Accelerating Phase Based Motion Estimation with Hierarchical Search Technique Using Parallel Threading in Graphical Processing Unit (GPU)

Rosa A. Asmara† and M. Hariadi††
State Polytechnics of Malang Institute of Technology Sepuluh Nopember, Surabaya, Indonesia

Summary
This paper presents Phase Only Correlation (POC) methods in hierarchical search motion estimation for high resolution digital video using Graphical Processing Unit (GPU). Using the POC function, one can estimate the translational displacement as well as the degree of similarity between two image blocks from the location and height of the correlation peak, respectively[1]. Motion Estimation is a process for defining object movement in digital video sequences. Motion Estimation is a system used in some field such as image processing, image analysis, video coding, and computer vision. A POC based hierarchical search is a high cost algorithm results in long processing time, thus the system developed in this paper proceed POC function in Graphical Processing Unit using parallel threading technology. The evaluation counts processing time speed of the methods using Graphical Processing Unit in high definition video with 1280 x 720 pixel resolution. The results show that the methods using GPU performs accelerating speed more than two times faster processing 2 layer hierarchical search in 256x256 POC block size than doing the same methods using CPU. Using the NVidia GeForce 9600GT GPU, kernel execution with 256 thread per block, 9 32-bit register per thread, and 36 bytes of memory shared for every thread block, the multiprocessor maximum occupancy is 100%, with 768 active threads per multiprocessor, 24 Active Warps per multiprocessor, and 3 active thread blocks per multiprocessor.

Key words:
Motion Estimation, Phase Only Correlation, Graphical Processing Unit Programming, hierarchical search.

1. Introduction

Motion estimation is the process of determining the movement of the objects of a video sequence. The movement is usually expressed in terms of the motion vectors of selected points within the current frame with respect to another frame known as the reference frame. A motion vector represents the displacement of a point between the current frame and the reference frame.

Motion estimation is a fundamental task in numerous fields, such as image processing, image analysis, video coding, and computer vision. Robust high accuracy motion estimation is essential for applications such as mesh-based motion compensation for video coding, stereo vision 3D measurement, and super-resolution imaging (the reconstruction of a high-resolution image using multiple low-resolution images). Here, robustness refers to consistent pixel level estimation of motion vectors with minimal false detection [1].

Strategies for finding the best matching block are broadly classified into two types: full search methods and hierarchical search methods. The former is suitable for detecting local motion of individual objects, while the latter is suitable for detecting global motion of the scene. A POC based hierarchical search is a high cost algorithm results in long processing time, thus the system developed in this paper proceed POC function in Graphical Processing Unit using parallel threading technology.

2. Theory and related Algorithm

2.1 Phase Only Correlation
Consider two \(N_1 \times N_2\) images, \(f(n_1,n_2)\) and \(g(n_1,n_2)\), where we assume that the index range are \(n_1 = -M_1,...,M_1\) and \(n_2 = -M_2,...,M_2\), for mathematical simplicity, and hence \(N_1 = 2M_1 + 1\) and \(N_2 = 2M_2 + 1\). Let \(F(k_1,k_2)\) and \(G(k_1,k_2)\) denote the 2D Discrete Fourier Transforms (2D DFTs) of the two images. \(F(k_1,k_2)\) and \(G(k_1,k_2)\) are given by

\[
F(k_1,k_2) = \sum_{n_1 \in \mathbb{Z}} \sum_{n_2 \in \mathbb{Z}} f(n_1,n_2) e^{-j \frac{2 \pi n_1 k_1}{N_1}} e^{-j \frac{2 \pi n_2 k_2}{N_2}} = A_F(k_1,k_2) e^{i \theta_F(k_1,k_2)}
\]

\[
G(k_1,k_2) = \sum_{n_1 \in \mathbb{Z}} \sum_{n_2 \in \mathbb{Z}} g(n_1,n_2) e^{-j \frac{2 \pi n_1 k_1}{N_1}} e^{-j \frac{2 \pi n_2 k_2}{N_2}} = A_G(k_1,k_2) e^{i \theta_G(k_1,k_2)}
\]

Where \(k_1 = -M_1,...,M_1\), \(k_2 = -M_2,...,M_2\), \(W_{N_1} = e^{-j \frac{2 \pi}{N_1}}\), \(W_{N_2} = e^{-j \frac{2 \pi}{N_2}}\), and the operator \(\sum_{n_1,n_2}\) denotes
\[
\sum_{n_1=-M_1}^{M_1} \sum_{n_2=-M_2}^{M_2}, A_r(k_1, k_2) \text{ and } A_c(k_1, k_2) \text{ are amplitude components, and } e^{j \theta_r(k_1, k_2)} \text{ and } e^{j \theta_c(k_1, k_2)} \text{ are phase components.}
\]

The cross-phase spectrum (or normalized cross spectrum) \( \hat{R}(k_1, k_2) \) is defined as
\[
\hat{R}(k_1, k_2) = \frac{F(k_1, k_2) G(k_1, k_2)}{|F(k_1, k_2) G(k_1, k_2)|} = e^{j \theta(k_1, k_2)}
\]
(3)

Where \( G(k_1, k_2) \) denotes the complex conjugate of \( G(k_1, k_2) \) and \( \theta(k_1, k_2) = \theta_r(k_1, k_2) - \theta_c(k_1, k_2). \) The Phase Only Correlation function \( \hat{R}(n_1, n_2) \) is defined as
\[
\hat{R}(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1, k_2} \hat{R}(k_1, k_2) W_{N_1}^{-k_1 n_1} W_{N_2}^{-k_2 n_2}
\]
(4)

Now consider \( f_c(x_1, x_2) \) as a 2D image defined in continuous space with real number indices \( x_1 \) and \( x_2. \) Let \( \delta_1 \) and \( \delta_2 \) represent sub pixel displacement of \( f_c(x_1, x_2) \) to \( x_1 \) and \( x_2 \) directions, respectively. So, the displaced image can be represented as \( f_c(x_1 - \delta_1, x_2 - \delta_2). \) Assume that \( f(n_1, n_2) \) and \( g(n_1, n_2) \) are spatially sampled images of \( f_c(x_1, x_2) \) dan \( f_c(x_1 - \delta_1, x_2 - \delta_2), \) and are defined as,
\[
f(n_1, n_2) = f_c(x_1, x_2) \Big|_{x_1 = n_1 T_1, x_2 = n_2 T_2},
\]
\[
g(n_1, n_2) = f_c(x_1 - \delta_1, x_2 - \delta_2) \Big|_{x_1 = n_1 T_1, x_2 = n_2 T_2}
\]

Where \( T_1 \) and \( T_2 \) are the spatial sampling intervals, and index ranges are given by \( n_1 = -M_1, ..., M_1 \) and \( n_2 = -M_2, ..., M_2. \) The POC function \( \hat{R}(n_1, n_2) \) between \( f(n_1, n_2) \) and \( g(n_1, n_2) \) will be given by
\[
\hat{r}(n_1, n_2) = \alpha \frac{\sin[\pi n_1 + \delta_1]}{N_1} \frac{\sin[\pi n_2 + \delta_2]}{N_2}
\]
where \( \alpha \ll 1. \) The peak position of the POC function corresponds to the displacement between the two images, and the peak value \( \alpha \) corresponds to the degree of correlation between the two images. Figure 1 shows an example of function fitting to estimate the true position and height of the correlation peak

2.2 Hierarchical Search Method

In the POC-based hierarchical search motion estimation method (known thereafter as POC-HS), some coarser versions of the original input images are created. This method is also known as the coarse-to-fine correspondence search technique [6]. The POC-based block matching starts at the coarsest image layer and the operation gradually moves to the finer layers. The motion vector detected at each layer is propagated to the next finer layer in order to guide the search at that layer. An overview of the technique is shown in Figure 2. Let \( p_0 \) be the given point in the current image, and \( q_0 \) be the corresponding point in the reference image, and let \( p_l \) and \( q_l \) be the matching points at the \( l \)-th layer. The aim of the correspondence search is to find the corresponding point \( q_0 \) of point \( p_0 \) and in doing so, we obtain the motion vector of \( p_0 \) as \( q_0 - p_0. \)

**Procedure for POC-HS**

**Input:**
- Current image \( I_c(n_1, n_2) = I(n_1, n_2) \)
- Reference image \( I_r(n_1, n_2) = J(n_1, n_2) \)
- Point \( p_0 \) in \( I_c(n_1, n_2) \)

**Output:**
- Corresponding point \( q_0 \) of point \( p_0 \) in \( I_r(n_1, n_2) \)
- motion vector \( v_{p_0}^{HS} \) of point \( p_0 \)

**Step 1:** For \( l = 1, 2, \ldots, l_{max} \), create the \( l \)-th layer images \( I_c(n_1, n_2) \) and \( I_r(n_1, n_2) \), i.e., coarser versions of \( I_c(n_1, n_2) \) and \( I_r(n_1, n_2) \), recursively as follows:

\[
\sum_{n_1=-M_1}^{M_1} \sum_{n_2=-M_2}^{M_2}, A_r(k_1, k_2) \text{ and } A_c(k_1, k_2) \text{ are amplitude components, and } e^{j \theta_r(k_1, k_2)} \text{ and } e^{j \theta_c(k_1, k_2)} \text{ are phase components.}
\]
Step 2: For every layer \( l = 1,2, \ldots, l_{\text{max}} \), calculate the coordinate \( p_l = (p_{l1}, p_{l2}) \) corresponding to the original point \( p_0 \) recursively as follows:

\[
p_l = \left( \frac{1}{2} p_{l-1} - \frac{1}{2} p_{l-1} \right) = \left( \frac{1}{2} p_{l-1}, \frac{1}{2} p_{l-1} \right).
\]

Step 3: We assume that \( q_{l_{\text{max}}} = p_{l_{\text{max}}} \) in the coarsest layer, let \( l = l_{\text{max}} - 1 \).

2.3. Parallel Threading GPU

The GPU is especially well-suited to address problems that can be expressed as data-parallel computations – the same program is executed on many data elements in parallel – with high arithmetic intensity – the ratio of arithmetic operations to memory operations. Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control; and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches.

Data-parallel processing maps data elements to parallel processing threads. Many applications that process large data sets can use a data-parallel programming model to speed up the computations. In 3D rendering, large sets of pixels and vertices are mapped to parallel threads. Similarly, image and media processing applications such as post-processing of rendered images, video encoding and decoding, image scaling, stereo vision, and pattern recognition can map image blocks and pixels to parallel processing threads. In fact, many algorithms outside the field of image rendering and processing are accelerated by data-parallel processing, from general signal processing or physics simulation to computational finance or computational biology.

The data types in GPU that we use can only process single precision types, so we have to convert our data types first if our data is in double precision.

Procedure for POC GPU

Step 1: Convert data types for the input from double precision to single precision.

Step 2: Move the single precision data from CPU RAM to GPU RAM.

Step 3: Computing parallel FFT on GPU.

Step 4: Create the GPU Kernel for Cross Correlation to estimate the displacement. In our experiments, we use 256 thread per block.

Step 5: Computing parallel IFFT on GPU.

Step 6: Move the displacement results back from GPU to CPU.

Step 7: Convert the data types from single precision to double precision.

Step 8: Find the motion vector.
3. Experiments and Evaluation

To manage hundreds of threads running several different programs, the multiprocessor employs an architecture called SIMT (single-instruction, multiple-thread). The multiprocessor maps each thread to one scalar processor core, and each scalar thread executes independently with its own instruction address and register state. The multiprocessor SIMT unit creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. The multiprocessor maximum occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU. Each multiprocessor on the device has a set of N registers available for use by thread programs. These registers are a shared resource that are allocated among the thread blocks executing on a multiprocessor. The compiler attempts to minimize register usage to maximize the number of thread blocks that can be active in the machine simultaneously. If a program tries to launch a kernel for which the registers used per thread times the thread block size is greater than N, the launch will fail. The kernel created use 256 thread per block. From the experiments, known that it used 9 registers and 36 shared memory.

<table>
<thead>
<tr>
<th>Table 1 : Physical Limitations of GPU used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads / Warp</td>
</tr>
<tr>
<td>Warps / Multiprocessor</td>
</tr>
<tr>
<td>Threads / Multiprocessor</td>
</tr>
<tr>
<td>Thread Blocks / Multiprocessor</td>
</tr>
<tr>
<td>Total # of 32-bit registers / Multiprocessor</td>
</tr>
<tr>
<td>Register allocation unit size</td>
</tr>
<tr>
<td>Shared Memory / Multiprocessor (bytes)</td>
</tr>
<tr>
<td>Warp allocation granularity (for register allocation)</td>
</tr>
</tbody>
</table>

Using the physical limitations of the GPU as shown in table 1, we can calculate the multiprocessor maximum occupancy as follows:

\[
x_1 = 8
\]

\[
x_4 = 512
\]

\[
x_5 = 3
\]

\[
x_6 = 3
\]

\[
x_7 = 32
\]
Where \( x_7 \approx \text{Max. Shared memory per multiprocessor} \)

Took the minimum results from the equation (8), (9), and (10), which is equal to Active thread block per multiprocessor.

\[
\begin{align*}
\text{Active Warp per multiprocessor} &= \frac{\text{Active thread block per multiprocessor} \times \text{Warp per thread block}}{24} \\
\text{Active Threads per Multiprocessor} &= \frac{\text{Active thread block per multiprocessor} \times \text{thread per block}}{768} \\
\text{Multiprocessor Maximum Occupancy} &= \left( \frac{\text{Active Warp per multiprocessor}}{\text{Warp per multiprocessor}} \right) \times 100% = 100%
\end{align*}
\]

Fig. 4. (c) Motion Vector from Mobile Calendar using 32x32 size block taken from reference image (a) and current image (b)

The chart below is the results of the processing time in Full Search, 3 Layer and 2 Layer Hierarchical Search using 32x32 size POC block to 256x256 size POC block. The yellow bar is the processing time in CPU, and the red one is the processing time in GPU.

Fig. 5. Processing time of Full Search POC GPU and CPU

Fig. 6. Processing time of Hierarchical Search POC GPU and CPU 3 layer
The chart below is the results of the processing time ratio between CPU and GPU in 2 layer hierarchical search, 3 layer hierarchical search, and full search using 32 POC block to 256 POC block. The yellow bar is the processing time ratio in Full search, the red one is the processing time ratio in 3 layer hierarchical search, and the green one is the processing time ratio in 2 layer hierarchical search.

4. Conclusion

This paper presents an acceleration of POC-based motion estimation with hierarchical search using GPU for video sequences. In the proposed method, the motion vector result achieved from displacement result processed in GPU. We have demonstrated that the proposed method is generally more than twice faster than processing the same method using CPU. Using the NVidia GeForce 9600GT GPU, kernel execution with 256 thread per block, 9 32-bit register per thread, and 36 bytes of memory shared for every thread block, the multiprocessor maximum occupancy is 100%, with 768 active threads per multiprocessor, 24 Active Warps per multiprocessor, and 3 active thread blocks per multiprocessor.

Acknowledgments

The author wish to thank Mr. Suryo Hapsoro as a Research Director in Higher Educational Directorate of National Education Department Indonesia and the team for their financial support.

References

Rosa A. Asmara received the B.E. degree in electronics engineering from Brawijaya University, and the M.S. degree in Multimedia engineering, from Institute of Technology Sepuluh Nopember, Surabaya, Indonesia, in 2004 and 2009, respectively. He is currently a lecturer at State Polytechnics of Malang, Indonesia. His research interests include signal processing, image processing, parallel processing, and computer vision.

M. Hariadi received the B.E degree in electronics engineering, Institute of Technology Sepuluh Nopember, Surabaya, Indonesia, M.S. degree and PhD. Degree in Tohoku University, Japan, in 2003 and 2006, respectively. He is currently a senior lecturer in Institute of Technology Sepuluh Nopember, Surabaya, Indonesia. His research interests include image processing, video processing, Artificial Intelligence, multimedia and computer vision.