Automating FPGA Offload with Symbolic Execution

1 Background

Field-programmable gate arrays (FPGAs) are an essential component of the latest datacenters and compute clusters. They are especially valuable as compute accelerators since they can often outperform CPUs at specific tasks [1]. However, their design also presents a substantial challenge to the users of these clusters and datacenters. FPGAs require the user to write program descriptions in a hardware description language (HDL) such as Verilog, which has a significant learning curve compared to high-level languages such as C.

While solutions such as high-level synthesis (HLS) can transpile a high-level language such as C to HDL, integrating the FGPA into a compute workflow still requires significant effort. Since not all logic is suitable to be run on an FPGA, some code segments still have to be run on the host. Thus, communication libraries between the CPU and FPGA need to be written carefully to ensure maximum performance and often require domain-specific knowledge on interconnects such as PCIe [2]. Validation of FPGA code is also extremely time-consuming due to the inability of CPUs to effectively emulate FPGAs [3]. Thus, it may often be impractical to spend time integrating FPGAs into a workflow, despite their substantial performance benefits. A potential solution may be to find a way automatically select which components of a program to offload to an FPGA.

2 Proposal

This project will address the difficulty in creating performant and reliable FPGA offloading solutions by creating a compiler that will automatically convert code segments into HDL. This automation can work since the semantics of a program can reflect its suitability for offloading.

The most primary requirement for logic that is suitable for offload to an FPGA is that the code must be able to be pipelined [3]. Namely, the code must have minimal overlapping operations, and side effects cannot modify the execution of the function. Both of these requirements can be tested for through binary analysis.

Symbolic execution is especially suitable for checking code segments for offload suitability. A symbolic execution engine such as `angr` can generate an abstract syntax tree (AST) from a given binary [4]. The AST represents the set of operations that will define the value of a variable at some time during the program's execution. Heuristics to determine if the function is pipelineable can be created using this data.

The data dependency graph (DDG) will provide an important metric for determining offload suitability. Each node in this graph is a variable in the function, and each directed edge represents a variable that influences the final state of another variable. If cycles exist in such a graph, it may indicate that this function might not be efficiently pipelined. The control flow graph (CFG) of the function can also be used to identify loop conditions that may not be efficiently pipelined.

Other suitability analyses will need to be made as well to ensure that the FPGA can implement the required design. For example, the total size of all variables cannot exceed the total number of registers available on the FPGA, and allocated memory cannot exceed the total memory on the FPGA. Other considerations may include identifying code regions that may work well on specialized components within the FPGA aceclerators. For example, many FPGA vendors offer intellecutal property cores (IPs) that offer better performance than custom implementations of the same function. The compiler may need to take these IPs into account by generating profiles for each FPGA.

This project will create a compiler that can compile C source code into both a CPU portion and an FPGA portion. The compiler will compile the input C program into a binary and then analyze each function of the binary as described above. This allows the program to make use of compiler optimizations such as code inlining and loop unrolling detect code regions that may not otherwise be offloadable.

The compiler will then automatically determine which sections of the program can be offloaded to FPGAs through the heuristics above. It will extract the largest selection of code regions that was found to be offloadable and decompile that binary to HLS C. This preserves the semantics of the compiled program and allows conversion to the subset of C used by the HLS suite while keeping compiler optimizations. It will also add functions for communication between the host and the FPGA, and finally flash the compiled HLS project to the FPGA.

This project will also include a library for both the FPGA and the CPU which will enable communication between them. This can be done by building a kernel driver and a Verilog `module` that will interface between the offloaded code segments and the vendor's PCIe IP.

3 Evaluation

This project can be evaluated on any computer with an FPGA card supporting PCIe. Real-world programs making heavy use of parallelization will be evaluated. This will include applications such as the NAMD molecular dynamics library [5] or the Stockfish chess engine. These programs are highly parallelized and will test the pipeline heuristic of the prototype. Furthermore, some of these projects also support common APIs such as OpenMP and OpenMPI, which this project should also support.

Benchmarks such as those in NAMD and Stockfish also have custom metrics that can also be compared between the two versions. These results could take into account side effects that may not be apparent when looking solely at latency and throughput.

Performance comparisons can then be made between the host-only version and the offloaded version. Significant metrics to test include the latency and throughput of the application when offloaded, as well as total runtime of each version. The performance of the communication library will play a substantial role in the test results as well.

4 Related Work

Sommer et al. presented an addition for the OpenMP API that runs user-specified blocks on an FPGA using Vivado HLS [2]. While this presents an option to write FPGA offloads without writing any communication libraries, the project cannot automatically select code locations to offload. Instead, they are specified using a `#pragma` directive, like a standard OpenMP call. However, the project also implemented a communication library between the host and the FPGA using PCIe.

Yamato presented a project that allowed for automatic offloading of specific function blocks [6]. This project relies on static code analysis on the source code and therefore cannot account for compiler optimizations that may unroll loops and present new opportunities for offloading. Use of symbolic execution on a compiled binary will take advantage of compiler optimizations such as loop unrolling to identify offloadable blocks that may not be identifiable in the source. Furthermore, Yamato's project also requires the source code of the original program and thus cannot support offloading from a binary.

5 Relevance to the Department of Defense

This project will enable software developers to deploy their applications to FPGA accelerators with minimal effort. If successful, the compiler developed by this project may significantly increase the performance of many common computing operations.

This is especially relevant for ARL's research focus on supercomputing technologies, under section KCI-CS-1 of BAA W911NF-17-S-0003. In particular, this work will allow users to leverage large-scale on-chip parallelism without the specialist knowledge in RTL programming usually required.

This project may also be applicable to ARL's dynamic binary translation focus, by allowing developers to target FPGAs with pre-existing binaries. While the proposed project will work with source code, it can also be used to translate binaries as it operates at an intermediate representation level.

6 Societal Impacts

Highly parallel processing is an essential part of many scientific discoveries that may prove to be societally significant. For example, the SUMMIT supercomputer was used to model the COVID-19 spike protein and discovered a set of molecules that could disrupt the virus' infectivity [7]. Similar techniques are used to model the Earth's climate and present possible futures [8]. However, these calculations require considerable amounts of compute time, and the clusters often draw a large amount of power while doing so.

FPGAs also often offer better power efficiency and can potentially speed up computations by orders of magnitude [9]. Accelerating similar scientific projects through FPGA offloading could benefit other societally significant work by decreasing computation time and power usage. Having their results sooner also allows scientists to spend less time waiting for their experiments to finish or allow previously infeasible calculations to be undertaken.

7 Relevant Qualifications

A significant portion of my undergraduate experience focused on building high-performance distributed systems for various networked use cases. My research on defenses against denial of service attacks familiarized me with the Linux kernel. Similarly, my project at a NNSA laboratory on building network testbed technology provided me with a solid background in building high performance software systems. Furthermore, I am familiar with software development through my participation in the Mars 2020 project that will be essential for building a prototype with real-world applicability.

Furthermore, I also have extensive experience working with FPGAs in a high performance setting. My work with FPGA SmartNICs this past summer provided me with the experience needed to build designs for such accelerators. I am also currently working on a related project that is developing a high performance Verilog-to-C transpiler for networking tasks. These experiences has provided me with additional insights into designing for FPGAs with both HDL and Verilog.

I also have significant experience working with the `angr` binary analysis toolkit, including on a DARPA-funded project to extract mathematical expressions from cyber-physical binaries. This work familiarized me with symbolic execution and static analysis methods, and will be invaluable for the success of this project.

References

[1] Yufei Ma, Yu Cao, Sarma Vrudhula, et al. “Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks”. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017, pp. 45–54.

[2] Lukas Sommer, Jens Korinth, and Andreas Koch. “OpenMP device offloading to FPGA accelerators”. In: *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2017, pp. 201–205.

[3] Philip Simpson. *FPGA design*. Springer, 2010.

[4] Fish Wang and Yan Shoshitaishvili. “Angr-the next generation of binary analysis”. In: *2017 IEEE Cybersecurity Development (SecDev)*. IEEE. 2017, pp. 8–9.

[5] James C Phillips, Rosemary Braun, Wei Wang, et al. “Scalable molecular dynamics with NAMD”. In: *Journal of computational chemistry* 26.16 (2005), pp. 1781–1802.

[6] Yoji Yamato. “Automatic offloading method of loop statements of software to FPGA”. In: *International Journal of Parallel, Emergent and Distributed Systems* (2021), pp. 1–13.

[7] Micholas Smith and Jeremy C Smith. “Repurposing therapeutics for COVID-19: Supercomputer-based docking to the SARS-CoV-2 viral spike protein and viral spike protein-human ACE2 interface”. In: (2020).

[8] MS Mizielinski, MJ Roberts, PL Vidale, et al. “High-resolution global climate modelling: the UPSCALE project, a large-simulation campaign”. In: *Geoscientific Model Development* 7.4 (2014), pp. 1629–1640.

[9] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. “Performance comparison of FPGA, GPU and CPU in image processing”. In: *2009 international conference on field programmable logic and applications*. IEEE. 2009, pp. 126–131.