

CUDA 環境下での DGEMV 関数の性能安定化・ 自動チューニングに関する考察

今 村 俊 幸^{†,††}

GPGPU のコア技術となる線形計算代数において BLAS の性能向上と安定化が重要となる。DGEMM ルーチンの性能チューニングによって LINPACK ベンチマークの勝者が決まることから、その事実は理解できる。本研究は NVIDIA の CUDA 環境を想定して CUBLAS や MAGMABLAS など major な BLAS 実装における行列ベクトル積 (DGEMV-N と DGEMV-T) に対する自動もしくは半自動、手動チューニングを報告する。既存の性能報告にあるように、CUBLAS と MAGMABLAS はノコギリ歯状の性能特性を示す。著者らは測定結果とハードウェアの物理的な制約からモデル化の試みをする。それらモデルを組み合わせて性能チューニングを施した CUDA 環境上での MYBLAS について、GT200 系と Fermi コアでの結果と今後の課題について報告する。

Performance-stabilization and automatic performance tuning for DGEMV routines on a CUDA environment

TOSHIYUKI IMAMURA^{†,††}

Computational linear algebra is one of the core fields for GPGPU. To achieve higher and stable performance on BLAS can be thought as the most important issues, as the performance tuning for DGEMM routine leads to the LINPACK benchmark winner. This paper reports a case study for automatic, semi-automatic or manual performance tuning for the kernel function, DGEMV-N and DGEMV-T of the CUDA BLAS library. As many reports reveal that the performance on CUBLAS and MAGMABLAS shows a big saw figure. We build a performance model from the observations and the hardware specification. By using a simple modeling, we tune up the kernel function, namely AT-based MYBLAS on a CUDA environment. This study is examined on GT200 series and Fermi core series, and a future direction for automatic tuning on GPGPU is described.

1. はじめに

GPGPU のコア技術となる線形計算代数において BLAS の性能向上と安定化が重要となる。GPU クラスタ型のスパコンの登場などで、一躍脚光を浴びる DGEMM ルーチンの性能であるが、300GFLOPS や 500GFLOPS に達する性能を出すまでにチューニングがなされている¹⁾。また、GPU のマルチコア演算能力を利用して 4 倍精度の BLAS も開発が進んでいる²⁾。

本研究は NVIDIA の CUDA 環境を想定して CUBLAS や MAGMABLAS など主要な BLAS 実装における行列ベクトル積 (DGEMV) に着目する。既に

示したように DGEMM は各種ベンチマークのため十分なチューニングがなされているが、PC 用の BLAS 同様に Level2 BLAS に属する関数は GPU 向け BLAS もチューニングがなされているとは言い難い。Level 2 BLAS はメモリバンド幅がボトルネックとなり、単体の計算量が少量でも、アプリケーションの性能を左右する場合がある。著者が手掛ける固有値計算でも、対称行列の三重対格化や非対称行列のヘッセンベル格化に Level 2 BLAS の DGEMV 関数が利用されており、反復毎に呼び出されている。結果的に、全体コストの 70% を占めるとの報告があり (例えば文献 3) など)、メモリバンド幅に制限され理論ピーク性能には遠く及ばないが、それ故に、性能チューニングや性能安定化の結果はアプリケーション全体の性能に強く影響する。

過去の GPU の性能報告には、CUBLAS と MAGMABLAS がノコギリ歯状の性能特性を示すも

[†] 電気通信大学大学院情報理工学研究所
Graduate School of Informatics and Engineering, The
University of Electro-Communications
^{††} 戦略的創造研究推進事業/科学技術推進機構
CREST/JST

のが見受けられる。特に、DGEMV 関数は変動幅が 10% 以上に達する場合もある。前段でも述べたように、DGEMV 関数の 10% の変動幅はアプリケーションの性能に大きく影響を与えるため、性能チューニングの観点から無視することはできない。ノコギリ歯状の性能特性の発生原理の究明には様々なアプローチが考えられるが、著者は測定結果とハードウェアの物理的な制約からモデル化を試みる。それらモデルを組み合わせて、DGEMV 関数の GPU 上での自動もしくは半自動、手動チューニングにより性能安定化の方法論を議論する。最終的に、性能チューニングを施した MYBLAS/CUDA について、GT200 系と Fermi コアでの結果を示すとともに今後の課題についても説明する。

以下、本論文の構成を示す。2 章で BLAS ならびに CUDA 上での BLAS 実装系の実測データに基づく分析を行う。3 章で GPU に特徴的な性能のモデル化並びに実際のチューニング方法と速度安定化のための手法について示す。4 章で、チューニングの結果を示し、最後 5 章でまとめを行う。

2. BLAS and CUDA

CUDA 上で動作する BLAS として、NVIDIA 社が CUDA 環境と共に配布する CUBLAS⁴⁾ が有名である。また、テネシー大学の MAGMA プロジェクト⁵⁾ 内で開発している MAGMABLAS も高性能な BLAS である。特に、MAGMA プロジェクトでは自動チューニングにより一部の関数がチューニングされている⁶⁾。

本章の目的はこれら主要な BLAS 実装について DGEMV 関数の性能特性を調べることにある。使用する GPU は NVIDIA 社 GeForce GTX280(GTX280 と略記する) と Tesla C2050 である。ホスト-デバイス間のデータ転送は行わず、ホスト側からカーネル関数を呼び出し、デバイス側が終了するまでを測定する。全ての測定結果は 6 回のうち結果の悪い 1 試行を捨て平均をとっている。また、nvcc コンパイラのオプションは共通で以下のとおりである。

```
--compiler-options -fno-strict-aliasing \
-DUNIX -O3 \
-ptxas-options=-v \
-gencode arch=compute_20,code=compute_20 \
-gencode arch=compute_13,code=compute_13
```

2.1 DGEMV-N

本節では DGEMV-N (dgemv の行列の転置オプションを指定しない通常版、つまり、 $y := \alpha Ax + \beta y$.) の性能を中心に議論する。測定に使用した BLAS 実装は CUBLAS 3.1, MAGMABLAS 1.0RC2(論文執筆

```
__shared__ double buff[BLOCK_SIZE];
A += threadIdx.x+blockIdx.x*BLOCK_SIZE;
x += threadIdx.x;
for(int i=0; i<n_col; i += BLOCK_SIZE){
    __syncthreads();
    buff[threadIdx.x] = x[i];
    __syncthreads();
    #pragma unroll
    for(int j=0; j<BLOCK_SIZE ; j++){
        res += A[0]*buff[j];
        A += lda;
    }
}
```

図 1 MAGMABLAS DGEMV-N のコアループ抜粋 (変数名など一部変更している)

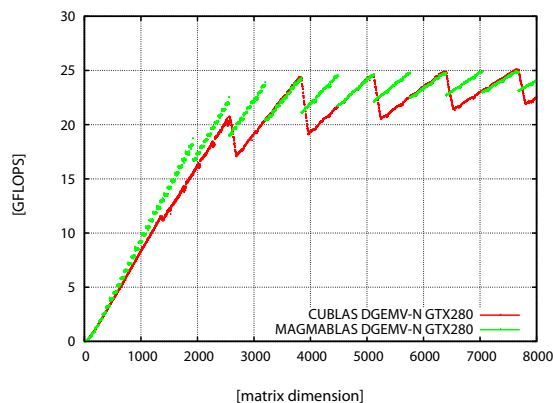


図 2 GTX280 での DGEMV-N の性能 (CUBLAS3.1 と MAGMABLAS1.0RC2 関数は magmablas_dgemv_tesla() を使用)

時の最新版) である。DGEMV-N 関数のコア部分を取り出したものが図 1 である。

2.1.1 GT200(GTX280) での性能特性

図 2 は、GTX280 での DGEMV-N の性能曲線である。横軸に行列の次元、縦軸に GFLOPS 値をとっている。CPU で見られる性能とは明らかに異なるノコギリ歯状の特性を示していることが分かる。性能の立ち上がりは直線的で、振動しながら 25GFLOPS に収束している。CUBLAS, MAGMABLAS 共に一定周期の振動をしているが、それぞれの周期は 1280, 640 と異なる。

実際のコードでは、定数 BLOCK_SIZE の値は 64 であり、ノコギリ歯周期の 1/10 である。BLOCK_SIZE は、ブロック内でのスレッド数であるとともに 1 重ループを 2 重ループに折畳む際の分割幅である。

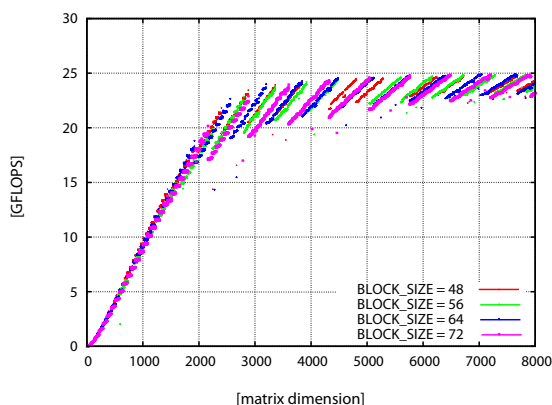


図3 ブロックサイズを変化させた DGEMV-N コアループの性能

各スレッドは対応する行要素の計算に割り当てられる。行列の次元が BLOCK_SIZE 増える毎に、スレッド・ブロックが増え、SM (Streaming Multiprocessor) に順々に割り当てられていく。GTx280 では搭載される SM の数が 30 であるため、全ての SM にブロックが割り当てられるのは、行列の次元が $30 * \text{BLOCK_SIZE} = 30 * 64 = 1920$ のときである。仮に、1920 次元の周期性を有する場合は、その理解は容易である。しかしながら、1920 は実際の DGEMV-N の性能特性上の周期よりも大きい。GT200 コアは 3SM で TPC (Texture Processor Cluster) を構成しており、TPC 内で共有される資源 (例えばメモリコントローラなど) の競合が $30 / 3 * \text{BLOCK_SIZE} = 640$ 次元周期で発生する。この周期が MAGMABLAS の DGEMV-N のノコギリ歯周期に一致する。BLOCK_SIZE を変化させた場合を見ても、 $10 * \text{BLOCK_SIZE}$ と一致する周期が確認できる (図 3)。

CUBLAS はソースコードが公開されていないため周期的な振る舞いははっきりと説明できない。しかしながら、両者の振る舞いが極めて類似していることから、もし MAGMABLAS と同様の理由でノコギリ歯周期が起きているのであれば、MAGMABLAS の倍の周期であるから $\text{BLOCK_SIZE}=128$ と推測される。

2.1.2 GF100(Tesla C2050) での性能特性

GF100(Fermi コア) の Tesla C2050 は 14 個の SM を持つため、 $14 * \text{BLOCK_SIZE} (=14 * 64=896)$ 次元もしくはその倍数での周期があると予想される。しかし、図 4 に示す性能特性にあるように、特段の短い周期性は見られない。MAGMABLAS では 7169 ($=14 * 64 * 8 + 1$) 次元に性能の大きな落ち込みが確認できる (搭載メモリの制限があるためこれ以上の次元で実行して、周期性を確認することはできない)。CUBLAS では 7169 次元に小さな落ち込みが見られるが、8193 次元

に MAGMABLAS と同程度の性能の落ち込みが確認できる。

ハードウェア構成から性能の落ち込みの要因に次のものが考えられる。

- (1) 共有メモリ容量
- (2) レジスタ数
- (3) SM 上でキューイング可能なスレッド・ブロック数
- (4) SM 上で動作可能な Warp 数

DEGEMV-N での各スレッド・ブロックの共有メモリ使用量は $\text{BLOCK_SIZE} * 8 (=512)$ バイトである。7169 次元では 9 スレッド・ブロックが必要であるから、4.5K バイトあれば十分である。Tesla C2050 は最大 48K バイトの共有メモリを持つため、要因 (1) となる共有メモリを使い切ることはない。

実際は、要因 (3) であるキューイング可能なスレッド・ブロック数が 8 に制限されて、8 スレッド・ブロック分しか同時実行されないことに起因すると考えられる。7169 次元では 8 スレッド・ブロック + 1 スレッド・ブロックに分けられて実行される。CUDA では複数のスレッド・ブロックを一つの SM の中に共存させ、Warp の多重キューイングを促進させ性能が向上する。当然、8 スレッド・ブロックの実行状態と 1 スレッド・ブロックの実行状態では、前者の方が性能面で優れている。

8 スレッド・ブロックと 1 スレッド・ブロックでの性能のアンバランスが 7169 次元での急激な性能の落ち込みの原因と解釈できる。検証のため、共有メモリの使用が 7 スレッド・ブロックで 48K バイトを超えるように設定し、5 から 8 スレッド・ブロックまでしか動作できない状況にして DGEMV-N を測定した (図 5)。スレッド・ブロック数が 5 から 8 に変化する際の性能特性が連続に変化していることが読み取れる。従って、SM に割り振られるスレッド・ブロック数のアンバランスが生じる際に性能の落ち込みが起きているといえるだろう。この現象は、 $8k$ スレッド・ブロック + 1 スレッド・ブロック (k は整数) が必要となった際に起こるので周期性がある。

また、図 5 から読み取れるように、ブロックサイズを固定した場合にスレッド・ブロック数を $8+1$ から $7+2$, $6+3$, $5+4$ と強制的にキューイングされるスレッド・ブロック数を制御することで性能のアンバランスを解消して僅かではあるが性能劣化を改善することができる。

2.2 DGEMV-T

本節では DGEMV-T (DGEMV の行列の転置オ

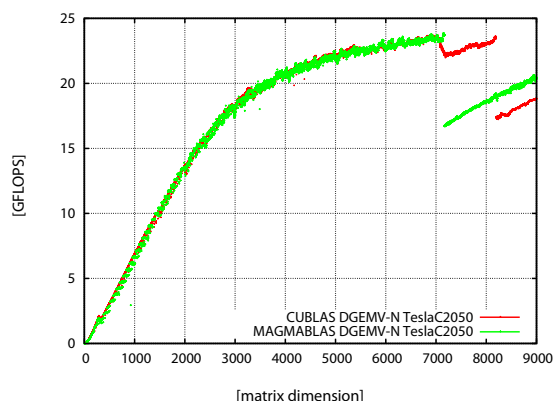


図 4 TeslaC2050 での DGEMV-N の性能 (CUBLAS3.1 と MAGMABLAS1.0RC2 関数は `magma blas_dgemv_fermi()` を使用)

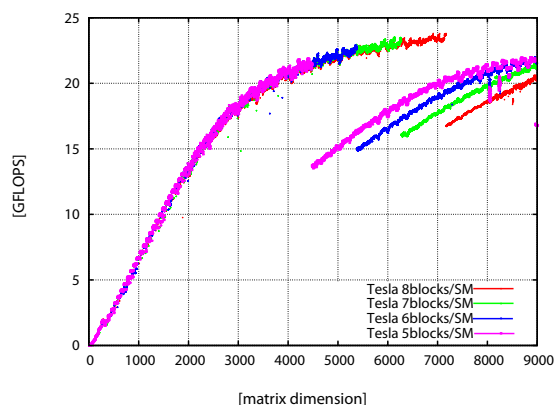


図 5 TeslaC2050 で共有メモリ利用を制御して、同時動作可能なスレッド・ブロック数を 8 以下にし測定した MAGMABLAS DGEMV-N の性能

ブションを指定するもの。つまり、 $y := \alpha A^T x + \beta y$ の性能を中心に議論する。性能測定に使用した環境は DGEMV-N と同様の CUBLAS 3.1 と MAGMABLAS 1.0RC2 (図 6) に加えて、今回独自に開発したもの (図 7, 本稿では以下 MYBLAS と呼ぶ) を取り扱う。

3 コアの特徴は次のようになる。

- MAGMABLAS のコアその 1 は行列ベクトル積を複数の内積計算に分け、それぞれ複数スレッドを 1 内積計算に割り当てて計算している。Fermi コア (GT100) 向けに利用される。
- MAGMABLAS のコアその 2 は、スレッドを 2 次元に配置し、行列データをできるだけコアレスにアクセスし、共有メモリ上で行列を転置してアクセスする。
- MYBLAS コアは MAGMABLAS のコアその 1

```
// コアその 1
__shared__ double w[THREAD_SIZE];
A += BLOCK_ID; x += threadIdx.x;
for(int i=0; i<n1; i+= THREAD_SIZE)
    r += A[i] * x[i];
sumup(&r,w);

// コアその 2
#define th_x    threadIdx.x
#define th_y    threadIdx.y
__shared__ double buff[32];
__shared__ double a[16][4][4];
A += START_COL*lda; x += THREAD_ID%32;
for(int i=0; i<n1; i+= 32 ){
    buff[THREAD_ID] = x[i];
    for(int itr=0; itr<2; itr++, A+=16){
        for(int j=0; j<4; j++){
            a[j][th_y][th_x]=A[j*4*lda];
            __syncthreads();
            for(int j=0; j<4; j++){
                r += a[th_x][th_y][j]
                    * buff[j+16*itr+th_y*4];
            }
        }
    }
}
sumup(&r,w);
```

図 6 MAGMABLAS DGEMV-T のコアループその 1 とその 2 抜粋 (変数名などは変更している、主要部分のみを掲載しており実際のコア 2 はループ展開がされている。関数 `sumup(r)` は共有メモリ `w` を通じてレジスタ `r` の総和を計算する。Fermi コアでは行列の次元が 128 以下の時コアその 2 を使用する。GT200 ではコアその 2 にブロックサイズが異なる関数も用意されている)

が 1 内積計算を割り当てている部分を、複数の内積計算にしているため、ベクトル x のアクセスにおいて他の 2 コアに比べて有利となる。また実コードでは、2 ないし 8 段展開があり、パラメタ `BLOCK_SIZE` を持つ。さらに、 x のアクセスにテクスチャメモリを使用する。

2.2.1 GT200(GTX280) での性能特性

図 8 は GTX280 での DGEMV-T の性能曲線である。CUBLAS, MAGMABLAS いずれもノコギリ歯周期が確認できる。周期はそれぞれ 1024 次元と 960 次元である。一方、MYBLAS は周期性はあるものの CUBLAS や MAGMABLAS ほどにはシャープな形状ではない。高次元では周期が確認しにくい、2000 次元までの形状から 480 次元周期と判断できる。

```

#define UMAX 4
__shared__ double w[THREAD_SIZE];
col_m = (n_col/BLOCK_SIZE)*BLOCK_SIZE;
row = blockIdx.x * UMAX;
A += threadIdx.x; x += threadIdx.x;
ak = A + row*lda;
#pragma unroll 8
for(int i=0; i<col_m; i+=BLOCK_SIZE ){
    s0 += ak[0]*x[i];
    s1 += ak[lda]*x[i];
    s2 += ak[lda*2]*x[i];
    s3 += ak[lda*3]*x[i];
    ak+=BLOCK_SIZE;
}
{ int i=col_m; if(i+threadIdx.x<n2){
    s0 += ak[0]*x[i];
    s1 += ak[lda]*x[i];
    s2 += ak[lda*2]*x[i];
    s3 += ak[lda*3]*x[i];
}}
sumup(&s0,w); sumup(&s1, w);
sumup(&s2,w); sumup(&s3, w);

```

図 7 MYBLAS DGEMV-T のコアラープ抜粋 (4 段展開を示した。他に 2,8 段展開がある。変数名などは変更している。関数 sumup は共有メモリ w を通じて引数レジスタの総和を計算する。)

MAGMABLAS と MYBLAS はそれぞれ 16, 8 行分の計算を 1 ブロックとして SM にタスクを割りつけるので、この周期は丁度 60 ブロックを割り付けた直後のタイミングである。SM 上では 2 スレッド・ブロックが動作する状況であり、スレッド・ブロックや warp 数の制約からの周期性ではない。MYBLAS コアの nvcc でのコンパイル結果からレジスタの消費が非常に多く (実際、ptx ファイルでは 64 ビット浮動小数点レジスタが 332 個使われている)、そのため 3 ブロック目の割り当てでレジスタスピルを起こした可能性がある。

2.2.2 GF100(Tesla C2050) での性能特性

図 9 は Tesla C2050 での DGEMV-T の性能曲線である。CUBLAS, MAGMABLAS いずれもノコギリ歯周期が確認できる。周期はそれぞれ 1024 次元と 2240 次元である。MAGMABLAS では行列の次元数分のスレッド・ブロックが作成されるため、2240 次元では 2240 スレッド・ブロック、1SM あたり 160 スレッド・ブロックである。Tesla C2020 での DGEMV-T の周期性の原因は、これまで挙げてきた各種物理的な

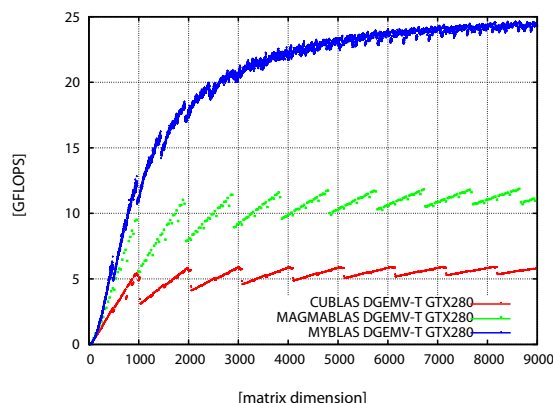


図 8 GTX280 での DGEMV-T の性能 (CUBLAS3.1 と MAGMABLAS1.0RC2, MAGMABLAS は dgemvt を使用 & 32 の倍数次元のみ動作確認し測定した。MYBLAS は 8 段展開を使用し BLOCK_SIZE=128.)

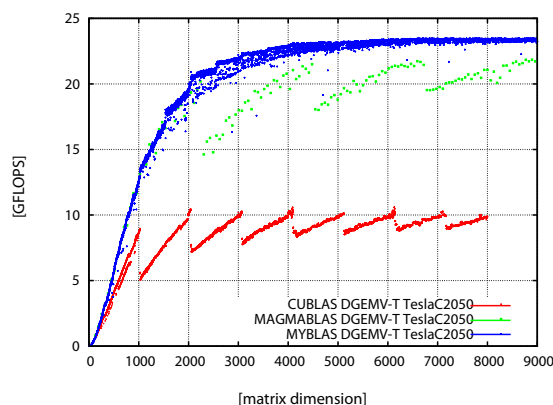


図 9 TeslaC2050 での DGEMV-T の性能 (CUBLAS3.1 と MAGMABLAS1.0RC2, MAGMABLAS は dgemvt を使用 & 64 の倍数次元のみ動作確認し測定した。MYBLAS は 2 段展開を使用し BLOCK_SIZE=128.)

制約に当てはまらない数字であるが、GPU 内部で何らかの資源競合が起こっており、周期的にそれが顕著に現れると判断される。現時点では、資源競合の要因がはっきりとしないが詳細な解析によって明らかにすることが今後の課題の一つといえる。なお、MYBLAS は周期的な振る舞いは認められるが、極めて小さい。

2.3 CUDA BLAS の性能特性まとめ

本章で示した DGEMV 関数の測定結果から、CUDA で作成された関数でノコギリ歯状の周期特性をもつものは、何らかの資源競合が原因となっていると推測される。多くは、有限の資源を多数のタスクで均等に割り付けたり分配するときに起こる、分配のアンバランスさやハードウェア上の制約に起因していると考えられる。

表 1 ノコギリ歯周期の原因となる可能性のあるハードウェア構成要素

	GTX280	Tesla C2050
TPC	3SM/TPC	—
SM 上でキューイング可能なスレッド・ブロック数	(8)	8
共有メモリ容量	(16KB)	16KB or 48KB
レジスタ容量 (32 ビット長レジスタ数)	16K	(32K)

ここまでの議論で、周期性の要因となる可能性のあるものについて表 1 にまとめた。なお、本論文中で直接周期性の要因となっていないものは括弧で示した。

3. 性能自動チューニング型 BLAS/CUDA

3.1 性能特性から解析モデルの導出

性能特性の測定結果から、BLAS/CUDA の性能は各種の資源配分の周期性によって大きな影響を受けることが分かっている。カーネルループの内部のコストが $O(N)$ であると仮定し、資源配分が均等に行われれば、それぞれのスレッドのコストはカーネルループ単体コストに重複度分の係数が乗じられることになる。行列ベクトル積のコストは一般的に次の様に書くことができる。

$$T(N) := o_0 + \sum_{i=1}^m (o_i + \alpha_i N) \lceil N/\beta_i \rceil \quad (1)$$

ここで、 o_* はループ立ち上がり等のオーバーヘッド、 α_* は速度係数である。また、 β_* は周期を表す。DGEMV-N のソースコードの様にカーネルループをさらに分割して多重ループ化し、式 (1) を更に複雑にすることもできる。

$$T(N) := o_0 + \sum_{i=1}^m (o'_i + \sigma_i \lceil N/\gamma_i \rceil + \alpha'_i N) \lceil N/\beta_i \rceil \quad (2)$$

なお、 N がとる値の範囲を限定すれば、式 (2) は区分的に式 (1) の形に帰着できるし、実用上、多くで切り上げ演算を括弧で置き換えるので、本論文のモデル化では式 (1) を標準形として用いる。

式 (1) より DGEMV の性能は

$$P = \frac{2N^2}{T(N)} = \frac{2N^2}{o_0 + \sum_{i=1}^m (o_i + \alpha_i N) \lceil N/\beta_i \rceil} \quad (3)$$

$\lceil N/\beta \rceil - N$ が周期と最大値が $\beta - 1$ のノコギリ歯状の特性を持つ。これを $S(N)$ で表せば、 $\lceil N/\beta \rceil = N + S(N)$ であるので、式 (1) が複数のノコギリ歯状の波を重ね合わせた性質を表現することになる。

となる。 $m = 1$, $o := o_0 + o_1 \ll \alpha_1 N \lceil N/\beta_1 \rceil$ とすると、1 次近似として次式を得る。

$$P \sim \frac{2N}{\alpha_1 \lceil N/\beta_1 \rceil} \left(1 - \frac{o}{\alpha_1 N \lceil N/\beta_1 \rceil} \right) \quad (4)$$

DGEMV-N on GTX280

上式 (4) で第 2 項を落とせば β_1 周期のノコギリ歯関数となる。GTX280 上で実行した DGEMV-N では短周期の振動は微小で、特徴的な 1280 もしくは 640 の周期 (それぞれ CUBLAS と MAGMABLAS) が主要なモードとして見る事ができるので、式 (4) で $\beta_1 = 1280$ or 640 とすることで DGEMV-N の性能をよく近似したモデル化ができる。

DGEMV-T on GTX280 and Tesla C2050

GTX280 と Tesla C2050 上で実行した DGEMV-T でも 1024 もしくは 960 周期 (それぞれ CUBLAS と MAGMABLAS) の主要モードとしてノコギリ歯周期が見えるので、式 (4) を利用するとうまく説明ができる。

MYBLAS の性能特性の解釈は o が相対的に大きくなり、短周期、つまり β_* が小さいときに対応できる。この場合は、整数切り上げ演算 ($\lceil \cdot \rceil$) を通常の括弧に書き換えても、良い近似として扱うことができる。つまり、 $\beta_i \ll \alpha_i N$ のときは、

$$P \sim \frac{2N^2}{o_0 + \sum_{i=1}^m (o_i + \alpha_i N) N/\beta_i} \quad (5)$$

$$= \frac{2N^2}{o + aN + bN^2} \quad (6)$$

としてよい。

DGEMV-N on Tesla C2050

コスト式 (3) で α_i が小さいとき、 P をプロットしたグラフは巨視的には滑らかなグラフに近くなる。また、 $m > 1$ とすれば、複数の振動モードが性能曲線に乗ることになる。Tesla C2050 での DGEMV-N の性能は、ほぼ滑らかな関数の上にノコギリ歯周期の関数が乗っているので、 β_1 のみ大きな周期で、それ以外の短周期の項は前節のように近似してやればよい。

$$P \sim \frac{2N^2}{o + aN + bN^2 + (o_1 + \alpha_1 N) \lceil N/\beta_1 \rceil} \quad (7)$$

本節で示したコストモデルは非常に単純なもので、7) や 8) にある様に、定性的な分析から詳細なコストモデルを組み上げる研究がいくつか存在する。本研究では、DGEMV のコアソースに限定して実測データからモデル化の議論を進めるため、モデル構築のアプローチが異なるといえる。

3.2 性能安定化・性能チューニング

前節までの実験観測や議論から、長周期のノコギリ歯状の性能特性と短周期の場合はコスト関数の近似が異なり、長周期は無視することは難しいが短周期は滑らかな関数の誤差ととらえても問題ないといえる。

滑らかな性能を示す場合は一般的に巨視的なチューニング (大雑把なサンプリングで行うチューニング) でも、ある程度の高性能が保証できる。しかしながら、ノコギリ歯状の特性が残る場合は局所的な詳細なモデルを用いたチューニングを行わなくてはならない。最悪の場合、全ての可能性のある問題サイズでチューニングが必要となる。そのような、最悪のケースでのチューニングを前節でモデル化したコスト関数を用いて CUDA 環境における DGEMV 関数のチューニングコストを削減するのが本節の目的である。従って、ノコギリ歯状の性能を示す GTX280 上での DGEMV-N を中心に議論を進めるが、他の関数についても大域的にチューニングにおける CUDA 特有の注意点やチューニングした結果を組み合わせる更なるチューニングを行うプロセスについても説明していく。

3.2.1 DGEMV-N on GTX280

GTX280 での DGEMV-N の性能はコアループから BLOCK_SIZE を変化させることができる。図 3 での測定結果から分かる様に、行列の次元に応じて適切な BLOCK_SIZE を選択すれば、シャープなノコギリ歯周期はなくなる。今回の性能チューニングでは主モードのみをモデル化することで、コスト関数を単純化する (つまり式 (3) で $m = 1$ とする)。

モデル化に必要なパラメタ $\alpha := \alpha_0 + \alpha_1, \alpha_1$ は、実測データ、ただしデータ数削減のため、各 BLOCK_SIZE で決まるノコギリ歯周期の頂点に限定し測定する。測定データをもとに、最小自乗近似によってパラメタを推定する。実際には、gnuplot の fit 機能を使用するが、一連の測定・推定の流れは自動で行うことができる。各 BLOCK_SIZE でのパラメタが定まれば、式 (3) に従って、与えられた行列の次元に対して最も良い性能を与える BLOCK_SIZE のコアループを選択し実行すればよい。つまり、図 3 の曲線群の中から上限に位置する BLOCK_SIZE のみで動作する DGEMV-N 関数を自動作成するというイメージである。

3.2.2 DGEMV-N on Tesla C2050

Tesla C2050 での DGEMV-N の性能特性は $N=7169$ での大きな落ち込みは見られるが、GTX280 の様に適切な BLOCK_SIZE を選択することで改善が期待できる。前節でも推測したように、CUBLAS の BLOCK_SIZE は 128 と見られる。従って、BLOCK_SIZE=

128 として、レジスタの利用数が利用可能数を超えないように (レジスタスピルを起こさないよう) アンローリング段数 (プラグマ unroll へのオプション) を決めればよい。通常は 16 まではレジスタスピルせずに十分な性能が得られる。

Fermi コアでは SM 上で動作可能な Warp 数が 48warp であるため、SM 上でキューイング可能なスレッド・ブロック数との制約から BLOCK_SIZE の上限は 196 となる。従って、BLOCK_SIZE とアンローリング段数についてそれぞれ 128 から 196、16 以下でレジスタスピルせず最高性能を出す組み合わせを選び出せばよい。BLOCK_SIZE はスレッド数指定に使われるので、32 の倍数が望ましい。従って、BLOCK_SIZE の候補は、128、160、192 の 3 候補に絞られる。また、アンローリング段数はできるだけ BLOCK_SIZE を割り切り、かつ、2 のべきが望ましいので、候補はあまり多くならない。

3.2.3 DGEMV-T on GTX280 or Tesla C2050

DGEMV-T は MYBLAS が他の 2 実装よりも高速で安定である。MYBLAS の DGEMV-T は展開の段数と BLOCK_SIZE の 2 パラメタを有する。これらパラメタの決定には Tesla C2050 での DGEMV-N と同様に考えていけばよい。展開段数は最内コアループのアンローリング段数を適切に選びレジスタスピルを避けるようにすればよい。また、BLOCK_SIZE はスレッド数指定に使われるので、16 または 32 の倍数から探索する。

3.2.4 補 足

複数のパラメタ候補から探索する場合は、サンプリング点を予め決めておき各サンプリングポイントでの点数をつけその総和で順位をつけるポイント方式を採用する。総合ポイント上位のパラメタについて、コストモデルに基づく推定を行い、サンプリング点以外の性能推定に基づき最良パラメタを決定する。アルゴリズムをまとめると以下になる。項目 (1) から (6) まではプログラムの実行前 (インストール時) に行うものである。

- (1) パラメタ候補を $\Lambda = \{\lambda_1, \dots, \lambda_k\}$ とおく。
- (2) サンプリング点集合 $X = \{x_1, \dots, x_m\}$ を決める。
- (3) 各パラメタ候補値でのコアループを作成し、 X

なぜなら、196 スレッドを 1 ブロックに割り当てることになるので $196/32=6[\text{warps/block}]$ 、最大キューイング可能なスレッド・ブロック数を乗じると $8*6=48[\text{warps/SM}]$ となり上限値に達する。

でのサンプリングを行う。

- (4) 各サンプリング点 x_i で、順位をつけ各パラメタ候補 λ_j のポイント p_{ij} を決定する。
- (5) パラメタ λ_j のポイント $p_j = \sum_{i=1}^m p_{ij}$ を算出。
- (6) ポイント上位のパラメタ集合 $\Gamma = \{\gamma_1, \dots\}$ に対して、最小自乗近似により式 (6) にフィッティングし、コスト近似関数 $P_j(x)$ を定める。
- (7) プログラム実行時には、 x に対して、最良コストを与えるパラメタ s を定め、パラメタ λ_s に対応するオブジェクトを選択し、実行する。

次章での実際のチューニングでは (1000, 3000, 5000, 7000, 9000) の 5 点をサンプリング点として使用した。

4. チューニング結果とその評価

本論文で扱った DGEMV-N, DGEMV-T は実行環境 GT200, GF100 においてその性能特性が異なることはすでに説明してきた。MYBLAS を用いて性能が滑らかだと判断される DGEMV-N on Tesla, DGEMV-T on GTX280, DGEMV-T on Tesla C2050 についてはポイント方式に基づきチューニングを施したが、本論文で利用したカーネル関数の範囲では 3 つの場合で全て 1 つのパラメタが最良の結果となった。すでに図 8, 9 に示した性能特性のグラフにあるものがチューニングの結果と一致している。グラフの再掲は行わないので、当該図を参照してほしい。

DGEMV-N on Tesla については、ポイント方式で BLOCK_SIZE=128 (アンローリングは 8 段) が滑らかで最良なカーネル関数と判定された。図 10 に、チューニングの結果を示す。パラメタ選定の過程でコスト関数が滑らかと仮定してフィッティングをしたが、図 10 は 6832 次元以降にノコギリ歯形状 (性能の落ち込み) を示している。当初の滑らかであるという仮定が許容できるかは判断付きにくい、現データでは 1 位, 2 位が逆転する僅かな区間で最良パラメタを選択できていない。したがって、ポイント方式で上位を選択した後に、サンプリングを細かくしフーリエ解析などで周期性をチェックし、コスト関数の形状の設定を行うなど改良すべきであることが分かる。

最後に、GT200 コア (GTX280 と GTX285 を使用) で DGEMV-N をチューニングした結果を図 11 に示す。ノコギリ歯の形状は残るものの変動幅は確実に小さくなっている。

5. ま と め

本研究では、まず、GPGPU の 1 環境である CUDA において線形計算代数のコアとなる BLAS (特に

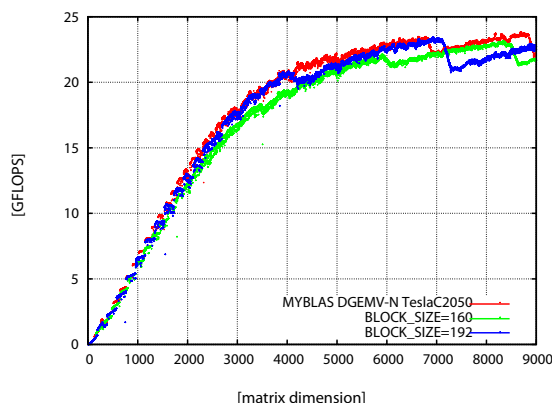


図 10 Tesla C2050 での DGEMV-N のチューニング結果 (赤がチューニングで選択されたもの BLOCK_SIZE=128, 緑は BLOCK_SIZE=160, 青は BLOCK_SIZE=192, アンローリング段数はいずれも 8 が選択されている。)

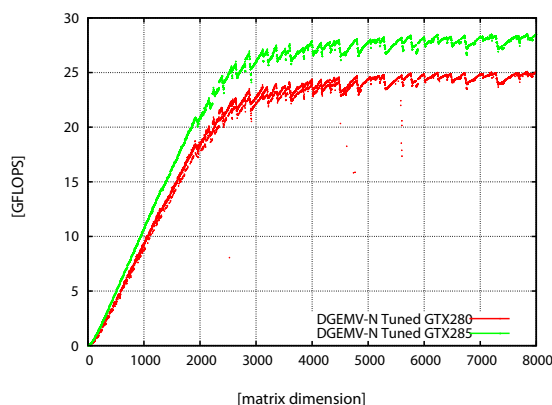


図 11 GTX280 と GTX285 での DGEMV-N のチューニング結果

DGEMV 関数) の性能特性を調査し、その中にノコギリ歯状の周期が現れることを確認した。これらは、GPU 上の物理上の制約によるものであると知られているが、本論文では DGEMV のコアループコードに限定してその周期性の原因要素を調査した。また、ノコギリ歯状の性能特性を示すコアループのパラメタを制御することでノコギリ歯のシャープな山の高さをできるだけ低くし、最良性能を選択できる自動チューニング手法を提案した。また、荒いサンプリングからポイント方式でよりよいパラメタを決定する方法の適用によってほぼ半自動的に最良の性能チューニングが達成された。

周期性の殆どは、多くのタスクが計算資源に均等に割り当てられる過程で発生する。つまり、負荷アンバランスに起因すると解釈できる。本論文では、ハードウェアの仕様から明らかになっている 2 要素について周期性への寄与が示唆された。これら要素は GPU

の世代が変わるごとに仕様変更が加わるが、計算コアの構成は大きく変わらないので周期性の寄与は将来に渡って関連し続けることが予想される。

GPU の世代が変わってもノコギリ歯状の性能特性を示すのであれば、本論文で示した DGEMV-N on GTX280 の性能チューニング手法によって性能安定化することができる。ハードウェアの機能拡張によってノコギリ歯のシャープさが緩和されても、資源の等分配割り当ての不均衡は必ず生じる。本論文では、滑らかな関数にノコギリ歯状関数が乗ったコスト関数(本文中では式(7))を用いたチューニングの提案は行っていないが、今後は必要になると考える。実際、DGEMV-N on Tesla において式(7)を利用したほうがよいとの結果も示されている。本研究では、周期性の判別に人間の判断を必要とした。つまり、半自動的なチューニングにとどまっている。フーリエ解析などのデータ解析手法を利用して完全自動化することが次のステップとして必要な項目である。今後は、DGEMV 以外の Level 2 BLAS 関数を中心に(当然、Level 3BLAS 関数についても)調査を行い、より一般的・汎用的な GPGPU での自動チューニングの研究を進める予定である。

最後に、GPGPU の性能特性モデルに関して有用な情報を頂いた須田礼仁先生ならびに滝沢寛之先生に感謝いたします。また、本研究は科研費(21300013 ならびに 22104003)の支援を受けています。

参 考 文 献

- 1) 中里: 行列乗算カーネルの性能評価, 情報処理学会研究報告, Vol. 2010-HPC-127, No. 3, (2010/10/13).
- 2) 棕木, 高橋: GPU による 4 倍精度 BLAS の実装と評価, 情報処理学会研究報告, Vol. 2009-ARC-186, No. 13, & Vol. 2009-HPC-123, No. 13, (2009/12/1).
- 3) Tomov, S., Nath, R., and Dongarra, J. : Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing, *Parallel Computing*, Vol. 36 Issue 12, pp. 645–654 (2010)
- 4) NVIDIA : CUDA CUBLAS Library, http://developer.download.nvidia.com/compute/cuda/2.3/toolkit/docs/CUBLAS_Library_2.3.pdf, (2009).
- 5) MAGMA プロジェクト, <http://icl.cs.utk.edu/magma/>.
- 6) Nath, R., Tomov, S., and Dongarra J. : Autotuning dense linear algebra libraries on GPUs and overview of the MAGMA library, *Proceedings of PMAA'10*, (2010).
- 7) Hong, S., Kim, H., : An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness, *Proceedings of ISCA'09*, (2009).
- 8) Baghsorkhi, S.S., et al. : An Adaptive Performance Modeling Tool for GPU Architectures, *Proceedings of PPOPP'10*, (2010).