Vol.2013-ARC-204 No.1
2013/3/26

RTL静的解析によるFPGAアクセラレータ向けアプリケーション特化メモリプリフェッチャー

高前田（山崎） 伸也[1,2,a)]　吉瀬 謙二[1,b)]

概要：より良い電力効率と高い性能の達成するために，汎用 CPU に加えて FPGA などを用いたアクセラレータを利用する計算機が普及しつつある．本稿では FPGA アクセラレータのメモリ階層をキャッシュを用いることにより抽象化し，プログラマビリティを高めつつ，高いメモリ性能を達成することを目的に，FPGA に搭載するアプリケーションに特化したメモリプリフェッチャーの構成手法を提案する．また，そのための解析ツールの構成について述べる．Verilog HDL で記述されたアプリケーションの RTL 記述を静的解析し，アプリケーションが含む状態遷移と各状態におけるメモリアクセス情報を取得する．そしてループ中の各メモリアクセスについて，次回同じ状態に到達したときにアクセスするであろうアドレスの定義木を構成し，プリフェッチャーとしてアプリケーションに加える．本稿では，簡単なアプリケーションを用いて，提案手法と現段階の構成ツールの初期的な評価を行った．

1. はじめに

より高い性能と電力効率を持つ計算機の構成を目的に，従来の CPU に加えて GPU や FPGA などのアクセラレータを組み合わせたヘテロジニアスな計算機がスーパーコンピュータをはじめとして，広く普及しつつある．FPGA を用いてアクセラレータを構成するには，従来 Verilog HDL や VHDL といった RTL（レジスタ・トランスファー・レベル）で回路の振る舞いを表現する低位言語である HDL（ハードウェア記述言語）を用いるのが一般的であった．しかし，近年では Vivado HLS や Impulse C などといった，より抽象的に回路の振る舞いを表現できる高位合成処理系も普及しつつあり，FPGA を用いたカスタムアクセラレータの利用はより盛んになると考えられる．

汎用 CPU においては，オフチップメモリはキャッシュにより抽象化されており，オンチップメモリとオフチップメモリのデータ転送はキャッシュ置き換えアルゴリズムとプリフェッチによって行われる．アプリケーションの振る舞いに応じたキャッシュラインの置き換えとプリフェッチを行うことにより，より高いメモリ性能を達成することが可能である．一方，FPGA アクセラレータにおいて，FPGA がチップ内にもつローカルメモリの容量よりも大きなデータを扱う場合には，外部メモリとチップ内メモリとの間でデータの入れ替えを明示的に行う必要がある．そのため，外部メモリを意識したハードウェア構成をとる必要があり，高性能なアクセラレータを容易に開発することを困難にする要因の一つとなっている．

本稿では，FPGA アクセラレータを対象とした抽象化と高性能を両立するメモリシステムの実現に向けて，アプリケーションに特化したプリフェッチ機構の構成手法を提案する．我々は，キャッシュを介してオフチップメモリにアクセスするアクセラレータ回路を対象として，アプリケーションの RTL 記述からプリフェッチ機構の回路記述を自動的に生成するツールを開発した．RTL 記述の静的解析により，メモリアクセスが発生する条件を特定し，当該メモリアクセスがアクティブになるのに先駆けてキャッシュ側にリクエストを生成する回路の RTL 記述をプリフェッチ回路として生成する．

本稿では，プリフェッチ回路の生成に用いた Verilog HDL の RTL 静的解析ツールの構成と，そのツールによって自動生成されたプリフェッチ回路の性能に関する初期評価の結果について述べる．

2. キャッシュを用いるFPGAアクセラレータ

図 1 に，オフチップメモリとオンチップメモリとの間のデータ転送スケジューリングを開発者があらかじめマニュアルで行う，従来型の FPGA アクセラレータの一般的な構成を示す．FPGA アクセラレータは主に，HDL や高位合成言語により記述されたアプリケーションのカーネル部

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
a) takamaeda@arch.cs.titech.ac.jp
b) kise@cs.titech.ac.jp

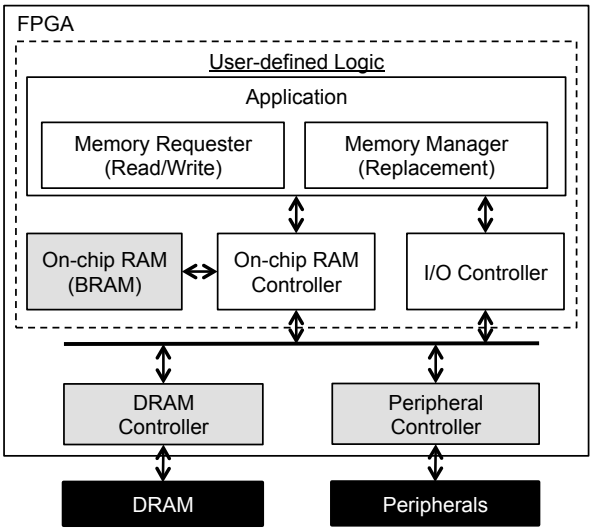


図 1 マニュアルでデータ転送を行う従来型の FPGA アクセラレータ

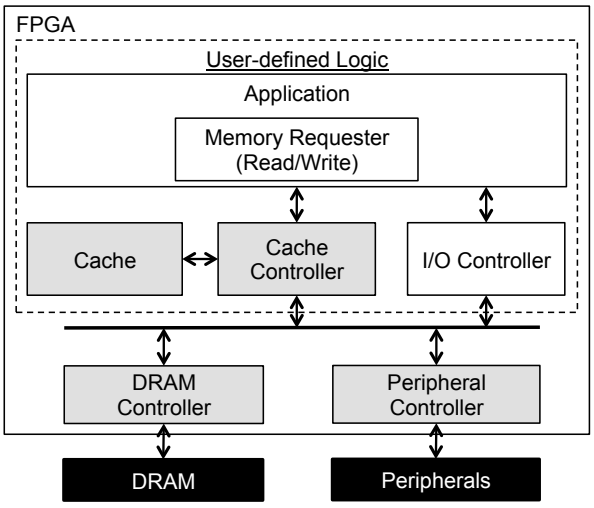


図 2 キャッシュを持つ FPGA アクセラレータ

分と，そこにデータを供給しまたそこからデータを取得するメモリコントローラ，および周辺回路などのコントローラなどから構成される．

カーネルの処理に持ちいられる一時データは，チップ内のローカルメモリ（On-chip RAM）に保存される．開発者は，カーネルとローカルメモリとの間のデータ転送を適切にスケジューリングし，カーネルが利用するデータを前もってローカルメモリに配置するようなオンチップメモリコントローラを実装する必要がある．大容量のデータセットを扱う場合など，FPGA チップ外部に接続された DRAM を利用する際には，DRAM とローカルメモリとの間のデータ転送，および，カーネルとオフチップメモリとの間のデータ転送を適切にスケジューリングする仕組みを実装する必要があり，これは開発者の大きな負担となる．

図 3 に，キャッシュを持つ FPGA アクセラレータの一般的な構成を示す．マニュアルでデータ転送をする場合とは異なり，カーネルからのメモリリクエストに応じて，オンチップメモリを利用して構成したキャッシュにキャッシュコントローラがデータを転送し格納する．キャッシュを用いることにより，メモリシステムをカーネルに対して抽象化しているため，開発者はデータ転送部分を実装する必要がないため，開発は容易になる．しかし，不必要なオフチップメモリアクセスやキャッシュラインの置き換えに起因するデータ供給に遅延により，マニュアルのデータ転送を行う場合と比べて，達成しうる性能は低くなることが多い．カーネルのメモリアクセスの特性に応じたキャッシュ置き換えアルゴリズムやプリフェッチ機構などを用いることにより，より高い性能を達成することが可能であるが，より良いアルゴリズムの選択および実装は，マニュアルでデータ転送を行う場合と同様に，開発者の大きな負担となる．

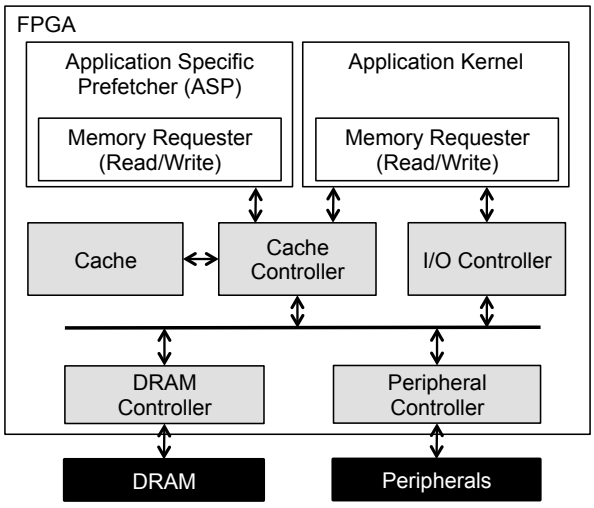


図 3 アプリケーション特化プリフェッチ機構をもつ FPGA アクセラレータ

我々は，キャッシュによりメモリシステムの抽象化を行いながら，高いメモリ性能を達成する手法を模索する．図 3 に，我々が提案するアプリケーションに特化したプリフェッチ機構を持つ FPGA アクセラレータの構成を示す．アクセラレータはアプリケーションのカーネルとキャッシュ，そしてアプリケーション特化プリフェッチャー (ASP: Application Specific Prefetecher) から構成される．キャッシュは通常のアプリケーションからのリクエストポートに加えて，プリフェッチ用のリクエストポートを持つ．しかしプリフェッチの場合はデータを下位のメモリ階層からキャッシュにデータを先行的に移動するだけであるため，キャッシュからの読み出しポートを追加する必要はない．アプリケーション特化プリフェッチャーは，アプリケーション内の状態遷移に同期して，プリフェッチリクエストをキャッシュに対して発行する．キャッシュは，アプリケーションからのリクエストがない場合はプリフェッチャーからのリクエストに基づいて，キャッシュラインの更新を行う．

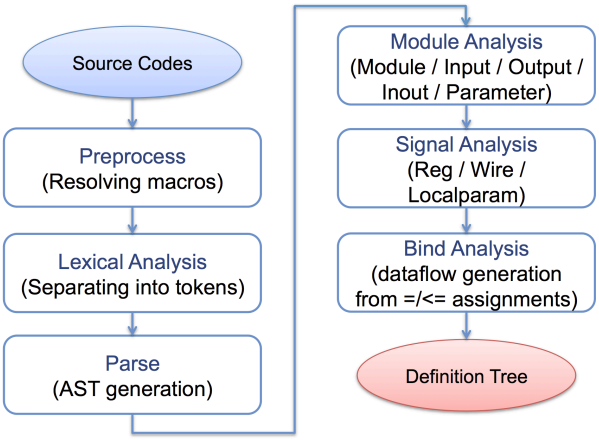


図 4 前処理・字句解析・構文解析・定義木生成のフロー

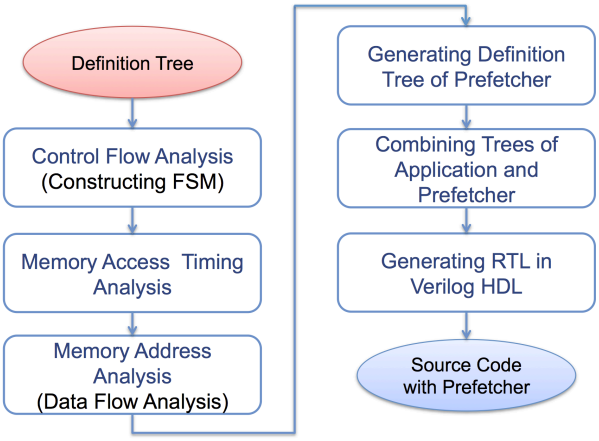


図 5 コントロールフロー解析・データフロー解析・プリフェッチャー定義木生成・コード生成のフロー

3. アプリケーション特化プリフェッチャーと RTL 解析ツール

3.1 解析およびコード生成の流れ

本章では，我々がアプリケションの RTL 記述からプリフェッチャーの RTL コードを生成するために用いた，RTL 解析ツールの構成と，生成されるについて述べる．アプリケーション特化プリフェッチャーが生成されるまでのフローを**図 5** および**図 ??**に示す．プリフェッチャー回路をもつアプリケーション RTL が生成されるまでの流れを以下に述べる．まず，Verilog HDL で記述されたアプリケーションのソースコードを入力として，(1) 前処理，(2) 字句解析，(3) 構文解析，(4) モジュール定義解析，(5) 信号の定義解析 (6) 各信号への代入解析を行い，各信号の定義木を生成する．次に，生成された定義木を元に，(7) コントロールフロー解析による状態遷移グラフの取得，(8) 状態遷移の各状態におけるメモリアクセスタイミング解析，(9) メモリアドレスに関するデータフロー解析，(10) プリフェッチャーの定義木生成，(11) プリフェッチャーおよびアプリケーションの定義木の合成, (12) 定義木から RTL コードへの変換，のステップを経て，最終的に，Verilog HDL で記述されたプリフェッチャー付きのアプリケーションのコードが生成される．

本稿で提案するアプリケーション特化プリフェッチャーは，アプリケーション中の反復処理に含まれるメモリアクセスを対象とする．(7) にて状態遷移グラフを取得し，その状態遷移グラフ中からループを探し出す．次に，(8) にてループに含まれる各状態においてメモリアクセスがあるかどうかを判定し，そのメモリアクセスが発生する条件を特定する．その後，(9) にて，ループを一周し，次に同じ状態遷移の状態に到達するときのメモリアクセスアドレスの定義木を，アドレス信号の定義木および，その定義に用いられる信号の定義と状態遷移から生成する．

本ツールの開発には Python を用いた．コード行数は，定義木の最適化処理器や定義木および状態遷移の可視化ツールなどを含めて，現時点でおおよそ 9000 行である．

3.2 プリフェッチャーの例

ここで，プリフェッチャーのコードの例を用いて，将来にアクセスするであろうアドレスの値を推論方法について解説する．**図 6** にメモリアクセスを制御する状態遷移のコードの例を示す．図 6 の例の場合，状態遷移中に，状態 1 における読み出し，および状態 4 における書き込みの 2 つのメモリアクセスが存在する．状態 1 における読み出し時のメモリアドレスは，変数 *cnt* によって定義され，状態 4 における書き込み時のメモリアドレスもまた変数 *cnt* によって定義される．また，変数 *cnt* は状態 6 において，4 ずつインクリメントされる．

図 7 にこの状態遷移におけるメモリアクセス情報から生成されるプリフェッチャーのコードを示す．生成されたプリフェッチャーのコードは，アプリケーションと同じソースコードに埋め込まれる．そのため，アプリケーションの定義と同じの変数を参照する．プリフェッチャーでは，ループが 1 周し，次回同じ状態に到達したときに発生するメモリアクセスのアドレス値を，現時点における変数の値を基準にオフセットを加えることによって定義する．例の場合，アドレス信号の定義に用いられている変数 *cnt* はループを 1 周する間に，4 インクリメントされるため，プリフェッチャーがリクエストするアドレス値は現時点の変数 *cnt* の値に 4 加えたものとなる．もし，アドレスの定義に用いられている変数の変化分が外部から入力などに依存し，解析できない場合にはプリフェッチは行わない．そのような場合に，汎用 CPU で用いられているストライドプリフェッチャーなどを導入するなどの方法の検討が，今後の課題として挙げられる．

4. 評価

本稿では，初期評価として，Verilog HDL で記述した簡

```
always @(posedge CLK) begin
  if(RST) begin
    state <= 0;
  end else begin
    if(state == 0) begin
      memory_write    <= 0;
      memory_read     <= 0;
      memory_dataout  <= 0;
      memory_address  <= 0;
      cnt             <= 0;
      state           <= 1;
    end else if(state == 1) begin
      memory_write    <= 0;
      memory_read     <= 1;
      memory_address  <= cnt;
      state <= 2;
    end else if(state == 2) begin
      memory_write <= 0;
      memory_read  <= 0;
      state        <= 3;
    end else if(state == 3) begin
      if(memory_read_done) begin
        state <= 4;
      end
    end else if(state == 4) begin
      if(!memory_write_full) begin
        memory_write    <= 1;
        memory_read     <= 0;
        memory_address  <= cnt + 1024;
        memory_dataout  <= 'hffff;
        state           <= 5;
      end
    end else if(state == 5) begin
      memory_write <= 0;
      memory_read  <= 0;
      state        <= 6;
    end else if(state == 6) begin
      state <= 1;
      cnt   <= cnt + 4;
    end
  end
end
```

Read

Write

Source of Address

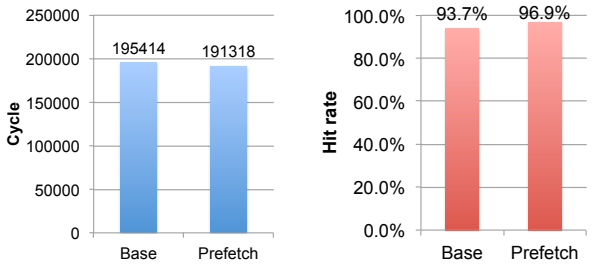
図 6 Verilog HDL で記述したメモリアクセスを制御する状態遷移コード例

```
always @(posedge CLK) begin
  if(RST) begin
  end else begin
    if(state == 1) begin
      prefetch_enable <= 1;
      prefetch_address <= (cnt + 4);
    end else if(state == 4) begin
      if(!memory_write_full) begin
        prefetch_enable <= 1;
        prefetch_address <= (cnt + 4) + 1024;
      end
    end
  end
end
```

図 7 生成されるプリフェッチ用コード例

単なベンチマークを用いて，提案手法による性能向上の度合いを評価する．

性能およびキャッシュヒット率を Icarus Verilog[1] を用いてシミュレーションにより評価する．ベンチマークにはベクター加算を用いた．キャッシュには，C++で記述した



(a) 実行サイクル数　(b) キャッシュヒット率

図 8 実行サイクル数とキャッシュヒット率

サイクルレベルのタイミングシミュレータを VPI (Verilog Programming Interface) を介して HDL シミュレーションに組み込み使用した．キャッシュの構成は，ラインサイズを 64 バイト，ウェイ数を 4，キャッシュ容量を 16K バイト，アクセスレイテンシを 1 とした．メインメモリには，アクセスレイテンシは 16 サイクル固定としたシンプルなモデルを用いた．ベクター加算の扱うデータのメモリフットプリントは 96K バイトとした．1 回のベクター加算の処理には 8 サイクルのレイテンシを要するもとして，演算はパイプライン化されていないものとした．

図 8(a) に基準のアプリケーションの実行サイクル数とプリフェッチャーを用いた場合の実行サイクル数を示す．また，図 8(b) に両者のキャッシュヒット率を示す．プリフェッチャーの導入により，2.1%の性能向上を達成した．またキャッシュヒット率が 3.1%向上した．性能向上率が伸び悩んだ理由としては，キャッシュが許可するアウトスタンディングミスの数を 1 としたため，プリフェッチリクエストが後続の読み出しを妨害したことと，今回のプリフェッチ対象が，ループ中の同状態における次回のアクセス先であったため，時系列において後続のリクエストに対する先行読み出しが行えなかったことなどが挙げられる．前者を回避するには，アプリケーションカーネルのリクエストを優先し，カーネルからリクエストが発行された場合には，プリフェッチャー側の処理をアボートするなどの処置を施すことなどが必要である．後者を回避するには，時系列順に次のアクセスを対象としてプリフェッチするようなプリフェッチャーの構成を検討する必要がある．

5. 関連研究

FPGA 向けのメモリシステムの最適化の研究としては，Samuel ら [2] による，高位合成言語で記述されたカーネルのコースコードを解析し，オフチップ SDRAM へのメモリアクセスを並べ替えることにより，メモリバンド幅を有効利用する方式や，Eric ら [3] による抽象度の高いメモリモデルを用いてアプリケーションを記述し，外部メモリとのカーネルの間にキャッシュとデータ転送機構を自動的に挿入するフレームワークの CoRAM などが挙げられる．前者は，高位合成系をターゲットしており，またループ中のイ

ンデックスにのみ着目して最適化を施す点で本研究と異なる．後者は，アプリケーションの記述を容易にする点では本研究とは類似しているが，キャッシュのデータの先読み等は行わない点で本研究とは異なる．

また，マルチコアや SMT プロセッサ上で，本来のアプリケーションのスレッドとは別に，キャッシュのプリフェッチを行うことを目的としたヘルパースレッディングという手法がある [4], [5]．本研究は，FPGA アクセラレータのアプリケーションに対するヘルパースレッディングととらえることも可能であり，これらの研究で提案された手法は同様に活用できると考えられる．

6. まとめ

本稿では，FPGA アクセラレータ向けアプリケーション特化プリフェチャーの生成手法の提案および，プリフェチャーによる性能向上率の初期評価を行った．

今後の課題として，より現実的なアプリケーションを複数用いた評価を行うこと，プリフェチャーの回路面積などの評価などを行うことが不可欠である．また，既存のプリフェッチ技術に対する優位性を定量的に評価する必要がある．現状の解析ツールでは Verilog HDL のフルセットを解析することができないため，高位合成系などで生成した RTL 記述からプリフェッチャーを構成することができない．より現実的な評価を行うためにはツール実装の改善が求められる．また，より高いメモリ性能を達成するために，キャッシュの置き換えアルゴリズムやラストユース予測などのハードウェアを静的解析の結果に基づいて構成する手法を検討したい．

謝辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の「ディペンダブルネットワークオンチッププラットフォームの構築」の支援による．

参考文献

[1] Williams, S. and Baxter, M.: Icarus verilog: open-source verilog more than a year later, *Linux J.*, Vol. 2002, No. 99, pp. 3– (online), available from ⟨http://dl.acm.org/citation.cfm?id=513581.513584⟩ (2002).

[2] Bayliss, S. and Constantinides, G. A.: Optimizing SDRAM bandwidth for custom FPGA loop accelerators, *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA '12, New York, NY, USA, ACM, pp. 195–204 (online), DOI: 10.1145/2145694.2145727 (2012).

[3] Chung, E. S., Hoe, J. C. and Mai, K.: CoRAM: an in-fabric memory architecture for FPGA-based computing, *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, New York, NY, USA, ACM, pp. 97–106 (online), DOI: 10.1145/1950413.1950435 (2011).

[4] Lu, J., Das, A., Hsu, W.-C., Nguyen, K. and Abraham, S. G.: Dynamic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor, *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, Washington, DC, USA, IEEE Computer Society, pp. 93–104 (online), DOI: 10.1109/MICRO.2005.18 (2005).

[5] Kamruzzaman, M., Swanson, S. and Tullsen, D. M.: Inter-core prefetching for multicore processors using migrating helper threads, *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, New York, NY, USA, ACM, pp. 393–404 (online), DOI: 10.1145/1950365.1950411 (2011).