
ELSEVIER

Parallel Computing 24 (1998) 1523–1536

PARALLEL COMPUTING

Implementation of an environment for Monte Carlo simulation of fully 3-D positron tomography on a high-performance parallel platform

Habib Zaidi *, Claire Labbé, Christian Morel

Division of Nuclear Medicine, Geneva University Hospital CH-1211 Geneva 4, Switzerland

Received 15 January 1998; received in revised form 15 April 1998

Abstract

This paper describes the implementation of the *Eidolon* Monte Carlo program designed to simulate fully three-dimensional (3-D) cylindrical positron tomographs on a MIMD parallel architecture. The original code was written in Objective-C and developed under the NeXTSTEP development environment. Different steps involved in porting the software on a parallel architecture based on PowerPC 604 processors running under AIX 4.1 are presented. Basic aspects and strategies of running Monte Carlo calculations on parallel computers are described. A linear decrease of the computing time was achieved with the number of computing nodes. The improved time performances resulting from parallelisation of the Monte Carlo calculations makes it an attractive tool for modelling photon transport in 3-D positron tomography. The parallelisation paradigm used in this work is independent from the chosen parallel architecture.

Keywords: Monte Carlo simulation; Positron emission tomography; Random numbers; Image reconstruction; Parallel computer

1. Introduction

Positron emission tomography (PET) is a well-established imaging modality which provides physicians with information about the body's chemistry not available through any other procedure. Three-dimensional (3-D) PET provides qualitative and

* Corresponding author. E-mail: hzaidi@dmnu-pet5.hcuge.ch

0167-8191/98/$ – see front matter
PII: S0167-8191(98)00069-6

quantitative information about the volume distribution of biologically significant radiotracers after injection into the human body. Unlike X-ray computed tomography (CT) or magnetic resonance imaging (MRI), which look at anatomy or body morphology, PET studies metabolic activity or body function. PET has been used primarily in cardiology, neurology, and oncology. Mathematical modelling is widely used for the assessment of various parameters in imaging systems since no analytical solution is possible when solving the transport equation describing the interaction of photons with non-uniformly attenuating body structures and complex detector geometries. The Monte Carlo method is thoroughly used for solving problems involving statistical processes, particularly in nuclear medical imaging since it can reproduce the stochastic nature of radiation emission and detection [2]. Apart from being used to model detectors and to estimate tomograph performances, the Monte Carlo method was also employed to assess 3-D image reconstruction algorithms and their implementations [11,27,28]. The usefulness of Monte Carlo techniques in the development and evaluation of attenuation and scatter correction techniques for both SPECT [26] and PET [10,25] is also well established.

Although variance reduction techniques have been developed to reduce computation time, the main drawback of the Monte Carlo method is that it is extremely time-consuming. To obtain the statistics required for image reconstruction studies needs to track hundreds of millions of particles. Consequently, a large amount of CPU time (weeks or even months) may be required to obtain useful simulated data sets. As parallel computers are becoming increasingly accessible to computational scientists, problems that may otherwise be computationally prohibitive can be performed much faster than with a scalar machine. Historically, however, most programs and subroutine libraries have been developed to run on serial, single-processor computers. A modification or adaptation of source code is therefore a prerequisite to run it on a parallel computer. Nevertheless, among all simulation techniques of physical processes, the Monte Carlo method is probably the most suitable one for parallel computing since photon histories are completely independent from each others. Although parallel processing seems to be the ideal solution for Monte Carlo simulation, very few investigations have been reported and a limited number of papers have been published on the subject [2]. The aim of this paper is to place the status of parallel Monte Carlo in perspective and describe the implementation of a simulator for 3-D positron tomography on a high performance parallel platform.

2. Parallelisation strategies for Monte Carlo codes

Sequential programs make the most effective use of the available processing power: they alone guarantee maximum use of the CPU. In parallel programs, communication management introduces an unavoidable overhead, resulting in less efficient use of the overall CPU power. Moreover, according to Amdahl's law [1], parallelisation efficiency is decreased by a factor representing the fraction of operations that must be executed in sequential order. When this fraction reaches one, we are confronted with a wholly unparallelisable code, and the speed-up is zero no

matter how many processors are used. The efficiency of parallel programs is furthermore reduced by a factor equal to the fraction of processor idle time, which is highly dependent on the software parallelisation techniques used by the programmer.

In photon transport simulation, scalar or serial Monte Carlo codes track the history of one particle at a time, and the total calculation time is the sum of the time consumed in each particle history. Many Monte Carlo applications have characteristics that make them easy to map onto computers having multiple processors. Some of these parallel implementations require little or no interprocessor communication, and are typically easy to code on a parallel computer. Others require frequent communication and synchronisation among processors and in general are more difficult to write and debug. A common way to parallelise Monte Carlo is to put identical "clones" on the various processors; only the random numbers are different. It is therefore important for the sequences on the different processors to be uncorrelated so each processor does not end up simulating the same data [8]. That is, given an initial segment of the sequence on one process, and the random number sequences on other processes, we should not be able to predict the next element of the sequence on the first process. For example, it should not happen that if random numbers of large magnitude are obtained on one process, large numbers are more likely to be obtained on another. Furthermore, in developing any parallel Monte Carlo code, it is important to be able to reproduce exactly Monte Carlo runs in order to trace program execution.

The basic principles of parallel and vector processing are illustrated in Fig. 1. In history-based parallel processing, each particle $(p_1, p_2, \ldots, p_m)$ is assigned to one process which tracks its complete factual history $(e_1, e_2, \ldots, e_n)$. In event-based vector processing, a process treats only part of each particle history $(e_1, e_2, \ldots, e_n)$ and particles $(p_1, p_2, \ldots, p_m)$ "flow" from process to process. The first approach seems to be best suited to the physics of the problem. By spreading out the work among many processors, a speed-up that approaches the number of processors being used could be attained. This can be achieved through the use of parallel processing environments including arrays of transputers [3,14], vector parallel supercomputers [19,22], massively parallel computers [20], or a cluster of workstations in a local area network using a parallel computing simulator such as Parallel Virtual Machine (PVM) [13]. While vector processors were designed for single instructions operating on multiple data (SIMD), parallel computers are for multiple instructions operating independently on multiple data (MIMD). In this later case, memories are distributed on each processor and the communications between processors are carried out by passing messages. Processors in MIMD machines are subject to asynchronous and unbalanced external effects and are thus, for all practical purposes, impossible to keep aligned in time in a predictable way. If the assumption is made that random number generation from a single generator will occur across processors in a certain predictable order, then that assumption will quite likely be wrong. A number of techniques have been developed that guarantee reproducibility in multiprocessor settings and with various types of Monte Carlo problems [5]. Although PVM [9] or MPI [23] can be used to provide a machine independent message passing interface,

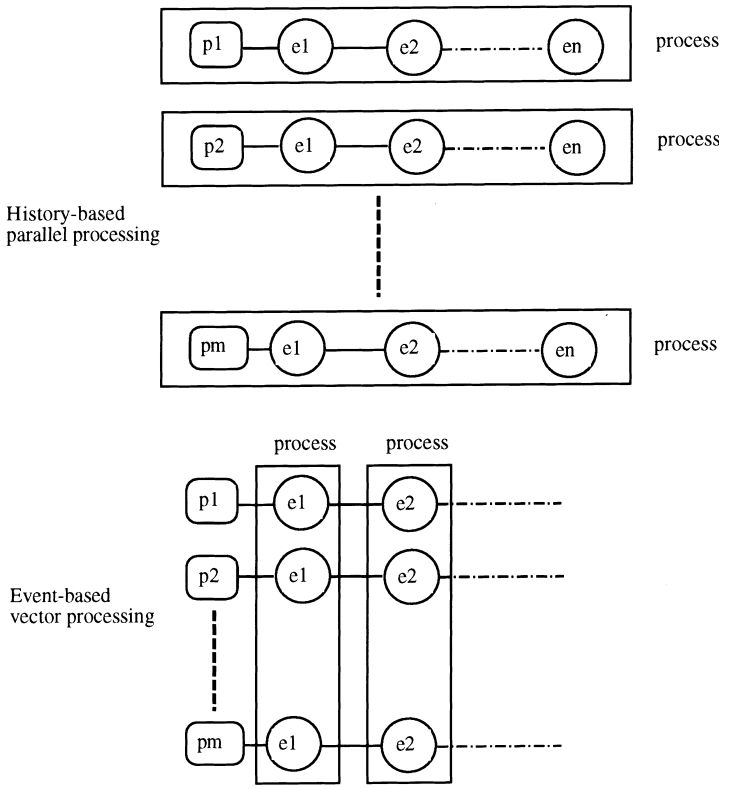

Fig. 1. Comparison of history-based parallel processing and event-based vector processing. In history-based parallel processing, each particle $(p_1, p_2, \ldots, p_m)$ is assigned to one process which tracks its complete factual history $(e_1, e_2, \ldots, e_n)$. In event-based vector processing, a process treats only part of each particle history $(e_1, e_2, \ldots, e_n)$ and particles $(p_1, p_2, \ldots, p_m)$ "flow" from process to process.

uniform assignment of particles to processes is not suited for workstation clusters because all the processors in the cluster may not have the same computation power in a heterogeneous environment, and the processing speed is bound by the slowest process. Moreover, the programmer needs to handle the different formats (big-endian and little-endian) computers use to keep data in memory.

During the last decade, investigations were carried out to run the EGS4 photon-electron Monte Carlo transport code on vector machines [7,17] and a speed up of about 8 was reported with the vectorized code [19]. The same code was also implemented on a multiple-transputer system [14] and a parallel computer [20]. Different approaches for parallel execution of this code including large-grain data flow techniques were reported [4]. Last but not least, Smith [22] described a vectorized code for simulation of SPECT imaging that uses an event-based algorithm in which photon history computations are performed within DO loops. The indices of the DO loops range over the number of photon histories, and take advantage of the vector processing unit of the Stellar GS1000 computer for pipelined computations.

3. Implementation of the Eidolon Monte Carlo code on the parallel system

3.1. System architecture

The Parsytec CC system (Parsytec GmbH, D-52068 Aachen, Germany) is an autonomous unit at the card rack level. The CC card rack subsystem provides the system with its infrastructure including power supply and cooling. The system is configured as a standard 19'' rack mountable unit which accepts the various 6U plug-in modules. The Parsytec CC system is a distributed memory, message passing parallel computer and is globally classified into the MIMD category of parallel computers. Its architecture is based on the mainstream Motorola MPC 604 processor running at 133 MHz with 512KB L2-cache. The modules are connected together at 1 Gbits/s with high speed (HS) link technology according to the IEEE 1355 standard, allowing data transfer at up to 75 Mbytes/s. The communication controller is integrated in the processor nodes through the PCI bus. PCI is also the standard which is used in the I/O modules and customer-specific I/O functions. All the nodes are directly connected to the same router which implements an active hardware 8 by 8 crossbar switch for up to 8 connections using the HS link. A schematic representation of the CC node is given in Fig. 2. It combines a memory controller (MPC 104) with a MPC 604 PowerPC processor and its local memory through an on-board PCI bus. The system board uses the MPC 105 chip to provide memory control, DRAM refresh and memory decoding for banks of DRAM and/or Flash. The CPU bus speed is limited to 66 MHz while the PCI bus speed is 33 MHz at maximum.

The software is based on IBM's AIX 4.1 UNIX operating system together with Parsytec's parallel programming environment Embedded PARIX (EPX). Thus, it combines a standard UNIX environment (compilers, tools, libraries) with an advanced software programming development environment. The system was integrated

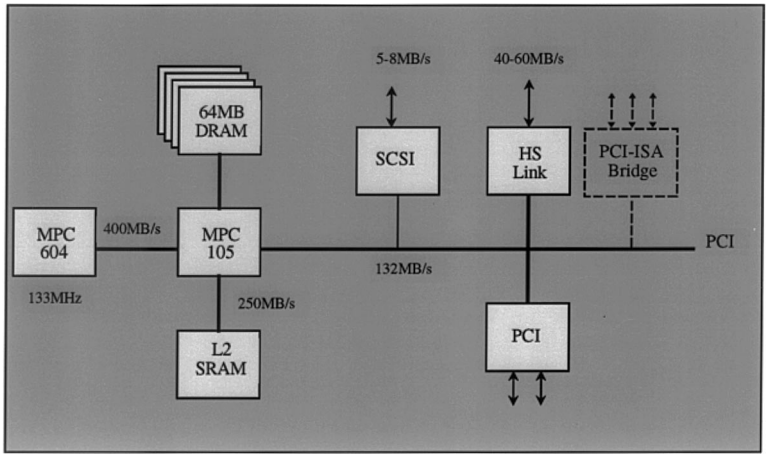

Fig. 2. Schematic layout of the parsytec CC node, showing the main PowerPC 604 CPU, its local memory through an on-board PCI bus, and the MPC 105 chip for memory control.

to the local area network using standard Ethernet. Currently a CC node has a peak performance of 266 MFlops. The peak performance of the 8-node CC system installed at Geneva University Hospital is therefore 2.1 GFlops. A schematic view is shown in Fig. 3. To develop parallel applications using EPX, data streams and function tasks are allocated to a network of nodes. The data handling between processors requires just a few system calls. Standard routines for synchronous communication such as send and receive are available as well as asynchronous system calls. The full set of EPX calls establishes the EPX application programming interface (API). The destination for any message transfer is defined through a virtual channel that ends at any user defined process. Virtual channels are user defined and managed by EPX. The actual message delivery system software utilises the router.

3.2. Porting the simulator on the parallel system

The *Eidolon* Monte Carlo simulation software was developed for cylindrical 3-D positron tomographs [27]. The main application sought is to generate data sets in a controllable manner to compare different volume reconstruction algorithms [28]. The original code was written in Objective-C and run under the NeXTSTEP (version 3.3) object-oriented development environment (NeXT Computer, Inc., Redwood City,

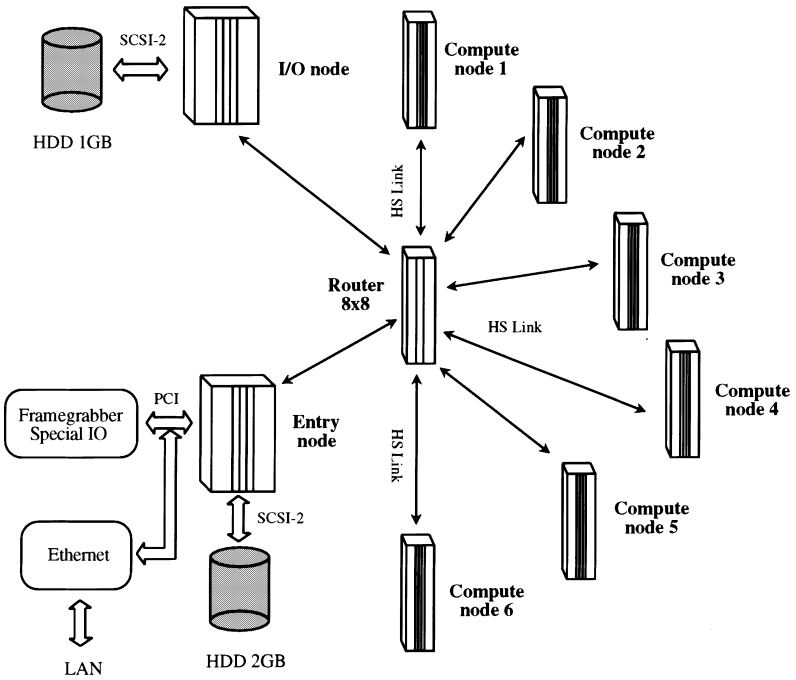

Fig. 3. Schematic view of the parsytec CC8 system installed at Geneva University Hospital and integrated to the local area network (LAN). It consists mainly of a router, a networked entry node, one I/O node and six computing nodes housed in the same 19" rack.

CA 94063) taking advantage of its powerful graphical user interface libraries and toolkits. Objective-C is a superset of ANSI C and provides classes and message passing similar to Smalltalk. Although, the NeXTSTEP environment provided a great tool allowing to reduce the amount of time necessary for writing applications, the portability issue still remained hard to solve since these applications were tied to machines running NeXTSTEP. Fortunately, the GNU C compiler (gcc) available from the Free Software Foundation (Free Software Foundation, Boston, MA 02111-1307 USA) comes with an Objective-C compiler since version 2.7.1. The current distributions of gcc (version 2.8.0) includes an Objective-C compiler and a runtime library. Hopefully, this makes possible to port *Eidolon* on most of the current platforms and operating systems. Eventually, a problem was encountered during installation of gcc on the Parsytec CC system because Objective-C with gcc was broken for the PowerPC-AIX 4.1 architecture. Though we were able to get gcc and the Objective-C runtime library compiled properly, there was a problem apparent during run-time which was due to the fact that the Objective-C constructors were not gathered and executed upon application start-up as they were supposed to. This functionality is provided through the collect2.c program within gcc. Thanks to Scott Christley from the GNUstep project [1], we were able to hand patch gcc in order to make it work. A general-purpose class library of non-graphical Objective-C objects designed in the Smalltalk tradition was also installed and method calls changed accordingly. The libobjects library is to GNU's Objective-C what libg++ is to GNU's C++. The parallelised code consists of the following processes:

1. a master 'control' process to generate tasks;
2. a number of slave 'simulation' processes to execute tasks and to generate results;
3. an 'analysis' process to collect data and to analyse results.

A simplified flow chart of the parallel simulator is given in Fig. 4. Information about the number and identity of nodes on which to perform the simulation, interaction within scatter medium and/or detector blocks (scatter-free, or considering scattering and attenuation) as well as details of scanner geometry, source and scatterer parameters are specified within series of ASCII files. The control process reads those files from the disk, performs the simulation, then writes the results. To avoid possible file corruption when two processes are editing the same file, a small delay between execution of the program on the selected processors has been arranged. Each processor writes a file named 'phantom' followed by its identification number (1–12). The random nature of the Monte Carlo method results in the fact that each processor could be writing data to its files at any time, making it impossible to safely add the resulting data files from many processors for analysis. A separate program is called to carry on the analysis phase and is launched only when all processors

[1] The GNUstep project (http://www.gnustep.org/) is an effort of many different parties to deliver a free implementation of the OpenStep specification as published by NeXT & Sun companies in 1994. The motivation behind GNUstep is to make porting applications easier, to leverage the benefits of the Objective-C language, and to push the idea of OpenStep as a common object oriented API; especially on systems which are not covered by commercial implementations of this standard.

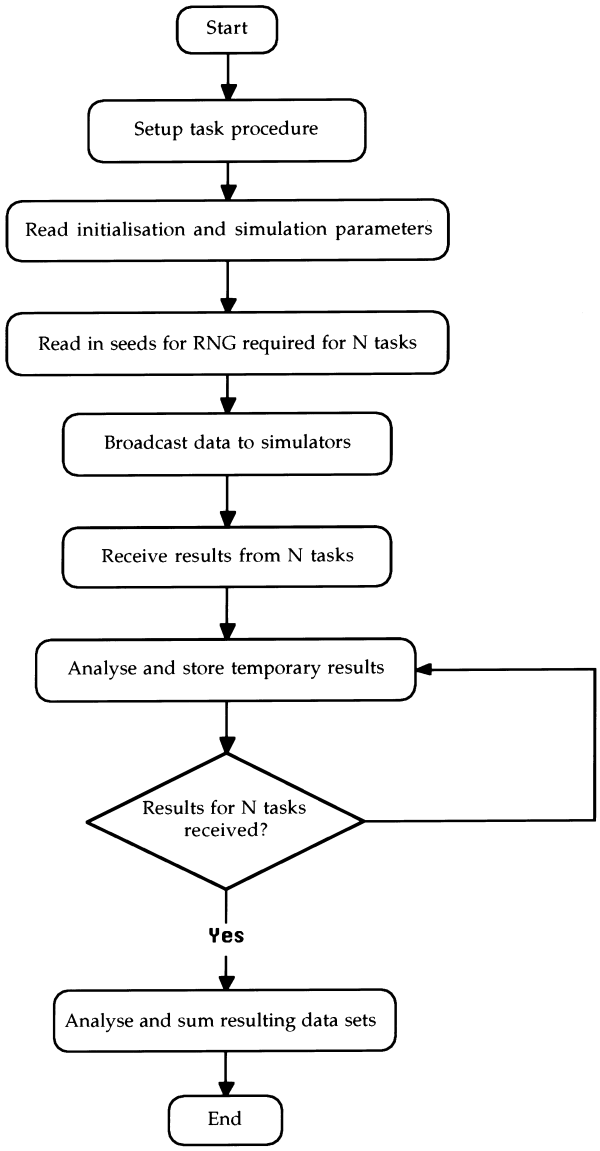

Fig. 4. Flow chart of the parallel simulation processes.

finished the calculations. The EPX partition management software (NRM) was used to define sets of partitions consisting of one processor each. Each individual processor run its own simulation with different initial configurations. One designated processor (entry node) serves as a controller which supplies other processors with some parameters that govern the simulation processes. Besides running its own sequence, the controller assesses the status of the whole simulation.

3.3. Distribution of random number sequences

For many computational science applications, such as Monte Carlo simulation, it is crucial that the generators have good randomness properties. This is particularly true for large-scale simulations done on high-performance parallel computers. Although powerful random number generators (RNG) have been suggested including shift register, inversive congruentional and combinatorial, the most commonly used generator for random numbers is the linear congruential RNG (LCRNG) [2,14] which has the following form

$$u_{n+1} = a(u_n + c) \bmod(m), \tag{1}$$

where m is the modulus, a the multiplier, and c the additive constant or addend. The size of the modulus constrains the period, and is usually chosen to be either prime or a power of 2. Recently, Monte Carlo researchers have become aware of the advantages of lagged Fibonacci series. With extremely long periods, they are generally faster than LCRNG and have excellent statistical properties. The lagged Fibonacci series RNG (LFRNG) have the following general form [15]

$$u_n = u_{n-1} \otimes u_{n-k} \bmod(m), \qquad 1 > k, \tag{2}$$

where $\otimes$ may be one of the following binary arithmetic operators +, −,*; l and k are the lags, and m is a power of 2 ($m = 2^n$). In recent years the additive lagged Fibonacci RNG (ALFRNG) has become a popular generator for serial as well as scalable parallel machines because it is easy to implement, it is cheap to compute and it does well on standard statistical tests, especially when the lag k is sufficiently high (such as $k = 1279$). The maximal period of the ALFRNG is $(2^k - 1)2^{n-1}$ [6] and has $2^{(k-1)(n-1)}$ different full-period cycles [18]. Another advantage of the ALFRNG is that one can implement these generators directly in floating-point to avoid the conversion from integer to floating-point that accompanies the use of other generators. However, some care should be taken in the implementation to avoid floating point round-off errors.

Instead the ALFRNG can be parameterised through its initial values because of the tremendous number of different cycles. Different streams are produced by assigning each stream a different cycle. An elegant seeding algorithm that accomplishes this is described in Ref. [18]. An interesting cousin of the ALFRNG is the multiplicative lagged-Fibonacci RNG (MLFRNG). While this generator has a maximal-period of $(2^k - 1)2^{n-3}$, which is a quarter the length of the corresponding ALFRNG, it has empirical properties considered to be superior to ALFRNGs [15]. An interesting aspect for parallel computing is that a parameterisation analogous to that of the ALFRNG exists for the MLFRNG. This later algorithm was used in this work for generating uniformly distributed random numbers. The sequence of 24 bit random numbers has a period of about 2^{144}, and has passed stringent statistical tests for randomness and independence [16].

There are many approaches to vector and parallel random number generation in the literature [21,24]. We can distinguish three general approaches to the generation of random numbers on parallel computers: centralised, replicated, and distributed.

In the centralised approach, a sequential generator is encapsulated in a task from which other tasks request random numbers. This avoids the problem of generating multiple independent random sequences, but is unlikely to provide good performance. Furthermore, it makes reproducibility hard to achieve: the response to a request depends on when it arrives at the generator, and hence the result computed by a program can vary from one run to the next. In the replicated approach, multiple instances of the same generator are created (for example, one per task). Each generator uses either the same seed or a unique seed, derived, for example, from a task identifier. Clearly, sequences generated in this fashion are not guaranteed to be independent and, indeed, can suffer from serious correlation problems. However, the approach has the advantages of efficiency and ease of implementation and should be used when appropriate. In the distributed approach, responsibility for generating a single sequence is partitioned among many generators, which can then be parcelled out to different tasks. The generators are all derived from a single generator; hence, the analysis of the statistical properties of the distributed generator is simplified.

A Monte Carlo particle history is a Markov chain because the next interaction or movement of a particle is always determined by the current state of the particle. The histories of two particles became identical only when the same random number sequence is used to sample the next state. The MLFRNG is initialised through the function `start_random_number (int seed_a, int seed_b)`. Supplying two seeds to `start_random_number` is therefore required once at program start-up before requesting any random numbers. The correspondence between pairs of seeds and generated sequences of pseudo-random numbers is many-to-one. If we choose the seeds carefully, then we can ensure that each random sequence starts out in a different cycle, and so two sequences will not overlap. Thus the seeds are parameterised (that is, sequence i gets a seed from cycle i, the sequence number being the parameter that determines its cycle). The question now becomes, how to initialise separate cycles to ensure that the seed tables on each processor are random and uncorrelated? This problem is addressed by Mascagni et al. [18] where they describe a canonical form for initialising Fibonacci generators. This canonical form is determined by l and k, but is independent of m. In general, the canonical form for initialising Fibonacci generators requires word $(l-1)$ to be set to all zero bits and the least significant bits of all words in the register to be set to zero, with the exception of one or two characteristic bits that depend on l and k. Using this methodology, one can actually distribute random number seeds to different processors for doing the same calculation and ensure that particles for a specific process will not start at the same position of the sequence.

4. Performance and timing results

Besides the fidelity of the simulated PET data and the ease with which software models can be prepared, the remaining important consideration in performing

Table 1
Comparison of the computing times for different simulation studies

	HP 9000	Parsytec	Parsytec	Parsytec	Parsytec
	712/60	CC1	CC4	CC8	CC12
Scatter-free imaging	72.33	25.88	6.38 (4)	3.7 (7)	2.2 (11.2)
Detector-scatter imaging	1425.65	436.55	110.26 (4)	54.6 (8.2)	36.7 (11.9)
Object-scatter imaging	2256.2	493.35	122.1 (4)	63.97 (7.7)	42.5 (11.6)
Full-scatter imaging	2540.45	657	170.5 (3.9)	83.66 (7.8)	55.7 (11.8)

Computing times required to track 10 million annihilation pairs on both the HP 9000 712/60 workstation and the Parsytec CC system having 1, 4, 8 and 12 computing nodes are given in minutes. The speed-up achieved is also reported.

simulation of PET imaging systems is the computational time required to generate adequate data sets. Timing of a single MPC 604 processor was carried out against the HP9000/712 station. The same simulation benchmark was used in the timing in which a line source was simulated in the centre of a water-filled cylindrical phantom for the ECAT-953B PET scanner (CTI PET Systems, Knoxville, TN 37933) operated in 3-D mode (16 rings of 384 detectors each with a ring radius of 38 cm) in scatter-free imaging, and when considering attenuation and scattering within the phantom and detector blocks. The resulting computing time in minutes and the ratios between 4,8,12 and one single node to track 10 million annihilation pair photon histories are given in Table 1. Time required by slave processors to write data sets on disk is not included in the time quoted in Table 1. The time required to perform the I/O operations and summation of data sets is negligible compared to that required for large simulation studies. A linear scaling of the computing time with the number of processors has been achieved.

Since high statistics are necessary to model imaging simulations, the computing time needed to track photons becomes of paramount importance. To illustrate the effect of statistics on the quality of reconstructed images, projections of the Jaszczak phantom (Data Spectrum Corporation, Hillsborough, NC 27278) with spheres insert having total counts ranging from 1 to 50 million were generated. The simulation included all of the degrading factors present in the physical imaging process. Before reconstruction, attenuation correction was applied to the data sets; attenuation correction files were created by forward projecting the 3-D density map estimated with a constant linear attenuation coefficient of 0.096 cm^{-1}. Scatter correction was not performed. Fig. 5. shows transaxial slices of the simulated phantom containing six cold spheres with diameters ranging from 9.5 to 31.8 mm reconstructed using the widely-used reprojection algorithm of Kinahan and Rogers [12]. According to visual inspection, the quality of the reconstructions is superior when the projections are obtained with good statistics. As a result of the enhanced quality, the small cold lesions are clearly visible when compared to those of the low count studies. This type of studies is useful to assess contrast recovery and to quantitatively predict the effect of different image reconstruction techniques on human accuracy for lesions detection.

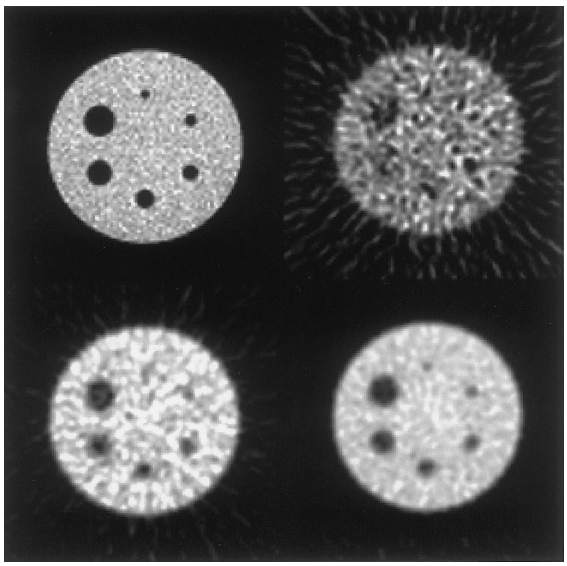

Fig. 5. Effect of limited statistics on the quality of reconstructed images and lesion detection. The phantom consists of a water-filled cylindrical phantom containing uniform activity in which six cold spheres with diameters ranging from 9.5 to 31.8 mm are included. The reference image (top left) and reconstructed slices generated with 1 Mcounts (top right), 10 Mcounts (bottom left), and 50 Mcounts (bottom right) are shown.

5. Discussion and conclusion

The object-oriented *Eidolon* Monte Carlo code developed to simulate fully 3-D cylindrical positron tomographs was successfully implemented on the Parsytec CC parallel system using PowerPC 604 nodes under AIX 4.1. We anticipated that the possibilities offered by the software need to increase steadily according to the research application of interest. Therefore software changes would be unavoidable within the following years. Besides extendibility, maintainability was thus very important as existing code must be reused and new features must be introduced in a straightforward manner. The object-oriented programming paradigm meets all these requirements and has proven to improve productivity, quality, and innovation in software development. It provides modelling primitives, a framework for high-level reusability, and integrating mechanisms for organising knowledge about application domains. In order to get higher speed, Monte Carlo applications make extensive use of parallel computers, since these calculations are particularly well suited to such architectures and often require very long runs.

A linear increase in computing speed was achieved with the number of computing nodes used. There is no theoretical limit on the number of processors to be used. Upgrade of the system is reasonable to obtain better performance. The techniques developed in this work are also suitable for the implementation of the Monte Carlo code on other parallel computer systems. A potential application of *Eidolon* will be

to compute photon detection kernels, which will be used to build system matrices for simulating PET projection data for use during the forward projection step in iterative image reconstruction algorithms.

Acknowledgements

This work was supported by the Swiss Federal Office for Education and Science under grant 96.193 within the European Esprit LTR project PARAPET (EP23493). The authors are indebted to Scott Christley from the GNUstep project for his help during the installation of the GNU compiler. They also gratefully thank their partners within the PARAPET project.

References

[1] G.M. Amdahl, Validity of the single processor approach to achieving large-scale computing capabilities, AFIPS Conference Proceedings Washington, DC, Vol. 30 (1967) 483–485.
[2] P. Andreo, Monte Carlo techniques in medical radiation physics, Phys. Med. Biol. 36 (1991) 861–920.
[3] C.R. Askew, D.B. Carpenter, J.T. Chalker et al., Monte Carlo simulation on transputer arrays, J. Parallel Computing 6 (1988) 247–258.
[4] R.G. Babb, L. Storc, Developing a parallel Monte Carlo transport algorithm using large-grain data flow, Parallel Comput. 7 (1988) 187–198.
[5] V.C. Bhavsar, J.R. Isaac, Design and analysis of parallel Monte Carlo algorithms, SIAM J. Statistical and Scient. Comput. 81 (1987) 73–95.
[6] R.P. Brent, On the periods of generalized Fibonacci recurrences, Math. Comput. 63 (1994) 389–401.
[7] F.B. Brown, W.R. Martin, Monte Carlo methods for radiation transport analysis on vector computers, Progr. Nucl. Energy 14 (1984) 269–299.
[8] A. De Matteis, S. Pagnutti, Controlling correlations in parallel Monte Carlo, Parallel Comput. 21 (1995) 73–84.
[9] A. Geist, A. Beguelin, J. Dongarra et al., PVM–A users' guide and tutorial for networked parallel computing, MIT Press, Boston, 1994.
[10] D.R. Haynor, R.L. Harrison, T.K. Lewellen, Energy-based scatter correction for 3D PET: A Monte Carlo study of best possible results, in: P. Kinahan, D. Townsend (Eds.), Conf. Rec. of the International Meeting in Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Nemacolin Woodlands, UPMC, Pittsburgh, 1997, pp. 52–54.
[11] A.K. Herrmann Scheurer, M.L. Egger, C. Joseph, C. Morel, A Monte Carlo phantom simulator for positron emission tomoraphy, in: H.J. Hermann, D.E. Wolf, E. Pöppel (Eds.), Proceedings of the Workshop on Supercomputing in Brain Research: From Tomography to Neural Networks, Jülich, 1994, World Scientific, Singapore, 1995, pp. 205–209.
[12] P.E. Kinahan, J.G. Rogers, Analytic 3D image reconstruction using all detected events, IEEE Trans. Nucl. Sci. 36 (1989) 964–968.
[13] D.R. Kirkby, D.T. Delpy, Parallel operation of Monte Carlo simulations on a diverse network of computers, Phys. Med. Biol. 42 (1997) 1203–1208.
[14] C.-M. Ma, Implementation of a Monte Carlo code on a parallel computer system, Parallel Comput. 20 (1994) 991–1005.
[15] G. Marsaglia, A. Zaman, Some portable very-long-period random number generators, Computers in Physics 8 (1994) 117–121.
[16] G. Marsaglia, A. Zaman, Monkey tests for random number generators, Comp. Math. Applic. 23 (1993) 1–10.

[17] W.R. Martin, F.B. Brown, Status of vectorized Monte Carlo code for particle transport analysis, Int. J. Supercomputer Appl. 1 (1987) 11–32.
[18] M. Mascagni, S.A. Cuccaro, D.V. Pryor, M.L. Robinson, A fast, high-quality, and reproducible lagged-Fibonacci pseudorandom number generator, J. Comput. Phys. 15 (1995) 211–219.
[19] K. Miura, EGS4V: Vectorization of the Monte Carlo cascade shower simulation code EGS4, Comput. Phys. Commun. 45 (1987) 127–136.
[20] K. Miura, R.G. Babb, Tradeoff in granularity and parallelization for a Monte Carlo shower code (EGS4), Parallel Comput. 8 (1987) 91–100.
[21] R. Sarno, V.C. Bhavsar, E.M.A. Hussein, Generation of discrete random variables on vector computers for Monte Carlo simulations, Int. J. High Speed Comput. 2 (1990) 335–350.
[22] M.F. Smith, C.E. Floyd, R.J. Jaszczak, A vectorized Monte Carlo code for modeling photon transport in SPECT, Med. Phys. 20 (1993) 1121–1127.
[23] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, MIT Press, Boston, 1996.
[24] A. Srinavasan, D.M. Ceperley, M. Mascagni, Random number generators for parallel applications, in: D. Ferguson, J.I. Siepmann, D.G. Truhlar (Eds.), Monte Carlo Methods in Chemical Physics, Advances in Chemical Physics series, Wiley, New York, 1997.
[25] C.C. Watson, D. Newport, M.E. Casey, A. deKemp, R.S. Beanlands, M. Schmand, Evaluation of simulation-based scatter correction for 3-D PET cardiac imaging, IEEE Trans. Nucl. Sci. 44 (1997) 90–97.
[26] H. Zaidi, Quantitative SPECT: Recent developments in detector response, attenuation and scatter correction techniques, Physica Medica 12 (1996) 101–117.
[27] H. Zaidi, A. Herrmann Scheurer, C. Morel, An object-oriented Monte Carlo simulator for 3D positron tomographs, Comput. Methods Programs Biomed (1998), in press.
[28] H. Zaidi, A. Herrmann Scheurer, C. Morel, Development of an object-oriented Monte Carlo simulator for 3D positron tomography, in: P. Kinahan, D. Townsend (Eds.), Conf. Rec. of the International Meeting in Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Nemacolin Woodlands, UPMC, Pittsburgh, 1997, pp. 176–179.