Vol.2019-HPC-170 No.12
2019/7/24

メモリアクセスデータを用いた機械学習による アプリケーションの類型化

土川 稔生[1,2,a)]　遠藤 敏夫[1,2]　大山 洋介[1]　野村 哲弘[1]　近藤 正章[3,4]　松岡 聡[4,1,2]

概要：ポストムーア時代の計算機アーキテクチャの開発のためには，様々なアプリケーションを用いて，システム構成やパラメータの異なる大量のケースを評価・解析する必要がある．しかし，世の中の全てのアプリケーションを詳細に解析することは不可能なため，性能特性を代表するアプリケーションを選定することが望ましい．そこで本研究ではアプリケーション群を代表するアプリケーションの選定と特性解析を行うために，アプリケーションの特徴量として Reuse Distance を用いてメモリアクセスデータのトレースを取得し，機械学習のクラスタリングを用いた手法によるアプリケーションの類型化を行った．また，アプリケーションのアルゴリズムの特性とクラスタリング結果の関係について検証した．

1. はじめに

2020 年代に，半導体の微細化によりプロセッサの性能・コスト比を改善することが難しくなる「ポストムーア時代」が到来すると予想される．その時代に向けて，高性能なスーパーコンピューターを構成するためには，デバイス・ハードウェア・システムソフトウェア・プログラミング・さらにはアルゴリズムまで考慮したコ・デザインが必要となる．そのためには様々なアプリケーションを用いて，システム構成やパラメータの異なる大量のケースを評価・解析する必要がある．しかし世の中全てのアプリケーションを詳細に評価・解析することは困難である．

またコンピューターシステムは近年のアプリケーションのデータ要件を満たすために，膨大な量のメモリを管理する必要がある．そのため，キャッシュの使い方などを分析するためにアクティブなメモリアクセスの様子を計測する必要がある．

そこで，様々なコンピューターシステムで評価・解析できるアプリケーションの選定には，特定のハードウェアに依存することなく，メモリアクセスの様子を計測できる特徴量がふさわしいと考えられる．

本研究ではこれらの条件を満たす特徴量である Reuse Distance を用いてメモリアクセスの様子をデータ化し，さらにそれらを機械学習のクラスタリング手法を用いることでアプリケーションを類型化した．また，それらの結果とアプリケーションのアルゴリズムの Reuse Distance における特性の関係について検証した．アプリケーションを類型化できるとアーキテクチャシミュレーションを用いて様々なメモリ構成で相関関係を検証することができ，コンピュータアーキテクチャの開発に役立てることができる．

[1] 東京工業大学
[2] 産業技術総合研究所
[3] 東京大学
[4] 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 Reuse Distance

プログラムが順次実行されると仮定すると，メモリやキャッシュへのアドレスアクセスも時間的に順次に行われる．そのようなアドレスへのアクセスを考えた時，同じアドレスにアクセスする間にアクセスされた重複を除いたアドレスの数を Distance として定義することで Reuse Distance [1] を定義する．また最初にアクセスされるアドレスは以前にアクセスされたデータがないため，Reuse Distasnce は無限大として定義する．こうして定義された Reuse Distance はアプリケーション固有の特徴量であり、キャッシュやメモリの構造に依存しない．そのためアプリケーションの局所性を定量的に評価することができる．またアプリケーション全体の局所性を測定するために，ヒストグラムの形式で使用されることが多い．Reuse Distance の計算例が**図 1** となる．

2.2 Reuse Distance 計算の効率化

アプリケーションのメモリアクセスは多大な回数行われるため，Reuse Distance を計算するには，多大なデータ数を処理する必要があり長い時間を必要とする．そのた

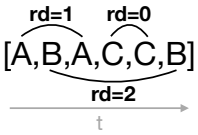

図 1: Reuse Distance の計算例．Reuse Distance を rd として表記した．また A,B,C はそれぞれアドレスの種類を示す．

め Reuse Distance Analytics の論文 [2] と同種の手法を用いて，Reuse Distance 計算のアルゴリズムの計算量を削減するために 3 つの手順で実装する．最初に各アドレスの最後のアクセスのみをトレースに残し，その前に使用したアドレスのセルを空にする．例えばアドレス A が B の前に複数回アクセスされている場合は最後のアクセスのみがトレースに残る．この時 B の Reuse Distance は 2 つの B の参照間の空でないセルの数になる．2 つ目の手順として，空でないセルの数をより早く数え上げるため，トレースを一定サイズ M のセグメントに分割する．各ブロックはそれぞれの空でないセルの総数を保持する．こうすることで空のセルを数え上げる際，セルではなくブロックを調べることになる．最後にブロックを調べる際 $B-tree$ の構造を用いる．こうすることで N 個のアドレスのトレースを考えたとき計算量を $O(NMlogN)$ に抑えることが可能になる．

2.3 クラスタリング手法

クラスタリングは，分類対象の集合を内的結合と外的分離が可能な部分集合 (クラスタ) に分割する手法で KMeans [3,4] や VBGMM(Variational Bayesian Gaussian Mixture) [5] などの手法がある．

KMeans は最初にクラスタの数 k を指定する．その後クラスタごとの代表点を選択し，与えられたデータを一番近い距離のクラスタに配置する．割り当てが終了したらそれらのクラスタの重心点を求め，そこを新たなクラスタの代表点として設定する．その後また新たな代表点との距離で与えられたデータの割り当てを繰り返し，代表点に変化がなくなるまで続ける．VBGMM は KMeans と同じ作業をガウス分布を用いて行うが，クラスタ数の指定せず，クラスタ数はベイズ分布で推定する．

3. 提案手法

メモリアクセスの取得には Intel の Pin ツール [6] を用いた．これを用いて取得したアドレスのトレース情報を 2.2 節の Reuse Distance を求めるアルゴリズムに適用することによってアドレスのトレース数と同数の Reuse Distance 情報が取得できる．Reuse Distance のデータは各アプリケーションによって大きさが異なるため，機械学習に適用するためにヒストグラムの形式に変形し前処理を行う．

3.1 入力データの整形

クラスタリングの入力データを作成するにあたり，最初に Reuse Distance に対する頻度によってベクトルを整形する．すると**図 2** のヒストグラムが描ける．ヒストグラムを図示する場合は見やすさのため初回アクセスに対応する Reuse Distance が無限大のデータは省略する．しかし，アプリケーションごとに無限大を除いた最大の Reuse Distance が異なるため，Reuse Distance を小さい順に並べた後，分割数が等しくなるようにヒストグラムの階級を分け，それらに対する頻度のベクトルに整形する．また，ヒストグラムを確認すると頻度は非常に大きいため，対数をとった数値を各ベクトルの要素とする．このベクトルの最後の要素に無限大扱いである，前に参照がなかったトレースの頻度に対数を取ったものをベクトルの要素として加える．これが一つのアプリケーションから生成されたトレースの特徴量のベクトルとなる．これらのベクトルをアプリケーションの数だけ用意した行列がクラスタリングの入力データになる．

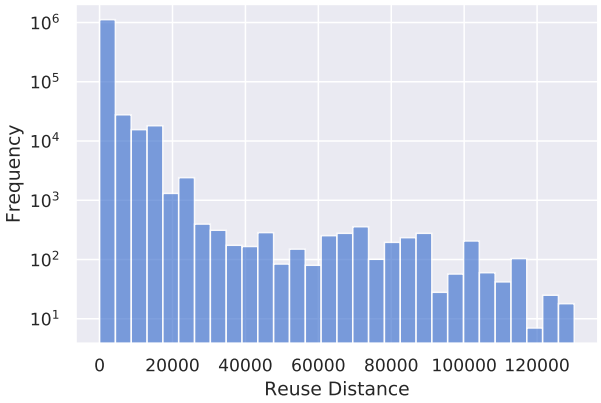

図 2: 入力データ整形後のヒストグラムの例

3.2 使用したアプリケーションとクラスタリングモデル

クラスタリングの対象として，HPC 分野で多用されているアプリケーションから**表 1** に示すアプリケーションを選択し用いた．NPB [7](NAS Parallel Benchmark) は NASA が提供し，BOTS [8](Barcelona OpenMP Task Suite) は Barcelona Supercomputing Center が提供する並列コンピューティング用のアプリケーション群である．また，Rodinia [9,10] はバージニア大学が提供し，ECP [11] は Exascale Computing Project のプロジェクトが提供する並列コンピューティング用のアプリケーション群である．これらのうちいくつかのアプリケーションは入力サイズを変更して実験を行った．またクラスタリングのモデルは scikit-learn [12,13] の cluster と mixture からそれぞれ KMeans と VBGMM の手法を試した．

表 1: 使用したアプリケーションの種類と特性

名前		特性	言語
NPB	BT	3 重対角行列のソルバー	Fortran
	CG	共役勾配法	Fortran
	DC	データキューブへの負荷	Fortran, C
	EP	モンテカルロ法	Fortran
	FT	3 次元 FFT	Fortran
	IS	整数のソート	Fortran
	LU	LU 分解	Fortran
	MG	マルチグリッド法	Fortran
	SP	5 重対角行列のソルバー	Fortran
	UA	非構造適合格子	Fortran
BOTS	Alignment	動的計画法	C
	FFT	高速フーリエ変換	C
	Floorplan	フロアプランの最適化	C
	Fib	フィボナッチ数	C
	Health	医療システムシミュレーション	C
	Sort	マージソート	C
	Strassen	密行列演算	C
Rodinia	pathfinder	2 次元最短経路探索	C++
ECP	miniFE	有限要素法	C++
	miniTri	行列のトライアングル探索	C++

4. 評価

4.1 実験環境

実装の実験環境として，国立研究開発法人 産業技術総合研究所の ABCI [14] を使用した．**表 2** に ABCI の実験時のハードウェア構成とソフトウェア構成を示す．

表 2: 実験環境 (ABCI 計算ノード)

CPU	Intel(R) Xeon(R) Gold 6148 × 2
周波数	2.4 GHz
コア数	20
スレッド数	40
L3 キャッシュ	27.5 MB
メモリ	384 GiB
インターコネクト	InfiniBand EDR × 2
バンド幅	100 Gbps
C コンパイラ	gcc 4.8.5
Pin	version 3.7

4.2 実験結果

実験は表 1 に示したアプリケーションと一部入力サイズを変更したものを別のアプリケーションとして用いた．NPB における入力サイズに対応するクラスは S を選択するとともに，一部のベンチマークではより大きな入力となるクラス W も選択し実験を行った．BOTS アプリケーション群は Fib を除いてデフォルトで設定されている入力サイズで実験を行った．Fib はデフォルトの入力が小さすぎたため，デフォルトの 3 倍の入力サイズで実験を行った．Rodinia の pathfinder は入力サイズを 100_100 と 1000_1000 を用いて実験を行った．ECP の miniFE は入力サイズ 8_8_8 と 16_16_16 と 32_32_32 を用いて miniTri は Matrix Market [15] にある 5 つの行列 can__24,dw2048,fidapm05,jgl009,saylr3 を用いて実験を行った．今後アプリケーションは (アプリケーションの名前_入力サイズ) で表記する．例えば NPB の bt アプリケーションの入力サイズ S については bt_S となる．実験には合計で 30 種類のアプリケーションを用いた．またプロセス数とスレッド数はそれぞれ 1 つで実験を行った．

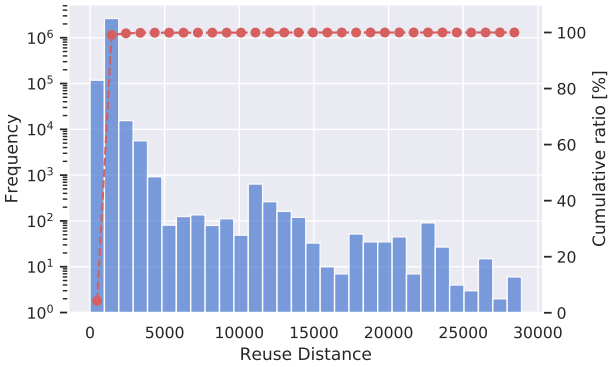

図 3: Floorplan のヒストグラムと累積比率

4.2.1 ヒストグラムの作成

はじめに，アプリケーションの実行ファイルと Pin ツールの中の pinatrace.so をリンクしてアプリケーションを実行するとアドレスの Read/Write 情報が取得できる．これらのアドレスのトレース情報を Reuse Distance に変形したものをヒストグラムの形に変形して視覚的に Reuse Distance がどのような値になっているか，例として Floorplan アプリケーションを用いて，累積比率と共に**図 3** に表す．

このヒストグラムは無限大の扱いとなる初回アクセスとなるアドレスについては省略してあり，x 軸の Reuse Distance の最大値は無限大を省いたものの最大値となっている．アドレス情報のトレースはプログラムのコードの長さ，問題サイズの大きさによって大きく異なるためヒストグラムの最大値はそれぞれ大きく差がある．これらのデータをクラスタリングモデルに適用するために 3.1 節の入力データの整形で述べたように等しい数で分割してベクトルの大きさを揃えていく．図 3 のヒストグラムは 30 個の分割数でデータを分割している．実験ではヒストグラムにあるデータを 99 個に分割しそれらの Reuse Distance に含まれるデータの頻度の対数をそれぞれのベクトルの要素とし，無限大の扱いである初回アクセスとなるアドレス情報の数に対数を取ったものをベクトルの最後の要素として追加し，合計で 100 の要素数のベクトルを 1 つのアプリケーションの入力サイズとして実験した．

4.2.2 クラスタリング結果

図 4 は KMeans を用いた時のクラスタリング結果である．クラスタの数は 5class で，各 marker の種類と色が

class の種類を示す．また 100 次元のベクトルをもつアプリケーションのクラスタリング結果を視覚的に確認するために scikit-learn の t-SNE(t-distributed Stochastic Neighbor Embedding) [16] を用いて 2 次元のベクトルに圧縮した．また VBGMM を用いた 5class のクラスタリング結果が**図 5** である．2 つの図は marker の種類と色は異なっているが *miniFE_16_16_16* のクラスタリング結果を除いて他のアプリケーションは同じクラスタに分類されていることがわかる．また一部のアプリケーションで同じ marker にも関わらず，図上で離れて表示されているアプリケーションがあるが，これは 100 次元のベクトル同士では近い距離に存在するが，t-SNE で 2 次元に圧縮したため，2 次元上では離れて図示されている．

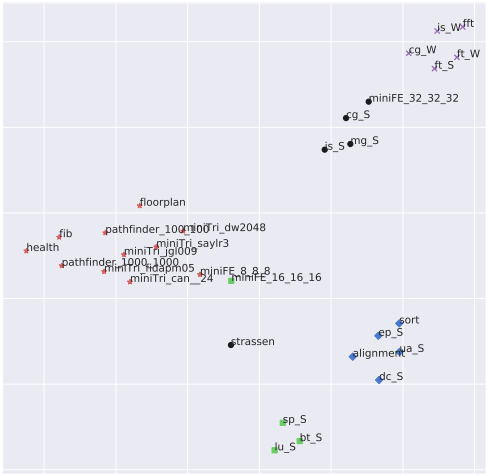

図 4: KMeans による 5class のクラスタリング結果

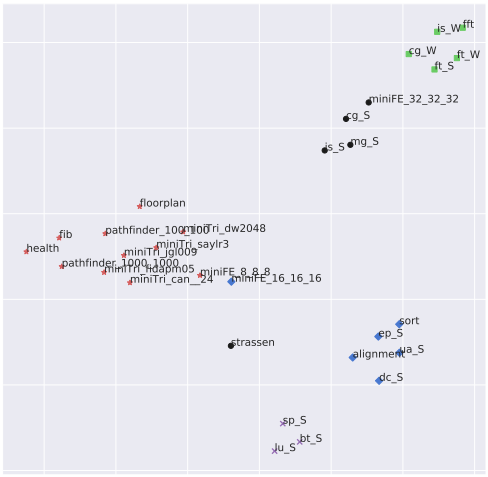

図 5: VBGMM による 5class のクラスタリング結果

5. 考察

5.1 クラスタリング結果について

図 4 と図 5 で近くに分類されている fib と pathfinder(100_100) のアプリケーションについて詳しく考察する．fib はフィボナッチ数を計算するアプリケーションであり，アルゴリズムは以下となる．

Listing 1: fib のアルゴリズム (一部)

```
1  define n;
2  if (n<2);
3      return n;
4  x = fib(n-1);
5  y = fib(n-2);
6  return x + y;
```

このプログラムについて Reuse Distance について考えていくと，最初 write される n のアドレスは前に参照がないため Reuse Distance は無限大となる．また 4,5 行目については Reuse Distance のオーダー O(1) で前のアドレスを read するので fib のアルゴリズムの Reuse Distance は O(1) 付近か無限大に分布するという特徴がある．実際に fib のヒストグラムと累積比率を**図 6** で確認すると 0 付近に大きく値が偏っていることがわかる．

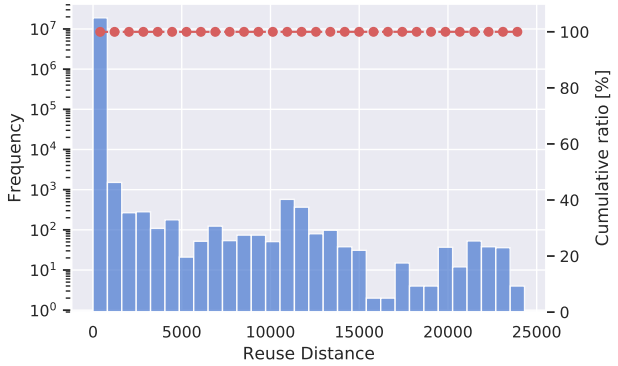

図 6: fib のヒストグラムと累積比率

次に 2 次元経路探索の pathfinder のアルゴリズムは以下となる．

Listing 2: pathfinder のアルゴリズム (一部)

```
1  wall[cols][rows] = rand() % 10;
2  dst = wall[0][rows];
3  src = int[cols];
4  for (int t = 0; t < rows-1; t++) {
5      tmp = src;
6      src = dst;
7      dst = tmp;
8      for(int n = 0; n < cols; n++){
9          tmp = Min(src[n-1], src[n], src[n+1]);
10         dst[n] = wall[t+1][n]+min;
11 return dst;
```

入力サイズを cols= N，rows= N とした時，このアルゴリズムの Reuse Distance について考えていくと，1 行目は初回のアクセスであるため無限大である．2 行目は 1 行目で write したものを read するため $O(N^2)$ である．このアルゴリズムで特徴的な部分は 9 行目の src の read と 10 行目の wall の read である．オーダーはそれぞれ $O(N)$ と $O(N^2)$ だが src の read の方が回数が多いため，このプログラムの Reuse Distance については $O(N)$ の影響が強くなると考えられる．

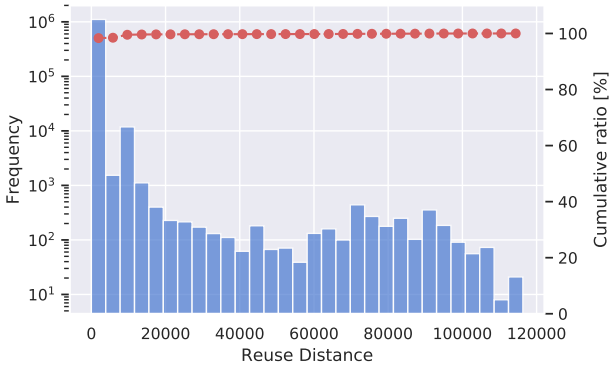

図 7: pathfinder(100_100) のヒストグラムと累積比率

図 7 を確認すると pathfiner の Reuse Distance のヒストグラムと累積比率では $O(N)$ と 0 付近に固まっており，fib のアルゴリズムと似たような特徴を持つといえる．この 2 つのアプリケーションはその他のアプリケーションのヒストグラムと比べ 0 付近に Reuse Distance の特徴が大きく出るアルゴリズムであるため，クラスタリング結果が近くなっていると考えられる．また pathfinder で入力サイズを変更したアプリケーションや miniTri で入力の行列の種類を変更したアプリケーションも同じアルゴリズムをもつアプリケーションであるため，それぞれ同じクラスタに分類されていることがわかる．しかし，この考察では miniFE の入力サイズが異なるアプリケーションも同じクラスタに分類されるべきだが，問題サイズが大きい miniFE_32_32_32 のアプリケーションは異なるクラスタに分類されていることがわかる．ここで miniFE_16_16_16 と miniFE_32_32_32 のアプリケーションについて詳しく考察する．まずヒストグラムはそれぞれ**図 8** と**図 9** になる．またこれらのヒストグラムは見やすさのため初回アクセスに対応する Reuse Distance が無限大のデータは省略し，30 個の階級に分割している．

これらのヒストグラムの特徴として，それぞれ 0 付近の階級の頻度が高い他に，図 8 では 250K 付近，図 9 では 2M 付近に頻度が高い階級が存在する．この 2 番目に高い階級あたりについて，実際にクラスタリングの入力に使用した 100 個の階級に分割したデータを用いて詳しく確認していく．これらのデータに関して miniFE で使用した 3 つの入力サイズを用いて**表 3** に示した．

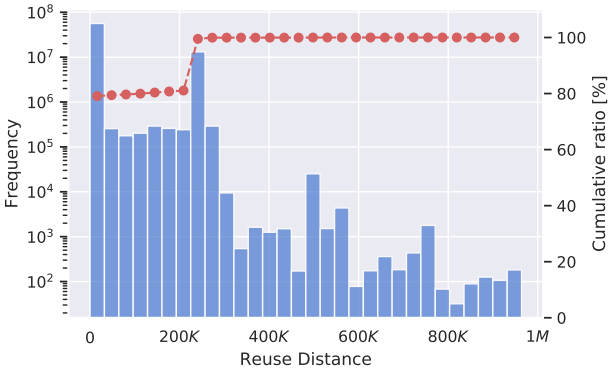

図 8: miniFE(16_16_16) のヒストグラムと累積比率

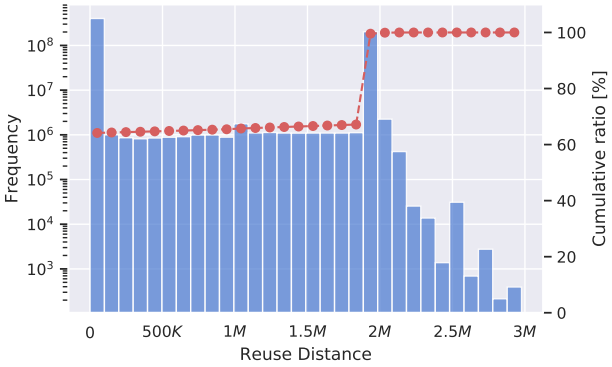

図 9: miniFE(32_32_32) のヒストグラムと累積比率

表 3: miniFE の入力サイズごとの頻度の比較

入力サイズ	1 番目に頻度が高い階級	2 番目に頻度が高い階級
8_8_8	0〜7101	28407〜35509
16_16_16	0〜9734	253096〜262829
32_32_32	0〜30066	1954340〜1984406

表 3 の 2 番目に頻度が高い階級に注目すると，Reuse Distance の大きさが，入力サイズが 8_8_8 から 16_16_16 に増加すると Reuse Distasnce の大きさが約 9 倍増加し，入力サイズが 16_16_16 から 32_32_32 に増加すると約 8 倍増加していることがわかる．入力サイズの変化は 8 倍であり，2 番目に頻度が高い階級の増加の割合と大体同じである．これは有限要素法で 3 次元のデータへのアクセスが順に行われていくようなメモリアクセスの特徴量が設計できれば解析できると考えられるが，現時点では実験データ数が少ないため今後の課題とする．

6. 関連研究

Hashimoto らの研究 [17] では，パフォーマンスチューニングをする際に最適なチューニング方法を検索できるよう

に計算集約型アプリケーションのパフォーマンス履歴からチューニング方法と期待される効率をデータベース化した．最初に経験豊富なエンジニアがアプリケーションのループカーネルのパフォーマンス改善を行いアプリケーションの分類を行う．さらにループカーネルの予測を行うために，Fortran で書かれた parser を使いカーネルの分類器を構築することで Fortran 言語のアプリケーションの分類をした．一方で本研究で分類をするためのデータを集める際に用いる Pin ツールは Fortran に限らず，様々な言語のトレースを取得し分類することが可能である．

Malakar らの研究 [18] では HPC で用いられるようなアプリケーションのパフォーマンスモデリングを 11 種類の機械学習の教師あり学習の手法を用いて，3 つのベンチマークと 4 つのハードウェアで実験し，機械学習の精度について検証した．実験は 3 つのアプリケーションのプロセス数と入力サイズを変更したものを入力とし，出力結果を予測した．その結果，一つの機械学習モデルだけではなく，複数のモデルを組み合わせるバギングやブースティングを用いたモデルの方が良い結果が出たこと，DNN を用いると同じアプリケーション内で他のハードウェアで実験した学習内容を転用して活用することで限られたデータで予測精度が向上することできること，機械学習モデルのアルゴリズムと計算は複雑ではあるが，最近のハードウェアによって推論時間，精度に大きな影響を与えないことなどを示した．

Carrington らの研究 [19] では MPI アプリケーションのトレースを用いてアプリケーションを分類し，アプリケーションのパフォーマンスの予測を行った．分類には独自のツールを利用し，通信，バンド幅，レイテンシなどの様子を追跡し，そのアプリケーションの計算なのかメモリバンド幅なのか通信なのか，どこが律速になっているのかを分類する．この時，異なる数のネットワーク構成で実験した際のパフォーマンスを予測する．このモデルを構築するにはアーキテクチャ固有の知識が必要となることが多く，プロセッサによっては独自の情報の収集が難しいこともある．一方で本研究で用いる Reuse Disrtance はアーキテクチャに依存することがないため，幅広いアーキテクチャへのモデルの適用が可能となる．

7. まとめと今後の課題

本研究ではアプリケーションを選定するためにメモリに関する特性に着目し，様々な HPC アプリケーションのメモリアクセスのトレースを Reuse Distance を用いて取得した．また，これらのトレースを機械学習のクラスタリング手法を複数適用し，似た Reuse Distance の特徴をもつアルゴリズムのアプリケーションは近いクラスタになることがわかった．しかし，同じアルゴリズムでも入力サイズを変更すると違うクラスタリング結果になるアプリケーションも存在しているため，今後は Reuse Distance の他にも，アプリケーション選定にふさわしい特徴量を増やし，アプリケーションの特徴を差別化し，より正確なクラスタリングを目指す．また今回はプロセス数，スレッド数を固定して実験を行ったが，これらの値を動かした時にアプリケーションのクラスタリングがどのようになるかも検証していく．また本研究では HPC アプリケーションだけではなく，他の様々な言語のアプリケーションにも適用できるため，今後スーパーコンピューターなどで使われていくであろう AI アプリケーションのクラスタリングも行う．

謝辞 本研究の一部は，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たものであり，また NEDO 委託業務および JST-CREST(JPMJCR1687) の成果を含む．

参考文献

[1] Mattson, R. L., Gecsei, J., Slutz, D. R. and Traiger, I. L.: Evaluation techniques for storage hierarchies, *IBM Systems Journal*, Vol. 9, No. 2, pp. 78–117 (online), DOI: 10.1147/sj.92.0078 (1970).

[2] Ding, C. and Zhong, Y.: Reuse Distance Analysis, Technical report, Rochester, NY, USA (2001).

[3] Lloyd, S.: Least squares quantization in PCM, *IEEE Transactions on Information Theory*, Vol. 28, No. 2, pp. 129–137 (online), DOI: 10.1109/TIT.1982.1056489 (1982).

[4] MacQueen, J. et al.: Some methods for classification and analysis of multivariate observations, *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, No. 14, Oakland, CA, USA, pp. 281–297 (1967).

[5] Attias, H.: A Variational Baysian Framework for Graphical Models, *Advances in Neural Information Processing Systems 12* (Solla, S. A., Leen, T. K. and Müller, K., eds.), MIT Press, pp. 209–215 (online), available from ⟨http://papers.nips.cc/paper/1726-a-variational-baysian-framework-for-graphical-models.pdf⟩ (2000).

[6] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J. and Hazelwood, K.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation, *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, New York, NY, USA, ACM, pp. 190–200 (online), DOI: 10.1145/1065010.1065034 (2005).

[7] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. and Weeratunga, S.: The Nas Parallel Benchmarks, *Int. J. High Perform. Comput. Appl.*, Vol. 5, No. 3, pp. 63–73 (online), DOI: 10.1177/109434209100500306 (1991).

[8] Duran, A., Teruel, X., Ferrer, R., Martorell, X. and Ayguade, E.: Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP, *2009 International Conference on Parallel Processing*, pp. 124–131 (online), DOI: 10.1109/ICPP.2009.64 (2009).

[9] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H. and Skadron, K.: Rodinia: A Benchmark Suite for Heterogeneous Computing, *Proceedings of the 2009 IEEE International Symposium on Work-*

load Characterization (IISWC), IISWC '09, Washington, DC, USA, IEEE Computer Society, pp. 44–54 (online), DOI: 10.1109/IISWC.2009.5306797 (2009).

[10] Che, S., Sheaffer, J. W., Boyer, M., Szafaryn, L. G., Wang, L. and Skadron, K.: A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads, *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, IISWC '10, Washington, DC, USA, IEEE Computer Society, pp. 1–11 (online), DOI: 10.1109/IISWC.2010.5650274 (2010).

[11] U.S. Department of Energy and National Nuclear Security Administration (NNSA): ECP Proxy Apps Suite, (online), available from ⟨https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/⟩ (2018).

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E.: Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830 (2011).

[13] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. and Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project, *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122 (2013).

[14] National Institute of Advanced Industrial Science and Technology(AIST): AI Bridging Cloud Infrastructure, (online), available from ⟨https://abci.ai/ja/⟩.

[15] Boisvert, R. F., Pozo, R., Remington, K., Barrett, R. F. and Dongarra, J. J.: Matrix Market: A Web Resource for Test Matrix Collections, *The Quality of Numerical Software: Assessment and Enhancement*, Chapman Hall, pp. 125–137 (1997).

[16] Maaten, L. and Hinton, G.: Visualizing High-Dimensional Data using t-SNE, *Journal of Machine Learning Research*, Vol. 9, pp. 2579–2605 (2008).

[17] Hashimoto, M., Terai, M., Maeda, T. and Minami, K.: An Empirical Study of Computation-Intensive Loops for Identifying and Classifying Loop Kernels: Full Research Paper, *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ICPE '17, New York, NY, USA, ACM, pp. 361–372 (online), DOI: 10.1145/3030207.3030217 (2017).

[18] Malakar, P., Balaprakash, P., Vishwanath, V., Morozov, V. and Kumaran, K.: Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications, pp. 33–44 (online), DOI: 10.1109/PMBS.2018.8641686 (2018).

[19] Carrington, L., A. Laurenzano, M. and Tiwari, A.: Inferring Large-Scale Computation Behavior via Trace Extrapolation, pp. 1667–1674 (online), DOI: 10.1109/IPDPSW.2013.137 (2013).