A project-oriented course in computational physics: Algorithms, parallel computing, and graphics

C. Rebbi[a)]
Department of Physics and Center for Computational Science, Boston University, 590 Commonwealth Avenue, Boston, Massachusetts 02215

(Received 24 September 2007; accepted 11 December 2007)

I describe an undergraduate course in computational physics in which by focusing on a limited number of physics projects, the students are gradually introduced to numerical algorithms, parallel computing, graphics rendering, and visualization. © *2008 American Association of Physics Teachers.*

[DOI: 10.1119/1.2839093]

I. INTRODUCTION

In 1988 Boston University became one of the first academic institutions to embrace the emergent field of massively parallel computing with the acquisition of a Connection Machine CM-2 from Thinking Machines Corporation.[1] Many graduate students, research associates, and faculty eagerly began to port their codes to the novel architecture and in a few years it became apparent that the venture into massively parallel computing had been very successful.[2] In the wake of these efforts a group of faculty from the Departments of Chemistry, Computer Science, Electrical and Computer Engineering, and Physics (Professors Azer Bestavros, Richard Brower, David Coker, Roscoe Giles, Abdelsalam Heddaya, Steven Homer, John Straub, and the author), working under the auspices of the Center for Computational Science, formulated a new undergraduate curriculum in parallel computing. With the help of a National Science Foundation educational infrastructure grant the group developed a coordinated set of new courses: an introductory course in parallel computing to be offered to undergraduate students in all of the aforementioned departments, followed by specialized courses in the individual disciplines. In the context of this effort I developed the course Advanced Computing in Physics. The curriculum project was successful in part: the new courses were developed as planned, approved, offered, and taken by several cohorts of students. The main problem was that because of the heavy course requirements in all the disciplines involved, very few undergraduate students could take both the introductory course and one of the more advanced courses. Thus the goal of providing an integrated offering in parallel computing ultimately failed and some of the courses that were originally introduced were eventually discontinued. However, Advanced Computing in Physics continued to be offered as a stand-alone course in modern computational techniques as applied to physical systems.

In this article I describe the challenges in designing this course and the way I tried to meet them. Many graduate students found the topics treated in this course quite interesting and began attending it. To give them credit, the course was crosslisted as a graduate course, with the additional requirement that the graduate students had to complete a term project. The topics continue to be presented at the level appropriate for an undergraduate class.

II. THE CHALLENGES

Students taking the course are expected to have at least some minimal knowledge of programming, although this background is not a formal prerequisite. There have been students who have taken the course without prior programming knowledge and developed programming skills along the way, even though this lack entailed a somewhat steep learning curve at the beginning of the course. A classroom equipped with individual workstations running UNIX (now LINUX) with a shared file system has been a very valuable tool for this course because it has allowed students to check, edit, modify, compile, and execute programs during the lectures.

An immediate question that I faced when developing the course is what programming language I should adopt. Eventually I opted for FORTRAN as soon as F90 became available. My decision was motivated mostly by its array intrinsics: having simple instructions that can manipulate distributed data structures as an inherent part of the language turned out to be very useful for the algorithms I was planning to illustrate. I also use C for selected projects, including some where C is used for graphics drivers with OpenGL calls, while FORTRAN is used for subroutines that implement the physics aspects of the simulation. The simultaneous use of C and FORTRAN in graphics simulations helps to show how the two languages can be interoperated.

However, the major challenges of the course were not to be found in programming or the choice of a programming language. Rather I was facing the problem of providing a good grasp of numerical algorithms, parallel programming, and efficient computational techniques to undergraduate students who were, for the most part, still coping with the main sequence of theoretical physics courses. I decided to base the entire course on a limited number of physics projects, elementary at the beginning, where as we went deeper, notions of numerical analysis, mathematical physics, distributed data processing and, eventually, elementary graphics rendering would all come together, providing an integrated view of what computational physics is. Because many of the students begin the course with very limited or nonexistent programming skills, the code for the projects is given to them at the beginning, with the instructor spending a substantial amount of time explaining the code's features. For the first few homework assignments the students are asked to expand the code slightly. For example, they introduce statements that open a file and write data to it and then analyze the results saved in the file on the basis of what they have learned about the algorithms. Gradually they are asked to perform more challenging programming tasks. Nevertheless, most of the project programs are given to them already written. Even if the students do not develop the code for the projects themselves, the approach that is followed is one of building the code together. So, for example, for the project on the time-dependent Schrödinger equation (see Sec. VI), which involves more advanced graphics rendering, a simple written C program with OpenGL calls that just opens a window and draws a multicolored line is first illustrated to the students. Then the students are shown how to integrate some physics into the graphics through a FORTRAN subroutine that evolves the wave function and returns the data to the main graphics program. Finally, the full version of the simulation code is presented to the students.

The term projects, which are done with the assistance of the instructor, also help the students develop good programming skills. The project, for which the students need to devise a computational problem of their own choosing, think of a strategy for its solution, and implement the appropriate code, is not required of the undergraduate students, who are, however, offered the option and encouraged to do it for extra credit. Most of the undergraduate students decide to do the term project and show remarkable initiative and enthusiasm.

In the following I describe in some detail the five projects around which the course is organized.

III. BLURRING OF AN IMAGE AND DIFFUSION

To lead the students gradually into the complexities of computing, the course begins with an elementary project, namely the blurring of a discretized monochrome image. At each step the current intensity f of a pixel is replaced by a linear combination of itself and the average of its four neighbors. This step is done in parallel over the whole image, as implemented by the FORTRAN instruction

$$f=(1-a)^{*}f+(a/4)^{*}(\mathrm{cshift}(f,1,1)+\mathrm{cshift}(f,-1,1) + \mathrm{cshift}(f,1,2)+\mathrm{cshift}(f,-1,2)), \tag{1}$$

where a is the blurring factor. Students are presented with the image file and the FORTRAN program, which implements the blurring. The program also calls a simple function, written in C with X11 calls, which displays the image as it is being blurred. Students are warned that having a main "physics" program that calls a display subroutine is not the proper way to implement the graphics: the graphics code should be the driver and call a physics subroutine to provide the data. Using a physics driver does not allow the program to respond to window events such as the temporary covering of the display window, which calls for a redrawing. This simplified approach suffices for the initial projects and better graphics will come later.

Running the code shows that the blurring of the image is a diffusion process. The relation between the blurring and diffusion processes is explained by assuming that the image field can be thought of as an array that discretizes a smooth field $f(x,y)$ over a uniform square lattice:

$$f(i,j)\equiv f(x=i\delta,y=j\delta), \tag{2}$$

where δ is the distance between neighboring vertices on the lattice, that is, the lattice spacing. The assumption of smoothness is not necessarily satisfied by a real image, but it is a good start, and the consequences of its violation are discussed later. With the identification of Eq. (2) the blurring step takes the form

$$f(x,y)\to f'(x,y)=(1-a)f(x,y)+\frac{a}{4}[f(x+\delta,y)+f(x-\delta,y)+f(x,y+\delta)+f(x,y-\delta)], \tag{3}$$

where f and f' are the image field before and after the blurring step.

A Taylor series expansion of f gives

$$f(x,y)\to f'(x,y)=f(x,y)+\frac{a\delta^2}{4}\nabla^2 f(x,y)+O(\delta^4), \tag{4}$$

where $\nabla^2=\partial_x^2+\partial_y^2$ is the Laplace operator.

If the iterations are interpreted as the discretization of a continuous time evolution, where f and f' represent the values of a continuously varying field at t and $t+\epsilon$, we can write

$$f'(x,y)\equiv f(x,y,t+\epsilon)=f(x,y,t)+\frac{\partial f(x,y,t)}{\partial t}\epsilon+O(\epsilon^2). \tag{5}$$

We substitute this result into Eq. (4) and find that, neglecting terms of $O(\delta^2)$ and $O(\epsilon)$, the blurring is described by the diffusion equation

$$\frac{\partial f}{\partial t}=D\nabla^2 f, \tag{6}$$

with $D=a\delta^2/(4\epsilon)$. More importantly, if we run the mathematical steps backward, the derivation shows how we can discretize a second derivative, and hence also a Laplacian, by the central difference approximation and use this discretization to implement a numerical solution of the diffusion equation.

If the blurring factor a is greater than one, the algorithm becomes unstable. To understand the reason for the instability, the students are led to analyze the algorithm in terms of normal modes. For a rectangular image of size $N_x\times N_y$ and periodic boundary conditions the normal modes are of exponential form and are given by

$$u_{k_x,k_y}(x,y)=\exp\left[2\pi i\left(\frac{k_x x}{N_x\delta}+\frac{k_y y}{N_y\delta}\right)\right]. \tag{7}$$

Students are shown that in this basis the algorithm becomes diagonal; namely, if the image field is expanded by

$$f(x,y)=\sum\nolimits_{k_x,k_y}c(k_x,k_y)u_{k_x,k_y}(x,y), \tag{8}$$

then the blurring step induces the following transformation of the expansion coefficients

$$c(k_x,k_y)\to c'(k_x,k_y)=[1-a\lambda(k_x,k_y)]c(k_x,k_y), \tag{9}$$

where

$$\lambda(k_x,k_y)=1-\frac{\cos(2\pi k_x/N_x)+\cos(2\pi k_y/N_y)}{2}. \tag{10}$$

Because $\lambda(k_x,k_y)$ ranges between 0 and 2, the image normal mode components undergo exponential growth for $a>1$. Correspondingly, if the time step in the discretized diffusion process is too large, the algorithm to simulate the diffusion equation also fails. This argument also shows that the first mode to become unstable is the one for which the field only changes sign from site to site, which the students can check by running the code and displaying some of the values of f. [Strictly speaking, our discussion is valid only for even N_x and N_y, in which case $\lambda(k_x,k_y)=2$ for $k_x=N_x/2$, $k_y=N_y/2$. If one or both N_x, N_y are odd, the maximum of λ is less than 2 by $O(1/N_x^2)$ or $O(1/N_y^2)$, and the algorithm becomes unstable for a value of a slightly greater than 1. Details of this kind are mentioned to the students, but will be omitted for brevity in the rest of this article.]

The notion of normal modes underlying the algorithm, with the associated notions of eigenvectors, eigenvalues, and eigenfunctions, is emphasized repeatedly during the course.[3] Equally emphasized is the fact that normal modes play a similar role in mathematical physics and in numerical analysis. Developing a good grasp of one role helps us understand the others.

This first project offers a good example of data parallel calculations. The two major paradigms of parallel computing, namely symmetric multiprocessing and message passing[4] are explained to the students. The students have access to a high-end symmetric multiprocessor, currently an IBM p690 server, and the code for the blurring of the image is run in symmetric multiprocessor mode. The students are taught the basics of automatic parallelization and open multiprocessing directives and can experiment with the code, timing its performance. They are also shown how some code rewriting, in this case replacing the use of cshift by explicit indexing, might be necessary to achieve efficient parallelization. An explicit illustration of message passing is deferred to the Potts model simulation project (see Sec. V).

Knowing how to discretize the second derivative allows students to implement a numerical solution of the wave equation. To provide insight into the method, the image field is used to provide the initial data and the algorithm for solving the wave equation is applied to this field; the output is visualized every few evolution steps. Because the two-dimensional evolution of a wave is not that illuminating, the wave equation is solved in one dimension along the horizontal axis. If we denote the evolving field by $f(x,y,t)$, the wave equation is

$$\frac{\partial^2 f(x,y,t)}{\partial t^2} = v^2 \frac{\partial^2 f(x,y,t)}{\partial x^2}. \qquad (11)$$

If we impose $\partial_t f(x,y,t)|_{t=0} = -v\partial_x f(x,y,0)$, the image is seen to move in one direction with speed v, as expected from the solution $f(x,y,t)=f(x-vt,y,0)$. To show that this drift follows from the properties of the equation and is not an algorithmic quirk, the students are then asked to change the initial condition to $\partial_t f(x,y,t)|_{t=0}=0$, which leads to the solution $f(x,y,t)=[f(x-vt,y,0)+f(x+vt,y,0)]/2$, and results in an image that splits into two images traveling in opposite directions. We can build a lot on such a simple problem: for example, we can discuss aliasing of the image, relating it to the discretized versus continuum dispersion formulae.

In the final part of the project the students learn how to unblur the image. In general, the students can be expected to have learned the basics of the Fourier series and integral, but most of them are not familiar with the finite Fourier transform, which is needed to undo the blurring of the image. Thus they are taught about the finite Fourier transform, which is introduced by starting from the eigenvectors and eigenvalues of the cshift operation. After having learned about the finite Fourier transform, the students are introduced to the fast Fourier transform (FFT) algorithm, which is used to undo the effect of the blurring operation. The students are asked to write the portion of the code that performs the unblurring as a homework assignment, substitute it into the blurring program, and use an FFT subroutine that has been distributed to them. The results are spectacular, provided that the original blurring is not too pronounced. The observation that the unblurring fails if the original blurring was pushed too far, which is a consequence of finite precision arithmetic, leads naturally to a discussion of machine precision and also to a way to improve on the limitations introduced by round-off errors: the failure of the unblurring algorithm manifests itself if we try to restore the Fourier components of the image that have fallen below machine precision and are thus dominated by round-off errors. We can partially remedy the failure by foregoing the unblurring of those modes whose magnitude has fallen below the level of machine precision. Because the modes with fastest decay are those of small wavelength, we can recapture the most relevant features of the original image even if these low wavelength modes are not restored. Of course, if the original blurring proceeds to the level where the information about most modes of the image has fallen below the level of round-off errors, the blurring can no longer be undone.

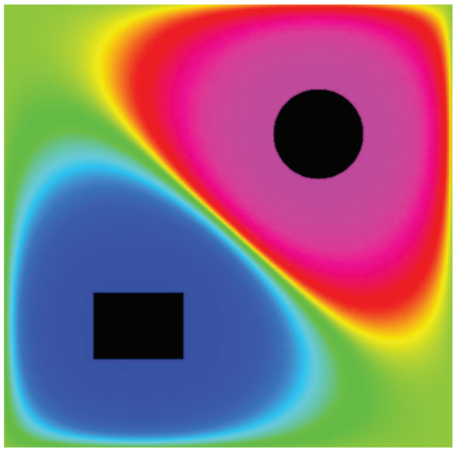

Fig. 1. Representation of the electrostatic potential obtained by solving the discretized Laplace equation. The black circle and rectangle represent two conductors held at fixed positive and negative potentials. The boundaries of the box are grounded conductors.

IV. THE POISSON EQUATION AND MULTIGRID TECHNIQUES

The material developed for the first project is used as a starting point for the solution of the second project on the evaluation of an electrostatic potential in the presence of two conductors kept at a fixed potential and surrounded by grounded, conducting surfaces. The Laplace equation is discretized on a regular lattice and a simple relaxation algorithm[5] is used for solving it. The code that illustrates this project sets up a two-dimensional geometry, with a circle and a rectangle as the two conductors within a square box, and uses color coding for visualization. As in the blurring of the image, a display subroutine is called at regular intervals by the main code, providing a good illustration of the way the relaxation algorithm works (see Fig. 1).

The problem is then generalized to solving the Poisson equation and the equation

$$-\nabla_L^2\phi + m^2\phi \equiv A\phi = \rho, \tag{12}$$

where ∇_L^2 stands for the lattice discretized Laplacian. Although Eq. (12) might have direct applications to some physical systems, the generalization is done mostly to study the behavior of the relaxation algorithm on a rectangular lattice with periodic boundary conditions, where, as for the case of the blurring of an image, the eigenvectors and eigenvalues are analytically known. Without the m^2 term in Eq. (12), the matrix A would have a zero eigenvalue, which can make the problem ill defined and complicates the discussion of the relaxation technique.

To give the students a better understanding of the relaxation algorithm, the algorithm is interpreted as a diffusion process in a fictitious time, which is proportional to the number of iterations. This interpretation makes it easy to introduce the ideas of overrelaxation and underrelaxation and to relate the rate of convergence as well as the limits of stability to the eigenvectors and eigenvalues of the operator A in Eq. (12). The students are also taught about the Gauss–Jacobi and Gauss–Seidel variants of the relaxation algorithm and how the algorithm of Gauss–Seidel can be implemented in a data parallel fashion by using a checkerboard subdivision of the data.

The study of the rate of convergence makes it clear that the algorithm undergoes critical slowing down when the resolution is increased by reducing the lattice spacing and that the modes responsible for critical slowing down are the long range small wave number modes. The observation that critical slowing down is due to the slow convergence of the long range components of the solution leads to a discussion of multigrid methods.[6] Students are presented with a program that implements a multigrid procedure for the solution of Eq. (12). The speeding up of the convergence that follows from the application of the multigrid technique is impressive.

I take advantage of the development of the simple multigrid solver to introduce students to some more sophisticated notions of programming. The multigrid cycle is defined in terms of a linked list, which is built using FORTRAN types and pointers, and the implementation of the cycle makes use of a recursive subroutine. Although pointers, linked lists, and recursive subroutines may be considered rather advanced elements of programming, from the response of the students to problems in the midterm exam where they are asked to apply some of these elements in a simplified context, it appears that most of the students develop an understanding of how they can be used.

V. STOCHASTIC SIMULATION

Students are introduced to stochastic simulations by calculating observables such as the internal energy and spin-spin correlations as functions of the temperature in the Potts model. (The Potts model is a generalization of the Ising model where the spin variables are allowed to take q different values. The Potts model reduces to the Ising model for $q=2$.) As a preliminary they are taught about pseudorandom numbers and shown the details of the congruential pseudorandom number generator as implemented in the rand48 UNIX function. They are shown how to use it to produce numbers with the desired distributions, in particular a Gaussian distribution, and how to modify RAND48 to generate pseudorandom numbers in parallel.[7]

The Ising and Potts model are then illustrated. The students are taught about stochastic simulations and given the details of the Metropolis Monte Carlo technique. Code that implements the Monte Carlo simulation of the Potts model serially is introduced first. The simplified graphics framework used for the blurring of the image and the calculation of the potential is used again to provide a visualization of the simulation. The illustration of the serial code is followed by a discussion of the parallel implementation of the Monte Carlo technique. Various methods of parallelization, based on the division of the spins into statistically independent sets, are considered. The students are shown codes that use SMP (symmetric multiprocessing) and MPI (message passing interface). The SMP code uses a checkerboard subdivision of the spins into two sets such that the spins on either sets are statistically independent, and therefore can be upgraded in parallel. In the MPI example, the various processors upgrade the spins on sets of neighboring columns, proceeding serially through the columns and making sure, through the usage of barriers, that the statistical independence of the spins that are upgraded in parallel is preserved. A considerable amount of time is spent on a detailed examination of the MPI code, which is written in C, to make the students well acquainted with the basic MPI function calls.

For this project the assignment for the students consists of slightly modifying the serial simulation code so that it writes the appropriate data to a file and running extensive simulations to explore the phase diagram of the system. Students must postprocess the data to calculate and plot the internal energy, specific heat, and the magnetization as a function of the temperature for at least two different lattice sizes, and find the signatures of a phase transition. The students do a minimal amount of finite size scaling analysis by comparing the results obtained on the two different lattice sizes.

VI. THE TIME-DEPENDENT SCHRÖDINGER EQUATION AND PATH INTEGRALS

In this project the students learn how to solve the time-dependent Schrödinger equation numerically in one dimension. They are also introduced to more sophisticated graphics techniques.

The equation to be solved is

$$i\hbar\frac{\partial\psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m}\nabla_L^2\psi(x,t) + V(x)\psi(x,t) \equiv K\psi + V\psi, \tag{13}$$

where $\psi(x,t)$ is the wave function of the particle, m is its mass, $V(x)$ is the potential energy, and ∇_L^2 represents the second derivative with respect to x discretized by the central difference approximation. The operator $-(\hbar^2/2m)\nabla_L^2$ is denoted by K because it represents the kinetic energy of the particle, and the potential is denoted by V as a reminder that it is also an operator, albeit diagonal, in x space.

Before tackling the solution of Eq. (13) the students learn some elements of graphics rendering and animation. The goal is to represent the complex wave function $\psi(x,t)$ in a way that conveys information not only about the magnitude of the wave function but also about its all important complex phase. For this purpose the simulation shows the evolution in time of $\rho(x,t)=|\psi(x,t)|^2$, which is proportional to the probability density for finding the particle at x at a given t, as well

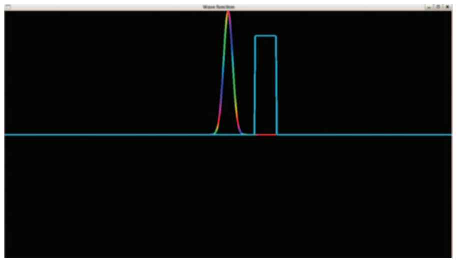

Fig. 2. Representation of the wave function $\psi(x,t)$. The profile of the curve shows $\rho(x,t)=|\psi(x,t)|^2$, which is proportional to the probability density of finding the particle at a given value of x; the color represents the phase of the complex wave function. The curve in light blue shows a potential barrier.

as of the phase $\phi(x,t)=\arg\psi(x,t)$. Figure 2 illustrates; the graphics rendering of these two quantities: the profile of the graph gives $\rho(x,t)$ and the color represents the value of $\phi(x,t)$ following the convention that $\phi=0$; that is, real ψ, is represented by red and that the color varies following the colors of the rainbow as ϕ increases from 0 to 2π, where the color returns to red.

The program displays the evolution of the wave function in time. The main idea is that the graphics component of the program is the driver, and a physics subroutine provides the data. The students are first shown how to write a program in C which uses OpenGL and the glut library to open a window and display a multicolored polyline. Then the idle function, which is repeatedly called whenever the graphics program does not need to respond to any input, is used to invoke a FORTRAN subroutine that performs a few animation steps; that is, the latter implements a few time steps of the algorithm that solves the equation and returns the wave function data. The use of a graphics driver written in C, with OpenGL calls, together with FORTRAN subroutines that initialize the wave function and potential data and implement the evolution is also an opportunity to show the students how to link C and FORTRAN programs.

At this point we consider the numerical solution of Eq. (13). A straightforward discretization of the time evolution by the one-step Euler method

$$\psi(t+\Delta t)=\left[1-\frac{i}{\hbar}(K+V)\Delta t\right]\psi(t) \tag{14}$$

produces catastrophic results. The algorithm is very unstable: the animation shows that the wave function blows up after a few time steps. The students are led to understand this instability by considering the eigenvectors and eigenvalues of the operator

$$H=K+V, \tag{15}$$

which is the Hamiltonian for the (space-discretized) system. Although the continuous time evolution, which is formally given by

$$\psi(t)\rightarrow\psi(t+\Delta t)=e^{-(i/\hbar)(K+V)\Delta t}\psi(t), \tag{16}$$

preserves the norm of all normal mode components of ψ, giving rise to a unitary evolution, the discretization of Eq. (14) violates the unitarity constraint. In the one-step method the amplitudes of all the normal modes increase at widely different rates, with the smallest wave number modes growing the fastest. A more precise explicit algorithm for the time evolution (for example, Runge–Kutta) does not help. It is crucial to maintain the unitarity of the evolution, which is accomplished by using a split operator approximation[8] and the FFT.[9] The exact evolution given by Eq. (16) is approximated by

$$\psi(t+\Delta t)=e^{-(i/\hbar)K\Delta t}e^{-(i/\hbar)V\Delta t}\psi(t). \tag{17}$$

Equation (17) is an approximation because K and V do not commute, but the error is $O[(\nabla t)^2]$. We take advantage of the fact that K is diagonal in Fourier space so that the evolution can now be implemented by repeated FFTs:

$$\psi\rightarrow\psi'=e^{-(i/\hbar)V\Delta t}\psi, \tag{18a}$$

$$\psi'\rightarrow\tilde{\psi}'=\mathrm{FFT}(\psi'), \tag{18b}$$

$$\tilde{\psi}'\rightarrow\tilde{\psi}''=e^{-(i/\hbar)K\Delta t}\tilde{\psi}', \tag{18c}$$

$$\tilde{\psi}''\rightarrow\psi''=\mathrm{FFT}^{-1}(\tilde{\psi}''), \tag{18d}$$

where ψ, ψ', $\tilde{\psi}'$, $\tilde{\psi}''$, and ψ'' are the arrays encountered in the steps leading from the space discretized value of $\psi(x,t)\equiv\psi$ to $\psi(x,t+\Delta t)\equiv\psi''$. Unitarity of the evolution can also be maintained by an implicit algorithm or by separating the real and imaginary parts of the wave function and taking advantage of the symplectic nature of the evolution equation. We use the split operator formalism because it provides an application of the FFT. The formalism also allows us to add an imaginary component to the time step in a straightforward manner, which with a suitable renormalization of the wave function causes a decay to the ground state, and gives us an natural avenue to the introduction of the path integral formulation of quantum mechanics.

The project is concluded by adding a few features that make it more user friendly: a menu, invoked by pressing a mouse key, is added to the graphics code to give the user the option of stopping the simulation, resuming it, restarting it with different parameters, or exiting. An initialization subroutine is added to allow the user to input various parameters, such as the initial particle momentum and even the value of Planck's reduced constant $\hbar$. The visualization offered by this project also turns out to be a valuable tool to help students develop a better understanding of various aspects of quantum mechanics, such as the dispersion of a wave packet, the difference between group and phase velocity, tunneling, or the classical limit (a reduction of the input value of $\hbar$ by a factor of 4 is sufficient to see the motion of a wave packet develop distinctly classical features).[10]

After having taught this project a couple of times, I realized that I had all the ingredients to explain the path integral formulation of quantum mechanics. To make a computer simulation possible, I start from the problem of calculating the partition function

$$Z=\sum_n e^{-E_n/kT}, \tag{19}$$

where E_n are the eigenvalues of the Hamiltonian H of Eq. (15). Z is recast in the form

$$Z=\mathrm{Tr}\,e^{-H/kT}=\mathrm{Tr}\,e^{-\beta H}, \tag{20}$$

with the standard notation $\beta=1/(kT)$. It is convenient to introduce a fictitious time variable $t=\hbar\beta$. Equation (20) is then recast in the form

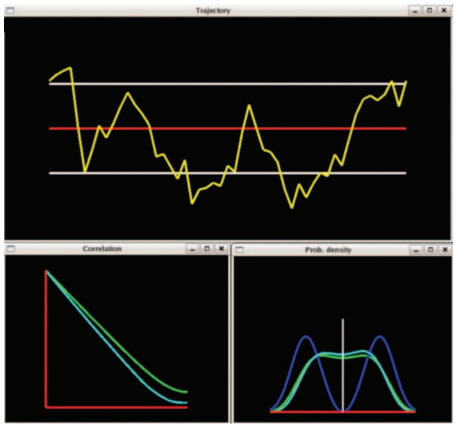

Fig. 3. Simulation of a path integral. The top window displays the trajectories as they are sampled. The left bottom window displays the correlation $\langle x_0 x_i \rangle$ as computed in the course of the simulation (in light blue) versus the approximate result obtained from the two lowest energy eigenstates (in green). The right bottom window shows the distribution of values of x as measured in the simulation (in light blue) versus the corresponding distributions in the lowest (green) and first excited (dark blue) quantum levels.

$$Z = \mathrm{Tr}\, e^{-Ht/\hbar} = \mathrm{Tr}\, e^{-H\Delta t/\hbar} e^{-H\Delta t/\hbar} \cdots e^{-H\Delta t/\hbar} \ (N \text{ factors}). \tag{21}$$

If we use the split operator approximation [see Eq. (17)] and repeatedly apply the Fourier transform, we arrive at the path integral formula[11]

$$Z = \left[\sqrt{\frac{m}{2\pi\hbar\Delta t}}\right]^N \int dx_0 \int dx_1 \int dx_2 \cdots \int dx_{N-1} \exp\left\{ -\frac{1}{\hbar}\sum_{i=0}^{N-1}\left[m\frac{(x_{i+1}-x_i)^2}{2\Delta t} + V(x_i)\Delta t \right]\right\}, \tag{22}$$

with $x_N = x_0$. (All the intermediate steps are explained to the students slowly and in great detail.)

The exponential in Eq. (22) can be considered as a probabilistic measure and a program that simulates the corresponding distribution of the variables x_i (a trajectory) using the Metropolis algorithm developed for the Potts model in Sec. V is illustrated. The graphics framework used to represent the evolution of the wave function is slightly revised to accommodate three windows (the students are shown how to do so), and we arrive at a program that displays a "live" path integral, that is, the trajectories as they are sampled and the gradual build up of observables (see Fig. 3).

VII. MOLECULAR DYNAMICS

The students are given a very concise introduction to molecular dynamics (MD) simulations. They are told that, in general, MD refers to simulations where we reproduce, usually on a smaller scale and often with suitable simplifications, the time evolution of a physical system. So defined, MD is a very broad field, encompassing a wide range of physical phenomena. Molecular dynamics is exemplified in the fifth project, which consists of simulating the evolution of a system of atoms interacting with the Lennard–Jones potential

$$V(r) = 4\left[\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6}\right]. \tag{23}$$

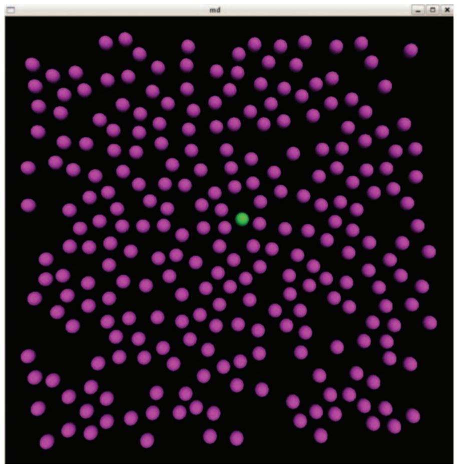

Fig. 4. Snapshot of the evolution of a set of atoms interacting with the Lennard–Jones potential. The atoms move in a plane. One atom is given a different color to make it easier to follow its motion.

The students are taught about the leap-frog (or velocity Verlet) algorithm for solving the equations of motion and are shown how to use it. The graphics program used for displaying the evolution of the wave function is modified to render three-dimensional graphics and the atoms are represented by colored spheres. As in the programs for the Schrödinger equation and the path integral, the C graphics program is the main program and calls a physics subroutine, written in FORTRAN, when it is in an idle state. The subroutine implements the evolution. A menu allows the user to stop and resume the animation, change the temperature and volume, and exit. A snapshot of the molecular dynamics simulation is shown in Fig. 4.

This project leads to a discussion of how the complexity of the calculation increases with N, the number of particles. The program calculates all the interatomic forces and thus the total number of operations increases as N^2. But the Lennard–Jones interaction is short range, and thus we expect the force of interaction between atoms that are farther apart than some maximum distance d to be negligible. Accordingly, the code is modified to perform the calculation of the force only when the difference between all of the atoms' coordinates is less than d. Imposing a cutoff on the maximum distance for which the force interaction will be evaluated reduces the total number of force calculations to $O(N)$, but not the total number of operations, because the number of coordinate comparisons is still $O(N^2)$. The discussion is continued by explaining to students how we can reduce the calculation to $O(N)$ operations by dividing space into cells and keeping track of which atoms occupy the various cells. Then only the forces between atoms in neighboring cells have to be calculated, leading to a calculation that scales like N. However, the programming needed to implement the simulation becomes substantially more complex, and thus is not tackled in class. A very brief overview of methods that can be used to accelerate MD in the presence of long-range forces concludes the course.

VIII. CONCLUSIONS

The feedback questionnaires distributed at the end of the course show that quite generally the students found the course very useful. One of the students who took the course was awarded a Department of Energy computational science graduate fellowship. The students' performance on homework assignments and on the term project (see the following) shows that even students who started the course with little or no previous knowledge of programming were able to develop basic skills in the methods of modern scientific computing as applied to physics. As mentioned in Sec. I, the graduate students attending the course needed to do a term project as one of the course requirements. Although a term project is not mandatory for the undergraduate students, they are offered the option of doing one for extra credit. The choice of topic for the individual term projects is left to the students, who must come up with an interesting problem: the instructor only provides guidance about feasibility, methods, and implementation. The wide range of term projects done by the students over the years, from the solution of the motion of a triple pendulum to a simulation of traffic, and the enthusiasm that the students generally have shown in carrying out the projects are further indication that they derived much benefit from the course.

The number of undergraduate students enrolled in the course has been small, of the order of five students out of a typical graduating class of approximately 20 physics majors. The number of graduate students taking the course has been larger, of the order of eight to ten students. Although these numbers make the class size reasonable, there is sentiment within the physics faculty that more undergraduate students could profit from the course than are actually taking it. Consideration has been given to teaching a revised version of the course, offered only to undergraduate students, as a requirement for the physics curriculum, but the large number of courses that are already required has prompted the faculty to decide against imposing an extra course requirement. Rather it was decided to maintain the course as an elective, but to lower the barriers that may dissuade larger numbers of students from taking the course. Some of these barriers may just be a matter of perception. It is likely that many undergraduate students who would otherwise like to attend the course are afraid that it may be too difficult without prior experience in computing. The word "advanced" in the title of the course, a vestige of the original multidisciplinary curriculum development project, might have deterred some undergraduate students from taking it. So the course is now being slightly redefined to make it more accessible and attractive to a wider audience. Although the general spirit of the course will be maintained, explicit programming instruction will be offered at the beginning of the course, the title will be changed to Introduction to Computational Physics, and some of the more advanced topics, such as multigrid methods and the simulation of the path integral, will be taught seminar style in discussion sessions. Also, the course will no longer be crosslisted as a graduate course, leaving it as a course with a strictly undergraduate audience. (We already offer a more traditional graduate course in computational physics and the introduction of a strictly graduate version of the course described in this article will be considered in a later year.) The revised undergraduate course will be offered first in the 2008/2009 academic year and for many years to come, offering to physics majors basic training in computing, which is becoming ever more important for so many physics applications.

ACKNOWLEDGMENTS

The course described in this article was developed with assistance from NSF Grant No. CDA-9214902. The code for the molecular dynamics project was developed based on a code originally written by Richard Brower. The code for the MPI simulation of the Potts model was developed in collaboration with Francis W. Starr of Wesleyan University, who was then a graduate student at Boston University.

[a]Electronic mail: rebbi@pthind.bu.edu

[1] See, for example, Andrew I. Mel'uk, Roscoe C. Giles, and Harvey Gould, "Molecular dynamics simulation of liquids on the Connection Machine," Comput. Phys. **5**(3), 311–318 (1991) or ⟨en.wikipedia.org/wiki/Connection_Machine⟩.

[2] R. Giles and C. Rebbi, "Computational Center develops practical experience in parallel computing," Comput. Phys. **6**(2), 122–130 (1992).

[3] See for example, Gene H. Golub and Charles F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, MD, 1996), 3rd ed.

[4] See for example, Barbara Chapman, Gabriele Jost, and Ruud van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming* (MIT Press, Cambridge, MA, 2007); Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, *MPI: The Complete Reference* (MIT Press, Cambridge, MA, 1998).

[5] See for example, David M. Young, *Iterative Solution of Large Linear Systems* (Academic Press, New York, 1971), Dover reprint, 2003; Louis A. Hageman and David M. Young, *Applied Iterative Methods* (Academic Press, New York, 1981), Dover reprint, 2003.

[6] Achi Brandt, "Multilevel adaptive solutions to boundary-value problems," Math. Comput. **31**, 333–390 (1977); William L. Briggs, Van Emden Henson, and Steve F. McCormick, *A Multigrid Tutorial* (SIAM, Philadelphia, 2000), 2nd ed.

[7] Fred James, "A review of pseudorandom number generators," Comput. Phys. Commun. **60**, 329–344 (1990).

[8] H. Gould, J. Tobochnik, and W. Christian, *Introduction to Computer Simulation Methods* (Addison–Wesley, San Francisco, 2007), pp. 689–698.

[9] M. D. Feit, J. A. Fleck, and A. Steiger, "Solution of the Schrödinger equation by a spectral method," J. Comput. Phys. **47**, 412–433 (1982).

[10] See EPAPS Document No. E-AJPIAS-76-009803 for a tar file with the code for this project. This document can be reached through a direct link in the online article's HTML reference section or via the EPAPS homepage (http://www.aip.org/pubservs/epaps.html).

[11] R. P. Feynman and A. R. Hibbs, *Quantum Physics and Path Integrals* (McGraw-Hill, New York, 1965); J. Tobochnik, G. Batrouni, and H. Gould, "Quantum Monte Carlo on a lattice," Comput. Phys. **6**(6), 673–680 (1992).