Vol.2022-ARC-248 No.44
Vol.2022-SLDM-198 No.44
Vol.2022-EMB-59 No.44
2022/3/11

高帯域幅メモリ搭載FPGAを用いたランダムアクセス指向メモリアーキテクチャとプログラミングモデルの検討

菅 研吾[1,a)]　高前田（山崎） 伸也[1,b)]

概要：High Bandwidth Memory (HBM) は最近の多くの FPGA ボードで利用可能である．HBM はそのバス幅の広さによって，メモリアクセスの観点からアプリケーションの高速化を図ることが可能である．しかしながら HBM が複数の擬似チャネルを持つ構造や，各チャネルの比較的長いレイテンシのため，HBM から理想的な性能を引き出すのが難しいことが知られている．
本稿では，ランダムメモリアクセス待ち時間とスループットが性能に関わる FPGA ベースのアクセラレータ上で，HBM 固有の広いバス幅を利用した高性能汎用メモリアーキテクチャとプログラミングモデルを提案する．プログラミングモデルはロードリクエスト，レイテンシの待機，データ読み込みを異なる命令に分割し，この命令の分離は計算時間を利用したデータ移動時間の隠蔽を容易にする．データは読み込み要求時に発行されたタグに基づき管理が行われる．シミュレーションに基づく初期評価により，提案するメモリシステムが低い導入コストでグラフ処理を 3.7 – 7.6 倍高速化することを示した．

1. はじめに

近年，限られたトランジスタを効率的に利用するドメイン固有アーキテクチャ [1], [2], [3] が，性能向上のアプローチとして注目されている [4]．ドメイン固有アーキテクチャが高い性能を発揮するためには，多くのアプリケーションでボトルネックとなっているメモリアクセスを効率的に行うことが必要である．高帯域幅メモリ（High Bandwidth Memory; HBM）[5] は，従来の DDR4 といった規格よりも小型で省電力でありながら，大きなバス幅により広帯域を実現する技術であり，スーパーコンピュータのプロセッサ [6] やデータセンター FPGA に採用されている．しかし，HBM を用いた高速アクセラレータの構築は，多くのチャネルを並列にアクセスする設計や各疑似チャネルの長いレイテンシの隠蔽が必要であり，困難な作業となる．これらの要件が適切なアーキテクチャの設計を困難にしている．

メモリアクセスのレイテンシを短縮するため，ワークロード全体のデータの一部を小さく高速なオンチップバッファに保持するキャッシュメモリが広く使用されている．キャッシュは現在の商用汎用プロセッサの全てにキャッシュが搭載されている [7]．キャッシュは，局所性の法則が成り立つ多くの状況でメモリアクセスを高速化し，メモリアクセス中に暗黙的に操作できるため，プログラマにとって使いやすい．一方で，キャッシュは連想メモリや擬似 LRU データ置換のように複雑な構成要素を持ち，キャッシュを搭載するプロセッサは，そのエネルギーの多くをキャッシュ操作に費やしている [8]．アクセラレータにおけるメモリシステムは，ドメイン固有のアプリケーションのために電力効率を高める軽量なメカニズムである必要がある．また，グラフ処理や疎行列積などのアプリケーションでは，キャッシュのメモリアクセスパターンが不規則であるため，メモリアクセス待ち時間の低減が十分に機能しない．

本研究では，HBM などのマルチバンクメモリに適した汎用メモリシステムとして，機構が簡単でプログラマブルであり，ランダムアクセスレイテンシが実行時間に関係する場合に有効なフレームワークを提案する．このメモリシステムは，メモリアクセスをメモリアクセス要求，レイテンシの待機，データ読み出しの 3 段階に切り離したプログラミングモデルを持っている．アクセスリクエストの管理は，リクエスト発行時に発行されるタグに基づいて行われる．このプログラミングモデルは扱いやすく，プログラマはシステム固有のプログラミング技術を使ってレイテンシを隠蔽することができる．

[1] 東京大学
〒 113–8656 東京都文京区本郷 7 丁目 3–1
[a)] suga-kengo@g.ecc.u-tokyo.ac.jp
[b)] shinya@is.s.u-tokyo.ac.jp

Vol.2022-ARC-248 No.44
Vol.2022-SLDM-198 No.44
Vol.2022-EMB-59 No.44
2022/3/11

2. 提案手法

2.1 プログラミングモデル

一般的なメモリアクセス命令では，ロード命令を発行してからデータを受け取るまで計算回路は待機しなければならない．プロセッサのメモリ階層がバッファを持たない場合，メモリアクセスのたびにレイテンシに等しい待機時間が発生する．

本研究が提案するメモリアクセスインターフェースは，従来のメモリロードをロードリクエスト，レイテンシの待機，データ読み出しの 3 種のマイクロ命令に分離する．

ロードリクエスト: `load(tag,mem,addr,offset)`
計算回路は，メモリ `mem` と内部アドレス `addr` を指定して，データのロードを要求する．リクエストを発行すると，その要求に対応する通し番号が計算回路の `tag` レジスタに格納される．要求されたロードは計算回路のバックグラウンドで実行され，データはプールと呼ばれるバッファの計算回路が指定した位置 `offset` に格納される．

レイテンシ待機: `wait(tag)` `wait` 命令が呼ばれた後, `tag` レジスタに紐づけられたデータのロードが完了するまで，計算回路は待機する．

データ読み出し: `read(dst,offset)` 計算回路は，プールの内部位置 `offset` を指定し，この位置に格納されているデータを読み出して，計算回路の `dst` レジスタに格納する．

上記で定義したインターフェースを使用して，我々はソースコード 1 で示される従来のロード命令をソースコード 2 での通りに分割できる．

```
1  load(dst,mem,addr)
```

ソースコード 1 従来のロード命令

```
1  load(tag,mem,addr,offset)
2  wait(tag)
3  read(dst,offset)
```

ソースコード 2 分離されたロード命令

上で定義された `load`，`wait` 命令の間にこれらと依存関係を持たない命令を挿入することで，チャネルの長いメモリレイテンシを隠している．さらに，`load` 命令はメモリから指定されたアドレスのデータのみをロードするように命令するため，提案するメモリシステムはランダムアクセスの連続がシーケンシャルアクセスに比べて性能を低下させることがなく，局所性の法則に基づいてキャッシュラインを持つキャッシュメモリとは異なる．

提案するプログラミングモデルは，ロードリクエストと同様に従来のストア命令をストア要求とレイテンシ待機の 2 つの動作に分離する．ロードリクエストとストアリクエストに使用される待機命令は共通である．

ストアリクエスト: `store(tag,mem,addr,src)` 計算回路は，メモリ `mem` と内部アドレス `addr` を指定して，`src` レジスタに格納されているデータを要求する．リクエストが送信されると，計算回路のタグレジスタにはリクエストに対応した通し番号が格納される．要求されたストアは，計算回路のバックグラウンドで実行される．

上記で定義したインターフェースを用いると，ソースコード 3 で示した従来のストアを，ソースコード 4 で示すように分割できる．

```
1  store(mem,addr,src)
```

ソースコード 3 従来のストア命令

```
1  store(tag,mem,addr,src)
2  wait(tag)
```

ソースコード 4 分離されたストア命令

ロードリクエストと同様に，従来のストア命令を分割する方法は自然な定義であり，ユーザーの利便性を損なうことはない．さらに，本提案で定義したストア命令を連続して実行したり，`wait` 命令の前に依存関係のない命令を挿入することで，メモリレイテンシを隠蔽することができる．また，ランダムなアドレスへのストア要求でも帯域を劣化させることはない．

提案するメモリシステムは，キャッシュと異なり，計算回路が明示的にプールを操作する．このため，プール内のデータを更新するためのインターフェイスを用意している．

プールへの書き込み: `write(offset,src)` 計算回路は，プールの内部位置 `offset` を指定して，`src` レジスタに格納されているデータを書き込む．

2.2 アーキテクチャ

図 1 は，M 個の擬似チャネルで構成される HBM で動作するアーキテクチャの概要を示している．計算回路は，前述したメモリアクセスリクエストを発行できるような FPGA の LUT 等で構成されるアクセラレータ回路に対応し，その実装はマイクロプロセッサに限られない．

アーキテクチャは，各チャネルの入力に FIFO を備えている．FIFO はチャネルと 1 対 1 で接続し，データ通信がユニットスイッチをまたがないようにすることで，スループットを向上させる．FIFO はメモリアクセスの役割を計算回路から切り離すため，計算回路はメモリがビジー状態でも待機せずに連続してアクセスを要求できる．FIFO

Vol.2022-ARC-248 No.44
Vol.2022-SLDM-198 No.44
Vol.2022-EMB-59 No.44
2022/3/11

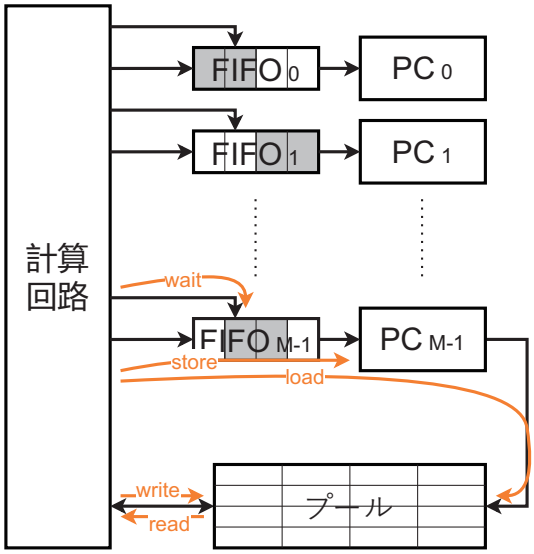

図 1 アーキテクチャ

は最適なタイミングでチャネルにリクエストを送るため，HBM のバンド幅を最大限に活用することができる．

また，計算回路はアクセス要求時に，FIFO からリクエストの通し番号を受け取る．FIFO は対応するチャネルのメモリアクセス要求の処理状況を管理するため，計算回路は処理の進捗を FIFO に問い合わせれば良い．

またアーキテクチャでは，チャネルの出力部にプールと呼ばれる小さなテーブルを配置する．ロード要求が完了すると，メモリはプールに出力を保存する．メモリからプールへのデータ転送は，計算回路の状態とは無関係に適用されるため，計算回路は他の演算を行うことでレイテンシを隠蔽することができる．

2.3 最適化技術

これまで，プログラミングモデルの定義とそれを実現するアーキテクチャの構築に焦点をあててきた．本章では，定義したメモリシステムを活用するためのプログラミング技術を具体的な事例を交えて紹介する．

2.3.1 レイテンシ隠蔽

プログラミングモデルが定義する通り，アーキテクチャは計算回路のバックグラウンドでロード要求を処理する．したがって，ロード命令を発行してから対応する待機命令を発行するまでの間に他の命令を実行することで，計算回路は挿入された命令に要する時間に相当するレイテンシを隠すことができる．図 2 はレイテンシ隠蔽前の命令列と実行の流れを表す．`add` 命令を `wait` 命令の前に挿入することで，図 3 のように実行時間の短縮を実現する．

2.3.2 タイル化

タイル化とは，図 4 のように，大きなサイズの入力データをプールに格納できるサイズに分割し，分割した各タイルをプールにロードし処理することを繰り返す手法である．このように処理を分割することで，計算回路はプール

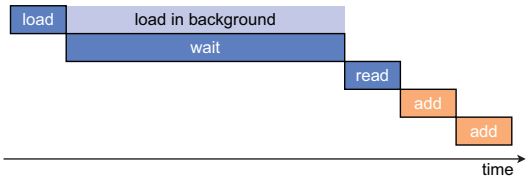

図 2 レイテンシ隠蔽前の命令実行

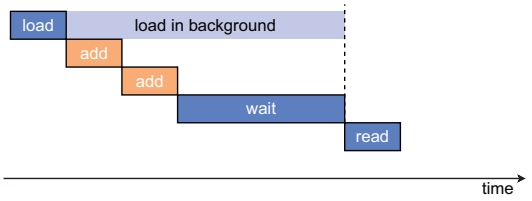

図 3 命令並び替えによるレイテンシ隠蔽後の命令実行

の容量に制限されずデータを扱えるようになる．プールの容量より大きなデータを処理する場合，タイル化は効率的なデータロードのための有効な手段である．

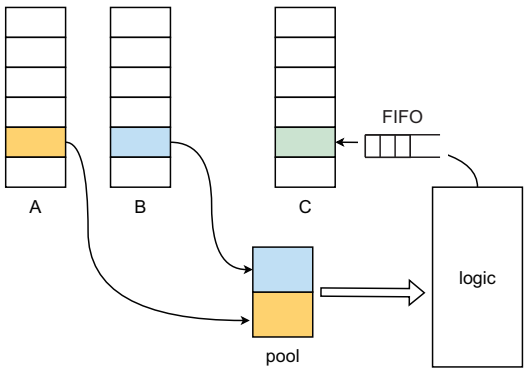

図 4 タイル化を使用したデータ処理分割

2.3.3 メモリインターリーブ

メモリインターリーブ [9] とは，主記憶装置を複数の擬似チャネルに分割することで，主記憶装置とプロセッサ間のデータ転送を高速化する方法である．

HBM の複数の擬似チャネルを 1 つの大きなメモリとみなし，メモリインターリーブ技術を使用する際に統一されたアドレス空間とする．このアドレス空間における任意のアドレスは，メモリの内部アドレスの上位ビットと，メモリインデックスの下位ビットに対応させる．この技術により，ユーザーには，メモリの抽象化によるプログラマビリティの向上と，メモリアクセスの分散によるシステム性能の向上という 2 つのメリットがもたらされる．ロード要求はランダムアクセスでランダムな擬似チャネルに発行されるため，メモリアクセスの作業負荷は容易に分散される．

3. 評価

3.1 評価手法

提案するメモリシステムの性能を定量化するために，サイクルレベルで動作するシミュレータを記述した．システ

Vol.2022-ARC-248 No.44
Vol.2022-SLDM-198 No.44
Vol.2022-EMB-59 No.44
2022/3/11

ムは Xilinx Alveo U280 が搭載する HBM を使用すると想定し [10], [11]，メモリの簡易モデルによる初期評価を行った．評価に用いたパラメータは表 1 の通りである．性能評価において，計算回路と HBM の間にバッファを持たないアーキテクチャ，バッファとしてキャッシュを持つアーキテクチャの 2 通りを比較対象とした．

パラメータ	値
擬似チャネル数	32
総容量	8 GB
レイテンシ	108 サイクル

表 1 シミュレーションに用いたメモリのパラメータ

また，FIFO とプールのパラメータを，比較対象のキャッシュのパラメータと合わせて表 2 に示す．

パラメータ	値
FIFO あたりの容量	64 リクエスト
プール総容量	32 KB
キャッシュ総容量	32 KB
キャッシュラインサイズ	32 words
キャッシュの特性	フルアソシアティブ

表 2 シミュレーションに用いた FIFO，プール，キャッシュのパラメータ

本アーキテクチャでは，計算回路としてインオーダーシングルコアプロセッサを使用した．計算回路は，演算命令，条件分岐命令をサポートし，これらの命令の実行を 1 クロックサイクルで完了すると仮定した．また，各擬似チャネルは 1 サイクルごとにロード要求を受け付け，各要求は所定のクロックサイクル後に完了すると仮定した．比較対象の計算回路は，上記の命令セットに加えて，分離されていない従来のメモリアクセス命令（ロードとストア）をサポートするものとする．これに対し，提案するシステムの計算回路は，プログラミングモデル章で定義した `load`, `read`, `store`, `write`, `wait` 命令をサポートする．また，評価対象のアーキテクチャは，いずれもメモリアクセスにワードアドレッシングを採用している．

ベンチマークでは，複数のグラフに対して BFS(幅優先探索) を実行し，実行サイクル数を指標として性能を評価した．データセットは SuiteSparse Matrix Collection [12] から表 3 にある 4 種類のグラフを選択し使用した．

グラフ		頂点数	辺数	性質
p2p-Gnutella04	(FS)	10K	40K	有向
rajat09	(CS)	24K	106K	無向
vsp_barth5_1Ksep_50in_5Kout	(MS)	32K	204K	無向
finance256	(LP)	37K	298K	無向

表 3 評価に用いたデータセット

ベンチマークに用いたアセンブリとして，比較対象の計算回路は共通のプログラムを使用し，提案手法の計算回路はこれに最適化技術で触れた 3 通りの技術を適用したプログラムを使用した．

3.2 結果

図 5 は，各グラフデータにおける BFS の速度向上率を，バッファを用いないアーキテクチャを基準として示している．この実験結果では，提案手法はすべての入力に対して，平均で 5.9 倍，最大で 7.6 倍，システムを高速化した．さらに，提案手法は LP 以外の全ての入力に対して処理速度を上回った．

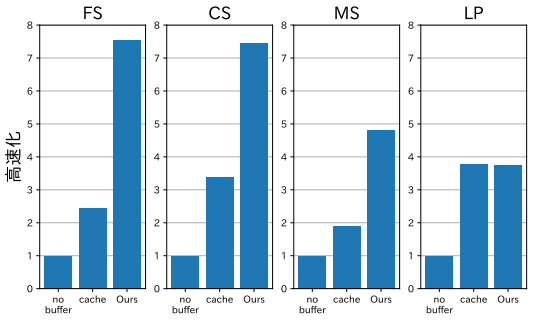

図 5 各グラフにおける BFS の速度向上率

また，各システムがどのように実行時間を費やしているかを明らかにするために，バッファを用いない場合で正規化された，各ベンチマークの実行に費やしたサイクル数の内訳を図 6 に示す．また表 4 には，各ベンチマークで実行された命令数とキャッシュヒット率を示している．

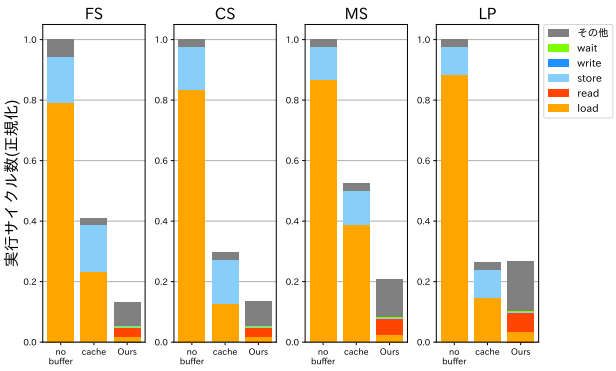

図 6 各グラフにおける BFS の実行サイクル数の内訳

	Ours		comparison		
	load	store	load	store	ヒット率
FS	118139	22862	111815	21626	77.1%
CS	288910	49684	284592	48964	88.9%
MS	775553	99874	503856	64424	62.2%
LP	1377733	151322	709120	151322	87.3%

表 4 各グラフにおけるメモリアクセス命令実行回数

サイクル数の内訳から提案手法の性能向上要因を考察する．アーキテクチャがシステムを高速化したのは，レイテンシ隠蔽技術により，計算回路がグラフのエッジや頂点の距離に対するメモリアクセス要求を連続して発行できるようにしたためである．すなわちリクエストの並行性が高まることで，計算時間が短縮されたと考えられる．また，メモリアクセス命令の実行回数から，特に MS のようにキャッシュミス率が高いアプリケーションでは，提案手法はキャッシュより高い性能を発揮することがわかる．

しかし，LP を入力としたシミュレーション結果では，キャッシュが提案手法の処理速度を上回った．この理由として，メモリアクセスのオーバーヘッド，最適化技術の適用によるプログラムコードの変質の 2 点が考えられる．まず，提案手法では，メモリアクセスリクエスト発行時に FIFO から受け取ったタグを管理するオーバーヘッドを計算回路が負担しなければならない．このオーバーヘッドにより，提案手法は図 6 が示す通り，「その他」の命令に余計な時間を費やしている．

第二に，タイル化を使用してプログラムコードを書き換える過程では，コードの等価性を保てない場合がある．提案手法で実行されるプログラムは，データの一部分を一括してプールにロードし，プールのデータを基に計算を実行する．この計算がロード元のデータの変更に相当する場合，メモリとプールの間でデータの一時的な不整合が発生する．ベンチマークにおいてもデータの不整合によってコードの等価性は保たれておらず，表 4 によると，LP において提案手法ではメモリロードの回数が 94.3%増加している．

4. まとめ

本研究では，ランダムアクセスを考慮したアクセラレータのためのメモリシステムフレームワークを提案した．このメモリシステムは，従来のメモリアクセスをメモリアクセス要求，待ち時間，データ読み出しに分離するプログラミングモデルを含み，優れた計算速度を実現するために活用される．アーキテクチャは，HBM の構造に特化した軽量なトポロジであり，これをベースにした様々な設計の可能性を示唆するものである．また，初期評価用シミュレータに HBM の簡易モデルを実装することで，提案するメモリシステムの導入によってシングルコアプロセッサのグラフ処理性能を向上させることを確認した．

本稿では，HBM のモデルを単純化したモデルを使用した初期評価を行ったが，さらなる正確な評価のため，DRAMSim2 [13] のような正確なメモリシミュレータの導入や，実機を用いた実験が必要である．また，本稿で提案したシステムの改善のため，プログラミングモデルの改良やコンパイラの提案によるユーザビリティの向上，HBM のバースト転送などを活用したアーキテクチャのより一層の高速化について，さらなる研究の余地がある．

謝辞

本研究の一部は，JSPS 科研費 19H04075 および 18H05288 の支援により行われたものである．

参考文献

[1] Jouppi, N. P., Young, C., Patil, N. and Patterson, D.: A domain-specific architecture for deep neural networks, *Comm.ACM*, Vol. 61, No. 9, pp. 50–59 (2018).
[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z.: Rethinking the Inception Architecture for Computer Vision, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
[3] Hsu, J.: Nervana systems: Turning neural networks into a service [Resources_Startups], *IEEE Spectrum*, Vol. 53, No. 6, pp. 19–19 (2016).
[4] Shalf, J.: The future of computing beyond Moore's law, *Philosophical Transactions of the Royal Society A*, Vol. 378, No. 2166, p. 20190061 (2020).
[5] Wissolik, M., Zacher, D., Torza, A. and Da, B.: *Virtex UltraScale+ HBM FPGA: A revolutionary increase in memory performance* (2017).
[6] Jackson, A., Weiland, M., Brown, N., Turner, A. and Parsons, M.: Investigating Applications on the A64FX, *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 549–558 (2020).
[7] Patterson, D. A. and Hennessy, J. L.: *Computer Organization and Design*, Morgan Kaufmann, fifth edition (2013).
[8] Horowitz, M.: 1.1 Computing's energy problem (and what we can do about it), *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14 (2014).
[9] Burnett, G. J. and Coffman, E. G.: A Study of Interleaved Memory Systems, *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, AFIPS '70 (Spring), Vol. 36, New York, NY, USA, Association for Computing Machinery, pp. 467–474 (1970).
[10] Xilinx, Inc.: *Alveo U280 Data Center Accelerator Card Data Sheet* (2021).
[11] Xilinx, Inc.: *AXI High Bandwidth Memory Controller v1.0 LogiCORE IP Product Guide* (2021).
[12] Davis, T. A. and Hu, Y.: The University of Florida Sparse Matrix Collection, *ACM Trans. Math. Softw.*, Vol. 38, No. 1, pp. 1–25 (2011).
[13] Rosenfeld, P., Cooper-Balis, E. and Jacob, B.: DRAMSim2: A Cycle Accurate Memory System Simulator, *IEEE Computer Architecture Letters*, Vol. 10, No. 1, pp. 16–19 (2011).