ハードとソフトの融合による環境適応実現のための性能優位性モデル

野田 竜平[1,a)]　趙 謙[2]　吉田 隆一[2]

概要：分散システムは実行環境の変化の影響を受けやすく, システムが自律的に環境適応を行うことが求められる. そこで, 我々は先行研究において, ソフトウェアモジュールの入れ替えによって, 分散システムで発生する環境変化への適応を実現してきた. しかし, CPU の性能向上の鈍化に伴い, ソフトウェアのみでは更なる性能向上は望めなくなってきた. そのため我々は, システムの性能向上を目的として, 回路の動的再構成による環境適応が可能となる FPGA(Field Programmable Gate Array) をアクセラレータとして導入することとした. これにより, ハードウェア／ソフトウェアの入れ換えによる環境適応が可能な高性能なシステムの構築を目指す.
これを実現する上での技術的課題として, ソフトウェア／ FPGA の役割分担がある. FPGA を利用することによって, 必ずしもシステムの性能が向上するとは限らないため, どの機能をソフトウェア／ FPGA で実現するのか判断する必要がある. そこで, 本研究ではソフトウェア／ FPGA のどちらで処理を実現した方が性能面において優位であるのかを評価するモデルとして, 性能優位性モデルを提案する.

キーワード：FPGA，分散システム・構成論， ハードウェア・ソフトウェア・コデザイン

Performance Evaluation Model for Realizing Environmental Adaptation by Integrating Hardware and Software

Tappei Noda[1,a)]　Qian Zhao[2]　Takaichi Yoshida [2]

Abstract: Distributed systems are susceptible to changes in the execution environment. Therefore, the systems are required to adapt to the environment autonomously. In our previous work, we have realized adaptation to environmental changes that occur in distributed systems by replacing software modules.
However, due to the slowdown in CPU performance improvement, no further performance improvement can be expected with software alone. Therefore, in order to improve the performance of the system, we decided to introduce an FPGA as an accelerator that can be adapted to the environment by dynamically reconfiguring the circuit. Our goal is to build a high-performance system that can be adapted to the environment by exchanging hardware and software.
One of the technical issues in achieving hardware/software integration is the division of roles between software and FPGA. Since the use of FPGA does not necessarily improve the performance of the system, it is necessary to determine which function is realized by software or FPGA. Therefore, in this study, we propose a performance evaluation model for the purpose of evaluating whether processing with software or with FPGA is superior in terms of performance.

Keywords: FPGA，Distributed System， Hardware/Software Co-design

1. はじめに

過去 50 年間, CPU はムーアの法則に基づき, 性能が向上

[1] 九州工業大学 大学院 情報工学府
Kyushu Institute of Technology
[2] 九州工業大学 大学院 情報工学研究院
Kyushu Institute of Technology
[a)] noda@ylab.ai.kyutech.ac.jp

してきた. その結果, ソフトウェアが社会に浸透し, 情報システムが社会に対して重要な役割を担うようになった. さらに, 社会における情報システムの重要度が増す事により, 必然的にシステムの更なる性能向上が求められる. しかし, CPU の性能は飽和しており, ムーアの法則が成り立たなくなった. そのため, ソフトウェアのみので構成されるシステムでは社会から求められる役割を担うことが出来なくなってきている. そこで, システムの性能向上を実現するために, 様々なアクセラレータを用いたヘテロジニアスな計算環境 [1] が注目されている.

我々が先行研究において提案した適応型分散システム [2] は, 分散システムにおける環境変化に対して, ソフトウェアモジュールの入れ替えによって対応している. 例えば, 分散システムではネットワークの実効帯域幅の増減という環境変化が発生する. その際, 実効帯域幅の増減に応じて通信データを圧縮するソフトウェアモジュールをロードすることによって, システム全体として性能向上を実現することができる。

しかし, 従来の適応型分散システムはソフトウェアのみで実装されていることから, 性能面に課題があった. そこで, 我々は FPGA をオフロード先のハードウェアとして用いることにより, システムの性能向上を実現する. 即ち, 先行研究においてソフトウェアの入れ替えのみで実現していた環境適応を, FPGA の動的再構成を含めた環境適応に拡張する.

ソフトウェアと FPGA の融合による環境適応を実現する上で解決しなければならない課題として, ソフトウェアと FPGA の役割分担が挙げられる. ソフトウェアが仮想記憶によってあたかも無制限に記憶容量を扱うことが出来るのに対して, FPGA は資源的に制限がある. そのため, 一部の機能はソフトウェアで実現する必要がある.

また, FPGA をアクセラレータとして用いることによって必ずしもシステムの性能が向上するとは限らない. システムから FPGA を利用する場合, システムからアクセラレータに対してデータを送信する際の, DMA 転送によるハードウェア的なオーバヘッドや, システムから FPGA を直接コントロールするドライバを呼び出す際の, ソフトウェア的なオーバヘッドが発生する。これらのオーバヘッドが原因となり, 図 1 が示すように, 処理量が小さい時はソフトウェアの方が性能面で有利であると考えられる。

つまり, ソフトウェアと FPGA の融合による環境適応を実現するためには, FPGA を用いた方がソフトウェアのみに比べて性能が有利になる点 (損益分岐点) を予測するモデルの開発が必須であり本研究の課題とする. このモデルを性能優位性モデルと名付ける.

本論文の構成は以下のとおりである. 2 章ではハードウェアアクセラレーションの性能予測に関する関連研究について述べる. 3 章では, 2 章で言及した LogCA モデルの詳細と, そのモデルをベースとした本研究の提案である性能優位性モデルについて述べる. 4 章では性能優位性モデルの有用性の評価について述べる. 最後に 5 章でまとめについて述べる.

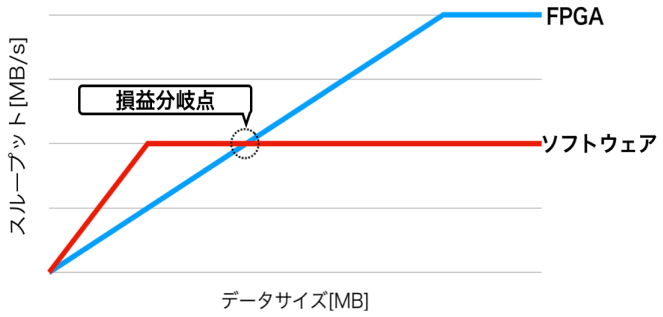

図 1 損益分岐点

2. 関連研究

ヘテロジニアスな計算環境では様々なプロセッサが協調して動作しているため, システム全体の性能を予測するのが非常に困難である. また, 1 章でも記述したようにアクセラレータを利用することによって必ずしもシステムの性能が向上するとは限らない. これらの理由から, アクセラレータの性能を予測する分析モデルは価値があり, 様々な提案がされている.

提案された方法としてルーフラインモデル [3] がある. このモデルはマルチコアアーキテクチャのための視覚的な性能モデルで, 性能はメモリの帯域幅もしくは ALU のスループットに制限されるという仮定に基づき大まかに性能を予測する. マルチコア以外のアーキテクチャに対しても柔軟に対応できることが証明されており, FPGA に対する拡張モデル [4][5] も提案されている. ルーフラインモデルではアクセラレータを抽象化しているため比較的単純に性能を予測することができる. しかし, ルーフラインモデルはあくまでもアクセラレータの最大性能の予測に特化したモデルであり, 我々が目的とする損益分岐点の予測には適していない.

ルーフラインモデルに比較的近いアプローチの分析モデルとして LogCA モデル [6] がある. このモデルもルーフラインモデルと同様に視覚的な性能モデルで, 比較的単純にアクセラレータの性能を予測することが出来る. また, ルーフラインモデルでは考慮されていなかった Host とアクセラレータとのインターフェースにおける設計上のボトルネックを明らかにすることが出来, 尚且つ我々の目的である損益分岐点の予測も実現することができる. 一方で, このモデルはマルチコアプロセッサや GPU の性能予測に特化しており, FPGA の性能をモデリングするにはパラメータが不十分である. そこで, 本研究では LogCA モデルの FPGA への拡張を行う.

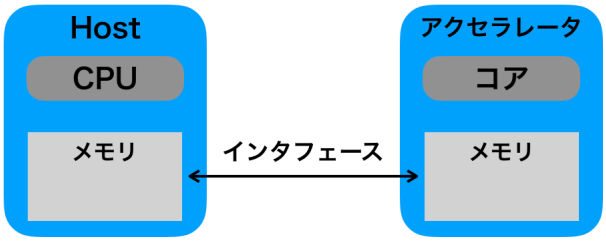

図 2　LogCA モデルが想定するシステム

3. 性能優位性モデル

3.1 LogCA モデル

一般的なハードウェアアクセラレーションはホストプロセッサ, デバイス, 通信リンクの 3 つの主要コンポーネントで構成されおり, LogCA モデルも上記の三つの主要コンポーネントからなる抽象的なシステムを想定している (図 2). 図 2 で示す Host はホストプロセッサ, アクセラレータは特定の機能に特化したデバイス, インタフェースは PCIe やシステムバスのような Host とアクセラレータを繋ぐ様々な通信リンクを抽象化している. このモデルではアルゴリズムの処理時間を, オフロードするデータ量を変数とした関数として定義している. ここで, オフロードするデータ量 g に対して, Host がアクセラレータを利用しなかった場合の処理時間は $T_0(g)$ と表し, Host がアクセラレータを利用した場合の処理時間は $T_1(g)$ と表す.

Host での処理時間を $C_0(g)$ と表すと, $T_0(g)$ は式 (1) と表現できる.

$$T_0(g) = C_0(g) \tag{1}$$

$O(g)$ をアクセラレータを利用する際に Host で発生するセッティング時間, $L(g)$ をインタフェースでのオーバヘッド, $C_1(g)$ をアクセラレータでの処理時間とすると, $T_1(g)$ は式 (2) と表現できる.

$$T_1(g) = O(g) + L(g) + C_1(g) \tag{2}$$

その為, アクセラレータを利用しなかった場合とアクセラレータを利用した場合の性能比 $speedup(g)$ は式 (3) のように表現できる.

$$speedup(g) = \frac{T_0(g)}{T_1(g)} = \frac{C_0(g)}{O(g) + L(g) + C_1(g)} \tag{3}$$

更に, CPU の最大性能とアクセラレータの最大性能の比を最大性能向上率として変数 A で表現した場合, $C_1(g) = \frac{C_0(g)}{A}$ と表現出来る.

我々の適応型分散システムの構成を抽象化すると, ソフトウェアモジュールを Host, FPGA をアクセラレータ, PCIe をインタフェースと捉えることが出来るため, LogCA モデルの主要な原則は採用することが出来る. 例えば環境変化が発生し, ソフトウェアモジュール／ FPGA 回路のいづれをロードするかを選択する際, 式 (3) に処理前のデータ量を入力し, $speedup(g) > 1$ の場合は FPGA, それ以外の場合はソフトウェア, というポリシーを設定することで課題であるソフトウェアとハードウェアの役割分担を解決することが出来る. しかし, LogCA モデルは出来るだけ少ないパラメータを使用するシンプルなモデルの提案を目的としていることや, 上述のようにマルチコアや GPU などのアーキテクチャを想定していることから, そのまま FPGA の性能予測に利用するにはパラメータが不十分であると考えた. そこで, LogCA モデルの主要な原則を採用しつつ, FPGA の性能予測に必要な新たなパラメータを追加した拡張モデルを構築する.

3.2 LogCA モデルの拡張

この節では, 提案するモデルに含まれる新たなパラメータについて説明する.

3.2.1 拡張式

式 (3) を我々が開発を進めている FPGA を導入した適応型分散システムへ拡張するため, インタフェースによるオーバヘッド $L(g)$ を FPGA との通信を考慮したオーバヘッド $L_{fpga}(g)$ に, アクセラレータでの最大性能向上率 A を, FPGA の特性を考慮した A_{fpga} に置き換える. その結果, 式 (3) を拡張した式 (4) が得られる.

$$speedup(g) = \frac{C_0(g)}{O(g) + L_{fpga}(g) + \frac{C_0(g)}{A_{fpga}}} \tag{4}$$

$L_{fpga}(g)$ と A_{fpga} についての説明を 3.2.2 項, 3.2.3 項で行う.

3.2.2 通信モデル

ここでは, FPGA との通信を考慮したインタフェースよるオーバヘッドである $L_{fpga}(g)$ を求める. 表 1 は, 通信に関して新たに導入するパラメータとそのパラメータに対応するシンボル, パラメータの導出方法を示している.

LogCA モデルの変数 $L(g)$ はデータの送受信にかかるインタフェースでのサイクル数からレイテンシを算出している. その為, 大凡の時間は予測可能だがより正確にレイテンシを求めるためには FPGA ベースのアクセラレータでの Host-デバイス間の通信をモデル化する必要がある.

Host から FPGA の呼び出しは, Host から FPGA に対するデータ書き込み, FPGA カーネルの実行, Host から FPGA に対するデータの読み込みの 3 つのステップから構成される. 一連のステップをまとめて FPGA トランザクションと名付け, その呼び出し回数を表すシンボルを FT とする. FPGA トランザクションに含まれるデータの書き込みとデータの読み込みの時間を表すシンボルをそれぞれ WT, RT とする.

Host から FPGA に対してデータを書き込む際, 予め FPGA 側のメモリに決められたサイズのバッファを割り当

表 1　通信に関する新たなパラメータ

パラメータ	シンボル	導出
Host から FPGA に対する呼び出し回数	FT	$\frac{g}{FB}$
Host から FPGA に対するデータ転送時間	WT	$\frac{FB}{PBW}$
FPGA から Host に対するデータ転送時間	RT	$\frac{FB}{PBW}$
Host プログラムに依存するインターバル時間	HI	N/A
入力データを格納する FPGA 側のバッファ容量	FB	N/A
PCIe の帯域幅	PBW	N/A

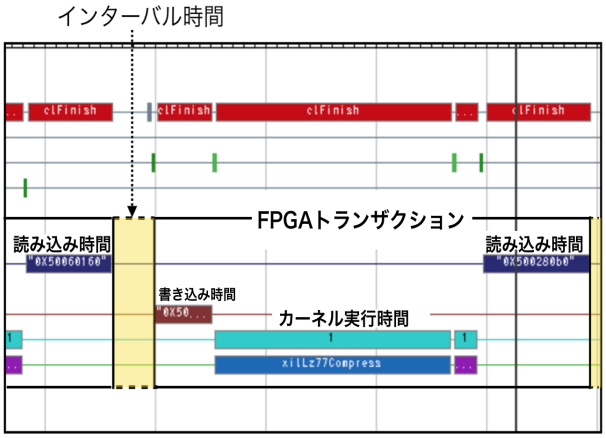

図 3　インターバル時間

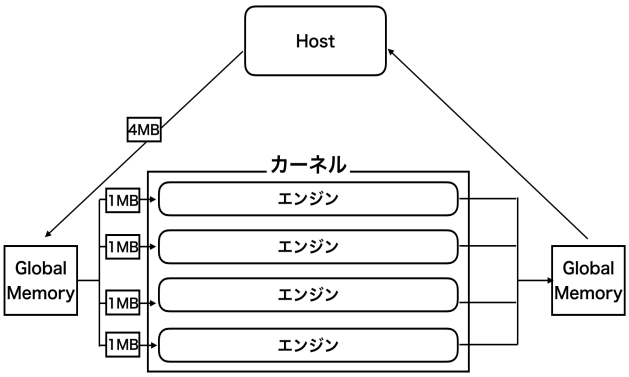

図 4　カーネルモデル

てる必要がある. バッファを割り当てる方法はオフロードするデータサイズに応じて動的に割り当てる方法と, カーネルが FPGA にロードされるのと同時に割り当てる方法の 2 通りがあるが, バッファを割り当てる処理は時間的なコストがかかる. その為, 前者の方法の場合, バッファ割り当ての時間がオーバヘッドとなりアクセラレーションによる性能向上が望めないことから, 後者の方法を選択した. このバッファを FPGA バッファと名付け, その容量を表すシンボルを FB とする. 例えば, このバッファ FB を 8MB, オフロードするデータサイズ g を 16MB とした場合, $FT = \frac{g}{FB}$ で求めることができるため, FT は 2 となる.

Host と FPGA をつなぐ PCIe の帯域幅を表すシンボルを PBW とする. すると, WT と RT はどちらも FB の値を PBW で除算することによって求めることが出来る.

更に, 各 FPGA トランザクションの間にはインターバル時間が発生する. この時間を表すシンボルを HI とする. 図 3 は FPGA を用いたあるアプリケーションの実行を Vitis アナライザー [7] という解析ツールを用いて実際に解析した際に生成されるレポートである. これにより, インターバル時間の発生が確認できる.

以上の結果から, 一回の FPGA の呼び出しにかかる時間的コストは WT, RT, HI の和で求めることが出来る. よって, $L_{fpga}(g)$ は式 (5) のように求めることが出来る.

$$L_{fpga}(g) = (WT + RT + HI) \times FT \qquad (5)$$

3.2.3　並列性

ここでは, FPGA の並列性を考慮したアクセラレータによる最大性能向上率である A_{fpga} を求める. 表 2 は, 並列性に関して新たに導入するパラメータとそのパラメータに対応するシンボル, パラメータの導出方法を示している.

LogCA モデルがターゲットとしているアクセラレータは固定的なハードウェアであることから, A が定数となる. そのため, アクセラレータでの実行時間を $\frac{C_0(g)}{A}$ と表現することができる.

これに対して FPGA はプログラマブルな性質上, 同じ機能であってもリソースの利用状況によってカーネルの並列性が変化する為, それに伴って実行時間も変化する. 即ち, 並列性という要素をパラメータとして考慮する必要がある. 図 4 はある機能が実装されたカーネルのモデルである. このようにアルゴリズムの性質によって入力データを並列に処理できるものがあるので, この理由から, 並列性のあるアルゴリズムを実装したカーネルには図 4 のように並列実行可能なエンジンを複数搭載しているものがある. そこで, カーネルに搭載されているエンジンの数を表すシンボルを NE とする. また, そのエンジン一つあたりのスループットを表すシンボルを TE とする. すると, カーネル全体の最大スループットは式 (6) で求めることが出来る.

$$\text{カーネル全体のスループット} = TE \times NE \qquad (6)$$

このカーネルでの処理プロセスはまず, Host からの入力データを FPGA 側の Global Memory に送信する. Global Memory から非処理ブロックをラウンドロビン方式で個々

表 2　並列性に関する新たなパラメータ

パラメータ	シンボル	導出
cpu の最大スループット	MTC	N/A
エンジン 1 つあたりの最大スループット	TE	N/A
カーネル 1 つあたりのエンジン数	NE	N/A
並列実行可能なエンジン数	NAE	N/A
FPGA に乗っているカーネル数	NK	N/A
並列性を利用する為に必要なデータサイズ	TB	N/A

のエンジンに送る. 処理済みのブロックをエンジンから読み出し, グローバルメモリに書き込む. このプロセスの中で, Global Memory から転送されるブロックのサイズはプログラマーが設定可能であるが, このブロックサイズを超えるデータ量を Host から入力しなければエンジンの並列性を活かすことが出来ない. 例えばブロックサイズを 1MB に設定した場合 1MB の以上のデータが入力されるまではエンジン一つ分の性能しか出すことができない. そこで, エンジンの並列性を利用することが出来るデータ量の閾値を表すシンボルを TB とする. また, あるデータ量が与えられた時, 利用可能なエンジン数を NAE とすると, NAE は式 (7) のように求めることができる.

$$NAE = \begin{cases} \frac{g}{TB} & (\frac{g}{TB} \leq NE) \\ NE & (\frac{g}{TB} > NE) \end{cases} \tag{7}$$

以上の結果から, ある処理を FPGA で行う場合, 並列性に関する最小のモジュールはエンジンである為, FPGA の性能は, エンジンのスループット TE, 並列実行可能なエンジンの数 NAE, 現 FPGA リソースに搭載可能なカーネル数を表すシンボル NK の積で求めることが出来る. よって, A_{fpga} は Host の CPU の最大スループットを表すシンボル MTC と, 式 (6)(7) より式 (8) のように求めることが出来る.

$$A_{fpga} = \frac{TE \times NAE \times KN}{MTC} \tag{8}$$

3.3　パラメータの取得方法

本モデルをシステムに適用する場合, 新たに導入したパラメータの取得方法について考える必要がある. そこで, 新たに導入したパラメータをアプリケーション固有のパラメータ, システム固有のパラメータ, モジュール固有のパラメータに分類し, 取得方法をまとめた.

- アプリケーション固有のパラメータ
  - 入力データサイズの将来による予測
    - g
- システム固有のパラメータ
  システム管理者があらかじめシステムに対して記述する.
  - CPU の性能
    - MTC
  - Host でのセッティング時間
    - $O(g)$
  - PCIe の帯域幅
    - WT
    - RT
- モジュール固有のパラメータ
  ソフトウェアモジュール作成者及び, FPGA 回路の作成があらかじめシステムに対して記述する.
  - アルゴリズムを実装したソフトウェアの性質
    - $C_0(g)$
  - アルゴリズムを実装した FPGA の性質
    - HI
    - NK
    - NE
    - TE
    - FB
  - 通信回数
    - FT

表 3　評価環境

項目	説明
Java VM	openjdk version “11.0.11”
OS	Ubuntu 18.04.1 LTS
FPGA	Xilinx 社製 Alveo U200
Memory	94GB
CPU	AMD Ryzen Threadripper 3960X 24-Core Processor

4.　評価

この章で本研究で提案したモデルの有用性の評価について述べる. 評価環境は表 3 に示す. 提案モデルを評価するにあたり, 式 (4) より得られた理論値と実測値から得られる最大性能の比と, 損益分岐点について, 相対誤差を求めた. 今回処理対象として処理時間が異なる二つの圧縮アルゴリズム, Gzip と Lz4 を用いた. この二つのアルゴリズムに対してソフトウェア実装は Java のライブラリ [8][9] を利用した.

今回我々 FPGA アクセラレータとして Xilinx 社製の Alveo U200 を用いており, この FPGA をターゲットとしたアプリケーションの開発環境として Vitis[10] を利用した. Xilinx はデータ圧縮やセキュリティとなどの Vitis 上で動作するアクセラレータのためのライブラリ (Vitis ライ

表 4　拡張モデルパラメータの計算値:Gzip

パラメータ	シンボル名	パラメータ
cpu の最大スループット	MTC	14[MB
エンジン 1 つあたりの最大スループット	TE	60[MB
カーネル 1 つあたりのエンジン数	NE	
FPGA に乗っているカーネル数	NK	
Host プログラムに依存するインターバル時間	HI	$9.4 \times 10^{-}$
Host から FPGA に対するデータ転送時間	WT	$6.2 \times 10^{-}$
FPGA から Host に対するデータ転送時間	RT	$3.2 \times 10^{-}$

表 5　拡張モデルパラメータの計算値:Lz4

パラメータ	シンボル	パラメータ値
cpu の最大スループット	MTC	180[MB/s]
エンジン 1 つあたりの最大スループット	TE	267[MB/s]
カーネル 1 つあたりのエンジン数	NE	8
FPGA に乗っているカーネル数	NK	1
Host プログラムに依存するインターバル時間	HI	9.4×10^{-4}[s]
Host から FPGA に対するデータ転送時間	WT	6.2×10^{-4}[s]
FPGA から Host に対するデータ転送時間	RT	3.2×10^{-4}[s]

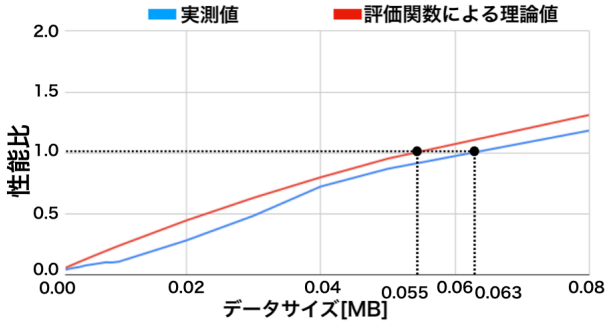

図 5　損益分岐点:Gzip

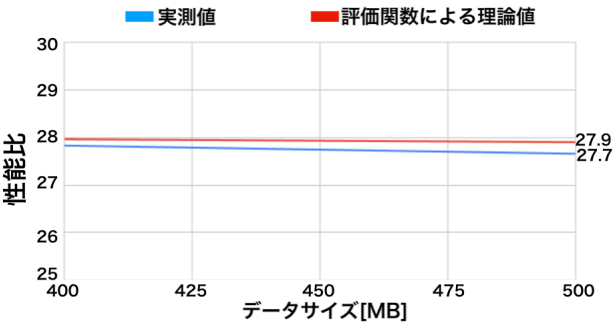

図 6　最大性能比:Gzip

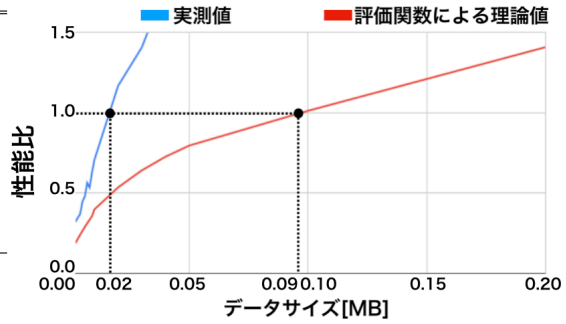

図 7　損益分岐点:Lz4

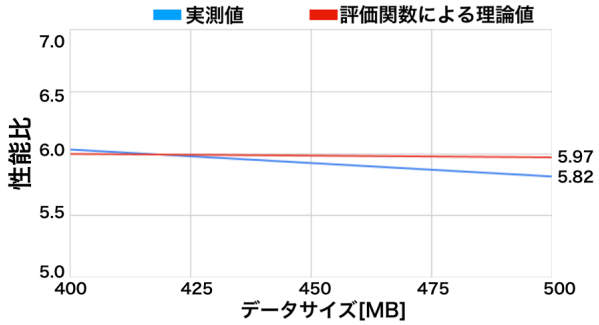

図 8　最大性能比:Lz4

ブラリ)[11] を提供しており, ハードウェア実装はこの Vitis ライブラリを利用した.

データサイズを最小値 0MB から最大値 500MB まで 100KB 間隔で変化させながらソフトウェア実装とハードウェア実装それぞれの処理時間を測定した. 圧縮対象としてバイナリがランダムに記述されたファイルを用いた. 各データサイズごとに 100 回データを測定し, 平均を求めた. ソフトウェアの処理時間は Java での実行時間のみを測定している. 本実験では, JVM から FPGA を直接コントロールするドライバを利用するためのインタフェースとして JNI(Java Native Interface) を利用した。その為、ハードウェアの処理時間は Host から JNI とドライバを介して FPGA にメッセージを送信し, 処理結果がドライバと JNI を介して Host に返ってくるまでの時間を計測している. 図中の $C_0(g)$, $O(g)$, $L_{fpga}(g)$, $\dfrac{C_0(g)}{A_{fpga}(g)}$, $speedup(g)$ はそれぞれ式 (4) のパラメータに対応している. また式 (4) のパラメータに入力する具体的な計算値は表 4, 表 5 に示す.

4.1　Gzip

圧縮アルゴリズムとして Gzip を用いた場合の各入力データサイズにおける式 (4) により得られた理論値と, 実際に動作させたハードウェア実装とソフトウェア実装の処理時間の比を性能比として, 図 5, 6 のグラフで示す. 図 5 は横軸のデータサイズを最小値 0[MB] から最大値 0.08[MB] まで示し, 図 6 は横軸のデータサイズを最小値 400[MB] から最大値 500[MB] まで示す.

図 5 より, 理論値の損益分岐点は 0.055[MB], 実測値の損益分岐点は 0.063[MB] となる. また, 図 6 より理論値の最大性能比は 27.9, 実測値の最大性能比は 27.7 となる.

4.2　Lz4

圧縮アルゴリズムとして Lz4 を用いた場合の各入力データサイズにおける式 (4) により得られた理論値と, 実際に動作させたハードウェア実装とソフトウェア実装の処理時

	損益分岐点	最大性能比
理論値	0.055[MB]	27.9
実測値	0.063[MB]	27.7
相対誤差	15[%]	0.717[%]

表 6 実験結果:Gzip

	損益分岐点	最大性能比
理論値	0.097[MB]	5.97
実測値	0.017[MB]	5.82
相対誤差	82[%]	2.51[%]

表 7 実験結果:Lz4

間の比を性能比として, 図 7, 8 のグラフで示す. 図 7 は横軸のデータサイズを最小値 0[MB] から最大値 0.20[MB] まで示し, 図 8 は横軸のデータサイズを最小値 400[MB] から最大値 500[MB] まで示す.

図 7 より, 理論値の損益分岐点は 0.097[MB], 実測値の損益分岐点は 0.017[MB] となる. また, 図 8 より理論値の最大性能比は 5.97, 実測値の最大性能比は 5.82 となる.

4.3 考察

表 6, 表 7 はそれぞれ Gzip, Lz4 の損益分岐点と最大性能比の相対誤差を示している. この, 結果から Gzip, Lz4 共に最大性能比の相対誤差はそれぞれ 0.717%, 2.15%と, 高い予測精度を示している. 一方で, Lz4 の損益分岐点が 82%と大きな誤差が生じていることがわかる. 原因として, 今回の実験で用いた式 (4) の Host での実行時間を表すパラメータ $C_0(g)$ は, 最小二乗法により直線で近似し, 処理時間を予測している. 処理するデータサイズが小さい時, 直線近似によって得られた予測時間と実測値の間で誤差が大きく, その結果, 損益分岐点の相対誤差が大きくなったと考えられる.

但し, 損益分岐点は実測値, 理論値のいずれも 100KB 以下に収まっている. Lz4(ソフトウェア) で, データサイズ 100KB の本実験で利用した圧縮用ファイルを圧縮した際の処理時間は約 9[ms], 同じファイルを Lz4(FPGA) で圧縮した際の処理時間は約 3[ms] であった. このことから, 実際のアプリケーション運用時を考えると, 100KB 付近のデータを圧縮する場合, ソフトウェアで処理する場合と FPGA で処理する場合とで, 絶対値としての処理時間に大きな差がないため, 損益分岐点の誤差は許容範囲だと考える。

5. 結果

本研究の評価により, 必要なパラメータを正しく取得する事が出来れば, 式 (4) を利用することで FPGA の性能を正確に予測可能であることがわかった. 本モデルに必要なパラメータを取得し, 式 (4) より得られる値に応じてソフトウェアで処理をするのか, もしくは FPGA で処理をするのかを決定することにより, 本研究の課題であったソフトウェアと FPGA の役割分担を解決することができると考える.

6. おわりに

6.1 まとめ

我々はシステムにおける環境変化に対して広く適応可能な適応型分散システムを提案している. しかし, 従来のシステムは全てソフトウェアで実装していた為, 性能面での課題があった. そこで, システム全体の性能向上を目的として FPGA によるハードウェアアクセレーションを行いハードウェアとソフトウェアの融合による環境適応の実現を提案した.

ハードウェアとソフトウェアの融合による環境適応の実現を達成する上での技術的課題としてソフトウェア／FPGA の役割分担が挙げられる. そこで本研究ではどのような状況においてソフトウェア／ FPGA を利用するのが性能面において有効であるのかを決定する性能優位性モデルを提案した. これは, システムの構成パラメータや FPAG の回路設計者から容易に取得可能なパラメータから, FPGA を用いたほうがソフトウェアのみの場合に比べて性能が有利になる点を予測するモデルである.

提案したモデルで求めた理論値と実測値を二つの圧縮アルゴリズム Gzip, Lz4 を用いて比較し予測制度を評価した. これにより, 式 (4) より得られる値に応じてソフトウェアで処理をするのか、もしくは FPGA で処理をするのかを決定することにより、本研究の課題であったソフトウェアと FPGA の役割分担を解決することができると考える.

6.2 今後の課題

今後の課題として, 性能優位性モデルに対する課題について述べる.

- 非線形アルゴリズムへの適応
- 再構成時間も含めた FPGA への切り替えの判断
- 性能以外のパラメータの検討

まず, 非線形アルゴリズムへの適応である. 今回用いた二つの圧縮アルゴリズムはどちらも線形アルゴリズムである. その為, 非線形なアルゴリズムに対しても同じように高精度に予測可能であるのかを検証する必要がある.

次に再構成を含めた FPGA への切り替え判断である. 本研究で開発したモデルは, あらかじめ目的の回路が FPGA にロードされた状態で性能の優位性を判断するものである. FPGA 回路の再構成が発生した場合, 回路の再構成には大きは時間的コストが発生する為, その時間を加味した性能予測を行う必要がある. 本研究で提案したモデルを利用することによって, どれくらいの時間, FPGA を利用すれば再構成時間の採算ラインを超えるのか予測ができる為, 今後このモデルを利用して再構成時間を含めた FPGA の切り替え判断を行う必要がある.

最後に性能以外のパラメータの検討である．本研究で開発した予測モデルは性能に対してのみ有用なモデルである．しかし, アクセラレータを用いる上で消費電力のようなエネルギー効率に関する指標は重要なパラメータである．その為, 今後そのようなパラメータを補完する必要がある．

参考文献

[1] Yan, L., Cao, S., Gong, Y., Han, H., Wei, J., Zhao, Y. and Yang, S.: SatEC: A 5G Satellite Edge Computing Framework Based on Microservice Architecture, *Sensors*, Vol. 19, No. 4 (online), DOI: 10.3390/s19040831 (2019).

[2] K. Oda, S. Izumi, Y. Y. and Yoshida, T.: A Simple Reconfigurable Object Model for a Ubiquitous Computing Environment, *International Journal of Computer Science and Network Security*, No. 5 (2007).

[3] Williams, S., Waterman, A. and Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures, *Commun. ACM*, Vol. 52, No. 4, p. 65–76 (online), DOI: 10.1145/1498765.1498785 (2009).

[4] Yali, M.: FPGA-Roofline: An Insightful Model for FGPA-based Hardware Acceleration in Modern Embedded Systems (2015).

[5] Da Silva Gomez, B., Braeken, A., Erik, H. and Touhafi, A.: *Performance and Resource Modeling for FPGAs using High-Level Synthesis tools*, Advances in Parallel Computing, pp. 523–531, IOS Press (2014).

[6] Altaf, M. S. B. and Wood, D. A.: LogCA: A high-level performance model for hardware accelerators, *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 375–388 (online), DOI: 10.1145/3079856.3080216 (2017).

[7] Xilinx.2020: Vitis analyzer, `https://apan.xilinx.com/html_docs/xilinx2020_2/vitis_doc/jfn1567005096685.html` (Accessed: 2022-01-11).

[8] JavaSE: Gzip ライブラリ ，`https://docs.oracle.com/javase/jp/8/docs/api/java/util/zip/GZIPOutputStream.html` (Accessed: 2022-01-17).

[9] Collet, Y.: Lz4 ライブラリ，`https://github.com/lz4/lz4-java` (Accessed: 2022-01-17).

[10] Xilinx.2020: Vitis, `https://japan.xilinx.com/products/design-tools/vitis.html` (Accessed: 2022-01-17).

[11] Xilinx.2020: Vitis library, `https://github.com/Xilinx/Vitis_Libraries` (Accessed: 2022-01-17).