DOI: 10.2352/ISSN.2470-1173.2016.14.IPMVA-375

Real-time Machine Vision with GPU-acceleration using Quasar

Jonas De Vylder, Simon Donné, Dirk Van Haerenborgh, Bart Goossens; dept. of Telecommunications and Information Processing, iMinds, IPI, Ghent University, Belgium

Abstract

The computational performance of graphical processing units (GPUs) has improved significantly, achieving even speed-up factors of 10x-50x compared to single-threaded CPU execution are not uncommon. This makes their use for high throughput machine vision very appealing. However, GPU programming is challenging, requiring a significant programming expertise. We present a new programming framework that mitigates the challenges common for GPU programming while maintaining the significant acceleration.

Introduction

There is an increasing interest in high throughput machine vision, both for industrial as bio-medical applications. These applications are typically constrained by a set of hard constraints related to accuracy and real-time performance. Often these constraints are contradictory, i.e. in order to achieve the required accuracy more complex techniques are needed, whereas achieving real-time performance often requires simple and computational efficient algorithms. One approach to mitigate this contradicting requirements is the use of hardware accelerators such as GPUs. Unfortunately, efficiently implementing algorithms on heterogeneous hardware is even for expert developers a challenging, time consuming task.

We present a new programming framework, Quasar, which facilitates GPU programming. Our high-level programming language relieves the developer of all implementation details such that he can focus on the algorithm and the required accuracy.

Background

Given the great speedup that can be achieved by using hardware accelerators, several approaches have been proposed in the past to facilitate programming on heterogeneous hardware. Most approaches fall within one of the following groups:

- The use of classic programming language, (e.g. java, MATLAB, python, etc.) in combination with a low-level programming language (e.g. CUDA, OpenCL, …) for the hardware accelerated parts.
- The use of programming languages with integrated support such as Mozilla Rust.
- The use of existing libraries as a set of accelerated building blocks (e.g. Thrust, CUSP, Armadillo MAGMA, FLAME, GPU-accelerated functions in OpenCV, …).
- The use of domain-specific languages (e.g. Halide, OpenACC, Rootbeet, …)

While all these approaches have their merit, they generally require fixed programming patters which limits the flexibility of a developer. Moreover, most of these approaches still require a lot of manual optimization in order to achieve proper acceleration, thus relying on the available programming expertise for specific hardware, instead of being properly device agnostic.

Quasar programming language

With this paper we present Quasar, a new programming framework consisting of a simple high-level programming language, an advanced compiler system, a runtime system and IDE.

The Quasar language is a high level scripting language with an easy to learn syntax similar to python and MATLAB (see Fig. 1 for an example). This makes Quasar well suited for fast development and prototyping. A Quasar program is first processed by a front-end compiler that automatically detects serial and parallel loops that could be accelerated by heterogeneous hardware. In the code of Fig. 1, the four for-loops are for example merged into a single kernel that can run in parallel, and this without any specific parallel constructs or other requirements of the programmer. In a second compilation phase, a number of back-end compilers processes the output of the front-end compiler, thus generating C++, OpenCL, LLVM and/or CUDA code. Based on the generated code the runtime system can dynamically switch between CPU and GPU. This automatic scheduling at runtime is done by analyzing the load of all devices, the memory transfer cost and the complexity of the task. This way, the programmer is relieved from complicated implementation issues that are common for programming heterogeneous hardware.

```
function [x,y]=matchTemplate(im:mat,mask:mat)
    [smx,smy]=size(mask)
    simil=zeros(size(im)-size(mask))
    %iterating over the image
    for ix=0..size(im,0)-smx
        for iy=0..size(im,1)-smy
            %iterating over the mask
            for mx=0..size(mask,0)-1
                for my=0..size(mask,1)-1
                    simil[ix,iy]+=(im[ix+mx,iy+my]-mask[mx,my])^2
                end
            end
        end
    end
    [x,y]=find_index_min(simil)
end
```

Figure 1 example Quasar code - calculation of the location of the mask in the image based on the MSE.

High throughput machine vision

We evaluate the usability of our programming framework using three different machine vision methods. We chose these three methods for their usability in both industrial and biomedical inspection systems.

1. Super-pixel segmentation: An algorithm that clusters similar pixels in groups, so called super-pixels. The clustering is done based on intensity, while regularizing the shape of the super-pixels based on compactness [1].
2. Template matching: An algorithm that locates the location in an image that best resembles an input template. A simple implementation is shown in Fig. 1. In our implementation we

DOI: 10.2352/ISSN.2470-1173.2016.14.IPMVA-375

create an extra speedup by the use of an image pyramid with two levels.

3. Model based segmentation: robust segmentation by jointly optimizing a data-fit term based on intensity and a term regularizing the shape. While this method is robust against noise and clutter, it is computationally demanding since it requires an iterative optimization step [2].

Results

For each of the three mentioned algorithms we evaluate the performance of three different runs: a single-threaded C++ implementation and two Quasar runs, one only using a multi-core CPU, a second one combining CPU and GPU. All runs where executed on a machine with an Intel i7-4712HQ CPU and NVIDIA GeForce 750M graphics card. Table 1 summarizes the performance of all runs. Note that the Quasar implementation that runs on GPU achieves real-time performance for all three methods.

Table 1. Performance results for different applications.

	C++	Quasar (CPU)	Quasar (CPU+GPU)
Super-pixel segmentation	8.85 fps	9.54 fps	34.48 fps
Template Matching	2,91 fps	4.93 fps	64.01 fps
Model based segmentation	0.79 fps	1.18 fps	21.09 fps

Conclusion

Quasar allows developers to focus on design aspects of the algorithms rather than implementation aspects. We showed that using quasar, real-time performance can be achieved for several machine vision methods and this without any change to the algorithm or implementation.

References

[1] R. Achanta, A. Shaji, K. Smith, and A. Lucchi, "SLIC Superpixels Compared to State-of-the-art Superpixel Methods," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274–2281, 2012.

[2] Bresson, X.; Esedoglu, S.; Vandergheynst, P.; Thiran, J. P. & Osher, S. Fast global minimization of the active Contour/Snake model Journal of Mathematical Imaging and Vision, vol. 28, pp. 151-167, 2007.

Author Biography

Jonas De Vylder received his M.Sc. in informatics from Ghent University (2007) and his PhD in engineering science from Ghent University (2014). Since then he has worked in the Quasar team of iMinds – Ghent University. His work has focused on the development of efficient load balancing on heterogeneous hardware.