Vol.2013-ARC-207 No.35
Vol.2013-HPC-142 No.35
2013/12/17

HP3C (HPC Compiler in the Cloud)：最適化シナリオを機械学習で求めるコンパイラ

田中 美帆[1]　Trouvé Antoine[2]　Cruz Arnaldo[1,2]　福山 博識[1,2]　眞木 淳[2]　新井 正樹[3]　中平 直司[3]
山中 栄次[4]　村上 和彰[2,5]

概要：
本稿では，ユーザプログラムのコンパイル過程で，プログラム実行時の性能を向上させることを目的とした，HP3C（HPC Compiler in the Cloud）プロジェクトの概要を示す．HP3C は，コンパイラのバックエンドにおいて，ソースコードからバイナリを生成する際にベンダーコンパイラを用いる．これに対しコンパイラの上層部（フロントエンド）では，機械学習を用いることによってソースレベルでのコード変換を行い，ベンダーコンパイラのためのコンパイルオプションを決定する．機械学習により生成するモデルは，個々のユーザの実行プラットフォーム上でプログラムが最大の実行性能を得られるようにするために，対象プログラムとその実行プラットフォームに関する様々な特性を考慮している．また，機械学習を行うための訓練データを収集するために本システムをクラウド化し，世界中の多数のユーザから自動的にフィードバックを得られるようにする予定である．クラウド化により，一般ユーザは実行プラットフォームにベンダーコンパイラを用意するだけで，機械学習によるソースレベルでのコード変換とコンパイルオプションの機能を利用することができる．

キーワード：HPC, コンパイラ, 最適化空間探索, 機械学習, クラウド

HP3C (HPC Compiler in the Cloud): a Compilation System using Machine Learning Techniques for Optimization Scenarios Exploration

Tanaka Miho[1]　Trouvé Antoine[2]　Cruz Arnaldo[1,2]　Fukuyama Hiroki[1,2]　Maki Jun[2]
Arai Masaki[3]　Nakahira Tadashi[3]　Yamanaka Eiji[4]　Murakami Kazuaki[2,5]

Abstract: This paper introduces the HP3C (HPC Compiler in the Cloud) project. The HP3C project aims at improving the performance of user programs from the compilation flow. In the backend of the flow, it uses vendor compilers to generate binaries from source code as in traditional flows. Upstream the compiler however, it leverages machine learning in order to decide the options of the compiler, and to apply some source-to-source transformations. These models take into account various characteristics about the users' programs and environment in order to yield the best performance on their individual machines. In order to gather enough training data, the HP3C project will rely on a cloud service that automatically gathers the feedbacks from as many users as possible all around the world. This cloud service will also host all of the HP3C functionalities for code transformation and machine learning. In consequence, users will only need to install on their personal machines the vendor compiler.

Keywords: HPC, Compiler, Optimization Space Exploration, Machine Learning, Cloud

[1] 九州大学大学院 システム情報科学府, Kyushu University, Japan
[2] （公財）九州先端科学技術研究所
ISIT, Momochihama, Fukuoka 814–0001, Japan
[3] （株）富士通研究所, Japan
[4] 富士通（株）, Japan
[5] 国立大学法人九州大学 大学院システム情報科学研究院
情報知能工学部門, Kyushu University, Japan

Vol.2013-ARC-207 No.35
Vol.2013-HPC-142 No.35
2013/12/17

1. はじめに

コンパイル時にプログラムに対して施す最適化は，プログラム実行時の性能に大きな影響を与える．最適化を適切に施すことで実行時の性能を向上させることができるが，どのようなプログラム・実行プラットフォームに対しても有効な最適化手法が存在するわけではない．不適切な最適化手法を用いると，プログラム実行時の性能が何も施さない場合よりも低下する場合があるためである．そのため，個々のプログラムに対して最も適切な最適化手法を決定するためには，コンパイラがプログラムに対して最適化を施す際に，それぞれの最適化手法の効用，つまり最適化後のプログラム実行時の性能向上の度合いを知る必要がある．

しかし，プログラムに対する最も適切な最適化手法を知るために，一連の最適化手法を全通り施して実行時間を測定することは現実的ではない．また，プログラム実行時の性能に影響を与える要因は，プログラムの静的・動的な特徴，ベンダーコンパイラ，プログラムの実行プラットフォーム，プログラムに対する入力値・データなど，多様で複雑である．このように最適化後のプログラムの性能を事前に知ることは難しい．既存手法では，プログラムそのものの特徴を考慮していないことや，予測を行うモデルが単純であることから，複雑な性能決定要因を踏まえた精度の良い性能の推測をすることが難しい [1].

これに対し我々は，プログラム実行時の性能を機械学習を用いることによって推測するシステムを開発している．機械学習を用いてプログラムの多数の性能決定要因を考慮したモデルを生成することで，精度よく実行時の性能を予測し，最も適切な一連の最適化手法とコンパイルオプションを決定できるようになる．

しかし予測精度の高いモデルを生成するためには，質の良い訓練データが必要となる．質の良い訓練データとは，先述の多様な性能決定要因を網羅し，未知のケースに対しても正しく実行性能を予測できるようになるモデルを生成するためのデータである．このような多様な訓練データを収集するために，筆者らは本システムをクラウド上に構築し，HP3C（HPC Compiler in the Cloud）と名付けた．HP3C は，ユーザ側のベンダーコンパイラと連携し，機械学習を用いてコンパイラの最適化空間探索を行うシステムである．本稿では，HP3C の概要を報告する．

本稿の構成は以下のとおりである．まず 2 章で HP3C 開発の背景について述べた後，3 章で HP3C の概要について，4 章でクラウド上でのシステムアーキテクチャについて示す．最後に 5 章でまとめとする．

2. 開発背景

2.1 プログラムの最適化と実行性能

図 1 は，テンソル縮約を行うプログラムに対して異なる手順でベクトル化を施した場合の性能の変化を，昇順に並べたものである．横軸は同一プログラムに対して施した，それぞれ異なる手順の最適化手法の ID，縦軸はそれぞれのプログラムの性能の変化を表している．図中の Baseline は，コンパイラ（ICC）によってベクトル化を行わない標準設定でコンパイルを行った場合の性能を表しており，マニュアルによれば，このオプションによる設定は “icc -O2” に近いとされている．それぞれのプログラムの性能はこの Baseline によって正規化されている．

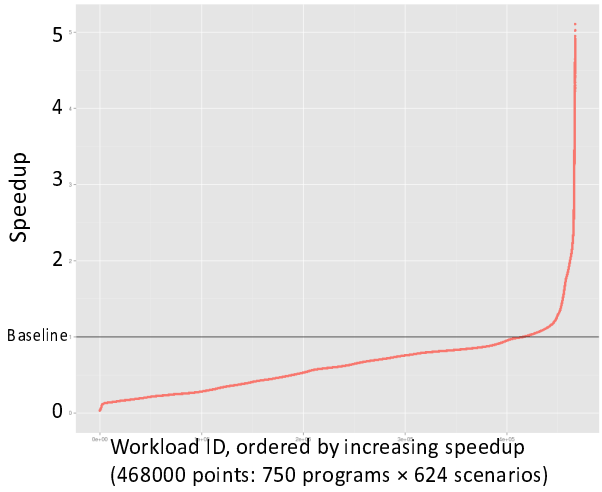

図 1 プログラムへの最適化手法と実行性能の関係

図 1 から，ベクトル化を施しても，大抵の場合は性能が向上するどころか逆に低下してしまうことが分かる．しかし，対象プログラムの特徴とその実行プラットフォームを考慮し適切にベクトル化を施せば，その性能は数倍にも向上することを示している．このように，コンパイル時に施す最適化は，プログラムに対して適切に施す必要がある．

2.2 機械学習と訓練データ

多数の最適化手法の中でベクトル化が有効であるかのみに注目しても，プログラムの性能がベクトル化を施さない場合に比べ向上するかどうかを事前に知ることは難しい．これはプログラムの特徴や，その実行プラットフォーム，プログラムへの入力などの性能決定要因が多様かつ複雑であるためである．

図 2 は，プログラムに対してベクトル化が有効かどうかを機械学習によって判断し最適化を行った場合の性能の変化を，ヒストグラムで表したものである．コンパイル対象となるプログラムは，約 750 のそれぞれ異なるテンソル縮約を行うもので，さらにそれぞれテンソルの次元を 10〜100 まで変化させたものを用いた．ここで “without ML” は機械学習を用いない場合（icc -O2，ベクトル化は有効）のプログラムの性能を，“with ML” は機械学習によってさ

Vol.2013-ARC-207 No.35
Vol.2013-HPC-142 No.35
2013/12/17

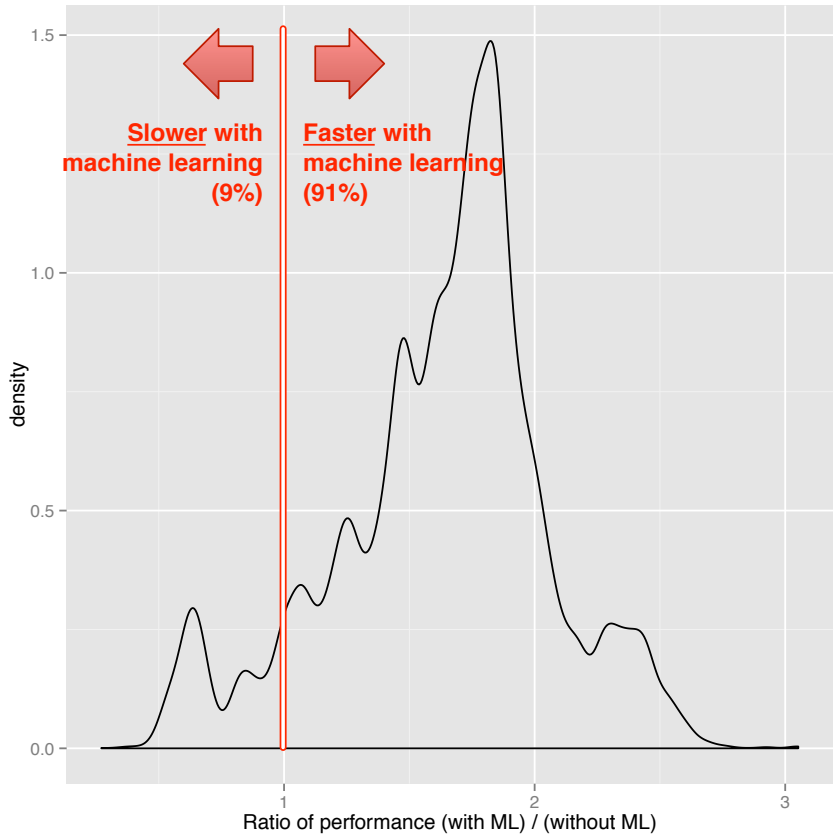

図 2 最適化手法の違いによる性能変化のヒストグラム

らにベクトル化を強いるオプション（-vec_threshold0）が有効かどうかを決定し実行した場合のプログラムの性能を表している．横軸は機械学習を用いた場合にどのように性能が変化したかの比を表しており，縦軸はその性能比の度数を表している．図より，機械学習によってパラメータを決定した場合，全てのプログラムの 90%以上で性能が向上していることが分かる．また，性能が向上したものの多くは 1.5〜2 倍近い性能向上を達成している．性能決定要因を考慮して適切な最適化手法を決定する上で，機械学習を用いる手法は有効であるといえる．

ここで対象プログラムに対して特定の最適化手法が有効かどうかについて精度の良い予測結果を得るためには，質の良い訓練データが必要である．訓練データの質とは，性能決定要因の範囲を網羅しており，例えば訓練データの内容であるプログラムの特徴や実行プラットフォームに関する値が多様であることを指している．質の良い訓練データを用いて生成されるモデルによって，未知のケースに対しても精度の良い予測ができるようになる．しかし，多様な性能決定要因の範囲を網羅し，学習するための訓練データを，どのようにして集めるかという課題がある．

3. HP3C（HPC Compiler in the Cloud）概要

HP3C は，コンパイル時にプログラムに対して最適化を施すことで，プログラム実行時の性能を向上させることを目標としている．従来の手法では，対象プログラムの特徴を無視して多数の性能決定要因の中の一部のみに着目し，単純化されたモデルによって，最適化手法の有効性を推測していた．しかし，このような手法の場合，複雑なプログラムやコンパイラ，ハードウェア上では役に立たない．そこで HP3C では，コンパイル後のプログラム実行時の性能を向上させるための一連の最適化手法・コンパイラオプションを機械学習によって決定する．

3.1 問題定義

本稿におけるコンパイラ最適化空間 Ω は，ユーザ側のベンダーコンパイラでの処理（バックエンド）を表すコンパイラオプションの集合 X と，HP3C 側で行う処理（フロントエンド）であるソースレベルで行う一連のコード変換の最適化手法の集合 Y との直積集合 $\Omega = X \times Y$ で表される．

ここで X，Y の各要素について見てみると，コンパイラオプション $\boldsymbol{x} \in X$ は，i 個のオプションのパラメータ

$a \in A$ の値を表す i 次元ベクトルで表される．そのため，X の要素数 $|X|$ は，コンパイラオプションのパラメータの有効範囲の組み合わせの数だけ存在する．次に，ソースレベルで行う一連のコード変換の最適化手法 $y \in Y$ は，ベクトル化，ループ展開，インライン展開などの n 個の最適化手法 $b \in B$ を用いて，$y = b_1\ b_3\ b_1\ b_1\ b_2 ...$ のように重複を許す順列で表される．またプログラムによって最適化手法の適用回数やその順序が異なるため，その長さは可変となる．そのため，Y の要素数 $|Y|$ は，各最適化手法 $b \in B$ とそのパラメータの有効範囲，適用回数の組み合わせとなり，爆発的に増加する．

$$X = \{\boldsymbol{x} \mid \boldsymbol{x} = (a_1, a_2, ..., a_i),\ 0 < i,\ a \in A\}$$
$$Y = \{y \mid y = (b_j)^*,\ 0 < j \leq n,\ b \in B\}$$

本稿では，Ω をコンパイラ最適化空間とし，フロントエンドで行う一連のコード変換の手法とバックエンドで行うコンパイラオプションを合わせて，最適化シナリオ（$\omega \in \Omega$）とする．X，Y の要素数は組み合わせによってともに爆発的に増加するため，結果としてその直積集合で表されるコンパイラ最適化空間 Ω の要素数も爆発的に増加する．

次にプログラムの集合を P，プログラムの静的特徴の空間を表す集合を SWC とする．プログラムの静的特徴とは実行プラットフォームに依存しないものを指しており，例えばプログラムの規模や，入れ子構造，配列のストライドなどが SWC の要素となる．SWC はプログラムの集合 P と射影の関係にあり，$swc(p)$ は，P から SWC への射影を表す．プログラムに対する静的特徴項目の数を k とすると，$\boldsymbol{swc} \in SWC$ は実数の k 次元ベクトルで表される．

$$SWC = \{\boldsymbol{swc} \in \mathcal{R}^k \mid \boldsymbol{swc} = swc(p),\ p \in P\}$$

HP3C では，プログラム p に対して最適化シナリオ ω を施したときの性能 $performance(swc(p),\ \omega)$ が最大となるような最適化シナリオ ω_0 を探索し決定する．

$$argmax(performance(swc(p),\ \omega))$$

または，

$$performance(swc(p),\ \omega_0) \geq performance(swc(p),\ \omega),\ \forall \omega \in \Omega$$

制約条件は，推測された最適化シナリオ $\hat{\omega}$ を施したプログラム p の実行時間の精度が α 以上であることとし，$accuracy(swc(p),\ \hat{\omega}) \leq \alpha\ (0 \leq \alpha \leq 1)$ と表す．

本稿では，特定の実行プラットフォーム・ベンダーコンパイラごとにモデルを生成するとして，この 2 つについては考慮しない．実行プラットフォームとベンダーコンパイラの違いも考慮して機械学習による探索を行う場合は，それぞれの集合を EP，VC とすると，目的関数と制約条件は以下のように表される．

$$performance(swc(p),\ \omega_0,\ ep,\ vc)$$
$$accuracy(swc(p),\ \hat{\omega},\ ep,\ vc) \leq \alpha \quad (0 \leq \alpha \leq 1)$$
$$ep \in EP,\ vc \in VC$$

3.2 最適化空間の探索手法

ベンダーコンパイラと連携して HP3C がプログラム p の最適化シナリオを決定する際に，対象プログラムの静的特徴 $swc(p)$ と実行プラットフォーム ep，ベンダーコンパイラ vc についての情報と合わせて用いることで，実行性能または最も適切な最適化シナリオ ω_0 を決定する．最もナイーブな解法は，最適化空間 Ω に対して全解探索を行うことである．つまり，対象プログラム p に対して，全通りの最適化シナリオを施した場合のプログラム実行時の性能 $performance(swc(p), \omega),\ \omega \in \Omega$ を，実際に実行して測定することで適切な最適化シナリオを決定する．しかし，探索すべき空間 Ω は，コンパイラオプションの集合 X とソースレベルでの最適化手法の集合 Y の直積集合であることから，Ω の要素数が組み合わせ爆発する．そのため，実際にプログラムを実行する全解探索は現実的ではない．

そこで HP3C は機械学習を用いてコンパイラ最適化空間 Ω を探索し，適切な最適化シナリオ ω_0 を決定する．機械学習を用いて，最適化シナリオを決定し性能を向上させる関連研究については，6 章で示す．HP3C は，最適化手法の中で特にベクトル化に着目し，より予測精度が高く，ベクトル化を施したプログラムの実行性能をより向上させる手法を研究している．具体的には，最近傍法（NN）を用いる場合は，プログラムの静的特徴 $swc(p)$ とプログラム実行時の性能が高くなる最適化シナリオ ω_0 の関係を学習し分類（多値分類）することで，与えられたプログラムの静的特徴 $swc(p')$ に対する適切な最適化シナリオ ω_0' を決定する．また線形回帰を用いる場合は，プログラムの静的特徴 $swc(p)$ に対して各最適化シナリオ $\forall \omega \in \Omega$ を施した場合の実行性能 $performance(swc(p), \omega),\ \forall \omega \in \Omega$ を予測し，その予測結果から最適化シナリオ $\hat{\omega}_0$ を決定する．

3.3 機械学習と訓練データ

推測した段階では，実際に決定した最適化シナリオ $\hat{\omega}_0$ によって性能が向上するかどうかが実際に分かるわけではない．そのため出力の最適化シナリオ $\hat{\omega}_0$ として選ばれるのは，その段階のモデルにより性能が「最大になると考えられる ω」である．

機械学習を用いて $performance(swc(p), \omega)$ を推測することによって，多数の性能決定要因 $swc(p)$ を入力としてモデルをすることができる．しかし，これらの多数の入力から精度よく実行時の性能を推測するためには，推測のために十分に学習しモデルを生成する必要がある．そのため，これらの多様な性能決定要因 $swc(p)$ を網羅し，未知のケー

スに対しても正しく実行性能 $performance(swc(p), \omega)$ を予測できるような訓練データが必要となる．また HP3C において考慮すべき性能決定要因の数は多数であることから，訓練データには，十分な数であることと，その内容が多様であることが求められる．

4. システムアーキテクチャ

4.1 要件定義

HP3C は本システムをクラウド上に構築することで，一般ユーザのプログラムに対しても最適化シナリオを提供できる形態にする予定である．これは，世界中の多くの一般ユーザが利用することで，多様な訓練データを収集するためである．収集した多数のデータは，最適化シナリオを推測するモデルの生成と，その精度の向上に役立てられる．

本システムは，ユーザ側で開発されているプログラムに対して，コンパイルを行う際にベンダーコンパイラと連携することで，クラウド上で最適化シナリオを決定し，ユーザに提供する．オンプレミス側でベンダーコンパイラを用いてプログラム開発を行う，一般ユーザを対象とする．図 3 に，ユーザ側から見た HP3C の動きを示す．ユーザはプログラムをコンパイルする際に，対象プログラムとともに，ユーザ側のベンダーコンパイラと実行プラットフォームに関する情報をクラウド上に送信する．HP3C は受信した情報にもとづいて，最適化シナリオを決定する．そのため一般ユーザを対象とした機能要件は次の通りである．

- プログラムのコンパイルを行う（hp3c-cc）
  - ユーザ側のコンパイル対象プログラム，ユーザ情報をクラウド上へ送る
  - クラウド上から，コード変換（フロントエンド）済のコンパイル対象プログラム，コンパイルオプションを受け取る
  - 受け取ったコンパイルオプションを用いて，ベンダーコンパイラによるコンパイル（バックエンド）を行う
- クラウド側へフィードバックを送る（hp3c-fb）
- データベース化された訓練データへアクセスする

本システムをクラウド化することによる利点として，ユーザは HP3C の所在を気にする必要がなくなるということ，コンパイルオプションを決定する際に，ユーザがローカルにシステムをダウンロード・インストールするなどして準備する必要がないということが挙げられる．またユーザが HP3C を利用し，コンパイル後の実行性能に関するフィードバックを収集することにより，最適化シナリオの予測精度も向上させることができる．さらに，機械学習のために収集したデータは，個人情報が漏洩しない形に処理してデータベース化し，一般に公開する予定である．これにより関連研究の発展にも貢献できると考えている．

本システムのクラウド側では，それぞれの役割を持つ 2 つのサーバ，収集した訓練データを蓄積するデータベースによって構成されている．図 4 に，各サーバ・データベースの構成を，図 5 にその動作の詳細を示した．詳細については，次節以降で述べる．

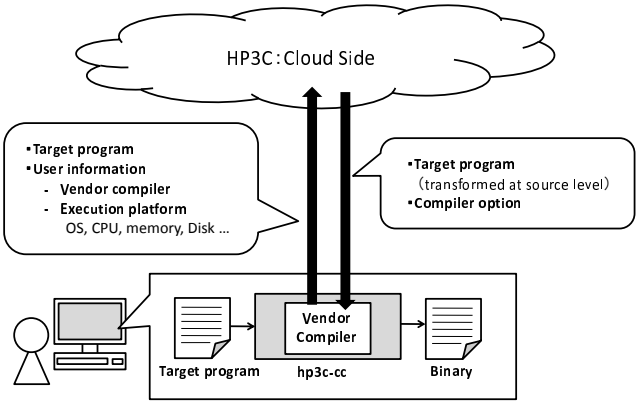

図 3 ユーザ側から見た HP3C の動き

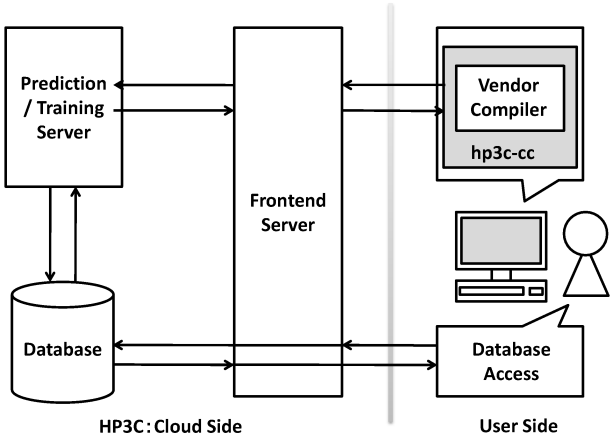

図 4 HP3C のシステム・アーキテクチャ

4.2 Frontend Server

Frontend Server は，ユーザからのアクセス・リクエストを受け付けるサーバ（図 4 右）である．コンパイル時のフロントエンド処理を行うほか，セキュリティの観点から，一般ユーザがデータベースへアクセスする際にはこのサーバを経由してアクセスする．ユーザが HP3C と連携してプログラムのコンパイルを行う際の機能要件を次に示す（図 5 参照）．

1. コンパイル対象のプログラム，ユーザ情報（ベンダーコンパイラ，実行プラットフォームに関する情報）を受け取る
2. プログラムから静的特徴の抽出
3. 機械学習による最適化シナリオ推測のリクエスト（Prediction/Training Server，4.3 節へ）
4. 最適化シナリオに沿ってソースレベルでのコード変換

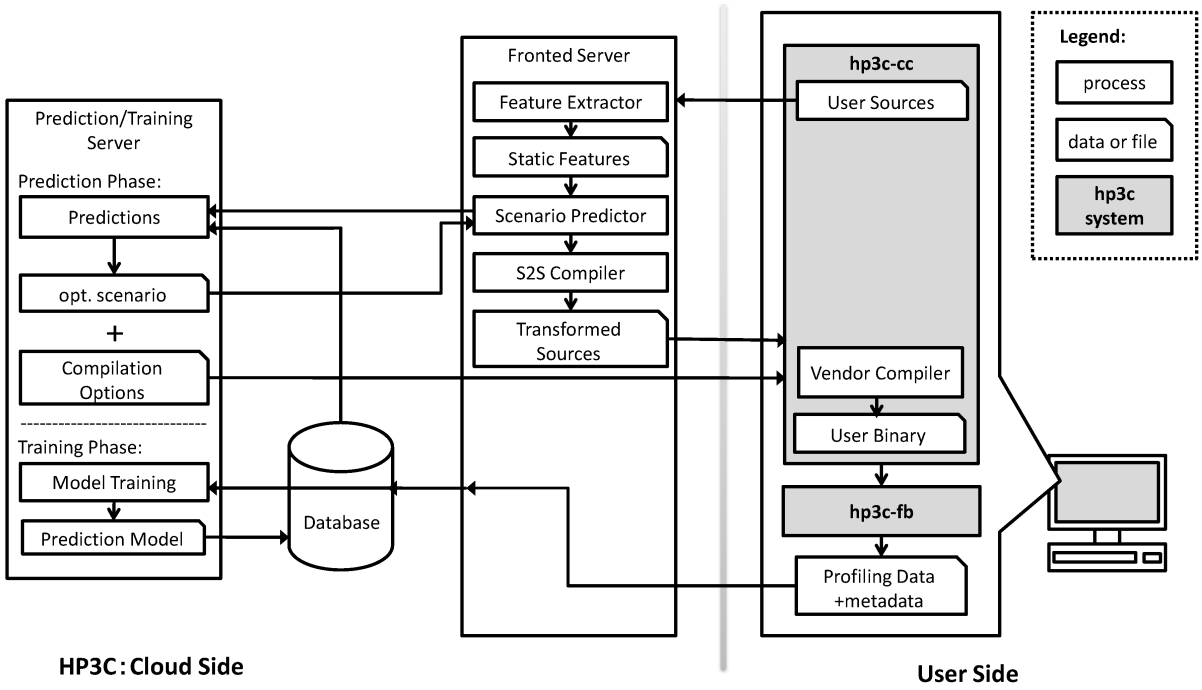

図 5 HP3C の内部動作の詳細

5. ユーザへコンパイルオプション，コード変換済みのプログラムを送る
6. ユーザ情報の匿名処理，データベースへのストア
7. プロファイルデータの受け取り・ストア

上記項目のうち，1，5 がユーザ対して行う動作となっている．Frontend Server は，ユーザに対するインターフェースとなっている他，最適化シナリオを推測する Prediction/Training Server へリクエストを行うための，プログラムの特徴抽出の準備やコード変換などの役割も担っている．

ベンダーコンパイラと連携して動作する他の機能要件としては，以下のとおりである．

- ユーザがプログラムを実行した際に得られる，実行時間などのプロファイルデータの収集・ストア
- プロファイルデータを用いた再学習のリクエスト
- 一般ユーザが訓練データにアクセスする際のインターフェース

4.3 Training/Prediction Server

Training/Prediction Server は，コンパイラ最適化空間の探索と最適化シナリオの決定を行うサーバ（図 4 左上）であり，機能要件は以下に示す通りである．

- 訓練データを用いた学習とモデルの生成
- 送られたプログラムの特徴・ユーザ情報から最適化シナリオを決定
- プロファイルデータを用いた再学習

これらの動作は，Frontend Server からのリクエストによって行われる．1 つ目は事前にモデルを生成するために，データベースにストアされている訓練データを用いて学習とモデルの生成を行い，2 つ目は Frontend Server から推測のリクエストが起こる度に行う．この際，Training/Prediction Server は，Frontend Server で抽出されたプログラムの静的特徴・ユーザ情報をもとにプログラム実行時の性能を推定し，最適化シナリオの決定を行う．3 つ目はプロファイルデータを用いたモデルの再学習であり，HP3C を利用した一般ユーザからのフィードバックとして得られるプロファイルデータを含めて行う．これにより，多様なデータに対してより正確に最適化シナリオを決定できるようになる．

4.4 データベース

HP3C のデータベース（図 4 左下）には，

- 学習のための訓練データ
- コンパイル時にユーザから得られるユーザの実行プラットフォームに関する情報とプログラムの特徴，決定した最適化シナリオ
- コンパイル後のプログラム実行時に得られる実行時間を含む，プロファイルデータ

が蓄積されている．コンパイル後にユーザから得られるデータが訓練データではなくプロファイルデータとなっているのは，このときユーザから得られるデータには最適

化シナリオに関する情報が含まれていないためである．再学習のための訓練データとして構成するために，Frontend Server がコンパイル時の情報とつなぎ合わせてからデータベースへ蓄積する．

また，これらの訓練データは一般に公開される予定であり，一般ユーザが Frontend Server を通じてデータベースへアクセスできるようになる．これにより，関連研究の発展に貢献できると考えられる．

5. おわりに

本稿では，コンパイラ最適化空間探索に機械学習を用いる HP3C の概要について示した．HP3C は，対象プログラムの静的特徴とユーザ側のベンダーコンパイラ，実行プラットフォームを考慮して最適化シナリオを決定することで，プログラム実行時の性能を最大化することを目的としている．また，最適化シナリオ決定の精度を高めるために本システムをクラウド上に構築することで，一般ユーザが利用できる形態を目指し，学習のための訓練データの収集を行う．

今後はより複雑なプログラムを対象として機械学習を用いてコンパイラ最適化空間探索に関する研究を引き続き行い，またシステムクラウド化のプロトタイプを構築する予定である．

6. 関連研究

本章では，機械学習を用いることによってコンパイラの性能を改善する手法についての関連研究を示す．

Stephenson らによる研究 [1] では，サポートベクターマシン（SVM），最近傍法（NN） を用いた多値分類によって，対象プログラムの性能が最大となるようなループ展開の因子を決定する．SPEC ベンチマーク[*1]を対象しており、最適ループ展開因子は 65%の精度で推測し、SPEC の性能は平均で 5%（最大で 23%）上がった．

Kulkarni らによる研究 [3] では，Java 仮想マシンの実行時コンパイラを対象として，マルコフ過程によって最適化シナリオのモデリングを行っている．コンパイルの各過程で ANN により，1 つずつ最適化手法を決定していく．この手法により，実行時間は最大で 20%削減されている．

Stock らによる研究 [4] では，6 つの機械学習のアルゴリズムを用いることによって，プログラムに対して自動ベクトル化を行い，実行時の性能を改善する．プログラムの特徴を抽出し，それを元にベクトル化を行う点で 2.2 節の図 2 の実験に類似しているが，次の点から，HP3C での最適化空間探索には向いていない．(1) 存在する最適化シナリオ毎に性能を推測するため，最適化空間が大きい場合に推測に時間がかかりすぎる．(2) [4] ではプログラムの特徴をバイナリから抽出するため，HP3C が想定するコンパイル前での抽出ができない．(3) ベンチマーク 2 つ利用した結果が示されているが，うち 1 つについては予測がランダムな探索により決定された最適化シナリオを施したケースをわずかに上回る程度である．

Milepost GCC [5] では，HP3C の開発動機に類似しており，自動的に最適化シナリオを決定できるように，汎用のベンダーコンパイラに対して機械学習の能力を加えたコンパイラを提供している．対象プログラムから静的特徴を抽出する点でも類似しているが，どのようなプログラムに対して評価を行っているのかは明らかにされていない．加えて [6] の予測精度は，プログラムの静的特徴をランダムに記述してコンパイルしたものとほぼ同じであった．

[*1] Online: http://www.spec.org/cpu2000/

参考文献

[1] Stephenson, M. and Amarasinghe, S.: Predicting Unroll Factors Using Supervised Classification, *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '05, IEEE Computer Society, pp. 123–134 (2005).

[2] Antoine, T., Arnaldo, C., Hiroki, F., Jun, M., Hadrien A., C., Kazuaki, M., Masaki, A., Tadashi, N. and Eiji, Y.: Using Machine Learning in Order to Improve Automatic SIMD Instruction Generation., *ICCS*, Procedia Computer Science, Vol. 18, Elsevier, pp. 1292–1301 (2013).

[3] Kulkarni, S. and Cavazos, J.: Mitigating the Compiler Optimization Phase-ordering Problem Using Machine Learning, *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, New York, NY, USA, ACM, pp. 147–162 (2012).

[4] Stock, K., Pouchet, L.-N. and Sadayappan, P.: Using machine learning to improve automatic vectorization, *ACM Trans. Archit. Code Optim.*, Vol. 8, No. 4, pp. 50:1–50:23 (2012).

[5] Fursin, Grigoriand Kashnikov, Y., Memon, A. W., Chamski, Z., Temam, O., Namolaru, M., Yom-Tov, E., Mendelson, B., Zaks, A., Courtois, E., Bodin, F., Barnard, P., Ashton, E., Bonilla, E., Thomson, J., Williams, C. K. I. and O'Boyle, M.: Milepost GCC: Machine Learning Enabled Self-tuning Compiler, *International Journal of Parallel Programming*, Vol. 39, pp. 296–327 (2011).