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
SIS2025-69(2026-03)

環境適応ソフトウェアにおけるメニーコアCPU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，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，ブロック行列計算，自明並列処理等の処理の計算タイプに応じた，メニーコア CPU 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 OpenMP がある計算タイプか把握する．OpenMP が見つかった場合は，その OpenMP 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，AMD Ryzen Threadripper 3995WX の 64 コア CPU 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，メニーコア CPU，計算タイプ，パターンマッチング

Proposal for expanding many-core CPU 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 propose environment-adaptive software that automatically converts and configures software code written by ordinary programmers for small-core CPUs according to the deployment environment, resulting in high-performance processing. This paper focuses on automatic offloading to many-core CPUs according to the computation type of processing, such as block matrix calculations and trivially parallel processing. We semantically analyze the existing application to be offloaded using an abstract syntax tree with pattern matching to determine whether the computation type has a replaceable OpenMP. If OpenMP is found, we replace it with that OpenMP and confirm the performance improvement. We confirm the effectiveness of the proposed method for automatic offloading using an AMD Ryzen Threadripper 3995WX 64-core CPU.

Key words　Environment-Adaptive Software, Automatic Offloading, Many-Core CPU, 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，30 コア以上のメニーコア CPU や FPGA（Field Programmable Gate Array），GPU（Graphics Processing Unit），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，マルチコア CPU, FPGA, GPU 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，メニーコア CPU では OpenMP（Open Multi-Processing）[9], FPGA では OpenCL（Open Computing Language）[10]，GPU では CUDA（Compute Unified Device Architecture）[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．メニーコア CPU は電力自体は GPU と同程度であるため，同程度の性能のオフロードは意味がないが，メニーコア CPU の方が GPU よりも高速で計算が早く終わる場合は，メニーコア CPU へのオフロードは電力削減でも意味がある．しかし，現在メニーコア CPU で高性能処理するには OpenMP 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（メニーコア CPU，FPGA，GPU 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，FPGA 向けや GPU 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，メニーコア CPU，FPGA，GPU 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，メニーコア CPU 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 OpenMP を作成した高速化には及ばなかった．本稿は，ブロック行列計算，自明並列処理等の計算タイプに応じた，メニーコア CPU 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 OpenMP がある計算タイプかを把握する．置換可能な OpenMP 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 OpenMP が見つかった場合は，その高速化 OpenMP 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，AMD Ryzen Threadripper 3995WX [27] の 64 コアメニーコア CPU 機を用いて，処理時間を計測して確認する．

2. 既存技術

2.1 市中技術

メニーコア CPU，FPGA，GPU 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）．また，OpenCL の改善に SYCL [28] があり，SYCL 解釈実行ツールに DPC++ [29] 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 [30] 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 [31] が，OpenACC 解釈実行ツールに PGI コンパイラ [32]（現在 nvc）がある．

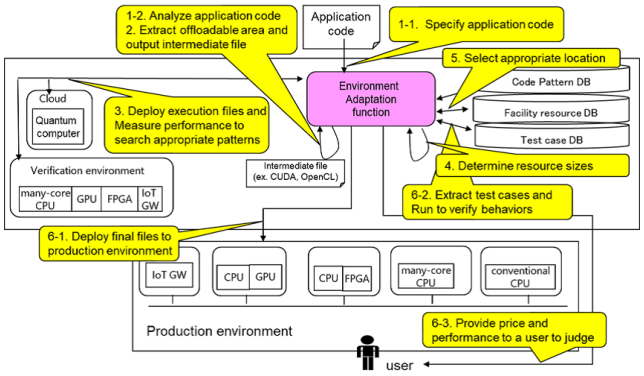


図 1　環境適応ソフトウェアの処理概要

OpenCL，OpenMP 等を用いて，メニーコア CPU 等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 [33] 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，メニーコア CPU 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あり，OpenCL や OpenMP の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 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 OpenMP 化し，ループ文のオフロード有無を遺伝的アルゴリズムにより，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，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，メニーコア CPU や FPGA や GPU 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作成 OpenMP での高速化には及ばなかった．本稿は，ブロック行列計算，自明並列処理等の計算処理の計算タイプに応じた，メニーコア CPU 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．実メニーコア CPU の AMD Ryzen Threadripper PRO 3995WX の 64 コア機で，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メニーコア CPU へのオフロード

3.1 ループ文のメニーコア CPU への自動オフロード方式

メニーコア CPU は，多数の計算コアを生かして，処理を並列化することで高速化を行う．GPU と異なる点は，メニーコア CPU の場合，メモリは共通であるため，GPU へのオフロードでしばしば問題となった CPU と GPU のメモリ間のデータ転送によるオーバヘッドは考慮する必要がない．また，メニーコア CPU でのプログラム処理の並列化には，OpenMP 仕様を用いる．OpenMP は，#pragma omp parallel for 等の指示句で for 文に対して，並列処理等を指定する．OpenMP での処理並列化は，OpenMP プログラマーが責任を持つこととなっており，並列化できない処理を並列化した場合には，コンパイラがエラー出力をするわけでなく，計算結果が誤って出力される．

これらを踏まえ，ループ文のメニーコア CPU 向け自動オフロードは，ループの並列処理可否を，OpenMP の#pragma で指定するパターンを複数作成し，検証環境で実際の性能測定を繰り返すことで，徐々に高速化していく進化計算手法のアプローチをとる．ここで，GPU 自動オフロードで用いていた PGI コンパイラは，並列化不能時はコンパイラがエラーを出力していた．しかし，gcc 等の OpenMP コンパイラはそのようなエラーはプログラマーの責任となる．そこで，自動化するために，OpenMP 指示句で行う処理は，ループ文をメニーコア CPU で並列処理するかどうかだけと単純化するとともに，並列処理した場合の最終計算結果が正しいかどうかのチェックも性能測定時にセットで行うことで，正しい計算結果が出るパターンだけ進化計算の中で残る仕組みとする．

具体的には，コードが入力されたら Clang [35] 等で構文解析を行い，ループ文を判定する．ループ文に対して，#pragma omp parallel for の追加により，並列処理を指定した OpenMP コードを作成する．ここで，メニーコア CPU で並列処理の場合は 1，並列処理しない場合は 0 として，遺伝子パターンとする．準備した複数のパターンを gcc 等の OpenMP コンパイラでコンパイルし，メニーコア CPU を備えた検証環境マシンで性能測定をする．性能測定の結果高速処理のパターンを高い適応度に，低速処理のパターンを低い適応度に設定し，次世代のパターンを遺伝的アルゴリズム [34] のエリート選択，交叉，突然変異等の処理により作成する．ここで，性能測定の際に，最終計算結果が並列処理しない場合と同じ結果であることを，オリジナルコードを少コア CPU で処理した場合と比較し，もし差分が許容できない程大きい場合はそのパターンの適応度は 0 として次世代には選ばれないようにする．

3.2 中粒度計算のメニーコア CPU オフロードのアイデア

3.1 節方式を用いて，メニーコア CPU 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，メニーコア CPU は多数コア処理の特性を生かし，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，ブロック行列計算，自明並列処理等，計算タイプに応じてメニーコア CPU 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メニーコア CPU 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，メニーコア CPU 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，その計算処理のメニーコア CPU 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, [36] 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 [37] 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，メニーコア CPU にオフロードできる計算タイプ（ブロック行列計算，自明並列処理計算等）のコード，その検索パターン，それをメニーコア CPU で処理する場合の OpenMP 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，メニーコア CPU オフロード高速化に用いられるので，メニーコア CPU の VM を提供するクラウド事業者がメニーコア CPU 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メニーコア CPU で処理する OpenMP に関しては，別の方論文等で検討され実装された OSS 物を用いる．

高速化検討された OpenMP で，例えば，NASA が提供している OSS の OpenMP がある．また，多くの既存研究でも，OpenMP 使ったメニーコア CPU での計算タイプに応じた高速化検討がされている．

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 節のループ文のメニーコア CPU 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 に保持されたメニーコア CPU にオフロードできる計算タイプの計算処理を含むか，コードかの判定ができる．

3.4 OpenMP 置換による高速化

メニーコア CPU にオフロードできる計算タイプを含むコードか判定できるため，検索された部分をメニーコア CPU で処理する場合の OpenMP のコードに置換することで高速化する．OpenMP のコードは，ブロック行列計算，自明並列処理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 OpenMP 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，OpenMP 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，OpenMP 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

4.1 評 価 条 件

4.1.1 評 価 対 象

評価対象は，ユーザがメニーコア CPU で利用すると想定されるブロック行列計算と自明並列計算とする．なお，私達は環境適応ソフトウェアを提案し，ループ文オフロードに関して，メニーコア CPU，FPGA，GPU の混在環境で最高速にできるオフロード先を検証して選ぶ技術 [24] を実現しており，ブロック行列計算や自明並列計算はメニーコア CPU が最高速である．単純なフーリエ変換，単純な行列計算等は GPU の方が高速であり，電力消費量も同程度なメニーコア CPU にオフロードする意味はないと言える．

ブロック三重対角行列計算とは，係数行列がブロック三重対角構造を持つ連立一次方程式を高速に解くための数値計算法である．ブロック版 Thomas 法（Block Thomas algorithm）が使われる．小行列を成分とする三重対角構造の連立一次方程式を，構造を利用して高速に解く方法であり，特定構造の活用が前提である．検証で用いるブロック三重対角行列計算 BT（Block Tri-Diagonal Solver）[38] のデータサイズは Class = B とする．

自明並列計算は，浮動小数点演算の性能の達成可能な上限，プロセッサ間通信をほぼ必要としない性能の推定値を提供する．様々な利用例があるが，本 EP は数値流体力学アプリケーションから派生したベンチマークで，流体力学で利用される．検証で用いる自明並列計算 EP（Embarrassingly Parallel）[39] のデータサイズは Class = B とする．

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メニーコア CPU 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，単コア CPU 処理とメニーコア CPU 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，メニーコア CPU と GPU に対して既存手法で行う．

メニーコア CPU，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：BT 120，EP 12

個体数 M：ループ文数以下（BT 60，EP 12）

世代数 T：ループ文数以下（BT 60，EP 12）

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．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．

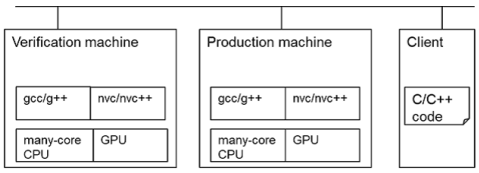
選択：ルーレット選択．ただし，世代での最高適合度遺伝子は交叉も突然変異もせず次世代に保存するエリート保存も合わせて行う．

交叉率 Pc：0.9

突然変異率 Pm：0.05

4.1.3 評 価 環 境

評価用メニーコア CPU として 64 コアの AMD Ryzen Treadripper PRO 3995WX を用いる．AMD Ryzen Treadripper PRO 3995WX 搭載機は，Iiyama LEVEL-R6W8-LCTP39-XAX (OS：Ubuntu 24.04.3 LTS，GPU：NVIDIA GeForce



Name	Hardware	CPU	RAM	GPU	OS	gcc/g++	nvc/nvc++
Verification machine for many-core CPU and GPU	LEVEL-R6W8-LCTP39-XAX	AMD Ryzen Threadripper PRO 3995WX (64 Core)	128 GB	NVIDIA GeForce RTX 3090	Ubuntu 24.04.3 LTS	13.3.0	25.9.0
Production machine for many-core CPU and GPU	LEVEL-R6W8-LCTP39-XAX	AMD Ryzen Threadripper PRO 3995WX (64 Core)	128 GB	NVIDIA GeForce RTX 3090	Ubuntu 24.04.3 LTS	13.3.0	25.9.0
Client	Lenovo ThinkPad X1 Carbon Gen10	Intel Core i7-1260P	32 GB		Windows 11 Pro		

図 2　性能測定環境

RTX 3090，RAM：128GB DDR4-3200) である．メニーコア CPU の制御は，gcc/g++を用いる．OpenMP の文法に従い，C 言語プログラムを拡張し，メニーコア CPU オフロード処理が OpenMP でされる．また，事前確認で GPU オフロード処理も性能測定するが，GPU の制御は nvc/nvc++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 EP 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 EP 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．BT 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 BT と EP 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メニーコア CPU 自動オフロードした場合の，1 コア CPU の処理時間，取得した OpenMP に置換した 64 コア CPU の処理時間，1 コア CPU に対する処理性能倍率を示しており，以前手法である遺伝的アルゴリズムを用いてループ文オフロードを 64 コア CPU にした処理性能倍率も示している．

BT では, 1 コア CPU の処理時間が 110sec, 取得した OpenMP に置換した 64 コア CPU の処理時間が 6.64sec で，処理性能倍率は 16.6 倍である．また，ループ文オフロードを 64 コア CPU にした処理性能倍率は 8.46 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．EP では，1 コア CPU の処理時間が 66.2sec，取得した OpenMP に置換した 64 コア CPU の処理時間が 2.08sec で，処理性能倍率は 31.8 倍である．また，ループ文オフロードを 64 コア CPU にした処理性能倍率は 16.6 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 VM を提供している [3]．月額利用料は，通常 CPU の VM は 60 USD/Month，マルチコア CPU の VM は 140 USD/Month，FPGA の VM は 700 USD/Month，GPU の VM は 200 USD/Month 程度であり，VM のコア数は違うが 15 倍以上性能が出ている 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．

Search pattern

```
EP rules.yaml
rules:
 - id: detect-EP
   pattern-either:
    - pattern: |     # pattern #1   [175]
       for (...) {
         ...
         for (i=1; i<=100; i++) {
           ...
           t3 = randlc(&t2, t2);
           ...
         }
         ...
         vranlc(2 * NK, &t1, A, x);
         ...
         for (i=0; i<NK; i++) {
           x1 = 2.0 * $X_0 - 1.0;
           x2 = 2.0 * $X_1 - 1.0;
           t1 = pow2(x1) + pow2(x2);
           if (t1 <= 1.0) {
             t2 = sqrt($CON_M2 * log(t1) / t1);
             t3 = (x1 * t2);
             t4 = (x2 * t2);
             l = max(fabs(t3), fabs(t4));
             q[l] += 1.0;
             sx = sx + t3;
             sy = sy + t4;
           }
         }
         ...
       }
   message: "The formula for EP has been found."
   languages: [c, cpp]
   severity: INFO
```

Search result

```
❱ detect-EP
   The formula for EP has been found.

      for(k=1; k<=np; k++){
        kk = k_offset + k;
        t1 = S;
        t2 = an;

        for(i=1; i<=100; i++){
            ik = kk / 2;
            if((2*ik)!=kk)t3=randlc(&t1,t2);
            if(ik==0){break;}
            t3=randlc(&t2,t2);
            kk=ik;
        }

        if(timers_enabled)(timer_start(2);)
        vranlc(2*NK, &t1, A, x);
        if(timers_enabled)(timer_stop(2);)

        if(timers_enabled)(timer_start(1);)

        for(i=0; i<NK; i++){
            x1 = 2.0 * x[2*i] - 1.0;
            x2 = 2.0 * x[2*i+1] - 1.0;
            t1 = pow2(x1) + pow2(x2);
            if(t1 <= 1.0){
                t2 = sqrt(-2.0 * log(t1) / t1);
                t3 = (x1 * t2);
                t4 = (x2 * t2);
                l = max(fabs(t3), fabs(t4));
                q[l] += 1.0;
                sx = sx + t3;
                sy = sy + t4;
            }
        }
        if(timers_enabled)(timer_stop(1);)
      }
```

図 3　EP のパターンマッチング検索条件と検索結果

Applications	Single core CPU processing time	Proposed method processing time (Change searched OpenMP)	Proposed method improvement ratio	Loop statements offloading improvement ratio
BT (Block Tri-Diagonal Solver	110 sec	6.64 sec	16.6	8.46
EP (Embarrassingly Parallel)	66.2 sec	2.08 sec	31.8	16.6

図 4　メニーコア CPU オフロード後処理時間結果

実験を通じて，パターンマッチングによりメニーコア CPU オフロードできる計算タイプの計算対象を見つけ，メニーコア CPU 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．

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，ブロック行列計算，自明並列計算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メニーコア CPU 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，ブロック行列計算，自明並列計算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 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 OpenMP がある計算タイプの計算処理を含むか検索できる．OpenMP がない場合は，以前検討の遺伝的アルゴリズムを用いたループ文高速化の試行を行う．置換可能な OpenMP が見つかった場合は，その OpenMP に置換し，性能向上されるか性能測定を行う．価格的にもメニーコア CPU 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，ブロック行列計算の BT，自明並列計算の EP を計算タイプの題材に，Semgrep で分析し，対応する OpenMP に置換して性能測定し，メニーコア CPU VM の価格的にもオフロード意味がある，15 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, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[6] Y. Yamato, et al., "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.

[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] 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.

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

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

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

[14] 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.

[15] 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.

[16] 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.

[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] AMD Ryzen website, https://www.amd.com/ja/products/processors/desktops/ryzen.html

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

[29] DPC++ web site, https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-library.html#gs.flx6xq

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

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

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

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

[34] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.

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

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

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

[38] Block Tri-Diagonal Solver website, https://www.nas.nasa.gov/software/npb.html

[39] Embarrassingly Parallel website, https://www.nas.nasa.gov/assets/nas/pdf/techreports/1994/rnr-94-007.pdf