大規模流体アプリケーションのGPUによる高速化手法の評価

星野 哲也[†] 丸山 直也[†,††] 松岡 聡[†,††]

1. はじめに

格子法を用いた流体アプリケーションにおいて，支配的な計算カーネルであるステンシル計算に対して，GPU を利用する研究[1] は盛んに行われており，その有効性が示されている．また，GPU による計算環境も一般的になりつつあり，東京工業大学に設置されたスーパーコンピュータ TSUBAME2.0 は，演算性能の大部分が GPU によるものである．GPU は性能に対する価格・消費電力が CPU と比べ低いこと等から，今後も GPU を利用した計算環境は増加することが予想される．

しかし，CPU 向けに開発された従来のアプリケーションは，何らかの方法で GPU 向けに移植する必要があるが，この移植コストは小さくなく，実際に用いられる大規模なアプリケーションの移植には大きなコストを伴う．さらに，計算カーネル部分の GPU 利用など，個々の小さな問題については多くの研究がなされているが，実際に使われる大規模な流体アプリケーションにおける研究は十分にされていない．

本稿では，実際に利用されている大規模流体アプリケーションである UPACS[2] を例として，手動で段階的に CUDA 化した．この手動実装について，TSUBAME2.0 の単一ノードを用いての性能評価，書き換え行数をもとにしたコストの評価を行った結果を示し，大規模流体アプリケーションの GPU 適用における，性能上の課題を示す．

2. 研究目的

既存の CPU 向けアプリケーションの GPU 環境への移植を考えたとき，性能向上の度合いと移植コストを踏まえて移植方法を決定するべきである．GPU 環境への移植方法として，手動による CUDA 化，ディレクティブベースの OpenACC，PGI アクセラレータコンパイラを利用する方法等が考えられるが，得られる性能や移植コストは明らかではない．これらの性能・コスト・課題について明らかにし，定量的評価の指標を作ることが本研究の目的である．

† 東京工業大学 Tokyo Institute of Technology
†† JST/CREST

3. CUDA 化による性能・コストの評価手法

手動により CUDA 化した場合の性能と実装コストの比較評価をするために，実際の大規模流体アプリケーションを例に実装し評価する．実装・評価の方針として，まずはベースとなる単純な実装を施し，その後既知の最適化を施すことで，どの程度の性能向上が得られ，実装コストがかかるのかを評価する．本研究では例として，独立行政法人宇宙航空研究開発機構 JAXA により研究・開発されている，UPACS[2] を手動により CUDA 化する．UPACS は開発言語が Fortran，数万行に及ぶ大規模流体アプリケーションである．Fortran のプログラムを CUDA 化する場合，CUDA Fortran を使う方法も考えられるが，一度 C 言語に書き換えた上で CUDA に書き換える．まずは，UPACS の既存 CPU 実装に対してプロファイラを用いて解析を行う．TSUBAME2.0 の 1CPU ソケットの 1 コアを用いて解析した結果，**表 1** に示すソルバが主なボトルネックとなっていることを確認した．

表 1 ボトルネックとなるソルバ

ソルバ名	CPU 時間 [sec]	割合 [%]
blk_rhsviscous_	20.79	36.4
blk_mfgs_	15.20	26.5
blk_rhsconvect_	13.35	23.3

特定されたボトルネックのソルバ内の大きなループ構造のみを CUDA 化したものを V1 と定義する．V1 をベースラインとして既知の最適化を段階的に適用したものを V2∼V7 と定義し (**表 2**)，これらについて性能・コストの比較評価を行う．

表 2 段階的 CUDA 実装

V1	ボトルネックのソルバ内 3 重ループのみ CUDA 化
V2	V1 + ボトルネックのソルバ内の全 CUDA 化
V3	V2 + 定数データの転送削除
V4	V3 + ボトルネックのソルバ以外も CUDA 化
V5	V4 + 対流項・粘性項ソルバでデータ構造変更
V6	V5 + MFGS ソルバの並列性の増加
V7	V6 + その他 (アドレス計算削減，共有メモリ利用等)

© 2012 Information Processing Society of Japan

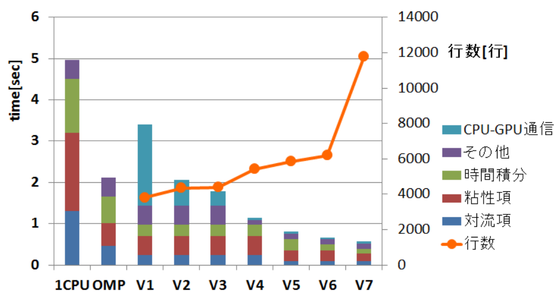

図 1 1 タイムステップ辺りの実行時間 (左軸)
書き換え行数 (右軸)

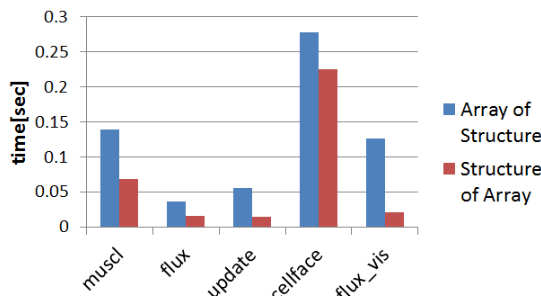

図 2 データ構造の変更

4. 実装・評価

前述の実装を用いて，120×120×120 の格子データに対して，TSUBAME2.0 の 1GPU で実験を行い，1 タイムステップにかかる実行時間を計測した．なお，UPACS は様々な解法を選択することが出来るが，今回はひとつに固定しており，CUDA 化したのもその入力で使うごく一部である．比較検証のため，各ソルバに対して OpenMP による並列化を行い，比較を行った．**図 1** に示した OpenMP は 6 並列時のものであり，オリジナルの 1CPU コアでの実行と比べて 2.5 倍程度高速化していることがわかる．

OpenMP と V1 を比較すると，各ソルバには性能向上が見られるのものの，全体としてはむしろ遅くなっている．原因は各ソルバの前後で行っている CPU-GPU 通信であり，極力通信を避けるようにするべきである．これを受けて V2～V4 では CPU-GPU 間の通信を削減している．書き換えのコストは比較的に小さく，V1 から 3 倍程度性能向上している．

V5 では，対流項・粘性項で局所的に使われているデータ構造を Array of Structure から Structure of Array に変更している．**図 2** は，対流項・粘性項の内部で呼ばれる各サブルーチンの，データ構造変更に伴う実行時間の変化を示している．ルーチンごとに差はあるが，最大で 6 倍程度の性能向上が得られていることがわかる．V6 では，時間積分ルーチンが陰解法を用いているため，対流項・粘性項ほどの並列性がない．そこでアルゴリズムを変更することで，並列性を増やしている．この変更により，時間積分部分では 2 倍程度の高速化が得られた．V7 の変更では，1.2 倍程度性能向上したが，コード量が倍増してしまっている．

V1～V4 で行ったのは，主にデータ転送の最適化であり，比較的簡単な変更で 3 倍程度性能向上しているが，V5～V7 ではアルゴリズムの変更を伴い，難しい変更であるといえる．また，V7 から更に高速化するためには，プログラム全体で共有されたデータ構造を適切な形に変更する必要がある．この変更を図 2 の update ルーチンに対して適用したところ，1.8 倍程度の性能向上を確認した．同様にデータ構造を変更することで，他のルーチンにおいても性能向上が期待できるが，この変更はプログラム全体に影響を及ぼすため，大きな書き換えコストを伴う．

5. まとめと今後の課題

大規模流体アプリケーションの GPU による高速化手法を評価するために，一例として UPACS の手動による CUDA 化を行い，既知の最適化を段階的に適用した．その結果，1CPU コアの 8 倍，6CPU コアの 3 倍程度の性能向上が得られることを確認し，各最適化に伴うコストを評価した．また，データ構造の変更が性能向上をもたらすことを確認したが，大規模アプリケーションでは書き換えコストが大きくなることを確認した．一般的に大規模なアプリケーションでは，プログラムは機能ごとに分割して管理されているが，データ構造は共有されているために，同様の問題が生じると考えられる．

今後の課題として，現在の実装の性能を調べるための性能モデルの構築，プログラム全体で共有されたデータ構造の変更とその方法の考察，複数のサブルーチンを融合することによるデータの読み書き回数の削減，今回の手動実装をベースとした他の手法との比較評価があげられる．

謝辞 本研究にあたり，UPACS を提供してくださった，独立行政法人宇宙航空研究開発機構准教授の高木亮治先生をはじめとする皆様に，感謝の意を表する．

参 考 文 献

1) Nguyen, A., Satish, N., Chhugani, J., Kim, C. and Dubey, P.: 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs, SC '10, Washington, DC, USA, IEEE Computer Society, pp. 1–13 (2010).
2) Takaki, R., Yamamoto, K., Yamane, T., Enomoto, S. and Mukai, J.: The Development of the UPACS CFD Environment, Vol. 2858, Springer Berlin / Heidelberg, pp. 307–319 (2003).

© 2012 Information Processing Society of Japan