POSTER: High-Level Synthesis of the OpenMP runtime to improve the generation of parallel accelerators

Giovanni Gozzi
giovanni.gozzi@polimi.it
Politecnico di Milano
Milan, Italy

Michele Fiorito
michele.fiorito@polimi.it
Politecnico di Milano
Milan, Italy

Serena Curzel
serena.curzel@polimi.it
Politecnico di Milano
Milan, Italy

Fabrizio Ferrandi
fabrizio.ferrandi@polimi.it
Politecnico di Milano
Milan, Italy

CCS CONCEPTS

• **Hardware → Electronic design automation**; **Reconfigurable logic and FPGAs**; **Emerging languages and compilers**.

KEYWORDS

FPGA, OpenMP, High-Level Synthesis

ACM Reference Format:
Giovanni Gozzi, Michele Fiorito, Serena Curzel, and Fabrizio Ferrandi. 2023. POSTER: High-Level Synthesis of the OpenMP runtime to improve the generation of parallel accelerators. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3587135.3592182

1 INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are reconfigurable hardware components that can be programmed using a hardware specification language such as Verilog or VHDL. Writing code to program an FPGA requires a set of different skills than software programming, for example, related to the use of signals and registers instead of variables and the explicit description of parallelism. High-Level Synthesis (HLS) tools can help by automatically translating software specifications into hardware descriptions.

This work aims to further improve the HLS process by adding the possibility to synthesize parallel applications to increase the performance of the resulting hardware accelerator. In particular, we extend the open-source Bambu HLS tool [2] to support the generation of parallel architectures described with OpenMP directives (pragmas). We draw inspiration from the *Svelto* architecture [3], but we employ a fundamentally different approach: instead of pattern matching and substituting a small subset of pragmas, we use Bambu to synthesize part of the OpenMP LLVM runtime. In fact, using pattern recognition, the compilation flow applies specific transformations on code regions marked by the programmer, and it becomes difficult to move instructions outside or inside the parallel region. Moreover, the set of patterns that can be recognized is always limited and difficult to extend; the implementation of the runtime, instead, improves the performance and flexibility of the HLS compiler.

In the proposed synthesis flow, all the OpenMP runtime functions and primitives are mapped on low-level C or Verilog components. The resulting architecture is highly configurable: several parallel cores are instantiated, and a group of OpenMP threads is allocated on each of them; context switches are performed to avoid idle cycles when memory requests are issued. The architecture is particularly suitable for complex applications (e.g., graph processing) with irregular memory access and work distribution patterns.

CF '23, May 9–11, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0140-5/23/05.
https://doi.org/10.1145/3587135.3592182

2 PROPOSED SYNTHESIS FLOW

Users identify parallel regions in the input code (C/C++) using OpenMP pragmas, which can also specify other important information like synchronization mechanisms and the number of threads. Then, the hardware generation flow in Bambu starts from the high-level OpenMP representation and uses the Clang LLVM frontend to lower it to an LLVM intermediate representation (LLVM IR) containing calls to OpenMP runtime functions. Calls to runtime functions are generated from the translation of OpenMP constructs, and they describe how to perform the parallelization. They contain particular operations called primitives that are an extension of the C language and serve for low-level synchronization and thread handling. Bambu replaces runtime function calls in the LLVM IR with corresponding calls to functions from a modified implementation of the LLVM OpenMP runtime library, which has been streamlined and reduced to the essential in order to generate small and fast components when synthesized on FPGA. For example, the logic to support the reuse of threads has been removed because hardware threads cannot be reassigned to other tasks that require a different datapath and controller. Low-level synchronization primitives such as 'wait for thread' or 'barrier reached' have been implemented as Verilog components. Bambu HLS performs its standard set of analyses and transformations on the IR to optimize the code and prepare it for hardware synthesis. Finally, the remaining calls to OpenMP primitives are mapped on hardware components from the Bambu library and connected to the rest of the design.

3 ARCHITECTURE

The architecture generated by Bambu is highly configurable: the user can tune every element using parameters and, in this way, improve the performance of the resulting hardware. Bambu replicates each OpenMP parallel region in the input code and creates parallel hardware cores; a set of OpenMP threads is statically allocated to each core. Every core has a hardware module that manages the active context and performs a context switch between threads when there is an operation with a latency greater than one. A typical situation that requires context switching is when a thread accesses the external memory: instead of stalling the execution for tens of clock cycles, the execution state (context) is saved, and the core is dedicated to the execution of a different thread. Registers and

Configuration	Cycles	LUTs	Register	Slice	DSP	Frequency	Speed Up
CC-Baseline	22463720	1404	1198	499	0	192.20	1
CC-1core-8cs	3061152	5944	1893	1958	0	111.63	7.34
CC-8core-1cs	2889317	15470	11829	5286	0	124.35	7.77
CC-8core-8cs	610005	43601	13916	13474	21	105.70	36.82

Table 1: GAPBS Connected Components
Results obtained on the Connected Components (CC) test of the GAP Benchmark Suite [1] on a Xilinx xc7vx690t FPGA with a target frequency of 100 MHz. The external memory has 16 channels with a latency of 200 ns, connected through the NoC to 8 accelerator channels. The design does not utilize any BRAMs. DSPs are used to compute the bounds of the loop iterations; on core 0, and when there is only a context on the core, they are simplified by the synthesis tool.

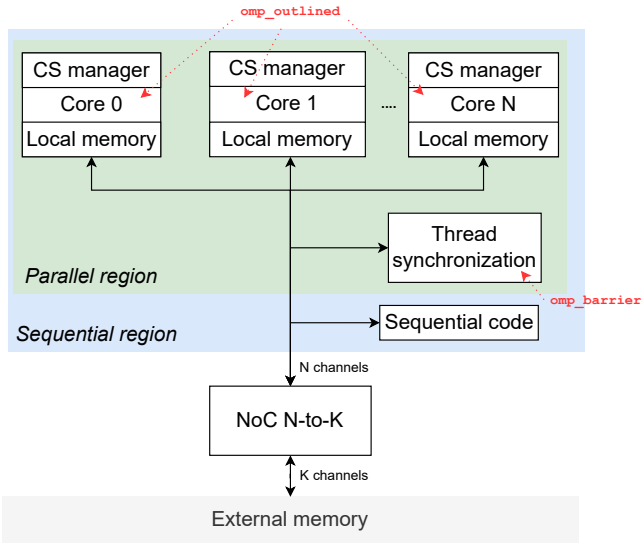

Figure 1: Multi-threaded architecture with a network-on-chip to arbiter requests to the external memory.

local memory are replicated on each core to store data from multiple contexts; Bambu performs a liveness analysis to only replicate hardware elements that are actually needed when a context switch is performed.

Several components are allocated to guarantee the correct execution and synchronization between threads, corresponding to the critical, wait, barrier, and reduction OpenMP primitives. The architecture is generated starting from the OpenMP runtime, so it is not limited to a specific set of patterns and it can be adapted to input programs containing e.g. nested loops or OpenMP sections.

Storage for variables can be allocated in local or external memory, or memory belonging to other cores (this is the case of reduction variables). Every core has a memory channel to retrieve variables that are allocated outside of it, filters are automatically generated to redirect the request to the correct location. The number of cores is not linked to the number of external memory channels going out of the FPGA, which can lead to interconnection problems. Instead of using hardware modules to arbiter the incoming requests, we introduced a custom network-on-chip (NoC) interconnect that connects multiple sources to multiple destinations keeping the clock period low. The NoC consists of a group of interconnected nodes that can be crossed by a memory request in a number of cycles proportional to the logarithm of the number of nodes. The proposed architecture can be connected to different kinds of external memory, banked or not banked, and it can manage multiple pending requests.

4 CONCLUSION

Preliminary results reported in Table 1 show that our approach can speed up applications by adding more cores, performing context switches, or both. Until the load on the memory channels is limited, the performances obtained are near the theoretical limits, and they are satisfying even as traffic increases due to an increase in cores or contexts issuing requests.

By synthesizing the OpenMP runtime, we extended Bambu with a new approach to the problem of generating parallel hardware for FPGA, improving the integration between the front-end compilers and the back-end synthesis engine. The proposed approach features a highly configurable architecture that can be adapted to parallelize complex applications written in OpenMP. The architecture can achieve both SIMD (e.g., OpenMP for) and MIMD (e.g., OpenMP sections) parallelism and it can hide the latency of external memory accesses through context switching without losing clock cycles.

There are more OpenMP constructs that we would like to support in the future, like for example tasks and atomic operations. Another possible improvement is the introduction of a broadcasting mechanism to forward a response from the external memory to multiple threads; this would enable the generation of efficient accelerators for sparse applications, as multiple accesses to the same memory location would quickly become a bottleneck otherwise. All of these developments will allow us to improve the quality of the generated accelerators, but also to further extend the applicability of the proposed approach to more and more application domains[1].

REFERENCES

[1] S. Beamer, D. Patterson, and K. Asanović. 2021. GAP Benchmark Suite. https://github.com/sbeamer/gapb
[2] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo. 2021. Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*. 1327–1330.
[3] M. Minutoli, V. G. Castellana, N. Saporetti, S. Devecchi, M. Lattuada, P. Fezzardi, A. Tumeo, and F. Ferrandi. 2022. Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics. *IEEE Trans. Comput.* 71, 3 (2022), 520–533.

[1]This research has been partially supported by the HERMES project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 101004203.