Vol.2013-HPC-138 No.7
2013/2/21

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

今村 俊幸[1,5,a)]　林 熙龍[2]　内海 貴弘[3]　山田 進[4,5]　町田 昌彦[4,5]

概要：異なる 2 世代の GPU コアにおいて基本線形計算の一つである SYMV カーネルの自動チューニングについて報告する．SYMV カーネルは対称性を利用したループ変換と Atomic 操作によって排他制御を行うことで Fermi コア上で従来の CUDA-BLAS ライブラリの実装よりも高速化される．この実装には 4 つのパラメータがあり行列の次元を数えるとその探索空間の大きさは膨大なものとなる．本研究ではチャンピオン方式によるパラメー タ篩い落としとともに d-spline を用いたデータ内挿と推定を実施して最良のパラメータを選択する．実機 Tesla C2050, GeForce GTX580, Tesla K20 においてこれらを実施してほぼ最良の SYMV カーネルの構築がなされた．

Performance Tuning for the SYMV kernel on multiple GPU generations, Fermi and Kepler

Abstract: We report automatic performance tuning for the SYMV kernel, which is one of the basic linea algebra operations, on different two generation GPU cores. SYMV kernel is optimized by exclusive control using an atomic operation and loop tansformation taking into account the symmetry of matrix. It outperforms the existing CUDA-BLAS libraries on the Fermi core GPU. There are four parameters to be tuned up, and the size of parameter space becomes enourmous when we count up the dimension of the matrix to be calculated. In this study, the parameters of the best performing is seleceted via parameter sieving by the champion ranking scheme and estimation and interpolation data using d-spline function. Building the best SYMV kernel has been made in applying these on a Tesla K20, a Tesla C2050, and a GeForce GTX580.

1. はじめに

GPU カーネルの世代交代はかなり頻繁でありかつコア数や動作周波数, 搭載メモリなどの違いによる GPU の種類は膨大なものになる．現在主流の GPU コアは Fermi と 2012 年にリリースされた Kepler であるように常に複数の GPU コアが存在している．表 1 はスペック表に現れる数字上の違いが見て取れる．ハードウェア種の豊富さは性能最適化のチューニングフェーズにおけるコスト増大に直結する．また, CUDA5 で採択されたダイナミックな GPU コアの柔軟な利用形態は, GPU 上に再構成可能な GPU を配置していることに近く, GPU 単体であってもその性能チューニングは固定的ではなくなることが予想される．

表 1　主要な GPU カーネルのスペック
Table 1　Specification of the typical GPU kernels

	Tesla K20 (Kepler)	Tesla C2075 (Fermi)
CUDA コア数	2496	448
GPU メモリ	5GB GDDR5	6GB GDDR5
メモリバンド幅	208 [GB/s]	144 [GB/s]
ピーク性能 (DP)	1.17TFLOPS	515GFLOPS
(SP)	3.52TFLOPS	1.03TFLOPS

行列線形計算の基本演算を行う BLAS(Basic Linear Algebra Subprograms) は各種プロセッサに最適化され, GPU においても NVIDIA が CUDA[1] にバンドルしている

[1] 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
[2] 電気通信大学電気通信学部
University of Electro-Communications
[3] 電気通信大学大学院情報理工学研究科
Graduate School of Informatics and Engineering, University of Electro-Communications
[4] 日本原子力研究開発機構
Japan Atomic Energy Agency
[5] 科学技術振興機構 CREST
CREST JST
[a)] imamura.toshiyuki@riken.jp

CUBLAS[2], CULA[4] にバンドルされている CULABLAS, MAGMA[3] にバンドルされている MAGMABLAS, その他に KBLAS, GLAS[5], ASPEN.K2 などが存在している. MAGMABLAS, GLAS, ASPEN.K2 では自動チューニング技術が導入されており, 一部の GPU カーネルに特化したチューニングの施されたカーネルコードが公開されている. しかしながら, パラメタ探索や自動チューニング機構を含む形では公開がなされてはいない.

CUDA-BLAS(GPU 上での CUDA で実装された BLAS の総称を指す) の自動チューニングの方法論は GLAS の開発者によって議論がなされており [6], GEMV(行列ベクトル積) に対して縦横アスペクト比の違いに応じて最適カーネル関数がことなり, それらをテーブル方式により選択する. テーブルの作成方法について詳細は不明であるが, 性能チューニングされた GEMV を提供している. 著者らは先行研究 [7] や SIAM PP2012[8] において CUDA における資源配分 (割付け) 方式がラウンドロビンが原則であることに着目したコストモデルを導入して, チャンピオン方式によるパラメータ探索方式と組み合わせてテーブルを使用しないモデル関数評価方式による GEMV-[T|N] の最適カーネル関数選択方法を報告している. なお, コストの近似には非線形最小二乗近似法を用いるが, 関数の形状を経験的に定める必要のない d-spline という手法も知られており, 汎用的なチューニングの枠組み組み込みの可能性がある.

さらに, 著者今村は VECPAR2012[9] においてアトミック操作を使用した SYMV カーネルの実装方式を提案し, MAGMABLAS などの既存 CUDA-BLAS よりも高性能な SYMV カーネルの実装に成功している. Tesla コア (GT200) までではアトミック操作が重たく想像できない実装であったが, Fermi コア以降は高速化されたアトミック操作のサポートによりシリアライズされないような工夫で本手法が現実的になった.

2012 年に登場した Kepler コアは CUDA コア数が飛躍的に増加し, カーネル関数に指定すべきパラメタ値は確実に異なることが予想される. また, Tesla K20 では GeForce なみにメモリバンド幅が増強されており, メモリバンド幅で性能がバウンドされる Level 2 BLAS には魅力的な環境の出現ともいえる. 本報告の主眼は, アトミック操作を使用した DSYMV カーネルを Kepler コアならびに Fermi コア上でチューニングし, その過程で d-spline 関数を用いた性能推定を行う試みをすること. さらにそれらの問題や最終性能を報告することを目的とする.

2. SYMV カーネル

SYMV カーネルは BLAS ライブラリに含まれる次のような対称行列ベクトル積 API である. 対称行列の格納方法に上三角, 下三角のオプションが存在する.

$$y = \alpha Ax + \beta y \tag{1}$$

特にプロセッサを想定しないが, 対称性を考慮した FORTRAN コードは次のようになる. 一般的に GEMV のような行列ベクトル積では, 2 次元配列の 1 アクセスに対して 1 積和演算しかなされないが, 最内ループにおける 2 次元配列 a のアクセスに対して 2 積和演算ができるような最適化がなされている. これにより, 要求 Byte/flop が 1/2 にできるため, 他の Level2 BLAS に比較して最大で 2 倍のスループット向上が期待できる.

2.1 Atomic アルゴリズム

CUDA で実装するにも図 1 同様の計算の流れを採用する. CUDA での実装では最外側ループをスレッドブロック単位での分割に充てて, 内側ループにスレッド並列に充てる. ここで, 配列 w にスレッドブロック間の競合性, 変数 y0 にスレッド間の競合性が存在する. 本研究では, この競合性の解消性をアトミック操作による排他制御を行うことにより実現する.

(1) スレッド間の競合性排除は共有メモリを介した総和計算で実現できる. 共有メモリを介した総和計算は NVIDIA のチュートリアル資料にあるためここでは説明は控えたい.
(2) ブロック間の競合性排除は対象配列 w に対するものである. 以下の Atomic アルゴリズムにより排他制御する.
  (a) まず, 配列 w のインデックスをスレッド数単位で処理する区間に分ける.
  (b) その各区間に対して mutex 変数を設定する. つまり, スレッドのブロックサイズを BLOCK_SIZE とすれば (n-1)/BLOCK_SIZE 個の要素の mutex を配列として確保する.
  (c) スレッドブロック中のマスタースレッドが更新しようとする配列 w の区間に対応する mutex を捕まえるまで空ループを回し, マスタースレッドが mutex を捕まえた後でスレッドブロック内で同期をとる.

```
w(1:n)=0
do i=1,n
   y0=0
   do j=1,i-1
      y0  =y0  +a(j,i)*x(j)
      w(j)=w(j)+a(j,i)*x(i)
   enddo
   y(i)=y0+a(i,i)*x(i)
enddo
y(1:n)=y(1:n)+w(1:n)
```

図 1 SYMV カーネル簡易コード
Fig. 1 SYMV kernel, simplified code

(d) 対応する区間の配列 w の更新を実施する.
(e) スレッドブロック内で同期をとった後, マスタースレッドは捕まえた mutex を開放する.

実際, mutex の実装には atomicCAS と atomicExch 関数を使用して実装する.

2.2 L+U ハイブリッドアルゴリズム

VECPAR2012 の報告 [9] では行列サイズが小さいときに, オーバヘッドが少なく小規模問題で高性能な L+U アルゴリズムを切り替えて使用している.

$$Ax = (L + D + U)x = \tilde{L}x + Ux \quad (2)$$

本研究でも L+U アルゴリズムを採用したカーネル候補として Atomic アルゴリズムのカーネル群と切り替えて使用する.

3. 自動チューニング

本節では SYMV カーネルの自動チューニングについて説明する. まず, 性能を左右するパラメータについて述べ, パラメータ探索方式, 性能予測に使用する d-spline による内挿方法について述べていく.

3.1 SYMV カーネルの性能パラメータ

Atomic アルゴリズムに基づく SYMV カーネルには 4 つの性能パラメータが存在する. 本節はそれらの詳細を示していく.

3.1.1 ブロックサイズ: BLOCK_SIZE

SYMV カーネルの説明にもあったように, スレッドブロック内のスレッドグループ形状が第一のパラメータとなる. これは, 一般的な CUDA アプリケーションでもどうようである. アルゴリズムの構成上スレッドは 1 次元形状であるため BLOCK_SIZE のみで制御される. 一般的にワープの倍数が望ましいので BLOCK_SIZE={32, 64, 96, 128, 160, 192, 224, 256} の 8 通りが候補である.

3.1.2 ループ展開係数: UX

次に, SYMV カーネルは行列 A の 1 列ずつをスレッドブロックに割り当てるのではなく複数列 (UX 本) ずつスレッドブロックに割り当てていく. これは, ベクトル x のアクセス回数の削減と排他制御対象の w へのアクセス回数を減らす目的がある. UX={8, 9, ..., 32} の 25 通りが候補である.

3.2 多重処理係数: MULTIPLICITY

さらに, 1 マルチプロセッサに同時に起動できるスレッドブロック数 MULTIPLICITY を本実装では制御変数としている. GPU は実行時に共有メモリとレジスタの使用状況を見て実行可能なスレッドブロック数を制御しているようであるが, これをカーネル側から制御する. 実際には, スレッドブロック数を固定するような API は提供されていないため, 共有メモリ上の配列サイズを動的に変更する形の実装でこの制御を行う. Kepler コアは最大 16 まで可能であるが, Fermi コアとの整合性等とを考慮して MULTIPLICITY={1, 2, ..., 8} とする.

3.3 順序変更: MX

最後に, これがなぜ性能に影響を与えているかの解析が必要であるが, 演算順序に関する変更を定めるパラメタ MX がある. 一般的に

```
a0 = ak0[Lda*(0)];
a1 = ak0[Lda*(1)];
a2 = ak0[Lda*(2)];
a3 = ak0[Lda*(3)];
......
```

といったストライドのあるアクセスが各スレッドから発行される例を考える. これはスレッドに割り当てられた複数列を処理する際に必要なアクセスであり, 所謂マルチストリームである. この様なアクセスパターンを ak0[Lda*(0)], ak0[Lda*(1)], ak0[Lda*(2)], ak0[Lda*(3)], ... ではなく ak0[Lda*(0)], ak0[Lda*(2)], ..., ak0[Lda*(1)], ak0[Lda*(3)], ... などのように順序を変更する. この変更が後述の性能実測で大きく影響を及ぼすことになる. MX のパターンはストリームの個数である UX の階乗通り存在するが, あまりにも探索空間が広大となってしまうので 10 通りのパターンをあらかじめ作っておくこととする.

3.4 パラメータ間の制約

探索すべき総パラメタ空間の個数は $8 \times 25 \times 8 \times 10 = 16,000$ 個となる. しかしながら, MULTIPLICITY は共有メモリとレジスタの使用量で制約を受けるため, 全ての組み合わせが有効ではない.

$$\mathtt{UX} + \max\left\{\frac{16\mathrm{K}}{\mathtt{MULTIPLICITY} * 8},\ (\mathrm{UX})^2\right\} < 16\mathrm{K} \quad (3)$$

のときは共有メモリを使い果たさない. すなわち, 上条件を満足しない組み合わせ, さらに, システムが使用する共有メモリ (不確定) の影響のために上式では判別できない組み合わせがあるため, 実行時のメモリの利用状況や計算結果を CUBLAS などと比較しながら篩い落とさなくてはならない.

実際 Tesla K20 で実行したところ, 有効なパラメータ数は 7,276 個に限られ, 探索候補 16,000 の半分以下に減ることがわかっている.

3.5 チャンピオン方式パラメータ篩い分け

探索すべきパラメータ総数が約 7 千個であるが, 実際にはカーネルごとに行列サイズがパラメータとして追加されている. 従って, 代表的な行列の次元を実行するだけに限定しても各カーネルコアの実行には数秒を要する. さらに, パ

表 2　SYMV カーネルのチューニングパラメタ候補
Table 2　Parameter candidates for the SYMV kernel

変数名	探索候補	総数
BLOCK_SIZE	{ 32, 64, . . . , 256 }	8
UX	{ 8, 9, . . . , 32 }	25
MULTIPLICITY	{ 1, 2, . . . , 8}	8
MX	{ 0, 1, . . . , 9 }	10

ラメータが変化した際にカーネルを再コンパイルしなければならない場合もある. スレッドブロックサイズなどループ最適化に影響を与えるものは確実にコンパイルが必要である. 行列の次元も含めた総当たり (ブルートフォース) 探索は事実上難しく, 代表的な行列次元における測定から上位のカーネルを選別してから詳細なデータ測定をもとに内挿等により低コストで性能を予測することが望ましい.

以下にチャンピオン方式のパラメータ篩分けの方式をまとめておく.

(1) 実行可能なパラメータ空間を選定する.
(2) 実行可能なパラメータに対して代表的な次元の計算を行う.
(3) 各次元の計算時間を基に順位をつけ, ポイントを配分する.
(4) 配分ポイントの上位からカーネルを再度選ぶ.
(5) 選ばれた上位のカーネルに対して再度詳細な計算を実施する.
(6) 詳細計算から得られた情報から, データフィッティングの手法により未計算箇所の性能推定を行う.
(7) 各次元毎に性能推定データの中から最良の性能を与えるカーネルを決定する.
(8) 実行時に指定された次元で最良のカーネル関数を実行できるようにデータを記録しておく.

ポイントの配分やデータフィッティングの選択方法に自由度がある.

3.6 d-spline

d-spline は田中 [10] らにより提唱された離散型データ列に対するスプライン近似手法である. 詳細は田中らの原著に委ねるが, フィッティング結果のデータ列 $f = (f_1, f_2, \ldots, f_N)^T$ に対して, 標本点データ列 $y = (y_1, y_2, \ldots, y_k)^T$, 標本データの位置データ $s = (s_1, s_2, \ldots, s_k)^T$ から次のコストを最小化する f を決定するものである.

$$\|y - Ef\|^2 + \alpha^2 \|Df\|^2 \tag{4}$$

ここで, E は $E_{i,s(i)}$ を 1 としてそれ以外が 0 の行列. D は 2 階差分演算から由来する行列であり係数 α により滑らかさを調整する.

実際少数の標本データから多数の他データを推定する. 推定結果はベクトルとして出力される. 上位ランクのカーネルに対して標本点数を十分にとった上で推定を行う.

引き続き上位ランクのカーネルの d-spline による推定値から各次元における最良パラメータ (カーネル) を決定する. この情報もベクトルとして格納されるが, d-spline で推定されるデータは滑らかであるため, 通常は区間で最良のカーネルが決まる. したがって, 最終的には if 文等でカーネルを決定する関数の形で出力をする.

d-spline は逐次添加型のサンプリング追加をすることでサンプリングコストを削減することができる. 本研究ではまず単純化のためサンプリング数を固定して精度向上のためのサンプリング点追加操作は行わないこととする.

4. 自動チューニング結果とその結果の解析

4.1 サンプリング

まず, データ数を少なくし全パラメータ候補でのサンプリングに要する時間に関して表 3 にまとめた.

サンプリングすべき有効パラメータ数はコアによって変動するため総サンプリング時間が大きく異なる結果となった. 特に Tesla C2050 は CUDA4.0 と古い環境であり, CUDA5 に比べてカーネル候補をコンパイル時間に倍以上の時間を浪費する. 24 時間以内にサンプリングがなされており, 何とか許容範囲の中に入っているといえよう. ブルートフォースサンプリングではこれ以上のパラメタ数増加には対応が難しいので, d-spline の逐次添加型のサンプリング等によりサンプリングコストを削減するべきであろう.

4.2 チャンピオン方式による上位カーネルの選抜結果

今回のチューニングではチャンピオン方式で上位ランクのカーネルに配分するポイントを上位 25 位までに 'rank/25' で定まるポイントを付与し, 各カーネルごとに集計したカーネルのポイントの上位 10 を選定した (図 4).

この結果からわかることは, GPU コアによって完全に最適なカーネルが異なるということである. つまり, GPU コアが刷新されるたびに最良パラメータを選択し直していかなくてはならない. 自動チューニングの必要性を再確認する計算結果である.

次に, 上位ランキングに入賞したカーネルでパラメタ MX が 0 でないものが多数存在している. これは複数のストリームアクセスがあった時にプログラム上で順序を変更した方が性能が上がる可能性を示している. Tesla C2050 では 0 のみが選ばれているが, MX が 0 以外でも高い性能を示している.

4.3 自動チューニングされた関数の性能

今回の自動チューニングでは d-spline 計算に標本点を 100 から 10000 次元までの間から 48, 評価点を 9901 点とっている. d-spline 計算に要する時間は数秒であり, 前段のサンプリングに比較すれば無視できる程度である.

表 3　サンプリング時間内訳
Table 3　Dsitribution of sampling time

	Tesla C2050	GeForce GTX580	Tesla K20
総サンプリング数	6,646	7,276	7,200
総サンプリング時間	23:09	9:59	10:37
平均実行時間/サンプル	6 秒	5 秒	5 秒

図 2 が Top10 の第二段階目の詳細サンプリング一例 (Tesla C2050) であり, この情報から得た d-spline を基に推定される top10 カーネルの切り替えるべきポイント (次元) の情報を表にまとめている. 自動チューニングされた SYMV ではこの情報から適切なカーネルを選択して実行する. 図 3, 4, 5 に d-spline で算出された top10 カーネルの性能と, 自動チューニングされた SYMV の実測性能を併せて示している.

d-spline の特性でもあるが, 上に凸な曲線は曲率が抑えられるように近似されるため, 実測値よりも若干下回る数値を予測する. これは過大評価ではないので, 高性能化を期待した予測では十分であろう. また, d-spline のもう一つの特性として 1 点ノイズの影響を避けるために, 曲線が細かく揺れることがある. この現象は本来逐次添加型の操作によってより詳細な近似をする必要があるものを省いたことに起因する. 今後の研究では d-spline の簡易的な利用から逐次添加方式を行う適切な利用にシフトしていく必要がある.

最後に, 自動チューニングされた SYMV カーネルは上位カーネルの性能をほぼ上回っており性能チューニングはなされている. ただし, Fermi コア世代ではあまり顕著ではないが, 特定の次元においてスパイクが確認できる. ストリームのアクセス順序を変更することが性能向上につながることから, キャッシュの他に TLB の競合などに注目したチューニングも今後考慮しなくてはならない.

グラフ中で, 自動チューニングされた SYMV カーネルは小さい行列次元では Ranking したカーネルよりも性能が高い. これは L+U アルゴリズムが選択された結果であり. L+U アルゴリズムも同様に d-spline を用いて性能推定し, カーネル選択させている.

4.4　世代間のパラメータの違い

チューニング結果から最良パラメータは GPU コアアーキテクチャによって大きく異なることがわかる. GPU コア世代では `BLOCK_SIZE` と `UX` に似通った傾向がある. Kepler コアは Fermi コアに比べて CUDA コア数が 3 倍になっているために, `BLOCK_SIZE` を増やす必要があるのであろう. これらの最良値をカタログスペックや経験則から導くのは困難である. 現状は 24 時間程度かかってしまうが, 最適化された CUDA-BLAS ライブラリの提供の方法論としてはよきものと判断できる.

5. まとめ

SYMV カーネルの性能自動チューニングを異なる二つの GPU コアアーキテクチャに対して実施した. GPU コアに応じた最適なパラメータが存在し, 世代間の差は大きいことが分かった. GPU には自動チューニングが必須であることを再認識させるとともに, 新世代 GPU コア出現のたびに新たなチューニング技法が存在することも判明した. 今後はチューニングプロセスの大部分を占めているサンプリング操作のコスト削減を進めていく必要がある.

本研究は科研費基盤 A (課題番号：23240005) 並びに新学術領域研究 (課題番号：22104003) の支援を受けている.

参考文献

[1] NVIDIA: whitepaper NVIDIA’s Next Generation CUDA Compute Architecture: Fermi. `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf`
[2] NVIDIA : CUDA CUBLAS Library. `http://developer.download.nvidia.com`
[3] Agullo, E., Demmel, J., et al. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects, J. of Physics: Conference Series 180 (2009)
[4] Humphrey, J. R., Price, D. K., et al., CULA: Hybrid GPU Accelerated Linear Algebra Routines, SPIE Defense and Security Symposium (DSS) (2010)
[5] Sorensen, H. H. B., Auto-tuning Dense Vector and Matrix-Vector Operations for Fermi GPUs, Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, Vol. 7203 (2012) 619–629
[6] GPUlab: GLAS library version 0.0.2, `http://gpulab.imm.dtu.dk/docs/glas_v0.0.2_C2050_cuda 4.0_linux.tar.gz`
[7] 今村俊幸, 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, 情報処理学会論文誌コンピューティングシステム, Vol.4, No.4 (Oct. 2011) 158–168
[8] 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), `http://www.siam.org/meetings/pp12/`
[9] Imamura, T., Yamada, S., Machida, M., A High Performance SYMV Kernel on a Fermi-core GPU, Proc. 10th Intl. Mtg. High-Performance Computing for Computational Science (VECPAR2012) (2012)
[10] 田中 輝雄, 数値計算ライブラリを対象としたソフトウェア自動チューニングにおける性能パラメタ推定法に関する研究, 電気通信大学博士学位論文 (2008)

表 4　上位ランクカーネルコアリスト (BLOCK_SIZE, UX, MULTIPLICITY, MX)
Table 4　The list of Top10 kernel cores (BLOCK_SIZE, UX, MULTIPLICITY, MX)

	Tesla C2050	GeForce GTX580	Tesla K20
Rank 1	(128, 15, 5, 0)	(128, 14, 6, 9)	(160, 14, 5, 7)
Rank 2	(128, 16, 5, 0)	(128, 14, 5, 9)	(160, 13, 5, 2)
Rank 3	(96, 15, 6, 0)	(128, 14, 6, 1)	(128, 14, 6, 6)
Rank 4	(96, 16, 5, 0)	(160, 14, 4, 9)	(160, 13, 6, 2)
Rank 5	(96, 20, 3, 0)	(128, 19, 4, 5)	(96, 16, 1, 8)
Rank 6	(128, 17, 4, 0)	(96, 20, 3, 4)	(96, 14, 7, 6)
Rank 7	(128, 14, 5, 0)	(96, 19, 4, 2)	(192, 24, 1, 6)
Rank 8	(128, 14, 6, 0)	(96, 17, 1, 0)	(160, 13, 5, 3)
Rank 9	(160, 15, 4, 0)	(128, 14, 5, 1)	(160, 13, 6, 3)
Rank 10	(160, 15, 5, 0)	(160, 18, 4, 5)	(128, 12, 6, 6)

図 2　Tesla C2050 での Top 10 カーネルのサンプリング結果 (Rank 0 は L+U アルゴリズム)
Fig. 2　Sampling results of the Top 10 kernels on a Tesla C2050 ('Rank 0' refers to the L+U algorithm)

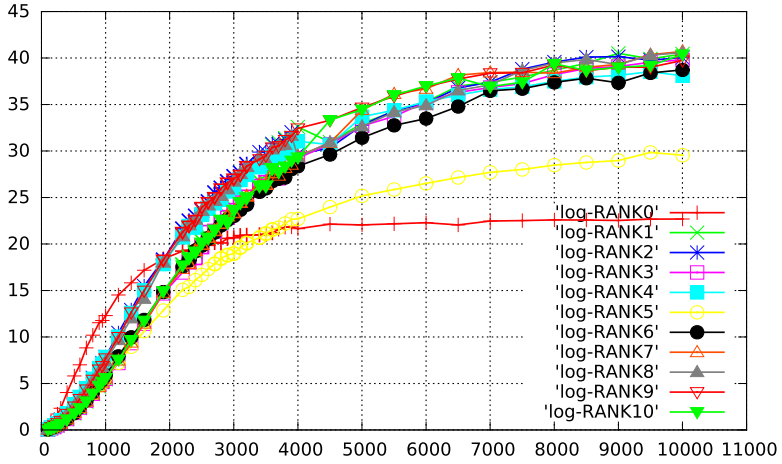

表 5　生成された最良カーネル選択ルール (Rank 0 は L+U アルゴリズム)
Table 5　Generated rule to determine the optimal kernel ('Rank 0' refers to the L+U algorithm)

Tesla C2050	GeForce GTX580	Tesla K20
Rank 0 ($100 \leq N < 2020$)	Rank 0 ($100 \leq N < 2070$)	Rank 0 ($100 \leq N < 2336$)
Rank 2 ($2020 \leq N < 3574$)	Rank 5 ($2070 \leq N < 2417$)	Rank 1 ($2336 \leq N$)
Rank 9 ($3574 \leq N < 5245$)	Rank 10 ($2417 \leq N < 3179$)	
Rank 10 ($5245 \leq N < 5835$)	Rank 4 ($3179 \leq N < 4716$)	
Rank 7 ($5835 \leq N < 7187$)	Rank 2 ($4716 \leq N < 6748$)	
Rank 9 ($7187 \leq N < 7441$)	Rank 1 ($6748 \leq N < 8059$)	
Rank 2 ($7441 \leq N < 9114$)	Rank 2 ($8059 \leq N < 8060$)	
Rank 1 ($9114 \leq N < 9625$)	Rank 1 ($8060 \leq N < 8061$)	
Rank 7 ($9625 \leq N$)	Rank 2 ($8061 \leq N$)	

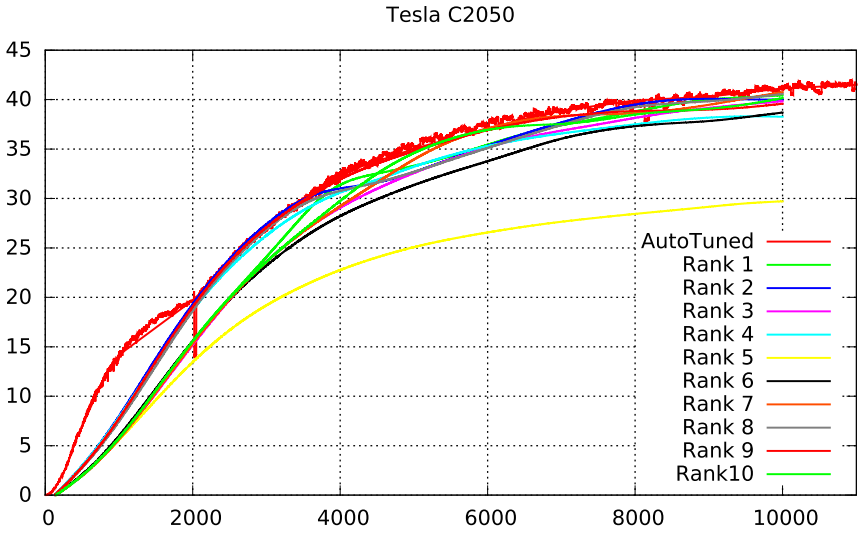

図 3 Tesla C2050

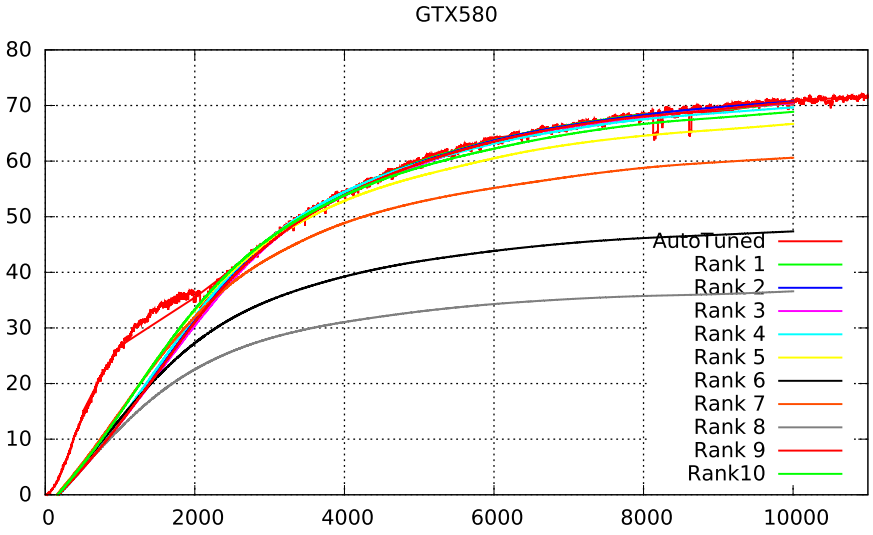

図 4 GeForce GTX580

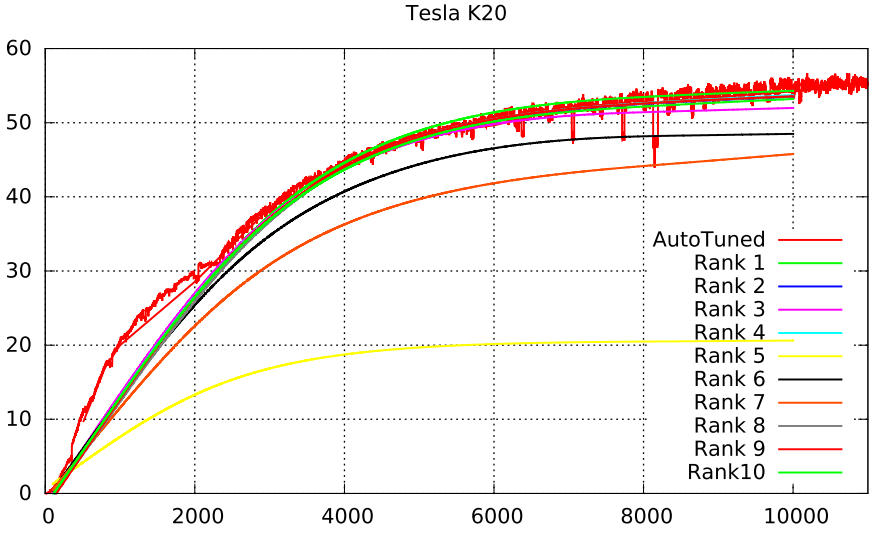

図 5 Tesla K20