よりよい超大規模並列CFDコード開発にむけて

Toward Better Development of Extreme-Scale CFD codes

○ 小野 謙二，理研 計算科学研究機構, 神戸市中央区港島南町 7-1-26, E-mail：keno@riken.jp
Kenji Ono, RIKEN AICS, 7-1-26 Minatojima-minami-cho, Chuo-ku, Kobe, Japan

Many challenges are required to develop large-scale parallel simulation codes, which include considering productivity and maintenance as well as to improve the performance and accuracy of the codes. In this paper, the author would like to share useful information for developing large-scale CFD codes.

1. はじめに

「京」コンピュータを始めとする超並列計算機の出現により，高並列計算環境が現実的なものになってきている．これらの巨大な計算機資源は，科学的な発見や工学的な知見を得るための道具として利用されるが，そのためには，計算科学の研究者が超並列計算機の性能を引き出すプログラムを開発することが重要なタスクとなる．一般的に，効率的なプログラムを書くために必要な知識やツールの研究開発は計算機科学分野の範疇にある．これまでの研究教育のカリキュラムでは，一般的には，物理化学分野のシミュレータを開発する研究者はこのような知識を詳細に系統的に学習する機会はあまりなかった．大規模な計算機資源を使い先端的な研究成果を創出していくためには，高い性能をもつプログラム開発が必要であり，そのような高性能プログラムを効率よく開発する技術を習得する意義は大きい．本稿の目的は，オープンソースのプロダクトを中心として，より効率的な大規模並列プログラム開発の生産性を高めるための視点を共有することである．

2. 高生産性ソフトウェア開発における視点

いわゆるスパコンで動作させる科学技術計算のプログラム開発は一般的なコードとは異なり，実行性能をきわめて重視しているため，そのソフトウェア開発には特殊な課題がある．しかしながら，一般的なプログラム開発で行われているような生産性を高める工夫は同様に適用可能である．ここではコード開発において，考慮すべき項目をリストアップする．

2.1 言語とコンパイラ

科学技術計算コードの記述には，Fortran 言語が良く用いられる．Fortran は簡便な記法で数値計算アルゴリズムを効率よく記述できる特長がある．記述の上で C/C++ に比べて制限があるが，逆に，コンパイラにとっては解析しやすく，ハードウェアに最適化したコードを生成しやすい．一方，C/C++は柔軟な記述が可能であるが，高性能化のためにはコンパイラが最適しやすいソースコード記述を心がける必要がある．

別の観点からは，Fortran と C は手続き型言語であるが，C++はオブジェクト指向である．オブジェクト指向は，一般的に，大規模コード開発に適していると言われるが，従来の手続き型言語でも適切なモジュール化やインターフェイスを定義することにより，メンテナンスしやすい大規模コードは開発可能である．オブジェクト指向言語特有のポリモーフィズムや STL の使用は性能の上でボトルネックになることもあるので，利用法には注意を要する．また，同じ STL でも処理系によりその内部実装が異なるため，プラットホームが異なると性能が大幅に異なることもある．

近年は，処理系の言語間サポート（Interoperability）も良くなり，Fortran と C/C++の混合言語で記述することも多くなってきた．混合言語のメリットは，それぞれの言語系の得意なところを利用する点にある．その一方で，言語学習の手間がかかる．また，C と Fortran では配列変数の記述順が $A[k][j][i]$ と $A(i,j,k)$ のように逆になる点などにも注意を払う必要がある．混合言語としては，他にもコンパイル型の言語とスクリプト（インタープリタ）の混合利用がある．この場合，スクリプトをモジュールを接続する糊（glue）として利用し，実行時の多様性を持たせることに役立つ．

2.2 OpenMP と MPI

プロセス数が非常に多くなると，いわゆる FlatMPI のみでは性能のボトルネックが現れるため，OpenMP との Hybrid 並列モデルが推奨されている．MPI で領域分割型の実装を行うと，MPI ライブラリがライブラリ内部で隣接間の通信テーブルを作成・保持する．大規模並列実行ではこのテーブルが巨大になるため，ユーザメモリ空間を圧迫することにも注意する必要がある．MPI の通信実装はハードウェアに依存した部分もあり，高性能通信のためには RDMA などが利用できれば積極的に利用したい．

MPI の代替として，例えば XMP 言語 [6] などがある．対応している言語やプラットホームは少ないが，簡便な記述で隣接間通信を記述できる特徴がある．また，一部 RDMA による実装で通信性能を高めている．

一方，高性能な通信アルゴリズムとして，通信隠蔽実装や非同期アルゴリズムなどの研究も進んでいる．

2.3 チューニング，性能プロファイル

ソフトウェアの性能向上を図るためにチューニング作業が行われるが，その際に様々なプロファイリングツールが利用される．ソフトウェアベースで性能を測定するツールやハードウェアカウンタにアクセスし性能情報を取得する PAPI インターフェイス [7] などがある．オープンソースのツールとしては TAU[8] などがある．しかしながら，動的な挙動により性能が変化するプログラムの場合には，アルゴリズムの設計時に全ての挙動を想定することは難しい．このような場合，実行時の性能を記録し，分析するツールが必要となる．プロダクションランなどの実行時における，このような問題を解決するために，実行時性能を取得するライブラリ PMlib を開発している．PMlib[11] は並列実行時のタイミングや演算数をサンプリングし，簡易・詳細レポートを出力する．プロセスグループが異なる場合や PAPI にも対応している．

2.4 デバッガ

最も簡単なデバッグ法としては，プログラム中に想定動作の応答を出力するデバッグライトがあるが，詳細な情報を得る標準的なツールとして gdb や TotalView などのデバッガが利用できる．一般的にアルゴリズム開発においては，データを可視化しながらデバッグすることが有用な場合が多い．この点では可視化ツールと組み合わせたデバッガなどが期待される [26]．

2.5 バージョンコントロール

バージョン管理システムは，プログラムを構成する各種ファイルの更新履歴を追跡する．これにより，任意の時点の構成ファイルへアクセスでき，過去に遡った修正やブランチによるバグ修正や複雑な管理が可能になる．また，このバージョン管理システムは複数の開発者によって開発を進める場合にその威力を発揮する．CVS, Subversion, Mercurial, Git などのシステムがある．最近では github[5] の利用により，開発過程の共有やデプロイ（展開）までもオープンソース化されている．

2.6 統合開発環境，ビルドツール

ソースコードを記述するエディタには，簡単な機能のものから多機能のものまで様々なツールが存在する．vi や Emacs を始めとして，統合開発環境（IDE）として Eclips や Xcode(Apple) など生産性を高める上で効果がある．

ライブラリ生成やアプリケーションのコンパイルには make システムが利用される．様々なプラットホームでの自動コンパイルを実現するためのツールとして，各プラットホームに応じた makefile を自動生成する GNU autotools や Cmake がある．これらのツールは，ビルド過程を自動化し，クロスコンパイルもサポートしている．また，Cmake は Windows へも対応している．

2.7 ドキュメンテーション

ソースコードの技術文書は，様々な形で提供される．例えば，独立した文書として利用マニュアルや設計書がある．一方，プログラムがどのように動作するか，あるいは実装の意図などを説明する場合には，ソースコード中にコメントとして埋め込まれている場合が多い．ソースコード中に埋め込まれた指定形式のコメントを自動抽出して成形し，文書を自動生成するツールがドキュメンテーションツールである．例えば，Doxygen[(3)], Javadoc などは，pdf や html に文書を成形し，出力する．また，クラス間の関係を示したクラス階層図なども出力可能である．

2.8 継続的インテグレーション

Continuous Integration(CI) は，高品質なプロダクトを効率よく開発するためには是非とも検討すべきツールである．予め記述されたテストを自動で実行し，以前の（正しい）結果との比較を行い，開発プログラムの正しさを担保することが主な目的である．Github リポジトリなどとの連携も可能で，リポジトリにコミットすると自動でテストを実行，レポート作成，メール通知などができる．Jenkins[(9)], Circle CI[(10)] など多くのツールが存在する．開発の過程では，プログラムの改善や機能追加，リファクタリング時に思わぬ副作用や間違いが生じる場合がある．このようなとりにくいバグを潰すことに効果がある．

2.9 プロジェクト管理・コミュニケーションツール

数万行にもなる大規模コードの開発には様々な要素が含まれ，また長期間にわたるメンテナンスを行っている場合には複数の開発者が関わることになる．開発者間の開発状況の情報共有，あるいは開発者とユーザ間のコミュニケーションを図るために，プロジェクト管理システムが利用される．代表的なものに Redmine があり，タスク管理、進捗管理、情報共有などの機能を提供する [(2)].

2.10 数値計算ライブラリ

汎用の高性能な数値計算ライブラリとして，BLAS, LAPACK, ScaLAPACK, FFTW などが利用される．これらは科学技術計算でよく利用される微分方程式，固有値計算，連立方程式に加え，統計処理の様々な関数群を提供する．また，汎用的な線形ソルバや反復ソルバとして PETSc, Aztec, Lis, Hypre など [(12)] 多くのライブラリが提供されている．特に大規模で高速な固有値計算には EigenExa[(4)] が「京」コンピュータなどで動作する．

2.11 領域分割

並列計算時の領域分割時，あるいはタスクのロードバランスをとる場合に分割アルゴリズムが必要となる．グラフ分割によるアルゴリズムをライブラリ化した Metis[(13)], Zoltan[(14)], Scotch[(15)] などが提供されている．ユーザが自分で分割アルゴリズムを記述する場合には，空間充填曲線（Space Filling Curve, SFC）を分割することによりロードバランスをとることが多い．

2.12 Mesh/Geometry

大規模計算のための格子作成は，重要な課題の一つである．複雑形状の大規模格子生成方法として，計算時の自動格子生成を提案しているが，このアプローチでは解析ソルバー側で形状情報を保持管理することが必要となる．このため，領域分割型の並列計算に対応した形状管理ライブラリ Polylib[(16)] を構築・提供している．Polylib はポリゴンベースの形状データ管理ライブラリで，領域分割に基づく並列処理に対応している．現在，細分割曲面にも対応できるように長田パッチベース [(17, 18)] の形状表現にカーネルを換装中である．

2.13 可視化・ファイル入出力

大規模並列計算において，ポスト処理は解決すべき研究対象の一つである．大規模かつ多数のデータファイルを対象とするため，まずファイル管理機能が必要となる．多数のファイルは，メタデータにより管理され，アプリケーション間のデータ授受はライブラリを利用することで自動化できる．ファイル入出力のライブラリとしては，階層的なデータ構造を記述し，そのファイル入出力を行う HDF5[(19)]，気象分野で用いられる netCDF[(20)]，様々なファイル I/O 機能を提供する Adios[(21)] などがある．大規模並列データの可視化システムとしては，米国 DOE 開発の Paraview[(22)], VisIt[(23)] などがあるが，国内では，HIVE[(24)], V-Isio[(25)] が提供されている．

2.14 ミドルウェア・フレームワーク

並列アプリケーションを記述するミドルウェアとしては，多くのミドルウェアやフレームワークが提供されている [(27, 28, 29, 30, 31, 32)]．これらの領域分割型のアプリを記述するミドルウェアは，データ構造の定義，通信処理，領域分割法，その他ユーティリティなどの機能を提供する．提供する機能により，ライブラリ形式にするのがよいか，あるいはフレームワーク形式にするのがよいか，実装は異なってくる．また，利用に際しては，要求されるミドルウェアの理解の程度も異なってくる．

2.15 Verification & Validation

開発したシミュレータの機能や性能・信頼性などを担保するためには，そのドキュメントとともに，ユーザがアクセスし検証できるデータセットの提供も必要である．さらに，検証問題や実験データなども併せてオープンにできるとよい．このような動きには，OpenFOAM の検証データを集めたサイト [(33)] などがある．著者らは，計算工学ナビ [(34)] にて国プロで開発したシミュレータの検証データを関連情報とともに提供している．

3. まとめ

大規模並列プログラム開発における生産性向上や信頼性の担保，展開，長期にわたるメンテナンスなど，ソフトウェア工学の点から検討すべき項目について列挙した．紹介したもの以外にもコード開発に役立つ様々なツールは数多く提供されている．使い方を学習しなければならない手間はあるものの，経験上，それ以上の恩恵に浴することが多い．

謝辞　本研究は，文部科学省科学技術試験研究委託事業「近未来型ものづくりを先導する革新的設計・製造プロセスの開発」の助成を受けている．

参考文献

(1) Lorin Hochstein and Victor R. Basili: The ASC-Alliance Projects: A Case Study of Large-Scale Parallel Scientific Code Development *Computer*, Vol.**41** No.**3** (2008) 50–58.

(2) `http://redmine.jp`

(3) `http://www.stack.nl/~dimitri/doxygen/`

(4) `http://www.aics.riken.jp/labs/lpnctrt/index.html`

(5) `http://github.com`

(6) http://omni-compiler.org/xcalablemp.html

(7) http://icl.cs.utk.edu/papi/

(8) https://www.cs.uoregon.edu/research/tau/home.php

(9) https://jenkins-ci.org

(10) https://circleci.com

(11) http://avr-aics-riken.github.io/PMlib/

(12) http://acts.nersc.gov/index.html

(13) http://glaros.dtc.umn.edu/gkhome/views/metis

(14) http://www.cs.sandia.gov/zoltan/

(15) http://www.labri.fr/perso/pelegrin/scotch/

(16) http://avr-aics-riken.github.io/Polylib/

(17) Takashi Nagata, “Simple local interpolation of surfaces using normal vectors”, Computer Aided Geometric Design, ELSEVIER, Vol.22, No.4, May 2005, pp.327-347.

(18) 清水 保弘，長田パッチの 3 次への拡張，UNISYS TECHNOLOGY REVIEW 第 114 号, pp.141–162, DEC. 2012.

(19) https://www.hdfgroup.org

(20) http://www.unidata.ucar.edu/software/netcdf/

(21) https://www.olcf.ornl.gov/center-projects/adios/

(22) http://www.paraview.org

(23) https://wci.llnl.gov/simulation/computer-codes/visit/

(24) http://avr-aics-riken.github.io/HIVE/

(25) http://avr-aics-riken.github.io/V-Isio/

(26) 松田勝之，武宮博，”データ可視化機能を持つ並列プログラムデバッグツール： vdebug”，JAERI-Data/Code 2000-014, (2000).

(27) Baden, S.B., Colella, P., Shalit, D., Van Straalen, B., ”Abstract KeLP”, 10th SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, March (2001).

(28) Wijesinghe, H.S., Hornung, R.D., Garcia, A.L., and Hadjiconstantinou, N.G., ”Three-dimensional Hybrid Continuum-Atomistic Simulations for Multiscale Hydrodynamics,” Journal of Fluids Engineering, Vol 126, pp. 768-777 (2004).

(29) Hornung, R.D., and Kohn, S.R., ”Managing Application Complexity in the SAMRAI Object-Oriented Framework,” in Concurrency and Computation: Practice and Experience (Special Issue), 14, pp. 347-368 (2002).

(30) Henshaw, W.D., “Overture: An Object-Oriented Framework for Overlapping Grid Applications,” AIAA conference on Applied Aerodynamics (2002), also UCRL-JC-147889.

(31) 小野 謙二，玉木 剛，”SPHERE - 物理シミュレーションのフレームワークと実行環境の開発”，日本計算工学会論文集，No. 20060031 (2006).

(32) 太田高志，白山晋，”オブジェクト指向フレームワークによる流体計算統合環境”，日本計算工学会論文集，No. 19990001 (1999).

(33) http://www.ercoftac.org/fileadmin/user_upload/bigfiles/sig15/database/9.4/index.html

(34) http://www.cenav.org