Software-Backed Caching and Virtual Addressing for Generated Accelerators in SoC FPGAs

Gerald Hempel, Markus Vogt, Jeronimo Castrillon
Technische Universität Dresden
Faculty of Computer Science
Dresden, Germany
Email: {forename.surname}@tu-dresden.de

Christian Hochberger
Technische Universität Darmstadt
Computer Systems Group
Darmstadt, Germany
Email: hochberger@rs.tu-darmstadt.de

I. Introduction

Hybrid platforms like Xilinx Zynq or Altera Cyclone/Arria V have become increasingly popular. Their power lies in the combination of a powerful RISC processor with a relatively large Field Programmable Gate Array (FPGA), that can be utilized for application specific accelerators. While the FPGA allows to create a custom system to meet application specific needs, the CPU delivers the performance to run complex applications on top of a modern operating system (OS). However, exploiting the performance gain offered by application specific accelerators is a complex design task. For that reason many techniques have been presented in the past that try to automate that process.

In [1] we introduced a GCC extension that generates accelerators from loops in unmodified, arbitrary C code. The accelerators interface with a bare-metal application running on a soft-core CPU while the whole system is implemented on a plain FPGA. Bare-metal systems use direct physical memory addressing, hence, accelerators can access main memory just using the pointers originating from the application code.

Interfacing accelerators in today's hybrid platforms has become more involved, since platforms typically run an OS, e.g. Linux. In such an OS, virtual addressing hinders the usage of pointers for direct memory access from within the accelerators. In this paper we present a novel solution that makes it possible to use virtual addresses together with accelerators. We propose a virtual addressed cache for values to the accelerator. The key idea is to handle cache misses by the processor. This is feasible since the processor is usually idle while the accelerator is running. Additionally, we show how this scheme simplifies accelerator design.

II. Problem Statement

As mentioned above, having a virtual memory space is key for running large applications on hybrid systems featuring a full-blown OS. By automatically migrating portions of application code to a hardware unit, pointers are silently moving from virtual to physical address space and therefore become invalid.

To solve this issue, recent desktop platforms provide an Input/Output MMU (IOMMU), translating addresses during peripheral DMA access. Unfortunately, IOMMUs are not widespread in the embedded domain. They are for example inexistent in today's hybrid processor-FPGA architectures.

In the absence of an IOMMU, a partial solution consists in allocating all application data pinned to physically contiguous memory regions. In this case, a single translation of all addresses passed to the accelerator before invocation suffices. However, this approach is limited, since it misses arbitrary pointers calculated by the accelerator at runtime. Apart from that, allocating all data in contiguous pinned memory contradicts the principle of virtual memory. Such a design would quickly run out of resources.

III. Proposed Implementation

A. IOMMU-Emulation

To overcome the issues mentioned above, one can emulate the behavior of an IOMMU. For this purpose, memory requests from the accelerator must be back delegated to software and transformed into ordinary memory accesses form the application domain. Explicit address translation even becomes obsolete and the fetched data can be returned to the accelerator. This basic attempt must be backed by caching, prefetching mechanisms and fast memory transfer techniques to deliver reasonable performance.

Assuming an accelerator implementing a loop that sequentially accesses memory, simple linear prefetching will already improve things greatly. Admittedly, access patterns are not necessarily that simple, but often follow similar patterns.

The work in progress presented in this paper targets the Xilinx Zynq platform [2], featuring a dual core ARM CPU coupled with an FPGA fabric. Implementing an IOMMU on the FPGA is not feasible, because it is not possible to synchronize with the internal MMU of the ARM subsystem. Although, the initial mappings for the IOMMU could be obtained via software callback, these mappings could be altered by the OS afterwards without notice of the IOMMU. Considering the facts that (i) an application idles while waiting for the accelerator to return and (ii) the CPU operates 5 times faster than the FPGA (1 GHz vs. 200 MHz) it sounds plausible to use the power of the CPU to implement a software-backed IOMMU.

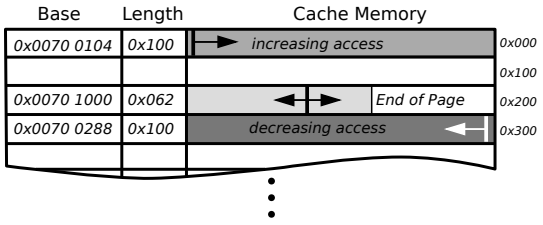

Figure 1. Cache layout optimized for various access scenarios.

B. Cache Hardware Implementation

As described earlier, caching is essential for the IOMMU performance. We propose a local cache in the range of a few kilobytes for each accelerator, which could be implemented in FPGA Block RAM. For a small footprint, the cache is split in a hardware and a software part. The former guarantees fast access in case of a cache hit, while the latter handles cache misses by fetching data from memory. This functionality integrates very well with the emulated IOMMU.

A static code analysis during compile time provides information to optimize caching strategies in order to meet the characteristics of each accelerator. For each pointer, the analysis tries to determine the bounds of the underlying array and whether its address is increasing or decreasing during execution. In case one of those properties could not be determined, we consider the pointer unbounded and/or random.

Our cache is organized in equally-sized lines, while their size and number could be adapted during design time. For each line, a table holds start address and length of the mapped memory region (Figure 1). The latter allows mapping of areas smaller than a cache line. Mappings are not bound to e.g. memory pages and can start at an arbitrary address. For consistency reasons, the cache control avoids overlapping of two distinct cache lines.

C. Cache Software Functionality

The software part is invoked to handle cache misses, cache write back and replacement. For the latter case, the software keeps track of the cache's current mapping. Implementing cache control in software saves hardware resources and enables flexibility while exploring caching strategies in order to minimize cache misses.

On a cache read miss for a_{read}, the chunk of data to be fetched from memory is determined as follows. The predicted memory access order defines whether a_{read} is mapped to the begin, end or middle of the cache line (Figure 1). The latter occurs if the access order is unknown. The size of the memory chunk is determined such that (i) no overlap with another cache line occurs, (ii) the bounds of the underlying array are not exceeded or (iii) no page boundary is crossed. While (i) avoids aliasing issues due to ambiguous mapping, (ii) and (iii) avoid accessing memory outside the applications scope. Rule (iii) only applies in case of unknown bounds. It is a best guess which only guarantees not causing segmentation faults. When no cache line is available, cache replacement is triggered. A proper strategy for handling replacement and write misses still has to be defined. In order to transfer data to and from the accelerator, we use DMA transfers.

IV. Related Work

Nymble [3] and COMRADE [4] are compilers aiming at generating accelerators from annotated C code. Both use the MARC II [5] memory model, providing highly parallel and cache-coherent access for hardware accelerators. To handle virtual memory addresses, all relevant memory regions are placed in a single contiguous DMA buffer. Although feasible, this approach does not scale very well. The authors in [6] present a solution similar to our proposal. However, their memory operates at the granularity level of pages. Our cache will handle arbitrary aligned memory efficiently in smaller chunks and uses static code analysis to optimize the cache implementation.

V. Conclusion and Perspective

We presented a software-backed local cache for arbitrary generated accelerators combined with an IOMMU-emulation technique. Key points of our approach are (i) use of virtual addresses within cache and accelerators, (ii) small cache hardware footprint, (iii) easy exploration of caching strategies and (iv) static memory access analysis during compile time for optimized caching. A future prototype implementation is targeting the Xilinx Zynq platform.

The overall performance of our proposed cache basically depends on the DMA throughput. However, additional latency introduced by software, e.g. kernel operation to initialize DMA transfers, have to be considered. To gain speedup, one could move some of the software functionality to the hardware (trading time vs. space). In the context of having lots of accelerators, this has to be wisely considered.

On the Zynq platform, the AXI interfaces reach up to 1200 MB/s and the DMA controller reaches up to 3500 MB/s transfer rate. Even though we do not have implementation results yet, these numbers look promising to achieve good overall performance.

Acknowledgements

This work is partly supported by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden".

References

[1] G. Hempel, C. Hochberger, and M. Raitza, "Towards gcc-based automatic soft-core customization," in *FPL*, IEEE, Ed., 2012, pp. 687–690.

[2] Xilinx, "Zynq-7000 All Programmable SoC Overview," Product Specification, 2014.

[3] J. Huthmann, B. Liebig, J. Oppermann, and A. Koch, "Hardware/software co-compilation with the nymble system," in *ReCoSoC*, 2013, pp. 1–8.

[4] H. Lange and A. Koch, "An execution model for hardware/software compilation and its system-level realization," in *FPL*, 2007, pp. 285–292.

[5] H. Lange, T. Wink, and A. Koch, "Marc ii: A parametrized speculative multi-ported memory subsystem for reconfigurable computers," in *DATE*, 2011, pp. 1–6.

[6] M. Vuletid, L. Pozzi, and P. Ienne, "Seamless hardware-software integration in reconfigurable computing systems," *Design Test of Computers, IEEE*, vol. 22, no. 2, pp. 102–113, March 2005.