An FPGA Solution for Radiation Dose Calculation

Kevin Whitton[1] X. Sharon Hu[1] Cedric X. Yu[2] Danny Z. Chen[1]

***Abstract*— Radiation dose calculation is an important step in the treatment of cancer patients requiring radiation therapy. It ensures that the physician prescribed dose agrees with the dose delivered to the patient. Current methods use software implementing either three-dimensional (3-D) convolution/superposition algorithms or Monte Carlo analysis. These software methods create a bottleneck in radiation therapy. The required computation time limits both the accuracy of the calculation and the number of patients whom can be treated.**

This paper presents a novel FPGA implementation for radiation dose calculation. The implementation is based on the 3-D convolution/superposition collapsed cone algorithm [1]. To achieve higher accuracy and performance, the original algorithm has been modified and advanced design techniques were applied. Experimental data demonstrate that the FPGA implementation shows significant improvements over software implementation.

I. Introduction

Technology advances in programmable logic devices (PLDs) have made designs based on PLD chips ever more popular. This is particularly true for applications where performance is a critical consideration yet the system quantity makes the application specific integrated circuits (ASIC) approach economically infeasible. However, designing high performance FPGA-based systems is not a trivial task as it requires intimate understanding of the applications from other disciplines. In this paper, we present an FPGA implementation for a task in radiation therapy, i.e., radiation dose calculation.

Radiation therapy involves the use of ionizing radiation to kill cancer cells and shrink tumors [2]. Radiation dose calculation, a critical step in radiation therapy, involves computing the amount of energy, released via a photon beam, that is deposited within a region of interest in a patient. These areas of interest are further broken down into small cubic volume elements referred to as voxels. The amount of energy deposited in each voxel is accumulated and the final dose is calculated by dividing the total energy per voxel by the average density of the matter in that voxel.

Desirable dose calculation approaches require both high accuracy and high speed. High accuracy enables the physician to be confident in his prescription and to be assured that enough radiation reaches the tumor, while not damaging critical structures (such as heart, lungs, etc.). Faster calculation allows more patients to be treated because treatment plans can be developed quicker.

Up till now, dose calculation is done in software exclusively. Depending on the actual algorithm and computer platform used, dose calculation can take anywhere from several minutes to hours. However, the dose calculation results obtained within minutes are often too inaccurate to be used clinically.

To investigate the feasibility of implementing radiation dose calculation in FPGA, we have selected a collapsed cone dose calculation algorithm and realized it on a commercial FPGA chip. The algorithm has some unique characteristics that demand careful evaluation of different design alternatives, particularly with respect to the tradeoff between computational load and memory bandwidth. The original algorithm was modified for increased accuracy and flexibility. The design was also pipelined to improve the performance of the final solution.

Our FPGA-based design was created with the use of Handel-C and VHDL. The design has been fully tested with ModelSim™[3]. In the future we plan to fully test it on an FPGA development board. This new implementation is shown to be faster than a commodity CPU. In tests performed, it has outperformed a CPU by up to a factor of 25. This improvement would allow more patients to be treated and allow for accuracy corrections to be added without increasing overall calculation time.

The rest of this paper is organized as follows. In Section II we present a brief overview of the dose calculation algorithm implemented along with existing efforts to speed up dose calculation. Section III gives an introduction of the overall system design. Sections IV and V present details on modifications of the original algorithm and tradeoffs made between computational load and memory bandwidth. Some experimental results are shown in Section VI and conclusions are presented in Section VII.

II. Preliminaries

In this section, we first give a brief overview of some necessary background information followed by a discussion on the dose calculation algorithm that we have implemented. Then, we review existing efforts to speed up dose calculation.

[1]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. Email: {kwhitton, shu, dchen@nd.edu}. The work of these authors is supported in part by the Notre Dame Faculty Research Program grants.

[2]Department of Radiation Oncology, University of Maryland School of Medicine, Baltimore, MD 21201, USA.

A. Background Information

Generally speaking, dose calculation determines the amount of energy deposited within a phantom (material used for the absorbing medium in experimental dose measurement) from photons produced by a linear accelerator or other radiation emitting device. To accurately calculate radiation dose, many different factors must be considered. First, the source of the radiation is used to determine beam energy. When this source is known, the composition of tissues in the patient is determined. Once the composition of tissues is known, and the energy (fluence) of the photon beam is known, the distribution of radiant energy released locally in a patient can be easily calculated [1]. This energy is then further distributed by secondary particles (electrons, positrons, and photons) until it is absorbed as dose, or it escapes from the boundaries of the patient and is no longer of interest. This two-step process is the basis for several methods of dose calculation where the distribution of primary energy is convolved with a kernel describing the energy spread by the secondary particles. The distribution of primary energy is typically referred to as the TERMA (Total Energy Released to MediA).

There are three main methods of dose calculation in use today. One class of algorithms are known as convolution, another as superposition, and the last as Monte Carlo analysis. Monte Carlo methods are the most accurate and also the most computationally intensive methods available. They are not used in clinical practice due to the enormous computational load they generate. Convolution and superposition methods are less accurate, but can typically be computed in much less time than Monte Carlo methods. An overview of a convolution method can be found in [4] and a Monte Carlo overview can be found in [5]. Current running times for dose calculation range from 5 to 45 minutes. (These numbers were provided by the University of Maryland School of Medicine, where two different dose calculation systems are in use, the Pinnacle system from Philips, and Prowess Panther from Prowess, Inc.)

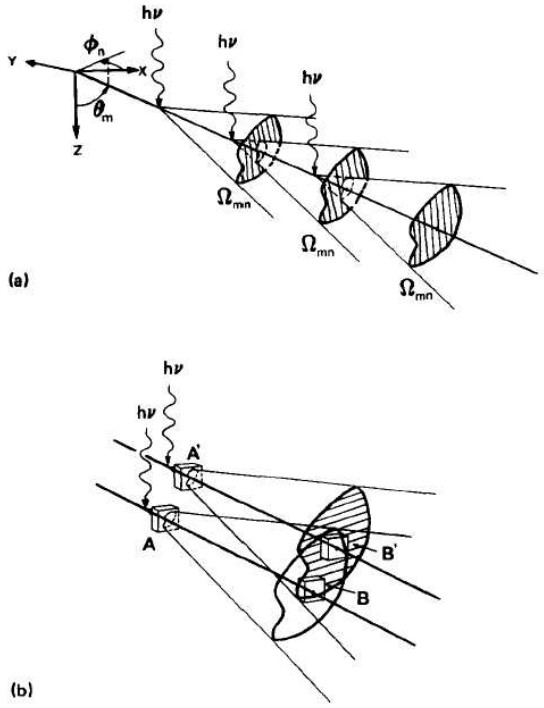

Fig. 1. (a) In the collapsed cone approximation, all energy released from primary photons at elements on a cone axis of direction (Θ_m, ϕ_n) and fbwing in coaxial cones of solid angle Ω_{mn}, is rectilinearly transported and deposited on the axis. In order to transport all energy released, several cone axes must be used, such that $\sum_{m,n} = 4 * \pi$, and parallel axes of each direction, must cover the irradiated volume. In the fi gure, only one axis is shown. An infi nite number of axes, where $\Omega_{mn} \rightarrow 0$, yields the continuous case of rectilinear motion. (b) Close to an interaction point the corresponding cone covers only a fraction of a Cartesian voxel, as indicated by the voxels and $A^{'}$. Further away, also off axis voxels are covered by the cone, as indicated by B and $B^{'}$. In the collapsed cone approximation, the energy that should have been deposited in voxel $B^{'}$ from interactions at the vertex of the lower cone is deposited in voxel B and vice versa. The relative accuracy of the calculated energy deposition therefore decreases with the distance from the scattering point. The error is small since the magnitude of polyenergetic point spread functions decreases rapidly with the distance from the interaction point

B. Algorithm overview

In this paper, we adopt the collapsed cone convolution algorithm first proposed by Ahnesjö in [1]. It is believed to be one of the best algorithms in terms of acceptable computational accuracy and efficiency. The essence of the collapsed cone approximation can be described by Figure 1 (taken from [1]).

The "Collapsed Cone" approximation works as follows: all energy released into coaxial cones of solid angle Ω_{mn} from volume elements on the axis is rectilinearly transported, attenuated and deposited in elements on that axis [1]. This allows for accurate computation of the dose deposited to all voxels while performing many less computations. As stated in [1], the dose can now be calculated with $M * N^3$ calculations, where M is the number of cones (typically between 48 and 384) and N is the number of Cartesian voxels along one side (i.e., there are N^3 voxels). A performance-comparable FFT-based method described in [6] requires $(2N)^3 * log_2 2N$. However, FFT-based methods require an invariant kernel, cannot be extended to handle inhomogeneity, and thus lead to less accurate dose calculation.

The kernel used in this implementation is the exponentially calculated kernel presented in [1]. Along with the kernel, the paper presented the necessary equations for calculating radiation dose, given an initial TERMA matrix in Cartesian coordinates. The exponentially calculated kernel is composed of two different components. One component calculates the primary energy (which has a rather sharp falloff as distance from the interaction point increases) and the other component calculates the scatter energy (which

does not have near as sharp a drop-off as the primary energy). The paper presents the equations to calculate each of these, and they are presented in Equations (1), (2), and (3).

$$\Delta R^p_{mn}(r_i) = T_i \rho_i \Omega_{mn} d^2 u \int_{r_i-1}^{r_i} \frac{A_m}{a_m} e^{-a_m \eta_i (r_i - s)} ds$$
$$= T_i \rho_i \Omega_{mn} d^2 u \frac{A_m}{\eta_i a_m^2} (1 - e^{-a_m \eta_i \Delta r}) \quad (1)$$

$$R^p_{mn}(r_i) = R^p_{mn}(r_{i-1}) e^{-a_m \eta_i \Delta r} + \Delta R^p_{mn}(r_i) \quad (2)$$

$$R^s_{mn}(r_i) = R^s_{mn}(r_{i-1})(1 - b_m \eta_i \Delta r) + T_i \rho_i \Omega_{mn} d^2 u \frac{B_m}{b_m} \Delta r \quad (3)$$

In the above equations, T_i represents the TERMA, ρ_i represents the mass density, and η_i the relative electron volume density of the segment i. Also, mn represents the current cone being calculated and d^2u represents a volume associated with the receiving voxel element. Equation 1 calculates the amount of primary energy released from the segment i, where r goes from r_{i-1} to r_i, into the cone of solid angle Ω_{mn}. Equation 2 represents the radiant primary energy through all segments on a line. Equation 3 is used to calculate the scatter energy. It does not explicitly require the exponentials because the scatter energy is only a small fraction of the total energy and hence can be approximated by the first two terms of the serial expansion. These equations serve to determine the amount of energy deposited in different voxels, which can then be used in calculating the final dose shown in Equation (4).

$$D(r) = \frac{\eta(r)}{\rho(r)} \frac{1}{d^2 u} \sum_m \sum_n [a_m R^p_{mn}(r) + b_m R^s_{mn}(r)] \quad (4)$$

In this equation, $\eta(r)$ is divided by $\rho(r)$ in order to convert energy per voxel into dose per voxel. These equations allow computation of the dose in a phantom, provided values are given for the TERMA, the mass density, and the relative electron volume density. Suitable values for A_m, a_m, B_m and b_m are provided as constants in the paper. The value of Ω_{mn} can easily be calculated using Equation (5). In the equation, the 4π radians of the solid sphere are simply divided by the number of cones used.

$$\Omega_{mn} = \frac{4 * \pi}{max\{mn\}} \quad (5)$$

These equations serve to compute the dose, but, are not the best method for implementation in an FPGA. They involve a lattice of parallel lines which must be computed and laid out. Basically, if there are 48 cones emanating from each dose calculation point ($\Delta\Theta = 3.75°, \Delta\phi = 360°$), then 48 different lattices would need to be constructed such that, for each lattice, no voxel was crossed by more than one line for that lattice. Then, for each line in the lattice, starting at the point where that line intersects with a voxel in the region of interest, Equations (1)–(3) would be run. Once these equations finished, the final dose could be calculated using Equation(4). This lattice computation introduces additional calculations and hardware into the dose calculation phase, and use of the lattice does not allow for easy use of dose correction factors.

One of the correction factors that this method does not easily accommodate is kernel tilting. Kernel tilting allows for the computation kernel to be tilted in various ways to allow more accurate dose calculation and is described in [7]. While the goal of our research is to increase speed, it is not to do so at the expense of accuracy. The goal is, as FPGAs become faster, to allow the hardware to be easily modified for increased accuracy. That is not easily accomplished using the above equations. Instead, a slightly different method is used to calculate R^p_{mn} and R^s_{mn} which is presented in Sections IV-A and IV-B and is easily adaptable to include factors such as kernel tilting.

C. Related Work

Much of the research available concentrates on providing parallel implementations of different radiation dose calculation algorithms. The results obtained demonstrate the potential in using multiple processors to compute the dose, and illustrate the parallelism available in dose calculation. Current parallel processing machines, however, present additional facility concerns, such as cost, cooling, space, and power requirements, that may be mitigated by using dedicated hardware.

In [8], Murray and others present a multicomputer approach to speed up dose calculation using transputers. They were able to achieve a 7.81x speedup by using 8 transputers in a tree network. Matthews and others present work aimed at using parallel processing to quickly compute and display dose in [9]. Their algorithm however is not accurate enough for a final dose calculation. Finally, Alderson et al. present a parallel pencil-beam redefinition algorithm that is able to achieve a 5.56x speedup by using a 12 processor cluster over a single processor [10].

In order to achieve the desired dose accuracy, we opt to employ floating point processing in our implementation. With remarkable advances in FPGA technology as well as the availability of several FPGA floating point libraries (e.g., [11], [12]), floating point processing is becoming more popular. These libraries have appeared due to the fact that floating point calculations are more efficient than fixed point calculations when the dynamic range of the numbers is large.

Underwood has compared FPGA and CPU past, current, and future floating point performance in [13]. This is done by selecting representative processors and FPGAs from different years and constructing a trend line to estimate

future performance. It was found that in the future FPGAs will be more efficient, in terms of peak MFLOPS (Millions of Floating Point Operations Per Second), than CPUs in performing all types of floating point operations and that currently they are capable of performing more MFLOPS than CPUs for most floating point operations.

III. System Overview

Figure 2 illustrates the major components that make up our implemented dose calculation system. The components are grouped according to their functionality. Four memory modules and three computation modules are included in the system. The controller handles the communication among these individual components.

The memory modules are used to store the constants needed by the calculation. A major portion of the constants are used in the R^p_{mn} or R^s_{mn} calculations. The x, y and z memory modules contain direction vectors and voxel sizes, and are used chiefly in the ray tracing calculations to be described below. The direction memories are realized by on-chip embedded memory while the "Memory Subsystem" is implemented with off-chip DRAM. Details of the computation modules will be given in the following section.

All of the components have been implemented in VHDL and combined using Handel-C from Celoxica [14]. Channels provided in Handle-C are used to communicate between the components. Channels are an object to synchronize processing and communicate across objects and clock domains. In the case of our system, there is only one clock domain, but different pieces could be easily moved into different clock domains for processing. In this manner, it is a fairly simple task to synchronize the communication between different components and ensure correct functionality.

As can be seen from Figure 2, our dose calculation system currently has no interface with an outside system. In its current form, memory locations must be initialized with the required data prior to start of the calculation. Our ultimate goal is to develop an interface between a host computer and our FPGA board so that memory locations can be initialized by the host computer. Implementing the interface is our future work.

IV. Design Details

It seems tempting to directly convert a software implementation of the collapsed cone algorithm to an FPGA implementation. (In fact, such software was provided to us by researchers at the University of Maryland School of Medicine.) However, our initial analyses suggested that a straightforward conversion would have yield minimal speed improvements. In order to improve both computation efficiency and accuracy, we have made several modifications to the original collapsed cone algorithm. These modifications were mainly made in the R^p_{mn} and R^s_{mn} calculations presented in (2) and (3). The following subsections describe the new algorithm, as well as other implementation details.

A. R^p_{mn} Calculation

The original dose calculation method has been presented as Equations (1) and (2). Unfortunately, the use of these equations makes the addition of dose correction factors difficult. The main dose correction factors of interest are kernel tilting and kernel hardening, the exclusion of which can have a detrimental effect on the accuracy of the final result of the calculation as shown in [15]. A major reason that these correction factors are difficult to implement is the requirement of a constant Δr due to the parallel lattice of lines needed for the original equations. The reason for this requirement of a constant Δr will be discussed later.

We have modified the original collapsed cone algorithm such that a constant Δr is no longer required. This enables us to start a calculation from each voxel v and to easily perform kernel tilting by providing different A_m and a_m values as needed by v. In our modified algorithm, the cones are extended from each voxel with non-zero TERMA value. TERMA indicates the energy released, and the kernel describes how that energy is dissipated from the interaction site. Therefore, if a voxel contains no TERMA, then there is no reason to ray trace from it as there will be no energy deposited. Algorithm 1 presents the key procedure for calculating the primary energy distribution.

Procedure `Primary_Dose` is called iteratively and the number of iterations depends on the voxel size, which varies depending on both the clinical software used and the region of interest within a patient. With this algorithm, there are more variables that must be tracked throughout the calculation. However, it enables a much more flexible implementation than (1) and (2). There are many input values to the modified algorithm. $T_i, \rho_i, \Omega_{mn}, \Delta_r, \eta_i, A_m,$ and a_m represent the same parameters as in [1]. Added to the algorithm are p_diff, which represents a difference value from the previous iteration, num_η showing the total number of iterations performed, sum_η giving the previous sum of the η values and p_sum which represents the previous radiological distance. These additional variables are used in order to perform the algorithm with a different Δr for each iteration. They are used to keep track of quantities which are lost if only Equations (1) and (2) are employed. Equation (1) calculates the energy released within a cone. Our modified algorithm, in essence, calculates and saves the previous amount of energy released in this cone, calculates a new value for energy released and subtracts from it the previous value in order to obtain a more accurate dose than possible with Equations (1) and (2) when Δr is changing.

An explanation regarding why a constant Δr must be used for (1) and (2) is described here. Assuming that $T_i, \rho_i, \Omega_{mn}, \eta_i, A_m,$ and a_m remain constant, then only Δr may change. Here, assume that the original Δr_1 is large, i.e. $\Delta r_1 >> 1$ and the second Δr_2 is small, i.e.

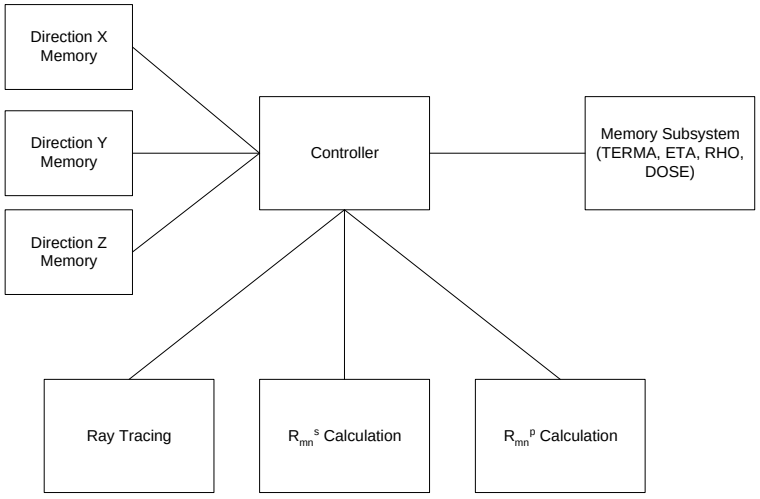

Fig. 2. System overview of the Collapse Cone implementation

Algorithm 1 Primary Energy Distribution Calculation

procedure: Primary_Dose($T_i, \rho_i, \Omega_{mn}, A_m, a_m, \eta_i, \Delta r_i,$ $p\_diff, p\_sum, num\_\eta, sum\_\eta$)
/*T_i: TERMA value, ρ_i: mass density, Ω_{mn}: Solid angle, A_m and a_m: exponential kernel values, η_i: electron volume density, Δr_i: distance traveled in voxel, p_diff: difference from previous iteration, num_η: number of iterations performed, sum_η: previous sum of η values, p_sum: previous radiological distance;*/
    $temp\_difference = (1 - e^{-a_m(\eta_i * \Delta r_i + p\_sum)})$;
    $p\_sum = p\_sum + \eta_i * \Delta r_i$;
    $difference = temp\_difference - p\_diff$;
    $p\_diff = temp\_difference$;
    $sum\_\eta = sum\_\eta + \eta_i$;
    $num\_\eta = num\_\eta + 1$
    $R^p_{mn} = \frac{sum\_\eta}{num\_\eta} * T_i * \rho_i * \Omega_{mn} * \frac{A_m}{a_m^2} * difference$;
    return $p\_diff, num\_\eta, sum\_\eta, p\_sum, R^p_{mn}$
end procedure

$\Delta r_2 << 1$. Then, the $(1-e^{-a_m \eta_i \Delta r})$ factor in Equation (1) is essentially $(1-e^{-\Delta r_1})$ or ≈ 1. So, the initial ΔR^p_{mn} used to calculate R^p_{mn} would be very large as most of the energy would have been deposited within the cone. Now, the exponential factor in Equation (2) is meant as an attenuation factor. So, if Δr_2 is small, such that the exponential factor is ≈ 1, then nearly the same amount of energy will be deposited in the region of Δr_2 as was deposited in Δr_1, which is not sensible. The same procedure can be repeated to show that the opposite effect occurs if Δr_1 is small and Δr_2 is large.

To see why our modification works, observe that by using 2 Δr's, Δr_1 and Δr_2 where $\Delta r_1 = \Delta r_2$, the equations will yield consistent results. That is, the same amount of energy will be deposited by Equations (1) and (2) as would be deposited by (1) assuming that a Δr_3 was used where $\Delta r_3 = \Delta r_1 + \Delta r_2$.

A block diagram for the R^p_{mn} calculation is given in Figure 3, where the circles represent functional blocks. The figure illustrates the initial inputs along with the different steps required to produce an output. Most of the functional blocks (such as addition/subtraction, multiplication and division) are realized by the corresponding pipelined modules provided by the Quartus design tool from Altera [16]. The entire implementation is fully pipelined with buffers inserted to balance the execution paths, and is capable of producing a result every clock cycle after an initial period of latency.

B. Modified R^s_{mn} Calculation

Equation (3) was modified also to be equivalent to that of Algorithm 1. The reason that (3) was more simple than (1) and (2) was due to the fact that scatter dose has less impact on the total dose than does primary dose. However, in the case of dedicated hardware, because it is fully pipelined, it requires the same amount of time to compute the scatter dose using the complex or the simple method. As the more complex method given in Algorithm 1 yields more accurate results, this method was chosen over a potentially simpler implementation.

C. Ray Tracing

The calculations used in the collapsed cone method require a Δr value as shown in Algorithm 1. This Δr value is the physical path length that a ray travels through a particular voxel. It is then used to determine a radiological path length. The ray tracing portion of the program is used to determine the physical path length.

The idea of ray tracing in our situation is to determine the distance a line travels between a starting point and its first intersection with a voxel wall boundary. That is, given

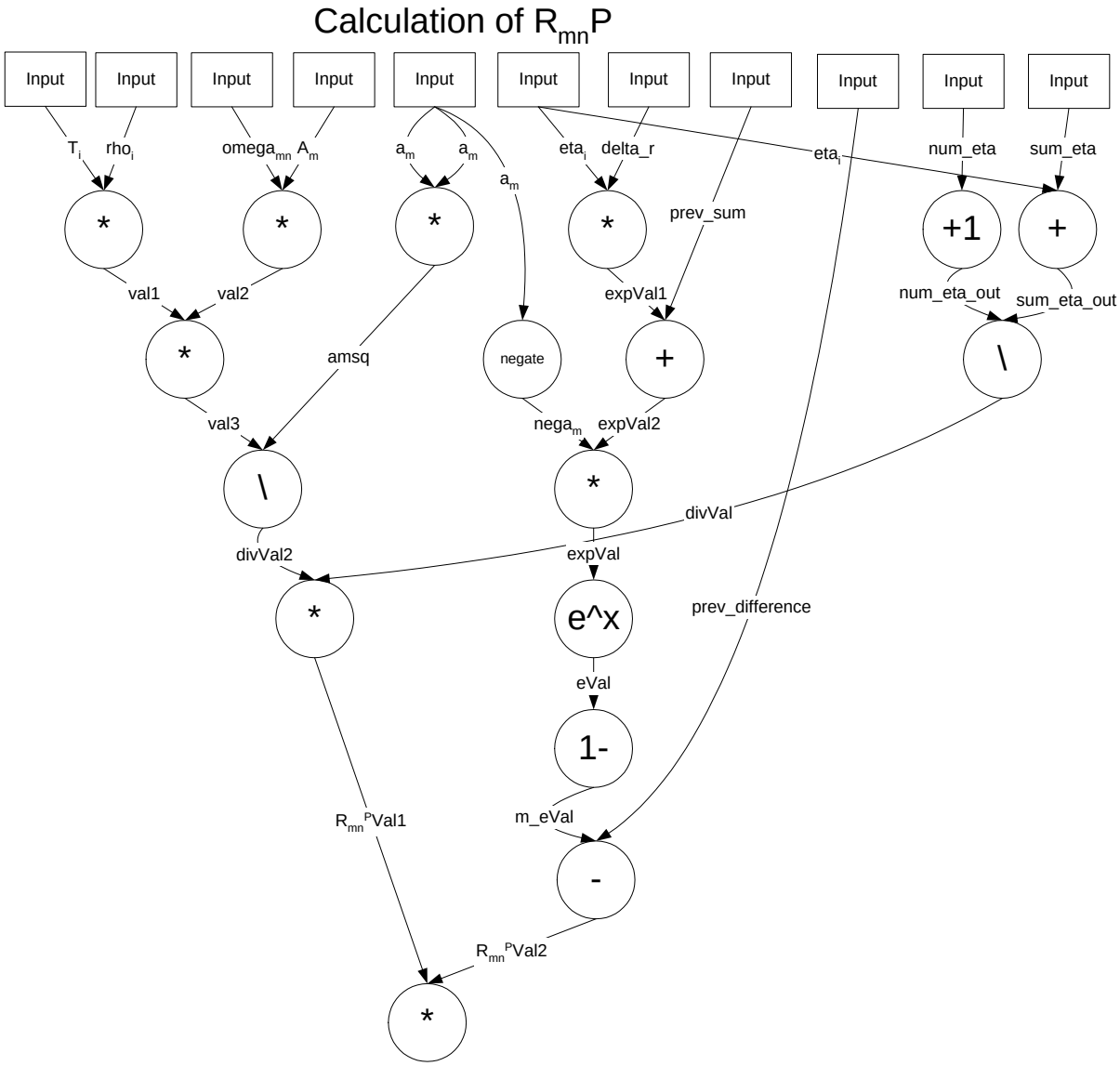

Fig. 3. Calculation of R^p_{mn}

a starting point $\overrightarrow{position}$ and direction vector $\overrightarrow{direction}$, a line l is created originating at $\overrightarrow{position}$ and traveling to infinity in the direction described by $\overrightarrow{direction}$. Next, a determination is made as to which voxel wall boundary has been crossed first by l. Calling this intersection point $\overrightarrow{intersection}$, it is then a simple calculation to determine the distance between the originating point $\overrightarrow{position}$ and the intersection point $\overrightarrow{intersection}$.

V. Memory Tradeoffs

Implementing the collapsed cone algorithm on an FPGA required a thorough examination of the memory usage of the algorithm and several tradeoffs between processing element usage and memory usage. Software from the University of Maryland School of Medicine that performed dose calculation was very memory intensive and a straight forward conversion of that algorithm to hardware would have yielded minimal speed improvements.

Several modifications were made that enabled the hardware algorithm to achieve its speedup over a traditional CPU. The modifications are: the use of an exponentially calculated kernel instead of table lookup, the use of a pipelined ray tracing module instead of precalculating values and using a large table lookup, and the use of several small on-chip memories to store values. The following subsections present information relating to the choices of these modifications.

A. Exponential Kernel Calculation

Instead of performing a table lookup for the kernel calculation, which is typically done in many existing dose calculation systems, the use of an exponential kernel calculation first proposed in [1] was performed. In [1], Ahnesjö showed that a kernel calculated using an exponential function closely resembled that obtained by using table lookup. Most of the time, with modern CPUs, it is more efficient to lookup a value from a table than it is to perform a lot of additional computation, but with FPGAs that is not the case.

Our implementation of the exponential kernel calculation is based on the exponential function evaluation discussed in [17]. The authors of [17] presented the design and implementation of an IEEE-754 compliant exponential function on an FPGA. We modified the design in [17] to suit our application. Strict compliance with the IEEE-754 standard was removed due to foreknowledge regarding the use of the inputs in our application and the design was pipelined. Once a pipelined exponential function was available on an FPGA, it was a simple task to implement the exponential kernel calculation. With the hardware exponential kernel computation, additional costly external memory lookups were avoided.

B. Pipelined Ray Tracing

The reference software algorithm from the University of Maryland School of Medicine relied on table lookup in order to perform ray tracing. For that implementation, all ray tracing calculations are performed prior to the start of the dose calculation. When a value is needed, it is simply looked up from a table. This is not an efficient implementation on an FPGA due to the slow access of external memory on FPGAs.

Instead of precalculating and storing ray tracing values, it is more efficient to simply calculate a value when it is needed. We have designed a fully pipelined ray tracing module. Though this module adds some initial latency to the overall calculation, it does not affect the overall throughput at all. In this way, we were able to eliminate an unnecessary external memory access by replacing a table lookup with calculation.

C. Table Usage

The use of several small tables enabled the algorithm to obtain needed values in parallel from on-chip embedded memory blocks instead of resorting to loading them sequentially through external memory. One example of this is the direction x, y, and z memory used in conjunction with the ray tracing module. These memories designed such that they can be loaded in parallel. With such parallel data access, the ray tracing module is able to compute a new result every clock cycle. If a sequential external memory were used, the best that could be accomplished would be a new result every 3 cycles.

In order to reduce the impact of external memory access on system performance, we have examined several factors to determine which required data could be moved from the external memory to on-chip memory. First, the size of the matrices needed was a concern. Most modern FPGA chips contain no more than 9 Mb of on-chip memory. The matrices could only take up a small portion of this. Second, it needed to be shown that data brought into on-chip memory could be used in such a way as to indeed accelerate the computation of the algorithm. The direction x, y, and z memory meet both of these criteria. The rest of the data are from large matrices and hence are kept in external memory.

VI. Experimental Results

The three computation modules (ray tracing, R^p_{mn} and R^s_{mn}) as well as the controller have been implemented in VHDL. The computation modules have been successfully compiled using the Quartus software for the Stratix EP1S40 chip [18]. Each of these components is able to achieve a cycle time of 10 ns (100 MHz). All these modules have been individually downloaded to an FPGA board and tested to ensure correct functionality.

The memory subsystem for storing the various matrices, such as the TERMA, the η, and the ρ matrices turned out to be a design challenge. The size of a typical TERMA matrix can be as large as 100x100x255. Therefore, the memory subsystem cannot be placed on-chip. This necessitates the use of memory external to an FPGA chip. As there is limited bandwidth between the FPGA chip and an external memory, the algorithm cannot be run every clock cycle. Currently, initial calculations of the bandwidth of external memory indicate that a calculation could be performed every 2 clock cycles.

The integration of all the modules (memory, computation and control) was done in HandelC from Celoxica [14]. The hardware design for the entire algorithm can handle one new input every clock cycle. Unfortunately, the entire implementation of the collapsed cone algorithm cannot fit on the FPGA chip on the Stratix board [19] currently available to us.

In order to analyze the performance of our design, we resorted to simulation via ModelSim from Mentor Graphics [3]. The clock frequency for the entire algorithm was set to 50 MHz, which should be easily attainable based on compilation data obtained for each individual component. While the hardware can handle the input of new data every clock cycle, it is currently not feasible to get required data from memory every clock cycle. For the simulation data used, a new calculation was started every other clock cycle to allow for some latency in the memory accesses. Below, we present these experimental results.

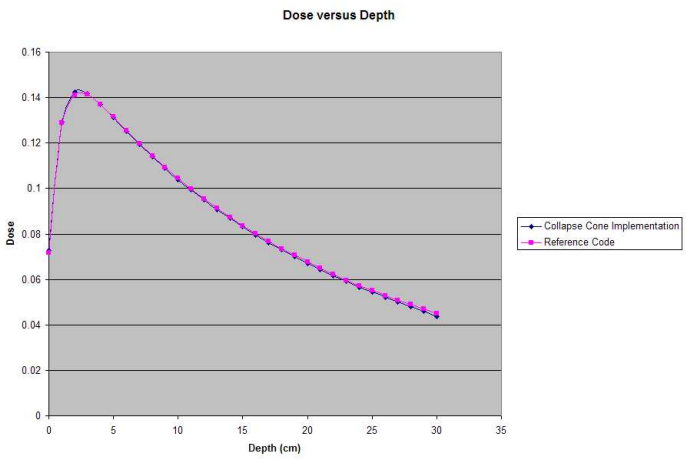

Fig. 4. Comparison of the dose computed by different dose calculation algorithms

A. Algorithm Validation

To verify our modified collapsed cone algorithm indeed calculates dose correctly, we implemented a software version of our algorithm and compared with the dose calculation software supplied by the University of Maryland School of Medicine. This effort was also indispensable for our hardware design and analysis.

Real patient data were used for testing. For the test cases investigated, our modified algorithm has a maximum off-axis error of 5.83% and an average error of only 1.15% compared to the software provided by the University of Maryland. Figure 4 presents a graph comparing our software collapsed cone implementation results with the reference code provided to us by the University of Maryland School of Medicine. The results are presented for the center axis of the phantom and show very close agreement between the two implementations. Due to the close correspondence of the two lines in Figure 4, the software was judged to be correctly implemented. The software was run on a computer with a 1.4GHz Intel Pentium M processor and 1 GB of RAM.

Our FPGA design performs exactly the same calculation as the software version of our modified collapsed cone algorithm. 32-bit floating point operations were used in the C++ software created and also in the Celoxica floating point functions provided. In this way, the accuracy of the hardware is ensured by the comparison work done for the software version. Furthermore, performance improvement can be analyzed strictly for different implementations to exclude the influence of algorithmic differences.

B. Dose Calculation Results

The TERMA calculation has been performed using the software from the University of Maryland. This calculation is not included in the processing time of any of the algorithms. Only the actual dose computation time is presented in these results.

Presented in Table I are the results of using the collapsed cone hardware and software. The number of cones computed for each voxel directly affects the number of calculations. If the number of cones increases by a factor of 2, the number of calculations increases proportionally. There is not an exact doubling due to differences in the number of voxels encountered during ray tracing. The number of cones is calculated as the number of radial divisions multiplied by the number of azimuthal divisions ($\Delta\phi$). The table presents how the number of calculations varies depending on the number of cones. It can been seen that the hardware is much more efficient at performing this calculation than the software implementation, which is to be expected.

In order to compute the required amount of time for the hardware implementation, simulation data were used. As the simulation was not fully tested on real hardware and was based on a simulated memory interface, the final timing information may change somewhat. However, ignoring initial latency, it requires 2 cycles to complete a result. Therefore, in order to obtain the hardware calculation time, a clock cycle rate of 20 ns was chosen (50 MHz) and final calculation time was based on that clock rate. Initial tests indicate that this cycle rate is reasonable.

The total computation time for the hardware implementation is shown in Equation (6). For the size of the calculations performed (the smallest requires 7 million computations) the setup time along with initial and final latency are negligible.

$$\text{Total Time} = \text{Setup Time} + \text{Initial Latency} + \frac{\text{Num\_Calculations}}{\text{Frequency}} + \text{Final Latency} \quad (6)$$

As noted in Table I, the computation time varies with the number of cones. Figure 5 shows this graphically. Also, the computation time varies as the size of the matrix changes. When the matrix is larger, there are more voxels that must be ray traces through and more voxels where dose is deposited. Figure 6 shows the impact that changing the matrix size has on the computation time. The x-axis lists total number of voxels and the graph illustrates how changing the number of voxels results in increased computation time.

C. Hardware v.s. Software Speedup Analysis

Table I has shown an impressive speedup of the hardware over the software implementation. We would like to examine in more detail the factors that attribute to this speedup.

Due to the algorithm differences between the software supplied by the University of Maryland School of Medicine and the collapsed cone implementation, comparisons between the two methods are made nearly impossible. In

TABLE I
CALCULATION TIMES FOR HARDWARE AND SOFTWARE COLLAPSED CONE IMPLEMENTATION

# of Cones	# Calculations	Time (s)		Avg. time/calc. (ns)	
		Software	Hardware	Software	Hardware
48	7,655,573	7.2	0.306	940.49	40
384	70,909,088	67.79	2.836	956.01	40
1152	209,969,888	209.52	8.399	997.86	40

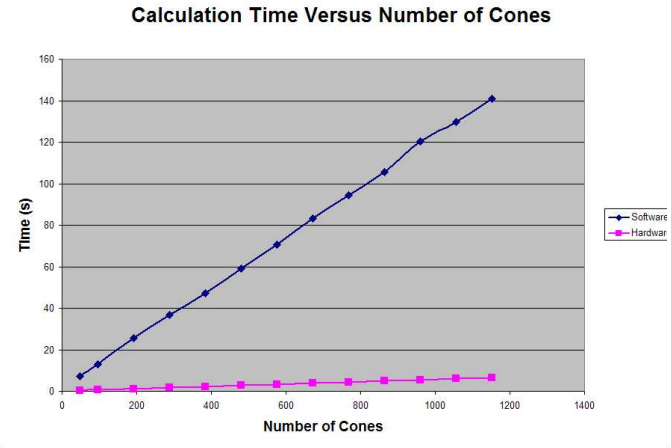

Fig. 5. Time required for hardware and software collapsed cone implementations as the number of cones is varied

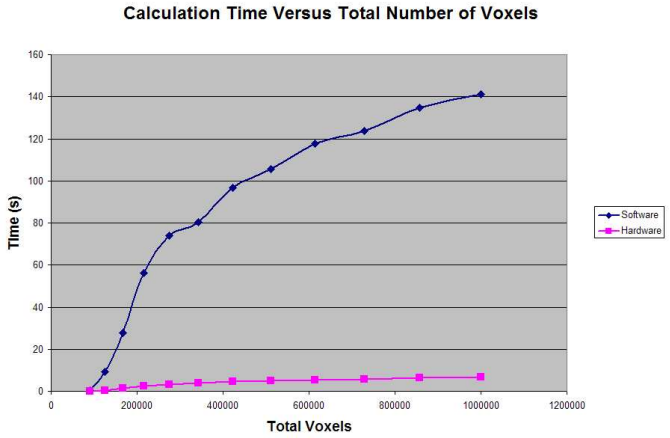

Fig. 6. Time required for hardware and software collapsed cone implementations as the matrix size is varied

the University of Maryland software, the dose is calculated from a dose deposition point of view, which is to say, the dose is only calculated for voxels within a region of interest and at no other locations. The collapsed cone hardware implementation is calculated from a dose interaction point of view, meaning that the calculation is performed originating at every non-zero TERMA value. There are advantages to each different method of dose calculation. For the dose interaction point of view, it is possible to determine the effect of changed beams on the dose without recalculating the entire beam [4]. The dose deposition point of view is more efficient if only a few voxels are required, but is at a disadvantage for a large number of voxels due to additional information that must be pre-calculated and stored. Because of these algorithm differences, we used the hardware and software versions of our modified collapsed cone algorithm are analyzed to conduct our speedup analysis.

To begin with, the maximum sustained floating point performance for both the hardware implementation and the CPU shows that some advantages are gained. According to [20], the maximum floating point performance in MFLOPS for an Intel Pentium M 1.6 GHz is 694.9. This was obtained by performing a matrix multiply that fit entirely in the processors cache. Calculations done for the hardware implementation show that it is capable of running at approximately 5600 MFLOPS. The number of floating point units of different types present in the hardware implementation are presented in Table II. The number of MFLOPS was obtained by assuming that each floating point unit ran at 50 MHz and the knowledge that each unit is completely pipelined. If $NumFP$ stands for the number of floating point units present and f represents the frequency they are running at, then total FLOPS is $NumFP * f$. Dividing the result by 10^6 yields MFLOPS. The difference in MFLOPS between the hardware and software implementations leads to a factor of 8 increase in the number of floating point operations which can be performed.

Another factor to examine is the instruction mix of the software. Using the Pin binary instrumentation tool provided by Intel and available at [21], the dynamic instruction counts of the software program were obtained. These instructions were then divided into 4 different categories, integer arithmetic, memory operations, floating

TABLE II
NUMBER AND TYPE OF FLOATING POINT UNITS IN COLLAPSED CONE HARDWARE IMPLEMENTATION

Unit Type	# Present
Multiply	45
Add	36
Subtract	17
Divide	13
Sqrt	1

point arithmetic, and control instructions. The breakdown of the percentages of total instructions executed by category is given below. The total number of dynamically executed instructions was 94,024,735,126 with 384 cones.

1) Integer Arithmetic ($IntOps$) - 12.1%
2) Memory Operations ($MemOps$) - 46.0%
3) Floating Point Arithmetic ($FPOps$) - 26.6%
4) Control ($ControlOps$) - 15.3%

Looking at the mix of instructions, with the hardware implementation, most of the memory operations are not of importance and neither are the control instructions. The memory operations can be ignored because they are overshadowed by computation. In a normal CPU, memory locations must be loaded before they can be used, but in this implementation, memory is loaded in parallel with computation, so while loads and stores do take place, they do not add to the overall computation time. If needed, these instructions can typically be shadowed by the parallel computation being done. If those are eliminated, then only 38.7% of the original instructions remain. If it is assumed that the integer operations can be run with the same increase in performance as the floating point operations (a factor of 8), the the overall speedup of the hardware over the software would be approximately $20.7\times$ and is shown in Equation (7).

$$\begin{aligned} Speedup &= 1/(\text{Original Computation Time} * \\ &\quad (100\% - MemOps - ControlOps)/8) \\ &= 1/(1 * (38.7\%)/8) \\ &= 20.7 \end{aligned} \tag{7}$$

Table I shows an overall speedup of between $23.51\times$ and $24.95\times$. The $20.7\times$ speedup is a reasonable estimation of the overall speedup.

VII. Summary and Future Work

We have implemented a FPGA-based system for implementation of radiation dose calculation. This method implements a collapsed cone convolution and has been shown to decrease computation time when compared to a commodity CPU. The presented method is efficient both in terms of number of calculations and in memory bandwidth. Such achievements can be of great benefit to improving the quality of radiation therapy.

This method shows the ability of new FPGA chips to handle a large amount of calculation element and successfully compute a complex software algorithm. In addition to providing a quicker collapsed cone convolution method, we have implemented the algorithm in FPGA using state of the art design technology. Much work was done in hardware design and in algorithm selection and modification for improving execution efficiency and required memory bandwidth.

In the future, we would like to obtain a new development board that would allow us to implement and test the entire algorithm in hardware. In addition, an interface with an existing commercial dose calculation engine needs to be designed would allow for thorough testing and would facilitate the use of this new technology. Also, the addition of kernel tilting to the current implementation would improve the accuracy and put us one step closer to obtaining a clinically viable dose calculation chip.

References

[1] A. Ahnesjo, "Collapsed cone convolution of radiant energy for photon dose calculation in heterogeneous media," *Medical Physics*, vol. 16, no. 4, pp. 577–592, Jul-Aug 1989.

[2] N. C. Institute, "Radiation therapy for cancer: Questions and answers," Online: Available http://www.cancer.gov/cancertopics/factsheet/ Therapy/radiation, 8 2004.

[3] M. Technology, Online: http://www.model.com.

[4] M. T. R., S. J. W., and B. J. J., "A convolution method of calculating dose for 15 MV x rays," *Medical Physics*, vol. 12, pp. 188–196, 1985.

[5] J. V. Siebers, "Monte carlo for radiation therapy dose calculations," Online: Available www.radonc.rdo.vcu.edu/AAPM, 2002.

[6] A. L. Boyer, R. Wackwitz, and E. Mok, "A comparison of the speeds of three convolution algorithms," *Medical Physics*, vol. 15, pp. 224–227, 1988.

[7] H. Liu, T. Mackie, and E. McCullough, "A dual source photon beam model used in convolution/superposition dose calculations for clinical megavoltage x-ray beams," *Medical Physics*, vol. 12, no. 24, pp. 1960–74, December 1997.

[8] D. C. Murray, P. W. Hoban, W. H. Round, I. D. Graham, and P. E. Metcalfe, "Superposition on a Multicomputer system," *Medical Physics*, vol. 18, no. 3, pp. 468–473, May 1991.

[9] J. W. Matthews, F. U. Rosenberger, W. R. Bosch, W. B. Harms, and J. A. Purdy, "Real-time 3D dose calculation and display: A tool for plan optimization," *Int. Journal of Rad. Onc.*, vol. 36, no. 1, pp. 159–165, August 96.

[10] P. Alderson, M. Wright, A. Jain, and R. Boyd, "Parallel pencil-beam redefi nition algorithm," *Lecture Notes in Computer Science*, vol. 2840, pp. 537–544, January 2003.

[11] P. Belanovic and M. Leeser, "A library of parameterized fbating-point modules and their use," *12th International Conference on Field-Programmable Logic and Applications*, pp. 657–666, 2002.

[12] A. Jaenicke and W. Luk, "Parameterised fbating-point arithmetic on FPGAs," *ICASSP IEEE INT CONF ACOUST SPEECH SIGNAL PROCESS PROC*, vol. 2, pp. 897–900, 2001.

[13] K. Underwood, "FPGAs vs. CPUs: Trends in peak fbating-point performance," *International Symposium on Field Programmable Gate Arrays*, vol. 12, pp. 171–180, 2004.

[14] Celoxica, "Dk design suite," http://www.celoxica.com/products/dk/.

[15] H. H. Liu, T. R. Mackie, and E. C. McCullough, "Correcting kernel tilting and hardening in convolution/superposition dose calculations for clinical divergent and polychromatic photon beams," *Medical Physics*, vol. 24, no. 11, pp. 1729–1741, November 1997.

[16] Altera, "Quartus ii design software," Online. Available http://www.altera.com/products/software/products/quartus2.

[17] H. Bui and S. Tahar, "Design and synthesis of an IEEE-754 exponential function," *IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 450–455, May 1999.

[18] Altera, "Stratix devices," http://www.altera.com/products/devices/stratix/.

[19] ——, "Nios II development kit, Stratix professional edition," Online. Available http://www.altera.com/products/devkits/altera/kit-nios_1S40.html.

[20] J. F. Carter, "Dell inspiron 6000d," Online: Available http://www.math.ucla.edu/ jimc/insp6000/p-proc.html, March 2005.

[21] Intel, "Pin," Online: Available http://rogue.colorado.edu/Pin/index.html, June 2005.