産業応用工学会論文誌, Vol. 8, No. 2, pp. 260-262 (Sept. 2020)
The Japanese Journal of the Institute of Industrial Applications Engineers
Online edition: ISSN 2187-5146, Print edition: ISSN 2189-373X, DOI: 10.12792/jjiiae.8.2.260

レター

環境適応ソフトウェア実現に向けた全体フロー提案

山登　庸次 [a,*]

Processing Flow of Software Adaptation Based on Deployed Environment

Yoji Yamato[a,*]

(Received June 6, 2020; revised July 7, 2020; accepted July 20, 2020)

Abstract

Recently, heterogeneous hardware such as GPU and FPGA utilization is increased in systems. However, to utilize them, the hurdles are high because of much skill. Therefore, I propose environment adaptive software to operate an application with high performance by automatically converting the code and configuring so that we can utilize heterogeneous hardware easily.

キーワード : 環境適応ソフトウェア，GPGPU，FPGA，自動オフロード，性能
Keywords : Environment Adaptive Software, GPGPU, FPGA, Automatic Offloading, Performance.

1. はじめに

近年，ムーアの法則が鈍化すると言われている。その状況も踏まえ，GPU や FPGA といったヘテロなハードウェアをシステムに用いることが増えており(1)，それを用いたサービスも IoT やサービス連携技術(2)-(4)等により数多く開発されている。しかし，ヘテロなハードウェアを様々なアプリケーションで活用して，高性能に運用するためには，ハードウェアに合わせたプログラムや設定が必要で，CUDA，OpenCL 等(5)高度技術が求められる。

本稿は，アプリケーションを高性能に運用するため，コード変換や設定を配置環境に合わせて自動で行う，環境適応ソフトウェアを提案する。これにより，GPU，FPGA 等のハードウェアを容易に活用することを目指す。本稿貢献は，新規な環境適応ソフトウェア全体フロー提案であり，個々の要素技術の詳細提案，評価は別稿に譲る。

2. 環境適応ソフトウェアの処理フロー

ソフトウェアの環境適応には，コード変換，リソース量/配置場所の設計，運用時環境適応が必要と考えている。これらは，現在，SIer が手動で行っている領域である。例えばコード変換では，根本的課題として，どの部分をオフロードすれば高速化できるかの判断は，スキル，経験が必要で，CUDA や OpenCL を熟知した技術者が手動設計しているのが基本である。本稿では，これらのステップを機械により自動で行う事が最大の差異点である。これらの処理を実現する処理フローを，図 1 で説明する。

Step1 コード分析:ユーザは動作させたいアプリケーションコードと利用を想定したテストケース，要望する性能とコストを，環境適応機能に指定する。環境適応機能は，コードを分析する。分析では，ループ文や変数の参照関係，処理する機能ブロック等の，構造を把握する。

Step2 オフロード可能部抽出 : 環境適応機能は，アプリケーションコードの並列処理可能なループ文や FFT 処理等の機能ブロック等，オフロード可能な処理をコードパターン DB を参照して特定し，オフロード先に応じた中間言語（OpenCL 等）を抽出する。なお，中間言語抽出は一度で終わりでなく，適切なオフロード領域探索のため，検証環境で試行して最適化するため反復がされる。

Step3 適切なオフロード部探索:次に，環境適応機能は，GPU や FPGA，IoT デバイス用 GW 等を備えた検証環境に，中間言語から導かれる実行ファイルを配置する。配置ファイルを起動し，テストケースを実行して，オフロードした際の性能を測定する。ここで，GPU や FPGA 等オフロード先に応じて，性能結果を参照して，より適切なオフロードとするため，Step2 の中間言語抽出に戻り，別パターン抽出を行い，性能測定を試行する。検証環境での性能測定を繰り返し，最終のコードを決定する。

Step4 リソース量調整 : コードパターンを決定後は，環境適応機能は，適切なリソース量の設定を行う。Step3

* Corresponding author. E-mail: yoji.yamato.wa@hco.ntt.co.jp
a 日本電信電話（株）　ネットワークサービスシステム研究所
〒180-8585　東京都武蔵野市緑町 3-9-11
Network Service Systems Laboratories, NTT Corporation
3-9-11, Midori-cho, Musashino-shi, Tokyo, Japan 180-8585

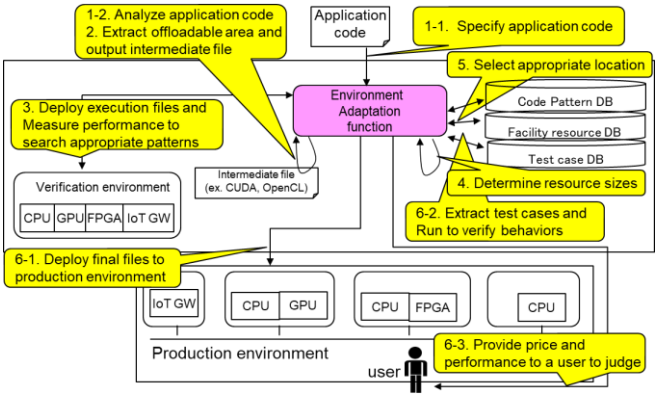

Fig. 1. Processing flow of environment adaptive software.

の検証環境性能測定で取得される，想定テストケースの処理時間の中で，CPU 処理時間と GPU 等の CPU 以外ハードウェアの処理時間を分析し，適切なリソース比を定める。次に，ユーザの要望する性能，コストと適切なリソース比を鑑みて，実際に確保するリソース量を定める。

Step5 配置場所調整：実行ファイルを配置する際に，環境適応機能は，性能，コストが適切になる場所を計算し配置先を決める。実行するアプリケーションの想定するテストケースの特性，設備リソース DB の情報から，性能とコストが適切になる配置場所を計算する。ここで，配置場所（エッジ等）に，リソース量制限から，必要なリソースを確保できない場合は，Step4 に処理を戻す。

Step6 実行ファイル配置と動作検証：商用環境に配置後，期待通りの動作となるかを，環境適応機能は，動作検証試験を行う。想定テストケースや，テストケース DB に保持されているリグレッションテストケースを用いて，動作検証する。この際に，テストケースの商用環境での実性能を，確保した全リソースのスペックやコストも含めて，ユーザに提示し，ユーザにサービス開始判断をもらい，OK の場合にアプリケーションの運用を開始する。

Step7 運用中再構成：開始した運用にて，リクエスト特性変化等で当初の性能が出ない場合に，環境適応機能は，ソフトウェア設定，ソフトウェア/ハードウェア構成を再構成する。ソフトウェア設定とは，リソース量や配置場所の再変更を意味し，ソフトウェア/ハードウェア構成とは，GPU ではオフロードする処理部分を，FPGA ではハードウェアロジックを再構成することを意味する。

本全体フローは，全ステップが実装できているわけではない。しかし，一部は実装しており，Step1-3 を GPU に適用した例として(6)がある。(6)では，映像認識を深層学習で行う Darknet の 3 倍自動高速化を行っており，動体展示も NTT R&D フォーラム 2018 秋に展示された。

3. 環境適応ソフトウェア処理フローの要素技術

3.1 コード変換

Step1 のコード分析については，Clang(7)-(9)等の構文解析ツールを用いて，分析を行う。分析は，ループ文や変数の参照関係等のコードの構造や，FFT 処理等の特定処理を行う機能ブロックである事や特定処理のライブラリを呼び出している等を把握する。機能ブロックの判断には，Deckard(10)等の類似性検知ツールを用いる。オフロード性能は，最大性能になる設定を一回で発見するのは難しいため，性能測定を検証環境で繰り返し試行し，高速化パターンを見つけることを，Step2，3 で行う。

(a) GPU 向けオフロード

私達は既に，GPU 自動オフロード方式を提案している(6)(11)。提案方式では，GPU 処理に適切なループ文を，検証環境での繰返し測定により抽出を行うが，その際に進化計算手法の一つである遺伝的アルゴリズムを用いて探索を行っている。さらに，CPU と GPU 間のメモリデータ転送を削減するため，ネストループ内の変数で一括化できる変数は一括転送を行っている。(11)では行列計算等の小アプリについて 10 倍以上，(6)では Darknet 等の大規模アプリについて 3 倍以上高速化を実現している。

(b) FPGA 向けオフロード

FPGA のオフロードは，多くはアプリケーションに依存するため，機械がオフロードロジックを抽出するのは難しい。そこで，FPGA では，今までにプログラマーが蓄積したノウハウ（Well-known パターン）を生かすため，機能ブロックでオフロードする。Step1 のコード分析で把握した機能ブロックがコードパターン DB に登録されている場合に，登録 FPGA ロジックに置換してオフロードする。この置換のため，コードパターン DB は，FPGA にオフロード可能な処理のライブラリ呼び出しや機能ブロックと，オフロードする FPGA 処理ロジックの OpenCL コードを，登録している。もし，オフロード機能ブロックが見つからない場合は，GPU と同様ループ文を試行するが，ROSE(12)等の Arithmetic Intensity ツールで測定した計算密度が高いループ文を優先的に試行する。

3.2 リソース量調整

Step4 のリソース量調整については，まず適切なリソース比を決め，次に性能，コスト要件に合うリソース量に設定する。例えば，CPU と GPU で動作させるコードに変換したとして，CPU と GPU のリソース量が適切なバランスでない場合は，性能が出ない。(13)では，CPU と GPU で MapReduce を行う際，CPU と GPU の実行時間が同じになる Map タスクを配分し高性能化している。私達も，リソース比を決める際は，(13)等を参考に，何れかのハードウェアでの処理がボトルネックとなる配置を避けるため，CPU とオフロード先の測定処理時間が同等で適切なオーダーになるよう，リソース比を決定する。リソース比決定後は，想定テストケースの処理性能が，Step1 でユーザ指定の要求性能及びコスト要求を満たす様に，リソース比はキープして，リソース量を決定する。

3.3　配置場所調整

Step5 の配置場所調整では，性能，コストが適切な場所を計算し配置先を決める。配置先を決めるため，配置アプリケーションの想定テストケースの性能情報（処理遅延やスループット）と，利用できる設備リソース情報（クラウド，エッジ，Home GW 等の計算リソース，帯域，既利用量，利用コスト）がある。想定テストケースの性能結果から，アプリケーションを配置した際の計算量と発生トラフィックをまず算出する。合わせて，クラウド，エッジ等のリンク関係をモデル化しておく。アプリケーションを特定ノードに配置した際，コストが要求条件に収まる事を制約条件に，性能を最大化する配置，又は，性能が要求条件を満たす形でコストが最低になる配置を，線形計画手法等の最適化計算を用いて計算する。

3.4　動作検証

Step6 の動作検証では，Step1-3 で決まった実行ファイルを Step4 の指定リソース量で Step5 の指定場所に配置した後に，期待通りの動作を，性能テストケースやリグレッションテストケースで確認する。性能テストケースは，ユーザが指定した想定テストケースを Jenkins(14)等の自動実行ツールを用いて行い，処理時間やスループット等を測定する。リグレッションテストは，システムにインストールされるミドルウェアや OS 等のソフトウェア情報を取得して対応リグレッションテストを実行する自動検証技術が検討されており(15)，それらを用いる。

3.5　運用中再構成

Step6 の運用開始後，リクエスト特性変化等で当初の性能が出ない場合に，環境適応機能は，ソフトウェア設定，ソフトウェア/ハードウェア構成を再構成する。

ソフトウェア設定変更は，Step4-5 処理を，周期的，又は，性能がある閾値以下となった場合に試行模擬し，性能向上やコスト低減度合を計算する。リソース量変更や配置場所変更で性能やコストが改善できる見込みがある場合は，ユーザに再構成を提案する。

ソフトウェア/ハードウェア構成変更は，Step1-3 処理を，周期的，又は，性能が閾値以下となった場合に試行模擬し，コード変換して，GPU オフロード部分のロジック変更や FPGA のハードロジックの変更(16)を行う。性能やコスト改善の見込みがある場合は，ユーザに再構成を提案する。ユーザ了承を得て，再構成を実施する際に，GPU オフロード部分の変更等，ソフトウェア構成変更の場合は，実行ファイルを起動する環境を複製後，クラウド技術(17)-(19)を用いてデータをマイグレーションする。

効果を考察する。Step3 は，ユーザにとって高性能化によりサーバ数を減らせ設備コスト減効果がある。Step4 も，効率的にサーバ数を設定でき，設備コスト減効果がある。Step5 は，クラウドだけでなくエッジ等の最適配置により応答時間減効果がある。Step7 は，オフロード部再構成による性能改善で，増設コスト減効果がある。

4.　おわりに

本稿では，GPU，FPGA 等を適切に活用し高性能を実現するため，アプリケーションを配置環境に適応させる，環境適応ソフトウェアの全体フローを提案した。

文　献

(1) AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/

(2) Y. Yamato, et al., "Predictive Maintenance Platform with Sound Stream Analysis in Edges," Journal of Information Processing, Vol.25, 2017.

(3) Tron project web site, http://www.tron.org/

(4) Y. Yamato, et al., "Proposal of Lambda Architecture Adoption for Real Time Predictive Maintenance," IEEE CANDAR 2016, 2016.

(5) K. Ishizaki, "Transparent GPU exploitation for Java," CANDAR 2016.

(6) Y. Yamato, "Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications," Journal of Intelligent Information Systems, Springer, 2019.

(7) Clang website, http://llvm.org/

(8) gcov website, http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

(9) gprof website, http://sourceware.org/binutils/docs-2.20/gprof/

(10) Deckard web site, http://github.com/skyhover/Deckard

(11) Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, Sep. 2018.

(12) ROSE website, https://github.com/rose-compiler/rose-develop

(13) K. Shirahata, et al., "Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters," IEEE CloudCom2010, 2010.

(14) Jenkins web site, https://jenkins.io/

(15) Y. Yamato, "Automatic verification technology of software patches for user virtual environments on IaaS cloud," Journal of Cloud Computing, Springer, 2015, 4:4, Feb. 2015.

(16) Y. Yamato, "Server Selection, Configuration and Reconfiguration Technology for IaaS Cloud with Multiple Server Types," Journal of Network and Systems Management, Springer, Aug. 2017.

(17) Y. Yamato, et al., "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Trans. on Cloud Computing, 2015.

(18) Y. Yamato, "Cloud Storage Application Area of HDD-SSD Hybrid Storage, Distributed Storage and HDD Storage," IEEJ Transactions on Electrical and Electronic Engineering, Vol.11, pp.674-675, 2016.

(19) Y. Yamato, "Use case study of HDD-SSD hybrid storage, distributed storage and HDD storage on OpenStack," ACM IDEAS 2015, 2015.

山登　庸次

2000 年東京大学理学部卒。2009 年同大大学院総合文化研究科にて博士（学術）。2002 年日本電信電話株式会社入社。同社にて，クラウド基盤技術，IoT 技術研究開発に従事。現在，同社 NTT ネットワークサービスシステム研究所特別研究員。