Parallel Implementation of Inverse Adding-Doubling and Monte Carlo Multi-Layered Programs for High Performance Computing Systems with Shared and Distributed Memory

Svyatoslav Chugunov, Changying Li*

Bio-Sensing and Instrumentation Laboratory, College of Engineering, University of Georgia, 200 D.W. Brooks Dr., Athens, GA 30602, USA.

*Corresponding author: Changying Li, Associate Professor, College of Engineering, 712F Boyd Graduate Studies, University of Georgia, Athens, Georgia 30602. Tel: +1 706 542 4696; Email address: cyli@uga.edu

Abstract

Parallel implementation of two numerical tools popular in optical studies of biological materials - Inverse Adding-Doubling (IAD) program and Monte Carlo Multi-Layered (MCML) program – was developed and tested in this study. The implementation was based on Message Passing Interface (MPI) and standard C-language. Parallel versions of IAD and MCML programs were compared to their sequential counterparts in validation and performance tests. Additionally, the portability of the programs was tested using a local high performance computing (HPC) cluster, Penguin-On-Demand HPC cluster, and Amazon EC2 cluster. Parallel IAD was tested with up to 150 parallel cores using 1223 input datasets. It demonstrated linear scalability and the speedup was proportional to the number of parallel cores (up to 150x). Parallel MCML was tested with up to 1001 parallel cores using problem sizes 10^4 - 10^9 photon packets. It demonstrated classical performance curves featuring communication overhead and performance saturation point. Optimal performance curve was derived for parallel MCML as a function of problem size. Typical speedup achieved for parallel MCML (up to 326x) demonstrated linear increase with problem size. Precision of MCML results was estimated in a series of tests - problem size of 10^6 photon packets was found optimal for calculations of total optical response and 10^8 photon packets for spatially-resolved results. The presented parallel versions of MCML and IAD programs are portable on multiple computing platforms. The parallel programs could significantly speed up the simulation for scientists and be utilized to their full potential in computing systems that are readily available without additional costs.

Keywords

Parallel computing, Monte Carlo, simulation, Inverse Adding-Doubling, tissue, photon

Program summary

Manuscript Title: Parallel Implementation of Inverse Adding-Doubling and Monte Carlo Multi-Layered Programs for High Performance Computing Systems with Shared and Distributed Memory

Authors: Svyatoslav Chugunov, Changying Li

Program title: MCMLMPI, IADMPI

Licensing provisions: free license

No. of lines in distributed program, including test data, etc.: 3 668 (MCMLMPI), 18655 (IADMPI)

No. of bytes in distributed program, including test data, etc.: 314 827 (MCMLMPI), 7 299 762 (IADMPI)

Distribution format: zip

Programming language: C

Computer: Variable: up to and including HPC/Cloud CPU-based clusters

Operating system: Variable: Windows, Linux, Unix, MacOS – requires ANSI C-compatible compiler

RAM: Variable: from megabytes to gigabytes (MCMLMPI), kilobytes to megabytes (IADMPI)

Classification: 2.2, 2.5, 18

External routines: dcmt-library (MCMLMPI), cweb-package (IADMPI)

Nature of problem: Photon transport in multilayered semi-transparent material, estimation of optical properties (IADMPI) and optical response (MCMLMPI) of multilayered material samples

Solution method: Massively-parallel Monte-Carlo method (MCMLMPI), Inverse Adding-Doubling method (IADMPI)

Unusual features: Tracking and analysis of photon packets in turbid media (MCMLMPI)

Running time: Many small problems can be solved within seconds, large problems might take hours even on HPC clusters (MCMLMPI); hours on single computer and seconds on HPC cluster (IADMPI)

Key words: Parallel computing, Monte Carlo, simulation, Inverse Adding-Doubling, tissue, photon

Introduction

Study of light transport in turbid media is a significant topic in bio-optical research. The typical subjects of interest in such studies are optical responses (reflectance, transmittance, and absorbance) and optical properties (coefficients of absorption and scattering as well as scattering anisotropy factor) of biological tissues. While basic optical responses for an examined tissue is acquired experimentally, data processing and modeling of complex cases require special numerical tools. Inverse Adding Doubling (IAD) method [1] and Monte Carlo Multi Layered (MCML) simulation program [2] are often employed at this stage as prospective tools for numerical study of optical parameters of biological materials. In the context of tissue examination, IAD handles conversion of optical responses into optical properties and MCML uses optical properties to simulate light transport in complex multi-layered biological tissues. Principles and numerical implementations of sequential version of IAD are well explained in the work of Scott Prahl [1, 3] and details on sequential implementation of MCML are given by Lihong Wang [4] and Steven Jacques [5]. In the past years, sequential versions of these tools were extensively tested by the scientific community [1, 6-11] and were accepted as gold standard for study of light propagation in turbid media. At the same time, necessity of more productive parallel versions of these tools was identified for processing of plentiful optical data generated by the spectrometric imaging technique and for comprehensive study of complex multi-layered tissues under different simulated scenarios. Numerical processing of spectrometric data with IAD and subsequent simulations with MCML are time consuming tasks: typical computation time for IAD processing is within hours; for MCML this time is in the order of tens of hours. Study of different case-scenarios using MCML increases processing time to days. Parallel implementation of IAD and MCML attempted in this study is intended to significantly accelerate the computation process and acquire simulation results within short period of time.

Prior to the attempts of parallel conversion we considered available parallel versions of IAD and suitable Monte Carlo (MC) programs. To date there was no parallel version of IAD available. However, there existed a few parallel MC-based programs simulating photon transport [2, 12-15] in different types of materials. Versions of parallel MC programs for photon migration were previously proposed by a variety of authors [16-21]. For example, Page et al [16] developed a Java-based application with in-house-made MC-code and tested it on a distributed network of general purpose personal computers. Colasanti et al [17] converted MC-code designed earlier by their group into a parallel version using F90 version of Fortran-language. They used “SHared MEMory access library” that brings communication time in distributed systems close to that of shared memory systems, using low-level features of hardware developed by Silicon Graphics Inc. Lo et al [18] developed hardware-based simulations of photon propagation in 5-layered materials using field-programmable gate arrays (FPGA) electrically driven in accordance with principles implemented in MCML. Badal et al [19] proposed a set of Linux scripts that distribute MC-code (or any other sequential code) over a network of computers, execute the code via SSH commands, and gather results after computations are finished. Wang et al [20] implemented MPI-based parallel scheduler in EGS5 MC-code [14] and tested it on Amazon Elastic Cloud Compute (EC2) cluster. Pratz et al [21] evaluated MC321 MC-code [15] using Hadoop framework [22] deployed over Amazon EC2 cluster. They adopted MapReduce [23] programming model to provide parallel capabilities to their program. Among these programs, only MCML provided accurate, well-tested, and simple approach to simulate samples that have geometry of a parallel slab – such geometry is specific to biological materials. Based on the current literature, there existed two parallel implementations of MCML: the first one was developed by Alerstam et al [24] with parallel optimization focused on General Purpose Graphic Processing Units (GPGPU) [24]; the second one was developed by Lo et al [18] and featured hardware-based implementation. The GPGPU version [24] was closely bound to NVidia chipsets due to substantial low-level optimization introduced into the code and the use of NVidia’s genuine Compute Unified Device Architecture (CUDA). While demonstrating significant speedup (up to 621x at 480 CUDA cores), it was designed to work only with NVidia hardware and did not allow easy modification of the code as the addition of new features for scientific

applications compromised the low-level optimization. The hardware version of MCML [18] was able to simulate light absorption in 5-layered model of human skin featuring 80x acceleration in comparison to sequential version of MCML executed on a 3GHz Intel Xeon processor. This version was focused on simulation of only light absorption in tissues, dropping light reflectance and transmittance in favor of the overall performance of the system. According to Lo et al, it was subject to memory constraints, and required 1 person-year development time using the known MCML algorithm to produce simulations for 5 layers of tissue. The listed MC-programs for simulation of light propagation in multi-layered biological tissues have limited applicability and GPGPU-optimized and FPGA-optimized MCML implementations have hardware restrictions. These limitations facilitated the authors' decision to proceed with parallel conversion of MCML which was based on x86 instruction set, standard C-language, and Message Passing Interface (MPI). Compliance with these standards made the parallel MCML suitable for execution at most High Performance Computing (HPC) clusters, as well as on regular desktop computers with multi-core CPUs. In addition, it provided endless possibilities for easy modification of the parallel code.

We exercised a conservative approach in parallel conversion of IAD and MCML: alterations of the original code were kept at minimum in the presented parallel versions to sustain conceptual integrity of the programs' core. Compliance with HPC standards was maintained at a high level which ensured compatibility of the parallel versions with a broad range of computing systems. Because no parallel version of IAD existed, it was advantageous to apply MPI-based conversion approach to both IAD and MCML, sharing some details of parallel implementation. The overall goals of this study were to develop accurate parallel versions of IAD and MCML, as well as to demonstrate their great efficiency and portability. Specific objectives were to: (1) convert IAD and MCML programs into parallel versions using MPI-libraries and keeping the core generic algorithms of these programs intact; (2) test the parallel code against its sequential counterpart verifying correctness of the conversion and demonstrating the parallel advantage; (3) estimate optimal problem size and optimal number of parallel cores for parallel IAD and MCML using local and remote HPC clusters; (4) demonstrate the portability of parallel MCML using a desktop computer, local and remote HPC clusters, as well as a loosely-coupled Amazon EC2 cluster.

Parallel implementation of IAD

An MPI-based parallel wrapper was developed for IAD program exploiting specific features of the generic algorithm. IAD program processes multiple independent entries of the optical response listed in the input file on one-by-one basis and outputs corresponding sets of optical parameters as a result. Because of independence of the entries, the input file could be split into a series of smaller files each containing only a portion of the original input. The small files are generated in accordance with the number of available parallel processes and the original entries are distributed within these files uniformly, providing a simple load balancing mechanism. Each parallel instance executed a copy of modified IAD code (Figure 1). The core of IAD code was maintained relatively unaltered, only a few modifications were introduced: 1) text output to display was suppressed for parallel instances to not produce overlapping messages, 2) initialization of MPI library was added to main() function, 3) timing of code performance was included, and 4) file name convention for input/output subroutines was altered to account for rank of parallel instances. Because of the last alteration, each parallel instance used its own dedicated input file designated with rank marker and produced a unique output file. An additional code was developed to cover two steps of parallel conversion of IAD: the code that splits the original input file into a series of smaller input files ("File_Splitter" in Figure 1) and the code that joins a series of small output files into one output file ("File Joiner" in Figure 1).

Parallel Implementation of MCML

Bio-photonic simulations based on Monte Carlo algorithm are comprised of multiple repetitions of photon packet's travel through turbid media and its interactions (specifically, reflection and refraction at boundaries, scattering and absorbance in the media) with the tested material. In the main cycle of MCML code, individual photon packets are treated independently; their contribution to the overall optical response of the sample is scored during the simulation

and averaged when simulation is finished. Such numerical algorithm allows intuitive approach to parallel implementation of MCML – photon packets must be distributed equally among parallel processes where the main computation cycle takes place; their response should be scored but not averaged until all packets at parallel processes are simulated. By the end of the simulation, optical response from all parallel processes should be collected at one dedicated node and averaged. Flow-chart of parallel conversion procedure developed for MCML is outlined in [Figure 2](#).

In [Figure 2](#), parallel processes are referred to as cores, in agreement with a good HPC practice of assigning a separate CPU core for each parallel instance. In the proposed scheme, one core (rank 0) was reserved for Master process and the rest of the cores (ranks 1...N-1) were served as Solvers. The functions of Master process were to handle the initialization stage (read input data from hard drive, modify and broadcast input data to Solvers, and generate initialization records for parallel Mersenne Twister) of parallel MCML, provide user interface to control simulation process in real-time, and produce correct post-processing of MCML results. The functions of the Solver processes were to acquire inputs from the Master, perform Monte-Carlo simulations of photon packets' travel through the media, and transfer the result to the Master process.

A few changes that were necessary for parallel conversion were introduced into the original MCML code. An additional section responsible for parallel operations was added in the form of a separate file "mcmlmpi.c". This section contained a number of parallel functions, including parallel "main()" implementation. The original "main()" function was modified to provide transition to parallel "main()" when parallel execution was specified. Disabling of a pre-processor flag responsible for engagement of parallel mode returned parallel MCML to the original sequential version. The parallel version of "main()" did initialize MPI library and distributed jobs among Master and Solver processes in accordance with their rank.

As for any Monte Carlo-based method, accurate operation of MCML depends on the performance of pseudo-random number generator (RNG). The standard RNG implemented in MCML provides relatively good results when executed within a single instance of the code. When parallel execution is in question, the generator produces random sequences containing same "random" numbers - such behavior is due to initializing the generator with a timestamp which, in most cases, is not unique across parallel instances. Standard RNG was abandoned in this study in favor of the state-of-the-art parallel Mersenne Twister (MT) [25] random generator that was employed for parallel version of MCML to generate pseudo-random numbers with a period of up to $2^{19937}-1$. Dynamic creation [26] of MT generators ensured availability of the necessary number of disjoint random sequences for parallel execution of the code. A special library dcmt v.0.6.1, distributed with MT code, was employed at the Master process to generate unique MT initialization records in accordance with the number of Solvers used in the simulations. These records were shared with the Solver processes via "MPI_Send()" messages. Master process also logged these records into a data-file for future reference, as such logging assisted in repeatability of numerical experiments. Each Solver process received the corresponding MT initialization record and invoked its own unique sequence of random numbers.

The input data provided to MCML by a user was modified by Master process to account for the correct number of photon packets that need to be processed by each Solver. These data were broadcasted to Solver processes via "MPI_Bcast()". At this step, a dedicated buffer was initialized by Master and Solver processes to exchange results of calculation as well as the timing data for performance evaluation. Solvers started the main cycle of Monte Carlo simulations in accordance with the algorithm developed and implemented by Wang [2] - no changes were introduced into the core of MCML code to ensure smooth and accurate transition from the original version of MCML to its parallel counterpart. By the end of each Monte Carlo cycle, Solvers submitted the collected results to Master using "MPI_Gather()" method. As the final step, Master averaged the simulated data replicating the same approach as was employed in the original MCML code, wrote data into output files, and logged information regarding code performance.

Testing Parallel Code

In order to differentiate sequential and parallel versions of IAD and MCML in the context of comparison tests, the sequential versions were referred to as IAD-Single and MCML-Single (due to the code being executed on a single CPU core), whereas parallel versions were referred to as IAD-MPI and MCML-MPI. Standard numerical samples for comparison tests were artificially devised for both IAD and MCML. For MCML, the sample (Table 1) was designed using optical responses $R = T = 1/3$ – this response is located at the center of R - T triangle (boundaries: $R = 0$, $T = 0$, $R + T = 1$); it also falls within the typical response range of biological tissues. Three thickness values were considered to form three subsamples: 0.01 cm, 0.1 cm, and 1 cm. The first and the second values replicated typical thickness of biological tissues, the third value replicated thickness of a typical cuvette holding optical phantoms in bio-optical experiments. Collimated transmittance was set at 10% of total transmittance value and refractive index (n) was 1.34. The optical response of the standard sample was processed with IAD program and corresponding optical parameters μ_a (absorption coefficient), μ_s (scattering coefficient), and g (scattering anisotropy factor) were generated for the three subsamples – these parameters constituted input to the MCML program. Greater number of thickness values was initially considered for the tests, but was reduced to the three presented cases due to the dramatic increase in computation time when $\sim 10^9$ photon packets and small number of parallel cores were used in MCML. A small number of subsamples were found adequate for the MCML testing as the core algorithm of the program remained intact.

For IAD, the range of applicability of reflectance (R) and transmittance (T) was encircled by boundaries: $R = T = 0$ and $R + T = 1$. The entire applicability range was discretized with a step of 0.02 along R and T axes resulting in 1326 pairs of $\{R, T\}$. Unphysical values of $R = 0$, $T = 0$ and $\{R, T\}$: $R + T = 1$ were removed from the data set; two elements $\{0.02, 0.92\}$ and $\{0.04, 0.96\}$ that caused problems with scattering anisotropy factor in IAD were also removed. The final set of parameters designed for the tests consisted of 1223 pairs of $\{R, T\}$ covering almost the entire range of applicability of reflectance and transmittance. Such large number of data-points is typical for studying optical properties of biological tissues, where spectral analysis of tens of samples often results in thousands of measured $\{R, T\}$.

Software and Hardware configuration

Numerical tests presented in this work used default configurations of IAD-Single and IAD-MPI with no command-line options applied, the only exception was made for the number of integrating spheres which was specified in the input file as zero. Under the listed conditions, computation time for IAD was only determined by the number of datasets in the input file. Computation time of MCML-Single and MCML-MPI depended on the number of datasets in the input file as well as the problem size (the number of photon packets) that had to be simulated. The problem size was deduced from $9! = 362880$, because this value is divisible by any number in the range from 1 to 150 with the only exception for prime numbers greater than 9. Such selection allowed broad variation of the number of parallel cores in the range of 1-150 while keeping the number of photon packets balanced among CPU cores. Problem sizes representing orders of magnitude, 10^4 , 10^5 , 10^6 , 10^7 , 10^8 , and 10^9 were defined as 9×10^{-1} , 9×10^0 , 9×10^1 , 9×10^2 , 9×10^3 , and 9×10^4 , respectively. To avoid potential confusion, problem sizes were referred to by their order of magnitude, i.e. $10^4 - 10^9$.

Configuration of hardware employed for numerical tests was the major factor in evaluation of performance of parallel programs. Usually CPU frequency, the size of cache and RAM, and inter-node connection speed are taken into account when performance of parallel code is estimated. Both MCML and IAD are methods that do not demand large amount of RAM. Effect of cache size can be taken into account when optimization of the core algorithms of these programs is performed. Since this study focused rather on parallel conversion than on optimization of the core algorithms, estimation of cache-size effect was found unnecessary.

CPU has the greatest effect on the performance of IAD and MCML due to the fact that most of the time these programs are calculating optical properties of a material in the main cycle. Computational units employed for numerical tests featured modern microarchitecture such as Intel's Sandy Bridge, Ivy Bridge, Nehalem, and Westmere with CPU frequency in the range of 2.4 - 3.2 GHz and cores' number in the range of 4 - 8 cores per CPU.

Due to independent propagation of photon packets in the media, data exchange between parallel processes in the main cycle of MCML-MPI was limited. Minor communication events spiked at the end of the main cycle when simulation results were sent to Master process via short MPI messages. Major data exchange took place at initialization and post-processing stages of the code. Similar network use was attributed to IAD-MPI. To ensure efficient parallel performance of the programs, computational facilities with 1 Gigabit, 10 Gigabit, and Infiniband connections were selected for the testing. The 1 Gigabit connection was implemented at a local HPC cluster at the Georgia Advanced Computing Resource Center (University of Georgia, GA). The Infiniband connection was used at a remote Penguin-On-Demand HPC cluster - both of these facilities had tightly-coupled parallel topology built over a low latency network. The 10 Gigabit network was implemented at Amazon EC2 cluster. While the speed of this connection was relatively high, tight-coupling of hardware nodes could not be guaranteed in cloud compute model. Therefore, additional effects due to distant hardware nodes were possible in performance metrics of the code executed at the Amazon EC2 cluster.

Table 2 summarizes four hardware configurations employed in the numerical tests of IAD and MCML. A Desktop Computer was used for development of parallel versions of IAD and MCML, their preliminary evaluation, and validation tests. A local HPC cluster with 160 hardware cores managed by a variant of Sun Grid Engine queuing system was used for Performance and Precision tests. Both a remotely accessed Penguin-On-Demand (POD) cluster and Amazon EC2 cluster were used for evaluation of code compatibility. Additionally, POD cluster was used for assessing MCML-MPI performance for large number of parallel processes (51...1001 parallel cores). POD cluster featured parallel jobs' managing system based on PBS Torque. Amazon EC2 did not have jobs' managing system – initial configuration of the cluster and parallel computations at Amazon EC2 were performed through SecureShell protocol using "--hostfile" option for "mpirun" command.

Validation test

Validation test was performed to ensure accurate parallel conversion of IAD and MCML. For IAD-Single and IAD-MPI five runs were performed using the standard sample as the input. Results of the five runs were averaged with calculation of mean and standard deviation values. Mean values corresponding to single and parallel versions of IAD were compared within 1223 entries of the standard sample.

Validation tests for MCML required five simulations of standard sample using problem sizes 10^4 - 10^9 . Results of five runs for each problem size were averaged for MCML-Single and MCML-MPI. Mean and standard deviation values for total reflectance, transmittance, and absorbance were calculated and plotted for comparison. Granting the stochastic nature of Monte Carlo-based algorithms a certain mismatch in mean value of results was expectable, especially with small problem sizes. The mismatch decreased with the growth of the problem size.

Validation results for IAD demonstrated a precise match between single and parallel versions of the program for all 1223 entries. Such outcome suggested that no errors were introduced into the algorithm of IAD by parallel conversion procedure. Because of coincidence of sequential and parallel results of IAD for 1223 data-points, no graphical illustration was presented.

Validation results for MCML were assessed individually for reflectance, absorbance, and transmittance. **Figure 3** presents mean and standard deviation values of optical responses for problem sizes 10^4 - 10^7 . Problem sizes greater than 10^7 produced well-coinciding results that graphically appeared similar to results of 10^7 , thus they were not included in **Figure 3**. With small problem sizes (e.g., 10^4 and 10^5) MCML-Single and MCML-MPI demonstrated a

mismatch in mean values and relatively large standard deviation which were due to stochastic nature of Monte Carlo-based algorithm. Despite the mismatch, mean values of sequential and parallel versions were located within one standard deviation of each other – this correlation improved with the increase of the number of photon packets used for the simulation. Relatively large standard deviation implied that no accurate match of MCML results could be achieved for small problem sizes ($<10^6$). When problem size increased to 10^6 and 10^7 standard deviation diminished and the mean values closely approached to each other. Based on the validation results, uncertainty in MCML output could be greatly reduced when problem size 10^6 or greater was used for simulations.

Performance Tests

Performance of IAD-MPI and MCML-MPI was evaluated in two aspects: (i) speedup as a function of the number of parallel cores and (ii) location of performance saturation point as a function of problem size. Speedup, S_p , was defined as a ratio between computation time of sequential and parallel versions of the code (*). Amdahl's law [27] provides a correlation between speedup and parallel fraction (P) of the converted code. Parallel fraction determines a portion of the code that is truly executed in parallel mode, the rest of the code is executed in the sequential form. Calculation of parallel fraction required, besides the speedup, the number of parallel cores, N .

$$S_p = \frac{T_{Sequential}}{T_{Parallel}}, \quad P = \frac{(1/S_p) - 1}{(1/N) - 1} \quad (*)$$

For IAD, speedup was estimated from numerical tests by timing performance of sequential and parallel versions. For MCML, speedup was calculated for each problem size using the computation time of MCML-Single and the time corresponding to optimal number of parallel cores for MCML-MPI. Parallel fraction of the code was estimated in accordance with equation (*). Parallel fraction approaching 1 was considered as an indication of efficient parallel code.

Performance saturation point of MCML-MPI was assessed as a function of the number of parallel cores and problem size. The nature of performance saturation point is to depict the number of parallel cores that deliver the shortest computation time for a particular problem size. A small number of parallel cores set high computational load on each CPU core while communication time remains relatively low. Alternatively, a large number of parallel cores reduce the load of each CPU, but results in long communication time. Optimal number of parallel cores is located between the small and large numbers outlined in this example. It balances computation and communication load and constitutes performance saturation point.

Prior to major evaluations of the code performance, a numerical test was conducted on the desktop PC comparing MCML-Single (1 core) to MCML-MPI (2 cores corresponding to Solver and Master processes). The purpose of this test was to determine whether parallel MCML executed with 1 Solver process is able to represent the sequential version in performance tests - this would enable batch computations that cover the entire test in one run. According to results presented in [Figure 4](#), MCML-Single performed better (smaller computation time was used as a criterion for better performance) than MCML-MPI at small problem sizes (42% faster at 10^4 and 8% faster at 10^5) where communication overhead between Master and Solver processes had adverse effect on the total time of simulation. At large problem sizes MCML-MPI performed better than MCML-Single (4% faster at 10^6 and 6% faster at 10^7) which could be attributed to differences in implementation of RNG in sequential and parallel versions of MCML. As a consequence, when MCML-MPI with 1 Solver process represented MCML-Single, speedup was expected to be slightly overestimated for small problem sizes and slightly underestimated for large problem sizes. Due to the small difference between sequential and parallel versions of MCML at problem sizes 10^5 - 10^7 , performance data of MCML-MPI with 1 Solver process was used to represent MPI-Single in performance tests.

Performance tests for IAD-MPI consisted of five runs of the same simulation for the standard sample using 1, 5, 10, 25, 50, 75, 100, 125, and 150 parallel cores on the local HPC cluster. Results of the code performance were

averaged over the five runs and compared between sequential and parallel versions of the program. According to the results shown in [Figure 5](#), computation time of IAD-MPI scaled linearly with the number of parallel processes. There existed no saturation point for IAD-MPI due to the linear scaling. The number of parallel processes for this program was only limited by the number of entries in the input data-file.

Performance tests for MCML-MPI consisted of 5 runs of simulation of the standard sample on the local HPC cluster using 32 options for the number of parallel processes for problem size 10^4 , 49 options for problem size 10^5 , 54 options for 10^6 , and 55 options for larger problem sizes – each option represented the number of parallel processes in the range 1...151 that served as divisor to the corresponding problem size. Because Master-Solver parallel paradigm was applied in MCML-MPI, the number of parallel processes determined from the options was incremented by 1 to account for Master process. Performance saturation points were collected for problem sizes 10^4 - 10^9 and the performance curve was passed through these points. In the curves shown in [Figure 6](#), the saturation point represents a spot of minimal computation time – it can be clearly seen in [Figure 6.a](#) and is relatively hard to distinguish in [Figure 6.b](#). For instance, although the minimum computation time was reached around 85 cores for problem size 10^7 , the performance curve of this problem size is quite flat between 40 and 80 cores. Using 80 cores provides only a little more speed up, but the user has to use almost twice the amount of computation time as compared to using 40 cores. Therefore, if the user has to pay for the core-hours, it is probably wise to use 40 cores as the optimal core numbers. For problem sizes 10^8 and 10^9 , estimation of performance saturation point required more than 151 parallel cores. Results presented in [Figure 6](#) demonstrate classical performance curves for problem sizes 10^4 - 10^7 with computation overhead to the left of the saturation point and communication overhead to the right.

Optimal number of parallel processes that corresponded to performance saturation points is shown in [Figure 7.a](#) as a function of problem size. The resultant curve helps in quick selection of optimal number of parallel processes for any problem size in the range of 10^4 - 10^9 photon packets. The choice of any other number of parallel processes, besides the optimal one, would necessarily lead to elongation of the computation time due to operation of MCML-MPI in unfavorable mode with high computation or communication overhead. [Figure 7.b](#) illustrates typical computation time of MCML-Single and optimal computation time of MCML-MPI. At large problem sizes ($>10^7$), the difference between results of sequential and parallel versions reached two orders of magnitude.

Speedup data were deduced from results shown in [Figure 7.b](#) for problem sizes 10^4 - 10^7 . For problem sizes 10^8 and 10^9 speedup data were estimated using POD results, which are discussed in the following sections. Based on the shape of speedup curve ([Figure 8.a](#)), there was no advantage of using MCML-MPI for problem sizes smaller than 10^6 . At problem size 10^6 , a significant gain of over 11x was achieved when using MCML-MPI. This gain increased almost linearly with problem size. Speedup value for problem size 10^9 was calculated with 1001 parallel cores which were smaller than optimum. Therefore, the estimated speedup at 10^9 could be improved by using larger number of parallel cores. Almost linear speedup as a function of the number of parallel cores was previously reported [16, 17, 21] for a few other MC-based parallel programs. In this study, the speedup curve approached the ideal linear case (the dashed line in [Figure 8.b](#)) but did not coincide with it. In the referenced works performance results were typically reported for a single MC simulation, while data presented in this study were based on 15 (3 sub-samples and 5 runs per sub-sample) consequent MCML-MPI simulations of R , T , and A . Performance data for the presented tests account for communication between Solver and Master processes with typical MPI-message size in the range of megabytes and for HDD input/output events. These data resemble real life application of the tool, and therefore, linear speedup close to the ideal line was not expected in performance tests of MCML-MPI.

Parallel fraction of MCML-MPI code ([Figure 8.c](#)) was estimated in accordance with Amdahl's law – problem sizes 10^4 - 10^7 corresponded to the performance results acquired from the local HPC cluster and 10^8 - 10^9 corresponded to the results acquired from the POD cluster. Parallel fraction reached relatively large values (>0.9) for problem size 10^6 and approached 1 when problem size increased - parallel advantage of MCML-MPI was attained at large

problem sizes. Dependence of parallel fraction on problem size (Figure 8.c) was in good correlation with previously published results [27].

Precision test

Precision test was conducted for MCML-MPI in order to find a relation between problem size and decimal-point precision of the result. This test did not cover IAD-MPI as this program was not affected by stochastic processes. There were two types of results generated in the MCML program – the spatially-resolved data and total optical response. According to the generic algorithm of MCML, total optical response was acquired via integration of spatially-resolved data over the space of computational domain. Its precision was elevated in comparison to spatially-resolved data due to mathematical properties of integration. Precision of the MCML result was sensitive to the number of photon packets simulated in the program - a straightforward relation was confirmed between the decimal-point precision of Monte Carlo-based results and the problem size.

Precision level of MCML results was determined from values of standard deviation calculated for reflectance, transmittance and absorbance. Mean and standard deviation values were calculated from averaging results of 5 consecutive runs of MCML-MPI performed for problem sizes 10^4 - 10^9 . For spatially resolved analysis only results of the subsample with thickness 1 cm were used to estimate the lower boundary of precision - precision was higher for thinner subsamples. The number of stable decimal places in standard deviation values served as an indicator of the precision level - these data were tabulated as a function of problems size and shown in Table 3. In the case of the total optical response the table contained three numbers in the form of 1-2-3, representing samples of thickness 0.01 cm, 0.1 cm, and 1 cm, respectively. In the case of spatially-resolved analysis, the table contained only one number which corresponded to the subsample of thickness 1 cm.

Problem size 10^6 or greater was necessary to deliver precision of 3 decimal places to the total optical response (Table 3.a). Similar precision level for spatially-resolved response (Table 3.b) could be obtained from simulations of 10^8 photon packets. The demonstrated relation between the level of decimal precision and problem size suggested the use of at least 10^6 photon-packets for modeling the total optical response and 10^8 photon packets for modeling spatially-resolved responses. According to Table 3.b, problem size 10^8 was optimal for spatial analysis as it provided precision level comparable to larger problem size of 10^9 and ten times shorter computation time. For spatially-resolved response, precision was estimated along optical axis of the light beam close to the upper surface of the sample. Departure from this axis led to a decrease of precision due to smaller number of photons that scored at distant locations of computational domain.

Compatibility Test

Numerical tests were conducted at POD and Amazon EC2 clusters to demonstrate the broad compatibility of the MCML-MPI code with different computing configurations as well as to estimate its performance at large number of parallel cores. IAD-MPI was also tested on these clusters, but it showed no compatibility issues and demonstrated the same behavior as was reported in the previous sections. Therefore, this section focuses only on tests of the MCML-MPI code. Figure 9.a illustrates the computation time of MCML-MPI for problem sizes 10^8 and 10^9 estimated at the POD cluster. Performance saturation point was observed for problem size 10^8 at 350 parallel cores. For problem size 10^9 no saturation point was observed because the right branch of performance curve, responsible for communication overhead, still could not be reached at 1001 parallel cores. Thus, MCML-MPI could benefit from >1001 parallel cores at problem size 10^9 . Performance data acquired from POD cluster (Figure 9.a) indicated that the performance saturation point were shifted to the right (less likely to saturate) in comparison to that of the local HPC cluster (Figure 6) because of faster (Infiniband) inter-node connection of the POD cluster.

Performance results depicted in [Figure 9.b](#) revealed longer computation time for POD cluster when compared to the local HPC cluster. This result was analyzed in greater detail in an additional performance test, which is discussed in the next section. Computation time at Amazon EC2 was the longest of the three tested computing facilities. Performance drop at Amazon EC2 could be attributed to two major factors: (i) HVM (Hardware-assisted Virtual Machine) virtualization layer employed at the cloud compute cluster and (ii) physical distance between hardware nodes (loose-coupling). Effect of inter-node distance in cloud network could not be completely alleviated by 10 Gigabit Ethernet connection. Despite the relatively long computation time for POD and Amazon EC2 clusters, MCML-MPI simulations for problem size 10^9 were accomplished at these facilities in less than 200 seconds using 151 parallel processes, and in less than 40 seconds using 1001 parallel processes at the POD cluster. This result can be compared to 11676 seconds that are necessary for MCML-Single to perform similar computations.

Performance results acquired from Amazon EC2 were compared to those of Pratz and Xing [21]. Inverse dependence between computation time and the number of parallel cores was observed in both cases. However, the slope of the inverse dependence reported by Pratz and Xing was 267.7 hours/core using log-log scale, while similar slope calculated for MCML-MPI was ~ 4.7 hours/core. Thus, drop in computation time with increase of the number of parallel cores was smaller for MCML-MPI which could be explained by 10^9 photon packets used in MCML-MPI simulations vs. 10^{11} used in the work of Pratz and Xing and by additional communication overhead in MCML-MPI due to running 15 simulations in a row while the referenced work did not include such communication data.

Additional Performance Test for HPC, Amazon EC2, and POD clusters

Performance results acquired from the POD cluster were analyzed in order to find the reason for longer computation time of MCML-MPI at this facility when compared to generally slower HPC cluster. Time spent by parallel code in initialization section (Init in [Figure 10](#)), the main cycle (Main in [Figure 10](#)), and post-processing section (Post in [Figure 10](#)) was recorded at 51, 101, and 151 parallel cores for problem size 10^9 . The recorded values were normalized to corresponding values acquired from the local HPC cluster and represented in the form of column plot ([Figure 10](#)). In [Figure 10](#), columns corresponding to HPC cluster have the height equal to 1 due to self-normalization. Columns with the height smaller than 1 indicate better performance of corresponding hardware configuration and vice versa.

At the initialization section, Amazon EC2 and POD spent smaller or comparable amount of time than the local HPC cluster. Amazon EC2 spent significant amount of time at the post-processing section when 101 parallel cores were used, which was the consequence of imbalanced traffic load in the cloud network. In contrast, POD spent less time in this section exercising benefits of fast Infiniband connection. The main cycle was the part of the code (section “Main” in [Figure 10](#)) where both, Amazon EC2 and POD, consistently spent larger amount of time than the local HPC cluster. While for Amazon EC2 such behavior was expectable due to the virtualization layer, for POD it was not.

In order to find the reasons for the unusual behavior of POD in the Main section, an additional performance test was conducted evaluating computation time of MCML-MPI at POD and the local HPC clusters with the use of problem size 10^9 and 51 parallel cores. [Table 4.a](#) summarizes the configurations of software and hardware employed in this test. There were 10 runs of each configuration accomplished. Minimal, maximal, and average values of computation time were analyzed with SASTM software Version 9.1.3 [28]. Cases 1 and 2 ([Tables 4.a, 4.b](#)) featured the same MPI-libraries and compilers, but different hardware components and Operating Systems (OS). These cases were designed to assess the effect of hardware and OS of the two facilities. Cases 2, 3, and 4 featured the same hardware and OS and therefore cases 2 and 3 were designed to assess the effect of MPI-library, whereas cases 3 and 4 to assess the effect of compiler.

Discrepancy in hardware and OS resulted in 3.3% average advantage of POD over the local HPC cluster, where POD results were used as a reference. Overall advantage of POD over the local HPC cluster in terms of hardware and OS was in the range of 0.3-9.4% among the 10 runs. Discrepancy in MPI-libraries resulted in 0.15% average difference between MPICH2 1.4.1 and OpenMPI 1.5.5 favoring the former, where OpenMPI was used as a reference. Overall difference for 10 runs spanned over interval -6.9% (favoring MPICH2) to +9.5% (favoring OpenMPI). Discrepancy in compilers produced the greatest effect on code performance – the average difference between the computation time of the code compiled with gcc 4.4.7 and pgcc 12.10-0 was 34.7% (favoring pgcc), where gcc results served as a reference. Overall difference due to compiler selection fell in the range -33.3 – -39.6%, favoring pgcc compiler.

Summarizing the results of the additional test, optimization level of compilers was the prime reason for the performance increase: the use of pgcc compiler corresponded to >33% performance increase in 100% of cases. Implementation of MPI-Libraries was the second major reason affecting MCML's efficiency: 50% of observations demonstrated no difference in performance, 40% of observations favored MPICH2 library by up to 6.9%, and 10% of observations favored OpenMPI library by up to 9.5%. Dissimilarity in hardware components and OS produced mild effect: 20% of observations scored up to 9% of performance increase at POD, 10% of observations scored 6% increase, and 70% corresponded to no more than 4% increase in the computation performance of the POD cluster. These results confirmed the discrepancy between the efficiency of POD and the local HPC clusters depicted in [Figure 11](#). A more detailed delineation could be obtained from studying optimization strategies employed in gcc and pgcc compilers, as well as profiling of the generic algorithm responsible for MCML's core computations. Since the focus of this study was set on parallel conversion of MCML and the main goal of the presented computations was to demonstrate compatibility of the code, no profiling of MCML's core code was attempted.

Discussion

The maximum speedup observed in the performance tests of MCML-MPI reached 326x when using 1001 parallel cores at the POD cluster. This value was underestimated due to non-optimal mode of parallel operation (>1001 parallel cores were required) and relatively inefficient gcc compiler used at POD. In addition, computation time measured in this study refers to the wall-clock time instead of CPU time. Wall-clock time measures the elapsed time between the start and the end of the program, including the time that passes due to programmed delays or waiting for resources to become available. CPU time only measures the time during which the processor is actively working on the execution of the program, which could be substantially shorter than the wall-clock time. Typical speedup figures for other MC-programs for photon transport were 32.2x using 32 processing elements of CRAY-T3E computer [17] to run in-house-made MC-code, 58x using a network of 60 computers for Java-based MC-code [16], 47.2x using 100 nodes of Amazon EC2 cluster (they used 2-core EC2 nodes = 200 virtual cores) to run EGS5 MC-code [20], and 1258x using 240 c1.xlarge nodes (they used 8-core c1.xlarge EC2 nodes = 1920 virtual cores) at Amazon EC2 cluster to run MC321 code [21].

The discussed MC-programs could be compared in terms of their parallel efficiency: $326x / 1001 \text{ cores} = 32.6\%$ for MCML-MPI, 100.63% for [17], 97% for [16], 23.6% for [20], and 65.5% for [21]. In terms of overall efficiency MCML-MPI fits between results of [20] and [21]. Assuming that the use of pgcc instead of gcc increases the performance of the code by 30%, parallel efficiency of MCML-MPI could potentially reach $326x * 1.3 / 1001 \text{ cores} = 42.3\%$ when pgcc compiler is employed. Since saturation point was not reached when estimating the maximum speedup, increase in the number of parallel cores could further enhance the parallel efficiency. Great efficiency values estimated for [16, 17] were due to several reasons. Specifically, Colasanti et al [17] reported high parallel advantage based on individual runs of the parallel code, rather than on values averaged over a few runs. Additionally, they intensively exploited special features of SGI hardware in the way similar to shared memory systems. Page et al [16] did not provide details on the size of communication messages or the type of result returned from Java-clients to Java-server. In the case when communication between clients and server was not included into

the performance metrics or the size of the messages was extremely small, almost linear scaling of parallel code could be expected.

Both IAD-MPI and MCML-MPI share certain advantages of using CPU-based parallel approach. One advantage is the ability to run computations at HPC facilities regardless the actual hardware directly available to scientists. An example of such approach was demonstrated in series of compatibility tests when Penguin-On-Demand and Amazon EC2 clusters were used to cut computation time of MCML simulations 300-fold compared to the sequential code. Configuration time and cost of operation for these clusters were at minimum – POD cluster is pre-configured and immediately available for computations; Amazon EC2 requires certain configuration steps (they take less than 15 minutes), but provides cluster model in the form of “spot-instances” which significantly cuts the cost of operation. Both of these facilities are publicly available and allow management of high performance computations from ordinary computers, such as laptops and office-level desktops.

The second advantage of CPU-based parallel approach is the ability to use any x-86 compatible hardware to run IAD-MPI and MCML-MPI. An MPI-library and compilers for C-language are the only necessary components required for these parallel programs, both of which are distributed in the form of freeware. These components are also installed among standard packages on any HPC cluster. If computations cannot be performed directly on a physical computer due to restrictions of an operating system, a virtual machine can be set up using freely available packages such as VmWare [29] or VirtualPC [30]. A free Linux-based operation system, e.g. Ubuntu [31], and free OpenMPI [32] library could be used to run parallel computations using multiple CPU cores with no hardware alterations necessary. Despite the fact that a virtual machine adds up to the burden of computation time, parallel computations in multi-core virtual machine are accomplished considerably faster than similar computations executed on a physical computer with the use of a single core.

The third advantage of CPU-based parallel code is the ease of implementation of additional features without significant losses in performance. In this aspect, GPGPU-optimized parallel code lacks versatility as all the subroutines in such code feature high-level of optimization for specific hardware [23]. Introduction of additional subroutines that are focused on an arbitrary task, e.g. tracking and analysis of millions of individual photon packets in the simulation, often leads to significant decrease in performance and, sometimes, to rebuilding of the code structure.

Regarding the comparison between CPU and GPGPU-optimized versions of MCML, no performance evaluation was conducted in this study. This was largely because the hardware and software configuration of an HPC-system-in-focus can be always settled in the way to favor one version over the other, hence a comparison test would be always biased. It is matter of availability of computational resources and acceptable cost of operation for researchers when CPU or GPGPU-optimized version is considered for scientific applications. Each of these parallel versions of MCML fills a certain niche in high-performance computing for light-propagation in turbid media. Together, they open opportunities for advances in bio-optical studies.

The presented parallel versions of MCML and IAD programs could be utilized to their full potential in computing systems that are readily available or easy to set up without additional costs. Parallel versions of IAD and MCML are available for public use and distributed under the same license as the original IAD and MCML. They are available for download at http://sensinglab.engr.uga.edu/?page_id=523

Conclusion

Parallel versions of IAD and MCML programs presented in this work reflect an attempt of parallel conversion of well-established and accurate sequential software packages that are extensively used in processing and modeling of optical parameters for biological tissues. The attempted validation, performance, precision, and portability tests demonstrated accuracy and versatility of the developed parallel programs. Compliance with major standards in the

HPC field, such as standard C and MPI specifications, ensured compatibility of the parallel versions with a broad range of HPC-configurations. Considering the fact that most desktop computers used in research facilities have CPUs with at least two cores, computations with IAD and MCML could be accelerated proportionally to the number of cores when MCML-MPI and IAD-MPI are used. Such acceleration could be achieved with no changes in hardware configurations and minimal rearrangement of software components.

Acknowledgements

This study was funded by the USDA NIFA Specialty Crop Research Initiative (Award No. 2009-51181-06010). This study was also supported in part by resources and technical expertise from the Georgia Advanced Computing Resource Center, a partnership between the University of Georgia's Office of the Vice President for Research and Office of the Vice President for Information Technology. Additionally, this study was supported in part by a grant provided by Amazon Web Services and resources and technical expertise provided by the University of Georgia Office of the Vice President for Information Technology.

Authors appreciate the technical expertise provided by Dr. Shan-Ho Tsai in facilitating the access to Penguin-On-Demand HPC Service and her invaluable help in the work with a local HPC cluster at GACRC at University of Georgia. We also thank her for the comments and suggestions that help improve the manuscript.

References

- [1] S.A. Prahl, M.J.C. van Gemert, A.J. Welch, Determining the Optical-Properties of Turbid Media by Using the Adding-Doubling Method, *Appl Optics*, 32 (1993) 559-568.
- [2] L.H. Wang, S. L. Jacques, L. Zheng, MCML – Monte Carlo modeling of light transport in multi-layered tissues, *Comput Meth Prog Bio*, 47 (1995) 131-146.
- [3] S.A. Prahl, The Adding-Doubling Method, in: A. J. Welch, M.J.C.v. Gemert (Eds.) *Optical-Thermal Response of Laser-Irradiated Tissue*, Springer1995.
- [4] L.V. Wang, H.-i. Wu, *Biomedical optics : principles and imaging*, Wiley-Interscience, Hoboken, N.J., 2007.
- [5] S.L. Jacques, Monte Carlo Modeling of Light Transport in Tissue (Steady State and Time of Flight), in: A.J. Welch, M.J.C. van Gemert (Eds.) *Optical Thermal Response of Laser-Irradiated Tissue*, Second Edition, Springer, London, New York, 2011, pp. 109-144.
- [6] D. D. Royston, R. S. Poston, S.A. Prahl, Optical Properties of Scattering and Absorbing Materials Used in the Development of Optical Phantoms at 1064 nm, *J Biomed Opt*, 1 (1996) 110-116
- [7] J.W. Qin, R.F. Lu, Monte Carlo simulation for quantification of light transport features in apples, *Comput Electron Agr*, 68 (2009) 44-51.
- [8] D.G. Fraser, R.B. Jordan, R. Kunnemeyer, V.A. McGlone, Light distribution inside mandarin fruit during internal quality assessment by NIR spectroscopy, *Postharvest Biol Tec*, 27 (2003) 185-196.
- [9] H.Y. Cen, R.F. Lu, Quantification of the optical properties of two-layer turbid materials using a hyperspectral imaging-based spatially-resolved technique, *Appl Optics*, 48 (2009) 5612-5623.
- [10] E. Drakaki, M. Makropoulou, A.A. Serafetinides, In vitro fluorescence measurements and Monte Carlo simulation of laser irradiation propagation in porcine skin tissue, *Laser Med Sci*, 23 (2008) 267-276.
- [11] G. Ma, J. F. Delorme, P. Gallant, D.A. Boas, Comparison of simplified Monte Carlo simulation and diffusion approximation for the fluorescence signal from phantoms with typical mouse tissue optical properties, *Appl Optics*, 46 (2007) 1686-1692.
- [12] S. Ren, X. Chen, H. Wang X. Qu, e.a. G. Wang, Molecular Optical Simulation Environment (MOSE): A Platform for the Simulation of Light Propagation in Turbid Media, *PLoS ONE* 8(4): e61304, 2013.

- [13] D.A. Boas, J.P. Culver, J.J. Stott, A.K. Dunn, Three dimensional Monte Carlo code for photon migration through complex heterogeneous media including the adult human head, *Opt Express*, 10 (2002) 159-170.
- [14] Y. N. H. Hirayama, A. F. Bielajew, S. J. Wilderman, W.R. Nelson, The EGS5 code system, SLAC Report, 2005.
- [15] S. Jacques, Light distributions from point, line, and plane sources for photochemical reactions and fluorescence in turbid biological tissues, *Photochem. Photobiol.*, 67 (1998) 23-32.
- [16] A. J. Page, S. Coyle, T. M. Keane, T. J. Naughton, C. Markham, T. Ward, Distributed Monte Carlo Simulation of Light Transportation in Tissue, *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, (2006) 254.
- [17] G.G. A. Colasanti, A. Kisslinger, R. Liuzzi, M.Quarto, P. Riccio, G. Robberti, F. Villani, Multiple processor version of a Monte Carlo code for photon transport in turbid media, *Comput. Phys. Communi.*, 132 (2000) 84-93.
- [18] W. C. Y. Lo, J.L. K. Redmond, P. Chow, J. Rose, L. Lilge, Hardware acceleration of a Monte Carlo simulation for photodynamic therapy treatment planning, *J. Biomed. Opt.*, 14 (2009) 014019.
- [19] J.S. A. Badal, A package of Linux scripts for the parallelization of Monte Carlo simulations, *Comput. Phys. Communi.*, 175 (2006) 440-450.
- [20] H. Wang, Y. Ma, G. Pratx, L. Xing, Toward real-time Monte-Carlo simulation using commercial cloud computing infrastructure, *Phys. Med. Bio.*, 56 (2011) N175-N181.
- [21] G. Pratx, L. Xing, Monte Carlo simulation of photon migration in a cloud computing environment with MapReduce, *J Biomed Opt*, 16 (2011) 125003-125001 – 125003-125009.
- [22] Apache Hadoop framework, hadoop.apache.org, last accessed December 23, 2013.
- [23] Google MapReduce, <http://research.google.com/archive/mapreduce.html>, last accessed on December 23, 2013.
- [24] E. Alerstam, W. C. Y. Lo, T. D. Han, J. Rose, S. Anderson-Engels, L. Lilge, Next-generation acceleration and code optimization for light transport in turbid media using GPUs, *Biomedical Optics Express*, 1 (2010) 658-675.
- [25] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*, 8 (1998) 3–30.
- [26] M. Matsumoto, T. Nishimura, *Dynamic Creation of Pseudorandom Number Generators*, Springer2000.
- [27] G. Karniadakis, R. M. Kirby, *Parallel scientific computing in C++ and MPI : a seamless approach to parallel algorithms and their implementation*, Cambridge University Press, New York, 2003.
- [28] SAS Institute Inc., SAS/BASE and SAS/STAT software, SAS System for Linux, SAS Institute Inc., Cary, NC, USA, 2002-2003.
- [29] VmWare Player Plus, <http://www.vmware.com/products/player/>, last accessed on September 17, 2013.
- [30] Virtual PC, Microsoft, <http://support.microsoft.com/kb/958559>, last accessed on September 17, 2013.
- [31] Ubuntu, Canonical, <http://www.ubuntu.com/>, last accessed on September 17, 2013.
- [32] OpenMPI, <http://www.open-mpi.org/>, last accessed on September 17, 2013.

Figure captions

Figure 1. Flow chart of parallel IAD algorithm.

Figure 2. Flow-chart of parallel MCML algorithm.

Figure 3. Comparison of results of sequential (one core for MCML-Single) and parallel (two cores for MCML-MPI) versions of MCML using average values of five runs. The test was held on the desktop computer. Shift in Single data along x-axis was artificially introduced to not overlap with Parallel data.

Figure 4. Comparison of computation time between MCML-Single and MCML-MPI using 1 Solver process. The test was held on the desktop computer.

Figure 5. Performance of IAD-MPI estimated at the local HPC cluster. The program demonstrated linear scaling with the number of parallel processes.

Figure 6. Computation time of MCML-MPI estimated at the local HPC cluster (a) for problem sizes 10^4 - 10^6 , (b) for problem sizes 10^7 - 10^9 . Circles represent performance saturation points. For problem sizes 10^8 - 10^9 , more than 151 CPU cores are necessary for evaluation of performance saturation points.

Figure 7 (a) Optimal number of parallel processes estimated at the local and remote HPC clusters as a function of problem size, (b) Comparison of computation time of MCML-Single and MCML-MPI evaluated at the local and remote HPC clusters.

Figure 8. Performance characteristics of MCML-MPI estimated at the local and remote HPC clusters (a) Speedup of MCML-MPI as a function of problem size, (b) Speedup of MCML-MPI as a function of the number of parallel cores for problem size of 10^9 , and (c) Parallel fraction of MCML-MPI as a function of problem size.

Figure 9. (a) Computation time of MCML-MPI estimated at POD cluster for problem sizes 10^8 and 10^9 , (b) Computation time of MCML-MPI estimated at the local HPC, POD, and Amazon EC2 clusters for problem size

Figure 10. Performance of different sections of MCML-MPI code at the local HPC, POD, and Amazon EC2 clusters. Subfigures contain timing data acquired at 51, 101, and 151 parallel cores (the smaller column height the better).

Figure 11. Average results of the additional performance test of POD and the local HPC clusters at 51 parallel cores, using problem size 10^9 . The black columns represent reference value (100%), the gray columns are normalized to the reference. The number in the brackets refers to the corresponding case.

Figure1

Figure2

Figure3

Figure4

Figure5

Figure6

Figure7

Figure8

Figure9

Figure10

Figure11

Table 1. Standard Sample for MCML Tests.

Subsample #	Thickness (cm)	R	T	T_c	μ_a (1/cm)	μ_s (1/cm)	g	n
1	0.01	1/3	1/3	1/30	13.33	322.53907	0.3793	1.34
2	0.1	1/3	1/3	1/30	1.333	32.253907	0.3793	1.34
3	1	1/3	1/3	1/30	0.1333	3.2253907	0.3793	1.34

Table 2. Hardware & Software configuration of a Desktop Computer, local and remote HPC clusters, and Amazon EC2 Cluster used for numerical tests.

Desktop Computer		Local HPC Cluster	
Name	Dell Precision T1650	Name	UGA GACRC zCluster
Number of Cores	1 core was used for “Single” code 2 cores were used for “MPI” code	Number of Cores	1...160
CPU	Intel Core i5-3470 (Ivy Bridge)	CPU	Intel Xeon E5530 (Nehalem)
CPU Clock Speed	3.2 GHz, 4 cores	CPU Clock Speed	2.4 GHz, 4 cores
Memory	4 GB	Internode Connect	1 Gigabit Ethernet
Operating System	Linux Ubuntu 12.10, 64bit	Operating System	Red Hat Enterprise Linux Server release 5.8, 64bit
Parallel Library	OpenMPI 1.4.5	Parallel Library	MPICH2 1.4.1
Compiler	GNU Project C compiler, gcc 4.7.2	Compiler	Portland Group Inc. C compiler, pgcc 12.10-0
POD Cluster		Cloud Compute Cluster	
Name	Penguin-On-Demand	Name	Amazon Elastic Compute Cloud
Number of Cores	50...1000	Number of Cores	50...150 (cc2.8xlarge nodes)
CPU	Intel Xeon X5675 (Westmere)	vCPU (HVM virtualized CPU)	Hardware hyper-thread of Xeon E5- 2670 (Sandy Bridge)
CPU Clock Speed	3.06 GHz, 6 cores	CPU Clock Speed	2.6 GHz, 8 cores
Internode Connect	Infiniband	Internode Connect	10 Gigabit Ethernet
Operating System	Linux CentOS 6.3	Operating System	Amazon Linux AMI 3.4
Parallel Library	OpenMPI 1.5.5	Parallel Library	OpenMPI 1.5.4
Compiler	GNU Project C compiler, gcc 4.4.7	Compiler	GNU Project C compiler, gcc 4.7.2

Table 3. Precision (decimal points) results for (a) Total Optical Response, (b) Spatially-Resolved Data

(a)	Problem Size	R*	A*	T*
	10^4	2-2-2	2-2-2	2-2-2
	10^5	2-3-2	2-2-2	2-3-2
	10^6	3-3-3	3-3-3	3-3-3
	10^7	3-3-3	4-3-4	3-3-3
	10^8	4-4-4	4-4-4	4-4-4
	10^9	5-5-5	5-5-4	5-5-4

(b)	Problem Size	R**	A**	T**
	10^4	1	1	1
	10^5	1	2	1
	10^6	2	2	2
	10^7	2	3	2
	10^8	3	3	3
	10^9	3	4	3

*Results are shown for 3 subsamples of thicknesses: 0.01 cm – 0.1 cm – 1 cm

**Results are shown for the subsample of thickness 1 cm

Table 4. (a) Software and Hardware configurations used in the additional performance test of POD and the local HPC clusters, (b) cases used for comparison in different test categories.

(a)	Case #	Tested Facility	MPI-Library	Compiler	(b)	Tested Effect	Compared Cases #
	1	POD Cluster	OpenMPI 1.5.5	gcc 4.4.7		Hardware + OS	1 - 2
	2	HPC Cluster	OpenMPI 1.5.5	gcc 4.4.7		MPI-Library	2 - 3
	3	HPC Cluster	MPICH2 1.4.1	gcc 4.4.7		Compiler	3 - 4
	4	HPC Cluster	MPICH2 1.4.1	pgcc 12.10-0			