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

信学技報
TECHNICAL REPORT OF IEICE.

環境適応ソフトウェアオフロード対象計算タイプの整理

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，通常スキルプログラマーが少コア CPU 向けに記述したソフトウェアコードを，配置環境に応じて，自動で変換，設定等して，高性能に処理する環境適応ソフトウェアを提案している．本稿は，フーリエ変換等の処理の計算タイプに応じた，多様なハードウェア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，置換可能な実装がある計算タイプか把握する．実装が見つかった場合は，その実装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，NVIDIA GeForce RTX 3090 の GPU 搭載した実機を用いて確認する．

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

Organizing the offloading target computation types 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 propose environment-adaptive software that automatically converts and configures software code written by skilled programmers for low-core CPUs according to the deployment environment, enabling high-performance processing. This paper focuses on automatic offloading to a variety of hardware according to the calculation type of processing, such as Fourier transforms. The existing application to be offloaded is semantically analyzed using an abstract syntax tree with pattern matching to determine whether a replaceable implementation exists for the calculation type. If an implementation is found, it is replaced with that implementation and performance improvements are confirmed. The ability of the proposed method to automatically offload is confirmed using an actual device equipped with an NVIDIA GeForce RTX 3090 GPU.

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

1. はじめに

近年，CPU（Central Processing Unit）の集積度向上を予想した，ムーアの法則が当てはまらなくなってきたと言われる．そのため，通常使われる少コアの CPU だけでなく，GPU（Graphics Processing Unit）や FPGA（Field Programmable Gate Array）や 30 コア以上のメニーコア CPU や量子コンピュータや IoT 機器などの多様な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，少コア CPU の VM（Virtual Machine）だけでなく，GPU，FPGA，マルチコア CPU の VM をクラウド（例えば, [1] [2]）で提供している [3]．Microsoft 社は FPGA の検索利用を述べており [4]，またクラウドで多種の VM を提供している．さらに，IoT PF をクラウドで提供し，接続する IoT 機器を用いた IoT サービスも増えている（例えば, [5]- [8]）．

しかし, 多様なハードウェアを高性能で利用するためには, ハードウェアの特性を生かしたプログラムが必要であり，GPU では CUDA（Compute Unified Device Architecture）[9], FPGA では OpenCL（Open Computing Language）[10]，メニーコア CPU では OpenMP（Open Multi-Processing）[11] 等のプログラム言語が前提となることが多い．そのため，Python，PHP 等のスクリプト言語にノウハウがある多くのプログラマーにとって，難度が高い．

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 は電力消費は GPU に比べ大きく下がるため，同程度の性能であれば，GPU でなく FPGA にオフロードした方が環境負担は下がる．しかし，現在 FPGA をはじめ多様なハードウェアで高性能処理するには OpenCL 等のスキルが必要で難度が高い．そこで，難度を下げ，多様なハードウェアを高性能に利用できるように，通常の少コア CPU 利用時と同様に記述したプログラムを，動作環境（GPU，FPGA，メニーコア CPU 等）に合わせて，自動変換や設定をし環境適応させるプラットフォームが必要になる．

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，GPU や FPGA やメニーコア CPU 向けに自動変換し，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のコンセプトを提案してきた．更に，環境適応ソフトウェアの要素技術として，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 [12]- [26].

しかし，これまでの私達は，多様なハードウェア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アルゴリズムを考えた手動で改造した高速化には性能は及ばなかった．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．オフロードしたい既存プログラム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，置換可能な高速化実装がある計算タイプかを把握する．置換可能な実装がない場合は，従来方式のループ文高速化を試行する．置換可能な実装が見つかった場合は，その高速化実装に置換する．提案する方式で自動でオフロードできることを，NVIDIA GeForce RTX 3090 の実 GPU 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

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（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）．

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）を使い，特定処理を行う部分を指示句で指定し，指示句に従ってバイナリファイルを作成する仕様がある．例えば，メニーコア CPU 等で多数コアを用いて計算処理するための仕様として OpenMP がある．OpenMP は，並列計算機環境において共有メモリマルチスレッド型の並列アプリケーションソフトウェア開発するための標準 API である．#pragma omp の指示句をコードに追加することで，その指示句に対応した処理が，OpenMP 解釈実行ツールを介して行われる．OpenMP 解釈実行ツールには gcc [27] 等がある．なお、#pragma の指示句で GPU の処理を行わせる仕様に OpenACC [28] が，OpenACC 解釈実行ツールに PGI コンパイラ [29] 等がある．

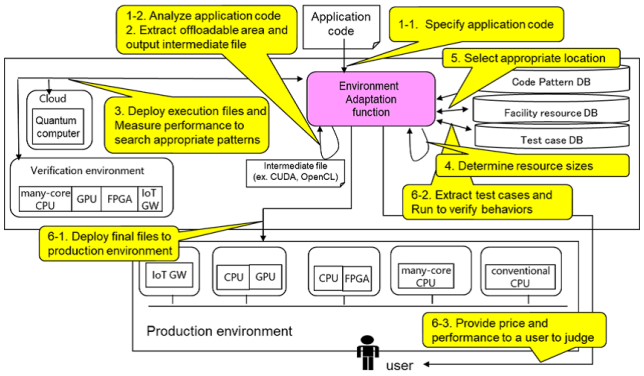


図 1　環境適応ソフトウェアの処理概要

OpenACC，OpenCL，OpenMP 等を用いて，多様なハードウェアを用いた処理は可能になっている．しかし処理は行えても，高速化するにはまだ壁がある．例えば，メニーコア CPU を活用するために，Intel コンパイラ [30] がある．これは，for 文等のループ文の並列処理可能部を並列処理する．しかし，データが効率的に利用されない場合，ループ文を並列処理しても高速化されない．また，多様なハードウェアでは，ハードウェア特性を生かしたパイプライン処理等でアルゴリズムを考え効率的に処理することも多く，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 OpenACC や OpenCL や OpenMP の作成がされている．著者はループ文オフロード自動化をねらい，ループ文オフロードパターンを OpenACC や OpenMP として作り，ループ文のオフロード有無を遺伝的アルゴリズム [31] 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．

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

本稿の課題を示す．多様なハードウェアを用いたアプリケー

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，GPU や FPGA やメニーコア CPU 等への自動オフロードも要素技術として実現してきた．しかし，今までは個別の 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．本稿は，フーリエ変換等の計算処理の計算タイプに応じた，多様なハードウェア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ることで，アルゴリズム考慮して自動で高速化する．GPU の NVIDIA GeForce RTX 3090 搭載の実機で，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多様なハードウェアへのオフロード

本節では，計算タイプに応じた計算処理の多様なハードウェアへのオフロードを提案する．3.1 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．3.2 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．3.3 節では判定した計算処理に応じた実装置換による高速化を提案する．

3.1 中粒度計算の多様なハードウェアオフロードのアイデア

従来方式を用いて，GPU やメニーコア CPU へのオフロードで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 10 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，多様なハードウェアは多数コア等の特性を生かし，パイプライン処理等も駆使して高速化することが多いため，フーリエ変換等，計算タイプに応じて多様なハードウェア処理のアルゴリズムから考える，手動での高速化が主流だからである．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多様なハードウェア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，多様なハードウェア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 [32]．それが含まれている際に，その計算処理の多様なハードウェア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 3.2 節で，2 ステップ目を 3.3 節で説明する．

3.2 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, [33] 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 [34] 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，多様なハードウェアにオフロードできる計算タイプ（フーリエ変換等）のコード，その検索パターン，それを多様なハードウェアで処理する場合のコード（GPU なら OpenACC/CUDA，FPGA なら OpenCL，メニーコア CPU なら OpenMP）をコードパターン DB に保持しておく．この DB の情報は，多様なハードウェアオフロード高速化に用いられるので，多種の VM を提供するクラウド事業者が 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多様なハードウェアで処理する実装に関しては，他者論文等で検討され実装された OSS 物を用いる．

例えば [35] は，姫野ベンチマーク [36] 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，従来方式のループ文の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 に保持された多様なハードウェア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．

3.3 実装置換による高速化

多様なハードウェア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，検索された部分を多様なハードウェアで処理する場合のコード（GPU なら OpenACC/CUDA，FPGA なら OpenCL，メニーコア CPU なら OpenMP）に置換することで高速化する．コードは，フーリエ変換等で他者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 の実装に置換するため，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，実装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，実装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 OpenMP 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，OpenMP 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多様なハードウェア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多様なハードウェア自動オフロードの提案方式の有効性を評価する．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，本稿は GPU を例にする．

4.1 評 価 条 件

4.1.1 評価対象と評価手法

評価対象は，ユーザが GPU で利用すると想定されるフーリエ変換とする．

フーリエ変換処理 FFT（Fast Fourier Transform）は，振動周波数分析等で使われ IoT モニタリング等様々な場面で利用されている．NAS.FT [37] は，NASA が提供しているベンチマークで FFT 処理の OSS の一つで離散 3 次元 FFT の計算を行う．FFT 処理高速化のため，3 次元 FFT 処理も可能な CUDA ファイル cuFFT [38] 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の GPUU 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 処理と GPU オフロード時の処理時間を測定し，オフロード効果を見る．また，比較対象として，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に，ループ文のオフロードを GPU に対して既存手法で行った結果も比較する．

GPU に既存の遺伝的アルゴリズムを用いた手法でオフロードする際の条件は以下で行う．

ループ文数：NAS.FT　81

個体数 M：ループ文数以下（NAS.FT　50）

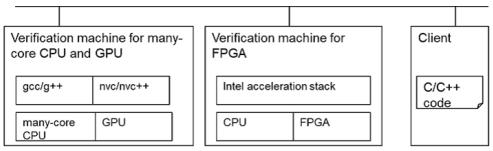
世代数 T：ループ文数以下（NAS.FT　50）

適合度：(処理時間)$^{-1/2}$　処理時間が短い程高適合度になる．また，(-1/2) 乗とすることで，処理時間が短い特定の個体の適合度が高くなり過ぎて，探索範囲が狭くなるのを防ぐ．

選択：ルーレット選択．ただし，世代での最高適合度遺伝子は交叉も突然変異もせず次世代に保存するエリート保存も合わせて行う．

交叉率 Pc：0.9

突然変異率 Pm：0.05



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++ 25.9
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 Acceleration Stack 2.0
Client	Lenovo ThinkPad X1 Carbon Gen10	Intel Core i7-1260P			32 GB	Windows 11 Pro	

図 2　性能測定環境

4.1.2 評 価 環 境

評価環境とスペックを図 2 に示す．GPU は NVIDIA GeForce RTX 3090，FPGA は Intel Stratix 10 GX，メニーコア CPU は 64 コアの AMD Ryzen 3995WX を用いる．制御は GPU は nvc/nvc+ 25 [39]，FPGA は Intel Acceleration Stack 2，メニーコア CPU は gcc/g++ 13 を用いる．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 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．

図 4 は NAS.FT 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 GPU 自動オフロードした場合の，CPU の処理時間，取得した CUDA ファイルの cuFFT に置換した GPU の処理時間，CPU に対する処理性能倍率を示している．また，[20] 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も示している．

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 7.13sec 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 [20] で GPU にした処理性能倍率は 2.54 倍であり，今回提案手法が性能倍率ではよいことが分かる．

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 [3]．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，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，フーリエ変換等の処理の計算タイプに応じ

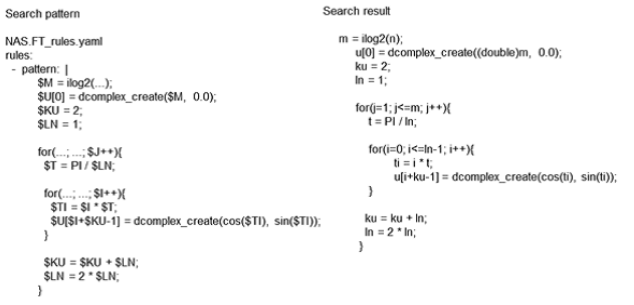


図 3　NAS.FT のパターンマッチング検索条件と検索結果

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

図 4　GPU オフロード後処理時間結果

て，適切な処理アルゴリズムで多様なハードウェア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，フーリエ変換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 OpenACC/CUDA や OpenCL や OpenMP 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，置換可能な実装がある計算タイプの計算処理を含むか検索できる．置換可能な実装がない場合は，以前検討の遺伝的アルゴリズム等を用いたループ文高速化の試行を行う．置換可能な実装が見つかった場合は，その実装に置換し，性能向上されるか性能測定を行う．価格的にも多様なハードウェア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．

今回検証では，フーリエ計算の NAS.FT を計算タイプの題材に，Semgrep で分析し，対応する CUDA ファイルの cuFFT に置換して性能測定し，GPU VM の価格的にもオフロード意味がある，119 倍以上の性能向上を確認し，方式有効性を示した．

文　献

[1] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.

[2] 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.

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

[4] A. Putnam, et al., "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.

[5] M. Hermann, et al., "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), May 2006.

[7] Y. Yamato and H. Sunaga, "Context-Aware Service Composition and Component Change-over Using Semantic Web Techniques," IEEE International Conference on Web Services (ICWS 2007), pp.687-694, July 2007.

[8] M. Takemoto, et al., "A Service-Composition and Service Emergence Framework for Ubiquitous Computing Environments," The 2004 Symposium on Applications and the Internet (SAINT 2004) Workshops, pp.313-318, Jan. 2004.

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

[10] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.

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

[12] 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.

[13] 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.

[14] 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.

[15] Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.

[16] Y. Yamato, "Study and evaluation of optimal placement of applications in mixed environments including quantum computers for environmental adaptation," IEEE Transactions on Cloud Computing, 2026.

[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, "Study and evaluation for adopting environmental adaptation of low-resource devices," IEEE Access, DOI: 10.1109/ACCESS.2024.3440918, Aug. 2024.
[27] gcc website, https://gcc.gnu.org/
[28] S. Wienke, et al., "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] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.
[31] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
[32] Clang website, http://llvm.org/
[33] Haskell description website, https://wiki.haskell.org/Declaration_vs._expression_style
[34] Semgrep website, https://github.com/semgrep
[35] I. Firmansyah, "OpenCL-based design methodologies for FPGA implementation," tsukuba repository, 2020.
[36] Himeno benchmark web site, http://accc.riken.jp/en/supercom/
[37] NAS.FT website, https://www.nas.nasa.gov/software/npb.html
[38] cuFFT website, https://developer.nvidia.com/cufft
[39] nvc/nvc++ website, https://developer.nvidia.com/hpc-sdk