GPU Implementation of Belief Propagation Using CUDA for Cloud Tracking and Reconstruction *

Scott Grauer-Gray and Chandra Kambhamettu
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
{*grauerg, chandra*}*@cis.udel.edu*

Kannappan Palaniappan
Dept. of Computer Science
University of Missouri
Columbia, MO 65211
palaniappank@missouri.edu

Abstract

This paper describes an efficient CUDA-based GPU implementation of the belief propagation algorithm that can be used to speed up stereo image processing and motion tracking calculations without loss of accuracy. Preliminary results in using belief propagation to analyze satellite images of Hurricane Luis for real-time cloud structure and tracking are promising with speed-ups of nearly a factor of five.

1. Introduction

The generation of an accurate disparity map given a pair of stereo images and motion vectors extracted from a set of sequential images are well-studied problems in computer vision. Robust estimation of 3D structure and motion for natural scenes remain challenging areas of research. One recent advance involves the use of Markov random field (MRF) models to generate an NP-hard energy minimization problem, and then using graph cuts or belief propagation (BP) to generate an approximate solution with reasonable computational cost. Implementations of these global methods often generate disparity maps and motion vectors that are closer to the ground truth than implementations of local methods, but it takes longer to generate the results. Ideally, one wants the accuracy achieved via the global methods with the running time cost of local methods. One path towards this goal is to speed up a BP implementation without losing accuracy, by taking advantage of the high performance capabilities of Graphics Processing Units (GPUs) available on most personal computing platforms today. This paper describes a General Purpose GPU (GPGPU) implementation of the BP algorithm using the nVidia Compute Uniform Device Architecture (CUDA) language environment.

*This work was partially supported by a U.S National Aeronautics and Space Administration award NASA NNX08AD80G under the ROSES Applied Information Systems Research program.

2. The GPU and CUDA

General-purpose processing on the GPU, known as GPGPU is currently an active research area since GPUs are widely available and continue to improve in performance faster than CPUs. The capabilities of the GPU have increased dramatically in the past few years and the current generation of GPUs have higher floating-point performance than the most powerful (multicore) CPUs [1]. The GPU contains hundreds of cores that are well suited for parallel implementations, using a single-instruction multiple data (SIMD) programming model. Many algorithms have been implemented on the GPU, and the results are often a significant speed-up over the sequential CPU implementation of the same algorithm.

However, until recently a graphics API and a shader language such as Cg had to be used to take advantage of the GPU's processing power. In addition, GPU programs could not scatter data to any part of the DRAM on the GPU. These requirements and limitations added overhead to any GPU implementation and made programming the device more difficult. Recently, nVidia released CUDA, which allows GPUs to be programmed using a variation of C with specific parallel extensions including the capability to perform the scatter memory operation. This enables algorithms to be implemented on any CUDA-capable GPU with greater ease of programming.

3. Belief Propagation

Belief propagation (BP) is an iterative algorithm that is used in a number of vision tasks. In the stereo vision

structure estimation problem, BP is used to perform approximate inference on a NP-hard energy minimization problem to find an accurate disparity map between a pair of images. The total energy represents the sum of the data costs $D_p(d)$ for each pixel p and the discontinuity costs $V(d_p, d_q)$ for each pair (p, q) of neighboring pixels. In the stereo vision problem, the data cost $D_p(d)$ represents the cost of assigning disparity d to pixel p using the brightness consistency assumption; the data cost increases as the difference in intensity between corresponding pixels increases. The discontinuity cost $V(d_p, d_q)$ represents the cost of assigning disparities d_p and d_q to neighboring pixels; this cost increases as the difference between d_p and d_q increases. The goal is to find the disparity map that minimizes the total energy. This problem is equivalent to finding the maximum a posteriori (MAP) estimator of a MRF [3], so an approximate solution can be found using the loopy BP algorithm that is used for inference on MRFs.

The BP algorithm runs for a number of iterations. In each iteration, messages are computed and sent from each pixel to each of its neighbors, and the values in the received messages and the data costs are used to compute the messages to send in the next iteration. Each message can be viewed as a vector containing a value for each possible label, the label being the disparity value in the stereo algorithm. When all the iterations are complete, the values for each label in the messages as well as the data costs are used to retrieve the estimated disparity at each pixel.

Implementations of stereo BP produce good results, but the running time is longer than local stereo methods. Many iterations are required to ensure convergence of the message values, it takes $O(n^2)$ running time to generate each message where n corresponds to the number of possible disparity values (labels), and separate messages must be generated and passed to each pixel's four neighbors in each iteration. In [3], Felzenszwalb presents the following methods to speed up these aspects of BP.

First, a hierarchical scheme is introduced where the output messages from a coarser scale are used to initialize the messages at a finer scale. This causes the message values to converge in fewer iterations.

Second, the pixels are divided into sets A and B in a 'checkerboard' manner; each pixel in one set passes messages to its neighbors in the other set. The workload in each iteration is cut in half by alternating between updating the messages in sets A and B.

Finally, the running time to generate each message is reduced from $O(n^2)$ to O(n) when the discontinuity cost $V(d_p, d_q)$ is computed using certain models. When the truncated linear model is used where the discontinuity cost is equal to the difference between d_p and d_q bounded by some value t, the message m can be computed in O(n) time as follows (m_d corresponds to the message value at disparity d):

1. Initialize the value m_d at each disparity d by aggregating previous message and data cost values corresponding to d.
2. Set m_{max} to the sum of the minimum m_d and t.
3. Update m in two passes that are performed sequentially and 'in place', so each update influences future ones
   (a) $m_d = \min(m_d, m_{d-1} + 1)$ from d = 1 to n-1
   (b) $m_d = \min(m_d, m_{d+1} + 1, m_{max})$ from d = n-2 to 0

4. Belief Propagation on CUDA

BP has been implemented on the GPU in the past; both [2] and [5] describe GPU implementations of BP on a set of stereo images. However, each of these implementations use a graphics API rather than CUDA, adding overhead and complexity to the implementations. In particular, the lack of the scatter memory operation forces adjustments to the implementation that may adversely affect the running time, particularly in the generation of messages. The BP algorithm maps more naturally to CUDA, and the remainder of this paper describes a CUDA implementation of the BP algorithm as well as the results.

CUDA is structured such that the GPU works as a co-processor that processes a kernel function on multiple threads in parallel. These threads are organized into a grid of thread blocks. Up to 512 threads can be grouped together in a thread block where each thread has a unique index as a member of a 1-D, 2-D, or 3-D array, and the thread blocks that run the same kernel are placed together in a 1-D or 2-D structure called a grid. In many ways, the CUDA kernel is analogous to the fragment shader; the GPU is able to process multiple threads in parallel that are running the CUDA kernel or the fragment shader. However, the capabilities of the CUDA kernel go beyond the capabilities of the fragment shader. Unlike the fragment shader, the CUDA kernel allows the programmer to define the thread block dimensions, provides the capability to synchronize threads within a thread block, and is able to scatter data anywhere in GPU memory.

In the CUDA BP implementation, the truncated linear model with truncation values T_{data} and T_{disc} is used to compute the data and discontinuity costs, and the parameter λ sets the relative weight of the data cost. For

all stereo experiments shown in this paper, T_{data} = 15.0, T_{disc} = 1.7, λ = .07, and the disparity space runs from 0 to 14. The implementation contains the following steps:

1. Calculate the data cost D_p(d) for pixel p at each point (x, y) at the bottom level 0.
2. Iteratively calculate the data cost for each pixel at each succeeding level by aggregating the data cost of the corresponding 2 X 2 block of pixels at the preceding level.
3. For each level L-1 down to 0
   (a) For each pixel in set A or B, compute the messages to send to neighboring pixels using the current message values and data costs. Repeat for i iterations alternating between sets A and B.
   (b) If not at final level, copy the message values at each pixel to the corresponding 2 X 2 block of pixels at the next level down.
4. Retrieve the estimated disparity map by finding, for each pixel, the disparity that minimizes the sum of the data cost and message values.

Each step of the algorithm can be mapped to the SIMD model of the CUDA kernel, as the same operations are independently performed on every pixel. In total, the CUDA implementation of BP contains five kernels, one for each step/sub-step.

The message values are computed using the O(n) algorithm described in section 3 that includes two passes through the disparity space. It is notable that the message updates can be performed in a single kernel invocation. When the message updates are performed using fragment shaders in [2], a shader is called for each disparity d in each of the two passes; it is not possible to write the messages values for multiple disparities in the same invocation without the scatter operation.

5. Results on BP for Stereo Estimation

The BP algorithm that was implemented in CUDA is based on the algorithm introduced in [3] and the resulting GPU-based disparity maps were validated to be nearly identical with the sequential version. It should be noted that belief propagation based stereo analysis incorporating occlusion handling, sophisticated data terms and other improvements are among the best performing algorithms in the Middlebury benchmark set (http://vision.middlebury.edu/stereo/eval/). Figure 1 shows the results of running the implementation on two stereo sets of images; the first set is the 384 X 288 Tsukuba set, and the second set is a pair of 512 X 512 satellite images taken by NOAA GOES-9 and GOES-8 during Hurricane Luis on 6 September 1995. In both cases, the images are first smoothed using a CUDA-implemented Gaussian filter with σ = 1.0, and then the CUDA BP implementation is run using 5 levels and 10 iterations per level.

The implementation is then benchmarked and compared to the sequential CPU implementation on two systems. The first system is an Intel Core 2 Duo CPU running at 2.00 GHz and a nVidia 8600M GT GPU with two multiprocessors. The second is a desktop with a Intel Quad-core Xeon CPU running at 2.00GHz and a nVidia 8800 GT GPU with 14 multiprocessors. In all the CUDA tests, the thread block size is set to 32 X 4.

Benchmarking is first performed on the 384 X 288 Tsukuba stereo set; the BP implementations are run on the set using 5 levels with 6 iterations per level. This configuration was also benchmarked by Brunton in [2] on a GPU implementation of BP using the graphics API. The CPU implementation on the first system (second system) runs in 2.2 seconds (.35 seconds), while the CUDA implementation runs in .7 seconds (.33 seconds) when the time to transfer the image data to the GPU and the disparity values back to main memory is included, and .47 seconds (.086 seconds) when the transfer time is not included. Brunton's implementation is run on a system with a 3.4 GHz Pentium 4 and a nVidia GeForce 6800 GT GPU. The running time of the sequential CPU implementation on that system is 1.189 seconds, and the running time of the GPU implementation using the graphics API is .610 seconds. However, it is not given if this includes the time to transfer data between the GPU and the CPU.

The CUDA BP implementation is also benchmarked on the stereo set of 512 X 512 satellite images to explore the feasibility of real-time cloud tracking using this implementation. As stated above, these images are first smoothed using a CUDA-implemented Gaussian filter with σ = 1.0, and then BP is run on the images using 5 levels and 10 iterations per level. Including the time to smooth the images, the total running time of the GPU implementation (sequential implementation) on the images is 1.6 seconds (10 seconds) on the first system and .5 seconds (1.3 seconds) on the second system.

6. Results on BP for Motion Estimation

On 6 September 1995, the satellite NOAA GOES-9 generated a nearly 12-h sequence of observations covering Hurricane Luis. The visible images at 1-minute interval were available, and it is important to generate accurate motion vectors showing the cloud movement

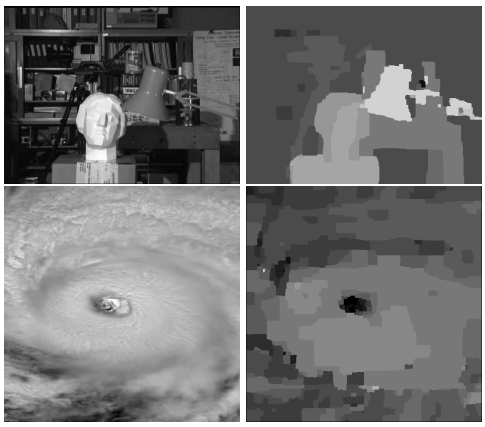

Figure 1: Results of the BP implementation on the Tsukuba stereo set and on a pair of satellite images.

in these images for modeling purposes. BP can be used to estimate these vectors, but a large number of possible motion vectors can lead to high storage requirements and a long running time.

It was shown in section 5 that CUDA can be used to speed up BP for stereo, and the general structure of a BP implementation for motion is the same as for stereo. The main difference is that the labels represent 2D motion vectors rather than 1D disparity values.

Unfortunately, there is limited storage on the GPU. This presents a challenge when the BP motion implementation is run on 512 X 512 satellite images; there is not enough DRAM available to store all the data costs and messages to run BP on the full images. One solution is to divide the images into multiple blocks, run BP on the blocks, and combine the results. This concept of block-based BP is presented in [4], and the increase in error rate using this method is minimal.

Each 512 X 512 image is divided into four 256 X 256 images. The range of motion in the satellite images is [-5, 5] in the x and y directions, resulting in 121 possible motions. Each 256 X 256 image is processed using 4 levels with 10 iterations per level, and the results combined to generate a full set of motion vectors.

The resulting motion for a pair of sequential satellite images is shown in Figure 2. Benchmarking is performed on the two systems described in section 5. On the first system (second system), the total running time of the GPU implementation is 15 seconds (4.5 seconds). Then, to establish a basis of comparison, a sequential CPU implementation of the algorithm is also benchmarked. There is more space in main memory than in GPU memory, so it is not necessary to divide the images when running the CPU implementation. Five levels with 10 iterations per level are processed on the complete images, and the running time is 64 seconds (15 seconds) on the first system (second system). It is clear that the use of the GPU results in a faster running time.

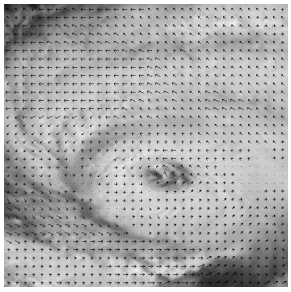

Figure 2: Results of the BP implementation for motion analysis on a set of satellite images.

7. Conclusions

The CUDA BP implementation presented in this paper can be used as part of a cloud-tracking system as discussed in Section 1. Given a set of cloud images, the implementation is capable of quickly producing an accurate disparity map or set of motion vectors. Future steps will include exploring the possibility of temporal smoothing of the output values over a sequence of images, and investigating semi-fluid based regularization schemes for robust disparity or motion estimation.

References

[1] *NVIDIA Corporation: NVIDIA CUDA compute unified device architecture programming guide*. NVIDIA Corporation, Jan 2007.

[2] A. Brunton, C. Shu, and G. Roth. Belief propagation on the GPU for stereo vision. In *Proc. 3rd Canadian Conf. Computer and Robot Vision*, page 76, 2006.

[3] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR'04)*, pages 261–268, 2004.

[4] Y. Tseng, N. Chang, and T. Chang. Low memory cost block-based belief propagation for stereo correspondence. In *IEEE Int. Conf. Multimedia and Expo*, pages 1415–1418, 2007.

[5] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér. Real-time global stereo matching using hierarchical belief propagation. In *British Machine Vision Conf.*, pages 989–998, 2006.