Transparent Code Offloading on FPGA

Anthony Convers and Baptiste Delporte and Xavier Ruppen and Roberto Rigamonti and Alberto Dassatti

HES-SO — REDS Institute, HEIG-VD — School of Business and Engineering Vaud
CH-1400 Yverdon-les-Bains, Switzerland (name.surname@heig-vd.ch)

Genomics, molecular dynamics, and machine learning are just the most recent examples of fields where FPGAs could provide the means to achieve interesting breakthroughs [1–3]. FPGAs have proven indeed to be a viable solution for energy efficient high-performance computing [4], [5]. However, HDL programming requires considerable multidisciplinary skills, experience, large budgets, time, and a bit of wizardry [6]. The use of FPGAs has dramatically raised the overall complexity of the system, as exploiting the available capabilities now demands a wide range of competencies which are not always in the background of a company or institution. Furthermore, the required effort often limits the applicability to a reduced set of supported brands/models, while being effective only when predicted usage patterns match the actual ones. Given that most cloud applications are short-lived — their lifespan can be as short as one week [7] —, the investment simply does not pay off.

High-Level Synthesis (HLS) [8] partially mitigates the above-mentioned problems by removing the language-barrier, but compiling and deploying a bitstream could take up to a few days (depending on the complexity of the design) [6], and again its development requires establishing a-priori usage patterns that might lead to a suboptimal usage of the available resources.

In this demo we propose a multi-vendor LLVM-based automated framework that can transparently — that is, without the user or the developer being aware of it — offload computing-intensive code fragments to FPGAs [9]. Our solution requires no changes in the code, not even pragma indications to guide the optimization, and dynamically adapts its behaviour to the available data and the workload of the system. The system operates on the Intermediate Representation of the original code to achieve language-agnosticity, and it identifies parallelizable, computationally-intensive code fragments that are then dispatched to a data flow overlay architecture built on top of the FPGA. The overall process requires hundreds of microseconds, since the bitstream we use is fixed, and can be easily reverted should the outcome be unsatisfactory.

At the heart of our system, depicted by Fig. 1, lies a Just-In-Time (JIT) compiler, coupled with the Linux perf_event performance monitor to automatically detect hot code fragments. Once a region is identified, it is analyzed to expose parallelization opportunities. The Control Flow Graph and the Data Flow Graph are then extracted and merged, and the overlay pre-synthesized on the FPGA is reconfigured on-the-fly to execute the new model. We have chosen to adopt the overlay architecture for pipelined execution of data flow graphs presented in [10], performing the Place&Route operation using a custom-made randomized algorithm. Finally, we alter the execution flow of the code and we feed the FPGA with the data provided by the application at hand.

Our approach has shown to be effective also on more traditional heterogeneous systems, embedding GPUs [11] or DSPs [12]. To the best of our knowledge, no other approach proposed thus far is capable of achieving these goals.

REFERENCES