FPGA アクセラレーションを可能にするストリーミングフレームワークの拡張

味曽野　智礼[†]　吉瀬謙二[†]

1. はじめに

近年、回路構成を変更できる FPGA によるアクセラレータの研究が多く行われている。この理由としては、FPGA アクセラレータが特定のアプリケーションに対して電力消費量を抑えながら高い性能を実現できることや、高位合成技術の進歩により FPGA アクセラレータの実装難易度が下がり始めてきていることが挙げられる。特に FPGA アクセラレータはストリーミング処理に向いていると言われており、リアルタイムデータ処理の高速化が期待できる。しかし現在、ストリーミング処理を行うソフトウェアから FPGA アクセラレータを簡単に使用できる環境は無い。

そこで本研究ではストリーミングフレームワークの 1 つである Heron[1] を拡張し、FPGA リソースを扱えるようにする。PC と FPGA ボード間の通信は PCIe を介して行なう。初期評価として、拡張したソフトウェアが仮想マシン上で正しく動作することを確認し、また CPU のみによる処理と比較して FPGA によるアクセラレーションが達せられることを示す。

2. 背　　景

2.1 ストリーミングフレームワーク

ベースとなるフレームワークとして Heron を使用する。Heron はオープンソースの分散ストリーミングフレームワークであり、クラスタ管理に Mesos[2] を、ジョブスケジューラとして Aurora[3] を使用する ☆。

Heron ではユーザーアプリケーションを非循環グラフとして記述する (**図 1**)。ここで全体を Topology、他のノードから入力を受け取らずにデータを出力するノードを Spout、入力を受け取りその処理結果を出力するノードを Bolt と呼ぶ (出力しなくても構わない)。Topology は Java か Python で記述される。ユーザーはそれぞれの Spout や Bolt に対応するクラスを作成し、main 関数において Topology の構成情報を定義する。ノード間の通信には Protocol Buffer が用いられており、通信単位はユーザーが決めることが出来る。例えば Bolt を作成する際は、ベースとなる抽象クラスを継承し、入力データが届く度に呼ばれる execute 関数をオーバーライドして処理内容を記述する。

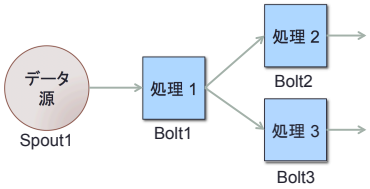

図 1　Heron topology の例

2.2 ホスト PC-FPGA 間の通信

本研究ではホスト PC と FPGA が PCIe を介して接続されている環境を想定する。PC-FPGA 間の通信には RIFFA[4] を用いる。これはオープンソースの PCIe インタフェース (FPGA の PCIe パケット処理回路およびデバイスドライバ) を提供する。様々な FPGA ボードを対象としており、最大で Gen2x8 の通信を行なうことが出来る。

3. 拡張方針

Heron ユーザーが特定の Bolt で FPGA を使用できるようにフレームワークを拡張する。ユーザーから見た時の要件を以下のように定める。

- ユーザーはあらかじめ FPGA アクセラレータの回路を作成し、コンフィギュレーションファイルを用意しておく。
- FPGA を使用する Bolt を FPGABolt 抽象クラスを継承して作成する。
- 初期化時に呼ばれる prepare 関数でアクセラレータのコンフィギュレーションファイルを指定する。
- execute 関数において、FPGA に対してデータの送受信 (send/recv 関数を使用) を行なう。
- ユーザーから FPGA に対しては、reset, send,

† 東京工業大学大学院 情報理工学研究科

☆ クラスタ管理やスケジューラには他に yarn 等を使用可能であるが、本研究では Mesos/Aurora を使用する

表 1 開発に用いるソフトウェアのバージョン

Mesos	0.28.2
Aurora	0.15.0
Heron	0.14.2
RIFFA	2.2.1

recv 以外の処理は行わない。

実際の FPGA の制御はユーザーから見えない部分に記述することで、将来的に FPGA インタフェースが変わった場合にもユーザーアプリケーション側の変更を最小にする。

4. 初期実装

前節の方針に従い、ソフトウェアに変更を加えた。主要な変更点は以下の通りである。

- Mesos にカスタムリソースとして FPGA を追加
- Aurora のリソース情報に FPGA を追加
- Heron に FPGABolt 抽象クラスを追加
- FPGABolt クラスが RIFFA を用いて FPGA との通信を行なうコードを追加

Mesos にはカスタムリソースを追加する仕組みがすでにあるため、実際にコードの修正を行ったのは Aurora と Heron に対してのみである。開発に用いるソフトウェアのバージョンは**表 1** の通りである。

今回の初期実装では 1 台の PC に 1 台の FPGA ボードのみ接続されることを前提にしており、あるアプリケーションが実行中はそれが FPGA を専有する。また簡単のため FPGA の動的コンフィギュレーションはサポートしておらず、事前に FPGA に対してコンフィギュレーションを行なう。

5. 評価

5.1 評価環境

拡張したソフトウェアをインストールした 1 台の仮想マシン上で評価を行う。OS は Ubuntu 14.04、CPU は Xeon E5 2687W であり、Xilinx Virtex-7 を搭載する vc707 評価ボード (PCIe Gen2x8) を 1 台接続する。CPU の VT-x を有効にし、また PCIe パススルーを用いて仮想マシンから FPGA ボードを使用する。

5.2 アプリケーション

評価アプリケーションとして 8 ビットグレースケール RAW 画像に対する 5 点メディアンフィルタを使用する。FPGA アクセラレータのコア回路の実装には Vivado HLS 15.4 を用い、プロジェクト全体は Vivado 15.4 で論理合成を行なう。

評価する topology は Java で記述し、画像データを出力する Spout を 1 つ、メディアンフィルタの処理を行う Bolt1 つから構成する。ただし Spout から出力する画像データは機械的に生成した値とし、各 Spout、Bolt には CPU2 コア、メモリ 2GB を割り当てる。

表 2 メディアンフィルタ処理を行なう Bolt の execute 関数の平均実行時間 (ms)

サイズ	512x512	1024x1024
CPU	151	688
FPGA	0.59	1.83

5.3 結果

メディアンフィルタの処理を CPU1 スレッドで行う場合と FPGA を用いる場合のそれぞれにおいて、10 分間動作させた時の、Bolt の execute 関数 1 回分の平均実行時間を**表 2** に示す。これはすなわち 1 枚の画像に対する平均処理時間である。 FPGA による実行結果とソフトウェアによる実行結果の値が一致する事を確認した。結果より、FPGA アクセラレータを用いることにより数百倍の高速化が達成されることが分かる。メディアンフィルタでは各要素を中心とした領域毎にソートを行なうために CPU の処理コストが高く、各要素の値を並列に計算することが出来る FPGA アクセラレータが高い性能を達成したと考えられる。

6. まとめ

ストリーミングフレームワークである Heron を拡張し、FPGA アクセラレータが使用できることを仮想マシン上で確認した。今後の目標はクラスタ環境での評価や、より現実的なアプリケーションを用いた評価を行なうことある。

参考文献

1) Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter Heron: Stream Processing At Scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 239–250, 2015. ACM.
2) Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform For Fine-Grained Resource Sharing in The Data Center. Technical Report, 2010.
3) Apache Aurora. http://aurora.apache.org/.
4) M. Jacobsen and R. Kastner. RIFFA 2.0: A Reusable Integration Framework For Fpga Accelerators. In *2013 23rd International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, Sept 2013.