Towards Breast Anatomy Simulation Using GPUs

Joseph H. Chui[1], David D. Pokrajac[2],
Andrew D.A. Maidment[3], and Predrag R. Bakic[4]

[1] Department of Radiology, University of Pennsylvania, Philadelphia PA 19104
{Joseph.Chui,Andrew.Maidment,Predrag.Bakic}@uphs.upenn.edu
[2] Applied Mathematics Research Center, Delaware State University, Dover, DE 19901
dpokrajac@desu.edu

Abstract. We have developed a method for massively parallelized breast anatomy simulation and a corresponding GPU implementation using OpenCL. The simulation method utilizes an octree data structure for recursively splitting the simulated tissue volume. Several strategies to optimize the GPU utilization were proposed and evaluated, including the use of synchronization constructs in the language and minimization of buffer allocations. The task of tissue classification was separated from the voxelization to further improve the balance of the control flow. The proposed anatomy simulation method provides for fast generation of high-resolution anthropomorphic breast phantoms. Currently, it is possible to generate an octree representation of 450 ml breasts with 50 μm voxel size on a AMD Radeon 6950 GPU with 2GB of memory at a rate of 7 phantoms per minute, 32 times faster than a multithreaded C++ implementation.

Keywords: Digital mammography, anthropomorphic breast phantom, Parallelization, GPU.

1 Introduction

Breast tissue simulation is of great importance for pre-clinical testing and optimization of imaging systems or image analysis methods. Currently, the standard for imaging systems validation includes pre-clinical evaluation performed with simple geometric phantoms, followed up by clinical imaging trials involving large numbers of patients and repeated imaging using different acquisition conditions. Such an approach frequently causes delays in technology dissemination, due to the duration and cost of these trials. In addition, there are many factors which place strict limitations on the number of test conditions, such as the use of radiation in x-ray imaging trials.

Use of software anthropomorphic phantoms for pre-clinical evaluations offers a valuable alternative approach which can reduce the burden of clinical trials. In this paper, we present a GPU (Graphical Processing Unit) implementation of a method for generating software anthropomorphic breast phantoms. The breast anatomy simulation method is based upon recursive partitioning of the simulated volume utilizing octrees. The octree-based algorithm allows generation and processing of octree nodes at the same tree level independently (i.e., in any arbitrary order), which makes the

A.D.A. Maidment, P.R. Bakic, and S. Gavenonis (Eds.): IWDM 2012, LNCS 7361, pp. 506–513, 2012.

algorithm a good candidate for parallelization. Using profiler analysis we have identified the bottleneck steps in the CPU implementation of the algorithm and developed a corresponding GPU implementation using OpenCL. The performances of the GPU and CPU implementations were compared in terms of the time needed for generating phantoms of various voxel sizes. The effects of several implementation parameters are discussed.

2 Methods

Our proposed method of breast anatomy simulation using GPUs is based on the algorithm originally proposed by Pokrajac et al [1]. The paper proposed a method of using octrees to represent simulated volumes of various tissue types. We recently proposed a roadmap [2] to migrate its implementation to a platform that directly utilizes massively parallel processors such as GPUs. Specific milestones were defined to allow incremental migration in implementations and regression testing. A multiple threaded, concurrent version targeting multiple-core CPUs had been implemented along the roadmap. Figure 1 shows the flowchart of this version of algorithm.

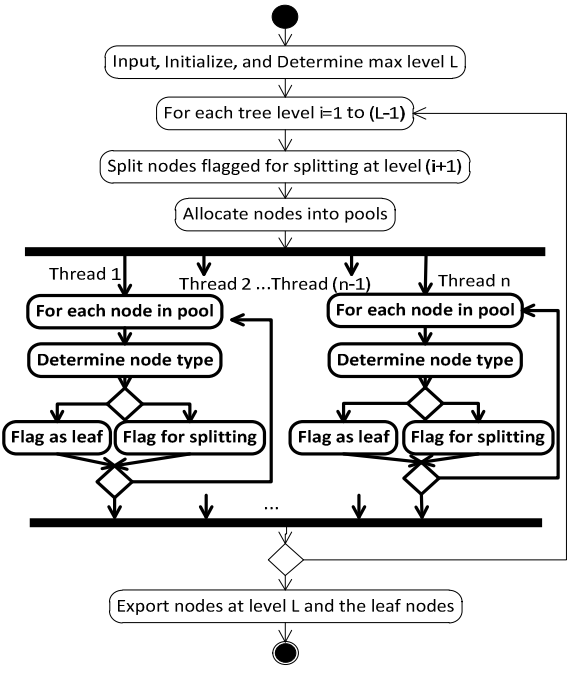

Fig. 1. Flowchart of the concurrent version of the octree-based algorithm, where nodes are processed concurrently to determine their tissue types

We chose OpenCL [3] as our software platform to implement a massively parallel version of the algorithm. Each individual octree node is identified as the finest granularity in the parallelization. To map it to OpenCL, each OpenCL work item is indexed to a unique node at each tree level. The concurrent part of the algorithm is ported into OpenCL kernels which are functions invoked and executed by the GPUs.

Profiling was performed on the initial OpenCL implementation to identify its potential bottlenecks using AMD APP SDK v2.6 [4]. The data transfer between the host memory and the device memory was identified as the major bottleneck in the pipeline. To reduce the amount of data transferred between the host and the devices, the process of splitting the nodes into child nodes was ported as an OpenCL kernel, so that the uploading of octree data to the devices was no longer needed. Figure 2 shows a float chart where the node splitting is parallelized on the GPU.

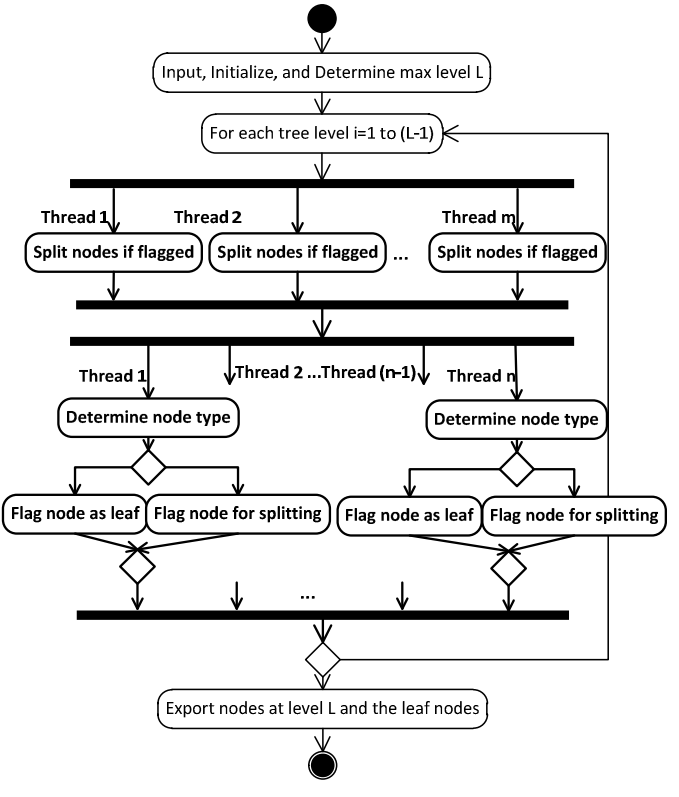

Fig. 2. A massively parallel version of the octree algorithm. At each octree level, two parallelized steps are performed. The first step is to split each splitable node into 8 child nodes. The second step is determining the tissue type of each node.

Because OpenCL does not allow allocation of memory by its kernels, buffers of sufficient sizes have to be allocated by the host in advance. Therefore, the GPU implementation has to determine, in advance, the number of octree nodes requiring for splitting. A technique similar to reduction [5] is used to accelerate the counting

process. The implementation first counts the number of nodes which require splitting in each work group using a counter in local memory. Next, the counts of each workgroup are accumulated so that the accumulation result multiplied by 8 would be the index where each workgroup starts splitting its nodes in parallel. Figure 3 shows an example of the parallelized splitting process.

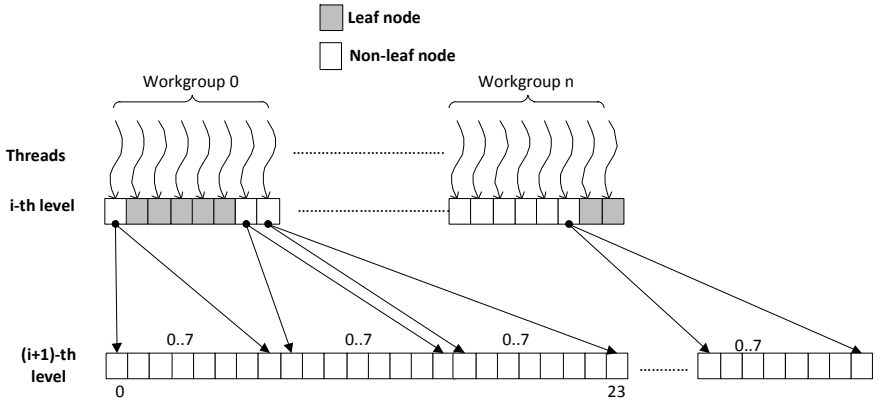

Fig. 3. Illustration of GPU threads splitting its each node into eight nodes in parallel. In this example, workgroup 0 has 3 nodes (0, 6, and 7) requiring splitting. Indexes 0 to 23 (= 3 x 8 - 1) are reserved for workgroup 0, while the next workgroup splits the nodes into child nodes starting from index 24.

Built-in OpenCL atomic functions `atom_inc()` and `atom_add()` were utilized to increment and add the counters on multiple threads to guard against a race condition.

During software profiling, several other GPU-specific bottlenecks were also identified. First, buffer allocations on GPUs require significant time. Secondly, excessive use of flow control in the kernels running on the GPUs slows down the execution of work groups.

To address the buffer allocation problem, instead of re-allocating new buffers for every level of octrees, buffers were retained on the devices until the current ones were no longer big enough for the next tree level. This was especially effective for phantoms of high resolution, where the buffers created for an octree section could often be reused for subsequent sections.

To tackle the issue of excessive use of flow control, the OpenCL kernels implemented in this study were refactored manually. Programming methods using branching that are designed for sequential computation are often unsuitable for parallel computation [6]. Instead, costly functions called on different control paths can be consolidated into a single call on the main path.

Our concurrent, non-parallel version of the algorithm conditionally voxelizes volumes on some of its control paths based on each node's tissue types. The whole workgroup is blocked when there is a work item in this group requires voxelization of

its octree node. To improve the utilization of the GPU, the voxelization was separated from the kernel that determines each node's tissue type.

We validated the implementation by comparing the generated octrees with the ones generated by previous implementations using the same set of parameters. In order to assess the performances of various implementations, the simulation times at different target resolutions were compared. We also measured the effects of workgroup sizes on the performance. Performances of the implementations were assessed by their duration times on a desktop PC with Intel® Core™ i7-2600K CPU @ 3.40GHz and 16GB of RAM and Radeon 6950 GPU with 2GB of VRAM.

3 Results

Figure 4 shows the orthogonal sections of a phantom with 400 μm and 50 μm voxel resolutions. With the same inputs, the identical octrees were constructed by the different implementations.

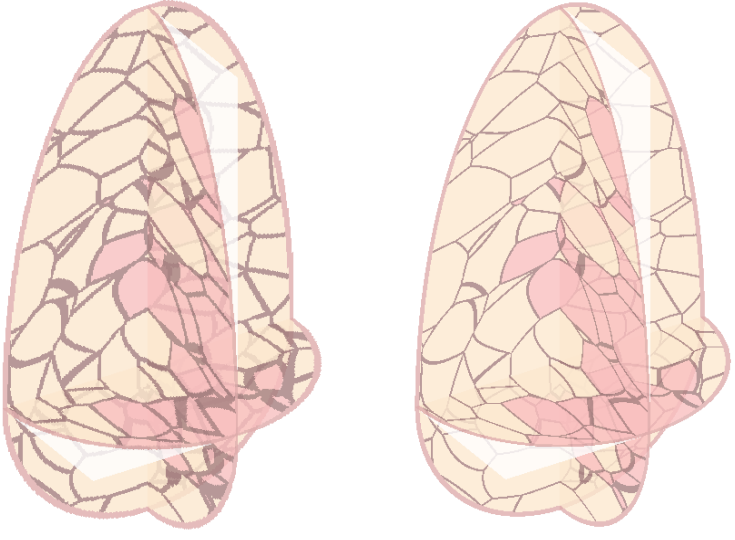

Fig. 4. Orthogonal sections of a simulated breast phantom of (a) 400μm and (b) 50μm resolutions

The performance of the OpenCL implementation was assessed by comparing the duration times to generate phantoms of various voxel sizes. The duration time of each configuration was measured by averaging the duration times of 5 independent phantoms; each phantom was generated from a different set of ellipsoids modeled randomly inside the simulated breast. Figure 5 is a graph showing the duration times of 2 implementations at different voxel resolutions. Figure 6 shows the duration times measured for 25μm resolution using different OpenCL workgroup sizes.

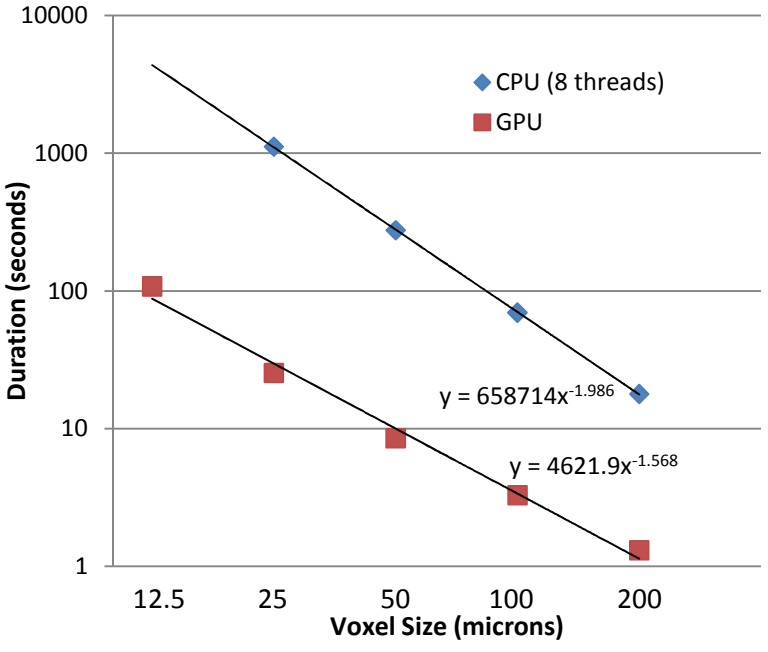

Fig. 5. Average duration times of different implementations of the octree-based algorithm for various voxel sizes (12.5, 25, 50, 100 and 200 μm)

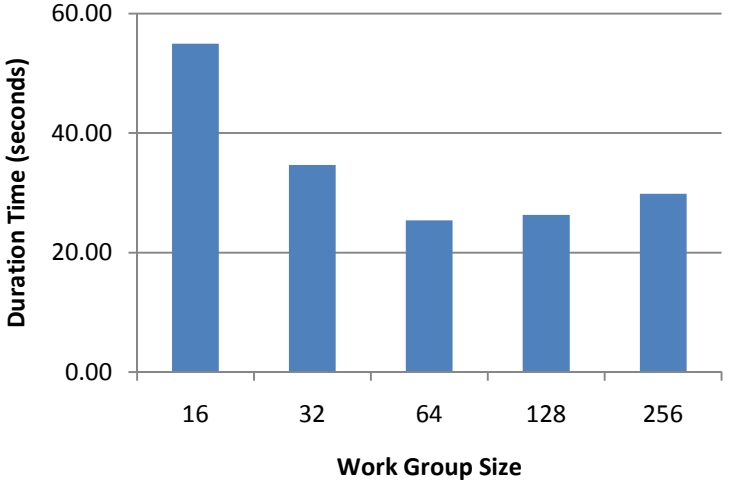

Fig. 6. The duration times using different OpenCL workgroup sizes (16, 32, 64, 128, and 256)

4 Discussion and Conclusions

We have successfully implemented an efficient parallelized version of an algorithm to simulate the breast anatomy for anthropomorphic phantoms by utilizing some of the strategies targeted for GPUs such as reuse of buffers and reduction of flow control. We measured, on average, a 32-fold improvement for the GPU implementation over the multi-threaded CPU implementation when simulating 50 μm phantoms.

Based on the measured duration times using different workgroup sizes, a workgroup of 64 yielded the best performance. Since the GPU used in this study has a wavefront size of 64 work items, any work group size less than 64 may underutilize the GPUs. On the other hand, a workgroup of more than 64 items would increase the memory contention among the units. Since the optimal workgroup size is hardware dependent, benchmarking on individual hardware is required to determine the optimal work group size.

The performance of the implementation is sufficient to create phantoms of reasonably high resolution in near real time. By generating and storing the data on the GPU, it becomes feasible to develop real time visualization software that interoperates with the same set of data on the GPU. This arises, in part, because the octree data structure offers a superior memory footprint compared to a 3D voxel representation. Therefore, an octree is an ideal data structure for storage on GPUs (that are typically available with limited memory). For simulations requiring higher resolution, the simulated phantom can be subdivided into sub-volumes small enough for the individual GPUs.

We observed a CPU usage of 2% by the application when the octrees are generated on the GPU. Thus, porting the code to the GPU not only resulted in the performance being significantly improved, but shifting the processing from the CPU to the GPU frees the CPU for other operations such as voxelization, data compression and I/O. Our GPU implementation can be further enhanced by operating it upon multiple GPUs; a feature supported by most mainstream performance computing hardware. It is noteworthy that it is more feasible to assemble hardware with multiple GPUs than hardware with multiple CPUs.

Our latest profiling results indicate that further improvements in performance can be achieved by extending the parallelization to the evaluation of shape functions for each octree. Please note that the estimated slope of the dependence of the computation time vs. voxel size for the GPU implementation (Fig. 5) is less than two. The computation time consists of two components. The first component, related to building and maintaining the octree structure of the phantom, is believed to be quadratic function of the inverse voxel size [1]. The second component includes overhead of initializing the OpenCL kernels that has linear or constant complexity as a function of the inverse voxel size. For larger voxel sizes, this linear component becomes dominant, influencing the estimate slope of the regression line.

It is further observed that when the resolution is sufficiently high, the duration increased slightly more than a quadratic as a function of the inverse voxel size. This is caused mainly by the overhead of the data transfers between the host and the devices, which accrue a cost proportional to the cube of the inverse voxel size. For simulations that require resolutions higher than 25 μm, further investigations of performance

improvement are needed. Such work should emphasize the reduction of the cost of operations for each sub-volume, such as voxelization and communication between the host and devices. Finally, the frequency of buffer allocation on the devices can be reduced if an accurate maximum buffer size can be estimated in advance for different sets of parameters.

Acknowledgements. This work was supported in part by the US Department of Defense Breast Cancer Research Program (HBCU Partnership Training Award BC083639), and the US National Institutes of Health (grant 1R01CA154444). The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding agency.

References

1. Pokrajac, D.D., Maidment, A.D.A., Bakic, P.R.: Optimized generation of high resolution breast anthropomorphic software phantoms. Medical Physics 39(4), 2290–2302 (2012)
2. Chui, J.H., Pokrajac, D.D., Maidment, A.D.A., Bakic, P.R.: Roadmap for efficient parallelization of breast anatomy simulation. In: Pelc, N.J., Nishikawa, R.M., Whiting, B.R. (eds.) Proc. of SPIE, Medical Imaging 2012: Physics of Medical Imaging, vol. 8313, pp. 83134T-1–83134T-10, SPIE, Bellingham (2012)
3. OpenCL 1.2 Specification, Khronos Group, `http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf`
4. AMD APP SDK v2.6, `http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx`
5. Harris, M.: Optimizing parallel reduction in CUDA, `http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`
6. AMD Accelerated Parallel Processing OpenCL Programming Guide (v1.3f), `http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf`