Realtime Motion Detection Based on the Spatio-Temporal Median Filter Using GPU Integral Histograms

Mahdieh Poostchi*
Dept. of Computer Science
University of Missouri
Columbia, MO, USA
mpr69@mail.missouri.edu

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

Filiz Bunyak
Dept. of Computer Science
University of Missouri
Columbia, MO, USA
bunyak@missouri.edu

Guna Seetharaman
Air Force Research Laboratory
Rome, NY, USA
gunasekaran.seetharaman@rl.af.mil

ABSTRACT

Motion detection using background modeling is a widely used technique in object tracking. To meet the demands of real-time multi-target tracking applications in large and/or high resolution imagery fast parallel algorithms for motion detection are desirable. One common method for background modeling is to use an adaptive 3D median filter that is updated appropriately based on the video sequence. We describe a parallel 3D spatiotemporal median filter algorithm implemented in CUDA for many core Graphics Processing Unit (GPU) architectures using the integral histogram as a building block to support adaptive window sizes. Both 2D and 3D median filters are also widely used in many other computer vision tasks like denoising, segmentation, and recognition. Although fast sequential median algorithms exist, improving performance using parallelization is attractive to reduce the time needed for motion detection in order to support more complex processing in multi-target tracking systems, large high resolution aerial video imagery and 3D volumetric processing. Results show the frame rate of the GPU implementation was 60 times faster than the CPU version for a $1K \times 1K$ image reaching 49 fr/sec and 21 times faster for 512×512 frame sizes reaching 194 fr/sec. We characterize performance of the parallel 3D median filter for different image sizes and varying number of histogram bins and show selected results for motion detection.

Keywords

spatio-temporal median filtering, 3D median, GPU, integral histogram

*Corresponding author

ICVGIP '12, December 16-19, 2012, Mumbai, India

1. INTRODUCTION

Motion detection is often a critical early stage of video object tracking. A common approach is to learn the stationary part of a video sequence using background modeling and detect moving foreground objects using background subtraction [1, 2, 3]. Reliable background subtraction based motion detection depends on several parameters such as spatio-temporal illumination compensation, shadow detection, background update and the background model learning rate. In terms of background modeling, fast learning leads to better adaptation to background changes but often results in more spurious detections from small background motions or missed detections when the object moves slowly or becomes temporarily stationary and blends with the background model. On the other hand slow learning can result in ghosting artifacts [4, 5]. The Median filter is a spatio-temporal nonlinear filter used for image enhancement and analysis [2, 6]. Median filter is commonly used in many computer vision tasks like segmentation, detection, tracking or recognition to discriminate moving objects from the background [1, 7, 8]. In this case the motion detection filter is

$$|I(x, y, c) - I_{Med}(x, y, c)| < \tau \quad (1)$$

where $I(x, y, c)$ is the current image pixel intensity value at (x, y) and $I_{Med}(x, y, c)$ is the spatio-temporal median value at pixel (x, y) in the current image, c,

$$I_{Med} = \underset{\Delta x, \Delta y}{Median}\{I_{c-\frac{T}{2}}, ..., I_c, ..., I_{c+\frac{T}{2}}\}$$

Otsu thresholding can be used to adaptively compute the threshold value τ for each time step. The most important advantage that median filter has over other well-known background modeling techniques such as mixture of Gaussians [9], Kalman filter [10] or Wallflower [11], is that it avoids blending pixel values. The output of the median corresponds to one of the values in the neighborhood and it does not create unrealistic pixel values out of the image content [12]. However, the median operation is computationally expensive which limits its application for real-time, large scale, high resolution imagery. In this paper we propose a fast implementation for background estimation model using 3D median filtering for motion detection, based on the integral histogram and utilizing many core Graphics Processing Unit

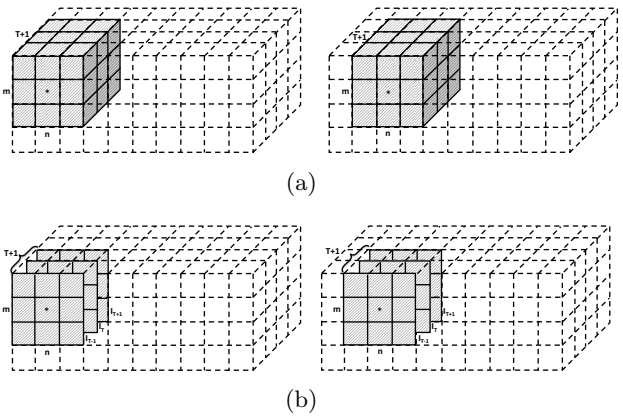

Figure 1: Spatio-temporal median filter sliding window operation. (a) 3D box median filtering [28], (b) 2D+t median filtering

(GPU) architectures using the CUDA programming model.

Fast and approximate foreground motion estimation is quite useful for a number of video analysis and tracking applications including video summarization [13, 14, 15], tracking in wide area imagery [16, 17, 18, 19], object detection and tracking in biomedical applications [20, 21, 22, 23, 24]. Our previous work in parallelizing motion detection include [25, 26, 27].

The main contributions of this paper include implementation of an integral histogram-based multi-scale 3D median filter that can handle large temporal windows, design of an efficient data structure for integral histogram, its layout in GPU memory, and optimization of the kernel configuration to maximize the resource utilization of the GPUs and to minimize the data movement. In the case of spatio-temporal median filter $T+1$, image arrays are transferred to the GPU using double buffering. Meanwhile the individual integral histograms are computed for each image array and the joint integral histogram is computed for $T+1$ individual integral histograms. Finally, the medians are computed for each pixel using the joint local integral histograms of kernel size $m \times n$ in parallel which results in high GPU utilization. Results show the frame rate of the GPU implementation versus the CPU implementation.

Section 2 presents a short review of the spatio-temporal median filter based on histogram. Section 3 includes the proposed spatio-temporal median filtering based on GPU integral histogram, followed by experimental results and conclusions.

2. SPATIO-TEMPORAL MEDIAN FILTER

Several techniques are typically used to perform 3D spatio-temporal median filtering. Figure 1 shows two types of sliding window operations for computing the 3D median filter [28]. Having a three-dimensional array mask (Fig. 1(a)) with size $m \times n \times (T+1)$, the real spatio-temporal median value for each pixel at (x, y) of current frame z, is obtained by

$$M_{3D}(x,y,z) = \underset{\substack{\Delta x \in [-\frac{m-1}{2}, \frac{m-1}{2}] \\ \Delta y \in [-\frac{n-1}{2}, \frac{n-1}{2}] \\ \Delta t \in [-\frac{T}{2}, \frac{T}{2}]}}{Median} \{I(x+\Delta x, y+\Delta y, z+\Delta t)\} \tag{2}$$

Algorithm 1 3D Median Filtering Based on Histogram

Input : Vector v stores pixel values located in the mask filtering box with size $m \times n \times (T+1)$ centered at (x, y)
Output : med returns the median of v

```
Hist_v = Hist(v)
cumsum = 0 //cumulative function of the histogram
for med=0:bins do
    if Hist_v(med) > 0 then
        cumsum+ = Hist_v(med)
        if cumsum ⩾ (m×n×(T+1))/2 then
            break;
        end if
    end if
end for
Return med
```

where $I(x+\Delta x, y+\Delta y, z+\Delta t)$ are pixel values located in the mask kernel with size $m \times n$ centered at (x, y) over the previous and following T frames around the current frame. Median(.) is the median operation which can be sort-based or selection-based.

A fast separable approximation to the 3D spatio-temporal median filter is to decouple the spatial and temporal median computation which we refer to as M_{2D+t}. In this case, median computations are separated into two steps: first, spatial 2D median filtering using an $m \times n$ kernel size is applied to each image; then the temporal median at each pixel (x, y) is obtained by finding the median over $T+1$ spatial medians (Fig. 1(b)). Note that this is an approximation of the 3D median across a block of $m \times n \times (T+1)$ pixel values inside the filtering 3D kernel. The fast separable median filter approximation for the stationary background estimate is,

$$M_{2D+t}(x,y,z) = \underset{\Delta t \in [-\frac{T}{2}, \frac{T}{2}]}{Median} \{ \underset{\substack{\Delta x \in [-\frac{m-1}{2}, \frac{m-1}{2}] \\ \Delta y \in [-\frac{n-1}{2}, \frac{n-1}{2}]}}{Median} \{I(x+\Delta x, y+\Delta y, z+\Delta t)\} \} \tag{3}$$

The key point is that all of the spatial median operations are computationally independent which can be executed in parallel to improve the performance.

2.1 Median Operation

Median filtering techniques can be grouped as sorting-based or histogram-based. The sort-based approaches need to generate a list of ordered data; then the median value is the middle value of the sorted list. Sorting is computationally expensive and as the size of 3D median filtering box increases, the computational cost of sorting increases significantly. In the best case, the complexity is $O(r^3)$ using bucket sort [29], where r is the filtering kernel radius. But it is still not practical especially in the case of large-scale high resolution image sequences.

The time complexity of the histogram based median computation is much more efficient. The classic Huang's fast 2D median filtering algorithm has $O(r)$ complexity [30]. Recently, Perreault presents a simple, fast median filter sequential implementation that has O(1) constant time [29]. In general, once the cumulative histogram of the median mask filter kernel has been calculated for a pixel, the median value is the first bin histogram which has a greater or equal value than the median index. The median index is defined as half of the median mask filter kernel (i.e. box) size. Algorithm 1 shows the spatio-temporal median filter using the histogram approach. The use of integral histograms make the median filter computation even faster by providing the

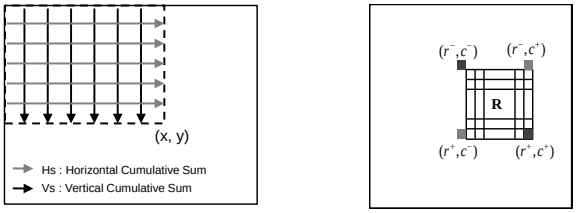

Figure 2: (a) Computation of the histogram up to location (x, y) using a cross-weave horizontal and vertical scan on the image. (b) Computation of the histogram for an arbitrary rectangular region R (origin is the upper-left corner with y-axis.)

local histograms of an arbitrary-sized box kernel in constant time to find the medians or even other statistical moments like mean, variance and standard deviation.

3. SPATIO-TEMPORAL MEDIAN FILTER BASED ON GPU INTEGRAL HISTOGRAM

We propose a fast median operation based on integral histograms and its parallel implementation utilizing many core Graphics Processing Unit (GPU) architectures using the CUDA programming model for motion detection.

3.1 Integral Histogram

The integral histogram is a recursive propagation preprocessing method used to compute local histograms over arbitrary rectangular regions in constant time [31, 32]. The efficiency of the integral histogram approach enables real-time histogram-based exhaustive search in vision tasks [33, 34, 35]. The integral histogram is extensible to higher dimensions and different bin structures. The integral histogram at position (x, y) in the image holds the histogram for all the pixels contained in the rectangular region defined by the top-left corner of the image and the point (x, y) as shown in Figure 2. The integral histogram for the region defined by the spatial coordinate (x, y) and bin variable b is defined as:

$$H(x, y, b) = \sum_{r=0}^{x} \sum_{c=0}^{y} Q(I(r, c), b) \tag{4}$$

where $Q(I(r, c), b)$ is the binning function that evaluates to 1 if $I(r, c) \in b$ for the bin b, and evaluates to 0 otherwise. Sequential computation of integral histograms is described in Algorithm 2. Given the image integral histogram $\mathbf{H}$, computation of the histogram for any test region R delimited by

Algorithm 2 Sequential Integral Histogram

Input : Image $\mathbf{I}$ of size $h \times w$
Output : Integral histogram tensor $\mathbf{IH}$ of size $h \times w \times b$

Initialize IH:
  $\mathbf{IH} \leftarrow 0$
  $\mathbf{IH}(\mathbf{I}(\mathbf{w}, \mathbf{h}), \mathbf{w}, \mathbf{h}) \leftarrow 1$
for z=1:b **do**
  for x=1:w **do**
    for y=1:h **do**
      $\mathbf{IH}(x, y, z) \leftarrow \mathbf{IH}(x-1, y, z) + \mathbf{IH}(x, y-1, z)$
      $-\mathbf{IH}(x-1, y-1, z) + Q(I(x, y), z)$
    end for
  end for
end for
Return $\mathbf{IH}$

Algorithm 3 GIH-STS: GPU Integral Histogram Calculation Using Scan-Transpose-Scan

Input : Image $\mathbf{I}$ of size $h \times w$, number of bins b
Output : Integral histogram tensor $\mathbf{IH}$ of size $b \times h \times w$

Initialize IH
  $\mathbf{IH} \leftarrow 0$
  $\mathbf{IH}(\mathbf{I}(\mathbf{w}, \mathbf{h}), \mathbf{w}, \mathbf{h}) \leftarrow 1$
for all $b \times h$ blocks in parallel **do**
  //horizontal cumulative sums
  Prescan(IH)
end for
//transpose the histogram tensor
$\mathbf{IH^T} \leftarrow$ 3D_Transpose($\mathbf{IH}$)
for all $b \times w$ blocks in parallel **do**
  //vertical cumulative sums
  Prescan($\mathbf{IH^T}$)
end for
Return 3D-Transpose($\mathbf{IH^T}$)

points $\{(r^-, c^-), (r^-, c^+), (r^+, c^+), (r^+, c^-)\}$ reduces to the combination of four integral histograms:

$$\begin{aligned} h(R, b) = \; & H(r^+, c^+, b) - H(r^-, c^+, b) \\ & - H(r^+, c^-, b) + H(r^-, c^-, b) \end{aligned} \tag{5}$$

Figure 2 illustrates the notation and accumulation of integral histograms using vertical and horizontal cumulative sums (prescan), which is used to compute regional histograms.

However, the intrinsic parallel characteristics of the integral histogram motivates further improvements in speed up by parallelizing it for high performance computing configurations. Bellens, *et al.* developed parallel implementations of the integral histogram for Cell/B.E. processors [32] and we describe a GPU version in [36].

3.2 Parallelized Integral Histogram

The cross-weave scan mode (Fig. 2), enables cumulative sum tasks over rows (or columns) to be scheduled and executed independently allowing for inter-row and column parallelization. In fact the integral histogram computations can be divided into two prescans over the data. First, a horizontal prescan that computes cumulative sums over rows of the data, followed by a second vertical prescan that computes cumulative sums over the columns of the first scan output. Taking the transpose of the horizontally prescanned image histogram, enables us to reapply the same (horizontal) prescan algorithm effectively on the columns of the data.

An image with dimensions $h \times w$ produces an integral histogram tensor of dimensions $h \times w \times b$, where b is the number of bins in the histogram. We have mapped the histogram tensor to a 1D row major ordered array for efficient access as shown in Figure 3. In order to reduce the communication overhead between host and device, data transfer is limited to send an image to the GPU and the final integral histogram tensor back to the CPU. There is no other data back and

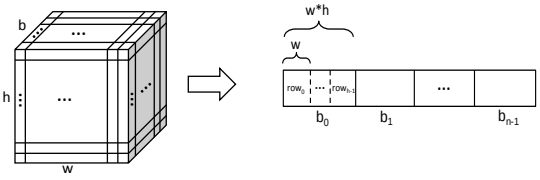

Figure 3: Integral histogram tensor represented as 3D array data structure (left), and equivalent 1D array mapping (right).

```
#define BLOCK_DIM 32
void Transpose3D (int* TArray, int* Array, int w, int h, int b)
{     dim3 grid(b, w/BLOCK_DIM, h/BLOCK_DIM);
      dim3 threads(BLOCK_DIM ,BLOCK_DIM ,1);

       3DTranspose_kernel <<< grid,threads >>> (T_Array, Array, w, h);
 }
__global__ void 3DTranspose_kernel (int *odata, int *idata, int width, int height)
{
__shared__ int block[BLOCK_DIM][BLOCK_DIM+1];
// read the matrix tile into shared memory
unsigned int xIndex = blockIdx.y * BLOCK_DIM + threadIdx.x;
unsigned int yIndex = blockIdx.z * BLOCK_DIM + threadIdx.y;
if ( (xIndex < width) && (yIndex < height) )
{unsigned int index_in = blockIdx.x * width * height + yIndex * width + xIndex;
 block[threadIdx.y][threadIdx.x] = idata[index_in];
}
__syncthreads();
// write the transposed matrix tile to global memory
xIndex = blockIdx.z * BLOCK_DIM + threadIdx.x;
yIndex = blockIdx.y * BLOCK_DIM + threadIdx.y;
if ( (xIndex < height) && (yIndex < width) )
{unsigned int index_out = blockIdx.x * width * height + yIndex * height + xIndex;
 odata[index_out] = block[threadIdx.x][threadIdx.y];
}
}
```

Figure 4: CUDA 3D transpose kernel code

forth between CPU and GPU until the integral histogram calculations have been complected on the GPU.

Our GPU implementation of integral histogram [36] which is described in Algorithm 3 combines cross-weave scan mode with an efficient parallel prefix sum operation. Harris, *et al.* present an efficient implementation of *all-prefix-sums* operation using the CUDA programming model [37], that can be used to implement Algorithm 3. For an array of size n the prescan operation requires only $O(n)$ operations: $2\times(n-1)$ additions and $(n-1)$ swaps. We apply prefix-sums to the rows of the histogram bins (horizontal cumulative sums or prescan), then transpose the array and reapply the prescan to the rows to obtain the integral histograms. In order to allow a single transpose operation, we extended the optimized 2-D transpose kernel described in [38] to a 3D transpose kernel by using the bin offset in the indexing (Fig. 4). The optimized transpose kernel uses zero bank conflict shared memory and guaranties that global reads and writes are coalesced.

Our GPU integral histogram implementation benefits from minimum data transfer between CPU and GPU, double buffering and high GPU utilization. The number of threads is automatically determined based on the image size to ensure maximum occupancy per kernel (for more info refer to [36]).

3.3 3D Median Filter Based on Integral Histogram

The integral histogram of an image provides the local integral histogram of every arbitrary target region in constant time (equation 5). Taking advantages of this property enables us to compute the medians in constant time for all target pixels using its local integral histogram. Algorithm 4 shows spatio-temporal median computations using the integral histogram (assuming image sequences of size k which are transferred to the GPU using double buffering and a spatio-temporal median filter of size $m \times n \times (T+1)$). In the initialization phase, the individual integral histograms are computed for each image array. Meanwhile the joint integral histogram is computed for the first $T+1$ individual integral histograms (assume that T is an even number). After creating the joint integral histogram, the median calculation phase started for frame $\frac{T}{2}+2$. In each iteration, first, the joint integral histogram is updated by adding the integral histogram of head image and subtracting the integral histogram of tail image (Fig. 5). Then medians are computed for each pixel using the joint local integral histograms of kernel size $m \times n$ where local kernel integral histograms can be obtained in $O(1)$.

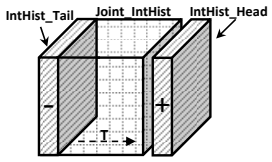

Figure 5: Updating the joint integral histogram for median filtering.

Algorithm 4 3D Median Filter Based on Integral Histogram

Input : Image sequences $\mathbf{I}[\mathbf{k}]$ of size $h \times w$,
number of bins b,
size of image history $T+1$

Output : Medians $\mathbf{M}[\mathbf{k}-\mathbf{T}]]$ of size $h \times w$

Initialize JointIH
IH_tail=JIH=Integral_Hist(Quantized(image(1)));
//Compute the first joint integral histogram for the first $T+1$ frames
for $Fr = 2 : T+1$ **do**
  $IH[Fr] = Integral\_Hist(Quantized(image(Fr)))$;
  JIH = JIH + IH[Fr];
end for
//Calculate the Median of current frame
for $Fr = T+2 : k$ **do**
  //Update the IH_head
  IH_head=Integral_Hist(Quantized(image(Fr)));
  //Update the Joint Integral Histogram
  JIH= JIH + IH_head - IH_tail;
  //Update the IH_tail
  IH_tail=Integral_Hist(Quantized(image(Fr-T)));
  //Compute Median $[Fr - \frac{T}{2}]$
  Median $[Fr - \frac{T}{2}]$ = ComputeLocalMedian(JIH)
end for

4. EXPERIMENTS

We implemented and evaluated background subtraction using spatio-temporal median filter based on CPU and parallel GPU integral histogram implementations. Our experiments were conducted on a 2.0 GHz Quad Core Intel CPU (Core i7-2630QM) and two GPU cards: an NVIDIA Tesla C2070 and an NVIDIA GeForce GTX 460. The former is equipped with fourteen 32-core SMs and has about 5GB of global memory, 48KB shared memory with compute capability 2.0. The latter consists of seven 48-core SM and is equipped with 1GB global memory, 48KB shared memory with compute capability 2.1. The parallel GPU integral histogram implementation is based on three kernel invocations: a single horizontal scan, a 3D transpose, and a vertical scan. The integral histogram computations start after transferring the image to the GPU, complete the calculation of the integral histogram on the GPU then transfer the final integral histogram tensor back to the CPU (this has

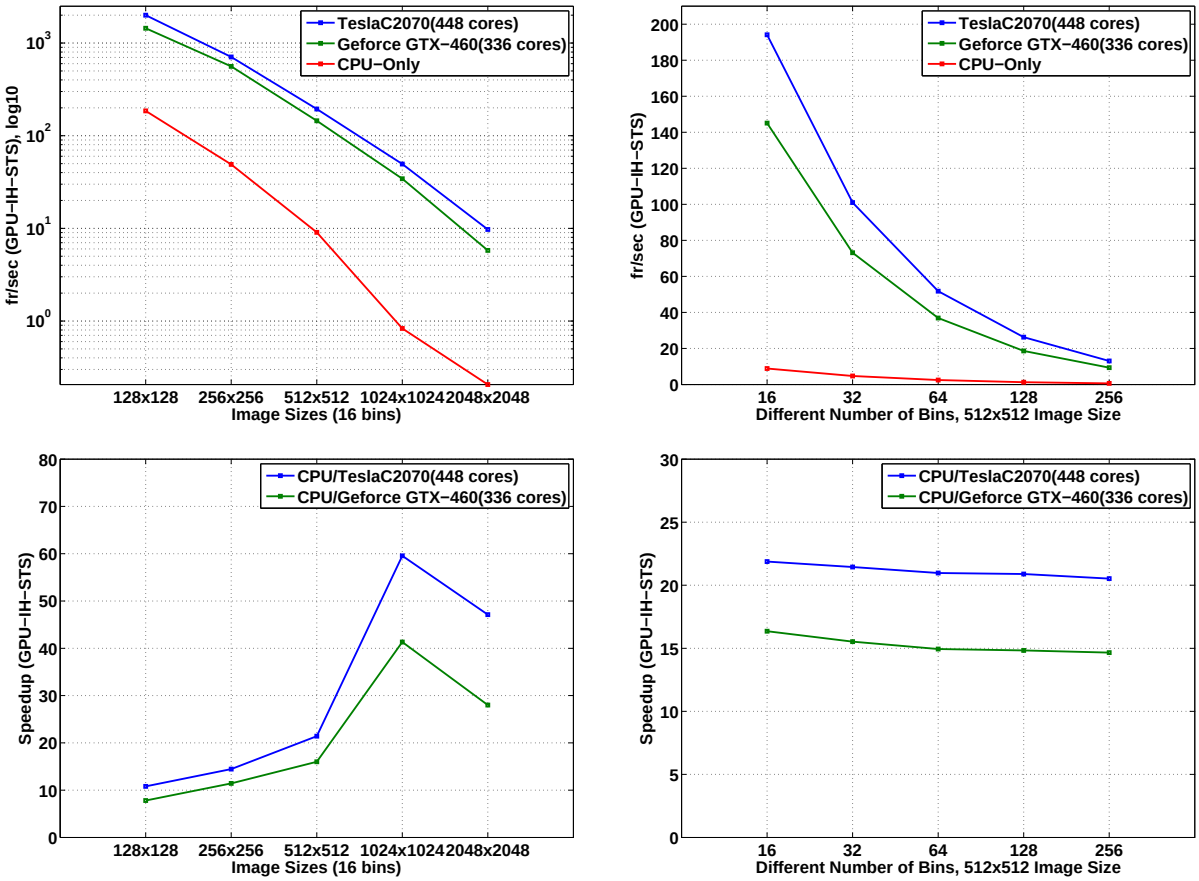

Figure 6: UP: Frame rate of our 3D median filter based on GPU integral histogram and CPU-only integral histogram implementations: (UL) frame rate for different image sizes and 16 bins, (UR) frame rate for 512x512 image size and different number of bins; Bottom: Speedup of proposed 3D median filter based on GPU integral histogram over CPU using two NVIDIA graphic cards, (BL) Speedup for different image sizes and 16 bins with respect to CPU-only, (BR) Speedup with varying number of bins for 512x512 image size.

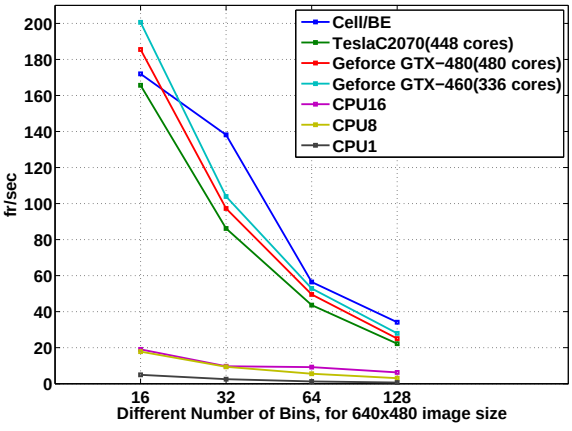

Figure 7: Frame rate performance comparison of the GPU design versus CPU implementation using different degrees of multi-threading, CPU1, CPU8, CPU16 and Cell/B.E. performance results presented for wavefront scan mode using 8 SPEs in[32].

the least communication overhead and the most GPU utilization).Figure 6 summarizes the frame rate performance and speed-up of the GPU implementations compared to the sequential CPU-only implementation. The frame rate is defined as the maximum number of images processed per second. Since we use double buffering, the frame rate equals $(kernel\_execution\_time)^{-1}$ for compute bound cases, or $(data\_transfer\_time)^{-1}$ for data-transfer bound cases. Considering double buffering timing, our 3D median filter based on GPU integral histogram achieves 194 fr/sec to compute 16-bin integral histogram for a 512×512 image (21 times faster than CPU-only implementation) and 60 times faster than the CPU version for $1K \times 1K$ image reaching 49 fr/sec using the NVIDIA Tesla C2070 GPU. Figure 7 compares the frame rate performance of our GPU integral histogram implementation using different GPU hardware, with a sequential CPU implementation using different degrees of multi-threading: CPU1 (1 thread), CPU8 (8 threads), CPU16 (16 threads). We have compared our performances to the best performance of the IBM Cell/B.E integral histogram parallel implementation computation using wavefront scan mode [32]. In most cases, our performance is data-transfer-bound: the achieved speedup is limited by transferring the data between CPU and GPU over the PCI-express interconnect,

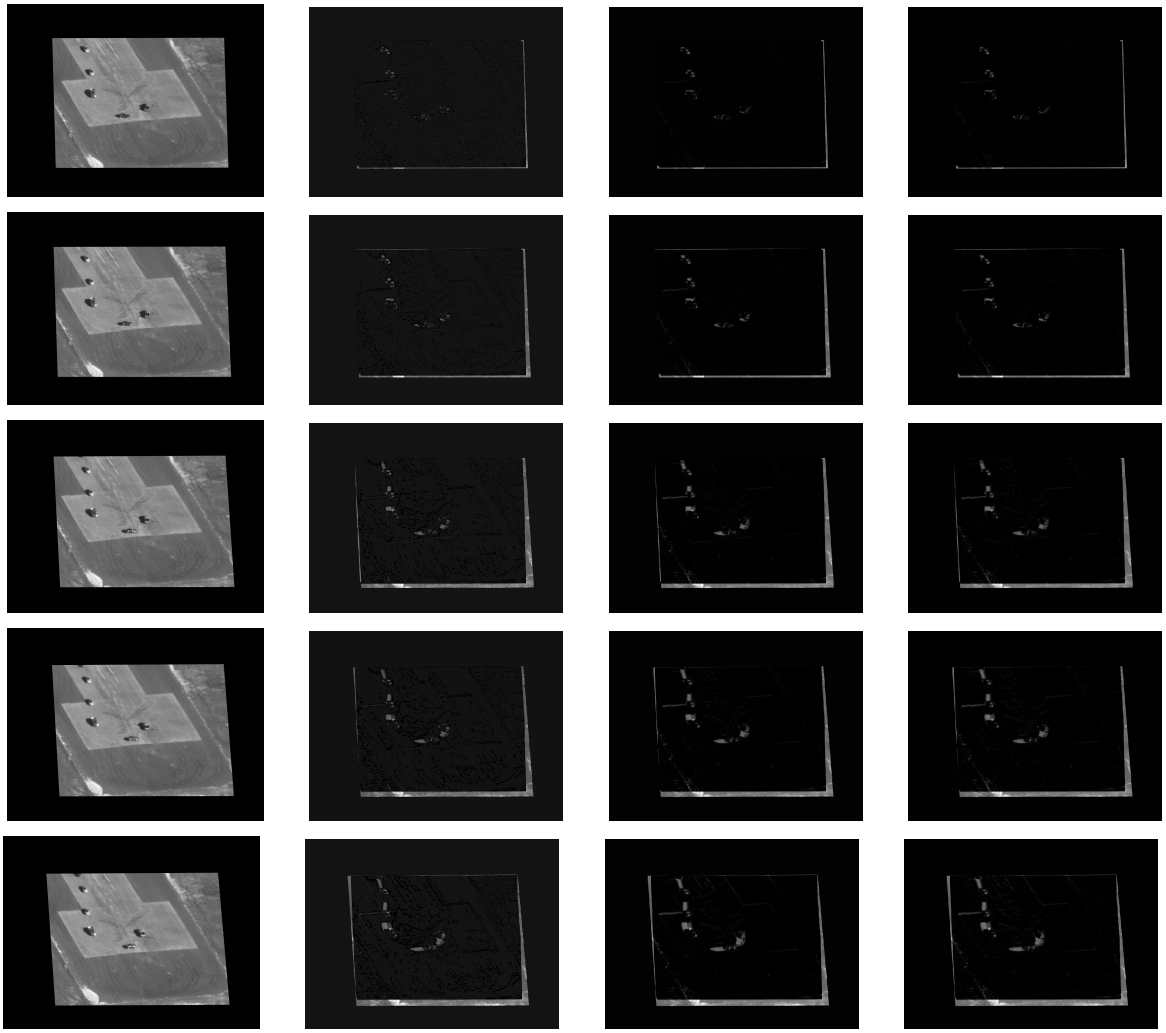

Figure 8: Background subtraction for a sample DARPA VIVID stabilized aerial sequence using 3D median filter based on GPU integral histogram using different number of bins, from left column to the right column: original image, foreground using 16 bins integral histogram, foreground using 128 bins integral histogram, foreground using 256 bins integral histogram.

rather than from the parallel execution on the GPU. Figure 8 shows sample background subtraction results for a DARPA VIVID stabilized aerial sequence using the 3D median filter based on GPU integral histogram using different number of bins (16, 128 and 256). Figure 9 shows sample background subtraction results for two image modalities visible and IR from the PETS2013 Benchmark background dataset (first three rows) [39], and indoor hallway motion from the OTCBVS Benchmark Dataset [40] (last three rows), using the 3D median filter based on GPU integral histogram with varying number of bins (16, 128 and 256). These results suggest that in general for natural images with large dynamic ranges using a lower number of bins in computing the integral histogram does not adversely degrade the detection results.

5. CONCLUSIONS

Although the median filter is widely used in computer vision tasks, it is computationally expensive which limits its application for large scale, high resolution imagery and real-time requirements. We showed that an efficient GPU-based parallel implementation of the spatio-temporal median filter can be achieved using integral histograms. The median filter based on GPU integral histograms not only takes advantage of fast GPU processing but also enables us to compute the median across multiple scales for all target pixels in constant time. Results show the frame rate of the GPU implementation was 60 times faster than the CPU version for a $1K \times 1K$ image reaching 49 fr/sec and 21 times faster for 512×512 frame sizes reaching 194 fr/sec. This opens up the possibility for speeding up a variety of computer vision detection tasks.

6. ACKNOWLEDGMENTS

This research was partially supported by U.S. Air Force Research Laboratory (AFRL) under agreement AFRL FA8750-11-1-0073. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of AFRL or the U.S. Government. The U.S. Govt. is authorized to reproduce and distribute reprints for Govt. purposes notwithstanding any copyright notation thereon.

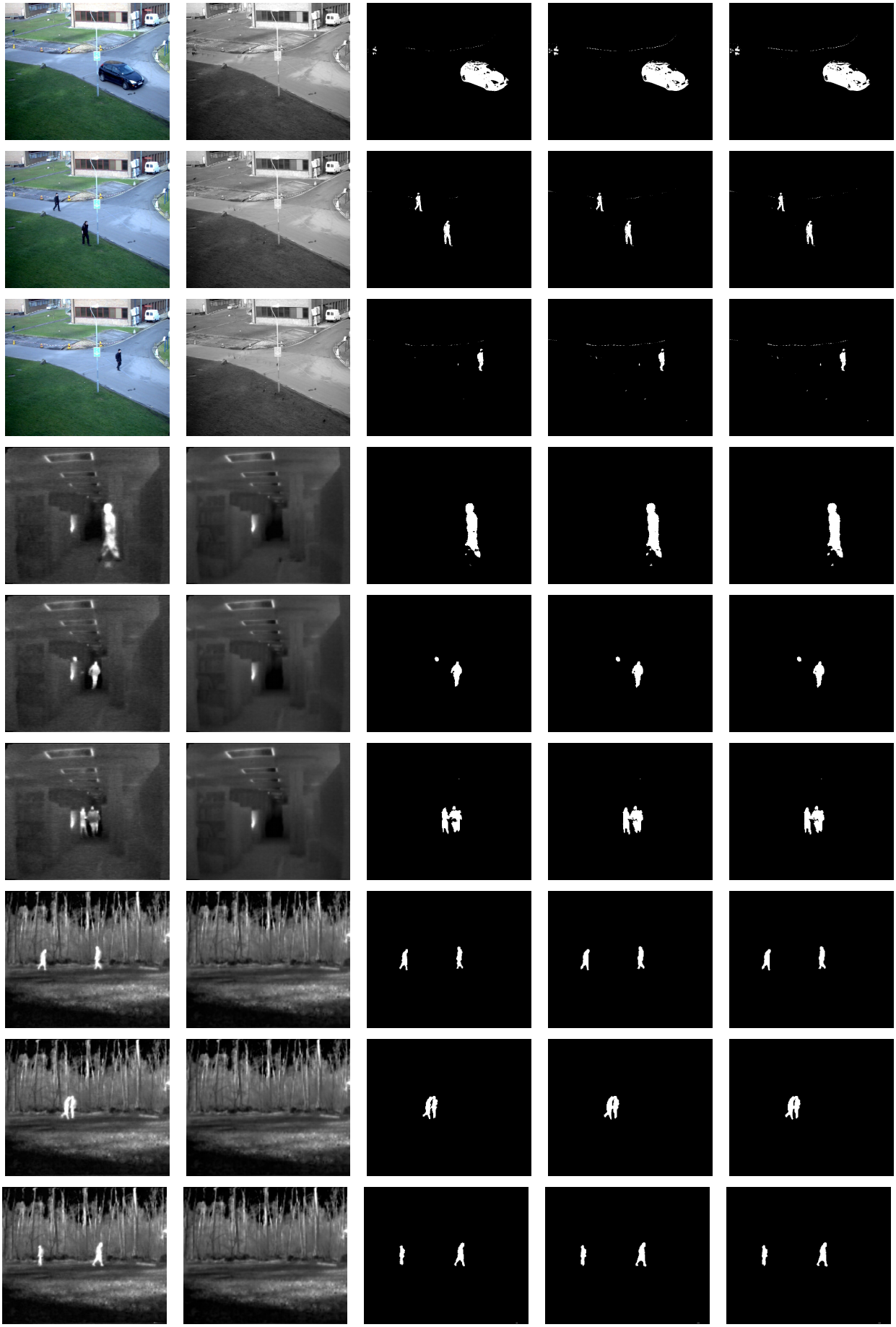

Figure 9: Background subtraction for PETS2013 Benchmark background dataset (first row frame 73, second row frame 161, third row frame 195) [39], an indoor hallway motion (seq. irin01) from OTCBVS Benchmark IR Database(row four frame 5150, row five frame 8290 and row six frame 8329) [40], and outdoor motion and tracking scenarios (seq. irw01) from OTCBVS Benchmark IR Database (row seven frame 332, row eight frame 399, row nine frame 480)[40] using 3D median filter based on GPU integral histogram using different number of bins, from left column to the right column: original image, background model (256 bins), foreground using 16 bins integral histogram, foreground using 128 bins integral histogram, foreground using 256 bins integral histogram.

7. REFERENCES

[1] SC Cheung and C. Kamath, "Robust techniques for background subtraction in urban traffic video," in *Proc. SPIE Visual Communications and Image Processing*, 2004, vol. 5308, pp. 881–892.

[2] A. Elgammal, R. Duraiswami, D. Harwood, and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *IEEE Proceedings*, vol. 90, no. 7, pp. 1151–1163, 2002.

[3] E. Maggio and A. Cavallaro, *Video Tracking: Theory and Practice*, Wiley, 2011.

[4] S. S. Cheung and C. Kamath, "Robust background subtraction with foreground validation for urban traffic video," *EURASIP Journal on Applied Signal Processing*, vol. 14, pp. 2330–2340, 2005.

[5] R. Cucchiara, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts, and shadows in video streams," *IEEE Trans. Pattern Analysis Machine Intelligence*, vol. 25, no. 10, pp. 1337–1342, 2003.

[6] M. Piccardi, "Background subtraction techniques: a review," in *IEEE Int. Conf. Systems, Man and Cybernetics*, 2004, vol. 4, pp. 3099–3104.

[7] JCS Jacques, C.R. Jung, and S.R. Musse, "Background subtraction and shadow detection in grayscale video sequences," in *SIBGRAPI 2005*. IEEE, 2005, pp. 189–196.

[8] P. Viola and M.J. Jones, "Robust real-time face detectin," *Int. J. Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[9] C. Stauffer and W.E.L. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE CVPR 1999*, 1999, vol. 2.

[10] K.P. Karmann, "Moving object recognition using an adaptive background memory," *Proc. Time Varying Image Processing*, 1990.

[11] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in *IEEE ICCV 1999*, 1999, vol. 1, pp. 255–261.

[12] D. Gutchess, M. Trajkovics, E. Cohen-Solal, D. Lyons, and A.K. Jain, "A background model initialization algorithm for video surveillance," in *IEEE ICCV 2001*, 2001, vol. 1, pp. 733–740.

[13] M. H. Kolekar, K. Palaniappan, S. Sengupta, and G. Seetharaman, "Event detection and semantic identification using Bayesian belief networks," 2009, pp. 554–561.

[14] M. H. Kolekar, K. Palaniappan, S. Sengupta, and G. Seetharaman, "Semantic concept mining based on hierarchical event detection for soccer video indexing," *J. Multimedia*, vol. 4, no. 5, pp. 298–312, October 2009.

[15] M. H. Kolekar, K. Palaniappan, and S. Sengupta, "Semantic event detection and classification in cricket video sequence," in *IEEE Indian Conference on Computer Vision, Graphics and Image Processing*, 2008, pp. 382–389.

[16] K. *et al.* Palaniappan, "Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video," in *13th Conf. Information Fusion*, 2010, pp. 1–8.

[17] R. Pelapur and *et al.*, "Persistent target tracking using likelihood fusion in wide-area and full motion video sequences," in *15th Int. Conf. Information Fusion*, 2012.

[18] I. Ersoy, K. Palaniappan, G. Seetharaman, and R. Rao, "Interactive tracking for persistent wide-area surveillance," in *Proc. SPIE Conf. Geospatial InfoFusion II (Defense, Security and Sensing: Sensor Data and Information Exploitation)*, 2012, vol. 8396.

[19] I. Ersoy, K. Palaniappan, and G. Seetharaman, "Visual tracking with robust target localization," in *IEEE Int. Conf. Image Processing*, 2012.

[20] I.Ersoy, F.Bunyak, M.A.Mackey, and K.Palaniappan, "Cell segmentation using Hessian-based detection and contour evolution with directional derivatives," in *IEEE Int. Conf. Image Processing*, Oct. 2008, pp. 1804–1807.

[21] F. Bunyak, K. Palaniappan, O. Glinskii, V. Glinskii, V. Glinsky, and V. Huxley, "Epifluorescence-based quantitative microvasculature remodeling using geodesic level-sets and shape-based evolution," in *30th Int. IEEE Engineering in Medicine and Biology Society Conf. (EMBC)*, Vancouver, Canada, Aug. 2008, pp. 3134–3137.

[22] K. Palaniappan, I. Ersoy, and S. K. Nath, "Moving object segmentation using the flux tensor for biological video microscopy," *Lecture Notes in Computer Science (PCM)*, vol. 4810, pp. 483–493, 2007.

[23] A. Mosig, S. Jaeger, W. Chaofeng, I. Ersoy, S. K. Nath, K. Palaniappan, and S.S. Chen, "Tracking cells in live cell imaging videos using topological alignments," *Algorithms in Molecular Biology*, vol. 4, pp. 10p, 2009.

[24] K. Palaniappan, H. S. Jiang, and T. I. Baskin, "Non-rigid motion estimation using the robust tensor method," in *IEEE CVPR Workshop on Articulated and Nonrigid Motion*, Washington DC, USA, June 27–July 2 2004, vol. 1, pp. 25–33.

[25] K. Palaniappan, I. Ersoy, G. Seetharaman, S.R. Davis, P. Kumar, R.M. Rao, and R. Linderman, "Parallel flux tensor analysis for efficient moving object detection," in *Int. Conf. Information Fusion*, 2011, pp. 1–8.

[26] P. Kumar, K. Palaniappan, A. Mittal, and G. Seetharaman, "Parallel blob extraction using the multi-core cell processor," in *Advanced Concepts for Intelligent Vision Systems*. Springer, 2009, pp. 320–332.

[27] S. Mehta, A. Misra, A. Singhal, P. Kumar, A. Mittal, and K. Palaniappan, "Parallel implementation of video surveillance algorithms on GPU architectures using CUDA," in *17th IEEE Int. Conf. Advanced Computing and Communications (ADCOM)*, 2009.

[28] M. Jiang and D. Crookes, "High-performance 3d median filter architecture for medical image despeckling," *Electronics Letters*, vol. 42, no. 24, pp. 1379–1380, 2006.

[29] S. Perreault and P. Hébert, "Median filtering in constant time," *IEEE Transactions on Image Processing*, vol. 16, no. 9, pp. 2389–2394, 2007.

[30] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 27, no. 1, pp. 13–18, 1979.

[31] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *IEEE CVPR*, 2005, vol. 1, pp. 829–836.

[32] P. Bellens, *et al.*, "Parallel implementation of the integral histogram," *LNCS (ACIVS)*, vol. 6915, pp. 586–598, 2011.

[33] F. Porikli, "Constant time o (1) bilateral filtering," in *IEEE CVPR 2008*, 2008, pp. 1–8.

[34] Q. Yang, K.H. Tan, and N. Ahuja, "Real-time o (1) bilateral filtering," in *IEEE CVPR 2009*, 2009, pp. 557–564.

[35] A. Adams, J. Baek, and M.A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," in *Computer Graphics Forum*. Wiley Online Library, 2010, vol. 29, pp. 753–762.

[36] M. Poostchi, K. Palaniappan, F. Bunyak, M. Becchi, and G. Seetharaman, "Efficient GPU implementation of the integral histogram," in *LNCS ACCV, Workshop on Developer-Centred Computer Vision*, 2012.

[37] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," in *GPU Gems*, vol. 3, chapter 39, pp. 851–876. 2007.

[38] G. Ruetsch and P. Micikevicius, "Optimizing matrix transpose in CUDA," *Nvidia CUDA SDK Application Note*, 2009.

[39] *PETS 2013 Benchmark Data*, 2012, `http://www.cvg.rdg.ac.uk/PETS2013/a.html#s3`.

[40] Roland Miezianko, *IEEE OTCBVS WS Series Bench; Terravic Research Infrared Database.*