

メニーコア・アクセラレータの比較を目的とした クロスプラットフォーム解析法の提案

岡 慶太郎^{1,a)} Wenhao Jia² 小野 貴継¹ Margaret Martonosi³ 井上 弘士¹

概要：High Performance Computing (HPC) 分野における多くのコンピュータ・システムは高い電力あたりの性能を達成するためにメニーコア・アクセラレータを搭載する。システムの構築する場合、システムで実行され得るアプリケーションに応じて、複数のアクセラレータからシステムに適したものを選択して搭載する必要がある。そのためには複数のアクセラレータ間で性能が異なる原因を解析する必要がある。実行性能解析にはパフォーマンス・モニタリング・カウンタ (PMC) が一般的に用いられる。しかしながら、アーキテクチャが大きく異なる2種のプラットフォームではPMCの対応がとれず、単純にPMC値を比較することが困難である。そこで、アーキテクチャが異なるメニーコア・アクセラレータを比較する手法としてクロスプラットフォーム解析手法を提案する。本研究ではクロスプラットフォーム解析を用いて Intel Xeon Phi と Tesla K20 の性能を比較した結果を示す。

1. はじめに

High Performance Computing (HPC) 分野におけるコンピュータシステムは高い性能を達成できるが、大きな消費電力を必要とする。たとえば、2014年に世界最高性能を達成した Tianhe-2 は 17 MW という大きな電力を消費する。一方、現実的に供給できる電力は 20 MW とされているため [1], HPC 分野のコンピュータ・システムでは、電力あたりの性能を高めることが重要となっている。そこで、HPC 向けコンピュータシステムの多くは、Intel Xeon Phi や Tesla GPU といったメニーコア・アクセラレータを搭載している [2], [3]。実際、Top 500 の上位 10 のシステムの内、Xeon Phi は 2 つのシステム、Tesla GPU は 3 つのシステムに搭載されている。メニーコア・アクセラレータは汎用 CPU に比べ周波数が低く、単純な構成のコアを多数用いて並列処理することで、高い電力性能を達成できるポテンシャルを有する。

メニーコア・アクセラレータにはアーキテクチャの異なるものが複数存在しており、ワークロードによってどのアクセラレータが最も高性能、低消費電力を達成できるかは異なる [4], [5]。したがって、HPC 向けコンピュータシステムを構築する場合、実行され得るワークロードに対して、性能や電力あたり性能がより高いメニーコア・アクセ

ラレータを導入することが望ましい。そのためには複数のアクセラレータの性能や電力を比較し、その原因を解析することが重要である。

性能解析、電力解析においては、あるプラットフォームにおけるプログラムの振る舞いを取得するためにパフォーマンス・モニタリング・カウンタ (PMC) が一般的に用いられる [6], [7]。各 PMC はあるアーキテクチャ要素での、ある動作 (イベントと呼称) の発生回数記録するレジスタである。たとえば、PMC のイベントの一つには L1 キャッシュメモリにおけるキャッシュミスがあるが、その PMC 値は L1 キャッシュミス回数を表す。PMC イベントの項目は多数存在するため、全ての PMC 値を対象に人手で解析するのは非効率である。そこで、性能や電力に強く依存する PMC を特定するためにそこで、回帰分析に基づく性能モデルが提案されている [8]。Xeon Phi については命令レベルのエネルギーモデルが提案されている [9]。GPU を対象とした性能モデル、電力モデルが多く提案されている [10], [11], [12]。しかしながら、複数のアクセラレータの性能の優劣を比較することを前提とした性能モデリング手法は未だ提案存在しない。

複数のアクセラレータにおける性能優劣を解析する場合について考える。各々のプラットフォームで取得した PMC に基いて解析する場合、性能にとって重要な PMC イベントの種類やその PMC 値を比較することが考えられる。しかしながら、PMC イベントの種類はプラットフォームによって異なるため、異なるプラットフォーム間で PMC

¹ 九州大学

² Qualcomm Research

³ Princeton University

^{a)} keitaro.oka@cpc.ait.kyushu-u.ac.jp

イベント同士を比較することはできない。

そこで、本研究ではクロスプラットフォーム解析法を提案する。本手法では特定のプラットフォーム上で取得した PMC 値を用いて各アクセラレータそれぞれの性能モデルを構築する。これにより、性能にとって重要な PMC イベントを比較することが可能となる。特に本稿では、Xeon Phi と Tesla K20 を対象として、様々なプログラムを対象に性能評価を行い、クロスプラットフォーム解析により Xeon Phi と Tesla GPU の性能優劣の原因を解析した。

本稿の構成は以下の通りである。まず、第 2 節で関連研究とその問題点について述べ、第 3 節でクロスプラットフォーム解析法を説明する。次に、第 4 節では評価環境について述べ、第 5 節では Xeon Phi と Tesla GPU を対象にクロスプラットフォーム解析法を適用する。最後に第 6 節で本稿をまとめる。

2. クロスプラットフォーム解析法

2.1 複数プラットフォーム性能解析の選択肢

第 1 節で述べたように、重回帰分析などの統計的手法により性能モデルを合成し、モデル中に出現する PMC 値に基づき性能決定要因を解析する様々な手法が提案された。このような手法は単一プラットフォームに閉じた世界であれば極めて有効である。しかしながら、利用可能な PMC はプラットフォーム依存である場合が殆どである。そのため、異なる複数プラットフォーム間の性能比較解析へと拡張するには、PMC の種類の違いを吸収した性能モデリングが必要となる。その手段として、性能モデリングに利用する PMC 値の選択方法に関して以下の選択肢が考えられる。

- **共通 PMC 方式**: 評価対象となる全てのプラットフォームにおいて利用可能な共通の PMC のみを利用し、各プラットフォームそれぞれにおいて固有の性能モデルを合成する。
- **個別 PMC 方式**: 評価対象となるプラットフォーム固有の PMC を用い、それぞれにおいて性能モデルを合成する。その後、何らかの方法によりプラットフォーム間で異なる PMC それぞれの相関を求め、各性能モデルに反映させる。
- **単一 PMC 方式**: 基準となる単一のプラットフォームを選定し、アプリケーション実行時の PMC の値を取得する。また、同一アプリケーションを各プラットフォームで実行し、それぞれの実効性能を取得する。これらを用いて各プラットフォームの性能モデルを合成する。

ここで、各方式の利点欠点について議論する。共通 PMC 方式ではプラットフォーム間で共通の PMC を利用するため、合成した性能モデルの比較は適切に行うことができる。しかしながら、利用可能な PMC の種類が限定的となるため、性能モデルの精度が低下するといった問題が生じ

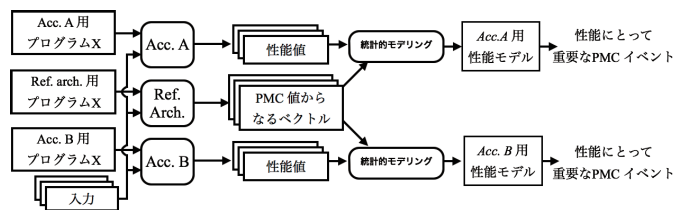


図 1 クロスプラットフォーム解析法の全体像

る。次に、個別 PMC 方式では各プラットフォーム固有の PMC を取得するため、性能モデルの精度を高く保つことが期待できる。しかしながら、プラットフォーム間で種類の異なる PMC の相関を求めることが極めて難しく、これを実現する手段は見いだせない。最後に単一 PMC 方式では、単一のプラットフォームで PMC 値を取得するため、合成される全ての性能モデルの適切な比較が実現できる。しかしながら、共通 PMC 方式と同様、性能モデルの精度低下の可能性がある。これら 3 つの方式から実現手段を検討した場合、まずは実装可能性の観点から共通 PMC 方式、もしくは、単一 PMC 方式が候補となる。次に、様々なプラットフォームへの適用可能性を考慮した場合、各プラットフォームにてサポートされる PMC の種類に依存しない単一 PMC 方式が有望となる（プラットフォーム間で共通の PMC が一つも存在しない場合は共通 PMC 方式は適用できない）。そこで本稿では、単一 PMC 方式となる**クロスプラットフォーム解析法**を提案する。

2.2 クロスプラットフォーム解析のフロー

提案するクロスプラットフォーム解析法のフローを図 1 に示す。ここでは、評価対象プログラム *Pro-X* と入力データ *Inp-X* が与えられた際、2 つのアクセラレータ・プラットフォーム（*Acc.A* ならびに *Acc.B*）の性能比較解析を行う例を示している。処理手順は以下の通りである。なお、消費電力解析を行う場合には、以下のフローにおいて、各プラットフォームでの実行時間（*Texe-A* と *Texe-B*）の代わりに、実効消費電力値を用いれば良い。

- (1) プログラム *Pro-X* と入力データ *Inp-X* を用いて、ある特定プラットフォーム（**参照プラットフォーム**と呼ぶ）で実行する。その際、参照プラットフォームが有する PMC（*PMC_{ref}* と略す）の値を取得する。これにより、参照 PMC を要素とするベクトルを抽出する。
- (2) 評価対象プラットフォーム *Acc.A* において、入力データ *Inp-X* を用いてプログラム *Pro-X* を実行し、実行時間 *Texe-A* を得る。
- (3) *PMC_{ref}* と *Texe-A* を入力とする機械学習により性能モデルを合成する。ここでは説明のために 1 個のプログラムと入力データの組を対象とした場合で説明したが、実際には効率のよい学習を行うために複数セットが必要となる。ここで、本稿では入力データ・サイズに対する性能センシティブリティを解析対象としたた

め、図1では複数入力データを利用する場合を示している。他の解析を目的とする場合には、それに適した学習データを準備すれば良い。なお、性能の統計的モデリングの詳細に関しては第2.3節で説明する。

- (4) 評価対象プラットフォーム *Acc.B* において、入力データ *Inp-X* を用いてプログラム *Pro-X* を実行し、実行時間 *Texe-B* を得る。また、(3)と同様、*PMCref* と *Texe-B* を用いた性能モデリングを行う。
- (5) 上記(3)と(4)で合成した性能モデルを比較し、アプリケーション特性が各プラットフォームに与える影響を比較解析する。

2.3 統計的モデリング

モデルはその生成方法によって機構的モデルと統計的モデルに分類される[13]。機構的モデルは入力と出力の関係についての仮説に基づいて生成されるため、PMCにて性能を説明するには、評価対象アーキテクチャを十分理解することが必要になる。一方、統計モデルは統計解析に基いて生成されるため、評価対象アーキテクチャの理解を必要としない。理解度に依存しないモデルを構築するため統計的モデルを構築する。

これまで様々な統計分析による性能モデルが提案されている[12], [14]。本稿では、モデリングアルゴリズムが単純であり、かつ、モデリングのためのツールが入手可能なStargazer[12]を用いる。Stargazerは段階的回帰分析により性能にとって重要なPMCのみをモデルに含める。したがって、モデルから性能に大きな影響を与えるPMCを特定できる。

モデルで説明する対象を従属変数と呼ぶ。本稿における従属変数は性能である。従属変数は独立変数によって表される。本稿での独立変数はPMCに対応する。なお、本手法では複数のPMCを用いてモデル生成するため独立変数は複数存在する。Stargazerではモデルを単純にするために従属変数にとって影響の大きい独立変数のみを選択する。それら独立変数は3次スプライン曲線にて表現され、従属変数はそれら3次スプライン曲線の線形結合にて表される。

モデリングの過程で暫定モデル（最終的なモデルとなる最有力候補）を生成する。暫定モデルの初期状態は空状態である。Stargazerはこの暫定モデルの質が向上するよう更新作業を繰り返す。そして、ある条件を満たした時の暫定モデルを最終モデルとして出力する。暫定モデルの更新は次のように行われる。まず、現状の暫定モデルに含まれていない新たなPMCを1つ追加し、新たな線形回帰モデルを生成する。たとえば、現暫定モデルが空状態であり、この現暫定モデルに含まれないPMCがN個ある場合は、N個の新暫定モデル候補が生成される。次に、生成した新暫定モデル候補において自由度調整済決定係数（以降、決

定係数と略す）が最大となるものを最有力新暫定モデル候補として選択し、現暫定モデルの決定係数と比較する。もし、最有力新暫定モデル候補の決定係数が現暫定モデルのそれより大きければ、現在の暫定モデルを最有力新暫定モデル候補で置き換える。ここで、決定係数とは回帰モデルのあてはまりの良さを表すための指標であり、値が大きいほど良い[15]。また、選択されたPMC同士の相互関係の一部は従属変数との相関に応じてモデルに含められる。

3. 準備

3.1 対象とするメニーコア・アクセラレータ

Intel Xeon Phi: Intel Xeon Phiの概略図を図2に示す。Xeon Phiは多数のコアから成る。たとえば、Intel Xeon Phi 5110Pには60コアが搭載されている。また、各コアは512KBのL2キャッシュメモリが搭載されており、これらL2キャッシュのコヒーレンスを管理するためにタグディレクトリ(TD)が用いられる。

Xeon Phi コアは2命令インオーダー発行を可能とする。各コアは32KBのL1命令キャッシュとL1データキャッシュを搭載する。Xeon Phi コアはスカルユニットとベクトルユニットの2つの実行ユニットを有する。スカルユニットはIntel社の従来のプロセッサと互換性のある命令を実行するために用いられ、ベクトルユニットはXeon Phi専用のベクトル命令を実行するために用いられる。ベクトルユニットのSIMD幅は512ビットであるため、1ベクトル命令は最大16要素に対して単精度計算が可能である。さらに、ハードウェア資源の利用率を高めるために、Xeon Phiはマルチスレッディングをサポートしており、コアあたりに最大4スレッドを割り当てることが可能である。

Tesla GPU: Nvidia Tesla GPUはHPC向けに用いられる代表的なGPUのプラットフォームである。Tesla GPUのアーキテクチャを図3に示す。Tesla GPUは複数のストリーミング・マルチプロセッサ(SM)を有し、各SMにはSIMDレーンと呼ばれる多数の実行ユニットが備わっている。たとえば、Tesla K20m GPUはSMを13個搭載し、各SMは192個のSIMDレーンを有する。SMにはL1キャッシュメモリに加えてシェアードメモリ(SMEM)、コンスタントメモリ(L1C)、テキストチャメモリ(L1T)といったTesla GP特有のメモリが搭載されている。さらに、Tesla GPUは分散共有型のL2キャッシュを有する。たとえば、Tesla K20m GPUはSMあたりに64KBのL1キャッシュ、GPUあたりに1.5MBのL2キャッシュを有する。

1SIMDレーンに割り当てられる一連の命令列をCUDAスレッドと呼ぶ。GPUプログラムは一般的に大多数のCUDAスレッドから構成される。1つのSMにはCUDAスレッドからなるグループ(スレッドブロック)が割り当てられる。そして、スレッドブロックからある一定数の

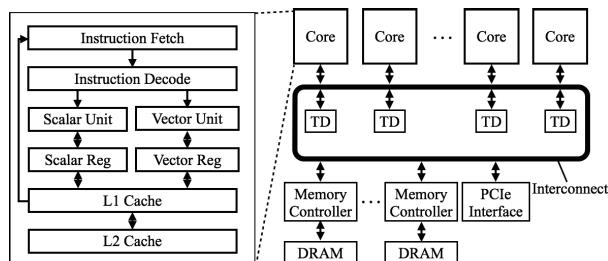


図 2 Intel Xeon Phi コプロセッサのアーキテクチャ

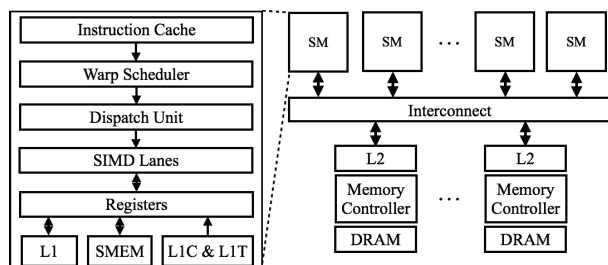


図 3 NVIDIA Tesla GPU のアーキテクチャ

CUDA スレッドからなるグループ（ワープと呼称）を複数生成し、各ワープをワープ内の CUDA スレッド数と同数の SIMD レーンに割り当てて実行する。Tesla K20m GPU ではワープに含まれる CUDA スレッドの数は 32 個である。ワープに含まれる CUDA スレッドは命令とプログラムカウンタを共有するが、演算対象のレジスタが異なる。

3.2 参照用プラットフォーム

Xeon Phi のアーキテクチャは Intel 社の汎用 CPU をベースとするため、PMC イベントの意味を理解しやすい。そこで、参照用プラットフォームの一例として Intel Xeon Phi を用いる。

参照用プラットフォームにおける PMC を用いて他のプラットフォームの性能を解析する場合、参照用プラットフォーム特有の PMC を排除することが望ましい。そこで、Xeon Phi と Tesla K20 両方で性能にとって重要と成り得る PMC のみを用いてワークロードの特徴とする。選択した PMC を表 2 に示す。これらは [16] に基づき、性能への影響が大きいと考えられるものを選択した。各項目の値は表 2 に示す算出方式によって求める。

3.3 ベンチマークプログラム

評価に用いた各プログラムの名前とその説明を表 1 に示す。SHOC ベンチマークには GPU 向けに CUDA [17] にて書かれたソースコードの他に Xeon Phi 向けに OpenMP にて開発されたものが存在するが [18]、それぞれを Xeon Phi よび Tesla GPU の評価に用いる。Xeon Phi 用プログラムは icc 15.0.1 を用いて -O3 オプションにて、GPU 用プログラムは icc 12.1.5 と nvcc 5.0 を用いて -O3 と arch=compute_35, code=sm_35 オプションにてコンパイルする。

SHOC ベンチマークはサイズの異なる 4 つの入力を用い

表 2 参照プラットフォームにて取得する PMC

項目	略称	算出式
ベクトル演算要素数とデータアクセス数の比	V-D-R	$\frac{\text{全ベクトル命令の演算要素数}}{\text{データ読み書き回数}}$
ベクトルユニットの効率	V-I	$\frac{\text{全ベクトル命令の演算要素数}}{\text{ベクトル命令の数}}$
L1 読みミス数	L1-M	$\frac{\text{L1 読みミス回数} \times 1000}{\text{実行命令数}}$
ローカル L2 読みミス数	L-L2-M	$\frac{\text{ローカル L2 読みミス数} \times 1000}{\text{実行命令数}}$
リモート L2 読みミス数	R-L2-M	$\frac{\text{リモート L2 読みミス数} \times 1000}{\text{実行命令数}}$
L1 TLB ミス	L1-TLB-M	$\frac{\text{L1 TLB ミス数}}{\text{データ読み書き回数}}$
L2 TLB ミス	L2-TLB-M	$\frac{\text{L2 TLB ミス数}}{\text{データ読み書き回数}}$
読みバンド幅	R-BW	$\frac{(\text{リモート L2 読み書き回数} + \text{L2 HWP ミス数}) \times 64}{\text{実行サイクル数}}$ HWP: ハードウェアプリフェッチ
書きバンド幅	W-BW	$\frac{\text{L2 でのラインの追出し回数} \times 64}{\text{実行サイクル数}}$

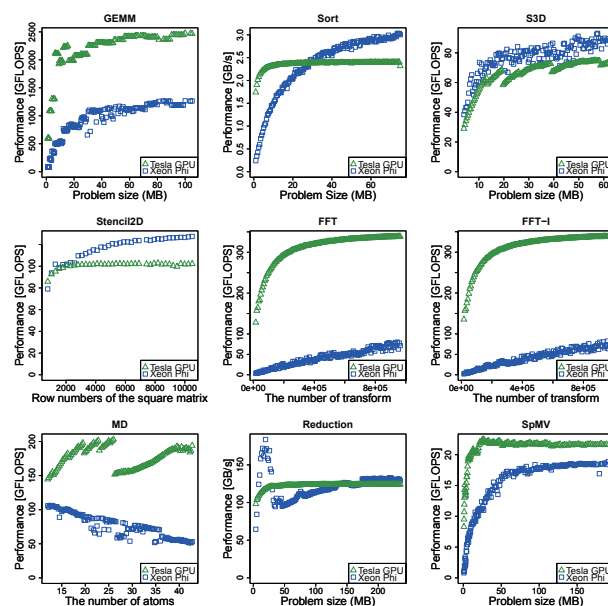


図 4 入力サイズを変化させた場合の SHOC ベンチマーク実行性能で、それから任意の入力を選択して実行する。本研究では入力サイズが細粒度に変化した場合の性能の優劣を評価するために、入力サイズを任意に変更できるようにプログラムに改良を加えた。表 1 に示す要領で入力サイズを変更して評価する。実行時間の制約から Stencil2D のみ 40 個の入力、その他のプログラムには 150 個の入力を用いる。

SHOC には性能値を取得するためのソースコードが用意されている。本評価では性能の評価指標として GFLOPS または GB/s を用いる。実機による評価では性能値がばらつくため、プログラムを 10 回実行して得られる性能値の算術平均を求めることとする。

4. メニーコア・アクセラレータを対象とした性能解析ケーススタディ

4.1 各プラットフォームの性能と入力データサイズ依存性

本節では、提案するクロスプラットフォーム性能解析法の適用事例として、HPC 分野においてアクセラレータと

表 1 本稿で用いる SHOC ベンチマーク

ベンチマーク名	説明	入力の変化方法
GEMM	行列行列積	2 つの正行列の行数と列数を 256 要素から 16 要素づつ増加
SpMV	疎行列ベクトル積	1 次配列の要素数と疎行列の行数と列数を 1024 要素から 100 要素づつ増加
Sort	基数ソート	1 次配列のサイズを 1,024 KB から 512 KB づつ増加
S3D	ナビエ-ストークス方程式の計算	1 次配列のサイズを 4,096 KB から 400 KB づつ増加
FFT	1 次高速フーリエ変換	変換回数を 16384 から 6400 づつ増加
FFT-I	1 次高速逆フーリエ変換	変換回数を 16384 から 6400 づつ増加
MD	ポテンシャルエネルギー計算	原子の数を 12,288 から 256 づつ増加
Reduction	縮約演算による総和計算	1 次配列は 4096 KB から 1600 KB づつ増加
Stencil2D	2 次元データに対する 9 点ステンシル演算	正行列の行数と列数を 768 要素から 256 要素づつ増加

して広く利用されている Xeon Phi ならびに Tesla GPU の性能解析を行う。特に、入力データサイズ依存性の解析に着目する。各 SHOC ベンチマークの入力サイズを変化させた場合の Xeon Phi, ならびに, Tesla GPU の性能の推移を図 4 に示す。これらの結果より, GPU 性能が勝る場合, Phi の性能が勝る場合, ならびに, 単一アプリケーションながら入力サイズによって優劣が逆転する場合があることが分かる。以降の節では, 提案するクロスプラットフォーム解析法を適用し, ケーススタディとして GEMM に関して性能の優劣が生じる原因を分析する。

4.2 プラットフォーム性能モデリング

クロスプラットフォーム性能解析により生成された Phi 性能モデル, ならびに, Tesla 性能モデルについて, モデルに含まれる PMC の項目を表 3 に示す。表 3 での PMC のイベントの項目(表の左から 3 列目)では, 左側に記載されたもののほどこ性能と高い相関があることを示す。また, 表 3 には Phi 性能モデル, Tesla 性能モデルの質を表す決定係数を示す。一般的に, 決定係数が 0.8 以下となる場合には回帰モデルの精度が十分でないと言われる。MD および FFT-I の決定係数は 0.8 以下であり, これらのベンチマークにおけるモデルの質の改善は今後の課題である。

4.3 性能解析 (GEMM)

Phi 性能モデルと Tesla 性能モデルの双方において, 共通して V-D-R, L1-M が性能にとって重要な PMC として選択されている。また, Phi 性能モデルでは L-L2-M, Tesla 性能モデルでは V-I と異なる PMC が選択された。図 5 に示すように GEMM における V-D-R は, 他のベンチマークと比較して高い値を示していることが分かる。一方, L1-M と L-L2-M が性能モデルに含まれており, キャッシュの利用効率が性能に影響を与えている事を示している。実際, 1,000 命令当たりの平均 L1 キャッシュミス回数 (L1 MPKI) および L2 MPKI を測定したところ, 中間値 (異なる入力での MPKI の値を昇順に並べたものの中間の値) はそれぞれ 0.19 および 0.69 であった。これらは比較的小さな値であり, キャッシュを効率的に利用できていると予想される。また, V-I は 15 に近い値程, ベクトルユニットを効率的に

表 3 性能モデリングに基づく性能決定要因の抽出

Bench.	Acc.	Selected Categories by Order of Importance	R^2
GEMM	Phi	V-D-R, L1-M, L-L2-M	0.969
	Tesla	V-D-R, L1-M, V-I	0.963
FFT	Phi	V-D-R, W-BW, V-I	0.854
	Tesla	V-D-R, L1-M, V-I	0.958
SpMV	Phi	W-BW, L1-M, V-I	0.986
	Tesla	V-D-R (W-BW), V-I, L1-TLB-M	0.970
MD	Phi	W-BW (L-L2-M), L1-TLB-M, L1-M, V-D-R	0.979
	Tesla	L1-M, L1-TLB-M, V-I	0.524
Reduction	Phi	R-BW (V-D-R, L-L2-M, R-L2-M), V-I, W-BW	0.850
	Tesla	V-D-R (L-L2-M, R-L2-M, R-BW), V-I, L1-M, W-BW	0.990
S3D	Phi	V-D-R, V-I, R-BW, L1-M	0.893
	Tesla	V-D-R, R-BW, V-I, L-L2-M (R-L2-M)	0.921
GEMM-T	Phi	V-D-R, L1-M (V-I)	0.977
	Tesla	V-D-R, V-I (L1-M), R-L2-M	0.959
FFT-I	Phi	V-D-R, V-I, L1-TLB-M	0.703
	Tesla	V-D-R, V-I, L1-M	0.942
Sort	Phi	W-BW, V-D-R	0.957
	Tesla	V-D-R, W-BW, V-I	0.984
Stencil2D	Phi	V-D-R (L1-M, L-L2-M, R-L2-M, R-BW, W-BW), V-I	0.995
	Tesla	V-D-R (L1-M, L-L2-M, R-L2-M, R-BW, W-BW)	0.968

利用できていることを表すが, 最低でも 10.3 であるためベクトル命令を効率よく実行できている。これから, GEMM のワークロードの特徴としてはベクトル命令の割合が極めて高く, ベクトルユニットとキャッシュを効率的に利用できること。これと図 4 の結果から, GEMM のようなプログラムでは全入力を通して Tesla が Xeon Phi に対して高性能を示す。この原因としては Tesla が Xeon Phi と比較して多数の SIMD 演算器を有することが考えられる。実際, Tesla は Xeon Phi に対して 2.6 倍の SIMD 演算器を有する。

5. おわりに

本稿では, 異なる複数プラットフォームの性能比較解析を可能とする手法としてクロスプラットフォーム解析法を提案した。また, メニューコア型アクセラレータを対象とし

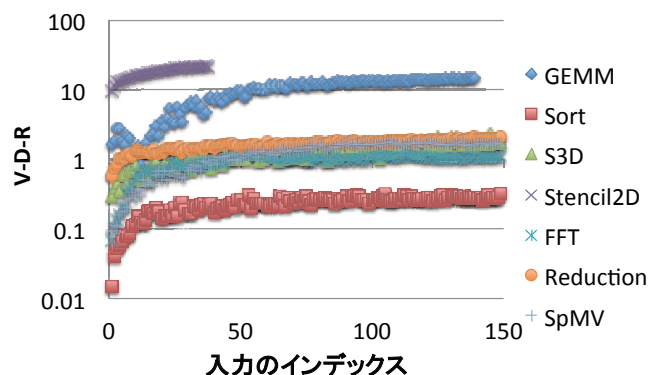


図 5 各ベンチマークにおいて入力を変化させた場合における V-D-R の値

たケーススタディを行い、提案方式の利用可能性を検討した。評価を行った結果、多くのプログラムで高い決定係数を実現でき、性能モデルとしては質の良い結果を得ることができた。その一方、幾つかのベンチマークにおいては決定係数が低く、さらなる改善が必要であることが分かった。今後は、モデリング精度の改善、ならびに、提案解析法そのものの詳細評価を実施する予定である。

謝辞 本研究は、一部、JST CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題「ポストペタスケールシステムのための電力マネジメントフレームワークの開発」の支援による。また、計算機の利用にあたり九州大学情報基盤研究開発センターの協力を得た。

参考文献

- [1] P., K. and K., B.: *Exascale computing study: Technology challenges in achieving exascale systems* (2008).
- [2] Intel: *Intel Xeon Phi Coprocessor: Developer's Quick Start Guide* (2013).
- [3] NVIDIA: *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110* (2012).
- [4] Teodoro, G., Kurc, T., Kong, J., Cooper, L. and Saltz, J.: Comparative Performance Analysis of Intel Xeon Phi, GPU, and CPU: A Case Study from Microscopy Image Analysis, *Proc. IEEE Intl. Parallel and Distributed Processing Symp.* (2014).
- [5] Li, B., Chang, H.-C., Song, S., Su, C.-Y., Meyer, T., Mooring, J. and Cameron, K.: The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications, *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp. 1448–1456 (2014).
- [6] Intel: *Intel Xeon Phi Coprocessor: Performance Monitoring Units* (2012).
- [7] NVIDIA: *Tuning CUDA Applications for Kepler* (2014).
- [8] Nagasaka, H., Maruyama, N., Nukada, A., Endo, T. and Matsuoka, S.: Statistical Power Modeling of GPU Kernels Using Performance Counters, *Proc. Intl. Green Computing Conf.* (2010).
- [9] Shao, Y. S. and Brooks, D.: Energy Characterization and Instruction-Level Energy Model of Intel's Xeon Phi Processor, *Proc. Intl. Symp. Low Power Electronics and*

Design (2013).

- [10] Hong, S. and Kim, H.: An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness, *Proc. 36th Intl. Symp. Computer Architecture* (2009).
- [11] Hong, S. and Kim, H.: An Integrated GPU Power and Performance Model, *Proc. 37th Intl. Symp. Computer Architecture* (2010).
- [12] Jia, W., Shaw, K. A. and Martonosi, M.: Stargazer: Automated Regression-Based GPU Design Space Exploration, *Proc. IEEE Intl. Symp. Performance Analysis of Systems and Software* (2012).
- [13] Eyerman, S., Hoste, K. and Eeckhout, L.: Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware, *Proc. Intl. Symp. Performance Analysis of Systems and Software* (2011).
- [14] Lee, B. C. and Brooks, D. M.: Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction, *Proc. 2006 ACM Intl. Conf. of Architectural Support for Programming Languages and Operating Systems* (2006).
- [15] Kutner, M. H., Nachtsheim, C. J., Neter, J. and Li, W.: *Applied Linear Regression Models*, McGraw-Hill/Irwin, 5th edition (2005).
- [16] Cepeda, S.: Optimization and Performance Tuning for Intel Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events.
- [17] NVIDIA: *CUDA C Programming Guide* (2014).
- [18] Spafford, K.: Pre-release of SHOC for Intel Xeon Phi, <https://github.com/vetter/shoc-mic>.