Accelerating Statistical LOR Estimation for a High-Resolution PET Scanner using FPGA Devices and a High Level Synthesis Tool

Zhong-Ho Chen and Alvin W.Y. Su
Department of CSIE
National Cheng-Kung University
Tainan, Taiwan
zhonghochen@gmail.com

Ming-Ting Sun and Scott Hauck
Department of Electrical Engineering
University of Washington
Seattle, WA, USA
hauck@u.washington.edu

***Abstract*—In this paper, we use an FPGA platform and a high level synthesis tool, called Impulse C, to speedup a statistical Line Of Reaction (LOR) estimation for a high-resolution Positron Emission Tomography (PET) scanner. The estimation algorithm provides a significant improvement over conventional methods, but the execution time is too long to be practical for clinic applications. Impulse C allows us to rapidly map a C program into a platform with a host processor coupled to an FPGA device. However, the generated HDLs from the original codes are very inefficient, and the execution time is even worse than the software code. We describe some optimization methods for the algorithm using Impulse C. These methods could also be applied to other applications or used to improve the high level synthesis tools. The results show that the FPGA implementation can obtain a 82x speedup over the optimized software.**

Keywords-component; FPGA; High Level Synthesis; LOR estimation; PET scanner;

I. Introduction

Positron Emission Tomography (PET) is a nuclear medical imaging technique which produces a three-dimensional scan of the body. This technology can be used in hospitals to diagnose patients and in laboratories to image small animals as well. In the beginning of the process, radiotracers are injected into bodies and absorbed by specific tissues. A radiotracer is a molecule labeled with a positron emitter, such as ^{11}C, ^{13}N, ^{18}F or ^{15}O. In the nucleus of a radiotracer, a proton decays and a positron is emitted. After traveling a short distance, the emitted positron annihilates with an electron and produces two 511 keV photons which travel in essentially opposite directions.

In order to determine the distribution of radiotracers, a patient is placed between gamma detectors. A gamma detector is composed of crystals that interact with photons. When a pair of photons is detected by the detectors within a short timing window, the system registers the event as a coincidence. The line joining two relevant detectors is called a line of response (LOR). However, it is difficult to estimate the exact depth of interactions with the crystals. Moreover, a single photon may interact with multiple crystals, which makes it harder to estimate the true LOR (Figure 1). Conventional PET scanners employ versions of Anger logic to position events [1][2]. Anger logic positions interactions by calculating the energy-weighted centroid of the measured signals. The depth of interaction is assigned a constant value, which is based on the attenuation coefficient of the detector crystals. This causes the well known parallax error [2], shown in figure 1. Therefore, it is important for a high resolution PET scanner to estimate the depth and order of interactions.

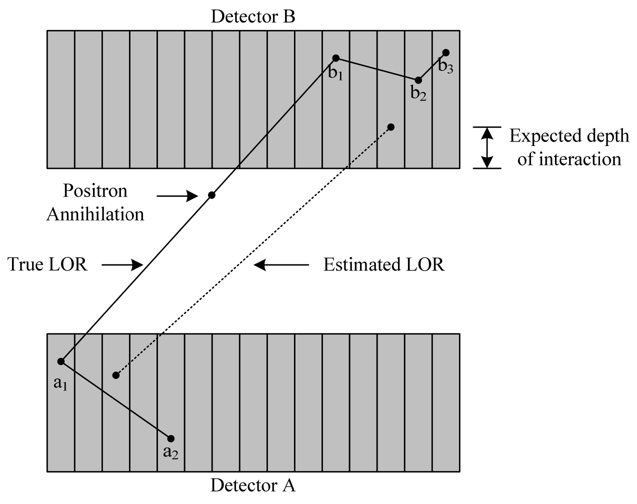

Fig. 1. The inaccuracy of a conventional PET scanner. The estimation error is cause by ignoring the depth and the order of the interaction.

In order to estimate the first interaction positions of photons, a statistical LOR estimation method is proposed in [3]. It uses a Bayesian estimator for determining a LOR by estimating three-dimensional interaction positions of a pair of annihilation photons. This estimation algorithm provides a significant improvement over Anger logic, but it also involves huge computations. In our experiment, the optimized algorithm takes 1.76 hours to estimate LORs from one million coincidences running with one thread in a computer with a 1 GHz AMD Opteron Processor 168. It requires processing millions of LORs in order to construct a high-resolution image visualizing the distribution of the radiotracers. For example in [4], it needs 250 million LORs for reconstructing an image. The execution time for the algorithm to construct such a high-resolution image will be 18.3 compute-days. This is totally unacceptable for clinic applications.

The goal of this work is to reduce the execution time of the LOR estimation algorithm by using Field-Programmable Gate Arrays (FPGA). Traditional FPGA implementations usually use hardware description languages (HDLs), such as VHDL and Verilog. HDL programming methodologies aimed at chip designs can be unsuitable for programming large-scale scientific applications [5]. In this papers, we use a C-to-FPGA tool-flow, called Impulse C. Impulse C allows us to rapidly map a C program into a platform that a host processor can cooperate with an FPGA device. In [6], it shows that the design time using Impulse C is less than the HDLs. Moreover, a program written in Impulse C is portable. Without modifying any codes, it could be re-mapped into a new FPGA device or platform.

The contribution of this paper is that we speed up the LOR estimation of a PET scanner using high level synthesis. On the XtremeData XD2000F platform, the system estimates LORs from one million coincidences in 79 seconds at 100MHz. A speedup of 82x is achieved compared to the optimized software. Another contribution is that we propose some improving methods for the application. These methods could also be applied to other applications or used to improve the high level synthesis tools.

II. Software Optimization

A. The Statistical LOR Estimation Algorithm

The kernel of the LOR estimation algorithm is to accumulating the probability of specific interaction sequences. Given an interaction sequence, $\{s_{A,} B_j\}$, the photon has I total interactions in detector A and the first interaction crystal in panel B is B_j. In detector A, the first interaction crystal is s_{A1}, the second one is s_{A2} and the *i-th* interaction crystal is s_{Ai}. Hence, the estimation confidence for the given interaction sequence is:

$$conf(s_A \mid B_j) = \sum_{dep_1=0}^{19} \cdots \sum_{dep_i=0}^{19} p(e_1, a_1, \ldots e_k, a_i \mid b_j), \quad (1)$$

where dep_k is a possible depth of interaction in crystal s_{Ak}. e_k is the computed estimated energy of a photon after the *k-th* interaction at position a_k, and e_0 is assumed to be 511 keV. The detail of computing the probability for an interaction sequence could be found in [3].

In this work, we only consider coincidences with 3 or less interactions in a detector. Because almost all coincidences have 3 or less interactions in a detector, there is no significant improvement of resolution to compute LOR from a coincidence with more than 3 interactions. Moreover, the performance will be decreased because of the algorithms factorial complexity.

B. Software Optimization

The original algorithm was implemented in C++ and most data types are double precision. To speed up the computation, we modify the data types to single precision and fixed-point.

Another optimization is loop invariant relocation. When computing Equation (1), there are some loop invariants which could be relocated out of a loop to reduce the computation. However, the iteration space varies from different number of interactions, and a recursion is used to compute Equation (1). It is not a trivial problem to relocate loop invariants for a recursion function. The recursion is mapped to an in-order tree traversal problem. The tree is *I*-depth for *I* interactions, and the leaves represent one iteration. All loop invariants are computed in non-leaf nodes and reused by its children nodes. The resulting optimized software version is used as our comparison point for our hardware speedup numbers.

III. FPGA Optimization

A. Impulse C

In this work, we use Impulse C [7] as the design tool to accelerate LOR estimation on the XtremeData XD2000F FPGA platform. The Impulse C tools include the CoDeveloper C-to-FPGA tools and the CoDeveloper Platform Support Packages (PSPs). A PSP includes platform-specific hardware/software libraries that support communication between the host and an FPGA device. With the PSP, one could port a program written in Impulse C to different platforms without modifying the source code.

The Impulse C programming model has two main types of elements: processes and communication objects. Processes are independently executing sections of code that interface with each other through communication objects. There are two kinds of processes: hardware processes that could be synthesized to HDLs, and software processes that run on the host computer.

Impulse C is a subset of the C language that has some restrictions [7] on hardware processes, and one of the constraints is no recursion is allowed. However, in our application, Equation (1) involves a recursion which is not allowed in Impulse C. To comply with the constraint, we started by only handling interaction sequences with 3 interactions and unrolling the recursion. Handling other sequences is discussed in Subsection E. Since the number of interactions is fixed, Equation (1) can be implemented with 3 nested loops.

Impulse C does not provide some of the necessary mathematics libraries used in our application, such as the square root and the exponential functions. We implemented these functions in VHDL and integrated them into our application by using Impulse C pragmas. The square root is implemented using the FPGA vendor's IP. The exponential function is implemented with two small lookup tables and a single-precision multiplier.

B. Impulse C Optimization

The Impulse C tool could automatically generate HDL codes from the original C codes for the selected FPGA platform. However, the generated HDLs are very inefficient, and the execution time is even longer than the software code. Hence, we use a loop pipelining technique and other optimizations to reduce the execution time.

Loop pipelining is an optimization technique that reduces the number of cycles required to execute a loop by allowing the operations of one iteration to execute in parallel with operations of one or more subsequent iterations. The pipelining efficiency is reported as the rate and the latency. The latency of a pipeline is the number of cycles to produce the first result, and the rate of a pipeline is the number of cycles to produce the next result. The number of total cycles to execute *N* iterations is: latency + rate * (*N*-1). If a loop has sufficient iterations, the dominant factor of execution time will be the rate. Unfortunately, in our application the innermost loop has only 20 iterations, and thus loop pipelining is inefficient in this case. Some efforts [8] can apply loop pipelining to outer loops, but Impulse C does not support it yet. Hence, we merge the three nested loops to one loop with 8,000 iterations. The original loop indexes are generated using division from the new index.

C. Improving the pipeline rate

Since the pipeline rate is the dominant factor, we focus on reducing the pipeline rate. The pipeline rate is dependent on three factors:

1) Non-pipeline operators. The pipeline rate is limited by the operators which take multiple cycles to execute them. For example, an external memory access in a loop would limit the pipeline rate, because it usually takes multiple cycles to access an external memory.

2) Data dependence. The input of an iteration can depend on the output of previous iterations. The iteration could not be executed until the dependence is solved, and thus reduces the pipeline rate. However, data dependence in the same iteration would not limit the pipeline rate.

3) Resource constraints. If a resource, such as a memory or an operator, is used in multiple stages of a pipeline, it also reduces the pipeline rate due to the competition of the same resource.

The first step to optimize the pipeline rate is to use pipeline operators. Most C language operations, such as addition, subtraction, multiplication and shift, are synthesized as pipelined operators. However, Impulse C synthesizes the division and modulo to multi-cycle implementations requiring one cycle for each bit in the result. It significantly decreases the pipeline rate. Hence, we use pipelined external HDLs to implement the divisions and modulos.

The second step to optimize the pipeline rate is to remove data dependence between iterations. Although there is no data dependence in Equation (1), accumulation may be considered as data dependence. Fortunately, Impulse C supports pipelined floating-point accumulation and it is enabled by a compiler option. The option forces the compiler to detect the accumulation operations and implement them using pipelined accumulators rather than adders.

Some coding styles may confuse the compiler and cause data dependencies. For example, incomplete conditional expressions can cause this kind of data dependences. The coding style needs to be corrected in order to improve the pipeline rate.

The last step to improve the pipeline rate is to remove resource constraints. In practice, we duplicate lookup tables that are referenced multiple times.

D. Reducing the pipeline latency

Although the latency of a pipeline has less impact on the performance, it has significant impact on the register utilization. Because a register is used to pass a signal from one stage to the next stage, each additional pipeline stage would increase the number of registers. Since Impulse C uses a C front-end to synthesize the data path, the operators are evaluated left-to-right. The latency might not be optimal. Hence, we manually rearrange the operations to avoid getting a very high pipeline latency.

E. Reusing Data Paths

As mentioned, the previous subsections describe the optimization for interaction sequences with three interactions. Once the 3-interaction sequences are accelerated, shorter interactions become the bottleneck. One might write another code to handle these sequences, but it is inefficient. We observe that the computation for sequences with different interactions is similar.

We reuse these operations by manually inserting a piece of code that selects inputs before invoking operators. Therefore, all sequences are handled by the same code and it is more efficient.

F. Reducing the communication between FPGA and the host

Since most computation is moved to the FPGA, the communication between FPGA and the host becomes a bottleneck. The next step to improve the performance is to reduce the communication time. We use stream objects to communicate between the host and the FPGA, and we observed the PSP of our platform implements the stream objects with polling. It queries the status of a stream object before the real data are sent. In our application, the data are crystal ids and signal pairs; we pack multiple data into a larger transfer. It required only one status query to send packed data. It efficiently reduces the communication time.

IV. EXEPERIMENT RESULTS

The design is implemented on the XtremeData XD2000F platform. The detail specification of the platform can be found in XtremeData website.

A. Performance Analysis

Figure 2 shows the software execution time of our implementations. The fixed-point software is very slow and we improved the performance using the methods described in Section II. The speedup is about 1.8x, but it is not efficient enough for medical applications. The program could be ported straight to the FPGA using Impulse C. However, the performance is even worse than the original program. This is because the non-pipelined FPGA implementation does not use the parallelism of the application and the clock rate is slower than the host. In FPGAs, the speedup is not achieved by operating the design at very high clock frequencies, but rather by exploiting the parallelism in the design.

The pipeline implementation exploits the loop level parallelism, with multiple iterations of a loop executed in parallel. The result shows that the pipelined implementation can obtain an 82x speedup over the optimized software.

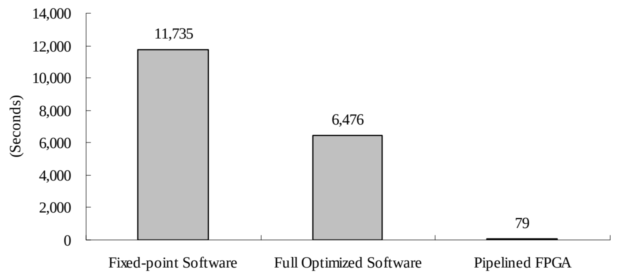

Fig. 2. Execution time of different implementations.

By applying the proposed methods, we accelerate the application. Figure 3 shows the detailed execution time of the application. Since we offload some computation to the FPGA, the execution time is divided into computation time and communication time. The communication time is the time that the host sends inputs (crystal ids and signal pairs) to the FPGA. The computation time is the time that the host waits for the result from the FPGA, and it includes the FPGA computation time and the time for transmitting the results from the FPGA to the host. Because shorter sequences occur frequently, they take most of the communication time.

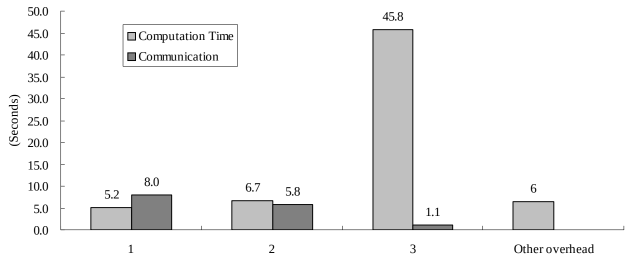

Fig. 3. Detailed execution time for sequences with different numbers of interactions.

We use loop pipelining to achieve the performance. All non-pipeline operators, data dependences and resource constraints are eliminated, and we are able to achieve the optimized pipeline rate 1 and the latency 258. The first result of the loop is generated in the 258-th cycle and one result is generated every successive cycle.

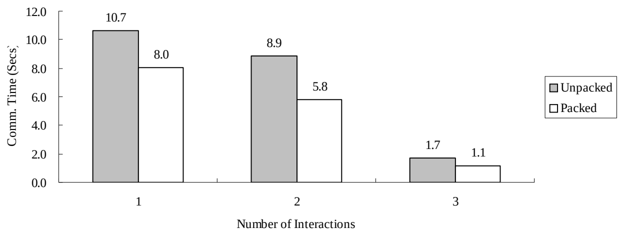

Fig. 4. Communication time for unpacked and packed stream.

In Section III.E, we describe methods to reduce the communication time by packing the data. Figure 4 shows the communication time for sending the inputs to the FPGA. Most communication time is consumed in sending inputs of sequences with 1 or 2 interactions, because most sequences are 1 or 2 interactions. Total communication time is reduced by 30% by packing the data.

V. Conclusion

Impulse C provides a design methodology that rapidly maps a C program to an FPGA platform, and the communication between the host and the FPGA platform is supported by the PSP. The program could be ported to another platform without modifying codes if the PSP of the platform is available. However, the generated HDLs of the original codes are very inefficient, and the parallelism of the program needs to be exploited to speed up the operations.

In this work we use loop pipelining to reduce the number of cycles required to execute a loop by allowing the operations of one iteration to execute in parallel with operations of one or more subsequent iterations. We propose methods to optimize the rate and the latency of a pipeline. These methods can also be applied to other applications, or used to improve the high level synthesis tools. The optimized pipeline rate is achieved by removing non-pipeline operators, data dependences between iterations, and resource constraints. Moreover, we reuse hardware to compute interaction sequences with different numbers of interactions. The results show that the FPGA implementation can obtain an 82x speedup over the optimized software.

Acknowledgements

We would like to thank Tom Lewellen, Robert Miyaoka, and Kyle Champley for the original source code and data sets for this project. We also thank Impulse Accelerated Technologies for the access to the Impulse C compiler and Altera for the FPGA compilers.

References

[1] S. R. Cherry, J. A. Sorenson, and M. E. Phelps, "Physics in Nuclear Medicine. Saunders," Elsevier Science, Philadelphia, PA, 2003.

[2] M. Wernick and J. Aarsvold, Emission tomography: the fundamentals of PET and SPECT: Academic Press, 2004.

[3] K. Champley, T. Lewellen, L. MacDonald, R. Miyaoka, and P. Kinahan, "Statistical LOR estimation for a high-resolution dMiCE PET detector," Physics in Medicine and Biology, vol. 54, p. 6369, 2009.

[4] J. J. Scheins, L. Tellmann, C. Weirich, E. R. Kops, C. Michel, L. G. Byars, M. Schmand, and H. Herzog, "High resolution PET image reconstruction for the Siemens MR/PET-hybrid BrainPET scanner in LOR space," in Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE, 2009, pp. 2981-2984.

[5] S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga, "Using FPGA Devices to Accelerate Biomolecular Simulations," Computer, vol. 40, pp. 66-73, 2007.

[6] J. Xu, N. Subramanian, A. Alessio, and S. Hauck, "Impulse C vs. VHDL for Accelerating Tomographic Reconstruction," in Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, 2010, pp. 171-174.

[7] Impulse-c. [Online]. Available: http://www.impulsec.com/

[8] K. Turkington, G. A. Constantinides, K. Masselos, and P. Y. K. Cheung, "Outer loop pipelining for application specific datapaths in FPGAs," IEEE Trans. Very Large Scale Integr. Syst., vol. 16, pp. 1268-1280, 2008.