Floating Point Acceleration for Stream Processing Applications in Dynamically Reconfigurable Processors

Karlsruhe Institute of Technology, Chair for Embedded Systems
Lars Bauer, Artjom Grudnitsky, Marvin Damschen, Srinivas Rao Kerekare and Jörg Henkel
{lars.bauer, grudnitsky, marvin.damschen, srinivas.kerekare, henkel}@kit.edu

Invited Special Session Paper at ESTIMedia 2015

Runtime reconfigurable processors provide a large degree of flexibility that allows them to dynamically adapt to different applications and requirements. They couple a standard processor with a runtime reconfigurable fabric (like an embedded FPGA) to offload computationally intensive kernels.

In this paper we present the design and architecture of a flexible accelerator for floating point operations in stream processing applications. To integrate it in an existing reconfigurable processor, the different frequencies between the sequential processor (high frequency) and parallel accelerators (low frequencies) have to be managed. The results show 63.70× and 3.85× better performance-per-area efficiency when using our accelerator and the reconfigurable processor compared to the baseline processor with a soft-float implementation and a high-performance floating point unit, respectively.

I. Introduction

Reconfigurable processors combine the idea of design-time optimized accelerators and runtime flexibility by using a runtime reconfigurable fabric. That makes them suitable for different types of application domains, especially as accelerators for new applications can be developed and integrated even after the actual processor is deployed. Streaming applications are particularly suitable as they combine a limited degree of control flow with a large degree of data-level parallelism, which is beneficial for accelerators. Several streaming applications only perform integer operations (e.g. H.264), but many algorithms also use floating point arithmetic, which is computationally more challenging.

In this paper we investigate the suitability to accelerate floating point arithmetic in an existing reconfigurable processor, the *i*-Core [1], that was designed and implemented for integer accelerators. The goal of the *i*-Core was not to optimize for streaming applications or to support floating point operations, but it was meant to be a general-purpose reconfigurable processor that should provide support for many types of compute-intensive applications. Its runtime system performs dynamic runtime reconfiguration to automatically provide the most suitable accelerators for the executed applications. In this paper we take advantage of the provided flexibility and present the design and architecture of a multi-mode accelerator for floating point operations, with a special focus on the challenges addressed to integrate it in the existing reconfigurable *i*-Core processor.

II. *i*-Core Reconfigurable Processor

The *i*-Core is a reconfigurable processor, capable of loading application-specific accelerators at runtime in order to improve execution speed. Each application can deploy its own accelerators, which provides for a high degree of flexibility, as the accelerators can be developed even after the processor is deployed and in use. The *i*-Core architecture (shown in Figure 1) is based on a general-purpose RISC core pipeline that is tightly coupled with a *reconfigurable fabric*, i.e. a special area that contains runtime reconfigurable regions (embedded FPGAs) to house the accelerators. The fabric also provides memory ports to access the memory hierarchy and an interconnect

978-1-4673-8164-2/15/$31.00 © 2015 IEEE

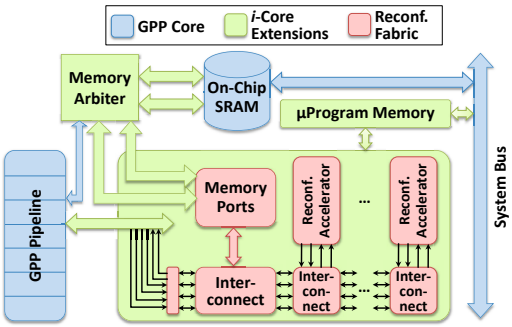

Fig. 1: Architecture of the *i*-Core reconfigurable processor

for communication between the accelerators. In contrast to simpler reconfigurable architectures that only support a single reconfigurable accelerator at a time, using a multi-accelerator fabric supports far more dynamic use scenarios: (i) demanding computational kernels can be fully offloaded onto the fabric by combining multiple accelerators, (ii) in multi-tasking scenarios task switching is very fast, as the fabric can hold accelerators for multiple tasks [2], (iii) the fabric can be shared between multiple cores, which extends the benefits of reconfigurable accelerators to the whole system [3].

In order to support such a degree of flexibility, the fabric interconnect needs to be highly configurable, e.g. allowing one accelerator to retrieve input data via a memory port in cycle i, processing it in cycle i+1, and forwarding the results to a second accelerator in cycle i+2. The *i*-Core supports this via μOps that control the configuration of the fabric interconnect and memory ports. An application therefore uses a μProgram (i.e. sequence of μOps) along with the accelerators for the kernels that should be run on the fabric.

The fabric interconnect can result in long combinatorial paths, thus the fabric frequency is lower than that of the GPP core. However, lowering the frequency of the fabric would also lower the frequency of the accelerators, even though especially integer accelerators can have much shorter critical paths. Instead, the reconfigurable fabric is operated at the frequency of the GPP core, but each μOp is a multicycle operation with a configurable duration of n cycles. This allows an accelerator to internally operate at a higher frequency than the fabric interconnect, while at the same time simplifying the connectivity to the GPP core.

III. Design of the Floating Point Accelerator

Scale-invariant feature transform (SIFT) feature matching is part of object recognition applications and involves thousands of floating point subtractions and multiplications. The complex datapath required for its floating point arithmetic and its memory bound characteristic make SIFT a good candidate for acceleration on the *i*-Core fabric and we use it as an example to develop a flexible accelerator that supports multiple operation modes.

To accelerate SIFT feature matching on the *i*-Core, the reconfigurable FMAV (floating point multiply-accumulate-vector) accelerator was designed. Figure 2 shows its block level architecture. The interface to the accelerator is compliant to the fabric interconnect of the *i*-Core and comprises of two 32 bit inputs, one 32 bit output, 6 bit control inputs and the clock signal. The FMAV accelerator performs

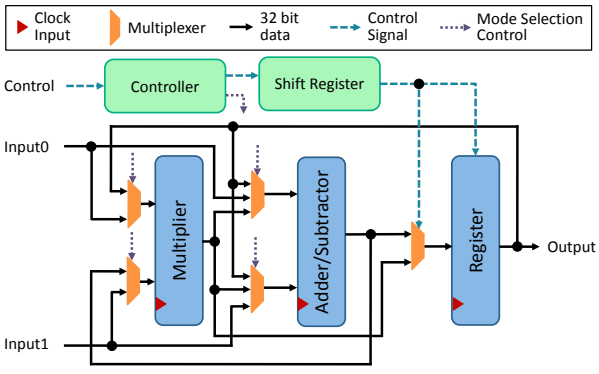

Fig. 2: FMAV Architecture

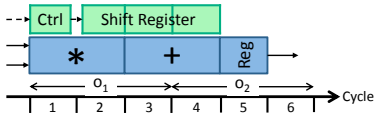

Fig. 3: FMAV Schedule of a MAC operation

single precision floating point operations and supports 12 operation modes. This allows flexible combinations of multiplication, addition and subtraction with optional storing of the result in a register.

Due to the long critical path of some FMAV modes (e.g. multiply→add→register), the accelerator's latency is nearly two times longer than the existing integer accelerators. Rather than configuring the μOps to last twice as long in case they use FMAV, we decided to internally pipeline FMAV, i.e. each FMAV operation lasts for 2 μOps, where the first μOp starts it and the second μOp may start another operation, while the first FMAV finishes at the same time. In our current prototype, each μOp lasts for 3 clock cycles and thus the latency of one FMAV operation is 6 clock cycles, while a new input can be provided every 3 clock cycles.

Depending on the FMAV mode, input data may come either from the interconnect or the internal register (controlled by the left four MUXes in Figure 3). Results from the multiplier or adder can optionally be stored in the register to serve as an input for the next operation. A typical schedule of a MAC operation is shown in Figure 3. At cycle 1, μOp O_1 enters the first pipeline stage of the FMAV accelerator. The FP multiply and add operations actually just last for 2 cycles each, however, as the μOp duration is 3 cycles (due to the other accelerators and the interconnects), we have to control the execution accordingly. As it is a MAC operation, the controller sets the data path of multiplier, adder, and register accordingly. From the latencies of the individual operations it is clear that the output of the adder will not be available during μOp O_1, but just during the subsequent μOp O_2, which enters the pipeline at cycle 4. To avoid having O_2 being responsible to store the result of O_1 (which would complicate μOp programming), we use a Shift Register which delays the control of O_1 until cycle 5, where the output of the Adder of μOp O_1 is stored in the Register.

IV. Results

The *i*-Core and our proposed floating point accelerator are integrated into a state-of-the-art many-core system [4] and were evaluated on its Virtex-6 prototyping platform. Table I shows the area results of our accelerator. Each reconfigurable region covers 5×20 CLBs, rather than the full 40 CLBs of a configuration frame, to get a better aspect ratio and routability of the reconfigurable region at the cost of bigger partial bitstreams. It takes 0.35 ms to reconfigure an FMAV accelerator at an average bandwidth of 200 MB/s.

To evaluate the performance, we use the FMAV accelerator for

Resource Type	Available	Required	Utilization [%]
LUT	800	562	71
DSP48E1	8	5	63

TABLE I: Lookup tables and DSP-blocks (multipliers) available in a reconfigurable region and required to implement our floating point accelerator

SIFT feature matching and matrix multiplication. Table II shows the execution times for the kernels using the *i*-Core with our accelerator in comparison to using the LEON3 High Performance Floating Point Unit (FPU-HP), the non-pipelined FPU-Lite, and a version compiled with the soft-float library.

	LEON3 soft-float	LEON3 FPU-Lite	LEON3 FPU-HP	*i*-Core
Matrix Multiplication	370.194	14.473	8.454	1.154
SIFT Comparison	1.172	0.113	0.050	0.009

TABLE II: Execution times in ms on different LEON3 configurations and the *i*-Core

	LEON3	LEON3 FPU-Lite	LEON3 FPU-HP	*i*-Core
LUT	5,420	8,152	14,357	27,296
Block-RAM (36 kbit)	13	13	13	40
DSP	4	4	20	60

TABLE III: Resources required for different LEON3 configurations and the *i*-Core

The area overhead of the *i*-Core and the two LEON3s with hardware FPUs compared to a LEON3 without FPU (soft-float) is shown in Table III. The *i*-Core is based on the LEON3 and includes all *i*-Core extensions (μProgram Memory, Memory Arbiter, and bistream loader) and the reconfigurable fabric (including 5 Reconfigurable Accelerators with DSPs and the interconnect) as shown in Figure 1.

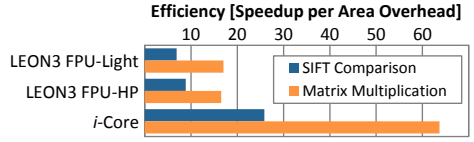

Fig. 4: Area efficiency of different LEON3 configurations and the *i*-Core

Figure 4 shows the area efficiency of the *i*-Core and the LEON3s with hardware floating point, i.e. it puts the speedup (relative to LEON3 soft-float) into perspective with the area overhead (relative to LEON). The area efficiency of the *i*-Core for matrix multiplication is 3.85× better than the LEON3 FPU-HP and 63.70× better than the LEON3. These application-specific optimizations are possible due to the large flexibility that is offered by the reconfigurable fabric. In addition to the presented floating point accelerator, the *i*-Core can also be reconfigured at runtime to support a variety of integer accelerators (e.g. for cryptography, audio/video/image processing) without the need to modify the *i*-Core itself [1–3].

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Invasive Computing" (SFB/TR 89). We thank Eduard Maier for his contribution on the accelerator implementation.

References

[1] J. Henkel, L. Bauer, M. Hübner, and A. Grudnitsky, "i-Core: A run-time adaptive processor for embedded multi-core systems", in *Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2011.

[2] L. Bauer, A. Grudnitsky, M. Shafique, and J. Henkel, "PATS: a performance aware task scheduler for runtime reconfigurable processors", in *Field-Program. Custom Comp. Machines (FCCM)*, 2012, pp. 208–215.

[3] A. Grudnitsky, L. Bauer, and J. Henkel, "COREFAB: Concurrent reconfigurable fabric utilization in heterogeneous multi-core systems", in *Int'l Conf. on Compilers, Arch., and Synthesis for Emb. Sys. (CASES)*, 2014.

[4] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and K. S., "Invasive manycore architectures", in *16th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 193–200.