C-023

プロセッサ自動選択機能を有するBLASの実現に向けた性能評価

Performance Evaluation towards BLAS with Automatic Processor Selection

小松一彦* 小山賢太郎† 佐藤功人† 滝沢寛之† 小林広明*†
Kazuhiko Komatsu Kentaro Koyama† Katsuto Sato† Hiroyuki Takizawa† Hiroaki Kobayashi*†

1 緒言

近年の著しい半導体技術の向上により，現在，汎用プロセッサ (Central Processing Unit, CPU) だけでなく特定の演算に特化したプロセッサなど様々な特徴を持つ高性能なプロセッサが広く普及している．例えば，演算コアを複数搭載している汎用プロセッサや，描画処理に用いられるメニーコアプロセッサである GPU(Graphics Processing Unit)，ストリーミング処理に特化したヘテロジニアスプロセッサである Cell プロセッサ [1] などが挙げられる．次世代のプロセッサでは，異なる演算コアを複数搭載し，汎用演算だけでなく特定の演算コアに特化した演算も 1 つのプロセッサ上で行う事ができる．

特徴の異なる複数のプロセッサを搭載する複合型計算システムでは，それぞれのプロセッサの長所と短所が異なるため，演算の種類やデータサイズなど演算の特性に応じて，実行に適切なプロセッサが異なる可能性がある．そのため，複合型計算システムの性能を最大限に発揮するためには，それぞれのプロセッサの特徴を考慮した処理の割り当てが重要となる．CPU は様々な処理を実行する事が可能であるが，GPU は描画処理に，Cell プロセッサは演算コアを利用したストリーミング処理に，それぞれ特化したプロセッサであるため，その性能を十分に発揮できる処理は限られている．それらのプロセッサで特化した処理以外の処理を実行する場合，実効演算性能が著しく低下してしまい，複合型計算システム全体の性能低下を招く可能性がある．

本研究では，複合型計算システムにおいて処理に応じて適切なプロセッサを自動で選択する手法の確立を目的としている．本報告では，線形代数演算ライブラリである BLAS (Basic Linear Algebra Subprograms)[2][3][4] を対象とし，各プロセッサの得手不得手が BLAS 関数実行時の性能に与える影響を調査する．その性能評価に基づいて，自動的に適切なプロセッサを選択するための指針を示し，プロセッサ自動選択機能付き BLAS ライブラリの実現可能性を明らかにする．

2 BLAS の概要

科学技術計算などの大規模な高性能計算でよく利用される最も代表的な線形代数演算が，BLAS と呼ばれる線形代数ライブラリとしてまとめられており，広く利用されている．高性能計算の最も基本となる BLAS には，高度に最適化された実装が様々なベンダーや研究者によって提供されており [2][3][4]，その BLAS の関数群を利用した LAPACK[5] などの上位ライブラリも提供されている．BLAS に含まれる各関数には，様々な最適化が適用されており，その結果として高速な線形代数演算が実現されている．例えば，プロセッサの演算性能やそのキャッシュメモリの特性などを考慮し，適切なサイズでブロッキングを行う事で高速化を図っている．

BLAS では，演算対象の組み合わせによって各関数が 3 段階のレベルに分類されている．スカラとベクトル単体の演算，もしくはベクトル同士の演算を行う関数群が BLAS レベル 1 に分類されている．BLAS レベル 2 にはベクトルと行列の演算を行う関数群が，BLAS レベル 3 には行列同士の演算を行う関数群が分類されている．

2.1 BLAS レベル 1

BLAS レベル 1 の関数群では，スカラとベクトル単体の演算，もしくはベクトル同士の演算を行う．BLAS レベル 1 ではデータの再利用性はなく，演算量が他のレベルに比べて少ないという特徴がある．BLAS レベル 1 の主な関数として，スカラの乗算 (*scal*)，内積演算 (*dot*)，乗算と加算 (*axpy*)，コピー (*copy*)，絶対値が最大の要素の検出 (*iamax*) などが挙げられる．

BLAS レベル 1 の関数には，マップ型とリダクション型という 2 種類の性質の異なる演算形式がある．表 1 に BLAS レベル 1 のそれぞれの関数が属している演算形式を示す．

マップ型の演算形式とは，複数の入力データの各要素から対応する出力データを算出する演算形式である．例えば，*scal* 関数は，スカラとベクトルを入力として，ベクトルの各要素とスカラの積を出力する．それぞれのベクトル要素に対して完全に独立して演算ができるため，マップ型の演算形式では効率的な並列処理を行う事ができる．

リダクション型の演算形式とは，複数の入力データに対してひとつの出力データがある演算形式である．例えば，*iamax* 関数は，ベクトルの各要素から絶対値が最大の要素を求める関数であり，ベクトルの要素間で比較を行い，絶対値が最大の要素を出力する．逐次処理が必要となり演算の全てを並列に処理する事ができないため，リダクション型の演算形式では並列化効率が低下してしまう可能性がある．

2.2 BLAS レベル 2

BLAS レベル 2 の関数群では，ベクトルと行列の演算を行う．BLAS レベル 2 ではベクトルデータのみ再利用性があり，演算量は比較的多いという特徴がある．

表 1 BLAS レベル 1 の演算形式

演算形式	BLAS 関数
マップ型	*scal*，*axpy*，*copy*
リダクション型	*dot*，*iamax*

* 東北大学サイバーサイエンスセンター
† 東北大学大学院情報科学研究科

BLAS レベル 2 の主な関数である *ger* は，一般行列のランク 1 更新を行う関数である．*gemv* は，一般行列とベクトルの積を求める関数であり，1 つの行列と 1 つのベクトルを入力として，その積を出力する．BLAS レベル 2 の各関数内では，下位レベルの関数である BLAS レベル 1 の関数が利用されている．また，三角行列とベクトルの積を求める関数である *trmv*，対称行列とベクトルの積を求める関数である *symv* など，特殊な行列における演算を行う関数も用意されている．

2.3 BLAS レベル 3

BLAS レベル 3 の関数群では行列同士の演算を行う．BLAS レベル 3 ではデータの再利用性は高く，演算量が極めて多いという特徴がある．

BLAS レベル 3 の主な関数である *gemm* は，2 つの一般行列の積を求める関数で，2 つの行列を入力として，行列積の結果を出力する．BLAS レベル 2 と同様に，関数内部で下位レベルの BLAS を呼び出す事で演算を行う．また，三角行列の行列方程式の解を求める関数である *trsm*，対称行列と一般行列の積を求める関数である *symm* など，特殊な行列における演算を行う関数も用意されている．

3 各プロセッサに最適化された BLAS ライブラリ

本報告では，CPU に対して最適化を施した GotoBLAS[3], NVIDIA の CUDA (Compute Unified Device Architecture) [6] に対応した GPU に提供されている CUBLAS[7]，そして Cell プロセッサに対して最適化を施した Cell 用 BLAS[8] の 3 種類の BLAS を対象として，それらの性能や特性を議論する．

3.1 GotoBLAS

GotoBLAS は後藤が開発した CPU 用の BLAS であり，様々な CPU 向けに高度な最適化が施されている [3]．GotoBLAS の特徴として，幅広い CPU アーキテクチャに対応している事が挙げられる．GotoBLAS は古い CPU から最新の CPU まで対応しており，いずれの CPU においてもその性能を十分に発揮する事が可能である．更に，新しい CPU への対応が早く，SIMD 演算命令によるデータ並列処理や複数コアを利用したマルチスレッド処理による高速な線形代数演算も可能である．そのため，GotoBLAS は，汎用 PC での演算だけでなく，IBM BlueGene などのスーパコンピュータにも採用されている．

本研究では，この GotoBLAS を CPU 用の BLAS として採用し，その性能を測定し CPU を用いて線形代数演算を行う場合の特性を調査する．

3.2 CUBLAS

CUBLAS[7] は，NVIDA が提供している CUDA 上で動作する BLAS ライブラリである．CUBLAS を利用する事で，GPU の高い演算性能を効率的に引き出す事ができる．

GPU は従来，描画処理専用ハードウェアとして発展してきたが，近年，その高い演算能力が描画処理以外にも利用されている．NVIDA は GPU 上で描画以外の処理を実現するためのプラットフォームとして CUDA を提供しており，CUDA に対応した GPU を用いる事で描画以外の処理をより容易に行う事ができる．CUDA において，GPU は図 1 に示すように抽象化される．GPU はいくつかの Stream Multi-processor(SM) から構成され，各 SM は基本的な演算を行う 8 つの Strem Processor(SP) から構成されている．SP は低速だが大容量の Global memory と，小容量だが高速の Shared memory を共有している．

CUDA において高い演算性能を引き出すためには，高速な Shared memory を効率的に利用してデータ供給を行うと同時に，各 SP 上で多数のスレッドを実行して並列処理を行う必要がある．そのため，実行される CUBLAS の関数の並列性が高い場合には，処理を分割し各 SP に割り当てる事で GPU による高速な線形代数演算を期待できる．しかし，実行される CUBLAS の関数の並列性が乏しい場合，十分な数のスレッドを各 SP に割り当てる事ができないため，GPU の高い演算性能を引き出す事ができない．

また，GPU の高い実効性能を引き出すためには，統合メモリアクセス (coalesced memory access) と呼ばれる効率的なメモリアクセスを行い，データ転送を行う事が重要となる．CUDA では，データ転送を高速に行うために，16 スレッドを 1 つにまとめて，メモリにアクセスを行う．このメモリアクセスを行う際に，先頭アドレスが 64 バイト境界にアラインされており，かつ，16 スレッドが連続するメモリ領域にアクセスする場合，効率的にデータを転送する事ができる．逆に，メモリアクセスを行う先頭アドレスが 64 バイト境界にアラインされていない場合などには，効率的なデータ転送が実現できず，実効メモリバンド幅が低下し，実効性能に影響が出る．

このように，実行される CUBLAS の関数に内在する並列性の有無や，効率的なデータ転送ができるか否かによって，GPU の実効演算性能は大きく変化する．演算対象となる行列やベクトルのサイズに応じて，並列実行可能なスレッドの数は変化するため，それらのサイズを変化させて CUBLAS の各関数の性能を評価する事により，GPU を用いた線形代数演算を行う場合の特性を調査する．

3.3 Cell プロセッサ用 BLAS

Cell プロセッサは，図 2 に示されるように，制御用プロセッサコアである PPE(PowerPC Processor Element)1 基と演算用プロセッサコアである SPE(Synergistic Processor Element)8 基から構成されるヘテロジニアス・マルチコアプロセッサである．PPE は 64 ビット PowerPC アーキテクチャをベースにした汎用プロセッサコアであり，主にメインメモリ上のデータ処理や外部への入出力制御，SPE の制御などの処理を行う．SPE は比較的単純な処理を大量のデータに適用する，マルチメディア処理に特化した演算コアである．3.2GHz で動作時に，8 基の SPE の単精度浮動小数点演算性能は合計で 204.8Gflop/s に達する．

SPE は専用のメモリである LS(Local Store) を搭載しており，その容量は 256KB である．SPE はメインメモリ上のデータを直接参照できないため，DMA(Direct Memory Access) 転送命令によってデータをメインメモリから LS へ転送し，LS 上のデータを参照する．DMA 転送とは，プロセッサを介さずに各装置とメインメモリの間で直接行われるデータ転送であ

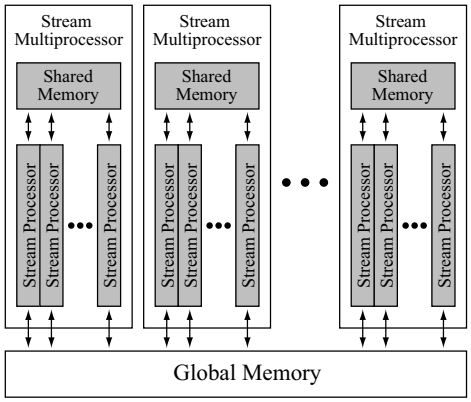

図 1　CUDA における GPU アーキテクチャの概要

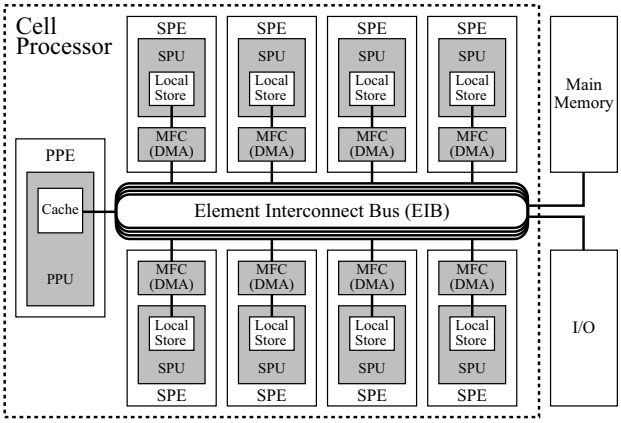

図 2　Cell プロセッサのアーキテクチャの概要

る．Cell プロセッサではメインメモリとのメモリバンド幅は 25.6GB/s である．

Cell プロセッサを用いた効率的な線形代数演算を実現するために, Cell プロセッサ用の BLAS ライブラリ (以下, CellBLAS) が CellSDK の一部として提供されている [8]．CellBLAS では，SPE を効率的に利用するための最適化が施されている．

Cell プロセッサの演算性能を最大限に引き出すためには，全ての SPE を効率よく稼働させ並列処理を行う必要がある．SPE が遊休状態にならないように常にデータを供給し続けるためには，容量が限られている LS の有効活用が必要である．また，各 SPE においても SIMD 命令によるデータ並列処理が必要となる．そのため，データの再利用性が高く，データ量に対して並列実行可能な演算回数が多い BLAS 関数の場合，SPE の稼働率を高める事ができるため，Cell プロセッサによる高速な線形代数演算を期待できる．一方，データの再利用性が低く演算回数が少ない BLAS の関数の場合，データ転送がボトルネックになるため，Cell プロセッサの高い演算性能を活用できない．また，データ並列性が乏しい BLAS 関数の場合にも，効率的なデータ並列処理を行えず，演算性能が低下する．

このように，並列性の有無やデータの再利用性，データのサイズによって Cell プロセッサの各 BLAS 関数における実効演算性能は大きく変化する可能性がある．そのため，演算対象となる行列やベクトルのサイズを変化させて，CellBLAS の各関数の性能を評価する事により，Cell プロセッサの線形代数演算実行時の特性を調査する．

4　性能評価

4.1　性能指標

本報告では，GotoBLAS, CUBLAS, および CellBLAS の 3 種類の BLAS ライブラリを用いて，行列やベクトルのサイズを変化させながら各プロセッサの性能を計測する．ここで，BLAS における 1 つの関数の処理が完了するまでに行われる浮動小数点演算の回数を，**演算回数**と定義する．

レベル 1 において，ベクトルの要素数を N とすると，演算回数は $O(N)$ である．レベル 2 において，行列の列サイズを M，行サイズを N とすると，演算回数は $O(M \times N)$ となる．レベル 3 において，演算する行列のサイズ $M \times K$，$K \times N$ とすると，演算回数は $O(M \times N \times K)$ となる．この演算回数と BLAS 関数の実行に要する時間との関係から，各プロセッサの BLAS 関数実行時の特性を評価する．

4.2　評価対象

GPU や Cell プロセッサは単精度浮動小数点演算に対して高い理論性能を有しているため，本報告においても，単精度浮動小数点演算を行う BLAS の関数を評価対象とする．

BLAS レベル 1 の関数ではマップ型の関数である *sscal* とリダンクション型の関数 *isamax* を，BLAS レベル 2 の関数では *sger*，BLAS レベル 3 の関数では *sgemm* と *strsm* を評価対象とする．

4.3　評価条件

BLAS の性能評価には，4 種類のプロセッサを用いて実験を行う．CPU として，2.4GHz で動作する Intel Core 2 Quad Q6600(以下, C2Q) を用いる．GPU として, NVIDIA GeForce 8800 GTX(以下, 8800GTX) と NVIDA GeForce GTX 280(以下，GTX280) を用いる．また，Cell プロセッサを搭載するシステムとして，IBM BladeCenter QS20(以下，QS20) を用いる．QS20 は Cell プロセッサを 2 基搭載し，最大で合計 16 基の SPE を同時に利用する事が可能である．表 2 に, 各プロセッサにおける演算コア数，その動作周波数，理論性能，メモリ容量，バンド幅をそれぞれ示す．

CPU，GPU の性能評価に用いた PC には，OS として Linux(kernel 2.6.18) が導入されており，コンパイラとして gcc 4.1.2 が利用可能である．また，NVIDA グラフィックスドライバのバージョンは 180.44 で，CUDA SDK 2.1 が導入

表 2　実験で用いる各プロセッサの仕様

プロセッサ	C2Q	8800GTX	GTX280	QS20
演算コア数	4	128	240	(1+8)×2
動作周波数 (GHz)	2.4	1.35	1.296	3.2
理論性能 (Gflops)	76.8	518	933	460.8
メモリ容量 (MB)	4096	768	1024	1024
バンド幅 (GB/s)	12.8	86.4	141.7	25.6

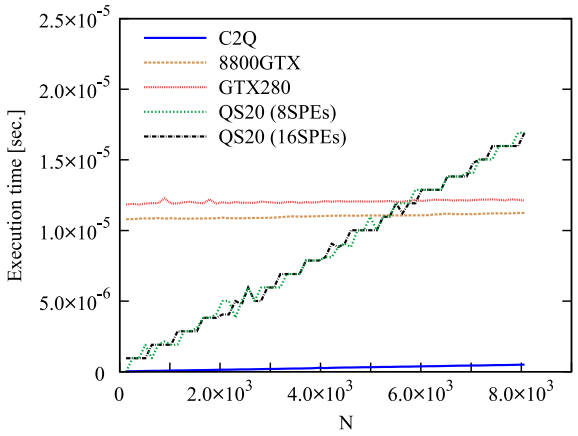

図 3　*sscal* における演算回数と実行時間

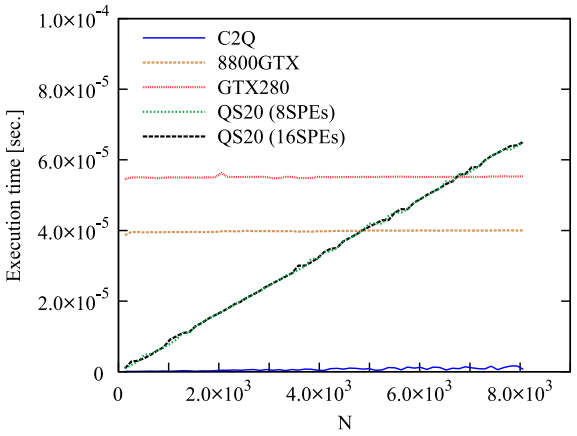

図 4　*isamax* における演算回数と実行時間

されている．また，Cell プロセッサの性能評価に用いた QS20 には，Linux(kernel 2.6.27) が導入されており，コンパイラとして IBM XL C/C++ for Multicore Acceleration for Linux V10.1 が利用可能である．

4.4　BLAS レベル 1 の関数における特性

BLAS レベル 1 の関数において，マップ型の関数である *sscal*，リダクション型の関数である *isamax* の演算性能を評価した結果を，それぞれ図 3，図 4 に示す．横軸に演算回数，縦軸に実行時間を示す．

図 3 と図 4 の結果から，いずれのプロセッサにおいても，演算回数に対して実行時間が線形に増加している事が分かる．QS20 では他のプロセッサよりも傾きが大きい事も分かる．また，図 3 と図 4 のいずれの場合も，C2Q の実行時間が他のプロセッサと比較して短い．BLAS レベル 1 の関数では，マップ型とリダクション型のいずれの場合においても，メモリへのアクセス数に対して演算回数が少ないため，メモリへのアクセスレイテンシを隠蔽しにくい．そのため，メモリへのアクセスレイテンシが長い GPU や LS へデータ転送が必要な Cell プロセッサに比べ，レイテンシが短いキャッシュを備えている CPU での実行時間が短くなる．

QS20 による実行時間を見ると，図 3 と図 4 のいずれの場合も，演算回数に対して階段状に実行時間が増加している．これは，CellBLAS において，行列の行と列が 64 の倍数の時に性能が高くなるように最適化されているためである [8]．64 の倍数を超えた直後に，実効性能の明らかな低下が見られる．また，8 基の SPE を用いる場合と 16 基の SPE を用いる場合とで，実行時間の差がほとんど見られないのが分かる．演算回数が少ないレベル 1 では，データ転送がボトルネックになり，SPE の台数を効率的に利用できないためである．

以上の結果より，CPU，GPU および Cell プロセッサを混載する複合型計算システムにおいては，ベクトルのサイズや BLAS の演算形式に依らず，CPU を用いて BLAS レベル 1 の関数を実行すべきである事が分かる．また，GPU と Cell プロセッサを比較すると，演算回数が非常に少ない場合には Cell プロセッサの実行時間が短く，演算回数が多くなると GPU の方が実行時間が短くなる事が分かる．このため，演算回数に応じて演算を行うプロセッサが切り替える必要がある．GPU と Cell プロセッサの実行時間はほぼ線形であるため，最小自乗法による直線近似で，演算回数に応じて適切なプロセッサを選択できる事が期待される．

4.5　BLAS レベル 2 の関数における特性

図 5 は，一般行列のランク 1 更新を行う BLAS レベル 2 の関数である *sger* に対する性能を評価した結果である．横軸に演算回数，縦軸に実行時間を示す．

図 5 を見ると，全てのプロセッサにおいて，演算回数の増加に伴い実行時間がほぼ線形に増加している．特に，GTX280 の実行時間は演算回数に対して線形になっている．

C2Q の実行時間を見ると，演算回数が少ない場合においては直線の傾きが小さく，演算回数がある回数を超えると傾きが大きくなる事が分かる．これはベクトルと行列のデータが 2 次キャッシュに収まる場合，より高速な演算が可能であるため，2 次キャッシュに入りきらない場合に比べ，実効時間が短くなり傾きが小さくなるためである．

8800GTX の場合，行列の行サイズが 16 の倍数，8 の奇数倍，4 の奇数倍，2 の奇数倍の時に実行時間が短くなる．これは，特定の行サイズの場合に，統合メモリアクセスが行われ実効メモリバンド幅が高くなり，高い演算性能が引き出せたためである．*sger* では，行列の行の要素とベクトルの要素との積をそれぞれ求め，その積の総和をとる事で最終的な結果を算出する．この総和をとる時にビデオメモリ上にある積の結果にアクセスし，SP へデータ転送する必要がある．CUBLAS では行列が 1 次元配列で列優先でメモリ上に配置されており，行列の行サイズが 16 の倍数の時，データ転送によりアクセスされる要素の先頭アドレスは常に 64 バイト境界にアラインされるため，常に効率的なメモリアクセスが行われる．その結果，実行時間が大幅に短縮される．行列サイズが 4 の奇数倍である時は，メモリアクセス 4 回中 1 回だけ先頭アドレスが 64 バイト境界にアラインされているため，4 回中 1 回のみ統合メモリアクセスを行い，それ以外では，非効率な非統合メモリアクセスになる．このような効率的なメモリアクセスの頻度に変化が生じる事によって，特定の行サイズにおいて実行時間に差が生じる．

図 6 に，メモリアクセスする先頭アドレスのアライメントを

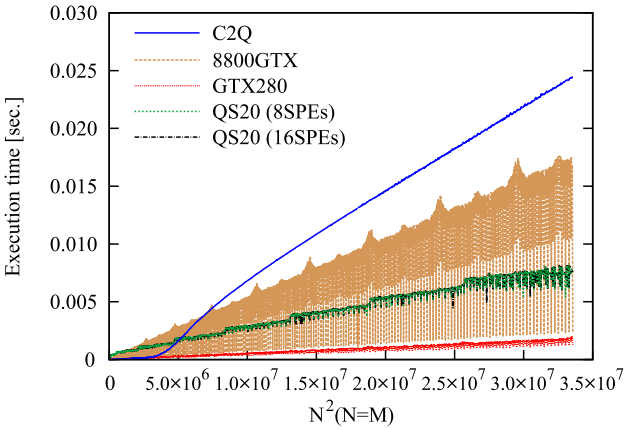

図 5　*sger* における演算回数と実行時間

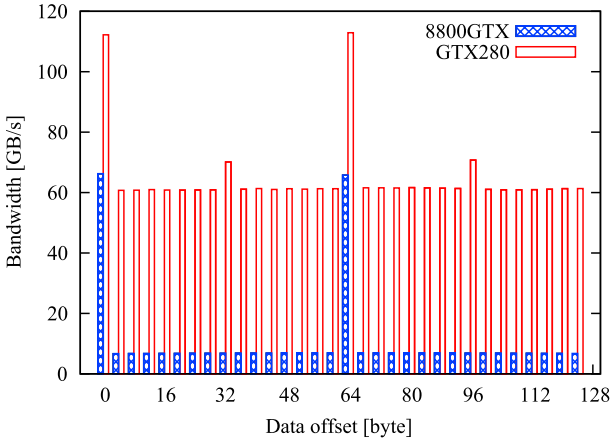

図 6　データアライメントの違いによる実効メモリバンド幅

変更した場合の実効メモリバンド幅の関係を示す．横軸に 64 バイト境界から先頭アドレスへのオフセット，縦軸に実効メモリバンド幅を示す．図 6 に示すように，8800 GTX では統合メモリアクセスと非統合メモリアクセスのバンド幅の差が大きいため，メモリアクセスによる実効メモリバンド幅の影響が大きくなる．一方，GTX280 では統合メモリアクセスと非統合メモリアクセスのバンド幅の差が 8800GTX と比較して小さいため，アラインメントによる性能への影響が小さくなる．

図 5 の結果を見ると，8800GTX では，アラインメントによる実効メモリバンド幅への影響が顕著に表れるため，特定行列サイズにおける実行時間が短くなる．そのため，特定の数の倍数に対応する演算回数とそれ以外の演算回数との実行時間の差が大きくなる．しかし，GTX280 では，アラインメント状態による影響が小さいため，特定演算回数とそれ以外の演算回数との実行時間の差は 8800GTX と比べて小さい事が分かる．

次に，図 5 に示す QS20 による *sger* の実行時間を見ると，実行時間が階段状に増えている事が分かる．これは，BLAS レベル 1 の場合と同様に，64 の倍数を超えた直後に実効性能の低下が引き起こされるためである．また，8 基の SPE の場合と 16 基の場合とで，実行時間の差がほとんど見られない．これも BLAS レベル 1 の場合と同様に，BLAS レベル 2 の関数においても，未だにメモリアクセスに対して演算回数が少ないため，データ転送がボトルネックになり SPE の台数を効率的に利用できないためである．

各プロセッサの実行時間を比較すると，演算回数に応じて演算が速いプロセッサが切り替わっているのが分かる．演算回数が少ない場合では C2Q, GTX280, 8800GTX, QS20 の順に演算が速く，多くなると GTX280, 8800GTX または QS20, C2Q の順になる．これは演算回数が多い場合，効率的なデータ並列処理が求められるため，多数の演算コアを持つ GPU や Cell プロセッサが有利になるためである．また，8800GTX の実効メモリバンド幅や QS20 の理論メモリバンド幅に比べ，GTX280 の実効メモリバンド幅が高いため，より高い演算性能を引き出す事ができた．

以上の結果より，各プロセッサを混載する複合型計算システムにおいては，レベル 2 の BLAS 関数実行時に演算回数に応じて適切なプロセッサを選択する事で実行時間を削減できる可能性が示された．CPU と Cell プロセッサの実行時間はほぼ線形であるため，BLAS レベル 1 の場合と同様に，実行時間を最小自乗法によって線形近似する事によって，演算回数から実行時間を予測する事ができる．また，GPU の場合には，統合メモリアクセスを行う場合と行わない場合を分けて近似する事で，同様に実行時間を予測する事ができる．これにより，複合型計算システムにおいて，演算回数に応じて適切なプロセッサが選択できる事が期待される．

4.6　BLAS レベル 3 の関数における特性

図 7，図 8 に，一般行列同士の積を行う BLAS レベル 3 の関数である *sgemm* の実験結果を，三角行列の行列方程式の求解を行う BLAS レベル 3 の関数である *strsm* の実験結果をそれぞれ示す．横軸に演算回数，縦軸に実行時間を示す．これらの関数における実験では，$M \times K$ と $K \times N$ の行列同士を演算する場合に，常に $N = M = K$ となるような条件下で演算回数を変化させ，実験を行う．

図 7 および図 8 に示されるとおり，レベル 3 の BLAS の関数の実行時間にも他のレベルの関数と同様に，いずれのプロセッサにおいても演算回数に対する線形性を見る事ができる．

8800GTX や GTX280 の実行時間を見ると，BLAS レベル 2 の時と同様に，統合メモリアクセスの効果が見られ，実行時間が短くなる場合がある．図 7 の場合，8800GTX では $M = N = K$ が 64 の倍数，32 の奇数倍，16 の奇数倍の時に，GTX280 では $M = N = K$ が 64 の倍数，32 の奇数倍の時に実行時間が短くなる．図 8 の場合，8800GTX では 32 の倍数，16 の倍数，8 の奇数倍，4 の奇数倍，2 の奇数倍の時に，GTX280 では 32 の倍数の時に実行時間が短くなる．

また，C2Q や QS20 の場合，図 7 や図 8 に見られるように，複数の直線が重なっている．これは，CPU や Cell プロセッサで利用されている SIMD 演算命令の効果である．$M = N = K$ が 4 の倍数の場合に，SIMD 演算命令が効率的に実行され，実行時間が短くなる．一方，それ以外の場合では 4 の倍数の場合と比べ，実行時間が長くなる．BLAS レベル 3 の関数では他のレベルと比べデータの再利用性が高く演算回数が大きいため，データ並列処理による効果がより顕著に表れている．

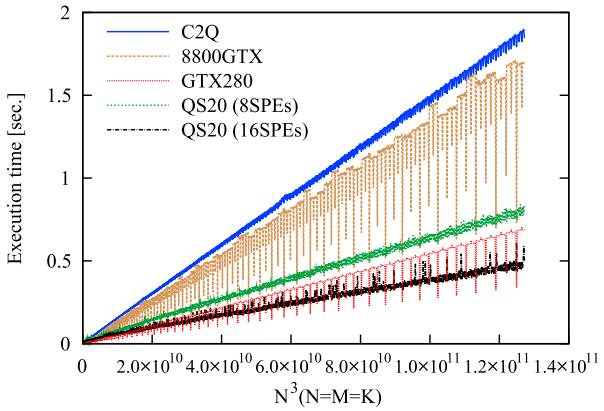

図 7　*sgemm* における演算回数と実行時間

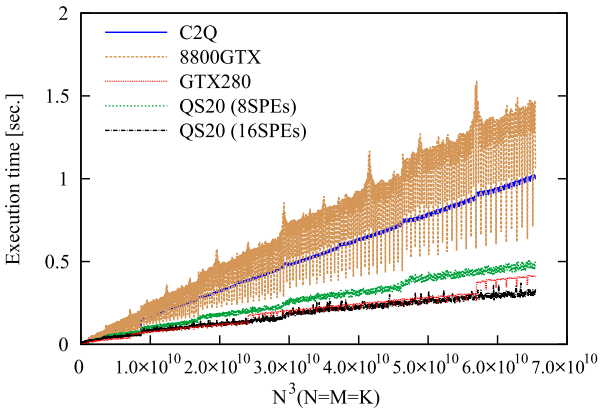

図 8　*strsm* における演算回数と実行時間

QS20 において 8 基の SPE の場合と 16 基の SPE の場合を比較してみると，他のレベルの場合とは異なり，16 基の SPE を用いた方が実行時間が短い．BLAS レベル 3 の関数では演算回数が多いため，演算を効率的に SPE 分割し並列処理を行う事ができている．

各プロセッサの実行時間を比較すると，BLAS レベル 2 の場合と傾向が似ており，演算回数が非常に少ない場合では C2Q の実行時間が短いが，演算回数が多くなると GTX280 や QS20 の実行時間が短くなる．演算回数が多い場合，効率的なデータ並列処理が求められるため，多数の演算コアを持つ GPU や Cell プロセッサが有利になる．特に，Cell プロセッサを搭載する QS20 では 8800GTX や GTX280 よりも実行時間が短い．QS20 では最大で理論性能の 62.5% と高い演算性能を引き出しているのに対し，GTX280 では 39.2% の演算性能しか引き出せていない．これは，BLAS レベル 3 の関数ではアルゴリズム上の工夫も効果的であり，よりプログラミングの柔軟性が高い Cell プロセッサの方が高い演算性能を引き出せるためである．したがって，大規模な線形代数演算においては，GPU に比べ Cell プロセッサの方が実効性能が高くなると考えられる．

以上の結果より，各プロセッサを混載する複合型計算システムにおいては，BLAS レベル 3 の場合においても演算回数に応じて適切なプロセッサを選択する必要ある．BLAS レベル 2 の場合と同様に，各プロセッサの実行時間はほぼ線形で近似できるため，演算回数に基づいて実行時間を予測する事によって，適切なプロセッサを選択できる可能性が示された．

5　結言

本報告では，様々な条件における BLAS の関数の実行時間を測定し，CPU 用 BLAS である GotoBLAS, GPU 用 BLAS である CUBLAS ライブラリ，および Cell プロセッサ用 BLAS と比較，評価を行った．これにより，3 種類の異なるプロセッサの演算特性が異なる事を示した．CPU は演算回数が少ない場合に高い実効性能を示した．GPU や Cell プロセッサは演算回数が多い場合に，高い実効性能を示す事が分かった．

また，CPU や GPU，および Cell プロセッサを混載する複合型計算システムにおいて，その演算性能を最大限に引き出すためには，BLAS のレベルや演算回数に応じて適切なプロセッサを選択する必要がある事が分かった．BLAS レベル 1 の場合は，CPU での実効性能が GPU や Cell プロセッサに比べ高いため，CPU で実行するべきである事が分かった．その他の場合においては，演算回数に対する実行時間を近似し，演算回数から実行時間を予測し，演算性能が高いプロセッサを選択すべきである事が示せた．

今後の課題として，より多くの BLAS の関数を評価し，演算特性の調査を行う事と，その演算特性を利用した複合型計算システムにおける最適なプロセッサの自動選択機構の構築が挙げられる．

参考文献

[1] Flachs B. et al. A streaming processing unit for a cell processor. In *ISSCC '05: IEEE Solid-State Circuits Conference, 2005. Digest of Technical Papers.*, volume 1, pages 134–135, 2005.

[2] Netlib BLAS. `http://www.netlib.org/blas/`.

[3] TEXAS Advanced Computing Center. `http://www.tacc.utexas.edu/resources/software/#blas`.

[4] Antoine Petitet R. Clint Whaley and Jack J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, February 2000.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, S. Hammarling, and A. A. Greenbaum. *LAPACK users' guide Third Edition.* Siam, 1999.

[6] John Nickolls and I. Buck. Nvidia cuda software and gpu parallel computing architecture. In *Microprocessor Forum*, May 2007.

[7] *CUDA CUBLAS LIBrary.* NVIDIA Corporation, September 2008.

[8] *Basic Linear Algebra Subprograms Library Programmer's Guide and API Reference.* International Business Machines Corporation, 2008.