

Parallel operation of Monte Carlo simulations on a diverse network of computers

D R Kirkby† and D T Delpy

Department of Medical Physics and Bioengineering, University College London,
11–20 Capper Street, London WC1E 6JA, UK

Received 6 November 1996, in final form 9 January 1997

Abstract. Monte Carlo simulation methods are frequently used to determine light propagation in tissue and x-ray propagation as well as for solving other non-medically related problems. Such techniques are computationally slow, with the signal to noise ratio improving only as the square root of computation time. We present a method for the design of a Monte Carlo program that is capable of running on up to 24 computers simultaneously, with there being very few restrictions on the computer types as long as they run on a common network. This parallel operation is useful when the run time is expected to be long. A mixture of PCs and Sun workstations have been successfully used. The program as described was designed for the simulation of light transport in tissue, but the technique of achieving simple simultaneous execution on a number of different computers could be used wherever Monte Carlo techniques are used.

1. Introduction

Monte Carlo methods (Lux and Koblinger 1990, Raeside 1976, Andreo 1991) are used extensively in the fields of medical physics for solving problems of light transport in tissue (Keizjer *et al* 1989, Fishkin and Gratton 1993, Farrell and Patterson 1992, Jentink *et al* 1991) and x-ray radiation (Baba-Hamed and Speller 1995), as well as other fields such as the numerical integration of functions (Press *et al* 1992) or solving partial differential equations (Dworsky 1979).

Monte Carlo simulations have been used by many authors for modelling light transport in tissue. They impose few or no restrictions on the measurement geometry and are accurate near sources and boundaries, where other models, such as the increasingly popular finite-element method (Arridge 1993, Schweiger *et al* 1993), based on the diffusion equation, are not valid. In the Monte Carlo method, the source launches photons, which are assumed to be ballistic particles, one by one. Each photon then moves in the tissue for some distance, determined by the scattering coefficient μ_s and a random number before it scatters. It then scatters in some direction determined by a random number and the known scattering properties which may be determined from either measured data on real tissue (Van der Zee 1993), Mie calculations (Mie 1908) or use of the Henyey–Greenstein (Henyey and Greenstein 1937) approximation. After a number of scattering events N , the photon's weight (or intensity) is reduced by

$$I = I_0 \sum_{i=0}^N \exp(-\mu_0 l_i) \quad (1)$$

† E-mail address: davek@medphys.ucl.ac.uk

where I_0 is the initial intensity (usually 1.0), μ_a is the absorption coefficient and l_i is the distance travelled to the i th scattering site. The photon position and direction following each scattering event are calculated by geometry. The photon is followed until it either exits the volume of interest, is detected or has become so weak due to tissue absorption that it may be ignored. The whole process is repeated for a large number of photons, which will typically be 1 000 000 or more, depending on the problem. A number of variance reduction methods have been developed (Lux and Koblinger 1990) to speed the process, but in general Monte Carlo methods are very slow if data of good statistical significance is to be obtained, especially at large distances from the source.

Since the Monte Carlo technique simulates the paths of a large number of individual photons, with the results of one photon history being completely independent from any other, there is no need for all the photons to be simulated on the same processor. A multiple-processor computer can be used, or more than one separate computer, as long as care is taken that the random numbers used are independent (Press *et al* 1992), so each processor does not end up simulating the same data. Miura and Babb (1987) discuss how to do this for programs that use multiple processors.

With the increasing availability of networked computers within organizations, many of which spend considerable time unoccupied, a method of using these computers to simultaneously carry out the same Monte Carlo calculation would be advantageous. This short note describes a technique developed at University College London for this purpose. The technique is useful when the run time on a single computer would be a number of days or longer—it is not suitable for tasks that can be completed quickly on one computer.

This method was developed as a result of problems encountered with an earlier attempt to use Monte Carlo techniques to simulate laser Doppler blood flow, where 13 different Sun workstations were used. Unfortunately, with many computers writing to many different data files, it became difficult to monitor the progress of the calculation. The random nature of the Monte Carlo method meant that each computer could be writing data to its files at any time, making it impossible to safely add the data files from many computers for analysis, without first stopping the processes, summing the data, then restarting the computation, which can be very tedious and error prone.

2. Method

A Monte Carlo program was developed to allow calculation of the temporal profile of light exiting tissue of differing optical properties when the source was a very short pulsed laser. This was to generate data against which to compare experimental measurements of temporal profiles made with a fibre optic coupled cross-correlator (Kirkby and Delpy 1996a, b), enabling the optical properties of tissue to be estimated. The description of the problem to be solved is held in one small ASCII file. This contains the measurement geometry, tissue optical properties (absorption coefficient μ_a and scattering coefficient μ_s), position of source and detector, acceptance angle of fibres etc. The Monte Carlo simulation software reads this from disk, performs the simulation, then writes the photon history results, consisting of intensity, position, pathlength and angle with respect to the detection fibre, into a four-dimensional array of memory. After a minimum of 1000 photons are launched, the results are written to a file on a disk accessible to all computers on the network. One copy of the program runs on each machine desired, usually at reduced priority when being executed on Unix based machines that implement scheduling priorities.

Only one result file is stored on a computer network. Each computer performs the Monte Carlo calculations and writes the photon history results into this one result file. Since two computers editing the one file could easily result in file corruption, the time of day at which any computer may edit the data file is fixed. The first processor to run writes a file named '0' into the directory where the data is stored. The file need not contain any data at all, although when the computer is running under the Unix operating system, the machine name and process identification number (PID) are written into the file in ASCII to enable the user to determine on which machines the program was started. The second processor, on finding the file named '0', writes a file named '1'. A third invocation of the program writes a file named '2' and so on, up to the 24th invocation, which writes the number '23'. The program is written such that if the program was able to write the file '0', it only stores data to disk between the times of midnight and 00:55 am, and not at any other time (until this time the program continues to run and accumulate data into the local machine's memory). The second program to run (which wrote the file '1') can only store data between the times of 01:00 and 01:55, the third stores data between the times of 02:00 and 02:55 etc. This ensures that each invocation of the program can only edit the data file at a time when no other program is doing this. Obviously, this limits the number of separate invocations of the program to 24, but even this could be extended if necessary, by using time steps smaller than one hour. Conversely, if the number of computers available is smaller (say six) it would probably be preferable to write the data once every six hours, rather than once per day, as this would reduce the time that could elapse without the data file being updated. For safety, no programs can edit the data file in the last five minutes of any hour, since time must be allowed for writing the file. If for any reason (such as hardware failure) a process is unable to write the data to disk in the 55 minutes allocated for the task, the data is retained (and continues to further accumulate) in the computer's random access memory (RAM) until the next day.

Although it has not been tested, it should be possible to use a mixture of supercomputers, workstations and personal computers in different countries by use of the internet. For this system to work properly, it is essential that all the computers on a network have the same time stored in their clocks.

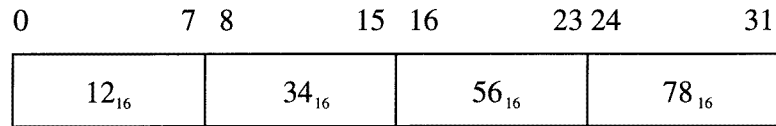
The first time sufficient data has accumulated for the computer to consider writing to disk, the computer continues simulations while waiting for its allocated time slot, whereupon the array used to store the photon histories is simply written to disk. Once a disk file exists with some photon histories, a program ready to write more data to the disk must first wait for the correct time, then read the existing disk file, add this to the data in its memory, then write the memory out, containing results for both the new data and the original data. This ensures that each time new data are ready to be stored on disk, the disk is updated with all the photon histories. The computer name, date, time and PID of the machine making the change is appended to a file called 'update' so that progress can be checked easily.

All the simulations are stopped at the same time—when data of sufficient statistical significance have been collected. No specific limits are put on the number of photons simulated on each computer, or the time each computer program will execute for. Hence a fast and/or lightly loaded computer will collect considerably more data in the same time as a slower and/or heavily loaded machine.

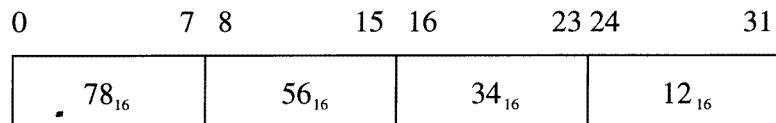
The random number generator used has a long period. This is seeded at a random location for each processor running. Whilst this does not absolutely guarantee that the number streams for each program are independent from another, it is highly probable, as only a small fraction of the total period of the random number generator is used. We intend implementing a method (Miura and Babb 1987) to guarantee this for certain.

2.1. Data storage methods

Since data are usually in the form of double-precision (approximately 16 significant digits) floating point numbers, it is better to store data in binary files, rather than text files, as the former uses eight bytes, rather than 17 to store each number—16 for the data, plus one byte as a separator. If single-precision data are used (seven significant digits), again binary requires less space—only four bytes instead of eight.



4-byte integer 12345678_{16} in 'big-endian' memory format



4-byte integer 12345678_{16} in 'little-endian' memory format

Figure 1. A diagram showing the different byte ordering of the data when a four-byte hexadecimal integer 12345678_{16} is stored in memory on two different computer types. Similar differences occur when floating point data are stored.

3. Using different computers

The system for two or more computers writing to the one data file worked well on our network, when only Sun workstations were used in the Monte Carlo simulation. However, since PCs are frequently left unused for long periods overnight and at weekends, and since PCs are particularly suitable for Monte Carlo simulations (Al-Affan 1996), using these too is beneficial. However, there are two different methods of storing binary numbers in memory on computers. Some computers, such as Suns and any computer originally based on Motorola microprocessors keep data in memory in a format known as 'big-endian'. PCs, or any computer based on Intel microprocessors (or equivalents such as the AMD series) use a different format called 'little-endian'. The differences between these two are shown in figure 1, which illustrates the alternative formats for storing hexadecimal number 12345678_{16} , which contains four bytes of data. Computers using the 'big-endian' format store the number 12_{16} lowest in memory, followed by the number 34_{16} one byte beyond this, followed by 56_{16} one byte further up in memory and finally store 78_{16} . Conversely, a PC, or other computer using the 'little-endian' format, will store the number 78_{16} lowest in memory, with the 12_{16} being stored highest in memory. When performing a binary read or write operation (using for example the C functions *fread()* and *fwrite()*), binary data are written to disk in the same format as it was stored in memory. Hence binary data written on one computer cannot be read on another, unless the byte ordering of the data is reversed. The 'big-endian'/'little-endian' problem affects any data type over one byte in

length, such as integers and floating point numbers, but not characters. To circumvent this, and to allow all Monte Carlo programs to update one data file, all data are written to disk in the ‘big-endian’ format.

A Sun, or other computer using the ‘big-endian’ form of data storage needs only to perform the following three steps if writing data, with no existing data file present.

- (i) Wait for the correct time of day.
- (ii) Write the data to disk, using a standard command for binary writes, such as *fwrite()* in the C language.
- (iii) Zero the array in memory, so only new simulations are later written to disk—the same data that have previously been written.

If however a data file containing previous results already exists, the computer program must perform the following five steps.

- (i) Wait for the correct time of day.
- (ii) Read in the data from disk. If using C, the *fread()* function would be used.
- (iii) Add the data about recent photon histories to those read from disk.
- (iv) Update the disk file by writing out the updated data in ‘big-endian’ format.
- (v) Zero the array in memory.

A PC, or other computer using the ‘little-endian’ format, must perform the following four steps before writing data to disk, if no disk file currently exists.

- (i) Wait for the correct time of day.
- (ii) Reverse the data in memory, so it is in ‘big-endian’ format.
- (iii) Write out the data to disk in the ‘big-endian’ format.
- (iv) Zero all the array locations.

If data already exist on the disk, a PC, or other computer that uses the ‘little-endian’ data format, needs to perform a more complex set of procedures.

- (i) Wait for the correct time of day.
- (ii) Read in the array, which will be in ‘big-endian’ format.
- (iii) Reverse the byte order, to convert to the PC’s ‘little-endian’ format.
- (iv) Add the data about recent photon histories to those read from disk.
- (v) Reverse the byte ordering to convert to ‘big-endian’ format.
- (vi) Update the disk file by writing out the updated data in ‘big-endian’ format.
- (vii) Zero the memory. There is no need to reverse the order prior to zeroing the memory.

The number of times that the data need to be byte reversed could be reduced if temporary storage were used, but this was not felt to be justified, since in practice the byte reversals take little time and are only performed once per day. Note that on PCs running DOS or Windows it is most important to open the binary files for writing using the following:

```
file_pointer=fopen(filename,“wb”);
```

Failure to add the ‘b’ for binary, which is never needed on Unix systems, will result in the files not being correctly written. Similar remarks apply when reading files using *fread()*. This is easy to overlook and will result in about 99% of disk accesses working properly, with just the odd few failing, when a particular byte has some special significance to the PC.

Since the data are stored in the native form used by the Sun workstations, any analysis tools written to run on a PC must take into account the fact that the data are stored in reverse format.

4. Conclusions

A simple method has been presented that allows a number of computers to work in parallel to solve Monte Carlo simulation problems. The method is easy to implement if data are stored in text, rather than binary files, but this will increase storage area requirements by approximately a factor of two. If binary files are used, which results in saving disk space, the problem becomes more complex as some computers store the higher bytes in low memory, but others do the reverse.

References

- Al-Affan I A M 1996 A comparison of the speeds of personal computers using an x-ray scattering Monte Carlo benchmark *Phys. Med. Biol.* **41** 309–13
- Andreo P 1991 Monte Carlo techniques in medical radiation physics *Phys. Med. Biol.* **36** 861–920
- Arridge S R 1993 A finite element model for modelling photon transport in tissue *Med. Phys.* **20** 299–309
- Baba-Hamed T and Speller R D 1995 The effect of scattered radiation in dual-energy analysis *Phys. Med. Biol.* **40** 1619–32
- Dworsky L N 1979 *Modern Transmission Line Theory and Applications* (New York: Wiley)
- Farrell T J and Patterson M S 1992 A diffusion theory model of spatially resolved, steady-state diffuse reflectance for the noninvasive determination of tissue optical properties *in vivo Med. Phys.* **19** 879–88
- Fishkin J R and Gratton E 1993 Propagation of photon density waves in strongly scattering media containing an absorbing semi-infinite plane bounded by a straight edge *Opt. Soc. Am. A* **10** 127–40
- Heney L G and Greenstein J L 1937 Diffuse radiation in the galaxy *Astrophys. J.* **86** 70–83
- Jentink H W, de Mull F F M, Hermsen R G A M and Greve J 1991 Monte Carlo simulations of laser Doppler blood flow measurement in tissue *Appl. Opt.* **29** 2371–81
- Keizjer M, Jacques S L, Prah S A and Welch A J 1989 Light distribution in artery tissue: Monte Carlo simulations for finite-diameter laser beams *Lasers Surg. Med.* **9** 148–54
- Kirkby D R and Delpy D T 1996a Measurement of tissue temporal point spread function (TPSF) by use of a gain modulated avalanche photodiode detector *Phys. Med. Biol.* **41** 939–49
- 1996b An opto-electronic cross-correlator using a gain modulated avalanche photodiode for measurement of the tissue temporal point spread function *Proc. Optical Society of America* **2** 108–12
- Lux I and Koblinger L 1990 *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations* (Boca Raton, FL: Chemical Rubber Company)
- Mie G 1908 Beitrage zur Optic trüber Medien spieziell kolloidaler Methhallösungen *Ann. Phys., Lpz.* **25** 377–445
- Miura K and Babb R G 1987 Tradeoff in granularity and parallelization for a Monte Carlo shower code (EGS4) *Parallel Comput.* **8** 91–100
- Press W H, Teukolsky S A, Vetterling W T and Flannery B P 1992 *Numerical Recipes in C* 2nd edn (New York: Cambridge University Press)
- Raeside D E 1976 Monte Carlo principles and applications *Phys. Med. Biol.* **21** 181–97
- Schweiger M, Arridge S R and Delpy D T 1993 Application of the finite-element method for the forward and inverse models of optical tomography *J. Math. Imaging Vision* **3** 263–83
- Van der Zee P 1993 Measurement and modelling of the optical properties of human tissue in the near infrared *PhD Thesis* University of London.