Microprocessors and Microsystems 51 (2017) 275–288

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Combining hardware and software codecs to enhance data channels in FPGA streaming systems

Marlon Wijeyasinghe*, David Thomas

Department of Electrical and Electronic Engineering, Imperial College London, United Kingdom

ARTICLE INFO

Article history:
Received 30 July 2016
Revised 3 May 2017
Accepted 6 May 2017
Available online 8 May 2017

Keywords:
FPGA
Codec
Compression
Streaming
Multithreading

ABSTRACT

This paper discusses a framework to apply compute-intensive data transforms (codecs) to data flowing through a channel between a CPU and an FPGA kernel on a heterogeneous streaming system. Codecs such as parallel compression are applied to data across a PCI-express channel in real-time, aiming to increase the effective bandwidth by using spare resources in both software and hardware. Other codecs such as encryption can be applied while minimising the loss in performance. Real-time encoding/decoding is performed by breaking up the data stream into segments for multiple threads to process different segments. This allows the interleaving of encoding, transmission and decoding on the CPU, hence the framework functions as a software pipeline. The capabilities of the framework are demonstrated using 4 compression codecs and an encryption codec on a Maxeler system. For example, 1.51x and 1.85x speed ups are observed respectively when delta compression and double to single precision floating point conversion are applied to the data being transmitted to/from an identity kernel on the FPGA. Similarly, a 1.61x speed up is observed when double-to-single precision floating point conversion is applied to data being transmitted to/from an FPGA kernel performing a Fast Fourier Transform.

1. Introduction

State-of-the-art FPGA streaming systems are ideal for high-performance computing (HPC) applications as they use pipeline parallelism and replicated pipelines to achieve high speed computation. Such applications have traditionally been built at the Register Transfer Level (RTL), but the time this takes is increasing due to the complexity of developing pipelines and coordinating dataflow. HPC programmers (kernel designers) now use high-level synthesis (HLS) tools to handle low-level aspects such as instantiating and re-timing pipelines in hardware kernels.

HLS tools have drawbacks despite their ability to simplify the design process – the kernel designer is restricted to using language/library primitives provided by the tool to describe logic, arithmetic and stream control. Hence they must revert to an RTL description to implement features that are not included in the streaming services provided by the tool.

A major challenge that streaming systems face is limited data bandwidth to/from the chip. FPGA kernels are capable of high-speed computation, but if the channel supplying data to the kernel is incapable of providing data at the required speed, it is not possible to make full use of the kernel's computational capabilities. One way to mitigate the bandwidth issue in an existing high-level streaming system is for the HPC programmer to design compression at RTL or include pseudo-RTL code within the high-level code. This adds complexity to the design process in both software and hardware thereby reducing the benefits of working with HLS tools.

This paper proposes a framework to add compute-intensive real-time codecs (e.g. compression to improve effective bandwidth) by using spare CPU cycles on the software side and spare FPGA resources on the hardware side of the channel in heterogeneous streaming applications. Software codecs must be multi-threaded and hardware codecs must be pipelined to operate at line rate (channel data rate) to avoid reducing bandwidth in the system. The discussion in this paper builds on a paper that was presented at the International Symposium on Applied Reconfigurable Computing 2016 [1], by applying the framework to a Fast Fourier Transform (FFT) application and testing the framework with encryption, a non-compression codec.

Hence, the contributions of this paper are:

- A multi-threaded framework to apply compute-intensive codecs to a segmented data stream by using spare resources in software and hardware for high-bandwidth applications.

* Corresponding author.
E-mail addresses: marlon.wijeyasinghe09@imperial.ac.uk (M. Wijeyasinghe), d.thomas1@imperial.ac.uk (D. Thomas).

http://dx.doi.org/10.1016/j.micpro.2017.05.003
0141-9331/

- A concrete implementation of the framework in a commercial platform using C code on the CPU side and OpenSPL code on the FPGA side.
- A model for predicting the end-to-end transmission latency when the framework is applied to a streaming system.
- Evaluation of data transmission performance across a PCI-express channel on a heterogeneous system when a range of different codecs such as compression and encryption are applied.
- A comparison of measured values with model-predicted performance for different codecs.

Section 2, the background section discusses related work in this field. Section 3 provides a detailed description of the framework proposed by this paper. Section 4 describes a mathematical model to predict the performance of streaming systems operating under this framework. Section 5 evaluates the performance of the framework on a commercial platform when different codecs were applied to streaming systems. Section 6 summarises the discussion in the paper.

2. Background

FPGA streaming systems accelerate computationally intensive algorithms by exploiting their parallelism and/or using replicated pipelines. This is done by transmitting data along a pipelined kernel at a high throughput. Previous work on streaming applications have demonstrated that FPGAs can outperform multi-core CPUs in terms of speed and energy efficiency. Examples are the Lattice Boltzmann Method (LBM) [2], Discrete Cosine Transform (DCT) [3] and Finite Impulse Response filters [3]. Such implementations are suitable where large amounts of data are transmitted such that the total transmission time is much larger than the pipeline latency; thereby making the implementation tolerant to latency.

The bandwidth problem arises due to limitations in the capability of data channels in high-bandwidth applications. The I/O bandwidth of the chip is restricted by the number of I/O pins and the clock frequency of their corresponding registers. The physical characteristics of the off-chip data channel(s) which are used to access the FPGA can also limit the bandwidth with which the FPGA can be accessed externally. For example, this work uses the Maxeler [4] heterogeneous CPU-FPGA platform where the CPU transmits data to and from the FPGA via PCI-express 2.0 x8. The PCI-express channel has a theoretical bandwidth of 4GB/s in either direction (total 8GB/s), but the Maxeler card itself only has a bandwidth of up to 2GB/s in either direction (total 4GB/s). When the duplex channel bandwidth was measured under the low-latency interface (discussed later), it was observed to be 2509MB/s. Assuming simultaneous transmission in both directions, if a pipelined computation kernel on the FPGA has a 128-bit wide data bus, the channel bandwidth is sufficient to support the kernel being clocked at up to 134 MHz. A kernel with a higher clock frequency will not run at full capacity since the channel supplying the kernel is a bottleneck.

To tackle the bandwidth bottleneck on streaming systems, several different compression methods have previously been implemented on FPGAs. For example, prediction-based floating point hardware compression and decompression have been design to be placed at either end of an LBM streaming kernel to improve effective memory bandwidth; one variation achieved a compression ratio of around 3.8 [5] and another variation, which used integer-based prediction achieved a compression ratio of around 3.5 [6]. Similarly, sparse matrix vector multiplication, a memory-bound algorithm was accelerated by compressing redundant non-zero data using spare FPGA resources and achieved average compression ratios of 1.14–1.40 and a maximum ratio of 2.65 [7]. The adaptive JPEG-LS algorithm has also been implemented previously achieving a speed of 75 megapixels per second [8]. These compression methods are specific to the data type being compressed and optimised for the specific algorithm for which they were designed. Additionally, the computation kernel was not modified in these two cases.

The framework that this paper presents works with a good degree of transparency, under the same principle of not modifying the kernel. However its emphasis is broader than purely accelerating bandwidth-bottlenecked HPC applications by designing state-of-the-art compression algorithms – but rather to use compression codecs to demonstrate the capability of the proposed framework. The framework aims to enhance communication between software-hardware boundaries.

Some general purpose compression algorithms have also been implemented on FPGAs previously. A streaming implementation of the general purpose ZLIB algorithm [9] was implemented on FPGA and it operates at 125MB/s. The LZMA algorithm [10] operates at up to 16MB/s. GZIP [11] attained a compression bandwidth of 110MB/s and a decompression bandwidth of 306MB/s. A particular implementation of LZW [12] achieved a compression bandwidth of 88MB/s and a decompression bandwidth of 160MB/s. An implementation of a processor for LZ-based algorithms had a compression speed of 16MB/s [13]. While it is important to note that these implementations were evaluated on older technology (therefore they will be faster on newer technology), all 3 are at least an order of magnitude slower than our measured channel bandwidth (2509MB/s). For real-time compression in high-bandwidth streaming systems, a fast compression speed is the most important factor rather than the compression ratio. These existing implementations are not able to meet the real-time requirements of state-of-the-art FPGA streaming systems.

There have been attempts at compression targeted at high-bandwidth FPGA applications. The Titan-R [14] hardware compressor core was shown to perform the LZ277 algorithm at 1096MB/s. Two more implementations of the LZ277 algorithm for multigigabit networks was shown to have a throughput of 1113MB/s and 1173MB/s [15]. The X-MatchPRO lossless data compressor has an internal throughput of 100MB/s and a duplex compression and decompression performance of 200MB/s [16,17]. An improved version of this, namely X-MatchPROv4 has an internal throughput of 200MB/s and a duplex compression and decompression performance of 400MB/s [18]. The LZSS algorithm (based on LZ277) was found to operate at 52MB/s [19]. An implementation of the 842B algorithm [20] (with two compression and two decompression engines) was measured to have a duplex compression and decompression throughput of 2723MB/s. The latter algorithm has duplex bandwidth that exceeds the bandwidth of the channel and is relatively small in terms of hardware resource usage. If a sufficiently fast software version of the algorithm is also available, then this is the only existing algorithm that would work across a software-hardware channel on a streaming system. The others do not have any corresponding software implementations.

Other proposals of general data channel enhancements have been made, using various approaches. For example, [21] initially proposed the idea of transparently adding capabilities such as compression and error correction to data channels without modifying the kernel.

Similiar platforms have been developed in other work, albeit to perform different functions. Such a platform is [22], which is a framework for simplifying communication between FPGAs and GPUs via PCIe on heterogeneous hardware accelerators which use both GPUs and FPGAs. The FPGA-GPU framework enables easy data transfer and coordination between a GPU and FPGA. In [24], the authors describe how a high-level streaming language called Brook can simplify the design of HW accelerators and achieve good performance including features such as kernel replication. In [25], the