Scientific Computing With Java

P. KNOLL, S. MIRZAEI

Department of Nuclear Medicine and PET Center, Montleartstr. 37, 1171 Vienna, Austria

Received 25 July 2007; accepted 19 October 2007

ABSTRACT: In this work we aimed to use examples from various fields (physics, medicine and structural biology) and several mathematical libraries for Java (COLT and JLAPACK) to demonstrate the advantages of scientific computing using Java. We also compared the runtimes of different Java compilers (Sun, IBM and Blackdown) and found that IBM's Java compiler results in the smallest CPU time. © 2010 Wiley Periodicals, Inc. Comput Appl Eng Educ 18: 495–501, 2010; View this article online at wileyonlinelibrary.com; DOI 10.1002/cae.20217

Keywords: scientific computing; Java

INTRODUCTION

Java is an object-orientated programming language that was developed by Sun Microsystems (http://www.sun.com). One of the most important features is the platform independence of Java programs, because it allows the exchange of software between several research groups very easily. Despite this important feature several limitations, such as speed and lack of mathematical libraries altered the use of Java for scientific computing in the past. Basically we can distinguish between Java applications, Java applets and Java servlets. Applications are invoked from the command line and executed by the Java interpreter. Servlets [1] are Java programs that run on a Web server, and service requests from a Web server providing an efficient way for a server side code to communicate with web based clients. Java applets are Java programs that execute locally within a Web browser. In general, applets do not have access to the local file system. This limitation can be overcome by signing the applets, resulting in 'trusted' applets.

The Conceptual Learning of Science (ColoS) project aims to promote the development of innovative teaching methods in science and technology. This project is focused on learning and understanding fundamental concepts in science. For teaching methods sophisticated tools, such as the 'Easy Java Simulations'-Toolkit are available (www.colos.org). The goal of this work was to analyse if Java can be used for effective scientific computing.

Before Java can be used as programming language for scientific computing, several aspects have to be considered:

Performance.
Advanced visualization.
Availability of mathematical libraries.

Correspondence to P. Knoll (peter.knoll@wienkav.at).

Java is an interpreted language and is therefore subject to a layer of processing that Fortran and C are not. Two different strategies are nowadays used to render this issue significantly: 'Hotspot' and 'Just-in-time' (JIT) compilation. Hotspot improves performance by using advanced adaptive optimisation techniques by identifying the 'hotspots', the parts of the application where most of the time is spent executing the code, and accelerating the rate of execution for this performance critical code (http://java.sun.com/products). The Hotspot performance engine is already implemented since the introduction of the Java Development Kit (JDK) 1.3. The idea behind JIT compilation is simple: Java byte-codes are compiled on a method-by-method basis, into native-code implementations for direct execution by the underlying hardware. Several JIT compilers exist, some of them freely available [2].

After performing a scientific simulation it is necessary to visualise the results. There are several possibilities in Java to perform this task. In this work we used the Berkeley's PtPlot library for curve processing (http://ptolemy.eecs.berkeley.edu/java/ptplot), Java2D to visualize results in 2D and Java3D for the 3D visualisation (http://java.sun.com/products).

As mentioned above several mathematical libraries (such as LAPACK and BLAS) are extraordinary important for scientific computations. Fortunately Java versions of these software packages already exist (http://www.netlib.org/java/f2j) and can be easily implemented. We also would like to mention CERN's Colt distribution (http://hoschek. home.cern.ch/hoschek/colt) that consists of several open source libraries for high performance scientific and technical computing.

METHODS

Fast Fourier Transformation (FFT)

The Fourier transform (FT) of a specific function is usually necessary in the analysis of experimental data. Numerical

procedures for the FT are inevitable in physics and other fields. Usually we have to deal with large number of data points and the speed of the algorithm for the FT becomes a very important issue. The straightforward implementation of the FT is very insufficient, since the computing time needed is proportional to N^2. The key element of the fast Fourier transformation (FFT) is to rearrange the term in the series and have the summation performed in a hierarchical manner. In our implementation the mixed radix FFT was applied on a two-dimensional data set. For comparison, the JDKs for Linux of Sun Microsystems (version 1.4.0) (http://java.sun.com/products) and Blackdown (version 1.3.0) (http://www.blackdown.org) and the Java distribution of IBM (version 1.3.1) (http://www.alphaworks.com) were installed. Each of the FFT calculations has been performed using each compiler using the same hardware equipment (Dell Inspiron 4100).

Ising Model

The Ising model is a simplified model of the ferromagnetism but also the grand canonical example can be mapped exactly to the canonical example of the Ising model. The mapping allows one to exploit simulation and analytical results of the Ising model to answer questions about the related model. Applying the Ising model in which spins S_i are placed on the sites i of a two-dimensional lattice each spin can take either of two values: $+1$ and -1. If there are N sites on the lattice, then the system can be in 2^N states, and the energy of any particular state is given by the Ising−Hamiltonian:

$$H = -J \sum_{\langle ij \rangle} s_i s_j - B \sum_i s_i$$

where J is the interaction energy between nearest-neighbour spins $\langle ij \rangle$, and B represents the external magnetic field. Due to the fact, that most of the interesting questions (magnetisation and specific heat) concerning the Ising model can be answered by performing simulations in zero magnetic field, we concentrated in our test on this case. The used Metropolis algorithm works by repeatedly choosing a new stat, and then accepting or rejecting it at random with a chosen acceptance probability. If the state is accepted, the computation changes the system to the new state. If not, the system remains in the same state. This process is repeated again and again. This study we wanted to compare different programming languages (IBM Java compiler (http://www.alphaworks.-com) and Lahey Fortran (http://www.lahey.com)).

Variational Calculation for the Hydrogen Atom

Quantum systems are governed by the Schrödinger equation. In particular, the solutions to the stationary form of this equation determine many physical properties of the system by hand. The stationary Schrödinger equation can be solved analytically in a very restricted number of cases, such as the harmonic oscillator or the hydrogen atom. It is of course possible to integrate the Schrödinger equation using discretisation methods but in most realistic electronic structure calculations we would need huge number of grid points, leading to important computer time and memory requirements [3].

The variational method enables us to solve the Schrödinger equation efficiently in many cases. Applying a variational calculation we have to solve a generalised eigenvalue problem. A description of the method for diagonalising a symmetric matrix and for solving the generalised eigenvalue problem can be found in many mathematical textbooks [4,5].

The electronic Schrödinger equation for the hydrogen atom reads:

$$\left[-\frac{\hbar^2}{2m} \nabla^2 - \frac{1}{4\pi\varepsilon_0} \frac{1}{r} \right] \psi(r) = E\psi(r)$$

Standard units in electronic structure physics are atomic units: the unit of distance is the Bohr radius a_0, masses are expressed in the electron mass m_e and the charge is measured in unit changes (e). In these units the Schrödinger equation for the hydrogen atom assumes the following simple form:

$$\left[-\frac{1}{2} \nabla^2 - \frac{1}{r} \right] \psi(r) = E\psi(r)$$

We try to approximate the ground state energy and wave function of the hydrogen atom in a linear variational procedure. Here we use only Gaussian basic functions; for the ground state, we only need angular momentum ($l = 0$) funczions, that have the form

$$\chi_p(r) = \mathrm{e}^{-\alpha_p r^2}$$

Optimal values for the exponents α have been found previously by solving the non-linear variational problem including the linear coefficients C_p and the exponents α. Using the values for α and filling the overlap matrix $S_{pq} = (\pi/(\alpha_p + \alpha_q))^{3/2}$ and the Hamilton matrix H it remains to determine the linear coefficients C_p in a computer program which solves the generalised eigenvalue problem

$$\mathrm{HC} = \mathrm{ESC}$$

Nevertheless, it is not advisable to program these routines, because several routines (e.g., LAPACK [4]) for a matrix diagonalisation are available. Fortunately Java versions of LAPACK are available and can be easily implemented (http://www.netlib.org).

Inelastic Hard Sphere Model

Granular materials are fascinating examples from the rich world of non-linear, dissipative, non-equilibrium systems. Granular materials consist of particles that interact only during contact with each other and dissipate energy. One of the outstanding effects in granular gases is the formation of clusters. This process of clustering should not be confused with the 'inelastic collapse', the divergence of the collision rate. Hard spheres, as a special case, are used as basic model for gases and liquids. Using the hard sphere model the particles are assumed to be perfectly rigid and follow an undisturbed motion until a collision occurs. Due to the rigidity of the interaction, the collisions occur instantaneously, so that an event-driven simulation can be performed. The change in velocity can only occur at a collision. The post-collision velocities v' of two collision partners in their centre of mass reference frame are given in terms of the pre-collision velocities v by

$$v'_{1,2} = \frac{v_{1,2} m \pm (1 + r) v_n}{2} \tag{1}$$

with $v_n = [(v_1 - v_2) \cdot \hat{n}]\hat{n}$, the normal component of the relative velocity $\mathrm{v}_1 - \mathrm{v}_2$, parallel to $\hat{n}$, the unit vector pointing along the line connecting the centres of the colliding particles. If two particles collide, their velocities are changed according to (1).

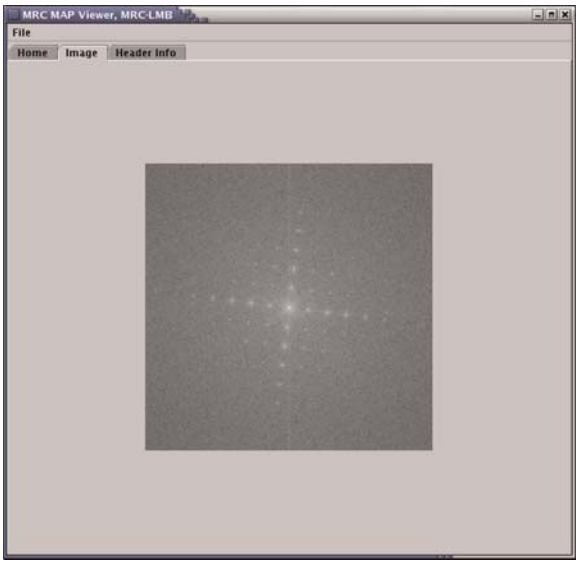

Figure 1 Fast Fourier Transformation I: Graphical User Interface.

When dissipative particles are left alone, their fluctuation energy decays due to collisions, clusters build up and grow with time until the system size is reached. The cluster growth is an interesting physical phenomenon and can be studied by computer simulations. In order to model clustering the inelastic hard sphere (IHS) is frequently used, since the event driven nature of modelling algorithm allows a sufficient simulations over many orders of magnitude in time [6].

The only drawback of the IHS, the inelastic collapse is related to a divergence of the collision frequency, a vanishing particle separation and relative velocity. To avoid the inelastic collapse, we used restitution coefficients, which value depends on the impact velocity [7].

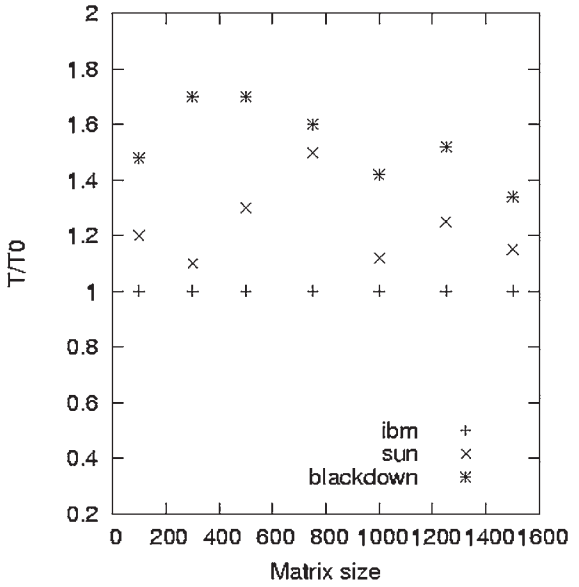

Figure 2 Fast Fourier Transformation II: Runtimes of different Java compilers and different matrix sizes (200, 400, 600, 800, 1,000).

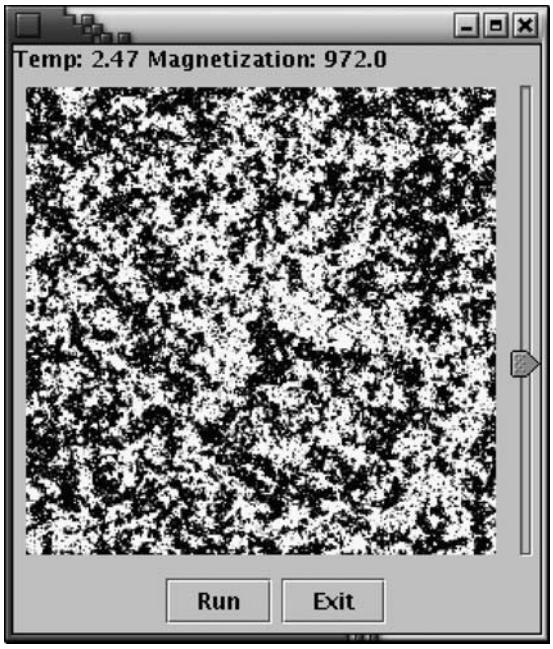

Figure 3 Ising model I: Graphical User Interface.

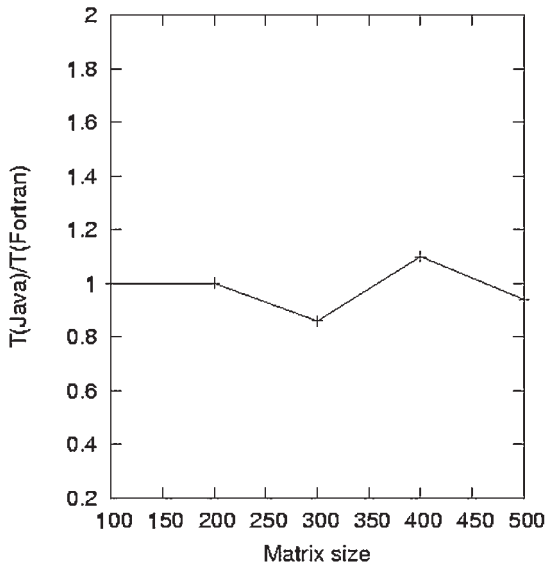

Figure 4 Ising model II: Comparison of Java with Fortran.

RESULTS

Fast Fourier Transformation

We compared the runtimes of several Java compilers using a TEM test image[1] and different matrix sizes (200, 400, 600, 800, 1,000) with each other using the IBM Java compiler as reference implementation (Fig. 1). Our results indicate that the code compiled and executed with IBM's Java compiler (http://www.alphaworks.

[1]The authors like to thank Richard Henderson, Laboratory of Molecular Biology, MRC, Cambridge for the TEM test image.

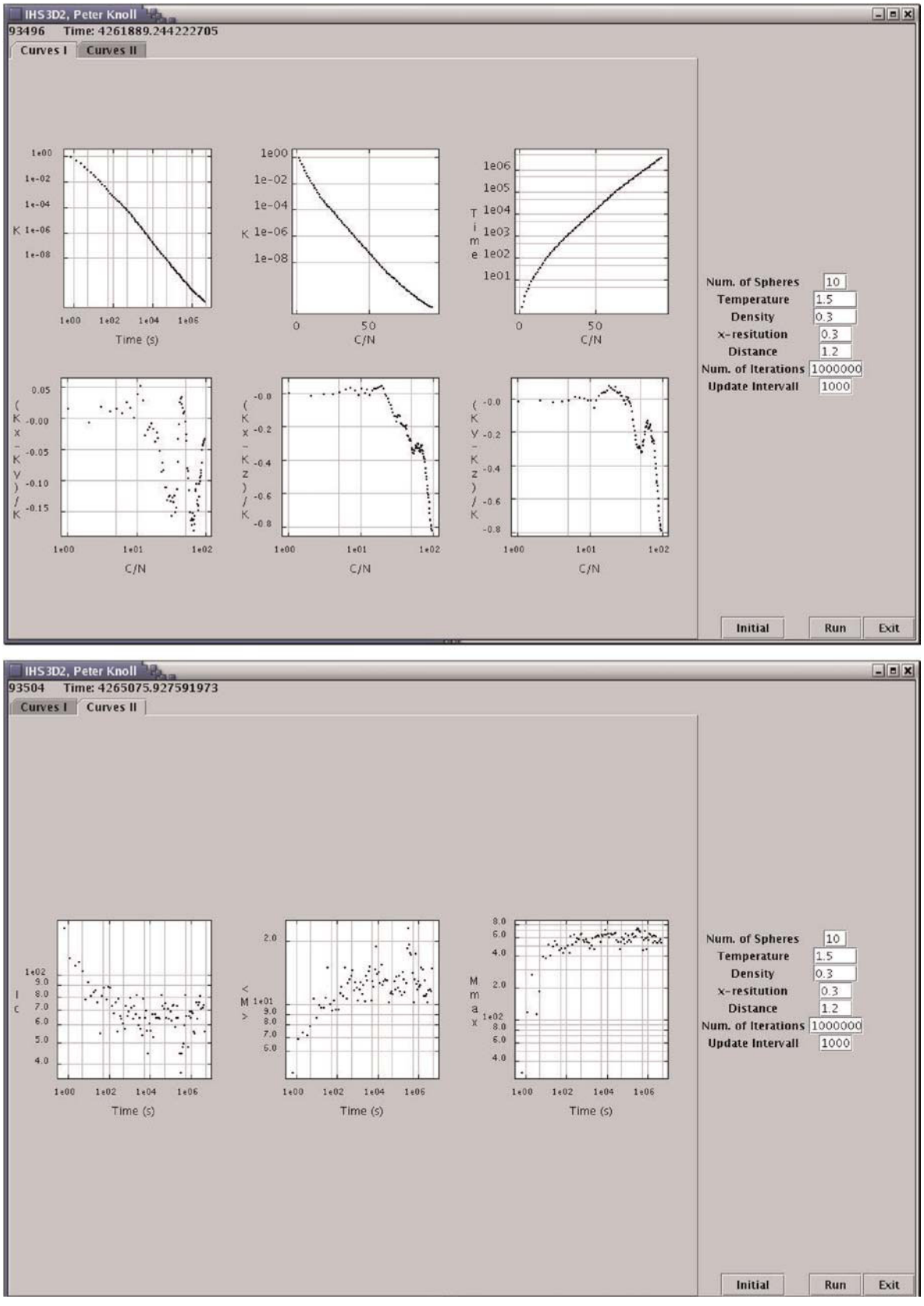

Figure 5 IHS model I: Online visualisation of the results of the simulation.

com) results in the smallest CPU time (Fig. 2). The runtimes of the Sun's Java compiler (www.colos.org) is ~1.2 times, the Blackdown implementation (http://www.blackdown.org) ~1.5 times higher than our reference implementation. The Graphical User Interface (GUI) allows the user to scale the image, to change the colour but also to change the intensity of the image.

Ising Model

We considered a system of spins, which are distributed on a square lattice. The number of Monte Carlo steps per spin (MCS) was chosen as 10,000 (Fig. 3). Each simulation has been performed 10 times for different lattice sizes (200, 400, 600, 800, 1,000). We compared the runtimes for the Java application with the runtimes of the Fortran program. In this application we did not found significant differences between the Java and the Fortran implementation (Fig. 4).

Variational Calculation for the Hydrogen Atom

A Java program was written, that fills the relevant matrices and solves the generalised eigenvalue problem. We found -0.499278 Hartree, which is close to the exact answer (-0.5 Hartree $=$ 13.0958 eV).

The Java program is listed in Appendix.

Inelastic Hard Sphere Model

The IHS model has been used to observe cluster formation of rigid particles. Initially the particles are arranged on a square lattice with velocities chosen from a Gaussian distribution. The system is evolved for 1,000 collisions using $r = 1.0$ (elastic collisions) to develop a homogeneous density. Afterwards the velocity dependent resitution coefficients are used until the ratio of collisions C and N is equal to 100. The energy loss of particles leads to a reduced separation velocity after collision and to a formation of clusters. Cluster growth was studied quantitatively using the following procedure [8]:

Two particles i and j with positions $\vec{r}_i$ and $\vec{r}_j$ and diameter d are assumed to be in contact if

$$\left|\vec{r}_i - \vec{r}_j\right| \leq S_c d \tag{2}$$

with the distance factor $S_c > 1$. Luding and Herrmann [9] showed, that the qualitative behaviour of the cluster formation is relatively insensitive from the value of S_c In our simulations S_c was set to 1.2. Using (2) to identify clusters the size of the largest cluster in the simulation is identified.

We developed a GUI, which allows changing parameters of the simulation such as the number of spheres, temperature, density and resitution coefficient. The results of the simulation are visualised using the PtPlot library (Fig. 5) and Java3D was used to visualise the clustering process of the particles (Fig. 6).

The curves in the GUI are used to visualize the results of the simulation immediately (Fig. 5). Snapshots of the evolution of the cluster formation are shown in Figure 6. The largest cluster is shown in white, the particles that do not belong to this cluster are coloured grey.

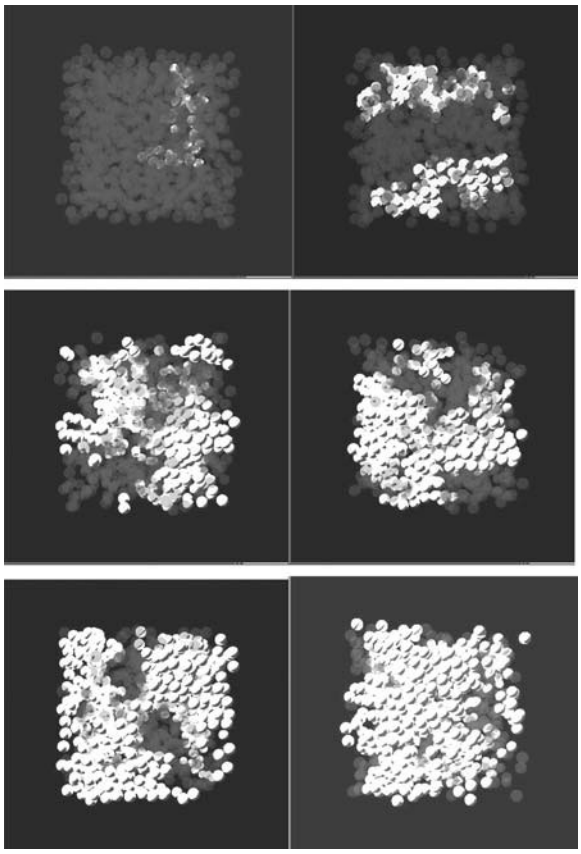

Figure 6 IHS model II: Visualisation of the cluster formation.

DISCUSSION

In this work we validated whether the programming language Java is suitable for scientific computing. We selected several aspects that are in our opinion key issues for scientific computing: Performance, visualisation and availability of mathematical libraries. The performance was evaluated for two different setups: comparison of runtimes between different Java compilers (FFT) and comparison with a different programming language, Fortran (Ising model). The FFT was performed for various matrix sizes (200, 400, 600, 800, 1,000) and several Java compilers (IBM, Sun and Blackdown). We found the smallest computation time for the IBM compiler. The two-dimensional Ising model was used to compare Java with Fortran. For this application we did not find significant differences between the two programming languages. In an article by Prechelt [10] a comparison between the performance of Java and C/C++ was made. In this article the author concluded that the programmer's efficiency and experience decidedly overwhelms the execution differences due to the programming language choice. Therefore we translated in our comparison the Java code to Fortran but did not change the programming structure. IBM's JIT compiler improves the performance of Java due to several optimisations such as monitor, method inlining and exception check (http://www.research.ibm.-com/ninja). The availability of mathematical libraries is an important issue for programming languages. In our example (hydrogen atom) we used the Java version of the BLAS and the LAPACK library, which allows an easy implementation of a routine to solve the generalised eigenvalue problem. IBM's Ninja project shows, that Java can perform BLAS matrix computations up to 90% as fast as optimised Fortran. In this context we would like to mention CERN's COLT library, which provides an

infrastructure for scalable and technical computing in Java (http://hoschek.home.cern.ch/hoschek/colt). The COLT distribution is particular useful in the domain of High Energy Physics: It contains among others, efficient data structures and algorithms for data analysis, linear algebra, statistics and Monte-Carlo simulations.

The visualisation of the results of a scientific simulation is an important issue, because images give the researcher a clearer understanding to the nature of the study. The online visualisation of the behaviour of key values such as kinetic energy or cluster size is extremely convenient to the user. The 3D visualisation of the cluster formation was done with Java3D that enables the user to observe this process online.

Appendix

```
import java.awt.*;
import java.io.*;
import org.netlib.lapack.*;
import org.netlib.util.*;

public class Hydro
{ int n, nSize;
  double z;
  double [ ] alpha;
  double [ ][ ] sMatrix, hMatrix;
  int i,k;

  public static void main (String args [ ])
  {new Hydro( );
  }

  public Hydro ( )
  {n=4;
  nSize=4;
  sMatrix=new double [nSize] [nSize];
  hMatrix=new double [nSize] [nSize];

  alpha=new double [n];
  alpha [0]=13.00773;
  alpha [1]=1.962079;
  alpha [2]=0.444529;
  alpha [3]=0.1219492;
  z=1;

  for (i=0; i<n; i++)
    alpha [i]=alpha[i]*(z*z);

  overlap ( );
  calcHMatrix ( );
  calcSpectrum ( );
  }

  public void overlap ( )
  {double factor;
  for (k=0; k<n; k++)
    {for(i=0; i<k; i++)
      {factor=Math.PI/(alpha [k]+alpha [i]);
      sMatrix [k] [i]=factor*Math.sqrt (factor);
      sMatrix [i] [k]=sMatrix [k] [i];
      }
    factor=Math.PI/(alpha [k]+alpha [k]);
    sMatrix [k] [k]=factor*Math.sqrt (factor);
    }
  }

  public void calcHMatrix ( )
  {double a,b,kinet,coul;
  for (k=0; k<n; k++)
    {for (i=0; i<k; i++)
      {a=alpha [k];
      b=alpha [i];
      kinet=calcKinet (a,b);
      coul=calcCoul (a,b);
      hMatrix [k] [i]=kinet+coul;
      hMatrix [i] [k]=hMatrix [k] [i];
      }
    a=alpha [k];
    hMatrix [k] [k]=calcKinet (a,a)+calcCoul (a,a);
    }
  }

  public double calcKinet (double a, double b)
  {double alph=a+b;
  double factor=Math.PI/alph;
  double kin=3*factor*Math.sqrt (factor)*a*b/alph;
  return kin;
  }

  public double calcCoul (double a, double b)
  {double value=−2*z*Math.PI/(a+b);
  return value;
  }

  public void calcSpectrum ( )
  {double [ ] diag=new double [nSize];
  int maxSize=100000;
  double [ ] work=new double [maxSize];
  double r, r2, val;
  intW err=new intW (0);

  DSYGV.DSYGV (1, ''V'', ''U'', n, hMatrix, sMatrix, diag, work,
maxSize, err);

  System.out.println (''Calculated value ''+diag [0]);
  System.out.println (''Exact value''+−z*z*0.5);
  }
}
```

REFERENCES

[1] S. Allamaraju, C. Buest and J. Davies, Professional Server Programming, Wrox Press Ltd., Birmingham, UK.

[2] N. Meyers, Java Programming on Linux. Waite Group Press, Indianapolis, USA.

[3] J. M. Thijssen, Computational physics. Cambridge University Press, Cambridge, United Kingdom.

[4] M. L. Boas, Mathematical methods in the physical sciences. John Wiley & Sons, New York, USA.

[5] C. Lang and N. Pucker, Mathematische Methoden in der Physik. Spektrum Akademischer Verlag, Heidelberg, BRD.

[6] F. J. Vesely, Einführung in die Computative Physik. WUV-Universitätsverlag, Vienna.

[7] N. V. Brilliantov, F. Spahn, J. M. Hertzsch, and T. Poschel, The collision of particles in granular systems, Physica A 321 (1996), 417−424.

[8] S. Miller and S. Luding, Cluster growth in two- and three-dimensional granular gases, submitted.

[9] S. Luding and H. Herrmann, Cluster growth in freely cooling media, Chaos 9 (1999), 673−681.

[10] L. Prechelt, Technical opinion: comparing Java vs. C/C++ efficiency differences to interpersonal differences, Commun ACM 42 (1999), 109−112.

BIOGRAPHIES

Peter Knoll received his PhD degree in computational physics from the University of Vienna, Vienna, Austria, in November 2000. He is currently Medical Physicist with the Department of Nuclear Medicine and PET Center.

Siroos Mirazei recived the MD degree from the University of Vienna, Austria, in 1987. Since 2007 he has been the head of the Department of Nuclear Medicine and PET Center, both at the Wilhelminenspital, Vienna, Austria.