アプリケーションループ文のFPGA自動オフロード手法の評価

2020年2月21日

NTTネットワークサービスシステム研究所　特別研究員

山登庸次

目次

1 はじめに

- 近年、ムーアの法則終焉が近いと言われており、それも踏まえて、GPUやFPGAといったヘテロなハードウェアの活用が増えている。
- IoTデバイスは増加し、2020年に3百億、2030年に兆と言われる。
- GPUやFPGAには、CUDA、OpenCLと言ったスキルが、IoTデバイスの細かい制御は組み込みソフトウェア知識が必要で障壁高い。
- これらを容易に活用するため、プログラマーは処理したいロジックだけ書き、配置環境に合わせて、ソフトウェアが適応して動作が求められる。
- Javaは、環境適応を実現したが、性能は考慮されていなかった。
- 私は、一度書いたソフトウェアを、配置先環境でコード変換等を自動で行う事で、高性能動作させる、環境適応ソフトウェアを提案している。
- 本稿はその中で、ループ文のFPGA自動オフロードを検討する。

2 既存技術

- Javaは、移行した先の性能チューニングやデバッグ等の稼働は小さくないことが課題であった（Write Once, Debug Everywhere）。

- GPGPU向けにCUDAが、ヘテロハード向け仕様にOpenCLがある。指示行仕様としてOpenACC、その解釈にPGIコンパイラがある。
- これらにより、GPU、FPGA処理は可能になっても、高性能化は難しい。
  - 自動並列化機能コンパイラは、ループ文等の並列可能部を抽出するが、メモリ間のやり取りオーバヘッドのため性能が出ないことも多い。

- IoTデバイスでは、アセンブリ等の組み込みソフトウェア知識が必要。
- Rasberry Pi等のシングルボードコンピュータでは、どのように処理を分担するか等は、利用形態によって異なり、環境に合わせた設計が必要。

2 本稿の目的とContribution

目的

- 一度書いたソフトウェアを、配置先環境でFPGAを活用できるように、コード変換等を自動で行う事で、高性能で動作させる。

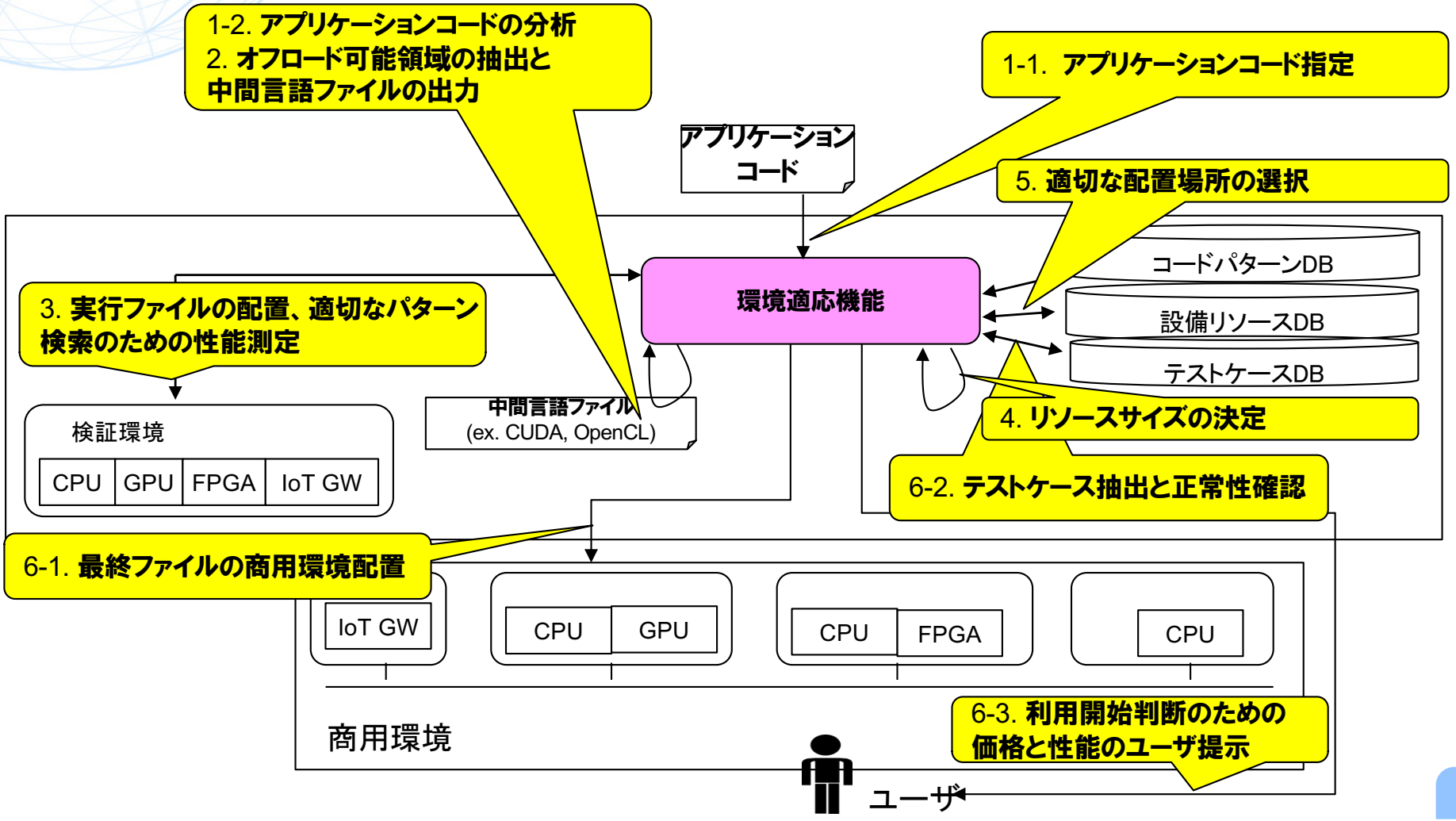
Contribution

- GPUでの自動オフロード手法も参考に、コンパイルに長時間かかることを考慮し、ループ文のFPGA自動オフロード手法を提案。

- 既存アプリケーションでFPGAオフロードの有効性を検証する。

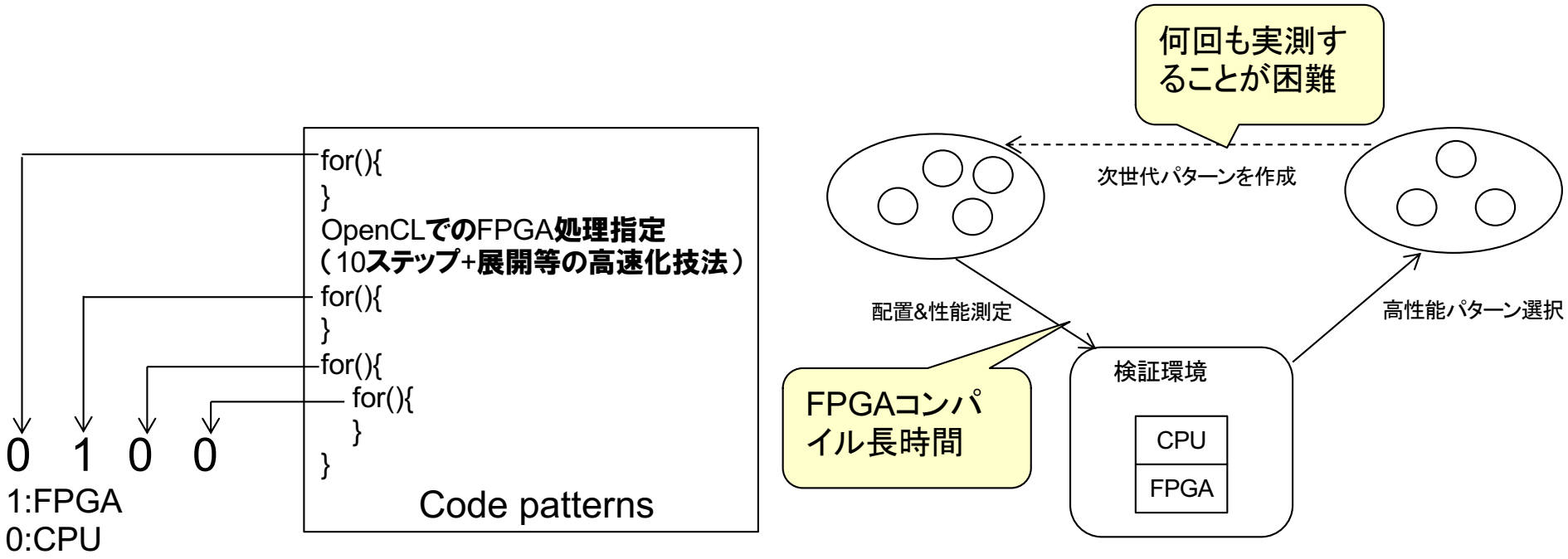
3-1　環境適応ソフトウェアの処理フロー

- Step1-3で変換、4で量、5で場所を決め、6で検証配置。
- 運用中に、性能をモニタして、定期的にStep1-5を試行し、再構成が有効な場合に、7として再構成を提案。



3-2 ループ文FPGA自動オフロード考慮点

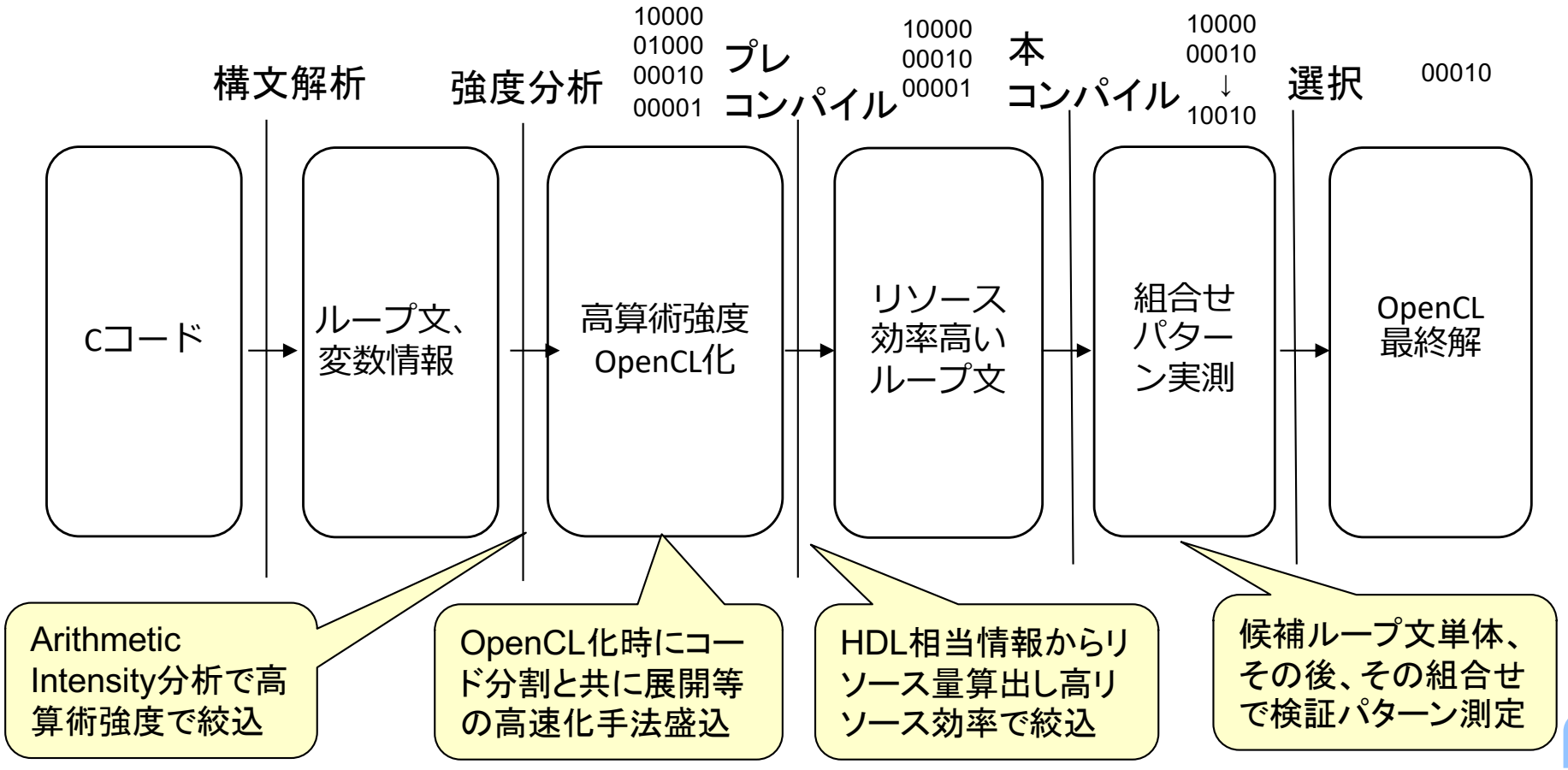
- 処理時間が多く費やされるループ文をまず対象とするが、FPGAはクロックが遅いこともあり、単純に並列可能ループ文をオフロードしても高速化できないことが多い。
- GPU時と同様に、検証環境で性能を実測し、高速パターンを探索する方式をとるが、FPGAではコンパイルに長時間かかるため、GPUのように多パターンを測定できない。



FPGA処理はOpenCLによる高位合成ツール利用（Intel FPGA SDK for OpenCL等）

3-2ループ文FPGA自動オフロード手法

- コンパイルが長時間の対応のため、オフロード候補とするループ文を絞り込んでから、実測して高性能パターン選択する。
- コード分析→ループ文発見→高算術強度ループ抽出→OpenCL化→高リソース効率ループ絞り込み→絞り込みループで複数検証パターン→実測し高性能パターン選択



4 実装（利用ツール等）

NTT

- 3節の方式を以下のツールを用いて以下条件で実装
  - Python 2.7：構文解析含む全処理
  - FPGA：Intel PAC with Intel Arria10 GX FPGA
  - FPGA制御：Intel Acceleration Stack Version 1.2
    - Intel FPGA SDK for OpenCL 17.1.1，Quartus Prime Version 17.1.1
  - 構文解析：Clang 6.0
    - 対象アプリ：C, C++
  - Arithmetic Intensity分析：PGIコンパイラ 19.4

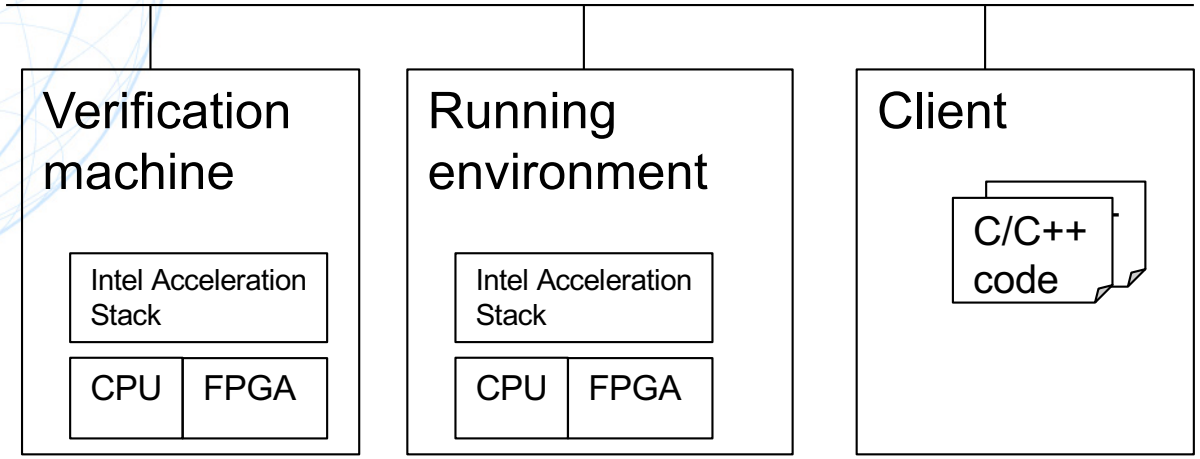
  - OpenCL化時の高速化手法：ループ文展開処理
  - 絞込ロジック：Arithmetic Intensity/リソース使用率 が高いc個選定
    - 計算密度が10，リソース量が0.5のループ文は10/0.5=20がリソース効率
  - 実測パターン作成数はd個上限に、候補ループ文単体と組合せ
    - 1と3番目のループが高リソース効率であったとしたら，1番をオフロード，3番をオフロード，1と3番両方をオフロードのOpenCLパターン作成

5　評価対象と条件

- 時間領域有限インパルス応答フィルタ、画像処理MRI-Q

- 条件
  - オフロード対象：ループ文数（時間領域有限インパルス応答フィルタは36、MRI-Qは16）
  - 算術強度絞り込み：算術強度分析の上位5つのループ文に絞り込み
  - ループ文展開数：1（検証ではOpenCLでのFPGAオフロードした効果だけ確認するため）
  - リソース効率絞り込み：リソース効率分析の上位3つのループ文に絞り込み（算術強度/リソース量が高い上位3つループ文を選定）
  - 実測オフロードパターン数：4（1回目は上位3つを測定し、2回目は1回目で高性能だった2つの組み合わせパターンで測定）

5 評価環境



Name	Hardware	CPU	RAM	FPGA	OS	Intel Acceleration Stack
Verification machine	Dell PowerEdge R740	Intel Xeon Bronze 3104	32GB*2	Intel PAC with Intel Arria10 GX FPGA	CentOS 7.4	1.2
Running environment	Dell PowerEdge R740	Intel Xeon Bronze 3104	32GB*2	Intel PAC with Intel Arria10 GX FPGA	CentOS 7.4	1.2
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 7 Professional	

5　性能結果

- 時間領域有限インパルス応答フィルタ、MRI-Q高速化
- 複数検証パターン測定を元に、各4倍、7倍の高速化
- 性能測定数は4つのため、半日で自動検証が完了

	Performance improvement of this paper implementation
Time domain finite impulse response filter	4.0
MRI-Q	7.1

6 まとめ

- アプリケーションを環境に適応させ、FPGA等を適切に活用し、高性能に動作させるための環境適応ソフトウェアを提案している。
- GPUに続き、ループ文のFPGA自動オフロードを、提案した。
  - コード分析→ループ文発見→高算術強度ループ抽出→OpenCL化→高リソース効率ループ絞り込み→絞り込みループで複数検証パターン→実測し高性能パターン選択
- 検討手法を実装し、FPGA自動オフロードが、市中アプリケーションで実現性があるかを検証した。
- 今後は、より多くのアプリケーションでの検証及び、GPU、FPGAと共通的に利用できる方式を検討する。