

# Mutual Information Computation and Maximization Using GPU

Yuping Lin and Gérard Medioni

Computer Science Department, University of Southern California  
3737 Watt Way, PHE 101 Los Angeles, CA, 90089

{yupingli, medioni}@usc.edu

## Abstract

*We present a GPU implementation to compute both mutual information and its derivatives. Mutual information computation is a highly demanding process due to the enormous number of exponential computations. It is therefore the bottleneck in many image registration applications. However, we show that these computations are fully parallelizable and can be efficiently ported onto the GPU architecture. Compared with the same CPU implementation running on a workstation level CPU, we reached a factor of 170 in computing mutual information, and a factor of 400 in computing its derivatives.*

## 1. Introduction

In information theory [3], mutual information is a quantity that measures the information shared between two random variables. Applied in image processing problems, it can be used to measure the similarity between two images. It is very powerful since there are no limiting constraints imposed on the image content of the modalities involved, which is particularly useful in measuring multi-modal images. Figure 1 shows some retinal images in different modalities.

One step further, aligning multi-modal images is always a challenging task. It is an important step in many applications, particularly in medical image processing. A thorough survey of mutual information based registration of medical images can be found in [8]. Maximizing mutual information is a commonly adopted approach which assumes the mutual information to be maximal if the two images are perfectly aligned [6, 12]. It is a very powerful criterion, since no assumptions are made regarding the nature of this dependence.

However, mutual information is computational demanding. It needs to compute the entropy and the joint entropy of two variables, which involves estimating the marginal probabilities and joint probabilities of each sample pixel. This requires an enormous number of exponential computations.

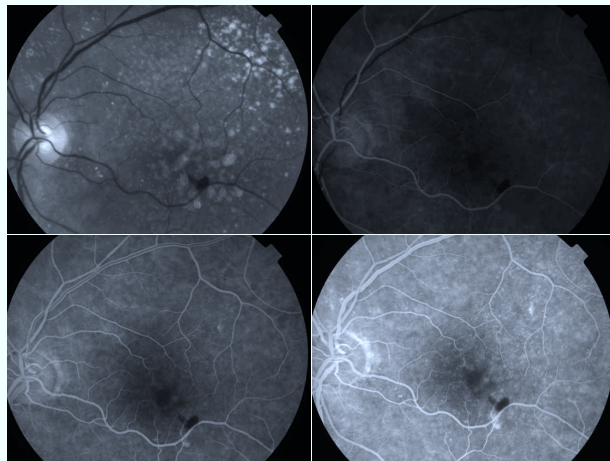


Figure 1. Retinal images in different modalities

Therefore, the speed up of computing mutual information becomes a critical issue.

We focus on the speed up of Viola’s approach [12] to computing mutual information, since the approach follows a close form solution of the mutual information derivatives, for which the computational time can be speeded up, too. In Viola’s approach, the number of exponential computations is  $n^2$ , where  $n$  is the number of samples required to estimate the entropy of a variable. The simplest way to reduce the number of exponential computations is by precomputing the Gaussian densities of all possible intensity values into a lookup table. However, if the intensity range is large, the precomputation becomes an overhead. Meihe [7] presents an approach that maps all the intensities into a smaller dynamic range, and computes/stores the Gaussian densities for them.

Nevertheless, such efforts only have limited speed up, which can still be impractical for many applications. In addition, extra interpolation is required if the intensity values are in floating point precision. We find way to parallelize the computation of the Viola’s algorithm, and achieve a significant speed-up by using the GPU. We program the

GPU using CUDA[1], a new hardware and software architecture for computing on the GPU. Although Shams[9] has presented a CUDA implementation of mutual information computation, he uses histograms to compute the probability and joint probability, which is hard to extend to a GPU version of computing mutual information derivatives.

The rest of the paper is organized as follows: Section 2 and 3 describe Viola’s approach to approximating mutual information and its derivatives respectively. A short description of CUDA and how the computation is implemented is described in section 3. Section 4 describes how maximizing mutual information can be employed in a retinal surface registration application. The experimental results are presented in section 5, followed by the conclusion and future work in section 6.

## 2. Mutual Information

Since our work is a GPU speed up of [12], we start by reviewing how the mutual information and its derivatives are approximated. First, from the information theory, the entropy of a random variable is defined as:

$$h(v) = - \int p(v) \ln p(v) dv, \quad (1)$$

and the joint entropy of two random variable is defined as:

$$h(u, v) = - \int p(u, v) \ln p(u, v) dudv. \quad (2)$$

The mutual information of two random variables  $u, v$  is then defined as

$$MI(u, v) = h(u) + h(v) - h(u, v). \quad (3)$$

### 2.1. Approximate Value

In [12], the probability density  $p(v)$  is estimated using Parzen Window method:

$$p(v) \approx \frac{1}{N_A} \sum_{v_j \in A} G_{\psi}(v - v_j), \quad (4)$$

where  $N_A$  is the number of samples in  $A$ , and  $G_{\psi}$  denotes a Gaussian function with variance  $\psi$ . Parzen Window is a widely adopted technique to estimate the probability density since it directly uses the samples drawn from an unknown distribution, and uses Gaussian Mixture model to estimate its density, which is robust to noise.

Equation 1 can be expressed as a negative expectation of  $\ln p(v)$ , *i.e.*,  $h(v) = -E_v(\ln p(v))$ . Together with equation 4, the entropy of a random variable  $v$  is approximated as:

$$h(v) \approx \frac{-1}{N_B} \sum_{v_i \in B} \ln \frac{1}{N_A} \sum_{v_j \in A} G_{\psi}(v_i - v_j), \quad (5)$$

where  $B$  is another sample set.

The joint entropy can be approximated in the same manner by drawing pairs of corresponding samples from the two variables. Hence the mutual information is approximated as:

$$MI(u, v) \approx \frac{-1}{N_B} \left\{ \sum_{u_i \in B} \ln \frac{1}{N_A} \sum_{u_j \in A} G_{\psi_u}(u_i - u_j) + \sum_{v_i \in B} \ln \frac{1}{N_A} \sum_{v_j \in A} G_{\psi_v}(v_i - v_j) - \sum_{w_i \in B} \ln \frac{1}{N_A} \sum_{w_j \in A} G_{\psi_{uv}}(w_i - w_j) \right\} \quad (6)$$

in which  $w = [u, v]^T$ . There exist other approximation formulas for computing mutual information[4, 10], which we have not considered here.

Algorithm 1 implements equation 6 to compute the mutual information of two images.  $I_u$  and  $I_v$  denotes the two images, while  $I(x)$  is the intensity value at  $x$  in image  $I$ .  $B_u, B_v, B_{uv}$  and  $A_u, A_v, A_{uv}$  are the values of the outer summation and inner summation in equation 5 respectively. Note that we assume the two images to be independent and approximate the joint probability as the product of the two marginal probabilities (line 9). We also assume the two images have the same variance  $\psi$ , *i.e.*,  $\psi_u = \psi_v = \psi$ .

**input** : Image  $I_u$  and  $I_v$ . Sample sets  $A$  and  $B$   
**output**: Mutual Information of  $I_u$  and  $I_v$

```

1 Initialize  $B_u, B_v, B_{uv}$  to 0;
2 foreach  $i$  in  $B$  do
3   Initialize  $A_u, A_v, A_{uv}$  to 0;
4   foreach  $j$  in  $A$  do
5      $G_u \leftarrow G_{\psi}(I_u(i) - I_u(j));$ 
6      $G_v \leftarrow G_{\psi}(I_v(i) - I_v(j));$ 
7      $A_u \leftarrow A_u + G_u;$ 
8      $A_v \leftarrow A_v + G_v;$ 
9      $A_{uv} \leftarrow A_{uv} + G_u \times G_v;$ 
10  end
11   $B_u \leftarrow B_u + \ln(A_u/N_A);$ 
12   $B_v \leftarrow B_v + \ln(A_v/N_A);$ 
13   $B_{uv} \leftarrow B_{uv} + \ln(A_{uv}/N_A);$ 
14 end
15 return  $-(B_u + B_v - B_{uv})/N_B;$ 

```

Algorithm 1: Mutual information

### 2.2. Approximate Derivative

In image registration problems, we usually want to estimate the transformation  $T$  that best aligns two images  $I_u$

and  $I_v$ . This is formulated as a mutual information maximization in [12] which looks for the optimal  $T$ :

$$\hat{T} = \arg \max_T MI(I_u(x), I_v(T(x))).$$

We can think of  $MI$  as a function of  $T$ . To maximize the mutual information, [12] approximates its derivative with respect to  $T$  as follows:

$$\frac{d}{dT} MI(I_u(x), I_v(T(x))) = \frac{1}{N_B} \sum_{x_i \in B} \sum_{x_j \in A} (v_i - v_j)^T W(x_i, x_j) \frac{d}{dT} (v_i - v_j). \quad (7)$$

The weighting factor  $W(x_i, x_j)$  is defined as:

$$W(x_i, x_j) = W_v(v_i, v_j) \psi_v^{-1} - W_{uv}(w_i, w_j) \psi_{uv}^{-1}, \quad (8)$$

$$W_v(v_i, v_j) = \frac{G_{\psi_v}(v_i - v_j)}{\sum_{x_k \in A} G_{\psi_v}(v_i - v_k)}, \quad (9)$$

$$W_{uv}(w_i, w_j) = \frac{G_{\psi_{uv}}(w_i - w_j)}{\sum_{x_k \in A} G_{\psi_{uv}}(w_i - w_k)}, \quad (10)$$

where  $u_i = I_u(x_i)$ ,  $v_i = I_v(T(x_i))$  and  $w_i = [u_i, v_i]^T$ .

Algorithm 2 shows the implementation of equation 7. The first inner loop computes the denominators in equation 9 and 10, and then the second inner loop computes the weighting factor in equation 8. In our implementation, the goal is to maximize the mutual information between two triangular patches by moving the triangle vertices in  $I_v$ , i.e.,  $dv_i/dT$  in line 16 is the derivative of a intensity value over the movements of the vertices. This can be done by

$$\frac{dv}{dT} = \frac{dv}{d\mathbb{A}} \frac{d\mathbb{A}}{dT},$$

where  $\mathbb{A}$  is an affine transformation.

We found that in both algorithms, the statistics computed for each element  $i \in B$  are independent from the others, which can be computed efficiently in GPU. Next we will describe how they are implemented in CUDA.

### 3. CUDA implementation

CUDA implements multiple memory spaces for different usage. It features the Single Instruction, Multiple Data architecture (SIMD), where each processor of the the multiprocessor executes the same instruction, but operates on different data. Figure 2 illustrates the memory model in CUDA. A grid, a block, and a thread are the logical representations of a GPU itself, a multi-core processor and a processor respectively. The key to speed up the computation is the usage of the shared memory. Shared memory is built on-chip, and can be accessed in parallel. By using the

```

input : Image  $I_i$  and  $I_j$ . Sample set  $A$  and  $B$ .
output:  $D$ , the derivative of  $MI$  with respect to a transformation  $T$ 

1 Initialize  $D$  to  $\vec{0}$ ;
2 foreach  $i$  in  $B$  do
3   Initialize  $A_v, A_{uv}$  to 0;
4   foreach  $j$  in  $A$  do
5      $G_u \leftarrow G_\psi(I_u(i) - I_u(j))$ ;
6      $G_v \leftarrow G_\psi(I_v(i) - I_v(j))$ ;
7      $A_v \leftarrow A_v + G_v$ ;
8      $A_{uv} \leftarrow A_{uv} + G_u \times G_v$ ;
9   end
10  foreach  $j$  in  $A$  do
11     $G_u \leftarrow G_\psi(I_u(i) - I_u(j))$ ;
12     $G_v \leftarrow G_\psi(I_v(i) - I_v(j))$ ;
13     $W_v \leftarrow G_v/A_v$ ;
14     $W_{uv} \leftarrow G_u \times G_v/A_{uv}$ ;
15     $W \leftarrow (W_v - W_{uv}) * \psi^{-1}$ ;
16    compute  $\frac{dv_i}{dT}$  and  $\frac{dv_j}{dT}$ ;
17     $D \leftarrow D + \frac{W}{N_B} (I_v(i) - I_v(j)) (\frac{dv_i}{dT} - \frac{dv_j}{dT})$ ;
18  end
19 end
20 return  $D$ ;

```

Algorithm 2: Mutual information derivative

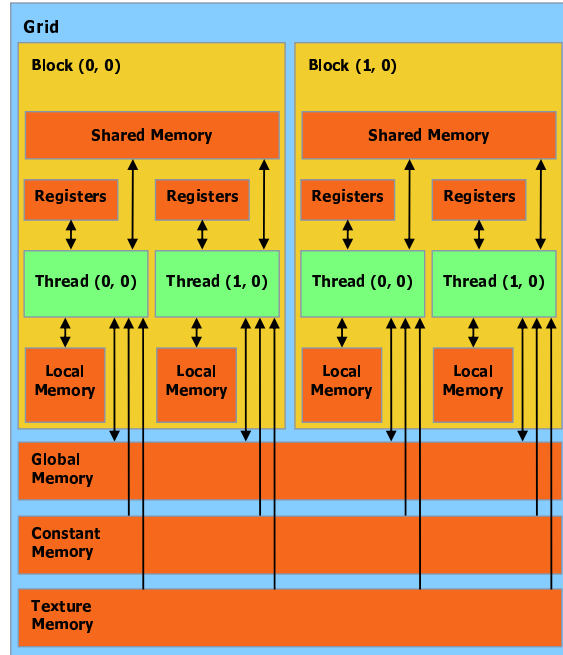


Figure 2. CUDA memory model

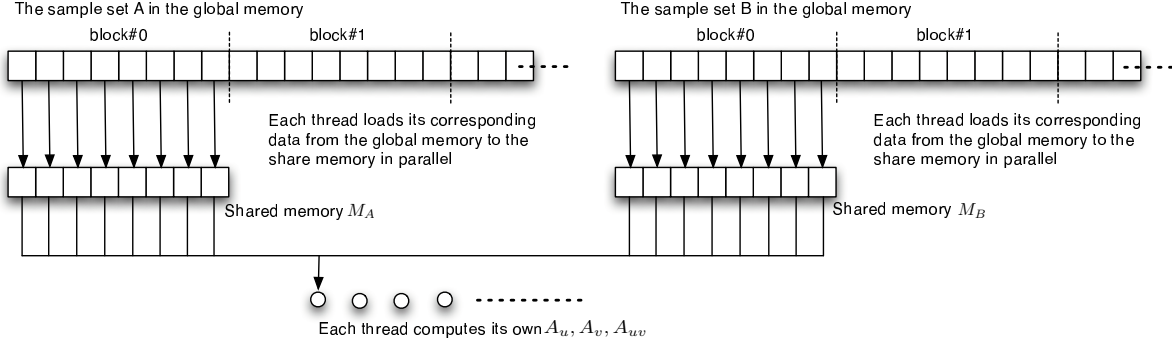


Figure 3. Parallel access of the shared memory

shared memory properly, we can minimize memory access latency and maximize the processor throughput.

Our strategy is to parallelize the computation of the inner summations of equation 6 and 7, since the statistics associated with each element  $i \in B$  are independent from the others. The only issue now is how the data associated with the elements in  $A$  can be loaded into the shared memory since it has a limited capacity (16K in our GPU). Figure 3 shows how the threads load data into the shared memory and compute the statistics in the inner loop of algorithm 1 in parallel. Computing  $A_u$ ,  $A_v$  and  $A_{uv}$  is performed on a block by block basis. Let  $BS$  be the block size (the number of threads in a block), we allocate two shared memory  $M_A$  and  $M_B$  and load the sample values in  $A$  and  $B$  from the global memory to them respectively. Once the  $M_A$  and  $M_B$  are loaded, each thread computes its own  $A_u$ ,  $A_v$  and  $A_{uv}$ . To maximize the bandwidth, there must be no bank conflicts while accessing the shared memory. Summing  $A_u$ ,  $A_v$ , and  $A_{uv}$  has no bank conflict since at iteration  $j$ , all threads read the same bank at  $M_A[j]$  and  $M_B[j]$ , and they can all be serviced simultaneously by broadcasting. Then the next block is loaded, until every block is loaded. The final summations from line 9 to line 11 in algorithm 1 can be done using reductions.

The statistics in the inner loops of algorithm 2 can also be computed in parallel in the same manner, and we skip the detailed description.

#### 4. Surface Alignment

We take advantage of this GPU speed-up to perform some computational demanding processes. Here we try to align two retinal images by maximizing mutual information. The alignment of retinal images is important for retina diagnosis, and the relative works can be found in [5, 2]. Since the retina is a near planar smooth surface, it is reasonable to model it as piece-wise planar, and model the transformation between each pair of planes with an affine transformation, as shown in figure 4. Sugimoto presented a

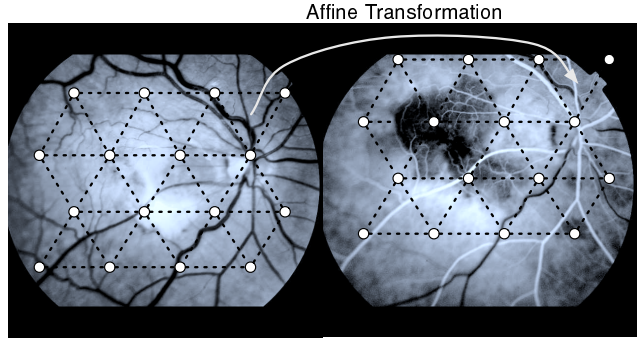


Figure 4. Piecewise planar retinal surface

similar setting in [11], where he minimizes the pixel intensity difference to align two terrain images.

For each triangular patch, we compute the derivatives of mutual information over the displacements of the three vertices. In general, a vertex in the mesh is associated with 6 triangles, therefore the derivatives are the average of its 6 associated derivatives. Once the derivatives for each vertex are computed, we find the step length which maximizes the mutual information along the gradient direction, and then recompute the derivatives from there. The process iterates until the mutual information has converged.

#### 5. Experimental results

The experiment is performed on the hardwares as follows:

Dell Precision 690	GeForce 8800GTX
Xeon Quad Core 2.33 GHz 4GB DDR2 667MHz	128 streaming processors 575 MHz 768MB DDR3 1.8GHz 16KB shared memory per block

Table 1. Hardware spec

In the first experiment, we run a thousand iterations of

both implementations with respect to different numbers of samples. The result is shown in figure 5. Using 1000 samples, the GPU has a factor of 170 and 400 in computing mutual information and its derivatives respectively, compared with a single CPU. As the number of samples increases, the computation time for the CPU grows quadratically. Although a mutli-core CPU can spread the tasks, the computation time can still be unacceptable when the number of samples is large. On the other hand, the computation time for the GPU is less than 3ms, and is relatively constant compared with that of the CPU. The result explains that GPU indeed computes mutual information more efficiently.

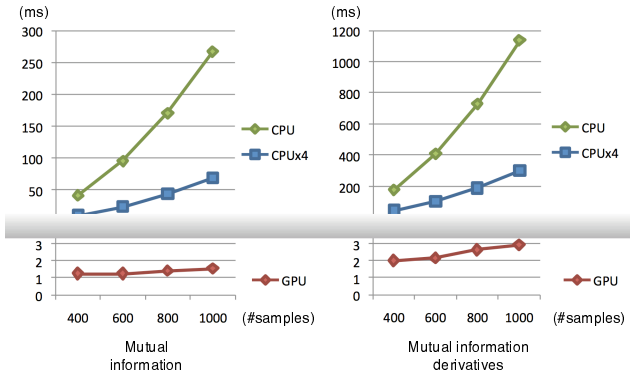


Figure 5. Time for computing mutual information derivatives with respect to point movements

In the second experiment, we evaluate the performance difference of a full image alignment process as described in the previous section. For a single triangular patch in  $I_u$ , we arrange all inner pixels in scan-line order, and group pixels with odd and even indices into sample sets  $A$  and  $B$  respectively. The corresponding samples in  $I_v$  are usually on fractional positions, where we use bilinear interpolation to approximate their intensities and intensity gradients. To find the step length, we start at 0.1, and double it until the mutual information cannot increase. Note that this requires mutual information to be computed at every step length, and the samples in  $I_v$  to be re-interpolated. We run 50 iterations for an image pair of size  $640 \times 480$  and compare the computation time with respect to the numbers of triangle sizes. We ignore the measurement of a single CPU since we know it is roughly 4 times longer than a quad-core CPU. The result is shown in figure 6.

We can see that the time savings in performing a full registration is not as much as computing mutual information or its derivatives alone. This is because the interpolation of the samples in  $I_v$  is performed on the CPU, and these samples have to be copied from the host memory to the GPU memory each time we compute the mutual information and its derivatives, which is a significantly large overhead. Nevertheless, the registration on GPU takes only a few minutes,

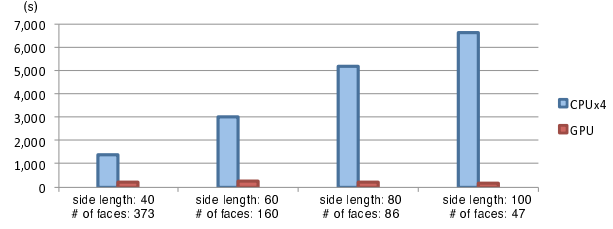


Figure 6. Time for computing mutual information derivatives with respect to point movements

where as CPU takes up to hours. In common medical image registration applications, the number of iterations is usually over a hundred, where the the CPU implementation is certainly impractical. On the other hand, the huge time saving in GPU implementation makes high computation demanding applications run in more reasonable time, such as registration in high resolution images.

## 6. Conclusions and Future work

We have presented a GPU implementation of computing mutual information and its derivatives. We showed how the computation and the shared memory access can be fully parallelized to maximize the GPU power. The computation time for both mutual information and its derivatives is greatly reduced up to a factor of 170 and 400 respectively compared with a work station level CPU. In the future, we plan to integrate it into mutual information applications and conduct experiments which we were unable to perform before. It will also be interested to compare the performances with other mutual information approximation methods [4, 10]. We intended to make this implementation publicly available once it is documented.

## Acknowledgement

This research was supported by the National Institutes of Health (NIH) under grant No. R21 EY015914-03. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH. We would like to thank the Doheny Eye Institute for providing data sets.

## References

- [1] *NVIDIA CUDA Programming Guide 1.1*. 2007. 2
- [2] T. E. Choe, I. Cohen, M. Lee, and G. Medioni. Optimal global mosaic generation from retinal images. In *ICPR '06*, pages 681–684. 4
- [3] R. M. Gray. *Entropy and information theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. 1
- [4] J. Kim, V. Kolmogorov, and R. Zabih. Visual correspondence using energy minimization and mutual information. In *ICCV '03*, page 1033. 2, 5

- [5] Y. Lin and G. Medioni. Retinal image registration from 2d to 3d. In *CVPR '08*. 4
- [6] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens. Multimodality image registration by maximization of mutual information. *Medical Imaging*, 16(2):187–198, 1997. 1
- [7] X. Meihe, R. Srinivasan, and W. L. Nowinski. A fast mutual information method for multi-modal registration. In *Information Processing in Medical Imaging*, pages 466–471, 1999. 1
- [8] J. Pluim, J. Maintz, and M. Viergever. Mutual-information-based registration of medical images: a survey. *IEEE Transactions on Medical Imaging*, 22(8):986–1004, 2003. 1
- [9] R. Shams and N. Barnes. Speeding up mutual information computation using nvidia cuda hardware. *Digital Image Computing Techniques and Applications*, pages 555–560, 2007. 2
- [10] C. Studholme, D. L. G. Hill, and D. J. Hawkes. An overlap invariant entropy measure of 3d medical image alignment. *Pattern Recognition*, 32(1):7186, 1999. 2, 5
- [11] S. Sugimoto and M. Okutomi. A direct and efficient method for piecewise-planar surface reconstruction from stereo images. *CVPR '07*, pages 1–8, 17-22 June 2007. 4
- [12] P. Viola and I. William M. Wells. Alignment by maximization of mutual information. *Int. J. Comput. Vision*, 24(2):137–154, 1997. 1, 2, 3