Fast, Power-Efficient Biophotonic Simulations for Cancer Treatment using FPGAs

Jeffrey Cassidy*, Lothar Lilge †, and Vaughn Betz*
* Edward S Rogers Sr Department of ECE, University of Toronto, Toronto, ON, Canada
† Princess Margaret Cancer Centre, Toronto, ON, Canada

Abstract—**Biophotonics, the study of light propagation through living tissue, is important for many medical applications ranging from imaging and detection through therapy for conditions such as cancer. Effective medical use of light depends on simulating its propagation through highly-scattering tissue. Monte Carlo simulation of photon migration has been adopted as the "gold standard" for its ability to capture complicated geometries and model all of the relevant problem physics. This accuracy and generality comes at a high computational cost, which limits the technique's utility.**

Greatly generalizing previous work, we present the first and only hardware-accelerated Monte Carlo biophotonic simulator that can accept complicated geometries described by tetrahedral meshes. Implemented on an Altera Stratix V FPGA, it achieves high performance (4x) and extremely high energy efficiency (67x) compared to a tightly-optimized multi-threaded CPU implementation, with demonstrated potential to expand the performance gains even further to 15-20x, which would enable important clinical and research applications.

I. Introduction and Outline

Light is useful for many medical applications. It is not inherently dangerous, as well as being inexpensive to produce, guide, and measure. Absorption spectra can be used to measure the composition and function of living tissue. Photochemical reactions like PDT (introduced below) can also be used to control biological processes. The ability to simulate the propagation of light through tissue is critical in all of these applications, for instance to calculate the tissue volume interrogated or the energy deposited by light.

This paper starts by giving a quick introduction to biophotonic simulations, highlighting the important features, and introducing Photodynamic Therapy as an application. Section III presents the basic algorithm used for Monte Carlo modeling of light propagation. Next, we present a summary of prior work on hardware acceleration of such simulations. Section V discusses computational features of the problem, and the reason for choosing FPGAs as an acceleration platform. We then discuss implementation details, area, and performance results. The paper concludes with a summary and directions for future work.

Our main contribution is an FPGA implementation of the algorithm, the first accelerated tetrahedral-mesh-based Monte Carlo simulator (GPU or FPGA). It achieves higher performance (4x) and power efficiency (67x) than a best-in-class tightly-optimized CPU implementation. We also summarize some important profiling results and analyze an architecture to enable still-higher performance on a single-FPGA system.

II. Background

A. Biophotonics

Fluence $\Phi(\mathbf{p})$ $[\mathrm{Jcm}^{-2}]$ is generally the quantity of interest in biophotonic applications, being the quantity received at a detector, imaged at a sensor, or deposited within a tissue for therapeutic purposes. Formally, it is the integral over time of the total light energy that has passed through an infinitesimal area at point $\mathbf{p}$. Calculation of fluence in heterogeneous tissues with complicated geometry is therefore an important tool in many medical applications for optically detecting, treating, and understanding disease.

Living tissues typically scatter light strongly, which is measured by the *scattering coefficient* μ_s, denoting the number of times a photon would typically be scattered per centimetre of path length ($\approx 10^1 - 10^3\,\mathrm{cm}^{-1}$). When a photon is scattered, its new direction is a random variable whose distribution depends on the material causing the scattering. Photons are also absorbed according to the absorption coefficient $\mu_a \approx 10^{-1} - 10^1\,\mathrm{cm}^{-1}$, generally less than the scattering coefficient.

While an approximate solution can be computed by the Finite Element Method [1] under some assumptions, Monte Carlo (MC) techniques are regarded as the "gold standard" for their ability to model the broadest range of materials, light sources, and geometries accurately. MC simulations work by launching a large number of photons, calculating their propagation according to the relevant statistical distributions, and tracking where they are absorbed. The method is computationally intensive, but has a regular structure and significant parallelism which make hardware acceleration attractive.

B. Inverse Problems

Many applications rely on solving mathematical *inverse problems* involving light propagation through tissue. The *forward problem* solved by the simulator presented here is to determine the distribution of fluence within the tissue given a description of the geometry, optical properties, and light emitters. In the inverse problem, we try to move "backwards" from a fluence distribution to the problem description that caused it. Generally, we wish to either find the pattern of light sources and tissue optical properties that best explain a measured light pattern, or we wish to create a certain light distribution within the tissue for therapeutic purposes. There are many degrees of freedom in terms of optical properties, source, and detector locations; closed-form solutions are not

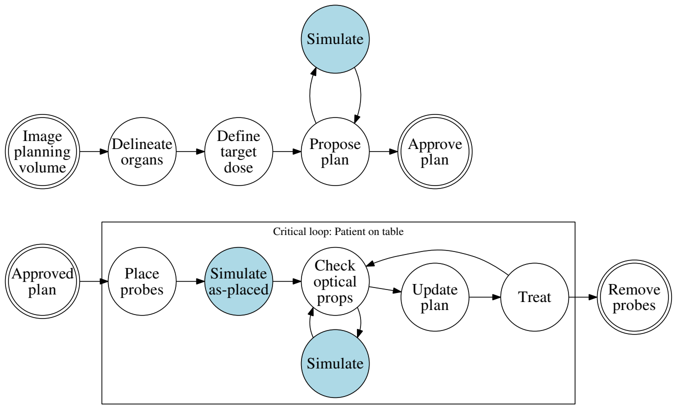

Fig. 1. IPDT workflow: planning (upper) requires $\approx 10^3$ iterations; on-line adjustment (lower) requires 10s of iterations, but is latency-critical

known for general cases, requiring iterative solutions using many forward simulations. This computational burden can render otherwise promising techniques such as Photodynamic Therapy (PDT) for cancer infeasible, which motivates this work.

C. Photodynamic Therapy (PDT)

Photodynamic therapy (PDT) is a light-mediated therapy used to kill diseased cells including cancer and bacterial infections. It uses a light-activated drug called a *photosensitizer* (PS) which, when it absorbs a photon, excites the oxygen normally present in tissue into a reactive form that damages cells. Oxygen radical production depends on the concentration of PS and oxygen in the tissues, and on the fluence; if the accumulated damage is sufficient, the cells in the immediate vicinity of light exposure die. Light may be delivered by superficial illumination (skin), through an endoscope (mouth and esophagus), during surgery, or using optical fibres inserted via needles. When optical fibres are inserted via needles and used to emit light within the tissue itself, the technique is known as *interstitial PDT* (IPDT). Treatment planning, as discussed in the next section, is critical to the safety and effectiveness of IPDT and remains a key obstacle to widespread adoption that can be addressed by faster, more accurate simulations. Further details about PDT are presented in a review by Wilson [2].

D. PDT Treatment Planning

To produce a PDT plan which is both safe and effective, it is necessary to propose and evaluate a number of candidate plans against physician-defined dose targets. The workflow occurs in two phases as illustrated in Fig 1. In the first phase, off-line planning, the region to be treated is imaged, minimum and maximum dose targets are defined, and many candidate plans are simulated with differing source configurations. Once an acceptable plan is generated, treatment can begin. The second phase, on-line adjustment, occurs when the patient is immobilized and the probes are inserted. Actual probe placement may differ from the simulation, and patient optical properties will also vary from those assumed in the planning phase. Tens of simulations may then be required to ensure safety given the as-placed locations and to optimize treatment for the patient's actual optical properties.

Evaluating the fluence distribution for a single PDT treatment plan is very computationally expensive. Achieving acceptable result variance requires simulation of $\approx 10^7 - 10^9$ photon packets, each of which undergoes hundreds or thousands of interactions before being retired. During the offline planning phase, approximately one thousand candidate plans may need to be evaluated to optimize a patient's treatment. Each plan requires approximately thirty minutes to complete on a single computing node, even using a highly-optimized software simulator. Consequently, the total effort to compute a plan is on the order of 500 node-hours: over twenty node-days per patient. Since Monte Carlo simulations are inherently parallel, virtually arbitrary performance speedup can be achieved through accumulating the results of multiple parallel instances. Given a rack of two hundred nodes, the calculation can be done in a few hours *if cost, space, and power consumption are not constraints*. When considering the process for multiple patients per day, the number of nodes required becomes non-trivial (≈ 20 per daily patient). Furthermore, power and cooling costs are first-order concerns when operating a computing facility given that the nodes required for each daily patient will run near 100% capacity and consume multiple kilowatts.

In the on-line phase, the simulations are on the critical path between immobilizing the patient and the completion of treatment so latency is of the utmost importance. Tens of simulations will be required in the course of a treatment that should last no more than an hour, meaning that each simulation must be reduced from a half-hour to minutes at most (≈ 20x). The cost of patient discomfort and operating-room time argue the case for extremely high reliability, quick compute times, and independence from remote facilities or communication links, ie. a self-contained portable system. These desiderata argue for small physical size, low power consumption (thermal dissipation), manageable cost, and fast computing capability.

Given the foregoing concerns, throughput per-dollar, per-watt, and per-unit-space are as important as total throughput, especially for the on-line calculations. Unfortunately, CPU-based cluster computing can address throughput concerns only; the other figures of merit remain constant since cost measures scale with cluster size, and clusters are non-portable. We therefore seek a system which performs 20x faster than a high-performance CPU in a package which is portable, uses no exotic cooling, and can be powered from a standard wall socket: goals which are likely to be met by an FPGA.

III. Algorithm

The basic algorithm for biophotonic Monte Carlo simulations, called "hop, drop, spin", was first implemented in open-source software (MCML) for layered materials by Wang, Jacques, and Zheng [3]. While acceptable for some applications like skin, infinite planar layers are oversimplified for many anatomical structures. The algorithm generalizes naturally to more complex problems by increasing the complexity

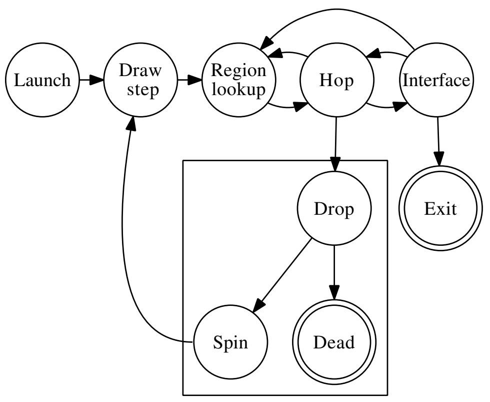

Fig. 2. Overview of Monte Carlo "hop, drop, spin" flow. This maps exactly to hardware functional blocks and data flows in our implementation.

of the geometry description along with related routines to find the intersection of a ray with region boundaries and calculate surface normals. Shen and Wang provided such a software implementation called TIM-OS [4], and we provide a faster, more flexible version called FullMonte [5] which is used to gather algorithm profiling information and to validate the hardware.

MC methods simulate photons in groups (called *packets*) which propagate together through the flow shown in Fig 2. Each packet is launched, travels in a straight line for an exponentially-distributed step length ("hop"), then interacts with the material. During that interaction, part of the packet energy (called *weight* w) is deposited ("drop"), the photon scatters into a new random direction ("spin") and the process repeats.

A. Launch

The packet's initial state is defined by a *source distribution*. In this work, we use an isotropic point source which radiates equally in all directions. The starting position and surrounding region are known constants and the direction $\hat{\mathbf{d}}$ is chosen as a random point on the unit sphere.

B. Draw Step

Based on the coefficients of scattering and absorption for the current region, μ_s, μ_a, the attenuation coefficient $\mu_t = \mu_s + \mu_a$ defines the average number of interactions a photon has per unit path length traveled. The length of a step s is exponentially distributed with mean $\frac{1}{\mu_t}$, and is generated from a uniform random variable r by

$$s = \frac{l}{\mu_t} = -\frac{\log r}{\mu_t} \quad r \sim \mathcal{U}_{01} \tag{1}$$

C. Region Lookup

One important novelty of our work is that we are the first hardware accelerator to support a tetrahedral mesh description, and hence the first which can approximate surface normals to general smooth curves properly - an essential feature for accuracy when dealing with reflection and refraction [6]. The problem geometry is represented as a set of tetrahedra, each of which is assigned a material ID. The set of points $\mathbf{p}$ inside a tetrahedron is defined by

$$\{\mathbf{p} : \hat{\mathbf{n}}_i \cdot \mathbf{p} - C_i \leq 0,\ i \in [1,4]\} \tag{2}$$

The description therefore consists of four faces and an index into the material-properties lookup table. Each face is defined by a normal vector $\hat{\mathbf{n}}_\mathbf{i}$, constant C_i, and the index of the adjacent tetrahedron.

D. Hop

To take a "hop", the packet is advanced by the drawn step length s along direction $\hat{\mathbf{d}}$ to its final position $\mathbf{p}' = \mathbf{p} + s\hat{\mathbf{d}}$. If it remains within the current tetrahedral region, then the hop is finished and the packet proceeds to "drop". Otherwise, the packet is passed to the interface logic which handles the boundary.

E. Interface

If the packet crosses a face during its hop, the intersection of the ray and the face crossed is calculated. In the refractive-index differs across the boundary, calculations to account for that interface must be done: Fresnel reflection gives a probability of reflection as a function of incidence angles and refractive indices; total internal reflection specifies an incidence angle above which reflection always occurs as a function of refractive index; and Snell's law specifies the angle of the transmitted ray if it does not reflect. If the scattering and absorption coefficients change, then the remaining step length is updated to $s' = \frac{\mu_t}{\mu_t'} s$ to preserve correct step-length statistics for the new interaction coefficient. If the photon passes beyond the edge of the mesh, ie. the adjacent tetra ID is 0, then we count it as having exited the mesh and stop tracing it.

F. Drop

At the conclusion of the step, the packet drops some of its energy to model the absorption probability. The *albedo* $\alpha = \frac{\mu_s}{\mu_s + \mu_a}, 0 \leq \alpha \leq 1$ gives the probability that a given interaction with the material will be scattering. In order to model the expected behavior of individual photons, the packet deposits weight $(1-\alpha)w$ in the region and continues onwards with decreased weight $w' = \alpha w$. The deposited weight is accumulated for that mesh element to calculate the fluence at the end of the simulation.

G. Dead

As the photon propagates and is fractionally absorbed at multiple steps, its weight declines so it contributes less to the absorption scores. The computational cost of tracking a packet is independent of its weight so the effort spent shows diminishing returns in terms of its impact on the accumulated fluence value. To avoid wasting effort, packets with a weight less than a constant $w_{\min}$ undergo a self-termination process called "Russian roulette" in which they are given a 1-in-P chance of surviving with weight Pw, otherwise terminating. An average packet will undergo hundreds of hop-drop-spin cycles before expiring in this energy-conserving way.

H. Spin

Living tissues consist of scatterers with varying size and optical properties, so precise modeling of each scatterer is impractical. Instead the Henyey-Greenstein (HG) phase function is often used to model the distribution of aggregate behavior for the deflection angle θ between $\hat{\mathbf{d}}$ and $\hat{\mathbf{d}}'$. The azimuthal angle ψ is uniformly distributed. The HG function ICDF shown below takes a parameter $g = \mathbb{E}[\cos\theta]$ which describes how forward-biased the distribution is in terms of the correlation between the incident and outgoing directions.

$$\cos\theta = \frac{1}{2g}\left[1 + g^2 - \left(\frac{1-g^2}{1-g(2q-1)}\right)^2\right] \tag{3}$$

Direct evaluation for a uniform random value of q gives an appropriately-distributed value of $\cos\theta$. The sine is calculated using $\sqrt{1-\cos^2\theta}$.

IV. Prior Work: GPU and FPGA Acceleration

A. Software

A number of different software models have been created over time with varying simplifications. Only MC simulators based on a tetrahedral mesh geometry are sufficiently general to deal accurately with complicated geometries including curves and refractive index changes. Of those, our previous work FullMonte-SW [5] runs slightly faster than the next-fastest such software, TIM-OS [4] while supporting the same problem definitions. Prior acceleration work exists as summarized below, but it is insufficiently general for IPDT planning.

Software implementations vary greatly in their level of optimization, requiring caution when interpreting reported speedup results for accelerated versions. In our experience, the speedup of running multi-threaded simulations is linear in the number of physical cores, and Simultaneous Multi-Threading[1] also offers a significant benefit. Single-threaded versions are therefore inherently at least 5-10x slower than potential multi-threaded performance on modern processors with 4-8 cores. During the evolution of FullMonte-SW, we also noted a performance difference of 2-3x due to optimizations in data structures and use of SIMD intrinsics. Unoptimized single-threaded software may therefore be 10-30x slower than an optimized implementation.

B. GPU

Fang and Boas [7] present MCX, a GPU-based simulator, for propagation within a voxelized geometry representation which has well-known drawbacks in representing curved surfaces [6] and offers no way to vary the level of detail throughout the simulation volume. The resulting implementation is unnecessarily memory-bandwidth-intensive, compute-intensive, and inflexible in its geometry representation.

Alerstam [8] implemented CUDAMC on a GPU, for a greatly simplified problem definition: semi-infinite geometry (the half-space below $z = 0$), non-absorbing (no requirement for atomic read-accumulate-write of fluence), and homogeneous (single material). That implementation achieved 1000x speedup against an unoptimized single-threaded program, which should be adjusted to approximately 30-100x.

[1]Intel "HyperThreading", in which multiple threads execute concurrently sharing a single computing core

Alerstam and Lo [9] and Lo [10] present the related packages CUDAMCML and GPU-MCML, both of which achieve approximately 100x speedup for a more complicated radially symmetric multi-layered model which, unlike CUDAMC, scores absorption and permits differing materials. The benchmark comparison is again a single-threaded unoptimized C implementation, so a fair estimate would place the speedup versus an optimized software implementation at 3-10x. In exchange for a modest increase in geometry complexity and output data versus CUDAMC, the GPU-CPU performance ratio has dropped by an order of magnitude. The geometry model is still simple: implicit $(+z)$ surface normals, and both the geometry description and the absorption grid are small.

C. FPGA

In a different vein, Lo [10] created FBM, an implementation of MCML on an Altera Stratix III FPGA, permitting up to 10 layers and a cylindrically-symmetric fluence distribution $\Phi(r, z)$ of up to 256x256 elements. He reported an advantage of 45x in speed and 700x in power efficiency over a CPU of the same process node, which provide a basis for comparison in the results of Sec VII. Adjusting for the unoptimized single-thread baseline, the speed difference is closer to 1.5-3x.

V. Choice of Accelerator Technology

We optimized our reference software code extensively using SIMD instructions, multithreading, and compact data structures. While some incremental improvements are possible, software implementations remain more than an order of magnitude away from the necessary performance defined in Sec II-D, so alternative platforms must be considered. To choose a platform, we examined the features of the algorithm and made the following observations:

1) Fixed dynamic range: The relevant problem variables have a well-defined range, consisting of bounded variables, unit vectors, and positions within a defined cube.

2) Limited precision needs: Precision needs are modest; single-precision float (24b mantissa) results were indistinguishable from double (54b) in software (Sec VII-A). Validation of the hardware implementation has shown further that 18b fixed point is sufficient. Each packet terminates after a few hundred steps so little error accumulates, unlike the numerical stability issues seen in iterative calculations.

3) Data parallelism: There exists nearly infinite data parallelism since each photon packet is independent of all others. The same results are expected from running N packets on one processor as from summing M parallel simulations of $\frac{N}{M}$ packets each with different random seeds.

4) Pipeline parallelism: The algorithm (Fig 2) exhibits significant pipeline parallelism, with well-defined operations, few branches, and no inter-packet dependences.

5) Unique memory structure: As discussed further in Sec VI-A, the problem involves a mix of large read-only geometry memory, medium-sized read-accumulate-write fluence memories, and small read-only constant memories. The ability to exploit the particular sizing, access patterns, and aspect ratios of the memories is important for scalability.

6) Special Functions: At several points in the algorithm, special functions (sine, cosine, square-root, division) are required. Due to the limited precision and dynamic range, it is possible to trade quality for speed in the implementation.

7) Atomic accesses: Some implementations simplify by not scoring absorbed energy, which is not appropriate in this case. Fluence scoring requires frequent atomic read-accumulate-write access to relatively large arrays ($\approx 10^4 - 10^6$ elements).

8) High memory access requirements: The present problem formulation is distinguished from previous accelerated work by its large memory access requirements. When dealing with semi-infinite or planar media, the problem description is on the order of kilobits. The tetrahedral meshes required to handle real-life geometries are 3-5 orders of magnitude larger.

9) Random number generation: As a Monte Carlo method, random number generation (RNG) is an essential step in the process. FPGAs excel at generating and manipulating random numbers of varying precision using bitwise operations.

Some factors listed above apply equally well to GPU and FPGA acceleration, however there are several significant factors which favour FPGAs uniquely. Given the limited precision needs and fixed dynamic range, fixed-point arithmetic can be used which offers significant savings in area and power on an FPGA. It also allows the instantiation of custom special-function blocks with the right precision and in the exact quantity required, in contrast to GPUs where special functions are slower, overly accurate, and less plentiful [11]. The relatively straight-forward data flow graph of Fig 2 suggests an efficient spatial layout, which saves power and area for routing data.

Due to its simplicity and negligible memory access needs, CUDAMC is fully compute-bound on a GPU and so gives a hard upper bound on speedup. CUDAMCML's layers introduce the need for a small constant array to describe the geometry, which can easily fit into per-processor shared memory and so geometry storage remains trivial. Since it outputs the spatial distribution of accumulated energy, however, it introduces a modest global memory requirement for accessing the absorption array which does not fit in shared memory.

Generalizing to a tetrahedral mesh on a GPU would add an entirely new memory access requirement since the size of the geometry description would increase by 3-4 orders of magnitude. That is a hugely important difference between fitting in local shared memory (64kB) and requiring access to the large, relatively slow main memory. Operating at peak efficiency, modern GPUs can provide hundreds of gigabytes per second in memory access, *if* accesses are coalesced into groups of 16 between adjacent threads. Given that a packet's location at a given moment is uncorrelated with any other packet, and that the access granularity is on the order of a few hundred bits, a very high proportion of bandwidth waste is probable. The situation becomes even worse when attempting to score absorption, where access granularity is just 64 bits for atomic read-accumulate-write operations to a large data set.

In contrast, modern FPGAs offer thousands of configurable RAM blocks with a variety of width, depth, and port configurations providing massive internal memory bandwidth in addition to plentiful and efficient computing resources. We propose and analyze a highly efficient custom memory system to exploit that configurability in Sec VII-D.

VI. Design Details

The algorithm of Sec III was implemented in an Altera Stratix V GS A7 chip (-C1 fastest speed grade). Architecturally, the design is a direct implementation of the algorithm flow shown in Fig 2, with one hardware block for each graph node. All compute units except the rarely-used launcher are fully unrolled and pipelined to accept a new input on each clock. Inter-packet parallelism is exploited by keeping multiple packets in-flight at once. We used the Bluespec SystemVerilog (BSV) high-level synthesis language, which excels at control-intensive designs, making writing and validating the high-level flow relatively simple.

Our current implementation focuses on performance-critical issues, placing two restrictions on the model. First, only 48k tetrahedral elements are supported in order to fit in on-chip Block RAM. For comparison, a commonly-used open-source mesh description of a mouse often used for imaging experiments, Digimouse [12], contains slightly over 300k elements to model the entire animal. Testing showed that over 95% of the absorption events covering 99+% of absorbed energy occur within the 48k most-frequently accessed elements. Such a limitation on mesh size therefore still permits useful results for clinical applications, and provides a meaningful generalization both in size (48k elements vs 10 layers) and type (tetrahedral mesh vs planar layers) over previous accelerated work. We are working on resolving this limitation without detracting from performance, and present analysis that shows it can be done using a memory hierarchy including off-chip DRAM.

Second, we do not currently handle refractive interfaces. While important to give accurate results, profiling of the software algorithm showed that packets encounter interfaces $10^2 - 10^3$x less often than they undergo intersection testing and scattering. Some resources will of course be necessary to accommodate such calculations, but due to its rarity the area requirements can be minimized through operator sharing. Our work has thus far focused on the performance-critical parts of the problem and demonstrating feasibility of the accelerated calculation. In our view, the essential challenges are memory access and the calculations for intersection testing and scattering. Development in progress suggests that the operator can be supported without degrading performance (throughput, $F_{\max}$) using 1-2 DSP and a few thousand ALMs. There is sufficient packet pipeline parallelism to avoid stalling the main pipeline, and the throughput requirement is low enough to permit sharing between multiple pipelines.

Dataset	Type	Bits per el.	Total size #El	Total size Bits
Geometry storage	RO	404	48k	18.5M
Volume accumulation	RW	64	48k	2.9M
Surface accumulation	RW	64	4.8k	320k
Material properties	RO	128	16	2k

TABLE I
MEMORY ACCESS REQUIREMENTS

Since hop and spin are very resource-intensive, the design keeps them as close to 100% occupied as possible. Packets which are killed in roulette are immediately replaced at the draw-step stage by a newly-launched packet to ensure that the "hop" core never stalls. All functional blocks have fixed latency since on-chip BRAM is used for all storage, however the branch decisions depend on random variables. Where paths merge (eg. region-lookup), all but one of the paths are queued while the other is given priority. Simulation assertions were used to ensure the queues never overflow.

A. *Memory Access*

The datasets accessed by the algorithm are summarized in Table I. Data storage is the single largest difference between the current and previous acceleration work. Though the geometry is far more general, it only adds marginal complexity to the propagation calculations, but its demands on storage are much higher (18.5Mbit, nearly half of available RAM). Each tetrahedron is described by a material ID and four face descriptions (see Sec III-C). Walking the mesh requires testing for ray-plane intersection to determine which of the four faces is crossed, followed by loading the adjacent tetrahedron corresponding to the index stored in the face description.

After geometry lookup, absorption accumulation is the most frequent operation (nearly every clock), and also the largest read-write data set. It is stored in M20k block RAM by reading, adding, and writing back in lock step for each datum received. Surface accumulation works similarly, though the array is smaller by about 10x and less frequently accessed ($\approx$ 100x) so it is not a major performance or area issue.

Since there are only a few properties to define the material (μ_s, μ_a, g, n), and 16 materials are sufficient for very general applications, the material properties are stored in MLABs near their point of use. Different properties are needed at different stages of calculation, so small distributed stores make sense.

B. *Random Number Generation*

Uniform $[0, 1)$ pseudo-random numbers are generated using the 800-bit Tiny Twister (TT800) generator of Saito and Matsumoto [13], using their open-source C code as a reference. A very fast and lightweight parallel implementation produces 800 random bits per clock in parallel at rates exceeding 500 MHz using only logic cells and registers.

To reduce the latency of the main loop, distributed random numbers are pre-calculated and queued to hide calculation latency. This reduces the number of delay balancing registers in the main loop, and the number of in-flight packets required to avoid pipeline stalling.

Unit vectors are generated using standard methods. For the 2D case, $(\cos 2\pi\psi, \sin 2\pi\psi)$ are evaluated using the standard CORDIC method [14] for uniform random ψ. To get a 3D unit vector, a 2D unit vector in the xy-plane is rotated towards the $+z$ direction by ϕ where $\cos\phi$ is uniform on $[0, 1]$.

The step-length random variable is calculated by taking the logarithm of a uniform random variable. Further savings are achieved by taking the base-2 logarithm and scaling the stored coefficients appropriately. Calculation of $\log_2 x$ is done using bit-shifts and a Taylor expansion for $\log(1 + y)$.

C. *Intersection Testing*

Intersection testing is performed by direct evaluation of the four conditions of Eq 2, which fits naturally into the 18x18 variable-precision DSP blocks in the Stratix V architecture.

D. *Spin*

Deflection $(\sin\theta, \cos\theta)$ scattering angles are calculated from uniform random numbers according to the Henyey-Greenstein inverse CDF (Eq 3) to 18 bit precision each. To minimize inner-loop latency, values are calculated and stored in a queue (one for each of 16 distinct materials) in advance of being requested. As values are dequeued, a new random variable is drawn and a calculation is launched to replenish the queue; since queue size exceeds calculation latency, a value is always available. A single shared queue is maintained for the azimuthal angle ψ. In the original software implementation, division and square-root operations are required to calculate the outgoing direction vector. The hardware version reduces latency and eliminates square-root and division entirely by carrying two extra column vectors $\hat{\mathbf{a}}, \hat{\mathbf{b}}$ orthonormal to the packet direction so that the spin may be calculated as a 3x3 matrix multiplication using fast, low-latency, plentiful, power-efficient hard multipliers by

$$\left[\hat{\mathbf{d}}\ \hat{\mathbf{a}}\ \hat{\mathbf{b}}\right]^T = \begin{bmatrix} \cos\theta & -\sin\theta\cos\psi & \sin\theta\sin\psi \\ \sin\theta & \cos\theta\cos\psi & -\cos\theta\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \left[\hat{\mathbf{d}}\ \hat{\mathbf{a}}\ \hat{\mathbf{b}}\right]^T \tag{4}$$

VII. RESULTS

The results discussion below is split into four sections: validation, area, speed and energy efficiency, and scale-up architecture. We simulated and debugged the design using Bluespec's Bluesim simulator. Performance and area results were produced by running the Verilog code produced by Bluespec through Altera Quartus II place-and-route targeting a Stratix V 5SGXMA7N1F45C1 28nm FPGA.

A. *Validation*

For validation, we used the "cube_5med" test geometry from the TIM-OS test suite [4]. It consists of a cube with five layers of differing optical parameters (μ_s, μ_a, g, n), which we adjusted to match the hardware simulator's capability by making the index of refraction homogeneous. Our own software implementation was used as a reference, which was

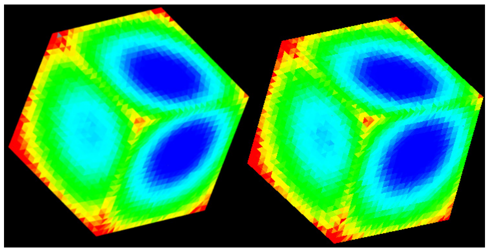

Fig. 3. Validation output for the "cube_5med" test case (L: hardware 1.6M packets, R: software 100M packets)

itself validated [5] against TIM-OS. We used the Bluespec Bluesim cycle-accurate hardware simulator to run 1.6 million packets ($\approx 10^9$ inner loop iterations) in 18 hours[2]. Both qualitative (Fig 3) and quantitative comparison showed no meaningful difference in output results (mean or variance) for surface or volume fluence. As intended, we achieved 100% utilization of the intersection-test core, and over 70% utilization of the scatter core which are the most resource-intensive functional blocks.

B. Area

The hardware resources required and maximum frequency for the principal blocks synthesized in isolation, and for a complete single pipeline instance are shown in Table II. Mesh storage and absorption/exit accumulation were not synthesized in isolation since they are not significant consumers of resources aside from BRAM. Step finishing is part of the main loop code and so is itemized only to explain the DSP count. The total resource utilization exceeds the sum of the parts because the main module contains glue logic, pipeline delays, and queuing which cannot be separated out. Physical synthesis also implements some resource duplication to improve performance. In the case of block RAM, the need for duplication to achieve a high clock rate could be reduced by additional pipelining. Only one of the two available ports is used to access the geometry ROM, so in the future two pipeline instances could share storage.

The design achieves high clock rate and area-efficiency by explicit instantiation of hard blocks for DSP functions such as multiply-add to compute dot products. On the -A7 device, DSP and Block RAM availability limit the number of instances to four.

C. Performance & Power Efficiency

In software testing, we found that the number of Million INTersection Tests per Second ("Mints") limits performance across a wide range of geometries and material properties. Intersection testing requires accessing the deepest and widest array in the algorithm (the geometry storage), as well as significant calculation. By design, the hardware implementation is also Mints-limited since the intersection-test ("hop") block of Fig 2 is on all cycles of the flow graph.

[2]Approximately 400x slower than the C++ reference, but providing bit- and cycle-accurate hardware queuing, flow control, and arithmetic

Functional block	$F_{\max}$ MHz	ALM	FF	DSP	M20k BRAM
Point source	290	1792	2014	2	2
Henyey-Greenstein	364	1740	2857	4	0
Scatter	302	280	546	19	0
TT800 RNG	590	804	800	0	0
Intersection test	329	510	799	20	0
Boundary	340	1707	2713	5	2
Step finish	*			3	0
Mesh storage	*			0	1034
Fluence accumulation	*			0	211
Total	280	16271	29154	59	1265
% of Available		7%	6%	23%	49%

* Not synthesized individually; no isolated $F_{\max}$ available

TABLE II
RESOURCES AND $F_{\max}$ FOR SINGLE INSTANCE ON STRATIX V A7

	Power (W)	Relative Speed	Relative Energy/pkt
CPU	76	1.0	67.5
Single-instance Stratix V	4.5	4.0	1.0
Estimated 4 instances	13.9	16.0	0.77

TABLE III
PERFORMANCE AND ENERGY-EFFICIENCY COMPARISON (FPGA VS CPU)

The FPGA (28nm technology) achieves 280 Mints performance at the maximum clock rate of 280 MHz. Power draw was estimated at 4.5W using Altera's post-synthesis vectorless power estimation and a 12.5% input toggle rate. To produce the estimate for four instances, the core power (excluding I/O) was quadrupled which explains the higher efficiency value.

Unlike prior works which used a single-threaded unoptimized software baseline, we use our own tightly-optimized multi-threaded best-in-class C++ software package [5]. Optimizations include using single-precision floating-point where appropriate, data structure packing to maximize cache performance, use of approximate math instructions, best-in-class libraries, and explicit use of Intel SSE vector instructions. Multiple methods were explored for critical sections including intersection testing and scattering calculations. We believe that only marginal improvements can be made through further software optimization. The CPU used for comparison is an Intel i7-2600K "Sandy Bridge" (32nm) with a thermal design power (TDP) rating of 95W. For a power-efficiency comparison, we assumed conservatively 80% of TDP (76W). It achieved 70 Mints on the "cube_5med" benchmark when compiled with all non-essential output disabled and full optimizations.

As shown in Table III, our FPGA implementation is 4x faster and conservatively 67x more power-efficient than a highly-optimized CPU implementation. Lo's FBM [10] achieved over 700x advantage in power efficiency using an FPGA versus a processor that were both two generations older. We note three factors to explain the difference in power-efficiency comparisons: first, the current problem definition is far more complex, with a working set over 10x as large (RAM is the leading dynamic power drain); second, our processor baseline is far more carefully optimized; and third, processor energy efficiency has improved since FBM was built.

D. Scaling Up

We have also investigated memory architectures for scaling the system up to more parallel pipelines and larger meshes. Based on memory traces gathered from the software simulator for various problem definitions, there is little temporal reuse beyond the 4-8 most recently used elements ($\approx 60\%$ hit rate). It would therefore be appropriate to use an eight-element first-level (L1) cache with LRU (least recently used) eviction policy using 11 BRAMs. Beyond that, reuse in time is low but access frequency by address is highly non-uniform, giving a Zipf-like [15] distribution. Because the distribution remains static over a simulation run, L1 misses would be served well by a hierarchy of *static* caches based on access frequency. A simulation would start with some naive cache set or best guess and assign elements to the appropriate cache level after a brief warm-up run to determine the distribution. Since the L1 hit rate exceeds 50%, two L1 caches can share a single L2 cache read port. Each L2 cache (4k elements, 88 BRAM) provides two read ports, so two L2 blocks could serve eight L1. Since each L2 has a hit rate over 50%, two of those could share an L3 cache (32k elements, 704 BRAM, 80% hit), with the remaining 5% of accesses served by off-chip DRAM. Such a scheme could handle a very large (>1M elements) mesh and keep 8 pipelines saturated using only 968 BRAM, which is *less* than the current system. The performance limit due to memory would then be $8F_{\max}$ tetrahedra per second, 25-35x faster than a CPU if sufficient computing pipelines could be instantiated (current limit is 4 based on DSPs for A7 device).

VIII. Conclusions and Future Work

We have demonstrated the first hardware-accelerated (FPGA or GPU) implementation of a tetrahedral mesh Monte Carlo biophotonic simulator. It can accommodate a mesh of up to 48k elements, and has been validated and benchmarked against highly-optimized software. It offers advantages in performance (4x) and power efficiency (67x), and demonstrates the feasibility of achieving the performance goals described in Sec II-D as necessary for widespread adoption of IPDT.

Since each pipeline instance requires slightly under one quarter of the chip, the existing prototype can be replicated four times within a Stratix V A7 chip to yield 4x the performance. We are therefore confident of at least 16x performance against a CPU, and additional tuning (pipelining, logic-arithmetic optimization) should yield further incremental speedup. The profiling and architecture design presented here (greater detail in [16]) have also shown convincing potential for more performance improvement and extension to larger meshes. Some development effort remains to realize a full system interfaced over PCIe, but the difficult research questions have been answered leaving very little technical uncertainty.

In creating the software implementation of FullMonte, we created a significant suite of profiling tools to analyze the memory access patterns. Using that data, we have designed and analyzed a custom memory hierarchy using both on- and off-chip memory (SRAM and DRAM) to accommodate larger meshes, eliminating the 48k element restriction.

Lastly, we intend to use the simulator for IPDT treatment plan evaluation with a goal towards creating an iterative treatment plan optimization algorithm. In collaboration with Princess Margaret Cancer Centre and Sunnybrook Hospital, we plan to use the created software and hardware simulators for other biophotonic research as appropriate including medical-device design, imaging system analysis, and bioluminescence tomography (BLT).

Acknowledgements

The authors acknowledge financial support from the Canadian Institutes of Health Research CGS-D scholarship and an NSERC-Altera Industrial Research Chair. We are grateful to Bluespec Inc for contributing the Bluespec Compiler, and to Altera for FPGA CAD tools. Thanks also to Kevin Murray and the anonymous reviewers for thorough comments.

References

[1] S. L. Jacques and B. W. Pogue, "Tutorial on diffuse light transport," *J Biomedical Opt*, vol. 13, no. 4, p. 041302, 2008.

[2] B. C. Wilson and M. S. Patterson, "The physics, biophysics and technology of photodynamic therapy," *Phys med biol*, vol. 53, no. 9, pp. R61–109, May 2008.

[3] L. Wang, S. L. Jacques, and L. Zheng, "MCML–Monte Carlo modeling of light transport in multi-layered tissues." *Computer Methods and Programs in Biomedicine*, vol. 47, no. 2, pp. 131–146, 1995.

[4] H. Shen and G. Wang, "A study on tetrahedron-based inhomogeneous Monte Carlo optical simulation." *Biomed Opt Exp*, vol. 2, no. 1, pp. 44–57, Jan. 2010.

[5] J. Cassidy, L. Lilge, and V. Betz, "FullMonte: a framework for high-performance Monte Carlo simulation of light through turbid media with complex geometry," in *Proc SPIE BiOS*, vol. 8592. San Francisco, CA: SPIE, Feb. 2013, pp. 85 920H–14.

[6] T. Binzoni, T. S. Leung, R. Giust, D. Rüfenacht, and a. H. Gandjbakhche, "Light transport in tissue by 3D Monte Carlo: influence of boundary voxelization." *Computer methods and programs in biomedicine*, vol. 89, no. 1, pp. 14–23, Jan. 2008.

[7] Q. Fang and D. a. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units." *Optics express*, vol. 17, no. 22, pp. 20 178–90, Oct. 2009.

[8] E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration." *J Biomed Opt*, vol. 13, no. 6, p. 060504, 2012.

[9] E. Alerstam, W. C. Y. Lo, T. D. Han, J. Rose, S. Andersson-Engels, and L. Lilge, "Next-generation acceleration and code optimization for light transport in turbid media using GPUs," *Biomed Opt Exp*, vol. 1, no. 2, pp. 658–675, 2010.

[10] W. C. Y. Lo, K. Redmond, J. Luu, P. Chow, J. Rose, and L. Lilge, "Hardware acceleration of a Monte Carlo simulation for photodynamic therapy treatment planning," *J Biomed Opt*, vol. 14, no. 1, p. 014019, 2009.

[11] N. Corp, "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110," Tech. Rep., 2012.

[12] B. Dogdas, D. Stout, A. F. Chatziioannou, and R. M. Leahy, "Digimouse: a 3D whole body mouse atlas from CT and cryosection data." *Physics in medicine and biology*, vol. 52, no. 3, pp. 577–87, Feb. 2007.

[13] M. Saito and M. Matsumoto, *SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator*. Springer, 2008, no. 18654021, pp. 1–15.

[14] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC8, no. 3, pp. 330–334, 1959.

[15] L. Breslau, P. Cao, and L. Fan, "Web caching and Zipf-like distributions: Evidence and implications," in *IEEE Infocom*, vol. XX, 1999, pp. 126–134.

[16] J. Cassidy, "FullMonte: Fast Biophotonic Simulations," Master's thesis, University of Toronto, 2013.