

## Parallelization of quantum few-body calculations

(Roman Kuralev, Department of Computational Physics, Saint-Petersburg State University)  
Joint Advanced Student School 2008

### Introduction

The main goal is calculation of the bound and resonant states properties of quantum three-body systems. This problem is important for the quantum mechanics and of course it presents a challenge from the computational point of view because few-dimensional Schrödinger equation should be solved. It is important to make calculations with high accuracy ( $\sim 4$  ppm) because some experimental methods allow to measure spectra with high accuracy and calculation methods must ensure such accuracy.

In this work the sequential code of the ACE program was parallelized. Then some test were performed (calculation of ground state energy of the helium atom).

### Problem statement

We have three-body quantum system with central-force interaction. This three particles interact with help of Coulomb potential. So, the problem is to calculate bound and resonance states of this system. During the solving of this problem, we need to solve the eigenvalue problem for large sparse matrices with up to 100 000 elements and with matrix sparseness of order 0,01. It means that we have only 1% of non-zero elements.

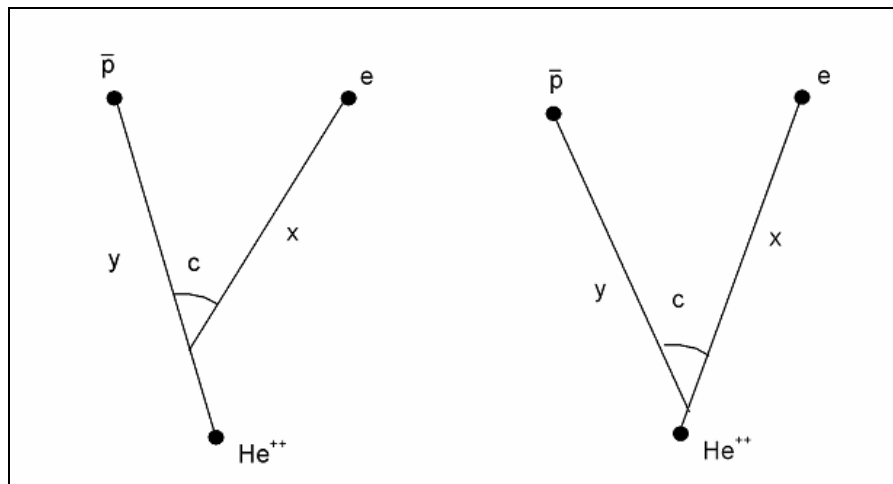
It is necessary to solve six-dimensional Schrödinger equation.

### Method of calculations

Here we will neglect nonradiative decay channels; therefore, the energy levels will be truly bound states. The total wave function of the system with the angular momentum  $J$  and its projection  $M$  can be expanded in terms of Wigner  $D$  functions: (1)

$$\pm\Psi^{JM} = \sum_{s=0,1}^J \frac{1}{\sqrt{2+2\delta_{s0}}} \left[ D_{Ms}^J(\alpha, \beta, \gamma) \pm (-1)^s D_{M-s}^J(\alpha, \beta, \gamma) \right] \Psi^{(Js)}(R)$$

Here plus and minus signs refer to states with normal and anomalous spatial parity, respectively. The Euler angles  $\alpha$ ,  $\beta$ , and  $\gamma$  define a rotation of the body-fixed frame with respect to the laboratory-fixed one and  $R$  is a three-dimensional coordinate in the body-fixed frame. As the states of the anomalous spatial parity decay rapidly via Auger transitions, the only states of interest can be described as  $^+\Psi^{JM}$ . In the following, we thus consider only such states and omit the plus in the designation of the wave function. There are a few different possible choices of the body-fixed coordinate  $R$ . The usual one is “standard” Jacobi coordinates (left picture).



Here the first vector joins two heavy particles helium and antiproton and the second connects the electron and the center of mass of the three-body system. The main reason for that choice is that it is possible to separate the electron motion for the case of infinite masses of the heavy particles. However, we have chosen another Jacobi coordinate system. Namely, let particles 1, 2, and 3 represent the antiproton, the electron, and the helium nucleus, respectively. Then  $x$  is the distance between particles 2 and 3 i.e., electron and helium nucleus,  $y$  is the distance between particle 1 (antiproton) and the center of mass of particles 2 and 3, and  $\phi$  is the angle between  $x$  and  $y$ . In short, this choice is caused by a simpler structure of the wave function in these coordinates. After substituting expansion (1) into the Schrödinger equation and using an orthogonality relation for the D function one can derive the system of equations

$$-i\sqrt{1+\delta_{s0}} \frac{\lambda_+(j,s)}{2\mu_{1,23}y^2} \left( \frac{\partial}{\partial \phi} + (1+s)\cot \phi \right) \Psi^{(Js+1)} = 0, \quad s=0,\dots,J,$$

where  $\lambda_{\pm}(J,s) = [J(J+1) - s(s \pm 1)]^{1/2}$  and  $\psi^{(J-1)} \equiv 0$  and the diagonal part of the kinetic-energy operator and the Coulomb potential  $V_{\text{Coul}}$  are defined as

$$-\Delta_x^{(s)} = -\frac{1}{2\mu_{23}x^2} \left[ x \frac{\partial^2}{\partial x^2} x + \left( \frac{\partial^2}{\partial \phi^2} + \cot \phi \frac{\partial}{\partial \phi} - \frac{s^2}{\sin^2 \phi} \right) \right],$$

$$-\Delta_y^{(s)} = -\frac{1}{2\mu_{1,23}y^2} \left[ y \frac{\partial^2}{\partial y^2} y - [J(J+1) - 2s^2] + \left( \frac{\partial^2}{\partial \phi^2} + \cot \phi \frac{\partial}{\partial \phi} - \frac{s^2}{\sin^2 \phi} \right) \right],$$

$$V^{\text{Coul}}(x,y,\phi) = -\frac{2}{r_{13}} - \frac{2}{r_{23}} + \frac{1}{r_{12}}.$$

The interparticle distances  $r_{ij}$  can easily be expressed in terms of Jacobi coordinates and the reduced masses  $\mu$ . In a numerical realization this produces a band matrix with a fixed bandwidth for any angular momentum  $J$ . This decreases computational requirements substantially. Numerically we have realized our equation using the finite-element method. Here we outline the general ideas of this method.

The three-dimensional space formed by  $x$ ,  $y$ , and  $c = \cos \phi$  is divided into some number of rectangular boxes numbered by  $i$ . Each component of the total wave function is expanded in a finite-element basis such that.

$$\tilde{\psi}^{(Js)}(x,y,c) = \sum_{im} v_{im} f_{im}(x,y,c).$$

For the sake of simplicity, we omit the indices ( $J_s$ ) on the right-hand of equation. Here  $v_{im}$  are expansion coefficients defined in each element  $i$  and restricted through continuity conditions for the wave function at element boundaries. A basis function  $f_{im}(x, y, c)$  has in the finite-element algorithm the property  $f_{im}(x, y, c) = 0$  for  $(x,y,c) \notin$  element  $i$ . Then the coefficients  $v_{im}$  and the energy  $E$  are obtained by means of minimization of a functional  $\tilde{H}v = E \tilde{S}v$ , where

$$(\tilde{H})_{im,jk} = \langle f_{im} | H | f_{jk} \rangle,$$

$$(\tilde{S})_{im,jk} = \langle f_{im} | f_{jk} \rangle.$$

The best approximation is evaluated by solving a generalized eigenvalue problem.

It is worth noticing that the global basis set is a particular case of the method described above where one box only is used. However, the FEM manifests its advantages in using many boxes, defined by the locality of the basis functions. One such advantage is that the final matrices H and S become banded and relatively sparse and the overlap matrix S is well defined in contradiction to the ill-defined one appearing for a global basis set. Let us now discuss some details of the general scheme used for our calculations. The basis functions are expressed as products of one-

dimensional basis functions  $f_{im}(x,y,c) = f_{i,n(m)}^{(x)}(x) f_{i,k(m)}^{(y)}(y) f_{i,l(m)}^{(c)}(c)$

Such a representation of the basis functions simplifies an evaluation of the matrix elements and reduces most three-dimensional integrals to a product of one-dimensional ones.

The number of boxes for the space coordinates x,y is defined by convergence properties.

However, one box only was chosen for the angular variable c. The one-dimensional basis functions in every box were chosen to be

$$f_{i,n}^{(x)}(x) = P_n(\hat{x}) \exp(-\nu x),$$

$$f_{i,k}^{(y)}(y) = P_k(\hat{y}) g_d(y),$$

$$f_{i,l}^{(c)}(c) = P_l(c).$$

Here  $P_n$  is a Legendre polynomial. Variables  $\hat{x}$  and  $\hat{y}$  appear instead of x and y due to a normalization of every one-dimensional box to the interval [-1, 1]. This normalization causes a maximum linear independence of the basis functions in the box and decreases the ill definiteness of the overlap matrix. The above choice of the basis functions is guided by the fact that we are dealing with a Coulomb potential only. This makes it possible to use simple analytical expressions for the potential matrix element and to reduce the CPU time of the calculations as well as to increase the accuracy. We have fixed the damping function  $g_d$  to meet the asymptotic behavior of the wave function at the origin and at infinity:

$$g_d(y) = C y^J \exp\left(-\frac{\mu_{1,23}}{J+1+q} y\right).$$

Here q is an arbitrary parameter. Thus the only nonlinear parameters to be optimized are n and q. We have found a slow eigenvalue dependence on these parameters.

### Arnoldi method

In numerical linear algebra, the **Arnoldi iteration** is an eigenvalue algorithm and an important example of iterative methods. The Arnoldi iteration was invented by W. E. Arnoldi in 1951.

The term *iterative method*, used to describe Arnoldi, can perhaps be somewhat confusing. Note that all general eigenvalue algorithms must be iterative. This is not what is referred to when we say Arnoldi is an iterative method. Rather, Arnoldi belongs to a class of linear algebra algorithms (based on the idea of Krylov subspaces) that give a partial result after a relatively

small number of iterations. This is in contrast to so-called *direct methods*, which must complete to give any useful results.

Arnoldi iteration is a typical large sparse matrix algorithm: It does not access the elements of the matrix directly, but rather makes the matrix map vectors and makes its conclusions from their images. This is the motivation for building the Krylov subspace.

An intuitive method for finding an eigenvalue (specifically the largest eigenvalue) of a given  $m \times m$  matrix  $A$  is the power iteration. Starting with an initial random vector  $b$ , this method calculates  $Ab, A^2b, A^3b, \dots$ . This sequence converges to the eigenvector corresponding to the largest eigenvalue,  $\lambda_1$ . However, much potentially useful computation is wasted by using only the final result,  $A^{n-1}b$ . This suggests that instead, we form the so-called *Krylov matrix*:

$$K_n = [b \quad Ab \quad A^2b \quad \dots \quad A^{n-1}b].$$

The columns of this matrix are not orthogonal, but in principle, we can extract an orthogonal basis, via a method such as Gram–Schmidt orthogonalization. The resulting vectors are a basis of the *Krylov subspace*,  $\mathcal{K}_n$ . We may expect the vectors of this basis to give good approximations of the eigenvectors corresponding to the  $n$  largest eigenvalues, for the same reason that  $A^{n-1}b$  approximates the dominant eigenvector. The Arnoldi iteration uses the stabilized Gram–Schmidt process to produce a sequence of orthonormal vectors,  $q_1, q_2, q_3, \dots$ , called the *Arnoldi vectors*, such that for every  $n$ , the vectors  $q_1, \dots, q_n$  span the Krylov subspace  $\mathcal{K}_n$ . Explicitly, the algorithm is as follows:

- Start with an arbitrary vector  $q_1$  with norm 1.
- Repeat for  $k = 2, 3, \dots$ 
  - $q_k \leftarrow Aq_{k-1}$
  - **for**  $j$  from 1 to  $k-1$ 
    - $h_{j,k-1} \leftarrow q_j^* q_k$
    - $q_k \leftarrow q_k - h_{j,k-1}q_j$
  - $h_{k,k-1} \leftarrow \|q_k\|$
  - $q_k \leftarrow \frac{q_k}{h_{k,k-1}}$

The  $j$ -loop projects out the component of  $q_k$  in the directions of  $q_1, \dots, q_{k-1}$ . This ensures the orthogonality of all the generated vectors. The algorithm breaks down when  $q_k$  is the zero vector. This happens when the minimal polynomial of  $A$  is of degree  $k$ . Every step of the  $k$ -loop takes one matrix-vector product and approximately  $4km$  floating point operations.

### Calculation algorithm

There are three main stages of calculation:

1. Basis definition
2. Matrix elements calculation (FEM)
3. Solving of generalized eigenvalue problem

So, we used the sequential code of the ACE program package, and its algorithm is:

1. Data input (\*.inp file)
2. Building a 3D grid, establishing the topology, implementing boundary conditions
3. Matrix building
4. Generalized eigenvalue problem solving
5. Data output (eigenvalue goes to the screen and saves to the \*.eig file)

## Message Passing Interface (MPI)

It is both a computer specification and is an implementation that allows many computers to communicate with one another. It is used in computer clusters. MPI "is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation", "MPI includes point-to-point message passing and collective (global) operations, all scoped to a user-specified group of processes." The MPI is a language-independent communications protocol used to program parallel computers. MPI's goals are high performance, scalability, and portability. MPI is not sanctioned by any major standards body; nevertheless, it has become the *de facto* standard for communication among processes that model a parallel program running on a distributed memory system. Actual distributed memory supercomputers such as computer clusters often run these programs. The principal MPI-1 model has no shared memory concept, and MPI-2 has only a limited distributed shared memory concept.

Most MPI implementations consist of a specific set of routines (API) callable from Fortran, C, or C++ and from any language capable of interfacing with such routine libraries. The advantages of MPI over older message passing libraries are portability (because MPI has been implemented for almost every distributed memory architecture) and speed (because each implementation is in principle optimized for the hardware on which it runs). MPI is supported on shared-memory and NUMA (Non-Uniform Memory Access) architectures.

MPI is a specification, not an implementation. MPI has Language Independent Specifications (LIS) for the function calls and language bindings. The first MPI standard specified ANSI C and Fortran-77 language bindings together with the LIS. The draft of this standard was presented at Supercomputing 1994 (November 1994) and finalized soon thereafter. About 128 functions comprise the MPI-1.2 standard as it is now defined.

There are two versions of the standard that are currently popular: version 1.2 (shortly called MPI-1), which emphasizes message passing and has a static runtime environment (fixed size of world), and, MPI-2.1 (MPI-2), which includes new features such as parallel I/O, dynamic process management and remote memory operations. MPI-2's LIS specifies over 500 functions and provides language bindings for ANSI C, ANSI Fortran (Fortran90), and ANSI C++. Interoperability of objects defined in MPI was also added to allow for easier mixed-language message passing programming. A side effect of MPI-2 standardization (completed in 1996) was clarification of the MPI-1 standard, creating the MPI-1.2 level.

It is important to note that MPI-1.2 programs, now deemed "legacy MPI-1 programs," still work under the MPI-2 standard although some functions have been deprecated. This is important since many older programs use only the MPI-1 subset.

The MPI interface is meant to provide essential virtual topology, synchronization and communication functionality between a set of processes (that have been mapped to nodes/servers/ computer instances) in a language independent way, with language specific syntax (bindings), plus a few features that are language specific. MPI programs always work with processes, although commonly people talk about processors. When one tries to get maximum performance, one process per CPU is selected, as part of the mapping activity; this mapping activity happens at runtime, through the agent that starts the MPI program, normally called mpirun or mpiexec.

The implementation language for MPI is different in general from the language or languages it seeks to support at runtime. Most MPI implementations are done in a combination of C, C++ and assembly language, and target C, C++, and Fortran programmers. However, the implementation language and the end-user language are in principle always decoupled.

The initial implementation of the MPI 1.x standard was MPICH, from Argonne National Laboratory (correctly pronounced MPI-C-H, not pronounced as a single syllable) and Mississippi State University. IBM also was an early implementor of the MPI standard, and most supercomputer companies of the early 1990s either commercialized MPICH, or built their own implementation of the MPI 1.x standard. LAM/MPI from Ohio Supercomputing Center was

another early open implementation. Argonne National Laboratory has continued developing MPICH for over a decade, and now offers MPICH 2, which is an implementation of the MPI-2.1 standard. LAM/MPI, and a number of other MPI efforts recently merged to form a new world-wide project, called the Open MPI implementation, but this name does not imply any connection with a special form of the standard. There are many other efforts that are derivatives of MPICH, LAM, and other works, too numerous to name here. Recently, Microsoft added an MPI effort to their Cluster Computing Kit (2005), based on MPICH 2. MPI has become and remains a vital interface for concurrent programming to this date.

Many Linux distributions include MPI (either or both MPICH and LAM, as particular examples), but it is best to get newest versions from MPI developer sites. Many vendors distribute customised versions of existing free software implementations which focus on better performance and stability.

### Parallel algorithm

1. Data input
2. Task distribution
3. Parallel matrix calculation
4. MPI\_Reduce
5. Eigenvalue problem solving
6. Data output

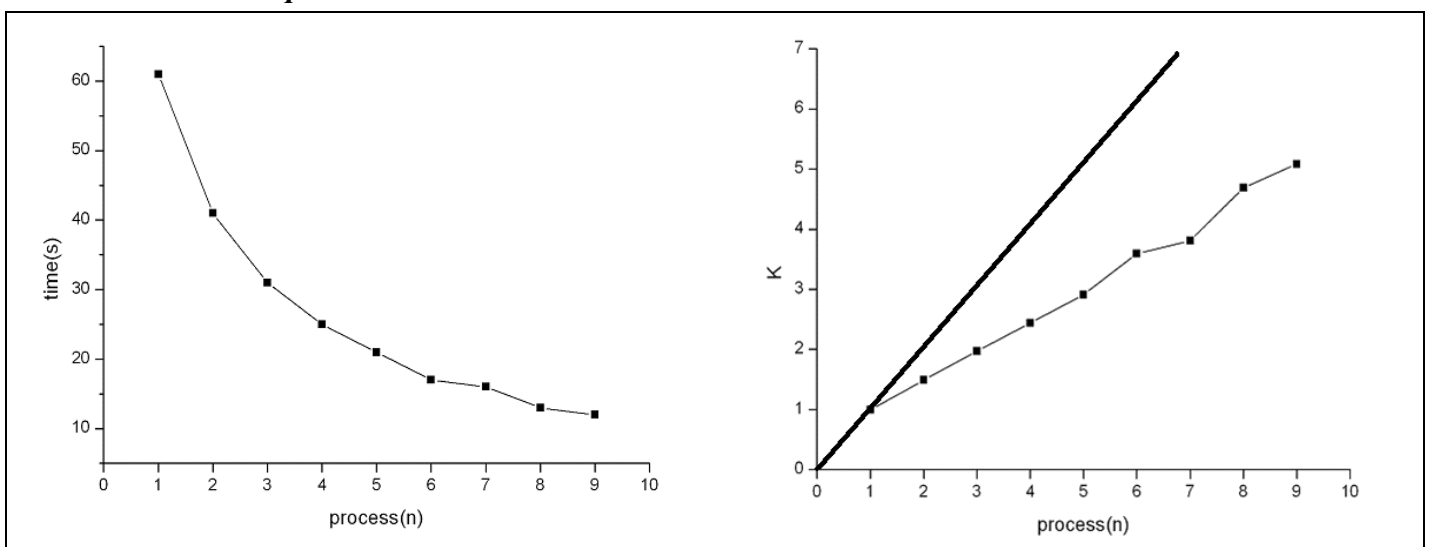
### Results & conclusions

- The program was parallelized. It is obvious that the parallel version is much more faster then sequential.
- The parallel version works correctly and it was confirmed by calculation of the helium atom energy. This result is in good agreement with the experiment.
- **Theoretical energy value (helium) is:**

$$E_{th} = -2.9032 \text{ conventional units (the proton mass is 1, the Plank's constant is 1)}$$

**Experimental energy value is:**

$$E_{exp} = -2.9037 \text{ c.u.}$$



Time of calculation

Speedup