Journal of Digital Imaging

Execution of the SimSET Monte Carlo PET/SPECT Simulator in the Condor Distributed Computing Environment

Karl G. Baum , and María Helguera

SimSET is a package for simulation of emission tomography data sets. Condor is a popular distributed computing environment. Simple C/C++ applications and shell scripts are presented which allow the execution of SimSET on the Condor environment. This is accomplished without any modification to SimSET by executing multiple instances and using its combinebin utility. This enables research facilities without dedicated parallel computing systems to utilize the idle cycles of desktop workstations to greatly reduce the run times of their SimSET simulations. The necessary steps to implement this approach in other environments are presented along with sample results.

KEY WORDS: SimSET, Monte Carlo, Condor, emission tomography, simulation, computer simulation, positron emission tomography (PET), single photon emission computed tomography (SPECT), high-performance computing, distributed computing, grid computing, open source, computers in medicine

BACKGROUND

Software simulators have been used extensively in the field of nuclear medicine since the 1990s. These simulators, designed to emulate the object being imaged, nuclear physics, and the detector system may be based on analytic or Monte Carlo models. They can be applied to study a number of different topics including system design, acquisition protocols, and reconstruction techniques. Anyone interested in learning more about positron emission tomography (PET) and single photon emission computed tomography (SPECT) simulators is referred to[1,2] for a review of different simulation software packages and their applications.

The work described in this paper is based on the SimSET simulator.[3] It was selected for its wide acceptance, and relatively extensive feature set. SimSET, which "uses Monte Carlo techniques to model the physical processes and instrumentation used in emission imaging", was initially released in 1993.[3] It was developed and is still being maintained and updated by the University of Washington Imaging Research Laboratory. SimSET, which can be used to model both SPECT and PET, models the important physical phenomena including photoelectric absorption, Compton's scattering, coherent scattering, photon noncolinearity, and positron range. It supports a variety of collimator and detector designs, and already includes the attenuation properties for many common materials. SimSET and its source code can be downloaded from.[3]

The code is written in a modular format, see Figure 1. The Photon History Generator is the central module that models the generation and transport of photons through attenuating media (object being imaged) to the face of the collimator or detector. The Collimator module allows the user to choose a collimator's configuration depending on the system to be simulated. This module

From the Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, 54 Lomb Memorial Dr., Rochester, NY, 14623, USA.

Correspondence to: Karl G. Baum, Chester F. Carlson Center for Imaging Science, Rochester Institute of Technology, 421 Countess Dr., West Henrietta, NY 14586, USA; Tel: +1-585-4757053; Fax: +1-585-4754568; e-mail: kgb5056@rit.edu

Online publication 10 August 2007

doi: 10.1007/s10278-007-9058-z

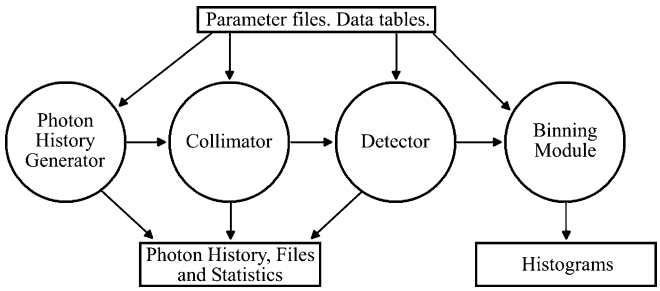

Fig 1. Block diagram of SimSET modules. Adapted from[3].

receives the photons from the Photon History Generator and tracks them as they move through the collimator to the detector. The Detector module allows the user to select the detector geometry and properties. The Detector module can receive photons from either the Photon History Generator module or the Collimator module. It tracks the movement and interactions of the photons until either they escape or all of their energy has been deposited. The Binning module is used to process photon history and detection records. It offers options to write the data in different histogram formats, such as sinograms. Photons can be selected for binning based on their deposited energy and scattering history. The Photon History Generator, the Collimator module, the Detector module, and the Binning module can all create photon history files, which can be used for further analysis of the simulation results. Together, these modules can be used to simulate a wide variety of systems. For more information on SimSET, see[3–9].

The computational complexity of simulators can often limit their utility. This becomes clear when trying to simulate realistic system configurations with high-resolution input data sets. It is not uncommon for simulations to run for a week or two on standard hardware. Much effort has been spent investigating techniques to reduce simulation run times. The most notable efforts in accelerating the simulation process have been applied to the SimSET simulator. Importance sampling techniques such as stratification, forced detection, non-absorption and weight windows [8,9] can lead to an acceleration by a factor of at least 2 and up to more than 10.[2]

Even with acceleration techniques, the time required to run a simulation may be too long for simulators to be used in a practical manner in research laboratories. Running simulators in parallel environments is a promising way to further decrease the run time. To the authors knowledge, there are currently no PET/SPECT simulators designed to take advantage of more than a single processor. At least one attempt has been made to modify SimSET for execution in a parallel environment.[10] The chosen approach, however, may limit its acceptance in other research facilities due to its complexity and required technical skills for proper deployment. The approach runs a modified version of SimSET on the NetSolve distributed environment, which increases maintenance both due to code modification and deployment throughout the grid.

In this paper, the authors introduce an alternative parallel implementation of SimSET based on the Condor high throughput computing distributed environment.[11] This implementation requires no modification of SimSET, decreasing maintenance and simplifying installation. Distribution throughout the cluster can be automated using Condor's file transfer mechanisms, and Condor's unique architecture means no dedicated computing resources are required.

Condor, developed at the University of Wisconsin-Madison, provides "a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management" for execution of serial (ex. batch) or parallel (MPI, PVM, or server-client) applications.[11] Because of its design, it can effectively run on a dedicated cluster or it can rely on the unused resources of idle workstations, similar to SETI@Home[12,13] or Folding@Home.[14] It supports heterogeneous environments consisting of multiple machine architectures and operating systems. It can run with or without shared file systems, and supports check pointing (resuming execution of an application on the same or another

computer due to the computer losing its idle status, crashing, or rebooting) of specially compiled applications. Condor has been widely accepted, recently reported running on over 88,000 hosts worldwide.[15]

Individual jobs or batches of jobs are submitted to Condor by providing a simple text file containing job resource requirements and Condor settings. Condor uses this information to allocate appropriate resources (assign jobs to computers that meet the hardware and software requirements), and to determine Condor-specific job settings dealing with job priority, file transfer, error handling, submitter notification, and emulation of the local execution environment on remote machines. Jobs continue to be monitored during execution allowing graceful error handling and job migration as available resources change. For more information on Condor, see[11,16–19].

METHODS

Several steps were required before successful execution of SimSET on Condor. This involved a wrapper for SimSET to match its interface to that required by Condor, a script to setup and launch a SimSET run on Condor, and an application to finalize the results. Only the script needs to be modified to run SimSET on another Condor cluster.

The first parameter of any C/C++ application is the invocation name of the program that is being run. SimSET is a single executable, multiple utility, package that utilizes the first parameter to determine which utility the operator intended to run. The utility names that are invoked when running SimSET (ex. phg, combinebin, etc.) are simply symbolic links to the SimSET application. SimSET examines the invocation name to determine which utility was meant to be run. For example, if the first argument is phg, then SimSET runs the photon history generator utility.

When executing applications using Condor, the invocation name is replaced by the name of a Condor executable. SimSET is then unable to determine which utility the operator intended to run. To address this issue, a simple C wrapper was written. This wrapper, named simset_wrapper.c, can be found in Appendix 1. simset_wrapper.c simply removes the first argument and reruns the command. For example, running the command "simset_wrapper phg phg.param" will cause the command "phg phg.param" to be executed. By running SimSET using this wrapper the reliance on the invocation name is avoided. When running on Condor, the unavoidable replacement of the first parameter will no longer interfere with the execution of SimSET.

It was determined that the simplest way to run the serial SimSET application in parallel would be to run it multiple times combining the results at the end using the combinebin utility. Since the events recorded in the histograms are independent of each other (one decay of a radioisotope does not affect another), this is an acceptable solution. A simulation of a large number of events can be accomplished by running multiple simulations of an appropriately smaller number of events concluded by combining their results using the combinebin utility. This approach, excluding the execution of combinebin, provides essentially a linear decrease in execution time for SimSET.

A sh script simset_batch.sh, found in Appendix 2, was designed to create the necessary Condor job files for executing multiple versions of SimSET. It takes as arguments the histogram name, the phg parameter file name, and the number of instances that you would like run. For example, running "sh simset_batch.sh tutor1 phg.param 100" will cause the command "phg phg.param" to be executed 100 times. Their results tutor1.weight, tutor1.weight2 and tutor1.count will be combined at the end to create a single tutor1.weight, tutor1.weight2 and tutor1.count. The output for each run of phg will be saved in its own subdirectory for reference at a later time.

For SimSET to correctly run using simset_batch.sh, relative path names to preexisting files in the SimSET parmeter files will need to be changed. During execution, the working directory will be "./data/nodeX". This means a reference such as "bin.param" will need to be changed to "../../bin.param".

Lines 5–23 of simset_batch.sh remove any data from previous runs. Lines 24–47 create the Condor job file that will be used to run SimSET the appropriate number of times. See[15] for information on modifying the Condor parameters in this section for appropriate execution in your environment. In the very least appropriate paths will need to be inserted. Specification of requirements may be necessary if SimSET is expected to produce

large histograms. File transfer will need to be appropriately set up if working in a nonshared file system environment, and macros might need to be used for specification of executables and arguments in a multi-architecture, multi-operating system environment.

Lines 48–68 create the Condor job responsible for combining the results from the different SimSET runs. This section may need to be appropriately modified to match your environment. simset_batch.sh concludes with lines 69–70 which are responsible for submitting the previously defined jobs for execution on Condor.

The C++ application simset_finalize.cpp, found in Appendix 3, is responsible for combining the results of the different SimSET runs. Other than the path to the SimSET combinebin utility, this application should never need changing. Simset_finalize.cpp knows how many SimSET instances have been launched by Condor, it loops through these looking for ones that have completed. As they complete simset_finalize.cpp takes the results and uses combinebin to sum them. Please note that there are bugs in the combinebin portion of SimSET version 2.6.2.6. Contact SimSET support or the authors of this paper for the updates.

RESULTS AND DISCUSSION

A Condor cluster consisting of 25 333 Mhz Sun Ultra 10 workstations and 1 1.503 GHz Sun Blade 1500 workstation was used to evaluate the results. A thorax phantom similar to that used in the SimSET tutorial[20] was simulated with 10^8 total events. A slice through the phantom is shown in Figure 2. The simulation was first run using the entire cluster. The 25 Ultra 10 workstations were used to do the photon tracking while simset_finalize was run on the Blade 1500 workstation. The entire simulation concluded after approximately 92 min. The reconstructed simulation result (reconstructed using 3DRP, a fully 3D back projection technique with reprojection[21,22]) can be seen in Figure 3.

For comparison, the same simulation was run using just the Blade 1500 workstation, taking approximately 523 min, and on an Ultra 10 workstation, taking approximately 1,975 min. The recon-

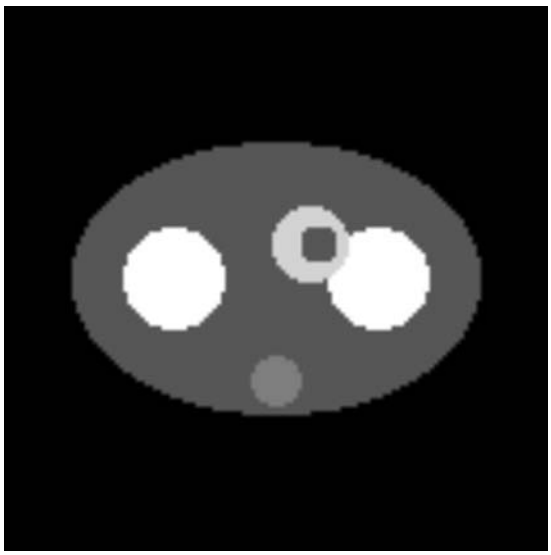

Fig 2. Thorax phantom for SimSET simulator.

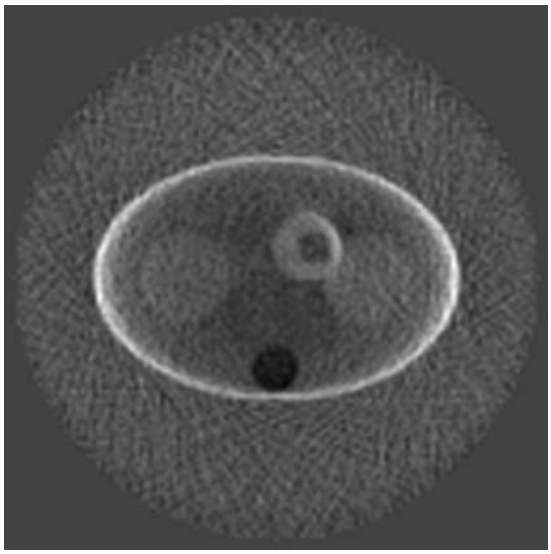

Fig 3. PET image of phantom generated using Condor.

structed results are shown in Figures 4 and 5, respectively. Other than variations due to noise, there appear to be no significant differences between the results from the different simulation runs.

Running SimSET in a distributed manner on the Condor system reduced runtime considerably. This savings will amplify in significance as the number of events being simulated increases, and as the

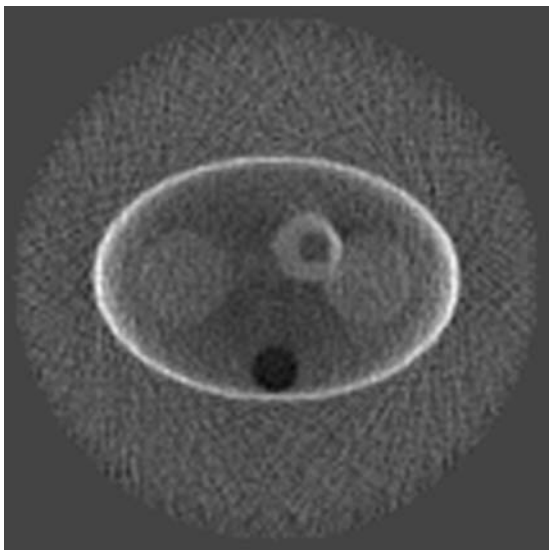

Fig 4. PET image of phantom generated using Sun Blade 1500 workstation.

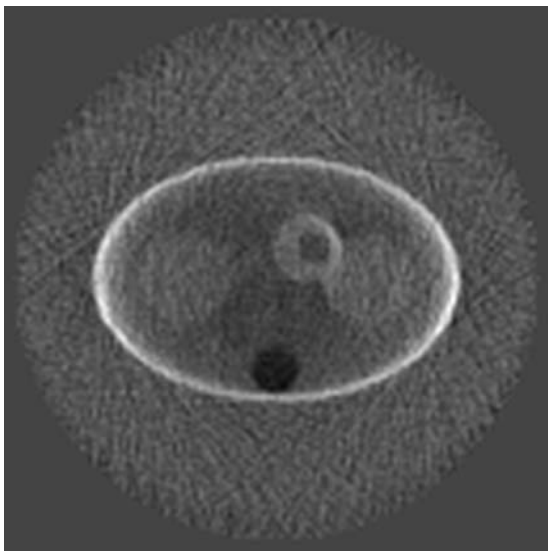

Fig 5. PET image of phantom generated using Sun Ultra 10 workstation.

simulations being run begin to more realistically model modern scanners.

Due to the way SimSET distributes decay events throughout the phantom, care needs to be exercised when deciding how many events to simulate and how many instances of SimSET to run in parallel. This becomes more important as the resolution of phantoms increases, the number of instances of SimSET running in parallel increases, and as the number of events being simulated decreases. See the empirical results section of[10] for a detailed discussion of this problem.

The time elapsed between completion of photon tracking and completion of simset_finalize will increase as the size of the histograms increase. If this time difference becomes too great then it may be necessary to consider other finalization schemes. One alternative would be to run the combinebin utility in a distributed format over the entire cluster using a multilayer pyramid scheme as shown in Figure 6. This arrangement is especially suited to homogeneous dedicated clusters. While when using this scheme the summation of the results is delayed until after all of the photon tracking is complete, decreases in total runtime may be achieved especially when simulating very large histograms.

SUMMARY

Significant decreases in the runtime of SimSET can be achieved by parallelization. This can be accomplished by running SimSET on a Condor cluster as described in this paper. This implementation required no changes to the SimSET source

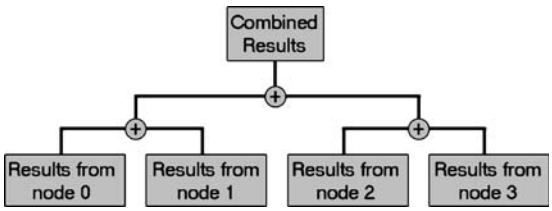

Fig 6. Triangle/pyramid scheme for using combinebin.

code and can be easily implemented in other environments. The only change that should be required is modification of the Condor job definitions specified in the script simset_batch.sh. This is a particularly attractive option for smaller research groups since Condor can utilize the unused CPU cycles in idle desktop workstations and does not require an expensive dedicated cluster.

ACKNOWLEDGMENT

The authors would like to thank Dr. Robert Harrison and the University of Washington Imaging Research Laboratory for supporting SimSET, and the University of Wisconsin-Madison computer science department for supporting Condor. They would also like to thank Research Computing at the Rochester Institute of Technology and its director Dr. Gurcharan S. Khanna for providing and supporting the computational resources used.

APPENDIX 1

simset_wrapper.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

main(int argc, char **argv)
{

    /* remove condor executable from arguments */
    int argIndex;
    for (argIndex = 0; argIndex < argc-1; argIndex++)
    {
        argv[argIndex] = argv[argIndex+1];
    }
    argc--;

    /* find length of arguments */
    int length = 0;
    for (argIndex = 0; argIndex < argc; argIndex++)
    {
        length = length + strlen(argv[argIndex]) + 1;
    }

    /* create new command */
    char command[length];
    command[0] = '\0';
    char space[2] = " \0";
    for (argIndex = 0; argIndex < argc; argIndex++)
    {
        strcat(command, argv[argIndex]);
        strcat(command, space);
    }

    /* execute command */
    printf(command);
    int error = system(command);
        if (error == 0)
            system("touch success");
        else
            system("touch failure");
    return error;
}
```

APPENDIX 2

simset_batch.sh

```
#! /bin/sh

imageName=$1
parameterFile=$2
N=$3

# -----------------------------------
# clear space for files
if [ -d "data" ]
    then
    while [ "$temp" != "y" ] && [ "$temp" != "n" ]
    do
        echo "Files in directory data will be overwritten. Continue [y,n]?"
        read temp
    done
    if [ "$temp" = "y" ]
        then
        rm -r data
    else
        echo "Aborting"
        exit
    fi
fi
mkdir data
# -----------------------------------

# -----------------------------------
# create job files
    cat <<EOM >> data/jobs.txt
Universe   = vanilla
Executable = /bin/simset_wrapper
Arguments  = /SimSET/2.6.2.6/bin/phg ${parameterFile}
Log        = job_log.txt
Output     = job_out.txt
Error      = job_error.txt
Rank       = Mips
notification = Error
copy_to_spool = False
EOM

X=0
while [ "$X" -lt "$N" ]
do
    mkdir "data/node$X"
    cat <<EOM >> data/jobs.txt
initialdir = ./data/node${X}
Queue
EOM
    X=`expr ${X} + 1`
done
# -----------------------------------

# -----------------------------------
# create finalize job
    cat <<EOM >> data/finalize.txt
Universe   = vanilla
Executable = /bin/simset_finalizer
Arguments  = ${imageName} ${N}
Log        = finalize_log.txt
Output     = finalize_out.txt
Error      = finalize_error.txt
Input      = no.txt
Rank       = Mips
notification = Always
copy_to_spool = False
initialdir = ./data/
Queue
EOM
# -----------------------------------

# -----------------------------------
# create input file for finalize
    echo "n" >> data/no.txt
# -----------------------------------

condor_submit ./data/finalize.txt
condor_submit ./data/jobs.txt
```

APPENDIX 3

simset_finalizer.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <cstring>
#include <new>

using namespace std;

main(int argc, char **argv)
{
    // argv[1] is histogram file name
    // argv[2] is number of histograms

    // get histogram file names
    char fileNameWeight[25];
    char fileNameWeight2[25];
    char fileNameCount[25];
    strcpy(fileNameWeight,argv[1]);
    strcpy(fileNameWeight2,argv[1]);
    strcpy(fileNameCount,argv[1]);
    strcat(fileNameWeight,".weight");
    strcat(fileNameWeight2,".weight2");
    strcat(fileNameCount,".count");

    // initialize parameters
    int N = atoi(argv[2]); // number of SimSET runs
    bool first = true; // is this the first run finished?
    int trueCount = 0; // number of completed runs
    bool *runs = new bool[N]; // track which runs have completed
    int x;
    for (x=0; x<N; x++)
        runs[x]=false;
    x=0;
    FILE *f;

    // sum results as runs complete
    // loop through SimSET runs looking for completed ones
    while (trueCount < N)
    {
        if (runs[x] == false) // if run has not be summed
        {
            // get file names indicating run completion
            char cx[5];
            sprintf(cx,"%d",x);
            char fileNameSuccess[25];
            strcpy(fileNameSuccess,"./node");
            strcat(fileNameSuccess,cx);
            strcat(fileNameSuccess,"/success");
            char fileNameFailure[25];
            strcpy(fileNameFailure,"./node");
            strcat(fileNameFailure,cx);
            strcat(fileNameFailure,"/failure");

            // if SimSET run completed successfully
            if ((f=fopen(fileNameSuccess,"r"))!=NULL)
            {
                // if this was the first run to complete
                // we just copy the results
                if (first)
                {
                    // create and run commands that copy results
                    char comm[50];
                    strcpy(comm,"cp ");
                    strcat(comm,"./node");
                    strcat(comm,cx);
                    strcat(comm,"/");
                    strcat(comm,fileNameWeight);
                    strcat(comm," ");
                    strcat(comm,fileNameWeight);
                    system(comm);

                    strcpy(comm,"cp ");
                    strcat(comm,"./node");
                    strcat(comm,cx);
                    strcat(comm,"/");
                    strcat(comm,fileNameWeight2);
                    strcat(comm," ");
                    strcat(comm,fileNameWeight2);
                    system(comm);
```

```
                        strcpy(comm,"cp ");
                        strcat(comm,"./node");
                        strcat(comm,cx);
                        strcat(comm,"/");
                        strcat(comm,fileNameCount);
                        strcat(comm," ");
                        strcat(comm,fileNameCount);
                        system(comm);

                        first = false;
                    }
                    // if it is not the first run we use combinebin
                    // to sum results
                    else
                    {
                        // complete and run commands that sum results
                        char comm[50];
                        strcpy(comm,"/SimSET/2.6.2.6/bin/combinebin ");
                        strcat(comm, fileNameWeight);
                        strcat(comm," ./node");
                        strcat(comm,cx);
                        strcat(comm,"/");
                        strcat(comm,fileNameWeight);
                        strcat(comm," < no.txt");
                        system(comm);

                        strcpy(comm,"/SimSET/2.6.2.6/bin/combinebin ");
                        strcat(comm, fileNameWeight2);
                        strcat(comm," ./node");
                        strcat(comm,cx);
                        strcat(comm,"/");
                        strcat(comm,fileNameWeight2);
                        strcat(comm," < no.txt");
                        system(comm);

                        strcpy(comm,"/SimSET/2.6.2.6/bin/combinebin ");
                        strcat(comm, fileNameCount);
                        strcat(comm," ./node");
                        strcat(comm,cx);
                        strcat(comm,"/");
                        strcat(comm,fileNameCount);
                        strcat(comm," < no.txt");
                        system(comm);
                    }

                    printf(fileNameSuccess);
                    printf("\n");
                    // update status
                    runs[x] = true;
                    trueCount++;
                    fclose(f);
                    f=NULL;
                }
                // if run failed we exclude its results
                else if ((f=fopen(fileNameFailure,"r"))!=NULL)
                {
                    printf(fileNameFailure);
                    printf("\n");
                    runs[x] = true;
                    trueCount++;
                    fclose(f);
                    f=NULL;
                }
            }
            x++;
            if (x>=N)
                x=0;
        }

        // cleanup
        delete [] runs;

        return 0;
    }
```

REFERENCES

1. Buvat I, Castiglioni I: Monte Carlo Simulations in SPECT and PET. Q J Nucl Med 46:48–61, 2002
2. Buvat I, Lazaro D: Monte carlo simulations in emission tomography and GATE: an overview. Nucl Instr Meth Phys Res 569(2):323–29, 2006
3. University of Washington. SimSET Home Page. Available at http://depts.washington.edu/simset/html/simset_main.html. Accessed 28 May 2007
4. University of Washington. SimSET User Guide. Available at http://depts.washington.edu/simset/html/user_guide/user_guide_index.html. Accessed 28 May 2007
5. Harrison RL, Haynor DR, Gillispie SB, Vannoy SD, Kaplan MS, Lewellen TK: A public-domain simulation system for emission tomography: photon tracking through heterogeneous attenuation using importance sampling. J Nuc Med 34(5):60P, 1993
6. Harrison RL, Vannoy SD, Haynor DR, Gillispie SB, Kaplan MS, Lewellen TK: Preliminary experience with the photon history generator module of a public-domain simulation system for emission tomography. Conf Rec IEEE Nuc Sci Sym 2:1154–8, 1993
7. Kaplan MS, Harrison RL, Vannoy SD: Coherent scatter implementation for SimSET. IEEE Trans Nuc Sci 46(6):3064–68, 1998
8. Haynor DR, Harrison RL, Lewellen TK: The Use of Importance Sampling Techniques to Improve the Efficiency of Photon Tracking in Emission Tomography Simulations. Med Phys 18(5):990–1001, 1991.
9. Haynor DR, Harrison RL, Lewellen TK, Bice AN, Anson CP, Gillispie SB, Miyaoka RS, Pollard KR, Zhu JB: Improving the Efficiency of Emission Tomography Simulations Using Variance Reduction Techniques. IEEE Trans Nuc Sci 37 (2):749–53, 1990.
10. Thomason MG, Longton RF, Gregor J, Smith GT, Hutson RK: Simulation of Emission Tomography Using Grid Middleware for Distributed Computing. Comput Methods Programs Biomed 75(3):251–8, 2004.
11. University of Wisconsin-Madison. Condor Project Homepage. Available at http://www.cs.wisc.edu/condor/. Accessed 28 May 2007.
12. Classic SETI@Home. Available at http://seticlassic.ssl.berkeley.edu/. Accessed 28 May 2007
13. SETI@Home. Available at http://setiathome.berkeley.edu/index.php. Accessed 28 May 2007
14. Pande V, Stanford University. Folding@Home Distributed Computing. Available at http://folding.stanford.edu/. Accessed 28 May 2007
15. Univsersity of Wisconsin-Madison. Condor Manuals. Available at http://www.cs.wisc.edu/condor/manual/. Accessed 28 May 2007
16. University of Wisconsin-Madison. Condor World Map. Available at http://www.cs.wisc.edu/condor/map/. Accessed 28 May 2007
17. Litzkow M, Livny M, Mutka M: Condor—A Hunter of Idle Workstations. Proc of the 8th International Conf of Distributed Comput Sys 104–111, 1988
18. Livny M, Basney J, Raman R, Tannenbaum T: Mechanisms for high throughput computing. SPEEDUP J 11(1), 1997
19. Thain D, Tannenbaum T, Livny M: Distributed computing in practice: the condor experience. concurrency and computation: Practice and Experience 17(2–4):323–356, 2005
20. University of Washington. SimSET Tutorial Scatter Fraction in 99mTc SPECT Imaging. Available at http://depts.washington.edu/simset/html/user_guide/user_guide_index.html. Accessed 28 May 2007
21. Kinahan PE, Rogers JG: Analytic 3D image reconstruction using all detected events. IEEE Trans Nuc Sci 36(1):964–8, 1989
22. Rogers JG, Harrop R, Kinahan PE: The theory of three-dimensional image reconstruction for PET. IEEE Trans Med Img MI-6(3):239–43, 1987