Vol.2021-HPC-178 No.20
2021/3/16

# MPIアプリケーションのキャッシュプロファイル予測

長谷川　健人[1,a)]　有馬　海人[1]　三輪 忍[1]　八巻 隼人[1]　本多 弘樹[1]

**概要**：並列アプリケーションの開発や性能チューニングに利用されるプロファイルは、通常は解析対象のアプリケーションを解析対象の計算環境で実行することにより取得する。プロファイリングに要するコストはアプリケーションの大規模化、複雑化にともなって増大するため、並列アプリケーション解析コストの削減はエクサスケール級のスーパーコンピューティングにおける重要な課題の一つである。本稿では、MPI 並列アプリケーションのキャッシュプロファイルを予測する手法を提案する。アプリケーションを実際に実行してプロファイルを取得するのは少ないコア数や小さな問題サイズに限定し、多いコア数や大きな問題サイズのプロファイルは上記のプロファイルから予測することで、大規模なプロファイルの取得に要するコストを削減する。TSUBAME3.0 と NAS Parallel Benchmark を用いて評価した結果、提案手法は問題サイズの変化に対して L1 データキャッシュミス数を平均絶対誤差 58.7%の精度で予測し、プロファイル取得に要するコストを 11.3%に削減した。

**キーワード**：MPI, キャッシュ, プロファイル, モデル

## 1. はじめに

スーパーコンピューティングシステムは、計算科学分野におけるアプリーケーションの大規模化、複雑化にともなってシステム規模を増大させてきた。例えば、現在世界最速のスーパーコンピュータ「富岳」は、その前進である京コンピューターの約 2 倍の数の CPU を有する [1]。このようなシステムでは、多くのエンドユーザーが、これまでにない規模の並列アプリケーションを開発、実行するようになると予想される。並列アプリケーションの開発やチューニングにはアプリケーションの解析が必須であり、超大規模環境に適した並列アプリケーション解析技術が必要とされている。

現在の並列アプリケーションの解析技術にはプロファイリングとモデリングの二つが存在し、これらが用途に応じて使い分けられている。

プロファイリングは、コンパイラや外部ライブラリの支援により、計測用のコードを解析対象アプリケーションに挿入しアプリケーションを実行することで関数単位の実行時間などのプロファイルを取得する [2], [5]。このように解析対象のアプリケーションそのものを 1 度実行しなければならないため、アプリケーションの実行時間と計測用コードの実行時間の合計の時間がプロファイリングには必要となる。またプロファイリングには、実際にアプリケーションを実行するときと同じ規模の計算資源も必要である。後述するモデリングに比べて情報取得に必要な時間や計算資源などは大きいが、正確な情報を多く得られるのがプロファイリングのメリットである。

一方、モデリングは、少数のコアで測定したアプリケーションまたは関数の実行時間から性能のスケーラビリティモデルを作成し、多数のコアで同アプリケーションまたは同関数を実行した際の実行時間を予測する [3], [7]。得られる情報は実行時間のみであるが、少数のコアでの実行結果から予測を行うためプロファイリングよりも少ない計算資源と短い時間でアプリケーションの実行時間を見積もることができる。

このように、プロファイリングとモデリングは取得できる情報の量や精度と、情報の取得に要するコスト（時間と計算資源）との間にトレードオフの関係がある。並列アプリケーションの複雑化、大規模化にともなってプロファイリングによる情報取得コストは増大し、モデリングとプロファイリングのコスト差は広がると考えられる。モデリングは少数コアでの実行結果から大規模実行時の性能を予測するため、アプリケーションの複雑化や大規模化によるコストの増大は軽微である。しかしモデリングでは予測対象がアプリケーションまたは関数の実行時間に限られており、性能チューニング等に使用する情報としては不十分である。プロファイリングはチューニングに利用可能な多量

[1] 電気通信大学
1-5-1, Chofugaoka, Chofu, Tokyo 182–8585, Japan
[a)] hasegawa@hpc.is.uec.ac.jp

の正確な情報を得られるが、プロファイリングではアプリケーションそのものを実行する必要があり、超大規模アプリケーションのプロファイリングは容易でない。

前述の問題に対して我々は、少数コアや小さい問題サイズなどの小規模での実行時のプロファイルから大規模実行時のプロファイルを予測する手法を提案する。予測により得られるプロファイルはモデリングにより得られる情報と同様の近似的な情報であるが、チューニングに使用する情報は必ずしも正確な情報が必要ではなく、近似的な情報でも十分な可能性が高い。そのため、プロファイル予測の技術を開発することにより、超大規模アプリケーションのプロファイルを短時間で得られるようになり、アプリケーション開発者の生産性の向上に寄与すると考えられる。プロファイルに含まれる様々な情報を予測することが本研究の最終目標であるが、本稿ではキャッシュミス数に着目し、キャッシュプロファイルの予測技術を開発する。

## 2. 関連研究

プロファイリングに要するコストの削減方法の一つにプロファイル取得のオーバーヘッドを削減する方法がある。例えば、HPC 向けのプロファイリングツールである TAU[5] には TAU_THROTTLE がある。TAU_THROTTLE は、関数コール回数が一定以上かつ関数の 1 コールあたりの実行時間が一定以下である関数のプロファイリングを制限する機能である [6]。これにより短い実行時間で多くの回数コールされる関数のプロファイル取得のオーバーヘッドを削減し、プロファイルの取得にかかるコストを削減できる。

また、並列アプリケーションの解析ツールとして Extra-P が開発されている [7]。Extra-P は並列アプリケーションの性能モデリングを行うツールであり、入力問題サイズやプロセス数などのパラメータを用いて、アプリケーション内の各関数の実行時間や消費電力を予測する。使用できるプログラミングモデルは MPI と OpenMP の二つである。

## 3. 提案手法

本章では我々が提案するキャッシュプロファイル予測手法を詳しく述べる。以下では、まず提案手法の全体像を説明し、続いて予測に使用するモデルを説明する。

### 3.1 概要

キャッシュプロファイルには様々な階層のキャッシュのアクセス情報が含まれるが、簡単のため、本稿で予測の対象とする情報は 1 プロセス当たりの L1 データキャッシュミス数のみとする。また、予測は関数毎に行い、予測する関数はユーザ定義関数に限定する。予測対象に MPI 関数などの組み込み関数を含めなかったのは、アプリケーション開発者が自身のアプリケーションの性能解析を行う際に最初に注目するのがユーザ定義関数だと考えたからである。

提案手法では少ないコア数や小さい問題サイズなどの小規模実行で取得した L1 データキャッシュミス数から、多いコア数や大きい問題サイズなどの大規模実行時の L1 データキャッシュミス数を予測する。提案手法はコア数に対する予測と問題サイズに対する予測の 2 つに分けられる。

提案手法の一連の流れを説明する。コア数に対する予測では、初めに解析対象アプリケーションを解析対象の環境でいくつかの少ないコア数 (例えば 8,32,128 コアなど) で実行し、複数のプロファイルを取得する。取得したデータよりコア数と L1 データキャッシュミス数の関係を表現するモデルを推定し、このモデルを用いて実際には実行していないコア数 (例えば 256 コア) で同アプリケーションを実行した際の L1 データキャッシュミス数を予測する。

問題サイズに対する予測も同様に行う。解析対象のアプリケーションを小さな問題サイズで実行したときの L1 データキャッシュミス数より、問題サイズと L1 データキャッシュミス数の関係を表現するモデルを推定する。そして、同アプリケーションを実際には実行することなく、推定したモデルを用いて大きな問題サイズにおける L1 データキャッシュミス数を予測する。

### 3.2 予測に使用するモデル

コア数に対する予測では linear,inverse,log,exponentail の 4 つのモデルを使用する。それぞれのモデルの式を以下に示す。

$$y = ax + b \tag{1}$$

$$y = a + \frac{b}{x}\,(0 \leq a) \tag{2}$$

$$y = \frac{\log x}{\log a} + b\,(1 < a) \tag{3}$$

$$y = ab^{-x} + c\,(1 < b, 0 \leq c) \tag{4}$$

上記の式において $x$ はコア数、$y$ は L1 データキャッシュミス数を表す。また、$a, b, c$ は係数であり、小規模実行によって取得したプロファイルを用いてフィッティングを行うことにより求める。フィッティングは残差平方和を最小とするように行う。なお、フィッティングは上記の 4 つのモデルすべてに対して行う。

4 つのモデルからプロファイル予測に使用するモデルを選択する際は、フィッティングに使用したデータに対して各モデルの平均絶対パーセント誤差（MAPE）をまず計算し、MAPE の値が最小となったモデルを予測に使用するモデルとして選択する。MAPE は以下の式で表される。

$$MAPE = \frac{100\%}{N} \sum_{t=1}^{N} \left| \frac{A_t - F_t}{A_t} \right| \tag{5}$$

上記の式において $A_t$ はプロファイリングを利用して取得した L1 データキャッシュミス数（実測値）、$F_t$ はモデルが予測する L1 データキャッシュミス数（予測値）である。

表 1: TSUBAME3.0 のハードウェア構成

| 計算ノード | 540 台 |
|---|---|
| CPU(ノードあたり) | Intel Xeon E5-2680 V4 Processor(Broadwell-EP, 14 コア, 2.4GHz) × 2 |
| RAM(ノードあたり) | 256GiB (DDR4-2400 32GB モジュール × 8) |
| ローカルストレージ | Intel DC P3500 2TB (NVMe, PCI-E 3.0 x4, R2700/W1800) |
| ネットワーク | Intel Omni-Path 100Gb/s × 4 |
| 外部ストレージ | DDN SFA14KXE 及び EXAScaler |

表 2: TSUBAME3.0 のキャッシュ構成

| キャッシュ名 | サイズ [KB] |
|---|---|
| L1 データキャッシュ | 32 |
| L1 命令キャッシュ | 32 |
| L2 キャッシュ | 256 |
| L3 キャッシュ | 35,840 |

$N$ はフィッティングに使用したデータの数である。

問題サイズに対する予測では、時間の都合により、今回は式 (6) に示す linear モデルのみを使用した。

$$y = ax + b\,(a > 0) \tag{6}$$

上記の式において $x$ は問題サイズを表す。$y, a, b$ の意味は式 (1) と同様である。

コア数に対する予測と同様、小規模実行で取得したデータを用いて上記のモデルに対してフィッティングを行い、各係数の値を求める。このようにして推定したモデルを用いて大きい問題サイズの L1 データキャッシュミス数を予測する。

## 4. 評価方法

### 4.1 実験環境

実験は東工大の TSUBAME3.0[4] を用いて行った。TSUBAME3.0 のハードウェア構成を表 1 に示す。TSUBAME は 540 台の計算ノードからなり、各計算ノードは 2 つの CPU を有している。各 CPU には 14 個のコアがある。本実験ではこの CPU を使用して実験を行う。

表 2 に TSUBAME3.0 の CPU のキャッシュメモリの構成を示す。L1 データ/命令キャッシュと L2 キャッシュはコア内に配置されており、L3 キャッシュはコア間で共有されている [8]。

実験には NAS Parallel Benchmarks (NPB) に含まれる 6 つのプログラム（表 3）を使用した [9]。実験に使用した問題サイズは A, B, C, D の 4 つである。問題サイズは、B と C がそれぞれ A と B の 4 倍，D が C の 16 倍の関係にある。

プログラムの実行に使用するコア数を 8~256 コアに変更して実験を行った。ただし、FT, IS, LU の 3 つのプログラムについては、問題サイズ D を 8 コアで実行した時の実行時間が長すぎたため、本実験では評価に含めない。プログラムを実行する際は、1 つの MPI プロセスが 1 つのコアを使用するようにした。

表 3: 実験に使用したプログラム

| プログラム名 | 詳細 |
|---|---|
| cg | 共役勾配法 |
| ep | 独立している Embarassingly parallel なタスク |
| ft | 離散 3 次元高速フーリエ変換 |
| is | 基数ソート |
| lu | ガウスザイデル法 |
| mg | 連続したメッシュに対するマルチグリッド計算 |

L1 データキャッシュミス数と実行時間の取得には，TAU と PAPI[10] を使用した。

### 4.2 評価目的

まず最初に、前章で述べた 4 つのモデルに関して、フィッティングに使用するデータに対する適合度を評価する。これは各モデルがコア数と L1 データキャッシュミス数の関係をどの程度正しく表せているかを評価するために行う。

次に、提案手法の予測精度を評価する。具体的には、フィッティングに使用しなかった規模の実行に対してモデルを用いて L1 データキャッシュミス予測を行い、モデルから得た予測値とプロファイリングを実際に行って取得した実測値とを比較する。

最後に、提案手法によるプロファイル取得コストをプロファイリングによるプロファイル取得コストと比較する。本稿では、プロファイル取得に要する実行時間と使用するコア数の積をプロファイル取得コストと定義する。これは多くのスーパーコンピューティングサービスの利用料はポイント制となっており、エンドユーザは計算に使用した資源と計算時間の積に応じてポイントを消費するためである。

上記以外にモデルのフィッティングと予測処理が提案手法には必要だが、これらの処理は通常の PC で高速に実行できるため提案手法によるプロファイル取得コストには含まれないものとした。

### 4.3 具体的な予測方法

コア数に対する予測では、8 ~ 128 コアでアプリケーションを実行した際の L1 データキャッシュミス数を用いてモデルの推定を行い、推定したモデルを用いて同アプリケーションを 256 コアで実行した際の L1 データキャッシュ

表 4: モデル選択割合及び絶対平均誤差率

| | モデルの割合 [%] (MAPE の最小値 [%], MAPE の最大値 [%]) | | | |
|---|---|---|---|---|
| プログラム | linear | inverse | log | exponential |
| cg | 17.86 (0.67, 9.35) | 57.14 (0.57, 14.86) | 1.79 (1.75, 1.75) | 23.21 (0.23, 10.06) |
| ep | 0.00 (-,-) | 100.00 (0.00, 30.24) | 0.00 (-, -) | 0.00 (-, -) |
| ft | 7.14 (1.51, 155.19) | 69.39 (0.00, 127.04) | 3.06 (8.35, 15.84) | 20.41 (0.31, 25.26) |
| is | 7.14 (1.10, 6.92) | 48.21 (0.29, 90.71) | 1.79 (1.27, 1.27) | 42.86 (0.10, 32.82) |
| lu | 9.85 (0.34, 16.90) | 57.58 (0.77, 61.84) | 0.76 (3.17, 3.17) | 31.82 (0.35, 29.87) |
| mg | 2.27 (2.26, 8.11) | 72.73 (0.11, 1241.03) | 0.00 (-, -) | 25.00 (0.81, 25.96) |

ミス数を予測した。モデルの推定に使用したデータセットは具体的には以下の 16 通りである。8:16:32, 8:16:64, 8:16:128, 8:32:64, 8:32:128, 8:64:128, 16:32:64, 16:32:128, 16:64:128, 32:64:128, 8:16:32:64, 8:16:32:128, 8:16:64:128, 8:32:64:128, 16:32:64:128, 8:16:32:64:128。また参考までに、8 ∼ 256 コアで実行した際のデータすべてを用いてモデルの推定を行い、256 コアで実行した際の L1 データキャッシュミス数の予測も行った。

一方、問題サイズに対する予測では、問題サイズ A∼C を 256 コアで実行したときのデータから問題サイズ D のキャッシュミス数の予測を行った。モデルの推定には、問題サイズ A と B の 2 種類、および、問題サイズ A∼C の 3 種類のデータを使用した。また参考までに、問題サイズ A∼D の 4 種類のデータすべてを使用してモデルの推定を行い、問題サイズ D の L1 データキャッシュミス数を予測する場合についても評価した。

## 5. 評価結果

### 5.1 モデル適合度

コア数と L1 データキャッシュミス数の関係を表す関数の評価を行った。モデル適合度の評価ではコア数 8,16,32,64,128,256 で実行したプロファイルを用いてモデルのフィッティングを行った。フィッティングによって得られた関数を使用して、各コア数に対応する予測値を計算によって求めた上で、各関数に対して予測値と実測値から MAPE の値を求めた。全関数の MAPE の平均値を各ベンチマークの問題サイズ A,B,C,D の 4 つで計算した。結果を図 1 に示す。全ベンチマーク、全問題サイズの MAPE の平均は 10.8% となった。

mg の問題サイズ D を除き、MAPE は 20%以下となった。mg の問題サイズ D では comm3_ex_という関数の MAPE 値が 1,000%を超える値となった。この関数は問題サイズ A,B,C,D の全てにおいてコア数 8,16,32 でキャッシュミス数が増加してコア数 32 で最大となり、コア数 32 から減少に転じてコア数 64,128,256 ではほぼ一定の値となる。今回用意したモデルは全て単調増加もしくは単調減少するモデルであり、したがってこの関数には適合しない。そのため comm3_ex_関数の予測精度を向上させるには、増加減少の傾向が途中で変わるモデルを追加する必要があると考えられる。

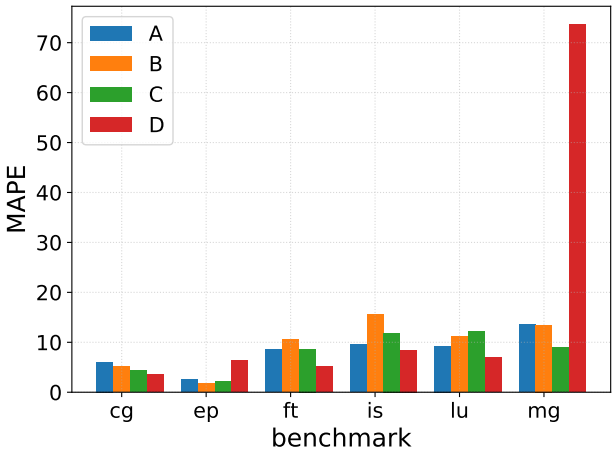


図 1: モデル適合度

4 つのモデル linear,inverse,log,exponential のそれぞれが選択された割合とそのモデルが選択されたときの MAPE の最小値と最大値を表 4 にまとめた。表 4 より linear モデルはベンチマーク ft を除いてモデルが適合していることが分かる。次に、inverse モデルは全ベンチマークで最も選択されたモデルであることが分かる。inverse モデルでは MAPE の最小値は 1%未満である一方、最大値は 127%や 1241%となるケースもある。これは inverse モデルが精度よく適合している関数がある一方で、精度が悪い関数も inverse モデルを選択してしまっていると考えられる。このことから精度を向上させるには新たなモデルの追加が必要と思われる。次に、log は選択される割合が低いが、選択されたときは MAPE の値が小さいことが分かる。最後に、exponential モデルは選択される割合が linear や log より多く、MAPE の値は最大でも 33%であり、精度よく適合していると考えられる。

### 5.2 予測精度

提案手法による予測精度を評価した。予測精度の評価には式 (7) で定義される平均誤差率を用いた。

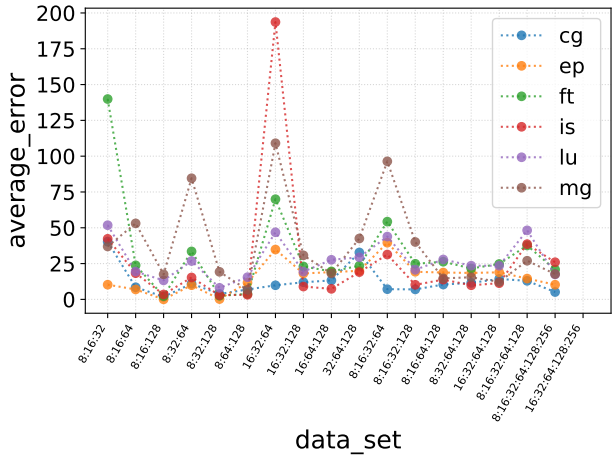


図 2: コア数に対する予測の精度 (問題サイズ A)

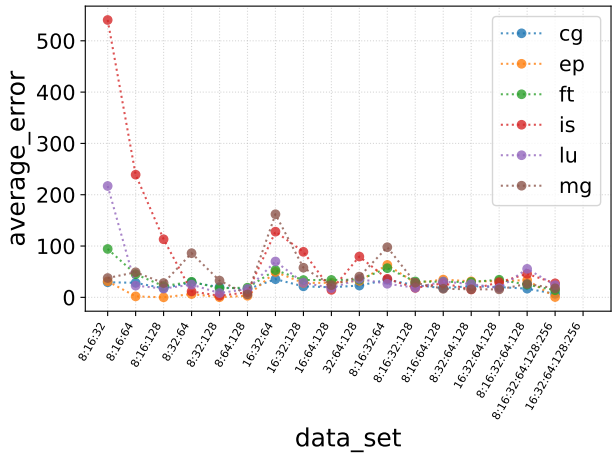


図 3: コア数に対する予測の精度 (問題サイズ B)

$$average\_error = \frac{100}{N_f}\sum_{t=1}^{N_f}\left|\frac{A_t - F_t}{A_t}\right| \tag{7}$$

$N_f$ はユーザ定義関数の総数、$A_t$ は実測値 $F_t$ は予測値である。

問題サイズ A の結果を図 2 に、問題サイズ B の結果を図 3 に、問題サイズ C の結果を図 4 に、問題サイズ D の結果を図 5 に示す。すべての問題サイズとプログラムに対して、コア数 8,64,128 のデータから予測を行った際の平均誤差率は 13.1%となった。

問題サイズ ABCD の 4 つで共通する傾向として、フィッティングに使用するデータセットの選択方法で平均誤差率が大きく変わる点が挙げられる。予測に使用するデータを 3 種類のコア数から 4 種類のコア数に増やすだけでは平均誤差率を改善することができないことが分かる。そのため、データの数ではなくデータの選択方法が平均誤差率に影響を与えることが分かる。

また、図 3 では、lu,ft,is の 3 つのベンチマークが 8,16,32 コアのデータから予測したときの平均誤差率が最悪の数値であった。また 16,32,64 コアのデータから予測したときの平均誤差率がほかのデータセットの値より悪くなっている。このことから使用するデータはコア数が大きく違うものを使用すると、精度の良い予測が行えると考えられる。




図 4: コア数に対する予測の精度 (問題サイズ C)




図 5: コア数に対する予測の精度 (問題サイズ D)

cg の問題サイズ A に対して、8,64,128 コアのデータから 256 コアのキャッシュミス数の予測を行った際の関数ごとの予測結果を表 5 にまとめる。各関数の予測精度は式 8 の relative_error で評価した。

$$relative\_error = 100 \times \left|\frac{A - F}{A}\right| \tag{8}$$

上記の式において $A$ は実測したキャッシュミス数、$F$ は予測したキャッシュミス数である。

表より、一部の関数は 1%以下の relative_error で予測できている一方で、中には relative_error が 10%を超える関数もあった。

問題サイズに対する予測の評価結果を図 6 に示す。図の縦軸は平均誤差率、横軸はフィッティングに使用したプロファイル数である。なお、プロファイル数 2, 3, 4 は、それぞれ、問題サイズ A~B、A~C、A~D のプロファイルを表す。平均誤差の全プログラムに対する平均値は、プロファ

表 5: コア数に対する予測の精度（cg, 問題サイズ A）

| function name | relative error[%] |
|---|---|
| .TAU | 3.8886 |
| main | 3.9389 |
| MAIN__ | 3.9416 |
| makea_ | 9.1129 |
| sprnvc_ | 9.8794 |
| conj_grad_ | 2.8559 |
| initialize_mpi_ | 0.3995 |
| randlc_ | 14.2681 |
| icnvrt_ | 23.1941 |
| vecset_ | 12.0894 |
| sparse_ | 2.6308 |
| alloc_space_ | 4.7269 |
| setup_submatrix_info_ | 0.0027 |
| setup_proc_info_ | 5.8198 |

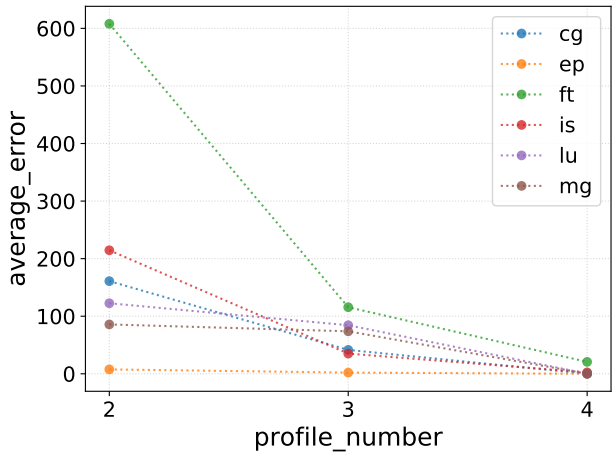


図 6: 問題サイズに対する予測精度

イル数 2 で 199%、プロファイル数 3 で 58.7% となった。

図 6 より予測に使用するプロファイル数を増やすと平均誤差率が減少することが分かった。一方、問題サイズ A~C の 3 つのプロファイルを使用した予測では、平均誤差率が 100%を超えているプログラムがあった。問題サイズ A,B,C,D は問題 A を基準として 1,4,16,256 倍の問題サイズに相当する。今回のプロファイル数 3 の予測は、1,4,16 倍の問題サイズから 256 倍の問題サイズを予測することに該当する。そのため 256 倍の問題サイズと比較して極端に小さい問題サイズ 3 種類から予測を行っていることになる。予測に 32,64,128 倍に相当する問題サイズのプロファイルを使用できれば予測精度が改善できる可能性がある。

### 5.3 予測コスト

コア数に対する予測に要するコストを評価した。各データセットに対して予測を行うのに要するコストを求めた。さらに 256 並列で実際にプロファイルを取得するコストと比較した。

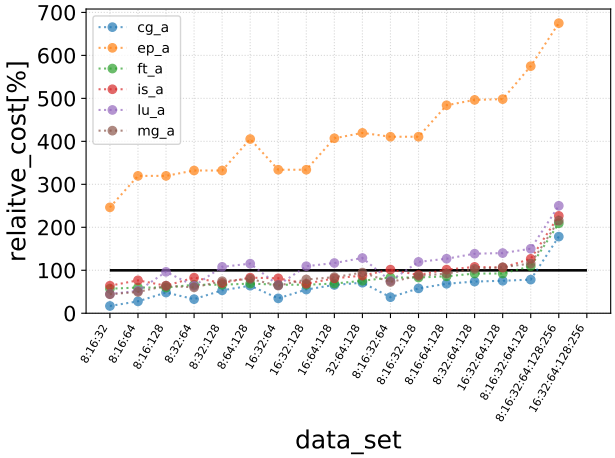


図 7: コア数に対する予測の相対コスト（問題サイズ A）

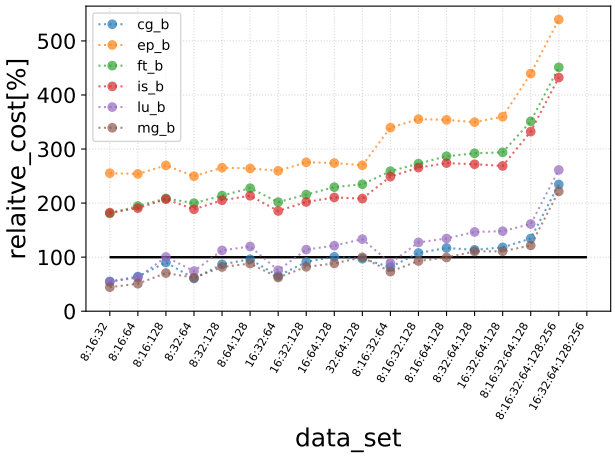


図 8: コア数に対する予測の相対コスト（問題サイズ B）

コア数に対する予測コストを式 (9) で評価した。

$$relative\_cost = 100 \times \frac{C_p}{C_E} \tag{9}$$

上記の式において $C_E$ は実際にプロファイルを取得する際のコスト、$C_p$ は予測に要するコストである。

問題サイズごとに相対コスト (relative_cost) を評価した結果を図 7~10 に示す。コア数 8,64,128 のデータから予測を行った際の全プログラムと全問題サイズの相対コストの平均は 180.1% となった。

コア数に対する予測のコストはプログラムによって大きく異なる結果となった。例えば cg では、問題サイズ A,B において一部のデータセットで予測のコストが実際にプロファイルを取得するコストを下回った。一方、ep ではすべての問題サイズで予測のコストが実際にプロファイルを取得するコストを上回った。

図 7 より、問題サイズ A では多くのデータセットで予測のコストが実際にプロファイルを取得するコストを下回った。一方、図 10 の問題サイズ D ではすべてのデータセットでの予測のコストが実際にプロファイルを取得するコス

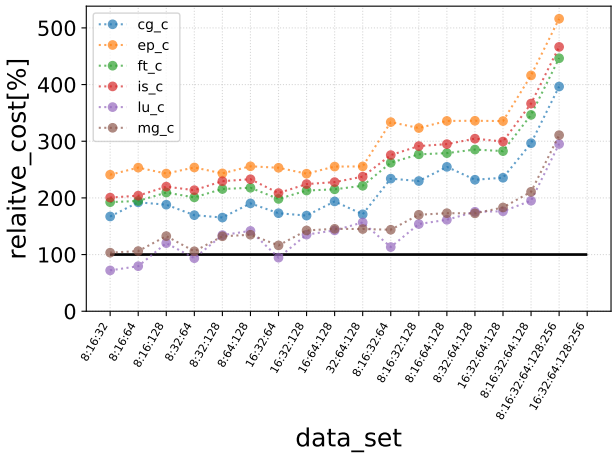


図 9: コア数に対する予測の相対コスト（問題サイズ C）

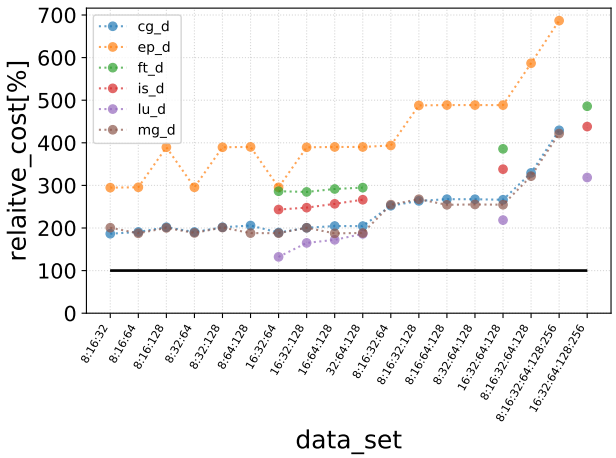


図 10: コア数に対する予測の相対コスト（問題サイズ D）

トを上回った。

問題サイズに対する予測では、問題サイズ D を実行してプロファイルを取得する際のコストと、同問題サイズのプロファイルを予測により取得する際のコストを比較して評価した。問題サイズ D を実行してプロファイルを取得するのに要するコストを $C_E$、予測に要するコストを $C_p$ として式 (9) で評価した。図 11 に結果を示す。

図 11 の Relative_cost は全プログラムの平均で、プロファイル数 2 で予測のコストは実際にプロファイルを取得するコストの 4.0%、プロファイル数 3 で予測のコストは実際にプロファイルを取得するコストの 11.3% となった。最も値が大きくなったプログラムでも実際にプロファイルを取得するコストと比べ、予測に使用するプロファイル数 2 で予測のコストを 20%以下、予測に使用するプロファイル数 3 で予測のコストを 25%以下に削減できた。

## 6. おわりに

### 6.1 まとめ

本稿では、モデルを用いて MPI 並列アプリケーション

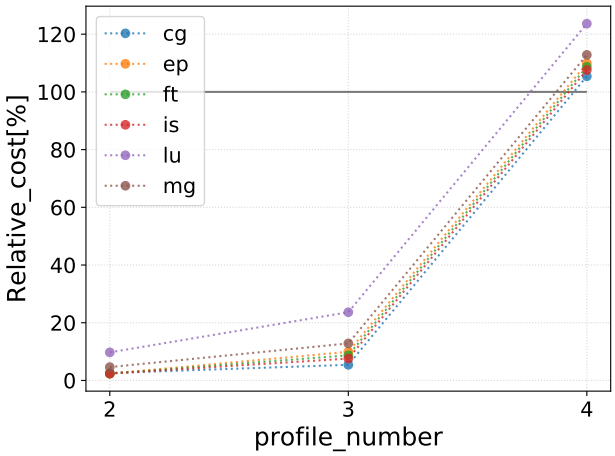


図 11: 問題サイズに対する予測のコスト（コア数 256）

の L1 データキャッシュミス数を予測する手法を提案した。モデルの適合度を TSUBAME3.0 と NPB を用いて評価した結果、モデルの評価は全プログラム、全問題クラスの平均で平均絶対誤差率が 10.8% となった。コア数とキャッシュミス数の関係を表現するモデルは一部の関数で値が悪く、改善が必要である。

またコア数に対する予測では、8,64,128 コアのデータセットを使用して 256 コアのプロファイルの予測を行った結果、全プログラムと全問題サイズに対する平均で 13.2% の平均誤差率で予測を行うことができた。一方でこの予測にかかったコストは、実際にアプリケーションを実行して取得するときの 1.8 倍のコストとなってしまった。

一方、問題サイズに対する予測では、問題サイズ A~C の 3 つのプロファイルから問題サイズ D のプロファイルを予測したところ、全プログラムに対する平均で 58.7% の誤差率で予測を行うことができた。予測を行うためのプロファイル取得に要したコストは全プログラムの平均で、問題サイズ D を実行したときに要するコストの 11.3% に削減できた。

### 6.2 今後の展望

本研究ではコア数に対する予測と問題サイズに対する予測を独立して行った。今後はコア数と問題サイズの両方を同時に変化させた際の予測も行う。本研究では扱わなかった L1 命令キャッシュや L2 キャッシュ、L3 キャッシュについても同様にプロファイルの予測を行う。また本研究では MPI 関数を除いたユーザー定義関数の予測を行った。今後は MPI 関数の予測も同様に行う。本研究では 8 から 256 のコア数及び A~D の問題サイズを扱った。今後は 256 以上のコア数や E や F などの問題サイズにも予測を拡張する。また、実用的な HPC アプリケーションに対する評価も行う予定である。

**謝辞** 本研究は JSPS 科研費 JP20H04193 の助成を受け

たものです.

## 参考文献

[1] TOP 500 November 2020 https://www.top500.org/lists/top500/2020/11/ (Accessed on 01/28/2021)
[2] Knüpfer A. et al. (2012) Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Brunst H., Müller M., Nagel W., Resch M. (eds) Tools for High Performance Computing 2011. Springer, Berlin, Heidelberg.
[3] PMaC Performance Modeling and Charactrization https://www.sdsc.edu/pmac/research-areas/index.html (Accessed on 01/26/2021)
[4] TSUBAME3.0 利用の手引き https://helpdesk.t3.gsic.titech.ac.jp/manuals/handbook.ja/jobs/ (Accessed on 2021/02/23)
[5] S. Shende and A. D. Malony, "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287-331, Summer 2006
[6] TAU throttle https://www.cs.uoregon.edu/research/tau/docs/tutorial/ch01s05.html (Accessed on 02/07/2021)
[7] Extra-P https://www.scalasca.org/software/extra-p/download.html (Accessed on 01/08/2021)
[8] Intel Xeon E5-2680 V4 https://ark.intel.com/content/www/jp/ja/ark/products/91754/intel-xeon-processor-e5-2680-v4-35m-cache-2-40-ghz.html (Accessed on 01/20/2021)
[9] NAS Parallel Benchmarks https://www.nas.nasa.gov/publications/npb.html#url (Accessed on 01/10/2021)
[10] Terpstra, D., Jagode, H., You, H., Dongarra, J. "Collecting Performance Data with PAPI-C", Tools for High Performance Computing 2009, Springer Berlin / Heidelberg, 3rd Parallel Tools Workshop, Dresden, Germany, pp. 157-173, 2010.