一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
NS2025-187(2025-12)

環境適応ソフトウェアにおけるFPGAオフロード対象拡大の提案

山登庸次†

† NTT株式会社　ネットワークサービスシステム研究所，東京都武蔵野市緑町3-9-11
E-mail: †yoji.yamato@ntt.com

あらまし　私達は，プログラマーが通常CPU向けに記述したコードを，配置環境に応じて，自動で変換等をして，高性能に運用可能とする環境適応ソフトウェアを提案してきた．本稿は，画像処理，信号処理等の処理の計算タイプに応じた，FPGAへの自動オフロードを対象とする．オフロードしたい既存のアプリケーションをパターンマッチングで抽象構文木を用いて意味的に分析し，置換可能なOpenCLがある計算タイプか把握する．OpenCLが見つかった場合は，そのOpenCLに置換し性能向上を確認する．提案方式で自動オフロードできることを，Intel Stratixの実FPGAを用いて，処理時間を計測して確認する．

キーワード　環境適応ソフトウェア, 自動オフロード，FPGA，計算タイプ，パターンマッチング

Proposal for expanding FPGA offloading targets for environment adaptive software

Yoji YAMATO†

† Network Service Systems Labs., NTT, Inc., 3-9-11, Midori-cho, Musashino-shi, Tokyo
E-mail: †yoji.yamato@ntt.com

Abstract　We have proposed environment-adaptive software that automatically converts code written by programmers for regular CPUs according to the deployment environment, enabling high-performance operation. This paper focuses on automatic offloading to FPGAs according to the computation type of processing, such as image processing and signal processing. The existing application to be offloaded is semantically analyzed using pattern matching and an abstract syntax tree to determine whether the computation type has an OpenCL that can be replaced. If OpenCL is found, the application is replaced with that OpenCL and performance improvements are confirmed. The effectiveness of the proposed method for automatic offloading is confirmed by measuring processing time using an actual Intel Stratix FPGA.

Key words　Environment-Adaptive Software, Automatic Offloading, FPGA, Computation Type, Pattern Matching

1. はじめに

CPU（Central Processing Unit）が年々集積度が上がり，安く高速化されることを予測した，ムーアの法則があるが，スピードは鈍ってきたと言われる．そのため，少コアのCPUだけでなく，メニーコアのCPUやGPU（Graphics Processing Unit），FPGA（Field Programmable Gate Array），IoT機器等のヘテロジニアスなハードウェアを利用したシステムが増えている．Microsoft社はFPGAの検索利用を述べている[1]，Amazon社は，GPU，FPGAをクラウド（例えば，[2] [3]）インスタンスとして提供している[4]．また，FPGAやGPU等のアクセラレータだけでなく，IoT機器も利用が増えている（例えば，[5]- [8]）．

しかしながら，ヘテロジニアスハードウェアを効率良く利用するためには，ハードウェアの特性を理解したコードや設定が必要であり，OpenCL（Open Computing Language）[9]，CUDA（Compute Unified Device Architecture）[10]，OpenMP（Open Multi-Processing）[11] 等のC言語拡張を用いたコードが必要となる．そのため，多くのプログラマーにとっては，ハードルが高い

現在生成AIでGPUが使われ，その電力消費が大きな問題となっている．FPGAはハードウェア処理で電力削減できる可能性が高く今後活用が期待されるが，FPGAの利用にはOpenCLやSYCL [12] 等のスキルが必要でハードルが高い．そこで，

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2025 by IEICE

ハードルを下げ，ヘテロジニアスハードウェアを容易に利用できるようにするため，少コア CPU の時と同じように記述したコードを，動作させる環境（メニーコア CPU，GPU，FPGA 等）に合わせて，自動で変換をし，環境適応させる，プラットフォームが必要になってくる．ここで，ヘテロジニアスハードウェア利用は手動改造が殆どである中で，自動で変換等を行う事がポイントである．

そこで，私達は，以前からあるコードを，動作させる環境で利用できるよう，GPU や FPGA 向けに自動変換し，アプリケーションを高速化する，環境適応ソフトウェアのコンセプトを提案してきた．環境適応ソフトウェア要素として，以前からあるコードのループ文を，GPU，FPGA 等に自動オフロードする方式等を提案し評価している [13]- [27].

しかし，これまでの私達は，FPGA にはループ文の自動オフロードを主に検証してきた．これは，ある程度の高速化は可能だが，計算タイプに合わせてハードウェア処理を意識したアルゴリズムを考えた手動で OpenCL を作成した高速化には及ばなかった．

本稿は，画像処理，信号処理等の処理の計算タイプに応じた，FPGA への自動オフロードを対象とする．オフロードしたい既存のアプリケーションをパターンマッチングで抽象構文木を用いて意味的に分析し，置換可能な高速化 OpenCL がある計算タイプか把握する．高速化 OpenCL がない場合は，以前検討のループ文高速化を試行する．置換可能な高速化 OpenCL が見つかった場合は，その高速化 OpenCL に置換する．提案方式で自動オフロードできることを，Intel Stratix の実 FPGA を用いて，処理時間を計測して確認する．

2. 既存技術

2.1 市中技術

FPGA，GPU，メニーコア CPU 等のヘテロジニアスハードウェアを統一的に扱う仕様に OpenCL が定義されており，OpenCL 解釈実行ツールも各社提供している．OpenCL は，C 言語拡張のソフトウェア仕様であり，ハードルは高い（FPGA 等デバイス側のカーネルとホストとの間のメモリデータの開放や移動や複製の記述を明示的に行う）．

SYCL [12] は，ヘテロジニアスハードウェアでの単一ソースプログラミングモデルである．OpenCL では，ホストコードとカーネルコードは別々に記述されるが，SYCL では，単一のソースで記述できる．SYCL は，GPU や FPGA などの複数のハードウェアで実行される単一のコードを対象としているため，その点は OpenCL より改善されていると言えるが，単一のコードはプログラマーによって新たに作成される必要がある．

OpenCL や SYCL と異なり，容易にヘテロジニアスハードウェアを使うため，指示行（Directive）を使い，特定処理を行う部分を指示行で指定し，指示行に従って GPU や FPGA に合わせたバイナリファイルを作成するツールがある．GPU 向け仕様，ツールには OpenACC [28] や PGI コンパイラ [29] が，メニーコア CPU 向け仕様，ツールには OpenMP や gcc がある．FPGA 向けには高位合成 HLS(High Level Syntesis) [30] ツールがあり，ツール毎に異なるが例えば#pragma 等を使いループ文展開等が指示句に応じてできる．

OpenCL，SYCL，HLS 等を用いて，FPGA 等のヘテロジニアスハードウェア処理は可能になっている．しかし処理は行えても，高速化は難しいのが現状である．例えば，メニーコア CPU 向けに，Intel コンパイラ [31] がある．これは，ループ文の並列処理可能部を並列化する．しかし，メモリデータ利用の効率があり，単にループ文を並列化しても高速処理されないことが多い．また，FPGA では，ハードウェア特性を生かしたパイプライン処理等で効率的に並列処理する事が高速化に必要となることが多く，OpenCL や HDL(Hardware Description Language) [32] の専門家等がチューニング作業を行い，ツールを繰り返し試行して適切な OpenCL や HDL の設定がされている．著者はループ文オフロード自動化をねらい，算術強度とループ回数とリソース効率から FPGA にオフロードする候補ループ文を選択し，候補に該当する複数パターンの実測を通じて，高速なオフロードパターンを選択決定する方法を提案している．

2.2 環境適応ソフトウェアの概要

図 1 で，私達は環境適応ソフトウェアの 7 ステップの処理を提案している．環境適応ソフトウェア処理では，クラウド等の事業者が提供する環境適応機能が中心に存在し，検証環境，商用環境，コードパターン DB，テストケース DB，設備リソース DB が連携する．

Step1　ユーザ提供コード分析：
Step2　オフロード可能部分抽出：
Step3　適切なオフロード部分探索：
Step4　商用リソース量調整：
Step5　商用デプロイ場所調整：
Step6　バイナリファイル商用配置と検証：
Step7　商用運用中再構成：

運用開始前処理として，Step1-6 で，ユーザ提供コードをまず分析し，検証環境での性能測定試験を繰り返して，適切なコードに変換，リソース量の決定，デプロイ場所の決定，正常動作の検証をする．運用開始後処理として，Step7 は，実利用データの傾向を分析して，動作コードやリソース量やデプロイ場所等の，商用構成を変更した方が適切な事が明らかな場合は再構成を行う．

2.3 本稿の課題

本稿で扱う課題を提示する．ヘテロジニアスハードウェア用いたアプリケーション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のコンセプトを提案し，FPGA や GPU 等への自動オフロード方式も実現してきた．しかし，今までは個々のループ文の自動オフロードが主な対象であった．そのため，数倍程度の高速化は可能だが，計算タイプに合わせてハードウェア処理を意識したアルゴリズム含めて考えた手動 OpenCL での高速化には及ばなかった．本稿は，画像処理，信号処理等の計算処理の計算タイプに応じた，FPGA への自動オフロードを対象とする．パターンマッチングにより，計算タイプに応じて，既存ノウハウが詰まった実装に置き換え

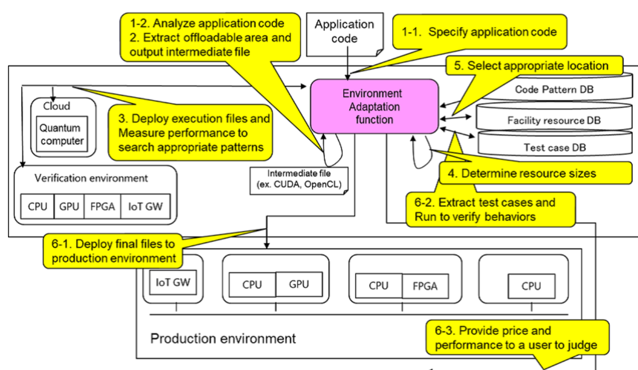


図 1　環境適応ソフトウェアの処理概要

ることで，アルゴリズム考慮して自動で高速化する．実 FPGA ボードの Intel Stratix 搭載機で，提案方式有効性を示す．

3. 計算タイプに応じた計算処理の FPGA へのオフロード

本節では，計算タイプに応じた計算処理の FPGA へのオフロードを提案する．3.1 節では以前論文提案の，ループ文の FPGA への自動オフロード方式を確認する．3.2 節では，個々のループ文オフロードを踏まえ，より大きい粒度でのオフロードについてアイデアを述べる．3.3 節では，パターンマッチングを用いた計算タイプの計算処理判定を提案する．3.4 節では判定した計算処理に応じた OpenCL 置換による高速化を提案する．

3.1　ループ文の FPGA の自動オフロード方式

まず，オフロードしたいコードを分析し，ループ文や変数情報を把握する．

把握したループ文に対して，FPGA オフロードを試行するかどうか候補を絞っていく．ループ文がオフロード効果あるかどうかは，算術強度が一つの指標となりうる．算術強度は，計算回数が多いと増加し，データサイズが大きいと減少する指標で，算術強度が高い処理は重い処理となり時間がかかる．そこで，ループ文の算術強度を分析し，強度が高いループ文をオフロード候補に絞る．算術強度分析には ROSE framework [33] が利用できる．また，ループ回数が多いループも重い処理となる．ループ回数はプロファイラーで分析し，ループ回数が多いループ文もオフロード候補に絞る．ループ回数分析には gcov [34] が利用できる．

高算術強度やループ回数が多いループ文であっても，それを FPGA で処理する際に，リソースを過度に消費してしまうのは問題である．FPGA にコンパイルする際の途中状態の HDL レベルで，FPGA 利用リソースは分かるため，利用リソース量は短時間で分かる．オフロード候補のループ文を OpenCL 言語化し，リソース量を算出することで，オフロードした際の算術強度とリソース量が決まるため，算術強度/リソース量をリソース効率とする．本手法では，高リソース効率のループ文をオフロード候補として更に絞り込む．ここで，ループ文を OpenCL 言語化する際には，CPU 処理のプログラムを，カーネル（FPGA）とホスト（CPU）に，OpenCL の文法に従って分割する．

高リソース効率ループ文が絞られるため，それらを用いて性能測定するパターンを作成する．絞り込まれた単ループ文とその組み合わせのパターンを一定数作り，FPGA で動作するようコンパイルする．最後に検証環境で，コンパイルされた複数パターンの性能測定を行い，高速のパターンを解として選択する．

3.2　中粒度での FPGA オフロードのアイデア

3.1 節方式を用いて，算術強度やループ回数が高くリソース効率が高いループ文に絞って，オフロードパターンを作り，検証環境測定を通じ高速パターン探索を行うことができる．しかし，個々ループ文に対しオフロードするか判定する方式では数倍程度のある程度高速化はできても，極めて大きい高速化は難しかった．

なぜなら，FPGA はハードウェア処理の特性を生かし，パイプライン処理等を駆使して高速化することが多いため，画像処理，信号処理，行列計算，ステンシル計算等，計算タイプに応じて FPGA 処理のアルゴリズムから考える，手動での高速化が殆どだからである．そこで，個々のループ文に対し判定するのでなく，より大きな粒度の計算タイプに応じた計算処理に対し，多くの方が別論文等で今までに検討している FPGA 処理アルゴリズムを適用する事で，自動での高速化を行う．

動作概要としては，以下の 2 ステップからなる．まず，オフロードしたいコードに，FPGA オフロードできる計算タイプの計算処理が含まれるかを分析する．それが含まれている際に，その計算処理の FPGA 処理に該当する既存ノウハウが含まれた実装に置換することで処理を高速化する．ここで，1 ステップ目を 3.3 節で，2 ステップ目を 3.4 節で詳細を説明する．

3.3　計算タイプに応じた計算処理の検索

コードを分析し，オフロードできる計算タイプの計算処理が含まれているか把握する．どのような計算タイプかを把握するためにはパターンマッチング（例えば, [35] 解説）が利用できる．パターンマッチングは，特定のパターンが含まれているか検索する技術である．計算タイプを判定するため，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．このようなパターンマッチング可能な市中ツールには，Semgrep [36] 等が OSS で利用できる．

パターンマッチング検索のため事前に，FPGA にオフロードできる計算タイプ（画像処理，信号処理，行列計算，ステンシル計算等）のコード，その検索パターン，それを FPGA で処理する場合の OpenCL のコードをコードパターン DB に保持しておく．この DB の情報は，FPGA オフロード高速化に用いられるので，FPGA インスタンスを提供するクラウド事業者が FPGA インスタンス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 FPGA で処理する OpenCL に関しては，別の方論文等で検討され実装された無料の物を用いる．

高速化検討された OpenCL で，例えば，Intel や Xilinx が提供している OSS の OpenCL がある．また，多くの既存研究で

も，OpenCL 使った FPGA での計算タイプに応じた高速化検討がされている．[37] は，姫野ベンチマークというメモリ重視の流体ステンシル計算で，時空間ブロッキングとシフトレジスタの実装技法を組合わせることで高速化している．

オフロードしたい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，3.1 節のループ文の FPGA オフロード高速化の試行に移行する．見つかる場合を，以下で詳説する．

パターンマッチングツールでの検索条件は，コードパターン DB に登録された検索パターンを順番で試行する事で自動で行う．ユーザが提供するオフロードしたいコードが検索対象となり，抽象語でパターンマッチングされる．

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の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に，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としてマッチするかどうかを判定する．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．

このようにすることで，コードパターン DB に保持された FPGA にオフロードできる計算タイプの計算処理を含むコードかの判定ができる．

3.4 OpenCL 置換による高速化

FPGA にオフロードできる計算タイプの計算処理を含むコードか判定できるため，検索された部分を FPGA で処理する場合の OpenCL のコードに置換することで高速化する．OpenCL のコードは，画像処理，信号処理，行列計算，ステンシル計算等で他論文等で手動改造で高速化が検討されてきた計算タイプであり，専門家の今までのノウハウが詰まった実装と言える．

ただし，オフロードしたいコードをパターンマッチングし，オフロードできる計算タイプをコードパターン DB の OpenCL に置換するため，引数や戻り値の数や型等の部分が，ユーザ要望と合っている保証はない．合っていない場合は，OpeCL は既存ノウハウであり頻繁に変更できるものでないため，オフロードを依頼するユーザに対して，元のコードの引数や戻り値の数や型について，OpenCL に合わせて変更するか確認し，確認了承後にオフロード性能試験を試行する．型の違いについて，float と double 等自動でキャストすればよいだけであれば，特にユーザ確認せずに試行に入ってもよい．また，引数や戻り値で，元のコードと OpenCL で数が異なる場合に，例えば，ユーザコードで引数 1, 2 が必須で 3 がオプションであり，OpenCL で引数 1, 2 が必須の場合等，省略しても問題ない場合は，ユーザに確認せず，オプション引数は自動で無しとしてもよい．

4. 評　　価

個々のループ文オフロードを判定し FPGA へ自動オフロードでなく，より大きな粒度の計算タイプに応じた計算処理の FPGA 自動オフロードの提案方式の有効性を評価する．

4.1 評 価 条 件

4.1.1 評 価 対 象

評価対象は，多くのユーザが FPGA で利用すると想定される画像処理と信号処理とする．

MRI-Q [38] は，キャリブレーションのためのスキャナー設定を表現する Q マトリックを計算する MRI 画像処理である．MRI-Q は非カルテアン空間で 3D MRI 再構成アルゴリズムで使用される．IoT 等で，画像処理はしばしばカメラ映像の自動監視等に必要となり，画像処理のスループット等の性能は，強化が要望される．オフロードパターン抽出時の性能測定では，MRI-Q は 3D MRI 画像処理を実行し，データサイズによるが，想定利用では，64*64*64 サイズのデータを使用して処理時間を測定する．

信号処理の有限インパルス応答フィルタ（tdFIR）は，システムにインパルス関数を入力したときの出力に対して有限時間で打ち切る処理を行う，フィルタの一種である．実装は種々あるが, [39] の C コードを用いる．IoT 等で，デバイスからの信号データをネットワーク転送するアプリケーションを考えた際に，ネットワークコストを下げるため，フィルタ等の信号処理をしてからクラウドにデータを送ることは想定される．そのため，信号処理の FPGA での自動高速化は応用範囲が広いと考える．

4.1.2 評 価 手 法

ユーザはオフロードしたいアプリケーションを 2 つ指定し，パターンマッチングされ，計算タイプに応じた計算処理の FPGA 自動オフロードがされる．オフロードされた際は，検索条件と結果のログ取得，CPU だけ処理と FPGA オフロードした処理時の処理時間（メモリデータコピー時間等あるため，実経過時間で比較）を測定し，オフロード効果を見る．また，パターンマッチングでオフロードできる計算タイプが見つからない場合はループ文のオフロードを行う．

FPGA オフロードの条件は以下で行う．

オフロード対象：ループ文数　MRI-Q 16，tdFIR 6.

パターンマッチング利用ツール：Semgrep 1

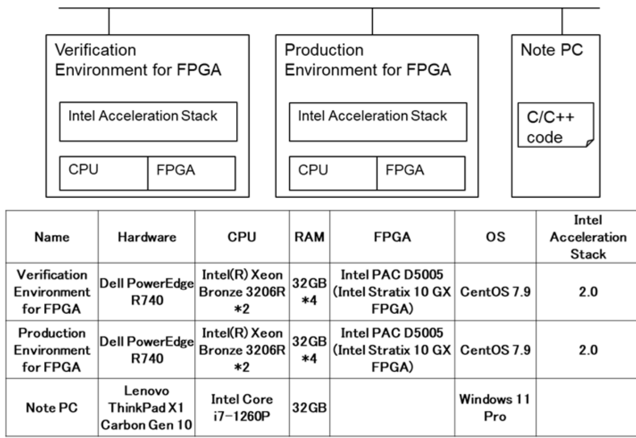
算術強度絞り込み：Rose framework で算術強度分析の上位 4 つのループ文に絞り込み

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

実測オフロードパターン数：4（1 回目は上位 3 つのループ文オフロードパターンを測定し，2 回目は 1 回目で高性能だった 2 パターン組合せを測定）

4.1.3 評 価 環 境

評価用 FPGA として Intel FPGA PAC D5005 (Intel Stratix 10 GX FPGA, Logic Element 2,800,000) を用いる．Intel FPGA PAC D5005 搭載機は，DELL EMC PowerEdge R740 (CPU：Intel Xeon Bronze 3206R *2, RAM：32GB RDIMM * 4) である．FPGA の制御は, Intel Acceleration Stack Version 2.0 を用いる．OpenCL の文法に従い，C 言語プログラムを，カーネルプログラムとホストプログラムに分割記載することで，FPGA オフロード処理が OpenCL でされる．



Name	Hardware	CPU	RAM	FPGA	OS	Intel Acceleration Stack
Verification Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9	2.0
Production Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9	2.0
Note PC	Lenovo ThinkPad X1 Carbon Gen 10	Intel Core i7-1260P	32GB		Windows 11 Pro	

図 2　性能測定環境

評価環境とスペックを図 2 に示す．ここで，ノート PC が，オフロードするアプリケーションコードを指定し，検証環境での性能測定を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 3 は，Semgrep のパターンマッチングで MRI-Q 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 MRI-Q 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．tdFIR でも同様に抽象語での抽象構文木マッチングにより発見できる．

図 4 は MRI-Q と tdFIR で，パターンマッチングしオフロードできる計算タイプを FPGA 自動オフロードした場合の，CPU だけの処理時間，FPGA オフロード含めた処理時間，処理性能倍率を示している．

MRI-Q では，CPU だけの処理時間が 27.5 sec，FPGA オフロード含めた処理時間が 2.23 sec で，処理性能倍率は 12 倍である．tdFIR では，CPU だけの処理時間が 0.27 sec，FPGA オフロード含めた処理時間が 0.0057 sec で，処理性能倍率は 47 倍である．

例えば, Amazon 社はクラウドで, 通常の CPU に加え, FPGA, GPU，メニーコア CPU のインスタンスを提供している [4]．月額利用料は，通常 CPU の VM は 60 USD/Month，FPGA の VM は 700 USD/Month，GPU の VM は 200 USD/Month 程度であり，12 倍以上性能が出る 2 つのアプリケーションはコスト的にもプラス効果があると言える．

実験を通じて，パターンマッチングにより FPGA オフロードできる計算タイプの計算対象を見つけ，FPGA オフロードする事で，コスト的に意味があるオフロードができ方式が有効であることを示した．

4.3 考　　察

以前のループ文の FPGA オフロードでは，複数のオフロードパターンを検証環境で性能測定し，高速なパターンに自動でしていく手法で，Intel Arria 10 FPGA を用いて自動高速化をしてきた．FPGA は，ハードウェア特性に応じたプログラムが必要で，手動での設計が主流で，自動オフロードは大きな差異

Search pattern

```
MRI-Q_rules.yaml
rules:
  - id: detect-MRI-Q
    pattern-either:
      - pattern: |
          for (...) {
            ...
            $PHI_MAG = $REAL * $REAL + $IMAG * $IMAG;
            ...
          }
      - pattern: |
          for (...) {
            $QR_ACC = $INIT_VAL;
            $QI_ACC = $INIT_VAL;
            for (...) {
              $EXP_ARG = $PIX2 * ( $KVALS_KX * $X +
$KVALS_KY * $Y + $KVALS_KZ * $Z );
              $COS_ARG = cosf( $EXP_ARG );
              $SIN_ARG = sinf( $EXP_ARG );
              $PHI = $KVALS_PHIMAG;
              $QR_ACC += $PHI * $COS_ARG;
              $QI_ACC += $PHI * $SIN_ARG;
            }
            $QR = $QR_ACC;
            $QI = $QI_ACC;
          }
    message: "The formula for MRI-Q has been found."
    languages: [c, cpp]
    severity: INFO
```

Search result

```
  for (indexK = 0; indexK < numK; indexK++) {
    float real = phiR[indexK];
    float imag = phiI[indexK];
    phiMag[indexK] = real*real + imag*imag;
  }

and

  for (indexX = 0; indexX < numX; indexX++) {

    // Sum the contributions to this point over all
frequencies
    float Qracc = 0.0f;
    float Qiacc = 0.0f;
    for (indexK = 0; indexK < numK; indexK++) {
      expArg = PIx2 * (kVals[indexK].Kx * x[indexX] +
                       kVals[indexK].Ky * y[indexX] +
                       kVals[indexK].Kz * z[indexX]);

      cosArg = cosf(expArg);
      sinArg = sinf(expArg);

      float phi = kVals[indexK].PhiMag;
      Qracc += phi * cosArg;
      Qiacc += phi * sinArg;
    }
    Qr[indexX] = Qracc;
    Qi[indexX] = Qiacc;
  }
```

図 3　MRI-Q のパターンマッチング検索条件と検索結果

Applications	CPU only processing time	FPGA offloading processing time	improvement ratio
MRI-Q	27.5 sec	2.23 sec	12
tdFIR	0.27 sec	0.0057 sec	47

図 4　FPGA オフロード後処理時間結果

点と言えた．今回，個々のループ文オフロード判定でなく，より大きな粒度の計算タイプに応じた計算処理の FPGA 自動オフロードを対象にすることで，ループ文オフロードより大きな性能改善が見込め，FPGA オフロードの大きな進展と言える．

5. ま と め

本稿では，私達が提案している環境適応ソフトウェアの拡張として，ユーザが提供するアプリケーションを，個々のループ文によらず分析し，画像処理，信号処理等の処理の計算タイプに応じて，適切な処理アルゴリズムで FPGA に自動オフロードする方式を提案した．

まず，ユーザアプリケーションを分析する．パターンマッチングツールの Semgrep で分析し，画像処理，信号処理等の計算タイプに応じた処理パターンがないか検索する．なお，Semgrep でのマッチング検索のため，事前にコードと検索パターンとそれに対応する OpenCL をコードパターン DB に保持しておく．Semgrep のパターンマッチングでは，抽象構文木を用いた意味的検索で，置換可能な OpenCL がある計算タイプの計算処理を含むか検索できる．OpenCL がない場合は，以前検討の算術強度とループ回数とリソース効率を用いたループ文高速化の試行を行う．置換可能な OpenCL が見つかった場合は，その OpenCL に置換し，性能向上されるか性能測定を行う．価格的にも FPGA にオフロードする意味がある場合はそのオフロードを行う．

今回，3D 画像処理の MRI-Q，信号処理の tdFIR を計算タイプ例に，Semgrep で分析し，対応する OpenCL に置換して性能測定し，FPGA インスタンスの価格的にもオフロード意味がある，10 倍以上の性能向上を確認し，方式有効性を示した．

文　献

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), pp.13-24, June 2014.

[2] O. Sefraoui, M. Aissaoui and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.

[3] Y. Yamato, "Automatic Verification Technology of Software Patches for User Virtual Environments on IaaS Cloud," Journal of Cloud Computing, Springer, Vol.4, No.4, DOI: 10.1186/s13677-015-0028-6, Feb. 2015.

[4] AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/

[5] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[6] H. Sunaga, et al., "Ubiquitous Life Creation through Service Composition Technologies," World Telecommunications Congress 2006 (WTC 2006), Budapest, May 2006.

[7] Y. Yamato, H. Ohnishi and H. Sunaga, "Study of Service Processing Agent for Context-Aware Service Coordination," IEEE International Conference on Service Computing (SCC 2008), pp.275-282, July 2008.

[8] Y. Yamato, Y. Nakano and H. Sunaga, "Study and Evaluation of Context-Aware Service Composition and Change-over Using BPEL Engine and Semantic Web Techniques," IEEE Consumer Communications and Networking Conference (CCNC 2008), pp.863-867, Jan. 2008.

[9] J. E. Stone, D. Gohara and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.

[10] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.

[11] T. Sterling, M. Anderson and M. Brodowicz, "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

[12] SYCL web site, https://www.khronos.org/sycl/

[13] Y. Yamato, "Proposal of Automatic GPU Offloading Method from Various Language Applications," The 9th International Conference on Information and Education Technology (ICIET 2021), pp.400-404, Mar. 2021.

[14] Y. Yamato, "IoT application adopting for automatic software division," 2024 6th International Electronics Communication Conference (IECC 2024), July 2024.

[15] Y. Yamato, "Study for division of general-purpose software that helps with customization," The 12th International Conference on Information and Education Technology (ICIET 2024), Mar. 2024.

[16] Y. Yamato, "A study for environmental adaptation of IoT devices," 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW 2023) pp.14-19, Nov. 2023.

[17] Y. Yamato, "Study of software reconfiguration after adapted service start," 2023 5th International Electronics Communication Conference (IECC 2023), pp.63-68, July 2023.

[18] Y. Yamato, "Evaluation of GPU Logic Reconfiguration after Service Start," The 11th International Conference on Information and Education Technology (ICIET 2023), pp.551-556, Mar. 2023.

[19] Y. Yamato, "Study and Evaluation of Automatic Offloading for Function Blocks of Applications," Automatika, Taylor & Francis, Vol.65, Issue.1, pp.387-400, DOI: 10.1080/00051144.2024.2301888, Jan. 2024.

[20] Y. Yamato, "Proposal and evaluation of GPU offloading parts reconfiguration during applications operations for environment adaptation," Journal of Network and Systems Management, Springer, DOI: 10.1007/s10922-023-09789-2, Nov. 2023.

[21] Y. Yamato, "Study and Evaluation of FPGA Reconfiguration during Service Operation for Environment-Adaptive Software," International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2023.2242639, Aug. 2023.

[22] Y. Yamato, "Study and Evaluation of Optimum Location Deployment for Environment Adaptive Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2022.2088749, June 2022.

[23] Y. Yamato, "Proposal and Evaluation of Adjusting Resource Amount for Automatically Offloaded Applications," Cogent Engineering, Taylor & Francis, Vol.9, Issue 1, DOI: 10.1080/23311916.2022.2085467, June 2022.

[24] Y. Yamato, "Study and Evaluation of Automatic Offloading Method in Mixed Offloading Destination Environment," Cogent Engineering, Taylor & Francis, Vol.9, Issue 1, DOI: 10.1080/23311916.2022.2080624, June 2022.

[25] Y. Yamato, "Study and evaluation of automatic division of general-purpose programs to facilitate addition of user functions," International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, DOI: 10.1080/17445760.2024.2375650, Aug. 2024.

[26] Y. Yamato, T. Demizu, H. Noguchi and M. Kataoka, "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.

[27] Y. Yamato, "Study and evaluation for adopting environmental adaptation of low-resource devices," IEEE Access, DOI: 10.1109/ACCESS.2024.3440918, Aug. 2024.

[28] S. Wienke, P. Springer, C. Terboven and D. an Mey, "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

[29] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.

[30] M.C.McFarland, A.C.Parker and R.Camposano, "The high-level synthesis of digital systems," Proceedings of the IEEE, Vol.78, No.2, pp.301-318, Feb. 1990.

[31] E. Su, X. Tian, M. Girkar, G. Haab, S. Shah and P. Petersen, "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.

[32] HDL website, https://curlie.org/Science/Technology/Electronics/Design/Hardware_Description_Languages/

[33] ROSE framework website, http://rosecompiler.org/ROSE_HTML_Reference/index.html

[34] gcc website, https://gcc.gnu.org/

[35] Haskell description website, https://wiki.haskell.org/Declaration_vs._expression_style

[36] Semgrep website, https://github.com/semgrep

[37] I. Firmansyah, "OpenCL-based design methodologies for FPGA implementation," tsukuba repository, 2020.

[38] MRI-Q website, http://impact.crhc.illinois.edu/parboil/

[39] Time domain finite impulse response filter web site, http://www.omgwiki.org/hpec/files/hpec-challenge/tdfir.html