OpenCL-based optimizations for acceleration of object tracking on FPGAs and GPUs

Amir Momeni Hamed Tabkhi Gunar Schirner David Kaeli
ECE Department of Northeastern University
momeni/tabkhi/schirner/kaeli@ece.neu.edu

Abstract

OpenCL support across many heterogeneous nodes (FPGAs, GPUs, CPUs) has increased the programmability of these systems significantly. At the same time, it opens up new challenges and design choices for system designers and application programmers. While OpenCL offers a universal semantic to capture the parallel behavior of applications independent of the target architecture, some customization should take place at the source-level to increase the efficiency of the target platform.

In this paper, we study the impact of source-level optimizations on the overall execution time of OpenCL programs on heterogeneous systems. We focus on Meanshift Object Tracking (MSOT) algorithm as a highly challenging compute-intense vision kernel. we propose a new vertical classification for selecting the grain of parallelism for MSOT algorithm across two mainstream architecture classes (GPUs and FPGAs). Our results show that both fine-grained and coarse-grained parallelism can greatly benefit GPU execution (up to a 6X speed-up), while the FPGA can only benefit from fine-grained parallelism (up to a 4X speed-up). However, the FPGA can largely benefit from executing both the parallel and serial parts of the program on the device (up to a 21X speed-up).

Keywords OpenCL, Object Tracking, FPGA, GPU

1. Introduction

Massively parallel programming languages such as OpenCL and CUDA have streamlined the programming of heterogeneous systems. These languages provide a universal semantic to capture the behavior of applications in a parallel fashion across multiple heterogeneous nodes. In particular, there have been recent developments of the Open Computing Language (OpenCL) for platforms combining GPUs and CPUs [6].

The availability of a rich programming platform has motivated both platform architects and software developers toward unification. Field-Programmable Gate Arrays (FPGAs) are one of the main architecture candidates that can significantly benefit from OpenCL unification. OpenCL can potentially be a significant step toward use of FPGAs in heterogeneous platforms. The top two FPGA companies (i.e., Altera and Xilinx) have already explored this path by releasing dedicated design suites to map OpenCL codes to the FPGAs, simplifying the task of FPGA integration.

OpenCL support across many heterogeneous nodes (FPGAs, GPUs, CPUs) opens up significant design choices, as well creates new design challenges for system designers and application programmers. In principle, OpenCL offers a universal description of an application, independent from the target architecture. While this is true, in reality, some customization should take place at the source code level that considers the actual target platform.

This challenge is more pronounced when we consider platforms that include GPUs and FPGAs. While GPUs efficiency relies on exploiting concurrent execution of many cores with massively parallel threads, FPGAs efficiency stems from customization of the data-path and the ability to exploit deep pipelines. As a result, while OpenCL abstracts away many implementation details from the programmer, the programmer is still responsible for understanding the impact of source-level decisions on the overall execution time when developing OpenCL program across different heterogeneous nodes. In many cases, programmers have to revisit the source-level design decisions that include choices impacting the granularity of parallelism and frequency of host-to device communication, to achieve fully exploit the speed-up offered on the target accelerator architecture (GPU or FPGA).

Optimizations implemented at the OpenCL programming layer have not been studied in depth. Most optimization studies have focused solely on GPU tuning, since these devices have dominated the heterogeneous computing market to date. Now that FPGAs have become potential targets, we need to consider how source-level choices impact application performance when targeting different accelerators. The general trend has been to compare a hand-crafted FPGA implementation with an OpenCL-programmed GPU execution to evaluate the performance and power efficiency [4, 5]. Furthermore, previous work mainly focuses on embarrassingly parallel applications, ignoring the broader class of irregular applications that possess lower degrees of parallelism, though plenty of potential for acceleration with the right device. We argue that new research needs to explore the effect of source-level design decisions across a wide range of architectures. The results of this study can help guide the OpenCL developer to better leverage the targeted accelerator. At the same time, there is a need to tackle OpenCL implementations of complex algorithms that possess irregular execution patterns and less obvious parallelism. A good representative class of applications are advanced vision algorithms, which are compute-intense kernels mixed levels of parallelism and regularity across parallel threads.

Overall, the aim of our work is to demonstrate the potential for new research path that consider the effect of programming decisions and architecture properties across heterogeneous platforms bridging the programming abstraction and underlying architecture. For the example of MSOT, this paper demonstrates that FPGAs sig-

nificantly benefit from fine-level parallelism (up to a 4X speed-up), while GPUs need a combination of coarse- and fine-grained parallelism (up to a 6X speed-up). Furthermore, a homogenous FPGA-only solution leads to skyrocketing speedup (up to a 21X speed-up) due to constructing a customized data-path for both parallel and serial portions of algorithm.

The remainder of this paper is organized as follows. Section 2 briefly overviews the Mean-shift algorithm as the case study for this paper. Section 3 categorizes different classes of parallelism. Section 4 explores parallelism across GPU and FPGA platforms. Section 5 concludes the paper.

2. Background and motivation

Next, we review the basic principles of the Mean-Shift Object Tracking (MSOT) algorithm and discuss inefficiencies in the single-threaded version of this code. The MSOT algorithm was originally proposed by Comaniciu et al. [3] and later became widely used for object tracking due to its high quality and robustness. Algorithm 1 highlights the major steps of the Mean-shift algorithm. Decomposing it at a high level, the algorithm is divided into two steps: 1) initialization and 2) adaptive tracking. During the initial step (frame 0 only), a color histogram is calculated per object in the scene (line 3 and per line 4). The histogram values are used as the reference model for tracking objects through the remaining sequence of frames. In the adaptive tracking phase (frames 1 to N), MSOT iteratively identifies the new location of the object with respect to the reference histograms. MSOT calculates the current histogram (line 10) and uses the Bhattacharyya distance measure to determine the similarity between the current and reference histograms (line 11). Next, the shift-vectors are calculated based on the Bhattacharyya distance, and the object moves one step toward its new location (line 12 and line 13 of the algorithm). Overall, the higher the iteration threshold (i.e., higher $Threshold$), the higher the similarity matching and thus the higher the quality. A thorough description of the original MSOT algorithm has been described previously [3].

Algorithm 1 MSOT

```
Require: A sequence of frames, F = {f_0, ...f_N}
 1: procedure OBJECT TRACK
 2:     for all objects in the scene do                    ▷ Initialization
 3:         b ← calculate the bins of f_0
 4:         q ← calculate the target color histogram on f_0
 5:     end for
 6:     for i = 1 to N do                                  ▷ Adaptive tracking
 7:         for all objects in the scene do
 8:             b ← calculate the bins of f_i
 9:             while k ¡ Threshold do
10:                 p ← target color histogram on f_i
11:                 d ← distance between p and q
12:                 (X,Y) ← calculate shift vector
13:                 Update the target position
14:             end while
15:             Draw the target window on the frame f_i
16:         end for
17:     end for
18: end procedure
```

To evaluate the performance of the serial MSOT on heterogeneous platforms, we start by developing a serial (single-thread) version of the code in ANSI C running on an Intel Core i7-3820 CPU. The input is a sequence of 120 frames of a soccer field, with 10 objects (players) being tracked. We also consider the maximum quality by selecting an iteration threshold of 60. Table 1 highlights the execution efficiency of serial executions in term of Frames per Second (FPS) over increasing the number of objects in the scene from 1 to 10 objects. The quality of FPS significantly degrades as the number of tracked objects increases (30 FPS for 1 object and only 2.5 FPS for 10 objects). This makes the single-thread (CPU-based) execution very inefficient. One possible solution is to reduce the value of $Threshold$, which results in significant quality loss. Vision markets are always demandomg higher quality and improved performance together, As a result, the trend is towards accelerator architectures, such as FPGAs and GPUs, that can extract parallelism present in algorithms.

Table 1. Serial execution of MSOT

# of Targets	1	2	4	6	8	10
Performance (FPS)	29	13	6.7	4.1	3.1	2.5

Some existing approaches have aimed at accelerating MSOT using GPUs [7, 9] and FPGAs [2, 8]. What these approaches have in common is a lack of insight into the possible performance opportunities available through tuning the implementation at a source level and by focusing optimizations at a very fine resolution. Furthermore, they mainly focused on first-order implementation possibilities, achieving very limited speed-up (e.g. [7] reported a 3.3X speed-up). In contrast, we look for opportunities through a deeper analysis of the algorithm and by finding the right grain of parallelism. We study the effect of source-level parallelism choices on both FPGAs and GPUs acceleration.

3. Parallelism Granularity

Section 2 showed that a serial (CPU-based) execution of MSOT algorithm is slow when tracking multiple objects at high resolution. Developing a parallel implementation of MSOT is not straightforward. Compared to many embarrassingly-parallel vision filters (e.g., Canny edge detection, convolution filtering), MSOT would appear to have much less inherent parallelism when working at a coarse granularity. In particular, the main factor hindering the parallelism potential is the inherently serialized execution model of MSOT. The algorithm computes a histogram of current position, calculating distance and gradually converting to a new position. This serialization factor makes MSOT parallel description very challenging.

Through exploring a spectrum of parallel implementations, we have tried to develop some insight about the best path to MSOT acceleration. We have identified how to leverage parallelism at multiple levels of granularity. Fig. 1 highlights the different levels, ranging from coarse to fine, and including object-level, neighbor-level, window-level and instruction-level parallelism. Next, we will describe each level further, starting at coarse-grained parallelism and ending at fine-grained strategies.

Object-Level Parallelism (OLP): The first and coarsest level of parallelism present in the Mean-shift algorithm is object-level parallelism. The procedure for tracking the individual objects in the scene are completely independent. This allows several objects in a sequence of images to be tracked concurrently.

***Neighbor-Level Parallelism (NLP)*:** OLP offers very limited parallelism – the number of OpenCL threads are bounded by the number of objects in the scene, leading to significant underutilization on the target architecture. One possible way to increase parallelism is to speculatively compute feature positions across the neighbors of the current position being computed.

The basic idea is to calculate the histograms and shift vectors not only for the current position of the objects, but also across a number of neighbors (which we will refer to as the search distance) in parallel. By speculating on these values, we are able to utilize a much higher number of parallel OpenCL threads (work-items).

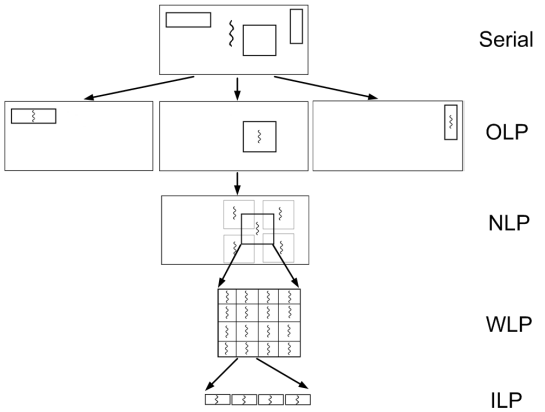

Figure 1. Parallelism granularity in Mean-shift

Speculatively computing feature positions can potentially lead to higher speed-up. However, in general this depends on the parallelism potentials of the target architecture. On the down side, speculative execution in NPL introduces a serialization factor at later in the execution. At the end of the neighbor histogram calculation, a serial thread needs to run to identify one neighbor position which matches the current position with the value that has been estimated by the shift vector.

***Window-Level Parallelism (WLP)*:** The histogram calculation involves all of the pixels of each object. One possible way to utilize a larger number of threads is to split pixels covering an object into smaller windows or segments (object segmentation), and then calculate the histogram across the segments in parallel. Window-level parallelism exposes far more parallelism and can potentially lead to higher speed-ups. On the down side, similar to NLP, there will be significant serialization delay when we need to gather the individual results and calculate the final histogram across all parallel executed threads.

***Instruction-Level Parallelism (ILP)*:** Even inside a window, there exists significant fine-grained instruction level parallelism between the pixels. It is almost infeasible to expose ILP to OpenCL source code. ILP extraction will solely depend on the capability of the underlying architecture, as well as the target compiler.

4. Parallelism Exploration

Next, we explore and evaluate the effect of source-level decisions on the execution efficiency of GPUs and FPGAs. The exploration focus is based on the parallelism approaches identified in Section 3.

4.1 Execution Setup

All accelerator codes consider in this exploration are based on the OpenCL 1.0 standard (the version supported by the Altera tools). We have targeted two state-of-the-art accelerator architectures: the NVIDIA Tesla K20 GPU and the Altera Stratix-V FPGA. We have also utilized the NVIDIA CUDA Compiler (NVCC) and the Altera SDK for OpenCL v14.0 [1] for compiling the OpenCL codes on the GPU and FPGA platforms, respectively. To evaluate parallelism approaches on the GPU and the FPGA, we have used two different systems (listed in Table 2 - we are not trying to compare these systems against one another. For both architectures, we consider the Object-Level Parallelism (OLP) as the baseline. The experiments are carried out on a sequence of 120 full HD (1080×1920) frames of a soccer field, tracking 10 objects simultaneously. Based on our quality explorations, the bin size and the threshold are set to 4096 and 50, respectively, to achieve high quality.

Table 2. System characteristics

	System I	System II
Host	Core i7-3820	Xeon CPU E5410
Host clock	3.60 GHz	2.33 GHz
Device	Tesla k-20 GPU	Stratix-V FPGA
Device resource	2496 processor cores	622000 LEs
Device clock	706 MHz	100 MHz
Device memory size	5 GB	∼8 GB

4.2 Heterogeneous Approaches

Utilizing the various levels of parallelism can generate serial computation overhead for the MSOT algorithm (i.e., there is no free lunch). Therefore, the OpenCL implementation of the MSOT is composed of both parallel and serial execution sections. In our heterogeneous implementation, the serial portion of the algorithm is executed on the CPU, while the parallel portion is executed on either the GPU or the FPGA.

NLP Evaluation: NLP is the coarsest level of parallelism that can be applied to our baseline implementation. Table 3 shows the impact of using NLP for the MSOT. When the search distance is 1, 9 neighbors are calculated in parallel for each object. Since the MSOT tracks 10 objects in our experiments, the total number of threads is 90. Similarly, the total number of threads is 250, 490, and 810 when the search distance is 2, 3, and 4, respectively. Increasing the search distance on the GPU platform increases the parallelism in the algorithm, and therefore, the resource utilization increases dramatically. However, the serialization factor added by NLP also increases. On the GPU, the best performance is achieved when the search distance is 2 (1.9X speed-up). Increasing the search distance to more than 2 overloads the GPU resources and the speed-up drops.

The Altera OpenCL SDK uses the concept of *pipelined parallelism* to map the OpenCL kernel code to the FPGA. It builds a deeply pipelined compute unit for the kernel. The compiler replicates the compute unit based on the available resources on the FPGA to expose parallelism. To gain the benefit of NLP optimization (which is a coarse-grained parallelism approach), the platform needs to have multiple compute units. However, since the MSOT kernel design is very large, we exhaust resources on the FPGA for the Altera OpenCL SDK to replicate the compute unit. Therefore, the FPGA base system, unlike the GPU base system, did not gain any benefit from the NLP approach.

Table 3. NLP speed-up on GPU and FPGA

Search Distance	# of neighbors	GPU	FPGA
1	9	1.58X	1.005X
2	25	1.91X	1.007X
3	49	1.77X	1.006X
4	81	1.63X	1.006X

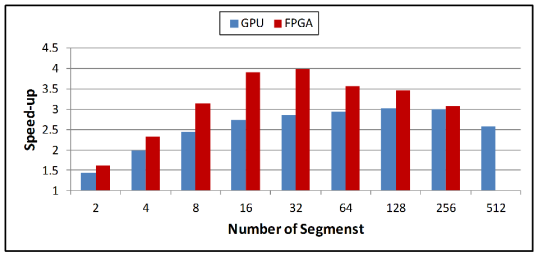

Figure 2. WLP speed-up on GPU and FPGA

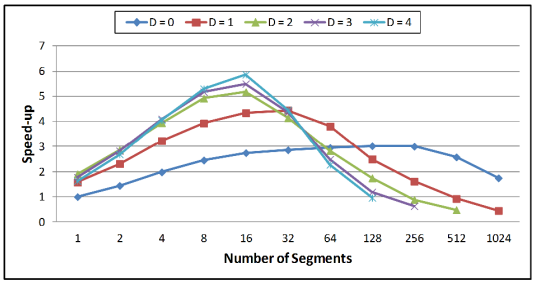

Figure 3. Hybrid approach speed-up on GPU

WLP Evaluation: The WLP optimization can provide benefits to both platforms. Increasing the number of segments, increases the parallelism and the serialization overhead at the same time. At some point, the overhead of the reduction process will dominate any performance improvements provided by Window-Level Parallelism. Based on Fig. 2, the best distribution choice on System I is 128 (3X speed-up). The total number of threads is 2560 (256×10) in this case. This number is 32 (320 threads) for System II. the WLP optimization achieves a 4X speed-up on the FPGA platform in this case.

Similar to NLP, the Altera OpenCL SDK builds only one compute unit for the MSOT kernel. However, the compute unit that was generated is capable of running many fine-grained threads in parallel. Based on Table 3 and Fig. 2, the FPGA platform is better suited to exploit fine-grained parallelism versus working at a coarser grain.

Hybrid Evaluation: Since both NLP and WLP showed performance improvements on the GPU platform, we want to explore a Hybrid implementation where we combine NLP and WLP to take advantage of both approaches. We varied the Search Distance (D) from 0 to 4, and the segmentation level from 1 to 1024. The combination of coarse-grained and fine-grained parallelism approaches is the best way to use the GPU resources efficiently. However, the best source-level decisions are different from using NLP or WLP individually. In the Hybrid approach, the best performance (6X speed-up) is achieved by choosing the search distance of 4 and 16 number of segments per object (Fig. 3). The total number of threads in this case is 12960 ($81 \times 16 \times 10$).

4.3 Homogeneous Approaches

A GPU is a massively parallel device that can outperform a CPU when executing regular, data-parallel, applications. However, the weakness of the GPU is executing serial computations. On the other hand, a FPGA can handle both parallel and serial computations. This potential can lead programmers to develop better designs for the FPGA and achieve significant performance improvements. In this section, we evaluate the FPGA performance when running both parallel and serial portions of the MSOT algorithm. The OpenCL kernel is designed in such a way, that lots of threads are executed in parallel to perform the parallel section of the algorithm. Then, the first thread executes the remaining serial portion. The key benefit of using this approach is that we can reduce the overhead of transferring data between the host and the device. However, the drawback is that the kernel executable is large and uses more FPGA resources in comparison to the size of only the parallel kernel. Fig. 4 shows that there is a huge benefit in using this homogeneous approach on the FPGA. We can achieve up to a 21X speed-up, which is much higher than previous approaches. The best performance improvement is seen when the number of segments per object is 32. The same pattern was seen before, using the WLP approach on the FPGA.

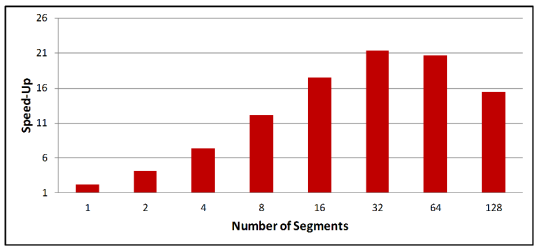

Figure 4. Homogeneous approach on FPGA

5. Conclusions

In this paper we presented a study that considered how to exploit different classes of parallelism on a GPU and FPGA platform. We report on the performance of the Mean-shift object tracking algorithm on each platform. Our experiments showed up to a 4X speed-up on a FPGA-based platform when using WLP approach, and up to a 6X speed-up on a GPU-based platform when using both NLP and WLP approaches. Also, if we execute both parallel and serial sections of the algorithm on a FPGA, this can produce a 21X speed-up.

References

[1] Altera sdk for opencl. `http://www.altera.com/literature/lit-opencl-sdk.jsp`.

[2] U. Ali and M. B. Malik. Hardware/software co-design of a real-time kernel based tracking system. *Journal of Systems Architecture - Embedded Systems Design*, 56(8):317–326, 2010. URL `http://dblp.uni-trier.de/db/journals/jsa/jsa56.html#AliM10`.

[3] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 142–149, 2000. ISBN 0-7695-0662-3. doi: 10.1109/CVPR.2000.854761. URL `http://dx.doi.org/10.1109/CVPR.2000.854761`.

[4] C. W. Fletcher, I. A. Lebedev, N. B. Asadi, D. R. Burke, and J. Wawrzynek. Bridging the gpgpu-fpga efficiency gap. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 119–122, 2011. ISBN 978-1-4503-0554-9. doi: 10.1145/1950413.1950439. URL `http://doi.acm.org/10.1145/1950413.1950439`.

[5] J. Fowers, G. Brown, P. Cooke, and G. Stitt. A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. In *In Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pages 47–56, 2012.

[6] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2013. ISBN 9780124055209, 9780124058941.

[7] P. Li and L. Xiao. Mean shift parallel tracking on gpu. In *Proceedings of the 4th Iberian Conference on Pattern Recognition and Image Analysis*, pages 120–127, 2009. ISBN 978-3-642-02171-8. doi: 10.1007/978-3-642-02172-5_17. URL `http://dx.doi.org/10.1007/978-3-642-02172-5_17`.

[8] E. Norouznezhad, A. Bigdeli, A. Postula, and B. C. Lovell. Robust object tracking using local oriented energy features and its hardware/software implementation. In *ICARCV*, pages 2060–2066, 2010. URL `http://dblp.uni-trier.de/db/conf/icarcv/icarcv2010.html#NorouznezhadBPL10`.

[9] F. Zhou, Y. Zhao, and K.-L. Ma. Parallel mean shift for interactive volume segmentation. In *Proceedings of the First International Conference on Machine Learning in Medical Imaging*, pages 67–75, 2010. ISBN 3-642-15947-8, 978-3-642-15947-3. URL `http://dl.acm.org/citation.cfm?id=1889135.1889144`.