jh170032-NAH

航空エンジンの翼列周り流れ解析のメニーコアシステム向け最適化

星野哲也（東京大学）

概要　近年主流となってきているメニーコアプロセッサを用いた計算機環境におけるアプリケーションの最適化は、より一層複雑化してきている。航空エンジンの翼列などの複雑形状の解析では特に、プロセス間の計算負荷のバランスが重要となる。本研究では、将来的な航空エンジンの解析に向け、解析のボトルネックとなる、疎行列連立一次方程式のメニーコアプロセッサ向けの高速化手法を検討した。

1. 共同研究に関する情報

(1) 共同研究を実施した拠点名

東京大学

(2) 共同研究分野

- ✓ 超大規模数値計算系応用分野
- □ 超大規模データ処理系応用分野
- □ 超大容量ネットワーク技術分野
- □ 超大規模情報システム関連研究分野

(3) 参加研究者の役割分担

星野哲也：全体統括,OpenACC+CUDA による Pascal GPU 向け最適化、ロードバランシング手法の開発

平川香林：シミュレーション実施、問題領域分割の最適化

塙敏博：プロセス間通信の最適化

大島聡史：KNL 向け最適化

河合直聡：Intel Broadwell-EP 向け最適化

青塚瑞穂：アプリケーションアドバイス

2. 研究の目的と意義

計算環境の大規模化・複雑化に伴い、アプリケーションの開発も複雑化している。特に近年主流となってきているメニーコアプロセッサに向けてアプリケーションを最適化するためには、アーキテクチャに関する知識を求められるため、数値計算を主として行う開発者にとっては難しい。さらに計算ノード間の通信を伴う複数メニーコアプロセッサを効率良く扱うための技術開発は、開発環境の構築の難しさもあり、一層困難を極める。株式会社 IHI が開発を進める UPACS_turbo は、航空エンジンの翼列周りの流れ解析などを目的とするアプリケーションであり、圧縮性完全気体の流れをセルセンター型有限体積法により解析する。対流項、粘性項、乱流、時間積分の計算部が実行時間の大部分を占め、乱流モデルに Spalart-Allmaras、時間積分には MFGS(Matrix Free Gauss Seidel)陰解法を用いている。また領域分割手法としては、複雑物体周りの計算格子の作成を容易にするために、構造格子を１ブロックとするマルチブロック構造格子法を採用しており、ブロック間は非構造に接続可能である。回転翼列においては、動翼列と静翼列が交互に並んでいるため非定常性のある流れとなり、翼列同士の相対的な回転を考慮する必要がある。UPACS_turbo では動翼列と静翼列周りをそれぞれのブロック群として分割し、ブロック群毎に単翼列として解析した上で、翼列同士の接続面に設けた新たなブロック群を仲介して情報の受け渡しをすることにより、翼列間の干渉を考慮した非定常な解析を可能としている。翼列の接続面に設けるブロック群は、大きさと形状、また処理内容が翼列周りのブ

ロック群とは異なるため、並列実行の際のプロセス間のロードバランシングをより難しくしている。提案者らのこれまでの研究では、相対的な回転を考慮しない単翼列周りの定常な流れ解析を行う 400 万格子点程度の比較的小さな問題を題材とし、単一の GPU(Kepler 世代）向けに OpenACC などを用いて UPACS_turbo を並列化し、評価を行ってきた。しかし実際の応用においては、動翼列と静翼列からなる多段翼列の相互の流れの干渉を考慮し解析することが重要であり、現実的な時間で結果を得るためには複数のメニーコアプロセッサを用いた並列化が必須である。そこで本課題においては、多段翼列解析のための 5,000 万格子点程度のデータセットを用い、NVIDIA 社の最新の Pascal 世代の GPU や、Intel 社の最新の Knights Landing (KNL)世代の Xeon Phi といった次世代のメニーコアプロセッサを用い、複数のメニーコアプロセッサを効率良く利用するための手法を開発し、UPACS_turbo の高速化を目指す計画であった。

3. 当拠点公募型共同研究として実施した意義

メニーコアプロセッサを扱うためには、アーキテクチャに関する知識が必須であり、計算科学と計算機科学のそれぞれの専門家による共同研究を行うことが好ましい。特に本課題では、プロセス間のロードバランシング、プロセス間の通信の最適化、データセットの分割方法の最適化など、アーキテクチャとアプリケーション双方に関する深い理解が必要であるために、本課題として実施する意義があった。また次世代のメニーコアプロセッサにはいくつかの選択肢があるため、様々な種類の計算環境を提供する本制度は我々の目的と合致していたために、本課題の申請を行った。

4. 前年度までに得られた研究成果の概要

本課題は今年度からの新規課題ではあるが、昨年度までの共同研究では IHI の所有する UPACS_turbo の高速化が進められていた。オリジナルの UPACS_turbo は MPI のみにより並列化されていたが、過去の共同研究においては乱流解析、粘性計算、移流計算など、シミュレーションに必要なほぼ全ての部分を MPI+(OpenMP or OpenACC+CUDA)により並列化し、Intel Xeon Phi や GPU を搭載したクラスタにおいて複数ノードによる並列実行可能なところまで開発を進めていた。本課題においてはこの成果を基とし、新しいメニーコアプロセッサ向けのさらなる最適化、複数プロセス実行向けの最適化などを行う予定であった。

5. 今年度の研究成果の詳細

IHI 側の事由により、本課題で UPACS_turbo を利用することができなくなってしまったため、UPACS_turbo でボトルネックとなっていた、疎行列の連立一次方程式解法のメニーコアプロセッサにおける高速化手法について検討した。

① 概要

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、疎行列の格納形式が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ことがわかっている。そこで本研究では、Reedbush スーパーコンピュータの GPU である NVIDIA Tesla P100 (P100)、 Oakforest-PACS スーパーコンピュータのプロセッサである Intel Xeon Phi Knights Landing (KNL) 向けの最適化・性能評価を実施し、比較評価を行なった。 結果として、ELL 形式の拡張である Sliced-ELL、SELL-C-σ が P100、KNL に双方において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 のベ

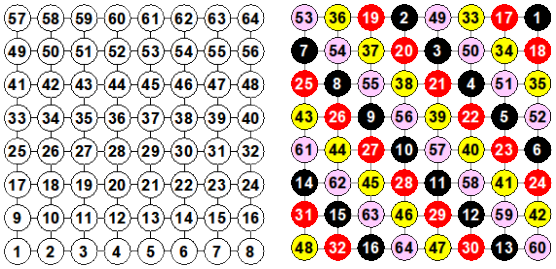

図 1 CM-RCM(k) 法による色付けとリオーダリング. 左) 元の疎 行列のオーダリング, 右) CM-RCM(4) 法によるリオーダリング.各色内の要素数は 16 でバランス.

クトル長の違いら、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。

② 計算環境

本研究において評価に使用した 3 種類の計算環境を表 1 に示す。表 1 中の BDW は、Reedbush スーパーコンピュータのプロセッサである、Intel Xeon E5-2695v4 の略称である。P100、KNL はそれぞれ Reedbush、Oakforest-PACS のものを用いている。表 1 中のメモリバンド幅性能は StreamTriad の実測値を示している。また 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。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 モードを使用している。

表 1 評価に用いたプロセッサのスペック

略称	P100	KNL	BDW
動作周波数(GHz)	1.48	1.40	2.10
コア数	3,584	68	18
理論演算性能(GFlops)	5,304	3,046	604.8
メモリバンド幅(GB/sec)	534	490	65.5

③ 実施ケースの概要

本研究では、ICCG 法による三次元のポアソン方程式ソルバーを対象とする。並列化に際しての色付け手法として CM-RCM(k)法 (図 1) を

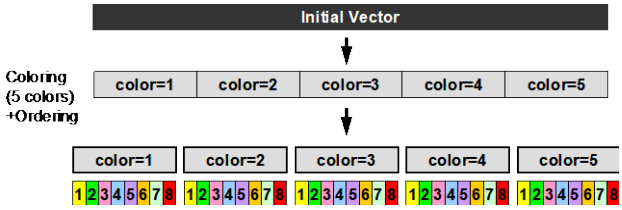

図 2 Coalesced 方式による要素番号割り付け

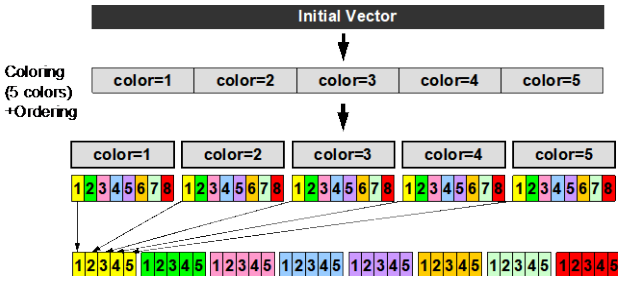

図 3 Sequential 方式による要素番号割り付け

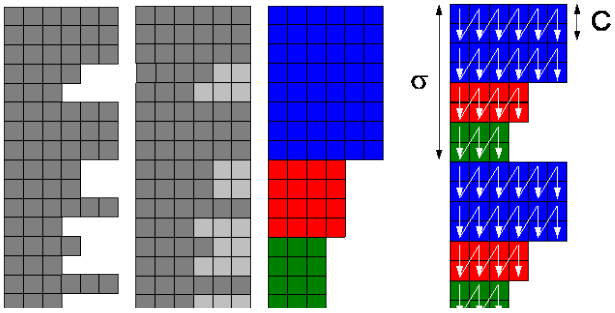

図 4 疎行列格納形式. 左から CRS, ELL, Sliced-ELL, SELL-C- σ (図は SELL-2-8) 形式

用い、Coalesced・Sequential 方式によるリオーダリング (図 2, 3)、CRS、SlicedELL、SELL-C-σ (図 4) による疎行列格納方式を適用する。実施ケースは c-CRS, c-Sliced-ELL, c-SELL-C-σ, s-CRS, s-Sliced-ELL, s-SELL-C-σ の 6 ケースであり、c-, s- は Coalesced, Sequential のリオーダリング方式を示す。なお、SELL-C-σ のパラメータは、σ = 1 であり、C は適宜変更している。対象問題の要素数は NX=NY=NZ=128 の総メッシュ数 2,097,152 である。P100 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。基本的には、OpenACC・OpenMP の指示文を無視すれば等価なプログラムとなるよう実装している (図 5)。 P100 向けの OpenACC 実装について説明する。図 5 の 1 行目に現れる!$acc data において、デバイ

```
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 5 前進後退代入部(実施ケース:表 2 s-Sliced-ELL)の Ope- nACC・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

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。

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.

次に、KNL・BDW 向けの OpenMP 実装について説明する。図 5 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、1 コアあたり 4 スレッド、最大 272 スレッドの実行が可能であるが、Oakforest-PACS の計算ノードでは、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

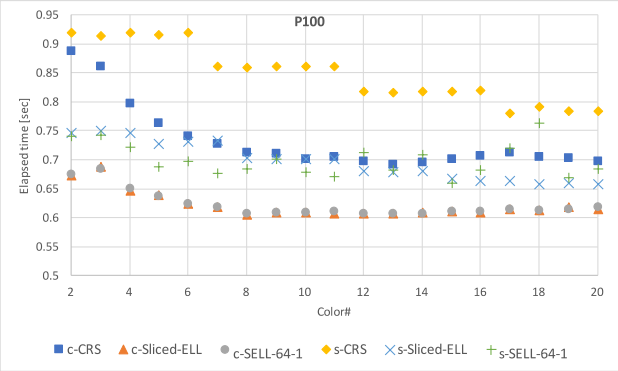

図 6 P100 における各ケースの色数と実行時間の関係」

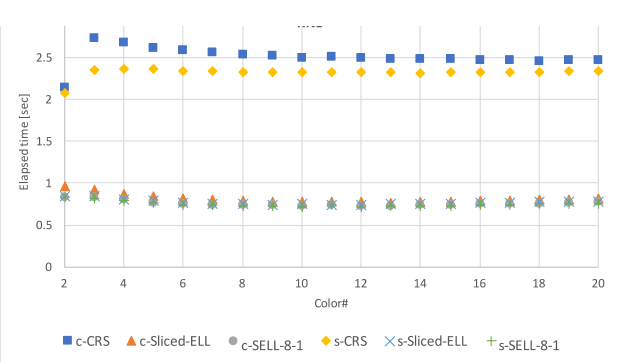

図 7 KNL における各ケースの色数と実行時間の関係

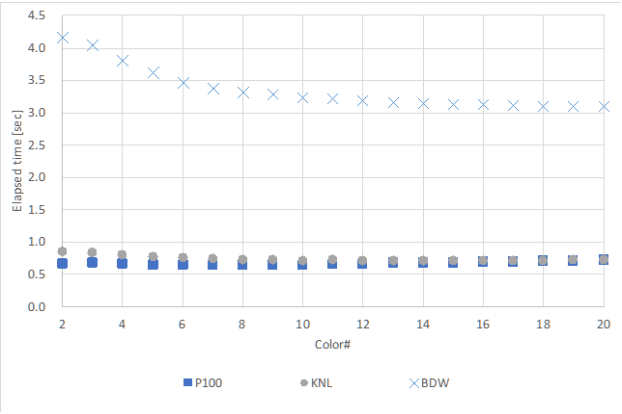

図 8 プロセッサ毎の最速ケースの比較

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 –xAVX2 -qopenmp をオプションとして設定した。

④ 性能評価

各実施ケースにおいて、色数を変化させた際の実行時間の変化について調べた結果が図 6・7 である。 P100 においては c- で始まる Coalesced 版が高速である一方で、KNL では s- から始まる Sequential 版 が高速な傾向であった。疎行列形式として最も一般的と言える CRS 形式に着目すると、P100 では最速ケースと比較して概ね 3 割程度低速であるのに対し、KNL では 2.5 倍ほど低速であった。また、各プロセッサの性能比較が図 8 である。各プロセッサにおいて最速の実施ケースによる実行結果である。具体的には、P100 は c-SELL-C-σ の C = 64 としたもの、KNL は s-SELL-C-σ の C = 8、BDW は C = 4 であり、いずれも σ = 1 である。C の値はプロセッサの SIMD 幅に依る。P100 と KNL を比較すると、概ねメモリバンド幅の比相当の結果であるため、妥当であると言える。 BDW の結果を見ると、色数を増やすごとに実行時間が小さくなっていることがわかる。色数を増やした場合収束性が良くなり、図 5 における収束判定ループの回転数が少なくなるため、BDW の振る舞いは自然である。一方で、色数を増加すると 4 行目の色ループが増加し、21-22 行目の !$omp end parallel do または!$acc end kernels で発生する同期コストが増加する。P100・ KNL で色数増により実行時間が変わらない、もしくは悪化している原因は、この同期コストによる。メニーコアプロセッサは文字通り多数のコアを並列に実行することにより性能を稼いでおり、多数の同期コストは相対的に大きくなるため、実行時間の増加は自然である。

⑤ P100・KNL 向け性能最適化

前述の通り、P100・KNL では同期コストが大きいことがわかっている。そこで主に同期コストに着目し、以下の性能最適化を施した。

P100 向け最適化

1. Baseline：全ての並列ループに!$acc kernels を適用
2. Async：!$acc kernels を!$acc kernels async(0)へ置き換え
3. Thread：gang, vector といった、スレッド数を調整するパラメータを最適化
4. Fusion：カーネルの融合による、暗黙の同期コストの削減

KNL・BDW 向け最適化

1. Baseline ：全ての並列ループに omp parallel do
2. mvparallel1：!$omp parallel を色ループ（図 14 行目）の外側に移動
3. nowait：!$omp end do nowait を SpMV 部分に適用し、暗黙の同期コストの低減
4. mvparallel2：!$omp parallel を収束判定ループ（図 12 行目）の外側に移動
5. rmompdo：!$omp do を使わずに、機械的にループ分割（reduction 部を除く）
6. rmreduction：reduction 節を用いず、手動により実装する
7. loopscheduling：アプリケーション固有の知識を用いたループスケジューリングにより、同期 コストを削減

最適化の結果を図 9・10 に示す。P100 においては、async の効果が大きいことがわかる。async 節を付けない場合、ホストプロセスはデバイスプロセスの終了を待ち、その後に次のデバイスプロセスを発行するため、この発行コストが大きな影響を及ぼしている。Fusion まで順次適用することで、色数増により高速化していることから、同期コストの低減に成功していることがわかる。KNL 向けの最適化では、OpenMP 指示文を利用した時に生じる、暗黙の同期の発生などのオーバーヘッドを抑制している。順次適用により、実行時間が減少していることがわかる。以上の通り、P100・KNL などの最新のメニーコアプロセッサにおいては、三次元積層メモリの搭載による大幅な高速化が見込める一方で、相対的に同期コストが大きくなっており、最適化の必要性が あることがわかった。同期回避のアルゴリズムの開発などが今後の課題である。

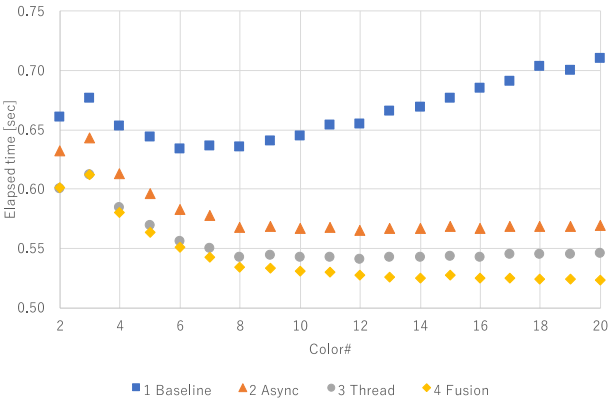

図 9 P100 向け最適化の効果

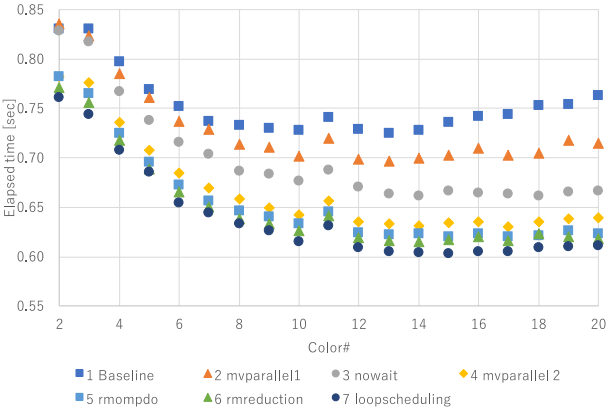

図 10 KNL 向け最適化の効果

6. 今年度の進捗状況と今後の展望

本年度は、エンジン周りの解析を行う際に必要となるであろう、疎行列連立一次方程式のメニーコアプロセッサ向けの高速化について研究した。今後は、本研究成果を実際の解析で役立てることを目指したい。

7. 研究成果リスト

なし