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
ICM2026-18(2026-11)

ソフト工学技術での大粒度計算オフロードの提案

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，通常プログラマーが通常 CPU 向けに記述したコードを，動作させる環境に応じて，自動で変換等をして，高性能処理を行う環境適応ソフトウェアを提案した．本稿は，離散 3 次元フーリエ変換，MRI 画像処理等の計算処理を行う機能ブロックの自動オフロードを対象にする．オフロードしたい既存のアプリケーションを，機械学習系コード検知で分析しオフロードできるか検索し，抽象構文木系コード検知で分析しオフロード箇所を高速実装で処理させ，オフロードする．提案方式で自動オフロードできることを，NVIDIA GeForce 3090 の実 GPU 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，GPGPU，機能ブロック，類似コード検知

Proposal for offloading large-grained calculation using software engineering technology

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 code written by programmers for typical CPUs according to the operating environment to perform high-performance processing. This paper focuses on the automatic offloading of functional blocks that perform computational processing such as discrete 3D Fourier transform and MRI image processing. Existing applications to be offloaded are analyzed using machine learning-based code detection to search for offloadable parts, and then analyzed using abstract syntax tree-based code detection to process the offloaded parts with a high-speed implementation and perform the offloading. The ability to automatically offload using the proposed method is confirmed by measuring the processing time using an actual NVIDIA GeForce 3090 GPU.

Key words　Environment-Adaptive Software, Automatic Offloading, GPGPU, Functional Block, Code Clone Detection

1. はじめに

CPU（Central Processing Unit）の集積度が上がり，性能も上がっていくことを予想したムーアの法則は近年その進展が鈍化している．そのため，少コア CPU だけだけでなく，ヘテロジニアスなハードウェアを使ったアプリケーションが増えている．ヘテロジニアスなハードウェアには，GPU（Graphics Processing Unit），FPGA（Field Programmable Gate Array），メニーコア CPU 等のアクセラレータや少リソースの IoT 機器等がある．Amazon 社はヘテロジニアスなハードウェアをクラウド技術（例えば，[1]- [4]）で提供しているし [5]，Microsoft 社は FPGA 用いた Bing 検索を報告している [6]．また，IoT 技術を用いた IoT 機器でのサービスも増えている（例えば，[7]- [14]）．

しかし，ヘテロジニアスなハードウェアの能力を十分発揮するためには，ハードウェアの性質を十分理解し，設定や製造をする必要があり，CUDA（Compute Unified Device Architecture）[15]，OpenCL（Open Computing Language）[16]，OpenMP（Open Multi-Processing）[17] 等の，技術を用いて動作させることが基本となる．そのため，大半の技術者には，難度が高いのが現状である．

現在生成 AI の電力使用量が社会問題になっている．電力使用が適切なヘテロジニアスなハードウェアで処理することで，電力削減が期待できるが，OpenCL 等の前提知識から難度が高い．そこで，容易にヘテロジニアスなハードウェアを利用する

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2026 by IEICE

ため，少コア CPU 向けに製造された既存のソフトウェアコードを，GPU，FPGA，メニーコア CPU 等の配置ハードウェアに合わせて，変換して環境適応させる技術が必要である．ここで，ヘテロジニアスなハードウェアの利用には，CUDA，OpenCL 等を手動で新規に製造が主流である中，既存コードを自動で変換等行うことが，本研究の最大の特徴である．

私達は，そのために，既存コードを，GPU や FPGA 等の配置ハードウェアで動作するように自動変換し処理を高速化する，環境適応ソフトウェアを提案している．環境適応ソフトウェアを具現化するため，既存のコードのループ文や機能ブロックを GPU や FPGA にオフロードする手法を評価している [18]- [26]．しかし，これまでの私達は，遺伝的アルゴリズムを用いたループ文のオフロードや抽象構文木を用いた機能ブロックのオフロードを検証してきた．ループ文オフロードでは，計算に応じてハードウェアを意識したアルゴリズムを考えたライブラリでの高速化には及ばなく，また機能ブロックオフロードでは類似度が非常に高い場合でないと利用できない問題があった．

本稿は，フーリエ変換等の計算処理をする機能ブロックのヘテロジニアスなハードウェアへのオフロードを対象とする．オフロードする既存アプリケーションを，ソフトウェア工学の機械学習と抽象構文木のコード検知技術を用いて，オフロード可能なライブラリ等の実装がある機能ブロックか検知する．高速化ライブラリ等がない場合は，以前研究であるループ文高速化を施行する．高速化ライブラリ等の高速化実装が見つかった場合は，その実装で処理を行う．提案方式で，機能ブロックを検知しオフロードできることを，NVIDIA Geforce 3090 の GPU ボードを用いて確認し，処理時間を測定し，有効性を示す．

2. 既存技術

2.1 市中技術

GPGPU（General Purpose GPU）のための環境に NVIDIA は CUDA を提供している．CUDA は C 言語を拡張した GPU 向けの仕様だが，GPU，FPGA，メニーコア CPU 等のヘテロジニアスなハードウェアを制御する標準に OpenCL があり，GPU 等を提供する各社はコンパイラも提供している．CUDA，OpenCL は，C 言語拡張の記述だが，GPU 等のデバイスと CPU 等のホストとの間のデータコピー記述を明示的に行う等，ハードウェア特性の理解が必要であり，難度は高い．

CUDA や OpenCL のような低レベル記述は不要で，ヘテロジニアスなハードウェアを迅速に利用するため，指示行（Directive）を用いて処理を行う箇所を指定し，Directive に基づいて GPU 等で動作する実行ファイルを作成するコンパイラがある．記述標準仕様に OpenACC [27] や OpenMP 等，コンパイラに nvc [28] や gcc [29] 等がある．

CUDA, OpenCL, OpenACC, OpenMP 等の技術で，GPU，FPGA，メニーコア CPU への処理オフロードは可能になっている．しかしオフロードはできても，高速化は難度が高い．例えば，メニーコア CPU 並列化に，Intel コンパイラ [30] がある．Intel コンパイラは，for 文等のループで並列処理可能箇所を CPU の複数コアで並列に処理する．しかし，メモリデータ利用が非効率だと，単にループを並列処理しても高速化できない．GPU 等多コアのハードウェアで高速化する際には，CUDA 専門家等がチューニングしながら，コンパイラを用いて適切な処理箇所の探索等がされている．処理が適切なループの探索自動化として，私達は遺伝的アルゴリズム等の進化計算法を用いたオフロードを提案している [22]．

ループの GPU 等へのオフロードは上記方式が提案されている．しかし，上記方式は個々のループのオフロード可否を見るため，処理計算に応じて計算アルゴリズムから考えた手動実装のライブラリ等には性能が及ばなかった．そこで，計算処理全体をオフロードするため，抽象構文木を用いてオフロードできる機能ブロックがあるか検索し，検知できた場合はその処理を CUDA ライブラリ等の処理を呼び出す方式も提案されている [21]．方式評価では Deckard [32] という抽象構文木検知ツールを用いているが，類似度が非常に高い場合しか，上手く検知できない問題があった．

2.2 環境適応ソフトウェアの処理フロー

私達は環境適応ソフトウェアの 7 つの処理のフローを図 1 で提案している．環境適応ソフトウェアでは，クラウド等のサービス事業者が提供する環境適応機能が存在し，コードパターン DB，設備リソース DB，テストケース DB，商用環境，検証環境が連携して動作する．

Step 1　ユーザ提供コード分析：
Step 2　オフロード可能部分抽出：
Step 3　適切なオフロード部分探索：
Step 4　商用リソース量調整：
Step 5　商用デプロイ場所調整：
Step 6　実行ファイル商用配置と検証：
Step 7　商用運用中再構成：

環境適応ソフトウェアでは，Step 1-6 を通して，ユーザが指定するアプリケーションのコードを分析し，性能測定を検証環境で反復して行い，高速処理コードに変換等をし，利用するリソース量を決め，配置場所を決め，動作を確認後，ユーザの実利用が始まる．ユーザの実利用開始後に，Step 7 として，商用の実利用頻度を分析して，変換コードや配置場所等の構成を変えた方が効率が良い場合は再構成をユーザに提案して，了承後再構成を行う．

2.3 本稿対象の課題

本稿の対象課題を示す．ヘテロジニアスなハードウェアによるアプリケーション高速化は，高度な技術ノウハウを用いた手動製造が主流である．私達は，環境適応ソフトウェアを提案し，GPU 等へのオフロードを検証してきた．しかしこれまでは，遺伝的アルゴリズムを用いたループ文のオフロードや抽象構文木を用いた機能ブロックのオフロードの検証であり，ループ文オフロードでは，ハードウェア特性を生かしたライブラリでの高速化には及ばなく，機能ブロックオフロードでは類似度が非常に高い場合でないと利用が難しかった．本稿は，フーリエ変換等の機能ブロックのオフロードの改善を対象にする．既存アプリケーションから，機械学習と抽象構文木のコード検知技術

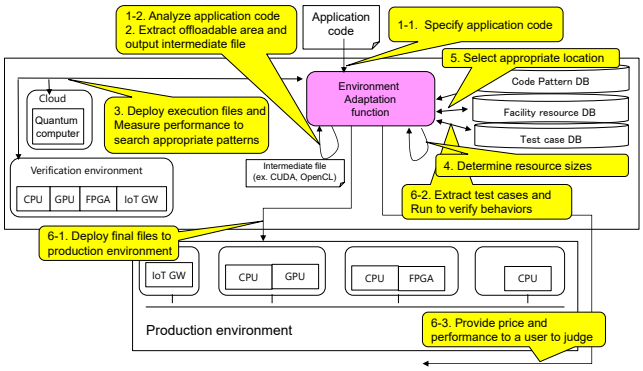


図 1　環境適応ソフトウェアの処理フロー

を用いて，オフロード可能なライブラリ等の実装がある機能ブロックか検知し，ある場合にそれを用いて処理する．提案方式で，機能ブロックを検知しオフロードできることを，NVIDIA Geforce 3090 の GPU ボードを用いて確認し，処理時間を測定し，有効性を示す．

3. 機能ブロックの自動オフロード

3.1 以前のループ文 GPU オフロード，機能ブロックオフロード

以前論文で提案の，ループ文 GPU オフロード方式 [22] と，多種ハードウェアへの機能ブロックオフロード方式 [21] をレビューする．

[22] では，GPU にオフロードする適切なループ文の発見を，Genetic Algorithms（GA）で自動的に行うことを提案している．並列化を想定していない汎用プログラムから，最初に並列可能ループ文のチェックをコンパイラ機能で行う．次に並列可能ループ文群に対して，GPU 実行の際を 1，CPU 実行の際を 0 と値を置いて遺伝子化し，検証環境で性能検証試行を反復し適切な領域を探索している．GPU で計算等の処理指定は，OpenACC ディレクティブで指定し，その実行は OpenACC コンパイラである nvc 等で行う．遺伝子の部分の形で，高速化可能な並列処理パターンを保持して次の世代の個体生成で組み換えていくことで，取り得る膨大な並列処理パターンから，効率的に高速化可能なパターンを探索している．

さらに，ループ文の適切な抽出に加えて，CPU-GPU 間の転送を減らすため，ネストループ文の中で利用される変数の転送を検討している．ループ文を GPU にオフロードする際に，ネストの下位で CPU-GPU 転送が行われると下位のループの度に転送が行われ効率的でないため，上位で CPU-GPU 転送が行われても問題ない変数については，上位でまとめて転送を行うことを提案している．処理時間がかかるループ回数の多いループは，ネストであることが多いため，転送回数削減による高速化に効果を示す．

この 2 つのアイデアを元に，ループ文が 150 を超える規模のアプリケーションでも数倍以上の自動高速化を確認している．しかし，計算のアルゴリズムからハードウェア性質に適したアルゴリズムで手動製造している場合に比べ，ループ文の単純なオフロードだけでは，性能が不十分なことが多かった．

機能ブロックオフロードは，ループ文の単純なオフロードでは性能が不十分なことから，計算全体である機能ブロックを高速処理実装に置換する方式で，オフロード先が GPU だけでなく，FPGA，メニーコア CPU でも同様である [21]．

まず，ソースコードの分析が行われるが，構文解析ツール（例えば，[31] 等）を用いて，ループ文構造等とともに，機能処理を把握する．把握した機能処理について，コードパターン DB と照合することで，GPU，FPGA，メニーコア CPU にオフロードできる高速実装がないか発見する．ここで，コードパターン DB マッチングには抽象構文木コード検知を用いている．オフロードできる処理について，GPU 向けライブラリや FPGA 向け IP コア等に置換して，CPU プログラムとのインタフェースを作成することでオフロードする．この際，オフロード処理が即高速化かやコスト効果が十分かは分からないので，検証環境での性能測定を通じて，オフロードするしないを試行することで，より適切となるオフロードパターンを抽出する．

ここで，抽象構文木コード検知として Deckard [32] を用いてマッチングしていたが，抽象構文木構造で類似コード検索は出来るが，オフロードのために高い類似度が前提となり，オフロード候補に発見できる機能ブロックが少なくなるという問題があった．

3.2 オフロードする機能ブロックの把握

コードを分析し，オフロードできる計算処理の機能ブロックかをコード検知技術で把握する．コード検知には，テキスト，トークン，抽象構文木，プログラム依存グラフ・意味，機械学習等の種類があり，DB 等の検索対象と入力クエリ等の検索条件をマッチングし，類似コードがないかを検知する．ここで，機械学習のコード検知ツールを用いる．機械学習のコード検知ツールには，UniXcorder [33] や NiCad [34] が利用できる．

コード検知に関連して事前に，オフロードできる計算処理の機能ブロック（MRI 系画像処理，離散 3 次元フーリエ変換等）のコード，それをオフロード先で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，オフロード高速化に用いられるので，ヘテロジニアスなハードウェアを提供するクラウド事業者等が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．機能ブロックの計算処理をハードウェアで処理する実装に関しては，別の方論文等で検討された無料の物を用いる．高速化検討された実装で，例えば，NVIDIA が提供している GPU 用の CUDA ライブラリに cuFFT [35] や cuRAND [36] 等がある．フーリエ変換を処理する cuFFT は GPU の多コアを生かして，Cooley-Tukey 分割された箇所を並列処理するが，FFT（Fast Fourier Transform）を GPU メモリ階層（Register - Shared - Global）にどう割り付けるかを最適化して，高速化している．

機械学習コード検知ツールで，オフロード可能な機能ブロックがあるか検索する．ここで，ない場合は，3.1 節のループ文のオフロード高速化の試行に移行する．ある場合は，改めて抽象構文木コード検知ツールで機能ブロックの計算処理を行う実装を検索する．抽象構文木のコード検知ツールには，Deckard

や CloneDR [37] が利用できる．

ステップ 1: 検索条件となるオフロードしたいアプリケーション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: 検索対象となる DB 側コードの抽象構文木化検索対象も検索条件と同様に，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条件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．抽象構文木部分木マッチングアルゴリズムで，検索条件抽象構文木を検索対象抽象構文木に対して走査し，部分木同型性を調べる．

機械学習系ツールと抽象構文木系ツールを組み合わせ，機械学習で低い類似度閾値等で多くの機能ブロックオフロード候補を発見できるようにし，抽象構文木でその機能ブロックの高速実装を発見して処理できるようにすることで，[21] 等であった，オフロード候補に発見できる機能ブロックが少なくなるという問題を解決する．

3.3 高速実装処理による高速化

GPU，FPGA，メニーコア CPU にオフロードできる計算処理の機能ブロックか判定できるため，発見された部分を GPU，FPGA，メニーコア CPU で処理する場合の実装に置換することで高速化する．実装は，GPU なら CUDA や OpenACC，FPGA なら OpenCL，メニーコア CPU なら OpenMP の実装である．実装は，MRI 系画像処理，離散 3 次元フーリエ変換等の計算処理機能ブロックで他論文等で手動改造で高速化が検討されてきた機能ブロックであり，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コード検知技術でマッチングし，オフロードできる計算処理をコードパターン 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，元のコードと実装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，実装で引数 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

個々のループ文の自動オフロードでなく，大粒度計算処理の機能ブロックの自動オフロードの提案方式有効性を，GPU を用いて評価する．

4.1 評 価 条 件

4.1.1 評 価 対 象

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．

フーリエ変換処理は，振動周波数の分析等，IoT でのモニタリングの様々な場面で利用されている．NASA の FT [38] は，FFT 処理し離散 3 次元 FFT を行う OSS（Open Source Software）の一つである．IoT で，デバイスから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，デバイス側で FFT 処理等の一次分析をして送ることは想定される．NASA の FT はグリッドサイズ 256*256*128 で，反復回数 6 回で処理を行う．コード検知に使う際は，旧版である v3.3 のコード [39] を新版である v3.4.1 のコード [38] にマッチングする．ユーザは旧版でアプリケーションを作りオフロードを依頼するが，事業者がコードパターン DB に準備しているのは新版等のケースを想定している．なお，新旧版をダウンロードした時間は 5 年以上の間隔があり，フーリエ変換処理は変わらないが実装コードはそれなりに異なる．FFT を GPU で高速化するため，CUDA ライブラリ cuFFT で自動処理することで高速化する．

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として v3.3 の FT を指定し，UniXcorder で v3.4.1 のコード検知がされ，更に Deckard でライブラリに処理が置換され，GPU 自動オフロードがされる．オフロードされた際は，コード検知を通じて得られた類似度の結果と，単コア 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，コード検知でオフロードできる機能ブロック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う．

GPU に GA を用いた既存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：FT 81

個体数 M：ループ文数以下（FT 30）

世代数 T：ループ文数以下（FT 30）

適合度：$(\text{処理時間})^{-1/2}$　処理時間が短い程高適合度になる．(-1/2) 乗で探索範囲狭くなるのを防ぐ．

選択：ルーレット選択．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

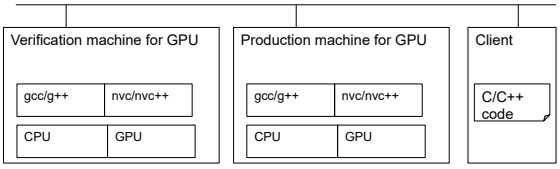
4.1.3 評 価 環 境

評価用 GPU として NVIDIA GeForce RTX 3090 を用いる．GPU 搭載機は，Iiyama LEVEL-R6W8-LCTP39-XAX (OS：Ubuntu 24.04.3 LTS，CPU：AMD Ryzen Treadripper PRO 3995WX，RAM：128GB DDR4-3200) である．GPU の制御は nvc/nvc++を用いる．C 言語プログラムを拡張し，GPU オフロード処理が CUDA でされる．また，比較でループ文 GPU オフロード処理も性能測定するが，nvc/nvc++で OpenACC を用いる．

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確定後，商用環境にデプロイされる．

4.2 結　　果

図 3 は，コード検知した際の，類似度，検索条件のファイル



Name	Hardware	CPU	GPU	RAM	OS	Tools
Verification machine for 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
Production machine for 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
Client	Lenovo ThinkPad X1 Carbon Gen10	Intel Core i7-1260P		32 GB	Windows 11 Pro	

図 2　性能測定環境

UniXcorder

```
3.7974 NAS.FT/ft.cffts1_neg.vec          NPB.FT/ft.fftz2.vec
3.7546 NAS.FT/ft.cffts1_neg.vec          NPB.FT/ft.evolve.vec
3.7949 NAS.FT/ft.cffts1_pos.vec          NPB.FT/ft.fftz2.vec
3.7503 NAS.FT/ft.cffts1_pos.vec          NPB.FT/ft.evolve.vec
3.7949 NAS.FT/ft.cffts2_neg.vec          NPB.FT/ft.fftz2.vec
3.7513 NAS.FT/ft.cffts2_neg.vec          NPB.FT/ft.evolve.vec
3.7910 NAS.FT/ft.cffts2_pos.vec          NPB.FT/ft.fftz2.vec
3.7954 NAS.FT/ft.cffts3_neg.vec          NPB.FT/ft.fftz2.vec
3.7585 NAS.FT/ft.cffts3_neg.vec          NPB.FT/ft.evolve.vec
3.8188 NAS.FT/ft.cffts3_pos.vec          NPB.FT/ft.fftz2.vec
3.7820 NAS.FT/ft.cffts3_pos.vec          NPB.FT/ft.init_ui.vec
3.7785 NAS.FT/ft.cffts3_pos.vec          NPB.FT/ft.evolve.vec
3.8950 NAS.FT/ft.checksum.vec            NPB.FT/ft.checksum.vec
3.7860 NAS.FT/ft.checksum.vec            NPB.FT/ft.fftz2.vec
3.7738 NAS.FT/ft.checksum.vec            NPB.FT/ft.compute_initial_conditions.vec
3.8939 NAS.FT/ft.compute_indexmap.vec    NPB.FT/ft.compute_indexmap.vec
3.7969 NAS.FT/ft.compute_indexmap.vec    NPB.FT/ft.fftz2.vec
3.7865 NAS.FT/ft.compute_indexmap.vec    NPB.FT/ft.compute_initial_conditions.vec
3.7726 NAS.FT/ft.compute_indexmap.vec    NPB.FT/ft.setup.vec
3.7705 NAS.FT/ft.compute_indexmap.vec    NPB.FT/ft.ftfft_init.vec
3.8772 NAS.FT/ft.compute_initial_conditions.vec NPB.FT/ft.compute_initial_conditions.vec
3.7901 NAS.FT/ft.compute_initial_conditions.vec NPB.FT/ft.compute_indexmap.vec
3.7638 NAS.FT/ft.compute_initial_conditions.vec NPB.FT/ft.fftz2.vec
3.7611 NAS.FT/ft.compute_initial_conditions.vec NPB.FT/ft.ftfft_init.vec
```

Deckard

```
0.971524 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:checksum
0.971174 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.cc:compute_initial_conditions
0.962914 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/common/c_randdp.c:randlc
0.962017 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/common/wtime.c:wtime
0.961760 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/common/c_randdp.c:vranlc
0.960315 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:init_ui
0.959246 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:cffts1
0.959246 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:cffts2
0.959246 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:cffts3
0.958902 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:evolve
0.954008 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:print_timers
0.950993 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:setup
0.948621 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:fftz
0.948031 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.cc:compute_indexmap
0.946777 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:ipow46
0.943705 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:ilog2
0.943210 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:ftfft_init
0.938425 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:fftz2
0.936872 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/common/c_print_results.c:c_print_results
0.923742 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:main
0.891532 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:verify
0.768427 NAS.FT/f/c_timers.c:timer_stop    NPB.FT/FT/ft.c:fft
0.952402 NAS.FT/ft.cc:cffts1_neg           NPB.FT/FT/ft.cc:cffts1
0.952402 NAS.FT/ft.cc:cffts1_neg           NPB.FT/FT/ft.cc:cffts2
```

図 3　FT のコード検知の UniXcorder，Deckard での出力結果

（NAS.FT と表記）と関数名，検索対象のファイル（NPB.FT と表記）と関数名の一部を，UniXcorder と Deckard で出力した結果を示しており，閾値より高い結果のみ表示されているが，全ファイルの 95%以上出力されており類似対象に問題ない場合は機能ブロックオフロードの対象となる．Deckard では処理対象の CUDA ライブラリの cuFFT も発見される．

図 4 は FT で，コード検知して機能ブロックオフロードした時の，1 コア CPU の処理時間，cuFFT で処理したに GPU 処理時間，1 コア CPU に対する処理性能倍率を示しており，以前手法で GA 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．

FT では，1 コア CPU の処理時間が 852 sec，cuFFT で処理した 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 GA 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，マルチコア CPU，FPGA，GPU のインスタンスを提供している．月額利用料は，通常 CPU は 60 USD/Month，マルチコア CPU は 140 USD/Month，FPGA は 700 USD/Month，GPU は 200 USD/Month 程度であり，インスタンスのスペックは一致ではないが 100 倍以上性能が出ているアプリケーションはコスト的にも高いプラス効果があると言える．

実験を通じて，コード検知により機能ブロックオフロードできる計算処理を見つけ，実際の GPU に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確認した．

5. ま と め

本稿では，環境適応ソフトウェアの拡張として，ユーザがオフロード依頼するアプリケーションを，ループ文以外も分析して，フーリエ変換等の計算処理を行う機能ブロックを発見し，適切なアルゴリズムで実装されたライブラリに自動オフロードする方式を提案した．

Application	CPU processing time	Proposed method processing time (Change searched CUDA files)	Proposed method improvement ratio	loop statements offloading improvement ratio using GA
FT (NASA Fourier Transform)	852 sec	7.13 sec	119	2.54

図 4　FT の GPU オフロード処理時間

まず，指定アプリケーションを分析する．機械学習コード検知ツール UniXcorder で分析し，フーリエ変換等の計算処理を行う機能ブロックを把握する．次に，抽象構文木コード検知ツール Deckard で分析し，フーリエ変換等の計算処理箇所とそれに該当するライブラリ等の実装を発見する．コード検知ツールを組み合わせ，機械学習 UniXcorder で，広い範囲で機能ブロックを把握し，抽象構文木 Deckard で，その機能ブロックの処理箇所と対応する実装を発見することで，Deckard だけだと類似度が非常に高い場合だけしか利用できなかったが，以前より広い範囲でオフロード機能ブロック候補を発見できる．実装が見つかった場合は，その実装を取得して置換して処理するようにし，性能測定を行う．

今回検証では，フーリエ変換を行う NASA の FT を，UniXcoder で発見し，該当箇所を Deckard で cuFFT に処理させることで，GPU に自動オフロードし，119 倍の性能向上を確認し，方式有効性を示した．

文　献

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.
[9] 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.
[10] 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.
[11] Y. Yamato, "Study and evaluation for adopting environmental adaptation of low-resource devices," IEEE Access, DOI: 10.1109/ACCESS.2024.3440918, Aug. 2024.
[12] M. Takemoto, T. Ohishi, T. Iwata, Y. Yamato, T. Tanaka, S. Tokumoto, N. Shimamoto, A. Kurokawa, H. Sunaga and K. Koyanagi, "Service-composition Method and Its Implementation in Service-provision Architecture for Ubiquitous Computing Environments," IPSJ Journal, Vol.46, No.2, pp.418-433, Feb. 2005.
[13] 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.
[14] Y. Yamato, Y. Tanaka and H. Sunaga, "Context-Aware Ubiquitous Service Composition Technology," The IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006), pp.51-61, Apr. 2006.
[15] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.
[16] 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.
[17] T. Sterling, M. Anderson and M. Brodowicz, "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.
[18] 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.
[19] Y. Yamato, "Study of software reconfiguration after adapted service start," 2023 5th International Electronics Communication Conference (IECC 2023), pp.63-68, July 2023.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[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, 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.
[26] Y. Yamato, "Study and evaluation of optimal placement of applications in mixed environments including quantum computers for environmental adaptation," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2026.3698110, May 2026.
[27] 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.
[28] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
[29] gcc website, https://gcc.gnu.org/
[30] 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.
[31] Clang website, http://llvm.org/
[32] Deckard website, https://github.com/skyhover/Deckard
[33] UniXcorder website, https://github.com/microsoft/CodeBERT/blob/master/UniXcoder/README.md
[34] NiCad website, https://github.com/bumper-app/nicad
[35] cuFFT website, https://developer.nvidia.com/cufft
[36] cuRAND website, https://developer.nvidia.com/curand
[37] CloneDR website, https://www.semanticdesigns.com/Products/Clone/CppCloneDR.html
[38] NASA parallel benchmarks v3.4.1 website, https://github.com/GMAP/NPB-CPP
[39] NASA parallel benchmarks v3.3 website, https://github.com/uhhpctools/openacc-npb