Study of combining GPU/FPGA accelerators for High-Performance Computing

BRUNO DA SILVA[1], AN BRAEKEN[1],
ERIK H. D'HOLLANDER[2], ABDELLAH TOUHAFI[1],
JAN G. CORNELIS[3] and JAN LEMEIRE[3]
[1]Erasmus University College, IWT Dept., Brussels
[2]Ghent University, ELIS Dept., Ghent
[3]Free University of Brussels, ETRO Dept., Brussels

Nowadays, processors alone cannot deliver what High-Performance applications are demanding. Often applications are faced with more and more complex algorithms. An alternative is to use hardware accelerators such as Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). GPUs have been increasingly successful due to their massive parallelism, unified programming model and regular design. While these accelerators are most appropriate for vector calculations in general, they are less suited for irregular calculations. On the other hand, FPGAs are a different kind of hardware accelerators that provides a programmable and massively parallel architecture. They have an open architecture which can be modeled after the data path and control path of the algorithm. This degree of freedom, however, poses an additional challenge to create efficient, error-free designs in a short time span.

The combination of the power of GPUs with the flexibility of FPGAs enlarges the scope of problems that can be accelerated [1][1]. We present a hybrid platform and a toolchain that allows to efficiently create programs using one or both technologies. The proposed architecture is composed of a multi-core Xeon E5506 CPU, a high-end Tesla 2050 GPU board and a modular accelerator board integrating two Virtex-6 LX240 FPGAs. In order to identify what kind of algorithms are best suited for each technology, the roofline models [6] of both technologies are superposed. The roofline model relates the maximum performance of the accelerator to the computational intensity (CI) of the algorithm. When the most compute-intensive part has been identified, through theoretical analysis or from experimental results, the roofline model is used to find the most appropriate hardware accelerator.

In the case of GPUs, the functional part to be accelerated is translated to OpenCL. For the FPGA part, as the development time of a design is one of the main problems, several High-Level Synthesis (HLS) tools such as Xilinx's AutoESL and the Riverside Optimizing Configurable Computing Compiler ROCCC were considered to reduce the FPGA design complexity. We have chosen the open-source ROCCC to develop our first designs. On the host side is the C++ code managing the GPU/FPGA communication and the functional parts. The GPU/FPGA modules are modeled as function calls of the main function running on the host. The C++ code is compiled and during the execution the GPU and the FPGA are initialized with the proper kernel or bitmap respectively. The FPGA code has to be previously compiled and synthesized to generate the bitmap file. The complete toolchain is depicted in figure 1.

Since the HLS tools offer many optimizations and allow a fast design exploration, we have studied the resulting designs in order to best exploit the HLS tool as well as the target FPGA. For each design the latency, the resource consumption and the throughput have been examined. Of special interest is the impact of the resource usage on

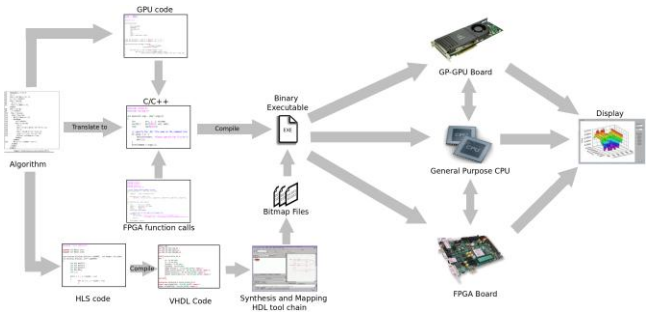

Figure 1: *An algorithm is converted into a C ++ program with mixed code fragments for the three platforms, CPU, GPU and FPGA. The executable communicates with the GPUs and FPGAs using API libraries.*

the performance estimated by the roofline model. This allows to identify the most appropriate compiler directives yielding the best performance within the limits of the resource footprint. The productivity of the C-to-VHDL compiler is estimated by the ratio of the number of lines in the original C code to the number of lines in the generated VHDL code. This is a well-known metric that has been used to compare the development time of compilers in [5]. Using this metric the impact of the different compiler options of ROCCC on the development time can be measured.

As a first step, several image processing algorithms were implemented both independently on the GPU and the FPGA. Experimental results comparing ROCCC with a hand-made implementation show that the modular FPGA high-performance board offers a scalable solution which is able to perform as good as or better than the corresponding GPU design. Once the results are depicted on the roofline model, it is possible to identify what kind of optimizations can be applied (figure 2). In the case of the FPGAs, as the computational performance is defined by the algorithm's resource consumption, to obtain the maximum attainable performance it is required to increase the parallelism, as for example replicating the functional blocks. On the other hand, to increase the CI, the number of operations per received data must increases. Thanks to the data-reuse by the smart buffer component of ROCCC, the memory accesses are significantly reduced, increasing the final CI and the resulting performance. However, while both the GPU and FPGA excel in particular applications, both devices suffer from the limited I/O bandwidth to the processor, in this case by the PCI express bandwidth. Despite the I/O bottleneck, several computationally hungry algorithms can be accelerated by executing the most intensive parts on the appropriate technology [2],[3].

As a second step, a promising algorithm has been explored and implemented using the aforementioned toolchain, splitting and

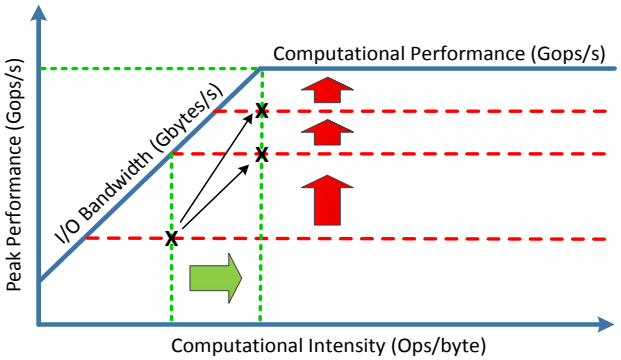

Figure 2: *The roofline model suggests several optimizations when the results of a measured algorithm are depicted on the roofline. In particular, an improvement of the CI (green arrow) can lead to a better performance by reducing the I/O bottleneck impact, while in the horizontal part the use of more resources to obtain more parallelism, can lead to a higher peak performance (red arrow).*

distributing the functionality between the GPU, FPGA and CPU in order to validate the hybrid concept. The candidate is an object recognition application called fastHOG which has been implemented in CUDA to run on GPUs [4]. The application consists of several algorithms such as the Histogram Oriented Gradients (HOG) and the Support Vector Machine (SVM) computations. Our estimations show that both algorithms are good candidates to be executed on the FPGA, in particular HOG since histogram implementations have been traditionally well fitted to FPGAs. The HOG part inputs the gradients of the image and computes normalized histograms of same sized blocks. The implementation of the histogram computation and normalization is able to exploit the benefits of the FPGAs using parallelism, pipelining and streaming. Thanks to those features, which can be mainly controlled through the HLS compiler directives, this approach is able to increase the final performance. In fact, our experimental results show that the computation of HOG on the FPGA is substantially faster than on the GPU. However, the external data rearrangement and the communication overhead greatly reduce the final performance. A further improvement is to implement the SVM algorithm on the second FPGA (figure 3) to avoid extra communication between the FPGA and the GPU.

To conclude, the FPGA is able to outperform high-end GPUs, however the performance of a combined system may be hampered by the extra cost of the PCIe communication overhead.

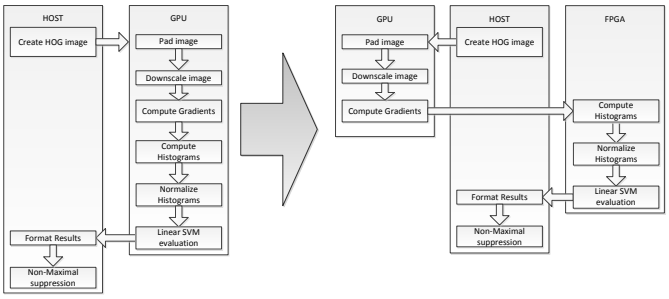

Figure 3: *The object recognition application called fastHOG, designed for GPUs, is adapted to be partially executed on the FPGA. The Histogram computation and the SVM are ideal candidates for FPGAs.*

REFERENCES

[1] Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J. and others 2009. A view of the parallel computing landscape. *Communications of the ACM*. 52, 10, 56–67.

[2] Bauer, S., Kohler, S., Doll, K. and Brunsmann, U. 2010. FPGA-GPU architecture for kernel SVM pedestrian detection. *Proceedings of the Computer Vision and Pattern Recognition Workshops (CVPRW)*, 61–68.

[3] Inta, R., Bowman, D.J. and Scott, S.M. 2012. The "Chimera": An Off-The-Shelf CPU/GPGPU/FPGA Hybrid Computing Platform. *International Journal of Reconfigurable Computing*. January 2012, Article 2, 10 pages.

[4] Prisacariu, V. and Reid, I. 2009. *fastHOG-a real-time GPU implementation of HOG*. Technical Report #2310/09. University of Oxford.

[5] Sackman, H., Erikson, W.J. and Grant, E.E. 1968. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*. 11, 1, 3–11.

[6] Williams, S., Waterman, A. and Patterson, D. 2009. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*. 52, 4, 65–76.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Heterogeneous (hybrid) systems —*Performance*; D.1.3 [**Programming Techniques**]: Parallel programming—*High-level synthesis*

General Terms: Performance, Experimental

Additional Key Words and Phrases: OpenCL, FPGA, GPU, accelerators, programming toolchain

This research was carried out within the framework of the IWT-TETRA project GUDI, "A combined GP-GPU/FPGA desktop system for accelerating image processing applications," IWT-100132.