Vol.2021-HPC-178 No.19
2021/3/16

MPIアプリケーションの関数コール回数予測

有馬　海人[1,a)]　長谷川　健人[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実行には多くの計算資源と時間を必要とすることから、超大規模なアプリケーションのプロファイリングは一般に困難である。この問題に対して、アプリケーションを小規模実行した際の結果から大規模実行した際の結果を予測するツールである Extra-P が開発されている。しかし、Extra-P が出力する情報は基本的には関数ごとの実行時間のみであり、プロファイラが一般に出力する関数ごとのコール回数は出力しない。そこで本稿では、アプリケーションを大規模実行した際の関数コール回数を、同アプリケーションを小規模実行した際の関数コール回数から予測する手法を提案する。実際にプロファイリングを行うケースを小規模実行に限定することによって、アプリケーションを大規模実行した際の関数コール回数の取得に要するコスト（時間と計算資源）を削減する。TSUBAME3.0 と NPB を用いて評価した結果、提案手法は 256 コアを用いて問題サイズ A∼C を実行した際の関数コール回数から同コア数で問題サイズ D を実行した際の関数コール回数を 19.3%の平均誤差率で予測し、関数コール回数の取得に要するコストを 5%にまで削減できることがわかった。

キーワード：MPI, プロファイル, 関数コール回数, モデル

1. 序論

並列アプリケーションのプロファイリングは、性能解析や性能チューニングなどを目的として、スーパーコンピュータのユーザ間で広く行われている。プロファイリングは、通常、解析対象のアプリケーションを解析対象のシステム上で実行することによって行われる。ところが、超大規模な並列アプリケーションの場合は実行そのものに膨大な時間と計算資源を必要とするため、そのようなアプリケーションのプロファイリングは一般に困難である。

上記の問題に対し、並列アプリケーションの性能解析ツール Extra-P が開発されている [5]。Extra-P はアプリケーションを小規模実行した際の結果から各関数のスケーラビリティモデルを構築し、構築したモデルを用いて同アプリケーションを大規模実行した際の各関数の実行時間を予測する。この方法により、大規模実行に必要な計算資源と時間を節約しつつ、アプリケーションを大規模実行した際の関数の実行時間を見積もることができる。

一方、アプリケーションのプロファイルには、関数の実行時間のほかにも関数のコール回数やコールグラフなどさまざまな情報が含まれている。これらの情報は解析対象のアプリケーションの挙動を理解し性能をチューニングする上で有益であるが、Extra-P は関数の実行時間以外の情報は提供しない。そのため、Extra-P では並列アプリケーションの細かな挙動を理解することができず、したがって、Extra-P から得られる情報のみを用いてアプリケーションの性能チューニングを行う場合はチューニング方法が限定的にならざるを得ない。

本論文では、MPI アプリケーションの関数コール回数を予測する手法を提案する。提案手法では、アプリケーションを小規模実行した際の関数コール回数から同アプリケーションを大規模実行した際の関数コール回数を予測する。アプリケーションのプロファイリングを行うケースを小規模実行に限定することによって、プロファイリングに必要なコスト（時間と計算資源）を削減する。提案手法によって並列アプリケーションの性能チューニングに利用可能な情報を増やし、並列アプリケーション開発者の生産性向上に資する。

2. 研究背景

2.1 プロファイリング

並列アプリケーションのプロファイリングは性能解析やチューニングを目的として高性能計算分野で広く行われている。アプリケーションのプロファイルには、実行された関数の名称、関数のコール回数、 関数の実行時間などの情報が含まれる。さらには、CPU のキャッシュメモリのア

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

ⓒ 2021 Information Processing Society of Japan

クセス数やミス数などのハードウェアカウンタ値を取得することも可能である [4]。

プロファイルを取得する際は、まず解析対象のアプリケーションのライブラリリンク時もしくはコンパイル時に計測用のコードを挿入し、その後、解析対象のシステム上でアプリケーションを実行する。計測用のコードを挿入するためのツールとして、TAU や Score-P などのツールがすでに開発されており、これらのツールを用いてアプリケーションにコードを挿入する。

後述する Extra-P とは異なり、プロファイリングによって取得された情報は実測値である。その一方で、プロファイルを取得するにあたっては実際に解析対象のアプリケーションを実行する必要があるため、プロファイルの取得に必要なコスト（時間と計算資源）は相対的に大きい。情報を取得する関数を限定するなどのプロファイリングコストを削減する技術も開発されているが [3][4]、解析対象のアプリケーションを解析対象の規模で実行する必要がある点に変わりはない。以上まとめると、プロファイリングの利点は詳細で正確なプロファイルを取得できることであるが、欠点は取得のためのコストが大きいことである。

2.2 Extra-P

並列アプリケーションの性能解析ツールである Extra-P[5] は、アプリケーションを小規模実行した際の実行結果から実行プロセス数や問題サイズをパラメータとする実行時間モデルを関数ごとに構築し、それらを用いて同アプリケーションを大規模実行した際の実行時間を予測する。モデルを構築するプロセスでは、関数ごとに計 21 個のモデルに対して入力データを用いてフィッティングを行い、自由度調整済決定計数が最大のモデルを同関数のスケーラビリティモデルとして出力する [2]。

Extra-P はアプリケーションを大規模実行した際の実行時間の見積もりを低コストで行うことができるが、プロファイリングとは異なり、出力する実行時間は実測値ではなく、モデルによって予測された近似的な値である。また、Extra-P は関数の実行時間を予測できるが、関数コール回数を予測することはできない。

2.3 関数コール回数と性能チューニング

関数コールには性能オーバーヘッドが存在する。そのため、多くの回数コールされる関数が性能上のボトルネックとなっている場合には、関数のコール回数を減らすことでアプリケーションの性能を改善できる可能性がある。関数のコール回数はアプリケーションのソースコードの解析によって求めることができる場合もあるが、アプリケーションを実際に実行してみなければわからない場合もある。例えば、関数が IF 文や収束計算などの不定回数ループの中でコールされる場合には、実際に入力データを与えて実行しなければ、その関数のコール回数はわからない。

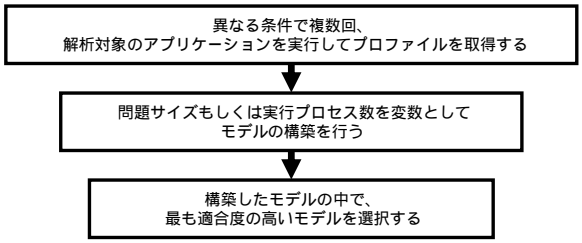

図 1: 提案手法

コール回数の多い関数に対する性能チューニング技術の 1 つであるインライン展開では、関数内で実行される命令列をコーラー側に展開することによって関数コールそのものをプログラム中から削除する。関数のインライン展開によってコードサイズは増加するが、関数コールによる性能オーバヘッドは削減されるため、結果としてアプリケーション性能が向上する可能性がある。そのため、コール回数の多い関数を特定することはアプリケーションの性能チューニングにおいて重要である。

3. 提案手法

3.1 概要

本稿では、アプリケーションを小規模実行した際の関数コール回数から、同アプリケーションを大規模実行した際の関数コール回数を予測する手法を提案する。提案手法では実際にアプリケーションのプロファイルを取得するケースを小規模実行に限定することにより、大規模実行時の関数コール回数の取得に必要なコストを軽減する。提案手法が出力する情報と Extra-P が出力する情報を組み合わせることによって性能チューニングの幅を広げ、アプリケーション開発者の生産性の向上に資する。

提案手法は次のような流れで行われる（図 1）。まず最初に、アプリケーションのプロファイリングを異なる条件下（コア数、問題サイズ）で複数回行い、各条件下における 1 プロセスあたりの各関数のコール回数を取得する。次に、異なる条件で取得した各関数のコール回数の情報を用いて次節で述べるモデルに対してフィッティングを行い、関数ごとにコール回数のモデルをいくつか作成する。最後に、作成したモデルの中から最も適合度の高いモデルを選択し、それぞれの関数の 1 プロセスあたりのコール回数を予測する。

3.2 関数コール回数のモデル

関数コール回数のモデルとして「線形モデル」「対数モデル」「反比例モデル」「線形飽和モデル」の 4 種類を用いる。以下、それぞれのモデルについて詳しく述べる。なお、以下の説明では、変数 x は数値化された問題サイズもしくは

コア数を表し、変数 y は関数コール回数を表す。

線形モデル

関数コール回数が問題サイズまたはコア数に比例すると仮定する。モデル式は式 (1) である。

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

対数モデル

関数コール回数が問題サイズまたはコア数に対して対数的に変化すると仮定する。モデル式は式 (2) である。

$$y = a \log_{10} x + b \tag{2}$$

反比例モデル

関数コール回数が問題サイズまたはコア数に反比例すると仮定する。モデル式は式 (3) である。

$$y = \frac{a}{x} + b \tag{3}$$

線形飽和モデル

問題サイズまたはコア数がある値 x' よりも小さい場合は、関数コール回数が問題サイズまたはコア数に比例すると仮定する。また、問題サイズまたはコア数が x' よりも大きい場合は一定の値 y' になると仮定する。モデル式は式 (4) である。

$$y = \begin{cases} ax + b & (x < x') \\ ax' + b & (x \geq x') \end{cases} \tag{4}$$

関数コール回数は一般に問題サイズとコア数の両方に依存するため、モデルには両方のパラメータが含まれていることが望ましい。それに対して本稿では、問題サイズまたはコア数のどちらか 1 つのみをパラメータに含むモデルを用いて関数コール回数予測の初期検討を行う。問題サイズとコア数の両方をパラメータに含むモデルの開発と評価は今後の課題である。

4. 評価

4.1 評価目的

少ないコア数で取得した関数コール回数から多いコア数で取得される関数コール回数を予測する手法、および、小さい問題サイズで取得した関数コール回数から大きい問題サイズで取得される関数コール回数を予測する手法の 2 つを評価する。

評価では、プロファイリングにより実測した関数コール回数と提案手法により予測した関数コール回数を比較し、提案手法の予測精度を確認する。また、プロファイリングと提案手法それぞれに対して、関数コール回数の取得に必要なコストも評価する。

表 1: TSUBAME3.0 のシステム構成

総理論演算性能	12.15PFlops（倍精度）
総ノード数	540
総主記憶容量	138,240GB
ネットワークトポロジ	フルバイセクション・ファットツリー

表 2: TSUBAME3.0 のノード構成

CPU	プロセッサ名 物理（論理コア数） 動作周波数	Intel Xeon E5-2680 V4 14(28) 2.40GHz
Memory	容量 メモリ帯域幅	256GB 153.6GB/s
GPU	プロセッサ名	NVIDIA Tesla P100

4.2 評価環境

実験に使用するシステム、ツール、ベンチマークプログラムは次の通りである。

TSUBAME3.0

コンピュータシステムには東京工業大学学術国際情報センターのスーパーコンピュータ TSUBAME3.0 を使用する。TSUBAME3.0 は 540 台の計算ノードにより構成され、1 ノード当たり CPU(Intel Xeon E5-2680 V4) が 2 基搭載されている。TSUBAME3.0 のシステム構成を表 1、ノード構成を表 2 に示す。

TAU

アプリケーションのプロファイリングには TAU (Tuning and Analysis Utilities) を使用する。TAU は、C 言語や Python などで記述された並列プログラムの性能分析を行うために開発された、包括的なプロファイリングツールキットである [4]。

NAS Parallel Benchmarks

ベンチマークプログラムには NAS Parallel Benchmarks (NPB) を使用する。NPB は、表 3 に示す 5 つの計算カーネルと 3 つの疑似アプリケーションで構成されている [1]。NPB には複数の問題サイズが用意されているが、本稿では A, B, C, D の 4 種類を実験に使用する。なお、問題サイズ B は問題サイズ A の 4 倍、問題サイズ C は問題サイズ B の 4 倍、問題サイズ D は問題サイズ C の 16 倍の規模となっている。

4.3 評価項目と評価方法

4.3.1 モデルの適合度

3.2 節で述べた 4 つのモデルに対してフィッティングを行い、フィッティングに使用したデータに対する適合度を評価する。評価には平均絶対パーセント誤差 (MAPE) を用いる。平均絶対パーセント誤差はモデルが出力する値 (F_t) と観測データ値 (A_t) の絶対誤差率の平均であり、式 5 で表すことができる。

表 3: NAS Parallel Benchmarks

プログラム名	詳細
IS	整数ソートとランダムメモリアクセスをする。
EP	各プロセッサがほぼ完全に独立して処理できる並列処理をする。
CG	連立一次方程式を解くためのアルゴリズムである共役勾配法、不規則なメモリアクセスとノード間通信をする。
MG	メモリ集約型で連続したメッシュ状でのマルチグリッド計算、長距離通信と短距離通信をする。
FT	離散 3 次元フーリエ変換と全対全通信をする。
BT	ブロック 3 重対角連立 1 次方程式のソルバ
SP	スカラー 5 重対角連立 1 次方程式のソルバ
LU	ガウス・ザイデル法による LU 分解のソルバ

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

4.3.2 予測精度

提案手法を用いて各関数のコール回数予測を行い、その予測精度を評価する。予測精度の評価には以下の式で表される絶対誤差率を使用する。

$$\text{絶対誤差率 [\%]} = \frac{|\text{予測値} - \text{実測値}|}{\text{実測値}} \times 100$$

関数コール回数の予測は次の 2 つのケースに対して行う。1 つ目は、問題サイズは変えずに実行に使用するコア数のみを変更することにより、プログラムを実行する規模を大きくした場合（コア数に対する予測）である。具体的には、1∼128 コアで実行した際の関数コール回数を用いて 256 コアで実行した際の関数コール回数を予測する。2 つ目は、実行に使用するコア数は変えずに問題サイズのみを変更することにより、実行規模を大きくした場合（問題サイズに対する予測）である。具体的には、問題サイズ A∼C の関数コール回数を用いて問題サイズ D の関数コール回数を予測する。

4.3.3 コスト

本稿では、関数コール回数の取得に必要なコストを (プロファイリングに使用するコア数) × (コアの使用時間) 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はポイント制を採用しており、エンドユーザは使用コア数とコアの使用時間の積に応じてポイントを消費するためである。

提案手法の場合、プロファイリング以外にモデルのフィッティングや予測などの処理が必要となる。ただし、これらの処理はアプリケーションのプロファイリングに比べて非常に軽微であり一般的な PC で実行可能なため、本稿では無視できるものとした。

4.4 評価結果および考察

提案手法の評価結果と結果に対する考察を述べる。なお、以下では、すべての規模（問題サイズとコア数）で実行される関数のみを評価対象とした。

4.4.1 モデルの適合度

問題サイズ、コア数に対する予測において採用されたモデルの割合とその適合度（MAPE の最小値と最大値）をそれぞれ表 4、表 5 に示す。表の各行はプログラムを表しており、列はモデルを表す。表内の割合は、当該プログラムの関数コール回数予測において当該モデルが採用された割合を示している。また、カッコ内の数値は、左が採用されたモデルの MAPE 値の最小値を、右が最大値を表す。NoData は、当該モデルが当該プログラムで採用されなかったことを示す。

表 4 は、実行に使用するコア数を 64 とし、問題サイズ A, B, C の実行結果を用いてモデルのフィッティングを行った際の結果を表している。また表 5 は、実行に使用する問題サイズを B とし、BT と SP についてはコア数 1, 4, 16, 64 の実行結果を、それ以外のプログラムについてはコア数 1, 2, 4, 8, 16, 32, 64 の実行結果を用いてモデルのフィッティングを行った際の結果を表している。なお、BT, SP とそれ以外のプログラムとで使用コア数が異なるのは、BT と SP はプログラムの制約によりプロセス数が平方数の場合しか実行できないためである。

表より、すべてのプログラムにおいて最も採用されたモデルは線形モデルであった。また、線形モデルを採用した場合の MAPE の最小値と最大値が 0.0 を示しているのは、1 プロセスあたりの関数コール回数が固定されており、コア数にも問題サイズにも依存しない関数が多かったためと考えられる。

問題サイズに対する予測では、線形飽和モデルが採用された場合の MAPE 値の最大値と最小値がともに 0.0 になっている。そのため、問題サイズに対する予測においては、線形飽和モデルは問題サイズの変更に対する関数コール回数の変化を良く表現できていると言える。一方で、コア数に対する予測では、線形飽和モデルが採用された場合でも MAPE 値の最大値が大きいケースもある。このような関数に対しては、今回利用した 4 つのモデルとは異なる別のモデルを検討する必要がある。

4.4.2 予測精度

フィッティングに使用するデータを変更した場合におけ

表 4: 問題サイズに対する予測において採用されたモデルとその適合度（コア数 64）

	採用される割合 (MAPE の最大値 [%]，MAPE の最小値 [%]) [%]			
プログラム名	線形モデル	対数モデル	反比例モデル	線形飽和モデル
BT	99(0.0, 0.0)	1(0.0, 0.0)	0(NoData)	0(NoData)
CG	69(0.0, 0.0)	13(21.2, 30.1)	0(NoData)	18(0.0, 0.0)
EP	100(0.0, 0.0)	0(NoData)	0(NoData)	0(NoData)
FT	57(0.0, 0.0)	6(22.5, 22.6)	0(NoData)	37(0.0, 0.0)
IS	100(0.0, 0.0)	0(NoData)	0(NoData)	0(NoData)
LU	81(0.0, 0.0)	19(0.1, 0.4)	0(NoData)	0(NoData)
MG	48(0.0, 0.0)	40(27.0, 28.2)	3(1.3, 1.3)	9(0.0, 0.0)
SP	98(0.0, 0.0)	2(0.0, 0.0)	0(NoData)	0(NoData)

表 5: コア数に対する予測において採用されたモデルとその適合度（問題サイズ B）

	採用される割合 (MAPE の最大値 [%]，MAPE の最小値 [%]) [%]			
プログラム名	線形モデル	対数モデル	反比例モデル	線形飽和モデル
BT	78(0.0, 0.0)	22(0.0, 0.5)	0(NoData)	0(NoData)
CG	69(0.0, 0.0)	0(NoData)	0(NoData)	31(0.0, 12.2)
EP	100(0.0, 0.0)	0(NoData)	0(NoData)	0(NoData)
FT	62(0.0, 0.0)	0(NoData)	0(NoData)	38(0.0, 88.7)
IS	82(0.0, 0.0)	14(14.0, 14.0)	0(NoData)	4(88.7, 88.7)
LU	77(0.0, 0.0)	21(0.0, 17.2)	2(0.0, 0.0)	0(NoData)
MG	72(0.0, 0.0)	0(NoData)	14(91.0, 91.0)	14(19.4, 19.4)
SP	79(0.0, 0.0)	21(0.0, 0.5)	0(NoData)	0(NoData)

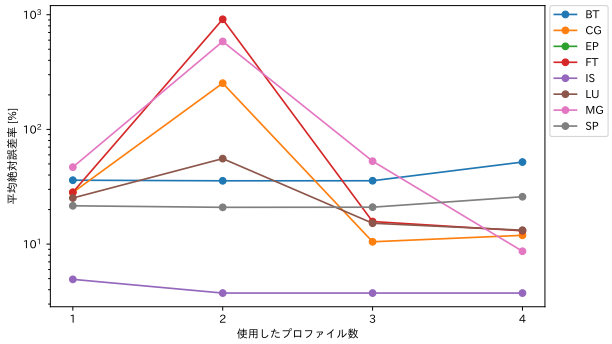

図 2: 問題サイズに対する予測精度（コア数 64）

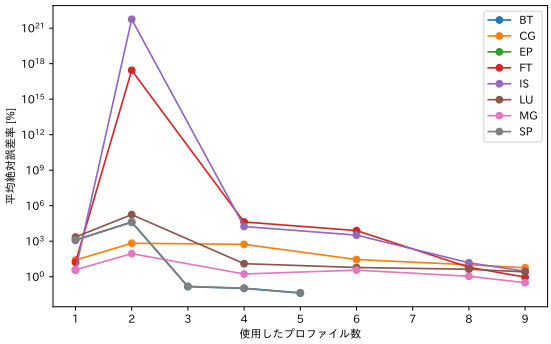

図 3: コア数に対する予測精度（問題サイズ B）

る提案手法の予測精度を図 2 と 図 3 に示す。 図 2 はコア数 64 で問題サイズに対する予測を行った場合、図 3 は問題サイズ B でコア数に対する予測を行った場合である。グラフの縦軸は平均絶対誤差率を表しており、横軸はフィッティングに使用したプロファイル数を表す。なお、プロファイル数とプロファイルの関係は表 6~8 のようになっている。

図 2 より、問題サイズに対する予測では、多くのプログラムにおいてフィッティングに使用するプロファイル数を 1 から 2 に増やすと予測精度は一旦悪化する。ただし、そこからさらにプロファイル数を増やすと予測精度は減少に転じ、プロファイル数 3 の時に予測精度が最も高くなった。

また、プロファイル数が 4 の時に最も精度が高くなると予想していたが、いくつかのプログラムでは予想に反してプロファイル数を 3 から 4 に増やした際に予測精度が悪化した。これは、3.2 節で述べた 4 つのモデルでは説明できない挙動を示す関数が存在したためである。そのような関数として、問題サイズが小さい時はコール回数が一定だがある問題サイズを超えるとコール回数が上昇に転じる関数、問題サイズの増加にともなってコール回数は増加するものの増加の方法が一定ではない関数などがあった。

図 3 より、コア数に対する予測では、すべてのプログラムにおいてフィッティングに使用するプロファイル数が 2 の時に提案手法の予測精度は最も悪くなる。この理由として、フィッティングに用いたデータが 2 種類の場合には基本的にどの関数の予測においても線形モデルが最適なモデ

表 6: 問題サイズに対する予測に使用したプロファイル

プロファイル数	使用されるプロファイルの問題サイズ
4	A, B, C, D
3	A, B, C
2	A, B
1	A

表 7: コア数に対する予測に使用したプロファイル（BT と SP）

プロファイル数	使用されるプロファイルのコア数
5	1, 4, 16, 64, 256
4	1, 4, 16, 64
3	1, 4, 16
2	1, 4
1	1

表 8: コア数に対する予測に使用したプロファイル（BT と SP 以外）

プロファイル数	使用されるプロファイルのコア数
9	1, 2, 4, 8, 16, 32, 64, 128, 256
8	1, 2, 4, 8, 16, 32, 64, 128
6	1, 2, 4, 8, 16, 32
4	1, 2, 4, 8
2	1, 2
1	1

ルとなることが考えられる。提案手法ではモデルの選択において MAPE 値が同じ場合は線形モデルを優先的に選択するようになっている。そのため、関数コール回数のモデルとして線形モデルが最適ではない関数がプログラム内に多いと、線形モデルによる予測値と実測値が大幅にずれてしまい、このような結果になったのだと考える。線形モデル以外のモデルはフィッティングに用いるデータが 3 種類以上ないと基本的には選択されないため、フィッティングに用いるデータは最低 3 種類以上必要である。

4.4.3 コスト

フィッティングに使用するデータを変更した場合における提案手法の予測コストを図 4 と図 5 に示す。図 4 はコア数 64 で問題サイズに対する予測を行った場合、図 5 は問題サイズ B でコア数に対する予測を行った場合である。グラフの縦軸は相対コストを表しており、提案手法による予測コストを（プロファイリングによる）実測コストで割った値である。したがって、相対コスト 1.0 は提案手法とプロファイリングのコストが同じであり、0.0 に近づくほど提案手法が低コストであることを表す。グラフの横軸はフィッティングに使用するプロファイル数である。

図 4 より、問題サイズに対する予測においては、フィッティングに使用するプロファイルが 3 以下の場合に提案手法はプロファイリングよりも低コストになる。使用プロファイル数が 3 の場合の平均絶対誤差率は、図 2 より、8 つのプログラム内の 5 つにおいて 20%未満であった。

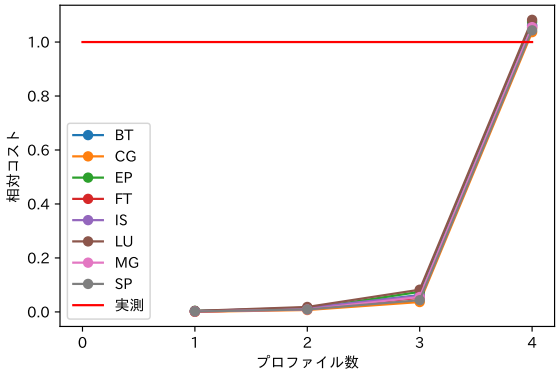

図 4: 問題サイズに対する予測のコスト（コア数 64）

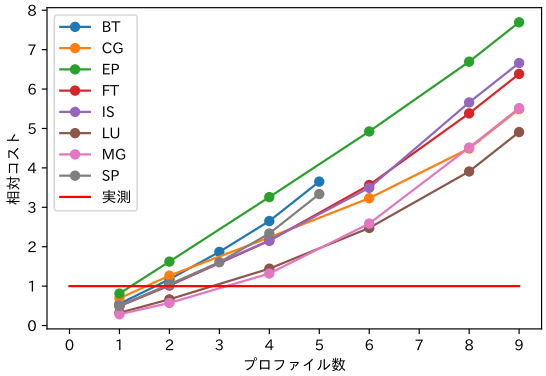

図 5: コア数に対する予測のコスト（クラス B）

図 5 より、コア数に対する予測においては、ほとんどのプログラムにおいてフィッティングにプロファイルを 2 つ以上使用すると提案手法のコストがプロファイリングのコストを上回る結果となった。フィッティングに使用するプロファイル数が 1 つの時は提案手法のコストがプロファイリングのコストを下回るものの、図 3 より、プロファイル数が 1 つの時の平均絶対誤差率はあまり低くない。

前項で述べたように、提案手法はフィッティングに使用するプロファイル数が 3 つ以上ないと線形モデル以外を選択しない。そのため、提案手法でフィッティングに使用するプロファイル数を 1 つにしてしまうと、関数コール回数のモデルとして線形モデル以外が最適な関数を多く含むプログラムに対して効果がない。

4.4.4 コストと平均絶対誤差率

最後に提案手法におけるコストと平均絶対誤差率の関係をまとめる。コア数 64 で問題サイズに対する予測を行った時の結果を表 9 に、問題サイズ B でコア数に対する予測を行った時の結果を表 10 に示す。表 9 で使用したプロファイル数は 3 である。表 10 で使用したプロファイル数は、BT と SP については 4 であり、BT と SP 以外につい

表 9: 問題サイズに対する予測の精度とコスト（コア数 64）

プログラム名	平均絶対誤差率 [%]	相対コスト [%]
BT	35.6	6.225
CG	10.4	3.667
EP	0.0	7.433
FT	15.7	5.240
IS	3.7	6.035
LU	15.1	8.229
MG	52.8	5.525
SP	20.9	4.341
平均	19.3	5.837

表 10: コア数に対する予測の精度とコスト（問題サイズ B）

プログラム名	平均絶対誤差率 [%]	相対コスト [%]
BT	0.1	265.1
CG	10.2	449.1
EP	0.0	669.4
FT	6.3	538.2
IS	15.0	566.0
LU	4.2	390.8
MG	1.0	451.5
SP	0.1	233.5
平均	4.6	445.5

ては 8 である。

問題サイズに対する予測では、表 9 より、提案手法は全プログラムの平均で平均絶対誤差率 19.32%の予測を実現できた。また、この時の相対コストは全プログラムの平均で 5.83%であった。

一方、コア数に対する予測では、表 10 より、提案手法は全プログラムの平均で平均絶対誤差率 4.65%の予測を実現できた。しかし、この時の相対コストは全プログラムの平均で 445.5%となってしまった。これは、提案手法ではフィッティングに使用するプロファイルをアプリケーションを少ないコア数で実行することによって取得するが、各プロファイルの取得にはアプリケーションを多いコア数で実行した場合とほぼ同等のコストが必要なためである。一般に、アプリケーションの問題サイズが同じならばコア数の少ない方がコア数の多い場合と比べて実行時間は長くなるため、実行に使用するコア数とコアの使用時間の積はほぼ一定となる。

5. 結論

5.1 まとめ

スーパーコンピュータの規模の増大にともないその上で実行されるアプリケーションも大規模化/複雑化しており、超大規模な並列アプリケーションのプロファイリングは困難となっている。この問題に対して Extra-P が開発されているが、Extra-P ではアプリケーションの関数単位の実行時間は予測できるものの、関数コール回数を予測することはできない。

そこで本論文では、小さい問題サイズもしくは少ないコア数で解析対象のアプリケーションを複数回実行してプロファイルを取得し、得られたプロファイルを用いて各関数のコール回数モデルを構築し、構築したモデルを用いて大きい問題サイズもしくは多いコア数で同アプリケーションを実行した際の関数コール回数を予測する手法を提案した。

少ないコア数で取得した関数コール回数から多いコア数の関数コール回数を予測した場合は、提案手法は平均絶対誤差率 4.65%の精度で予測できたものの、提案手法によるプロファイルの取得コストは従来の約 4.5 倍となってしまった。一方で、小さい問題サイズで取得した関数コール回数から大きい問題サイズの関数コール回数を予測した場合は、提案手法は平均絶対誤差率 19.3%の精度で予測することができ、プロファイルの取得コストを約 5%にまで削減できた。

5.2 今後の展望

本稿では問題サイズとコア数のどちらか一方をパラメータとするモデルの提案と評価を行ったが、今後は問題サイズとコア数の両方をパラメータとするモデルの開発と評価を行う予定である。また、今回の実験によって 3.2 節で述べた 4 つモデルには適合しない挙動を示す関数が存在することもわかったため、今後はモデルの種類を増やすことも検討する。さらには、より実用的なアプリケーションを用いて提案手法の評価を行う予定である。

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] Bailey, D. H.: The NAS Parallel Benchmarks, *RNR-94-007* (1994).
[2] Calotoiu, A., Hoefler, T., Schulz, M., Shudler, S. and Wolf, F.: Insightful Automatic Performance Modeling, `https://apps.fz-juelich.de/scalasca/releases/extra-p/slides/InsightfulAutomaticPerformanceModelingTutorialPartI.pdf`.
[3] Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A. D., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B. and Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir (2011).
[4] Performance Research Lab: TAU, `https://www.cs.uoregon.edu/research/tau/home.php`.
[5] Technical University of Darmstadt: Extra-P, `https://www.scalasca.org/scalasca/software/extra-p/download.html`.

ⓒ 2021 Information Processing Society of Japan