NVIDIA
CUDA™ RESEARCH CENTER

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

Tracking Objects Better, Faster, Longer

Assoc. Prof. Dr. Alptekin Temizel
atemizel@metu.edu.tr
Graduate School of Informatics, METU

18 March 2015
GPU Technology Conference

Video Object Tracking

- Real-time tracking of objects in video is an important problem in various domains such as
  - Robotics
  - Defense
  - Security
  - Immersive applications

- Many studies in the literature are based on short term tracking which often fails if the object is:
  - Occluded
  - Disappears from the field of view
  - Changes its appearance rapidly
  - Goes through a large displacement between consecutive frames.

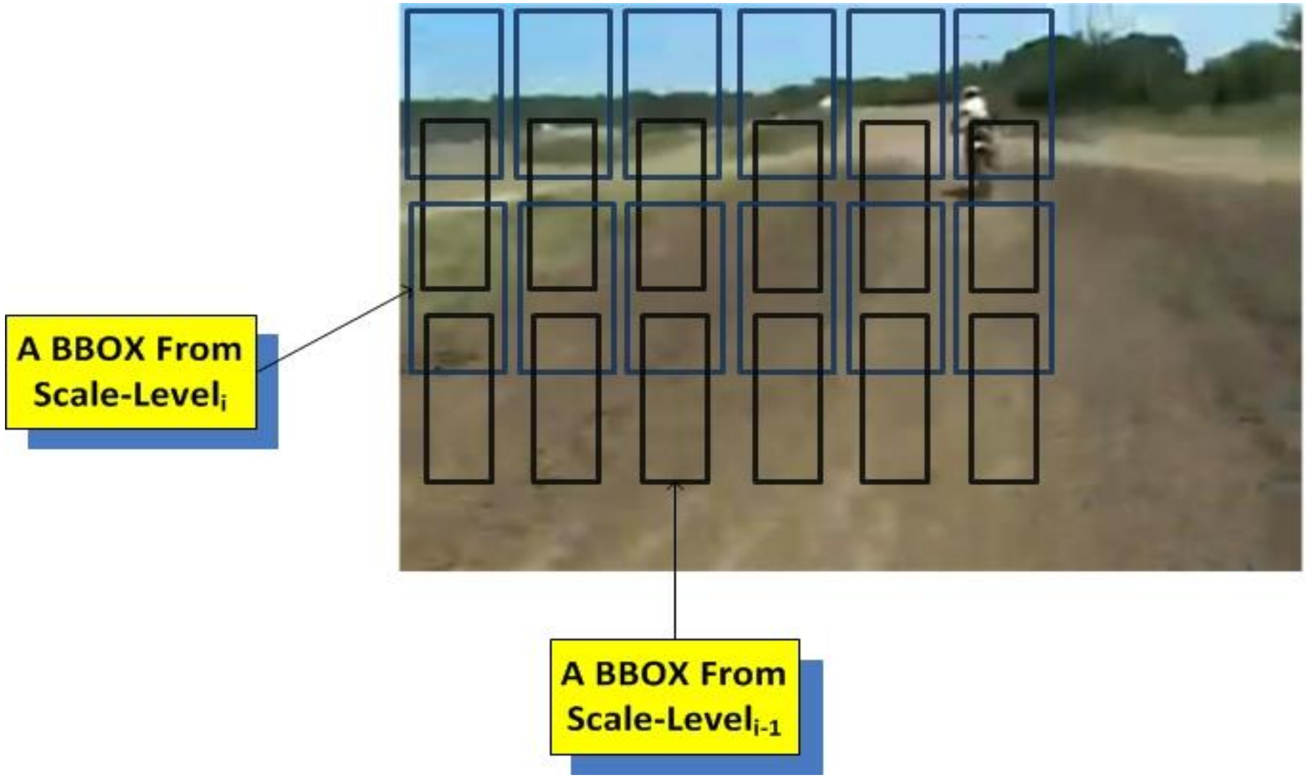
Tracking-**L**earning-**D**etection

- **Track** the object in real-time
  - The object location is expected to be provided by the tracker in most cases.
- **Learn** its appearance
  - The predicted location of the object is used by P-N experts in the learning component.
- **Detect** when it reappears after an occlusion or disappearance
  - when the detector has higher confidence than the tracker, the object is assumed to be at the location estimated by the detector and the tracker is reinitialized with this result.

Motivations for Optimization

- Increase the resolutions for which the algorithm can run in real-time,
- Allow running multiple instances of the algorithm to support multiple object tracking,
- Allow running the algorithm at higher accuracy.
  - Tuning the algorithm parameters for higher tracking accuracy requires higher computation power,

Computational Cost



Detector needs to check 30.000 Bounding Boxes even in a 320x240 frame!

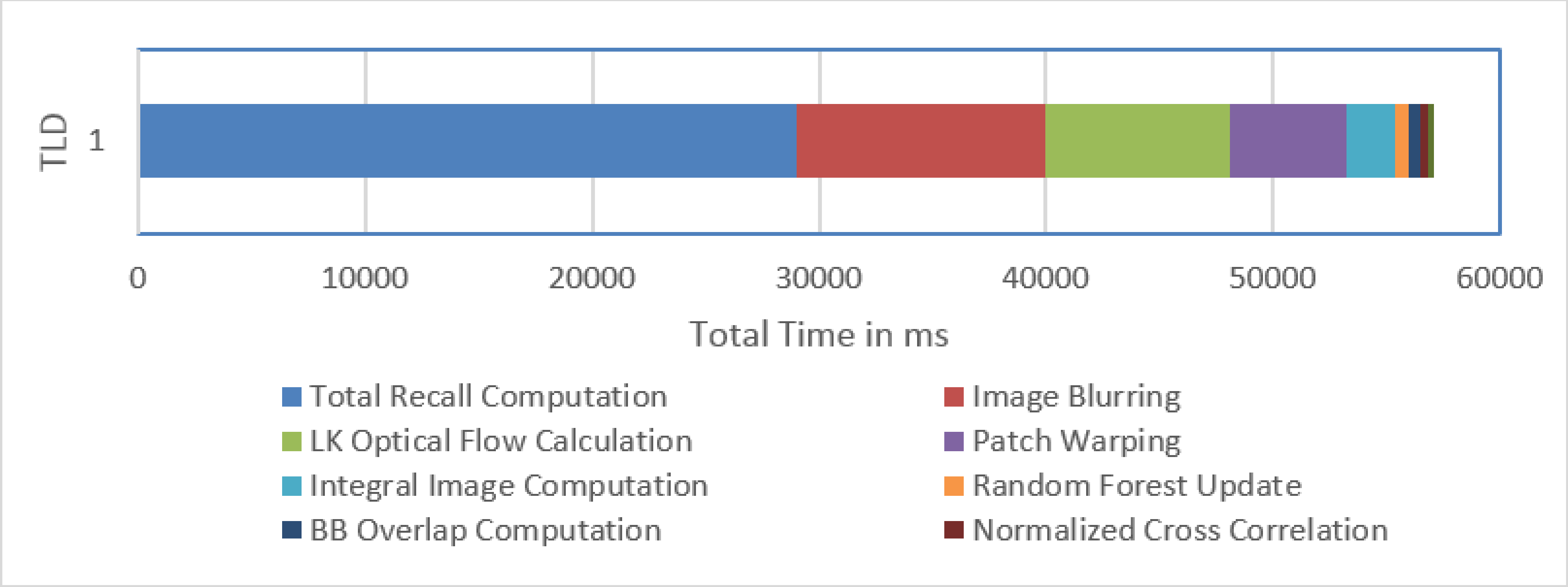
Test Platform

Operating System	Windows 7 x64
CPU	Intel i7 4770K 3.5 GHz, 4 Physical Cores, Hyper Threading Factor is 2
GPU	Tesla K40c, Compute Capability 3.5 15 Streaming Multiprocessors (SM) 192 Cores per SM (total of 2880 cores) 2 Async. Copy Engine, Hyper-Q Enabled
RAM	32 GB DDR3
Serial Computer Expansion Bus	PCIe 2.1
CUDA Toolkit	6.0
CUDA Driver Version	6.0
CUDA Run time Version	6.0
OpenCV Version	2.4.9
OpenMP Version	2.0

Analysis for various video resolutions

Component	Time per call (ms)			Time for whole sequence (ms)		
	480x270	960x540	1920x1080	480x270	960x540	1920x1080
Tracking						
LK Optical Flow	1.100	4.280	17.520	509	1982	8112
Normalized Cross Corr.	0.620	0.630	0.770	287	292	357
Learning						
Pattern Generation	0.010	0.020	0.080	32	65	258
Random Forest Update	0.440	1.200	1.890	141	386	608
Patch Warping	0.080	0.230	1.270	326	938	5180
BB Overlap	0.020	0.060	0.270	35	104	467
Detection						
Total Recall	5.930	20.400	62.500	2752	9466	29000
Integral Image	0.271	1.100	4.560	126	510	2116
Image Blurring	1.685	6.509	23.649	782	3021	10974

Analysis for 1920x1080 video



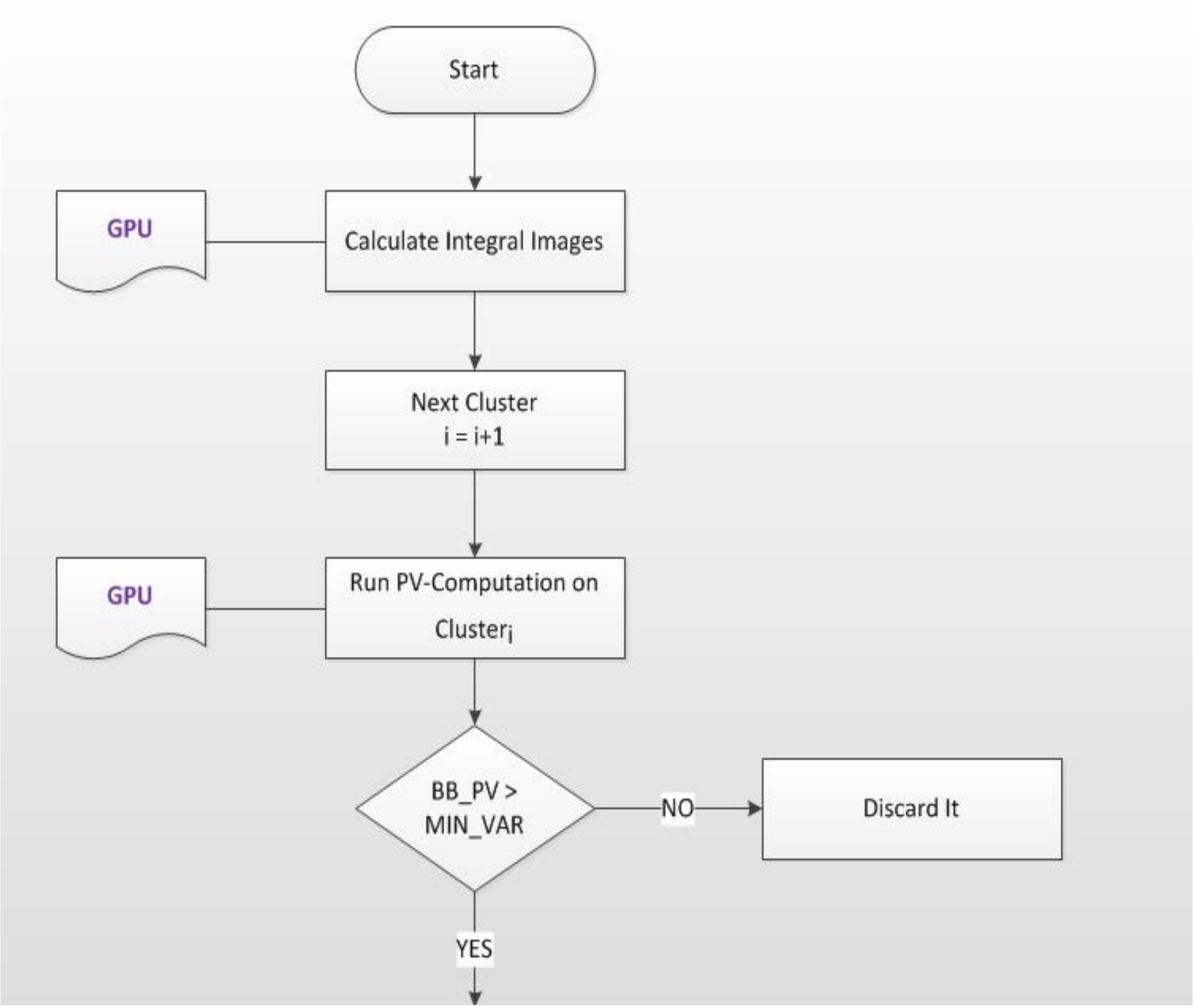
- ❑ Heterogeneous implementation
  - ❑ serial parts are run asynchronously on the CPU
  - ❑ The most computationally costly parts are parallelized on the GPU
- ❑ Apply stream compaction
- ❑ Design the data structures to allow coalesced access
- ❑ Use shared memory whenever suitable.
- ❑ Load balancing - this is achieved by the proposed grouping of the data.

- Lucas-Kanade Optical Flow
- Pyramidal Lucas-Kanade is used to handle large motion
- Open-CV’s GPU Module which has a large community support has been adopted

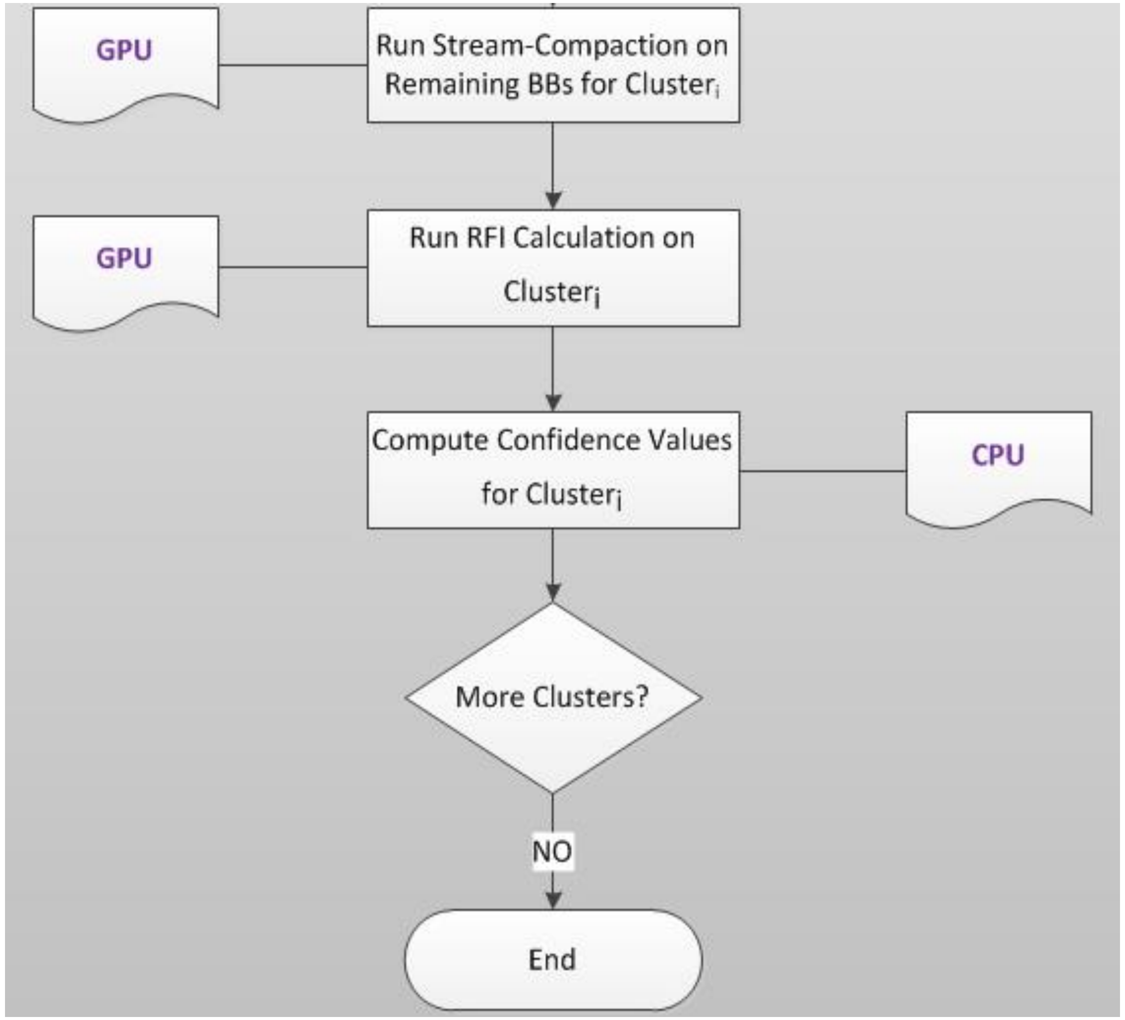
Implementation: Learning

- Patch Warping is the most computationally expensive part.
- The other parts do not take significant processing time as they involve calculation for a limited number of BBs and learning is invoked intermittently. As such, implementation of these parts on GPU were considered infeasible.
- Processing these parts on the CPU while processing patch warping on the GPU necessitates moving large amounts of data (i.e. warped patches) between CPU and GPU.
- As a result, we have decided to keep the learning component purely on CPU.

Implementation: Detection

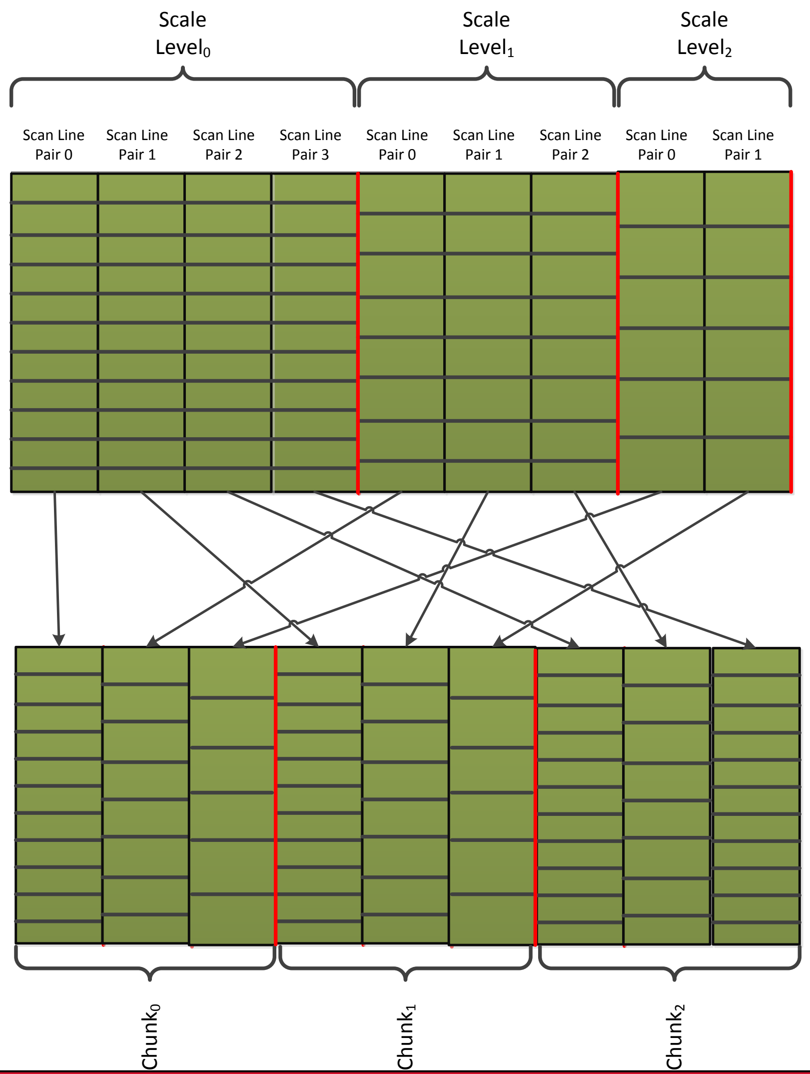


Implementation: Detection



Load Balancing for Patch Variance Calculation

- Ensure chunks to have similar number of BBs to be processed.
- Exploitation of spatial locality of BBs is also important.



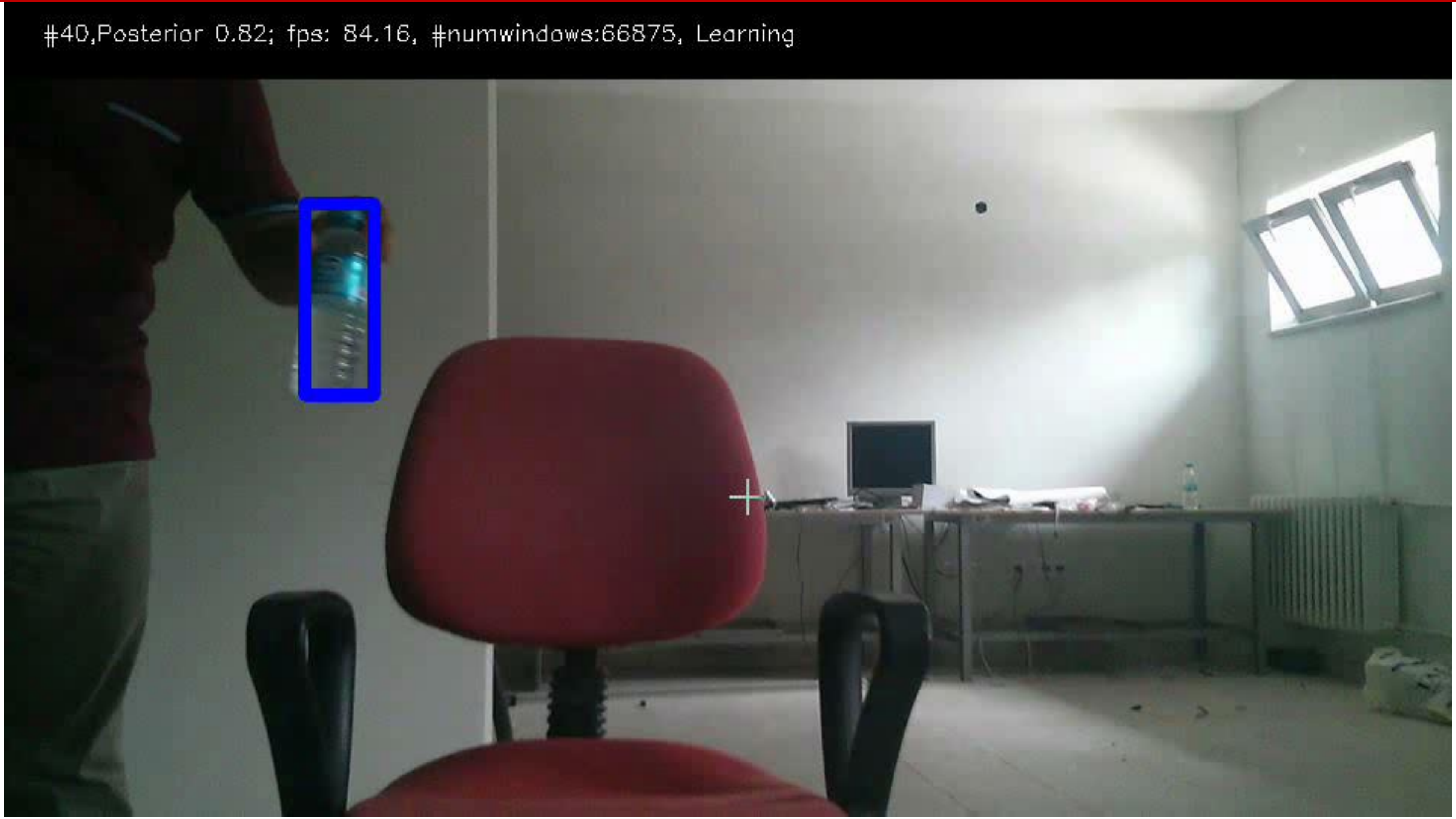
Stream Compaction

- Patches having low variance (marked with -1) need not to be transferred to the CPU
- Stream compaction is performed by calculating the shift amounts by prefix-sum

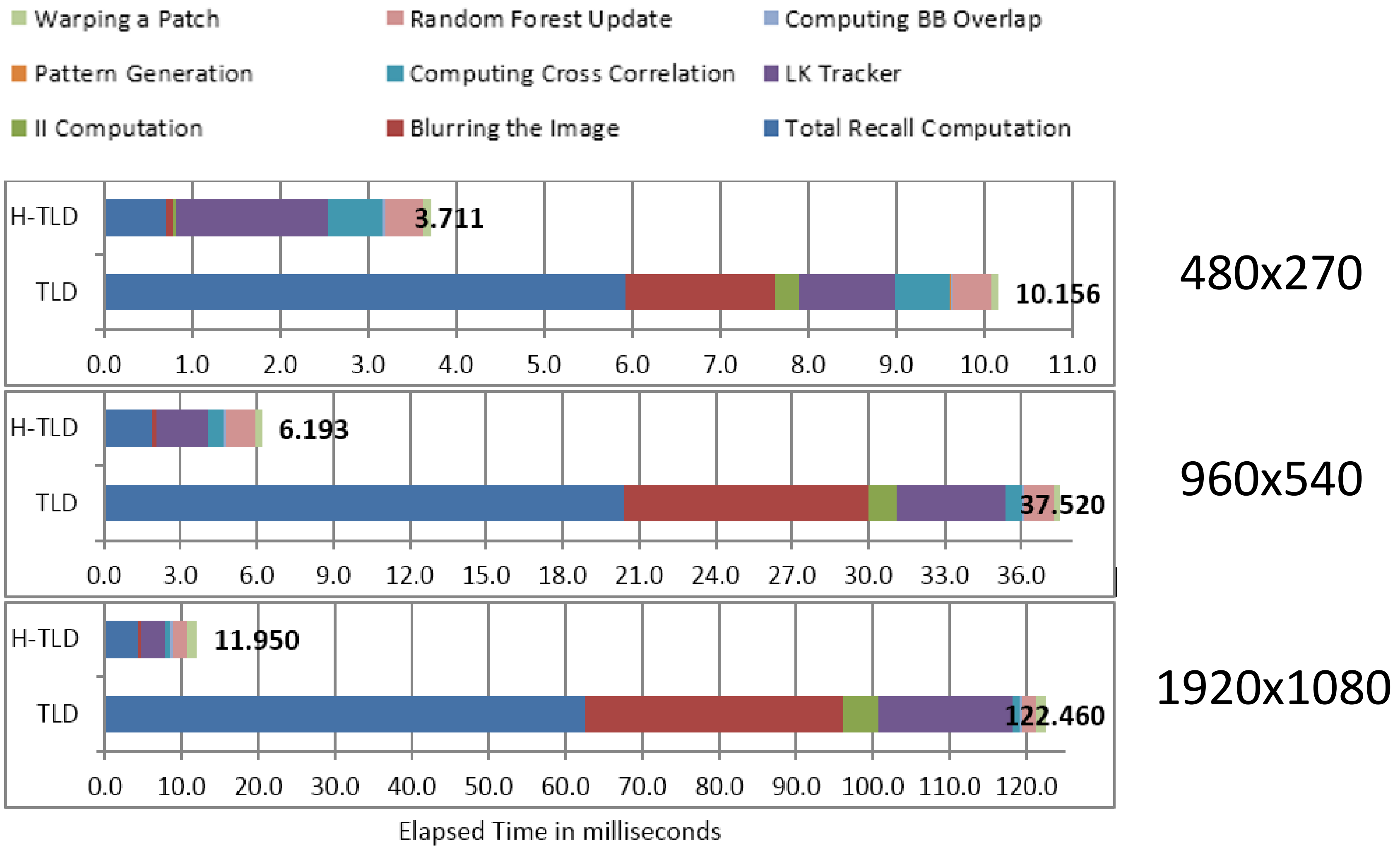
BB_0	BB_1	BB_2	BB_3	BB_4	BB_5	BB_6
0	-1	-1	0	-1	0	-1

BB_0	BB_1	BB_2	BB_3	BB_4	BB_5	BB_6
0	-1	-2	-2	-3	-3	-4

Results



Experimental Results



Discussion

- The main bottleneck is the data transfers between the CPU and GPU memory spaces.
- A further analysis of the framework reveals that approximately 45% of total recall calculation time is spent on RFI part; and approximately 78% of the RFI Calculation’s time is spent in moving the calculated RFIs to the host side.
- If this data transfer could have been eliminated, a theoretical speed-up bound of 13.13x at 1920x1080 resolution would be obtained.
- This theoretical analysis shows the potential impact of expected memory bandwidth enhancements and speed-up of data transfers between CPU and GPUs in the next generation architectures.

H-TLD library code repository

https://github.com/iliTheFallen/htld

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App.
Your feedback is important!

For further enquiries:

Dr. Alptekin Temizel
http://www.metu.edu.tr/~atemizel/
atemizel@metu.edu.tr