並列化はプログラマにとって大きな負担となる．加えて，並列化ができたとしても予測不可能なデータ通信レイテンシにより，並列化の恩恵を十分に受けることができない懸念がある．

本研究では，DMA 命令により明示的にコア間およびコア-メモリ間のデータ転送を行うメニーコアアーキテクチャを対象に，DMA 転送のオーバーヘッドを削減する手法について検討する[1)]．特に本稿では，DMA 転送元における通信と処理のオーバーラッピングをサポートするハードウェアの追加を提案する．各コアのローカルメモリとロードストアユニットの間にストアバッファを挿入し，ゼロコピー DMA 転送とロードストアの間のデータハザードを回避しながら，DMA 転送元の積極的な命令実行を支援する．また，ストアバッファが飽和することによる命令実行の停止頻度を削減するために，DMA 転送対象のローカルメモリ範囲を監視する機構を追加することで，DMA 転送を完了した範囲に対するストア結果を格納するエントリーを早期に解放する．

本稿の構成を次に示す．2 章では，本稿の対象としている DMA ベースのメニーコアアーキテクチャについて述べる．3 章では，提案手法である，ストアバッファと DMA 転送エリア監視ハードウェアの追加による，DMA 転送元の先行実行機構について述べる．4 章では，提案手法の効果をいくつかのマイクロベンチマークを用いて評価した結果について述べる．5 章では，関連研究について述べる．最後に 6 章で本稿をまとめる．

2. DMA ベースメニーコアアーキテクチャ

近年，階層型キャッシュを介して単一のメモリ空間を共有する，マルチコアプロセッサが

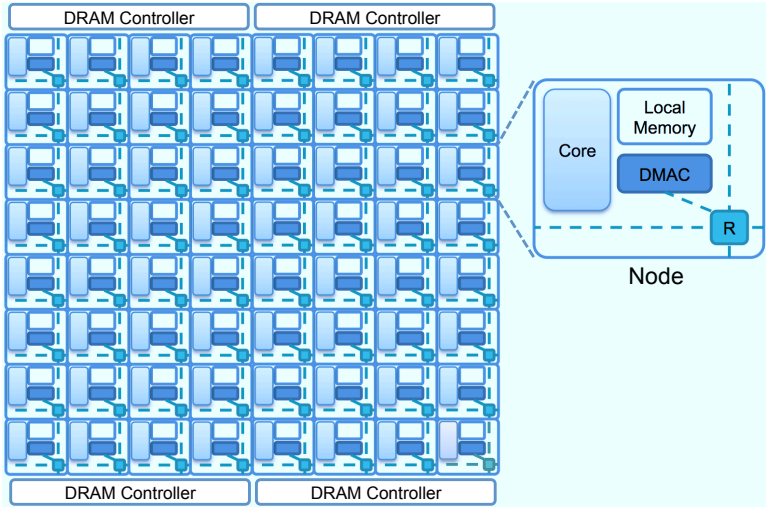


図 1 M-Core アーキテクチャ

広く利用されている．これらのアーキテクチャでは効率的なキャッシュ管理とプリフェッチ機構により，高いメモリ性能を達成している．その一方で，キャッシュの代わりに小容量のスクラッチパッドメモリを持つアーキテクチャとして，Cell/B.E[2] がある．Cell/B.E[2] では，従来型汎用プロセッサコアの PPE と，強力な SIMD 演算が可能なアクセラレータコアである SPE を複数持つ構成をとる．PPE は通常の汎用プロセッサコアと同様，階層型キャッシュによりメインメモリと接続される．一方，SPE はキャッシュを持たず，小容量のスクラッチパッドメモリを持つ．各 SPE のスクラッチパッドメモリの内容は，プログラム中の明示的な DMA 転送命令により，他の SPE のスクラッチパッドメモリやメインメモリとデータの共有を行う．

本稿では Cell/B.E のような DMA ベースにアーキテクチャについて，データ転送のオーバーヘッドを削減する手法について検討する．本稿では特に，Cell/B.E と同様に各コアがスクラッチパッドメモリを持ち，明示的な DMA 転送によりデータ共有を行う M-Core アーキテクチャ[3),4)] 上に提案手法の適用を行う．**図 1** にアーキテクチャ全体の構成を示す．M-Core では，コア (図中，Core)，ローカルメモリ (図中，Local Memory)，DMA コントローラ (図中，DMAC) およびオンチップネットワークのルータ (図中，R) をそれぞれ 1 つずつ含むノード (図中，Node) を単位とする．これらをメッシュネットワークで接続し，各ノードはそれぞれの位置に応じたユニークな ID(ノード ID) を持つ．

各ノードのコアは，自ノードのローカルメモリに格納されている命令を実行し，処理を行う．各ローカルメモリのアドレス空間は独立しており，各コアはそれぞれ自ノードのローカルメモリに対しては，ロード・ストア命令を介して直接アクセスすることができる．また，ローカルメモリのアドレスは仮想化されていない．

また，他ノードのローカルメモリまたはオフチップのメインメモリに対しては DMA コントローラを介して DMA 転送命令を発行することでアクセスすることができる．DMA 転送には，ノード ID により，どのノードのローカルメモリに対する DMA なのかを指定し，加えて転送元と転送先それぞれのローカルメモリ中のアドレスと転送サイズを指定する必要がある．DMA コントローラは DMA 転送命令が発行されると，指定されたローカルメモリアドレスから指定されたサイズをゼロコピーで送信する．すなわち，送信用バッファに送信データの退避は行わない．

DMA 転送命令は主に，命令発行元のコアから他のコアまたはメインメモリに対してデータを転送する DMA PUT，および，その逆に他のコアまたはメインメモリから命令発行元のコアへとデータを転送する DMA GET の 2 つに大別される．また，これらの DMA PUT/GET には，その命令が完了するまで後続の処理を行わないブロッキング転送と，DMA 命令の完了を待たずに後続の処理に移行するノンブロッキング転送の 2 種類が存在する．これは，MPI における MPI_Send()/MPI_Recv() と MPI_Isend()/MPI_Irecv() の関係とよく似ている．

M-Core では，DMA 転送のための API として，ブロッキング転送の MC_dma_put() および MC_dma_get()，ノンブロッキング転送の MC_dma_put_nowait() および MC_dma_get_nowait() が定義されている．しかし本稿では DMA PUT のみを議論の対象し，DMA GET に関する議論は今後の課題とする．

図 2 に DMA PUT を用いたサンプルコードを示す．配列 buf に対して処理を行った後で，その結果を他ノード (dst_id) に送信し，再度配列 buf に対して処理を行っている．ここで，MC_dma_put() はブロッキング転送として処理されるため，DMA によるデータ転送が完了するまでは，後続の for 文で表現されている処理には進まない．ブロッキング転送を用いることで送信元のメモリ空間を破壊することなく，かつゼロコピーで DMA 転送を行うことが出来る．

```
int buf [1024];
int i;

for(i=0; i<1024; i++){
   buf[i] = i * i;
}

MC_dma_put(dst_id, (void*)dst_buf, (void*)buf, sizeof(buf));

for(i=0; i<1024; i++){
   buf[i] = buf[i] + 1;
}
```

図 2 DMA 転送 API を用いたサンプルコード.

3. 提 案 手 法

先の例では，DMA 転送の範囲と DMA 転送後に使用するメモリ範囲が重なっているため，ブロッキング転送を行っている．しかしプログラマはダブルバッファリングなど手法を用いて，予め DMA 転送とその後の処理で使用するメモリ範囲の重なりを取り除くことが可能である．その場合，ブロッキング転送ではなく，ノンブロッキングの DMA 転送に置き換えることで，通信と処理がオーバーラップされ，通信オーバーヘッドを隠蔽できる．しかし，すべてのデータ転送に対してダブルバッファリングを適用することは，プログラマの大きな負担となる．

本稿では，DMA 転送元の DMA による通信とその後続の処理のオーバーラッピングを支援する機構を提案する．**図 3** に提案手法を適用した M-Core のノードの構成を示す．本稿では M-Core のノードに対して提案手法を適用したが，他の DMA ベースのアーキテクチャにも同様に適用できる．

提案手法では，コアと DMA コントローラ・ローカルメモリ間にストアバッファと Violation Detector というハードウェアを追加する[*1]．通常，M-Core アーキテクチャにおいて，自ノードから他ノードへのブロッキング DMA 転送を行う場合，コアは DMA 転送完了を待ってから後続の処理に進む．しかし，本手法を適用した場合は，ブロッキング転送に

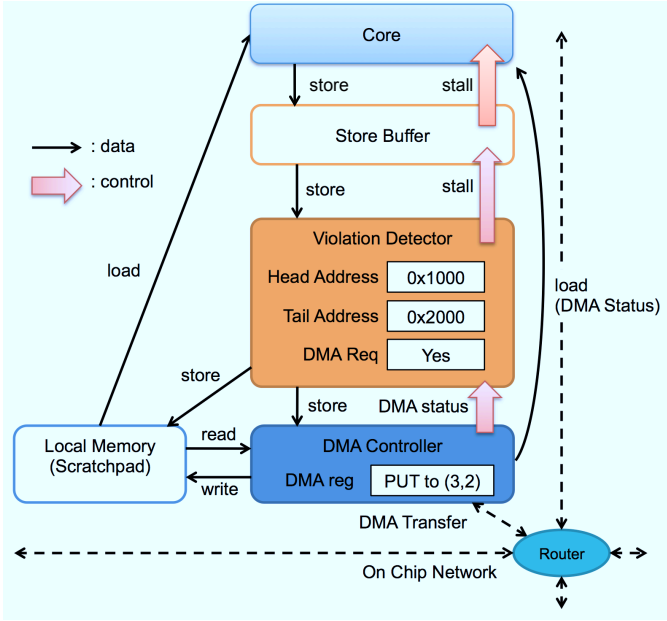


図 3 Violation Detector と Store Buffer を挿入したノード

おいてもコアは DMA 転送の完了を待たずに，先行的に転送完了後に行うはずの処理に進む．これにより通信と処理のオーバラップを行い，DMA 転送による通信レイテンシの隠蔽を目指す．しかし，DMA 転送の範囲と本来転送完了後に行われる処理でのメモリ範囲にに重なりがある場合，先行的に後続の処理に進んでしまうと，コアのストア命令と DMA 転送の間に Write after Read のデータハザードが発生する可能性がある．本手法では，データハザードの発生を防ぐ為に，ストア命令の結果はまずストアバッファに格納される．

Violation Detector は，DMA コントローラから DMA 転送の情報を受け取り，3 つのフィールド (Head Address, Tail Address, DMA Req) を保持する．Head Address は自ノードから他ノードへのデータ転送が未完了なデータ範囲の先頭アドレス，Tail Address は末尾アドレスを示す．DMA Req は自ノードから他ノードへの DMA 転送要求の有無を示す．Violation Detector は，これら 3 つの情報を用いて，DMA 送信元のメモリ範囲のうち，未転送範囲のアドレスを監視する．

⋆1 元よりコアがストアバッファを搭載する場合は，改めてストアバッファを追加する必要はない．

Vol.2011-ARC-196 No.3
2011/7/27

表 1 構成

コア	1 inst / cycle, single cycle
ローカルメモリ	512KB, 4 ports (Inst, Load/Store, DMA Read, DMA Write), 1 cycle
ルータ	1 hop / cycle, warm-hole, no virtual channel, Inbuf: 4 entry
ストバッファ	0 entry (Base, no Violation Detector). 8, 16 and 32 (Proposal)
ノード数	16, 64

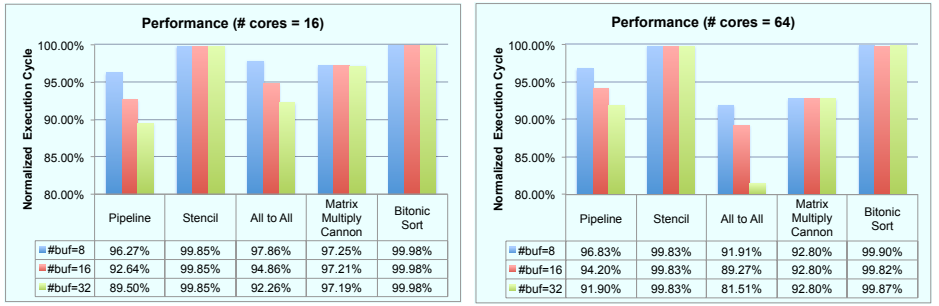
DMA 転送が行われていないときは，ストアバッファに格納されているストア命令の実行結果は，ストアバッファの先頭から随時ローカルメモリに反映され，ストアバッファから追い出される．DMA 転送中は，先頭エントリーの書き込み先アドレスが DMA 転送の対象ではないか，DMA 転送の対象であるが，既に転送が完了している場合には，通常と同様にローカルメモリにストア命令の結果を反映させ，ストアバッファから追い出される．しかし，先頭アドレスが DMA 転送の対象であり，かつ未転送範囲である場合には，ローカルメモリへの反映と追い出しは延期される．これにより，DMA 転送を追い越して先行実行の結果がローカルメモリに反映されるのを防ぐ．もし，ストアバッファに空きエントリーがない場合には，コアはストールする．加えて，M-Core では DMA 転送命令はメモリマップドレジスタへのストア命令で実現されており，DMA 転送命令もまずストアバッファに格納される．DMA 転送中に後続の新たな DMA 転送要求をコアが発行する場合は，先行する DMA 転送が完了するまで，後続の DMA 転送命令はストアバッファで待機する．

Violation Detector は 2 つのアドレス値を格納し，ストアバッファの先頭アドレスとそれらを比較するハードウェアである．前述の振る舞いは 2 つの整数比較器と 1 つの AND ゲートという小規模なハードウェア追加で実現できる．

4. 評　　価

本章では，提案手法をマイクロベンチマークで評価する．評価には M-Core アーキテクチャのソフトウェアシミュレータ SimMc を用いた．プロセッサ構成を**表 1** に示す．ノード数 16 ノード，64 ノードの 2 つの構成で評価を行った．また，ストアバッファ・Violation Detector なし，ストアバッファをそれぞれ 8 エントリー, 16 エントリー, 32 エントリーずつ持つ，計 4 構成で評価を行った．ベンチマークには次の 5 つを用いた．すべてのベンチマークは，データ転送にはブロッキング転送の MC_dma_put のみを利用している．

Pipeline: 配列に対する処理の後，1 つのノードに対してデータ転送．次のノードはデータを受信完了したら同様の処理を行う．配列サイズは 128 バイトとした．



(a) 16 コア　　(b) 64 コア

図 4 ブロッキング転送の完了を待機した場合の実行サイクル数を基準としたときの，提案手法適用による相対実行サイクル数

Stencil: 2 次元配列に対する平均化処理．配列を領域分割し，境界データのみを転送．配列サイズは 16900(130×130) バイトとした．

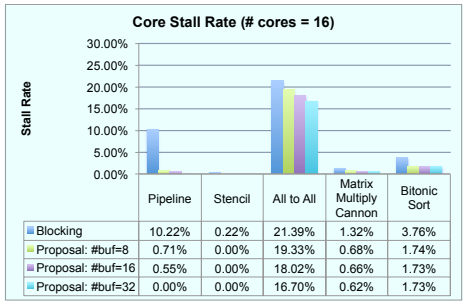
All to All: 各ノードに送信する値を生成後，そのデータを転送．全ノード分繰り返す．配列サイズは 128 バイトとした．

Matrix Multiply Cannon: Cannon アルゴリズム[5] により並列化された行列積．データサイズは 16384 (128×128) バイトとした．

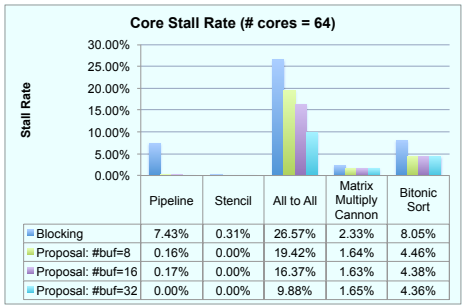
Bitonic Sort: 並列ソート[6]．データサイズは 40K バイトとした．

まずはじめに，提案手法適用による性能向上について示す．**図 4** に提案手法適用による相対実行サイクル数を示す．提案手法未適用の場合のプログラム完了に要するサイクル数を 100%としている．Pipeline，All to All では，ストアバッファのエントリー数の増加に従い，性能が向上しており，64 ノード・ストアバッファ32 エントリーの場合，約 19%の性能向上を達成している．また，Matrix Multiply Cannon では，ストアバッファエントリー数の増加による性能向上ないが，64 ノードの場合約 7%と，有意な性能向上を確認した．その一方で，Stencil および Bitonic Sort ではほとんど性能向上は見られなかった．

次に，提案手法適用よる通信オーバーヘッドの変化について示す．**図 5** に提案手法適用による相対ストールサイクル数を示す．提案手法未適用の場合に DMA 転送完了を待つためにコアがストールしたサイクル数を 100%として，提案手法適用時にコアがストールした相対量を示す．Pipeline では提案手法適用によりほとんどのストールを削減できた．All to All ではストアバッファのエントリー数の増加に従い，徐々にストール量が削減できている．

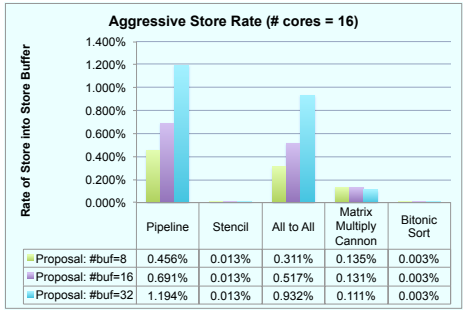


(a) 16 コア

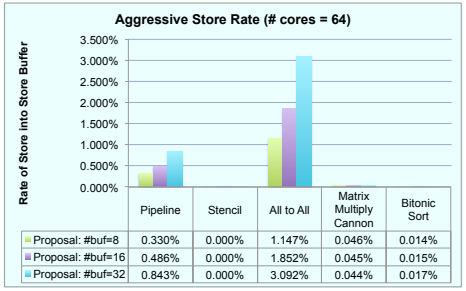


(b) 64 コア

図 5 プロセッサコアがストールしたサイクルの割合

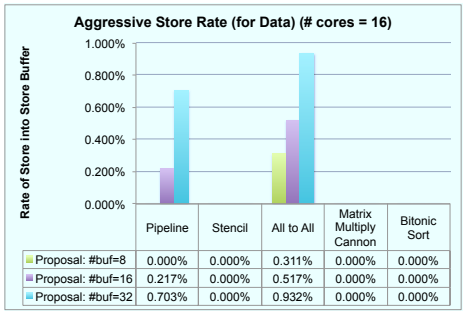


(a) 16 コア

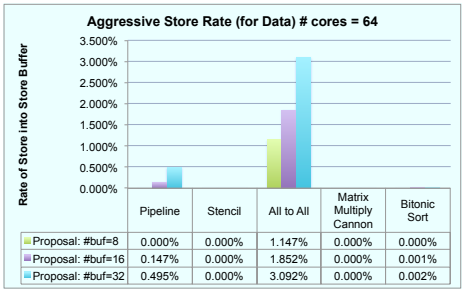


(b) 64 コア

図 6 先行ブロッキング DMA 転送中にストアバッファに対して発行した後続ストア命令がストアバッファに格納されたサイクル数の割合

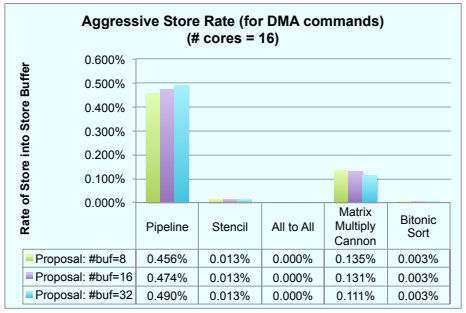


(a) 16 コア

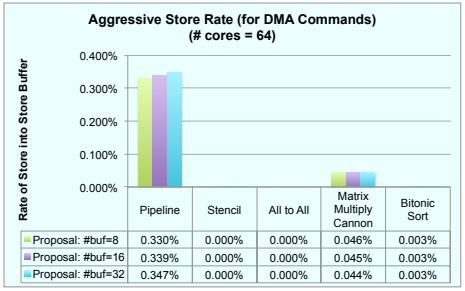


(b) 64 コア

図 7 先行ブロッキング DMA 転送中にストアバッファに対して発行した後続ストア命令のうち，データを格納する通常のストア命令の割合



(a) 16 コア



(b) 64 コア

図 8 先行ブロッキング DMA 転送中にストアバッファに対して発行した後続ストア命令のうち，DMA 転送要求を引き起こすストア命令の割合

Bitonic Sort ではエントリー数の影響は見られないが，約半分までストール量を削減できている．Matrix Multiply Cannon は元よりストール量が少ないが，提案手法適用によりストール量を削減できている．

図 6 に提案手法適用による先行実行を行っている間に，ストア命令がストアバッファに格納されたサイクル数の割合を示す．また，**図 7** に提案手法適用による先行実行を行っている間に，データを格納する通常のストア命令がストアバッファに格納されたサイクル数の割合を示す．加えて，**図 8** に提案手法適用による先行実行を行っている間に，DMA 転送要求のストア命令がストアバッファに格納されたサイクル数の割合を示す．図 6 より，実行サイクル数の削減量が多かった Pipeline, All to All, Matrix Multiply Cannon は先行する DMA の転送中に後続のストア命令が実行され，ストアバッファに格納されていることがわかる．しかし全実行サイクル数に対する割合はどれも低く，最大でも 3%である．また，図 7 と図 8 より，Pipeline はデータと DMA 要求のどちらのストア命令も先行実行できているが，All to All ではデータのみ，逆に Matrix Multiply Cannon では DMA 要求のみであることがわかる．これらのことより，DMA 要求とデータを別に管理し，後続の DMA 転

送要求に対して Violation Detector と同様の機構で監視をすることで，更なる性能向上が見込めることが考えられる．また，性能向上がほとんど見られなかったベンチマークではストアバッファの活用による先行実行がほとんどできていない．これらのベンチマークでは，他ノードに対するデータ送信の完了待ちのオーバーヘッドよりも，他ノードからのデータを集める際のオーバーヘッドが大きいのではないかと考えられる．今後，他ノードからデータを取得する DMA GET に対する通信と処理のオーバーラップ手法について検討する．

5. 関 連 研 究

本研究では単一プロセッサ内の DMA 転送オーバーヘッド削減を目的としている．関連する研究としては，従来よりクラスタ型計算機システムにおける並列処理のオーバーヘッド削減を目的とした研究がある．例えば，クラスタ計算機におけるゼロコピー通信の高速化に関する研究としては[7] がある．コンパイラなどの解析により，通信と処理のオーバーラップにより性能向上を図る研究としては[8)–11)] などがある．また，ハードウェア機構を活用した手法としては，SMT プロセッサのスレッドレベル並列性を活用して通信レイテンシを隠蔽する[12] がある．これは GPGPU/CUDA における多くのスレッドを単一の演算ユニットで実行し，メモリアクセスレイテンシを隠蔽する手法とも似ている．

6. ま と め

本稿では DMA 転送によりコア間のデータ共有を行うアーキテクチャを対象に，データ転送と命令実行のオーバーラップピングを支援する機構について提案した．機能レベルシミュレータを用いて，いくつかのマイクロベンチマークでは性能向上を確認した．しかし，今回取得したデータのみでは十分な通信オーバーヘッド解析ができていない．今後，DMA 転送先でのオーバーラッピングを行う手法の検討と合わせて，最もクリティカルなオーバーヘッドの解析とその隠蔽手法について検討する．また，ハードウェアサポートを活用した，よりアグレッシブな MPI ライブラリ等の検討を行いたい．

参 考 文 献

1) 高前田伸也，吉瀬謙二：メニーコアプロセッサにおけるコア間通信レイテンシ隠蔽手法の検討，情報処理学会第 72 回全国大会，pp.173–174 (2010).
2) : Cell Broadband Engine. http://cell.scei.co.jp/index_j.html.
3) 植原　昂，佐藤真平，吉瀬謙二：メニーコアプロセッサの研究・教育を支援する実用的な基盤環境，電子情報通信学会 システム開発論文特集号， Vol.J93-D, No.10, pp. 2042–2057 (2010).
4) : M-Core Project. http://www.arch.cs.titech.ac.jp/mcore/.
5) Cannon, L.E.: A cellular computer to implement the kalman filter algorithm, PhD Thesis, Bozeman, MT, USA (1969). AAI7010025.
6) Nassimi, D. and Sahni, S.: Bitonic Sort on a Mesh-Connected Parallel Computer, *IEEE Trans. Comput.*, Vol.28, pp.2–7 (1979).
7) Tezuka, H., O'Carroll, F., Hori, A. and Ishikawa, Y.: Pin-down cache: a virtual memory management technique for zero-copy communication, *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International … and Symposium on Parallel and Distributed Processing 1998*, pp.308 –314 (1998).
8) Danalis, A., Pollock, L., Swany, M. and Cavazos, J.: MPI-aware compiler optimizations for improving communication-computation overlap, *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, New York, NY, USA, ACM, pp.316–325 (2009).
9) Fishgold, L., Danalis, A., Pollock, L. and Swany, M.: An automated approach to improve communication-computation overlap in clusters, *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p.7 pp. (2006).
10) Danalis, A., Kim, K.-Y., Pollock, L. and Swany, M.: Transformations to Parallel Codes for Communication-Computation Overlap, *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, p.58 (2005).
11) Miyoshi, T., Kise, K., Irie, H. and Yoshinaga, T.: CODIE: Continuation-Based Overlapping Data-Transfers with Instruction Execution, *Networking and Computing (ICNC), 2010 First International Conference on*, pp.71 –77 (2010).
12) Goumas, G., Anastopoulos, N., Koziris, N. and Ioannou, N.: Overlapping computation and communication in SMT clusters with commodity interconnects, *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp.1 –10 (2009).