Vol.2018-HPC-167 No.3
2018/12/17

マルチグリッド法におけるファイルを介したメッシュ生成プロセスの高速化

塙 敏博[1,a)]　中島 研吾[1,2]

概要：Intel Xeon Phi による大規模クラスタ Oakforest-PACS でマルチグリッド前処理付き並列 CG ソルバーを実行する際のファイルを介した並列メッシュ生成プロセスの性能改善について検討した。マルチグリッド法のメッシュ生成プロセスにおいて、並列メッシュ生成プログラムにより作成したメッシュデータファイルをソルバーが読み込んで動作するが、計算時間に比べてファイル入出力時間がボトルネックになっていた。そこでファイル入出力を MPI-IO に置き換え、バッファリングを適切に行うことで、最大 200 倍以上の性能向上を得た。またバーストバッファ技術を用いたファイルキャッシュによる性能改善も試みた。

1. はじめに

マルチグリッド法は、線形方程式の求解やクリロフ部分空間法の前処理に用いられており、高い並列性を実現できるため大規模な問題に適した解法である。従って並列マルチグリッド法は、ポストペタ・エクサスケール時代における有用な解法として期待されている。近年、スーパーコンピュータシステムの実用的な性能を評価するためのベンチマークプログラムとして HPCG (High Performance Conjugate Gradients) [1] が用いられている。実際の HPCG では、有限要素法アプリケーションから導出される疎行列を、マルチグリッド法による前処理付きの CG (Conjugate Gradient) 法で解いている。

これまでに、筆者らは不均質な多孔質媒体中の三次元地下水流れを有限体積法（Finite Volume Method，FVM）を用いて解く pGW3D-FVM アプリケーションにおいて、OpenMP+MPI ハイブリッド並列によるマルチグリッド法による前処理を施した共役勾配法を適用してきた [2]。その結果、東京大学情報基盤センターで運用されていた富士通 PRIMEHPC FX10 (Oakleaf-FX) を用いて 4,096 ノード (65,536 コア) まで Weak scaling, strong scaling のいずれについても高い並列性を示すことが確認された。

一方、pGW3D-FVM においては、あらかじめ並列メッシュ生成プログラムによってマルチグリッドの各レベルに対応したメッシュデータファイルを生成し、それをソルバーが読み込むことで動作する。これによって、一度生成されたメッシュデータを再利用することで、ソルバーにおけるメッシュ生成のコストを短縮することができる。

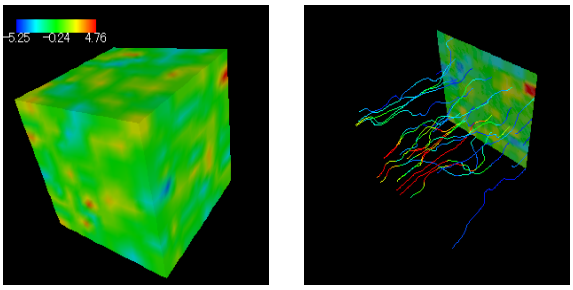
しかし、ソルバーにおける計算時間が短縮されるにつれて、メッシュデータの生成プロセスがボトルネックになってくる。さらに、Intel Xeon Phi(Knights Landing: KNL) のようなメニーコアプロセッサでは、コアの動作クロックが低いため、SIMD 演算などで高速化可能な計算に対して、逐次性能に頼る IO 処理は相対的に遅くなるため、その傾向が一層顕著になる。

そこで本研究では、Intel Xeon Phi による大規模クラスタ Oakforest-PACS でマルチグリッド前処理付き並列 CG ソルバーを実行する際のファイルを介した並列メッシュ生成プロセスの性能改善について検討する。Oakforest-PACS では、並列ファイルシステムとして Lustre ファイルシステムだけでなく、バーストバッファ技術を用いた高速ファイルキャッシュシステムとして IME (Infinite Memory Engine) を導入している。ファイル入出力には MPI-IO を用いることで、IME の native な利用や、全 MPI プロセス間での単一共有ファイル、Lustre ファイルシステムの特性を活かしたファイルアクセスなどが実現できる。

2. マルチグリッド法を用いた pGW3D-FVM

本研究では、**図 1** に示すような、不均質な多孔質媒体中の三次元地下水流れを並列有限体積法（Finite Volume Method，FVM）によって解くアプリケーション pGW3D-FVM を扱う [2]。対象とする問題は不均質場におけるポアソン方程式である。各メッシュは立方体で差分格子のような規則的な配列である。pGW3D-FVM から導かれる係数行列からなる連立一次方程式を，マルチグリッド法による

1　東京大学 情報基盤センター
2　理化学研究所 計算科学研究センター
a)　hanawa@cc.u-tokyo.ac.jp



(a) (b)

図 1 不均質な多孔質媒体中の三次元地下水流れの例：(a) 透水率の分布 (b) 流線

前処理を施した共役勾配法（Conjugate Gradient Method, CG）によって解く。このような前処理付き共役勾配法を MGCG 法と呼ぶ。8 つのメッシュから 1 つのメッシュを生成するような V-Cycle に基づく幾何学的マルチグリッド法を適用している。最も細かい格子レベル（level）を 1 とし，粗くなるにつれてレベルが増加する。最も粗いレベルにおけるメッシュ数は各 MPI プロセスにおいて 1 となる。ブロック Jacobi 型局所 IC（0）法を加法シュワルツ法と組み合わせた緩和演算子（smoothing operator）を各レベルにおいて適用している。

2.1 Coarse Grid Aggregation (CGA) 法

V-cycle のマルチグリッド法を適用した例を**図 2** に示す。各レベルではマルチグリッドの処理は並列に行われるが、最も粗いレベルでの処理 (coarse grid solver) では全プロセスの情報を 1 プロセスに集める必要がある。従って、coarse grid solver におけるメッシュ数は MPI プロセス数に等しくなる。しかし、計算時間に比べて通信のオーバヘッドは V-cycle のより粗いレベルで顕著になる。

そこで、**図 3** に示すように、より細かいレベルで MPI プロセスの情報をまとめて coarse grid solver を使うようにしたのが CGA (Coarse Grid Aggregation) 法である。これにより、coarse grid solver で解くべきサイズが大きくはなるが、通信オーバヘッドが減らせるだけでなく、収束の安定性が向上する。また、coarse grid solver は OpenMP 並列化を施し、全 MPI プロセスで冗長に実行することでさらに通信コストを減らすことができる。

粗いレベルでの通信コストを一層低減するため、すでに階層型 CGA も提案されている [2] が、今回は使用していない。

2.2 並列メッシュ生成

マルチグリッド法を用いる pGW3D-FVM において、マルチグリッドの各レベルで MPI プロセスごとにメッシュ情報が必要となる。メッシュ情報には、該当のレベル・MPI プロセスにおけるメッシュの格子点数、隣接ノードの MPI ランク番号、さらに各格子点において、ノード番号、境界

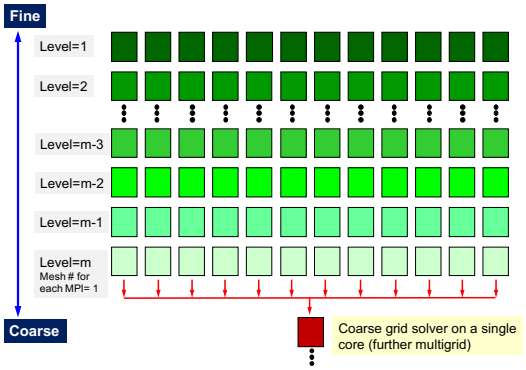


図 2 マルチグリッド法 (V-cycle, restriction の場合)

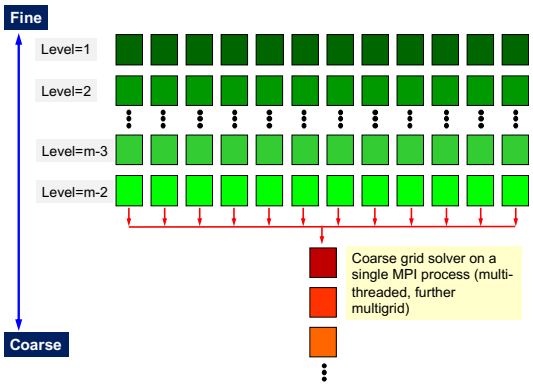


図 3 CGA (Coarse Grid Aggregation) の手順：Level=$m-2$ での各 MPI プロセスの結果を 1 プロセスに集めてしまう（実際には全プロセスが冗長に持つ）

に当たるか否か、親（次のレベルに対応する）のノード番号、その格子点における透水率などが含まれる。

図 4 に、並列メッシュ生成プログラムとソルバーとの関係を示す。図中の並列メッシュ生成プログラム “pmg” に対して、メッシュ生成のパラメータ情報として “ppp.dat” に格子点数と x, y, z 方向の各プロセス数、透水率の分布データ “porsim1.out” を与えることで、メッシュデータ “pmk.level.my_rank” のファイルが生成される。“grid.dat” には、各レベルでのメッシュ数の情報が記録される。“work.lst” は各レベルでのファイルサイズを示したものであり、確認のために作成される。“workfil.level.my_rank” は一時ファイルであり、pmg 終了後に削除してよい。

その後、ソルバープログラム “solver” では、実行条件を制御する “INPUT.DAT” ファイルとメッシュ情報の grid.dat を読み込んだ後、pmk.level.my_rank ファイルを各ランクが読み込み、マルチグリッド法による前処理を行う。

3. Oakforest-PACS とファイル IO

本章では、最先端共同 HPC 基盤施設 (JCAHPC) において運用されている Oakforest-PACS について述べる [5]。本研究ではファイル IO に着目するため、主にストレージについて述べる。

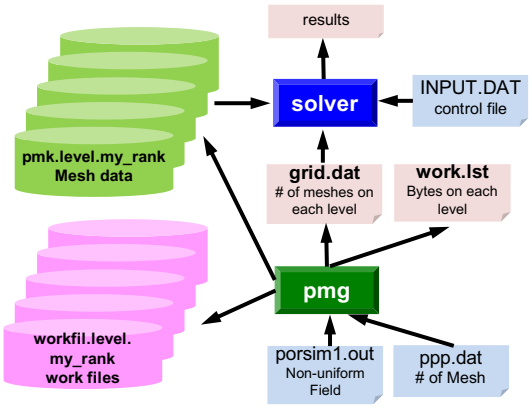


図 4　pGW3D-FVM における並列メッシュ生成プログラムとソルバーとの関係

3.1　OFP のストレージ

3.1.1　構成

OFP では、並列ファイルシステムとして、容量 26 PB の Lustre ファイルシステム [6] が採用されている。**図 5** に並列ファイルシステムを構成するラック群の一部を示す。ファイル管理情報を提供するメタデータサーバ (MDS) を 3 セット、ファイルの実体を保存するオブジェクトストレージサーバ (OSS) をそれぞれ 10 セット 40 台用意することで高いファイル入出力性能を実現している。理論ピークバンド幅は 500 GB/s となる。

さらに OFP では、ストレージ性能向上のため、高速ファイルキャッシュシステムとして、DDN 社 Infinite Memory Engine (IME) を導入している [7]（**図 6**）。25 台の IME14KX を用いて構成しており、各 IME14KX は 48 台の 800 GB NVMe SSD を搭載し、100 Gbps の Omni-Path ネットワーク 8 本で接続されている。各 NVMe SSD のデータ転送性能を 1,300 MB/s として，ファイルキャッシュシステム全体での理論ピークバンド幅は 1,560 GB/s となる．全体の物理容量は 960 TB であるが、10D+1P の erasure coding を行うため、利用可能な容量は約 864 TB となる。Lustre ファイルシステムのキャッシュとして扱うことができ、アプリケーションの IO 性能を大きく改善することが可能である。

並列ファイルシステム、高速ファイルキャッシュシステムは、前述の OPA を経由して全てのログインノード・計算ノードと接続されている。

3.2　IO 性能

表 1 に、2018 年 11 月の IO500 性能 (ただし easy のもののみ) を示す [8]。この結果より、IOR easy write の場合には IME が Lustre ファイルシステムより 7.67 倍性能が高いことがわかる。詳細については文献 [9] に詳しい。

なお、これらの結果は、

図 5　並列ファイルシステム (中央 3 ラックと同じものが他に 7 ラックあり、合計で 10 ラック)

図 6　高速ファイルキャッシュシステム

表 1　IO500 における IOR easy, mdtest easy 性能

	IOR easy (GiB/s)		mdtest easy (kIOP/s)	
	Read	Write	Stat	Write
IME	664.47	744.76	202.01	37.02
Lustre	121.77	97.08	152.55	58.46

3.3　MPI-IO によるファイル IO と IME

MPI-IO は、MPI Forum によって標準化が行われている Message Passing Interface 仕様において規定されている、ファイル IO のための仕様である [3]。MPI-IO では、通常のファイル IO(ここでは POSIX IO と呼ぶ) では実現できない、柔軟なファイルアクセス制御、移植性や、並列ファイルシステムの特性に合わせた性能最適化を可能にしている。代表的な MPI-IO 実装である ROMIO では、特に様々な並列ファイルシステムに向けた高速化が実装されている [4]。Oakforest-PACS システムでも用いられている Lustre ファイルシステム、IME についても最適化されたドライバが提供されている。

ただし、ROMIO ドライバにより IME が利用可能なのは、MVAPICH2 だけである。この場合は、MPI_File_open を用いてファイルをオープンする際に、“ime://” を先頭につけ、絶対パスを指定することによってアクセスすることで、IME ドライバが使用され、native に IME を使うことができる。

一方、Intel MPI でも ROMIO が内部的に使われているが、IME を使うためには下位層では POSIX を使ってアクセスする必要がある。OFP の計算ノードからは、IME 用のマウントポイント経由でアクセスできる。

IME は Lustre のキャッシュとして振る舞うが、CPU の

キャッシュとは異なり、自動的にデータのマイグレーションは行われないため、ユーザが明示的に行う必要がある (prestage)。また、キャッシュ上に作成したデータも、通常は自動的に Lustre に書き戻されることはないため、明示的に書き戻す操作を行う必要がある。OFP ではこれらを行うコマンドを用意している他、ジョブスケジューラと連動したステージング機能として実装しており、ジョブの割り当ての際に prestage、またはジョブ解放の際に sync を指示することで Lustre と IME との間で同期を取ることができる。

4. 並列メッシュ生成プロセスの高速化

4.1 オリジナルコード

まず、オリジナルの並列メッシュ生成プログラムを用いて測定を行った。測定には、Oakforest-PACS と、比較のために Intel Xeon Broadwell プロセッサを 2 ソケット搭載した Reedbush-U (東京大学情報基盤センター)[10] を用いた。両システムの構成は**表 2** の通りである。

プロセスあたり 128^3 メッシュとし、各システムでは同一のプログラムを用いて、いずれも 32 ノード、1024 プロセスのときの各レベルにおける処理時間を測定した。結果を**表 3** に示す。その結果、Level=3 までは OFP の方が Reedbush-U に比べて 3.7 倍程度性能が低い。これは単純なコアクロック比で考えるとはるかに性能が低く、理由は不明である。一方で、Level=5 以上になると、OFP の方が高い性能を示している。

その一方で、**表 4** のように、これらのメッシュデータファイルを使用するソルバーにおいて、計算時間は 9.14 秒であるのに対して、メッシュ読み込み等ファイル入力にかかる時間は 18.5 秒必要であり、ファイル入出力が明らかにボトルネックになっていることがわかる。

4.2 ファイル出力の最適化

図 7 にファイル入出力の要素だけを取り出した擬似コードを示す。この中で、10 行目の最後の要素 vvv だけが倍精度浮動小数点数 (real*8)、それ以外は全て整数 (integer*4) である。

まず、このコードの構造に合わせて MPI-IO をループ単位でまとめて入出力を行うようなコードを作成した。10 行目については、派生データ型を用いて実装を行った。これによって、ファイル入力については 1.5 秒程度まで高速化が実現できたが、ファイル出力については、まだ数倍遅い状態であった。

そこで、方針を変更し、KNL のラージページサイズに合わせて 2MB 単位でバッファリングして書き込んでいくことにした。その結果、**表 5** に示すように、1.42 秒でメッシュファイル生成が完了した。これにより、オリジナルに対して 255 倍の性能向上が得られた。

```
1  read (21)  NPX, NPY, NPZ
2  read (21)  NXcc, NYcc, NZcc
3  read (21)  nnn
4
5  read (21) (NEIBPE(k), k= 1, 6)
6  read (21) ICELTOT, intCELtot
7  read (21) levMGcelINDEX(lev-1)
8
9  do icel= 1, elmtot0
10   read (21) ii, i1, i2, i3, j1, j2, j3, vvv
11 enddo
12
13 read (21) nn
14
15 do icel= 1, elmtot0
16   read (21) ii, i1, i2, i3, i4, i5, i6
17 enddo
18
19 read (21) (IMPORT_index(kk), kk= iS0+1, iS0+6)
20 read (21) (ii, kk= 1, nq0)
21 read (21) (EXPORT_index(kk), kk= iS0+1, iS0+6)
22 read (21) (ii, kk= 1, nq0)
```

図 7 ファイル入出力の擬似コード

さらに、単一共有ファイル (Single Shared File:SSF) にまとめ、さらに IME を使った結果も表 5 に示す。ここでは、Intel コンパイラ 2018.3 と MPI として MVAPICH2-2.3 に対して、IME の ROMIO ドライバのパッチを当てたものを用いた。この場合、Level 1 での処理時間が長いのに加えて、Level 3 以降の処理が本来 1/8 になるところが減少していないことがわかる。SSF はプロセスごとにファイルを作った場合 (File Per Process:FPP) に比べて 8.2 倍時間がかかってはいるが、FPP では数千のファイルを取り扱うのに対し、各レベルごとに単一ファイルとしてまとまり扱いやすいため、利点も大きい。

5. おわりに

本研究では、Intel Xeon Phi による大規模クラスタ Oakforest-PACS でマルチグリッド前処理付き並列 CG ソルバーを実行する際のファイルを介した並列メッシュ生成プロセスの性能改善について検討した。その結果、ファイル IO を MPI-IO を用いて書き換えるとともに、アプリケーション内部でバッファリングを行うことで、オリジナルに比べて最大 200 倍以上の性能向上が得られた。

今後は、weak scaling, strong scaling について検証を行っていく予定である。

謝辞 本研究の一部は、学際大規模情報基盤共同利用・共同研究拠点、および、革新的ハイパフォーマンス・コンピューティング・インフラ (JHPCN-HPCI) の支援によるものです。(課題番号: jh180022-NAHI、課題名: Innovative Multigrid Methods)

表 2　Reedbush-U と Oakforest-PACS の仕様

	Reedbush-U	Oakforest-PACS
CPU	Xeon E5-2695v4 (Broadwell-EP)×2 ソケット	Xeon Phi 7250
周波数・コア数	2.1 GHz, 18 × 2	1.4 GHz, 68
メモリ	256 GB	16 GB+96 GB
メモリバンド幅	153.6 GB/s	490.0 GB/s(実測), 115.2 GB/s
インタコネクト	InfiniBand EDR 100 Gbps	OmniPath 100 Gbps
並列ファイルシステム	Lustre ファイルシステム	
容量	5 PB	26 PB
理論ピークバンド幅	150 GB/s	500 GB/s
コンパイラ, MPI	Intel Parallel Studio 2018.1	

表 3　Reedbush-U と Oakforest-PACS のメッシュ生成時間 (オリジナル, 32 ノード、1024 プロセス)

Level	1	2	3	4	5	6	7	8	全体
Reedbush-U	82.5	10.9	1.66	0.380	0.215	0.157	0.182	0.192	127.8
Oakforest-PACS	302.3	40.4	5.73	0.904	0.172	5.34E-02	3.10E-02	2.55E-02	362.7

表 4　ソルバーの内訳（coloring に要する時間は含めず）

	メッシュファイル入力	計算	通信
処理時間 (s)	18.51	9.14	1.56

参考文献

[1] HPCG: High Performance Conjugate Gradients, http://hpcg-benchmark.org/
[2] K. Nakajima, Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, Proceedings of the 20th IEEE International Conference for Parallel and Distributed Systems (ICPADS 2014) 25-32, 2014 (Best Paper Award)
[3] MPI: A Message-Passing Interface Standard Version 3.1, https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
[4] R. Thaker, W. Gropp, and E. Lusk, On Implementing MPI-IO Portably and with High Performance, Proceedings of the 6th Workshop on Input/Output in Parallel and Distributed Systems, pp. 23—32, 1999.
[5] 最先端共同 HPC 基盤施設 (JCAHPC), Oakforest-PACS スーパーコンピュータシステムについて、http://jcahpc.jp/ofp/ofp_intro.html
[6] Lustre filesystem, http://lustre.org/
[7] Infinite Memory Engine (IME), https://www.ddn.com/products/ime-flash-native-data-cache/
[8] IO-500, http://www.io500.org/
[9] 建部 修見、Oakforest-PACS における IO-500 の評価、情報処理学会研究報告 2017-HPC-162, p. 1–5, 2017 年
[10] 塙　敏博、他、データ解析・シミュレーション融合スーパーコンピュータシステム Reedbush-U の性能評価、情報処理学会研究報告 2016-HPC-156, p. 1–10, 2016 年

表 5 最適化後 (MPIIO 使用、128 ノード 8 スレッド、1024 プロセス)

Level	1	2	3	4	5	6	7	8	全体
最適化後 (FPP)	1.05	0.164	5.03E-002	2.22E-002	1.93E-002	1.67E-002	1.56E-002	1.45E-002	1.42
SSF+IME	7.21	0.771	0.511	0.514	0.497	0.487	0.460	0.505	11.6