これからの並列計算のためのGPGPU連載講座(I)
GPUとGPGPUの歴史と特徴

大島聡史
東京大学情報基盤センター

1 はじめに

高い演算性能を安価に得られる新たな並列計算ハードウェアとしてGPU(Graphics Processing Unit)への注目が集まっている。GPUを用いた計算は「GPGPU(General-Purpose computing on GPUs)」や「GPUコンピューティング」と呼ばれ、国内外において様々な研究が行われている。今日では動画像処理ソフトウェアなどのソフトウェアパッケージにおいてもGPGPUが活用され始めている。GPUは、一般の小売店(家電量販店、PCパーツショップ等)で安価に購入できる製品が多く存在することや、既存のデスクトップ型PCに容易に搭載できることから、自作PC市場を中心に利用が広がってきた。カスタマイズされたGPUを搭載したゲーム機の普及も進んでいる。今日ではノートPCを含む大手メーカー製PCにもGPUを搭載したモデルが普及している。スーパーコンピューターにおけるGPUの利用も始まっている。2008年には東京工業大学が所有するスーパーコンピューター「TSUBAME」がNVIDIAのGPU(Tesla S1070)を追加搭載してTOP500の29位となり[1]注目を集めた。さらに2009年には中国の国防科学技術大学が所有するスーパーコンピューター「天河一号」がAMDのGPU(RadeonHD4870X2)を搭載してTOP500の5位となり[2]注目を集めている。

現在、東京大学情報基盤センター(以下、本センター)が所有・提供しているスーパーコンピューターにGPGPUのためのGPUを搭載したものは存在しない。しかしながら、GPUはスーパーコンピューター使用者の多くが関わる科学技術計算の分野において多く使われているため、また高性能コンピューティング(HPC)の分野におけるGPGPUの影響は年々大きくなっているため、本センターのユーザーにGPUの情報を提供することは重要であると考えられる。

さらに、GPGPUの正しい姿を伝えることも本連載の重要な目的である。今日ではGPGPUの成果が強調されるあまり「どんなプログラムでもCPUの代わりにGPUを用いて実行すれば性能が上がるのではないか」「(既存のスーパーコンピューターを使わずに) 全てのプログラムをGPUを用いて実行すれば良いのではないか」という意見が散見される。しかしながら、これは誤りである。CPUとGPUにはそれぞれ得意不得意があり、最大の性能を得るには適切に使用する必要がある。現実的には世の中の(スーパーコンピューター上で実行されている)全てのプログラムが、GPUによってCPUよりも高速に実行可能なわけではない。本連載では、GPUとGPGPUについての解説を行い、強力ではあるが万能ではないGPUの正しい姿を伝える。

本連載は今回を含めて5回で以下のような内容である：

- 第1回：GPUとGPGPUの歴史と特徴
- 第2回、第3回：GPUプログラミング環境CUDAの基礎、CUDAを用いたプログラムの最適化
- 第4回、第5回：GPUに関する研究について

第1回の今回は、GPUとGPGPUがどのように発展を遂げてきたのかや、GPUとCPUの特徴および違いについて紹介する。

2　GPUの基礎と発展の歴史

2007年にNVIDIA社によってGPGPU開発環境CUDA[3]が一般公開されると、それまでにGPGPUを利用していたか否かに関わらず多くのユーザーがCUDAに興味を示し、GPGPUの研究が広く注目を集めることとなった。ここではCUDAが一般公開されるまでの、言わば初期のGPGPUの時期におけるGPUの発展に注目し、GPUとはどのようなハードウェアなのか、何故高性能な並列計算ハードウェアとして注目されるようになったのかについて解説する。

GPUはGraphics Processing Unitの名が示すように画像処理のためのハードウェアであり、具体的には画像処理用のLSIチップのことを指す。ただし、このLSIチップはCPUのようにチップ単体で店頭および消費者の手元に供給されることはなく、ビデオカード(グラフィックスカード)の形をとる。そのうえ、ビデオカードにはビデオメモリと呼ばれる交換不可能なDRAMが搭載されており、GPUの性能に大きな影響を与えている。そのため、LSIチップ単体とビデオカードを区別せずにGPUと呼ぶことが多い。マザーボード上にオンボードグラフィックスチップとして搭載されるGPU製品も非常に多く、さらに現在ではノートPCにもGPUが搭載されているため、GPUという独立した製品以外の形で販売されているGPUも多い。過去には2Dグラフィックス向けのビデオカードや3Dグラフィックス専用のビデオカードも多く存在していたが、現在のGPUは2Dグラフィックスと3Dグラフィックスを兼ねているのが普通である。

GPUの本来の主な役割は、グラフィックス表示のための様々な処理を行い、処理の結果をディスプレイに表示することである。PC上で3Dグラフィックスが使われるようになった当時は、CPUが3D描画のための演算を行い、ビデオカードが結果を表示するという役割分担が行われていた。しかし1999年にMicrosoft社がリリースしたライブラリDirectX 7.0においてハードウェアによるポリゴンの座標変換およびライティング処理(Transform&Lighting)がサポートされたのを機に、3Dグラフィックス処理の多くの部分がGPU(ビデオカード)へと移って行った。現在ではGPUがグラフィックス処理を担っており、CPUの仕事は描画内容の生成や生成したデータをGPUへ送ること、そしてGPUの動作を制御することである(図1)。

GPUを用いた3Dグラフィックス描画処理の基本的な流れを図2に示す。GPUによって3Dグラフィックスが描画される際には、大きく分けて以下の5つの手順が必要である:

1. CPUからGPUへ描画情報(頂点やピクセルに関する情報)が送られる
2. 座標変換が行われ、頂点やポリゴンの画面上での位置の決定や、頂点単位での照明計算などが行われる：頂点処理
3. 頂点やポリゴンと画面上のピクセルとの対応付けが行われる
4. テクスチャの参照やピクセル単位での照明計算などが行われ、画面上の各ピクセルの色が決まる：ピクセル処理
5. ピクセルの色情報がフレームバッファに書き込まれ、画面へと出力される

これらの処理はパイプライン処理されるため、グラフィックスパイプラインもしくはレンダリングパイプラインと呼ばれる。3Dグラフィックス処理におけるパイプライン内の処理は対象プログラムによって異なる。例えば陰影の生成を行う処理だけ見ても、光源の種類や、投影先

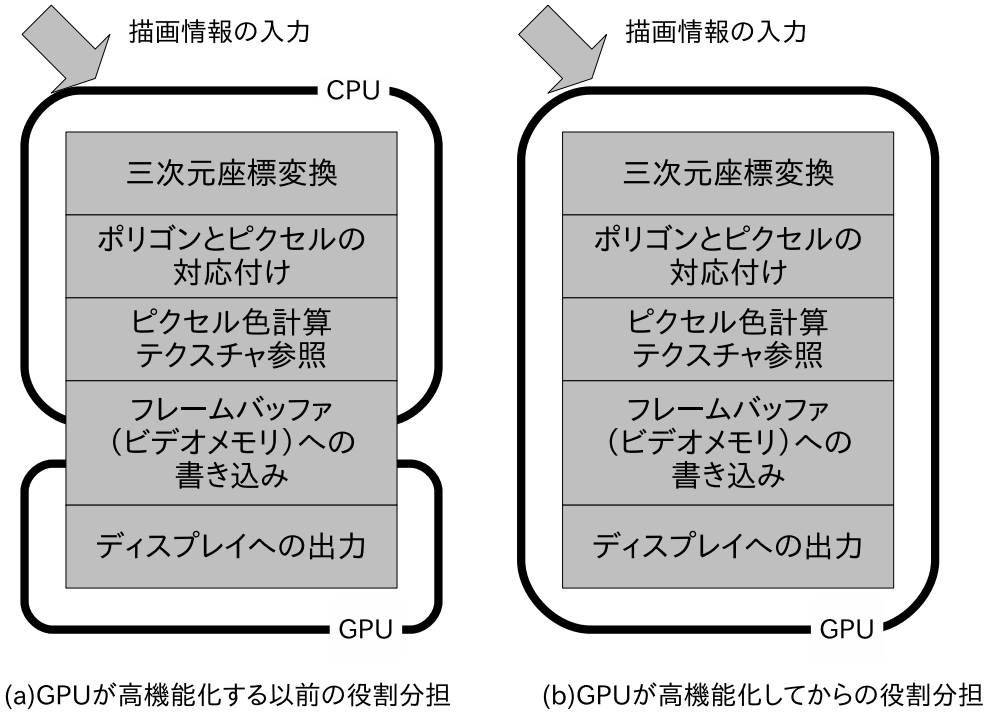


図 1　CPU と GPU の役割分担

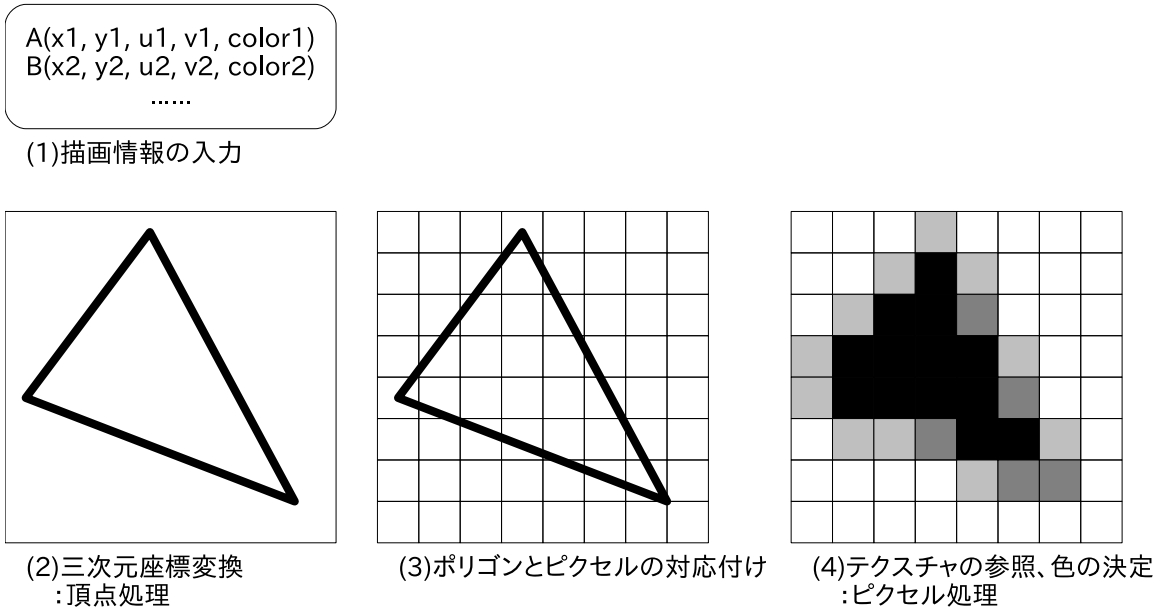


図 2　グラフィックス処理の基本的な流れ

の材質、陰影の元となる物体の形状をどれだけ細かく反映させるかなど、生成される陰影に影響を与えるパラメタは多岐に渡る。そのため陰影の生成方法は表現したい内容やハードウェアの性能によって様々な手法が用いられている。

描画処理技術が発展するにしたがい様々な描画手法が提案されると、提案された描画手法にあわせてGPUへのハードウェア実装が行われた。3Dグラフィックスが使われ始めた当時のGPUにおいては、グラフィックスパイプラインの動作制御はグラフィックスAPI(DirectXやOpenGL)を介した機能のON/OFFおよびいくつかの単純なパラメタによってなされていた。しかし様々な描画技法が開発されるようになると、以下のような問題が生じるようになった：

- 新たな描画手法を利用したいユーザー(ここでのユーザーにはプログラム作成者も含まれる)は最新のGPUを購入しなくてはならない。逆に、新たな描画手法を開発しても対象手法がGPUへ実装され製品としてリリースされるまで利用してもらえない。
- 最新の描画手法に対応したGPUを持つユーザーとそうでないユーザーが存在することにより各ユーザーが持つGPUの機能にばらつきが生じる。そのため、最新の描画手法を用いたプログラムをリリースしてもユーザーに利用してもらえない可能性が上昇する。
- GPUの回路規模が、対応する描画手法が増えるにしたがって大きくなる。しかし全ての手法を常に使うわけではないため、使用していない回路が増加しGPUの利用効率が低下する。

そこで、グラフィックスパイプラインをソフトウェアレベルで変更できるGPU、すなわち頂点処理とピクセル処理をユーザーが書き換えられるGPUが登場した。頂点処理に対応するプログラムは頂点シェーダ（VertexShader）、ピクセル処理に対応するプログラムはピクセルシェーダ（PixelShader）もしくはフラグメントシェーダ（FragmentShader）と呼ばれ、それぞれ専用のシェーダユニットが処理を担当した。これらの機構はプログラマブルシェーダと呼ばれ、DirectXとOpenGLの2大グラフィックスAPIおよび多くのGPU製品が対応したために広く普及することとなった。なお、プログラマブルシェーダに対して既存のシェーダは固定機能シェーダと呼ばれる。

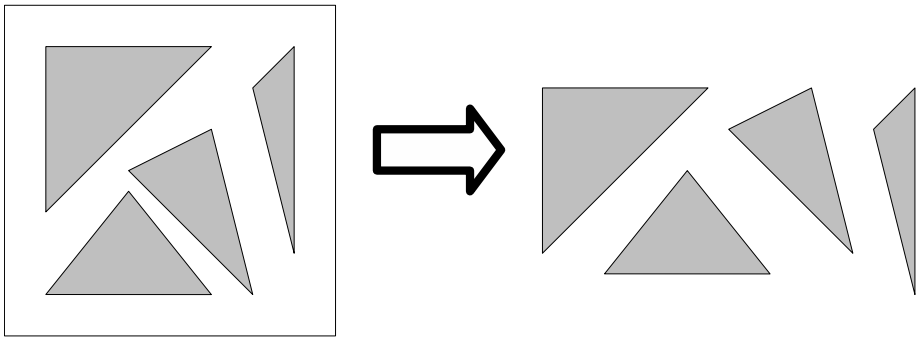
初期のプログラマブルシェーダはプログラミングの制限が厳しく、複雑なアルゴリズムを実装するのは困難であった。例えば、

- 対応している命令の種類が少ない
- 1プログラム中に記述可能な命令数の制限が厳しい
- 分岐を含むプログラムが記述できない
- 扱えるデータ型の制限が厳しい

などの制限が存在した。しかしながら、より高度なシェーダへの要求が高まるにつれて、GPUの世代が進むたびに制限の緩和が進み、様々な描画手法がGPU上で実現されることとなった。GPU内部で行われる処理をユーザーが自由に記述できるようになったことは、のちにGPUがGPGPUとして汎用演算に使われるようになる上で重要である。

ところで、3Dグラフィックス処理における頂点処理やピクセル処理は高い並列性を持った処理である(図3)。頂点処理においては、多数のオブジェクトの座標変換をする場合でもオブジェクト同士は互いに影響し合わない計算が多いため、オブジェクト単位で並列に座標変換処

理を行うことができる。またピクセル処理においても、ピクセル単位で並列にピクセル処理を行えることが多い。そのため、GPUは内部の演算器(シェーダユニット)の数を増やすこと、すなわちハードウェアレベルでの並列性を向上させることによって性能を大きく向上させた。



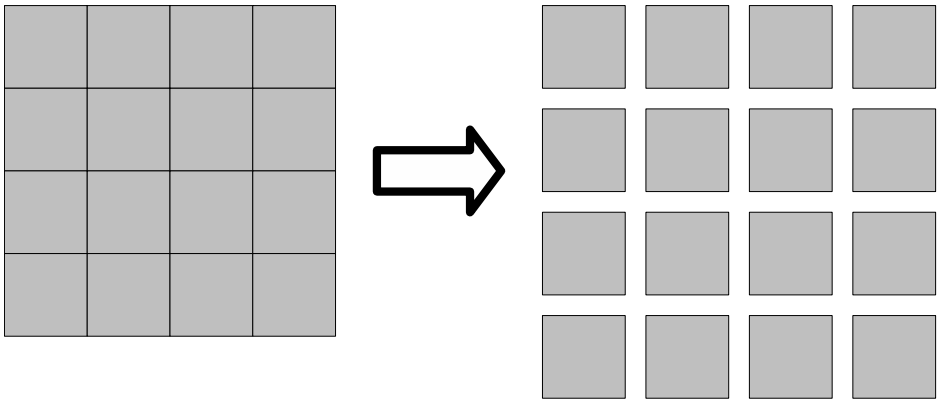


図 3　グラフィックス処理における並列性

たとえば2004年に発表・発売されたGPU GeForce6800は自由度の高いシェーダユニットを並列に搭載(最大6個の頂点シェーダユニットと最大16個のピクセルシェーダユニットを搭載、ユニット数を減らしたモデルがいくつか存在) することで高い性能を達成し、さらに従来より自由度の高いシェーダプログラムや精度の高い計算を扱えるようになったため、大いに注目を集めた。GPUがグラフィックス処理の高速化のために演算器の並列度を向上させたことは、のちにGPUが並列処理に強い計算ハードウェアとして注目される上で重要である。

ある程度のアルゴリズムが実装可能な演算の自由度や高いハードウェアの並列度といったGPUの特徴は、複雑で演算量の多い高度な画像処理に対応するためにもたらされたものであったものの、GPUが様々なアルゴリズムを実装できるだけのプログラマビリティと高い並列性を持ったことは、のちにGPGPUが大きく注目されることにつながったと言える。

2006年以降、プログラマブルシェーダのユニファイドシェーダ化が進んだ。これは、従来のGPUでは頂点処理とピクセル処理がそれぞれ専用の演算ユニットによって行われていたのに対して、共通のシェーダユニット(ユニファイドシェーダユニット)によって行われるものである。従来のシェーダユニット構成においては、頂点処理とピクセル処理の最適なバランスが対象となるプログラム(描画手法)によって違うために最適なユニット数を決めることが難しいと

いう問題が知られていた。例えば、頂点座標の変換や頂点色の計算には独創的で複雑な処理をしたい一方でピクセル処理は比較的単純なプログラムでは、頂点シェーダの演算量が多くなる。逆に頂点処理はシンプルだが複雑なピクセル単位の演算をしたいプログラムでは、ピクセルシェーダの演算量が多くなる。これらのようなプログラムでは、いずれか一方のシェーダユニットばかりに大きな負荷がかかり、もう一方のシェーダユニットにはほとんど負荷がかからない可能性がある(図4-a)。しかし、GPUを作る(購入する)時点ではどちらの処理ユニットに大きな負荷がかかるかは予測が困難なため、バランスの取れたGPUを作る(購入する)のは難しく、頂点シェーダユニットとピクセルシェーダユニットへの負荷の不均衡を避けることも難しい。

そこで、両方のシェーダユニットの役割を兼ね備えたユニファイドシェーダユニットが使われるようになった。ユニファイドシェーダユニットは単一のシェーダユニットが頂点処理とピクセル処理のどちらでも行うことが可能であり、状況に応じて頂点シェーダユニットもしくはピクセルシェーダユニットとして動作する。頂点処理を重視する描画処理では多くのシェーダユニットが頂点シェーダユニットとして、またピクセル処理を重視する描画処理では多くのシェーダユニットがピクセルシェーダユニットとして働くことで、対象処理の内容に関わらず常にシェーダユニットの使用率を高くすることができる(図4-b)。GPGPUにおいてはピクセルシェーダの需要が大きい傾向があったため、GPU上の多くのユニファイドシェーダユニットがピクセルシェーダユニットとして動作をすることで高い性能が得られることになる。

その後2007年にかけて、さらに自由で高度な表現を容易に行うために、頂点シェーダとピクセルシェーダに続く第3のシェーダであるジオメトリシェーダが導入された。ジオメトリシェーダは頂点シェーダが処理した頂点をさらに変化させるものであり、ポリゴンモデルの頂点を増減させるなどの処理が行えるようになった。

ちなみに、現在市販されている主なGPU製品は以下の通りである:

- NVIDIA社のGeForceシリーズ、Quadroシリーズ、Teslaシリーズ
- AMD社のRadeonシリーズ、FireProシリーズ、FireStreamシリーズ
- Intel社の統合グラフィックスチップ

ただし、Intel社の製品はビデオカードとして市販されておらず、マザーボード上に備え付けられたオンボードグラフィックスチップとしてのみ流通している。

3 初期のGPGPU

現在GPGPUプログラミングに用いられるプログラミング環境としてはCUDAが台頭しているが、CUDAが登場する以前の初期のGPGPUにおいてはグラフィックスプログラミングの手法が多く用いられていた。現在ではグラフィックスプログラミングの手法を用いてGPGPUを行うユーザは(高度なグラフィックス表現にGPGPUを利用するユーザを除いて) 少なくなっている。本章ではGPGPUがどのように発展してきたのかを概観するために、初期のGPGPUではどのようにグラフィックスプログラミングを用いて汎用演算を行っていたのか、そのアイディアと実装について紹介する。

GPUは高度なグラフィックス表現のために、プログラムの書き換え(プログラマブルシェーダ)により様々な処理が可能となり、演算器(シェーダユニット)を並列化することで並列演算性能を向上させてきた。特に高度な陰影処理や照明処理には従来の画像処理と異なり汎用的な演

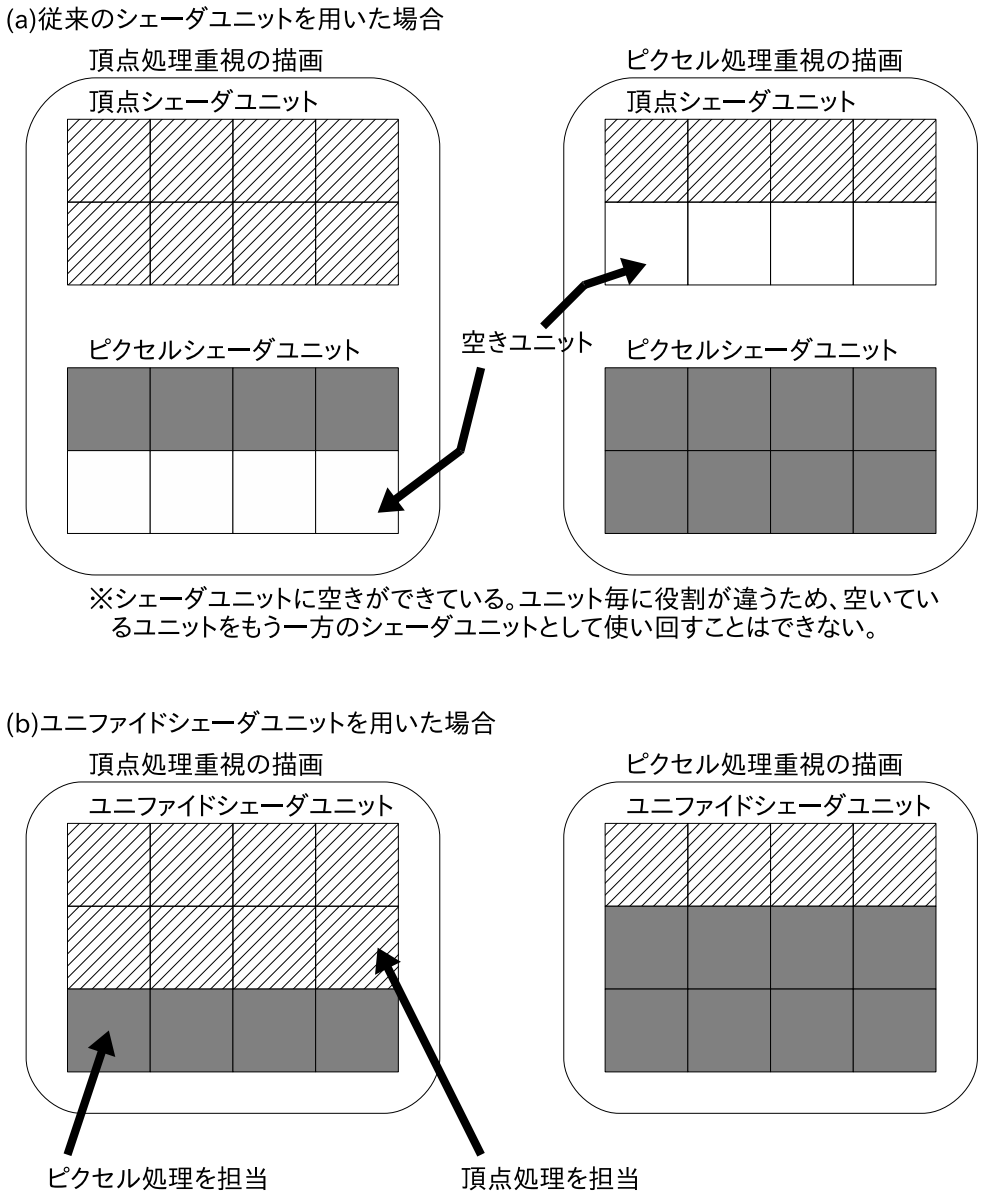


図 4　シェーダユニットの構成

算が含まれるようになった。さらに、従来のGPUでは描画結果はフレームバッファに書き込まれてディスプレイに表示されるものだったのに対して、GPU上のメモリ(テクスチャ)に書き込んで再利用(次のフレームを描画する際に参照)する使い方も可能となった。

ここで、2枚のテクスチャ(入力テクスチャ)を参照し、それぞれの色(RGBA値[*1]) を足し合わせて別のテクスチャ(出力テクスチャ)に書き込む(テクスチャに対して描画する)という描画処理を考えてみることにする。これはグラフィックス処理の観点で見ればテクスチャのブレンドと呼ばれる非常に一般的な処理である。一方でこれを別の観点から見ると、2つの入力配列のデータを入力テクスチャという形式でGPUに入力し、入力配列のデータを加算した新たな1つの出力配列を出力テクスチャという形で格納したとみなすことができる。入力テクスチャに配

[*1] Red,Green,Blue,Alpha の 4 値。RGB は光の三原色を、Alpha は透明度を意味する。

置するデータをCPUが生成し、出力テクスチャに格納されたデータをCPUが取得すれば、これは配列の加算という汎用的な演算と同じであると見なすことができる。(図5)

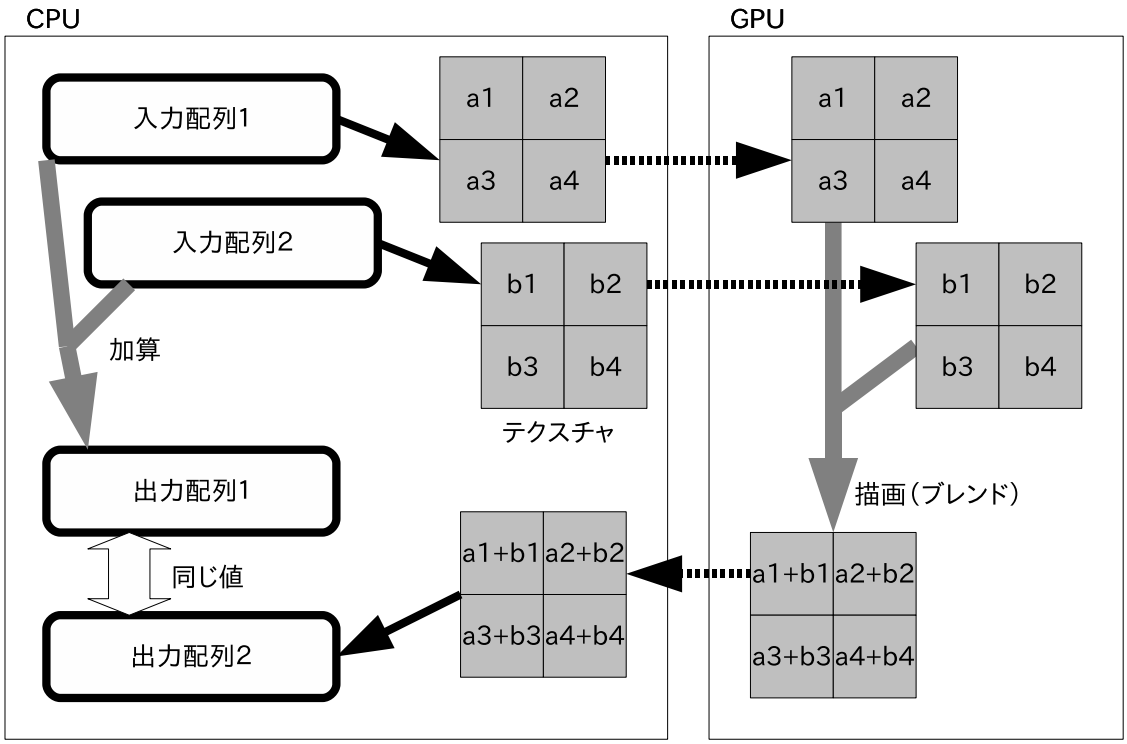


図 5　配列の加算

以上のように、CPUからGPU上のテクスチャへのデータ配置を入力、GPU上での(テクスチャに対する)描画処理を演算、GPUの描画結果(テクスチャのデータ)をCPUへ書き戻す処理を出力とみなすことで、GPUによる汎用演算が可能であることがわかった。特に、

- GPUが本来行ってきたグラフィックス処理の並列性が高いため、GPUはハードウェアレベルでの高い並列性を備えている
- グラフィックス処理にはXYZW(座標計算)およびARGB(色計算)の4要素ベクトル演算が多用されるため、GPUはベクトル計算に強い

という特徴があることから、GPUを用いることで対象問題によってはCPUを圧倒する性能が得られるのではないかと期待されるようになった。

なお、ここで例に挙げた描画処理はプログラマブルシェーダが必須な処理ではない。テクスチャのブレンド自体はプログラマブルシェーダが登場する前から使われている技術である。テクスチャへの描画やテクスチャに格納されたデータの取得も、プログラマブルシェーダとは直接関係なく発展したグラフィックスプログラミングの技術である。

このような初期のGPGPUにおけるプログラミング環境としては、グラフィックスAPIとシェーダ言語が使用された。グラフィックスAPIは画像処理を記述するためのAPIであり、具体的にはMicrosoftのDirectX(Windows専用)とThe Khronos GroupのOpenGL(マルチプラットフォーム対応)が多く用いられた。GPUを用いたグラフィックスプログラミングにおけるグラフィックスAPIは、CPUがGPUにアクセスするためのインターフェースであり、頂点やポリ

ゴンの設定、描画機能の選択、その他GPUの制御などを担当する。一方シェーダ言語はGPUの動作を記述するための言語であり、DirectX用のHLSL[4]、OpenGL用のGLSL[5]が用いられた。DirectXとOpenGL両方で利用可能なCg[6]も一部では利用された。いずれのシェーダ言語もC言語を拡張した高級言語であり、高級言語が使われる前にはGPU向けのアセンブラも使われていた。

しかし、グラフィックスプログラミング(グラフィックスAPIとシェーダ言語を用いたプログラミング)を用いたGPGPUはユーザーにとって使いやすいものではなかった。グラフィックスプログラミングは画像処理のために作られた仕組みである。そのため、描画処理とその他の処理とを組み合わせるようなGPGPUプログラミングには適していたが、描画処理とは関係のない汎用処理を実装する際には描画処理と対象処理とが容易に対応づけられるとは限らなかった。さらに、グラフィックスプログラミングそのものの難しさも利用の妨げとなった。例えば数値計算を行っているユーザーがGPGPUに挑戦しようと思った場合には、専門外であるグラフィックスプログラミングの勉強をしたうえで、自分のプログラムとGPUとの対応付け、たとえば数値データをどのようにテクスチャへ配置するかや限られた言語仕様の中でアルゴリズムをどのように記述すれば良いのかについて検討し、実装する必要があった。さらに、GPUベンダーがGPUの内部構造や詳細なアーキテクチャを公開していなかったことが最適化の妨げとなった。ユーザーは描画処理と対象問題との対応付けの工夫によって最適化を行ったが、グラフィックスAPIとシェーダ言語によって隠蔽された情報のみを用いてGPUの性能を最大限に引き出すことは容易ではなかった。こうした利用の難しさや手間がGPGPUが普及する上での課題となっていた。

こうした状況を踏まえて、GPGPU専用の言語やライブラリに関する研究等も行われた。これについては次章にて概要を紹介し、さらに、CUDAについては本連載の第2回および第3回で、その他のプログラミング環境については第4回および第5回において詳しく紹介する。

本章の最後に、グラフィックスプログラミングを用いた行列積の実装例を2つ簡単に紹介する。1つ目はプログラマブルシェーダを用いずにグラフィックスAPIのみで行う行列積であり、2つ目はプログラマブルシェーダ(実際に使うのはピクセルシェーダ)を用いた行列積である。

1つ目の実装例の概要を図6に示す。本手法はLarsenらによって2001年に実装されたものである[7]。本手法の特徴は、加算ブレンド描画や乗算ブレンド描画を活用している点である。プログラマブルシェーダが用いられる以前からテクスチャの色情報を加算したり乗算したりすることは可能だった。そこで、テクスチャ参照座標を調整した乗算描画を繰り返し行うことで行列積を実現した。図6には、大きさ4×4の正方行列に対する行列積の一部を示している。i回目の描画では、各ピクセルにおいて行列(テクスチャ)Aのi列目と行列(テクスチャ)Bのi行目に該当するテクスチャ座標の色を参照し取得して乗算ブレンド描画する。この描画を4回繰り返し、各回の結果を加算ブレンド描画すると、行列積を行ったのと同等の結果が得られる。

2つ目の実装例の概要を図7に示す。本手法はテクスチャを入出力の配列として、プログラマブルシェーダ(ピクセルシェーダ)を計算内容として使用している。ピクセルシェーダでは、各ピクセルにどのような値を書き込めば(出力すれば)良いかを記述する。スレッドプログラムに対応づけて例えるのであれば、描画されるピクセルの数すなわち行列の要素数分だけスレッドを生成し、各スレッドの動作をシェーダプログラムで指定しているのに近い。図の擬似コードではposX,posY関数を用いてピクセルの座標を取得しているが、これはスレッドプログラミン

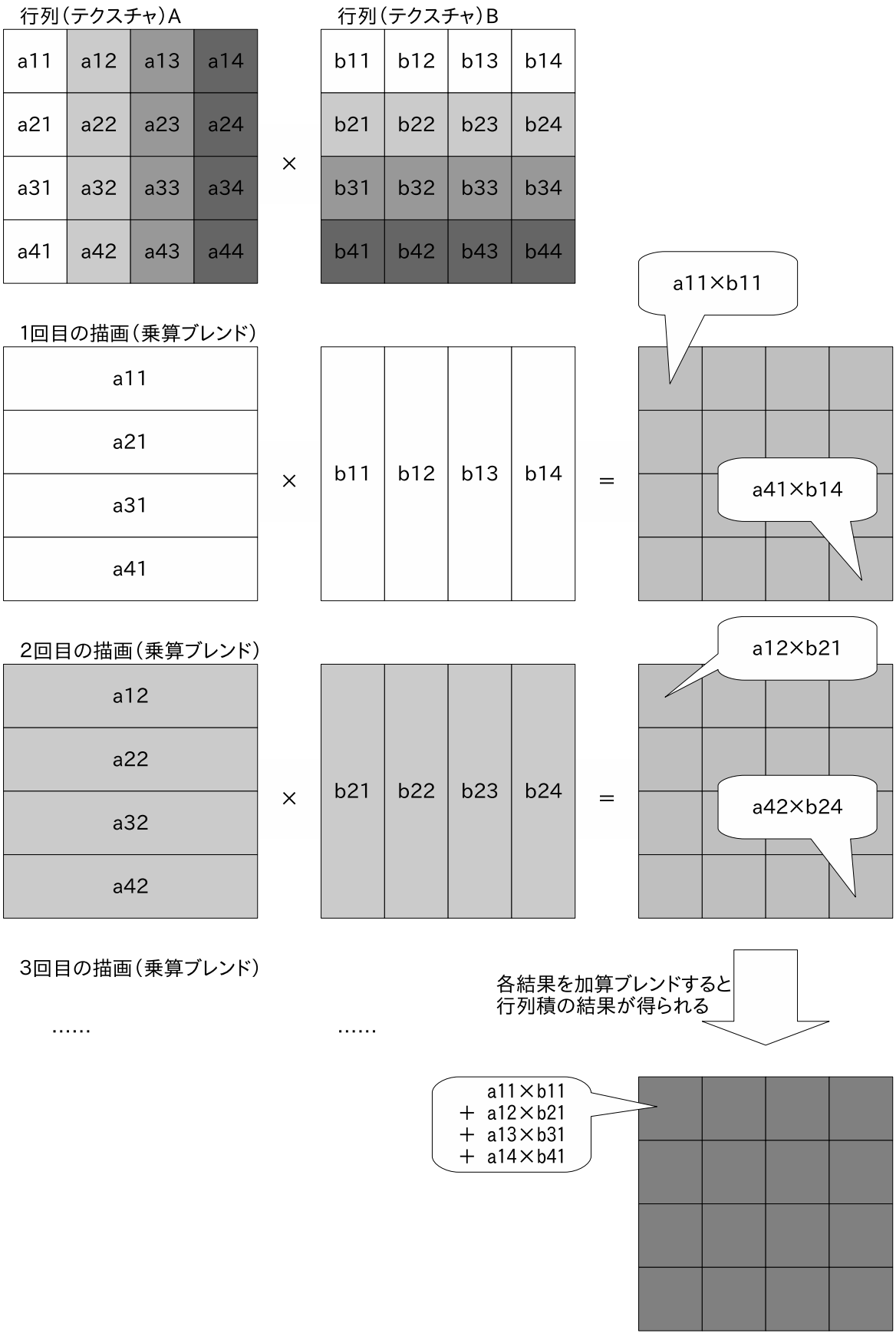


図 6　グラフィックス API を用いた行列積

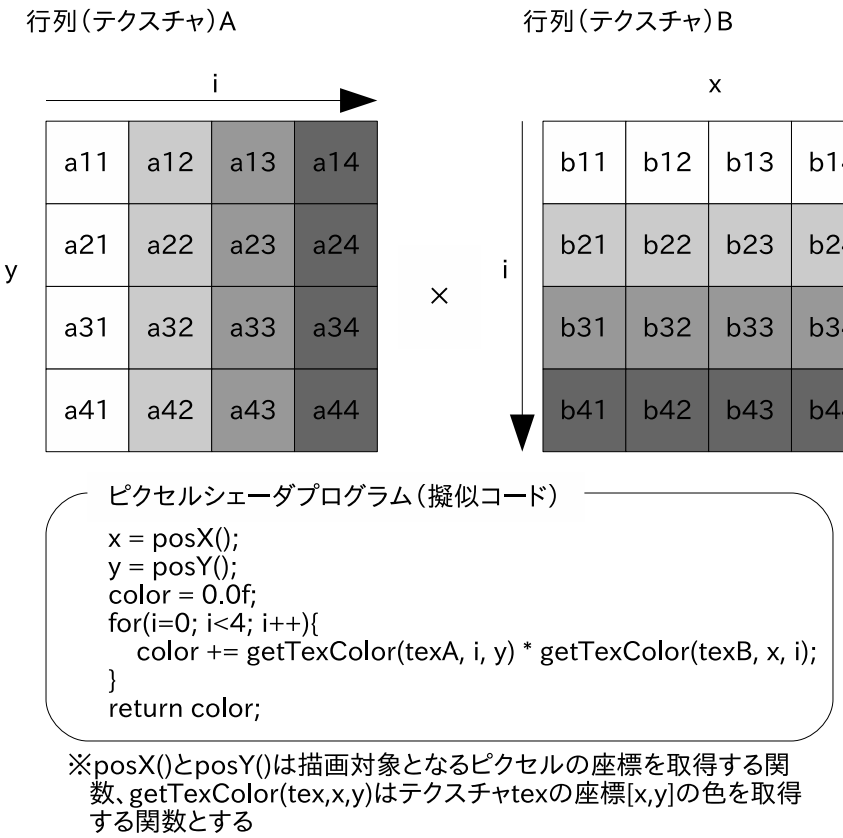


図 7　プログラマブルシェーダ (ピクセルシェーダ) を用いた行列積

グにおけるスレッドIDの取得に相当する。IDを元に参照すべきデータのインデックス、すなわちテクスチャ座標を計算して入力データを取得し、乗算と加算を行って行列積の結果を求めている。実際にグラフィックスプログラミングを行う場合にはプログラマブルシェーダを利用するための準備など手間がかかるコーディングが必要であるが、行列積のアルゴリズム自体は簡潔に記述することが可能である。プログラマブルシェーダを用いた行列積の研究は多数行われており、ブロック化やベクトル型の使用などの高速化手法が提案・実装された[8, 9]。

4　現在のGPUとGPGPU

GPUの性能はCPUを超える勢いで向上し続けている。特に、グラフィックス性能の向上が行われ続けているのみならず、GPGPUを意識した性能向上も行われている。これはGPUベンダーがGPGPUを一過性の流行以上の重要なものであると認識していることのあらわれであると考えられる。例えば、従来のGPUはポリゴンの描画速度や新たなグラフィックス機能を前面に出した製品開発や広告活動を行っていた。これにたいして、今日のGPUは理論浮動小数点演算性能やGPGPUとしての利用の成果も前面に出している。さらにHPC分野からの要求に応えて、グラフィックス目的では不要と言われている倍精度浮動小数点演算性能の強化やECCメモリの搭載なども行われつつある。GPGPU専用の、ディスプレイ出力を持たないGPUまでもがリリースされている。

GPUベンダーによるGPGPUプログラミング環境への注目も大きくなっている。NVIDIAは

表 1　性能比較表

ベンダー	Intel	AMD	AMD	NVIDIA
型番	XeonX5570	Opteron8435	FireStream9270	TeslaC1060
計算コア数	4	6	800	240
コアクロック周波数	2.93GHz	2.6GHz	750MHz	1.3GHz
メモリ帯域幅	25.6GB/sec	12.8GB/sec	108.0GB/sec	102GB/sec
対応するメモリの種類	DDR3	DDR2	GDDR5	GDDR3
最大メモリ容量	72GB	32GB	2GB	4GB
最高メモリクロック	1333MHz	800MHz	850MHz	800MHz
理論演算性能 (Float)	46.88GFlops	62.4GFlops	1.2TFlops	933GFlops
理論演算性能 (Double)	46.88GFlops	62.4GFlops	240GFlops	78GFlops
最大消費電力	85W	75W	220W	187.8W
電力あたり演算性能 †	0.55GFlops/W	0.83GFlops/W	5.45GFlops/W	4.97GFlops/W

† 最大消費電力 1W あたりの理論演算性能 (Float)

2007年にGPU向けの開発環境CUDA(Cuda Unified Device Architecture、開発環境だけではなくアーキテクチャの名前でもある)を、AMDも2008年にATI Stream SDK(AMD Stream SDK)とBrook+からなる開発環境の一般公開を開始した。特にCUDAは既存のC言語により近い記述ができること、公式に提供するドキュメントやサンプルが充実していたこと、各種イベント等でのアピールが効果的だったことなどが功を奏してか、2009年までには多くのCUDAユーザおよびCUDAに関する研究の事例を獲得するに至っている。

第1回の最後として、CPUとGPUとの違いについて簡単にまとめる。最近のGPUのアーキテクチャについては次回以降にプログラミング環境と一緒に紹介することとし、今回はいくつかの性能指標についてのみ言及する。

表1はCPUとGPUの主な性能数値を並べて表にしたものである。CPUとしてサーバ用のIntel XeonX5570およびAMD Opteron8435、GPUとしてHPC向けのNVIDIA TeslaC1060およびAMD FireStream9270 を挙げている。CPUとGPUのアーキテクチャが異なるのみならずGPU同士のアーキテクチャも大きく異なるものの、CPUとGPUの違いが明確に示されている項目がいくつか見受けられる。

まず、計算コアの数が大きく異なることがわかる。CPUのコア数がたかだか6(XeonのHyperThreadingを含めても8)であるのに対して、GPUは800コアないし240コアという極めて多数のコアを搭載している。一方でコアクロック周波数を比較してみると、CPUは2GHz以上の高い周波数であるのにたいして、GPUはたかだか1GHz程度の低い周波数である。しかし、CPUの計算コアとGPUの計算コアが同程度の機能・性能を備えているわけではない。またGPUの計算コアは全てのコアが互いに異なる演算を同時に行えるようには作られておらず、SIMDのようにいくつかのコアが同じ計算を行うことが前提となっている。現在のCPUはチップ面積の多くをキャッシュや分岐予測など計算以外の仕事を行う部分が占めている。一方GPUはチップ面積の多くを多数のシンプルな計算コアが占めている。そのためGPUは、シンプル

だが高並列な問題では著しく高い性能が得られる一方で、分岐の多いプログラムや並列度が低い問題では高い性能が得られない。このような計算コアの傾向の違いは、GPU本来の用途であるグラフィックスの処理においてGPUに求められてきたのはCPUのようなキャッシュや分岐予測などの機能の強化ではなく、多数の頂点やピクセルに対して次々に処理を行える高い並列演算性能であったことによるものである。

続いてメモリの性能について比較する。GPUはグラフィックス用に開発された高速なメモリGDDR SDRAMを搭載している一方で、搭載可能な最大容量はCPUと比べて小さいことがわかる。そのため大規模な問題を解こうとする場合には、CPU側で対象問題を1GPU上のメモリに乗る大きさに分割して実行する必要がある。

さらに最大消費電力を見てみると、GPUはCPUと比べて多くの電力を消費する可能性があることがわかる。しかし演算性能に大きな差があるため、電力あたり演算性能を比較するとCPUよりもGPUの方が優れている。

以上をまとめると、単純で並列性が高くGPU上のメモリに収まるサイズに分割可能な問題はGPUで解き、そうでない問題はCPUで解くことで、高い性能および電力あたり演算性能が期待できるということがわかる。

しかし、GPUによる演算にはCPUとの通信が必要である。GPUはあくまで演算ユニットであり、GPU単体ではユーザーからの入力を受けとることや結果をディスプレイ以外へと出力することができない。そのためGPUを利用する際には、CPUからGPUへデータを送信し、演算結果をGPUからCPUへ書き戻す必要がある。CPU-GPU間の通信には一般的にPCI-Expressバスが使われる。PCI-ExpressはGen2 x16で双方向合計16GB/sという高い転送性能を持つものの、GPU上のDRAMとGPUコア間の通信性能と比較すると低速であり、CPU-GPU間の送受信が多いプログラムを実行する場合にはGPUの演算を妨げ性能低下を引き起こす可能性が高い。また、対象プログラムに単純な並列処理によって高性能が得られる部分以外の部分が含まれる場合には、並列度が低くて性能が得られないことを理解した上でGPUに演算を行わせるか、CPU-GPU間の通信がオーバーヘッドになることを理解した上でCPUに演算を行わせる必要がある。さらにCPUがGPUを制御すること自体も多少のオーバーヘッドになる。以上のことから、プログラムの全てが単純に並列処理できる場合を除いては、単純にGPUを用いて並列高速化が行えるとは限らない。一方で、CPUとGPUはそれぞれが同時に別々の演算を行うことが可能なため、CPUとGPUによる並列演算によって性能向上可能な場合もありえる。

さらに、CPUと同様に複数のGPUを用いた計算機環境を構築することができる。複数のCPUを用いる方法としては、1台の計算機に複数のCPUを搭載するマルチソケットと、複数台の計算機を用いるマルチノード、そして両方を組み合わせた方法が考えられる。一方でGPUを複数用いる方法も、1台の計算機に複数のGPUを搭載する方法と、GPUを搭載した計算機を複数用いる方法、そして両方を組み合わせた方法が使用可能である。しかしながら、GPUはPCI-ExpressなどのバスでCPUやメインメモリと接続されるハードウェアであり、同一計算機上のGPU同士が直接通信することや、異なる計算機上のGPU同士が直接通信することはできない。(NVIDIAのSLIやAMDのCrossFireといった複数のGPUを利用する技術には同一計算機上のGPU同士を接続するケーブルが存在するが、ドライバレベルの制御によって複数GPUにより描画を高速化するためのものであり、ユーザーが細かく制御することはできない。) さらに、複数のGPUを搭載した計算機環境を構築する場合は、GPU上のメモリを管理するために十

分大容量のメインメモリを搭載する必要もある。このように、複数のGPUを利用するとCPUやメインメモリに負担がかかるため、台数効果を得るのは難しい。今日では複数のGPUを搭載可能なチップセットやマザーボードの流通が増えてはいるものの、複数のGPUを活用して高い性能を得るための知識や技術はまだ普及しているとは言い難い。

以上、第一回の今回はGPUとGPGPUがどのように発展を遂げてきたのかや、GPUとCPUの特徴および違いについて紹介した。次号では、現在広く利用されているGPGPU環境CUDAについて、概要やプログラミング方法を解説する。
(次号に続く)

参考文献

[1] 東京工業大学 学術国際情報センター. 東京工業大学が世界初の大規模GPGPUコンピューティング基盤を大規模スーパコンピュータ基盤TSUBAME上に実現. プレスリリース, http://www.gsic.titech.ac.jp/contents/news.html.ja?page=News/2008/1119/01, November 2008.

[2] TOP500 Supercomputing Sites. Tianhe-1, November 2009.

[3] NVIDIA. CUDA Zone – The resource for CUDA developers, http://www.nvidia.com/object/cuda_home.html.

[4] Microsoft. DirectX デベロッパーセンター, http://msdn.microsoft.com/ja-jp/directx/default.aspx.

[5] The Khronos Group. OpenGL - The Industry Standard for High Perfoemance Graphics, http://www.opengl.org/.

[6] NVIDIA. Cg Toolkit - GPU Shader Authoring Language, http://developer.nvidia.com/object/cg_toolkit.html.

[7] E.Scott Larsen and David McAllister. Fast matrix multiplies using graphics hardware. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, page CDROM, 2001.

[8] K.Fatahalian, J.Sugerman, and P.Hanrahan. Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication. In *Graphics Hardware 2004*, pages 133–137, 2004.

[9] Jesse D. Hall, Nathan A. Carr, and John C. Hart. Cache and Bandwidth Aware Matrix Multiplication on the GPU. Technical report, University of Illinois Dept. of Computer Science, 2003.