Vol.2018-HPC-165 No.28
2018/8/1

機械学習による計算機トレースの自動生成

土川 稔生[1,a)]　大山 洋介[1]　野村 哲弘[1]　松岡 聡[2,1]

概要：計算機トレースとはプログラムを実行した際の様々な情報を記録したものである．これを用いてプログラムのボトルネックなどを発見し，解決することでプログラムの性能向上に繋がる．しかし，計算機トレースを取得するにはプログラムを実際に実行する必要があり，プログラムのサイズによっては多大な時間を要する．本研究では，回帰木などの機械学習の手法を用いることによって，プログラムの特徴量を入力とし，計算機トレースの一種である L1 キャッシュミスなどの計算機トレースを部分的に生成することに成功した．Rodinia ベンチマークの Pathfinder プログラムに対して，プログラムの問題サイズを変更した際のキャッシュミス数とその累計値をそれぞれ予測誤差 19.57%，7.30%の精度で生成することに成功した．

1. はじめに

計算機を使ってプログラムを実行する際，期待した通りの性能が出ない場合がある．そうした場合，プログラムの性能が出ない原因を突き止めることが必要となる．そこで計算機トレースを取得し，プログラム内のどの部分で時間がかかっているのか，どこの処理がボトルネックになっているのか，などの情報を確認しプログラムを改善していくことが必要となる．計算機トレースはプログラム実行の様々な情報を記録したものである．しかし，計算機トレースはプログラムを実際に実行しないと取得することができないため，問題サイズが大きいプログラムの実行の際，一回の計算機トレースを取得するためには多くの時間がかかる．そのため，プログラムサイズなどのパラメータを変更し，複数回計算機トレースを取得することは困難になってくる．

本研究では，少数回のプログラム実行を通じて得たプログラムの特徴量から L1 キャッシュミスなどの計算機トレースを部分的に生成することに成功した．Rodinia ベンチマーク [1] の Pathfinder プログラムを TSUBAME3.0 を用いて 1 スレッドで実行した際，プログラムの問題サイズを変更したときに相対誤差 19.57%の精度で L1 キャッシュミス数のトレースを生成することに成功した．また，プログラム全体のキャッシュミス累計値を相対誤差 7.30%の精度で生成することに成功した．

[1] 東京工業大学
[2] 理化学研究所 計算科学研究センター
[a)] tsuchikawa.t.aa@m.titech.ac.jp

2. 背景

2.1 計算機トレースについて

計算機トレースはプログラムを実行した際の通信の情報やメモリ使用の情報などを記録したものである．計算機トレースの種類は複数あり，スレッド間，プロセス間の通信や同期の様子を記録したものや，メモリやキャッシュなどのハードウェアカウンターの様子，I/O の様子や，CPU，GPU の様子を記録したものなどがある．

今回の評価では，計算機トレースは Score-P [2]，PAPI [3] を用いて取得し，Vampir [4] を用いて可視化した．Score-P は計算機トレースを取得する中心的な要素であり，プログラム実行時の通信，同期の様子などを記録することができる．また Score-P は外部ツールで計測した結果を取り込むことが可能であり，その中で PAPI はキャッシュミス数などのハードウェアカウンターを中心に収集するツールで，Vampir はグラフなどで計算機トレースを可視化するツールである．図 1 は Vampir で可視化した計算機トレースの例である．

3. 提案手法

計算機トレースで記録できる計測項目は多岐にわたるため，L1 キャッシュミスなどの 1 つの計算機トレースに着目して，機械学習による生成を行う．Score-P の出力の標準の出力形式である Open Trace Format 2(OTF2) 形式 [8] のファイルから得られる EVENT 情報を入力データとした．出力データは otf2 ファイルから得られる L1 キャッシュのミス数などの計算機トレースとなる．図 2 に OTF2 ファイルに記録されるエントリの例を示す．図 2 で青色に着色した箇所はプログラムの関数や OpenMP で並列実行

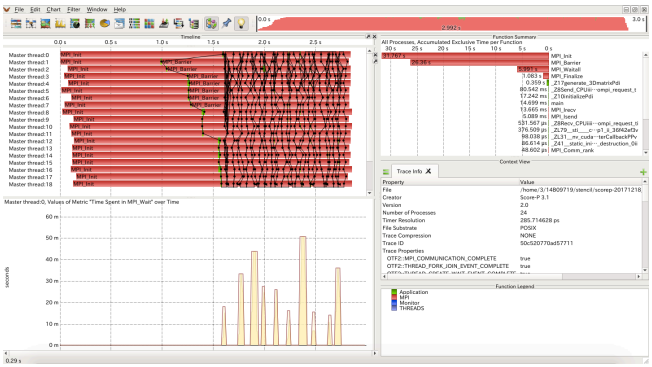


図 1: 可視化された計算機トレースの例

される領域への出入りの情報を表している．オレンジ文字は記録されている各イベントの発生時刻を UNIX 時間で表し，赤文字は各計測項目 (図中の例では L1 キャッシュミス数) の累計値を表している．青文字の部分は 1 回のプログラム実行から構成することができる．一方，オレンジ文字と赤文字は，ツールを用いてトレースとして取得することが必要であり，サイズを変更する度にプログラムを実行することがて必要となる．そこで本研究では，青文字で表されたプログラムの特徴量である METRIC の情報とベンチマークの中から対象としたプログラムの問題サイズを入力として，各 EVENT における L1 キャッシュミス数とプログラム全体のキャッシュミス数の累計値を機械学習の手法により生成する．

3.1 入力データの整形

機械学習モデルの教師データを作成するにあたり、次のように METRIC の情報とプログラムの入力サイズに対して前処理を行った．まず METRIC に関しては METRIC がもつ関数の情報に着目し、プログラム全体に存在する関数の数の次元の配列を作り，その関数ごとに関数の ENTER~LEAVE に入っている場合は 1 をそれ以外の場合を 0 とした配列とした．図 2 の前処理の手順を示す．

Listing 1: Pathfinder のプログラム (一部)

```
1  for (int t = 0; t < rows-1; t++) {
2      temp = src;
3      src = dst;
4      dst = temp;
5      #pragma omp parallel for private(min) //
         ①,②,⑤
6      for(int n = 0; n < cols; n++){
7          min = src[n];
8          if (n > 0)
9              min = MIN(min, src[n-1]);
10         if (n < cols-1)
11             min = MIN(min, src[n+1]);
12         dst[n] = wall[t+1][n]+min;
13     } //③,④
14 }
```

```
① ENTER   23047558475513548 Region: "!$omp parallel @pathfinder.cpp:98" <15>
① METRIC  23047558475518604 Metric: 2, 1 Values: ("PAPI_L1_DCM" <9>; UINT64; 1052585)
② ENTER   23047558475518604 Region: "!$omp for @pathfinder.cpp:98" <16>
② METRIC  23047558480777180 Metric: 2, 1 Values: ("PAPI_L1_DCM" <9>; UINT64; 1066146)
③ ENTER   23047558480777180 Region: "!$omp implicit barrier @pathfinder.cpp:106" <17>
③ METRIC  23047558480804248 Metric: 2, 1 Values: ("PAPI_L1_DCM" <9>; UINT64; 1066387)
③ LEAVE   23047558480804248 Region: "!$omp implicit barrier @pathfinder.cpp:106" <17>
④ METRIC  23047558480807724 Metric: 2, 1 Values: ("PAPI_L1_DCM" <9>; UINT64; 1066393)
④ LEAVE   23047558480807724 Region: "!$omp for @pathfinder.cpp:98" <16>
⑤ METRIC  23047558480809296 Metric: 2, 1 Values: ("PAPI_L1_DCM" <9>; UINT64; 1066395)
```

図 2: 取得された計算機トレースの例

図 2 のトレースを見ると，①から⑤と番号が振ってある 5 つの METRIC が存在する．これらは Pathfinder プログラムの中の①から⑤と対応している．まず①に注目すると，上に ENTER があり，Region: “!$omp parallel @pathfinder.cpp:98”<15>と書いてある．つまり，①の METRIC は pathfindef.cpp のプログラムの 98 行目にある openmp の parallel の範囲（Region）に含まれていることを意味する．<15>となっている箇所は計測対象の領域を識別するための内部的な ID であり，実行のたびに変化する．トレースの再構成および利用においてこの番号自体を再現する必要性はないため，以降は無視することとする．②から⑤も同様に ENTER と LEAVE を確認し，各イベントに発生時刻において各 region に入っているかどうかを，region ごとに 0/1 であらわしたベクトルで表現する．この例では region は 3 種類あるため，ベクトルは 3 次元のベクトルとなり，各 METRIC 文において，

$$[1,0,0][1,1,0][1,1,1][1,1,0][1,0,0]$$

と表される．

また出力としては各 METRIC の L1 キャッシュミス数を出力したいが，図 2 の赤文字で表される L1 キャッシュミス数は初期値が不定値になっている累積値となっているため，入力の METRIC の L1 キャッシュミス数から一つ前の METRIC のキャッシュミス数を引いたものを出力とする．図 2 の例では $[13561, 241, 6, 2]$ となる．また，一番初めの要素は初期値が不定値のため省く．

3.2 時系列の考慮

また，図 2 のオレンジ文字の計測時刻をみると上から時系列順のトレースになっていることがわかる．そこで時系列の情報を考慮し，予測精度を高めるために直近の入力ベクトルを複数個連結させたものを入力とした．例えば，3.1 章で求めた例の 5 つの配列に対して，時系列を考慮し，2 個ずつ直近のベクトルを結合すると以下のようになる．初めの 2 つのベクトルについては前に結合すべき 2 つのベクトルがないため削除したが，時系列情報の一番最初のほうのベクトルであるため削除しても実用上問題ないと判断した．

$$[1,1,1,1,1,0,1,0,0][1,1,0,1,1,1,1,1,0][1,0,0,1,1,0,1,1,1]$$

またこの時，出力は以下のようになる．

$$[241, 6, 2]$$

表 1: scikit-learn との対応

回帰分析モデル	scikit-learn のライブラリ名
線形回帰	LinearRegression
SGD	SGDRegressor
Ridge 回帰	Ridge
Lasso 回帰	Lasso
ElasticNet	ElasticNet
SVR	LinearRegression
回帰木	DecisionTreeRegression
ランダムフォレスト	RandomForestRegression
Extremely Randomized Trees	ExtraTreeRegression
回帰木+勾配ブースト	GradientBoostingRegression
回帰木+AdaBoost	AdaboostRegression
回帰木+バギング	BaggingRegression

3.3 プログラム情報の入力と圧縮

プログラムごとの特徴量を入力データに含めるため，データサイズや計算パターンに影響を与えているプログラムの引数の情報を付加した．また同じ入力に対して，同じ出力をするデータがあると，同じ訓練データを機械学習に学習させることになり，学習に偏りが生まれうまく学習できないため，同じ訓練データは 1 個以外のデータを除外して 1 つのベクトルにした．

3.4 使用した回帰モデル

分析には scikit-learn [8] を使用し，回帰モデルは線形モデル・決定木やこれらのモデルのアンサンブル回帰を用いた．また scikit-learn のライブラリとの対応を表 1 に示した．

4. 評価

4.1 実行環境

実装の実行環境として，東京工業大学学術国際情報センターの TSUBAME3.0 を使用した．表 2 に TSUBAME3.0 の実行時のハードウェア構成とソフトウェア構成を示す．

表 2: 実行環境 (TSUBAME3.0)

CPU	Intel(R) Xeon(R) CPU E5-2680 v4 × 2
周波数	2.4 GHz
コア数	14
スレッド数	28
L1 キャッシュ	32 KB
L2 キャッシュ	256 KB
L3 キャッシュ	35840 KB
メモリ	256GB
メモリバンド幅	76.8 GB/s
インターコネクト	Intel Omni-Path HFI x4
バンド幅	100Gbps
バス・インターフェイス	PCI Express Gen3 x 16
トポロジー	Fat tree
C コンパイラ	gcc 4.8.5
Score-P	scorep 3.1
cube	cube 4.3.5
vampir	vampir 9.4
papi	papi 5.5.1
OpenMP	OpenMP 3.1

4.2 実験結果

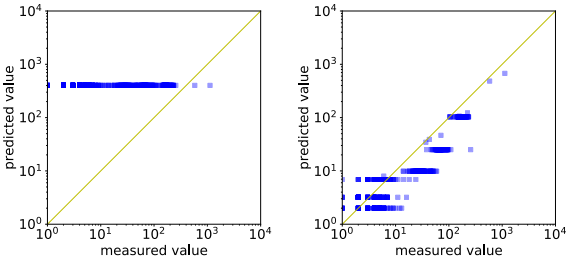
実験は Rodinia ベンチマークの Pathfinder プログラムを用いた．このプログラムは 2 次元グリッドデータの経路を探索するプログラムである．またこのプログラムの特徴量として，グリッドの横 (row) と縦 (col) のサイズの引数があるため，その 2 つのデータを入力データに含めた．訓練データには入力サイズとして (row,col)=(100,100)，(1000,1000)，(10000,1000)，(100000,100) を用い，テストデータとして (row,col)=(1000,100) を用いた．また，Pathfinder に出現する Region の種類は 9 つであったので，9 次元のベクトルに 2 次元のデータサイズを連結したものが一つの入力データとなる．

4.2.1 連結数 4 の時の予測結果

はじめに，今回作成したデータセットに対して適切なモデルを検討するために，連結数を 4 として L1 キャッシュミス数を予測した．また 4 つを連結させるため，$9 \times (4+1)+2 = 47$ 次元のベクトルが一つの入力データとなる．その結果を横軸を実測値，縦軸を予測値として両対数をとったグラフとして図 3 に示す．図の対角線上に点が近ければ近いほど実測値と予測値が近いことが言え，良いモデルと言える．表 1 の手法で試した結果，図 3 の (a) のように表 1 の上 6 つの回帰では予測値が一つの値に固まってしまった一方，(b) の回帰木やそれに伴うアンサンブル学習で比較的学習がうまく進んだので，表 1 の下 6 つのモデルに着目して，連結数を変更しモデルの予測を行なっていく．

4.2.2 連結数を変えた時の予測誤差の結果

それでは，回帰木やそれに伴うアンサンブル学習のモデルを用いて，連結数を変えた時のモデルを評価する．ここでは予測誤差を用いて評価する．図 4 は連結数を 0 から 9 まで変えた時のそれぞれのモデルの予測誤差を示したものになる．また，モデルの予測結果は実行ごとに変わるので，図 5 はそれぞれのモデルを 10 回計算した予測誤差の平均をとったものとなっている．図 4 より連結数が 4 未満の場合は予測誤差が極端に大きくなるケースがあり 4 以上の時に安定して学習できていることがわかる．予測誤差が一番小さいのは連結数 8 の時の ExtraTreesRegressor の時で予測誤差 65.93%であった．またこの時のキャッシュミス数の累計値の予測誤差は 9.40%であった．



(a) ElasticNet (b) ExtraTreesRegressor

図 3: 連結数 4 で用いた回帰モデルの一部

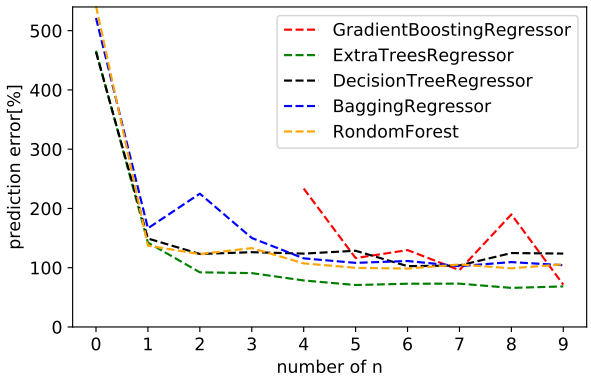


図 4: 連結数 n の時の L1 キャッシュミスの相対誤差 (相対誤差が大きい連結数 4 以下の GradientBoostingRegressor(1000%∼6000%) と AdaBoostRegressor(500%∼1000%) に関してはグラフを省略した)

4.2.3 連結数 6 の時の予測誤差の詳細

4.2.2 章で連結数が 4 以上の時に比較的よく学習することがわかったので，連結数 6 の場合の予測誤差の詳細を見てみる．図 5 は連結数 6 の時の実測値と予測値の両対数グラフを示したもので，図 6 は連結数 6 の時の予測誤差のヒストグラムに予測誤差の平均，最大値，中央値をそれぞれ，赤線，緑線，黄色線で示したものである．またこの時，勾配ブースト+決定木 (GradientBoostingRegressor) のモデルを用いた．

ここで図 5 を見ると，予測値の値が 10^0 から 10^1 に多く存在していることがわかる．しかし，予測値の値が 1 桁の数値の差は実用上はあまり問題のない微小な誤差にも関わらず，予測誤差の計算式では過大な誤差になってしまう．そこで 10 未満のデータを除いて評価しても実用上問題ないと判断し，実測値が 10 未満のデータを除いて，同じモデル，連結数を用いて再度評価を行った (図 7).

図 7 を見ると予測誤差が全体的に下がっていることがわかる．また，予測誤差の平均値は 80.0%から 24.5%，最大値は 1047.2%から 137.6%，中央値は 38.2%から 20.8%となり，実用

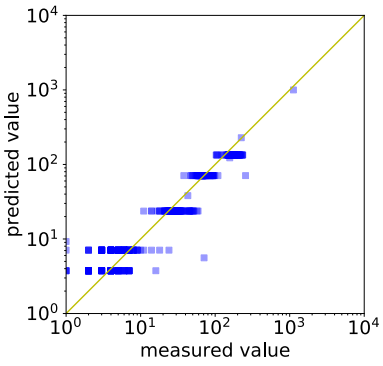


図 5: 実測値と予測値の両対数グラフ (連結数 6)

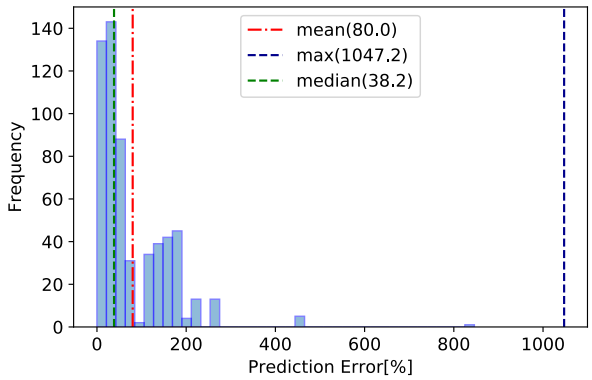


図 6: 相対誤差のヒストグラム (連結数 6)

上必要なデータではより高い精度で学習ができていることがわかる．

4.2.4 実測値 10 未満を除いたデータの予測誤差

実測値が 10 未満のデータを除外したモデルを評価していく．図 5 と同じく，それぞれのモデルで 10 回の平均の予測誤差をとり，連結数を 0 から 9 まで変えた時のグラフを図 8 に示す．

図 4 と比較すると，どのモデルも予測誤差の精度がよくなっている．一番予測精度がよかったのは連結数 8 の時の ExtraTreesRegressor のモデルで予測誤差 19.57%であった．またこの時，キャッシュミス数の累計値の予測誤差は 7.30%であった．

5. 考察

5.1 適切な連結数の選択

図 4 の 10 未満のデータを処理数前の予測誤差のグラフと図 8 の 10 未満のデータを処理した後のグラフをみると，どちらも 0 から 3 までの連結数では予測誤差が相対的に大きくなる場合がある．しかし，連結数が 4 以上になると比較的予測誤差が安定して減っていくのわかる．これは，連結数を増やすことで，入力ベクトルに一意性が生じたため，学習がうまく進み予測値に偏りが減ったためだと思われる．

5.2 元データの分布について

次に Pathfinder に一番多く出現する関数に着目し，4 種類の

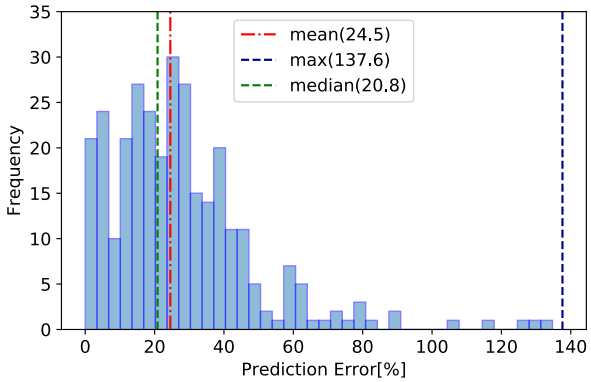


図 7: データ処理後の相対誤差のヒストグラム (連結数 6)

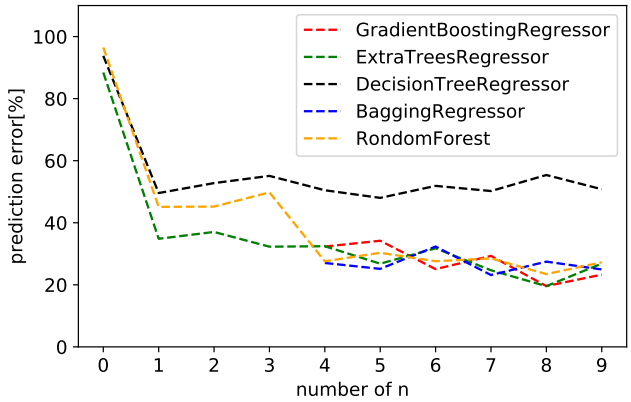


図 8: 連結数 n の時の L1 キャッシュミスの相対誤差 (実測値の処理後，また，相対誤差が大きい連結数 4 以下の BaggingRegressor(100%∼300%)・GradientBoostingRegressor(100%∼700%) や AdaBoostRegressor(100%∼200%) に関してはグラフを省略した)

問題サイズの訓練データのキャッシュミス数のデータ分布を確認し，学習後のモデルが訓練データとテストデータに対し予測した値とデータ分布の関係性をみる．図 9 の size は 100_100，1000_1000，100000_100，10000_1000 が訓練データとして使用した問題サイズ，1000_100 がテストデータとして使用した問題サイズと対応している．図 9 の箱ひげ図は関数のキャッシュミス数のデータ分布になっている．赤色の逆三角がテストデータの予測値となっていて他の青色の逆三角はそれぞれの訓練データの予測値である．またモデルは予測誤差の精度が最も高かった連結数 8 の時の ExtraTreesRegressor のモデルを用いた．

図 9 を見ると，問題サイズが大きくなるとそれに伴い，L1 キャッシュミスの予測値も大きくなっていることがわかり，テストデータに対する予測値もサイズに比例している．またテストデータのデータ分布を見ると，中央値の値とほぼ等しく，特定の関数に対し，キャッシュミス数を上手く学習ができていることがわかる．

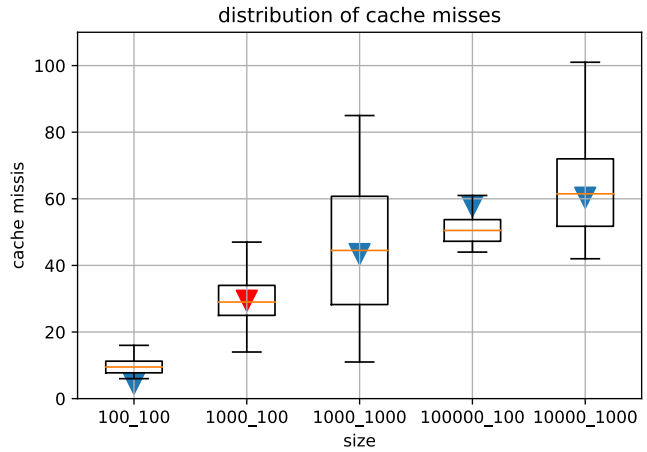


図 9: 訓練データとテストデータのデータ分布とそれぞれのデータの予測値

5.3 モデルごとの精度について

4.2.1 章より線形回帰を中心とした分析やロジスティック回帰などの線形な回帰分析の手法ではうまく学習ができないことがわかった．これは今回使用した入力データセットに対し，出力が線形的ではないためだと思われる．そこでアンサンブルを用いたがアンサンブル学習でもブースティングを使った分析とバギングを使った手法で予測誤差の精度に差が出ている．図 4 をみると，勾配ブースト+回帰木のモデルや Adaboost+回帰木のブースティングを用いた手法が他のバギングを用いた手法などのに比べ精度が悪い．これはブースティングは外れ値に大きな影響を受けやすいモデルであるため，連結数が低いときは入力ベクトルに一意性が小さく，外れ値の影響を大きく受けたためだと思われる．しかし，ブースティングはバギングよりも精度は出やすいモデルであるため，図 8 をみると全体的にブースティングのモデルの方が精度が高い傾向にあることがわかる．

6. 関連研究

6.1 計算機トレースの生成に関する研究

辻らの研究 [6] では，HPC システムとアプリケーションのコデザインにおいて，将来システム上でのアプリケーションの通信性能の推定に着目した．SCAMP(SCAlable Mpi Profiler) という手法を用い，将来システムにおけるアプリケーションの疑似 MPI イベントトレースを既存のシステムの MPI イベントトレースから生成し，さらにネットワークシュミレータを実行して通信性能を評価した．

ネットワークシミュレーションを行うにはトレースの情報が必要となる．しかし，将来システムにおけるネットワークシミュレーションを行うには，既存のトレース情報は使えず，そのシステムのノード数などに対応したシステムのサイズと同数のトレースファイルが必要となってくる．SCAMP 法の概要は図 10 に示す．

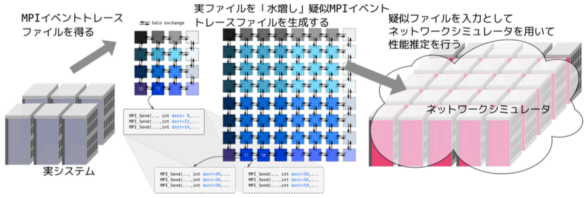


図 10: SCAMP 法の概要 (同論文より引用)

彼らは，プログラムコードの中間表現である LLVM-IR を用いて，コードから疑似 MPI イベントトレースファイルの自動生成にも取り組んでいる．LLVM-IR の MPI 関数を読み込み，関数のフローを追跡することで，疑似 MPI イベントトレースジェネレータを生成する．このジェネレータに実 MPI イベントトレースと将来システムのノード数と MPI ランクを与えること

で疑似 MPI イベントトレースを出力することが可能である．

一方，本研究で提案する手法は通信トレースに限らない一般的なトレースを対象としており，今回の L1 キャッシュミス数などを生成することができる．しかし，本研究では機械学習モデルに対する入力データセットを自動生成することはできていない．

6.2 計算機トレースを使用したベンチマークの生成

Wu らの研究 [7] では HPC のハードウェアとソフトウェアを評価する際に不可欠なベンチマークの自動生成に着目した．ScalaTrace と呼ばれる通信トレースを取得するツールを使用し，HPC 通信トレースを収集し，通信動作および実行時間に関して元のアプリケーションに非常に類似したベンチマークを生成することに成功した．ScalaTrace は Umpire と PMPI を使用して得た通信トレースを追跡し，ノード間の反復を圧縮し，最後は各ノード上のローカルトレースを単一グローバルトレースに結合する．またタイムスタンプも記録する．これを使い，ループに入るタイミングの詳細を記録する．ScalaTrace を得たトレースをトラバースすることで C 言語のベンチマークを自動生成することができる．評価として，各タイプの MPI イベントについて，生成されたベンチマークごとに測定されたイベント数およびメッセージ量が，元のアプリケーションに対して測定されたものと完全に一致した．

Wu らはソースコードを解析することにより性能予測を行うが、一方で本研究ではソースコードを解析せずとも機械学習による実験的なアプローチである程度の精度で性能予測を行えることを示した．

7. まとめと今後の課題

本研究では，Rodinia ベンチマークの Pathfinder プログラムのプログラムの特徴量とキャッシュミス数を学習することで，プログラムの問題サイズを変更した際のキャッシュミス数とその累計値をそれぞれ予測誤差 19.57%，7.30%の精度で生成することができた．これにより，プログラムの問題サイズを変更した際，そのキャッシュミス数をプログラムを実行することなく取得することができるようになった．今後の課題として，複数の計算機トレースを生成することや，一つのプログラムに固定するのではなく，任意のプログラムを当てはめた場合に，計算機トレースを生成したり，そのプログラムに適したアーキテクチャを生成することが必要である．

謝辞

本研究は，産総研・東工大実社会ビッグデータ活用オープンイノベーションラボラトリ (RWBC-OIL) の活動として実施し，JSTCREST(JPMJCR1303，JPMJCR1687) の支援を受けたものである．

参考文献

[1] Rodinia Benchmark: available from ⟨http://www.cs.virginia.edu/ skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators⟩
[2] Score-P User Manual 3.1: available from ⟨https://silc.zih.tu-dresden.de/scorep-current.pdf⟩
[3] PAPI overview: available from ⟨http://icl.cs.utk.edu/papi/overview/index.html⟩
[4] Vampir9 User Manual: available from ⟨https://tu-dresden.de/zih/forschung/ressourcen/dateien/projekte/vampir/dateien/Vampir-User-Manual.pdf⟩
[5] OTF2 documentation: available from ⟨https://silc.zih.tu-dresden.de/otf2-current⟩
[6] 辻美和子, 李珍泌, 朴泰祐, 佐藤三久. 疑似 mpi トレースプロファイルを用いた通信性能推定手法 scamp のための疑似トレースファイル作成手法の検討. Technical Report 14, 筑波大学, 理化学研究所計算科学研究機構, sep 2017.
[7] Wu Xing, Deshpande Vivek, and Mueller Frank. Scalabenchgen: Auto-generation of communication benchmarks traces. In Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pp. 1250 1260. IEEE, 2012.
[8] scikit-learn,Regression: available from ⟨http://scikit-learn.org/stable/supervised_learning.html#supervised-learning⟩
[9] 東京工業大学学術国際情報センター.TSUBAME3.0 ハードウェア・ソフトウェア仕様: available from ⟨http://www.gsic.titech.ac.jp/node/619⟩