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.

ユーザ機能追加のためのアプリケーション分割の提案

山登庸次†

† NTT 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　ヘテロジニアスなハードウェア利用を増やすため，私は一度記述したプログラムコードを，配置先環境を適切に利用できるように，変換等を自動で行い動作させる，環境適応ソフトウェアのコンセプトを提案してきた．本稿は，環境適応ソフトウェアの新たな要素として，汎用的プログラムでも，コードを分析して，ユーザが行いたい処理を追加変更できるようにする方式を提案する．提案方式はアプリケーションを関連する処理に基づいて分割して，分割境界を元に変更を局所化することで，サービス追加変更の容易化が可能になる．サンプルアプリケーションを提案方式で自動分割出来る事を確認する．

キーワード　環境適応ソフトウェア，汎用的プログラム，機能追加，自動分割，動的分析

A proposal for dividing the application to add user functions

Yoji YAMATO†

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

Abstract　To increase the use of heterogeneous hardware, we have proposed the concept of environment-adaptive software that automatically converts and operates program code once written so that it can be used appropriately in the environment in which it is placed. This paper proposes a new element of environment-adaptive software, a method that analyzes the code of general-purpose programs and allows users to add and change the processing they want to perform. The proposed method divides the application based on related processes and localizes changes based on the division boundaries, making it easier to add and change services. We check the sample applications can be automatically divided using the proposed method.

Key words　Environment-adaptive software, General-purpose programs, Function addition, Automatic division, Dynamic analysis

1. はじめに

AI（Artificial Intelligence）進歩も有り，少コアの CPU（Central Processing Unit）だけでなく，マルチコア CPU，GPU（Graphics Processing Unit），FPGA（Field Programmable Gate Array）や少リソースの IoT（Internet of Things）機器（[1]-[10] 等）等のヘテロジニアスなハードウェアが，多くのアプリケーションに用いられるようになっている．Microsoft 社は，FPGA を検索に用いているし [11]，Amazon 社は，クラウド技術等を使い（例えば [12]-[16]），GPU や FPGA インスタンスを提供している [17].

しかし，ヘテロジニアスなハードウェアを効率よく利用するためには，ハードウェア特性を意識したプログラムが必要となり，大半のソフトウェア技術者にとって壁が高い．マルチコア CPU では OpenMP（Open Multi-Processing）[18]，GPU では CUDA（Compute Unified Device Architecture）[19]，FPGA では OpenCL（Open Computing Language）[20]，IoT 機器では IoT PF（Platform）の知識が必要となってくることが多い．

ヘテロジニアスなハードウェアをより活用していくためには，高度な知識を持たない通常のソフトウェア技術者でも，それらを最大限に活用できるようにする PF が必要と考える．技術者が CPU だけしか使わない簡易プログラムと同様の手法で処理を記述したソフトウェアを，分析して，適用する環境（GPU，FPGA，IoT 機器等）にあわせて，適切に変換，設定を行い，環境に適応した動作をさせることを，PF が自動で行うことが今後求められていく．

そこで，私は，一度記述したプログラムコードを，配置先環境を適切に利用できるように，変換等を自動で行い動作させる，環境適応ソフトウェアのコンセプトを提案している．環境適応ソフトウェアの要素として，通常プログラムのコードを，GPU，

FPGA に自動オフロードする方式を提案している [21]- [26]．更に，アクセラレータだけでなく，少リソースの IoT 機器も適応対象とするため，ユーザは基本となるサービスと計算処理したいコードと IoT 機器を指定すれば，データ送信や蓄積等の共通処理を発見し，IoT GW（Gateway）と IoT PF に配置してサービス化する，IoT 適応方式を提案している．IoT 適応方式は，コード分析し IoT GW や IoT PF を適材適所で利用するが，IoT サービスに閉じるため，IoT サービス以外の汎用的プログラムを適材適所で利用することはできない．

これらの背景を踏まえ，本稿は，汎用的プログラムでも，IoT サービス同様にコードを分析して，ユーザが行いたい処理を追加変更できるようにする方式を提案する．アプリケーションを関連する処理に基づいて分割して，分割境界を元に変更を局所化することで，適材適所での利用とともに，サービス追加変更の容易化が可能になる．提案方式は，プログラムを動作させない静的状態と動作させる動的状態の両方での分析から，関数が含まれるファイル同士の呼び出し関係を把握し，関連が無いファイル群同士の分割を提案する．分割するファイル群と，元のファイル群の行数を比較し，ユーザが行いたい処理を追加する際の変更影響確認範囲が小さくなることを確認し，提案方式の有効性を示す．

2. 既存技術

2.1 市中技術と環境適応ソフトウェア

GPU を一般的計算にも用いる GPGPU（General Purpose GPU）（[27] 等）の環境に NVIDIA は CUDA を提供している．FPGA，GPU 等ヘテロジニアスなハードウェアを共通的に扱う仕様として OpenCL がある．OpenCL や CUDA は，C 言語拡張記述だが，ハードウェア知識が必要となる．容易に GPU 等を用いることができるようにするため，ディレクティブで GPU 処理等行う行を指定して，GPU 処理等のバイナリファイルを作成する取り組みがある．そのために，OpenMP や OpenACC [28] 等仕様と，それを解釈実行する gcc や PGI [29] 等コンパイラがある．

ヘテロジニアスなハードウェアを利用はできても，性能改善は容易でないのが現状である．Intel コンパイラ [30] 等は，並列処理可能なループを見つけ，複数のコアに処理を行わせているが，データコピー等により単に並列でループを処理しても性能が改善しないことも多い．マルチコア CPU でなく，GPU や FPGA の際はメモリも異なるためより複雑で，性能改善には OpenCL や CUDA を駆使した手動チューニングが必要となる．その中で，著者は進化計算手法である遺伝的アルゴリズム（GA）を用いた自動オフロードを提案している．

GPU 等のアクセラレータでなく，メモリ等のリソース量が少ない端末が多い IoT サービスも，ヘテロジニアスなハードウェアの利用である．IoT に関する標準としては，M2M プラットフォームの oneM2M [31] が標準化されている．IoT センサ等向けのプロトコルとして，HTTP より軽量な MQTT（Message Queuing Telemetry Transport）[32] が標準化されている．IoT センサからセンシングデータを IoT GW で集約し MQTT や

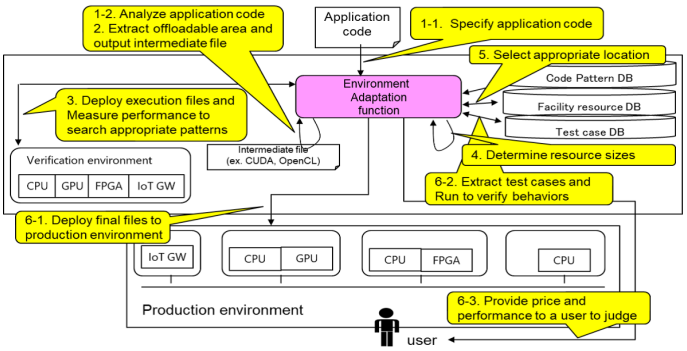


図 1　環境適応ソフトウェアの全体像

HTTP 等のプロトコルでクラウドに送り，データ集計や保持等共通的処理はクラウド上 IoT PF が行い，加工等処理を行って，その結果を企業幹部等のユーザにクラウドサーバで表示する形が多い．

IoT PF としては，大手クラウド事業者の Amazon や Microsoft が，AWS IoT や Azure IoT [33] 等の IoT PF をクラウドの中で提供しており，デファクトスタンダードに近くなっている．日本国内では，SORACOM が提供している IoT PF や，通信キャリアである NTT コミュニケーションズが Enterprise Cloud 上にシステム構築して，IoT PF 機能を提供等している．IoT GW としては，多くのセンサを集約し，標準プロトコルでデータをクラウドに送信することが大きな役割で，Intel 社の IoT GW やアットマークテクノ社の Armadillo IoT GW [34] 等があるが，IoT PF に比べ IoT GW は多くの社が提供しており，製品は極めて多い．

以前に著者は，環境適応ソフトウェアの処理として，図 1 の全体像を提案した．環境適応ソフトウェアの処理は，事業者が提供する環境適応機能を中心として，商用環境，検証環境，コードパターン DB，設備リソース DB，テストケース DB 等が連携して行われる．

Step1　コード分析：
Step2　オフロード可能部抽出：
Step3　適切なオフロード部探索：
Step4　リソース量調整：
Step5　配置場所調整：
Step6　実行ファイル配置と動作検証：
Step7　運用中再構成：

Step1-6 は，コード分析し，配置環境に応じたコード変換，リソース量調整，配置場所調整，検証の一連を行い，アプリケーション運用を開始する，Step7 は，アプリケーションの運用開始後に，実際の利用特性等を分析して，必要な再構成を行う．Step1-7 の流れを，GPU や FPGA 等のアクセラレータ，IoT 機器等の少リソース端末の両方で検証してきた．

2.2 本稿の課題の整理

本稿の課題を整理する．著者は環境適応ソフトウェアのコンセプトを以前提案し，GPU や FPGA 等のアクセラレータに通常プログラムを自動オフロードする方式や，ユーザは基本サービスとユーザが処理したい計算処理コードと IoT 機器を指定す

れば，共通的処理を検索して発見し，IoT GW と IoT PF に配置してサービス化する IoT 適応方式を検証してきた．IoT 適応方式では，コード分析し，IoT GW や IoT PF を適材適所で利用することができるが，自動で構築できるのは IoT サービスに閉じるため，IoT サービス以外の汎用的プログラムを適材適所で利用することはできない．

そこで，本稿では，IoT 適応方式の適用対象を拡大し，汎用的プログラムでも，IoT サービス同様に，分析してユーザが行いたい独自処理を追加変更できるようにする方式を対象とする．現在，保守により修正されてきたアプリケーションは，内部が複雑化して変更影響が広範囲に及ぶ構造になっていることが多く，追加変更に合わせたアプリケーション変更が大きな負担となっている．アプリケーションを関連する処理に基づいて分割して，分割境界を元に変更を局所化することで，適材適所利用とともに，サービス追加変更が容易化する．

勿論，ソフトウェアの追加変更等は，ソフトウェア工学で検討されてきた領域であり，それらの課題，成果を利用する必要があるが，環境適応という切り口で取組み，GPU や IoT 環境適応と同様の自動化を特徴に検討する．

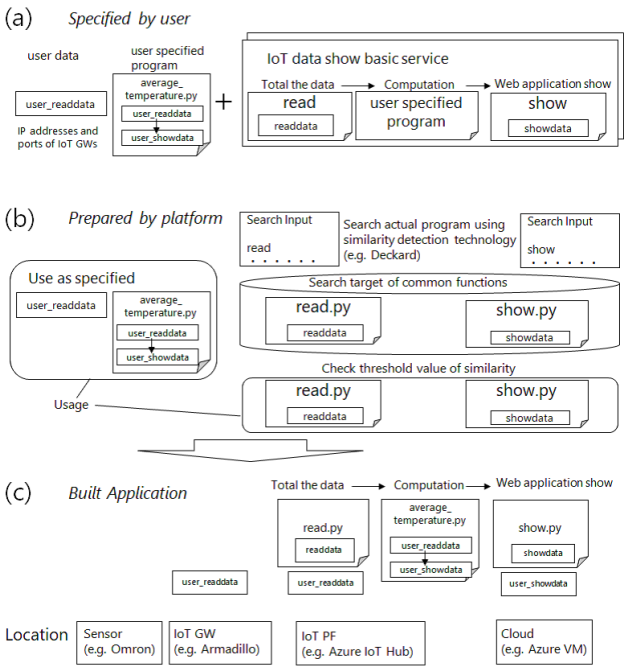


図 2　IoT 環境適応方式．(a) ユーザ指定，(b) プラットフォーム準備，(c) 生成アプリケーション．

3.　汎用的プログラムへの機能追加変更

著者は，環境適応ソフトウェアのコンセプトを具体化するために，これまでに，IoT 適応方式を検討してきた．これらの要素技術も参考にするため，3.1 では，IoT 適応方式についてレビューする，3.2 では，IoT 適応方式も発展応用して，汎用的プログラムの自動分割方式を検討する．3.3 では実装を述べる．

3.1　IoT 適応方式のレビュー

IoT PF や IoT GW に関する深い知識が無くても，IoT サービス化できることを目指し，ユーザは，基本となるサービスを選び，利用するユーザの IoT データとユーザ独自処理を指定すると，サービスに必要な共通的機能は類似機能ブロック検知技術により探して，ユーザ独自処理と組み合わせてサービス化する方式を提案した [?]．

図 2 を用いて，提案方式を説明する．事前に，サービス事業者は，複数の IoT サービスで用いることができる基本サービスと共通的機能を準備する．

(a) まず，ユーザは基本サービスを選ぶ．基本サービスには，データを集計し，ユーザ指定プログラムで加工し，結果を表示する等のサービスの基本動作が記載されている．ここで，ユーザは基本サービスをユーザ向けに使えるようにするため，利用する IoT データとユーザ指定プログラムを指定する．例えば，IoT データは user_readdata で定義される温度データ等であり，ユーザ指定プログラムは user_readdata の温度の平均値を計算し user_showdata に変換する等のユーザが行いたい処理である．

(b) ユーザに指定された情報を元に，サービス事業者が提供するプラットフォームの IoT 環境適応処理機能は，サービスに必要な機能を探す．データを集計する，結果を表示する等の基本サービスに記載された機能は，複数のサービスで利用できる共通的機能であるため，記載された動作に該当する実際のプログラムを，類似機能ブロック検知技術（Deckard [35] 等）を使って検索する．例えば，Azure IoT PF [33] でデータ集計や Web 表示を行う python プログラム等が見つかる．

(c) 発見された共通的機能とユーザ指定プログラムを基本サービスの記載（Main 関数相当）に沿って組み合わせる事でサービス化する．IoT GW から取得された user_readdata を，発見された共通的機能の read.py で集計し，ユーザ指定プログラムの average_temperature.py で平均を計算し，その結果を共通的機能の show.py で表示する．ここで，基本サービスには各処理の動作する場所も指定されており，各処理はその場所に配置される．

3.2　汎用的プログラムの自動分割方式

IoT サービス同様に，汎用的プログラムに，ユーザが行いたい独自処理を追加変更することを考える．それなりに行数があるアプリケーションは，変更影響が広範囲に及ぶことが多く，追加変更には関連する機能に影響がないかのチェックに大きな稼働がかかる．そこで，本稿では，アプリケーションを関連する処理で分割して，分割境界を元に変更を局所化することで，サービス追加変更を容易化する．

分析する手法で，アプリケーションを実際には動かさずに，ソースコードの関数の呼び出しや書き込み等のリレーションを見る，静的分析手法がある．静的分析では，まず，関数同士の呼び出し関係が把握できる．次に，ある関数が同じデータに書き込む関数かどうかも把握できる．一般に，呼び出し関係があったり，同じデータに書き込むでそれを使う関数は，関連が深いため，分割する際にはグループ化する必要がある．ただ，アプリケーションを動かさない静的分析は，ユーザには実際は使われないようなケースも分析される形となる．

一方，分析する手法で，サンプルテストケースを用いてアプリケーションを実際に動かし，実行されたログ等の情報を見る，動的分析手法がある．動的分析では，関数同士の呼び出し関係でも，ユーザが指定するサンプルテストケースで実際に使われる関数の呼び出し関係が抽出できる．さらに，データベース（DB）を用いるアプリケーションの場合，DB アクセスログを抽出して，連続的に実行しているデータアクセス命令を発見し，一連の処理として実行しているプログラム範囲を見つけることができる．また，DB でなくファイルアクセスの場合でも，ファイルアクセスログから，一連の処理として実行しているプログラム範囲を見つけることができる．ただ，動的分析で実際に動かすテストケースはサンプルの数に依存する形となる．

環境適応ソフトウェアでは今まで，GPU 自動オフロード等に取り組んできたが，GPU で計算処理可能かは静的分析で分かるが，GPU 処理した際の性能は実際に測定しないと分からないのが通常であり，静的分析と動的分析を組合わせて，オフロード部を自動探索していた．動的分析は，ユーザが使うサンプルテストケースを実際に動かし性能測定するため，個々のユーザ毎に異なる対応をするために重要な要素であった．そこで，本稿の汎用的プログラムの自動分割でも，個々のユーザに対応するために，静的分析と動的分析を組合わせた方式をとることにする．

提案方式の動作は図 3 のようになる．汎用的プログラムコード群を静的分析，動的分析し，呼出関係がある情報をもとに分割を行う．

ファイル A に関数 a1,a2,a3,,, ファイル B に関数 b1,b2,b3,,, が定義されているとする．

静的分析では，Main 関数から呼ばれた先の関数 a1 が b2 を呼んでいる場合に，AB が一回と加算する．それを全ファイルに対してカウントする．その結果，AB：20，AC：30，BA：10 等が分析結果になる．

動的分析では，サンプル試験を一定数実施した結果，Main 関数から呼ばれた先の関数 a1 が b2 を 10 回呼んでいる場合に，AB が 10 回と加算する．それをサンプル試験実施分だけカウントする．その結果，AB：10，AC：20 等が分析結果になる．

動的分析の結果，1 回でも呼び出しがあるファイル群は同じグループとする．静的分析の結果，3 回以上呼び出しがあるファイル群は同じグループとする．静的分析の結果，2 回以下呼び出ししかないファイル群は全て同じグループとする．

例えば，全ファイルとして，A,B,C,D,E があり，C → B,，B → A，E → D の呼出関係が動的分析であった際，A,B,C がグループ イ，D,E がグループ ロとなる．

このようにすることで，動的分析でユーザが指定するサンプルテストケースで呼び出し関係があるファイル群は必ず同じグループとなる．また，静的分析で全ファイルを分析した際に一定度の回数である 3 回以上呼び出し関係があるファイル群も同じグループになる．Stand Alone 的で他の関数とあまり関わりがない関数のファイルは全て残りのグループになる．呼び出し関係に基づいて自動分割することで，ユーザが機能追加する際の確認範囲を小さくすることができる．

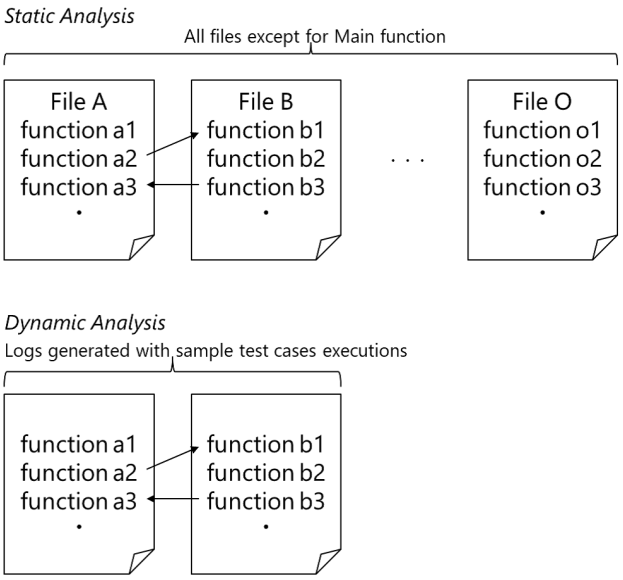


図 3　汎用的プログラム自動分割時の分析

3.3 実　　装

自動分割の検証対象は C 言語のアプリケーションとし，C 言語の解析には Clang [36] を用いる．プログラムの追跡等のデバッグは GDB（GNU Debugger）を用いる．C 言語アプリケーションを解析する実装は，Python 3 で行う．実装は，入力として，ソースコードファイル群と，サンプルテストケース情報を受け，出力として，静的分析の場合のグループ情報と，動的分析の場合のグループ情報を出力する．グループ情報は，イ（A,B,C），ロ（D,E）のような形で，分割されたグループとそれに所属するファイル群からなる．また，サンプルテストケースを実行する一定回数は 10 回とした．

4. 評　　価

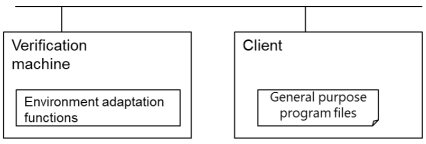
提案方式でサンプルアプリケーションが自動で分割出来る事，及び，提案方式での分割でユーザが機能追加した際の変更影響を確認する範囲が小さくなることを，プログラムコード行数で確認する．

4.1 評価対象と評価手法

評価する対象は，C 言語のアプリケーションでこれまでも環境適応の検証によく用いてきた，流体計算の姫野ベンチマーク，フーリエ変換の NAS.FT，深層学習フレームワークの Darknet とする．

姫野ベンチマーク [37] 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．GPU での手動高速化に頻繁に利用されており，著者の GPU 自動オフロード検証でも使われた．サンプルテストケースでは，データサイズ LARGE (512*256*256) のデータでサンプル試験を行う

フーリエ変換処理は，振動周波数の分析等，モニタリングの様々な場面で利用されている．NAS.FT [38] は，FFT 処理のオープンソースアプリケーションの一つである．センサ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



Name	Hardware	CPU	RAM	GPU	OS
Verification Machine	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX (32 Core)	64GB	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB)	Ubuntu 16.04.6 LTS
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro

図 4　評 価 環 境

トワークコストを下げるため，エッジ側で事前に行いたい FFT 処理を行ってからその結果だけ送ることは考えられる．サンプルテストケースは，NAS.FT に付属のサンプルテストケースを用いて，グリッドサイズは 256*256*128 で，イテレーション数は 6 で行う．

Darknet [39] は，C 言語のニューラルネットフレームワークであり，画像処理以外に，classification． detection 等様々な処理が出来る．画像処理の基本である，オブジェクト検知や画像加工が備わっており，それらにユーザ独自の機能追加等は想定される．サンプルテストケースは，Darknet に備え付けのサンプル処理で detection と呼ばれる検知と nightmare と呼ばれる画像加工を利用する．

3 アプリケーションともユーザ独自の機能追加も想定されるアプリケーションであり，本稿方式での自動分割の確認に利用する．自動分割では，3 節提案方式に従い実装したツールにアプリケーションのコードファイル群とサンプルテストケース情報を入力する．入力後分析が行われ，3 アプリケーションに対して，静的分析，動的分析の場合の分割グループが表示されるので，元々のアプリケーションの全行数と，分割されたアプリケーションの行数をカウントし比較する．

評価環境を図 4 に示す．本稿の環境適応は高速化するわけではないので，ツールはどのマシンで動かしても良いとは言えるが，静的分析，動的分析は長時間がかかるため，十分なスペックのマシンでの動作が必要である．

4.2　結果と考察

図 5 は，3 アプリケーションを提案方式で自動分割した際の，分割グループ数と，新機能を各アプリケーションに追加する先のグループの行数と，アプリケーション全体の行数を示している．

まず，姫野ベンチマークは分割されず，全体 166 行に対して，新機能追加グループは 166 行となっている．NAS.FT は 6 つに分割され，全体 1,547 行に対して，新機能追加グループは 723 行となっている．Darknet は 6 つに分割され，全体 24,834 行に対して，新機能追加グループは 2,366 行となっている．姫野ベンチマークはサイズが小さく分割されなかったが，NAS.FT と Darknet は自動で分割がされ，ユーザが独自処理を追加した際の確認する範囲が小さくなり，機能追加変更が容易になっていることがわかる．

著者の環境適応ソフトウェアは，これまで，既存のアプリケーションを GPU や FPGA へのアクセラレータに自動オフロードを行ったり，IoT 機器を用いたサービスを自動構築したりしてきた．そのため，利用できるアプリケーションは OSS で皆が利用できることが基本となり，大きな制約となっていた．今回，分析処理を発展させ，汎用的プログラムの自動分割をして，機能追加を容易にすることで，アプリケーションをまず自分用に機能追加してカスタマイズしてから，改めて GPU や FPGA 等のアクセラレータにオフロードすることも可能になった．そのため，環境適応の対象をより多くに増やし，利便性が大きく進展したと言える．

Application	Divided group #	total code #	target group code #
Himeno benchmark	1	166	166
NAS.FT	6	1,547	723
Darknet	6	24,834	2,366

図 5　提案方式での自動分割例

コストを考察する．汎用的プログラムに機能追加することを考えた際に，今回 C 言語の 3 アプリケーションをサンプルに利用した．姫野ベンチマークはサイズが小さく分割されなかったが，NAS.FT は機能追加する際に確認する行数が 1/2 以下になり，Darknet は 1/10 以下になった．影響確認稼働が低くなり，改造コストを抑えることが出来ると考える．

今回分析には，環境適応ソフトウェアの特徴であるユーザ毎の対応を行うために，従来も良く使われていた静的分析に，動的分析を組合わせる方式を提案した．分析自体は，関数の呼び出し関係を用いており，それで十分分割できることを確認したが，同じデータへの書き込み，DB やファイルのアクセスログ，連続実行等，更なる情報により，より詳細な分析が可能となる．

5. ま と め

本稿では，私が提案している，ソフトウェアを環境に自動適応させアプリケーションを運用する環境適応ソフトウェアの要素として，IoT 適応方式を発展応用し，汎用的プログラムでも容易に機能追加できるようにする，汎用的プログラムの自動分割方式を提案した．

提案方式では，汎用的プログラムを Clang 等で分析し，プログラムを動作させない静的状態と動作させる動的状態の両分析から，関数が含まれるファイル同士の呼び出し関係を把握し，呼び出し関係が無く関連が無いファイル群同士の分割を自動提案する．提案方式により，分割したファイル群の行数が，元のファイル群の行数に比べて小さくなり，ユーザが行いたい独自処理を追加する際の変更影響確認範囲が小さくなる．サンプルアプリケーションとして，姫野ベンチマーク，NAS.FT，Darknet の 3 つに提案方式を適用し，姫野ベンチマークは元が 166 行で分割できなかったが，NAS.FT は元が 1,547 行，分割後 723 行，Darknet は元が 24,384 行，分割後 2,366 行となり，機能追加する際の影響範囲が小さくなり，機能追加する際の確認が容易になることを確認した．

今後は，提案した汎用的プログラム自動分割方式を，検証し

た計算系アプリケーション以外の様々な種類でも有効性を確認する．特に，実際の IoT サービスにも適用し，IoT GW と IoT PF に処理を分割して，ユーザが容易に機能追加できることを確認する．

文　献

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

[2] 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.

[3] Y. Yamato, et al., "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, 2016.

[4] 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.

[5] Y. Yamato, et al., "Study of Service Processing Agent for Context-Aware Service Coordination," IEEE International Conference on Service Computing (SCC 2008), pp.275-282, July 2008.

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

[7] Y. Yokohata, et al., "Service Composition Architecture for Programmability and Flexibility in Ubiquitous Communication Networks," IEEE International Symposium on Applications and the Internet Workshops (SAINT 2006), pp.142-145, Jan. 2006.

[8] Y. Yokohata, et al., "Context-Aware Content-Provision Service for Shopping Malls Based on Ubiquitous Service-Oriented Network Framework and Authentication and Access Control Agent Framework," IEEE Consumer Communications and Networking Conference (CCNC 2006), pp.1330-1331, Jan. 2006.

[9] H. Noguchi, et al., "Device Identification Based on Communication Analysis for the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.

[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] 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.

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

[13] 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.

[14] Y. Yamato, "Server Selection, Configuration and Reconfiguration Technology for IaaS Cloud with Multiple Server Types," Journal of Network and Systems Management, Springer, DOI: 10.1007/s10922-017-9418-z, Aug. 2017.

[15] Y. Yamato, et al., "Evaluation of Agile Software Development Method for Carrier Cloud Service Platform Development," IEICE Transactions on Information and Systems, Vol.E97-D, No.11, pp.2959-2962, Nov. 2014.

[16] 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.

[17] Amazon Web Services EC2 web site, https://aws.amazon.com/ec2/instan
types/

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

[19] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, ISBN: 0131387685, 2010.

[20] 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.

[21] 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.

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

[24] 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.

[25] Y. Yamato, "Study of Parallel Processing Area Extraction and Data Transfer Number Reduction for Automatic GPU Offloading of IoT Applications," Journal of Intelligent Information Systems, Springer, DOI: 10.1007/s10844-019-00575-8, Aug. 2019.

[26] 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.

[27] J. Fung and M. Steve, "Computer vision signal processing on graphics processing units," 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp.93-96, 2004.

[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 (EWOMP'02), Sep. 2002.

[31] oneM2M web site, https://onem2m.org/technical/published-specifications/release-

[32] MQTT web site, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[33] Azure IoT Hub web site, https://learn.microsoft.com/ja-jp/azure/iot-hub/iot-concepts-and-iot-hub

[34] Armadillo web site, https://armadillo.atmark-techno.com/about/iot-gw

[35] Deckard web site, https://github.com/skyhover/Deckard

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

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

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

[39] Darknet website, https://pjreddie.com/darknet/