Vol.2017-HPC-158 No.18
2017/3/9

OpenACCを用いたICCG法ソルバーのPascal GPUにおける性能評価

星野 哲也[1]　大島 聡史[1]　塙 敏博[1]　中島 研吾[1]　伊田 明宏[1]

概要：不完全コレスキー分解前処理付き共役勾配法（ICCG 法）は，疎行列連立一次方程式の解法として，科学技術計算において広く使用されている．ICCG 法はデータ依存性を有する計算過程を含むため，並列計算を行うためには多色順序付け等により並列性を抽出する必要があるが，最適な色付け手法・行列格納形式は，実行する並列計算デバイスにより大きく異なることが知られている．本研究では，OpenACC を用いて ICCG 法ソルバーを並列化し，NVIDIA 社の最新の Pascal 世代の GPU（P100）向けの最適化・性能評価を実施し，同世代のメニーコアプロセッサである Intel Xeon Phi（Knights Landing）等と比較評価を行なった結果について報告する．

1. はじめに

スーパーコンピュータシステムの構築・運用において，今日最も重要視される要素の一つが消費電力あたりの演算性能である．消費電力あたりの演算性能を高めるための要素技術として注目されているのが，GPU や Intel Xeon Phi などに代表されるメニーコアプロセッサである．その証左として，スーパーコンピュータシステムの電力あたりの性能を競うランキングである Green500 List[1] の最新版 (2016 年 11 月) において，Top 10 にランキングされたシステム全てがメニーコアプロセッサを利用している．中でも本稿で評価を行う NVIDIA 社の Tesla P100 GPU を使ったシステムが同リストの 1, 2 位にランキングされており，Top 10 中 5 システムは Intel Xeon Phi (Knights Landing[2], 以下 KNL) を用いたシステムである．従ってメニーコアプロセッサに適した計算手法の開発は喫緊の課題であり，本研究では最新のメニーコアプロセッサにおける性能特性の評価・適した計算手法の開発を目的としている．

本研究では評価対象として，疎行列連立一次方程式の解法として科学技術計算において広く使用されている，不完全コレスキー分解前処理付き共役勾配法（Preconditioned Conjugate Gradient Method by Incomplete Cholesky Factorization, ICCG 法）ソルバーを用いる．これまでの研究で我々は，種々のプロセッサにおいて ICCG 法の最適化・性能評価を行なっている．[3] では，NVIDIA GPU (Kepler), Intel Xeon Phi (Knights Corner) を含む種々のプロセッサを用い，並列性を抽出するための多色順序付け手法（MC・RCM・CM-RCM）[4]，疎行列格納手法（CRS・ELL）の評価を行なった．[5] においては，CM-RCM による多色順序付けをベースとし，ELL 行列格納手法のメニーコアプロセッサ向けの拡張を行い，さらに [6] では，SIMD 並列性に優れる疎行列格納形式である SELL-C-σ[7] 法を ICCG 法ソルバーに適用し，KNL での性能改善を達成した．これらの研究が示す通り，疎行列格納形式は性能に大きく影響を与え，また有効な格納形式は実行するプロセッサによって異なる．

本稿では，NVIDIA 社の最新の Pascal 世代の GPU である P100 向けの最適化・性能評価を実施し，同世代のメニーコアプロセッサである KNL 等と比較評価を行なった．その結果，ELL 形式の拡張である Sliced-ELL，SELL-C-σ は P100 においても有効な疎行列格納形式であることが確認された．一方で，P100 と KNL の SIMD 長の違いなどから，P100 では SELL-64-1 形式，KNL では SELL-8-1 形式が最適であるなど，最適なパラメータは異なることを確認した．またプロセッサ同士の比較から，P100・KNL においては十分な性能を得られていないことを確認したため，その原因調査，解決策の模索を行なった結果について報告する．

2. 対象アプリケーション

本稿で対象とするアプリケーションは，図 **1** に示す三次元領域を以下のポアソン方程式を解くものである：

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f \qquad (1)$$

[1] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

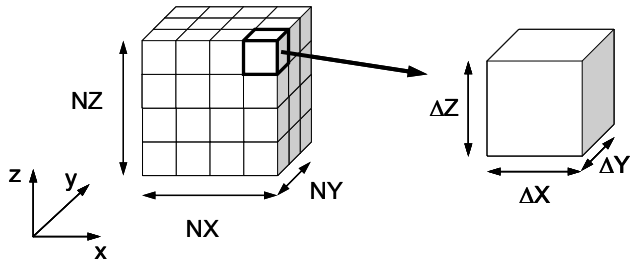

図 1　三次元ポアソン方程式ソルバーの解析対象．差分格子の各メッシュは直方体（辺の長さは ΔX，ΔY，ΔZ），X，Y，Z 各方向のメッシュ数は NX，NY，NZ．

$$\phi = 0@z = z_{max} \tag{2}$$

対象領域は規則正しい差分格子によってメッシュ分割されているが，プログラム中では一般性を持たせるために，有限体積法に基づき非構造格子型のデータとして取り扱う．図 1 における任意のメッシュ i の各面 (6 面) を通過するフラックスは，式 (1) より導かれる以下の式 (3) によって表される：

$$\left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\right]\phi - \left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\phi_k\right] = +V_i f_i \tag{3}$$

ここで，S_{ik}：メッシュ i と隣接メッシュ k 間の表面積，d_{ik}：メッシュ i-k 重心間の距離，V_i：メッシュ i の体積，f_i：メッシュ i の体積あたりフラックスである．この式は各メッシュ i について成立するため，全メッシュ数を N とすると，境界条件と N 個の方程式を連立させた，連立一次方程式 $[A]\{\phi\} = \{b\}$ を解くことにより解を得る．式 (3) の左辺第一項は $[A]$ の対角項，第二項は非対角項，右辺は $\{b\}$ に対応する．各メッシュ i に対応する非対角成分数は最大 6 個であるので，係数行列 $[A]$ は疎 (sparse) な行列である．係数行列 $[A]$ は対称かつ正定 (Symmetric Positive Definite, SPD) であるため，前処理付き共役勾配法 (Preconditioned Conjugate Gradient Method) を適用する．前処理手法としては，対称行列向けに広く使用されている不完全コレスキー分解 (Incomplete Cholesky Factorization, IC) を使用する．本研究では，fill-in を考慮しない IC(0) を使用している．不完全コレスキー分解を前処理手法とする共役勾配法を ICCG 法と呼ぶが，ICCG 法では不完全コレスキー分解生成時，前進代入，後退代入の計算過程でメモリへの書き込みと参照が同時に生じる．そのためデータ依存性が発生する可能があり，故にリオーダリングが必要である．

3. 計算機環境

3.1 概要

本研究において使用した 4 種類の計算環境を表 **1** に示す．P100[8] は NVIDIA Tesla シリーズの最新世代 (Pascal) であり，K20 は Kepler 世代の GPU である．KNL は Intel Xeon Phi シリーズの最新世代 (Knights Landing) であり，BDW は Intel Xeon シリーズの最新世代 (Broadwell-EP) である．また，K20 は東京工業大学学術国際センターの運用する TSUBAME2.5[9] の 1GPU，KNL は最先端共同 HPC 基盤施設 (JCAHPC) の運用する Oakforest-PACS[10] の 1 ノード，BDW は東京大学情報基盤センターの運用する Reedbush-U[11] システムの 1 ソケットを用いている．表 1 中のメモリバンド幅性能は，[12] より取得したベンチマークプログラム（OpenMP 並列）の Stream Triad の実測値を示している．KNL・BDW では取得したベンチマークプログラムをそのまま使用しているが，P100・K20 では独自に OpenACC による並列化を行なったベンチマークプログラムを用いた．表 1 に示した KNL は通常の DDR4 メモリの他，高速な三次元積層メモリ MCDRAM を搭載しており，主記憶要領・メモリバンド幅性能は MCDRAM のものである．また KNL のメモリモード・サブ NUMA クラスタリングモードは，それぞれ Flat・Quadrant モード (設定可能なモードの詳細は [6] 参照）を使用しており，本稿における KNL での実行は全て同モードで行われたものである．

表 **1**　計算環境概要

略称	P100	K20	KNL	BDW
名称	NVIDIA Tesla P100-PCIE	NVIDIA Tesla K20Xm	Intel Xeon Phi 7250 (Knights Landing)	Intel Xeon E5-2695 v4(Broad well-EP)
動作周波数	1.328 GHz	0.732 GHz	1.40 GHz	2.10 GHz
コア数	3,584	2,688	68	18
理論演算性能	4,759 GFlops	1,311.7 GFlops	3,046.4 GFlops	604.8 GFlops
主記憶容量	16GB	6GB	16GB	128GB
メモリバンド幅性能	534 GB/sec	179 GB/sec	490 GB/sec	65.5 GB/sec

3.2 P100 の特徴

従来の NVIDIA 社の HPC 向け GPU（Tesla シリーズ）と比較し，P100 のハードウェア構成は以下の点で異なる．

- 三次元積層メモリ (HBM2) 採用による，メモリバンド幅性能の向上
- 倍精度 AtomicAdd 演算のハードウェアサポート
- 半精度浮動小数点演算 (FP16) のハードウェアサポート
- NVLINK：チップ間接続の高速インタコネクトのサポート．ただし Tesla P100 SXM2 版のみの機能であり，本研究で用いた P100 ではサポートされておらず，本研究で用いた P100 は PCI-e Gen3 により CPU と接続されている．

本研究で対象とするアプリケーションはメモリバウンドなアプリケーションであるため，メモリバンド幅性能の向上

による効果が期待される．

4. 多色順序付け・疎行列格納形式

本章では，ICCG 法ソルバーの並列性を抽出するための多色順序付け手法，疎行列の格納形式について説明する．なお，本稿において用いた色付け手法・疎行列格納手法は，[6] にて用いた手法と同様であり，新たな手法の提案は本稿では目的としていない．そのため，それぞれの概要について簡単に述べる．

4.1 CM-RCM 法によるリオーダリング

ICCG 法の不完全コレスキー分解，前進代入，後退代入は，同じアドレスへの書き込みと参照が同時に生じる可能性のある，データ依存性のある計算過程である．このデータ依存性を回避し並列性を抽出する手法として，色付けによるリオーダリングが広く使用されている．本研究では [6] 同様，CM-RCM(k) 法を使用する．図 **2** は CM-RCM(k) 法による色付け，リオーダリングの例である．

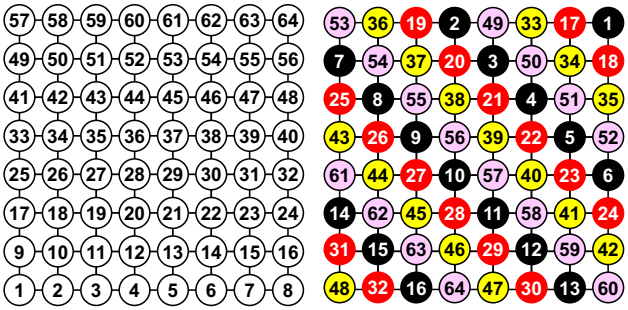

図 **2** CM-RCM(k) 法による色付けとリオーダリング．左) 元の疎行列のオーダリング，右) CM-RCM(4) 法によるリオーダリング．各色内の要素数は 16 でバランス．

CR-RCM(k) 法によるリオーダリングでは，同一の色に属する要素は独立であり，並列に計算可能である．本稿ではさらに，色内の要素を各スレッドに振り分ける方法として，Coalesced 方式と Sequential 方式を用いる．Coalesced 方式は図 **3** のように，色の順番に各要素を番号付した方法である．Sequential 方式は図 **4** のように，Coalesced 方式に対して再番号付けを行い，同じスレッドで処理するデータを連続に配置する方式である．

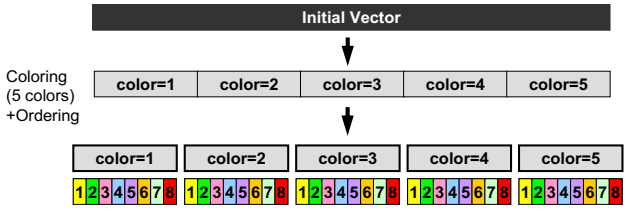

図 **3** 8 スレッドに対する要素の番号付方式．(Coalesced)

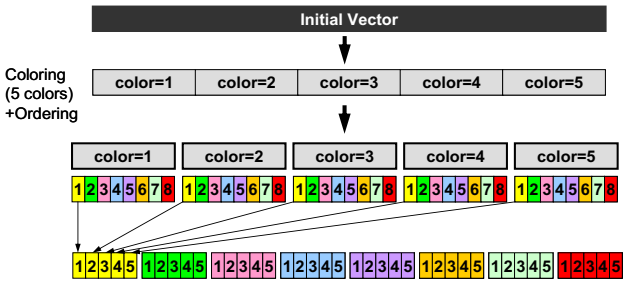

図 **4** 8 スレッドに対する要素の番号付方式．(Sequential)

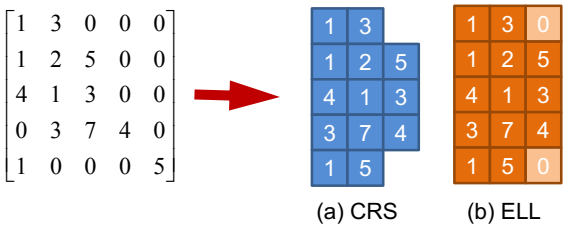

図 **5** 疎行列格納形式．(a) CRS，(b) ELL

4.2 疎行列格納形式

疎行列計算においては係数行列の格納形式が性能に大きく影響することが広く知られており，様々な格納形式が提案されている．図 **5** に示すように，Compressed Row Storage (CRS) 形式は疎行列の非零成分のみを記憶する形式であり，Ellpack-Itpack(ELL) 形式は各行における非零非対角成分数を最大非零非対角成分数に固定し，非零非対角成分の存在しない部分を 0 係数とし計算する形式である．

また，不規則行列に ELL 形式を適用する際，0 係数部の計算を減らすためには，対象行列を非零非対角成分の数の順に並び替え，ループ長を変化させつつ計算する方法が考えられる．このような ELL の拡張形式として，複数の配列を使用することでより効率的な計算を実施する，Sliced-ELL 形式（図 **6**）がある．この Sliced-ELL (SELL) 形式をさらに SIMD プロセッサ向けに拡張した格納形式が，SELL-C-σ（図 6）である．C (chunk size) は計算を実行する単位であり，σ (sorting scope) は疎行列の非零非対角成分の分布によって決定されるパラメータである．本研究においては，各色・各スレッドで処理する要素数が C で割り切れるよう padding を行なっている．

5. 性能評価

5.1 実施ケースの概要

本研究では，色付け手法として CM-RCM(k) 法を用い，Coalesced・Sequential 方式による要素の番号付，CRS，Sliced-ELL，SELL-C-σ による疎行列格納方式を適用した．実施ケースと略称の対応を表 **2** に示す．なお，SELL-C-σ のパラメータは，$\sigma = 1$ であり，C は実験ごとに変更している．要素数は NX=NY=NZ=128 (図 1) の総メッシュ数 2,097,152 である．基本的に [6] で用いた実装をベースとし

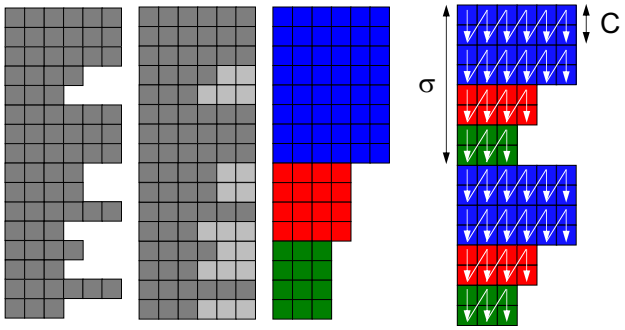

図 6　疎行列格納形式．左から CRS, ELL, Sliced-ELL, SELL-C-σ(図は SELL-2-8) 形式．

表 2　実施ケース

略称	Numbering	疎行列格納形式 (図 5, 図 6)
c-CRS	Coalesced(図 3)	CRS
c-Sliced-ELL		Sliced-ELL
c-SELL-C-σ		SELL-C-σ
s-CRS	Sequential(図 4)	CRS
s-Sliced-ELL		Sliced-ELL
s-SELL-C-σ		SELL-C-σ

ているが，作業用配列のローカル変数への置き換え，ループの融合などのコード上の最適化を一部に施している．さらに 6 章で行う最適化の評価のために，一部の最適化はあらかじめ取り除いている．そのため [6] において示した性能と一致しない部分があるが，全体的な傾向は同様であり，評価結果が矛盾しないことを確認している．

5.2　各プロセッサ向けの実装，環境設定

本稿では，P100・K20 向けには OpenACC による実装を，KNL・BDW 向けには OpenMP による実装を用いている．本節で用いる実装に関しては，OpenACC・OpenMP の指示文を無視すれば等価なプログラムとなる．以下では，OpenACC・OpenMP それぞれによる実装ついての説明，各プロセッサでの実行時の環境設定について説明する．

5.2.1　P100・K20

P100・K20 向けの実装としては，上述の通り OpenACC による並列化を施したプログラムを用いた．使用した指示文は`!$acc data`，`!$acc kernels`，`!$acc loop` の 3 種である．`!$acc data` は，CPU 側とは別のメモリ空間を持つデバイス（ここでは GPU）のメモリを管理するために導入された指示文であり，OpenMP にはなかった概念である（OpenMP4.0 以降の`!$omp target` 指示文の `map` 指示節が似た概念）．`!$acc kernels` は，デバイス側で実行すべき領域を指定するための指示文であり，OpenMP の `!$omp parallel` 指示文が似た概念である．`!$acc loop` は，並列化すべきループを指定し，また並列化されたループ各要素のスレッドへのスケジューリング方法を指定するための指示文であり，`!$omp do` と似た指示文である．

図 **7** に実際の実装例を示す．図 7 は OpenACC・OpenMP

```
1  !$acc data copy(... )
2     do itr = ...    ! 収束判定ループ
3        ....
4        do ic= 2, NCOLORtot-1  ! 色ループ
5  !$omp parallel do private(... )
6  !$acc kernels async(0)
7  !$acc loop independent gang
8        do ip= 1, PEsmpTOT
9          ip1= (ip-1)*NCOLORtot + ic
10 !$omp simd
11 !$acc loop independent vector
12         do i= SMPindex(ip1-1)+1, SMPindex(ip1)
13           ib0= i - SMPindex(ip1-1)
14           VAL= W(i,Z)
15         do k= 1, 3
16           VAL= VAL - AL(ib0,k,ip1)
17     &              * W(itemL(ib0,k,ip1),Z)
18         enddo
19           W(i,Z)= VAL * W(i,DD)
20         enddo
21       enddo
22 !$omp end parallel do
23 !$acc end kernels
24       enddo
25       ....
26    end do ! 収束判定ループ
27 !$acc end data
```

図 7　前進後退代入部（実施ケース：表 2 s-Sliced-ELL）の OpenACC・OpenMP 実装．NCOLORtot：総色数, PEsmpTOT：総スレッド数, SMPindex：各スレッドに属する総要素数, AL：非零非対角成分, itemL：非零非対角成分の列番号, W(i,DD)：対角成分

の指示文が混在しているが，コンパイルの際に一方が無視される．図 7 の 1 行目に現れる`!$acc data` において，デバイス側で必要なデータの転送を行い，27 行目の `!$acc end data` にてデータのコピーバックを行う．このデータ転送は時間計測の外側で行なっており，従って今回の計測時間中には含まれていない．

6 行目から 23 行目までが`!$acc kernels` により囲まれ，デバイス側で実行される部分である．`kernels` 指示文に付随する `async(0)` 指示節は，ホスト CPU 側とデバイス側で非同期な実行を行うためのものであり，CUDA プログラミングにおけるストリームを制御するためのものである．`async(0)` がない場合，CPU はデバイス側の実行終了を待ち，次の処理に進むが，今回の実装ではデバイス側の終了を待つ必要はない．`async(0)` をつける場合，4 行目のループにより，直前のカーネルが終了する前に次のカーネルに到達し得るが，同じ `async` ID を持つカーネルは逐次に実行されるため，カーネル間の依存性による問題は生じない．一方，例えば 5 行目において `async(ic)` などとし，それぞれのカーネルに独立の ID を指定すれば，全てのカーネルが同時に実行され得る．

7 行目と 11 行目に現れる`!$acc loop` により，ループの

各要素のスレッドへのスケジューリングを行なっている．independent 節はループが並列化可能であることを指示する指示節であり，特に 17 行目のような間接参照があるプログラムでは必須となる．また gang，vector 指示節は並列化を行う際の粒度を設定するためのパラメータである．GPU は複数のコアをストリーミングマルチプロセッサ（SM）と呼ばれる単位で管理している．例えば P100 は 56 の SM を持ち，その SM は 64 のコア（FP32 CUDA コア）を持つ構成であるため，スレッドも階層的に管理している．これに対し OpenACC は gang，worker，vector という 3 階層でスレッドを管理し，gang は worker の，worker は vector の集合である．つまり 8 行目のループ要素は SM 単位で振り分けられ，12 行目のループ要素は同一 SM 内のコアが実行するスレッドに割り付けられることが期待される．gang と vector それぞれの数はユーザが指定可能であるが，本節における実験ではコンパイラの自動設定に任せている．

また図 7 においては，8 行目を gang 指示節を用いて並列化している．これにより図 4 の Sequential 方式のリオーダリングは gang 単位で行われるため，実際のスレッド（vector 単位）が連続領域を担当することにはならず，厳密には Sequential 方式ではない．GPU では数万〜数億スレッドを扱うことが可能であり，Sequential なオーダリングでは各スレッドの連続実行領域が極端に短くなるため，Sequentail 方式はそもそも GPU に向いていないと言える．Sequential 方式を用いた s-CRS，s-Sliced-ELL，s-SELL-C-σ での実行においては，図 7 中の PEsmpTOT を P100・K20 の SM 数の 4 倍または 8 倍（P100：224 or 448，K20：56 or 112）に設定し，より性能の良い方を採用した．

コンパイラには pgfortran バージョン 16.10-0 を用い，P100 では -O3 -ta=tesla:cc60, K20 では-O3 -ta=tesla:cc35 をオプションとして設定した．環境変数などは特に設定していない．

5.2.2 KNL・BDW

KNL・BDW での実験では，OpenMP による並列化を施したプログラムを用いた．図 7 に示すように，OpenACC の kernels と同様のループネストを!$omp parallel do により並列化した．

KNL は 68 コアを搭載しているが，今回用いた計算環境では，スレッド ID 0 のコアにのみタイマー割り込みを行わせる tickless と呼ばれる設定が施されているため，当該スレッドに対応する 1 タイル上での計算を避けた 66 スレッドを用いて実行した．具体的には，OMP_NUM_THREADS=66, KMP_AFFINITY=granularity=fine, proclist=[2-67], explicit という環境変数の指定を行なった．一方 BDW においては，物理コア数と同数である 18 スレッドにより実行した．具体的には OMP_NUM_THREADS=18 の指定を行なった．

コンパイラには ifort バージョン 17.0.1 を用い，KNL では -align array64byte -O3 -xMIC-AVX512 -qopenmp -qopt-streaming-stores=always -qopt-streaming-cache-evict=0，BDW では -align array64byte -O3 -xHost -qopenmp をオプションとして設定した．

5.3 各プロセッサにおける色数と計算時間の関係

以下では，表 1 に示した計算環境において，表 2 の各ケースを実施し，各プロセッサにおける色数と計算時間の関係を評価する．図 **8** は各プロセッサにおいて，CM-RCM(k) 法 ($k = 2, 3, ..., 20$) により彩色，各ケースの Numbering 方式・格納形式を用い，各ケース各色数につき 5 回ずつ実行した際の実行時間の平均値を示している．

SELL-C-σ の C 値は，P100，K20，KNL，BDW それぞれで，$C = 64$，$C = 128$，$C = 8$，$C = 4$ を選択した．この値を選んだ根拠については 5.4 節で述べる．全体的な傾向として，以前 [6] の結果同様，CRS 形式より Sliced-ELL 形式，SELL-C-σ 形式が優れており，P100 においても同様であった．Numbering 方式に関しては，P100・K20 においては Coalesced が優れており，KNL・BDW では殆ど差がなかった．疎行列格納形式については，P100・K20・BDW では c-Sliced-ELL がわずかではあるが最も高速であり，KNL では s-SELL-8-1 が最も高速であった．

5.4 P100・K20 における，SELL-C-σ の最適値

SELL-C-σ の C は，プロセッサの SIMD 幅に合わせるのが良いとされる．そのため KNL・BDW においては，それぞれの SIMD 幅（512・256bit）に合わせ，$C = 8, 4$ を設定した．一方 GPU では明確にこの SIMD 幅が決まらない．GPU は warp と呼ばれる 32 スレッドを 1 単位として実行するため，$C = 32$ とするのが良さそうである．しかし GPU は複数の warp を高速に切り替えて実行することでメモリアクセスなどの遅延時間を隠蔽する都合上，複数の warp を用いるために 64 以上の 2 のベキ乗数 (64, 128, 256 など) を使うのが効率的とされる．

$C = 32, 64, 128, 256$ とし，P100・K20 それぞれにて実行した結果が図 **9** である．P100 においては $C = 64$ が最適であり，$C = 32, 128$ も近い性能を示したが，$C = 256$ では性能が劣化した．一方 K20 においては $C = 128$ が最適であり，$C = 64, 256$ も近い性能を示したが，$C = 32$ では性能が劣化した．

5.5 プロセッサ間の性能比較

P100 の性能が妥当であるかどうかを判断するために，プロセッサ間の性能を比較する．図 8 における各プロセッサにおける最速値とメモリバンド幅性能の対応をまとめたものが表 **3** である．ICCG 法ソルバーはメモリバウンドであ

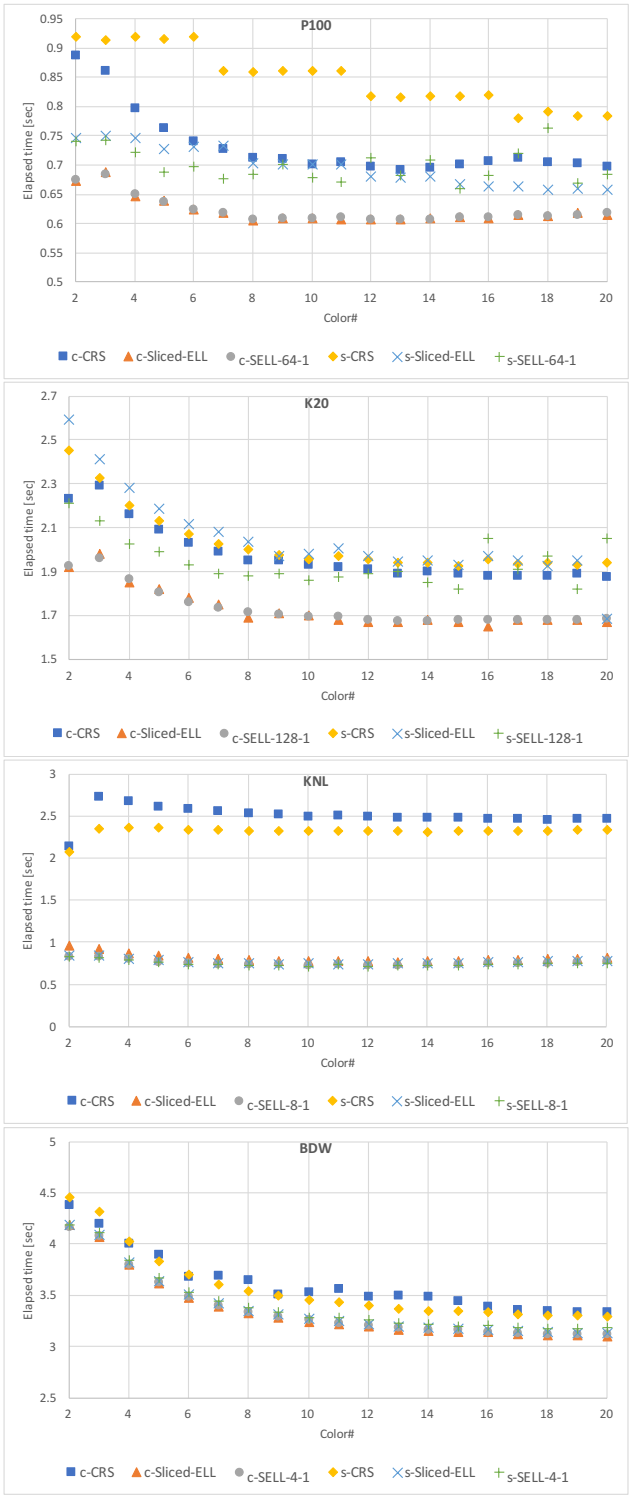

図 8　色数と計算時間の関係．上から P100，K20，KNL，BDW

るため，プロセッサのメモリバンド幅性能に比例して高速化することが期待される．表 3 中の相対速度，相対バンド幅性能は，それぞれ BDW の性能を 1 とした時の相対値を表しており，(相対速度)/(相対バンド幅性能) は高い方が優れている．従って，メモリバンド幅性能に対する実行性能としては BDW が最も優れており，K20，P100，KNL の順で効率が良いという結果になった．

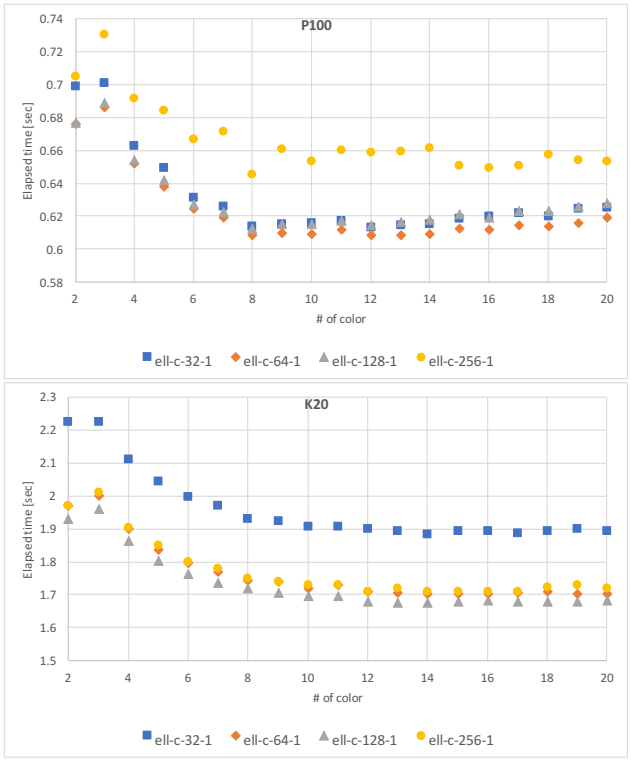

図 9　P100・K20 における SELL-C-σ の C 値と性能の関係．上）P100，下）K20．

表 3　実行時間とメモリバンド幅の関係

	P100	K20	KNL	BDW
最速実行時間 (sec)	0.605	1.652	0.723	3.102
相対速度 (BDW=1)	5.127	1.877	4.292	1
メモリバンド幅性能 (GB/s)	534	179	490	65.5
相対バンド幅 (BDW=1)	8.152	2.733	7.481	1
$\frac{\text{(相対速度)}}{\text{(相対バンド幅性能)}}$	0.629	0.687	0.574	1

6. P100・KNL 向けの最適化

5.5 節において述べた通り，BDW と比較して P100 と KNL の相対的な性能は十分とは言えない．本節では P100・KNL 向けの最適化を行う．

6.1 P100 におけるスレッド数の調節

5 節の実験においては，P100 で用いるスレッド数はコンパイラの決定に任せていたため，最適化の余地がある．図 **13** に，5 節の実験において最も高速であった c-Sliced-ELL（8 色）のスレッド数の調整（図 13 中：acc thread）を行なった結果を示す．この最適化により 9.0%程度の性能向上を得た．結果的に，リダクション部分のスレッド数調整（図 **10**）の効果が高く，それ以外の部分はコンパイラの選択が最適であった．具体的には，図 10 のように `gang(448)` とし，`verctor` の値はコンパイラに任せる方針とした．OpenACC の `gang` レベルでは，カーネルの終了以外に同期を取る方法がないため，リダクションを行うた

```
      RHO= 0.d0
!$omp do private(i) reduction(+:RHO)
!$acc kernels async(0)
!$acc loop independent reduction(+:RHO)
!$acc& loop gang(448) vector
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!$acc end kernels
```

図 **10** OpenACC Reduction 部分のスレッド数調整

めには一般的に，

(1) 同一 gang 内の vector 同士でローカルなリダクションを行い，gang 長の一時配列に各 gang でのローカルリダクション結果を記憶．

(2) gang(1) のカーネルを呼び，一時配列をリダクション．という 2 段階の手続きをとる．故に，gang を大きく取りすぎると 2 段階目のリダクションで取り扱う配列が大きくなり，並列性に乏しい gang(1) のカーネルの実行領分が大きくなるため非効率である．コンパイラに任せた場合，gang のデフォルト値は $(N-1)/(vector\_length)+1$ (N はループ長）となるのが一般的であり，実際に本稿で扱う問題設定の場合には gang(16384) が設定されていた．また，2 段階目のリダクションを高速化するために，gang(256) vector(256) などの設定も試みた．高速化は確認できたものの，gang(448) vector の方がわずかに高速であったためこちらを採用した．ここで，448 は P100 の SM 数 56 の 8 倍の値であり，K20 の最適値は異なると考えられる．このような値設定はプログラムの性能可搬性を損ねるため，実行プロセッサに合わせて自動的に最適な値が選ばれる仕組みが必要である．

6.2 KNL における OpenMP の同期削減

KNL・BDW 向けの OpenMP 実装は，OpenACC 版と同様の実装にて評価を行うため，図 7 に示したように，!$acc kernels 指示文と!$omp parallel do が一対一の対応となるようにした．しかし!$omp parallel 指示文と!$omp do 指示文を分割して用いる事により，スレッドの生成・破棄コスト，スレッド間の同期コストの低減が期待できる．例えば図 7 では，!$omp parallel を 4 行目の色ループの外に出す事により実現できる．実際，[6] で用いた実装では，この!$omp parallel を色ループの外に出す最適化が施されていた．

OpenMP 関連のオーバーヘッドを軽減するための最適化として，以下を順に適用した．

(1) (Baseline) 第 5 節の実装．

(2) (mv-parallel1) !$omp parallel の移動 1．(色ループの外，収束判定ループの内側)

(3) (nowait) 読み込み配列と書き込み配列が異なる，SpMV 部分に対する!$omp end do nowait の適用による同期の削減．

(4) (mv-parallel2) !$omp parallel の移動 2．(収束判定ループの外側)

(5) (rm-ompdo) リダクション部以外における，!$omp do に頼らない手動によるループ分割．(図 **11**)

(6) (rm-reduction) !$omp do reduction に頼らない手動によるリダクション実装．(図 **12**)

```
      ip = omp_get_thread_num()+1
      nth= omp_get_num_threads()
      ls = (N+nth-1)/nth
         ...
c$$$!$omp do private(i)
c$$$       do i= 1, N
      do i= (ip-1)*ls+1, min(ip*ls,N)
        W(i,Z)= W(i,R)
      enddo
c$$$!$omp end do
!$omp barrier
```

図 **11** 手動によるループ分割．c$$$で始まるコメント行が元の実装．

```
      ip = omp_get_thread_num()+1
      nth= omp_get_num_threads()
      ls = (N+nth-1)/nth
         ...
      W_RHO(ip)= 0.0d0
      do i= (ip-1)*ls+1, min(ip*ls,N)
        W_RHO(ip)= W_RHO(ip) + W(i,R)*W(i,Z)
      enddo
      RHO= 0.d0
!$omp barrier
      do i = 1, nth
         RHO= RHO + W_RHO(i)
      end do
```

図 **12** 手動によるリダクション実装．図 10 が元の実装．変数 RHO は OpenMP の private 変数としており，全てのスレッドが RHO を独自に計算する．

これらの最適化を，5 節の KNL の実行にて最も高速であった，s-SELL-8-1（12 色）に適用した結果を図 13 に示す．スレッド生成・破棄コスト，同期コストを軽減する，mv-parallel1, nowait, mv-parallel2 の最適化は大きな効果があり，結果として mv-parallel2 までの適用で 13.1%の性能向上を達成した．rm-ompdo, rm-reduction もわずかではあるが効果があり，全ての適用で 15.6%の性能向上を達成した．

一連の最適化による，収束ループ 1 イテレーションあたりの OpenMP 指示文出現数の変化を表 **4** にまとめた．暗黙の同期に関しては，parallel do 終了時に 1 回，parallel と do それぞれの終了時に 1 回，reduction 中（コア毎のローカルリダクションとグローバルリダクションの間）に 1 回，暗黙の同期が入るものとしてカウントしている．それぞれを図 13 と比較すると，KNL ではスレッドの生成・破棄，

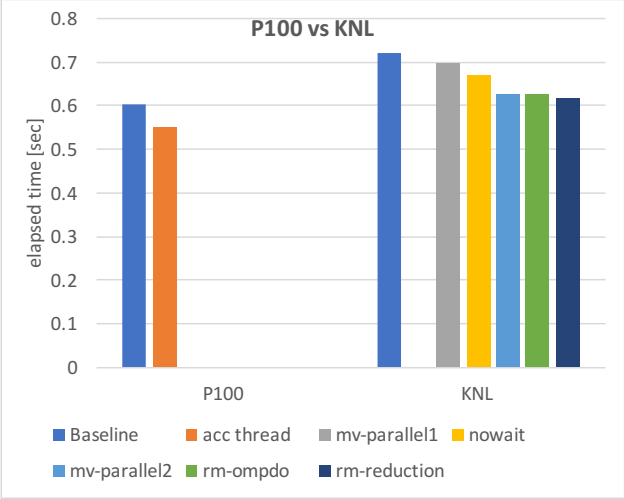

図 **13**　P100・KNL における最適化の効果

表 **4**　1 イテレーションあたりの OpenMP 指示文出現数（色数 12 のとき）．カッコ内の数字は上記最適化番号．

	(1)	(2)	(3)	(4)	(5)	(6)
parallel do	40	5	5	0	0	0
parallel	0	3	3	0	0	0
do	0	35	23	28	3	0
do (nowait)	0	0	12	12	0	0
reduction clause	3	3	3	3	3	0
barrier (explicit)	0	0	0	1	26	29
barrier (implicit)	43	46	34	31	6	0

同期にかかるコストが大きいことがわかる．

一方で，一連の最適化は従来の Xeon プロセッサにおいては必ずしも効果的でないことを確認した．図 **14** に同様の変更を BDW に適用した結果を示す．この結果は，s-SELL-4-1，12 色の時のものであり，OpenMP の指示文数などは表 4 と同様である．スレッドの生成・破棄を大きく現象させる mv-parallel1 の適用では 3%前後（色数を増やすことで効果増）の性能向上があったものの，他の最適化はほとんど効果がないか逆効果であった．

KNL はコア周波数が BDW と比較して低く，コア数は 68 と多いため，スレッドの生成・破棄コスト，同期コストが高くなるのは自然なことである．それに対し，メモリバンド幅性能は著しく向上したため，従来では見えてこなかったコストが表面化したものと考えられる．

7. おわりに

本稿では，疎行列連立一次方程式の解法として科学技術計算において広く使用されている，ICCG 法ソルバーを用い，NVIDIA 社の最新の Pascal 世代の GPU である P100 向けの性能評価・最適化を実施し，同世代のメニーコアプロセッサである Intel Xeon Phi (Knights Landing) 及び Kepler 世代の GPU との比較評価を行なった．これまでの研究において，メニーコアプロセッサ向けの行列格納手法として有効性が確認できた，ELL 形式の拡張である，

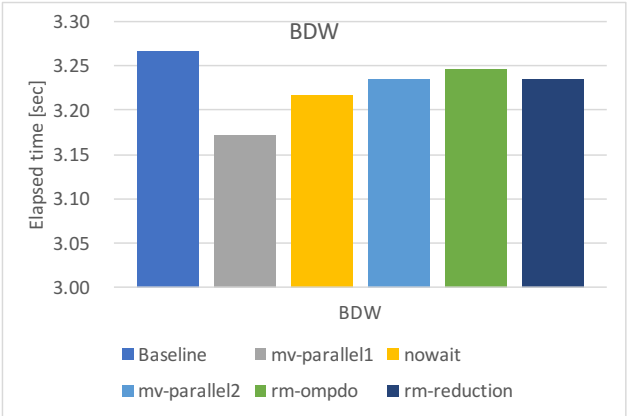

図 **14**　OpenMP オーバーヘッド削減，BDW での効果

Sliced-ELL，SELL-C-σ は，P100 においても有効な疎行列格納形式であることを確認した．

P100・KNL では，さらなるメニーコア化が進み，三次元積層メモリの搭載によりメモリバンド幅性能が大幅に向上したことで，新たな課題が表面化してきている．特に本稿では，KNL 向けの最適化として OpenMP の同期オーバーヘッド等の削減を行い，15.6%もの大幅な性能向上を達成した．

同期等のオーバーヘッドが増大した要因について，今後詳細に調べる必要があるが，相対的に低いコア周波数，コアの多数化などの要因を考慮すれば，同期のコスト増は自然な事である．メモリなどの性能が向上する一方で，同期コストなどが増加する傾向が今後も続くのであれば，チップ内であってもコア同士の同期を回避するようなアルゴリズム，それを容易に記述可能なプログラミングモデルなどが求められる事になる．

OpenMP の同期等のオーバーヘッドの調査，同期回避アルゴリズムの開発などについては今後の課題とする．

謝辞　本研究は JSPS 科研費 2611834 の助成を受けたものである．

参考文献

[1] The Green 500 List: `http://www.green500.org/`.
[2] Sodani, A., Gramunt, R., Corbal, J., Kim, H. S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R. and Liu, Y. C.: Knights Landing: Second-Generation Intel Xeon Phi Product, *IEEE Micro*, Vol. 36, No. 2, pp. 34–46 (2016).
[3] 大島聡史, 松本正晴, 片桐孝洋, 塙敏博, 中島研吾: 様々な計算機環境における OpenMP/OpenACC を用いた ICCG 法の性能評価, Vol. 2014-HPC-145, No. 21, pp. 1–10 (2014).
[4] Washio, T., Maruyama, K., Osoda, T., Shimizu, F. and Doi, S.: Efficient implementations of block sparse matrix operations on shared memory vector machines, *Proceedings of The 4th International Conference on Supercomputing in Nuclear Applications (SNA2000)* (2000).
[5] 中島研吾: 拡張型 ELL 行列格納手法に基づくメニィコア向け疎行列ソルバー, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2014-HPC-147, No. 3, pp. 1–8 (2014).

[6] 中島研吾, 大島聡史, 塙敏博, 星野哲也, 伊田明弘: ICCG 法ソルバーの Intel Xeon Phi 向け最適化, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol. 2016-HPC-157, No. 16, pp. 1–8 (2016).
[7] Kreutzer, M., Hager, G., Wellein, G., Fehske, H. and Bishop, A. R.: A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units, *SIAM Journal on Scientific Computing*, Vol. 36, No. 5, pp. C401–C423 (2014).
[8] Pascal Architecture Whitepaper: `http://www.nvidia.com/object/pascal-architecture-whitepaper.html`.
[9] TSUBAME 計算サービス: `http://tsubame.gsic.titech.ac.jp/`.
[10] Oakforest-PACS スーパーコンピュータシステム: `http://www.cc.u-tokyo.ac.jp/system/ofp/`.
[11] Reedbush スーパーコンピュータシステム: `http://www.cc.u-tokyo.ac.jp/system/reedbush/`.
[12] STREAM: Sustainable Memory Bandwidth in High Performance Computers: http://www.cs.virginia.edu/stream/.