Accelerating Spatial Clustering Detection of Epidemic Disease with Graphics Processing Unit

Sisi Zhao, Chenghu Zhou
Institute of Geographic Sciences and Natural Resources
Research, Chinese Academy of Sciences
Beijing, China
ricepig@gmail.com

***Abstract*—The statistics of disease clustering is of interest to epidemiologists. In order to detect spatial clustering of disease in all the regions of China, we adopted a likelihood ratio based method which utilizes Monte Carlo simulation and spatial exploring to analyze the real time updating data stored in database. However, large number of random tests for Monte Carlo simulation and large scale of the data set had made the speed of analysis too slow to detect and monitor potential public health hazards. Therefore, we explored to adopt graphics processing unit (GPU) and compute unified device architecture (CUDA) to accelerate the spatial exploring and analyzing process. The algorithm has been implemented efficiently on GPU and the access pattern to memory has been optimized to exploit the computing power of GPU. As a result, the GPU based spatial exploring and likelihood ratio test program performed more than forty times faster then the CPU implementation. The Monte Carlo simulation on GPU performed around thirty times faster than the counter part on CPU. By using GPU and CUDA, the usage of our application is changed from verification after the event to early warning.**

Keywords-Spatial analysis; GPGPU; Monte Carlo simulation; Clustering detection; Epidemic disease

I. INTRODUCTION

The statistics of disease clustering is of interest to epidemiologists and has been studied for many decades [1]. The results of the statistics have been used to further analysis for exploring the spreading pattern of diseases or early warning [2][3]. Such studies are useful to detect and monitor potential public health hazards.

A scan statistic is used to detect clusters in a point process. It has been studied in the one-dimensional setting [4], and has been extended in various directions [5][6]. Kulldorff proposed a spatial scan statistic to detect clusters in spatial process [1][4]. It is a likelihood ratio based spatial scan statistic for detecting clusters in a multi-dimensional point. The baseline process is Bernoulli or Poisson process. In this paper, we employed Kulldorff's method with Bernoulli model to detect spatial clustering of epidemic diseases in China.

Since we took villages as the minimal unit in exploring the large scale dataset, and we adopt the Monte Carlo simulations in inference step which requires large number of random tests, the potential amount of computation is very huge. Traditional implementations based on CPU may be too slow to detect and monitor potential public health hazards. A recent trend in computer science and related fields is the use of general purpose computing on graphics processing units (GPGPU) for reducing the required processing time. Applications with impressive performance improvement achieved in geographic information system can be found in [8-10] and GPU accelerating Monte Carlo simulation with high speedup in other areas can be found in [11][12]. Therefore, we explored to adopt graphic processing unit and CUDA to accelerate the process.

II. GPU AND CUDA

In traditional computer architecture, CPU acts as the only general computing device while graphics processing unit (GPU) is processing graphics. However, the rapidly increasing performance and the flexible GPU programmability make general purpose graphics unit processing a new trend in both high performance computing market and the consumer market. In the current generation, The peak performance of GPU has been leading an order of magnitude to CPU: The peak floating point performance of common GPU is around 800 giga floating-point operations per second (GFLOPS), while the common peak CPU floating point performance at about 45 GFLOPS [13]. Compared to CPU, the improvement of the GPU's performance is very impressive in some practical applications. On the other hand, the peak power of Geforce GTX 260+ is 182W [14] while the full power consuming of quad-core CPU is around 100w.

In the beginning of general purpose programming on graphics processing units, the typical way to program against CPU was to use shading languages such as OpenGL shading language or DirectX high level shading language. However they are designed for computer graphics and are difficult to program for general purpose computing. The common unified device architecture (CUDA) released by main stream graphics card manufacturers NVIDIA is a powerful and flexible framework which makes the transition to general purpose GPU computing much easier.

The modern GPU generation uses a graphics card architecture, which is composed of a scalable array of so-called streaming multiprocessors [15]. The basic units of executable code for CUDA are so-called kernels. Each kernel is written in a specific dialect of the C language (CUDA C) and is executed by the host (CPU). Whenever a kernel is executed, several instances of this kernel are created and mapped onto the streaming processors on GPU to be executed concurrently.

Supported by the Major Program of National Natural Science Foundation of China (Grant No. 40830529)

Each kernel instance is called a thread and can be identified by some CUDA predefined variables. Threads are organized into thread blocks for synchronization and communication purposes. The threads in a given block execute concurrently within a single multiprocessor while blocks execute concurrently among multiprocessors.

CUDA enabled GPU is consist of several multiprocessors, and each of them is consist of 8 stream processors. Stream processors in the same multiprocessor perform the single instruction multiple data (SIMD) pattern on execution. So the concept wrap was introduced for thread execution scheduling. CUDA uses warps of 32 threads as the fundamental unit for execution on a single multiprocessor to hide latencies instead of wraps of 8 threads.

There are several types of memory in CUDA architecture: register memory, local memory, shared memory, global memory and texture memory. Each thread has its own register memory and local memory. These two types of memories can only be accessed by the local thread. Shared memory is shared among threads within the same block. Global memory can be accessed by all threads in every thread blocks at the highest latency. Texture memory is the same as global memory except that it is read-only and has lower latency.

III. SPATIAL SCAN STATISTICS

As described in the introduction, we employed Kulldorff's method with Bernoulli model as the baseline spatial distribution model [1][4]. In this section, some details of this method will be stated.

Let G denote the study area, and Z is used to denote a subset of G. Each individual within that zone has probability p of being a point, while the probability for individuals outside the zone is q.

The μ is a measure which is defined as the population of specified areas, e.g. $\mu(Z)$ denotes the population in zone Z. Let n_z denote the observed number of disease cases in zone Z, and n_G the total number of observed cases. The likelihood function for the Bernoulli model is expressed as (1).

$$L(Z,p,q) = p^{n_Z}(1-p)^{\mu(Z)-n_Z} q^{n_G-n_Z}(1-q)^{(\mu(G)-\mu(Z))-(n_G-n_Z)} \quad (1)$$

And the likelihood ratio, λ, can be written as (2).

$$\lambda = \frac{\sup_{\hat{Z}\in Z, p>q} L(\hat{Z},p,q)}{\sup_{p=q} L(Z,p,q)} \quad (2)$$

In order to find the value of the test statistic, we need a way to calculate the likelihood ratio as it is maximized over the collection of zones. Once the value of test statistic has been calculated, we rely on Mote Carlo simulation for inference. The detailed steps are stated below:

- Step 1: For each village v in the study area G, we took v as the center of circles, and generated a series of circular zone with equally increased radiuses. Then we abandoned the zones which contains only a few diseases cases or villages to void small zones or zones with sparse diseases.
- Step 2: We calculated likelihood ratios for each of the circular zones obtained in the first step. For concentric circular, we eliminated all of them except for the one with maximal likelihood ratio. After this process, we obtained a collection of circular zones with different centers.
- Step 3: The zones obtained in the second step may overlap with each other. In this step, we iterated the zones and eliminated the overlapped zones according to the following criteria: If two circles are overlapped with each other, then reserve the one with greater likelihood ratio.
- Step 4: For the remaining zones, we employed Monte Carlo simulation to testify the P-value for each zone, and then reserved the zones with significance probability. These circular zones are the so-called hot spot regions.

IV. GPU IMPLEMENTATION

In this section, we present an implementation of Kulldorff's spatial scan statistic process with Bernoulli model on GPU. The process is consisting of two different parts: spatial exploring and likelihood ratio test part and the Monte Carlo simulations part. Thus the section is divided into two parts accordingly.

A. Spatial Exploring and Likelihood Ratio Test

Taking the degree of parallelism into consideration, spatial exploring and the likelihood ratio test part of the program was divided into two parts: the first part corresponds to the step 1 and the step 2 stated in the previous section and the second part corresponds to the step 3. Thread divisions for these two parts are completely different.

For the first part, we took the process of exploring concentric circular regions and filtering them based on likelihood ratio as a kernel. We assigned each village as the center point for each thread; in other words, threads for this kernel are different in center points.

Assuming that the number of villages for exploring is relatively small, if the number of blocks is decided only by the number of villages, then the number of blocks would be too small to take full advantages of stream multiprocessors on GPU, and it couldn't effectively hide memory access latency. It would lead to low performance and ineffective execution of GPU. Therefore, the number of blocks was decided by both the number of the exploring villages and the number of diseases. In other words, each time the application calls the kernel, the analyses for multiple diseases can be processed simultaneously.

For the second part, the exclusion of non-overlapped screening was performed based on non-concentric circular regions which had been obtained in the first part: If two regions are overlapped, then leave the region with greater likelihood ratio. We had these process mapped onto one kernel and for

each thread the process is based on different regions. Since data sharing and inter-operation among threads are necessary in this part, candidate regions from global memory were copied to shared memory. These sharing and copying operations had caused a problem: all the exclusion operations need to be processed within a single block. In order to avoid performance loss caused by insufficient block, we used the same method as part one: we processed several kinds of diseases simultaneously, and the analysis for each kind of diseases performed in a single block.

It is important for GPU to carefully align the input and output data structure to achieve better memory throughput. In addition, binding read-only data from global memory to texture memory may lead to additional performance improvement since GPU has built-in cache for texture memory for better performance for access. Therefore, we had the input data aligned to 16 bytes and bound the data of village coordinates, population and cases of disease which are read-only and frequently accessed to texture memory.

B. Monte Carlo Simulation

An efficient algorithm to generate random numbers parallel is essential for Monte Carlo simulations implemented on GPU. By referring to [11], we used an array of linear congruential random number generators (LCRNGs) applying classic algorithms to generate pseudo random numbers [16]. In a single thread j, starting at a seed value s_j, a sequence of random numbers x_i with N can be obtained by (3).

$$x_{i+1} = (a \cdot x_i + c) \bmod m \tag{3}$$

Where $x_0 = s_j$; a, c and m are integer coefficients. A careful decision should be made in choosing these coefficients to guarantee the quality of the random numbers. We used a = 1664525 and c = 1013904223 as suggested in [17]. The modulo operation m was set to the number of villages in the study area.

As we are using a set of LCRNGs in parallel, each LCRNG was initialized by a random number obtained by another LCRNG as suggested in [11]. The s_j was calculated using (4).

$$s_{j+1} = (16807 \cdot s_j) \bmod m \tag{4}$$

For each Monte Carlo simulation of a single region, the cases with specified types were distributed onto the whole study area based on the aforementioned series of random number. Then we traversed the study area to calculate the likelihood ratio for simulated distributions. In the next step, we compared the likelihood ratio with the real likelihood ratio of the region. Finally, the results of all these comparisons were summed up for calculating P-value and we extracted adequate regions based on the P-value.

In order to implement Monte Carlo simulation on GPU, we took each Monte Carlo simulation for a single region as a kernel. Because of a large amount of Monte Carlo simulations, the number of blocks is sufficient to exploit the performance of the multiprocessors in GPU.

In the next step, we utilized share memory within blocks for parallel reducing operation to calculate P-values among threads. There is no communication among blocks, and CPU is responsible to calculate the final P-value based on the P-values retrieved from blocks.

Similar to the implementation for spatial exploring and likelihood ratio test part, we aligned input data structure to 16 bytes and bound the read-only data (such as coordinates, population, cases, etc) to texture memory. Secondly, we take the characters of memory coalescing into consideration in storing and accessing random number series, and they were arranged carefully to achieve better memory bandwidth. Finally, in the reducing process, we managed to avoid bank conflicts by optimizing the access pattern to the results of the Monte Carlo simulations which had been stored in shared memory.

V. PERFORMANCE EVALUATION

The following benchmark was executed on an Intel Core 2 Quad CPU Q9550 running at 2.66GHz with NVIDIA GTX 260+ GPU. All test had been built with the Microsoft 64-bit C++ compiler version 14 and CUDA version 2.3. The block size for CUDA was set to 512.

A. Spatial Exploring and Likelihood Ratio Test

Table I, Table II and Table III display timings for a large problem set of 3 to 81 kinds of disease in 500 villages. Table I shows the comparison of the runtimes between GPU implementation and CPU implementation for spatial exploring. Table II shows the comparison in likelihood ratio test process and Table III is for the whole process.

The performance is quite impressive for spatial exploring implemented on GPU. The maximum speedup ratio is up to 80 and the GPU implementation runs almost two orders of magnitude faster than CPU's. The more kinds of diseases are processed in a batch, the greater speedup ratio is achieved until there are more than 27 kinds of diseases. As we stated in the implementation section, the spatial exploring processes among different diseases and different villages are completely independent with each others and have great similarity in instruction level. According to the gap in peak performance between GPU and CPU, such impressive performance gain for this part is reasonable.

TABLE I. RUNTIME COMPARISION FOR SPATIAL EXPLORING

Diseases	GPU	CPU	Speedup
3	17.708	272.194	15.37
9	19.786	751.870	38.00
27	26.318	2180.579	82.85
81	77.584	6551.567	84.44

The unit for GPU column and CPU column are millisecond

There is another point needs to be mentioned: the runtime of GPU implementation for less than 27 kinds of diseases

increases slightly while that of CPU implementation increase linearly. This is because NVIDIA GTX 260+ GPU has 27 multiprocessors. According to the GPU implementation we stated in the implementation section, we had 500 villages assigned to 500 threads, and each block has 512 threads. Therefore, each multiprocessor is able to handle one type of disease entirely. If less than 27 kinds of diseases are processed in a batch, some multiprocessors in GPU are not assigned work and will be idle until the next batch.

The performance for likelihood ratio test part is not so impressive as the spatial exploring part. The speedup ratio is only around 5 at peak, and the GPU implementation runs even a bit slower than CPU implementation when there are only a few kinds of diseases. This is caused by three reasons: the first reason is that the memory access pattern for this part is random access on which depends the input data. Thus it is impossible to optimize the code for memory coalescing to achieve better memory performance. The second reason is the communication between threads is unavoidable in this part, e.g. the exclusion operation of non-overlapped screening is cross-thread. The last reason is code divergence. Branching is expensive to GPU so that it will significantly reduce the performance of programs running on GPU. In conclusion, the algorithm and implementation of this part is not as suitable for GPU as for CPU. However, we still achieved performance gain from GPU.

For the whole process of this part, the speedup ratio is up to 40 when there are many kinds of diseases are analyzed concurrently. More accurate results are shown on Table III.

Fig. 1 shows the speedup ratios for spatial exploring part (Part 1), likelihood ratio test part (Part 2) and the whole processing. According to Fig. 1, the program running on GPU reaches the peak speedup in analysis of 27 kinds, 54 kinds and 81 kinds of diseases against CPU implementation. As mentioned before, this is because NVIDIA GTX 260+ GPU has 27 multiprocessors. If the number of diseases is a multiple of 27, the GPU is fully utilized for computing. Otherwise, some multiprocessors are idled and the computing power is wasted.

TABLE II. RUNTIME COMPARISION FOR LIKELIHOOD RATIO TEST

Diseases	GPU	CPU	Speedup
3	22.162	15.425	0.70
9	23.085	45.708	1.98
27	26.897	137.412	5.11
81	87.960	428.918	4.88

The unit for GPU column and CPU column is millisecond

TABLE III. RUNTIME COMPARISION FOR THE WHOLE PROCESS

Diseases	GPU	CPU	Speedup
3	38.727	252.058	6.51
9	42.466	786.665	18.52
27	54.953	2265.998	41.24
81	164.427	6836.915	41.58

The unit for GPU column and CPU column is millisecond

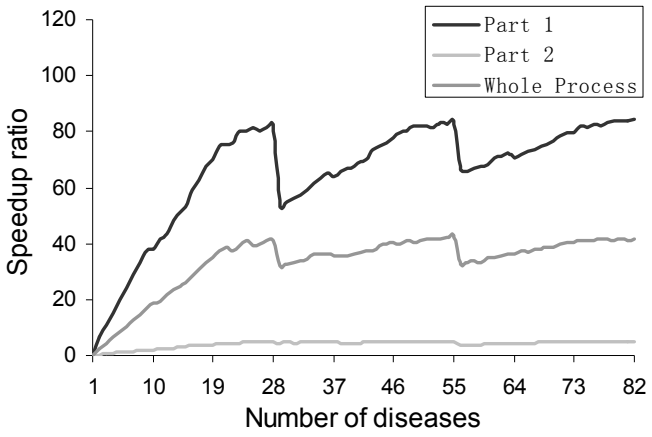

Figure 1. Speedup ratios for spatial exploring part, likelihood ratio test part and whole processing

B. *Monte Carlo Simulation*

Table IV shows the timings for a large scale of Monte Carlo simulation of one disease within a study area which is consist of 500 villages. The times of Monte Carlo simulation are from 1024 to 65535. Fig. 2 is the line chart for the runtime of the same process.

The performance of GPU implementation is fairly better than that of CPU's. The speed up ratio increases along with the growth of the times of tests for Monte Carlo simulations. The peak speedup ratio 30 is achieved on running 65535 times of Monte Carlo tests simultaneously. In this case, there were plenty of threads and blocks scheduled to be executed on GPU so that the memory access latency can be hidden adequately and all the multiprocessors on GPU are fully utilized for computing.

TABLE IV. RUNTIME COMPARISION FOR MONTE CARLO SIMULATION

Times of Monte Carlo Simulation	GPU	CPU	Speedup
1024	9.436	104.191	11.04
2048	12.765	214.695	16.82
4096	19.847	468.606	23.61
8192	41.715	1044.690	25.04
16384	81.545	2277.796	27.93
32768	156.167	4622.015	29.60
65536	303.797	9381.943	30.88

The unit for GPU column and CPU column is millisecond

However, the speedup ratios of this part are relatively lower than that of the spatial exploring part. There are two reasons which lead to relative lower performance of this part: firstly, series of random numbers were generated and the random distributions of diseases were created by the random numbers and stored in global memory. According to the pattern of this stage, the access to global memory was fairly random that

caused high latency and relatively inefficient memory access. For the second reason, the process of creating the random distributions of diseases by random numbers caused a certain degree of code divergence which is relatively not suitable for SIMD pattern employed by modern GPU.

Fig. 2 demonstrates that the speedup ratio of the Monte Carlo simulation part alters not so periodically and dramatically as that of spatial exploring part. This is due to the computing pattern of this part. The memory access time was playing an important role against the time spent on arithmetic. Thus the bottleneck of this part is the memory access speed which depends on memory access latency and memory bandwidth. The computing power of GPU was not fully utilized since some multiprocessors may wait for the data retrieved from memory.

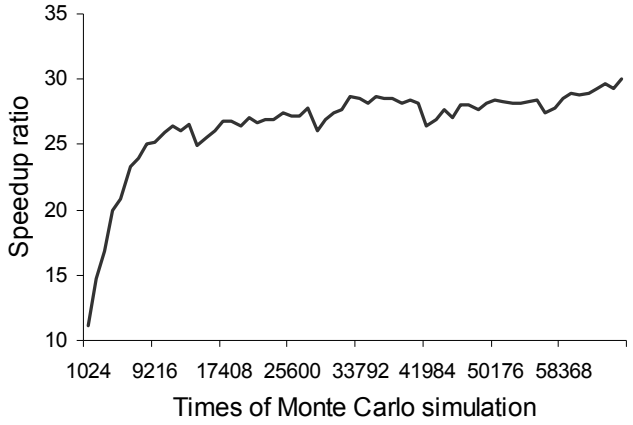

Figure 2. Speedup ratios for Monte Carlo simulation

VI. CASE STUDY

In this section, we applied our GPU implemented spatial scan statistics program on the epidemic disease dataset of China in the year 2008. There are up to forty thousand villages and dozens of diseases in the dataset. All the cases of diseases in the dataset are organized by day. We selected the data of three days as samples. For reasons of confidentiality, we are not permitted to expose the accurate dates, so Sample 1, Sample 2 and Sample 3 represent the selected datasets of three days.

TABLE V. RUNTIME COMPARATION FOR REAL DATASET

	GPU	CPU	Speedup
Sample 1	2.641	28.532	10.80
Sample 2	2.359	24.036	10.19
Sample 3	2.408	25.750	10.69

The unit for GPU column and CPU column is hour

The relevant parameters used in GPU implementation were determined as follows:

- The number of diseases in each computing batch was set to 27 to exploit computing power of each stream multiprocessors of NVIDIA GTX 260+ GPU.
- The thread number of each block was set to 256. This is the tradeoff between idling multiprocessors and costing of communication between threads.
- The number of villages explored in each computing batch was set to 1024 in consideration of the scale of the dataset and the capacity of video memory.

The comparison between the runtime of GPU implementation and that of CPU's is shown in the Table V. The speedup ratios are around ten, and the GPU implementation runs an order of magnitude faster than CPU's.

The speedup ratios when running against the real dataset are relatively lower than that of test data in the previous section. The reason is: the time spent on the data access operations against the database which stores the dataset was remarkable and equivalent between CPU and GPU implementations.

Noticing that the analysis running on CPU always costs more than 24 hours while the dataset is updated everyday, the results from CPU can only be used as verification after the event. In sharp contrast to this situation, the analysis running on GPU only costs several hours, this response time is sufficient for early warning.

VII. CONCLUSIONS AND FUTURE WORK

In this work, a likelihood ratio bases spatial scan algorithm for the detection clusters has been described. In order to manage the inefficiency of the implemention of this algorithm on CPU, the GPU implemention has been introduced and carefully optimized. In the spatial exploring and likelihood ratio test part of the GPU implemention, parallel processing was among not only villages but also diseases. In the Monte Carlo simulation part, all the sample tests were running simultaniously to exploit the computing power of GPU. And then the benchmark between CPU implemention and GPU implementation has been performed. The speedup ratio around 40 for the spatial exploring and likelihood test part and the ratio around 30 for the Monte Carlo simulation part were achieved by the GPU implementation agaist CPU's. The analysis about the differences of performance among stages has been given and the reason why the speedup ratio altered dramatically has been proposed in many aspects: code similarity and divergence, memory access pattern and idling multiprocessors. Finaly, the GPU implementation had been applied on the real world dataset, and its feasibility and high efficiency had been verified.

As a future work, we are working on three aspects:

- On the aspect of floating point precision. Our present implementation on GPU is based on single precision floating number. however, higher precision would be required in some cases. Calcuations for numbers with double precision have been supported by current GPU. but they are inefficient and is around an order of magnitude slower than the counter part base on single precision in the current GPU generation. we will

migrate our GPU implementation to double persicion once the calcuations for double persicion became faster in the next GPU generation.

- On the aspect of heterogeneous computation. Our present implementation is only for GPU while the traditionaly implementation is only for CPU. Although GPU one runs faster, an implementation running on both CPU and GPU would be much better. The next implementation will utilize both CPU and GPU simultaneously.
- On the aspect of hardware independence. Our present implementation on GPU is base on NVIDIA CUDA and can only run on NVIDIA's GPU. Further work is needed for hardware independent GPU implementation which can run on the GPU provided by other manufacturers besides NVIDIA.

REFERENCES

[1] M. Kulldorf, and N. Nagarwalla, "Spatial disease clusters: detection and inference," Statistics in Medicine, vol. 14, pp. 799–810, April 1995.

[2] B. Simon, C Sian, N. J. Kiambo, P. Sarah, M. Benbolt, et al. "Spatial clustering of malaria and associated risk factors during an epidemic in a highland area of western Kenya," Tropical medicine & international health, vol. 9, pp. 757–766, July 2004.

[3] M. Kulldorf, "A spatial scan statitic, " Communication in Statistics, vol. 26, pp. 1481–1496, 1997.

[4] J. I. Naus, "The distribution on the size of the maximum cluster of points on a line," Journal of the American Statistical Association, vol. 60, pp. 532–538, 1965.

[5] S. Wallenstein, C. R. Weinberg, and M. Gould, "Testing for a pulse in seasonal event data," Biometrics, vol. 45, pp. 817–830, 1989.

[6] C. R. Loader, "Large-deviation approximations to the distribution of scan statistics," Advances in Applied Probability, vol. 23, pp. 751–771, 1991.

[7] F. Mostashari, M. Kulldorff, J. J. Hartman, J. R. Miller and V. Kulasekera, "Dead bird clusters as an early warning system for West Nile virus activity," Emergeing Infectious Diseases, vol. 9(6), pp. 641–646, June 2003.

[8] Y. Wu, Y. Ge, W. Yan, and X, Li, "Improving the performance of spatial raster analysis in GIS using GPU," Proc. SPIE, vol. 6754, pp. 67540P.1–67540P.11, 2007.

[9] V. garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pp. 1–6, 2008.

[10] A. Bleiweiss, "GPU accelerated pathfinding," Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, pp. 65–74, 2008.

[11] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, "GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model," Journal of Computational Physics, vol. 228, pp. 4468–4477, 2009.

[12] E. Alerstam, T. Svensson, and S. A. Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration," Journal of Biomedical Optics, vol. 13, pp. 060501P.1–060504P.3, 2008.

[13] Intel Corporation, Intel microprocessor export compliance metrics, 2009.

[14] NVIDIA Corporation, Geforece GTX 260 specification, 2009.

[15] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, version 2.0, 2008.

[16] J. J. Schneider, S. Kirkpatrick, "Stochastic Optimization," Springer, Berlin, 2006.

[17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, "Numerical recipes: the art of scientific computing," Cambridge University Press, Cambridge, 2007