「並列分散処理による衛星画像データ処理の高速化」

佐藤裕幸（ソフトウェア情報学部、教授）、
前原秀明（三菱電機株式会社、主席研究員）、後町将人（三菱電機株式会社、研究員）

＜要旨＞
災害や脅威への対策など迅速な対応を目的に、衛星、航空機、ヘリ等から空撮した都市部の映像を対象に、映像から３次元形状への復元処理を行う面積相関法プログラムの高速化、CPU及びグラフィックス用プロセッサGPUによる並列化を行った。当初の単一CPUコアで動作していたプログラムに対して、４つのGPUを用いて、771倍（87秒→113ミリ秒）の高速化を達成した。

1　研究の概要

コンピュータによる処理の高速化、装置の小型化への要求は留まることはない。その要求に応えるために、近年のコンピュータアーキテクチャ、特にプロセッサのアーキテクチャはマルチコア化や、メモリの多階層化など、複雑で多種多様となってきている。そのため、アプリケーション実行において、その構造を理解して、それに適した実行制御方式を適用しなければ、プロセッサの本来の性能を最大限に引き出すことが出来ないという課題がある。

また近年、地震や津波などの災害予測・被害状況把握、地球温暖化などの環境問題、天然資源探索、諸外国からの脅威監視などの地球観測に対する重要性が高まってきている。地球観測は人工衛星や航空機等により地表を撮像することで行われるが、撮像データをそのまま利用者に提供するのではなく、利用目的に応じたデータの加工（画像処理）が必要となる。この撮像データは観測領域の広域化・高分解能化により大規模になっているが、災害や脅威への対策など迅速な対応が必要とされる分野が多くなっており、撮像データに対する画像処理の高速化が大きな課題となっている。

以上の課題と災害時の被災状況の把握等に重要と成ることから、衛星、航空機、ヘリ等から空撮した都市部の映像を対象に、映像から3次元形状への復元処理の並列分散処理による高速化を行った。

映像から３次元形状への復元処理では、映像中の連続するフレーム画像の中から、処理対象として特定の２フレームを選択し、ステレオ画像処理[1]を用いて、フレーム間の対応関係から距離情報を計算して立体化を行う。しかしながら、ステレオ画像処理は、処理対象のフレーム量とフレームサイズに応じた高負荷な処理となるため、リアルタイム性能を実現するためには、桁違いの高速化が必要不可欠となる。また精度向上のためには、複数フレーム間での処理が必要となり、更に高速化が必須となる。

この課題に対処するため、我々は、グラフィックス用プロセッサGPUを用いて高速化に取り組んだ。GPUは、CPUとは異なり、CGモデリングやコンピュータグラフィックス等のグラフィックス用途に最適化されたプロセッサであり、大規模な数の単純な演算器を搭載することで、CPUを大きく越えた理論性能を備えている。すなわち、このGPUアーキテクチャの特徴に適合した演算アルゴリズムを実装することができれば、CPUで扱ってきた汎用的な演算も、GPUを利用して高速に処理することができる[2]。このため、このGPUに適合した演算法の

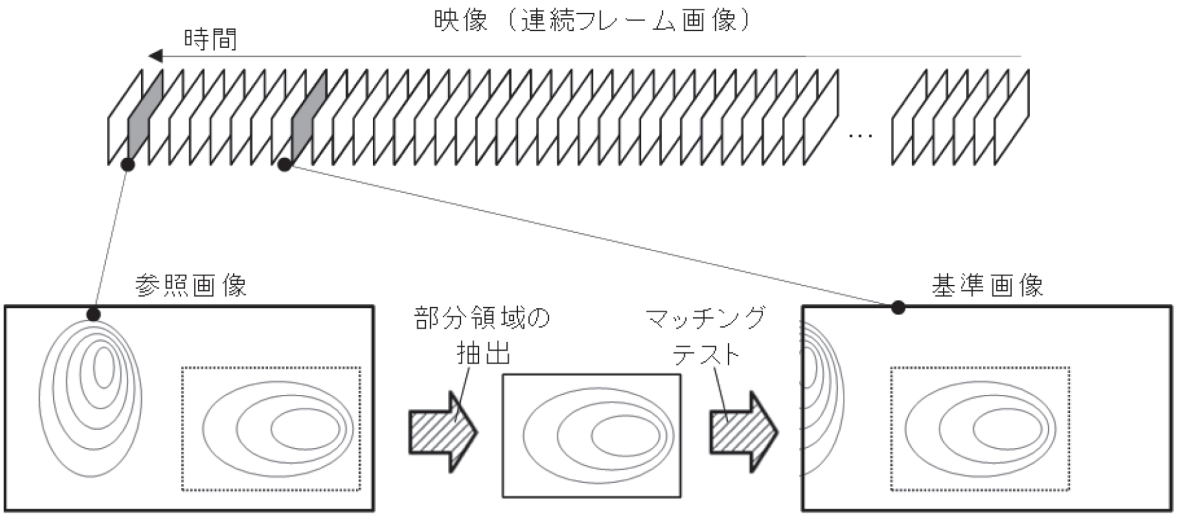

図１　ステレオマッチング処理の概要

研究開発が、近年、盛んに取り組まれており、画像処理[3]、気象予報[4]、暗号化処理[5]など、様々な分野で、実績が報告されている。

本報告書では、ステレオ画像処理アルゴリズムの１つである面積相関法[6]を実装した基本的なプログラムを対象とした高速化の方法と効果を示す。

2　研究の内容

異なる２種類の画像データ間の対応を求める処理（図）をステレオマッチングと呼ぶ。本章では、このステレオマッチングを実現するための手法として高速化の対象とした面積相関法について簡単に触れる。

2.1 面積相関法

面積相関法は、画像間の相関値を類似度として、同一領域の探索を行う。この相関値が最大となる点が画像間の対応点となる。数式等の詳細については、文献[6]等を用いて調べることができる。

ステレオ画像処理のその他の代表的な手法であるDP（Dynamic Programming）マッチング法[7]では、基準画像の１画素に対して参照画像の複数画素を探索している。一方、面積相関法では、基準画像の複数画像に対して参照画像の複数画素を探索しているため、面積相関法の方が演算量が高い。

2.2 高速化の関連研究事例

２次元パターン同士のDPマッチングを効率的に計算する方法[7]や、GPUを用いたステレオマッチングの高速化[9][10][11]は、実績が数多く報告されている。特に[11]では、複数のマルチコアプロセッサやメニーコアプロセッサであるGPUを対象として、それぞれに適した高速化を行っている。

3　これまで得られた研究の成果

本章では、面積相関法に対する高速化方法とCPUおよびGPUによる並列計算方法、そして、これらの高速化効果を示す。

3.1 実行環境

表１は、性能測定に使用したプロセッサとコンパイラの環境である。また、図２に、これらのプロセッサ間の関係を示す。本性能測定では、１つのグラフィックスカードに２つのGPUを搭載したモデルを計２枚、すなわち最大４GPUを利用した。CPUによるプログラムは、最大12スレッドを使用して並列に動作させた。なお、改良前のプログラムの実装にはCを使用していたが、プログラムの改良にあたり、Cの拡張形式でGPUを利用できる開発環境CUDA（Compute Unified Device Architecture）を利用した。

3.2 高速化の概要

GPU上で幾つかの最適化（改良）を行った。

表１　プロセッサ構成

	CPU	Graphics Card
Vender	Intel	NVIDIA
Model Name	Corei7-3930k	Geforce GTX690
Launch Date	2011.Q4	2012.Q2
# of Cores	6(12 Threads*2)	1536(192×8 SM*1)×2GPU
Core Clock	3.20-3.80 GHz*3	1.02 GHz
Theoretical Performance	153.6-182.0 GFLOPS*4	2,810.9 GFLOPS×2GPU
Compiler	GNU Compiler Collections(Ver.4.6.2)	CUDA(Ver.5.0)
Optimize Option	"-O3"	"-O3"
	(SIMD演算命令SSE4.2 AVXは無効)	"-prec-sqrt=FALSE"*5
		"-maxrregcount32"*6

*1：SM: Streaming Multiprocessor，グループ化されているGPUコアの塊の単位．上記のGPUの場合，192コア/ SMの構成．
*2：Intelハイパースレッディングテクノロジで，コア毎に，命令パイプラインに最大2スレッド分の命令を同時にスケジュールする．
*3：Intelターボブーストテクノロジで，負荷に応じて，コアのクロックが変化する．
*4：FLOPS，1行間に計算できる浮動小数点演算回数．GFLOPSは，Giga FLOPS．すなわち，単位を10^9回とした表記．
*5：-prec-sqrt=false，mathライブラリのsqrt関数の精度を落とし，演算速度を向上させるオプション．
*6：-maxrregcount 32，使用するレジスタサイズを制限する．チューニング用オプション．

以下に、プログラムを高速化するために実施した改良作業をステップ毎に示す。

① 画素数308×390を分割した並列計算を行うために、中間結果を格納する配列WinBの領域を確保する位置をループの内側に移動した。すなわち、配列WinBの並列化対応を実施している。

② ほとんど全ての演算を単精度に修正した。倍精度で演算しているのは、２箇所のみである。全ての演算を単精度に修正した場合は、演算結果が、X、Y方向それぞれ２か所ずつ異なる値（差異は１）となってしまう。このため、今回は改良前と結果が等しくなる範囲で、演算を単精度に変更している。なお、前述程度の違いなので、実際には全てを単精度で演算しても問題はない。

更に、openMPライブラリを使用して並列化した。並列性は画素単位であるが、CPUではそれでは多すぎるため、演算範囲をY方向（390並列）で分割した並列処理を実装した。

③ ①のプログラムをそのままGPUへ移植した。並列性を最大限に活かすため、計算領域の各画素（308×390並列）の処理を各スレッドに割付けている。

④ 計算領域に加えて、マージン領域も含めてGPUスレッドに割り付けるよう③を改良した。スレッド数をX：320×Y：408として、GPUが得意とするスレッド数（X方向が32の倍数）のパターンを設定した。

⑤ ほとんど全ての演算を単精度に修正した。CPU向けの改良であった②と同様である。今回使用したグラフィックスカードGeforce GTX690は、倍精度演算性能に比べて単精度演算性能が非常に高いモデルである。このため、本修正を施すことで、演算時間を大幅に短縮することができる。

⑥ 画像データをGPUのシェアードメモリにコピーして処理するように改良した。シェアードメモリは、スレッドブロック内の各スレッドからアクセスできるアクセス遅延の小さいメモリである。本処理では、同じ画素への再アクセスが頻繁に発生するため、低遅延のシェアードメモリにデータをわざわざコピーしてから演算しても、高速化の効果が高い。面積相関法では、担当画素の周辺のデータにアクセスするため、スレッドブロック内の四方端のスレッドは、計算領域だけでなく、外側のマージン領域もシェアードメモリにコピーする必要がある。

⑦ 各スレッドの使用レジスタ数を32に制限するようコンパイラの最適化オプションを追加した。このオプション指定がない場合、１スレッド当たり34個のレジスタを使用していた。GPUでは、SM（Streaming Multiprocessor）内で複数のスレッドブロックが時分割で平行して実行される。幾つのスレッドブロックが実

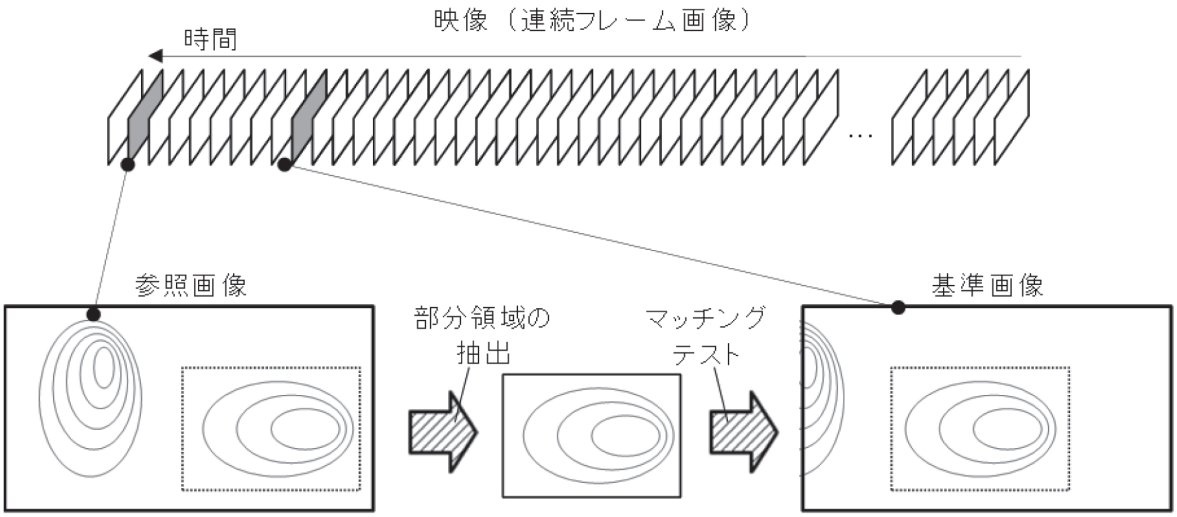

図２　実験に使用した計算機環境

行されるかは、SMに搭載されたレジスタ数とスレッドブロック（すなわち各スレッド）が使用するレジスタ数の関係で決まり、各スレッドの使用スレッド数が少ないほど、多くのスレッドブロックが平行実行される。従って、本オプションを指定して使用レジスタ数を32に制限することで、並行動作数を向上させている。

⑧　スレッド毎に単一の画素を計算する方式から、多数の画素を計算する方式へと修正した。これにより、並列粒度の調整が可能となり、各スレッドの最適な担当画素数を調整できるようになる。

3.3 高速化結果

表２に、単一画像データに対する実行時間を示す。表３は、32枚の画像データをまとめて処理した場合の実行時間である。面積相関法は、並列性が非常に高く、改良前のCPUプログラム１スレッド実行と比較して、GPU４枚を使用することで計771倍の高速化を達成した。なお、⑧で、並列粒度を調整可能にする修正を施した

表２　単一画像データに対する実行時間（単位：ミリ秒）

	改良前	改良後													
		①	②							③	④	⑤	⑥	⑦	⑧
		# of CPU Threads								1GPU					
	1	1	1	2	4	6	8	10	12						
実行時間	2,721	2,740	2,105	1,071	569	380	331	270	228	76	73	30	20	15	16
改良前比	1.0	1.0	1.3	2.5	4.8	7.2	8.2	10.1	11.9	35.8	37.3	90.7	136.1	181.4	170.1
対②比率	—	—	1.0	2.0	3.7	5.5	6.4	7.8	9.2	27.7	28.8	70.2	105.3	140.3	131.6

表３　32画像データに対する実行時間（単位：ミリ秒）

	改良前	改良後				
		②		⑦		
		# of CPU Threads		# of GPUs		
スレッド/PE数	1	1	12	1	2	4
実行時間	87,072	67,027	7,252	403	202	113
改良前比	1.0	1.3	12.0	216.1	431.0	770.5
対②比率	—	1.0	9.2	166.3	331.8	593.2
対②12比率	—	—	1.0	18.0	35.9	64.2
対1GPU比率	—	—	—	1.0	2.0	3.6

が、これによる高速化の効果は得られなかった。これは、修正前にはなかったループ処理のオーバヘッドが表れているものと思われ、１スレッドが１画素を担当するのが最適であることが分かった。

本稿では、ステレオ画像処理の１つである面積相関法のプログラムの高速化方法、CPUおよびGPUによる並列化方法、そして、これらの高速化の効果について示した。結果として改良前のプログラム１スレッド実行と比較して、32画像を同時処理する場合、４GPUを使用して計771倍の高速化（87秒→113ミリ秒）を達成した。

4　今後の具体的な展開

CPU、GPU共に以下に示すような高速化の余地が残されている。

- CPU：演算性能を高めるためにベクトル演算命令を発行する。Intel Compilerを用いて自動ベクトル化に関する最適化オプションの追加や、ディレクティブによるベクトル化補助などを行うことで、数倍程度の高速化が期待できる。
- GPU：グラフィックス処理に最適化されたGPUならではのキャッシュメモリ（テクスチャキャッシュ）を使用する。これは、２次元的なキャッシュメモリとして利用できる機構のため、画像処理を中心に、GPUプログラムのチューニング手段として利用されることが多い。本稿で高速化の対象としたステレオマッチング処理は、２次元配列中の同一データへ頻繁に再アクセスするため、このテクスチャキャッシュを用いた高速化が期待できる。

上記手法の適用や、DPマッチングなどの他のステレオ画像処理アルゴリズムの高速化、従来技術にない並列計算法の開発を今後の課題とする。

5　論文・学会発表等の実績

なし。

6　受賞・特許

なし。

7　その他

当初の目標であったマルチコアプロセッサ単体に対して2.8倍の高速化に関しては、64.2倍（表３より）であり、目標を充分に達成している。また、単一ノードでの最速の実行時間が0.1秒程度であり、目標も充分に達成していることから、今回は複数ノードでの評価は実施しなかった。

参考文献

[1] 津邑公暁，清水遊歩，中島康彦，五島正裕，森眞一郎，北村俊明，富田眞治：ステレオ画像処理を用いた曖昧再利用の評価，情報処理学会論文誌コンピューティングシステム，Vol.～44，No.～11，2003.

[2] Kirk, D. B., and Hwu, W. W.：*Programming Massively Parallel Processors: A Hands-on Approach*（*Applications of GPU Computing Series*），Morgan Kaufmann, 2010.

[3] Bhattacharya, C., Somawanshi, P., Khadtare, M., and Karandikar, S.K.：Accelerated SAR Image Generation on GPGPU Platform, in *Proc 2011 3rd International Synthetic Aperture Radar*（*APSAR*），pp.1-4，2011.

[4] Shimokawabe, T., Aoki, T., Muroi, C., Kawano, K., Endo, T., Nukada, A., Maruyama, N., and Matsuoka, S.：An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code, in *Proc. 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society, pp~1-11，2010.

[5] 西川尚紀，岩井啓輔，黒川恭一：CUDA環境における共通鍵ブロック暗号の高速実装，電子情報通信学会技術研究報告，Vol.~111，No.~163，pp.~25-30，2011.

[6] Sutton, M. A., Orteu, J. J., and Schreier, H. W.: *Image Correlation for Shape, Motion and Deformation Measurements*, Springer，2009.

[7] 内田誠一：DPマッチング概説－基本と様々な拡張－，信学技報，PRMU2006-

166，2006.

[8] Yaguchi, Y., Iseki, K., and Oka, R. : Full Pixel Matching between Images for Non-linear Registration of Objects, *IPSJ Trans. Computer Vision and Applications* (CVA), Vol.~2, pp.~1-14, 2010.

[9] Congote, J., Barandiaran, J., Barandiaran, I., and Ruiz, O. : Realtime Dense Stereo Matching with Dynamic Programming in CUDA, in *Proc. CEIG '09*, 2009.

[10] Sah, S., and Jotwani, N. : Stereo Matching using Multi-Resolution Images on CUDA, *International Journal of Computer Applications*, Vol.56, No.12, 2012.

[11] 岩田健司，中村良介，田中良夫，増田知記，町田亮介，小島功，関口智嗣：異なるアーキテクチャのメニーコアプロセッサにおけるステレオマッチングプログラムの高速化と性能評価, 情報処理学会論文誌コンピューティングシステム, Vol. 3, No. 3, pp.178-1188, 2010.