Vol.2016-HPC-157 No.7
2016/12/21

コンシューマレンジのGPUに最適化した固有値ソルバーの実装と評価

今村 俊幸[1,a)]　椋木 大地[1,b)]

概要：コンシューマレンジの GPU である GeForce や組み込み系 GPU の Tegra では HPC 向けの機能を削いでおり, デバイスメモリの転送能力は 200GB/s を超えるハイエンドモデル並みだが倍精度と単精度の演算性能バランスが 1:32 と著しく悪いものが多い．このような GPU を用いた数値計算ではハイエンド GPU とは異なるアルゴリズムや実装の選択が必要となる．本報告では GPU 向け固有値ソルバーである Eigen-G と MAGMA の固有値計算ルーチンと対して, その性能評価を DP 性能と B/F 値のバランスの観点から行う．B/F 値が相対的に高い GPU 環境下では古典的なアルゴリズムである 1-stage アルゴリズムが高速であり, 全固有値・固有ベクトルを計算する必要がある場合には B/F 値が低いハイエンド GPU や現代的な CPU 環境における選択とは異なることが数値実験からも明らかになった．さらに, 単精度演算器を用いて倍精度演算を模擬する double-float 技術 (DF) を使用して実装した DGEMM 関数を用いてコンシューマレンジ GPU 向けの最適化を施す．DF 版 DGEMM を用いた固有値ソルバ Eigen-G の実行性能は GeForce GTX1080 上で N=10000 の固有値問題を解いたときに 20 秒であり (DP 版では 21.4 秒), 7% 程度の高速化が認められる．測定誤差は相対残差が $|A - X^\top \Gamma X|_1/(N|A|) = 6.0 \times 10^{-18}$, 直交誤差が $|X^\top X - I|_1/N = 1.3 \times 10^{-15}$ であり, 倍精度での演算と比較して 10 進数で 1 ないしは 2 桁程度の劣化で済んでいる．実用上は DF 版 DGEMM を使用した Eigen-G を用いることで十分な速度性能と演算精度を保証できることが分かった.

1. はじめに

GPU 上での基盤技術開発の裏には, GPU プロセッサコアが持つ極めて多彩なハードウェア構成が存在している．特に, GPU ベンダの市場戦略の広がりにより, 多数のコアを搭載する GPU 本来のプロセッサであるものと, 組み込み系などの少数コアではあるが省電力に傾けたものなど, 構成は多数である．科学技術計算の特に複雑な数学計算に GPU を用いるときには, 数値線形代数 (行列やベクトルでの数式表現) の高性能計算は避けて通れない．我々は過去の GPU 向けのカーネルソフトウェア開発の過程で, 高性能 BLAS の最適スレッド数の数理をはじめとして double-double や double-float などの基盤技術の蓄積を続けてきた ([1], [2], [3], [4], [5], [6], [7], [8], [9] など)．更に, これら基盤技術を GPU 向け固有値ソルバーに活用して, 数値計算シミュレーションコードの高速化に寄与することを可能にしてきた [10].

1 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science, Kobe, Hyogo
a) imamura.toshiyuki@riken.jp
b) daichi.mukunoki@riken.jp

近年市場に投入される GPU はハイエンドクラスとコンシューマレンジとの二極化・差別化が顕著になされる傾向にある．科学技術計算や HPC 分野で使用するハイエンド機はスパコン並みもしくはそれを超える演算性能とメモリ性能を保持することがあるが, コンシューマ向けの GPU ではある程度のメモリ性能は保持するがあまり必要とされない倍精度浮動小数点演算器などは大幅に省かれている．しかしながら, 比較的安価で導入しやすいコンシューマレンジの GPU を HPC 分野で活用しない手はなく．その特性を見定めつつ, 適切な数値計算アルゴリズムや実装方式を適用することで十分に科学技術計算の武器になりえるはずである.

本報告の目的は, コンシューマレンジの GPU に最適化した固有値ソルバーの周辺技術とその性能評価結果の速報をまとめることである.

2. コンシューマレンジ GPU

本報告では, NVIDIA 社の GPU で HPC 向け Tesla ブランドや可視化サーバ向け Quadro ブランドではない GeForce ブランドの GPU をコンシューマレンジ GPU と呼ぶ．GeForce ブランドには高性能ゲーム向けの x80 番

台から, 廉価版の x70 番, さらにコンソール向けの x50 や x40 番台があるが, 本報告では, 科学技術計算に必要な倍精度演算機 (DP) を搭載する x50 番以上を想定する.

2.1 性能面での特徴・ハイエンド GPU との比較

現在の市場に投入されている GPU はコア世代, CPU コア数, CPU クロック, メモリ帯域, メモリ容量によって様々なバリエーションが存在する. ベンダーの市場戦略としてもこれらの数値を変え, 先に示した x80 番台や x50 番台などのエントリを用意し価格帯などを考慮した差別化が行われている. GPU を科学技術計算で演算アクセラレータとして使用する場合, 倍精度演算 (DP) 性能, 単精度演算 (SP) 性能, メモリバンド幅が大きな要因として考えられる. これらに加えて大規模演算を実践する場合には, メモリ容量と CPU–GPU 間データ転送性能などが必要項目としてあげられる.

特に近年のコンシューマレンジとハイエンドの違いは DP/SP にある (表 1 を参照). メモリ容量なども差別化されているが, 計算科学計算で必要とされる広帯域メモリ (CPU と比較して広帯域) は搭載されているので, 価格に見合った DP 性能が備わっていればある程度の科学技術計算に適用することは可能である.

2.2 B/F≈1 を実現する環境例

コンシューマレンジの GPU は高性能可視化デバイスとして, メモリ帯域の広さと単精度演算の高さを保証している. 一方, 科学技術計算で必要な倍精度計算はその演算能力を落としているため, 倍精度計算時には演算とメモリの性能バランスが均衡する. 実際, 表 1 を見ると, コンシューマレンジの GPU では B/F 値がほぼ 1 か 1 以上のものが多い. これは, 10 年前にスパコンアーキテクチャを席巻したベクトルプロセッサ環境と近いものになり, 近年のマイクロプロセッサでは B/F 値が 0.2 以下程度のものがほとんどであることを考慮すると非常に高い値であることがわかる.

近年, ポストムーア時代到来後の技術動向に関する議論の中で, Flops からデータ移動に中心を移すことによるアルゴリズムの変化についての研究が重要であるとされている. その議論の中では古典アルゴリズムも含めた新技術開発が必須であるとの認識がなされており, B/F 値の幅がここまであるコンシューマレンジ GPU 環境はポストムーア時代に向けたアルゴリズム検証の一プラットフォームになりうるものである.

3. GPU 固有値ソルバー

3.1 実装系

GPU 向けの固有値ソルバー 3 種類について説明する.

- CULA
- MAGMA
- Eigen-G

CULA[11] は LAPACK の CUDA 実装の先駆け的ソフトウェアであるとともに商用ソフトウェアでもある. しかしながら, MAGMA や Eigen-G とは異なり非常に古典的なアルゴリズム (QR 法, 二分法+逆反復法に基づく計算ルーチン) のサポートにとどまっているため, 確実で安定な動作は保証できるが計算速度の面で劣ってしまう.

MAGMA[12] には 1-stage アルゴリズムの `magma_dsyevd` と 2-stage アルゴリズムの `magma_dsyevdx_2stage` というルーチンとそのマルチ GPU 版が固有値ソルバとして実装されている. 固有値以外の連立一次方程式ソルバーや疎行列計算用のルーチンなども品揃えが充実してきており, 単一ノードでの GPU 数値計算ライブラリとしては業界標準といってもいいであろう.

Eigen-G[10] は今村らが開発する EigenK, EigenExa[13] のスタンドアローン版を GPU 化したものであり, 次に説明する 1-stage アルゴリズムを採用し. 各種の最適化技術により高速化を施している. 過去の HPC 研究会 [5] でもその性能解析や MAGMA ライブラリとの比較をしてきている.

3.1.1 1-stage アルゴリズム

1-stage アルゴリズムは実対称行列の固有値計算の古典的アルゴリズムの一つとして, Householder 三重対角化を経て, 行列を三重対角化し, Cuppen 分割統治法, 逆変換により固有値・固有ベクトルを求める方法である. 多くの数値計算の教科書で紹介されており, 当然ながら殆どの数値計算ライブラリに採用されている.

3.1.2 2-stage アルゴリズム

2-stage アルゴリズムは 1-stage アルゴリズムに対してマイナーな存在であるが, 近年の B/F 値が低い計算機アーキテクチャで高い性能をだすことを目指して開発されたアルゴリズムである. Level3 BLAS の DGEMM(行列行列積) を中心にアルゴリズムを構成することで, コア部分でピーク性能にかなり近い性能を出せるようにしている.

具体的には,

(1) Block Householder 変換により密行列を一旦ブロック三重対角もしくは帯行列に変換する.
(2) 次に, 村田の方法や Rutishauser の方法により三重対角に変換し, 固有値固有ベクトルを計算する.
(3) 逆変換は順変換の逆であるので, 三重 → 帯 → 密行列の 2 回の操作が必要となる.

MAGMA ライブラリでの実装については開発者らの論文 [14] に詳しく述べられているが, 帯 → 三重もしくは三重 → 帯の変換部分はデータ依存関係が存在するため, 効率的な実装はかなり難しいようである. しかしながら, 逆変換部分は固有ベクトルの本数に比例したコストになるため, 2-stage アルゴリズムは**部分的な固有値 (例えば最大から数%, 最小から数% など) 計算の時に威力を発揮する方法と**いわれている.

表 1　主要な GPU の各種性能比較

GPU	コア世代	演算比 DP/SP	DGEMM GFlops	sgemm GFlops	メモリ帯域 GB/s	Byte/Flop
GTX1080	pascal	1/32	280	7700	250	0.89
GTX980	maxwell	1/32	154	4500	170	1.10
GTX750Ti	maxwell	1/32	46	1340	67	1.45
Tesla K20c	kepller	1/2	1050	2500	135	0.128

3.2　コンシューマレンジ GPU 環境で期待される動作

コンシューマレンジ GPU では B/F 値が 1 に近いため, 10 数年前のコンピュータで通じた「**演算量が計算時間に比例**」するということが成り立つ.

2-stage アルゴリズムは 1-stage のメモリバンド幅律速の部分を, **演算量を増加させても DGEMM の加速によって全体の計算時間が短縮する**ようにしたものである. したがって, 相対的に DGEMM の性能が悪くなる状況では, 2-stage アルゴリズムは不利になる.

1-stage ソルバー, 2-stage ソルバーの計算時間は主要時間項のみのかなりラフな見積もりによって N が十分に大きいとき

$$T_1 = \gamma_f(2/3 + 4/3 + 2)N^3 + \gamma_m 8 * 2/3N^3 \quad (1)$$

$$= (12\gamma_f + 16\gamma_m)N^3/3 \quad (2)$$

$$T_2 = \gamma_f(2/3 + 2/3 + 4/3 + 2 + 2)N^3 \quad (3)$$

$$= (20\gamma_f)N^3/3 \quad (4)$$

とできる. ここで, γ_f=1/Flops, γ_m=1/(B/s), $\delta = \gamma_f/\gamma_m$=B/F 値とする. 両アルゴリズムの時間比は

$$\frac{T_1}{T_2} = \frac{12\delta + 16}{20\delta} \quad (5)$$

とできるが, $\delta = 2$ のときに T_1, T_2 は一致する. これは 1-stage ソルバーの対称行列ベクトル積 (DSYMV) の要求 B/F 値に相当し, 計算量の観点からの議論ではメモリバンド幅律速部分が解消されることを意味している. 2-stage アルゴリズムは帯 → 三重化またはその逆変換に由来するオーバヘッドが必ず生じるが, 現在これらを無視した理想的な動作の議論をしている. それ故に, 式 (5) の分母は大きくなるはずであり, B/F が 1 以下でも比較的 1 に近い値でも 1-stage アルゴリズムが有利になることが多くあることが推定される.

3.3　double-float 演算技術による加速

Dekker [15] はある仮数部長の浮動小数点演算でその 2 倍の仮数部長の計算を行う手法を示している．この手法に基づいて単精度演算のみを用いて倍精度相当の浮動小数点演算を行う手法を，本稿では double-float 演算（DF 演算）と呼ぶ．DF 演算では倍精度相当の積和演算（$a \times b + c$）を 16 回の単精度浮動小数点演算で実現できる．これにより倍精度と単精度の理論ピーク演算性能比が 1:32 であるコンシューマレンジの Pascal アーキテクチャ GPU では，積和演算で構成される行列積（GEMM）において，DF 演算を用いることでハードウェアによる倍精度演算と比べ，約 2 倍の性能が期待できる．

今回は Eigen-G で行われる計算のうち，GEMM のみに DF 演算を用いることとし，CUBLAS の DGEMM（cublasDgemm）が呼ばれている箇所を，我々が実装した DF 版の GEMM（mublasDBgemm）に置き換えた．DF 演算を用いた GEMM の実装については，研究報告?において Maxwell アーキテクチャ GPU における実装と性能評価を報告している．本研究では Pascal 向けに GEMM カーネルの設計を若干変更し性能最適化を行った．また mublasDBgemm は cublasDgemm と互換のインタフェースを実現するため，行列の入力は binary64 の倍精度型で行うが，内部で DF 型に変換して DF 演算を行い，最後に DF 型を binary64 型に戻して書き出す．このデータ型の変換について，以前の実装 [9] では，グローバルメモリ上の行列 A, B にアクセスするタイミングで binary64 型から DF 型への変換を行っていた．しかし binary64 型から DF 型への変換コストは，行列積においてブロッキングによるデータの再利用を行っても無視できないものであった．そこで今回は予め binary64 型で格納されている行列 A, B を DF 型へ変換（コピー）するカーネルを動かし，それから DF 型データに対して DF 演算で GEMM を計算するカーネルを実行する実装とすることで，性能を改善した．

4.　性能評価

本報告では 4 つの GPU を使用し性能評価を行っている. 4GPU の詳細については以下表 2 の通りである.

4.1　基本性能評価

本研究では倍精度浮動小数点性能 Flops, とメモリバンド幅が重要な指標となる．DGEMM(DP 版と DF 版), SGEMM の性能測定結果を通じて実際の倍精度演算と単精度演算の性能比, 更には DF 版での実装 mublasDBgemm の性能について示す．さらに, CUBLAS や MAGMABLAS, ASPEN.K2[1] の DSYMV を用いてその性能から実行のメモリバンド幅を算出する.

4.1.1　DGEMM(DP 版と DF(double-float) 版) の性能

まず，GeForce GTX 1080 において cublasDgemm と

表 **2**　本報告で使用する GPU の性能諸元

	GeForce GTX1080
GPU Name	GP104
Compute Capability	6.1
GPU Clock (MHz)	1607(boost 1733)
Multiprocessors	20
CUDA Cores	2560
Memory Capacity (GByte)	8 (GDDR5X)
Memory Clock (Gbps)	10 (256bit)
Memory Bandwidth (GB/s)	320
ECC Support	NA

	GeForce GTX980
GPU Name	GM204
Compute Capability	5.2
GPU Clock (MHz)	1126(boost 1216)
Multiprocessors	16
CUDA Cores	2048
Memory Capacity (GByte)	4 (GDDR5)
Memory Clock (MHz)	7012(256bit)
Memory Bandwidth (GB/s)	224
ECC Support	NA

	GeForce GTX750Ti
GPU Name	GM104
Compute Capability	5.0
GPU Clock (MHz)	1020(boost 1085)
Multiprocessors	5
CUDA Cores	640
Memory Capacity (MByte)	640(GDDR5)
Memory Clock (MHz)	2048(128bit)
Memory Bandwidth (GB/s)	86.4
ECC Support	NA

	Tesla K20c
GPU Name	GK110
Compute Capability	3.5
GPU Clock (MHz)	706(boost NA)
Multiprocessors	13
CUDA Cores	2496
Memory Capacity (GByte)	4(GDDR5)
Memory Clock (MHz)	5120(320bit)
Memory Bandwidth (GB/s)	208
ECC Support	Enabled

mublasDBgemm の性能を比較した結果を図 **1** に示す．ここでは正方行列に対して，Eigen-G で使用する NN カーネル（行列 A, B ともに転置なし）と TN カーネル（行列 A のみ転置）の性能を比較している．GeForce GTX 1080 の倍精度演算の理論ピーク演算性能は，仕様上のブーストクロック [*1] の 1733MHz から算出すると約 277.3 GFlops

[*1] GeForce GTX 1080 にはクロックの自動制御技術 GPU Boost が搭載されており，負荷に応じてクロックが自動で変動する．ブーストクロックは GPU Boost によって到達可能なクロックの平均値であるため，それ以上のクロックに到達する場合がある．なお使用した GPU ではこの機能をオフにすることはできない．

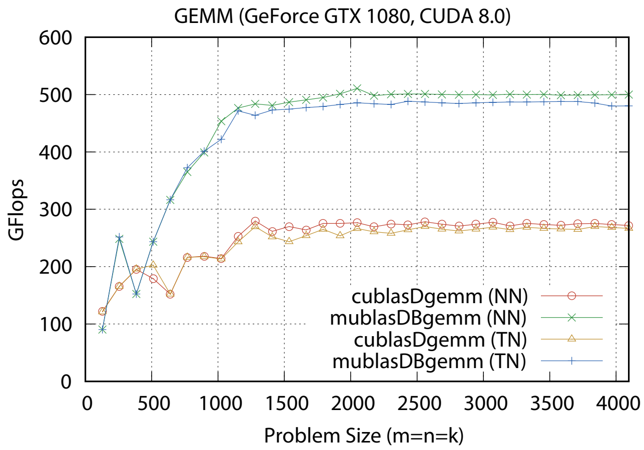

図 **1**　GeForce GTX 1080 における cublasDgemm と mublasDBgemm の比較（NN: 行列 A,B ともに転置なし，TN : 行列 A のみ転置）

である．NN カーネルにおいて，cublasDgemm は最大で約 280.0 GFlops の性能であるが，我々の mublasDBgemm は最大で約 510.8 GFlops を達成した．

図 2 に DGEMM, SGEMM, double-float で実装した DGEMM の性能を示す．いずれの GPU でも，DGEMM と SGEMM の性能比は表 1 中の DP/SP と同様であることがわかる．また，DF 版の DGEMM の性能は Tesla K20c を除いて，DP 版の DGEMM よりも 2 倍程度高速である．

4.1.2　メモリバンド幅 (DSYMV 性能)

図 3 に DSYMV の性能を示す (MAGMA が内部で称する MAGMABLAS の DSYMV, Eigen-G が使用する ASPEN.K2 の DSYMV を測定した)．DSYMV 実行時のメモリバンド幅は DSYMV の実行性能 (Flops) の 2 倍に相当するので，Flops と B/s の換算は容易である．GeForce GTX750Ti を除いて ASPEN.K2 が高速であり，CUBLAS や MAGMA は GPU ごとに異なる性能特性を示している．

ただし，CUBLAS の無印 DSYMV はアトミック関数を使用したスレッドブロック間の総和計算を行うため，実行毎に結果が丸め誤差の影響を受けて下位数ビットの違いが現れる．一方，ASPEN.K2 はアトミック演算を使用するが，適切な総和演算順序の制御により演算結果にビットレベルの一貫性を持たせることができる．また，MAGMA はワーク配列を使った gather-scatter 方式により結果一貫性を保っている．したがって，最高性能と結果一貫性を得るには ASPEN.K2 が最良の選択肢になる．

4.2　固有値ソルバーの性能

図 4 に

(1) 1-stage アルゴリズム (`magma_dsyevd`)
(2) 2-stage アルゴリズム (`magma_dsyevdx_2stage`)
(3) Eigen-G

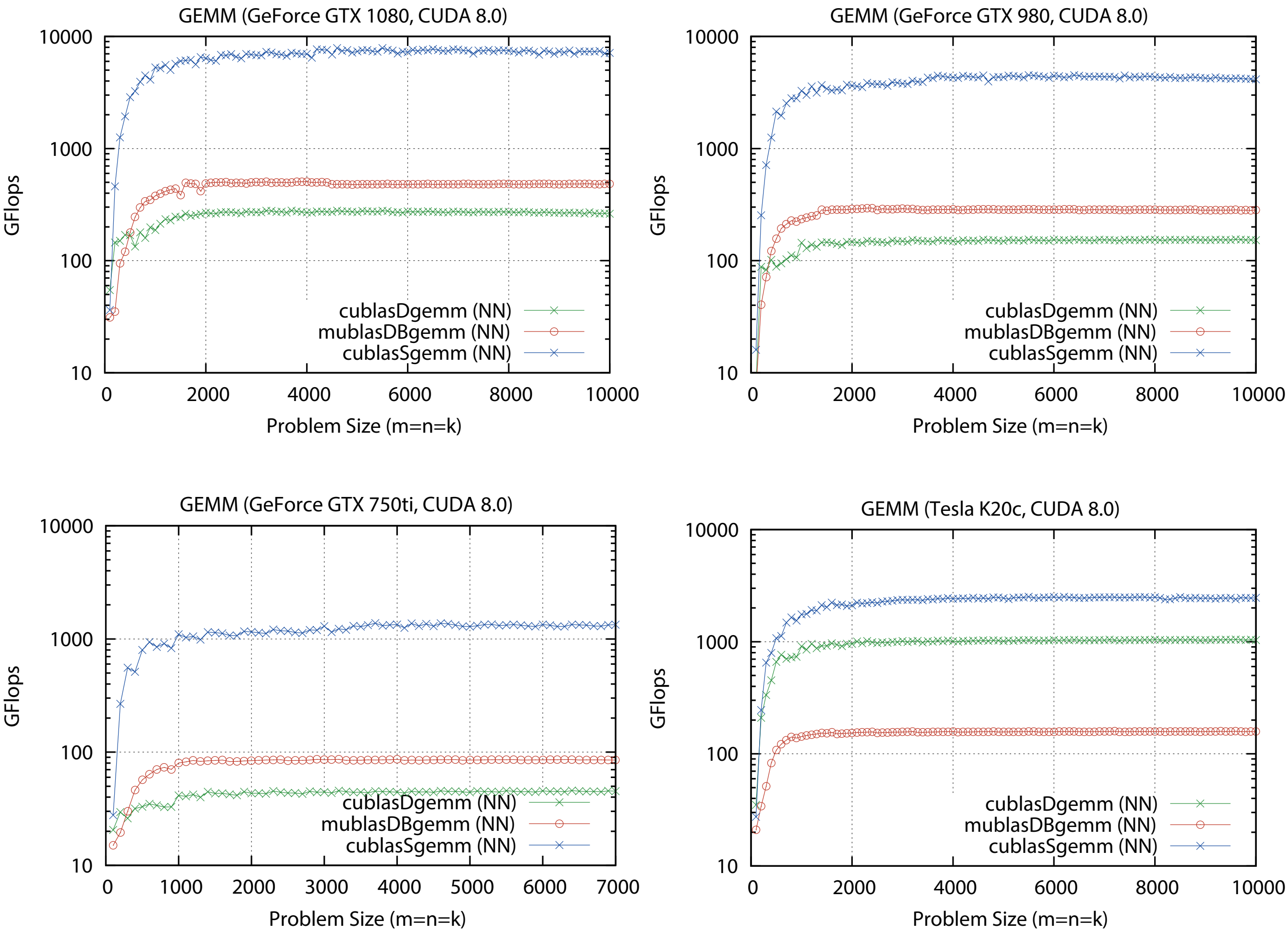

図 2 cublasDgemm と mublasDBgemm, cublasSgemm の比較（NN: 行列 A,B ともに転置なし，TN：行列 A のみ転置）,GeForce GTX1080(左上), GeForce GTX980(右上), GeForce GTX750Ti(左下), Tesla K20c(右下)

の乱数行列の全固有値・固有ベクトル計算に要する時間をプロットした．MAGMA の 2 関数は magma-2.1.0 に収納される testing コード群に含まれる当該 testing プログラムを使用し，コマンドラインオプションとして`-JV -N 100:10000:100` を指定してし実行している．*2

前節で説明したように，1-stage アルゴリズムと 2-stage アルゴリズムの性能特性は DGEMM 性能とメモリバンド幅によって決まることが予測されている．実際に図 4 から判断すると，DGEMM 性能が十分に高い Tesla 系の GPU で 2-stage アルゴリズムが 1-stage アルゴリズムに対して高速になる．逆の場合には旧来の 1-stage アルゴリズムが高速であり，B/F 値が低い環境に向けて開発された 2-stage アルゴリズムが有利にならないことがわかる．

また，Eigen-G は MAGMA の 1-stage アルゴリズムとほぼ同様のアルゴリズムを採用しており，実装面で若干の違いがある．対称行列ベクトル積 (DSYMV) に対して，現在最速となる ASPEN.K2-1.5p9 を使用し，逆変換部分に CPU+GPU のハイブリッド実行をしているため，MAGMA よりも性能面で 10〜15%程度高速である．特筆すべきは，DGEMM が高い性能を示す場合は，2-stage アルゴリズムの優位性が予測されるのだが，Eigen-G の 1-stage アルゴリズムが MAGMA の 2-stage アルゴリズムとほぼ同等という結果が得られている．2-stage アルゴリズムでは逆変換部分の三重 → 帯変換部分が計算量増加部分にあたり，実装もかなり困難であるため相当のボトルネックになるが，[14] にもあるように，`magma_dsyevdx_2stage` はカーネルルーチンの巧妙なスケジューリングを行う高度な最適化をしているため GPU 側での 2-stage アルゴリズムの性能上限にあるといえる．

しかしながら，2-stage アルゴリズムでは小規模な DGEMM が多数呼ばれるため，順変換の一部と逆変換部分で 30%程度の性能しかでないとして，分母を

*2 なお，Eigen-G は CPU と GPU を一定の比率で実行させることができるが，CPU と GPU の DP 性能比を考慮して各 GPU 毎に次の比を指定をしている．(GTX1080, GTX980, GTX750Ti, K20c) = (27:14, 11:5, 9:14, 50:7).

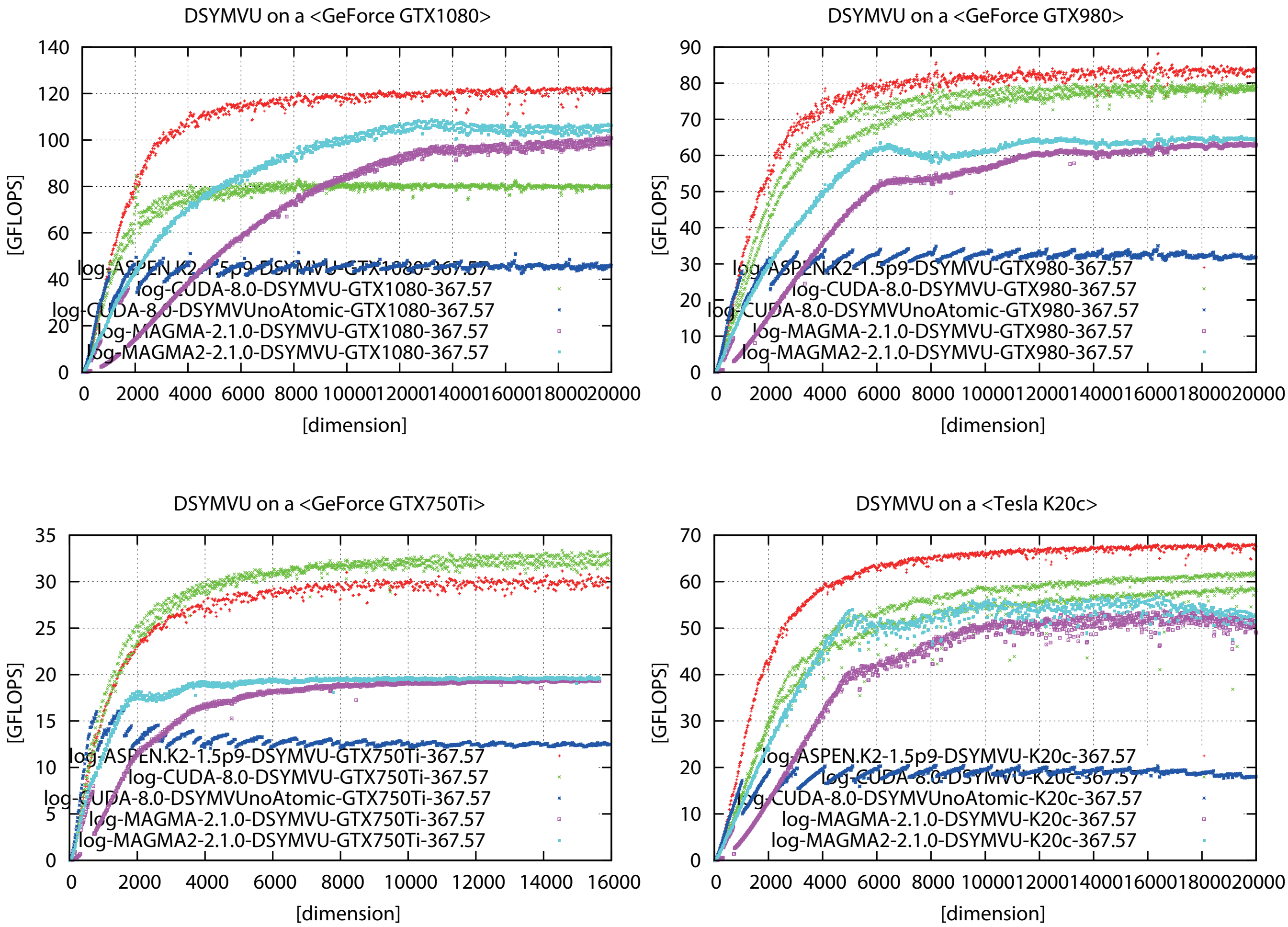

図 3 ASPEN.K2-1.5p9, CUBLAS (CUDA 8.0), MAGMA 2.1.0 での DSYMV の比較(U(上三角) フォーマット),GeForce GTX1080(左上), GeForce GTX980(右上), GeForce GTX750Ti(左下), Tesla K20c(右下)

2/0.3+2+4+3*2/0.3+3*2=38.7 とし $\delta = 0.70$ で両アルゴリズムの計算時間が一致するといった修正が必要になる．実際は, Eigen-G との時間比較で B/F=0.128 で両者が一致しているため, CPU 側での処理時間に非常に大きなオーバヘッドが含まれることが示唆される．Tesla K20c で MAGMA 2.1.0 を timer 出力を ON にしてコンパイルして, N=10000 を実行した際の出力を以下に示す．

magma_dsytrd の実行ログ

```
time dsytrd =   17.43
time dstedx =    3.20
time dormtr + copy =    2.90
```

magma_dsyevdx_2stage の実行ログ

```
  N=       10000  nb=    128 time dsytrd_sy2sb=    2.02
  N=       10000  nb=    128 time dsytrd_convert =    0.01
  Finish BULGE    timing= 4.273144
  Finish T's      timing= 0.093300
  time BULGE+T = 4.375466
  N=       10000  nb=    128 time dsytrd_sb2st=    4.44
  N=       10000  nb=    128 time dsytrd=    6.47
  N=       10000  nb=    128 time dstedx =    3.20
  N=       10000  nb=    128 time dbulge_back =    6.48
  N=       10000  nb=    128 time dormqr + copy =    2.68
  N=       10000  nb=    128 time eigenvectors backtransf.
=   12.36
```

上記ログを見る限り, 順変換の bulge chase に時間を要しており, 性能改善の余地が見られる．ソースコードを解析する限りでは buldge chase は CPU 上でスレッド並列に実行されている．演算量は $O(N*nb^2)$ であり, nb が小さい行列の HouseholderQR 分解を進めることから, N=10000 における DSYMV の処理に要する時間 3.26 秒 (Core i7-3940K, 3.2GHz, 6 コア使用時) は少なくとも必要である．この部分の処理をメモリ帯域の広い GPU に offload できれば, magma_dsytrdx_2stage の dsytrd_sb2st の 4.44 秒の部分

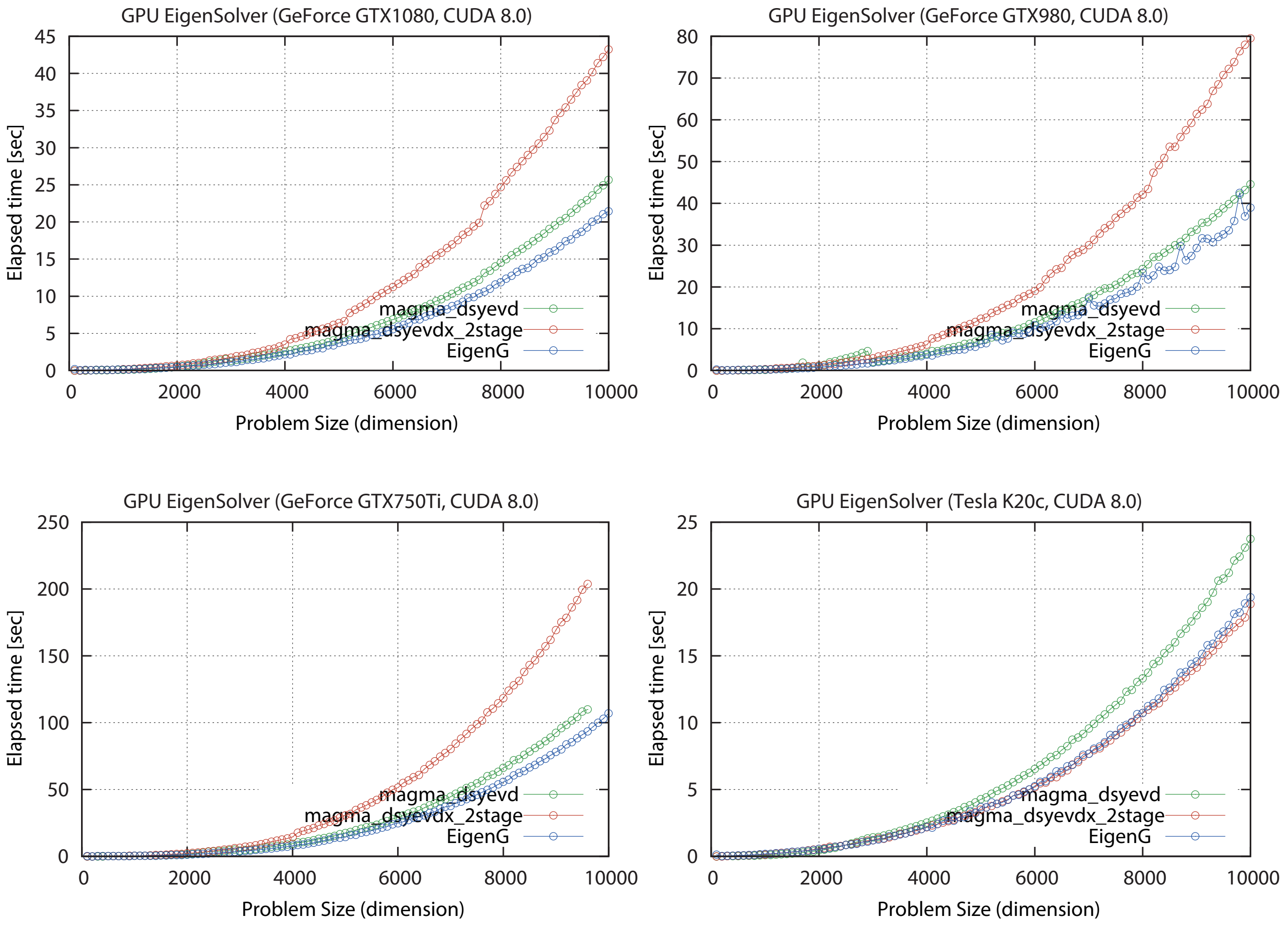

図 4 magma_dsyevd と magma_dsyevdx_2stage, Eigen-G の比較（100 次元から 10000 次元まで 100 次元毎, 乱数行列の全固有値・固有ベクトルの計算）, GeForce GTX1080(左上), GeForce GTX980(右上), GeForce GTX750Ti(左下), Tesla K20c(右下)

はまだ十分に性能改善が可能である．

4.3 double-float 版 DGEMM による性能改良

4.3.1 計算時間削減効果

図 5 に，DF 版 DGEMM(mublasDBgemm) を用いた Eigen-G による性能改良を示す．オリジナル Eigen-G と DF 版 Eigen-G による計算時間の削減が確認できる．分割統治法や逆変換部分は DGEMM が支配的な計算部分であるので計算時間の大きな削減を期待したのだが DSYMV や CPU での処理の方が全体に占める割合が高いため，DGEMM の DF 化によって削減できたのは 7～20% である．最も削減の効果があったのは B/F 値が最も高い GeForce GTX750Ti であるが，DGEMM の絶対性能が極めて低いことがそのまま反映されているといえる．

4.3.2 精度への影響

次に，倍精度演算を単精度数を 2 つ組み合わせて作る double-float に置き換えることで生じるより大きな丸め誤差の影響について確認する．以下はオリジナルの DP 版 Eigen-G の計算結果 (Tesla K20c, N=10000) の相対残差と直交誤差を出力したものである．

オリジナル DP 版 Eigen-G の精度検証ログ

```
machine.EPS        =   2.220446049250313E-016
|AZ-ZW|_F          =   3.365878808964467E-011
|A-ZWZ^t|/(N|A|)   =   6.934349219363451E-019
|Z^tZ-I|/(N)       =   7.789857617667546E-017
```

次に DF 版の Eigen-G で同じ問題を解いた際の精度を示す．

DF 版 Eigen-G の精度検証ログ

```
machine.EPS        =   2.220446049250313E-016
|AZ-ZW|_F          =   5.656747989080741E-010
|A-ZWZ^t|/(N|A|)   =   6.668970596105819E-018
|Z^tZ-I|/(N)       =   1.391176193510466E-015
```

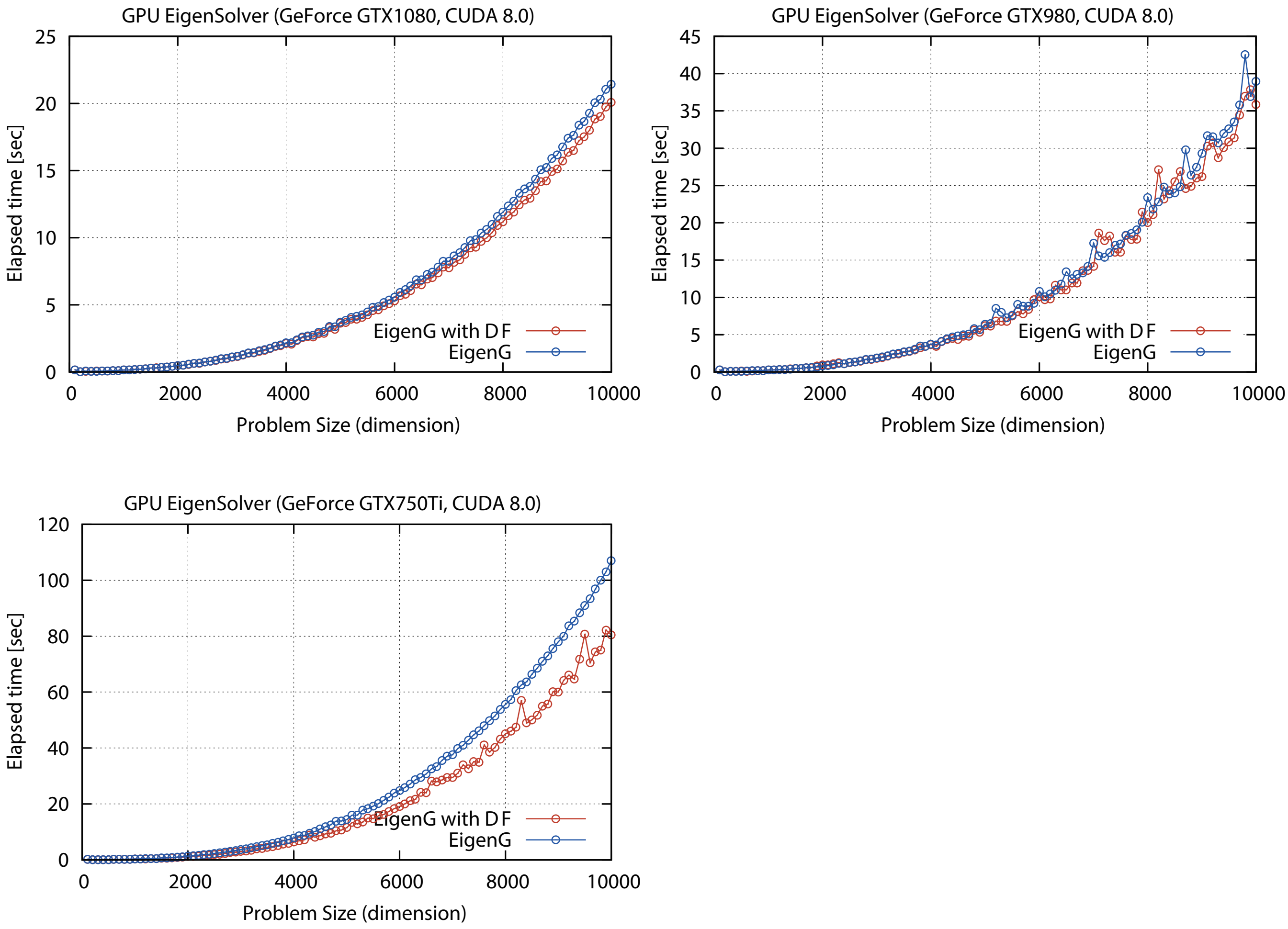

図 5　double-float 版 dgemmEigen-G (mublasDBgemm) を使用した Eigen-G の性能（100 次元から 10000 次元まで 100 次元毎，乱数行列の全固有値・固有ベクトルの計算），GeForce GTX1080(左上), GeForce GTX980(右上), GeForce GTX750Ti(左下), Tesla K20c(右下)

DP の仮数部分は 52 ビット，DF の仮数部は 2*23=46 ビットである．したがって，今回の固有値計算の範囲では，DP から DF にデータ形式が変わることで失われる 6 ビット (52bit-2*23bit=6bit) 分の 10 進 2 桁程度の精度劣化にとどまることが確認できる．なお，今回の固有値計算で DF 版 DGEMM に置き換わる部分は

(1) 分割統治法の 2 分岐のマージ処理の中で固有ベクトルを乗じる処理

(2) 逆変換における鏡像逆変換

となるため数値的にも非常に安定した計算に限定されていたといえる．これが，指数部のオーバーフローを起こすこともなく，極端な巨大数同士の演算も引き起こさなかったと考えられる．より一般の計算の中で DGEMM の代替として DF 版の DGEMM を使用する場合には注意が必要となるであろう．

5. まとめ

本研究では，コンシューマレンジ GPU が広帯域メモリを搭載するものの倍精度と単精度の演算性能バランスが著しく悪いという点に着目し，そのような計算環境における数値計算アルゴリズムや実装の選択方法について議論した．特に，固有値ソルバーの性能評価を DP 性能と B/F 値のバランスの観点から行うことで，B/F 値の高低が実際にアルゴリズム選択に影響を及ぼしており，近代的なアルゴリズムとして開発されてきた 2-stage アルゴリズムが必ずしも有効ではないなどの結果が導かれた．さらに，単精度演算器を用いて倍精度演算を模擬する double-float 技術の速度面での有効性や今回の固有値ソルバ実装においては許容できる誤差範囲に収まるなど新しい技術の展開の可能性を示すことができた．

現報告時点の実装はまだプロトタイプであり，一般に公

開できるレベルまでインターフェイスなどを改善する必要がある. また, 過去の HPC 研究会でマルチ GPU 化は実施済みであるが, 実際の固有値ソルバに対応するためにはデータ構造やマルチ GPU のハンドリング部分の改良など課題は多い. 同様に, Xeon Phi プロセッサなど GPU 以外の複数種類のデバイスに対応するなども今後の研究課題としたい.

本研究は科研費基盤 B(課題番号: 15H02709) ならびに計算科学振興財団 (研究助成研究教育拠点 (COE) 形成事業) の支援を受けている.

参考文献

[1] Imamura, T., ASPEN-K2: Automatic-tuning and Stabilization for the Performance of CUDA BLAS Level 2 Kernels, 15th SIAM Conference on Parallel Processing for Scientific Computing (PP2012)

[2] 椋木大地, 今村俊幸, 高橋大介, Kepler アーキテクチャ GPU における高速な SGEMV の実装, GPU Technology Conference Japan 2014, (2014)

[3] 今村俊幸, 内海貴弘, 林熙龍, 山田進, 町田昌彦, Fermi, Kepler 複数世代 GPU に対する SYMV カーネルの性能チューニング, 情報処理学会研究報告,「ハイパフォーマンスコンピューティング（HPC）」, Vol. 2012-HPC-138, No. 8 (2012) 1–7.

[4] 今村俊幸, 内海貴弘, 山田進, 町田昌彦, CUDA-xSYMV の実装と評価, 情報処理学会研究報告,「ハイパフォーマンスコンピューティング（HPC）」, Vol. 2014-HPC-146, No. 14 (2014) 1–12.

[5] 今村俊幸, 椋木大地, 山田進, 町田昌彦, CUDA-BLAS 等の選択による最速 GPU 固有値ソルバーの性能評価, 情報処理学会研究報告,「ハイパフォーマンスコンピューティング（HPC）」, Vol. 2015-HPC-148, No. 4 (2015) 1–9.

[6] 椋木大地, 今村俊幸, 高橋大介, NVIDIA GPU におけるメモリ律速な BLAS カーネルのスレッド数自動選択手法, 情報処理学会研究報告,「ハイパフォーマンスコンピューティング（HPC）」, Vol. 2015-HPC-150, No. 13 (2015) 1–13.

[7] Imamura, T., Yamada, S., and Machida, M., A High Performance SYMV Kernel on a Fermi-core GPU, High Performance Computing for Computational Science — VECPAR 2012, LNCS 7851 (2013) 59–7.

[8] Mukunoki, D., Imamura, T., and Takahashi, D., Fast Implementation of General Matrix-Vector Multiplication (GEMV) on Kepler GPUs, 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) (2015) 642–650, doi:10.1109/PDP.2015.66.

[9] 椋木, 大地, 今村, 俊幸, Maxwell アーキテクチャ GPU における疑似倍精度演算を用いた DGEMM の実装と評価, 情報処理学会研究報告,「ハイパフォーマンスコンピューティング（HPC）」, Vol. 2014-HPC-147, No. 26 (2014) 1–6.

[10] Imamura, T., Yamada, S., and Machida, M.: Eigen-G: GPU-Based Eigenvalue Solver for Real-Symmetric Dense Matrices. Proc. 10th International Conference, PPAM 2013, Part I. LNCS 8384 (2013) 673–682.

[11] Humphrey, J.R., Price, D. K., Spagnoli, D. K., Paolini, A. L., Kelmelis, E. J., CULA: Hybrid GPU Accelerated Linear Algebra Routines, SPIE Defense and Security Symposium (DSS), April, 2010.

[12] Innovative Computing Laboratory, University of Tennessee, Matrix Algebra on GPU and Multicore Architectures, `http://icl.cs.utk.edu/magma`

[13] EigenExa ホームページ: `http://www.aics.riken.jp/labs/lpnctrt/EigenExa.html`

[14] Haidar, A., Tomov, S., Dongarra, J., Solca, R., and Schulthess T., A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine-grained memory aware tasks, International Journal of High Performance Computing Applications, Vol. 28, No. 2 (2014) 196–209.

[15] T. Dekker, A Floating-Point Technique for Extending the Available Precision, Numerische Mathematik, Vol. 18 (1971) 224–242.