Vol.2013-HPC-140 No.33
2013/8/2

Xeon PhiにおけるSpMVの性能評価

大島 聡史[1,a)]　金子 勇[1]　片桐 孝洋[1]

概要：本稿では最新のメニーコアプロセッサ Xeon Phi(以下，Phi) の疎行列ベクトル積演算性能について述べる．Phi は高い演算性能およびメモリ転送性能を備えたハードウェアであり，様々なアプリケーションへの適用について盛んに研究が行われている．また Phi はその性能を従来の CPU 同様のプログラミング手法によって活用できることが重要な特徴・利点としてあげられているものの，実際にはアーキテクチャの特性にあわせた最適化の余地が多く存在することが知られている．一方で Phi はアーキテクチャとしても製品としても新しいものであるため，性能を十分に引き出すための知識や技術のさらなる共有が必要である．本稿では疎行列ベクトル積を対象として Phi の性能を測定し，他の並列計算ハードウェアと性能を比較して性能評価を行う．なお本稿では Phi として先行提供版の Preproduction Xeon Phi を用いている．

1. はじめに

近年の計算機システムにおいては高い演算性能を得るために並列計算ハードウェアの活用が進んでいる．特に大きな注目を集めているハードウェアの一つとして，汎用性と高い並列演算性能を兼ね備えたメニーコアプロセッサがあげられる．MIC(Many Integrated Core) の名で注目されていた Intel 社のメニーコアプロセッサは，本年初頭より Xeon Phi(以下，本稿では Phi と表記する) として一般消費者への流通が開始された．最新のスーパーコンピュータのランキング TOP500(2013 年 6 月版) においてはランキング第一位の Tianhe-2 を初めとして Phi を搭載した計算機システムが 12 システムランクインしており [1]，HPC 分野における Phi の本格的な利用が始まったと言える．2020 年頃の達成を目指す 1ExaFlops 級のスーパーコンピュータを実現するための技術の一つとしても Phi への期待は大きい．著者らの所属する東京大学情報基盤センターにおいてもシステムソフトウエアや通信機構に関する研究等 [2][3] を中心に Phi の活用が行われている．

Phi は既存の汎用 CPU と比べて単純な計算コアを多数搭載した並列計算ハードウェアである．Phi と同様に単純な計算コアを多数用いたハードウェアとしては GPU が広く普及しているが，既存の CPU 向けに作成されたプログラムを GPU 上で高速に動作させるには GPU 向けにアルゴリズムの変更やプログラムの大幅な書き直しが必要なことが多く，GPU の活用に消極的な立場の意見も少なくはない．これに対して Phi は既存の CPU と比べて高い理論演算性能や理論メモリ転送性能を持つ一方で，既存の CPU と同様の開発環境やプログラミング手法が利用可能である．そのため高性能への期待のみならずプログラム移植性などの観点からも Phi に期待しているという意見もある．

しかしながら，確かに Phi 上で実行可能なプログラムは容易に作成できる一方で，Phi の持つ高い性能を発揮するには Phi の特性に合わせて適切なプログラム記述を行い，コンパイルオプションや環境変数の設定を行う必要がある．さらに Phi はアーキテクチャとしても製品としても新しいものであるため，性能最適化手法や具体的なプログラム記述方法と得られる性能についての情報共有が急務である．

我々は疎行列ベクトル積などの数値計算問題の高速化について興味を持っており，これまでにマルチコア CPU や GPU を用いた高速計算手法の提案等を行っている [4][5]．そこで本研究では Phi 上で疎行列ベクトル積を実行し性能を評価する．またいくつかの性能最適化手法・パラメタの影響についての評価，他ハードウェアとの性能比較も行う．

本稿の構成は以下の通りである．2 章では Phi のアーキテクチャやプログラミング手法の概要を述べる．3 章では対象問題である疎行列ベクトル積について述べる．4 章では疎行列ベクトル積プログラムを用いて性能評価を行った結果を示す．5 章はまとめの章とする．

2. メニーコアプロセッサ Xeon Phi

本章では Phi のアーキテクチャとプログラミング手法，最適化手法の概要について述べる．Phi 向けのプログラム作成には原稿執筆時点で Intel 社製のコンパイラ群と GNU

[1] 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
[a)] ohshima@cc.u-tokyo.ac.jp

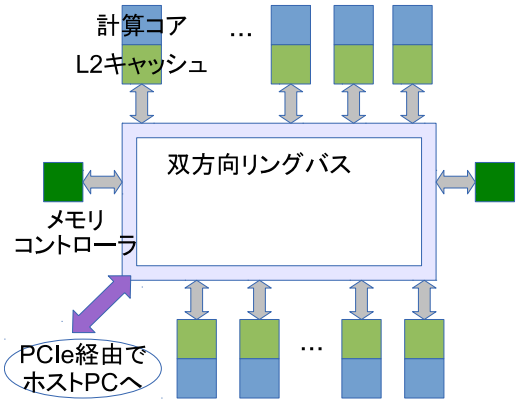

図 1　Xeon Phi の全体構成

コンパイラ群が利用可能であり利用するコンパイラ群によってプログラミング手法や性能にも差が生じる．本稿の記述は Intel 社製コンパイラの利用を前提としている．なお Phi に関する情報については Intel 社の web サイト [6] に多くの資料が掲載されているので参考にされたい．

はじめに Phi の全体的なハードウェア構成について述べる．**図 1** に示すように，Phi は計算コアとメモリを双方向の高速なリングバスで繋いだ構成となっている．各計算コアは既存の Intel 社製 CPU をもとにしたインオーダー実行の単純な計算コアであり，SIMD 処理 (AVX3) には対応しているものの，計算コア単体の性能は現在の CPU と比べて低い．Phi の命令セットアーキテクチャは k1om であり既存の x86 系とは別なため，既存の CPU 向けに作成されたバイナリをそのまま実行することはできない．

現在の Phi は HPC 用途にも利用されている高性能 GPU カードと同様に PCI-Express でホスト PC と接続するアクセラレータカードの形状で提供されている．以下本稿ではホスト PC に搭載された CPU のことをホスト CPU と呼ぶ．Phi 1 基 (カード 1 枚) あたりの計算コア数は 60 程度であり，例えば 5110P では 60 コア，7120P では 61 コア，本稿で利用している Preproduction Xeon Phi では 57 コアが利用可能である．また HyperThreading 機能に対応しており，1 コアあたり最大で 4 スレッド実行が可能である．メモリとしては既存の CPU 向けに用いられている DDR 系のメモリではなく，GPU と同様にレイテンシは大きいがスループットの高い GDDR5 メモリを搭載している．

Phi 向けのプログラム開発環境 (コンパイラやライブラリ) については，既存の CPU と同様に Intel コンパイラや Vtune，MKL 等が提供されている．そのためユーザは慣れ親しんだ CPU と大きな差異なく Phi を利用することができる．また現在 GPU とは異なり Phi 上で Linux OS を動作させることが想定されており，ホスト CPU 上の OS から Phi 上で動作する Linux OS に ssh ログインしてプログラムを実行することもできる．

Phi のプログラミングモデルと実行モデルについては，大きく分けて native モデルと offload モデルの 2 種類が利用可能である．

native モデルは Phi 上で起動し Phi 上で動作するプログラミングモデル・実行モデルを意味する．Phi に対応した Intel コンパイラでは通常の C/C++プログラムや Fortran プログラムに対して-mmic というオプションを付けることで native モデル向けの実行可能ファイルを作成することができる．プログラム実行方法は既存の CPU と同様であり，Phi 上で動作する Linux OS に ssh ログインして./a.out の形式でプログラムを起動する．なおホスト CPU から Phi への明示的な ssh アクセスを行わずに Phi 上のプログラムを実行するサポート機能等も存在するが，本稿では割愛する．

一方の offload モデルはホスト上で起動したプログラムから特定の部分のみを Phi で実行させるプログラミングモデル・実行モデルを意味する．この実行モデルは、GPU プログラムにおいて GPU カーネル関数のみを GPU 上で実行し，その他の部分は CPU 上で実行するのに近い．offload 実行向けのプログラムを作成するためには，対象プログラムのソースコードに対して専用の指示文を用いて Phi 上で実行させたい計算部位や CPU-Phi 間のデータ通信等について記述を行い，offload モデルに対応するコンパイラを用いて実行可能ファイルを作成する必要がある．プログラム実行時には，ホスト CPU 上でプログラムを実行すれば offload 対象部分のみが Phi 上で実行される．

Phi 向けに提供されている主な並列化プログラミング手法としては，MPI，OpenMP，そして TBB(Thread Building Block) があげられる．特に MPI と OpenMP はすでに HPC を含めた各分野で広く利用されてきた並列化プログラミング手法である．Phi はユーザから見ると多数の計算コアを持つマルチコアプロセッサであるかのように見えるため，OpenMP や MPI を従来と同様の感覚で利用可能であり，既存のプログラムの移植が容易に行えることが期待できる．

このように，Phi 向けのプログラム作成においては既存の CPU 向けの知識や技術が大いに活用できると考えられる．特に native モデルについては，もちろんプログラムの内容や用いるライブラリ等にもよるものの，CPU 向けに書かれたソースコードを Phi 向けにコンパイルすればそのまま Phi 上で実行できる可能性があり，容易に Phi 対応を行うことができると考えられる．しかしその一方で，既存の CPU 向けのプログラムをそのまま Phi 上で実行した場合，いくつかの要因によって良い性能が得られない可能性がある．

たとえば既存の多くの CPU には Phi と比べてずっと少ない数の計算コアしか搭載されていなかったため，プログラムが低い並列度に最適化されている可能性がある．仮に並列度の低い繰り返し処理 (for ループ，do ループ) が OpenMP によって並列化されていた場合，そのまま Phi で

実行すると一部の計算コアしか使われずに低い性能となることが考えられる．また既存の Intel 社製 CPU と Phi では対応する SIMD 命令 (AVX) が異なり，最適なメモリアクセス単位にも違いがある．そのため，配列の宣言・確保時にこれらの差異を考慮することでより良い性能が得られる可能性がある．手動で SIMD 化を行った (組み込み命令を用いて記述してある) プログラムについてはプログラムの書き換えも必要である．その他，キャッシュヒット率向上のためにブロック化を施してあるコードやプリフェッチのための命令を入れて最適化を施したコードなど，既存の CPU 向けに高度な最適化を施してきたコードを Phi 上で用いる場合はパラメタの変更などの対応を行わねば良い性能が得られない可能性が高い．実行時のスレッドとコアの割り当て方 (アフィニティ設定) についても違いがあり，Phi では既存の compact と scatter 以外に balanced というアフィニティが利用可能となっている．また対象プログラムに CPU に適した部位と Phi に適した部位が混在している場合には、CPU-Phi 間のデータ通信コストを考慮した上でどの部位をどちらのプロセッサ上で実行するかを考えなくては性能低下を引き起こすことになると考えられる．

以上のように，Phi において最大の性能を得るためには Phi 向けのプログラム最適化が必要である．そのためにも Phi 向けの最適化手法や性能へのについての調査や情報共有は重要である．

なお，本稿における性能評価では OpenMP を用いて記述した native モデルのプログラムを用いている．

3. 疎行列ベクトル積

疎行列ベクトル積 (sparse matrix vector multiplication, 以下 SpMV と表記する) は疎行列，すなわちゼロ要素の多い行列とベクトルとの乗算問題である．SpMV は CG(Conjugate Gradient) 法をはじめとした連立一次方程式の数値解法など様々な計算に用いられており，アプリケーション実行時間の多くを占めることが多々ある．そのため高速化の要求が高く，著者らもこれまでにマルチコア CPU や GPU を対象として SpMV に関する研究を行ってきた．

多くの疎行列計算においては大規模な疎行列 (行数・列数の多い疎行列) を扱う．しかしゼロ要素の多い疎行列を密行列として保存すると，ゼロを保存するだけのために非常に大容量のメモリが必要となる．そのうえ SpMV のような計算においてはゼロ要素については計算する必要がないためメモリ利用効率が非常に悪くなる．そのため疎行列の格納にはメモリ効率や計算効率の良い疎行列向けのデータ格納形式が用いられている．特に多く用いられる形式としては，CRS(Compressed Row Storage) 形式 (CSR 形式とも呼ばれる)，CCS(Compressed Column Storage) 形式 (CSC 形式とも呼ばれる)，COO(COOrdinate) 形式，

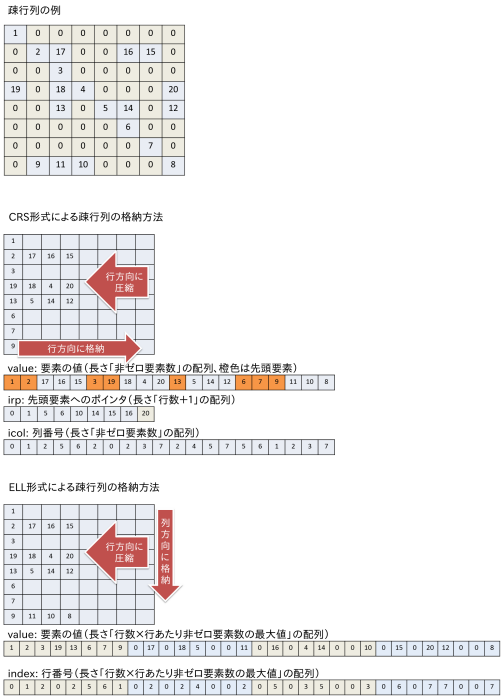

図 2 疎行列格納形式の例: CRS 形式, ELL 形式

ELL(ELLpack) 形式などがあげられる．これらの各疎行列格納形式にはそれぞれ適したハードウェアアーキテクチャや計算の種類などがあるため，一概にどの方式が優れたものであるといった判断はできない．そこで本稿では，広く汎用的に用いられている形式の 1 つである CRS 形式と，ベクトル計算機や GPU などにおいて高い SpMV 性能を得やすい ELL 形式に対象を絞り，SpMV 実装と性能評価および性能比較を行う．

CRS 形式は非ゼロ要素を行方向に詰めた形式である (**図 2**)．CRS 形式の疎行列データは詰められた非ゼロ要素に加えて，各行の先頭要素を指し示す情報 (irp) と各非ゼロ要素の列番号情報 (icol) から構成される．CRS 形式は余計なゼロ要素を保持する必要がないためメモリ効率が良く，格納可能な疎行列の非ゼロ要素配置に制限もないため，多くのアプリケーションやライブラリにおいて標準的な格納形式として利用されている．

CRS 形式を用いた SpMV 計算の OpenMP プログラム (ソースコード) 例を**図 3** に示す．プログラム全体は二重のループによって構成されており，外側ループは行ごとの計算を，内側ループは行内の計算を示している．OpenMP による並列化については，行ごとの計算を独立に行えることを利用し，外側ループに対して施すのが一般的である．しかし外側ループを用いた並列化を行う場合は，対象とする疎行列の行あたり非ゼロ要素数に偏りがあるとスレッドごとに計算する非ゼロ要素数にも偏りが生じることになり，最も多くの非ゼロ要素を担当するスレッドの実行時間が全体の実行時間を律速することになる．このように行列形状

```
int crsSpMVD_kernel
(int n, double *ans,
 double *val, int *irp, int *icol, double *vec)
{
  int i;
#pragma omp parallel for
  for(i=0; i<n; i++){
    int j;
    double tmp = 0.0;
    for(j=irp[i]; j<irp[i+1]; j++){
      tmp += val[j] * vec[icol[j]];
    }
    ans[i] = tmp;
  }
  return 0;
}
```

n : 行数
ans : 計算結果
val : 要素の値
irp : 先頭要素へのポインタ
icol : 列番号
vec : ベクトル

図 3　CRS 形式の疎行列に対する SpMV

```
int ellSpMVD_kernel
(int n, double *ans,
 double *val, int *index, int len, double *vec)
{
  int i, j;
  for(j=0; j<len; j++){
#pragma omp parallel for
    for(i=0; i<n; i++){
      ans[i] += val[j*n+i] * vec[index[j*n+i]];
    }
  }
  return 0;
}
```

n : 行数
ans : 計算結果
val : 要素の値
index : 行番号
len : 行あたり非ゼロ要素数の最大値
vec : ベクトル

図 4　ELL 形式の疎行列に対する SpMV

が性能に与える影響が大きい点については留意する必要がある．なお外側ループではなく内側ループを並列リダクション演算にて並列化するという実装も不可能ではない．しかし既存の CPU では外側ループを並列化した方が高い性能を得やすいことがわかっている．一方で GPU を用いる場合には高並列性と連続メモリアクセスの観点から外側と内側の両方の並列性を用いるのが良い．

ELL 形式も CRS 形式同様に非ゼロ要素を行方向に詰めて扱う (図 2)．しかし ELL 形式では，格納方向が列方向であり，また行ごとに非ゼロ要素数が異なる場合にはゼロで埋める必要がある．疎行列データとしては要素の値，各要素の列番号，そして行数と列数から構成されており，わかりやすい疎行列格納形式の 1 つであると言えるだろう．

ELL 形式を用いた SpMV 計算の OpenMP プログラム (ソースコード) 例を**図 4** に示す．CRS 形式と同様にプログラム全体は二重ループから構成されており，図 4 では行数に基づくループを内側ループとしているが，ループ順序を単純に入れ替えても正しい計算結果が得られる．どちらのループを内側にするか，どちらのループを用いて並列化を行うかについては計算対象ハードウェアのメモリアクセス特性や並列化の方法などに依存する．多くの疎行列においては疎行列の行数 n に比べて行あたり非ゼロ要素数の最大値 len の数が小さいため，非ゼロ要素数のループを外側，行数のループを内側にして内側ループを並列化するといわゆるベクトル長が長い問題となり，ベクトルプロセッサなどに適した実装となる．

なお ELL 形式はその方式の都合上，行ごとの非ゼロ要素数のばらつきが大きい疎行列，特に少数の行が他の多くの行よりも非常に多くの非ゼロ要素を持っているような疎行列についてはゼロで埋めねばならない要素の数が多くなる．ゼロで埋められた要素の多い疎行列は保持に必要なメモリ容量の増加と SpMV 実行時間の増加を招いてしまう．特に GPU や Phi のように搭載メモリ容量が多くない環境においては，対象の疎行列を CRS 形式であれば扱えるが ELL 形式では扱えないという問題が発生しやすいことに注意せねばならない．

4. 性能評価

4.1 評価内容

本章では SpMV プログラムを用いて Phi の性能を評価する．性能評価に用いた Phi の性能および比較対象として用いた各ハードウェアの性能諸元を**表 1** に示す．なお本稿では先行提供版の Phi (Preproduction Xeon Phi) を用いているため，製品版の Phi とは性能や傾向に若干の差が生じる可能性がある．

性能評価対象の疎行列には Florida Sparse Matrix Collection[7] から幾つかの行列を用いた．各行列の非ゼロ要素数などいくつかの形状特性情報を**表 2** に示す．いずれも行数と列数が同じ行列であり，非ゼロ要素の配置の対称性は行列により異なる．さらに**図 6** には各疎行列の行あたり非ゼロ要素数の平均値と分散値を示す．平均値は非ゼロ要素の総数を行数で割った値であり、分散値については (行あたり非ゼロ要素数 - 平均値)^2 の総数を行数で割った値を用いた．

4.2 CRS 形式における実装と性能

本節では CRS 形式で格納された疎行列に対する SpMV 性能について述べる．いずれも SpMV 計算を 101 回繰り返し計算して最初の 1 回以外の 100 回の平均実行時間から FLOPS 値を求めた．

Phi を用いた SpMV の実装については，基本的には図 3 に示したソースコードを用いた．ただしコンパイラオプションや実行時環境変数，OpenMP 指示文については以下に示すような幾つかのパラメタ (以下，これらをまとめて最適化パラメタと呼ぶ) を組み合わせて実行時間を測定し，最大性能を求めた．

- スレッド数: 57, 114, 171, 228 (物理コア数の 1 倍，2 倍，3 倍，4 倍)
- アフィニティ: compact, scatter, balanced
- const restrict の有無
- OpenMP スレッドスケジューリング: 指定無し, dynamic, guided(32), guided(64)
- SIMD 実装: コンパイラによる SIMD 命令生成，手動で単純な SIMD 命令 (AVX3) を記述

また MKL(スレッド並列版) の mkl_cspblas_dcsrgemv につ

表 1　実験環境

	Phi	CPU1	CPU2	GPU
プロセッサ名称・種別	Preproduction Xeon Phi	SPARC64IXfx	E5-2670 (SandyBridge)	TeslaK20c (Kepler)
コア数	57 (最大 228 スレッド)	16	8 (最大 16 スレッド)	2496 (192*13SMX)
動作周波数	1.10 GHz	1.848 GHz	2.60 GHz	706 MHz
メモリ種別	GDDR5	DDR3	DDR3	GDDR5
倍精度浮動小数点理論演算性能	1003.2 GFLOPS †	236.5 GFLOPS	166.4 GFLOPS	1170 GFLOPS
メモリバンド幅	320 GB/s †	85 GB/s	51.2 GB/s	208 GB/s
コンパイラ，etc.	icc 13.1.1 MPPS 2-2.1.5889-16	fccpx 1.2.1	icc 13.1.1	CUDA 5.5RC

† 先行提供版のためこれらの値は明確にされていない．1003.2 GFLOPS は次式により算出した：16 DP FLOPS/クロック/コア × 57 コア × 1.10 GHz．320 GB/s は Xeon Phi 5110P の値であり，本システムでは異なる可能性がある．

いても性能を測定した．

CPU1(SPARC64IXfx) を用いた SpMV についても図 3 にをもとに SpMV 関数を作成した．最適化パラメタについては，スレッド数を物理コア数にあわせて 16，アフィニティは compact のみ，OpenMP スケジューリングは指定無し，コンパイラによる SIMD 命令生成はありとした．SPARC64IXfx および対応する富士通コンパイラには性能に影響を与える多数の指示文やオプションが存在するが，今回はそれらの追求は行っていない．

CPU2(SandyBridge) についても基本的に Phi と同様である．図 3 をもとにした SpMV 関数を作成し，以下の最適化パラメタを組み合わせて実行時間を測定し，最大性能を求めた．

- スレッド数: 8, 16 (物理コア数の 1 倍，2 倍)
- アフィニティ: compact
- const restrict の有無
- OpenMP スレッドスケジューリング: 指定無し, dynamic, guided, guided 32, guided 64
- SIMD 実装: コンパイラによる SIMD 命令生成，手動で単純な SIMD 命令 (AVX) を記述

Phi 同様に MKL(スレッド並列版) の mkl_cspblas_dcsrgemv についても性能を測定した．

GPU(Kepler) の SpMV 性能については，図 3 に対して外側ループを Grid レベルの並列化，内側ループを WARP 単位の並列リダクション演算で実装したものを用いた．なお詳細な実装については割愛するが，これは主に 1 世代前の Fermi アーキテクチャ向けにある程度の最適化を行った自前のプログラムである．また CUDA 付属の cusparse ライブラリによる SpMV 性能についても性能を測定した．

各行列に対する SpMV 性能 GFLOPS 値を**図 7** に示す．横軸には対象とする疎行列の名称を，縦軸には性能 GFLOPS 値をとっており，グラフの左側ほど合計非ゼロ要素数が少ない行列，右側ほど合計非ゼロ要素数が多い行列となっている．

実験の結果，全体としてはグラフの左側，すなわち合計非ゼロ要素数が少ない行列においてはは CPU2(特に非 MKL 版) の性能が高く，グラフの右側，すなわち合計非ゼロ要素数が多い行列においては Phi(MKL 版) や GPU(cusparse 版) の性能が高かった．

Phi の性能は合計非ゼロ要素数が少ない行列においては CPU1 と同程度であり，今回用いた実行環境の中では特別優れた性能ではなかった．一方で合計非ゼロ要素数が多い行列においてはいくつかの行列において最も良い性能が得られていた．また Phi のグラフ全体の形状は GPU よりも CPU に近いものであった．最適化パラメタについては，まず const restrict は多くの行列に対して効果があった．また OpenMP のスケジューリング設定についてはデフォルト (静的なブロック割り当て) が良い性能を得たものが多かった一方で guided スケジューリングがその性能を上回った行列も散見された．並列度 (使用するスレッド数) については 171 スレッドまたは 228 スレッドにて良い性能が得られており，アフィニティ設定については compact か balanced が最大性能を得ていた．スレッド数やアフィニティについては，いずれの実装が良いかを行列形状等から簡単に判定できるような仕組みがあればとても有益であるが，現在のところそのような仕組みや基準は作れていない．

CPU1 の性能は全体的に Phi に似た傾向を示した．合計非ゼロ要素数が少ない行列においては Phi と同程度の性能であり，合計非ゼロ要素数の多い行列においては Phi(非 MKL 版) をやや下回った．一方 CPU2 については，合計非ゼロ要素数が少ない行列において特に高い性能を得た一方で合計非ゼロ要素数が少ない行列においては低性能となった．並列度 (使用するスレッド数) は HyperThreading を用いた 16 スレッドが良い性能を得たものが多かった一方で非ゼロ要素数が多い行列を中心に 8 スレッドの方が良い性能を得たものも少なくはなかった．また Phi と同様に，const restrict は性能に良い影響を与えており，OpenMP のスケジューリング設定については対象の疎行列によって

表 2　計算対象とする疎行列

matrix name	size	non-zeros	min	max
rdist1	4134	94408	3	24
epb1	14734	95053	3	9
memplus	17758	126150	2	574
epb2	25228	175027	3	87
wang3	26064	177168	4	7
wang4	26068	177196	4	7
chem_master1	40401	201201	3	5
airfoil_2d	14214	259688	4	23
ex19	12005	259879	1	50
chipcool0	20082	281150	5	24
poisson3Da	13514	352762	6	110
ecl32	51993	380415	1	22
viscoplastic2	32769	381326	4	93
nmos3	18588	386594	5	33
epb3	84617	463625	3	7
e40r0100	17281	553562	8	62
Baumann	112211	760631	4	7
sme3Da	12504	874887	24	345
scircuit	170998	958936	1	353
torso2	115967	1033473	6	10
xenon1	48600	1181120	1	27
language	399130	1216334	1	107
twotone	120750	1224224	1	188
mac_econ_fwd500	206500	1273389	1	47
raefsky3	21200	1488768	32	80
cant	62451	2034917	1	40
sme3Db	29067	2081063	24	345
mc2depi	525825	2100225	2	4
pdb1HYS	36417	2190591	1	162
rma10	46835	2374001	4	145
poisson3Db	85623	2374949	6	145
consph	83334	3046907	1	78
sme3Dc	42930	3148656	24	405
xenon2	157464	3866688	1	27
shipsec1	140874	3977139	1	78
torso3	259156	4429042	6	21
atmosmodl	1489752	10319760	4	7
memchip	2707524	14810202	1	27

size: 対象行列の一辺の長さ
non-zeros: 合計非ゼロ要素数
min: 行ごとの非ゼロ要素数の最小値
max: 行ごとの非ゼロ要素数の最大値

最適なものは異なるという結果であった．

GPU の性能は CPU や Phi とは異なる傾向を示した．まず自前のプログラムと cusparse については，cusparse の方が全体的に高い性能が得られている．そのうえ cusparse 版は性能のばらつきが他の実装と比べると小さい．対象とする疎行列の形状によって性能のばらつきが生じにくいアルゴリズムを用いているものと思われる．cusparse 版の性能をもう少し詳細に見てみると，合計非ゼロ要素数が少ない行列においては Phi や CPU1 と同程度であり CPU2 と比べると低い性能となった．一方で合計非ゼロ要素数が多い行列においては全体的に高めの性能が得られており，複数の行列において最も高い性能を得た．

以上の性能を図 6 とあわせて見てみると，全体としては ex19 や raefsky3 といった平均値が高い行列においては高めの性能が得られた．しかし sme3D*など平均値だけではなく分散値も高い行列においては大きく性能が落ちているケースも見受けられる．既に述べたように CRS 形式における SpMV の性能は非ゼロ要素の多い行がスレッドにどのように割り振られるかにより大きく左右するため，行あたり非ゼロ要素数の平均値や分散値では性能の大小を説明しきれないと考えられる．

4.3　ELL 形式における実装と性能

つづいて本節では ELL 形式で格納された疎行列に対する SpMV 性能について述べる．FLOPS 値の算出にあたっては ELL 形式にすることで非ゼロ要素数と計算量が増えることを考慮せず，CRS 形式における FLOP 値と ELL 形式における実行時間を用いて行っている．

Phi と CPU2 における SpMV の実装については，図 4 に示した OpenMP 並列化プログラムを利用し，CRS 形式と同様にスレッド数やスケジューリング、アフィニティの設定を調整して最大性能を求めた．また OpenMP のスレッド生成破棄オーバーヘッドの削減を主な目的として，外側ループの前で omp parallel 指示文にてスレッドを生成し，スレッド番号から内側ループの計算範囲を求めておき，内側ループでは求めた範囲のみを計算するという実装 (**図 5**) も行った．以下，この実装を外側ループ並列化と呼ぶ．

CPU1 における SpMV の実装についても図 4 に示した OpenMP 並列化プログラムを用いたが，スケジューリング設定などは全てデフォルトのままである．

GPU における SpMV の実装についても基本的には CPU/Phi 向けの実装と同様の考え方である．内側ループのループ長が対象行列の行数に等しい大きな数であるうえに，ループカウンタに従って順番に配列をアクセスすれば行列データの格納された配列が連続アクセスになるため，高い性能が期待できる．

各行列の性能 GFLOPS を**図 8** に示す．縦軸・横軸の構成は CRS 形式と同様である．全体的な性能の傾向として

```
int ellSpMVD_kernel
(int n, double * ans,
 double *val, int *index, int len, double *vec)
{
#pragma omp parallel
  {
    int i, j;
    int tid = omp_get_thread_num();
    int ntx = omp_get_num_threads();
    int iBegin, iEnd, iStep;
    iStep = (n+ntx-1)/ntx;
    iBegin = iStep * tid;
    iEnd = iBegin+iStep;
    iEnd = iEnd < n ? iEnd : n;
    for(j=0; j<len; j++){
      for(i=iBegin; i<iEnd; i++){
        ans[i] += val[j*n+i] * vec[index[j*n+i]];
      }
    }
  }
  return 0;
}
```

図 5 ELL 形式の疎行列に対する SpMV：外側ループ並列化

は，CRS 形式の結果と大きく異なり，合計非ゼロ要素数の大小とはあまり関係なく GPU が高い性能を得た．また合計非ゼロ要素数が多い行列では Phi が，合計非ゼロ要素数が少ない行列では CPU2 が GPU 並みの性能を得ることが多いという傾向も確認できる．

Phi は合計非ゼロ要素数が多い行列を中心に GPU に匹敵する性能を得たが，合計非ゼロ要素数が少ない行列においてはあまり良い性能が得られなかった．実装方法やパラメタについては，外側ループ並列化が良い性能を得た．ただし最適なスレッド数とアフィニティは対象の疎行列によって異なっており，全体としてはスレッド数は 117 か 228，アフィニティは compact か balanced が最大性能を得ているものが多かった．

2 種類の CPU については，CRS 形式と同様に合計非ゼロ要素数が少ないと CPU2 が，合計非ゼロ要素数が多いと CPU1 が良い性能を得る傾向が見られた．CPU1 は実装のバリエーションがないため特記することは無い．CPU2 は Phi と同様に外側ループ並列化が良い性能を得た．並列度 (スレッド数) については CRS 同様に行列によって適切な値が異なる結果となった．

性能と平均値・分散値との関係については，分散値が高い場合に性能が低いという関係が見られる．これは分散値が高い行列は ELL 形式として格納される際に多くの非ゼロ要素が加えられてしまう一方で FLOPS 値の計算には CRS 形式と同じ演算数を用いているという算出方法による部分も大きいと考えられる．一方で平均値が性能に与える影響は特に見受けられなかった．

5. おわりに

本稿では Preproduction Xeon Phi を用いて SpMV を実行し，得られた性能を報告した．また CPU や GPU と性能を比較して性能の傾向について述べた．疎行列格納形式としては CRS 形式と ELL 形式を用いて，並列度や OpenMP のスレッドスケジューリングなど幾つかの最適化パラメタを用いて最大性能を求めた．性能比較の結果，Phi は今回実験を行った範囲では CPU や GPU と比べて特別高い性能や性能安定性は得られなかった．特に対象行列の合計非ゼロ要素数が少ない際にあまり良い性能が得られなかったところを見ると，スレッド数が多いことにより OpenMP のオーバーヘッドであるスレッドの生成破棄などの処理時間が無視できない大きさとなっている可能性も推察される．

本稿では Florida Sparse Matrix Collection に含まれる行列を対象として，CRS 形式と ELL 形式を用いて性能評価を行ったが，他の行列や行列格納形式を用いた場合の性能や性能特性についても評価を行う価値があると考えられる．特に Phi が全体として合計非ゼロ要素数が少なめの行列に対して低めの性能，合計非ゼロ要素数が多めの行列に対して高めの性能だったことを踏まえて，今後は対象とする行列を絞ってより細かい性能解析を行いたい．また Phi に適した疎行列格納形式や SpMV 計算方法についても今後研究を進めていく予定である．

謝辞 日頃より最適化プログラミングについて議論をさせていただいている東京大学情報基盤センタースーパーコンピューティング研究部門の皆様に感謝します．本研究は JSPS 科研費 24700024(GPU プログラム最適化のための指示文を用いた自動チューニング機構の開発), 24300004(実行時自動チューニング機能付き疎行列反復解法ライブラリのエクサスケール化) の助成を受けたものです．

参考文献

[1] Top500 List - June 2013 — TOP500 Supercomputer Sites http://www.top500.org/list/2013/06/
[2] Balazs Gerofi, Akio Shimada, Atsushi Hori, Yutaka Ishikawa: Towards Operating System Assisted Hierarchical Memory Management for Heterogeneous Architectures, ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC) (2012).
[3] Min Si, Yutaka Ishikawa: Design of Communication Facility on Heterogeneous Cluster, 情報処理学会研究報告 (HPC-133-16) (2012).
[4] 櫻井隆雄，直野健，片桐孝洋，中島研吾，黒田久泰，猪貝光祥：自動チューニングインターフェース OpenATLib における疎行列ベクトル積アルゴリズム，情報処理学会研究報告 (2010-HPC-125)，pp.1-8 (2010).
[5] 大島聡史, 櫻井隆雄, 片桐孝洋, 中島研吾, 黒田久泰, 直野健, 猪貝光祥, 伊藤祥司: Segmented Scan 法の CUDA 向け最適化実装, 情報処理学会研究報告 (2010-HPC-126), pp.1-7 (2010).
[6] Intel: Xeon Phi Coprocessor, Intel Developer Zone http://software.intel.com/en-us/mic-developer
[7] T.A.Davis, Sparse Matrix collection, http://www.cise.ufl.edu/research/sparse/matrices/

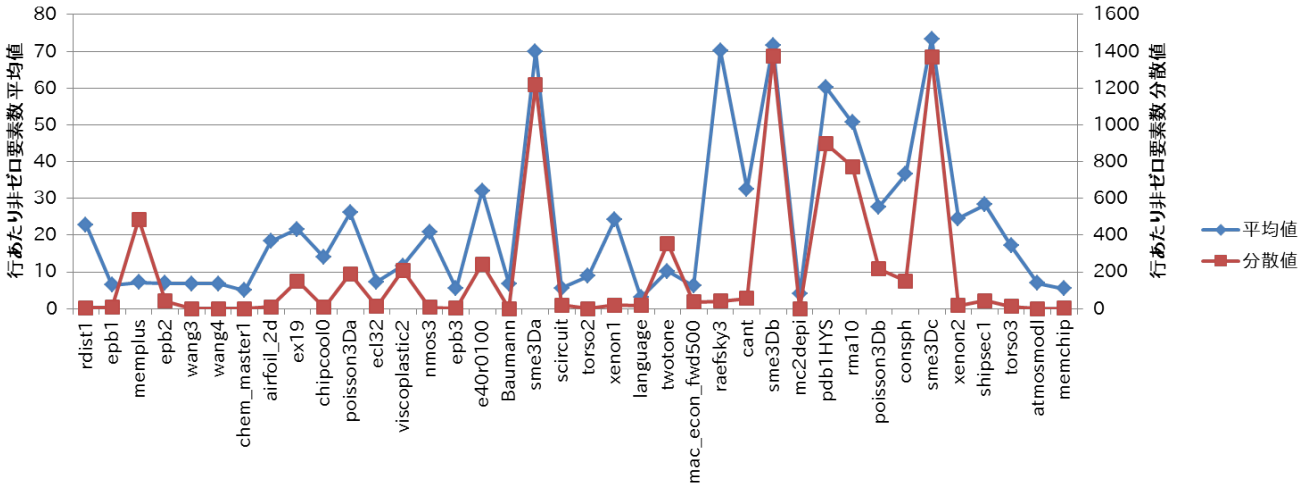

図 6　非ゼロ要素数の平均値と分散値

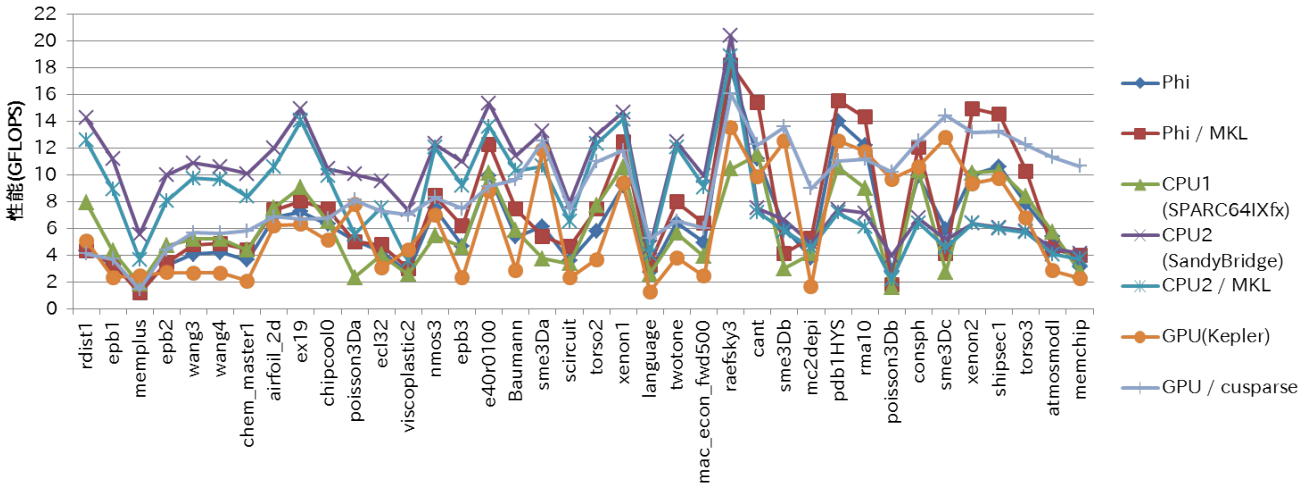

図 7　CRS 形式で格納された疎行列の SpMV 性能

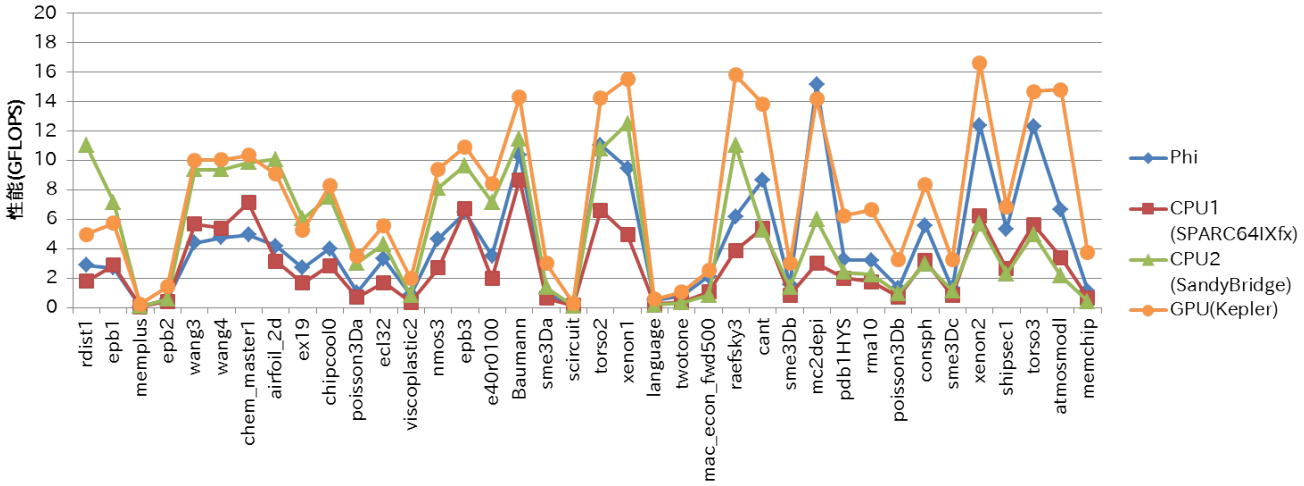

図 8　ELL 形式で格納された疎行列の SpMV 性能