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.
SC2024 (2025-03)

量子計算機含む混在環境でのプログラム最適配置の提案

山登庸次†

† NTT，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，通常 CPU，マルチコア CPU，FPGA，GPU，量子コンピュータが存在する混在環境の中で，適切なハードウェアへの自動オフロードを対象とする．各ハードウェアで通常 CPU の何倍性能が出るか確認し，コストパフォーマンスもとに，オフロードに適切なハードウェアを定め自動オフロードする．提案方式で自動オフロードできることを，実ヘテロジニアスハードウェアを用いて，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，量子コンピュータ，混在環境，ハードウェア選択

A proposal for optimal program placement in a mixed environment including quantum computers

Yoji YAMATO†

† NTT Corporation, 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 normal CPUs according to the deployment environment, enabling high performance operation. This paper targets automatic offloading to appropriate hardware in a mixed environment that contains normal CPUs, multi-core CPUs, FPGAs, GPUs, and quantum computers. We confirm how much the performance of normal CPUs can be achieved with each hardware, and based on cost performance, we determine the appropriate hardware for offloading and perform automatic offloading. We confirm that the proposed method can automatically offload by measuring the processing time using actual heterogeneous hardware.

Key words　Environment Adaptive Software, Automatic Offloading, Quantum Computers, Mixed Environment, Hardware Selection

1. はじめに

CPU（Central Processing Unit）の経年高速化を期待したムーアの法則に合わない部分が出てきていると言われている．そのため，通常の CPU だけでなく，マルチコア CPU や，FPGA（Field Programmable Gate Array），GPU（Graphics Processing Unit）等のヘテロジニアスなハードウェアのアプリケーション利用が増えている．例えば，Microsoft 社は FPGA の検索利用を述べているし [1]，Amazon 社は，FPGA，GPU をクラウド（例えば, [2]- [5]）インスタンス提供している [6]．また，FPGA や GPU だけでなく，IoT 端末も多くなり（例えば, [7]- [15]），新たな型のハードウェアとして，DPU（Data Processing Unit）や量子コンピュータ等も出てきている．

しかし，ヘテロジニアスハードウェアで高速化するためには，ハードウェアの性質を意識した設定やコード作成が必要で，OpenCL（Open Computing Language）[16]，OpenMP（Open Multi-Processing）[17]，CUDA（Compute Unified Device Architecture）[18] 等で C 言語拡張を用いたプログラミングスキルが望まれる．そのため，PHP や Python 等のスクリプト言語利用が主流となった多くのプログラマーにとっては，スキルのハードルが高い．

FPGA や GPU，量子コンピュータ等のヘテロジニアスハードウェアシステムは，電力削減も可能であり，今後発展が期待されるが，それらの利用にはスキルのハードルが高い．そこで，ハードルを除き，ヘテロジニアスハードウェアを十分活用できるため，プログラマーが通常と同じ様に処理を記述したソフト

ウェアを，配置先の環境（FPGA，GPU，量子コンピュータ等）に応じて，変換や設定をし，環境に適応させる，新たなプラットフォームが重要になってくる．

そのため，私達は，既存ソフトウェアコードを，配置先環境を利用できるよう，FPGA や GPU 向け変換，リソース量設定等自動でし，アプリケーションを高速化する，環境適応ソフトウェアを提案した．環境適応ソフトウェア要素として，既存コードのループ文や機能ブロックを，FPGA，GPU，量子コンピュータ等に自動オフロードする方式等を評価している [19]- [27]．

しかし，私達検証はこれまで，FPGA，GPU，量子コンピュータ等個別のハードウェアにオフロードする事の自動化が主流であり，それらの混在環境への適切なオフロードは検討されていなかった．例えば，GPU オフロードで性能 10 倍，FPGA オフロードで性能 20 倍も，クラウドの月利用料は FPGA インスタンスが GPU インスタンスの 3 倍ならば，GPU オフロードが適切となりうる．

本稿は，通常 CPU，マルチコア CPU，FPGA，GPU，量子コンピュータが存在する混在環境の中で，適切なハードウェアへの自動オフロードを対象とする．まず，オフロードしたい既存のアプリケーションを分析して，オフロード可能部分を発見し，既存提案方式を用いて各ハードウェアにオフロード試行する．各ハードウェアで通常 CPU の何倍性能が出るか確認し，そのハードウェアのクラウド利用料から定まるコストパフォーマンスもとに，オフロードに適切なハードウェアを定め自動オフロードする．提案方式で自動オフロードできることを，AMD Ryzen，Intel Stratix，NVIDIA Geforce，Microsoft Azure Quantum [28] の実ヘテロジニアスハードウェアを用いて，処理時間を計測して確認する．

2. 既存技術

2.1 市中技術

GPGPU（General Purpose GPU）向けの開発に NVIDIA は CUDA を提案している．CUDA は GPU 対象だが，GPU，マルチコア CPU，FPGA 等のヘテロジニアスハードウェアを扱う仕様に OpenCL が提案されており，そのツールも各社が提供している l CUDA，OpenCL は，C 言語拡張のプログラム記述が必要で難しい（FPGA 等のカーネルと CPU のホストとの間のメモリ上データリリースやコピー記述を行う等）

CUDA や OpenCL と異なり，容易にヘテロジニアスハードウェアを利用すべく，指示行（Directive）ベースで，処理を行う部分を指定し，指示行に従って GPU 等に合わせたバイナリファイルを作成するコンパイラがある．仕様に，OpenACC [29] や OpenMP 等，コンパイラに PGI コンパイラ [30] や gcc 等がある．

CUDA，OpenCL，OpenACC，OpenMP 等用いて，GPU，マルチコア CPU，FPGA オフロードは可能になっている．しかし処理は行っても，高速化は難しい．例えば，マルチコア CPU 並列化に，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処理する．しかし，メモリ等で単にループ文並列処理しても高速化できない．GPU 等で高速化する際には，CUDA 専門家等がチューニング作業を行い，コンパイラ用いて適切な並列処理部探索等がされている．探索自動化として，著者は進化計算法を用いたオフロードを提案している．

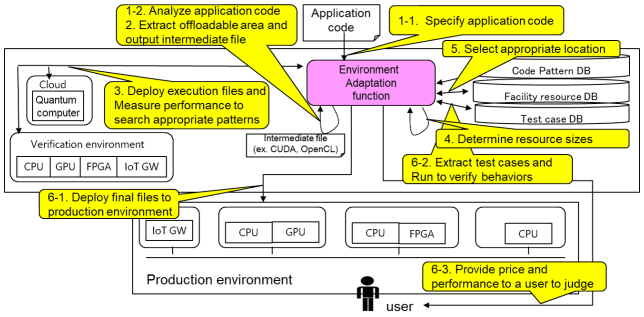


図 1　環境適応ソフトウェアの処理概要（[19]）

量子コンピュータは，量子力学を応用したコンピュータである．汎用性がある量子ゲート法での量子コンピュータの研究開発が増えており，Microsoft，Amazon，IBM，Google 等多くの企業が進めている．Azure Quantum や AWS Braket は，それぞれ Microsoft の Azure や Amazon の AWS クラウドの機能で提供されており，クラウドの時間利用が可能であり，量子コンピュータの初期費用は不要で利用できる．しかし，量子コンピュータを用いた素因数分解やナップサック問題の解法は，量子アルゴリズムを専門家が考案して，一つ一つ実現しているのが現状であり，一般ユーザが容易に利用できるものではない．

2.2 環境適応ソフトウェアの概要

私達は図 1 の処理概要の環境適応ソフトウェアを提案している [19]- [27]．事業者が提供する環境適応機能が中核に位置し，商用環境，検証環境，設備リソース DB，テストケース DB，コードパターン DB が連携することで，環境適応ソフトウェアは動作する．

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　商用配置場所調整：
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：

ユーザ提供コードを分析し，商用運用開始前に，Step1-6 で，検証環境性能測定試験通じてコード変換，リソース量調整，配置場所調整，動作検証を行う．Step7 は，商用運用開始後に，実際の利用データを分析して，構成変更した方が適切な事が明らかな場合に再構成を行う．

2.3 本稿の課題

課題を整理する．ヘテロジニアスハードウェア用いた高速化は専門家の手動が必要である．私は以前に環境適応ソフトウェアを提案し，FPGA や GPU，量子コンピュータの自動オフロード方式も実現してきた．しかし，今までは個別のハードウェアオフロード自動化が目的で，混在環境への適切なオフロードは十分検討されていなかった．例えば，GPU で性能 10

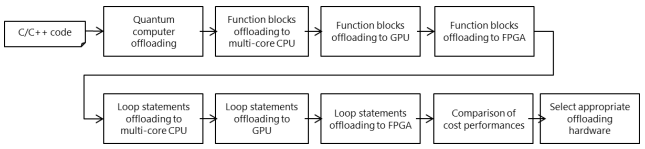


図 2　混在環境への適切な自動オフロード方式

倍，FPGA で性能 20 倍も，利用料で FPGA が GPU の 3 倍ならば，GPU が適切となる．そこで，本稿は，通常 CPU，マルチコア CPU，FPGA，GPU，量子コンピュータが存在する環境の中で，適切なハードウェアへの自動オフロードを対象とする．混在環境での適切な自動オフロード方式を提案し，実へテロジニアスハードウェア混在環境を用いて，提案方式を確認する．

3. 量子コンピュータ含む混在環境へのオフロード

個々ハードウェアへのオフロード試行は以前の方式で行う．まず，最初に Clang [32] 等の構文解析ライブラリでプログラム構造を把握した後，量子コンピュータオフロードを試行する．量子コンピュータオフロードでは，機能ブロックオフロードを行うので，量子コンピュータの後に，マルチコア CPU，GPU，FPGA で利用できるライブラリ相当が無いか検索し，見つかったら機能ブロックオフロードして性能測定する．機能ブロックオフロードの可否によらず，機能ブロックオフロード試行の後に，ループ文オフロードを，マルチコア CPU，GPU，FPGA の順番で試行し，進化計算の最終解で性能測定する．全パターンの性能測定結果が得られるので，各ハードウェアの利用料元に，コストパフォーマンスを求め，適切なハードウェアを選択する．

図 2 を用いて，提案方式の詳細を説明する．

3.1　機能ブロックオフロード試行

量子コンピュータ含めた機能ブロックオフロード試行を各ハードウェアに対して行う．各ハードウェアにオフロードできる機能ブロックが無いかは，Deckard [33] 等の類似性検出ツールで発見する．類似性検出ツールとは，コピー後変更したコードのような類似コードの検出を可能とするツールである．類似性検出には，抽象構文木，語彙，行，プログラム依存グラフ，フィンガープリント等の類似性を用いるが，抽象構文木を見るツールで解析する．

オフロードできる機能ブロックが発見された場合は，コードパターン DB に登録されている，そのコードに該当するハードウェアを利用するための処理部分（ライブラリ相当）に置換して，オフロードする．量子コンピュータ利用の場合は，量子アルゴリズムを元にしており以下の何れかを用いた実装となる．Qiskit [34] は IBM がオープンソース化した量子コンピュータフレームワークで，Cirq は Google がオープンソース化した量子コンピュータフレームワークで，Q#は Microsoft が提案する量子コンピュータ用プログラム言語である．量子コンピュータでなく，マルチコア CPU を利用する際はマルチコア CPU のライブラリを指定した OpenMP ファイル，GPU を利用する際は GPU のライブラリを指定した OpenACC ファイル，FPGA を利用する際は FPGA の IP コアとなる OpenCL ファイルに置換される．

オフロード機能ブロックが発見でき処理部分に置換できた際は，オフロードした場合の処理時間と，通常 CPU で実行した場合の処理時間を改めて測定し，オフロードした場合の性能改善度を取得する．

3.2　ループ文オフロード試行

ループ文オフロード試行を各ハードウェアに対して行う．ユーザアプリケーションを，構文解析ライブラリを用いて，for 文の構造を把握する．複数の for 文に対して，ハードウェアにオフロードするかしないかのパターンを作り，検証環境での反復性能測定を通じて高速なオフロードパターンを見つけるのは，全ハードウェア共通である．検証順序は変更も有りうるが，基本は，マルチコア CPU，GPU，FPGA の順とする．これは，特に FPGA はコンパイルから実機測定に長時間がかかるため，同程度の時間のマルチコア CPU，GPU オフロード試行である程度良い結果が出たら，スキップしても良いからである．

検証環境での試行では，マルチコア CPU，GPU は [19] 等の方式を元にして，進化計算手法の遺伝的アルゴリズムを用いて，for 文オフロード可否の多パターンを測定し高速なパターンを見つける．FPGA は [22] 等の方式を元にして，ROSE フレームワーク [35] で分析した算術強度やループ回数，リソース量で候補ループ文を絞り込んでから，検証環境での複数パターン性能測定を行い，高速なオフロードパターンを見つける．最終解は，マルチコア CPU の場合 OpenMP ファイル，GPU の場合 OpenACC ファイル，FPGA の場合 OpenCL ファイルである．

途中でも性能測定は繰返し行っているが，最終解でオフロードした場合の処理時間と，通常 CPU で実行した場合の処理時間を改めて測定し，オフロードした場合の性能改善度を取得する．

3.3　コストパフォーマンス元にしたハードウェア選択

置換できる処理が見つからずオフロードできない場合や，オフロードしても性能向上しない場合もあるが，機能ブロックを量子コンピュータ，マルチコア CPU，GPU，FPGA にオフロードしたパターンと，ループ文をマルチコア CPU，GPU，FPGA にオフロードした，最大 7 パターンが揃う．基本的に機能ブロックオフロードは専用の処理部分（ライブラリ相当）に置換するので，高速化度合は大きい事が多く，[25] で，CPU でのフーリエ変換処理を GPU のライブラリ cuFFT に置換した場合は 730 倍の性能が出ている．そのため機能ブロックオフロードできた場合は，処理部分に該当するハードウェア選択が有力な候補とはなる．一方，機能ブロックオフロードできず，ループ文オフロードのみの場合，どのハードウェアへオフロードが適切か決める必要がある．

そこで，ユーザ要望を考慮しコストパフォーマンス元に定める方法を提案する．オフロード 7 パターンに対して，通常 CPU に対する改善度が測定されている．ここで，量子コンピュータで処理の場合は，そもそも CPU では解けない問題が主となり，

通常 CPU ではタイムアウトするため，改善度は求めなくて良い．具体的には，その各オフロードパターンの改善度を，オフロードハードウェアの利用料（クラウドインスタンスの場合月額利用料等）で割り，コストパフォーマンスを求める．コストパフォーマンスが通常 CPU の場合よりも高く，全パターンで最高値の場合は，そのハードウェアが適切となるため，それをシステムとして選択しユーザ提案する．

例えば，通常 CPU の VM は 60USD/Month，GPU の VM は 200USD/Month，FPGA の VM は 700USD/Month の場合に，あるアプリケーションが GPU オフロードで通常 CPU の 10 倍，FPGA オフロードで通常 CPU の 20 倍性能が出た場合に，コストパフォーマンスを計算する．この場合 GPU オフロードが適切と定まるので，それをユーザに提案する．

4. 評　　価

ユーザが指定する 4 アプリケーションを量子コンピュータ，マルチコア CPU，GPU，FPGA の混在環境へ適材適所に自動オフロードができ，コストパフォーマンス良いハードウェアが選ばれていることを確認し方式の有効性を評価する．

4.1 評 価 条 件

4.1.1 評 価 対 象

アプリケーションは 4 つである．

評価対象は固有値解析問題である．高次元複素行列の場合，有限回の代数演算で固有値を正確に表現できる計算手順は存在しない．このため，固有値問題の数値解析には従来から反復法が用いられてきた．これは量子コンピュータの量子状態の重ね合わせ特性を利用することで高速化できる．問題の本質は一般的な n 次代数方程式の関連行列の固有値を求めることであり，各問題はパラメータの違いによって決まる．用いられる固有値解析問題は [36] である．[36] が示すように，量子コンピュータ上の固有値解析問題の解法も量子アルゴリズムを用いて Qiskit に実装されている．

ブロック対角ソルバ計算の NAS.BT [37] がある．ブロック対角ソルバは偏微分方程式の数値解法である．数値計算には様々な種類や実装があるが，ここではループ文が 100 を超える中規模の数値計算アプリケーションの例として NAS.BT(NASA ブロック三角ソルバ) [37] を使用する．パラメータは CLASS A 設定で，グリッドサイズは 64*64*64，反復回数は 200 回，時間ステップは 0.0008 である．

姫野ベンチマーク [38] は，非圧縮流体解析の性能測定に用いられるベンチマークソフトで，ポアッソン方程式解法をヤコビ反復法で解いている．アクセラレータでの手動高速化に頻繁に利用されており，提案の自動方式でも高速化できることの確認のため利用する．利用するデータサイズは，LARGE(512*256*256) である．

MRI-Q [39] 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 等多くの分野で，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向上のため，オフロードは重視される．MRI-Q は 3D MRI 画像処理を実行し，データサイズは，64*64*64 サイズのデータを使用して処理時間を測定する．

4.1.2 評 価 手 法

ユーザは 4 アプリケーションのオフロードを依頼する．事業者プラットフォームは依頼を受けたらアプリケーションを解析し，機能ブロックオフロード，ループ文オフロードを，量子コンピュータ，マルチコア CPU，GPU，FPGA に対して行う．

利用料：量子コンピュータは 90USD/hour，通常 CPU VM は 60USD/Month，マルチコア CPU VM は 140USD/Month，GPU VM は 200USD/Month, FPGA VM は 700USD/Month. 市中クラウドでの利用料を参考としている．

機能ブロックオフロード試行は以下で行う．

機能ブロック：固有値解析問題，NAS.BT，姫野ベンチマーク，MRI-Q

機能ブロック発見手法：コードに含まれる機能ブロックを類似性検出ツール Deckard を用いて，コードパターン DB に含まれるコードと照合．

ループ文オフロード試行は以下で行う．

マルチコア CPU，GPU では遺伝的アルゴリズム用いる．

オフロード対象とループ文数：固有値解析問題 0，NAS.BT 120，姫野ベンチマーク 13，MRI-Q 12.

個体数 M：ループ文数以下（NAS.BT 20，姫野ベンチマーク 10，MRI-Q 10）

世代数 T：ループ文数以下（NAS.BT 20，姫野ベンチマーク 10，MRI-Q 10）

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．また，性能測定が一定時間 3 分で終わらない際はタイムアウトさせ，処理時間∞として適合度計算する．

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

FPGA では絞り込みをしてから測定をする

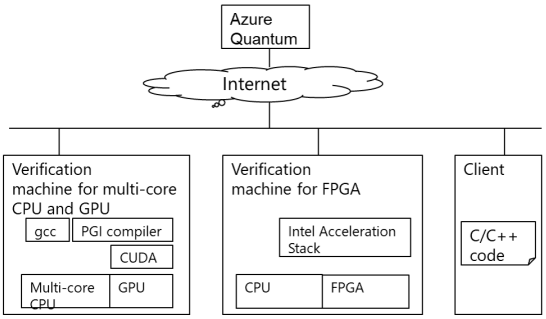
算術強度絞り込み：算術強度分析の上位 5 つのループ文に絞り込み

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（算術強度/リソース量が高い上位 3 つのループ文を選定）

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 つのループ文オフロードの組み合わせパターンで測定．

性能測定：サンプルパラメータそのままに，処理時間を測定する．

上記条件で，機能ブロックオフロード試行，ループ文オフロード試行を行い，オフロード処理した際の処理時間を測定し，4 アプリケーションの測定結果と利用料がユーザに提示される．



Name	Hardware	CPU	RAM	GPU/FPGA	OS	gcc	CUDA toolkit	PGI Compiler	Intel Acceleration Stack
Verification machine for multi-core CPU and GPU	LEVEL-F039-LCRT2W-XYVt	AMD Ryzen Threadripper 2990WX (32 Core)	32GB *2	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory:GDDR6 11GB)	Ubuntu 16.04.6 LTS	10	10.1	19.1	
Verification machine for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	Cent OS 7.9				2
Quantum computer	Azure Quantum								
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro				

図 3　性能測定環境

4.1.3　評 価 環 境

利用する量子コンピュータは Azure Quantum で，インターネット経由で, 量子コンピュータ用 Qiskit 実装を介して処理される．利用するマルチコア CPU は，AMD Ryzen Threadripper 2990WX (32 core) で，OpenMP 処理は gcc 10.1 を用いる．利用する GPU は NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory：GDDR6 11GB) で，OpenACC 処理は PGI コンパイラ 19.10 と CUDA Toolkit 10.1 を用いる．Ryzen と GeForce は同一ノードに搭載されている．FPGA は Intel PAC D5005 (Intel Stratix 10 GX FPGA) を用いる．コンパイルマシンは，DELL EMC PowerEdge R740 (CPU：Intel Xeon Bronze 3206R *2, RAM：32GB RDIMM * 4) で，Intel Acceleration Stack 2.0 で OpenCL をコンパイルする．評価環境とスペックを図 3 に示す．ここで，Client ノート PC から，ユーザが利用する C/C++言語アプリケーションを指定し，Verification machine が Deckard を用いて同一ノード上の DB と照合し検索する．処理時間は Verfication machine 等で測定され，適切なハードウェアが定められ，実際のユーザが使う商用環境に配置される．

4.2　結　　果

図 4 は，提案方式で 4 アプリケーションを混在環境にオフロードした際の，処理時間，性能改善度，通常 CPU VM を 1 とした時のコストパフォーマンスを示している．4 アプリケーションはコストパフォーマンスが高いハードウェアを自動選択している．

まず，固有値解析問題はそもそも通常 CPU では解けない事が多いが，量子コンピュータにオフロードすることで 15 秒程度で処理が終わっている．次に NAS.BT はループ文オフロード試行の結果，マルチコア CPU を適切なオフロード先に自動選択しており，通常 CPU の 5 倍以上の性能と 2.3 倍のコストパフォーマンスを示している．次に姫野ベンチマークはループ文オフロード試行の結果，GPU を適切なオフロード先に自動選択しており，通常 CPU の 20 倍以上の性能と 6.6 倍のコストパフォーマンスを示している．次に MRI-Q はループ文オフロード試行の結果，FPGA を適切なオフロード先に自動選択しており，通常 CPU の 11 倍以上の性能と 1.01 倍のコストパフォーマンスを示している（一般的に FPGA インスタンスは通常 CPU VM の 10 倍程度の利用料が多く，コストパフォーマンスはそれほど高くはなっていない）．

Applications	Offloading destination	Performance improvement	Processing time	Cost performance [/Offloading VM fee /normal CPU VM fee]
Eigenvalue solvers	Quantum computer		15.7 sec	
NAS.BT	multi-core CPU	5.39	24.1 sec	2.31
Himeno benchmark	GPU	22	1.87 sec	6.59
MRI-Q	FPGA	11.8	2.21 sec	1.01

図 4　4 アプリケーションの混在環境オフロード結果

オフロードするまでの時間は，機能ブロックオフロード試行は秒-分で終了するが，ループ文オフロード試行は時間が必要となる．マルチコア CPU や GPU の場合，遺伝的アルゴリズムでの測定数に依存するが，NAS.BT 等 for 文多く測定増える場合は，8 時間程度かかっている．また，FPGA の場合，OpenCL コンパイルして 1 回測定するまで 8 時間程度かかるため，測定数が増えると長時間となる．また，外部サービスとして，Azure を用いているため，Azure をサービス事業者でなくユーザ自身が契約する際は，審査等が必要なため，契約するまでに一定の時間と稼働が必要である．

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一部として，ユーザが提供するアプリケーションを，通常 CPU，マルチコア CPU，GPU，FPGA，量子コンピュータが混在する環境に，適切に自動オフロードする方式を提案した．

個別ハードウェアの自動オフロード試行自体は著者提案の以前方式を用いる．まず，ユーザアプリケーションを分析する．Deckard 等用いた構文木類似性元に，機能ブロックオフロード試行をマルチコア CPU，GPU，FPGA，量子コンピュータに対して行い，通常 CPU に対して何倍高速になったかを各ハードウェアに対し取得する．次に，機能ブロックオフロードの可否によらず，進化計算用いたループ文オフロード試行をマルチコア CPU，GPU，FPGA に対して行う．何倍かを各ハードウェアの月利用料で割ることで，コストパフォーマンスを求める．通常 CPU の場合よりも高く，最もコストパフォーマンスが良いハードウェアに対して，該当アプリケーションを自動オフロードする．

今回検証では，固有値解析問題，NAS.BT，姫野ベンチマーク，MRI-Q の 4 ユーザアプリケーションを，分析，性能測定して，それぞれ，量子コンピュータ，マルチコア CPU，GPU，FPGA に自動オフロードし，方式有効性を確認した．

今後は 2 つの方向性を予定している．一つはより多くの実用アプリケーションに対して混在環境オフロードを試行する．

もう一つは，自動オフロードは現在，構文木類似性見た機能ブロックオフロードと進化計算用いたループ文オフロードが主で粗粒度と細粒度の 2 つであるため，中粒度のオフロード，例えばステンシル計算ならこのオフロード等の形，を検討する．

文　献

[1] 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.

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

[3] Y. Yamato, "Automatic Verification for Plural Virtual Machines Patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, July 2015.

[4] Y. Yamato, "Use Case Study of HDD-SSD Hybrid Storage, Distributed Storage and HDD Storage on OpenStack," 19th International Database Engineering & Applications Symposium (IDEAS '15), pp.228-229, July 2015.

[5] Y. Yamato, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking Conference (CCNC 2015), pp.607-608, Jan. 2015.

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

[7] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[8] H. Noguchi, et al., "Autonomous Device Identification Architecture for Internet of Things," 2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018), pp.407-411, Feb. 2018.

[9] Y. Yamato, et al., "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.

[10] H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.

[11] H. Sunaga, Yet al., "Service Delivery Platform Architecture for the Next-Generation Network," ICIN 2008, Oct. 2008.

[12] Y. Nakano, et al., "Method of Creating Web Services from Web Applications," IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07), pp.65-71, June 2007.

[13] M. Takemoto, et al.,"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.

[14] Y. Yamato, et al., "Proposal of Shoplifting Prevention Service Using Image Analysis and ERP Check," IEEJ Transactions on Electrical and Electronic Engineering, Vol.12, Issue.S1, pp.141-145, June 2017.

[15] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), Nov. 2012.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] Y. Yamato, "Automatic Offloading Method of Loop Statements of Software to FPGA," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.

[23] 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.

[24] Y. Yamato, "Improvement Proposal of Automatic GPU Offloading Technology," The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.

[25] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," The 8th IIAE International Conference on Industrial Application Engineering 2020 (ICIAE 2020), pp.4-11, Mar. 2020.

[26] Y. Yamato, "Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.

[27] Y. Yamato, "Study and Evaluation of Improved Automatic GPU Offloading Method," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.

[28] Azure quantum website, https://azure.microsoft.com/products/quantum

[29] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

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

[31] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.

[32] Clang website, http://llvm.org/

[33] Deckard website, http://github.com/skyhover/Deckard

[34] Qiskit website, https://www.ibm.com/quantum/qiskit

[35] ROSE framework website, http://rosecompiler.org/ROSE_HTML_Refe

[36] Eigenvalue Solver website, https://learning.quantum.ibm.com/course/va algorithm-design/instances-and-extensions

[37] NAS.BT website, https://www.nas.nasa.gov/publications/npb.html

[38] Himeno benchmark web site, http://accc.riken.jp/en/supercom/

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