GPGPUを用いたソフトウェア高速化手法

Technique to Speedup of the software by GPGPU

大田 弘樹* 馬場 明子* 下田 雄一* 安田 隆洋* 山本 啓二*

Hiroki Ota, Akiko Baba, Shimoda Yuichi, Takahiro Yasuta, Keiji Yamamoto

PCやワークステーションにおいて画像処理に特化して使用されてきたGPUを、汎用的な数値計算処理に使用しソフトウェアを高速化するGPGPUが近年注目されている。GPUは多数コアによる並列処理を導入しており、CPUより高い並列処理能力を持っているが、実際にGPGPUを用いてソフトウェアを高速化するためには、GPUのハードウェア・ソフトウェアの各アーキテクチャを理解した上で、高速化対象のソフトウェアに応じて、速度性能を最大限に発揮するための手法を適用する必要がある。本報告では、通常のCPUより数倍程度広いGPUのメモリバンド幅に着目し、メモリバンド幅を効率的に有効活用することでソフトウェアを高速化する手法を紹介する。

Recently GPGPU (General-purpose computing on graphics processing units) has attracted much attention to realize high-speed software processing with GPU (graphics processing unit) which is specialized to perform for image processing in a PC and a work station, applying it to general-purpose numerical computation processing. GPU has thousands of cores and potentially better capability for parallel processing than that of CPU. Thus, to get the benefit from GPU performance, we must tune up the program based on the knowledge of both hardware and software architectures. In this report, we focus on memory bandwidth of GPU which is several times greater than that of CPU, also introduce the method to provide high-speed software processing with effective use of it.

1. まえがき

センサ技術において分解能・ダイナミックレンジが向上してきたことで、取り扱うデータ量が大規模となっている。当社が開発を担当している医用画像の世界においても、CT撮像装置等の性能向上により、得られるデジタル画像の解像度は高く、色深度は深く（ビット数が多く）なってきており、画像サイズが増大している。データ量の増加はソフトウェアの処理時間増加につながるため、必然的にソフトウェアの高速化が求められている。また、ソフトウェアを高速化することにより、同じ処理時間で、より精細な条件によるシミュレーションが実行可能となる。例えば医療現場では、より高精度なシミュレーションの結果、精密な治療を実現することで患者への負担を軽減することが求められており、ソフトウェアの高速化によるシミュレーション精度の向上が必須課題となっている。

本書では、ソフトウェアを高速化する手段として、GPU(*1)をより効果的・効率的に使用するための手法、および、開発事例を紹介する。

2．CPUによる高速化の限界とGPUによる高速化の始動

2.1 CPUによるソフトウェア高速化の限界

各CPUメーカは、CPUのクロック周波数を高くすることで、CPUの性能を向上させてきた。しかし、この方法では消費電力と発熱量が増加する問題が発生したため、「クロック周波数の向上」から「コア数の増加」へシフトしてきた（図１参照、横軸は発表年）。

ソフトウェアの高速化においては、CPUに搭載され

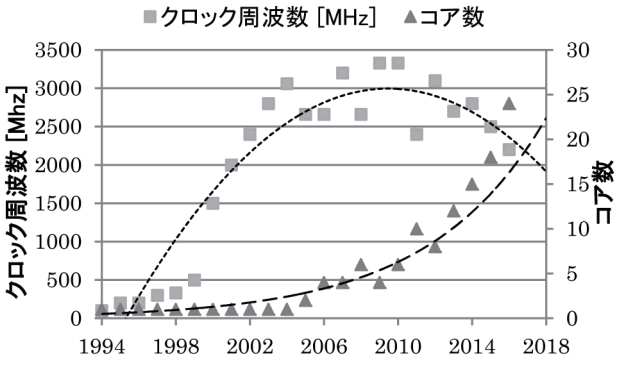

図１ CPUのクロック周波数とコア数の推移(1)(2)

＊関西事業部 第五技術部

たSIMD (*2) 対応の演算器で実行する並列ベクトル化、および、複数コアによるスレッド並列化により実現してきた。当社が開発に従事してきた粒子線線量計算エンジンにもこれらの手法を適用し、高速化を実現してきた (3)。

しかし、H/Wの進化により2016年現在では、1コアでも、ある程度の演算速度を実現できるようになり、また、複数コアによるスレッド並列化を実装しても、複数コアで共用するメモリとの接続バスがボトルネックとなるため、ソフトウェアの高速化には限界がある事が判明している。粒子線線量計算エンジンにおいても、9スレッド以上でオーバヘッドによる性能劣化が発生した（図２、図３参照）。

メモリアクセスが多いソフトウェアでは、メモリバンド幅の広いCPUを使用することが処理速度における重要課題となるが、図４に示す通り、CPUのメモリバンド幅の性能向上は計算性能のそれに対して遅く、今後も、その傾向は継続すると考えている。具体的には、2006年から2016年の間に、計算性能は63倍（18.6→1,164.8GFlops (*3)）に向上したが、メモリバンド幅は6倍（12.8→76.8GB/s）に留まっており、頭打ちとなっている。

2.2　GPUによるソフトウェア高速化の始動

CPUのメモリバンド幅において大幅な拡張が見込めないため、高速化に向けては、メモリバンド幅が広いハードウェアへ処理をオフロードする必要がある。例えば、表１に示す通り、GPUはCPUよりメモリバンド幅が広いため、メモリバンド幅がボトルネックとなっていたソフトウェアをGPGPU (*4) により高速化できると考えられる。

図５に示す通り、GPUはCPUより性能向上率が高く、今後もハードウェアの性能向上を享受できる可能性があると考えられる。また、スーパーコンピュータ等のメモリバンド幅が広いH/Wを新たに使用する場合と比較し、導入にあたり以下のメリットがある。

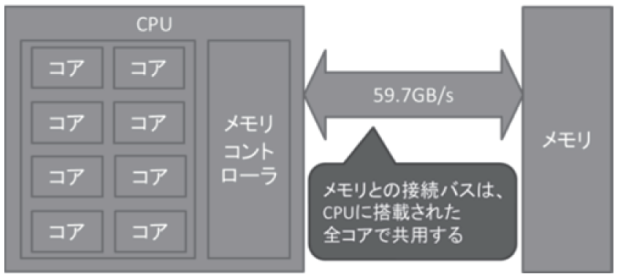

図２　CPU処理におけるボトルネック (1)

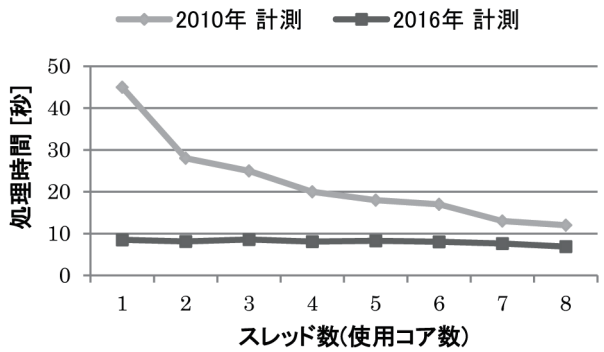

図３　粒子線線量計算エンジンの高速化

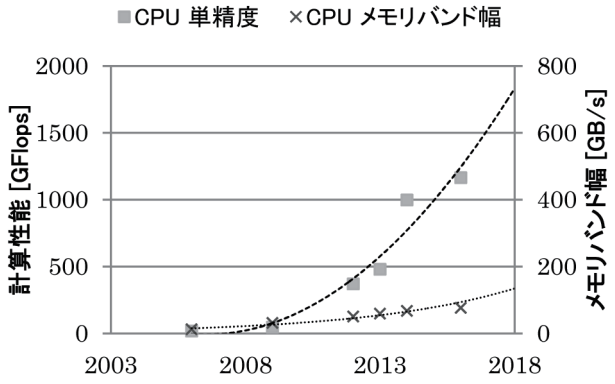

図４　CPUの計算性能とメモリバンド幅の推移 (1)

表１　CPUとGPUの性能比較 (1) (2)

プロセッサ名	単精度[GFlops]	メモリバンド幅[GB/s]
CPU Intel Xeon E5-2690 v4	1,164.8	76.8
GPU NVIDIA Tesla K40c	4,290.0	288.0

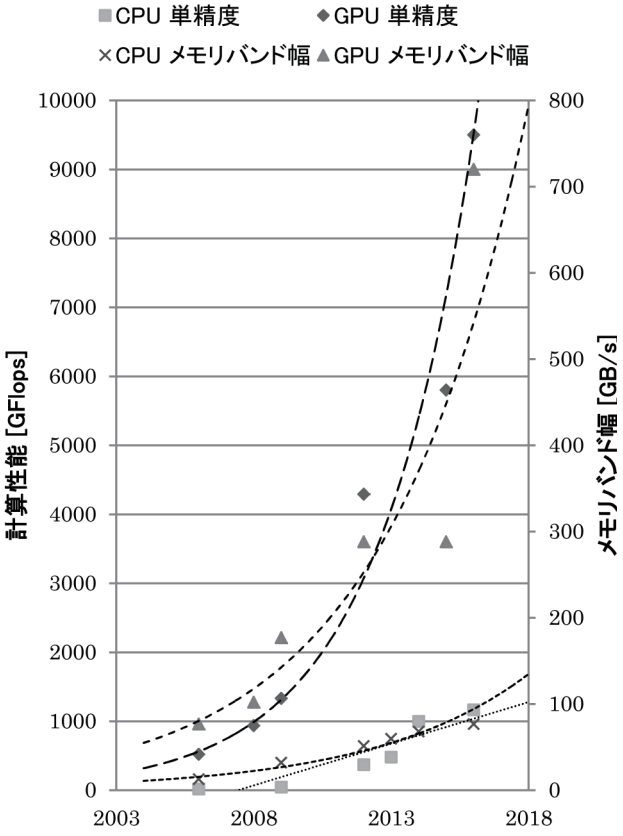

図５　CPUとGPUの性能推移 (1) (2)

1　スーパーコンピュータ等を使用する場合と比較して、GPUボードを用いた環境整備は導入コストが低い。
2　既存システムに対し、GPUボードを追加することで環境を構築することが可能であり、システム構成の変更による影響を抑えることができる。

3. GPUのアーキテクチャにおける特性

本章では、GPUのアーキテクチャにおける特性について、CPUとの差異を含め、説明する。

3.1　GPUとGPUにおけるコア数とキャッシュサイズの差異

CPUとGPUでは、想定している処理方法が異なるため、コア数とキャッシュサイズが大きく異なる（表2参照）。

CPUは、複数のデータを組み合わせて逐次処理・分岐処理・繰り返し処理を実行するため、ランダムにメモリへアクセスしても十分にキャッシング可能な大容量のキャッシュを搭載しており、メモリへのアクセスレイテンシを軽減できている。また、CPUは処理を効率的に実行するために、パイプライン処理や分岐予測、アウト・オブ・オーダ実行等を実施している。そのため、コアあたりの性能は高いが、コアの物理的なサイズが大きいため、搭載数が少なくなっている（図6参照）。

一方、GPUは画像処理を専門としており、大量のデータを同時かつ並列に演算処理できる。画像処理の特性上、メモリへシーケンシャルにアクセスするため、キャッシュサイズは小さい。また、1つのコアで大量のデータを同時に演算処理することで、アクセスレイテンシを隠蔽している。さらに、GPUは処理の特性上、条件分岐が無く、命令の実行順序を複雑化しないためにコアを簡素化し、物理的なサイズが小さくなっている。そのため、CPUと比較して多数のコアを搭載することができており、並列度を上げ、高スループットを実現している。

CPUと同様に、GPGPUにおいても1コアあたり1スレッドを動作させると、メモリへのアクセスレイテンシを隠蔽できず、速度性能をCPUよりさらに悪化させてしまう場合がある（図7①参照）。GPGPUで速度性能を向上させるためには、1コアあたり十分な数のスレッドを割り当て、メモリへのアクセスレイテンシを隠蔽させる必要がある（図7②参照）。

表2　CPUとGPUのハードウェアの違い[1][2]

プロセッサ名	コア数	キャッシュサイズ [MB]
CPU Intel Xeon E5-2690 v4	14	35.0
GPU NVIDIA Tesla K40c	2,880	1.5

図6　CPUによる演算

3.2　GPUに搭載されているメモリの特性

GPUには、用途ごとに特性の異なるメモリが搭載されており、それぞれアクセス速度が異なる（図8、表3、表4参照）。処理の特性に応じて、メモリを使い分けることでGPUの高い処理性能を活かすことが可能となる。

3.3　GPGPUにおけるメモリアクセス

GPUのアーキテクチャにおける特性により、GPGPUにおいてデバイスメモリに対するランダムなアクセス

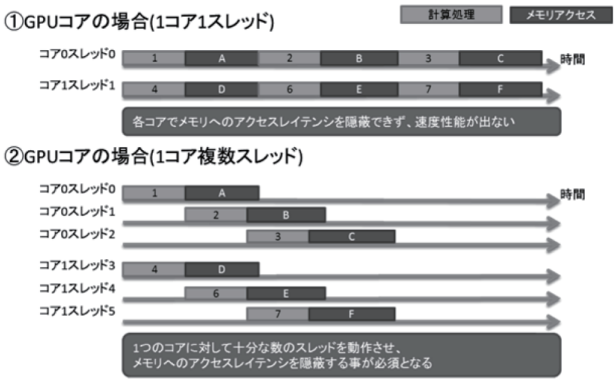

図7　GPUによる演算

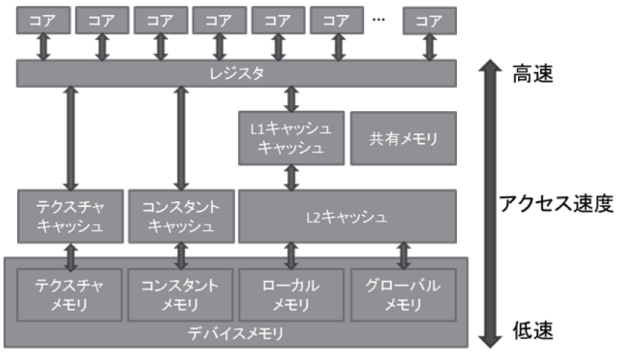

図8　GPUのメモリ配置とアクセスパス

表3　NVIDIA社製GPUに搭載されているメモリの特性(1/2)[2]

メモリ	特性
レジスタ	コアに最も近く、搭載されているメモリ内で最も高速にアクセスできる。GPGPU において、カーネル関数で使用するローカル変数に使用される。スレッドごとに独立しており共有できない。また、使用可能なレジスタ数を超過する場合、デバイスメモリ上に保持されるため、アクセスレイテンシへの対策が必要となる。

表４　NVIDIA社製GPUに搭載されているメモリの特性(2/2)[2]

メモリ	特性
共有メモリ	SM(*5)内に存在し、レジスタほどではないが高速な読み書きが可能である。同一SM内のスレッド間でデータを共有する。メモリへランダムにアクセスする場合、本メモリを使用することでアクセス回数増加による性能劣化を回避することが可能となる。
テクスチャメモリ	GPUからは読み込み専用であり、専用のロードユニット（テクスチャユニット）を経由してGPU全体からアクセスが可能である。また、専用のキャッシュによってグローバルメモリより高速にアクセス可能である。
コンスタントメモリ	GPUからは読み込み専用であり、専用のキャッシュ（最大64KB）にデータが格納されるため、GPU全体から高速なアクセスが可能である。
ローカルメモリ	使用するレジスタ数がGPUに搭載されたレジスタ数を超過する場合、一時的に超過分をローカルメモリとして保持する。
グローバルメモリ	CPUとのデータ転送に使用、および、GPU全体からアクセス可能である。容量は非常に大きいが、アクセスレイテンシが大きくGPUに搭載されるメモリとしては最も低速である。

は、アクセス回数が増加することにより、速度性能が劣化する。そのため、GPGPUでは、使用するデータの特性に合わせて、メモリを選択する必要がある。

4. GPGPUを用いたソフトウェアの高速化手法

GPGPU化を実現するための手法は、大きく以下の2種類に分類できる。次節以降で、各手法の特徴を示す。

1　CUDA(*6)を使用し、GPU用の処理を記述する手法
2　既存のC、または、Fortran言語のソースコードに対し、ディレクティブベースでGPU用の処理を生成するOpenACC(*7)を用いる手法

4.1　CUDAを用いたGPGPU化

CUDAは、NVIDIA社が製造するGPUで動作するGPGPU環境であり、コンパイラやライブラリから構成されている。NVIDIA社製GPUのアーキテクチャを十分に理解することで、最大限に最適化し、効率的に高速化を実現可能である。表５にCUDAによるGPGPU化の特徴を示す。

4.2　OpenACCを用いたGPGPU化

OpenACCは、AMD社・Cray社・NVIDIA社に代表されるメーカによって標準化されたアクセラレータ向けのプログラミングインタフェースである。表６にOpenACCによるGPGPU化の特徴を示す。

表５　CUDAによるGPGPU化の特徴

観点	特徴	評価
開発環境	NVIDIA社製のGPUが搭載されており、VisualStudioがインストールされていることを前提として、NVIDIA社から提供されているインストーラを用いて、統合開発環境をインストール可能である。	○
開発効率	独自言語CUDAを新たに覚える必要がある。使用するGPUの特性を考慮し、ソフトウェアを再設計する必要があるため、GPUコードの作成に時間を要する。	△
性能	GPUの特性を考慮したチューニングが可能なため、効果的・効率的にGPUを使用できる。	◎
互換性	NVIDIA社が製造するGPU以外では動作しない。	×

表６　OpenACCによるGPGPU化の特徴

観点	特徴	評価
開発環境	2016年現在、対応する統合開発環境が無い。OpenACCを使用するためには、それに対応したコンパイル環境を構築する必要がある。	△
開発効率	既存のソースコードを変更することなく、ディレクティブベースでGPGPU化できるため、開発効率が良い。	○
性能	ユーザが明示的に共有メモリを使用できず、効率的に高速化できない。また、複数GPUに対応していない等の制約がある。	×
互換性	複数メーカのGPUを利用可能であるが、使用するコンパイラに依存する。例えばPGIコンパイラの場合、NVIDIA社製とAMD社製のGPUに対応している。	○

4.3　本報告で使用したGPGPU化を実現するための手法

今回、GPGPU化の対象としたソフトウェアは、メモリバンド幅がボトルネックとなっている、かつ、メモリへランダムにアクセスしていることから、グローバルメモリへのアクセス回数を削減することが課題となることが明確であった。

そのため、共有メモリを使用してアクセス回数を削減する必要があるが、OpenACCでは共有メモリを明示的に使用できず高速化が望めない。一方、CUDAでは、共有メモリを使用してグローバルメモリへのアクセスを削減できるため、最適化による高速化が実現できると判断した。

5. CUDAを用いたGPGPUによる高速化事例

本章では、CUDAを用いてGPGPUにより高速化した以下の2例を紹介する。

1　処理時間の短縮、および、精度向上を目的とした粒子線線量計算エンジンのGPGPU化

2　スループットの向上を目的とした非線形画像位置合わせソフトウェアのGPGPU化

5.1　粒子線線量計算エンジンのGPGPU化

⑴　粒子線線量計算エンジン

粒子線治療装置において、標的に粒子線を照射した時の患者体内における線量値を、3次元分布としてシミュレーション計算するソフトウェアである[2]。

当社が開発に従事する本エンジンは、照射機器の設定値を計算する処理と粒子線線量計算を行う処理から成る。粒子線ビーム全体を局所的なビーム（ペンシルビーム）の集合体として表現し、3次元の計算グリッド上で線量値計算を行う（図９参照）。なお、並列化の対象は、その内、粒子線線量計算を構成する輸送計算と散乱計算とする。

⑵　輸送計算のGPGPU化

輸送計算において、計算すべきペンシルビームの総数は数千個である。CPUの場合、コアに対してペンシルビーム単位で並列化したが、最大8並列が限界であった。

一方、GPUの場合、メモリへのアクセスレイテンシを隠蔽するため、1コアあたり複数スレッドを割り当てる必要がある。しかし、数千個のGPUコアに対し数千個のペンシルビームでは、コアに割り当てるペンシルビーム数が十分ではなく、アクセスレイテンシを隠蔽できない。そのため、ペンシルビームの輸送経路によらず、輸送距離のみに依存するパラメータを使用した並列計算を実施することで数万スレッドでの並列処理を可能とし、GPGPUによる高速化を実現した。図10、表７にスレッド粒度の概念と生成スレッド数の変化を示す。

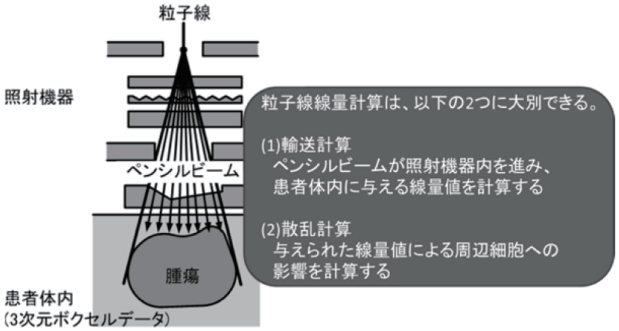

図９　粒子線線量計算エンジンの処理

⑶　散乱計算のGPGPU化

散乱計算では、ガウス分布にしたがって、散乱元となる1つの線量値を「散乱後の線量値」として周辺に加算していくため、メモリへのアクセス回数が多くなり、さらに、散乱範囲としてメモリへのアクセスはランダムとなる。CPUはキャッシュサイズが十分に大きいため、キャッシュヒットによりアクセス回数を削減できる。

一方、GPUはキャッシュサイズが小さくキャッシュヒットによるアクセス回数の削減は期待できない。そのため、散乱計算中のデータを共有メモリ上に保持するようCUDAでプログラミングし、デバイスメモリへのアクセス回数を削減することでGPGPUによる高速化を実現した（図11参照）。

⑷　GPGPUによる高速化効果

輸送計算と散乱計算において、GPGPU化する前のCPUのみの処理に対して、GPGPU化することによって、速度性能を6.7倍に高速化した。また、GPUボードを増設することで、さらなる高速化を確認できている（図12参照）。

5.2　非線形画像位置合わせソフトウェアのGPGPU化

⑴　非線形画像位置合わせソフトウェア

2つの画像間において発生した「変形」を検出し、画像を非線形に変形することで、画像に含まれる物体の位置合わせを行うソフトウェアである。

当社が開発に従事する非線形画像位置合わせソフトウェア[4]は、テンプレート画像として過去画像を格子状

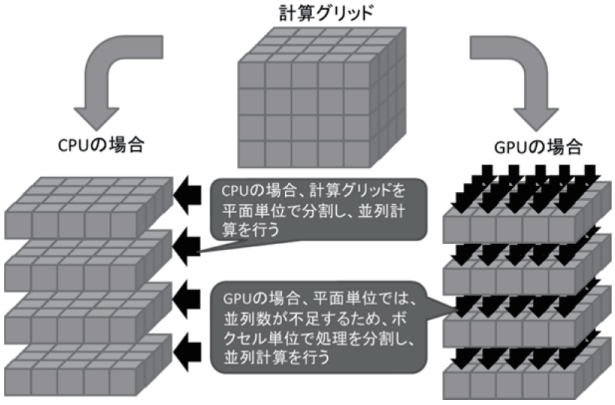

図10　輸送計算のGPGPU化

表７　並列化の比較

	CPU	GPU
並列化方法	1平面を 1スレッドで計算する	1ボクセルを 1スレッドで計算する
並列数	平面数分 1mmピッチの場合、 数100並列に分割する	ボクセル数分 1mmピッチの場合、 5.6万並列以上分割する
コアに割り 当てる計算	1コアあたり 1スレッドで計算する	1コアあたり 2000スレッド以上を計算する

の矩形領域に分割し、全てのテンプレート画像と現在画像でテンプレートマッチングすることで矩形領域ごとに過去画像から現在画像への動きベクトルを取得する。この動きベクトルから過去画像の各画素における移動量を算出しワーピング画像を作成する（図13参照 ）。

⑵　テンプレートマッチングのGPGPU化

テンプレートマッチングは、1組の画像あたり200～300回のマッチング処理が動作し計算量が多いため、必然的に高速化が必要である。

本処理をCPUで高速化する場合、CPUの実装コア数に応じたスレッド並列化が現実的である。同様に、GPGPUで高速化する場合、GPUに搭載されている数千個のコアに対し生成するスレッド数は不足しているため効率的に高速化できない。そのため、複数組の過去画像と現在画像を同時に処理することで、GPGPU化によるスループットを向上した（図14参照）。

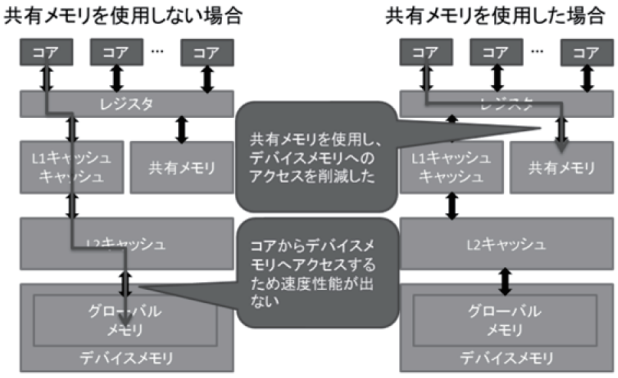

図11　散乱計算のGPGPU化

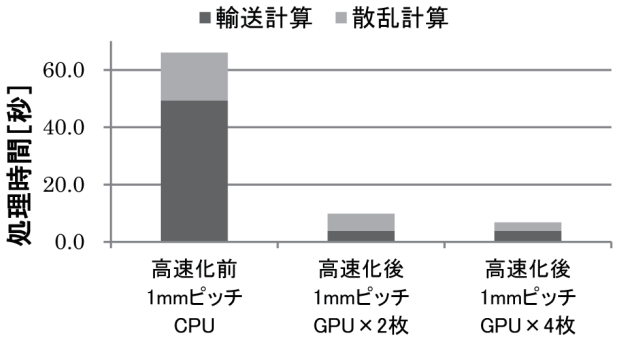

図12　GPGPUによる高速化成果

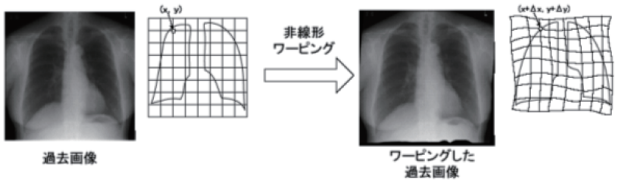

図13　非線形ワーピング処理

⑶　ワーピング画像作成のGPGPU化

ワーピング画像は、各画素位置から対応する元画像の画素位置を計算し作成するため、計算量は作成するワーピング画像の画素数に比例する。

CPUでは、各画素を順に計算しワーピング画像を生成していたが、GPGPUでは、全画素を並列に計算することで、ワーピング画像作成の時間を削減した（図15参照）。

6. GPGPU化のキーポイント

本章では、粒子線線量計算エンジン、および、非線形画像位置合わせソフトウェアのGPGPU化を実施する過程で得た設計上のキーポイントを一部紹介する。

6.1　CPU-GPU間のデータ転送にかかるオーバヘッドの考慮

GPGPUでは、処理対象となるデータをCPUからGPUへ転送、および、処理結果のデータをGPUからCPUへ転送する必要があり、転送するデータサイズ分のオーバヘッドが発生する。

そのため、CPUにおいて数百msで実現している処理をGPGPU化しても、CPUとGPU間のデータ転送時間が要因となり、GPGPU化による処理時間削減の効果は得られない。その場合、対象の処理時間短縮ではなく、処理のスループット向上によるシステム全体の高速化を設計する必要がある。

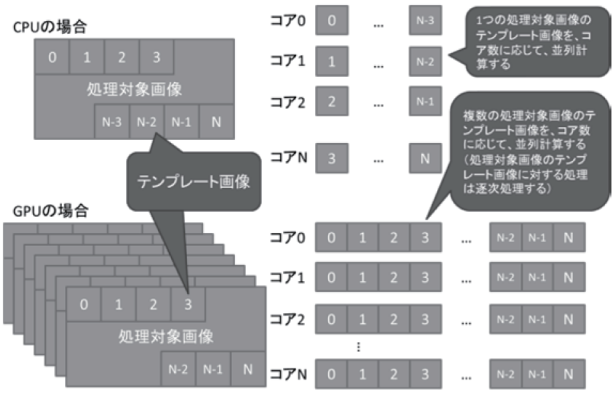

図14　テンプレートマッチング処理の並列化

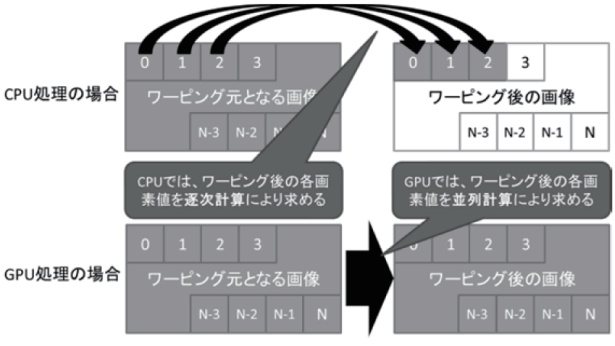

図15　ワーピング画像作成処理の概要

6.2　GPUのアーキテクチャを考慮した並列化粒度の設計

3.1節に述べた通り、GPUのアーキテクチャ特性上、メモリへのアクセスレイテンシを隠蔽するためにはコア数以上のスレッドを動作させる必要がある。そのため、CPUでは「スレッド数＝コア数」と設計していたのに対し、GPUでは「1つのコアあたり多数のスレッド」が動作するように処理の並列化粒度を設計する必要がある。

6.3　メモリへのアクセスパスを考慮した使用メモリの設計

GPUには、複数の異なる特性を持ったメモリが搭載されている。これらのメモリを有効利用することで、メモリへのアクセス回数を削減、メモリへのアクセスレイテンシを軽減し、GPGPU化による処理時間を短縮できる。

⑴　共有メモリの使用

GPGPU化する処理において、メモリへのアクセス回数が多い場合、または、メモリへランダムにアクセスする場合、共有メモリを使用し、デバイスメモリへのアクセス回数を削減する必要がある。図16に示す通り、GPUのSM内に搭載されている共有メモリは、デバイスメモリへのアクセスより、アクセスレイテンシが小さいため、高速化に大きく貢献する。

⑵　テクスチャメモリの使用

GPUにはメモリアクセス命令を実行するユニットが2つある。デバイスメモリへアクセスするユニットとテクスチャメモリへアクセスするユニットである。これらのユニットは独立して動作させることが可能であり、メモリアクセス命令の実行密度を上げることが可能である（図17参照）。

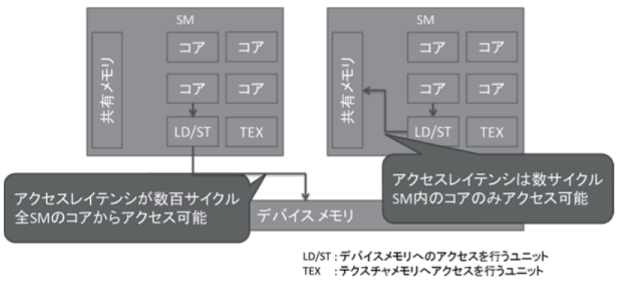

図16　共有メモリへのアクセス

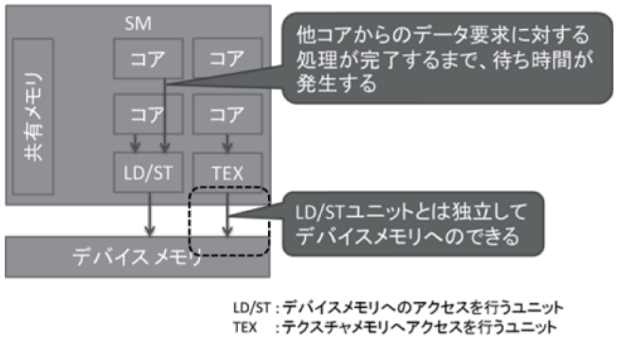

図17　メモリアクセスパス

6.4　複数のGPUボードを使用したスケールアウト

CUDAは、複数のGPUボードを同時に使用して高速化対象とする処理を分割し振り分けることが可能である。

ただし、複数のGPUボードを使用する場合、各GPUの制御（データ転送・同期処理等）を設計する必要がある。

7. むすび

本報告では、処理をCPUからGPUへオフロードするGPGPUによるソフトウェア高速化の手法と、GPGPUを実現する手法としてCUDAによるGPGPU化について述べた。また、各手法を適用し高速化したケース2例を紹介した。なお、粒子線線量計算エンジン（輸送計算と散乱計算）を高速化することによって、計算グリッドの間隔を半分（3次元分布のためデータ量は8倍、演算特性により演算量は32倍）としても、CPUのみの場合と同等の時間で処理を実行でき、高精度なシミュレーションを実現できた。

今後も継続して、これら手法を用いた技術ノウハウを獲得していくとともに、ドメインで求められる技術を追求しユーザの要求に応え続けるソフトウェア開発、および、ソフトウェア高速化に取り組んでいく所存である。

＊1　Graphics Processing Unit、3次元のコンピュータグラフィックスに必要な画像処理、および、画面表示を行うプロセッサ。

＊2　Single Instruction Multiple Data、1つ命令で複数のデータ列に対して処理を行う演算名の総称、Intel社製CPUでは、SSE®、AVX®がある。

＊3　Floating-point Operations Per Second、コンピュータの性能指標の1つで、1秒間あたりの浮動小数点演算回数を示す。本報告では、1クロックで実行可能なベクトル命令の数×動作クロック数×コア数より算出した。

＊4　General-Purpose computing on Graphics Processing Units、GPUの演算資源を画像処理以外の目的に応用する技術。

＊5　Streaming Multiprocessor、NVIDIA社製GPUにおいて、演算コアのグループを示す。

＊6　Compute Unified Device Architecture は、NVIDIA社が提供するGPU向けのGPGPU開発環境である。

＊7　OpenACCは、ディレクティブベースでGPUプログラム委具を可能とする標準規格である。

参考文献

⑴　Intel 製品仕様：http://ark.intel.com/

⑵　NVIDIA：http://www.nvidia.com/

⑶　粒子線用線量計算エンジンの開発、MSS技報 Vol.21 2010年度発行

⑷　胸部X線画像診断支援システムの開発、MSS技報 Vol.13 2001年度発行

執筆者紹介

大田 弘樹
2010年入社。関西事業部第五技術部所属。カーマルチメディア開発、医用画像システムのソフトウェア開発に従事。

馬場 明子
2005年入社。関西事業部第五技術部所属。カーマルチメディア開発、医用画像システムのソフトウェア開発に従事。

下田 雄一
2015年入社。関西事業部第五技術部所属。医用画像システムのソフトウェア開発に従事。

安田 隆洋
2014年入社。関西事業部第五技術部所属。医用画像システムのソフトウェア開発に従事。

山本 啓二
1992年入社。関西事業部第五技術部所属。数値シミュレーションや顔認識システム、医用画像システム、医用画像処理などの各種アルゴリズム開発及びソフトウェア開発に従事。