WLCSSCuda: A CUDA Accelerated Template Matching Method for Gesture Recognition

Mathias Ciliberto
m.ciliberto@sussex.ac.uk
Wearable Technologies Lab,
University of Sussex
Brighton, UK

Daniel Roggen
d.roggen@ieee.org
Wearable Technologies Lab,
University of Sussex
Brighton, UK

ABSTRACT

Template matching methods can benefit from multi-cores architecture in order to parallelise and accelerate the matching of multiple templates. We present WLCSSCuda: a GPU accelerated implementation of the Warping Longest Common Subsequence (WLCSS) pattern recognition algorithm. We evaluate our method on 4 NVIDIA GPUs and 4 multi-cores CPUs. We observe a 67-times speedup for the GPU implementation in the best case against the multithreaded CPU implementation.

CCS CONCEPTS

• **Computing methodologies** → *Parallel algorithms.*

KEYWORDS

WLCSS; CUDA; GPU acceleration; Template Matching

ACM Reference Format:
Mathias Ciliberto and Daniel Roggen. 2019. WLCSSCuda: A CUDA Accelerated Template Matching Method for Gesture Recognition. In *Proceedings of the 2019 International Symposium on Wearable Computers (ISWC '19), September 9–13, 2019, London, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3341163.3347718

1 INTRODUCTION

Template Matching Methods (TMM) have been successfully applied in activity recognition [2]. They compute a matching score between one or more templates and a stream of sensor data. Therefore, they can benefit from multi-core architectures in order to improve their speed of execution. They can take advantage of such architectures when multiple templates must be simultaneously compared to an incoming sensor signal as well as during training when multiple instances of the TMM must be run in order to evaluate different parameter sets. WLCSSCuda is a GPU accelerated implementation of Warping Longest Common Subsequence (WLCSS) using CUDA. This is a framework developed by NVIDIA for general purpose processing on GPUs (GPGPU) [3]. In the past, GPUs have been used to accelerate other TMMs, such as Dynamic Time Warping [5] and Longest Common Subsequence [6]. However, to the best of our knowledge, this is the first GPU implementation of WLCSS. WLCSSCuda allows to execute multiple instances of the TMM simultaneously. We compare this approach with a multi-core CPU implementation of WLCSS and show an drastic speedup in the matching score computation.

ISWC '19, September 9–13, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6870-4/19/09.
https://doi.org/10.1145/3341163.3347718

2 WARPING LONGEST COMMON SUBSEQUENCE

Warping Longest Common Subsequence (WLCSS) is a TMM suitable for continuous pattern recognition - such as spotting complex gestures - which is more robust than Dynamic Time Warping to noisy sensor data [2]. It computes a matching score ($M_{i,j}$) between a template (T) and a stream (S) by adding a reward (R) every time a sample i from S matches with a sample j from T within an acceptance threshold (ϵ). Otherwise, it decrements $M_{i,j}$ by a penalty (P) proportional to the mismatch.

Optimising WLCSS parameters requires an exhaustive grid search for R, P, ϵ, which benefits from computational speedups.

3 WLCSSCUDA

WLCSSCuda is our GPU implementation of WLCSS built using CUDA. Thanks to the high number of cores in modern GPUs, it is possible execute tasks, called kernels, with a high level of parallelization. CUDA abstracts the physical structure of the GPU in a grid of blocks. Each block can be addressed using a 1D, 2D, or 3D index. A block is a unit made by several threads that can be computed in parallel or serially on a GPU core. A 3D indexing is provided for the thread too. The scheduling of the threads' execution is transparent to the user. The number of maximum blocks and maximum threads per block depends on the GPU.

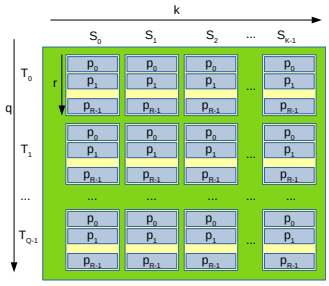

Figure 1: WLCSSCuda structure: blocks are represented in yellow. A template T_q and a stream S_k are assigned to each block. A set of parameters p_r is assigned to each thread, displayed in blue.

WLCSSCuda structures the computation using a 2D grid for the blocks and 1D structure for the threads, as shown in Figure 1: a single template/stream pair is assigned to each block. Then, within that block, a thread is used for every parameters set. During initialization, all the templates, streams and parameters are transferred to the GPU memory. Then, each kernel computes the pointer to the templates, stream and WLCSS parameters in memory using respectively the indexes q and k for the blocks, and r for the threads. Each triad template/stream/parameters is used by only one kernel which is executed by a single thread. Finally, when the matching scores are computed, they are transferred from the GPU memory back to the main memory. WLCSSCuda computes the entire score between the template and the stream.

WLCSSCuda is developed in C++ with a Python wrapper for loading the data and reading the results.

4 EVALUATION

We evaluated to which extent WLCSSCuda could accelerate multiple template matching, taking into account the time required to transfer the data to/from the GPU, which has been demonstrated to be a bottleneck in CUDA applications [1]. We compared the execution of WLCSSCuda on 4 GPUs and a multithreaded WLCSS on 4 CPUs (see Table 1). We simulates 4 test scenarios in which a different number of streams, templates and WLCSS parameters were employed (Table 2). We reported the average of 10 executions. The CPU implementation of WLCSS uses all the available threads in the CPU to run always the maximum number of possible WLCSS simultaneously.

We used OPPORTUNITY Dataset [4] as source of templates and streams, selecting a random subset of gestures for each test, to make them more realistic. The average length of the templates (and streams) is 98 samples, with a standard deviation of ± 46. The same subset of gestures was used for WLCSSCuda and the CPU implementation in each test.

GPU Model	CUDA cores	Cores Frequency	Memory
GTX 1080 Ti	3584	1645 MHz	11 GB GDDR5X
GTX 1050 Ti	768	1418 MHz	4 GB GDDR5
GTX 970	1664	1317 MHz	4 GB GDDR5
Titan XP	3840	1582 MHz	12 GB GDDR5X

CPU Model	Frequency (Max Turbo)	# of Cores	# of Threads
AMD Ryzen 1900X	3.8 GHz (4 GHz)	8	16
Intel i7-8750H	2.2 GHz (4.1 GHz)	6	12
Intel i7-4770K	3.5 GHz (3.9 GHz)	4	8
Intel i7-6700	3.4 GHz (4.0 GHz)	4	8

Table 1: CPU and GPU tested. For more details about the GPUs and CPUs (AMD and Intel), visit respectively [1], [2], [3]

Test	# Templates (Q)	# Streams (K)	# Params. sets (R)	Tot. WLCSS
a	10	1000	10	100000
b	20	2000	10	400000
c	50	5000	10	2500000
d	100	10000	10	10000000

Table 2: The number of templates, streams, parameters sets and the total number of WLCSS computation is shown, for each test scenario.

Platform / Test	*a*	*b*	*c*	*d*
Titan XP	0.49 ± 0.04	1.53 ± 0.15	9.66 ± 0.61	38.29 ± 1.87
GTX 1080 Ti	0.55 ± 0.07	1.98 ± 0.14	12.77 ± 1.51	51.38 ± 6.22
GTX 1050 Ti	0.79 ± 0.13	2.92 ± 0.26	18.15 ± 0.67	72.04 ± 3.59
GTX 970	0.91 ± 0.16	4.00 ± 0.67	18.97 ± 0.86	70.74 ± 3.69
AMD 1900X	8.29 ± 0.89	32.27 ± 1.74	231.75 ± 31.59	932.69 ± 86.19
i7-8750H	17.50 ± 2.57	81.23 ± 5.30	555.26 ± 15.39	2140.20 ± 141.37
i7-6700	20.35 ± 1.07	80.18 ± 8.28	523.05 ± 34.30	2008.19 ± 81.33
i7-4770K	22.78 ± 3.20	99.62 ± 10.00	647.21 ± 45.91	2452.15 ± 170.12
Improvement	10-46 times	8-65 times	12-67 times	13-64 times

Table 3: Results of WLCSS running on the 4 GPUs and the 4 CPUs. The values are in seconds and they are averaged across multiple running. The improvements are computed respectively between the best CPU against the worst GPU, and viceversa.

Table 3 shows the average time for each test for every CPU and GPU. For WLCSSCuda, all the values include the time to the transfer of data to/from the GPU memory. As we expected, the GPUs are faster in every scenario we evaluated. Moreover, it is possible to notice how WLCSSCuda scales better when the number of instances increase. Test *d* requires 100 times more evaluations than test *a*; the GPUs take on average only 85 times the time required by test *a* while the CPUs take 110 times more than *a*, on average, across all the different models.

[1] https://www.nvidia.com/en-gb/
[2] https://www.amd.com/en/products/ryzen-threadripper
[3] https://ark.intel.com

5 DISCUSSION AND CONCLUSION

We presented WLCSSCuda, a GPU accelerate multiple TMM. We demonstrated that WLCSSCuda can drastically increase the computation of multiple template matching, with an increase of 67 times in the best case compared to a multi-threaded CPU approach. However, there is still room for improvement: we plan to evaluate different organizations of data in order to better use the block/thread CUDA structure. Moreover, we aim to make WLCSSCuda automatically adapting such structure according with the number of templates/streams/parameters sets in order to increase the performance even further. Finally, WLCSSCuda is available as open source software at the address https://github.com/sussexwearlab/WLCSSCuda.

ACKNOWLEDGMENTS

This study was partly funded through the FFG project #5766494 "MinIAttention: Attention Management in Minimal Invasive Surgery". We also thank NVIDIA for their Titan XP donation.

REFERENCES

[1] Chris Gregg and Kim Hazelwood. 2011. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *International Symposium on Performance Analysis of Systems and Software*. IEEE.

[2] Long Van Nguyen-Dinh et al. 2012. Improving online gesture recognition with template matching methods in accelerometer data. *International Conference on Intelligent Systems Design and Applications* (2012).

[3] Nvidia. 2019. *CUDA Toolkit.* https://developer.nvidia.com/cuda-toolkit Retrieved on April 26, 2019 from https://developer.nvidia.com/cuda-toolkit.

[4] Daniel Roggen et al. 2010. Collecting complex activity datasets in highly rich networked sensor environments. In *International Conference on Networked Sensing Systems*. IEEE.

[5] Doruk Sart et al. 2010. Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In *International Conference on Data Mining*. IEEE.

[6] Jiaoyun Yang et al. 2010. An efficient parallel algorithm for longest common subsequence problem on GPUs. In *Proceedings of the World Congress on Engineering*, Vol. 1.