Trade-off Between GPGPU based Implementations of Multi Object Tracking Particle Filter

Petr Jecmen[1], Frederic Lerasle[2] and Alhayat Ali Mekonnen[2]

[1]*FM, Technical University of Liberec, Studentska 1402/2, 461 17 Liberec I, Czech Republic*

[2]*LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France*

Keywords: Particle Filter, MOT, GPGPU, Decentralized Tracking, Multi-template Appearance Model.

Abstract: In this work, we present the design, analysis and implementation of a decentralized particle filter (DPF) for multiple object tracking (MOT) on a graphics processing unit (GPU). We investigate two variants of the implementation, their advantages and caveats in terms of scaling with larger particle numbers and performance on several datasets. First we compare the precision of our GPU implementation with standard CPU version. Next we compare performance of the GPU variants under different scenarios. The results show the GPU variant leads to a five fold speedup on average (in best cases the speedup reaches a factor of 18) over the CPU variant while keeping similar tracking accuracy and precision.

1 INTRODUCTION

Visual tracking of targeted objects has received significant attention in the Vision community. In the last decade a number of robust tracking strategies that can track targets in complex scenes have been proposed. One such successful paradigm is the particle filter (PF) [(Isard and Blake, 1998), (Doucet et al., 2000)]. The most important property of a PF is its ability to handle complex, multi-modal (non-Gaussian) posterior distributions. Such distributions are approximated by a cloud of particles. Essentially, the number of particles required to adequately approximate the distribution grows exponentially with the dimensionality of the state space. PFs are computationally expensive as the number of particles needs to be large for good performances in terms of robustness and precision. Moreover, the observation models are often built on complex appearance models, and as a result the trackers have difficulties to operate in real time especially when the number of targets increases.
Currently, to the best of our knowledge, the literature is lacking works that showcase graphical processing unit (GPU) to improve performance of multi object tracking (MOT) algorithm. Hence, this paper aims to fill this void by presenting performance comparison and assessment of MOT algorithm running on GPU. The tracking is achieved by a decentralized particle filter with rich target appearance model. We demonstrate that the precision of GPU implementation is comparable to the standard CPU version and the execution time is significantly better – on average five times faster than CPU and on some datasets the improvement is by a factor of 18.
This paper is organized as follows: section 2 starts with background and overview of MOT and the use particle filters, the limits of CPU computation and overview of current state of GPU implementations. Section 3 presents the basics of MOT particle filter along with discussion on how to map it on GPU then Section 4 details our GPU implementations. In section 5 we present the results along with discussion and the paper finishes off in section 6 with conclusions and future works.

2 BACKGROUND AND OVERVIEW

2.1 Framework

People detection and tracking is an important research area with prominent applications in video surveillance, pedestrian protection systems, human-computer interaction, robotics, and the like. As a result, it has amassed huge interest from the scientific community [(Breitenstein et al., 2011), (Dollár et al., 2012), (Dollár et al., 2014)]. People tracking falls under Multi-Object Tracking (MOT) which

Jecmen P., Lerasle F. and Ali Mekonnen A.
Trade-off Between GPGPU based Implementations of Multi Object Tracking Particle Filter.
DOI: 10.5220/0006092301230131
In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, pages 123-131
ISBN: 978-989-758-227-1

deals with the process of accurately estimating the state of objects position, identity, and configuration - over frame from visual observations. Due to incurred challenges - heavy scene clutter, target dynamics, intra/inter-class variation, measurement noise, frame rate - it has long been established that coupling trackers with detectors, in a paradigm called tracking-by-detection, helps better tackle these challenges [(Breitenstein et al., 2011), (Li et al., 2008), (Khan et al., 2005)]. In the context of people tracking, tracking-by-detection approaches rely on a people detector to start, update, reinitialize, guide (avoid drift), or terminate a tracker.

In the literature, it is common to find many tracking-by-detection approaches applied to people tracking. However the usual trend it to select a single detector and directly couple it with the filter, e.g, [(Breitenstein et al., 2011), (Li et al., 2008)]. There are two important tracker configurations: a decentralized one which assigns an independent tracker per target [(Breitenstein et al., 2011), (Gerónimo Gomez et al., 2012)], and centralized one, also called a joint state tracker, in which all the states of the tracked targets are concatenated forming a single state vector that captures the entire system configuration (Khan et al., 2005). In our work, we have reused the observation model initied in an approach called Tracker Hierarchy (Zhang et al., 2012), which is similar to decentralized particle filter, but instead of simple single target representation, it adds a rich target representation model in the form of template ensembles.

One bottleneck is clearly the fps and we have to minimize it as much as possible. Another factor that has significant impact on tracking precision is frame rate of the video. Lower frame rate means larger time gaps between frames, which leads to bigger target's shifts. While it can be compensated by tuning the motion model accordingly, the tracking is generally more precise for videos recorded with larger frame rates. We focus on videos recorded with conventional cameras, which use frame rate of 25 frames per second.

2.2 Parallel Implementation of the Particle Filter

In typical scenarios, the computation for one particle is fast. However, the particle number typically ranges from tens to thousand (or more, depending on implementation), so the computation time necessary for one frame can get very large. As aforementioned in the literature, the time required for computation grows linearly with particle number. Due to the nature of the particle filter algorithm there was always an effort to create a parallel implementation of the algorithm to provide real time performance. (Rosn et al., 2010) studied how the use of multicore computers can help speeding up the computation. Their results show that the speedup is possible, but heavily depends on the type of particle filtering scheme used. (Medeiros et al., 2008) propose a parallel implementation of color based PF on a SIMD (Single Instruction Multiple Data) linear processor. Their analysis showed a possibility of substantial performance gain compared to current desktop computers.

With the introduction of CUDA (Compute Unified Device Architecture) the focus shifted to GPGPU. (Chao et al., 2010) offered a design guide on how to create an efficient filter running on GPU. They propose two approaches, FRIM prior editing and localized resampling, in order to reduce global operations and thus improve performance. The approach proposed by (Hendeby et al., 2010) does not use CUDA to achieve GPU computation. Instead, they propose a filter based on OpenGL shading language (Rost, 2006). Their simulations confirmed that GPU can outperform CPU for larger tasks, while for smaller tasks the overhead of GPU initialization is substantial and thus the CPU is faster for such tasks. (Rymut and Kwolek, 2010) present a particle filter with appearance-adaptive models running on GPU using CUDA. Their results again confirm the GPU speed gain, but they were able to outperform the CPU even for small particle numbers (tens or hundreds of particles). However they did not account for the GPU overhead, so the real performance of the GPU solution might be worse. (Chitchian et al., 2013) aim to provide a study on how to design a distributed particle filter on GPU. Their focus is not a people detection scenario, nonetheless their results again confirm that the GPU implementation is superior to CPU variants. Large focus has been also on the concrete parts of the particle filtering scheme, (Li et al., 2015) offer a detailed overview of resampling strategies along with discussion on how each variant performs on different platforms.

From the above insights, it is clear that a lot of detectors have GPU implementation, usually achieving significant speedup. However all of the aforementioned papers focus on SOT. As far as we know there is no paper focusing on implementing MOT on GPU. While the extension of SOT to MOT can be simple (as in case of decentralized particle filter), the performance gained in SOT case might not carry to MOT variant because of new challenges in memory handling and more complex logic of tracker creation and deletion. The theory behind decentralized PF based MOT is presented in section 3 and the GPU implementation will be discussed in section 4.

3 MOT PARTICLE FILTER BASICS

To perform MOT, we have used an approach called "Decentralized Particle Filter." In this approach, each target is assigned a unique instance of a Particle Filter as a tracker. In this work, this target specific tracker is implemented based on a tracking by detection paradigm (Perez et al., 2004) as it is the most widely used and suitable PF variant for detector integration. This is a sequential Monte Carlo approach which approximates the posterior over the target state x_t given all measurements up to time t, $Z_{1:t}$, using a set of N weighted samples, i.e., $p(x_t|Z_{1:t}) \approx \{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N$. Tracking is achieved sequentially with the notion of Importance Sampling whereby the particles at time $t-1$ are propagated according to a proposal density, $q(.)$, and their weights are updated in accordance with equation 1.

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p(z_t|x_t^{(i)})p(x_t^{(i)}|p(x_{t-1}^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)}, z_t)} \tag{1}$$

Where, $p(x_t^{(i)}|p(x_{t-1}^{(i)})$ is the target dynamic model, $p(z_t|x_t^{(i)})$ is the likelihood term, and $q(x_t^{(i)}|x_{t-1}^{(i)}, z_t)$ is the proposal density evaluated at the sampled state. To derive this filter with incoming detections, hence to perform tracking-by-detection, the proposal density shown in equation 2 is employed. According to this density, part of the particles will be sampled from the detector cues, $\pi(x_t^{(i)}|z_t)$, some from the dynamics, and some from the prior $p_0(x_t^{(i)})$, in accordance with the ratios α, β, γ which should sum to 1.

$$q(x_t^{(i)}|x_{t-1}^{(i)}, z_t) = \beta p(x_t^{(i)}|x_{t-1}^{(i)}) + \alpha\pi(x_t^{(i)}|z_t) + \gamma p_0(x_t^{(i)}) \tag{2}$$

The likelihood $p(z_t|x_t^{(i)})$ is a probabilistic measure based on the Tracker Hierarchy approach (Zhang et al., 2012). This approach consists of a rich appearance model of the target in the form of a template ensemble and uses hierarchy of expert and novice trackers for efficient multi-person tracking. One template consists of 2 histograms, each computed in different color channel. The color channels are picked at the template initialization based on the variance ratio between the background and foregrounds feature weight distributions (see (Zhang et al., 2012) for details).
During MOT, each incoming detection has to be associated with the different trackers distinctly. For that we use a greedy assignment algorithm (Breitenstein et al., 2011) which performs comparably to the Hungarian assignment algorithm.

The state of the filter is computed by minimum mean square error (MMSE) for the particle cloud. The result is a weighed average of the particle state vectors representing the current state of the filter (position and size in image plane).
In order to filter out invalid detections and noise the filter is not created at the moment of new object detection. Instead, the tracker is initialized after several subsequent detections with overlapping bounding boxes which are neither occluded nor associated to an already existing tracker. If there is no detection for already initialized tracker for several frames, we terminate it. During the frames with no detection the tracker is deactivated, so in case of new detection we can check if it isn't a detection of previously occluded target.
The MOT particle filtering introduces new kind of challenge for GPU implementation compared to single object tracking (SOT). Typically GPU is very efficient for large repetitive tasks, where a lot of data can be reused from frame to frame. In tasks, where intensive thread communication and / or frequent random data loading / storing is required, the GPU does not provide any significant performance improvement. The SOT particle filter is a task fit for GPU, because the only part where some synchronization and communication is required is the resampling step. The rest of particle handling can be trivially split to independent tasks. The introduction of multiple trackers in decentralized manner does not spoil the independence. However additional cost of computation handling arises due to more complex memory handling and the need to launch the computation for every tracker. In section 4 we will present two approaches that are fundamentally different in the way how the minimization of GPU computation overhead is handled.

4 GPU IMPLEMENTATION

First part of this section will provide an overview of programming in CUDA framework. Afterwards we will provide details on how we have mapped the MOT particle filter to GPU using CUDA framework.

4.1 Compute Unified Device Architecture (CUDA)

CUDA is a programming interface that uses the parallel architecture of NVIDIA GPUs for general purpose computing (Nickolls et al., 2008). The computation in CUDA is expressed in form of a kernels, which are programs written in Single Instruction Multiple

Threads (SIMT, which is a generalization of SIMD programming model) language defined by CUDA. The execution of the kernel is done by threads, where all threads execute the same code, but each thread has unique set of indexes to be able to determine on which data it should operate. In CUDA there are two levels of indexes - blocks (groups of threads) and grids (groups of blocks). The main difference between block and grids is that threads in one block can share a memory block, while blocks in mode grid cannot. While CUDA programming language allows programming constructs that create divergence (typically if-then-else), the physical execution is done in warps in true SIMD fashion. The device typically launches 16 or 32 threads in parallel in one warp and all threads execute the same code. This means that code with two possible ways of execution (one if-then-else block) the GPU first executes the "true" branch and then executes the "false" branch.
The memory model consists of 4 parts: registers (private memory for each thread, fastest), shared memory (all threads in one block can access the same segment), constant memory and global memory (accessible by anybody, very slow compared to other memory types). In order to hide global memory latency, GPU is able to very effectively switch between warps. So while one warp is waiting for the data, GPU switches the context to another warp, which can continue its execution. An efficient CUDA implementation must, apart from direct code conversion, also handle efficient data reading / writing due to memory latency, as little branching as possible and a good ratio of arithmetic calculations versus memory operations to achieve good GPU utilization.
While modern GPUs support double precision primitives, the performance of double arithmetic is always worse compared to single precision (Itu et al., 2011). In worst case scenario, where the performance of the algorithm is bound by the arithmetic, the performance drop might be up to 1/24 of single precision performance (the actual value depends on the concrete algorithm and GPU used). Due to this possible performance penalty we have used single precision arithmetic in our GPU implementation and we have tested the precision of the results to see if the reduced arithmetic precision has any impact on the results quality.

4.2 Our Single GPU based Tracker Variant

This variant mirrors the SOT approach, where one computation handles one tracker. This can be viewed as a naive and intuitive approach to create a GPU implementation. By launching this computation for every tracker in the frame we can perform MOT. Advantages of this approach is easier memory handling and independence from other trackers, so creation, deletion or other change in other trackers won't affect current tracker. A big disadvantage is sequential tracker computation (computation order can be seen in Figure 1).
The creation of GPU implementation of Importance Sampling introduced in section 3 consists of two parts. First part is fully parallel computation of particle weights (meaning evaluation of equation 1, when we know values of all terms). This is easily achievable because the computation of weight of one particle is independent from others. Second part consists of parallel implementation of component computation. Especially the term $p(z_t|x_t^{(i)})$ is computationally expensive and thus an effective parallel implementation would speedup the computation noticeably.
The mapping to CUDA takes advantage of indexing hierarchy, where a grid represents a complex task and the group of threads cooperate on solving given task (usually by splitting the problem space). For sampling there is no point to split the computation, so in this case the grid index is not used and the thread index identifies the particle.
Some parts of the algorithm have not been ported to GPU due to satisfactory performance on CPU, namely the detection and tracker association and filter state estimation.

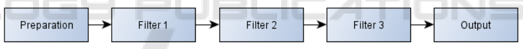

Figure 1: Sequential execution of filters.

4.3 Our Bulk Variant

One of the strategies to improve the performance of GPU computation is to group memory transfers and kernel executions together because the overhead (both CPU and GPU) of such tasks can be very high. This reflects the "bulk" mode, where we aggregate all the tracker data in current frame (particles, templates etc.) and transfer them at once (computation order can be seen in Figure 2).
In our GPU variant, we compute the particle weights with equation 1 by launching a one computation kernel for all particles in one tracker. Thus in a scene with N active trackers (targets), we launch the kernel N times with different particle data. The BULK variant groups all particles from all tracker into groups and executes the kernel just once.
The negative effect of this approach is the need for data aggregation and change monitoring. In single tracker variant, the data are tied to each tracker and its

layout does not change over time. In case of bulk variant the data are combination of data from all trackers, so in case a tracker is created or dies we have to regenerate the data on GPU to reflect this change.

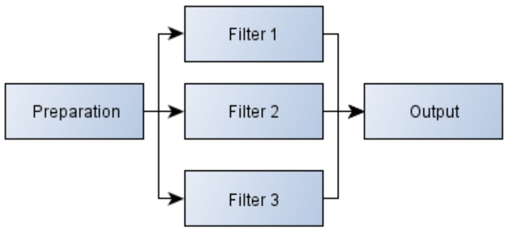

Figure 2: Bulk execution of filters.

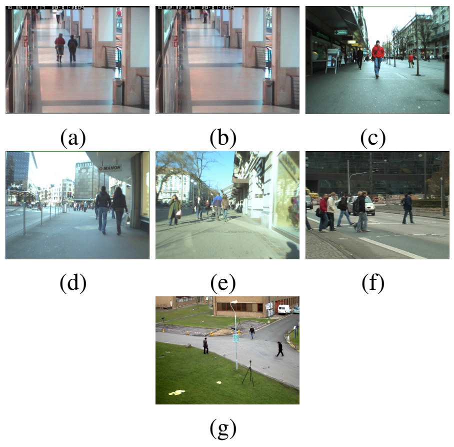

Figure 3: Images from used datasets.

5 EXPERIMENTS AND RESULTS

The objective of the paper is to assess the performance of GPU in MOT scenarios. The evaluations are split in two parts: first is to test if the precision of GPU framework is different from CPU implementation. Second part of the evaluations focus on the performance of the two proposed variants. Here below we provide an overview on used metrics, test data and obtained results.

5.1 Evaluation Metrics

To compare the quality of the results between CPU and GPU implementations, we have utilized the prevalent CLEAR-MOT metrics (Bernardin and Stiefelhagen, 2008). These metrics are principally based on computation of two quantities: the Multi-Object Tracking Accuracy (MOTA) and the Multi-Object Tracking Precision (MOTP). The $MOTA = 1 - (F_P + F_N + Id_{sw})$, where $F_P = \sum_t \frac{FP_t}{g_t}$ quantifies total false positives, $F_N = \sum_t \frac{FN_t}{g_t}$ quantifies the total false negatives, and $Id_{sw} = \sum_t \frac{Id_{sw,t}}{g_t}$ quantifies total id switches, all divided by the ground truth targets and summed over the entire dataset. The MOTP is the average bounding box overlap between the estimated target position and ground truth annotations over the correctly tracked targets. A tracker estimated rectangular position (noted R_T) is considered a correct track if its overlapping area score, $sc = \frac{R_t \cap G_t}{R_t \cup G_t}$, with the ground truth annotation (G_T) is above a given threshold τ, which is usually set to 0.5.
For the performance comparison we have measured total execution time including data preparation before the filtering and also data retrieval after the filtering. This approach penalizes the GPU version due to the need of GPU to CPU data copying for display, but it better reflects real world performance and behavior. The times were measured with millisecond precision using `std::chrono` class. The experiments were conducted on a PC with Intel Xeon E3-1241@3.5GHz, 8GB RAM, NVidia Quadro K420 with 1GB VRAM. The OS was Linux 12.04 LTS with all update available at the time, NVidia binary driver v364.19, NVidia CUDA 7.5 and as a compiler we have used nvcc v7.5.17 (bundled with CUDA package). All computations have been run ten times to account the stochastic nature of the algorithm.

5.2 Test Data

In order to properly evaluate the performance of the implementations we ran the tests on several datasets with different scene and people settings. We have used a total of 7 datasets (sample images can be seen in Figure 3): the CAVIAR[1] EnterExitCrossingPaths [3(a)] and OneShopOneWait[1] [3(b)] datasets, the ETH[2] Bahnhof [3(c)], Jelmoli [3(d)], Sunnyday [3(e)] datasets, TUD[3] Crossing [3(f)] dataset and PETS2009S2L12[4] [3(g)] dataset.

As for people detector, we have used several widely used detectors in order to take into account the dynamics of the scene.

HOG-SVM. This detector computes local histograms of the gradient orientation on a dense grid and uses linear Support Vector Machine (SVM) as a classifier. The HOG features have shown to be the most discriminant features to date, and in fact,

[1]http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/
[2]https://data.vision.ee.ethz.ch/cvl/aess/dataset/
[3]https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/software-and-datasets/
[4]http://www.cvg.reading.ac.uk/PETS2009/a.html

a majority of detectors proposed hence-after make use of HOG or its variant (Dollár et al., 2012). Further details can be found in (Dalal and Triggs, 2005).

ACF. This is a fast person detector that has shown state-of-the-art performance on various benchmark datasets (Dollár et al., 2014). It is based on aggregates (summed over blocks) of features represented as channels and a variant of Boosted classifier. Examples of features channels used include: normalized amplitude of the gradient, the histograms of oriented gradients (HOG, 6 channels) and color channels (LUV).

DPM. Contrary to HOG and ACF which detect persons full body, the DPM is a parts based detector that works by aggregating evidence of different parts of a body to detect a person in an image. Its trained model is divided in different parts. For instance, a target model could be made up of a head, upper body, and lower body sub-models. Each detected area has its own score. Thus, it is possible to tune the threshold in order to remove detections that have low scores. Since this detector relies on parts, and not solely on a full body, it detects partially occluded people rather well. Additionally, it also has better localization accuracy as it infers bounding box based on detected body parts. The detector uses variants of HOG features with Latent-SVM as classifier. Further details can be found in (Felzenszwalb et al., 2010).

5.3 Threading and Parallel Execution

In order to achieve good performance with CPU variant, we have used threads to run some parts of the computation in parallel. The first place where we have used threads, is concurrent particle filtering (resampling, sampling and weighing) and model update. We can update the model with information from current frame, while we run filtering (we call this mode of execution ”Interleaving”). Next place for parallel execution is at the level of particle filters. The filters in decentralized form of particle filter are completely independent, so we can run the computation of all filters concurrently. Another place suitable for parallel computation is inside the filter, namely the computation for each particle. However the number of particles is usually quite high, so the price of thread handling would be much bigger than the gain from parallel computation.
Table 1 sums all available execution modes. We have separated particle filtering and model update in terms of parallel computation because this allowed us to use threaded parallel computation for model update also

Table 1: Summary of available execution modes.

Phase	**Available options**		
Particle filtering	CPU	GPU	GPU-BULK
Interleaving	YES	NO	
Filter execution	SERIAL	PARALLEL	
Model update	SERIAL	PARALLEL	

for GPU computation. The main group is ”Particle filtering,” which denotes concrete implementation of the particle filter. The other groups can help speed up the computation, but we soon found out that the most parallel version (with interleaved filtering and model update, both run in parallel) is not always the fastest. This is due to possibly large count of targets in dataset, which leads to high number of running threads and thus high overhead of thread control. We always ran the test for one type of particle filtering implementation and all possible combinations of secondary groups (GPU-BULK variant does not allow serial filter execution) and always picked the best performing variant for each dataset.

5.4 CLEAR-MOT Evaluation

The results in Table 2 were obtained running all three variants on PETSS2L1 with 50 particles per tracker and with all people detectors. The results are presented as mean / standard deviation, annotated μ/σ. As can be seen, the differences between all variants are negligible. This means that the switch to single precision floating point numbers has no significant impact on result quality.

Table 2: CLEAR-MOT results on PETSS2L1 with 50 particles per tracker.

Device	**MOTP**	**MOTA**
CPU-PF	.73 / .00	.57 / .01
GPU-PF	.71 / .01	.54 / .02
GPU-BULK-PF	.72 / .00	.55 / .01

Device	T_p	F_p	F_n	ID_{sw}
CPU-PF	.67 / .01	.1 / .01	.32 / .01	35.1 / 3.8
GPU-PF	.68 / .02	.2 / .01	.35 / .02	38.4 / 3.8
GPU-BULK-PF	.66 / .00	.1 / .00	.33 / .01	36.2 / 4.3

5.5 Speedup

Figure 4 presents speedups of both GPU variants versus CPU variant for detector with 50 particles. The average speedup for GPU variant is 5 as for GPU-BULK variant the average speedup is about 4. The

variance in speedups for different datasets is due to varying people count and overall scene dynamics (e.g. number of people entering and leaving the scene, average life time of the tracker etc.). For 50 particles per tracker, the BULK variant always performs worse than GPU variant. But if we look at figure 5 representing configuration with 200 particles per tracker we can see that the performance of BULK improves. From the figure 5 we can see that the GPU variant's performance compared to BULK variant is best for EnterExitCrossingPath1cor and OneShopOneWait1cor. These recordings have the lowest dynamics in terms of people appearing and disappearing in the scene.
From the obtained results we have concluded that the BULK variant is more suitable for scenes with high number of tracker count changes. GPU is best suitable for tracking "closed" scenes, where the targets are not entering or leaving the scene too often.

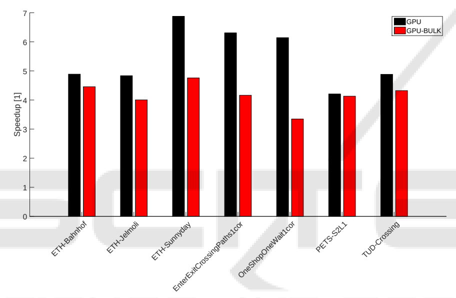

Figure 4: Speedup compared to CPU variant (50 particles per tracker, HOG tracker).

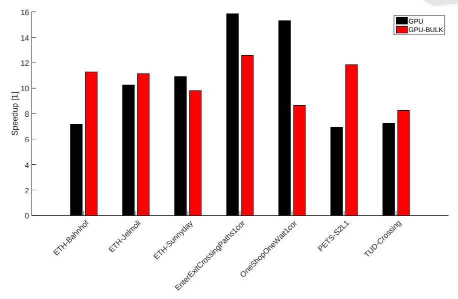

Figure 5: Speedup compared to CPU variant (200 particles per tracker, HOG tracker).

Figure 6 further emphasizes the influence of target dynamics on performance. The figure shows that BULK variant is more sensitive to dynamics, the speedup difference for DPM and LDCF INRIA object detectors is almost 0.5. The effect on GPU variant is much smaller (maximal difference is 0.2), but is still noticeable. This means that when considering which implementation to choose, it is not enough to base the choice on simple scene dynamics evaluation, but we also have to consider the properties of the object detector.

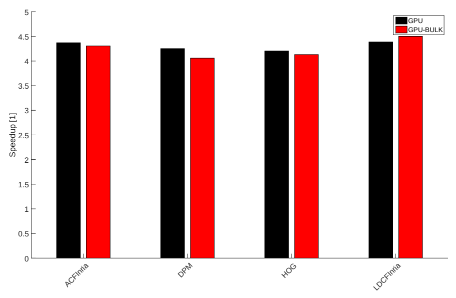

Figure 6: Speedup compared to CPU variant with different object detectors (50 particles per tracker, PETS2009S2L12 dataset).

5.6 Execution Time vs Particle Number

The performance scaling comparison with different number of particles per tracker can be seen in Figures 7 (PETS2009S2L12 dataset), 8 (TUD Crossing dataset) and 9 (OneShopOneWait dataset), all three figures used HOG people detector.
All three tested variants exhibit almost linear dependency on particle count in all three figures. CPU variant is always the slowest one, even for 20 particles per tracker. For OneShopOneWait dataset the performance for 20 particles per tracker is very close to GPU and BULK variant, but this is because of low tracker count. Due to low tracker count the GPU and BULK variants have almost constant execution time for all particle counts, because the graphic card is not fully utilized. The GPU-BULK variant exhibits best performance in terms of scaling, but due to the fixed cost of data preparation needed it is better suited for scenarios with larger data size. In cases where we have used low particle count per tracker or the scenes contain only small number of tracker the GPU variant is superior.

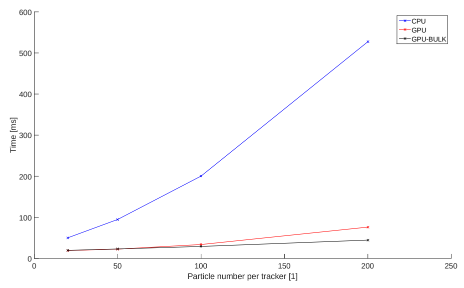

Figure 7: Execution time vs particle number (PETS2009S2L12 dataset, HOG detector).

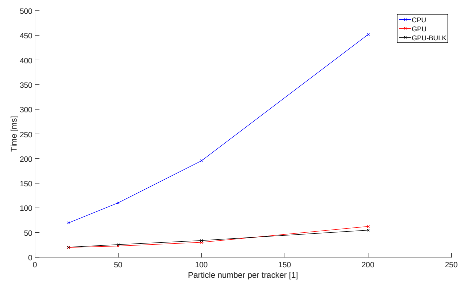

Figure 8: Execution time vs particle number (TUD Crossing dataset, HOG detector).

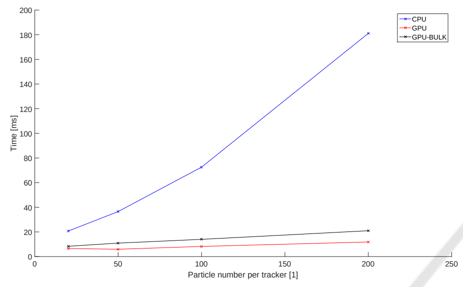

Figure 9: Execution time vs particle number (OneShopOneWait dataset, HOG detector).

5.7 Speedup Stability

Figure 10 provides a statistics of achieved speedups. Each bar represents count of data configurations (dataset - detector - particle number) that achieved given speedup (speedups were rounded to nearest integer).Notice that, for instance, 26 variants have achieved speedup of 4. We can also notice that GPU variant is generally faster, but the BULK variant provides more similar results. For general use the average speedup is more important, but Figure 10 highlights that both variants can achieve much higher speedups in some cases. This is due to CUDA work sizes used to launch the kernels. They have been optimized on one data configuration and then used for all tests. By optimizing the launch parameters for each data configurations separately the speedup would be much better with maximal values at the right side of the statistics (around 12x for BULK variant and 15 for GPU variant).

When we compile all the presented results, we can safely conclude that it is beneficial to use GPU to speedup the MOT algorithm. The choice of concrete implementation depends on two factors - required particle number and characteristics of captured video, mainly the dynamics of people entering / leaving the scene. The GPU variant offers good performance for more static scenes and for lower particle counts due to

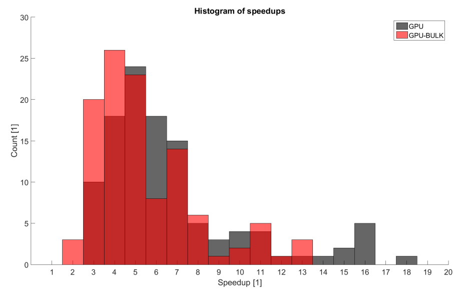

Figure 10: Statistics of speedups for GPU variants.

easier memory handling. The BULK variant handles better more dynamic scenes due to the fact that the cost of memory handling is almost independent from the number of filters. Another benefit of BULK variant is better performance for higher particle numbers due to batch processing. Such insights have not been yet highlighted in the literature.

6 CONCLUSIONS AND FUTURE WORK

In this work, we have presented two GPU implementations of decentralized particle tracker with rich appearance model based on templates. The results show overall performance boost by factor of at least 5, while for best cases the boost is around 18. These results have been achieved for relatively low particle numbers (50 particles per tracker, average number of tracker in one frame was around 5) compared to previous works. We have also showed that the use single precision arithmetic due to GPU limitations does not affect the precision of the results.

As a future work, an investigation on dynamic optimal group size computation could provide another significant boost to the performance due to always optimal use of GPU resources. Another interesting line of work could be full GPU implementation of tracking (both detection and filtering) by coupling existing GPU implementation of object detectors (for example (Sudowe and Leibe, 2011) or (Hirabayashi et al., 2013)) with our solution to further improve performances. The full GPU implementation would be especially suited for tracker with low particle numbers, where the speedup of the filtering with GPU is not high.

ACKNOWLEDGMENTS

The work of P. Jecmen was supported by the Ministry of Education of the Czech Republic within the SGS project no. 21176/115 of the Technical University of Liberec.
The work of F. Lerasle and A. A. Mekonnen was supported by grants from the French DGA under grant reference SERVAT RAPID-142906073.

REFERENCES

Bernardin, K. and Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*.

Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Gool, L. V. (2011). Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(9):1820–1833.

Chao, M. A., Chu, C. Y., Chao, C. H., and Wu, A. Y. (2010). Efficient parallelized particle filter design on cuda. In *2010 IEEE Workshop On Signal Processing Systems*, pages 299–304.

Chitchian, M., van Amesfoort, A. S., Simonetto, A., Keviczky, T., and Sips, H. J. (2013). Adapting particle filter algorithms to many-core architectures. In s.n., editor, *Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 427–438. IEEE Computer Society.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

Dollár, P., Appel, R., Belongie, S., and Perona, P. (2014). Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1532–1545.

Dollár, P., Wojek, C., Schiele, B., and Perona, P. (2012). Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(4):743–761.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645.

Gerónimo Gomez, D., Lerasle, F., and López Peña, A. M. (2012). *State-Driven Particle Filter for Multi-person Tracking*, pages 467–478. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hendeby, G., Karlsson, R., and Gustafsson, F. (2010). Particle filtering: The need for speed. *EURASIP J. Adv. Signal Process*, 2010:22:1–22:9.

Hirabayashi, M., Kato, S., Edahiro, M., Takeda, K., Kawano, T., and Mita, S. (2013). Gpu implementations of object detection using hog features and deformable models. In *2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 106–111. IEEE.

Isard, M. and Blake, A. (1998). Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28.

Itu, L. M., Suciu, C., Moldoveanu, F., and Postelnicu, A. (2011). Comparison of single and double floating point precision performance for tesla architecture gpus. *Bulletin of the Transilvania University of Brov Series I: Engineering Sciences*, 4:70–86.

Khan, Z., Balch, T., and Dellaert, F. (2005). Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1805–1819.

Li, T., Bolic, M., and Djuric, P. M. (2015). Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 32(3):70–86.

Li, Y., Ai, H., Yamashita, T., Lao, S., and Kawade, M. (2008). Tracking in low frame rate video: A cascade particle filter with discriminative observers of different life spans. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1728–1740.

Medeiros, H., Park, J., and Kak, A. (2008). A parallel color-based particle filter for object tracking. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–8.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6(2):40–53.

Perez, P., Vermaak, J., and Blake, A. (2004). Data fusion for visual tracking with particles. *Proceedings of the IEEE*, 92(3):495–513.

Rost, R. J. (2006). *OpenGL Shading Language*. Addison Wesley Professional.

Rosn, O., Medvedev, A., and Ekman, M. (2010). Speedup and tracking accuracy evaluation of parallel particle filter algorithms implemented on a multicore architecture. In *2010 IEEE International Conference on Control Applications*, pages 440–445.

Rymut, B. and Kwolek, B. (2010). *GPU-Accelerated Object Tracking Using Particle Filtering and Appearance-Adaptive Models*, pages 337–344. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sudowe, P. and Leibe, B. (2011). Efficient Use of Geometric Constraints for Sliding-Window Object Detection in Video. In *International Conference on Computer Vision Systems (ICVS'11)*.

Zhang, J., Presti, L. L., and Sclaroff, S. (2012). Online multi-person tracking by tracker hierarchy. In *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*, pages 379–385. IEEE.