高速度ビデオでの運動解析プログラムの開発
－ GPGPUによるトレース高速化 －

横山直樹（東海大学・総合科学技術研究所）

Development of Versatile Motion Analysis Porgram(VMAP)
– Parallel processing of correlation with GPGPU -

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
Keywords: Motion analysis, Template Matching, High-speed videography, GPGPU

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 CUDA and OpenCL were adapted to perform cross-correlation calculation. Using GPU as calculation device, processing time can be reduced drastically.

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。

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

このプログラムの主な特徴は以下のとおりである、

- 島津製作所の 100 万駒/秒超高速度ビデオカメラで記録されたファイルに関しては、その内部構造までを把握しており、ヘッダ部分から撮影速度等の記録時の主要なパラメータを読み取る。
- 上記システム以外で得られた動画であっても、Windows の標準的な AVI ファイルであれば解析可能である。この場合は撮影速度等をマニュアルで正しくセットする必要がある。
- 物体の位置把握は、相互相関計算により行われる。位置検出は、サブピクセルの精度で実行される。
- 変位後の物体位置をさがすパターンは、螺旋である。これをスパイラルサーチと呼ぶ。時間分解能が充分な場合は、効率的なサーチが可能である。

- テンプレートを回転させながら相関計算を行うことで、物体の動きに回転成分が含まれている場合にも、その角度を検出することができる。
- 静止画を扱うことができる。時間的に連続する静止画は、それらを一旦 AVI ファイルに連結してから取り扱うことができる。
- 運動解析結果の軌跡を、元の動画に重ねた形で動画として出力できる。
- マニュアル解析が可能である。
- 各フレームの画像を、ヒストグラムの等値化アルゴリズムにより強調できる。
- ROI の数が多いか、またはその領域が大きいときは、相関計算にかなり時間がかかることが見込まれるが、前者の場合はネットワーク上で複数の計算機を用意することで、分散処理が可能である。
- 相関計算を工夫することで、上記のネットワーク分散処理に加えて単一ノードでも処理の高速化を図ることができる。
- 最近よく使われるようになってきた GPU を用いて並列に相関計算を行うことで、CPU のマルチコアでの並列計算による高速化以上の効果が得られた。（本報）

Fig. 1 にこのプログラムの操作パネルを示す。

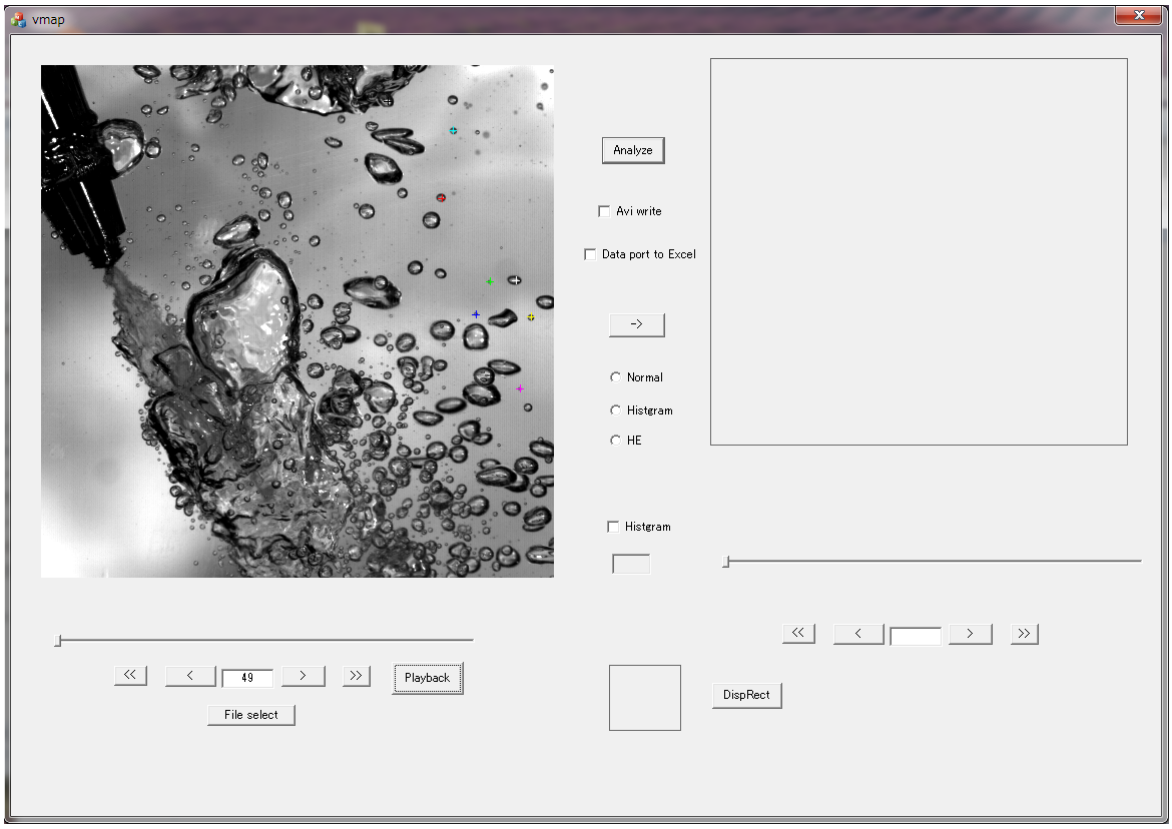

Fig. 1 Operational Panel of new VMAP

GUI スタイル(MFC を用いたダイアログベースのプログラム)での OpenMP 他を利用した相関計算並列化のために、開発システムを切り替えた。そのためにプログラムを全面的に書き直すこととなり、その課程で運動解析のためには必ずしも必要でない補助的な部分を省いたため、パネルがかな

り変更された。変更前のパネルを Fig. 2 に示す。

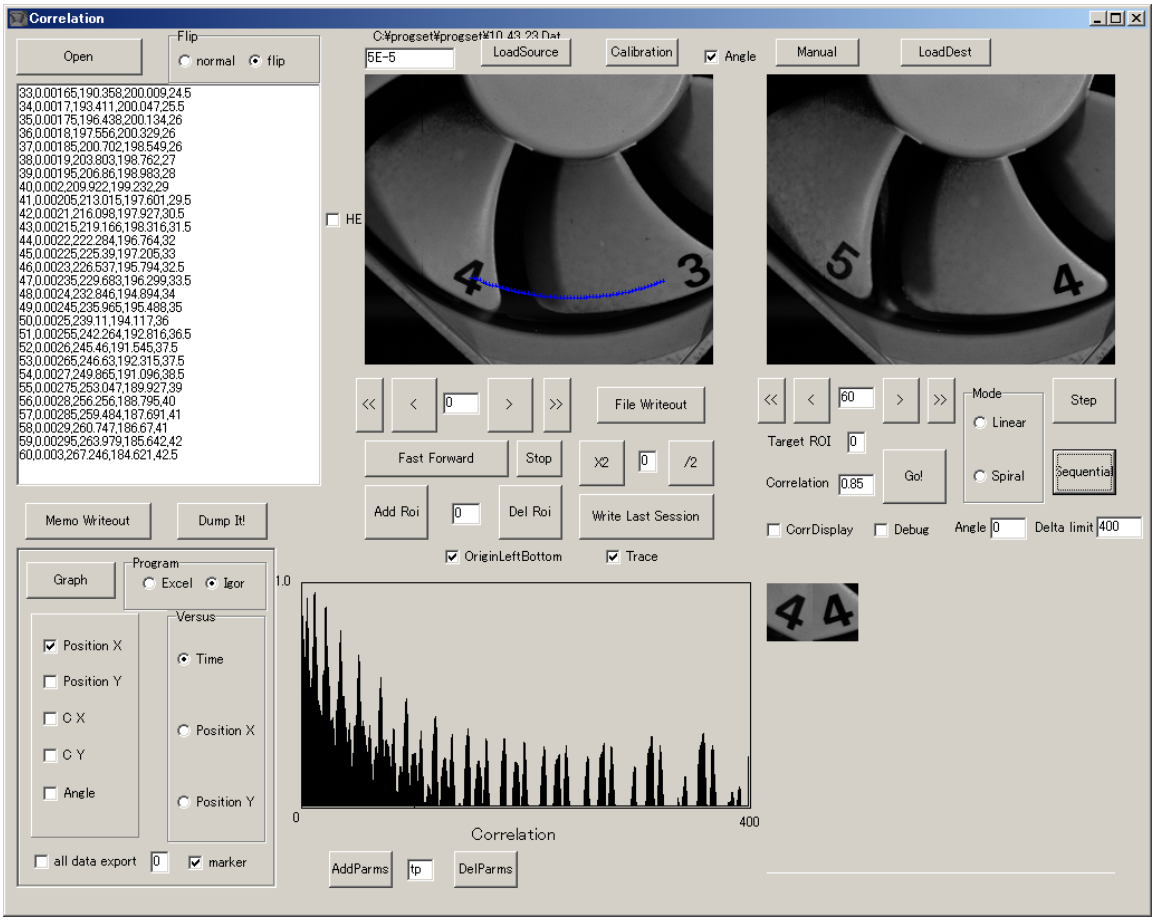

Fig. 2 Operational Panel of Old VMAP

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。$\bar{f}$ と $\bar{g}$ 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.負荷分散のストラテジ

対象部位が多数存在するような場合、負荷分散の方法は原則的に下記のものが考えられる。

1） フレームによる分割（時間軸での分割）

これは実際には、前のフレームにおける物体位置が後のフレームのスタート位置として影響するので、順次位置を求めることが必要なので、かなり困難である。

2） 位置による分割（空間軸での分割）

ROI によって分割する場合は実現容易であるが、ROI の個数が少なく、さらにそのサイズが大きい場合に負荷分散の効果があまり大きくならないことが予想される。つまり ROI の数以上には分割できないということである。その場合でも、相関計算自体を並列化することで、マルチコアを利用した単一ノードでの高速化は可能である。

空間軸での分割を用いて、ネットワーク分散処理により負荷軽減ができることは既に前報までで示した。本論文では単一ノードでの相関計算の高速化を CPU を用いて、さらには GPU を用いて行うことを目指す。

4. 実験　CPU ないし GPU による相関計算高速化

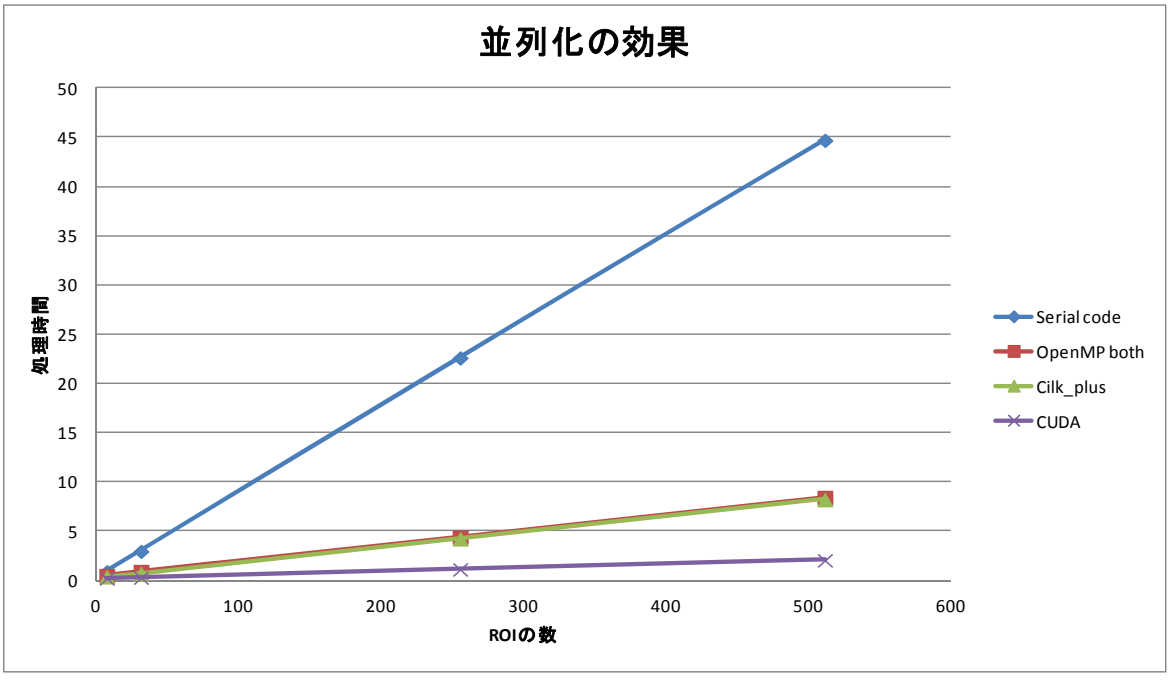

Fig. 3 Effects of various parallel calculation methods

前報までで、コマンドライン型のプログラムで MPI や OpenMP を用いて相関値計算を高速化する場合について述べたが、今回は VMAP そのものを並列化することを試みた。前述のように、そのために OpenMP や 64bit 化の進化が得られない処理系（C++ Builder）を捨て、Microsoft の Visual Studio に Intel 社の C コンパイラを加えた処理系に切り替えている。GPU を利用する場合は、Nvidia 系の

カードの場合は、CUDA ないし OpenCL、ATI 系の場合は、OpenCL を用いて相関値計算を極力並列化することを目指した。純粋に順次計算で対象物体をトレースした場合、CPU の複数のコアないしスレッドを利用して並列化した場合、GPU で並列化した場合のそれぞれの処理時間の比較を Fig.3 に示す。これは縦横 512 画素の白黒記録動画(前報 [1])と同一)を 50 フレームに渡って、それぞれ 8 個、32 個、256 個、512 個の ROI に対してトレースした際の所要時間をプロットしたものである。CPU は Corei7-2720QM、GPU は Nvidia GTX560Ti であり、コスト的にはほぼ同様なものを選んで比較した。ここで Cilk_plus は OpenMP と同様な仕組みの CPU による並列化の手法であり、僅かに OpenMP より高速であった。順次処理（シリアルコード）に対して CPU の並列化で数倍、GPU の利用でさらに数倍の高速化が達成され、いずれの場合も CPU に対して対費用効果がきわめて優れていることがわかった。

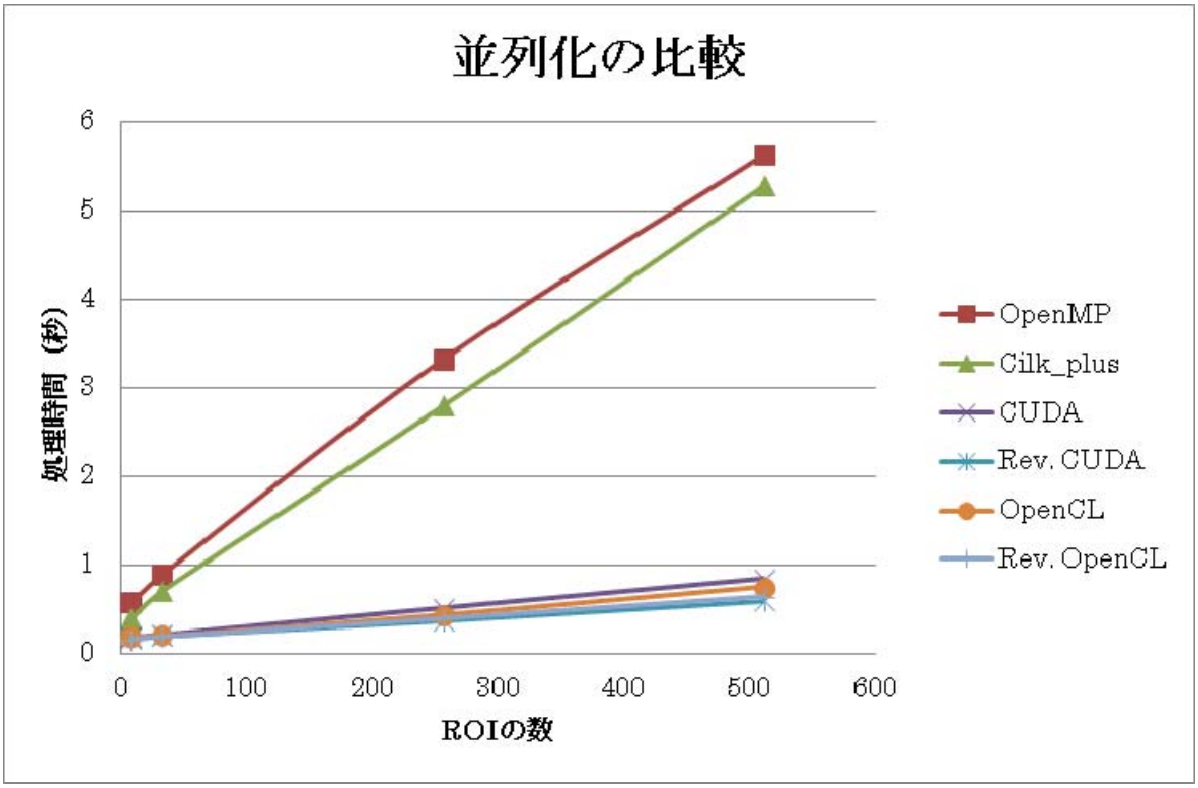

Fig. 4 Comparison of Various Parallelization methods in case of CPU

つぎに Fig.4 に CPU の場合の並列化手法による処理時間の比較を示す。縦軸と横軸は Fig.3 と同様である。ここで Rev. CUDA と Rev. OpenCL は前述の画素スキャンの回数を減らした場合の結果であり、無視できない程度の処理時間の改善が得られている。ATI のカードの場合は OpenCL の利用となるが、高速化の点で CUDA の場合とほぼ同様であった。また GPU 毎の処理時間の比較を Fig. 5 に示す。この場合も同様の AVI ファイルを 50 フレームに渡って、それぞれ 8 個、32 個、256 個、512 個の ROI に対してトレースした時間をプロットしたものである。Cilk_plus 3960X は、CPU によって並列化した際の現有マシンでの最速データである。この CPU はコア数が 6 で、ハイパースレッド有効なので実質 12 スレッドまでの並列化が有効である。古いタイプの GPU である GTX 280 以外ではほぼ同程度の高速化が達成できている。その GTX 280 でさえ、ROI の数が 8 や 32 の場合は CPU より遅いが、ROI の数が 256 を超えると CPU の倍程度の速さとなっている。これらの GPU の中では、浮動小数点演算速度では C2070 が抜きんでているはずであるが、結果はそうなってい

ない。これは GPU をドライブする CPU が非力なためか、並列化のスケジューリングが十分調整で

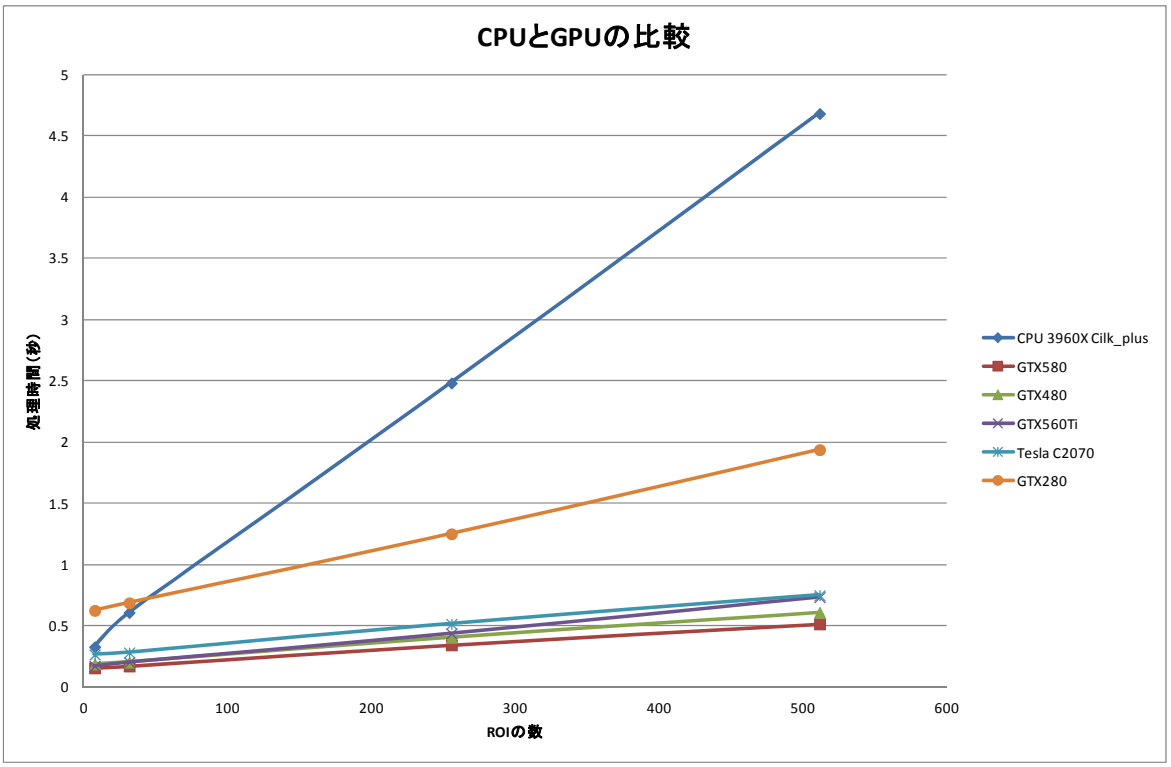

Fig. 5 Processing time of VMAP trace with CPU and Various GPUs

きていなかった可能性があり、高性能の GPU 使用時の高効率化も今後の課題である。いずれにしても GPU を用いることで CPU の場合に対して約 2 倍から 9 倍の高速化が達成できた。

5. 結論

CPU による並列化でほぼコアないしスレッド数倍の高速化が得られた。GPU を利用することで、さらに数倍の高速化が可能であることがわかった。しかし多数のバブルないし粒子を対象とし、相関計算を利用して運動解析を行う場合には、自動的にテンプレートを生成することが必要であり、今後の課題である。

参考文献

[1] 東海大学　紀要　総合科学技術研究所 Vol.31 2010 pp.12-21
[2] OpenMP http://openmp.org/
[3] CUDA http://developer.nvidia.com/category/zone/cuda-zone
[4] OpenCL http://www.khronos.org/opencl/