OpenMP Device Offloading to FPGAs using the Nymble Infrastructure

Jens Huthmann[1], Lukas Sommer[2], Artur Podobas[3], Andreas Koch[2], and Kentaro Sano[1]

[1] Riken Center for Computational Science, Japan
{jens.huthmann, kentaro.sano}@riken.jp
[2] Embedded Systems and Applications Group, TU Darmstadt, Germany
{sommer, koch}@esa.tu-darmstadt.de
[3] Royal Institute of Technology, KTH, Stockholm Sweden
artur@podobas.net

Abstract. Next to GPUs, FPGAs are an attractive target for OpenMP device offloading, as they allow to implement highly efficient, application-specific accelerators. However, prior approaches to support OpenMP device offloading for FPGAs have been limited by the interfaces provided by the FPGA vendors' HLS tool interfaces or their integration with the OpenMP runtime, e.g., for data mapping.
This work presents an approach to OpenMP device offloading for FPGAs based on the LLVM compiler infrastructure and the Nymble HLS compiler. The automatic compilation flow uses LLVM IR for HLS-specific optimizations and transformation and for the interaction with the Nymble HLS compiler. Parallel OpenMP constructs are automatically mapped to hardware threads executing simultaneously in the generated FPGA accelerator and the accelerator is integrated into `libomptarget` to support data-mapping.
In a case study, we demonstrate the use of the compilation flow and evaluate its performance.

Keywords: FPGA, OpenMP, Device offloading, Heterogeneous, LLVM, HLS

1 Introduction

As the end of transistor scaling [30] draws near, researchers are actively pursuing and evaluating alternative emerging architectures and computing paradigms, with which they hope to continue performance scaling we have grown used to rely on. Among the more salient of these emerging architectures are reconfigurable systems, whose silicon plasticity/reconfigurability provides a partial remedy for the end of Moore's law [22]– we do not need more transistors, we just need to repurpose the existing transistor to better fit the requirements of our applications.

Today, Field-Programmable Gate Arrays (FPGAs) are among the more popular and mature reconfigurable systems available. While early FPGAs had limited computing capabilities, and were primarily used for circuit simulation and

digital signal processing, modern FPGAs – on the other hand – feature tens of TeraFLOP/s of raw single-precision performance, and are capable of rivaling both general-purpose and graphics processing units (GPUs) in power efficiency and/or raw execution performance. Furthermore, with the increased maturity of High-Level Synthesis [12] tools, using FPGAs is no longer monopolized by hardware architectures, and instead, anyone with knowledge of C/C++/Java programming can map applications onto these exciting new architectures. Today, several research groups have already mapped important High-Performance Computing (HPC) applications onto FPGAs, with benefits illustrated over existing approaches [33, 16, 35, 34, 26]. These efforts have led to several research laboratories setting up large FPGA-based testbeds to investigate the role of these reconfigurable devices in a post Exa-scale era, such as the Noctua cluster at Paderborn University or the Cygnus cluster at University of Tsukuba.

In this paper, we present the Nymble OpenMP HLS infrastructure, which is a self-contained compilation tool-kit for running (a subset of) OpenMP constructs on FPGAs, and also visualize them using the Paraver [23] visualization tool. Unlike existing OpenMP HLS approaches, which use source-to-source compilation and rely on commercial black box compilers for hardware generation, Nymble is transparent and fully transforms OpenMP code down to Register Transfer Level (RTL) Verilog code without external dependencies. This, in turn, enables users to get a better insight into what hardware is actually generated, while at the same time providing an open platform for FPGA-based OpenMP research.

Our contributions in this paper are:

- A description over the Nymble infrastructure, including details on the front-end compilation and the hardware generation & architecture, including which OpenMP constructs Nymble supports and how they are implemented,
- A use-case showing how Nymble transforms well-known OpenMP code into hardware, including empirical performance evaluation, and
- A discussion on the future of OpenMP for FPGAs, including challenges and directions

2 Motivation

Today, FPGAs are being considered to complement (and compete with) the general-purpose processor and GPUs that currently reside in modern HPC infrastructure. Several research laboratories are already setting up large FPGA-based testbeds to investigate the role of these reconfigurable devices in a post-Exa-scale era, such as for example the Noctua cluster at Paderborn University or the Cygnus cluster at the University of Tsukuba.

Meanwhile, using these accelerators in a user-friendly way (that is, without resorting to writing RTL code), is often limited to using vendor-specific toolchains, such as for example Intel's OpenCL SDK for FPGA [8] or Xilinx SDSoC/SDAccel [32]. While these toolchains are often high-performing, they are also very tied to a specific execution model. Furthermore, adding or researching into alternative programming models using these vendor solutions (such as for

example OpenMP) is challenging, because tools are closed source, and even if some aspects can be changed (such as the Board Support Package, BSP), these changes become non-trivial.

There are methods to extend functionality, such as using source-to-source methods to transcompile OpenMP [10], but these methods have no way of even remotely controlling or dictating how the underlying hardware is generated. Finally, vendor tools and road-maps are not always necessarily aligned with what we as users or researchers need, meaning that it is imperative to look at alternative approaches, in particular for guiding and doing research on OpenMP execution on future FPGAs. The Nymble OpenMP infrastructure aspires to be one such alternative for OpenMP researchers and users.

3 The Nymble OpenMP Infrastructure

The goal of this work is to develop a compilation flow that maps OpenMP target regions to FPGA-based accelerators without requiring manual intervention by the user. The compilation flow is based on the LLVM compiler infrastructure [18] and its implementation of OpenMP. In contrast to many prior approaches that use source-to-source transformations on AST-level to extract target regions for HLS (see Section 6 for detailed discussion), the compilation flow in this work uses LLVM IR to interact with the HLS tool. This approach facilitates code transformations that can be used to transform and optimize target regions, described in more detail in Section 3.1.

As the commercially available HLS-tools only provide source-level interfaces and no official interface on IR-level, the state-of-the-art academic HLS compiler *Nymble* [15] is used for the actual High-Level Synthesis of the target regions. Besides providing an IR-level interface, Nymblle also supports true multi-threading in the generated accelerators [14], described in more detail in Section 3.2.

3.1 Compilation Flow

Fig. 1 presents an overview of our compilation flow. For OpenMP device offloading, LLVM's Clang frontend uses separate compilation passes for host- and device code. For this work, the host compilation remains completely unchanged and therefore supports any host code and OpenMP host constructs that Clang supports.

The device compilation flow (shown on the right-hand side of Fig. 1) does not only support the basic `target` directive to denote target regions and the *full* range of data-mapping constructs (`map`-clause, `target data`-directive, array-sections, etc.), but also provides two kinds of parallelism: The `teams` or `parallel` construct can be used inside a target region to express parallelism, Section 3.2 explains how this parallelism is realized in hardware. Note that in our current prototype, only one of these constructs can be used at a time and nested parallelism is not supported. For the `teams` construct, the `distribute` construct is also supported to specify worksharing for a loop nest.

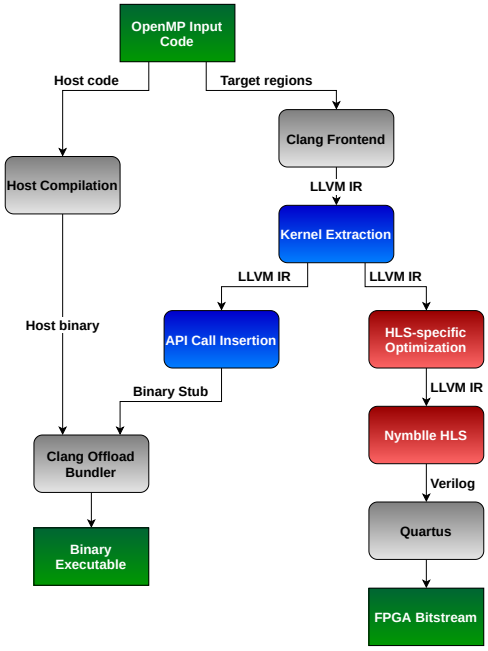

Fig. 1. Overview of the compilation flow.

Similar to many approaches investigated in the survey by Mayer et al. [21] (see Section 6 for detailed discussion), a binary stub for execution on the host machine is generated as one of the products of the device compilation flow. In this work, the binary stub is not only used to initiate the FPGA execution, but also to handle parallelism. Parallel constructs will spawn multiple software threads in the binary stub, these threads then interact with one hardware thread each in the FPGA-accelerator in an 1:1-relationship. This approach allows to re-use the standard mechanisms from LLVM's OpenMP runtime `libomp` to manage thread spawning and worksharing. Therefore, after generating LLVM IR in the Clang frontend, the *Kernel Extraction* splits the outlined target function into the stub to remain on the host and the actual target region kernel function for High-Level Synthesis.

The *API Call Insertion* then inserts calls to a thin wrapper library around Intel's *Open Programmable Acceleration Engine*[4] into the stub function to transfer function arguments and initiate hardware execution. Note that, in contrast to approaches such as [17], data-management is not handled via generated API calls, but rather through a plugin for LLVM's `libomptarget`, enabling the whole range of data-mapping clauses/constructs, including array sections and uni-directional transfers (`to`/`from` clause). The stub is then compiled for the host machine (`x86-64` in our case) and included in the binary executable using the Clang

[4] https://opae.github.io/

Offload-Bundler [1]. At runtime, the stub is loaded by `libomptarget` and initiates the execution on the FPGA accelerator.

The extracted HLS kernel undergoes a number of transformations and optimizations before actual High-Level Synthesis (*HLS-specific Optimizations* in Fig. 1). The transformations are mainly concerned with transforming OpenMP language constructs into constructs suitable for High-Level Synthesis. Currently, the prototype supports the OpenMP API runtime functions `omp_get_thread_num`, `omp_get_num_threads`, `omp_get_team_num` and `omp_get_num_teams`, which, in addition to `teams distribute`, can be used to assign individual workloads to the different threads. Besides that, the synchronization constructs `omp critical` and `omp barrier` are also supported inside target regions and mapped to efficient implementations using hardware semaphores.

Static allocation of thread-private memory inside the target region (`alloca` in LLVM IR) is also supported by the compilation flow and HLS backend and automatically mapped to low-latency accessible local memory (SRAM) on the FPGA device. Vector datatypes are also allowed in the target regions, but arithmetic operations on vectors are realized as individual operations on each vector element, as vector operations do not provide significant benefits in FPGA hardware. Therefore, to allow for more fine-grained scheduling during HLS, we automatically partition vector-wide thread-private memories into individual local memories for each element while preserving array semantics as one of the optimization steps.

The transformed LLVM IR is then passed to the Nymble HLS backend, which performs the typical HLS steps of allocation, binding and scheduling. For this purpose, the LLVM IR is transformed into a *control dataflow graph* (CDFG) representation, as described in [15]. More details on the mapping of different constructs to hardware will be presented in the next section.

The final product of the Nymble HLS backend is an HDL (Verilog) description of the accelerator, which is passed to Intel's Quartus software for synthesis and place-and-route, eventually yielding an FPGA bitstream.

3.2 Hardware Architecture

The overall hardware architecture of the generated FPGA accelerator is depicted in Fig. 2. The *Avalon slave interface* of the compute unit (CU) that is connected to the host is used as entry point for the hardware execution. The memory mapped register file can be used to pass kernel arguments and other information (e.g., thread ID) from the software thread to the corresponding hardware thread.

For larger data, the accelerator supports two different kinds of memory:

- Small, on-chip (SRAM) local memories (*LMEM*) are directly connected to the compute unit. These memories can be used as thread-private memory.
- *External memory* (DRAM) located on the FPGA-board can be used to hold large amounts of data and also for data-exchange with the host RAM using the OpenMP data-mapping constructs via the `libomptarget`-plugin. This memory is connected to the CU via an Avalon bus, with a dedicated Avalon master port per hardware thread.

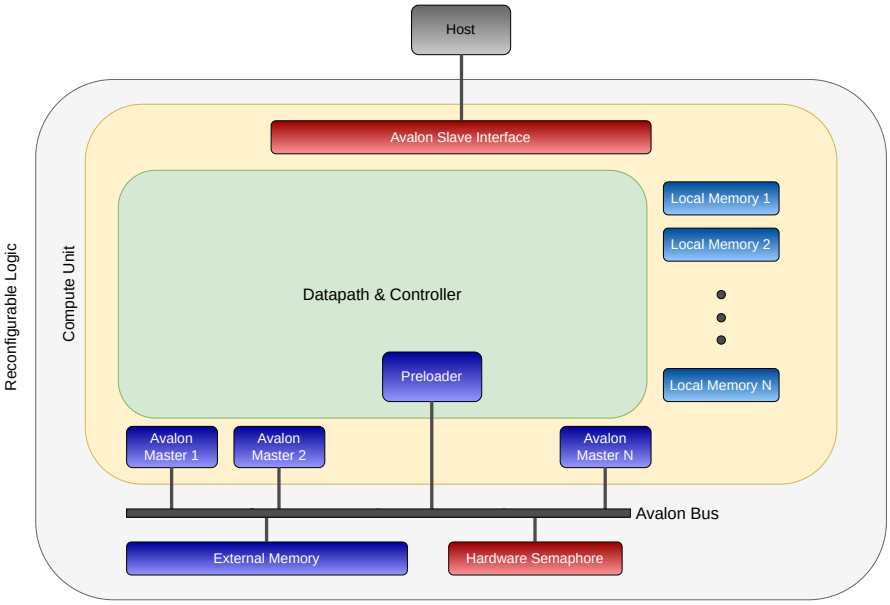

Fig. 2. Hardware architecture of the reconfigurable accelerator.

As the data-width of the external memory interface is usually higher than the size of single data-item of primitive type (e.g., `float`), vector data-types can be used in the OpenMP input code to improve the memory access efficiency. Where possible, vector-wide memory accesses are automatically mapped to Avalon burst accesses.

Another mechanism to further improve the memory access efficiency is the use of the *Preloader*. By using calls to the custom function `omp_target_preload` in the OpenMP input code to transfer data between global memory and thread-private local memory, the required data can be transferred efficiently in a single burst transfer. A more detailed discussion of the Preloader can be found in Section 4.1.

The Avalon bus system is also used to integrate the memory-mapped *Hardware Semaphore* that is used to realize the `omp critical` and `omp barrier` synchronization constructs.

The execution inside the *Datapath* is based on the Nymble-MT execution model presented in prior work by Huthmann et al. [14]. The unique feature of this execution model is the fact that it supports the simultaneous execution of multiple hardware threads in a *single* compute-unit, whereas most other FPGA-based approaches achieve thread-level parallelism through spatial replication of the compute-unit (e.g., [6], cf. Section 6 for discussion).

To allow for simultaneous execution of multiple hardware threads, the operations found in the data-flow graph of the kernel are organized into so-called *stages* according to their static HLS schedule. The different stages can operated

independently by the controller, allowing multiple threads to be active in different stages simultaneously. The stage-based execution model in addition also support loop pipelining.

The threads can operate completely independently of each other in this model, also allowing threads to start and finish at different points in time. Hardware threads are launched by their software counterpart (as stated in the previous section, we use a 1:1-relationship between software- and hardware threads) through the entry point in the Avalon slave interface. Parallelism in the OpenMP execution model (threads/teams) is automatically mapped to these simultaneously operating threads by the compilation flow presented here.

A major challenge in the stage-based execution model is the integration of operations for which the latency (in clock cycles) cannot be determined statically, e.g., accesses to external memory, which we call *variable-latency operations* (VLO). These operations are scheduled assuming their minimum latency. In case a VLO exceeds the assumed latency at execution time, the execution of the encountering thread is suspended until the VLO completes. To make sure that a single thread encountering a longer-than-expected latency does not block other threads, stages containing a VLO allow for thread re-ordering, i.e., threads can overtake each other in these stages.

3.3 Performance Visualization

Just as with any other device or target platform, the optimization of application code is an important step to achieve performance on FPGAs and is often an iterative process. To assist developers in this process, the compilation flow developed in this work provides mechanisms to automatically include various performance counters directly in the generated hardware. While the full details of the hardware implementation are out of scope for this work, the performance counters were designed to be as non-invasive as possible, i.e., to not have an impact on the performance of the investigated accelerator design, e.g. by increasing the initiation interval of pipelined loops.

The performance counters allow to capture important metrics such as memory bandwidth, arithmetic operations per time-interval (e.g. GFLOPs) or hardware thread idle times and facilitate the analysis and optimization of the target regions offloaded to the FPGA. After the execution on the FPGA completes, the collected performance data is exported in the Paraver trace format for use with the popular HPC performance visualization tool Paraver [23]. The integration with a state-of-the-art HPC visualization tool makes the performance analysis of the FPGA target regions more accessible for HPC domain experts.

4 Evaluation

To demonstrate the compilation flow from OpenMP with target offloading to FPGA-based accelerators, we use a well-understood benchmark as case study. The selected application allows to test the different features of the compilation

flow and architecture template by covering the supported OpenMP constructs as mentioned in the previous section, including synchronization.

For the application, a single compute unit is implemented inside the FPGA, supporting the simultaneous execution of up to four threads. The implementation of the compilation flow is based on LLVM release 9.0 and Quartus Prime version 18.1.2 is used for synthesizing the generated Verilog code to an FPGA bitstream.

The targeted FPGA is an Intel FPGA PAC D5005 card. The card is coupled via PCIe to the host processor, a quad-core Xeon Gold 5122 CPU which executes the host-portion of the applications and is also used for CPU benchmarking. Note that the performance figures always include data-transfers between host- and FPGA external memory via PCIe, initiated through `libomptarget`, i.e., the numbers reported here are end-to-end performance of the FPGA offloading.

4.1 Case Study: GEMM

As an example application, we use the general matrix multiplication (GEMM). The FPGA accelerator is compiled from a blocked version of GEMM and the different hardware threads compute distinct submatrices of the overall result matrix. Inside the computation of each thread, the computation is partially unrolled to exploit the potential of *spatial* parallelism provided by FPGAs. To reduce the number of expensive accesses to global, external memory, local memory is used to buffer inputs and intermediate results. To further improve the efficiency of memory access to the input matrices A and B, the threads preload blocks of the input matrices into the local memory using the preloader that is part of the compute unit. For users of the compilation flow, the preloading capability is available through a simple C++ template function called `omp_target_preload` (cf. Listing 1.1), which simply gets passed the relevant pointers to external and local memory and the number and type of the elements to load.

```
1 template <typename T, int ELEMENTS>
2 void omp_target_preload(size_t offset, size_t stride,
3   size_t num_transfers, void* globalSrc, void* localDst){...}
```

Listing 1.1. Definition of the `omp_target_preload`-function

The preloader will then collect the access to multiple elements in a single Avalon (burst) request, significantly improving the memory access efficiency. To further leverage the spatial parallelism, double buffering is implemented for the local memory and the preloading for the next block happens in parallel to the computation of the current block. All these optimizations have been implemented using standard OpenMP or, in case of unrolling (`pragma unroll`), compiler annotations and C++ constructs. The `omp_target_preload`-function was designed to be very generic and corresponds to a pattern often found in accelerator programming (e.g., GPU programming), the preloading of relevant input data from global memory to local memory. An usage example of the preload-function can be found in Listing 1.2.

```
1 void gemm(float* A,...){
2   [...]
3   VECTOR A_local[BUFFERING][BLOCK_SIZE];
4   omp_target_preload<float, BLOCK_SIZE>((i*DIM)+k, DIM,
        ↪ BLOCK_SIZE, (void*) A, (void*)
        ↪ &A_local[buffer%BUFFERING * BLOCK_SIZE]);
5   [...]
6 }
```

Listing 1.2. Usage example of the `omp_target_preload`-function (excerpt).

Fig. 3 shows the performance of the FPGA accelerator with different numbers of hardware threads executing simultaneously in the single compute unit for matrices of dimensions 8192 × 8192. While the performance of the accelerator almost doubles when going from a single to two threads, the increase slows down for three and four threads, respectively. In these cases, the threads do not only compete for compute resources in the multithreaded accelerator, but also for memory bandwidth to the external memory. The comparison with the BLAS implementation from the ATLAS library [31] on the Xeon CPU shows that the accelerator with a single thread outperforms a single thread on the CPU, but is not able to keep up with an execution with four threads on the CPU, partially also due to the data-transfers between host and FPGA.

In terms of hardware resource usage, the accelerator takes up 14% of logic resources, 16% of BRAM and 18% DSPs at a frequency of 183 MHz. Despite the relative low resource usage, it does not make sense to further increase the number of threads due to the negative impact on operating frequency. Instead, the remaining resources could be utitlized to duplicate the accelerator and compute on multiple compute units in parallel in future versions of the proposed architecture.

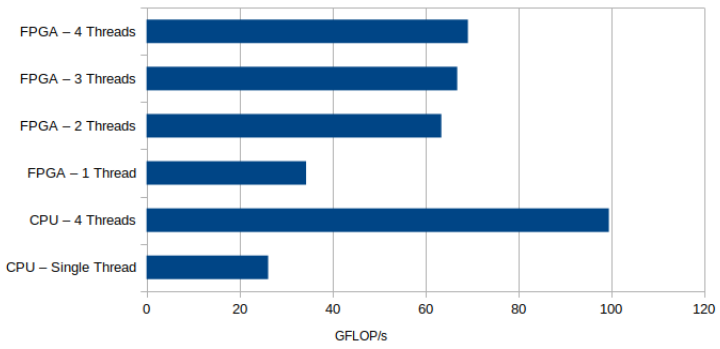

Fig. 3. Arithmetic performance of the blocked GEMM computation in GFLOP/s with different numbers of hardware threads simultaneously active in the compute unit.

In order to validate the support for OpenMP synchronization constructs via a lock implemented in the bus-attached hardware semaphore, an alternative version of GEMM, where each thread computes parts of the result for each element of the result matrix. The computed partial result is then added to the overall result inside a `critical` region. Even though the hardware semaphore allows for efficient locking, this version of GEMM, due to the very frequent access to global memory, delivers less performance than the optimized version using local memories described above.

5 Discussion

In this paper, we have demonstrated the Nymble infrastructure and shown that we can support a significant subset of OpenMP target offloading on FPGAs without much loss of generality, and that many of the properties (load-imbalance, scalability, etc.) materialize even in hardware. However, there are ample opportunities and future work for OpenMP on FPGAs, some of which we discuss herein.

OpenMP tasking, introduced in v3.0 (and dependent tasks in v4.0), is a construct that we would like to support in the Nymble subsystem. In theory, all necessary ingredients to support tasking is already provided by Nymble, and scheduling could be in a very software manner. However, such a solution would likely bloat the generated hardware, and a more customized approach is preferable (such as Nexus [9]), but a trade-off between consumed FPGA resources and the added performance must be performed. Alternatively, we could outsource task-management to a soft-core (e.g., a RISC-V [2]) that only orchestrates and resolves dependencies. More importantly, the FPGA allows for customizing communication between threads (and thus tasks), leading to interesting opportunities, particularly for dependent tasks.

One exciting future direction is concerning the synchronization and atomicity of operations. Today, Nymble uses a customized mutex hardware core (that is memory mapped) to support atomicity and synchronization. While this is a correct and functional way of supporting them, there are likely better ways that leverage the customization that FPGAs give us. For example, since we are working with an FPGA, we could, in theory, place the functionality of atom updates inside the external memory controller (DDR4 in our case). Similarly, rather than going through shared memory for synchronization, we could have a system-wide token bus that synchronizes all the threads (by sending and forwarding a synchronization token).

Another opportunity, unique for the FPGA, is concerning the recent memory allocations added in OpenMP. Because the memory hierarchy can be fully customized, we foresee that there are many future opportunities for tuning these for a particular performance criteria (e.g., execution time or power-consumption). For example, we could mark part of the FPGA that would be dedicated to the memory hierarchy as a partially reconfigurable region, and then *dynamically* adapt and optimize the actual hardware in real-time, such as for example

changing cache sizes or replacement policies, scratchpad memories, coherency (or coherency-less) islands of memory, and so on and forth, in order to facilitate high-performance, low-latency producer/consumer patterns in (for example) the OpenMP 4.0 dependent tasks.

The representation of floating-point numbers has recently become a hot topic, with multiple authors proposing (and evaluating) new representations such as Posit [13] and Elias encoding [20]. Today, OpenMP does not contain support for setting a particular region to use a specific representation, but in the future, it might. FPGAs can execute arithmetic operations on these exciting new representations at high speed [27]. If selecting number representation will be part of future OpenMP standard, then FPGAs will be the platform that can exploit it to the fullest.

Finally, scaling OpenMP onto multiple FPGAs is an open question. On hand, we could rely on OpenMP's accelerator directives, and treat each device a discrete system with little to no access to other systems. However, on FPGAs, we can do more, and create/include special hardware to (for example) support a shared-memory view across multiple FPGAs, or use tasks as containers that encapsulate produced/consumed data, that are exchanged among FPGAs.

In short, our understanding of OpenMP on FPGAs is just starting, and there are ample opportunities and future directions where this work affect OpenMP in the future.

6 Related Work

As OpenMP-based programming is very attractive for integrating FPGAs into HPC systems and toolflows, a number of previous works has presented approaches for mapping OpenMP to FPGAs. A good overview of these approaches can be found in the survey by Mayer et al. [21].

Early approaches tried to map OpenMP tasks [4, 24, 25, 11] or worksharing constructs [7, 6, 19], such as `parallel for` to FPGA accelerators. As these approaches date back to the time before the OpenMP target constructs were standardized, no OpenMP constructs for specifying data mapping and device-specific execution were available for these approaches.

More recent approaches combine the OpenMP device constructs with commercially available HLS tools. Many of these works take an approach where target regions are extracted from the input program on AST-level [28, 3], making OpenMP-specific optimizations before HLS difficult. The approach presented by Ceissler et al. [5] even requires the accelerator cores to be implemented in a hardware-description language and uses OpenMP only for the integration into the overall application. Only the work by Knaust et al. [17] uses IR (namely LLVM-IR) to interact with the HLS tool through an undocumented interface. However, as the data-transfers via the OpenCL API are statically generated during compile-time, their approach does not support array sections or mapping of data in only one direction (`to` or `from`), a limitation not found on our approach.

All of the tools mentioned above try to achieve a speedup over sequential execution through spatial parallelism (e.g., a dedicated accelerator core per thread) and classical HLS optimization techniques such as loop pipelining, but none of them supports actual hardware multi-threading inside the accelerator core. In contrast, in [29], OpenMP worksharing loops were mapped to multi-threaded accelerator cores. However, their threading model is much more limited than the one used in this work, as in their model, only a single thread can be active at a time and threads would only be switched when the active thread was suspended due to memory access latency.

As one of the key challenges for an effective mapping of OpenMP constructs to FPGA hardware, Mayer et al. [21] identified the code analysis and optimization across the border between compiler frontend and low-level HLS tool. With our fully integrated compilation flow from input program to Verilog, we are able to propagate information across this border and exploit knowledge of the underlying FPGA execution model for high-level, FPGA-specific transformations on IR-level in the compiler frontend.

7 Conclusion

This work presented a compilation flow for targeting FPGAs with OpenMP device offloading, in combination with a complete integration in `libomptarget` for complete data management support. The presented compile flow supports a significant subset of OpenMP for device offloading, including parallel constructs (e.g., `parallel, teams`) that are mapped to actual hardware threads executing simultaneously in the generated, multi-threaded accelerator, a unique feature of the presented approach.

By optimizing across the border between compiler front-end and the HLS-tool based on LLVM and the academic HLS-compiler Nymble, FPGA-specific optimizations were integrated in the compile flow. This insight could also be interesting for FPGA's vendor and a motivation to further open up their HLS-compiler IR interfaces for OpenMP-based compilation flows.

The case study showed that it is possible to target FPGAs from OpenMP programs, using only standard programming language constructs and annotations, without any HLS-specific extensions, and also showcased an integration of a data preloading functionality that could also be of interest on other accelerator architectures (e.g. GPUs). As described in Section 5, OpenMP is an interesting option for integrating FPGAs into parallel and heterogeneous applications, with a number of interesting research avenues.

References

1. Antão, S.F., Bataev, A., Jacob, A.C., Bercea, G.-T., Eichenberger, A.E., Rokos, G., Martineau, M., Jin, T., Ozen, G., Sura, Z., Chen, T., Sung, H., Bertolli, C., and O'Brien, K.: Offloading Support for OpenMP in Clang and LLVM. In: Third Workshop on the LLVM Compiler Infrastructure in HPC, LLVM-HPC@SC 2016,

Salt Lake City, UT, USA, November 14, 2016, pp. 1–11. IEEE Computer Society (2016). DOI: 10.1109/LLVM-HPC.2016.006. https://doi.org/10.1109/LLVM-HPC.2016.006

2. Asanović, K., and Patterson, D.A.: Instruction Sets Should Be Free: The Case For RISC-V. Tech. rep. UCB/EECS-2014-146, EECS Department, University of California, Berkeley (2014)
3. Bosch, J., Tan, X., Filgueras, A., Vidal, M., Mateu, M., Jiménez-González, D., Álvarez, C., Martorell, X., Ayguadé, E., and Labarta, J.: Application Acceleration on FPGAs with OmpSs@FPGA. In: International Conference on Field-Programmable Technology, FPT 2018, Naha, Okinawa, Japan, December 10-14, 2018, pp. 70–77. IEEE (2018). DOI: 10.1109/FPT.2018.00021. https://doi.org/10.1109/FPT.2018.00021
4. Cabrera, D., Martorell, X., Gaydadjiev, G., Ayguadé, E., and Jiménez-González, D.: OpenMP extensions for FPGA accelerators. In: Najjar, W.A., and Schulte, M.J. (eds.) Proceedings of the 2009 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2009), Samos, Greece, July 20-23, 2009, pp. 17–24. IEEE (2009). DOI: 10.1109/ICSAMOS.2009.5289237. https://doi.org/10.1109/ICSAMOS.2009.5289237
5. Ceissler, C., Nepomuceno, R., Pereira, M.M., and Araujo, G.: Automatic Offloading of Cluster Accelerators. In: 26th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2018, Boulder, CO, USA, April 29 - May 1, 2018, p. 224. IEEE Computer Society (2018). DOI: 10.1109/FCCM.2018.00058. https://doi.org/10.1109/FCCM.2018.00058
6. Choi, J., Brown, S.D., and Anderson, J.H.: From software threads to parallel hardware in high-level synthesis for FPGAs. In: 2013 International Conference on Field-Programmable Technology, FPT 2013, Kyoto, Japan, December 9-11, 2013, pp. 270–277. IEEE (2013). DOI: 10.1109/FPT.2013.6718365. https://doi.org/10.1109/FPT.2013.6718365
7. Cilardo, A., Gallo, L., Mazzeo, A., and Mazzocca, N.: Efficient and scalable OpenMP-based system-level design. In: Macii, E. (ed.) Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013, pp. 988–991. EDA Consortium San Jose, CA, USA / ACM DL (2013). DOI: 10.7873/DATE.2013.206. https://doi.org/10.7873/DATE.2013.206
8. Czajkowski, T.S., Aydonat, U., Denisenko, D., Freeman, J., Kinsner, M., Neto, D., Wong, J., Yiannacouras, P., and Singh, D.P.: From OpenCL to high-performance hardware on FPGAs. In: 22nd international conference on field programmable logic and applications (FPL), pp. 531–534 (2012)
9. Dallou, T., Engelhardt, N., Elhossini, A., and Juurlink, B.: Nexus#: A distributed hardware task manager for task-based programming models. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 1129–1138 (2015)
10. Filgueras, A., Gil, E., Jimenez-Gonzalez, D., Alvarez, C., Martorell, X., Langer, J., Noguera, J., and Vissers, K.: OmpSs@ Zynq all-programmable SoC ecosystem. In: Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays, pp. 137–146 (2014)
11. Filgueras, A., Gil, E., Jiménez-González, D., Álvarez, C., Martorell, X., Langer, J., Noguera, J., and Vissers, K.A.: OmpSs@Zynq all-programmable SoC ecosystem. In: Betz, V., and Constantinides, G.A. (eds.) The 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '14, Monterey, CA, USA - February 26 - 28, 2014, pp. 137–146. ACM (2014). DOI: 10.1145/2554688.2554777. https://doi.org/10.1145/2554688.2554777

12. Gajski, D.D., Dutt, N.D., Wu, A.C., and Lin, S.Y.: High—Level Synthesis: Introduction to Chip and System Design. Springer Science & Business Media (2012)
13. Gustafson, J.L., and Yonemoto, I.T.: Beating floating point at its own game: Posit arithmetic. Supercomputing Frontiers and Innovations 4(2), 71–86 (2017)
14. Huthmann, J., and Koch, A.: Optimized high-level synthesis of SMT multi-threaded hardware accelerators. In: 2015 International Conference on Field Programmable Technology, FPT 2015, Queenstown, New Zealand, December 7-9, 2015, pp. 176–183. IEEE (2015). DOI: 10.1109/FPT.2015.7393145. https://doi.org/10.1109/FPT.2015.7393145
15. Huthmann, J., Liebig, B., Oppermann, J., and Koch, A.: Hardware/software co-compilation with the Nymble system. In: 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Darmstadt, Germany, July 10-12, 2013, pp. 1–8. IEEE (2013). DOI: 10.1109/ReCoSoC.2013.6581538. https://doi.org/10.1109/ReCoSoC.2013.6581538
16. Huthmann, J., Shin, A., Podobas, A., Sano, K., and Takizawa, H.: Scaling Performance for N-Body Stream Computation with a Ring of FPGAs. In: Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, pp. 1–6 (2019)
17. Knaust, M., Mayer, F., and Steinke, T.: OpenMP to FPGA Offloading Prototype Using OpenCL SDK. In: IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019, Rio de Janeiro, Brazil, May 20-24, 2019, pp. 387–390. IEEE (2019). DOI: 10.1109/IPDPSW.2019.00072. https://doi.org/10.1109/IPDPSW.2019.00072
18. Lattner, C., and Adve, V.S.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In: 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO 2004), 20-24 March 2004, San Jose, CA, USA, pp. 75–88. IEEE Computer Society (2004). DOI: 10.1109/CGO.2004.1281665. https://doi.org/10.1109/CGO.2004.1281665
19. Leow, Y.Y., Ng, C.Y., and Wong, W.-F.: Generating hardware from OpenMP programs. In: Constantinides, G.A., Mak, W.-K., Sirisuk, P., and Wiangtong, T. (eds.) 2006 IEEE International Conference on Field Programmable Technology, FPT 2006, Bangkok, Thailand, December 13-15, 2006, pp. 73–80. IEEE (2006). DOI: 10.1109/FPT.2006.270297. https://doi.org/10.1109/FPT.2006.270297
20. Lindstrom, P.: Universal Coding of the Reals using Bisection. In: Proceedings of the Conference for Next Generation Arithmetic 2019, pp. 1–10 (2019)
21. Mayer, F., Knaust, M., and Philippsen, M.: OpenMP on FPGAs - A Survey. In: Fan, X., Supinski, B.R. de, Sinnen, O., and Giacaman, N. (eds.) OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings. LNCS, vol. 11718, pp. 94–108. Springer, Heidelberg (2019). DOI: 10.1007/978-3-030-28596-8_7. https://doi.org/10.1007/978-3-030-28596-8%5C_7
22. Moore, G.E.: Cramming more components onto integrated circuits. Electronics Magazine 38(8) (1965)
23. Pillet, V., Labarta, J., Cortes, T., and Girona, S.: Paraver: A tool to visualize and analyze parallel code. In: Proceedings of WoTUG-18: transputer and occam developments, pp. 17–31 (1995)
24. Podobas, A.: Accelerating Parallel Computations with OpenMP-Driven System-on-Chip Generation for FPGAs. In: IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs, MCSoC 2014, Aizu-Wakamatsu, Japan, September

23-25, 2014, pp. 149–156. IEEE Computer Society (2014). DOI: 10.1109/MCSoC.2014.30. https://doi.org/10.1109/MCSoC.2014.30
25. Podobas, A., and Brorsson, M.: Empowering OpenMP with automatically generated hardware. In: Najjar, W.A., and Gerstlauer, A. (eds.) International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2016, Agios Konstantinos, Samos Island, Greece, July 17-21, 2016, pp. 245–252. IEEE (2016). DOI: 10.1109/SAMOS.2016.7818354. https://doi.org/10.1109/SAMOS.2016.7818354
26. Podobas, A., and Matsuoka, S.: Designing and Accelerating Spiking Neural Networks using OpenCL for FPGAs. In: 2017 International Conference on Field Programmable Technology (ICFPT), pp. 255–258 (2017)
27. Podobas, A., and Matsuoka, S.: Hardware implementation of POSITs and their application in FPGAs. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 138–145 (2018)
28. Sommer, L., Korinth, J., and Koch, A.: OpenMP device offloading to FPGA accelerators. In: 28th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2017, Seattle, WA, USA, July 10-12, 2017, pp. 201–205. IEEE Computer Society (2017). DOI: 10.1109/ASAP.2017.7995280. https://doi.org/10.1109/ASAP.2017.7995280
29. Sommer, L., Oppermann, J., Hofmann, J., and Koch, A.: Synthesis of interleaved multithreaded accelerators from OpenMP loops. In: International Conference on ReConFigurable Computing and FPGAs, ReConFig 2017, Cancun, Mexico, December 4-6, 2017, pp. 1–7. IEEE (2017). DOI: 10.1109/RECONFIG.2017.8279823. https://doi.org/10.1109/RECONFIG.2017.8279823
30. Waldrop, M.M.: The chips are down for Moore's law. Nature News 530(7589), 144 (2016)
31. Whaley, R.C., and Petitet, A.: Minimizing development and maintenance costs in supporting persistently optimized BLAS. Software: Practice and Experience 35(2), 101–121 (2005)
32. Wirbel, L.: Xilinx SDAccel: a unified development environment for tomorrow's data center. The Linley Group Inc (2014)
33. Yang, C., Geng, T., Wang, T., Lin, C., Sheng, J., Sachdeva, V., Sherman, W., and Herbordt, M.: Molecular Dynamics Range-Limited Force Evaluation Optimized for FPGAs. In: 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 263–271 (2019)
34. Zohouri, H.R., Maruyama, N., Smith, A., Matsuda, M., and Matsuoka, S.: Evaluating and optimizing OpenCL kernels for High Performance Computing with FPGAs. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 409–420 (2016)
35. Zohouri, H.R., Podobas, A., and Matsuoka, S.: High-performance high-order stencil computation on FPGAs using opencl. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 123–130 (2018)