# 流体解析ソフトウェアAdvance/FrontFlow/redのGPU対応

大西　陽一*

# GPU support for CFD Software, Advance/FrontFlow/red

Ohnishi Yoich*

Advance/FrontFlow/red（以下 AFFr）の GPU を用いた高速化例を紹介する。

Keywords: 高速化、スーパーコンピューター、GPU、流体解析、CUDA、OpenACC、cuBLAS

## 1. はじめに

GPU とは Graphics Processing Unit の略で、主に CG の描画とそれに関係した計算を行うハードウェアである。画像処理を効率よく行うために並列処理に特化している。10 年以上前から、この並列処理能力を画像処理以外に用いる試みが盛んに行われている。いわゆる GPGPU(General Purpose computing on GPU　以下単純に GPU 化と呼ぶ)である。本稿では、AFFr の GPU 化例を紹介する。

## 2. 基本的な方針

一般にプログラムの高速化を行う場合、まず既存プログラムの計算時間を測定しボトルネックを特定することからスタートする。GPU では簡単な処理を CPU の何倍も高速に処理できるが、並列処理に特化した装置であり、また GPU と CPU 間のデータ転送に時間がかかる。そのためプログラムのどの部分を GPU 化するかよく見極めて行う必要がある。見極める基準としては、ある一部分の計算高速化率だけでなく GPU 化するための工数や、データ転送時間と高速化して得られる短縮時間の比較などがある。GPU 化するための工数については、CPU 版の既存プログラムから大きな変更なく GPU 化できることが工数短縮の観点から望ましい。そこで既存の CPU 版から指示行を入れるだけで対応できる OpenACC で GPU 化することを基本とし、必要に応じて CUDA ライブラリ等を用いることとした。また利用した計算機と GPU は以下の通りである。

表 1 CPU

| | |
|---|---|
| Processor name | Intel Xeon Silver 4310 |
| Packages(sockets) | 2 |
| Cores | 24 |
| Processors(CPUs) | 48 |
| Cores per package | 12 |
| Threads per core | 2 |

表 2 GPU

| | |
|---|---|
| Device Name | NVIDIA A30 |
| Device Revision Number | 8 |
| Global Memory Size | 25269698560 |
| Number of Multiprocessors | 56 |
| Concurrent Copy and Execution | Yes |
| Total Constant Memory | 65536 |
| Total Shared Memory per Block | 49152 |
| Registers per Block | 65536 |
| Warp Size | 32 |
| Maximum Threads per Block | 1024 |
| Maximum Block Dimensions | 1024, 1024, 64 |
| Maximum Grid Dimensions | 2147483647 x 65535 x 65535 |
| Maximum Memory Pitch | 2147483647B |
| Texture Alignment | 512B |

*アドバンスソフト株式会社　第 3 事業部
3rd Computational Science and Engineering Group, AdvanceSoft Corporation

| | |
|---|---|
| Clock Rate | 1440 MHz |
| Execution Timeout | No |
| Integrated Device | No |
| Can Map Host Memory | Yes |
| Compute Mode | default |
| Concurrent Kernels | Yes |
| ECC Enabled | Yes |
| Memory Clock Rate | 1215 MHz |
| Memory Bus Width | 3072 bits |
| L2 Cache Size | 25165824 bytes |
| Max Threads Per SMP | 2048 |
| Async Engines | 3 |
| Unified Addressing | Yes |
| Managed Memory | Yes |
| Concurrent Managed Memory | Yes |
| Preemption Supported | Yes |
| Cooperative Launch | Yes |
| Default Target | cc80 |

## 3. GPU 化対象の特定

AFF r は乱流非定常現象を精度よくとらえることに主眼を置いている。プログラムとしては、時間刻み幅を設定し、時間ステップごとに流れの支配方程式を解いていくことになる(図 1)。各時間ステップで最も計算負荷の高い部分は圧力ポアソンソルバーである。

$$\nabla^2 p' = b \qquad (1)$$

ここで P'は圧力補正値であり、方程式の右辺はまとめて b と置いている。

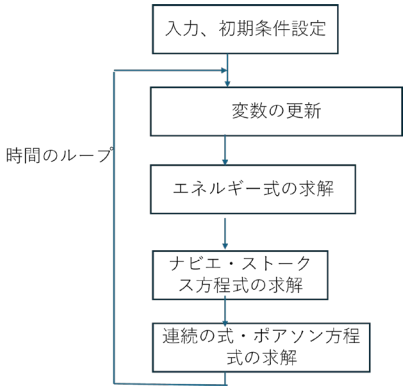


図 1　AFFr のフローチャート

このポアソンソルバーの解法として、反復法である前処理付き共役勾配法を採用している。非圧縮流れ場、500 万要素、48MPI 並列でポアソン方程式を ICCG 法で 100step 解いた場合の計算時間を計測したものが表 3 である。

表 3 非圧縮 CPU 計算時間における ICCG 比重

| | Time[s] | % |
|---|---|---|
| ICCG | 872.0 | 88.7 |
| それ以外 | 110.6 | 11.3 |

別途計算した 50 万要素程度の物理量を CPU から GPU へデータ転送速度した場合、0.1[s]以下程度の時間がかかったことから、もし GPU 化できたとしてデータ転送の時間を補って余りある高速化が期待できるとして、ICCG 法の GPU 化を決定した。

別の GPU 化対象部分の特定として、燃焼計算の高速化を目的とした検討を行った。燃焼という物理現象では、化学反応により生成消滅する大量の種類の化学成分を考慮する必要がある。反応で生成された化学成分が流れに乗ってどのように移流拡散するかを解く。その結果化学種の数だけ解くべき移流方程式が増加する。また反応による化学種の変化速度が通常の流れ場の典型的な時間スケールよりも何桁も小さいため、安定化・精度を追及すると時間刻みを小さくせざるを得ず、結果計算時間が膨大なものとなる。このような時間刻み幅を小さくして扱うことが求められる問題を時間積分硬直性という。AFFr ではこの化学反応を伴う燃焼計算特有の時間積分硬直性の問題に対して対象となる化学種の反応部分の方程式のみを分離して事前に時間積分し、流体の時間スケール当たりの変化率に平均化された反応速度を化学種の移流方程式に戻すことで全方程式の時間刻み幅を小さいものにそろえて解くことを避けている。反応式 200 式、化学種 100 程度の問題設定の時、CPU による燃焼計算を測定すると表 4 のようになる。反応の時間積分の占める割合が大きいが、これは炭化水素を燃料とした化学反応

を考慮するときさらに化学種が多くなる可能性がある。

表 4 CPU での化学反応計算時の反応時間積分の時間比率

| | Time[s] | % |
|---|---|---|
| 反応時間積分 | 15.2 | 74.5 |
| それ以外 | 5.2 | 25.5 |

反応の時間積分は以下のように離散化された方程式をさらに線形化して解いている。

$$\rho_i \frac{Y_{i,a}^{*} - Y_{i,a}^{n-1}}{\Delta t} = \omega_a(T_i, y_{i,0}, y_{i,1} \cdots, y_{i,ncomp}) \quad (2)$$

ここで化学種の質量分率を Y とし、反応式を $\omega$ としている。添え字 i はメッシュの index であり、添え字 a は化学種の種類を識別する index である。式を見るとわかるように、メッシュ index は i しか現れておらず、メッシュごとに独立して化学種の数 100×100 程度の行列サイズの問題を解くことになる。また非圧縮計算では支配的であったポアソン式の求解であるが、燃焼計算では温度圧力による密度変化が無視できないため連続の式を圧力補正値に関してまとめた式は以下のようになる。

$$\left\{\frac{1}{C_s^2 \Delta t} - \Delta t \nabla^2\right\} p' = b \quad (3)$$

AFFr では圧縮性流体計算の場合ポアソン式は不完全 LU 分解を前処理とした bicg 法を用いる。燃焼の場合は連続の式に現れる時間微分の項が圧力補正を未知数とした行列の優対角性を保証し、線型方程式の線形ソルバー反復回数は非圧縮と比較して一桁ほど少なくなる。したがって燃焼計算では GPU 化の優先度は低くなる。式(2)の GPU 化をすべきかどうか検討するときに、GPU へのデータ転送量を見積もると、メッシュ数×（化学種数+温度＋密度）となり、単一の物理量の線型方程式のデータ転送量よりも多くなる傾向がある。ニーズの高い解析に関する高速化ということでデータ転送時間を短縮する工夫を検討しつつ GPU 化することと決定した。

本稿では、上記の検討で決定した 2 種類の問題の GPU 化を報告する。2 種類の GPU 化とは以下のように要約できる。

表 5　GPU 化する対象の分類

| | ICCG 法の GPU 化 | 燃焼の GPU 化 |
|---|---|---|
| 行列サイズ | 数百万から数億を一回解く | 100×100 程度をメッシュ数分だけ解く |
| 行列成分 | 疎行列 | 密行列 |
| データ転送量 | メッシュ数程度 | メッシュ数×100 程度 |

## 4. ICCG の GPU 対応

AFFr は非構造格子であり四面体から 6 面体以上の任意多面体も扱える。このため、ポアソン式の非対角項の数は一定ではない。これまでの経験では高々20 程度であり、非対角項の数は 4 から 20 程度が不規則に現れる。データに有効にアクセスするためこの非対角項の現れ方に応じてデータを格納しておくことは有効である。ここには Ellpack-Itpack(ELL)形式を用いた。さらに ICCG 法は不完全コレスキー分解を前処理とした共益勾配法であり、GPU 化するときに注意して検討が必要な個所はこの前処理部分である。不完全コレスキー分解は前進後退代入を含み、GPU 内でもスレッドブロック内ではメモリ共有するため、代入操作により依存性が発生しないようにグループ分けしておくことが有効である。これらを達成するために、マルチカラー法（MC 法）による分類をさらに発展させた RCM-CM 法 (Reverse Cuthill-Mckee Cyclic multi-coloring)と呼ばれるリオーダー法を採用した。[1]

### 4.1. 計算速度

48CPU の MPI 並列計算と、A30 GPU 基 1 枚を利用したときの、ICCG 反復計算部分の計算結果比較を下記に示す。

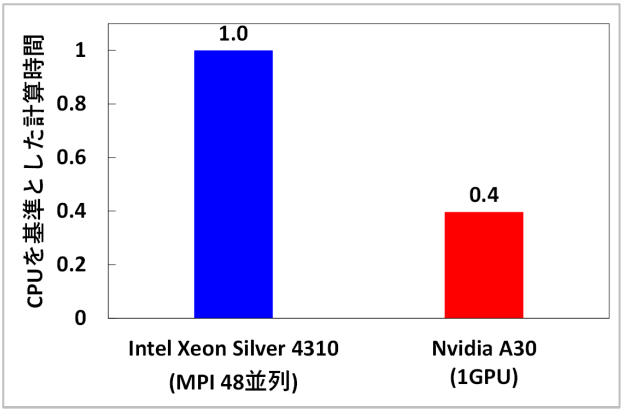


図 2　ICCG 法の計算時間比較

GPU 計算では、先に述べたデータ競合が起きないようグループ分けしたことにより GPU 内での高いスレッド並列性が確保され結果 48CPU よりも 1GPU で 2.5 倍高速化された。

### 4.2. 今後の取り組み

今後は CPU の並列計算との連成、複数 GPU 対応に取り組む。実はこれと同じ手法を用いたミニアプリでの試計算を NVIDIA A100 ８枚用いて実施したところ高い並列性能が得られることを確認している。A100 は A30 の 2 倍程度の性能をもちさらに 8GPU まで線形に高速化できれば 48CPU 計算の 40 倍程度の高速化が期待できる。これはこれまで 1 週間程度かかっていた非定常乱流計算がわずか数時間で終了する速度である。今後はこの成果を汎用流体ソフト特有の非構造格子でも達成すべく GPU 化に取り組む。

## 5. 燃焼計算の GPU 対応

ここではまず 100×100 程度の密行列の求解の GPU 化を実施する。CPU 版では、Lapack の密行列用の解法サブルーチン dgesv を用いていたため、CUDA 版の Lapck ライブラリを利用することとした。ただし、2024 年時点では、CPU で準備されていた Lapack サブルーチン dgesv そのものの CUDA 版は準備されていない。Dgesv の中の処理は、LU 分解 dgetrf と線形ソルバーdgetrs の組み合わせであり、これらの CUDA 版は準備されていたので組み合わせて利用した。この時のコンパイルコマンドは以下の通りである。

```
nvfortran -O2 -Minfo test_dgesv.f90 -acc –cuda
–cudalib=cublas,cusolver
```

言語はすべて Fortran である。

```
      subroutine
      cublas_dgesv(h,n,nrhs,a,lda,ipiv,b,ldb,dwork,lwo
      rk,info)
           use cublas_v2
           use cuSolverDn
            implicit none
           integer ,intent(in)::lda,ldb,n,nrhs,lwork
           integer,device,intent(out)::info
          integer,device,intent(out):: ipiv(*)
          real*8,device,intent(inout)::a(lda,*)
          real*8,device,intent(inout)::b(lda,*)
          real*8,device,intent(inout)::dwork(lwork)
          type(cusolverDnHandle),intent(in)::h
          integer :: istat,istat_f,istat_s
istat_s= cuSolverDndgetrf(h,n,n,a,lda,dwork,ipiv,info)
istat_f=cuSolverDndgetrs(h,CUBLAS_OP_N,n,nrhs,a
,lda,ipiv,b,ldb,info)
    RETURN
  end subroutine cublas_dgesv
```

図 3　cublas、cuSolver を用いた密行列

### 5.1. 計算速度 1

メッシュ数を 1000 と固定し、化学種数に相当する行列サイズの依存性を確認した。1 メッシュ当たりで平均化した計算時間を CPU と GPU で比較すると、行列サイズが 10 では CPU の方が早いが、サイズが 100 を超えると GPU は CPU の 3 倍程度高速化されている。

表 6　密行列の解法速度比較

| 行列サイズ | CPU の計算時間[s] | GPU の計算時間[s] |
|---|---|---|
| 10 | 1.91E-06 | 8.00E-05 |
| 100 | 1.60E-04 | 2.70E-05 |
| 1000 | 1.02.E-02 | 3.10E-03 |

また GPU へのデータ転送時間は表 7 のようになる。

表 7　データ転送時間のサイズ依存性

| 行列サイズ | データ転送時間[s] |
|---|---|
| 10 | 1.60E+00 |
| 100 | 1.67E+00 |
| 1000 | 1.80E+00 |

まだどこかに不適切な処理をしている可能性はあるが、行列ソルバーに対してかなりデータ転送に要する時間の比率が大きい。これまでの結果を総合すると炭化水素燃料のように化学種 100 程度を考慮し、かつメッシュ数が 10 万程度であれば、GPU 計算の方が早くなる。

### 5.2. Stream 計算と非同期データ転送

データ転送の時間をさらに短縮し高速化する方法として、CUDA の並行計算機能を検討した。Concurrent 計算とも呼ばれる並行計算は、CUDA3.0 から可能になった。利用している NVIDIA A30GPU では、CUDA 非同期機能を用いて、データのコピーとデバイス処理を並行して処理可能となる。

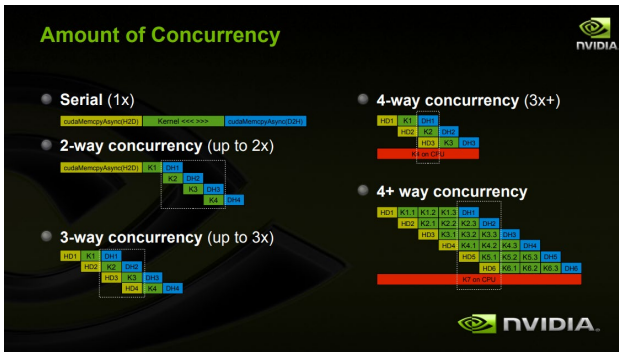


図 4　NVIDIA 資料より

A30 GPU は 3 エンジンであるので、図 4 の 3-way concurrency が可能となる。反応の時間積分はメッシュごとに独立であるので、全メッシュを stream 数で分割する。Host と Device 間のデータ転送、および Kernel での計算のオーバーラップを発生させることで全体の時間短縮を検討した。

### 5.3. 計算速度 2

現段階では、メッシュ分割による stream 並行計算まで実装が終わっているためこの部分の効果を報告する。

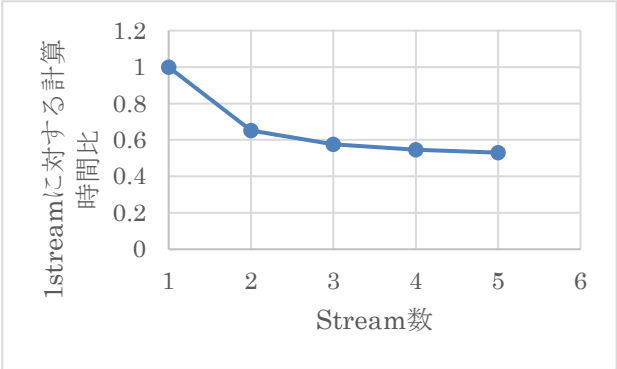


図 5 計算時間の stream 数依存性

Stream 数を増やすことで全体計算時間が短縮できることを確認した。

## 6. まとめ

AFFr による GPU 化の取り組みを紹介した。問題の特徴としては、行列サイズが大きいが呼び出す回数が少ない問題と行列サイズは小さいが呼び出し回数が多い問題である。どちらも流体解析のニーズの高い解析でボトルネックとなっていた箇所であり、本取り組みの過程で速度アップに効果があることを確認できた。今後は実用的なレベルで利用可能となるよう整備していく。

## 謝辞

本取り組みにおいて、東京大学中島研吾先生、Nvidia 成瀬彰様にはさまざまにご助言いただいた。また筑波大学 山菅昇太郎君にはプログラム作成に尽力いただいた。この場を借りて感謝申し上げます。

## 参考文献

[1] 河合直聡ほか、日本計算工学講演会論文集 2017-5.

[2] https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

※ 技術情報誌アドバンスシミュレーションは、アドバンスソフト株式会社 ホームページのシミュレーション図書館から、PDF ファイル（カラー版）がダウンロードできます。（ダウンロードしていただくには、アドバンス/シミュレーションフォーラム会員登録が必要です。）