Applying the Parallel GPU Model to Radiation Therapy Treatment PDPTA '13

J. Steven Kirtzic*, David Allen, Ovidiu Daescu**
Department of Computer Science
University of Texas at Dallas
Richardson, TX USA
{jsk061000, dallen, daescu}@utdallas.edu

Abstract— *With current advances in high performance computing, particularly the applications of GPUs, it is easy to see the need for a model for GPU algorithm development. We developed a model which offers a multi-grained approach intended to accommodate nearly any GPU.*

Radiation therapy is one of the most effective forms of cancer treatment available. In order to minimize the risk to the patient, physicians design treatment plans that expose the tumor to the prescribed levels of radiation while minimizing the exposure to the surrounding tissues. Our system allows users to quickly and easily visualize and compare treatment plans in order to identify the best one, with the most critical aspect of the simulation being implemented on the GPU using our parallel algorithm design model. In this paper, we show how the application of our model results in significant increases in algorithm performance, particularly in radiation therapy treatment simulation.

Keywords: GPU algorithm design, parallel processing, radiation therapy, cancer treatment

1. Introduction

The rapid advancement of the Graphics Processing Unit, or *GPU*, over the last few years has opened up a new world of possibilities for high-speed computation, ranging from biomedical to computer vision applications. Recent examples include [1], [2], and [3]. However, the GPU architecture is unlike that of any other, and designing algorithms to fully harness the capabilities of a GPU is not an easy task, especially when one considers the advantages and disadvantages of the various resources that a GPU has available to it.

Radiation therapy is a technique commonly used to eradicate malignant cancerous tumors. The therapy works by applying a controlled dosage of radiation to the tumor tissue in an attempt to damage the cancerous cells. Healthy tissue can also be damaged by the radiation, which raises the importance of optimizing the treatment in such a way that the tumor receives as much radiation as possible while the surrounding tissues receive as little as possible. We have developed a visualization system which provides treatment planners several different viewpoints to aid in choosing the best radiation treatment plan. The primary component of our system is an intensity mapping algorithm which utilizes a simple yet novel mapping scheme combined with a color-based representation of accumulated dosages to simulate the total amount of radiation delivered to a given target and the surrounding tissue. The result is an accurate, multi-view, navigable 3D representation of a given treatment plan that is beneficial for both practical clinical situations as well as educational environments.

The initial version of our system suffered from lag issues and was not capable of displaying multiple treatments with their accumulated dosages in real-time. Therefore, we utilized this as an opportunity to apply our Parallel GPU Model (PGM) to this system, more specifically to the mapping algorithm, as a validation of our model. The results were that our PGM version of the intensity mapping algorithm was able to not only perform in real-time, but also at a higher frame-rate, regardless of the granularity the various treatments were displayed at.

In this paper we present this application of our parallel algorithm design model for the GPU architecture, which demonstrates its effectiveness when applied to highly parallelizable tasks such as radiation treatment computation and simulation. In Section 2 we discuss the GPU architecture and the PGM; in Section 3 we present a brief overview of radiation therapy; in Section 4 we discuss the methods and materials used in the development of our simulation; in Section 5 we show the application of our PGM to the simulation itself, particularly to the intensity mapping algorithm; in Section 6 we discuss the results of the application of our model as compared to other implementations; and finally in Section 7 we conclude and remark on future work.

1.1 Contribution

We believe that our main contribution with this paper is to demonstrate the potential advantages of utilizing the high-performance computing power of the GPU. In this case we have chosen to apply our Parallel GPU Model to the task of simulating radiation therapy treatments for cancer patients.

*Kirtzic's research was supported by NSF award 0742477
**Daescu's research has been supported by NSF award CNS-1035460

With the application of our PGM, we are able to develop a version of our simulation which allows highly detailed simulations of treatments as they are delivered in real-time. We hope that this example of the benefit of utilizing GPUs will not only demonstrate the validity of our model, but will also encourage other researchers to take advantage of all that the GPU architectures has to offer.

1.2 Related Work

Borfield and Webb [4] have done work in the field of dosage calculation algorithms, as did Otto [5] and Vassilev [6]. Hamza-Lup et al [7] discuss the need for 3D treatment plan representation systems which allow both the clinician and the patient the ability to visually understand the advantages as well as the disadvantages of a given treatment plan. They present a system which allows the modeling of a plan in a 3D environment given a variety of input parameters. While producing fairly accurate results, their system does suffer from certain limitations, including the fact that it is web-based which makes its performance unpredictable due to bandwidth changes, as well as the lack of ability to view multiple treatments side by side, and the lack of a visual representation of accumulated dosage over a multi-beam treatment. While these methods have contributed considerably to the field of radiation therapy, they are limited when addressing the real-time, fine-grained needs of advancing current clinical treatment capabilities.

While there were several influences in the development of our Parallel GPU Model, the most noteworthy was the work of Leslie Valiant and his BSP model. The BSP, or bulk-synchronous parallel model, was proposed to overcome the limitations of the PRAM model [9], while maintaining its simplicity. In the BSP model, a BSP computer consists of a set of n processor/memory pairs (nodes) that are interconnected by a communication network. The BPS model is *Multiple Instruction Multiple Data* (MIMD) in nature, and uses the concept of a *superstep*, which is comprised of a computation step, a communication step, and a synchronization step. The BSP model is also variable grained, loosely synchronous, has non-zero overhead, and uses message passing or shared variables for communication.

The program executes as a strict sequence of supersteps. In each superstep, a process executes the computation operations in at most w cycles, a communication operation that takes gh cycles, and a barrier synchronization that takes l cycles. Note that in the communication overhead gh, g is the proportional coefficient for realizing a h relation. The value of g is platform-dependent, but independent of the communication pattern. In other words, gh is the time that it takes to execute the most time-consuming h relation.

Within a superstep, each computation operation uses only data in its local memory. This data is put into the local memory, either at the program start-up time or by the communication operations of previous supersteps. Therefore, the communication operations of a process are independent of other processes.

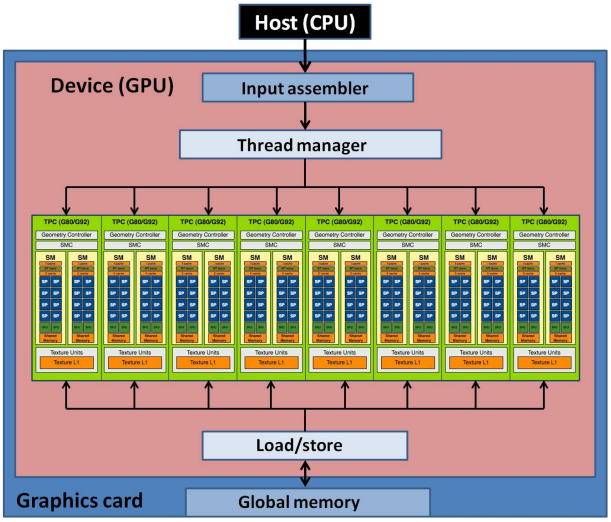

Fig. 1: NVIDIA GeForce 8800 architecture

The BSP model is more realistic than the PRAM model because it accounts for all overheads except for the parallelism overhead for process management. The time for a superstep is estimated by the sum

$$w + gh + l \tag{1}$$

This model is highly regarded and has formed the basis for other parallel models, such as the parallel phase model [9]. However, its generality is its shortcoming when one attempts to apply it to more specific architectures, such as that of the GPU. Valiant recently extended his model to include multi-core CPUs [10]. While this model is much more akin to the architectural nature of the GPU, it still does not take into consideration the complexities of the typical GPU architecture. Thus we developed a parallel algorithm design model for the GPU architecture which addresses these issues, which we first presented here [11].

2. The GPU architecture and the Parallel GPU Model

In this paper we will often refer to the machine containing the GPU as the “host" and the GPU itself as the “device”. The NVIDIA GeForce 8800 series is an example of a typical GPGPU (General Purpose GPU) device, which utilizes NVIDIA’s CUDA (*Compute Unified Device Architecture* GPU design. The GeForce 8800 contains 16 multiprocessors, each containing 8 semi-independent cores for a total of 128 processing units (see Figure 1). Each of the 128 processors can run as many as 96 threads concurrently, for a maximum of 12,288 threads executing in parallel.

The computing model is SIMD (*Single Instruction Multiple Data*), and the memory model is NUMA (*Non-Uniform*

Memory Access) with a semi-shared address space. This stands in contrast to a modern CPU, which is typically either SISD (*Single Instruction Single Data*) or MIMD, in the case of a multi-processor or multi-core machine. Additionally, from the perspective of the programmer, all memory is explicitly shared (in multi-threading environments) or explicitly separate (in multi-processing environments) on a desktop machine.

3. Radiation Therapy

In this paper we consider two main radiation therapy types: Intensity Modulated Radiation Therapy (IMRT), and Volumetric Modulated Arc Therapy (VMAT). In the IMRT method, beams of radiation calculated to be of a certain shape and intensity are administered to a particular *target*, which is typically a cancerous tumor [12]. The dosages are delivered in a discrete step-wise succession, a "step and shoot" method, with the typical treatment averaging around 6 - 9 beam dosages [13].

The VMAT method is similar to the IMRT method, however VMAT is a smoother, more contiguous method of delivery [14]. VMAT allows for the same dosage amounts as IMRT, but provides dosages in a pre-calculated arc of delivery. This results in the dosages being delivered in a continuous manner over a shorter span of time. Further discussion of the details of these methods can be found in [15], [16], and [17].

In both of these methods, a *multi-leaf collimator* or MLC is used to shape the beam of radiation. The "leaves" of the collimator can be moved back and forth to block parts of the source beam. Effective treatment plans adjust the leaf positions in such a way that they shield as much healthy tissue as possible from radiation exposure while the source beam is active.

4. Methods and Materials

We have developed a system which allows the visualization of a given treatment plan from a variety of viewpoints, allowing the physician to compare different treatment plans and determine which is the best suited to a given patient. Our system has the potential to simulate multi-beam IMRT treatments as well as VMAT treatments. The intended users of our software are medical students and physicians, with the aim of providing training in developing higher quality treatment plans as well as educating patients about the benefits and risks of individual plans.

The simulation work-flow is as follows: the user is presented with an interface which allows the user to load, edit, add, or remove a patient's treatment record from a treatment database. This database contains all of the necessary patient treatment information and could be made available online, allowing anyone with the proper permissions to access patients' treatment records.

Once a particular patient's treatment file is identified, our system can simulate potential treatments, allowing the physician to select the best of all possible treatments available. This is primarily accomplished by simulating an *intensity map* that maps levels of radiation absorbed by a section of tissue to its physical location. Our system can provide real-time simulations of a given treatment plan and present it in a variety of granularities. The interface allows the user to visualize a treatment from a variety of viewpoints, including a MLC view, a beam's-eye view of the target tumor, and a general 3rd-person view of the intensity map. In addition, the interface allows the user to switch between multiple views within the individual quadrants while the simulation is running. Perhaps the most useful feature of our system is the ability to have all four quadrants display four different treatment plans as intensity maps as they are running in real-time, allowing the user to compare the plans in high detail.

4.1 Mapping Algorithm

The input of our mapping algorithm consists of CT scans of the target at various time steps. We also require a data file representing the list of all MLC leaf movements (in mm) between times t_i and $t_i + 1$ for all t_i of a specific treatment plan *X*. We denote this file as the *input array*. We assume that the MLC leaves begin in a closed position at t_0.

The region represented by the CT scan is discretized into a grid of cells (Figure 2(a)). We equate each column on the grid to a millimeter of leaf movement and each row to a pair of opposing leaves in the MLC. The number of rows and columns represents the granularity of the system, as increasing the number of either rows or columns will increase the detail of the resulting intensity map with respect to the region in the CT scan image. Often these values are restricted by the physical characteristics of the MLC to be used, as different collimators may have different numbers and sizes of leaves, as well as different leaf motion parameters. In the examples below, we assume each leaf is 1 mm in width and moves 1 mm per unit of time to simplify the grid representation. In practice, the width of the leaves will be higher due to current manufacturing limitations.

Each cell in the grid is associated with a bar that represents the total amount of radiation that has accumulated in the area of the cell. The *input array* is parsed for every time step *t* of the simulation, and the position of each leaf in the MLC is updated by the appropriate amount. As each leaf moves it either exposes or covers cells on the grid, affecting the amount of radiation the cells receive and thus the growth rate of the associated bars.

Every cell that is exposed receives one unit of radiation intensity for every time step that the cell is exposed, increasing the corresponding bar height by one as well as changing the bar color based on the total amount of radiation received. An example of this process is shown in Figure 3, with numbers representing different intensity levels on the

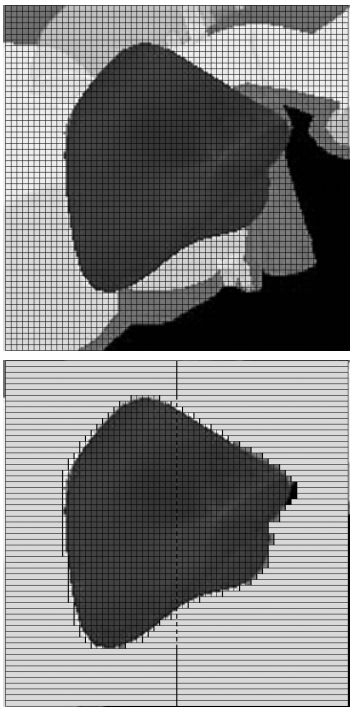

Fig. 2: (a) The intensity map bar grid overlaid on the CT scan of the target at t_0. (b) The leaves positioned around the target tumor to shield the surrounding tissue from the radiation beam

grid cells. After the simulation has completed, the resulting bar heights represent the total amount of dosage received by each cell grid. Different colors provide an extra means of visually processing the total dosage received at a cell. Blue indicates little or no radiation, red and orange indicates moderate amounts and bright yellow/white indicates a high amount. The example shown assumes a constant rate of radiation absorption, but in practice the dosage is not constant due to the differences in density and radiation absorption properties of different tissue types. We currently use density as the only factor affecting radiation absorption rates. The implementation identifies tissue density by the average lightness value of the grayscale colors in the CT scan image slice corresponding to a given grid cell. Brighter colors are assumed to be more dense and more resilient to radiation absorption, while darker colors are less dense and absorb radiation quickly.

From a time-space complexity perspective, given *m* number of time steps, *n* number of leaves, and *d* number of millimeters each leaf moves per time step, the run-time can be expressed as O(*mnd*) for a serial implementation of the algorithm.

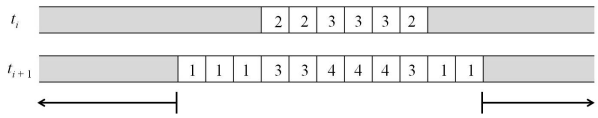

Fig. 3: A depiction of how the intensity values for each exposed bar increase from time t_i to time t_{i+1}.

4.2 Simulation Design Considerations

The basic flow of the system is the following: it first obtains from the list of leaf positions which bar sections are exposed to the beam, and which are not. Next it computes the dosage values received at each cell, and adjusts the height and color of the associated bar accordingly. The final radiation dosage values can be concatenated to a string and saved in a database, to be retrieved for later comparison or use. As discussed above, each kind of tissue absorbs radiation at a different rate, and so regions of the same tissue will have similar intensity values.

The colors and heights of the bars at the conclusion of the beam represent the total amount of radiation delivered to the corresponding area on the target or surrounding tissue. The user is able to toggle whether the bar heights are shown. With the heights disabled, the bars are simply squares of different colors overlaid onto the CT scan image and represent a heat map corresponding to the radiation delivered at each grid cell. The bars can also be made translucent, allowing the CT scan below to be seen and permitting the user to spatially visualize which tissue sections are receiving the most radiation.

We implemented our algorithm on the following hardware: we used a Dell desktop with a 2.99GH Intel Core 2 Duo with 3GB of RAM. We also used an NVIDIA 9800 GTX+ graphics card with 512 MB of memory and 128 streaming multiprocessors for the graphics duties. The fact that we implemented and ran our algorithm using commodity or "off the shelf" hardware indicates that our algorithm may be employed by practically anyone in a related field.

5. Application of the Parallel GPU Model

While the simulation that we have presented above can provide much in the way of designing and implementing highly effective radiation therapy treatments, the initial implementation of this system in a serial, CPU-based manner had its limitations. Particularly, the most critical part of our system (the intensity mapping algorithm visualization), suffered considerable lag when we attempted to apply it to higher detailed images. Specifically, we wanted the ability to view a given treatment in real-time, at various levels of detail, all the way to the millimeter level. Our experimentation proved that this was not possible with a serial implementation of our algorithm, given the hardware we had

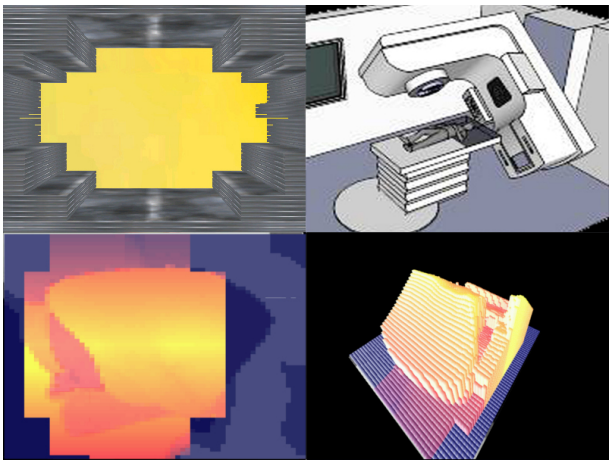

Fig. 4: This is the main interface of our visualization system as implemented strictly on the CPU in serial. Note that the various quadrants display differing views of a given treatment. Clockwise from the top-left: the first quadrant displays the position MLC leaves at time t_i; the second quadrant provides a view of the position of the gantry about the patient at t_i; the third quadrant displays accumulation of radiation dosages as a height map (before implementation on the GPU); and finally the fourth quadrant displays dosage accumulation as a heat map from a beam's eye view.

available. Consequently, this algorithm seemed to be a prime candidate for the application of our PGM.

We began applying our PGM to this problem by first developing the serial, CPU, brute-force version of our algorithm. This is the version that we had been using initially, which resulted in the performance lag identified earlier. Following the recommendations of the model, we then implemented an optimized version of this algorithm using OpenMP.

Again following the PGM model, we implemented a näive version of the intensity mapping algorithm by simply porting the algorithm to the GPU architecture, meaning we sent all of the data to the GPU and let the GPU kernels operate on the data, allowing the GPU thread scheduler to handle data distribution and thread allocation and management. We then went a step further in the model and considered data dependencies and preprocessing. This led to the optimized version of our algorithm, which includes preprocessing steps in which we import all of the necessary data from the database and convert it into simple arrays and image files before sending it to the GPU for computation and rendering. We have also off-loaded other preprocessing calculations to the CPU, including initial dosage calculations for each bar per each beam step, and total overall growth for each bar per each beam step. These results are temporarily stored in the host's RAM until this data is transferred to the device for real-time calculation and rendering.

Following with another aspect of our model, we considered the various types of GPU memory that we had access to, and distributed our data accordingly. It is not difficult to see that certain data elements of our system are read only (i.e. the treatment planning input, including the MLC movements and the CT scans), which according to the PGM should be loaded into the GPU's texture memory.

Finally, in following with the final step of the PGM, we fine-tuned our GPU-accelerated intensity mapping algorithm by adjusting it to suit the physical nature of our particular GPU. Knowing that our GPU (NVIDIA's 9800 GTX+) had 128 cores, with 512 MB global memory and 16K shared local memory for each SM, we broke down all computations so that they were allocated in groups of 32, aligning with the GPU's preferred warp format. However, considering that there were more available cores than there were calculations to be performed, even the use of memory coalescing and warp-filling does not result in a truly optimal use of the GPU structure, as a GPU operates the most efficiently when all of its threads are occupied.

6. Results and Discussion

By initializing the radiation levels at each cell to zero and applying our mapping algorithm and the leaf motions from the *input array*, we can generate the quad-view simulation of a given treatment plan as shown in Figure 4. This allows the user to select any quad and examine it in great detail. Each bar in the intensity map accumulates the dosage delivered over each beam in the treatment plan giving the physician or student a clear indication of how much radiation has been delivered to a given area of a target or the surrounding tissue.

This allows medical professionals to plan treatments of higher quality that minimize the exposure of healthy tissue to radiation while maximizing the dosage delivered to the target. Medical students studying radiology can use our system to simulate and visualize hypothetical treatment plans. Students receive feedback by comparing the resulting intensity maps of plans of their own design by observing the total amount of radiation delivered to the tumor versus the amount delivered to healthy tissues. This feedback assists in fostering an intuitive sense of what goes into planning an optimal treatment and therefore has the potential to improve the quality of the future real world treatment plans designed by the student. Moreover, the student or teacher may design example plans or exercises that highlight specific treatment complications and special situations. This has the effect of revealing to the student the best techniques for optimizing the plan under specific constraints.

As mentioned previously, patient education is another potential application. Doctors and radiologists can use our simulation to visually demonstrate the treatment process and explain exactly what the patient should expect. Treatment plans can be compared and contrasted for the patient using the multi-view interface mode, and the potential risks and

benefits of each can be explained and visualized in greater detail. This has the potential to increase patient understanding of and comfort with the therapy procedure.

In Table 1 we list the overall averaged run times and memory copy times for the different versions of our intensity mapping algorithm that were employed, including the näive CPU version, the optimal CPU version, the näive GPU version, and the optimal GPU version over 100 trials.

Table 1: Run time and data copy time for the Intensity Mapping algorithm, as represented for a näive CPU implementation, an optimized CPU implementation, a näive GPU implementation, and an optimized GPU implementation. All times are in milliseconds.

	Run Time	Copy Time
Näive CPU Intensity Mapping	192173.96	N/A
Optimized CPU Intensity Mapping	82186.34	N/A
Näive GPU Intensity Mapping	2007.12	197.78
Optimized GPU Intensity Mapping	199.94	2.04

As the table shows, optimizing the serial version of our algorithm using OpenMP did not show a significant increase in performance (approximately 2x speedup). This is due to the fact that we were using a dual core CPU, so considering the data transfer required between the two cores along with the synchronization step, the performance increase was fairly minimal (see Table 1).

When we simply ported our algorithm to the GPU architecture, our näive implementation resulted in a significant speedup (greater than an order of magnitude).

Regarding the optimized GPU version of the algorithm, we were able to reduce the overall runtime of the algorithm significantly by off-loading all of the necessarily serial work onto the CPU and only employing the GPU to do the parallel computation, specifically that of computing the values of each bar, as well as the overall accumulated values for each bar, at each cycle. This allows for optimal access speed as well as making this data available for the largest number of threads possible, resulting in a speedup of approximately an order of magnitude over the näive GPU implementation.

Regarding the visual performance of the Intensity Map (the calculations described above combined with the rendering), all implementations of the algorithm except the original näive CPU implementation allow the mapping to run in real-time. The GPU versions, however, allow the visualization to run in "hyper" real-time, at a variety of speeds, which should insure that the overall simulation can run in real-time or better regardless of the tasks it is performing, as shown in in Figure 5.

As shown above, we were able to provide a significant speedup in performance of our Intensity Mapping algorithm utilizing the high performance computing power of the GPU. Furthermore, we did so by applying our PGM to the problem of optimizing our Intensity Mapping algorithm, resulting in a significant speedup (a couple of orders of magnitude) over the original serial implementation.

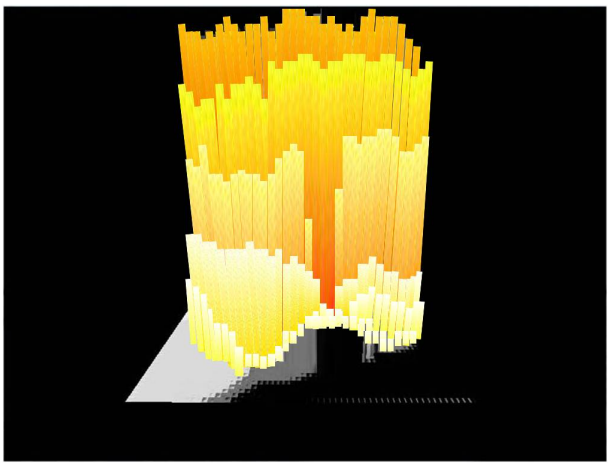

Fig. 5: The 3D rendering which shows the GPU implementation of how the radiation dosage will be delivered to the various areas of the target at time t_n, along with the total accumulated dosage (represented by the "heat scale" color gradation), according to this particular treatment plan.

7. Conclusion and Future Work

As presented in the previous sections, our visualization system offers a fast and inexpensive option for simulating radiation treatment plans in a real-time, multi-grained manner. This allows physicians and students the ability to quickly and easily compare multiple treatment plans to determine the optimal plan with considerable accuracy. We have then augmented this ability by applying our Parallel GPU Model to parallelize the computation being done on the intensity mapping aspect of our simulation, making it possible to now display a given treatment at a very fine-grained level of detail, in a fully navigable environment and in real-time, making it possible to view and compare multiple treatments as they would actually be delivered.

Based upon the the success of this application of our PGM, our future work with this system involves applying our model to the other aspects of the simulation to see which may also benefit from parallelization on the GPU. We will also be mindful to apply the same to any additional functionalities which we may include in future versions of our software.

In addition, the current version of our simulation handles single beam IMRT treatments. Future implementations will be able to handle multi-beam treatments as well as the continuous motion of the MLC during VMAT treatments. Another addition planned is to implement a module that *automatically* computes a radiation treatment plan. Although this treatment plan may not be optimal it can be made as

close to optimal as possible with computational methods. Such a module would allow users to easily obtain a baseline treatment plan to which they can compare the treatment plans they develop themselves, providing an extra mode of feedback and evaluation.

References

[1] D. Qiu, S. May, and A. Nüchter, "GPU-accelerated nearest neighbor search for 3d registration," *Computer Vision Systems*, pp. 194–203, 2009.

[2] P. Noël, A. Walczak, K. Hoffmann, J. Xu, J. Corso, and S. Schafer, "Clinical evaluation of GPU-based cone beam computed tomography," *Proc. of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI)*, 2008.

[3] J. Huang, S. Ponce, S. Park, Y. Cao, and F. Quek, "GPU-accelerated computation for robust motion tracking using the CUDA framework," in *Visual Information Engineering, 2008. VIE 2008. 5th International Conference on*. IET, 2008, pp. 437–442.

[4] T. Bortfeld and S. Webb, "Single-arc IMRT," *Physics in medicine and biology*, vol. 54, p. N9, 2009.

[5] K. Otto, "Letter to the editor on 'Single-Arc IMRT'," *Physics in medicine and biology*, vol. 54, p. L37, 2009.

[6] O. N. Vassiliev, T. A. Wareing, J. McGhee, G. Failla, M. R. Salehpour, and F. Mourtada, "Validation of a new grid-based Boltzmann equation solver for dose calculation in radiotherapy with photon beams," *Physics in Medicine and Biology*, vol. 55, no. 3, p. 581, 2010. [Online]. Available: http://stacks.iop.org/0031-9155/55/i=3/a=002

[7] F. Hamza-Lup, I. Sopin, and O. Zeidan, "Online external beam radiation treatment simulator," *International Journal of Computer Assisted Radiology and Surgery*, vol. 3, no. 3, pp. 275–281, 2008.

[8] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990. [Online]. Available: http://portal.acm.org/citation.cfm?id=79181

[9] K. Hwang and Z. Xu, *Scalable parallel computing: technology, architecture, programming*. WCB/McGraw-Hill, 1998.

[10] L. Valiant, "A bridging model for multi-core computing," *Journal of Computer and System Sciences*, vol. 77, no. 1, pp. 154–166, 2011.

[11] J. S. Kirtzic and O. Daescu, "A parallel algorithm development model for the GPU architecture," *International Conference on Parallel and Distributed Processing Techniques and Applications (accepted for publication)*, 2012.

[12] R. McMahon, R. Berbeco, S. Nishioka, M. Ishikawa, and L. Papiez, "A real-time dynamic-MLC control algorithm for delivering IMRT to targets undergoing 2D rigid motion in the beamâĂŹs eye view," *Medical physics*, vol. 35, p. 3875, 2008.

[13] X. Wu and J. Abraham, "The intensity level reduction in radiation therapy," in *Proceedings of the 2005 ACM symposium on Applied computing*. ACM, 2005, pp. 242–246.

[14] D. Rangaraj, S. Oddiraju, B. Sun, L. Santanam, D. Yang, S. Goddu, and L. Papiez, "Fundamental properties of the delivery of volumetric modulated arc therapy (VMAT) to static patient anatomy," *Medical physics*, vol. 37, p. 4056, 2010.

[15] D. Rangaraj, G. Palaniswaamy, and L. Papiez, "DMLC IMRT delivery to targets moving in 2D in beamâĂŹs eye view," *Medical physics*, vol. 35, p. 3765, 2008.

[16] M. Langer, V. Thai, and L. Papiez, "Improved leaf sequencing reduces segments or monitor units needed to deliver IMRT using multileaf collimators," *Medical Physics*, vol. 28, p. 2450, 2001.

[17] L. Papież, "DMLC leaf-pair optimal control of IMRT delivery for a moving rigid target," *Medical physics*, vol. 31, p. 2742, 2004.