Medical Imaging Process Accelerated in FPGA Hardware by 82x over Software

Line of Reaction Estimation for a PET scanner Optimized using C to FPGA Methodology

Zhongho Chen Department of CSIE, National Cheng-Kung University, Taiwan
Alvin W.Y. Su, Department of CSIE, National Cheng-Kung University, Taiwan
Ming-Ting Sun, Department of Electrical Engineering, University of Washington
Scott Hauck, Professor, Department of Electrical Engineering, University of Washington

Medical imaging tasks can require high-performance signal processing to convert sensor data into imagery to help with medical diagnostics. FPGAs are a compelling platform for these systems, since they can perform heavily pipelined operations customized to the exact needs of a given computation. In previous work we have benchmarked a CT scanner backprojection algorithm [http://ee.washington.edu/faculty/hauck/Tomography/]. In this article we focus on an FPGA platform and a high level synthesis tool called Impulse C to speed up 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 and an FPGA coprocessor. In this article, we describe some successful optimization methods for the algorithm using Impulse C. The results show that the FPGA implementation can obtain an 82x speedup over the optimized software.

Positron Emission Tomography (PET) is a nuclear medical imaging technique which produces a three-dimensional scan of the body. This technology is used in hospitals to diagnose patients and in laboratories to image small animals. 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 abundant gamma detectors. When a pair of photons is detected 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). A gamma detector is composed of crystals that interact with photons in order to improve the signal to noise ratio. However, it is difficult to estimate the exact depth of interactions. Moreover, a photon may interact with multiple crystals, which makes it harder to estimate the true LOR. 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. It 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.

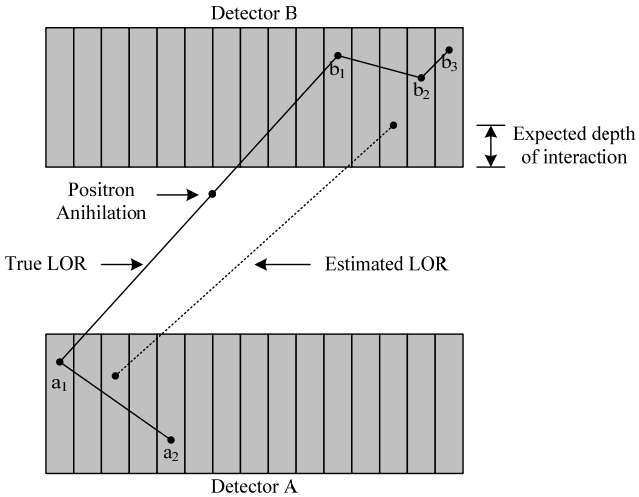

Fig. 1. The inaccuracy of conventional PET scanner. The estimation error is cause by ignoring the depth and order of interaction

Methods & Tools

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 under a variety of parameters, but it also involves huge computations. The optimized algorithm takes 1.76 hours to estimate LORs from one million coincidences running in a computer with an 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 unacceptable for clinic applications.

Our goal is to reduce the execution time of the LOR estimation algorithm by using Field-Programmable Gate Arrays (FPGA). FPGA devices can provide high computation power and flexibility. Traditional FPGA implementations use hardware description languages (HDLs) such as VHDL and Verilog. HDL programming methodologies aimed at chip designs are unsuitable for programming large-scale scientific applications [5]. In this project, we use a C-to-FPGA tool-flow called Impulse C. Impulse C allows us to rapidly map a C program into a platform such that the FPGA can co-process with a host processor. In [6] we showed that the design time using Impulse C is less than the HDLs. Moreover, a program written in Impulse C is portable. Without modifying any code it can be re-mapped into a newer FPGA device or platform.

We sped up the LOR estimation of a PET scanner using this high level synthesis approach. On the XtremeData XD2000F [13] 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 describe some methods for improving the application.

The Algorithm

The algorithm was developed for a high-resolution PET detector capable of providing depth-of-interaction information. Our collaborators at U.W. designed the PET system as an insert to a mammography unit; figure 2 shows the sketch of the PET system. There are four detector panels in the system, and each detector-panel is composed of depth-of-interaction micro-crystal element (dMiCE) crystal pairs. The size of the individual crystal element is $2\times2\times20$ mm^3 and each is coupled to its own 2×2 mm^2 micro-pixel avalanche photodiodes (MAPDs) [7]. A triangular-shaped light reflector is placed between two crystal elements in a dMiCE pair. The differential distribution of the scintillation light shared can be used to determine the depth of interaction.

To help the reader understand the complexity of the problem, we provide a quick overview of the calculations in the next sections. However, the exact formulas are not crucial for understanding the acceleration process, which begins at section "Computation Complexity" below.

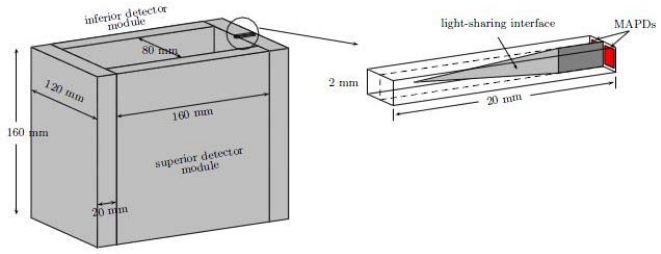

Fig. 2. Sketch of the PET system and the dMiCE crystal pair.

Depth of Interaction

The distribution of the light is currently modeled as a joint-normal distribution. The probability density function for the measured signal pair *sp* with a depth of interaction *d* mm and deposited energy λ keV is represented as:

$$\phi(sp \mid d,\lambda) = \frac{\exp(-\frac{1}{2}(sp - M(d,\lambda))' K(d,\lambda)^{-1}(sp - M(d,\lambda)))}{2\pi\sqrt{|K(d,\lambda)|}}, \qquad (1)$$

where $M(d, \lambda)$ and $K(d, \lambda)$ are respectively the mean vector and covariance matrix for the depth of interaction d and the deposited energy λ. The signal pair *sp* is a 2x1 vector. The deposited energy λ is the reduced energy of a photon after an interaction. These parameters are experimentally determined as discussed in [8]. Through these experiments, we find the M(d, 511) and K(d, 511) for d∈ {2, 6, 10, 14, 18}. The parameters for other values of d and λ are estimated through interpolation:

$$M(d,\lambda) = \frac{\lambda}{511} M(d,511)$$

$$K(d,\lambda) = \frac{\lambda}{511} K(d,511).$$

The Photon Process

The path length that a photon travels without interacting with the matter through which it passes is exponentially distributed. This is known as the Beer-Lambert Law, or simply as *attenuation*. The parameter of this exponential distribution, given by μ, is known as the attenuation coefficient, and it varies with the materials and the energy of the photon. Two dominate types of photon interactions present in PET imaging are photoelectric absorption and Compton scattering. Thus the attenuation coefficient of a photon with energy e is represented as:

$$\mu(e) = \mu_\tau(e) + \mu_\sigma(e),$$

where μ_τ is the photoelectric attenuation coefficient and μ_σ is the Compton scattering attenuation. After undergoing Compton scattering, the probability that a photon of energy e_{k-1} will have energy e_k is described by Klein-Nishina formula [9]:

$$KN(e_k \mid e_{k-1}) = \frac{1 + \left(\frac{e_k}{e_{k-1}}\right) + \left(\frac{e_k}{e_{k-1}}\right)\left[\left(\frac{511}{e_{k-1}} - \frac{511}{e_k} + 1\right)^2 + 1\right]}{e_k e_{k-1}} \times C(e_k \mid e_{k-1}) \quad (2)$$

where $C(e_k|e_{k-1})$ is a normalization factor:

$$C(e_{k-1}) = \begin{cases} 1, & \frac{511 e_{k-1}}{511 + 2e_{k-1}} \le e_k \le e_{k-1}, \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, the energy of the photon that scatters at θ degree is given by:

$$e_k = \frac{511 e_{k-1}}{511 + e_{k-1}(1 - \cos\theta)}. \quad (3)$$

The Statistical LOR Estimation Algorithm

The algorithm estimates an LOR from a coincidence, which is basically coupled signal-pairs measured by dMiCEs within a short timing window. A signal-pair is composed of two measured signals, and we assume that a signal-pair is caused by one interaction which happens in the crystal element with the larger measured signal. The interaction position is assumed to be in the center of the crystal, with an unknown depth. A LOR is the line that connects with two first interaction positions of a pair of photons. 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 *ith* interaction crystal is s_{Ai}. Hence, the estimation confidence for the given interaction sequence is:

$$conf(s_A \mid B_j) = \sum_{dep_1=0}^{19} \cdots \sum_{dep_i=0}^{19} p(e_1, a_1, \ldots e_k, a_i \mid b_j), \quad (4)$$

where dep_k is the depth of interaction in crystal s_{Ak}. The first interaction position is estimated by computing the expected depth of interaction:

$$\frac{\sum_{dep_1=0}^{19} \cdots \sum_{dep_i=0}^{19} p(e_1, a_1, \ldots e_i, a_i \mid b_j) \times dep_1}{\sum_{dep_1=0}^{19} \cdots \sum_{dep_i=0}^{19} p(e_1, a_1, \ldots e_i, a_i \mid b_j)}. \quad (5)$$

The probability density function for the interaction path is given by the equation:

$$
\begin{aligned}
& p(e_1, a_1, \ldots e_i, a_i \mid b_j) \\
& = \left[from\_PE + from\_CS \right] \\
& \times \prod_{k=1}^{I} \exp(-\mu(e_{k-1}) \| a_{k-1} - a_k \|) \qquad (6) \\
& \times \prod_{k=1}^{I-1} KN(e_k \mid e_{k-1}) \mu_\sigma(e_{k-1}) \phi(sp_k \mid dep_k, e_{k-1} - e_k) \\
& from\_PE = \mu_\tau(e_{i-1}) \phi(sp_i \mid dep_i, e_{i-1}) \\
& from\_CS = KN(e_i \mid e_{i-1}) \mu_\sigma(e_{i-1}) \phi(sp_i \mid dep_i, ML\lambda)
\end{aligned}
$$

where e_k is the energy of a photon after the *kth* interaction at position a_k, and e_0 is assumed to be 511 keV. The first line of the equation computes the probability of the last interaction. The last interaction might be photon absorption or Compton scattering. Hence, the algorithm computes both probabilities as *from_PE* and *from_CS*. The second line computes the probabilities according to the Beer-Lambert Law. The third line computes the probability of Compton scattering of each interaction except the last one.

An interaction position is derived from the depth of interaction with a specific crystal. In our application, the depths are equally sampled with 1 mm interval. The interaction depth of B_j is also estimated with the maximum likelihood method. In other words, the depth with the maximum $\phi(sp \mid d, \lambda)$ is assumed to be the interaction depth of B_j in order to compute the probability density function. If the last interaction is a Compton scattering, we assume the deposited energy is MLλ, which is also estimated from a signal pair using the maximum likelihood method.

The algorithm estimates the first interaction crystal by computing a confidence matrix:

$$
CM[i][j] = \sum_{s_A}^{s_{A1}=A_i} conf(s_A \mid B_j) \times \sum_{s_B}^{s_{B1}=Bj} conf(s_B \mid A_i) \qquad (7)
$$

The A_i and B_j with the maximum confidence $CM[i][j]$ are assumed to be the first interaction crystals. Therefore, the interaction sequence is the $\{s_A, B_j\}$ and $\{s_B, A_i\}$ with the maximum confidence. The estimated first interaction positions of the sequences are selected to construct an LOR for the coincidence.

Computation Complexity

Given a coincidence with I total interactions in detector A and J total interactions in detector B. The total computation of equation (3) is $J \times I! \times 20^I + I \times J! \times 20^J$. For example, it involves 288,000 computations of equation (6) to estimate an LOR from a coincidence with 3 interactions in both detectors. Figure 3 shows the percentage of photons with a given number of interactions. Because most coincidences have 3 or less interactions in a detector, there is no significant improvement in resolution to compute LOR from a coincidence with more than 3 interactions, but photons with 4 or more interactions require excessive time to analyze. In this work, we only consider coincidences with 3 or less interactions in a detector.

Software Optimization

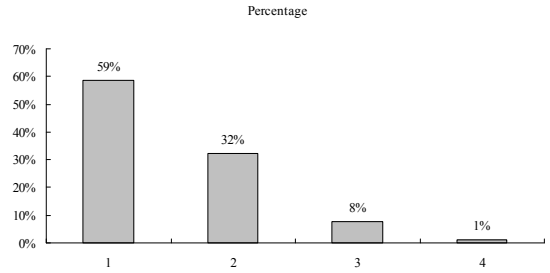

Fig. 3. Percentage of photons with a given number of interactions.

Some coincidences with too small signals are considered as noise and are not used in estimating LORs. Table I is the detailed profile of these one million coincidences, and it can be used to analyze the performance bottleneck. There are actually 657,273 coincidences, and they are used to evaluate the LOR estimation algorithm.

TABLE I. DETAIL PROFILE OF ONE MILLION COINCIDENCES

Number of Interactions in two detectors	1	2	3
1	249607	270369	40272
2		73566	21794
3			1665

The original algorithm [3] was implemented with C++ and most data types are double precision. To speed up the computation the data types are modified to single precision and fixed-point without introducing significant errors. The probability calculated from equation (6) is stored in single precision rather than fixed-point because of the exponential term; the dynamic range of the exponential values is too large to represent in a proper fixed-point value. Moreover, since the purpose of the algorithm is to estimate the depth of interaction rather than the exact probability, the constant scalars can be removed. For example, the constant scalar $1/2\pi$ in equation (1) is removed without affecting the results.

Another optimization is loop invariant relocation. When computing equation (4), there are some loop invariants which are relocated out of a loop to reduce the computation. However, the iteration space varies for different number of interactions, since a recursion is involved to compute equation (4). It is not a trivial problem to relocate loop invariants for a recursive 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 last optimization is to implement functions with lookup tables, such as computing the position from a given crystal and depth. One major lookup table is the Klein-Nishina formula. Because we use a 9-bit unsigned integer to represent photon energy, all results of the formula are pre-computed and stored in a two-dimensional lookup table. A lot of computation time is saved by using the lookup table.

Software Profiling

After optimization, we produce a 1.8x speedup with the new software version, but the performance is still limited. Figure 4 shows the execution time for estimating LORs from one million coincidences. Although coincidences with 3 interactions occur only 8% of the time, they take most of the execution time. The following sections describe the methods to speed up the computation using FPGA hardware.

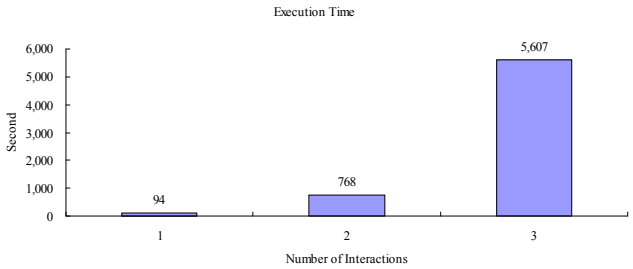

Fig. 4. Execution time for processing the one million coincidences from table I, broken out by number of interactions.

FPGA Optimization

Impulse C

Impulse C [10] was used as the design tool to accelerate the LOR estimation on the FPGA platform XtremeData XD2000F [13]. 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 the 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 source code.

The Impulse C programming model has two main types of elements: process 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 process that run on the host computer. We use a hardware process to compute equation (4) and (5), and a software process to compute the confidence matrix and perform I/O

tasks. The communication between the hardware and software processes is through stream objects that transmit crystal IDs and signals pairs.

Impulse C is a fully compatible subset of ANSI C that has some constraints [10] for hardware processes. One constraint is no recursion. However, in our application equation (4) involves a recursion which is not allowed in Impulse C. To comply with the constraint, we start with only handling interaction sequences with 3 interactions; handling other sequences is discussed in a subsequent section. Since the number of interactions is fixed, equation (4) can be implemented with 3 nested loops.

To add the necessary mathematics libraries used in our application, such as square root and exponential, we use an Impulse C pragma. The square root is implemented using the FPGA vendor's IP. In our application, the input of the exponential function is always negative. We keep 14 bits for computing the exponential functions. If we use a lookup table to implement the function, the table is too large for the FPGA. Instead, the 14 bits are divided into most and least significant portions, each 7 bits. Given the equation:

$$e^{A+B} = e^{A} \times e^{B},$$

the exponential function is implemented with two small lookup tables and a single-precision multiplier.

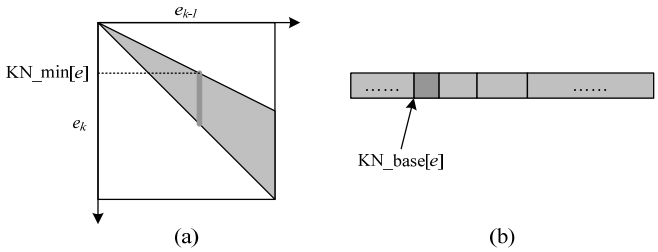

Fig. 5. Klein-Nishina Table. (a) The original two-dimension table (b) The compressed one-dimension table

The Klein-Nishina table is also too large for our FPGA device, since the original table contains 262,144 single-precision entries. Fortunately, we observe that most entries of the table are zeros. Figure 5a shows the two-dimension table, where the gray region of the table is non-zero and the others are zeros. We compress the table into a one-dimension table, *KN_tab*, and two index tables, *KN_min* and *KN_base*. The compression reduces the number of entries from 262,144 to 71,639, and the compression ratio is about 0.27. The Klein-Nishina formula is computed by these three tables:

if $e_k > e_{k-1}$ OR $e_k <$ KN_min[e_{k-1}]
  KN($e_k|e_{k-1}$) = 0;
else
  index = KN_base[e_{k-1}] +e_k- KN_min[e_{k-1}];
  KN($e_k|e_{k-1}$) = KN_tab[index];
end if

Impulse C Optimization

The Impulse C tool will automatically generate HDL from C code for the selected FPGA platform without modification. However, without refactoring the generated HDL is inefficient; the execution time is even longer than the software code. Hence, we use Impulse C's loop pipelining 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 subsequent iterations. The pipelining efficiency is reported as the rate and latency. The latency of a pipeline is the number of cycles to produce the first result, and the rate of a pipeline is the cycles to produce the next result. The number of total cycles to execute *N* iterations of a loop is: latency + rate * *N*. If a loop has sufficient iterations, the dominant factor of execution time will be the rate.

We merge three nested loops to one loop with 8,000 iterations. The original loop indexes are generated using division from the new index. Figure 6 shows the modified loop. The loop body computes equation (6). The division is implemented with external HDL in order to get both the quotient and remainder. It introduces a little logic, but significantly improves the performance.

Improving the pipeline rate

The pipeline rate is the dominant factor in the runtime. This rate is dependent on three factors:

1) Non-pipeline operators. The pipeline rate is limited by the operators which take multiple cycles to execute. 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. This is called a recursion in Impulse C. The iteration cannot be executed until the dependence is solved, which also reduces the pipeline rate. However, data dependences 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 competition for the same resource.

for dep=0; dep<8000; dep++;
  tmp = dep / 20 ..... dep_1;
  dep3 = tmp / 20 …… dep_2;
  calculate a_1, a_2, a_3 from depths and crystal ids;
  compute photon entrance position, a_0, from a_1 and b_j;
  calculate e_1, e_2, using equation (3)
  $e_3 = e_2$ - MLλ;
  d0_prob = $e^{-\mu(e_0)\|a_0-a_1\|}$;
  d1_prob = $e^{-\mu(e_1)\|a_1-a_2\|}$;
  d2_prob = $e^{-\mu(e_2)\|a_2-a_3\|}$;
  cs0_prob = KN[e_0][e_1];
  cs1_prob = KN[e_1][e_2];
  lr1_prob = $\phi(sp_1 \mid dep_1, e_0 - e_1)$;
  lr2_prob = $\phi(sp_2 \mid dep_2, e_1 - e_2)$;
  from_PE= $\mu_\tau(e_2)\phi(sp_3 \mid dep_3, e_2)$;
  from_CS= KN[e_2][e_3]$\mu_\sigma(e_2)\phi(sp_3 \mid dep_3, ML\lambda)$;
  integral += (from_PE+from_CS)*
        d0_prob* d1_prob* d2_prob *
        cs0_prob * cs1_prob* lr1_prob * lr2_prob;
end for

Fig. 6. The pseudocode of equation (4) after merging loops.

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. Impulse C synthesize the division and modulo to multi-cycles implementations requiring one cycle for each bit in the result. It significantly decreases the pipeline rate. Hence, we use external HDLs to implement the divisions and modulus.

The second step to optimizing the pipeline rate is to remove data dependence between iterations. Although there is no data dependence in equation (4), floating-point accumulation may be considered as data dependence. Fortunately, Impulse C supports the floating-point accumulation via a compiler option. The option forces the compiler to detect the accumulation operations and implement them using pipelined floating-point accumulators rather than adders.

Some coding styles may confuse the compiler and cause data dependence. For example, figure 7 shows a piece of C code that may confuse the compiler to cause unnecessary data dependence. The loop contains a switch statement, and the successive statement is dependent on the output of the switch statement. Although the dynamic rage of the *id* is 0 to 1, the compiler would believe that there is data dependence between iterations due to the missing default case of the switch statement. Incomplete

conditional expressions would also cause this kind of data dependence. The coding style needs to be fixed in order to improve the pipeline rate.

```
for (i=0; i<10; i++) {
        switch(id) {
        case 0:
          x = 1;
          break;
        case 1:
          x = 2;
          break;
        }
        y = x;
}
```

Fig. 7. A piece of C code that may confuse the compiler to cause unnecessary data dependence between iterations.

The last step to improve the pipeline rate is to avoid the resource constraints. In practice, we duplicate lookup tables that are referenced multiple times. We implement the exponential function and Klein-Nishina formula with lookup tables. These tables are referenced multiple times in the loop and limit the pipeline rate. The Impulse C compiler does not automatically duplicate these tables, but we could duplicate them manually. In figure 6 the Klein-Nishina table is referenced three times, which means it needs three duplicates. Unfortunately, it is too large to fit all three duplicates in our FPGA, even the compressed ones. We observe that it is unnecessary to duplicate the whole table for computing the probability of the first Compton scattering, because the energy of the photon is assumed to be 511 keV before the first interaction. Another table is removed by directly computing equation (2), but it involves a lot of FPGA resources.

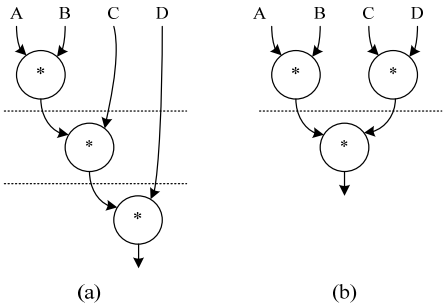

*Fig. 8. Two datapaths that perform the identical operation, A*B*C*D. (a) the operators are evaluated in left-to-right order with a latency of 3. (b) A balanced datapath with a latency of 2.*

Reducing the pipeline latency

Although the latency of our pipeline has a small 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 state would increase the number of registers. In figure 6, the variable *integral* is the product of multiple factors. According to the commutative law of multiplication, a product is not changed by the rearrangement of its factors. However, it does affect the pipeline latency to rearrange factors. Figure 8 shows an example for multiplication. Two datapaths perform the identical operation, but they have different latencies. Impulse C uses the C front-end to synthesize the datapath, where operators are evaluated left-to-right. Thus, we manually rearrange the operations to avoid getting a high pipeline latency.

Reusing Datapaths

As mentioned, the previous subsections describe the optimization for interaction sequences with three interactions. Other sequences still take a significant amount of execution time. The computation for sequences with different interactions is similar. For example, all sequences compute the probability of *from_PE* and *from_CS*. Thus, through careful design we can reuse the 3 interaction datapaths. In Figure 6, the number of iterations is modified to be queried with a lookup table in order to handle sequences with different number of interactions using the same loop.

Although the computation of different number of interactions is similar, they take different inputs. For example, $\mu_\tau(e_{i-1})$ in equation (4) is computed for all sequences, but the inputs are different. The sequences with 1/2/3 interactions take e_2, e_1, e_0 as the input respectively. We can reuse these operations by inserting a piece of code that selects inputs before invoking operators. Therefore, all sequences are handled by the same code, significantly reducing the required FPGA resources.

Reducing the communication between the FPGA and the host

Since most computation is moved to the FPGA, the communication between the 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 observe the PSP of our platform implements the stream objects with polling. It is required to query the status of a stream object before the real data is sent. In our application, this data is crystal ids and signal pairs, and we pack multiple data into a large block. It thus requires only one status query to send this packed data, reducing the communication times.

Experiment Results.

The design is implemented on the XtremeData XD2000F platform. Figure 9 shows the platform, which has two FPGAs: a bridge FPGA and an application FPGA. The bridge is responsible for communication with the AMD CPU via the HyperTransport protocol. The application circuit is placed in the Application FPGA, and communicates with the Bridge FPGA through the XOVE dedicated communication interface. These interfaces are generated by the Impulse C PSP, saving a lot of design time. Four QDR II SRAMs are connected to the application FPGA that are useful for applications that have large memory requirement. Two DDR-II DRAMs are shared with the Application FPGA and the AMD CPU. In our application, we use stream objects to communication between the FPGA and the host, and communication is through the Bridge FPGA.

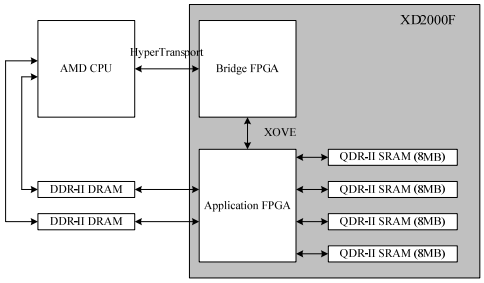

Figure 9. The XD2000F Platform

Performance Analysis

Figure 10 shows the execution time of our implementations: fixed-point software, optimized software, and pipeline FPGA. The fixed-point software is very slow, and we improved the performance using the methods described in Section III. The speedup is about 1.8x, but it is not efficient enough for medical applications. We produced a pipelined FPGA implementation that exploits the loop-level parallelism in the application. The result shows that the pipeline implementation obtains a 82x speedup over the optimized software.

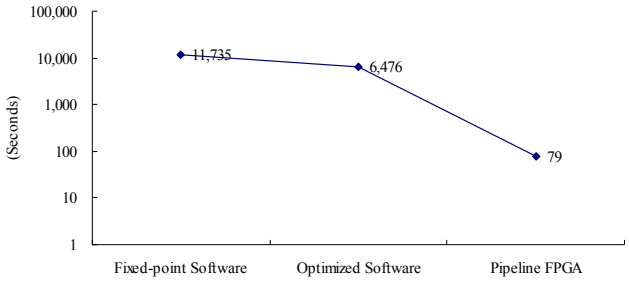

Fig. 10. Execution time of fixed-point software, optimized software, and pipeline FPGA implementations.

By applying the proposed methods, we significantly speed up the application. Figure 11 shows the real and estimated execution time of the application. The real execution time is measured with the computation 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 transmitting the results from the FPGA back to the host. We could estimate execution time from the profile (Table I) and the rate and latency of the pipeline. The estimated execution time is the best case for the implementation. The estimated and real computation times are very close. The communication time is the major difference between the real and estimated computation time.

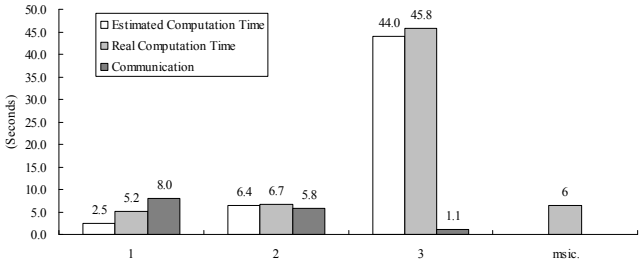

Figure 11. Execution time for sequences with different numbers of interactions.

We use loop pipelining to achieve this performance. The major loop is used to compute equation (4) and (5). All non-pipelined operators, data dependence and Resource constraints are eliminated, and we achieve an optimized pipeline rate of 1 and a latency of 258. The first result of the loop is generated in the 258th cycle, and one result is generated in every successive cycle.

Later we described methods to reduce the communication time by packing the stream data. Figure 12 shows the communication time for sending 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 interactions according to Table I. Total communication time is reduced by 30% by packing the stream data.

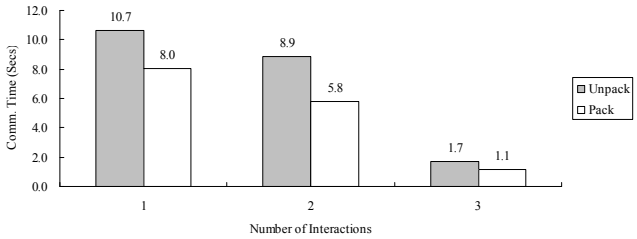

Figure 12. Communication time for packed and unpacked streams.

Resource Analysis

Impulse C generated HDLs are synthesized with the Altera Quartus II Tool. Table III shows the synthesis report of the same architecture with different pipeline latencies. Because we use very deep pipelining to achieve the performance, the implementation consumes more registers than ALMs (Adaptive Logic Modules). Although the latency of a pipeline has less impact on the performance, it has significant impact on the register utilization. We reduced the pipeline latency by using the method described in Section 4.D. 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. The memory bits are used in lookup tables, especially tables for calculating the Klein-Nishina formula.

TABLE III. SYNTHESIS REPORT OF THE PROPOSED ARCHITECTURE. DEVICE EP2S180F1508C3, CLOCK RATE IS 100MHZ

Multiplication Rearranging	Without	With
Pipeline Latency	291	258
ALMs	33,581	33,567
Registers	89,776	75,769
Memory Bits	2,969,408	2,969,408

Conclusion and future work

Impulse C provides a design methodology that rapidly maps a C program to an FPGA platform and the communication between the host and FPGA platform is supported by the PSP. The program could be ported to another platform without modifying the code, if the PSP of the platform is available. In this work, we use the loop pipeline 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 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 dependence between iterations and resource constraints. Moreover, we reuse the data that could compute all interaction sequences with different number of interactions. The results show that FPGA implementation can obtain an 82x speedup over the optimized software.

Although the estimation algorithm provides a significant improvement over Anger logic, it has assumptions used to reduce computation which may reduce the estimation precision. For example, the algorithm assumes that a signal pair is caused by exactly one interaction. In practice, a signal pair is sometimes caused by multiple interactions in the same crystal. With our FPGA acceleration, these assumptions could be removed, making the estimated results be more accurate

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] W. C. J. Hunter, R. S. Miyaoka, L. R. MacDonald, and T. K. Lewellen, "Measured temperature dependence of scintillation camera signals read out by geiger-Muller mode avalanche photodiodes," in Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE, 2009, pp. 2662-2665.
[8] T. K. Lewellen, L. R. MacDonald, R. S. Miyaoka, W. McDougald, and K. Champley, "New directions for dMiCE - a depth-of-interaction detector design for PET scanners," in Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE, 2007, vol. 5, pp. 3798-3802.
[9] Robley D. Evans. "The Atomic Nucleus. McGraw-Hill", New York, NY, first edition,1955..
[10] Impulse-c. [Online]. Available: http://www.impulsec.com/
[11] 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.
[12] Z. Guo, W. Najjar, and B. Buyukkurt, "Efficient hardware code generation for FPGAs," ACM Trans. Archit. Code Optim., vol. 5, pp. 1-26, 2008.
[13] http://www.xtremedata.com/