博士学位論文要約（平成29年３月）

ストリーム処理のための
高スループットFPGAアクセラレータに関する研究

立見 駿介
指導教員：青木 孝文

High-Throughput FPGA Accelerator for Stream Processing

Shunsuke TATSUMI
Supervisor: Takafumi AOKI

Processing large amounts of data arriving continuously is called stream processing, which is used for vairous applications such as video surveillance, image measurement, and financial analysis. Although stream processing is mainly implemented using CPUs and GPUs, achieving both high performance and low power is a major problem. Addressing this problem, this paper utilizes FPGAs that are reconfigurable LSIs. This paper proposes efficient design methods for maximizing the performance based on constraints of the external-memory bandwidth and hardware resources. The proposed methods are applied for FPGA implementations of phase-only correlation, particle filter, and PatchMatch multi-view stereo to demonstrate the energy efficiency.

1. はじめに

データ発生源から絶え間なく到来するデータを逐次的に処理していく処理方法はストリーム処理と呼ばれ，映像監視，画像を用いた計測および認識，市場シミュレーションなど幅広い応用において扱われている．このようなストリーム処理応用の拡大に伴い，大量のデータを高速に処理できる高スループットな演算器が求められている．

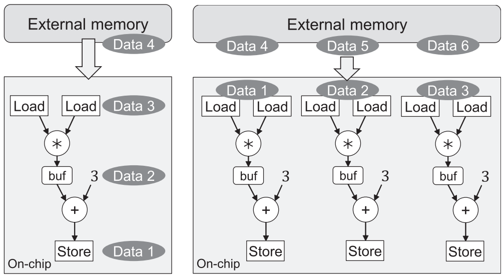
ストリーム処理の実装に用いられる主な演算器として CPU と GPU が挙げられる．CPU は，高い単一スレッド性能と実装容易さを持つが，消費電力当たりの性能の低さが問題であった．GPU は，並列処理による高い演算性能が特長であるが，消費電力の大きさが問題であった．つまり，両演算器において，高性能と低消費電力の両立が課題であった．これを解決するために，本論文では FPGA に着目する．

FPGA は，論理回路の構成を変更できる演算器であり，アルゴリズムをハードウェア実装できるため消費電力当たりの性能が優れていることが特長である．一方で，設計自由度が非常に高いため，性能を十分に引き出すための適切な設計が難しいことが問題であった．FPGA の性能を制約する要因は主に２つある．1 つは外部メモリ帯域であり，一度に FPGA チップへ読み込めるデータ量はこれに制約される．もう１つはハードウェアリソース量であり，並列に処理できるデータ数はこれに制約される．

そこで，本論文では，ストリーム処理向け FPGA アクセラレータ設計において，限られた外部メモリ帯域およびハードウェアリソースを効率よく利用できる設計手法を確立する．さらに，位相限定相関法 [1)]，パーティクルフィルタ [2)]，PatchMatch Multi-View Stereo (MVS) [3)] に対して提案設計手法を適用した FPGA 実装をそれぞれ行い，その有用性を示す．

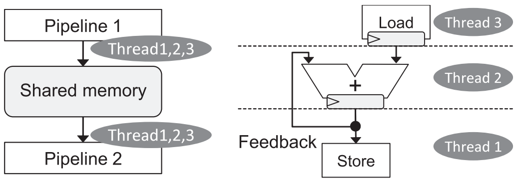
2. 高スループット FPGA アクセラレータ設計の基本方針

FPGA は，多数の加算器と DSP ブロックを持ち，並列処理によって高い性能を実現する．そこで，大量データを高スループットで処理するために，まず処理データをデータ並列処理が可能な小データへ分割する．一般に処理データは外部メモリへ置かれるため，並列処理のためには外部メモリから FPGA チップ内へ多数のデータを読み込まなければならない．しかしながら，FPGA は GPU と比較して外部メモリ帯域が非常に小さいことが問題となる．そこで，図 1(a) に示すように，各小データを連続的に到来するデータストリームとして読み込む．規則的かつ連続的なアドレスからデータを読み込むことにより外部メモリ帯域を効率よく扱える．さらに，処理回路をパイプライン実装することでデータの流れを止めることなく処理を行う．パイプライン実装によって，各小データに対する読み込みと処理を時分割して並列に行えるため，外部メモリ帯域を効率良く使用しつつ多数データを並列処理できる．また，全てのデータの処理



(a) パイプライン実装　(b) パイプラインを複製した場合

図 1: FPGA に適したハードウェア実装



(a) 共有メモリによる方法　(b) 明示的にデータ転送する方法

図 2: スレッド間のデータ共有方法

において同一パイプラインを再利用するため，ハードウェアリソース効率の良い実装ができる．リソースに余裕がある場合，図 1(b) に示すように，パイプラインを複製することで同時に処理できるデータ数が増加するため，スループットを向上できる．

異なる小データに対する処理の間にデータ依存関係が存在する場合，処理間でデータをやり取りする必要がある．これ以降，小データ（同一パイプラインステージのデータ）に対する処理の単位をスレッドと呼ぶ．図 2 に示すように，スレッド間でデータを共有する方法には共有メモリを用いる方法と明示的にデータ転送する方法が挙げられる．共有メモリを用いる手法は，前パイプラインにおけるスレッドが結果を共有メモリへ書き込み，全スレッドの書き込み終了後に次パイプラインの処理を開始することで次パイプラインで結果を共有できる．明示的にデータ転送する方法は，前スレッドの結果をフィードバックを用いて直接的に後スレッドへ渡すことで結果を共有する．明示的なデータ転送では共有メモリが不要でありハードウェアリソース効率が良いため，可能な限りこちらの方法を用いることが望ましい．

実装アルゴリズムに複数のタスクが含まれる場合，タスク毎にパイプラインを作成することで実装できる．ここで，あるパイプラインが異なるパイプラインの結果を使用する場合，結果を外部メモリを介さず内部メモリで転送することが望ましい．この場合，データの流れを止めずそのまま結果を受け渡すためには FIFO バッファが適している．また，FIFO バッファが空もしくは満たされているとき読み込みもしくは書き込みを行わないよう，同期的な転送機能を備えた FIFO バッファが望ましい．

以上のアーキテクチャがストリーム処理向け FPGA アクセラレータに適していると考えられる．

FPGA の性能は外部メモリ帯域およびハードウェアリソースによって制約される．そこで，これらの制約下で 1 スレッドが処理するデータ数を最大化することでスループットを最大化できる．一方で，制約のため，並列処理できるデータ群をさらに小さなデータ群へ分割しスレッドに割り当てる場合がある．このときは，明示的なデータ転送によってフィードバックによる効率良いスレッド間データ共有を行いながら処理を行うことが望ましい．高スループット化のための設計方針は以下のようにまとめられる．

性能のボトルネック

- 外部メモリ帯域: 外部メモリアクセスを最適化
- ハードウェリソース: リソース効率の良い設計

スレッド間のデータ依存

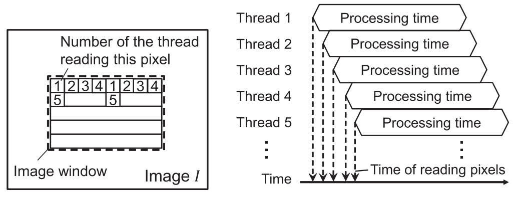
- 有り: 問題なし
- 無し: フィードバックによる効率良いデータ共有

本論文では，効率よい FPGA 設計のため高位合成ツールとして OpenCL を用いる [4)]．OpenCL では，各タスクの実装単位はカーネルと呼ばれ，カーネル間のデータ転送は “チャネル” と呼ばれる同期的な転送機能を備えた FIFO バッファで実現できる．

3. 位相限定相関法の FPGA 実装

位相限定相関法は，画像対応付け手法の 1 つであり，精度とロバスト性の高さからステレオビジョンや生体認証など様々な応用に使用されている．一方で，本アルゴリズムはフーリエ変換に基づく手法であるため，計算コストの大きさに伴う処理時間および消費電力が問題であった．これを解決するために，位相限定相関法のための FPGA アクセラレータを提案する．

位相限定相関法は，2 枚の画像間において参照画像の基準点に対応する点をもう一方の対象画像から推定する手法であり，ウィンドウマッチングによる推定を行う．このとき，対応付けは基準点ごとに全て独立して実行できる．一方で，外部メモリ帯域およびハードウェアリソース制約から，1 基準点の対応付け処理を 1 スレッドに割り当てることは望ましくない．そこで，図 3 に示すように，対応付けに用いる画像ウィンドウの処理を複数スレッドへ分散して割り当てる．このとき，スレッド間でデータ依存関係が生じるため，フィードバックによる効率よいデータ共有を行う．



(a) 画像ウィンドウの分割　(b) 各スレッドのタイムチャート

図 3: 画像ウィンドウに対する処理の割り当て方法

表 1: CPU，GPU，FPGA の各実装における性能

	Core i7 (12-thr.)	Geforce GTX 680	P395 D8 FPGA
処理時間 (ms)	49.09	11.51	10.39
消費電力 (W)	147.83	130.33	16.13
PD 積 (W×s)	7.26	1.50	0.17

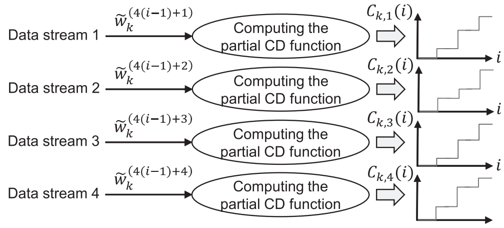
提案 FPGA 実装は複数カーネルによって実装されるため，カーネル間ではチャネルを用いてデータ転送を行う．処理データは各カーネルで逐次的に処理されていくが，相関および平均計算のためデータ数が変化する．そこで，これに応じて各カーネルのスループットを合わせることで無駄のない実装を行う．

提案 FPGA 実装を評価するために CPU および GPU 実装と比較する．CPU は Intel Corei7-3960X，GPU は NVIDIA Gerforce GTX680，FPGA は StratixV D8 搭載の Nallatech PCIe-395 D8 ボードを使用する．FPGA 実装では，外部メモリ帯域およびハードウェアリソース制約の下，1 スレッドに割り当てるデータ数を最大化した．表 1 に実装結果を示す．FPGA 実装によって，12 スレッド CPU 実装の 4.7 倍，GPU 実装の 1.1 倍の高速化が達成できた．さらに，FPGA 実装の電力遅延積（PD 積）は，12 スレッド CPU 実装の 2.3%，GPU 実装の 11%まで削減できた．以上から，提案 FPGA 実装によって高速かつ低消費電力な処理が実現できたことを確認できた．

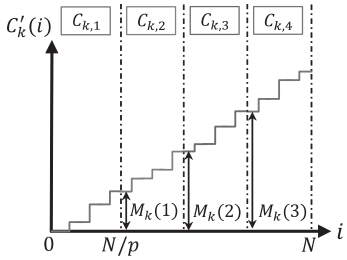
4. パーティクルフィルタの FPGA 実装

パーティクルフィルタは，一般状態空間モデルにおける状態分布を推定する手法の 1 つであり，線形・非線形を問わずシステムモデル・ノイズモデルを扱える高い汎用性からロボティクスや金融分析など幅広い応用で用いられている．そのため，車載システムや家電製品など低消費電力が要求される応用においてもニーズが高まっている．そこで，パーティクルフィルタのための FPGA プラットフォームを提案する．

パーティクルフィルタは外部メモリから読み込むデータ量が小さいため，性能は 1 スレッドに割り当てるデータ数（並列数）に依存する．そこで，並列数を増やすことで高スループット化を目指す．



(a) 部分的な累積分布関数（CD function）の並列計算



(b) 部分累積分布関数から求めた累積分布関数

図 4: 並列数 4 のときの累積分布関数の並列計算

表 2: CPU 実装と FPGA 実装の性能比較

	Core i7 (1-thr.)	DE5 FPGA
処理時間 (ms)	15.13	0.14
消費電力 (W)	56.8	67.9
PD 積 (W×s)	0.86	0.0095

表 3: 並列数に対する処理時間の変化

	DE5 FPGA	
並列数	1	4
合計の処理時間 (ms)	76.74	21.98

パーティクルフィルタは対象システム状態の候補である粒子を多数用いて状態を近似して推定を行うため，処理の大半は粒子ごとに並列に処理できる．一方で，残す粒子を選択するリサンプリング処理において，全粒子の重み（尤度）の累積分布関数を計算するために直列的な加算が必要となる．そこで，本論文では，累積分布関数を効率よく並列計算できる手法を提案し，1 スレッドが複数粒子を並列処理する．

図 4 に累積分布関数の効率よい並列計算方法を示す．まず入力データストリームごとに重み $\widetilde{w}_k^i$ の部分的な累積分布関数 $C_{k,j}(i)$ を計算し，次に各部分累積分布関数の最大値 $M_k(j)$ に対して累積分布関数を計算する．これら 2 つの部分的な累積分布関数を加算することで，全体の累積分布関数を並列に計算できる．

提案 FPGA 実装を評価するために，CPU 実装と性

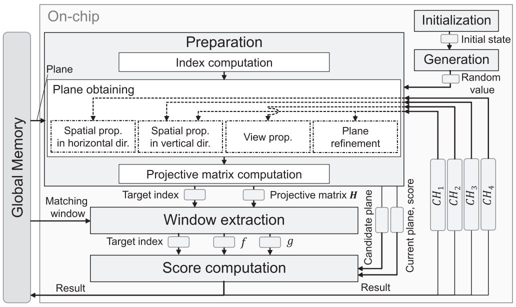


図 5: カーネル全体の構造

能を比較し，並列数を変化させた場合の性能を評価する．CPU は Intel Corei7-3960X，FPGA は StratixV A7 搭載の Terasic DE5-Net ボードを使用する．表 2 に 1 時間ステップの推定における性能の比較を示す．FPGA 実装によって，1 スレッド CPU 実装の 106 倍の高速化を達成し，電力遅延積（PD 積）を 1.1%まで削減できた．表 3 に並列度の変化に伴う時間ステップ 1–1000 までの合計の処理時間の変化を示す．提案並列実装によって，並列度を 1 から 4 へ増やすことで 3.5 倍の高速化が達成できた．

5. PatchMatch Multi-View Stereo の FPGA 実装

近年，計算機の性能向上に伴い，データ量の大きい 3 次元形状データが産業･医療･芸術などの様々な分野で利用されている．PatchMatch Multi-View Stereo (MVS) は，多視点画像から深度マップを作成する手法の 1 つであり，撮影対象や環境の制約が少なく安価なセンサで推定できる利点から様々な応用への利用が期待されている．一方で，全多視点画像の全ピクセルにおいて複数回のウィンドウマッチングが必要であるため，計算量の大きさが問題であった．そこで，高速かつ低消費電力のために PatchMatch MVS のための FPGA アクセラレータを提案する．

PatchMatch MVS は外部メモリから読み込むデータ量が大きいため，性能は外部メモリ帯域によって制約される．さらに，本アルゴリズムは各ピクセルの推定結果を隣接ピクセルへ伝播することで効率良い推定を行うが，このため外部メモリアクセスが頻繁になる．そこで，FPGA の柔軟なオンチップメモリを活用することで外部メモリアクセスを改善し，高スループット化を目指す．また，外部メモリ帯域制約によってスレッド間でデータ依存関係が生じるため，フィードバックを用いた効率良いデータ転送を行う．

図 5 に提案 FPGA 実装のカーネル構造を示す．Preparation, Window-extraction, Score-computation カーネルは 1 ピクセルに対する推定処理を行うパイプラインを構成し，Score-computation カーネルから推定結果が出力される．推定結果を伝播する際は，結果をチャネル $CH_1, \cdots, CH_4$ を用いて戻すことによってオンチップ内で再利用する．これにより，外部メモリアクセスを減らしかつ低レイテンシで結果を再利用できるため，性能向上に繋がる．

提案 FPGA 実装を評価するために，CPU および GPU 実装と比較する．演算器は 3 章と同様のものを用いる．FPGA 実装では，外部メモリ帯域およびハードウェアリソース制約の下，1 スレッドに割り当てるデータ数を最大化した．表 4 に実装結果を示す．FPGA 実装によって，12 スレッド CPU 実装の 6.8 倍，GPU 実装の 2.5 倍の高速化が達成できた．さらに，FPGA 実装の電力遅延積（PD 積）は，12 スレッド CPU 実装の 2.1%，GPU 実装の 6.9%まで削減できた．以上から，提案 FPGA 実装によって高速かつ低消費電力な処理が実現できた．

表 4: CPU，GPU，FPGA 実装の性能比較

	Core i7 (12-thr.)	Geforce GTX 680	P395 D8 FPGA
処理時間 (s)	127.0	47.3	18.7
消費電力 (W)	138.3	115.0	20.1
PD 積 (W×s)	17558.6	5439.8	374.9

6. まとめ

本論文では，ストリーム処理向け FPGA アクセラレータ設計において，限られた外部メモリ帯域およびハードウェアリソースを効率的に利用できるアーキテクチャの設計手法についてまとめ，それらの制約に基づいた高スループット化のための設計手法を確立した．また，位相限定相関法，パーティクルフィルタ，PatchMatch MVS の FPGA 実装を通じて提案設計手法の有用性を実証した．

文献

1) K. Takita et al., “High-accuracy subpixel image registration based on phase-only correlation,” *IEICE Trans. Fundamentals*, Vol. E86-A, No. 8, pp. 1925–1934, August 2003.
2) N.J. Gordon et al., “Novel approach to nonlinear/non-Gaussian Bayesian state estimation,” *IEE Proc. F*, Vol. 140, No. 2, pp. 107–113, August 1993.
3) M. Hiradate et al., “An Extension of PatchMatch Stereo for 3D Reconstruction from Multi-View Images,” *Proc. Asian Conf. on Pattern Recognition*, No. PS1-02, November 2015.
4) T.S. Czajkowski et al., “OpenCL for FPGAs: Prototyping a Compiler,” in *Proc. Int. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 3–12, July 2012.