環境適応ソフトウェアにおける自動オフロード対象の計算タイプ拡大

山登　庸次 [a,*]

Expanding the automatic offloading computation types of environment-adaptive software

Yoji Yamato[a,*]

(Received , ; revised , ; accepted ,)

Abstract

We have proposed the concept of environment-adaptive software that automatically converts software code so that it can appropriately utilize the environment. In this paper, we proposed a method to analyze user-provided applications without looking at individual loop statements, and automatically offload them to various hardware using appropriate processing algorithms.

キーワード : 環境適応ソフトウェア, 自動オフロード, GPGPU, 計算タイプ, パターンマッチング
Keywords : Environment-Adaptive Software, Automatic Offloading, GPGPU, Computation Type, Pattern Matching.

1. はじめに

近年，GPU や FPGA，メニーコア CPU 等の多様ハードウェアがアプリケーションで利用されるようになっている．Amazon 社は，GPU，FPGA，マルチコア CPU VM（Virtual Machine）をクラウド(1)で提供している(2).

しかし，多様ハードウェアを利用するためには，GPU では CUDA (3)，FPGA では OpenCL (4)，メニーコア CPU では OpenMP(5)等言語が前提となることが多い．そのため，多くのプログラマーにとって難度が高い.

そこで，私達は，既存プログラムを，動作環境で高性能に利用できるよう，自動変換等し，アプリケーション処理を高速化する，環境適応ソフトウェアを提案してきた．更に，要素技術として，既存プログラムのループ文を，GPU，FPGA，メニーコア CPU 等に自動オフロードする方式等を提案し評価している(6)-(19).

これは，ある程度高速化は可能だが，計算タイプに合わせてアルゴリズムを考えた手動改造した高速化には及ばなかった．本稿は，フーリエ変換等の計算タイプに応じた, 多様なハードウェアへのオフロードを対象とする.

2. 既存技術と本稿の課題

GPU，FPGA，メニーコア CPU 等の多様なハードウェアを共通的に扱う仕様に OpenCL がある．OpenCL と異なり，容易に多様なハードウェアを使うため，指示句（Directive）を使う仕様がある. 例えば，メニーコア CPU 等で計算処理するための仕様として OpenMP がある．OpenMP 解釈実行ツールには gcc(20)等がある．なお，指示句で GPU 処理に OpenACC(21)が，OpenACC 解釈実行ツールに PGI コンパイラ(22)（現 nvc）がある.

しかし処理は行えても，高速化するには壁がある．例えば，ループ文を並列処理する Intel コンパイラ(23)がある．しかし，データが効率的に利用されない場合，並列処理しても高速化されない．そのため多様なハードウェア利用では，専門家がチューニングして，ツールを繰り返し実行して適切な改造がされている．著者は，ループ文のオフロード有無を遺伝的アルゴリズム(24)により，検証環境での繰り返し測定を通じて，高速なオフロードパターンを徐々に発見していく方式を提案している.

課題を整理する．多様なハードウェアを用いた高速化は専門家による手動改造が主流である．私は環境適応ソフトウェアを提案してきた．しかし，今までは個別 for 文のオフロードが自動化の主な対象であった. そのため，少コア CPU に比べ 10 倍程度高速化は可能だが，計算タイプに合わせハードウェアを意識してアルゴリズム含めて考えた手動改造での高速化には及ばなかった．FPGA とメニーコア CPU は別論文で詳細評価しているので, 本稿は，全体とその中で GPU を題材に，フーリエ変換等の計算タイプに応じた，自動オフロードを対象とする.

3. 計算タイプに応じた計算処理のオフロード

3.1 中粒度計算のアクセラレータオフロードアイデア

今まで，個々ループ文をオフロードするしないのパターンを作り，検証環境での繰り返し性能測定を通じ徐々

により高速なパターンを探索してきた．しかし，10 倍程度の高速化はできても，大きい高速化は難しかった．

なぜなら，GPU，FPGA，メニーコア CPU 等はハードウェア特性を生かし，並列化だけでなく，パイプライン処理，メモリ利用効率化（インタリーブ等）等駆使して高速化することが多いため，行列計算，画像処理，フーリエ変換等，計算タイプに応じてアルゴリズムから考える，手動高速化が殆どだからである．そこで，個々ループ文に対し判定するのでなく，より大粒度計算タイプの計算処理に対し，他者が別論文等で今までに検討しているアルゴリズムを適用する事で,自動での高速化を行う．

3.2　計算タイプに応じた計算処理検索と発見実装置換

コードを分析し(25)，オフロードできる計算タイプか把握する．計算タイプ把握するためにはパターンマッチング（例えば(26)解説）が利用できる．パターンマッチングは，特定のパターンを検索する技術である．個々の変数名や関数名には依存しない抽象語を用いて意味的に,抽象構文木で計算タイプに応じたプログラム構造に対し,マッチングできることが必要である．可能な市中ツールには，Semgrep(27)等が OSS で利用できる．

事前に，オフロードできる計算タイプ（行列計算，画像処理, フーリエ変換等）のコード, その検索パターン, それを処理する場合の実装をコードパターン DB に保持しておく. この DB の情報は, 高速化に用いられるので, クラウド事業者が VM の利用活性化を狙い準備する事を想定している．検索パターンはコード中のメインとなる計算処理部の，変数名や関数名を抽象語で置き換えた物である．各計算タイプの計算処理を GPU，FPGA，メニーコア CPU で処理する OpeACC や CUDA，OpenCL，OpenMP 実装に関しては,別実装された OSS 物を用いる．

コードがユーザから指定されたら，パターンマッチングでオフロード可能な計算タイプが含まれているか検索する．ここで，見つからない場合は，既存のループ文適切化の試行に移行する. 見つかる場合を, 以下詳説する．

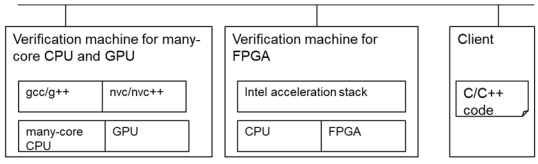
ステップ 1: オフロードしたいコードの構文解析

検索対象コードをパーサーで抽象構文木に変換する．

ステップ 2: 検索パターンの抽象構文木化

検索パターンもコードと同様，抽象構文木に変換する．

ステップ 3: 抽象構文木の木構造を走査マッチング



Name	Hardware	CPU	GPU	FPGA	RAM	OS	Tools
Verification machine for many-core CPU and GPU	LEVEL-R6W8-LCTP39-XAX	AMD Ryzen Threadripper PRO 3995WX (64 Core)	NVIDIA GeForce RTX 3090		128 GB	Ubuntu 24.04.3 LTS	gcc/g++ 13.3 nvc/nvc++ 25
verification machine for FPGA	Dell PowerEdge R740	Intel Xeon Bronze 3206R *2		Intel PAC D5005 (Intel Stratix 10 GX FPGA)	128 GB	CentOS 7.9	Intel Accelerat Stack 2.0
Client	Lenovo ThinkPad X1 Carbon Gen10	Intel Core i7-1260P			32 GB	Windows 11 Pro	

Fig. 1. Performance measurement environment

検索パターン抽象構文木を検索対象抽象構文木上に部分木のマッチを判定する．具体的には，抽象構文木部分木マッチングアルゴリズムで，パターン抽象構文木を対象抽象構文木に対して走査し,部分木同型性を調べる．

検索部分処理する OpeACC や CUDA, OpenCL, OpenMP は，行列計算，画像処理，フーリエ変換等高速化検討されてきた計算タイプで, ノウハウ詰まった実装と言える．

4. 評価

4.1　評価条件

FPGA とメニーコア CPU については，別論文で詳細評価しているため，本レターは GPU を題材にする．評価対象は, ユーザが GPU 利用想定されるフーリエ変換とする．

フーリエ変換（FFT : Fast Fourier Transform）は，振動周波数分析等様々な場面で利用されている．NAS.FT(28)は, FFT 処理の OSS の一つで離散 3 次元 FFT を行う（Class＝C）．FFT 処理高速化のため，従来から検討されてきた CUDA ファイル cuFFT(29)に置換する事で高速化する．

ユーザはオフロードしたいアプリケーションを指定し，Semgrep 1 でパターンマッチングされ，計算タイプに応じた計算処理自動オフロード（今回は GPU）がされる. オフロードされた際, 検索条件と結果ログ取得, CPU と GPU オフロード時の処理時間測定し，効果を見る．

評価環境とスペックを図 1 に示す．GPU は NVIDIA GeForce RTX 3090，FPGA は Intel Stratix 10 GX，メニーコア CPU は 64 コア AMD Ryzen 3995WX を用いる. 制御は GPU は nvc/nvc+ 25, FPGA は Intel Acceleration Stack 2, メニーコア CPU は gcc/g++ 13 を用いる．ノート PC が，オフロードするコードを指定し，検証環境性能測定通じオフロードパターン確定後, 商用環境にデプロイされる．

4.2　結果

図 2 は，Semgrep のパターンマッチングで NAS.FT を検索した際の検索条件と検索結果のログを示している．検索パターン自体は，変数名や関数名は抽象的に記述されているが，具体的な変数を持つ NAS.FT を抽象構文木の部分木マッチングにより発見できていることが分かる．

NAS.FT でパターンマッチングしオフロードできる計

Search pattern

```
NAS.FT_rules.yaml
rules:
  - pattern: |
      $M = ilog2(...);
      $U[0] = dcomplex_create($M, 0.0);
      $KU = 2;
      $LN = 1;

      for(...; ...; $J++){
        $T = PI / $LN;

        for(...; ...; $I++){
          $TI = $I * $T;
          $U[$I+$KU-1] = dcomplex_create(cos($TI),
sin($TI));
        }

        $KU = $KU + $LN;
        $LN = 2 * $LN;
      }
```

Search result

```
    m = ilog2(n);
      u[0] = dcomplex_create((double)m, 0.0);
      ku = 2;
      ln = 1;

      for(j=1; j<=m; j++){
        t = PI / ln;

        for(i=0; i<=ln-1; i++){
          ti = i * t;
          u[i+ku-1] = dcomplex_create(cos(ti),
sin(ti));
        }

        ku = ku + ln;
        ln = 2 * ln;
      }
```

Fig. 2. NAS.FT pattern matching search rules and results

Table 1. GPU offloading processing time result

Application	CPU processing time	Proposed method processing time (Change searched CUDA files)	Proposed method improvement ratio	(13)'s loop statements offloading improvement ratio
NAS.FT (Fourie Transform)	852 sec	7.13 sec	119	2.54

算タイプとし cuFFT に置換し GPU オフロードした場合の，1 コア CPU 処理時間，取得した CUDA ファイルに置換した GPU 処理時間，1 コア CPU に対する処理性能倍率と，(13)で遺伝的アルゴリズムを用いてループ文オフロードを GPU にした処理性能倍率を，表 1 に示す．

NAS.FT では，1 コア CPU の処理時間が 852 sec，取得した CUDA ファイル cuFFT に置換した GPU の処理時間が 7.13 sec で，処理性能倍率は 119 倍である．また，ループ文オフロードを GPU にした(13)の処理性能倍率は 2.54 倍であり，今回提案が性能倍率でよいことが分かる．

例えば，Amazon 社はクラウドで，少コア CPU に加え，GPU, FPGA, マルチコア CPU の VM を提供している(2)．月額利用料は，通常 CPU の VM は 60 USD/Month，GPU の VM は 200 USD，FPGA の VM は 700 USD，マルチコア CPU の VM は 140 USD 程度で，10 倍以上性能が出ている本手法はコスト的にも+効果がある．実験を通じ，パターンマッチングによりフロードできる計算タイプの計算対象を見つけオフロードする事で，コスト的にも意味あるオフロードができ方式が有効であることを示した．

5. まとめ

本稿では，ユーザが提供のアプリケーションを，計算タイプに応じて，適切な処理アルゴリズムで GPU, FPGA, メニーコア CPU に自動オフロードする方式を提案した．

今回検証では，フーリエ変換 NAS.FT を計算タイプ題材に，Semgrep で分析し，対応する CUDA ファイル cuFFT に置換して性能測定し，119 倍の性能向上を確認した．

文　献

(1) O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," IJCA, Vol.55, No.3, 2012.

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

(3) J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.

(4) J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," CiSE, Vol.12, pp.66-73, 2010.

(5) T. Sterling, et al., "High performance computing: modern systems and practices," Morgan Kaufmann, ISBN: 9780124202153, 2017.

(6) Y. Yamato, "Proposal of Automatic GPU Offloading Method from Various Language Applications," ICIET 2021, pp.400-404, Mar. 2021.

(7) Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet Things J., 2018.

(8) Y. Yamato, "Study for division of general-purpose software that helps with customization," ICIET 2024, Mar. 2024.

(9) Y. Yamato, "Automatic Verification Technology of Software Patches for User Virtual Environments on IaaS Cloud," J. Cloud Comput., Springer, Vol.4, No.4, 2015.

(10) Y. Yamato, "Study of software reconfiguration after adapted service start," IECC 2023, pp.63-68, July 2023.

(11) Y. Yamato, "Evaluation of GPU Logic Reconfiguration after Service Start," ICIET 2023, pp.551-556, Mar. 2023.

(12) Y. Yamato, "Study and Evaluation of Automatic Offloading for Function Blocks of Applications," Automatika, Taylor & Francis, Vol.65, Issue.1, pp.387-400, Jan. 2024.

(13) Y. Yamato, "Proposal and evaluation of GPU offloading parts reconfiguration during applications operations for environment adaptation," J. Netw. Syst. Manag., Springer, Nov. 2023.

(14) Y. Yamato, "Study and Evaluation of FPGA Reconfiguration during Service Operation for Environment-Adaptive Software," Int. J. Parallel Emergent Distrib. Syst., Taylor & Francis, Aug. 2023.

(15) Y. Yamato, "Study and Evaluation of Optimum Location Deployment for Environment Adaptive Applications," Int. J. Parallel Emergent Distrib. Syst., Taylor & Francis, June 2022.

(16) Y. Yamato, "Proposal and Evaluation of Adjusting Resource Amount for Automatically Offloaded Applications," Cogent Engineering, Taylor & Francis, Vol.9, Issue 1, June 2022.

(17) Y. Yamato, "Study and Evaluation of Automatic Offloading Method in Mixed Offloading Destination Environment," Cogent Engineering, Taylor & Francis, Vol.9, Issue 1, June 2022.

(18) Y. Yamato, "Study and evaluation of automatic division of general-purpose programs to facilitate addition of user functions," Int. J. Parallel Emergent Distrib. Syst., Taylor & Francis, Aug. 2024.

(19) Y. Yamato, "Study and evaluation for adopting environmental adaptation of low-resource devices," IEEE Access, Aug. 2024.

(20) gcc website, https://gcc.gnu.org/

(21) S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

(22) M. Wolfe, "Implementing the PGI accelerator model," GPGPU-3, 2010.

(23) E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," EWOMP 2002, Sep. 2002.

(24) J. H. Holland, "Genetic algorithms," scientific american, vol.267, 1992.

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

(26) Haskell site, https://wiki.haskell.org/Declaration_vs._expression_style

(27) Semgrep website, https://github.com/semgrep

(28) NAS.FT website, https://www.nas.nasa.gov/software/npb.html

(29) cuFFT website, https://developer.nvidia.com/cufft

山登　庸次

2000 年東京大学理学部卒．2009 年同大学院総合文化研究科博士（学術）．2002 年日本電信電話（株）入社．NTT（株）にて，IoT，適応ソフトウェアの研究開発に従事．現在，同社研究所特別研究員．電子情報通信学会フェロー，IEEE シニア会員．