高速度ビデオでの運動解析プログラムの開発
― CPU および GPU によるトレース高速化の検討 ―

横山直樹（東海大学・総合科学技術研究所）

Development of Versatile Motion Analysis Program(VMAP)
– Even more parallel processing of correlation with CPU and/or GPU -

Naoki YOKOYAMA (Research Institute of Science and Technology, Tokai University)

キーワード：運動解析、相関計算、高速度ビデオグラフィ、GPGPU、高速化
Keywords: Motion analysis, Template Matching, High-speed videography, Latest GPU

Motion analysis program for images acquired by high-speed videography was developed and tested so far. In the case of actual analysis scene, particle or object density can be very high. So some scheme of distributed computation will be required for practical use of this program. This time CPU and GPU were adapted to perform cross-correlation calculation. Using GPU as calculation device, processing time can be reduced drastically with CUDA. But CUDA codes can be difficult to modify for even more higher performance, CPU is much easier for that purpose using OpenMP.

1. はじめに

前報までで高速度ビデオシステムによって記録された動画を対象とし、対象物体の運動を自動解析するプログラムを開発し、その有効性を実験的に検証してきた。特に時間分解能を高めた高速度ビデオシステムにおいては、空間分解能に制限があり、各フレームにおける対象物体の位置の計測精度が低くなりがちであるが、画像相関値を評価関数とし、その極大値が得られる位置を物体位置として把握するアプローチでは、位置情報をサブピクセルの単位で求めることが可能であることを示した。しかし正規化相関を用いたアプローチでは、その計算量が膨大になるために、実用的にはなんらかの計算負荷軽減が必要である。このために、CPU の多重コア利用による並列化、ネットワーク分散処理などを試みてきたが、最近多用されるようになってきた GPU を活用する方法が有効であることを示した。本論文では CPU を用いた場合の並列処理を再検討し、また GPU の利用をさらに効率化する試みについて述べる。

2. VMAP(Versatile Motion Analysis Program for ultra high-speed videography)

物体の着目領域である ROI(Region of Interest)は、フレーム毎に相互相関値の極大点を探索することで追跡できる。この場合の相互相関値(Zero-mean Normalized Cross-Correlation: ZNCC)は次の式で与えられる。

$$\frac{\sum (f_i - \bar{f}) \times (g_j - \bar{g})}{\sqrt{\sum (f_i - \bar{f})^2} \times \sqrt{\sum (g_i - \bar{g})^2}}$$

ここで f_i と g_j はそれぞれ ROI と探索対象画像上にとったその対応部分上の画素を示す。$\overline{f}$ と $\overline{g}$ は対象領域にわたって計算された画素の輝度の平均値である。対象物体の運動部位については、基本的に剛体運動を仮定している。ただし純粋な並進運動だけでなく、物体内部および外部の基準点のまわりの回転運動にも対応できる。干渉像における干渉縞の広がりのような場合には、対象の運動が剛体運動ではないが、この場合は手動で解析するなどの方法で対応できる。この式に従って計算する場合、まず平均値を求めるために、対象領域の全画素の輝度値を拾い、その総和を求め、それを画素数で割り、次にこの平均値との差の積和を求めるというように 2 回の全画素スキャンが必要とされるが、計算の工夫をすることでこの画素スキャンを 1 回に抑えることができる。

3. 実験　CPU および GPU による相関計算高速化

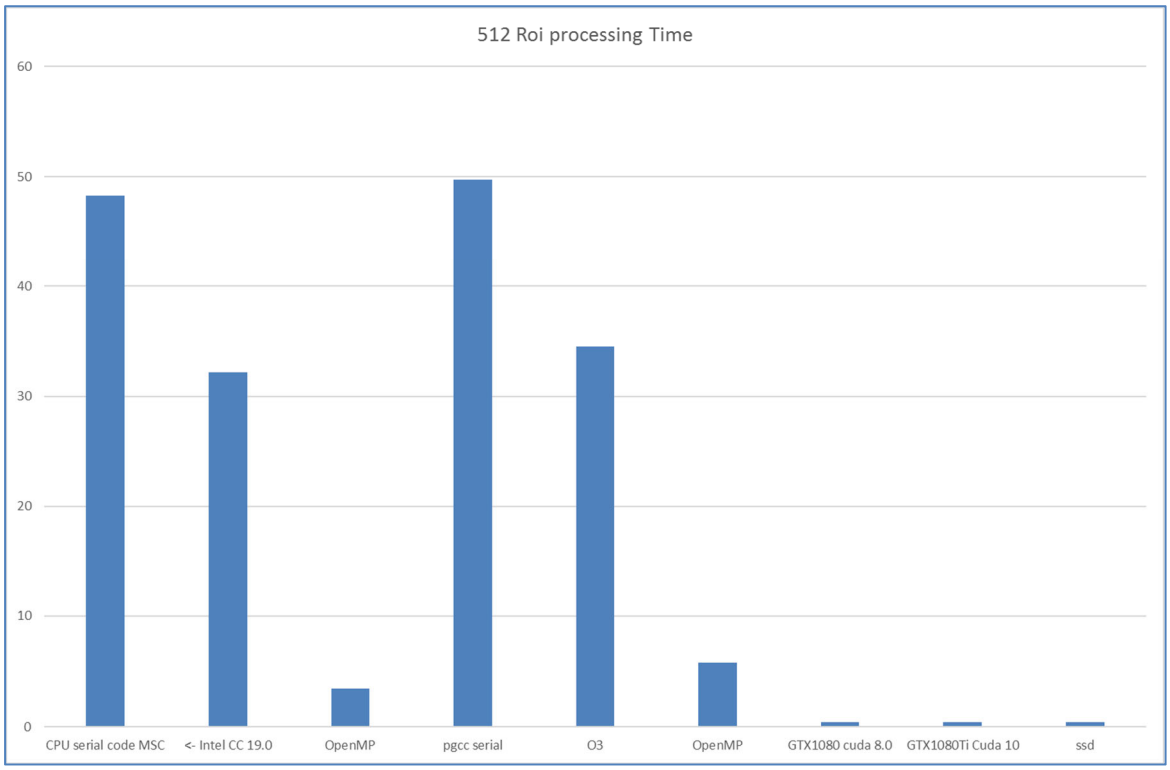

Fig. 1 CPU and Various GPUs comparison

種々の GPU や、CPU の多重利用による並列化の処理時間を Fig.1 に示す。ここで、比較的新しい GPU では処理時間の短縮化が達成できているが、この傾向はもっと新しい最近の GPU では頭打ちになっている。その状況を次の Fig.2 に示す。

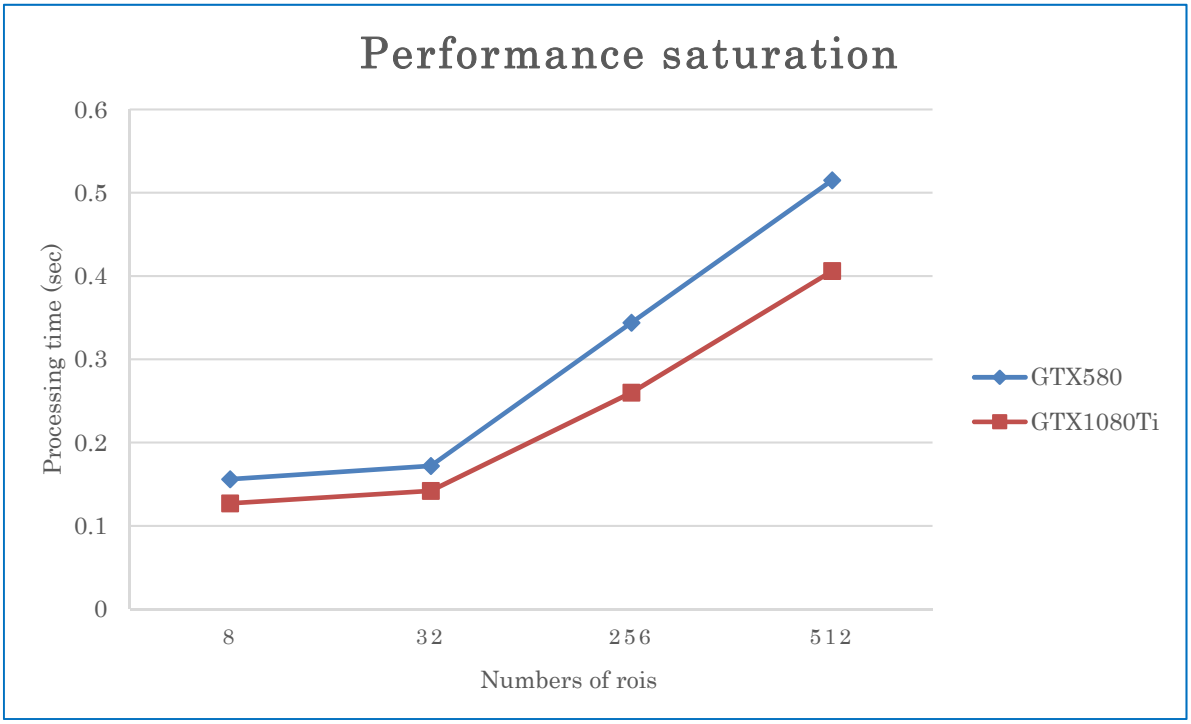

Fig.2　Performance saturation of GTX980 GPU

上図において、比較的新しい GPU である GTX1080Ti を用いてもパフォーマンスの向上はわずか 20%程度であった。本来の性能比が忠実に再現されたならば、50%程度の速度向上が見込まれるはずである。これには様々な原因があると考えられるが、GPU をサポートする Nvidia が供給するライブラリである CUDA のバージョンが古く、新しい GPU の効率的な演算を十分に利用できていないことが考えられる。上図で使用した CUDA のバージョンが当時の最新の 5.5 であったが、今では 8.0 が使用可能である。また GPU のアーキテクチャーも年々進歩してきており、新しい GPU とそれをサポートしている CUDA の新しいバージョンを用いることことで、全く同一のコードでも再ビルドするだけで数%以上のパフォーマンスの向上が期待できる。そこで上図の GTX580 および GTX980 に加えて、最新の GTX1080 を CUDA5.5 と 8.0 でビルドしたプログラムを作成し、パフォーマンスを計測した。その結果を次の Fig. 3 に示す。ここでプログラムは一行も変更しておらず、GTX1080 の新しい”PASCAL”アーキテクチャーを全く活用していない。しかし基本的な GPU-CPU 間のメモリー転送効率の向上などにより、図のように GTX980->GTX1080 の変更で、ROI512 個の場合の処理時間が、0.499 秒から 0.422 秒と 15%程度の処理時間の短縮が達成できた。CUDA5.0->8.0 の変更により GTX980 の場合で、0.49 秒から 0.459 秒へと 6%短縮、GTX1080 の場合で、0.422 秒から 0.409 秒へと 4%の短縮が達成できた。GTX1080 アーキテクチャーを活用して、CUDA コードを書き換えれば、さらに短縮化が可能なはずである。しかしこれははなはだ困難な作業となり、年々リリースされる新しい GPU の性能を自動的に反映するのは、自動ないし半自動ツールが必須である。そのためのツールとしては OpenACC や、本論文でも多用している OpenMP から GPU を活用する手法などが期待されているが、まだ開発段階である。

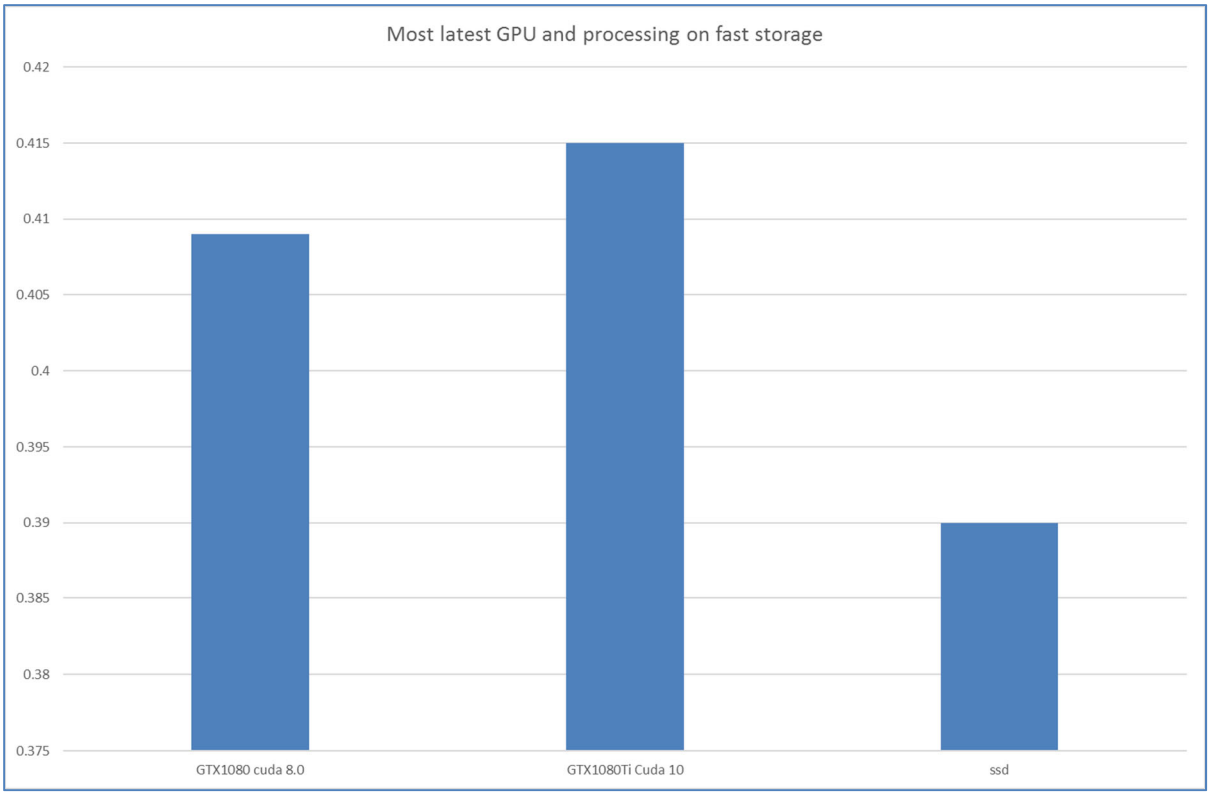

Fig.3　Performance of relatively recent GPU and/or CUDA

4. 結論

比較的新しい GPU と CUDA ライブラリを利用することで、元のコードを一行も変更することなく、パフォーマンスを数%程度向上させることができた。よりあたらしい GPU の新しく取り入れられたアーキテクチャーを積極的に活用するためには、GPU 側のコードを一から書き直すことが必要となり、かなり困難な作業となる。そのため OpenACC 等の半自動化高速化ツールを利用することが、使用者側には推奨される。OpenACC 等のツールは現在検証しているが、CPU の場合に効果的であったループに対して pragma 文で OpenMP を有効化するというような極めて単純化された変更だけでは現時点では十分な性能向上は達成できないことが分かった。

参考文献

[1] 東海大学　紀要　総合科学技術研究所 Vol.36 2015 pp.4-7
[2] OpenMP http://openmp.org
[3] CUDA http://developer.nvidia.com/category/zone/cuda-zone
[4] OpenACC http://www.openacc.org/