Impulse C vs. VHDL for Accelerating Tomographic Reconstruction

Jimmy Xu[1], Nikhil Subramanian[1], Adam Alessio[2], Scott Hauck[1]
[1] Department of Electrical Engineering, [2] Department of Radiology, University of Washington,
Seattle, WA. USA
{jimmyxu, niksubr, aalessio, hauck}@u.washington.edu

Abstract

While traditional methods of designing FPGA applications have relied on schematics or HDL, much interest has been shown in C-to-FPGA tool flows that allow users to design FPGA hardware in C. We evaluate a C-to-FPGA tool flow (Impulse C) by analyzing the performance of three independent implementations of the Computed tomography (CT) filtered backprojection (FBP) algorithm developed using C, Impulse C, and VHDL respectively. In the process, we compare the design process of Impulse C versus HDL, and discuss the benefits and challenges of using Impulse C. In addition, we explore the benefits of tightly-coupled FPGA acceleration offered by the XtremeData XD1000. The results demonstrate that Impulse C designs can achieve over 61x improvement over multi-threaded software, and similar performance as VHDL, while significantly reducing the design effort, and that tightly-coupled FPGA coprocessors like the XD1000 effectively overcomes the traditional communication bottleneck between CPU and FPGA.

1. Introduction

Tomography (CT) is a medical imaging technique used to create cross-sectional images from x-ray transmission data acquired by a scanner. In traditional tomographic systems, the primary computational demand after data capture by the scanner is the backprojection of the acquired data to reconstruct the scanned object. Backprojection can be viewed as the mapping of raw scanner data into the image space. In a typical CT system, the data has ~106 entries per cross-section, and the process of tracing each datum through the image space is computationally demanding. As a result, hardware acceleration of this process has been the focus of several studies [1].

Previous works often used FPGAs to implement such hardware accelerators, and have achieved dramatic improvements over microprocessor based software systems. However, the traditional method of designing FPGA based accelerators, using either schematics or HDL to describe the hardware, require the skill set and expertise of a hardware engineer. A promising method of reducing this barrier to entry is a C-to-FPGA tool flow that allows users to describe the hardware computations in C and automatically generates the hardware description for synthesis.

The general perception has been that while such high level tool flows are simple to use, they do not provide the same level of performance as hand coded HDL. We investigate this notion by considering a C-to-FPGA tool flow called Impulse C [2]. Our benchmark was an implementation of the filtered backprojection (FBP) algorithm running on a microprocessor. We partitioned the design such that the compute intensive backprojection step was run on an FPGA while the user interface and some filtering operations were performed on the host processor. The XD1000 development system was chosen for the implementation because it offers a tightly coupled CPU-FPGA platform, reducing communication latency that can overshadow the benefits of using hardware acceleration.

Overall, this paper provides a case study of the performance of Impulse C in the design and implementation of a typical application that can benefit from hardware acceleration. We hope to provide an accurate representation of the benefits and shortcomings of using Impulse C to replace traditional development methods for FPGAs.

2. Background

2.1. Tomographic Reconstruction

Tomographic reconstruction is the process of generating cross-sections of an object from a series of measurements acquired from many different directions. In Computed Tomography (CT scans), the measurements are x-ray attenuation data. The set of all projections acquired by a scanner is stored as a two dimensional matrix, with the dimensions being the angle at which the projection was acquired, and the position of the detector that recorded the x-ray. This data represents a 2-D spatial function termed a sinogram. The problem of reconstruction can be viewed as the problem of transforming the sinogram (θ,S) into the image space (X,Y).

2.2. XD1000 Coprocessor Accelerator

Frequently occurring or complex computations can overwhelm the primary microprocessor (host or CPU) in a computer system. A solution to this is utilizing coprocessors to shoulder some of the burden, allowing the CPU to offload these computations. The host processor would retain functions such as interfacing with the user or peripherals, and the coprocessor would be tasked with processing the raw data.

Companies such as DRC Computer and XtremeData take an approach with FPGA coprocessors that use a very tightly-coupled FPGA to CPU interface, with the FPGA coprocessor module situated in one of the CPU sockets in an AMD Opteron motherboard. This allows the FPGA module to communicate with the CPU via the HyperTransport bus, which has a bandwidth of 1.6 GB/s. This remedies the bandwidth concern with accelerating applications using FPGA coprocessors.

2.3. Impulse C

While FPGAs have traditionally been programmed with either schematics or HDL describing the hardware, interest has been growing in the use of C-to-FPGA tool flows. One example is Impulse C, which promises to reduce design and programming effort.

The Impulse C tools include the CoDeveloper C-to-FPGA tools and the CoDeveloper Platform Support Packages (PSPs). PSPs add platform-specific capabilities to the Impulse CoDeveloper programming tools. With the PSP, users can partition the application between the host and coprocessors, make use of other on board resources, and automatically generate the required interfaces. The interfaces are implemented as a wrapper function around the user logic. We used the Impulse C tools with the XtremeData XD1000 PSP.

3. Design

3.1. System Parameters

We chose to first implement 512x512 (θxS) backprojectors independently in VHDL and Impulse C, followed by 1024x1024 versions using each approach. The algorithm discussions in this section reflect the 1024x1024 implementations. Performance results for both the 512x512 and 1024x1024 implementations can be found in the results section.

3.2. Backprojection

The CT scanning process can be modeled as detectors acquiring line integrals of the attenuation coefficient along the scanned object. When reconstructing, we start with an image initialized to 0. Corresponding to each entry of the sinogram we identify all the pixels that lie on the ray. The values of those image pixels are then increased by the corresponding ray-value. When this process is performed for all entries in the sinogram, we have reconstructed the image. This is ray-by-ray reconstruction. Another way to approach this is to reconstruct one pixel at a time. Given a pixel and an angle of measurement, the ray-value that contributes to the pixel needs to be identified. If we go through all angles, identifying all rays that the pixel was on, and sum those values, that pixel is fully reconstructed. We can then go ahead and reconstruct all other pixels in the same way. This is pixel-by-pixel reconstruction.

Since both pixel-by-pixel and ray-by-ray reconstruction methods require the compute loop to touch every angle of the sinogram for every point on the image (an O(n3) operation), the main difference between these two methods is the outer loop of the processing algorithm [4]. The pixel-by-pixel approach places the x-y loops outside of the angle loop, with the opposite being true for the ray-by-ray method. Both of these methods can be made parallel by introducing blocks into the image and sinogram respectively. For the pixel-by-pixel method, each column of the image can be assigned to a separate processing engine. An example is a system with four 128x512 blocks, which will require 128 processing engines. The ray-by-ray method can be made parallel by partitioning the sinogram into blocks of 128x1024 (θxS) with each angle of the sinogram block assigned to a separate processing engine.

These two methods produce identical reconstructed images, and primarily differ in their resource requirements [4]. However, since the hand-coded VHDL and Impulse C versions of the backprojector were designed independently of each other, a split occurred at this point. Initial analysis indicated that the pixel-by-pixel algorithm appeared more compatible with the hardware hierarchical design abilities of HDL, and analysis on the ray-by-ray algorithm [3] indicated that it was compatible with the programming model of Impulse C.

4. Implementation

4.1. Impulse C Backprojector

The sequence of operations can be found in [3].

Initially, we experienced difficulty getting the Impulse C design to meet timing when the onboard accumulation algorithm was used. This forced us to create a version of the backprojector where partially created images would be sent back to the CPU, and later accumulated in software. While this is a fast computation, we still wanted to avoid the additional data transfers this would require. After analysis, it was determined that our timing issues originated from the declaration of an imgRAM that would span 8 MRAMs on the FPGA. This forced the Quartus tools to connect all 8 MRAMs together, which resulted in timing issues since the MRAMs are spread across the FPGA. A fix for this was found by manually declaring separate arrays for each MRAM, and introducing separate pipelined logic to tie them together.

Another issue we ran into during our implementation of the above system was the pipeline throughput generated by Impulse C. Since the XD1000 platform support package for Impulse C imposed a fixed 100 MHz clock on the design, we aim to have the pipeline accept inputs and produce results every cycle. However, the pipeline could only achieve a rate of 2 cycles per input/output due to operations within the compute loop requiring two values from the same memory. This can be resolved with the use of true-dual port memories; however, efforts to implement true-dual port memories in Impulse C resulted in the design failing to meet timing. Improvements to the PSP could likely fix this.

4.2. VHDL Backprojector

The VHDL Backprojector shares many of the same operation steps as the Impulse C backprojector. Details of the differences between the two backprojector implementations can be found in [4].

After the 512x512 VHDL backprojector was complete, several observations regarding the performance of the pixel-by-pixel and ray-by-ray methods of reconstruction were made. First, the pixel-by-pixel method would not scale as effectively to 1024x1024 due to resource utilization. Since ray-by-ray can use the MRAM to store the image data, much more of the M4K and M512 RAMs were available for the sinogram data. We realized that if we use pixel-by-pixel to construct a 1024x1024 projector, we would need to decrease the number of processing engines to 64. Second, we realized that the transfer time saved by overlapping communication with computation is not significant compared to the runtime of the compute loop. We decided the benefits of pixel-by-pixel did not outweigh the cost for the 1024x1024 projector, so we adopted ray-by-ray (similar to the Impulse C design) for the 1024x1024 version.

4.3. Software

The software benchmark performance numbers were obtained using two quad-core Intel Xeon L5320 processors with operating frequencies of 1.86 GHz. The base results represent a serial execution, and the multi-threaded result is obtained using OpenMP and 8 threads distributed across each of the eight available CPU cores.

5. Results

5.1. Performance Comparisons

Table I shows the VHDL implementation and Impulse C implementation, compared to the software benchmarks.

TABLE I. EXECUTION TIMES (SECONDS)

Design	512 Project	1024 Project
Base Software	5.17	20.93
Multi-threaded Software	1.06	3.95
Impulse C	0.03183	0.06452
VHDL	0.02168	0.03802

The major reasons for the FPGA implementations being significantly faster than the software benchmark are pipelining and loop unrolling. These two techniques allowed us to process 128 elements in parallel, resulting in a much faster implementation than the software versions.

The tight coupling of the FPGA and the CPU in the XD1000 also played a pivotal role in the results we achieved. Due to the high bandwidth and low latency between the CPU and the FPGA, transferring data to the FPGA did not prove to be a bottleneck for this application (Table II), and we were able to focus the resources and effort to optimize other areas of the system.

The compute pipeline in the HDL version produces a result every cycle, whereas the Impulse C version produces a result only once every two cycles. This makes the hand-coded compute pipeline 2x faster. Furthermore, in the hand-coded version the final image is streamed directly to the CPU instead of using the SRAM as an intermediate, making that process 30%

faster. Lastly, the hand-coded version has a custom SRAM controller that achieves 2x faster data access to the onboard SRAM than the Impulse C version. However, the Impulse C library for loading data to the SRAM was significantly faster than our VHDL version. Additional details can be found in [3, 4].

5.2. Design Effort

To quantify the ease of use of Impulse C compared to VHDL, we compare the design time and lines of code in the 512x512 projector designs. Both the Impulse C and VHDL designs were created by designers with similar hardware engineering background and FPGA design experience. Creating the initial version of the design using Impulse C took 25% less time than the HDL version. This includes the time it took us to get acquainted to the Impulse C tools, and understand the tool flow and design methodology. This represents the time it takes an experienced hardware designer to learn and use Impulse C, vs. implementing in an existing flow. To add an additional data point to this analysis, we created a 1024x1024 version of the VHDL pixel-by-pixel backprojector, and compared the implementation time to Impulse C. The incremental time taken to design, test and debug the 1024 version of the design using Impulse C was much less when compared to HDL.

TABLE II. COMPARISON OF DESIGN TIME

Design Version	VHDL	Impuse C
512 Projections	12 weeks	9 weeks
Time to extend design to support 1024 projections	1 week	1 day

6. Discussion

6.1. Strengths and Weaknesses of Impulse C

Impulse C is an effective design methodology for targeting a hardware platform. The ability to create applications entirely in C, but have them easily partitioned across the CPU and FPGA, is very attractive. Also, the ability to perform functional verification on the complete design is greatly enhanced. If one were creating the FPGA design separately by writing HDL and C to run on the host, the design verification is much more complex. Further, because the design of the software and hardware is so tightly coupled the eventual implementation is seamless. With traditional methods it takes extra effort to integrate the software and hardware execution stages.

One of the drawbacks of Impulse C is the loss of fine grained control over the resulting hardware. In certain situations we might want to make simple modifications like adding registers to the input and output of a computation. For example, we discovered that writing to the image cache was a step that failed timing. A simple work-around we wanted to implement was to postpone the write to the next clock cycle by adding a register to the input and output of the cache. These sorts of fine grained changes are not easily communicated to the compiler. An upshot of the above drawback is that it is extremely difficult to efficiently implement control logic in the pipeline.

6.2. XD1000 Bandwidth

We found the tightly-coupled FPGA coprocessor offered by the XD1000 is a significant advantage. Our results showed that, contrary to commonly held beliefs, the time spent transferring data to and from the FPGA was not the bottleneck of our application. Instead, data transfer occupied a small fraction of the total execution time. This suggests that as technology moves towards higher bandwidth and lower latency in FPGA coprocessor implementations, designers can instead focus on utilizing available hardware resources to produce the most efficient computation structure. There will always be applications where the communication to computation ratio rules out FPGA coprocessor implementations, but tightly coupled FPGA-CPU systems allows many more applications to benefit from these systems.

7. Acknowledgements

This project was supported by the Washington State Technology Center and Impulse Accelerated Technologies.

8. References

[1] M. Leeser, S. Coric, E. Miller, H. Yu, and M. Trepanier, “Parallel–beam backprojection: An FPGA implementation optimized for medicalimaging,” Int. Symposium on FPGAs, pp. 217–226, 2002.

[2] http://www.impulseaccelerated.com/ , Impulse Accelerated Tech-nologies.

[3] N.Subramanian, A C-to-FPGA Solution for Accelerating Tomographic Reconstruction, M.S. thesis, University of Washington, Washington, USA, June 2009.

[4] J. Xu, An FPGA Hardware Solution for Accelerating Tomographic Reconstruction, M.S. thesis, University of Washington, USA, June 2009