

Xeon Phi KNL プロセッサを搭載したスーパーコンピュータ Oakforest-PACS における OpenFOAM の高速化

今野 雅 [1†]

[1] 株式会社 OCAEL・東京大学情報基盤センター

Acceleration of OpenFOAM in the supercomputer Oakforest-PACS equipped with Xeon Phi KNL processors

Masashi IMANO[*†]

[*]OCAEL Co. Ltd. & Information Technology Center, The University of Tokyo

Abstract

In order to speed up OpenFOAM in the supercomputer Oakforest-PACS equipped with Xeon Phi KNL processors, vectorization improvement patch, vectorization of SpMV and conjugate gradient method by Chronopoulos/Gear algorithm were applied. As a result of evaluating the performance of the channel flow analysis, the speed up ratio was up to 20% compared with the original OpenFOAM. In addition, the combination of the above methods showed a stable speed improvement ratio of about 10% to 15% independent of number of processes.

Keywords: HPC, Xeon Phi, Vectorization, Conjugate gradient method, Oakforest-PACS, OpenFOAM

1. はじめに

東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営するスーパーコンピュータ Oakforest-PACS のプロセッサは，各コアが 512-bit 幅のベクタユニットを 2 基備える Intel Xeon Phi であるが，標準の OpenFOAM ではベクトルユニットを活用できず，性能が発揮できない問題があった．本研究では，ベクトル化を促進させる性能改善を施し，オープン CAE 学会の OpenFOAM ベンチマークであるチャネル流で性能評価を行なった．また，大規模並列解析時には，線型ソルバにおける内積の計算に伴なう MPI の集団通信のコストがボトルネックとなるため，通信回数削減型の共役勾配法についても併せて検討を行なった．

2. スーパーコンピュータ Oakforest-PACS の概要

Oakforest-PACS システム (以下 OFP) は計 8,208 台有する計算ノードを有するが，メニーコア型の Intel Xeon Phi 7150 (開発コード名：Knights Landing，以下 KNL) をプロセッサとして使用している [1]．OFP の KNL には 68 個の物理コアが搭載され，各コアが 512-bit 幅のベクタユニットを 2 基備えており，各ベクタユニットは 1 命令で 8 つの倍精度演算を行う SIMD 拡張命令の AVX-512 を実行可能である．KNL のコアの動作周波数は通常時 1.6GHz と低いため，ベクタユニットを活用しないと KNL の性能が発揮できないという特徴を持つ．

3. OpenFOAM のベクトル化阻害要因

OpenFOAM は主に有限体積法を用いて，解くべき支配方程式の離散化を行うが，格子の形状としては，任意面数のポリヘドラル (多面体) を用いる事が可能である．このため，OpenFOAM では，格子間の界面が，界面を保有する格子と，それに隣接する格子という 2 つの格子へのポインタを持つという，界面ベース探索 (アドレッシング) の格子格納形式を採用している．OpenFOAM が用いている有限体積法では，SIMPLE や PISO といった方程式のカップリングアルゴリズム，時間・空間の離散化スキーム等を用い，解くべき支配方程式を線型の連立方程式に変換し，最終的に線型ソルバによって解く．さらに，格子の格納形式が界面ベースであることから，

[†] *E-mail address of corresponding author: imano@ocael.co.jp*

連立方程式の係数行列も界面のアドレッシングによって作成する．また，OpenFOAM は，空間的に高々 2 次精度の離散化スキームを使用するため，界面でのフラックスを離散化した結果生じる係数行列の非零成分が，上対角成分 (以下，U) と下対角成分 (以下，L) にひとつずつ組で存在することになる．このため，OpenFOAM では，U と L を界面ベースのアドレッシングで格納しており，さらに対角成分 (以下，D) を格子のアドレッシングで格納する方式 (lduAddressing および lduMatrix クラス) を採用している．この格納方式は，D，L，U を分離して，L と U を Coo(Coordinate) 形式とした変形 COO 形式であり [2, 3]，単純に係数行列が生成可能で，対称行列では格納データとメモリアクセスを減少させることが可能である利点を有する．一方，疎行列ベクトル積 (以下 SpMV) において，U と L の成分に関する行列ベクトル積を行うコードを Code 1に示すが，行列成分の参照アドレスである `uPtr[face]` や `lPtr[face]` が界面ループ内で重複するので，ベクトル化ができない．また，2 重の間接参照であり，メモリアクセスがランダムとなり，高いメモリバンド幅が必要となる欠点を有する [2, 4]．

Code 1　Part of SpMV source code lduMatrixATmul.C.

```
1     for (label face=0; face<nFaces; face++)
2     {
3         ApsiPtr[uPtr[face]] += lowerPtr[face]*psiPtr[lPtr[face]];
4         ApsiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];
5     }
```

その他にも，AVX-512 用の最適化オプションをコンパイル時に指定しても，線型ソルバなどで多用されるベクトル内積の `sumProd` など，高速化されない実装が多く存在する [4]．

4. 性能評価

4.1. 高速化手法

検討した高速化手法は以下の通りである．

KNL　Intel コンパイラを用いる場合の標準の設定 `WM_COMPILER=Icc` ではなく，`WM_COMPILER=IccKNL` の設定でコンパイルする手法．コンパイラのオプションに`-xmic-avx512 -DvectorMachine` 等が加わる．

VEC　上記に加え，OpenFOAM-Intel[5, 6] のベクトル化改良パッチを適用する手法．上記で `vectorMachine` を定義すると，線型ソルバなどで多用されるベクトル内積の `sumProd` などは，ベクトル化が容易なループ構造に変更されるが，このパッチは，これらについて simd プラグマや ivdep プラグマを用いて，コンパイラに SIMD 化を指示するなどのベクトル化改良を行う．

CG　圧力方程式の線型ソルバとして，Chronopoulos/Gear アルゴリズムによる共役勾配法を使用する手法．OpenFOAM の PCG では，反復ループ内に内積の計算が 2 箇所にあるが，このアルゴリズムでは 1 箇所にまとめられているので，多並列 MPI 計算で大きな比重を占める MPI 集団通信のコストを下げることが可能である [7, 8]．なお，MPI 集団通信について，MPI-3 で採用された非同期集団通信 (MPI_Iallreduce) を使用し，パイプライン型などの共役勾配法を用れば，超大規模 MPI 並列では集団通信コストを隠蔽・回避することも可能だが，本検討では MPI プロセス数が高々 1,024 と小さいので検討しなかった [7, 9]．

SpMV　初期に係数行列の参照アドレスを CRS(Compressed Row Storage) 形式に変換しておき，線型ソルバにおける SpMV を CRS 形式で計算する手法．SpMV がベクトル化される．

4.2. 性能評価結果

上記に述べた高速化手法を用いた OpenFOAM を用い，オープン CAE 学会の OpenFOAM ベンチマーク [10] であるチャネル流を対象に性能改善の評価を行なった．計算条件を表 1に示す．

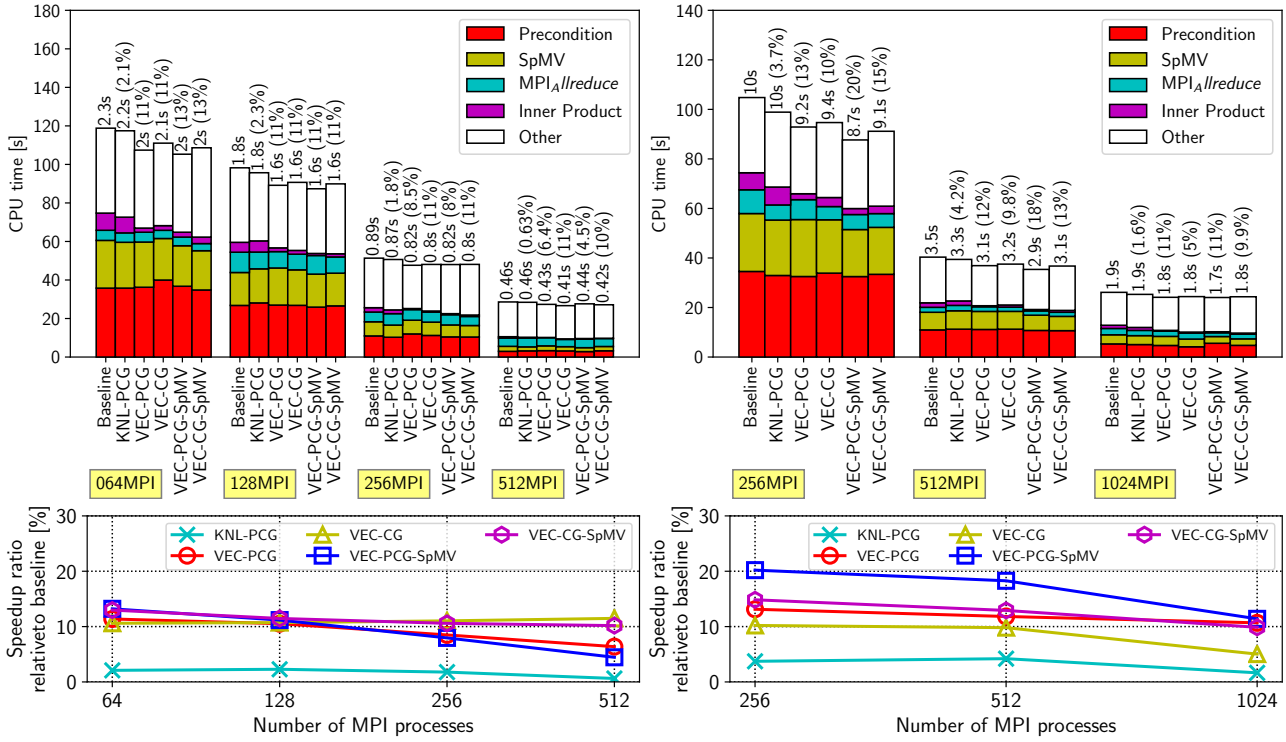
図 1の上図に，Intel VTune Amplifier プロファイラを用いて実行した場合での，主要なホットスポットを棒グラフで示す．また，初期ステップ以降のステップ毎の平均 CPU 時間 [s] と，標準の OpenFOAM-v1612+ のソースを `WM_COMPILER=Icc` の設定でコンパイルした baseline ケースに対する高速化率 [%] を，各ケースの棒グラフの上部に示す．また，高速化率 [%] は下図にも示した．なお，ステップ毎の平均 CPU 時間の計測時にはプロファイラを用いていない．また，ステップ毎の平均 CPU 時間は，格子数 3M では初期ステップから 51 ステップ間，格子数 24M では初期ステップから 9 ステップ間の `ExecutionTime` の差より求めた．

Table 1　解析条件

対象問題	オープン CAE 学会 OpenFOAM ベンチマーク [10] のチャネル流*1
計算機	JCAHPC Oakforest-PACS，Flat メモリモード (`numactl -p 1` を使用)
解析ソルバ	pimpleFoam (OpenFOAM v1612+)
MPI プロセス数/ノード	64 (領域分割方法: scotch)
格子数・解析ノード数	格子数 3M ($240 \times 130 \times 96$) の時，1, 2, 4, 8 ノード
	格子数 24M ($480 \times 260 \times 192$) の時，4, 8, 16 ノード
コンパイラ・MPI ライブラリ	Intel Compiler 2018.3.222, Intel MPI 2018.3.222*2
圧力方程式線型ソルバ	共役勾配法 (PCG)．前処理：不完全コレスキー分解 (DIC)

*1 摩擦速度 Re_τ=110，乱流モデル無し．

*2 `I_MPI_FABRICS=shm:tmi` と `I_MPI_DYNAMIC_CONNECTION=0` を設定．最初のタイルを除いてプロセスを各物理コアに配置した．



(a) 3M mesh　　(b) 24M mesh

Fig. 1　CPU time of main hotspots ,average CPU time per step and speedup ratio relative to baseline. In the above figures, average CPU time per step [s] and speedup ratio [%] are shown above bars.

標準のソースのまま，AVX-512 用のコンパイラオプションを付けてコンパイルし，線型ソルバとして標準の PCG を用いた KNL-PCG ケースでは，baseline に対する高速化率が格子数 3M で約 1〜2%，格子数 24M で約 2〜4% に留まった．また，主要なホットスポットについても，ほとんど減少していない．

ベクトル化改良パッチを適用した VEC-PCG ケースでは，KNL-PCG ケースに比べ，ベクトル内積 Inner Product(`sumProd`) の CPU 時間が大幅に減少した．また，最大の高速化率が格子数 3M で約 6〜11%，格子数 24M で約 11〜13% となり，ある程度高速化されるが，並列数の増加に伴ない高速化率が減少した．

ベクトル化改良に加え，圧力方程式の線型ソルバとして Chronopoulos/Gear アルゴリズムによる共役勾配法 CG を使用した VEC-CG ケースでは，VEC-PCG ケースに比べ，MPI 集団通信 MPI_Allreduce の CPU 時間が僅かに減少した．また，最大の高速化率が格子数 3M で約 11%，格子数 24M で約 5〜10% となり，MPI 通信の比重が大きい格子数 3M では並列数が上昇しても高速化率が減少しなかった．一方，MPI 通信の比重が小さい格

子数 24M では，VEC-PCG より低速となった．

SpMV を CRS 形式で計算する VEC-PCG-SpMV ケースでは，VEC-PCG ケースに比べ，SpMV の CPU 時間が減少した．高速化率が格子数 3M で約 5〜13%，格子数 24M で約 11〜20% となり，他のケースに比べ並列化数が小さい場合の高速化率が高いが，並列化数の増加に伴ない高速化率が急激に減少した．

圧力方程式の線型ソルバとした CG を用い，さらに SpMV を CRS 形式で計算する VEC-CG-SpMV ケースでは，最大の高速化率が格子数 3M で約 10〜13%，格子数 24M で約 10〜15% となり，並列数にあまり依存せず安定した高速化率が得られた．

以上の検討結果から，並列数が小さい場合には VEC-PCG-SpMV ケース，多い場合には VEC-CG-SpMV ケースを用いることで，概ね高い高速化が得られることがわかった．

5. まとめ

本報では，OpenFOAM のベクトル化を促進させる性能改善パッチを適用した上で，係数行列格納方式の CRS 形式変換による SpMV のベクトル化を行い，圧力方程式の線型ソルバとして Chronopoulos/Gear アルゴリズムによる共役勾配法 CG を用いるなど，各種高速化手法の組み合わせを検討した．メニーコア型の Xeon Phi KNL プロセッサで構成される Oakforest-PACS において，チャネル流を対象に性能評価を行なったところ，標準の OpenFOAM に比べ，最大で約 20% の高速化率が得られた．また，ベクトル化と CG を組み合わることで，並列数にあまり依存せず，約 10〜15% の安定した高速化率が得られた．今後は，メモリアライメントによるベクトル化向上，intrinsics 関数による高速化 [6]，Chronopoulos/Gear 共役勾配法での DIC 前処理の高速化 [8]，メニーコアプロセッサに適した前処理の検討などが課題である．

参考文献

[1] 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部門. https://www.cc.u-tokyo.ac.jp/, (accessed 2018-11-20).

[2] Mottaqiallah Taouil. A hardware accelerator for the openfoam sparse matrix-vector product. Master's thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2009. https://repository.tudelft.nl/islandora/object/uuid:ce583533-45ea-4237-b18d-fe31272ea1ee, (accessed 2018-11-20).

[3] 伊東聰. ppOpen-AT における OpenFOAM 自動最適化への取り組み. オープン CAE 学会オープン CAE ワークショップ 2012, 2012. http://www.opencae.or.jp/activity/workshop/opencae_workshop2012/, (accessed 2018-11-20).

[4] 井上義昭.「京」における openfoam の性能評価. 平成 26 年度「京」における高速化ワークショップ, 2014. http://www.hpci-office.jp/pages/ws_kei_141219, (accessed 2018-11-20).

[5] OpenFOAM-Intel Repository. https://github.com/OpenFOAM/OpenFOAM-Intel, (accessed 2018-11-20).

[6] N. Agrawal, P. Edwards, R. Ojha, A. Pandey, and P. Pawar. Performance Optimization of OpenFOAM on KNL. IXPUG Workshop, Application Performance on Intel Xeon Phi - Being Prepared for KNL and Beyond, 2016. https://www.ixpug.org/events/ixpug-isc-2016, (accessed 2018-11-20).

[7] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Comput.*, Vol. 40, No. 7, pp. 224–238, July 2014. http://dx.doi.org/10.1016/j.parco.2013.06.001, (accessed 2018-11-20).

[8] 内山学, ファム バン フック. 非構造格子に対応した PCG 法の thread 並列化手法. Technical report, 2016. https://www.ipsj.or.jp/kenkyukai/event/hpc153.html, (accessed 2018-11-20).

[9] 今野雅. オープンソース CFD コード OpenFOAM の性能評価および高速化に関する研究. Technical Report 19, 東京大学情報基盤センター年報, 2018. https://www.itc.u-tokyo.ac.jp/public/annual_report/, (accessed 2018-11-20).

[10] オープン CAE 学会 V&V 小委員会 OpenFOAM BenchmarkTest レポジトリ. https://gitlab.com/OpenCAE/OpenFOAM-BenchmarkTest/, (accessed 2018-11-20).