Published in final edited form as:

Phys Med Biol. ; 63(2): 025028. doi:10.1088/1361-6560/aa9c96.

Multi-GPU Configuration of 4D Intensity Modulated Radiation Therapy Inverse Planning Using Global Optimization

Aaron Hagan¹, Amit Sawant¹, Michael Folkerts², and Arezoo Modiri¹

¹University of Maryland, School of Medicine, Baltimore MD

²Varian Medical Systems, Palo Alto CA

Abstract

We report on the design, implementation and characterization of a multi-graphic processing unit (GPU) computational platform for higher-order optimization in radiotherapy treatment planning. In collaboration with a commercial vendor (Varian Medical Systems, Palo Alto, CA), a research prototype GPU-enabled Eclipse (V13.6) workstation was configured. The hardware consisted of dual 8-core Xeon processors, 256 GB RAM and four NVIDIA Tesla K80 general purpose GPUs. We demonstrate the utility of this platform for large radiotherapy optimization problems through the development and characterization of a parallelized particle swarm optimization (PSO) four dimensional (4D) intensity modulated radiation therapy (IMRT) technique. The PSO engine was coupled to the Eclipse treatment planning system via a vendor-provided scripting interface. Specific challenges addressed in this implementation were (i) data management and (ii) non-uniform memory access (NUMA). For the former, we alternated between parameters over which the computation process was parallelized. For the latter, we reduced the amount of data required to be transferred over the NUMA bridge. The datasets examined in this study were approximately 300 GB in size, including 4D computed tomography images, anatomical structure contours and dose deposition matrices. For evaluation, we created a 4D-IMRT treatment plan for one lung cancer patient and analyzed computation speed while varying several parameters (number of respiratory phases, GPUs, PSO particles, and data matrix sizes). The optimized 4D-IMRT plan enhanced sparing of organs at risk by an average reduction of 26% in maximum dose, compared to the clinical optimized IMRT plan, where internal target volume was used. We validated our computation time analyses in two additional cases. The computation speed in our implementation did not monotonically increase with the number of GPUs. The optimal number of GPUs (five, in our study) is directly related to the hardware specifications. The optimization process took 35 minutes using 50 PSO particles, 25 iterations and 5 GPUs.

1. Introduction

High dimensional radiation therapy treatment planning involves computationally-expensive processes, such as image registration, direct aperture optimization, dose calculation and inverse plan optimization. Maximizing computational efficiency and speed, in order to make such applications feasible within clinical time frames, has been focus of several studies in literature; e.g., [1–6]. On the software side, efficient techniques for parallel computation as well as data access and data storage management have played an important role in this endeavor. On the hardware side, the massively parallel processing design of graphics

processing units (GPUs) introduced a prominent platform for efficiently running general purpose computing algorithms (i.e., GPGPU programming).

The radiation therapy inverse planning processes, which involves large patient datasets and large numbers of variables have been limited in clinical realization due to their computational complexity. However, GPU implementations have made some radiotherapy applications not only feasible but also timely [1, 3]. A comprehensive review of high performance computation (HPC) in radiation therapy can be found in [7, 8]. Despite existing works, solving such computationally complex problems using multi-search-agent global optimization techniques has remained as an implementation challenge. We aim to address this challenge in our article.

The scope of this study is limited to the details and computational analysis of a multi-GPU implementation of particle swarm optimization (PSO) algorithm for four-dimensional (4D) intensity modulated radiation therapy (IMRT) inverse treatment planning. Our implementation technique can be applied to other computationally-expensive, higher dimensional radiation therapy techniques, such as inverse planned volumetric modulated arc therapy (VMAT), due to similarity in data sizes and problem scales [6]. Our implementation technique can also be used to accelerate the optimization process in adaptive planning, where a large number of non-coplanar beams are considered (e.g., 4π). 4D treatment planning can enhance the dosimetric quality of radiotherapy in thoracic and abdomen treatment sites, where anatomy can change over time due to respiration [9, 10]. We were particularly interested in creating 4D-IMRT plans for lung cancer using temporal dimension as an additional degree of freedom in inverse plan optimization. To preserve future clinical practicality, our optimization engine worked with a prominent treatment planning system, Eclipse (Varian, Palo Alto). We introduced a C++ CUDA GPU implementation of PSO algorithm for 4D-IMRT inverse planning. Our previous work on a MATLAB-based CPU implementation of PSO for 4D conformal radiotherapy (CRT) inverse planning was leveraged for this effort [11]. Deliverability analysis is out of the scope of this study. In addition, the non-convexity of the solution space, PSO algorithmic details and objective function formulation are explained in our previous works [11, 12], and thus, not repeated here.

Global optimization algorithms, like PSO, have been used in large-scale, non-convex problems with unknown global solutions [13–16]. The PSO algorithm is a highly-parallelizable, sometimes referred to as an embarrassingly parallel ([17]), global optimization algorithm. It uses a swarm of search agents (particles) to iteratively search for the optimal solution [18,19]. The PSO particles can be operated independent of each other on slave nodes, while a master node compares the updated objective functions and identifies the swarm's best achievement. PSO's algorithmic structure makes it an excellent candidate for GPU implementation because separate instances of same set of tasks are run [20]. Typical PSO implementations are parallelized over the swarm of particles. However, such an implementation of PSO can be limited by the amount of data required to be saved and accessed on GPUs [21]. In our study, since the size of the data was larger than maximum GPU memory, we introduced a new parallelized implementation approach, where, at each iteration, parallelization over particles (typical to PSO pipeline) was redesigned to be

performed in the sequences enforced by the registration operator. Our implantation technique can be applied to any similar high dimensional optimization problem where data needs to be partitioned in large chunks. Typically, in order to make data manageable, down-sampling or shrinking of the region of interest is performed [4, 22]. Here, we investigated data processing with and without down-sampling and analyzed processing speed when varying several parameters; i.e., number of respiratory phases, number of GPUs and number of PSO particles.

Several published studies on GPU implementation in radiation therapy have focused on dose calculation [1–4, 7, 23]. For inverse planning, Men et al. introduced an efficient GPU-based IMRT planning technique [4]. However, Men et al.'s study differed from ours in the following major points, making the two implementation methodologies distinct: (i) we studied 4D-IMRT where a registration operator was used to sum the dose over respiratory phases, while Men et al. studied internal target volume- (ITV-) based IMRT planning, (ii) we used a global meta-heuristic optimization technique to account for non-convexity of a dose-volume-based solution space, while Men et al. used a convex quadratic objective function and a gradient-based optimization algorithm, (iii) we optimized MU weights for apertures (as explained in [11, 12]), while Men et al. used fluence optimization. To the best of our knowledge, our study is the first study on GPU implementation of 4D-IMRT inverse planning using a global optimization algorithm. The closest work to our study was performed by RaySearch Laboratories on GPU-based 4D treatment planning using Minimax technique for inverse plan optimization [24]. In the Minimax technique, each objective is minimized only on the instance where it is demonstrating its maximum value. In our technique, however, the summation over all instances (here, respiratory phases) was minimized; therefore, our technique allowed for unique dose distributions across all respiratory phases. There are also several studies in literature which report on GPU implementation of PSO for non-radiotherapy applications [20,21,25]. The major difference between our work and such studies resides in the data size and also the presence of operators which partition datasets (such as deformable image registration operator which partitions data in terms of respiratory phases). This distinction significantly affected GPU implementation.

The remainder of the paper is structured as follows: Section 2 outlines our algorithm and the techniques used in our solver. Section 3 evaluates our proposed planning technique comparing treatment plans created by our optimization algorithm with those created in a conventional clinical process. We provide a discussion of our study's limitations and future works in Section 4, and our conclusion is given in Section 5.

2. Method

In general, the enhanced efficiency of computation on GPUs is mainly due to the single-instruction-multiple-data (SIMD) architecture of the GPU hardware coupled with the direct memory mappings in the GPGPU software libraries. However, memory size on GPUs is a limitation. Considering our available hardware, we designed a parallelization technique to efficiently implement our inverse plan optimization problem. Our multi-GPU computing platform consisted of eight NVIDIA Kepler GK210 GPUs (4 NVIDIA Tesla K80s), dual

Intel Xeon CPUs (8 cores, 16 threads a piece), and 256 MB of RAM with a maximum processing capability of 8.5 Tflops per K80 card (we used single precision float numbers). Each GK210 has 12 GB of memory, totaling 96 GB of GPU memory for our whole system. This system had a non-uniform memory access (NUMA) architecture; i.e., each CPU had direct access to their own system memory (DDR4) and peripheral component interconnect (PCI) slots. The hardware architecture was important to our implementation as it impacted performance notably. During the objective function calculation step, all of the updated parameters were read from the multiple GPUs in the system to be summed together and, there were process time limitations when accessing data on GPUs in adjacent PCI slots (Fig. 1 shows the layout of the components in the system).

2.1. Workflow

As the standard of care for thoracic patients, 4D CT scans are acquired to represent a sampled set of respiratory phases (generally ten phases). Similar to our conformal inverse planning studies ([11,12]), we used 4D CTs to convey time-dependent anatomical changes to the inverse planning optimization algorithm. The optimization task in our study was to optimize monitor unit (MU) intensity weights for all apertures (control points) of all beams and across all respiratory phases with the goal of reducing dose to organs at risk (OARs), while, first and foremost, the prescribed dose to planning target volume (PTV) was maintained.

The following were the three key steps in our treatment planning system:

1. For each respiratory phase, an initial IMRT treatment plan was created and optimized in Eclipse treatment planning system (V13.6, Varian, Palo Alto, CA). The beam angles were identical to those assigned by the physician for the clinical ITV-based plan.
2. Each plan consisted of a set of beams and each beam consisted of a set of apertures. For each aperture, a dose deposition matrix was calculated in Eclipse. This matrix represented the dose deposited at each voxel for an aperture weight of unity.
3. Dose deposition matrices were exported from the treatment planning system (TPS) and imported in our in-house PSO engine, where aperture intensity weights, and thus, aperture MUs, were optimized towards creating a desired dose distribution.

The implementation of step 3 was the focus of this manuscript. We performed the two initial steps in Eclipse using Eclipse scripting application programming interface (ESAPI). Voxels were equally-sized volume elements in the patient body. Note that a typical such optimization problem can have tens of thousands of variables (e.g., in our case study 1, we had $9 \text{ beams} \times 166 \text{ apertures per beam} \times 10 \text{ sampled respiratory phases} = 14940 \text{ variables}$).

2.2. PSO Algorithm

PSO basics have been studied in literature thoroughly [18, 19], hence, we solely review the key operation in the PSO kernel; i.e., velocity calculation. Each PSO particle's exploration

of the solution space is guided by (i) the particle's own initial velocity, (ii) its distance from the best solution that it has found individually, and (iii) its distance from the best solution found by the entire swarm. The goodness of each position inside the solution space is quantified by a problem-specific objective function. PSO velocity function is modeled as:

$$V_t^p = \omega_t V_{t-1}^p + C_1 \gamma_t^p (Pbest_t^p - X_t^p) + C_2 \delta_t^p (Gbest_t - X_t^p) \quad (1)$$

$$X_{t+1}^p = X_t^p + V_t^p \quad (2)$$

where, V , X , $Pbest$ and $Gbest$ represent the velocity, position, personal best and global best arrays, respectively. γ and δ are random-number arrays with uniform distribution in $[0, 1]$ and represent the stochastic behavior of the swarm. The dimensionality of all arrays in (1) is equal to the number of variables (aperture weights). The number of aperture weights is the number of beams $\times$ the number of apertures per beam $\times$ the number of respiratory phases. Iteration number and particle number are shown with subscript t and superscript p . As iterations progress, ω decreases from 0.9 to 0.4, in order to inspire wide exploration in initial iterations and convergence in final ones. In our study, C_1 and C_2 were chosen to be equal to 2 as recommended in the literature [26].

2.3. Partitioning Operator and Objective Function

In our study 4D-IMRT inverse planning, large datasets (total of around 300 GB per patient) were required to be analyzed for objective function calculations. In addition, the data processing involved a deformable image registration (DIR) operator that enforced a specific partitioning format on its input data. Since the use of our implementation technique could be extended to other applications using partitioning (piecewise) operators, we generalized our terminology by calling the registration operator a partitioning operator throughout this paper. The size of the partitioned data, in our study, was larger than maximum GPU memory; hence, we introduced a new parallelized implementation approach, where in the objective function calculation step, at each iteration, parallelization over particles (typical to PSO pipeline) was redesigned to be performed in the sequences enforced by the partitioning operator. Our implementation technique can be applied to any similar high dimensional optimization problem where data needs to be partitioned.

We used DIR to register dose distributions corresponding to distinct respiratory phases on a single phase in order to calculate summed dose. In general, DIR operators map varying images of a single object onto a reference image, so that they can be processed uniformly [27]. In our study, DIR had to be applied to subsets of our datasets corresponding to distinct respiratory phases, and therefore, we classified it as a partitioning operator. For the sake of consistency, we name the partitioning operator subsets as phases. To avoid deforming and keeping tens of thousands of dose matrices offline, we calculated deformation vector fields (DVF) prior to starting optimization iterations. DVFs were then applied to aperture dose matrices exported from our TPS in each optimization iteration. The overall dose (sum of scaled deformed dose matrices) was used to calculate the objective function. The summed

differences between desired doses and actual doses were used to formulate the objective function. We used the objective function modeled in equations (1)–(3) in [12]. The solution space was non-convex due to including noncontinuous dose-volume-based terms in the objective function.

For DIR, we incorporated Elastix 4.7, an open source library, which we modified to optimize memory transfers to and from the CPUs and GPUs [28]. Elastix is written in C++ and uses CUDA to compute its registration steps in parallel using GPU.

2.4. Optimization Framework

Figure 2 shows the structure of our optimization implementation involving operations that were parallelized on the GPUs and CPUs. The SIMD architecture of the GPU enabled us to update particle positions and velocities on the GPU(s) independently, in each PSO iteration cycle. The algorithm iterated over the phases in parallel on the CPU and started the partitioning operator on multiple GPUs. In Fig. 2, N is the number of partitioning operations. As an example, when considering 10 respiratory phases in 4D treatment planning, dose matrices are partitioned to 10 phases. One phase is considered as the reference phase, staying unmodified, while $N = 9$ DIR operations are performed to register the other 9 data sets to the reference phase. End-exhalation is the most clinically-relevant and preferred phase as reference phase owing to the stability and reproducibility of tumor target position as well as a longer duty cycle [29–31]. However, in terms of computational time and complexity, the choice of reference phase does not have a notable impact on optimization process.

We implemented the optimization code so that it could accommodate the partitioning operator with different number of phases and PSO with different number of particles. To further clarify, a 10-phase, 200-particle study would equate to 1800 DIR operations in parallel. For a typical patient with each aperture dose deposition matrix being 18.7 MB in size, 37.4 GB of dedicated GPU memory would be needed to be allocated by Elastix. Therefore, we redesigned parallelization and allocated enough memory to perform the DIR in parallel over particles per phase. This novel use of the partitioning operator makes our PSO GPU implementation distinct from common GPU PSO implementations [20, 21, 25].

For the implementation of the objective function, we used NVIDIA's Thrust libraries for computing the vector-based arithmetic operations and updating the swarm particle positions. The use of the Thrust libraries requires the programmer to simplify the algorithm into independent steps in the form of vector multiplication and addition operations [32]. As a very simple example, in $(a + b \times c)$ operation, b and c will be multiplied first and then added to a in two separate steps on the GPU instead of one single operation. The data for all particles was stored and shared across all the GPUs, structured as shown in Fig. 3.

2.5. Sparse Matrix Multiplication On GPU

Large datasets generally cannot be stored on limited GPU memory, and therefore, they are reformatted as sparse matrices, and/or down sampled or split to smaller sizes. One of the challenges in data management for a GPU implementation is indexing of large datasets. Kim et al. have shown an implementation of indexing multi-dimensional datasets on the GPU in

different tree type data structures [33]. For compression of the large datasets in our project, we used a Compressed Sparse Row (CSR) data structure and matrix indexing. Using CSR, the sparser the dataset was, the more reduction in storage could be achieved.

For our application, input data was prepared and copied to GPU memory along with the parameters for defining swarm population, termination criteria, objective function, and velocity function. We were able to use the NVIDIA cuSPARSE library to perform the compression and data scaling (computation steps are shown in Fig. 2). The 3D dose datasets in our study were highly sparse; therefore, the GPU memory saving was maximized by compressing the dataset using the CSR format. CSR saves the row indexes and the column extents used to index the non-zero values. cuSPARSE has libraries for creating sparse matrices in parallel; however, the data needs to be completely uploaded on the GPU. Subsection 2.6 highlights the size of the typical datasets and the savings achieved by sparsification. The CSR data structures were resided completely in GPU memory and accessed during the scaling step. CSR data structures can reconstruct the full matrix by storing the non-zero values, column indexes, and number of non-zero elements in the row. For the MU weight scaling step, we used the `cusparsescrmm` function in the cuSPARSE library and computed $M \times k$; i.e., multiplication of input sparse datasets with the PSO variable array at each iteration cycle. k is the number of elements in the dose matrix and M is the number of beams times apertures. Figure 3 shows the dimensions of the scaling operation with the first block being the input data stored in CSR format. The next steps in the PSO algorithm included analyzing the objective function, searching for G_{best} and P_{best} , calculating the velocity function and updating the PSO particle positions.

2.6. Data Size and Memory Requirement

To show the results of our PSO GPU application, we used anonymized clinical data from three lung cancer patients, retrospectively. Each case study contained datasets for 10 respiratory phases consisting of CT scans, dose matrix files, and structure masks for PTV and OARs. Each phase had one 3D CT image of size $512 \times 512 \times 217$, which was down-sampled to the dose resolution (typical voxel size of $2.5 \times 2.5 \times 2.0 \text{ cm}^3$) for the DIR step. As an example, for Case 1, each of the phases consisted of 1,494 (9 beams $\times$ 166 apertures) dose matrices of size $196 \times 118 \times 217$ equaling 29.9 GB before sparsification. After sparsification, the dose matrices for each phase ranged in size from 1.06 GB to 1.15 GB. For each IMRT beam aperture, we calculated the dose deposition matrix in Eclipse treatment planning system using the Analytical Anisotropic Algorithm (AAA). When creating sparse matrices, we set a threshold to consider any dose value less than 1% of the maximum dose in each matrix (i.e., max dose to PTV from that aperture) to be negligible and equal to zero. This value was chosen as it is well below the tolerance prescribed in AAPM Task Group 119 on IMRT guidelines [34]. After applying this threshold, we found that only 10%–20% of the dose matrix values were non-zero.

To give readers an idea of data transfer time, here we calculate it for a sample matrix size: Data is transferred between GPUs at a rate of 16 GB/s one way using the PCI-E 3.0 bus. A matrix of $196 \times 118 \times 217$, stored in 32 bit floating point values, has a total size of 20.08

MB and transferring this single matrix across the bus happens at theoretical duration of 0.00125 seconds.

2.7. GPU Clock Auto-Control

GPU clock rate becomes highly important when introducing multiple GPUs to the computational system. A summary on how the K80 Tesla card can automatically adjust its clock rate based on its load is shown in [35]. The K80 introduces 'auto-boost', which automatically selects the highest possible clock rate allowed by the thermal and power budget. Using the auto-boost feature, the speed of different processes may not necessarily scale reasonably with the addition of multiple GPUs in the system. That is because more GPUs will add to the thermal and power budget, and could result in lower clock rates. We observed an overall increase in our application time with the employment of auto-boost, and thus, chose to disable it. Even with auto-boost disabled, we observed no significant difference in the thermal characteristics of our machine.

2.8. Case Studies

We analyzed the performance of our treatment planning system and its implementation using real patient data for a lung cancer case. Figure 4 depicts a 3D view of the clinical 9-beam configuration and patient anatomy. The patient had a right lower lobe tumor with gross tumor volume (GTV) of 61 cc and left lung lobe removed prior to radiotherapy. Tumor motion due to respiration was 1.5 cm. We validated our computation time dependencies on various parameters in two additional patient datasets. The three patients reported herein encompass a variety of lung radiotherapy cases presented in the clinic. Case 2 had a 112.5 cc left upper lobe GTV and 0.7 cm respiration-induced GTV motion, for which an eleven-beam IMRT plan was clinically prescribed, and, Case 3 had a 479.4 cc central right lobe GTV and 0.5 cm respiration-induced GTV motion, for which a seven-beam IMRT plan was clinically prescribed.

3. RESULTS

Figure 5 shows the process time when different parameters are varied for the three cases. The process time is divided into two main parts: optimization (PSO) time and registration (partitioning operator) time. A total time is reported as well. Any computation performed merely because of registration was included in the partitioning operator time. The left column in Fig. 5 shows down-sampled datasets (e.g., $k=98 \times 59 \times 217$ for Case 1) and a swarm population of $P=50$. For all the three cases in the left column, increasing the number of GPUs did not vary the process speed significantly. In the middle column in Fig. 5, where original data sizes (e.g., $k=196 \times 118 \times 217$ for Case 1) and a swarm population of $P=200$ were used, the fastest process time was achieved by 5 GPUs and increasing the number of GPUs to more than 5 did not enhance the process speed. These figures demonstrate the impact of using different sizes of data on time and handling of the related processes (reading, sparsification and uploading to the GPUs). Process time increase for more than 5 GPUs was due to the cost of transferring data over the NUMA bridge and the GPU clocks running at a lower frequency when they were all being utilized in our system. As an example, in Case 1, for a swarm population size of 200, data size of $196 \times 118 \times 217$ voxels,

14940 variables and 9 deformable image registration operations, our C++ CUDA implementation took almost 5.0 minutes per iteration to run on 5 GPUs. Reducing the swarm population size to 50 and data matrix size to $1/4^{th}$ cut the process time by a factor of 5.

The right column in Fig. 5 compares the process times for fixed data sizes and swarm populations when number of partitioning operations is changing. In our implementation, data related to each partitioning operation were assigned to one exclusive GPU, and when the number of phases was greater than the number of GPUs, each GPU could be assigned to multiple phases. This implementation method explains why in the right column in Fig. 5 PSO times stay relatively constant for several phases and then increase. When 5 GPUs are used for 9 partitioning operations, multiple phases are sharing GPU resources.

The three patients studied in Fig. 5 had tumors with different sizes located at different parts of lungs and were treated with different numbers of beams, and thus, irradiated volumes were also different. Therefore, they encompassed a variety of lung radiotherapy cases presented in the clinic. The computational time results followed the same trend in all three patients, showing implementation robustness across variable scenarios. Table 1 demonstrates a more detailed analysis of the process time and speedup factors gained by increasing GPUs for Case 1.

Sparse dose deposition matrices are very typical of IMRT plans with dynamic MLC delivery, where apertures are fairly small. In the dose datasets of our studied cases, 10%–20% of the voxels contained positive non-negligible values. As expected, we observed a linear gain trend in memory consumption by application of sparsification. Our reported timing results do not include a data management time, which was spent on sparsifying 300 GB of data on multiple CPU threads. The data management process is a one-time step during which the sparse matrices for each patient are saved to the disk and may be used in future optimization tests for the same patient.

Figure 6 illustrates the process times versus swarm population size and data size for Case 1. Although the partitioning operation was performed in parallel for all particles, there was still overhead in copying data through the Elastix library. In addition, the process time was sensitive to swarm population size because it affected the number of elements needed to be scaled. Figure 3 shows why these parameters are important in our multi-GPU implementation and how they affect the data transfer time and, consequently, the analysis time. An approximately linear trend was observed for the process time with respect to the two parameters.

Finally, Fig. 7 shows the optimization results in terms of objective function convergence trend and dosimetric quality of the plans for Case 1. Figure 7 (b) and (c) contrast the dose distribution from PSO optimized plan and that of the clinical ITV-based plan. It is observed that dose to OARs was notably reduced (by an average of 26% for max dose) while the prescribed dose to tumor was maintained.

4. DISCUSSION

The scope of this study was limited to the details and computational analysis of our multi-GPU PSO implementation for 4D-IMRT inverse treatment planning. For PSO algorithmic details and objective function formulation, the reader is referred to our previous works [11, 12].

Our novel contribution was the introduction of (i) an Eclipse-based 4D-IMRT inverse planning system using a global optimization technique and (ii) an optimized implementation method for accelerating the inverse plan optimization process on multiple GPUs and CPUs. A key part of our implementation was the PSO parallelization design while having the partitioning operator as a part of the iterative process. Due to GPU memory limitations, it was not possible to maintain the common parallelization of PSO over particles for the entire optimization process [20]; therefore, as shown in Fig. 6, the process time in our implementation increased with the number of particles. Our solver was also time bound with respect to the data matrix size and required number of partitioning operations.

In this work, we limited our investigation to a swarm population size of 200. A general rule of thumb for choosing the number of particles for PSO is to have a large enough swarm to sufficiently sample the solution space, and therefore, the number of particles is defined based on the number of variables and the complexity of the solution [12]. In our specific application, complexity of solution space is determined by the number of dose-volume-based terms in the objective function. A large swarm can comprehensively explore solution space but at the expense of an increased computational complexity.

Process speed in our implementation was also dependent on the level of minimization we could achieve for data transfers between CPU and GPU memories, especially across the NUMA bridge. Although we had 8 GPUs on our computation platform, we found that process time would decrease by adding GPUs up to 5, but start increasing beyond 5. This time increase was due to (1) the cost of transferring data over the NUMA bridge and (2) the GPU clocks running at a lower frequency when they were all being utilized in our system. As GPU memory becomes more readily available, and the speeds across the system buses increase, memory limitations will be obsolete. We already see such technological leaps using the NVIDIA P100 cards, which have 16 GB of GPU memory versus 12 GB on the NVIDIA K80 GPU. One solution for addressing memory issues is to use a multi-GPU computing cluster. Cluster solution can also help with clock rate control, GPU memory fragmentation, and data transfer efficiency by utilizing high bandwidth networking interconnects.

Although we specifically studied 4D-IMRT, our implementation technique can potentially be used in other computationally-expensive, higher dimensional radiation therapy applications, such as inverse planned 3D- or 4D-VMAT [6], as the overall process of treatment planning optimization and data sizes are comparable. Our implementation technique could also help with speeding up the optimization processes in adaptive planning techniques, where large number of non-coplanar beams are used, such as in 4π radiotherapy [5].

5. CONCLUSION

Expediting computational applications that deal with large datasets is often limited by computational resource management capabilities. In this study, we investigated a GPU implementation technique for such problems in a multi-search-agent optimization setup. We tested our implementation on 4D-IMRT inverse planning for lung cancer. To help our method be clinically translatable, our optimization engine was designed to work with Eclipse TPS through a vendor-provided scripting interface. 4D planning with simultaneous spatial and temporal optimization of radiation therapy treatment plan has shown to significantly improve the dosimetric quality of treatment plans by using motion as a degree of freedom in planning process. However, the high computational complexity associated with such planning technique represents a significant barrier to its widespread clinical deployment.

While radiotherapy inverse planning processes generally use large data sets, they can further be limited by operators which partition the data in large chunks. Deformable image registration in 4D planning is an example of such operators. Our study showed a technique to maximize GPU performance efficiency and speed in the presence of a partitioning operator. Our processing platform had access to 8 GPUs; however, our time analysis showed that no more than 5 GPUs would perform efficiently. The most important factor in GPU implementation was the amount of data required to be stored on GPU memory. That data was mainly defined by our dose files (generated by Eclipse TPS) and also by the swarm population size in particle swarm optimization.

Acknowledgments

The authors would like to thank Wayne Keranen, Michelle Svatos and Camille Noel from Varian Medical Systems, Palo Alto, CA. Our study was partially supported by R01CA169102 grant from National Institute of Health (NIH) and Varian Medical Systems.

References

1. Li Y, Tian Z, Shi F, Song T, Wu Z, Liu Y, Jiang S, Jia X. A new Monte Carlo-based treatment plan optimization approach for intensity modulated radiation therapy. *Physics in Medicine Biology*. 2015; 60(7):2903. [Online]. Available: <http://stacks.iop.org/0031-9155/60/i=7/a=2903>. [PubMed: 25776792]
2. Tian Z, Shi F, Folkerts M, Qin N, Jiang SB, Jia X. A GPU OpenCL based cross-platform Monte Carlo dose calculation engine (gomc). *Physics in Medicine Biology*. 2015; 60(19):7419. [Online]. Available: <http://stacks.iop.org/0031-9155/60/i=19/a=7419>. [PubMed: 26352012]
3. Chi Y, Tian Z, Jia X. Modeling parameterized geometry in GPU-based Monte Carlo particle transport simulation for radiotherapy. *Physics in Medicine Biology*. 2016; 61(15):5851. [Online]. Available: <http://stacks.iop.org/0031-9155/61/i=15/a=5851>. [PubMed: 27427297]
4. Men C, Gu X, Choi D, Majumdar A, Zheng Z, Mueller K, Jiang SB. GPU-based ultrafast IMRT plan optimization. *Physics in Medicine and Biology*. 2009; 54(21):6565. [Online]. Available: <http://stacks.iop.org/0031-9155/54/i=21/a=008>. [PubMed: 19826201]
5. Men C, Jia X, Jiang SB. GPU-based ultra-fast direct aperture optimization for online adaptive radiation therapy. *Phys Med Biol*. 2010; 55:43094319. [Online]. Available: <http://iopscience.iop.org/article/10.1088/0031-9155/55/15/008/pdf>.
6. Tian Z, Peng F, Folkerts M, Tan J, Jiang XJS. Multi-GPU implementation of a VMAT treatment plan optimization algorithm. *Med Phys*. 2015; 42(6):2841–52. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pubmed>. [PubMed: 26127037]

7. Jia X, Ziegenhein P, Jiang SB. GPU-based high-performance computing for radiation therapy. *Physics in Medicine and Biology*. 2014; 59(4):R151. [Online]. Available: <http://stacks.iop.org/0031-9155/59/i=4/a=R151>. [PubMed: 24486639]
8. Pratz G, Xing L. Gpu computing in medical physics: A review. *Medical Physics*. 2011; 38(5):2685–2697. [Online]. Available: <http://dx.doi.org/10.1118/1.3578605>. [PubMed: 21776805]
9. Suh Y, Murray W, Keall PJ. IMRT treatment planning on 4D geometries for the era of dynamic mlc tracking. *Technology in Cancer Research & Treatment*. 2013; 13:505–512. [PubMed: 24354751]
10. Nohadani O, Seco J, Bortfeld T. Motion management with phase-adapted 4D-optimization. *Physics in Medicine Biology*. 2010; 55(17):5189. [Online]. Available: <http://stacks.iop.org/0031-9155/55/i=17/a=019>. [PubMed: 20714043]
11. Modiri A, Gu X, Hagan A, Bland R, Iyengar P, Timmerman R, Sawant A. Inverse 4D conformal planning for lung sbt using particle swarm optimization. *Physics in Medicine and Biology*. 2016; 61(16):6181. [Online]. Available: <http://stacks.iop.org/0031-9155/61/i=16/a=6181>. [PubMed: 27476472]
12. Modiri A, Gu X, Hagan A, Sawant A. Radiotherapy planning using an improved search strategy in particle swarm optimization. *IEEE Transactions on Biomedical Engineering*. 2016; PP(99):1–1.
13. Kalantzis, G., Lei, Y. A self-tuned bat algorithm for optimization in radiation therapy treatment planning. 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD); June 2014; p. 1-6.
14. Siauve N, Nicolas L, Vollaire C, Marchal C. Optimization of the sources in local hyperthermia using a combined finite element-genetic algorithm method. *International Journal of Hyperthermia*. 2004; 20(8):815–833. [Online]. Available: <http://dx.doi.org/10.1080/02656730410001711664>. [PubMed: 15764344]
15. Darzi S, Sieh Kiong T, Tariqul Islam M, Ismail M, Kibria S, Salem B. Null steering of adaptive beamforming using linear constraint minimum variance assisted by particle swarm optimization, dynamic mutated artificial immune system, and gravitational search algorithm. *Scientific World Journal*. 2014; 2014
16. Juang CF, Hung CW, Hsu CH. Rule-based cooperative continuous ant colony optimization to improve the accuracy of fuzzy system design. *IEEE Transactions on Fuzzy Systems*. Aug; 2014 22(4):723–735.
17. de Vega, FF., Pérez, JIH., Lanchares, J. *Parallel Architectures and Bioinspired Algorithms*. Springer Publishing Company, Incorporated; 2014.
18. Eberhart, R., Kennedy, J. A new optimizer using particle swarm theory. *Micro Machine and Human Science; Proceedings of the Sixth International Symposium on*; Oct 1995; 1995. p. 39-43. MHS '95
19. Kennedy, J., Eberhart, RC. *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc; 2001.
20. Silva EHM, Filho CJAB. PSO efficient implementation on GPUs using low latency memory. *IEEE Latin America Transactions*. May; 2015 13(5):1619–1624.
21. Souza, DL., Monteiro, GD., Martins, TC., Dmitriev, VA., Teixeira, ON. PSO-GPU: Accelerating particle swarm optimization in CUDA-based graphics processing units. *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, ser. GECCO '11*; New York, NY, USA: ACM; 2011. p. 837-838. [Online]. Available: <http://doi.acm.org/10.1145/2001858.2002114>
22. Neumann, L., Cséfalvi, B., Viola, I., Mlejnek, M., Gröller, E. Feature-preserving volume filtering. *Proceedings of the Symposium on Data Visualisation 2002, ser. VISSYM '02*; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association; 2002. p. 105-ff. [Online]. Available: <http://dl.acm.org/citation.cfm?id=509740.509757>
23. Robert Jacques JWTM, Taylor Russell. Towards real-time radiation therapy: GPU accelerated superposition/convolution. *Comput Methods Programs Biomed*. 2009; 98(3):285–92. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pubmed/19695731>. [PubMed: 19695731]
24. Albin Fredriksson BH, Forsgren Anders. Maximizing the probability of satisfying <https://www.sharelatex.com/project/591b5b05eb7b6f10773d008c> the clinical goals in radiation therapy

- treatment planning under setup uncertainty. *Medical Physics*. 2015; 42(7):3992–9. [Online]. Available: https://people.kth.se/~albfre/uncertainty_set_optimization.pdf. [PubMed: 26133599]
25. Liu ZH, Li XH, Wu LH, Zhou SW, Liu K. GPU-accelerated parallel coevolutionary algorithm for parameters identification and temperature monitoring in permanent magnet synchronous machines. *IEEE Transactions on Industrial Informatics*. Oct; 2015 11(5):1220–1230.
 26. Bratton, D., Kennedy, J. Defining a standard for particle swarm optimization. 2007 IEEE Swarm Intelligence Symposium; April 2007; p. 120-127.
 27. Sotiras A, Davatzikos C, Paragios N. Deformable medical image registration: A survey. *IEEE Transactions on Medical Imaging*. Jul; 2013 32(7):1153–1190. [PubMed: 23739795]
 28. Murphy K, Ginneken BV, Reinhardt JM, Kabus S, Ding K, Deng X, Pluim JPW. J.p.w.: Evaluation of methods for pulmonary image registration: The empire10 study. *Grand Challenges in Medical Image Analysis*. 2010
 29. Berbeco R, Nishioka S, Shirato H, et al. Residual motion of lung tumors in end-of-inhale respiratory gated radiotherapy based on external surrogates. *Med Phys*. 2006; 33:4149–4156. [PubMed: 17153393]
 30. Balter J, Lam K, McGinn C, et al. Improvement of CT-based treatment-planning models of abdominal targets using static exhale imaging. *Int J Radiat Oncol Biol Phys*. 1998; 41:939–943. [PubMed: 9652861]
 31. Coolens C, Webb S, Shirato H, et al. A margin model to account for respiration-induced tumour motion and its variability. *Phys Med Biol*. 2008; 53:4317–4330. [PubMed: 18653921]
 32. Thrust - CUDA toolkit documentation - CUDA V9.0.176. [Online]. Available: <http://docs.nvidia.com/cuda/thrust/index.html>
 33. Kim J, Jeong WK, Nam B. Exploiting massive parallelism for indexing multi-dimensional datasets on the GPU. *IEEE Transactions on Parallel and Distributed Systems*. Aug; 2015 26(8):2258–2271.
 34. TG-119 IMRT commissioning tests instructions for planning, measurement, and analysis version 10/21/2009. [Online]. Available: https://www.aapm.org/pubs/tg119/TG119_Instructions_102109.pdf
 35. Kraus, J. Increase performance with GPU boost and K80 autoboot. 2014. [Online]. Available: <https://devblogs.nvidia.com/parallelforall/increase-performance-gpu-boost-k80-autoboot/>

Figure 1.
NUMA architecture in our computing platform

Figure 2.

The optimization process flow chart highlighting parallelization steps on GPUs (green) and steps where there is parallelization across the CPUs and GPUs (orange). As an example, the memory requirement for creating summed dose matrix for a data matrix of size $196 \times 118 \times 217$ would be 21.65 GB for 200 particles and 10 phases.

Figure 3.
Layout of matrices used for scaling data with particle variable arrays.

Figure 4. 3D view of the target volume is shown along with the organs at risk in its close vicinity (right lung, heart, spinal cord and esophagus) for Case 1. The central axes of the 9 treatment beams are also depicted.

Figure 6. Time analysis for Case 1 using 5 GPUs and N=9 for different swarm populations (a) and data sizes (b) over 1 iteration

Figure 7.

For Case 1, the objective function evolution of a 200-particle swarm over 25 iterations is shown in (a). The OAR dose sparing achieved by the optimized plan over clinical plan is shown in dose volume histograms for planning tumor target volume (PTV), lung and heart in (b) and for esophagus and spinal cord in (c).

Table 1

Time analysis for Case 1 versus number of GPUs for $k=196 \times 118 \times 217$, $N=9$ and $P=200$ over 1 iteration. The speedup factors were calculated in reference to the single-GPU scenario (the first row).

# of GPUs	PSO Time (s)	Partitioning Operator Time (s)	Total Time (s)	GPU Speedup Factor
1	477.55	80.726	558.27	1.00
2	445.80	81.813	527.61	1.05
3	338.58	82.959	421.54	1.24
4	288.31	82.926	371.23	1.33
5	220.66	83.056	303.71	1.46
6	222.14	82.944	305.08	1.46
7	268.85	83.296	352.14	1.39
8	328.62	79.128	407.75	1.26