Computer Physics Communications 204 (2016) 121–131

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Fast GPU-based calculations in few-body quantum scattering

V.N. Pomerantsev [a,*], V.I. Kukulin [a], O.A. Rubtsova [a], S.K. Sakhiev [b]

[a] Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Leninskie gory 1(2), Moscow, 119991, Russia

[b] L.N. Gumilyov Eurasian National University, Astana, 010000, Kazakhstan

ARTICLE INFO

Article history:
Received 1 December 2015
Accepted 30 March 2016
Available online 8 April 2016

Keywords:
Quantum scattering theory
Discretization of the continuum
Faddeev equations
GPU

ABSTRACT

A principally novel approach towards solving the few-particle (many-dimensional) quantum scattering problems is described. The approach is based on a complete discretization of few-particle continuum and usage of massively parallel computations of integral kernels for scattering equations by means of GPU. The discretization for continuous spectrum of few-particle Hamiltonian is realized with a projection of all scattering operators and wave functions onto the stationary wave-packet basis. Such projection procedure leads to a replacement of singular multidimensional integral equations with linear matrix ones having finite matrix elements. Different aspects of the employment of multithread GPU computing for fast calculation of the matrix kernel of the equation are studied in detail. As a result, the fully realistic three-body scattering problem above the break-up threshold is solved on an ordinary desktop PC with GPU for a rather small computational time.

1. Introduction

Nowadays there appeared a new possibility to use the Graphics Processing Units (GPU) for many time-consuming problems. The efficient employment of GPU can transform an ordinary PC into a desktop supercomputer. There is no necessity to argue that such innovation is unmeasurably cheaper and more accessible for many researchers in the world than the employment of a modern high-power supercomputer. However, due to the special GPU architecture the usage of GPU is effective only for those problems where numerical scheme of solution can be realized with a high degree of parallelism. The high effectiveness of the so-called General Purpose Graphics Processing Unit (GPGPU) computing has been demonstrated in many areas of quantum chemistry, molecular dynamics, seismology, etc. (see the detailed description of different GPU applications in Refs. [1–4]).

Solution of few-body scattering problems, especially above the three-body breakup threshold, no matter in differential or integral formalism, is an appropriate candidate for such GPU implementation because it involves a very large amount of calculations and therefore requires an extensive use of modern computational facilities such as powerful supercomputers. As a vivid example, we note that one of the most active and successful groups in the world in this area – the Bochum–Cracow group guided up to recent time by Prof. W. Glöckle (who passed away recently) – employed for such few-nucleon calculations the fastest in Europe supercomputer from JSC in Jülich with the architecture of Blue Gene [5,6].

Quite recently, new methods for solving Faddeev and Faddeev–Yakubovsky few-body scattering equations using (in one way or another) the bases of square-integrable functions have been developed [7], which allow to simplify significantly the numerical solution schemes. Nevertheless, the treatment of realistic three- and four-body scattering problems still includes a huge numerical work and, as a result, can be done only by a few groups over the world that hinders the development of these important studies. However, according to present authors' knowledge, GPU computing still has not been used widely for a solution of few-body scattering problems (we know only two researches based on the few-body GPU computation but they are dedicated to the *ab initio* calculation of bound states [8] and also resonances in the Faddeev-type formalism [9]).

Moreover, in the case when the colliding particles have inner structures and can be excited in the scattering process, i.e. should be treated as composite ones (e.g., the nucleon isobars or nucleon clusters) the numerical complexity of the problem is increased additionally, so that without a significant improvement of the whole numerical scheme the practical solution of such

* Corresponding author.
E-mail addresses: pomeran@nucl-th.sinp.msu.ru (V.N. Pomerantsev), kukulin@nucl-th.sinp.msu.ru (V.I. Kukulin), rubtsova@nucl-th.sinp.msu.ru (O.A. Rubtsova), ssayabek@yandex.kz (S.K. Sakhiev).

http://dx.doi.org/10.1016/j.cpc.2016.03.018
0010-4655/

multichannel problems becomes to be highly nontrivial even for a supercomputer. Therefore, the development of new methods in quantum few-body scattering which can be adapted for highly parallel realization is of interest nowadays.

Recently we have proposed a novel approach in this area which includes two main components:

(i) A complete discretization of the continuous spectrum of the scattering problem, i.e. the replacement of continuous momenta and energies with their discrete counterparts, by projecting all the scattering functions and operators onto a space spanned on the basis of the stationary wave packets [10–13]. As a result, the integral equations of the scattering theory (like the Lippmann–Schwinger, Faddeev, etc. equations) are replaced with their matrix analogs.

(ii) The numerical solution of the resulting matrix equations with wide usage of the multithread computing on GPU.

The main feature of the above discretization procedure is that all the constituents in the equations are represented by finite matrices, all the elements of which are calculated independently. So, this approach is just quite suitable for parallelization and implementation on GPU. All the necessary details of this discretization method the reader can find in Refs. [10–13] while the detailed study of the GPU realization of the approach is done in the present paper.

Here we outline briefly the importance of just massively parallel GPU computing for such specific problems when employing a desktop computer. Let us remind that the Faddeev integral equations for solving the quantum three-body scattering problem with realistic interparticle interactions are the coupled two-dimensional singular integral equations with variable lower and upper limits in the kernel integrals. These many dozens of coupled channels when discretizing by means of wave packets result in huge matrix for kernel, with the dimension ca. 10^6–10^7. Such huge matrix cannot be even stored in RAM. This is a first serious problem. The second one is the high complexity of calculation for every matrix element of this huge matrix which includes evaluation of many thousands double integrals with complicated integrands combined with numerous sums. So, both these difficulties require a wide usage of powerful supercomputers.

However, the straightforward GPU implementation for massively parallel computation of the kernel matrix cannot be efficient because numerous exchanges between RAM and global GPU memory take a lot of time. This slows down all the computations and cancels all advantages due to parallel GPU realization of algorithm. So the GPU realization in this case looks to be highly uneasy and nontrivial. It should be emphasized here that the huge amount of computation leads to a rather long execution time even for powerful supercomputers. Thus the solution of such a problem with usual PC for a reasonable time seems completely unreachable.

The present paper reports about the ways to avoid all the above obstacles and to do really very fast calculation for the Faddeev equations using a desk computer. We illustrate the general discretization algorithm for GPU implementation by an example of calculating the elastic scattering amplitude in three-nucleon system with both simple s-wave and realistic interactions. Also different aspects related to GPU computing will be studied here and runtimes for CPU and GPU mode calculations will be compared.

The paper is organized as follows. The numerical scheme for a practical solution of the scattering problem in a system of three identical particles (fermions) in a discretized representation is described in Section 2. In Section 3 we discuss the properties of GPU realization for the above problem and test some illustrative examples while in Section 4 the results for the elastic scattering in three-nucleon system with realistic NN interaction are presented. The conclusions are given in Section 5. For the reader's convenience we add Appendix with a brief explanation of the wave-packet continuum discretization technique.

2. Discrete analogue for the Faddeev equation for a system of three identical particles in the wave-packet representation

2.1. The Faddeev equation for the transition operator

The elastic scattering observables in a system of three identical particles 1, 2 and 3 can be found from the Faddeev equation for the transition operator U, e.g. in the following form (the so-called Alt–Grassberger–Sandhas form):

$$U = P(G_0)^{-1} + Pt_1G_0U. \tag{1}$$

Here t_1 is the two-particle t-matrix, corresponding to interacting particles 2 and 3, G_0 is the resolvent of the free three-body Hamiltonian H_0, and P is the particle permutation operator explicit form of which depends on the type of particles we study.

After the partial wave expansion in terms of spin–angular functions, the operator equation (1) for each set of the conserved three-body quantum numbers Γ ($\Gamma = \{J, T, \pi\}$, where J is a total angular momentum, T is a total isospin and π is a parity) is reduced to a system of two-dimensional singular integral equations in momentum space. As an example, we give the explicit form of these equations in a form close to the one used in the works of the Bochum–Cracow group [6]. The transition operator in Eq. (1) depends explicitly on four momentum variables (p, q) and (p', q') (here p is the momentum of relative motion for pair {23}, q is the momentum of the third particle 1) and two sets of quantum numbers $\alpha = \{l, s, j, t\}$ and $\beta = \{\lambda, I\}$ where l, s, j, t are the orbital angular momentum, spin, total angular momentum and isospin of {23} pair respectively while λ and I are the orbital and total angular momenta of the third particle. Further we will use the symbol $\gamma = \{\Gamma, \alpha, \beta\}$ to denote the complete set of quantum numbers.

The particle permutation operator P in the integral Faddeev equation (1) is the most difficult challenge for evaluation. In the three-body plane wave basis $\{|p, q; \gamma\rangle\}$ defined for the every set γ, it can be written, e.g., in following form [6]:

$$\begin{aligned}&\langle pq\gamma|P|p'q'\gamma'\rangle\\ &= \int_{-1}^{1} dx \frac{\delta\left(p' - \sqrt{\frac{9q^2}{16} + \frac{p^2}{4} - \frac{3}{4}pqx}\right)\delta\left(q' - \sqrt{p^2 + \frac{q^2}{4} - pqx}\right)}{(p')^2(q')^2}\\ &\quad \times F^{\gamma\gamma'}(p, q, x),\end{aligned} \tag{2}$$

where δ is the Dirac δ-function, x is the cosine of the angle between the vectors $\mathbf{p}$ and $\mathbf{q}$. The functions $F^{\gamma\gamma'}(p, q, x)$ are the algebraic functions of p, q and x:

$$\begin{aligned}F^{\gamma\gamma'}(p, q, x) &= \frac{\sum\limits_{l_1,\lambda_1,k} p^{l_2+\lambda_2} q^{l_1+\lambda_1} P_k(x) g^{l_1\lambda_1 k}_{\gamma\gamma'}}{\left(\frac{9}{16}q^2 + \frac{1}{4}p^2 - \frac{3}{4}pqx\right)^{\frac{l'}{2}} \left(p^2 + \frac{1}{4}q^2 - pqx\right)^{\frac{\lambda'}{2}}},\\ l_1 + l_2 &= l',\\ \lambda_1 + \lambda_2 &= \lambda',\end{aligned} \tag{3}$$

where the intermediate indices l_1, λ_1, l_2, λ_2 arise from the rotation of spherical functions and range over all the possible values in accordance with the triangle rules in algebraic coefficient $g^{l_1\lambda_1 k}_{\gamma\gamma'}$ (see the formulas in [6]).

To find the scattering observables one needs to solve the following equation for the half-shell elastic amplitude matrix

$U^{\gamma}(p,q) = \langle p,q;\gamma|U|\phi\rangle$ where ϕ is the wave function of initial state:

$$U^{\gamma}(p,q) = U_0^{\gamma}(p,q) + \sum_{\gamma'\gamma''}\int_{-1}^{1} dx\, F^{\gamma\gamma'}(p,q,x)\int p''^2 dp'' \times \frac{t_1^{\alpha'\alpha''}\left(\sqrt{\frac{9}{16}q^2+\frac{1}{4}p^2-\frac{3}{4}pqx},\ p'';\ E-\frac{3(p^2+\frac{1}{4}q^2-pqx)}{4m}\right)}{E+i\varepsilon-\frac{p''^2}{m}-\frac{3}{4m}(p^2+\frac{1}{4}q^2-pqx)} U^{\gamma''} \times \left(p'',\sqrt{p^2+\frac{1}{4}q^2-pqx}\right). \quad (4)$$

Here m is a particle mass, $U_0^{\gamma}(p,q)$ is the driving term which we do not write explicitly and $t_1^{\alpha'\alpha''}(p',p'';Z)$ is the kernel of two-body t-matrix which depends on the {23} subsystem quantum numbers α and contains a pole at the bound state energy. The form of Eq. (4) includes additional integration over value x.

The practical solution of the set of coupled equations like (4) is complicated and time-consuming task due to special features of the integral kernel and a large number of coupled spin–angular channels γ which should be taken into account [6]. One can see that the Faddeev kernel at the real total energy E has singularities of two types: two-particle cuts corresponding to bound states in the two-body subsystems (they arise from two-body t-matrix t) and the three-body logarithmic singularity arising at energies above the breakup threshold from denominator of the kernel in (4). While the regularization of the two-body singularities is straightforward and does not pose any problems, the regularization of the three-body singularity requires some special techniques that greatly hamper the solution procedure. The practical tricks which allow to avoid such complications are e.g. a solution of the equation at complex values of energy followed by analytic continuation to the real axis or a shift for the contour of integration from the real axis into the plane of complex momenta.

However, the main specific feature of the Faddeev-like kernel is the presence of the particle permutation operator P, which changes the momentum variables from one Jacobi set to another one. This leads to variable limits in integral terms of the Faddeev equations. In Eq. (4), the variable limits of integration do not appear explicitly due to introduction the variable x. However, the second argument q' of the unknown solution U is a function of the variables x, p and q and can take values in finite limits, depending on the values p and q.

When one replaces as usually the integrals with quadrature sums, all the functions are defined only in quadrature points and in order to calculate the integral in Eq. (4) the numerous interpolations of the unknown solution $U(p,q)$ should be done at every iteration step. This cumbersome interpolation procedure takes most of the computational time and requires using powerful supercomputers. The usage of spline-type interpolations and L_2-type techniques helps to overcome this problem in conventional approaches [7].

The wave-packet discretization method which is used in our approach allows to circumvent completely the above difficulties in solving the Faddeev equations (see [10] and the Appendix).

2.2. The matrix analog of the Faddeev equation and its features

The main idea of the wave-packet (WP) approach is discretization of spectra for subHamiltonians h_0 and h_0^1 defining relative free motion in the subsystem {23} and the free motion of the third particle 1 respectively. For this purpose, the positive range of momenta p and q are divided into intervals $\{[p_{i-1},p_i] \equiv \mathfrak{D}_i\}_{i=1}^{N}$ and $\{[q_{j-1},q_j] \equiv \bar{\mathfrak{D}}_j\}_{j=1}^{\bar{N}}$ and two-body stationary wave-packets $|x_i\rangle$ and $|\bar{x}_j\rangle$ are introduced as integrals of exact two-body free motion wave-functions $|p\rangle$ and $|q\rangle$ over these intervals correspondingly (see Eq. (A.5)). The constructed WP states are L_2 normalized analogs for the exact continuum wave functions and they form a very convenient basis for the scattering theory calculations (see the Appendix and Ref. [10]).

The three-body free WP states needed for solving the Faddeev equation are built as direct products of the respective two-body WP states $|x_i^{\alpha}\rangle$ and $|\bar{x}_j^{\beta}\rangle$ with taking into account spin and angular parts Γ, α, β of the basis functions:

$$|X_{ij}^{\Gamma\alpha\beta}\rangle \equiv |x_i^{\alpha},\bar{x}_j^{\beta};\alpha,\beta:\Gamma\rangle = |x_i^{\alpha}\rangle\otimes|\bar{x}_j^{\beta}\rangle|\alpha,\beta:\Gamma\rangle, \quad \begin{array}{l} i=1,\ldots,N,\\ j=1,\ldots,\bar{N}.\end{array} \quad (5)$$

The state (5) is an L_2 analog of the exact plane wave state in three-body continuum $|p,q;\gamma\rangle$ for the three-body free Hamiltonian H_0. The three-body free WP basis functions (5) are constant inside the rectangular cell $\mathfrak{D}_{ij}$ of the momentum lattice in (p,q)-space built as composition of two one-dimensional cells: $\mathfrak{D}_{ij} = \mathfrak{D}_i\otimes\bar{\mathfrak{D}}_j$. Hence we refer to the free WP basis as *a lattice* basis. Using such a basis one can construct finite-dimensional (discrete) analogs of all the basic scattering operators.

It is also convenient to introduce the basis for the *channel Hamiltonian* $H_1 = H_0 + v_1 = (h_0 + v_1)\oplus h_0^1$ which defines asymptotic motion of the {23} subsystem relatively to the third particle 1. The channel WP basis functions are constructed similarly to the free ones (5) but $|x_i^{\alpha}\rangle$ states should be replaced with two-body *scattering* WPs $|z_k^{\alpha}\rangle$ corresponding to two-body subHamiltonian $h_1 = h_0 + v_1$ which includes the inner interaction in the {23} subsystem (see Eqs. (A.1) and (A.13)). It has been shown in our previous works that $|z_k^{\alpha}\rangle$ can be replaced by pseudostates found by means of a diagonalization procedure for the subHamiltonian h_1 matrix [10] on the basis of lattice-type WP $|x_i^{\alpha}\rangle$ (see Eq. (A.9)). In this way, one gets the basis WP states $|Z_{kj}^{\Gamma\alpha\beta}\rangle$ for the channel Hamiltonian H_1 as a linear combination of the lattice basis states (5) [10]:

$$|Z_{kj}^{\Gamma\alpha\beta}\rangle = \sum_{\alpha',i} O_{ki}^{\alpha\alpha'}|X_{ij}^{\Gamma\alpha'\beta}\rangle, \quad \begin{array}{l} k=1,\ldots,N,\\ j=1,\ldots,\bar{N}\end{array} \quad (6)$$

where the rotation matrix $\mathbb{O}$ consisting of the elements $O_{ki}^{\alpha\alpha'}$ has a block form because it affects only the {23}-subsystem indices k and α. Actually this matrix relates matrix elements of some operator given on the lattice and channel WP bases.

The constructed channel basis (6) can be used to find the transition operator U by using the Faddeev equation (1) in a modified form. Indeed, according to the identity

$$t_1G_0 = v_1G_1, \quad (7)$$

where $G_1 \equiv [E+i0-H_1]^{-1}$ is the resolvent of the channel Hamiltonian H_1, one can rewrite Eq. (1) in a half-shell equivalent form:

$$U = Pv_1 + Pv_1G_1U. \quad (8)$$

This form is especially convenient for solution in the representation of the channel WP states (6) because the three-body channel resolvent $G_1(E)$ which enters the kernel of Eq. (8) has only diagonal matrix elements in the basis (6), i.e.

$$\langle Z_{kj}^{\Gamma\alpha\beta}|G_1(E)|Z_{k'j'}^{\Gamma\alpha'\beta'}\rangle = G_{kj}^{\Gamma\alpha\beta}(E)\delta_{\alpha\alpha'}\delta_{\beta\beta'}\delta_{kk'}\delta_{jj'}, \quad (9)$$

and the complex eigenvalues $G_{kj}^{\Gamma\alpha\beta}(E)$ are defined by simple analytical formulas [10,11].

Projecting the integral equation (8) onto the three-body channel WP basis (6), one gets its matrix analog (for each set of conserved three-body quantum numbers Γ):

$$\mathbb{U} = \mathbb{P}\mathbb{V}_1 + \mathbb{P}\mathbb{V}_1\mathbb{G}_1\mathbb{U}. \quad (10)$$

Here $\mathbb{P}$, $\mathbb{V}_1$ and $\mathbb{G}_1$ are the matrices of the permutation operator, pair interaction and channel resolvent respectively defined in the channel WP basis. It should be emphasized that all the energy singularities of Eq. (8) are collected in the resolvent G_1. After a WP projection, these singularities are smoothed out according to integration over momentum cells, so that, Eq. (10) can be solved directly at real energies.

As was mentioned above, the channel-resolvent matrix $\mathbb{G}_1$ takes a simple diagonal form in the channel WP basis. The matrix element of the interaction $\mathbb{V}_1$ has the block form (the same as for the rotation matrix $\mathbb{O}$):

$$\langle Z_{kj}^{\Gamma\alpha\beta}|V_1|Z_{k'j'}^{\Gamma\alpha'\beta'}\rangle = \langle z_k^{\alpha}|v_1|z_{k'}^{\alpha'}\rangle\delta_{\beta\beta'}\delta_{jj'}, \tag{11}$$

where matrix element of the interaction $\langle z_k^{\alpha}|v_1|z_{k'}^{\alpha'}\rangle$ is taken in the subspace corresponding to {23} subsystem (they are related to the potential matrix elements taken in the lattice WP basis in Eq. (A.8)).

The permutation operator matrix $\mathbb{P}$ in the three-body channel WP basis can be expressed through the matrix $\mathbb{P}^0$ of the same operator in the lattice basis (5) using the rotation matrix $\mathbb{O}$ [10]:

$$\mathbb{P} = \mathbb{O}\mathbb{P}^0\mathbb{O}^{\mathrm{T}}. \tag{12}$$

Finally, the elements of the permutation matrix $\mathbb{P}^0$ in the lattice basis are evaluated by integrating Eq. (2) over two-dimensional lattice cells $\mathfrak{D}_{ij}$ and $\mathfrak{D}_{i'j'}$:

$$[\mathbb{P}^0]_{ij,i'j'}^{\gamma,\gamma'} = \int_{\mathfrak{D}_{ij}} pdpqdq \int_{\mathfrak{D}_{i'j'}} p'dp'q'dq' \frac{\langle pq,\gamma|P|p'q',\gamma'\rangle}{\sqrt{B_{ij}B_{i'j'}}}, \tag{13}$$

where B_{ij} and $B_{i'j'}$ are normalization coefficients. Due to an integration in Eq. (13), the singularities of the permutation operator kernel get averaged over the cells of the momentum lattice and, as a result, the elements of the matrix $\mathbb{P}^0$ in the WP basis become finite. The matrix element (13) is reduced to double integral with variable limits which can be calculated numerically [13]. It is important to note that due to the energy conservation rule the matrix $\mathbb{P}^0$ is highly sparse.

Thus, all the constituents of Eq. (10) have an explicit form in the channel WP basis. Below we show how to solve this equation practically and adapt it to a parallel GPU-realization. As numerical example we will consider scattering in three-nucleon ($3N$) system, particularly, *nd* elastic scattering in which case we can compare our numerical solution obtained with GPU with the benchmark solution found with traditional technique on powerful supercomputer [6].

2.3. Description of the numerical scheme for solution of Faddeev equation in WP approach

In the WP approach, we reduced the solution of integral Faddeev equation (8) to the solution of the very large system of linear algebraic equation (10) (where number of coupled channels is defined by possible values of spin–angular quantum numbers γ) and define the simple procedures and formulas for the calculation of the kernel matrix $\mathbb{K} = \mathbb{P}\mathbb{V}_1\mathbb{G}_1$. By this approach, we avoided the difficulties of solving the integral equation (8), which are met in the conventional approach, but the price which should be paid for this is a high dimension of the resulting system of algebraic equations. The high dimension is the most serious problem in the practical solution of the matrix analogue for the Faddeev equation.

In fact, we found [10] that quite satisfactory results can be obtained with a basis size along one Jacobi momentum $N \sim \bar{N} \sim$ 100–150. It means that even in the simplest one-channel case when all the quantum numbers in the set γ are conserved (e.g. for spin-quartet *s*-wave three-fermion scattering or *s*-wave three-boson scattering) one gets a kernel matrix with dimension $M = N \times \bar{N} \sim$ 10 000–20 000. However, in case of realistic three-body scattering it is necessary to include many spin–angular channels (up to 62 channels in the case of three-nucleon system) and thus the dimension of the kernel matrix increases up to $5 \cdot 10^5$–10^6. The high dimension of the algebraic system leads to the impossibility to place the whole kernel matrix into RAM of an ordinary PC and the impossibility to get the numerical solution for a reasonable time, using a successive CPU execution.

At the first glance, this problem seems to be solved by storing the whole kernel matrix in the external memory. However when using it the iterative process becomes very inefficient, since most of the processing time is spent for reading data from the external memory, while the processor is idle. Nevertheless the specific matrix structure of the kernel in Eq. (10) makes it possible to overcome this difficulty and *to eliminate completely the need for an external memory*. Indeed, the matrix kernel $\mathbb{K}$ for Eq. (10) can be written as a product of four matrices, which have the specific structure:

$$\mathbb{K} = \mathbb{P}\mathbb{V}_1\mathbb{G}_1 \equiv \mathbb{O}\mathbb{P}^0\tilde{\mathbb{V}}_1\mathbb{G}_1, \tag{14}$$

where $\tilde{\mathbb{V}}_1 = \mathbb{O}^{\mathrm{T}}\mathbb{V}_1$. Here $\mathbb{G}_1$ is a diagonal matrix, $\mathbb{P}^0$ is a highly sparse permutation matrix, while $\tilde{\mathbb{V}}_1$ and $\mathbb{O}$ are block matrices of the identical blocks with dimension $(N \times N)$ (see Fig. 1).

Thus, if to store in RAM only the individual multipliers of the matrix kernel $\mathbb{K}$, and to store highly sparse matrix $\mathbb{P}^0$ in a compressed form (i.e. to store only its nonzero elements), all the data required for the iteration process can still be placed in RAM. And although in this case three extra matrix multiplications are added at each iteration step, a computer time spent on iterations is reduced more than 10 times in comparison with the procedure employing an external memory.

Thus, our overall numerical scheme consists of the following main steps:

1. Processing of the input data.
2. Calculation of nonzero elements of the permutation matrix $\mathbb{P}^0$.
3. Calculation of the channel resolvent matrix $\mathbb{G}_1$.
4. Iterations of the matrix equation (10) and finding its solution by making use of the Pade-approximant technique.

The step 1 includes the following procedures:

– construction of two-body free WP bases, and calculating matrices of the interaction potential;

– finding parameters for the three-body channel basis including matrices of the rotation $\mathbb{O}$ between the lattice and scattering WPs;

– calculating algebraic coefficients $g_{\gamma\gamma'}^{l_1\lambda_1 k}$ from Eq. (3) for the recoupling between different spin–angular channels.

We found that the runtimes for the steps 1 and 3 are practically negligible in comparison with the total running time, so that we shall not discuss these steps here. The execution of the step 4 – the solution of the matrix system by iterations – takes about 20% of the total time needed to solve the whole problem in one-thread CPU computing. Therefore, here we did not optimize this step using the GPU.

The main computational efforts (in the one-core CPU realization) are spent on the step 3, viz. the calculation of the elements of the matrix $\mathbb{P}^0$. Because all of these elements are calculated with help of the same code and fully independently from each other, the algorithm seems very suitable for a parallelization and implementation on multiprocessor systems like GPU. However, since the matrix $\mathbb{P}^0$ is highly sparse, it is necessary to use special tricks in order to reach a high acceleration degree in GPU realization. In particular, we apply an additional pre-selection of nonzero elements of the matrix $\mathbb{P}^0$.

It should be stressed here that steps 1 and 2 do not depend on the incident energy. The current energy is taken into account

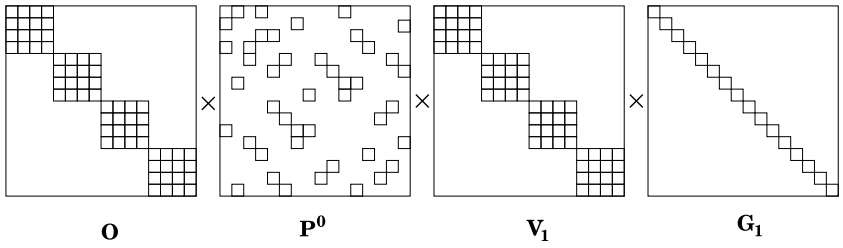

Fig. 1. The structure of matrix kernel (14) for the Faddeev equation (10): nonzero elements are marked by squares.

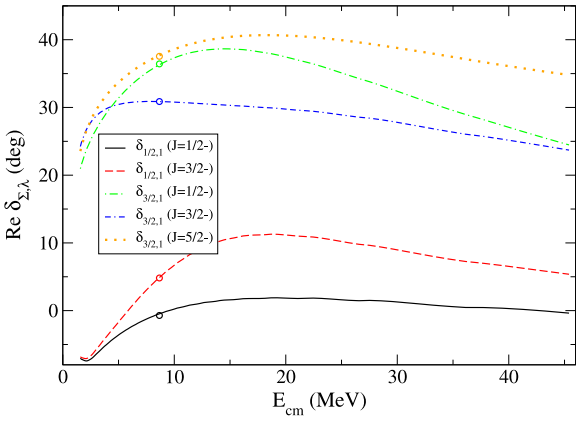

Fig. 2. (Color online) The *p*-wave partial phase shifts for the elastic *nd* scattering obtained within the WP approach (solid curves) and within the standard Faddeev calculations (circles) [6].

only at steps 3 and 4 when one calculates the channel resolvent matrix elements and solves the matrix equation for the scattering amplitude. Thus when one needs to get scattering observables in some wide energy region, the whole computing time will not increase largely because the most time-consuming part of the code (step 2) is carried out only once for many energy points.

We compared in Fig. 2 results of our calculation on PC for the case three-nucleon scattering with the conventional Faddeev calculation of Bochum group [6] who used a high-power supercomputer Blue Gene. In Fig. 2 the *p*-wave partial phase shifts $\delta^{J\pi}_{\Sigma\lambda}$ of the elastic *nd* scattering[1] for realistic the Nijmegen I *NN* potential [14] are shown.

In the next section we consider the specific features related just to GPU realization for the above numerical scheme.

3. GPU acceleration in calculation of kernel matrix elements

As was noted above, the calculation of elements of a large matrix looks very suitable task for effective application of GPU computing if these elements are calculated independently from each other and by one code. However, there are a number of aspects associated with the organization of the data transfer from RAM to the GPU memory and back and also with the GPU computation itself which renders the GPU realization in this case highly nontrivial. These aspects impose severe restrictions on the resulting acceleration in GPU realization. To measure an effectiveness of GPU execution let us introduce the GPU-acceleration degree η as a ratio of runtime for the one-thread CPU computation to runtime for the multithread GPU computation:

$$\eta = t_{\mathrm{CPU}}/t_{\mathrm{GPU}}. \tag{15}$$

This acceleration depends on the ratio of the actual time for the calculation of one matrix element, t_0, to the time of transmitting the result from the GPU memory back to RAM T, on the number of GPU cores N_c and their speed r_{GPU} as compared to speed of CPU core r_{CPU}, and also on the dimension of the matrix M:

$$\eta = f\left(\frac{t_0}{T}, N_c, r_{\mathrm{GPU}}, r_{\mathrm{CPU}}, M\right). \tag{16}$$

Note that the transition itself from a one-thread computing to multithread computing takes some time, so that any parallelization is not effective for matrices with low dimension. When using the GPU, one has to take into account that the speed of GPU cores r_{GPU} is usually much smaller than the CPU speed r_{CPU}. Also for the efficiency of multithread computing it is necessary that calculations in all the threads are finished at approximately the same time. Otherwise a part of threads, each of which occupies a physical core, will be idle for some time. In the case of independent matrix elements, this condition means that the numerical code for one element should not depend on its number, in particular, the code must not contain conditional statements that can change the amount of computation.

When calculating the permutation matrix $\mathbb{P}^0$ in our algorithm, the above condition is not valid: only about 1% of its non-vanishing matrix elements should be really calculated using a double numerical integration, while other 99% of elements are equal to zero and determination of this fact requires only a few arithmetic operations. Therefore, when one fills the whole matrix $\mathbb{P}^0$ (including both zero and nonzero elements) 99% of all threads will be idle, and thus we will not reach any real acceleration. So we have to develop at first a numerical scheme to fill effectively sparse matrices using GPU.

3.1. GPU acceleration in calculating elements of a model sparse matrix

In this subsection to study GPU acceleration for the calculation of elements of a matrix with a large dimension M, we consider two model examples in which the matrix elements are determined by the following explicit formulas:

(a) as a sum of trigonometrical functions:

$$A(i,j) = \sum_{k=1}^{K} \left(\sin^k(u_{ij}) + \cos^k(w_{ij})\right), \quad \text{or} \tag{17}$$

(b) as a sum of numerical integrals:

$$A(i,j) = \sum_{k=1}^{K} \int_{u_{ij}}^{w_{ij}} \left(\sin^k(t) + \cos^k(t)\right) dt. \tag{18}$$

Here u_{ij} and w_{ij} are random numbers from the interval [0, 1] and the parameter K allows to vary the time t_0 for calculation of each element in a wide range. The integrals in Eq. (18) are calculated numerically by the 48-point Gaussian quadrature. Therefore the example (b) with numerical integration is closer to our case of calculating the permutation matrix $\mathbb{P}^0$ in the Faddeev kernel.

[1] Here J, π and Σ are the total angular momentum, parity and total channel spin respectively while λ is the neutron orbital momentum.

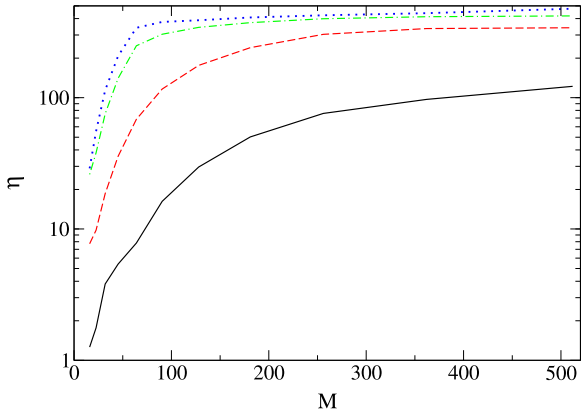

Fig. 3. (Color online) The dependence of GPU acceleration η in calculation of elements of the dense matrix (17) on the matrix dimension M for different values of t_0: 0.0009 ms (solid curve), 0.0094 ms (dashed curve), 0.094 ms (dot-dashed curve), 0.94 ms (dotted curve).

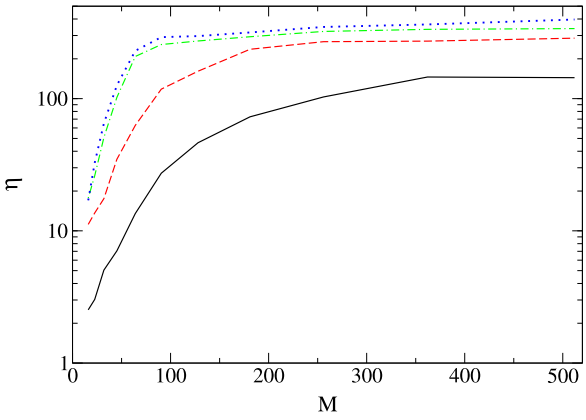

Fig. 4. (Color online) The dependence of GPU acceleration η in calculation of elements of the dense matrix (18) on the matrix dimension M for different values of t_0: 0.0017 ms (solid curve), 0.012 ms (dashed curve), 0.114 ms (dot-dashed curve), 1.13 ms (dotted curve).

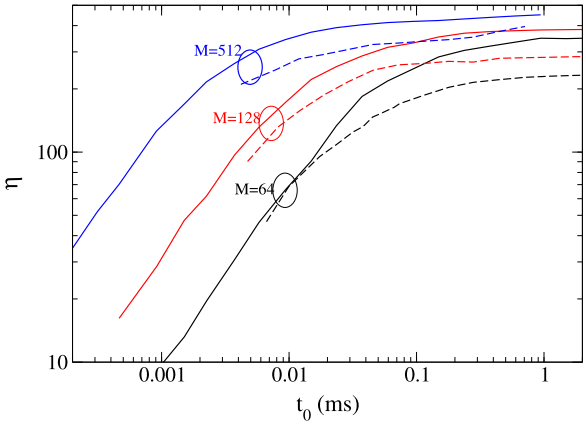

Fig. 5. (Color online) The dependence of GPU acceleration η in calculation of elements of dense matrix on the computational time of each matrix element, t_0, for different values of matrix dimension M: solid curves correspond to calculation of matrix elements using simple trigonometric functions (17), dashed curve — using numerical integrals (18).

Figs. 3–5 show the dependence of the GPU acceleration degree η on the matrix dimension N and the calculation time for each element t_0 when filling up the dense matrices defined by Eqs. (17) and (18). The GPU calculations were performed using M^2 threads, so that, each thread evaluates only one matrix element.

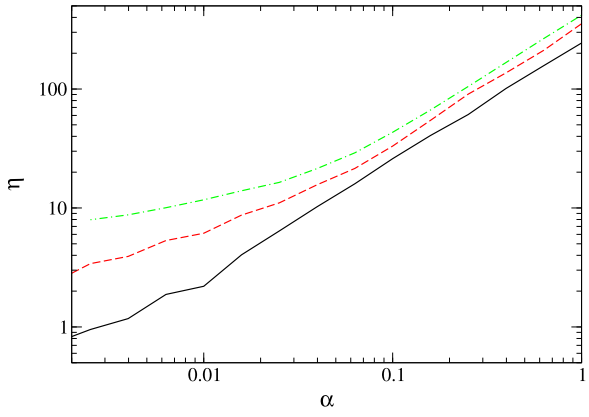

Fig. 6. (Color online) The dependence of GPU acceleration η in calculation of elements of sparse matrix with elements (19) on the sparseness parameter α: for $M = 64$ (dash-dotted curve), $M = 128$ (dashed curve) and $M = 256$ (solid curve).

The calculations are performed on a PC with the processor i7-3770K (3.50 GHz) and the video card NVIDIA GTX-670. We use the Portland Group Fortran compiler 12.10 including CUDA support and CUDA compiler V5.5. As can be seen from the figures, GPU acceleration rises noticeably with increasing the dimension M and the computational time for one matrix element t_0. The maximal acceleration that can be reached with our video card for this simple example is equal to 400–450(!) Such high degree of acceleration is achieved at the matrix dimension $M \sim 200$ and $t_0 \gtrsim 0.1$ ms. At further increase of the dimension M, the degree of acceleration does not change because in this case all the computing resources of the GPU are already exhausted. Note that the GPU acceleration for the example (b) with the numerical integration is somewhat lower than when calculating simple functions. This is due to repeated use of the some constants (the values of the quadrature points and weights) which should be stored in the global GPU memory. It should also be noted that the transition to the double-precision calculation of the matrix elements reduces greatly the maximal possible value of GPU acceleration η.

Consider now what efficiency of GPU computing can be reached in the case of a sparse matrix, when it is actually required to calculate only a small part of matrix elements. We introduce the following additional condition for the matrix elements (17) and (18):

$$\tilde{A}(i,j) = \begin{cases} A(i,j), & u_{ij} \le \alpha \\ 0, & u_{ij} > \alpha. \end{cases} \tag{19}$$

Since u_{ij} is a random number in the interval [0, 1], then one gets a sparse matrix with the degree of sparseness $\sim \alpha$ as a result of such filtration. In fact, the degree of sparseness is the ratio of number of non-zero matrix elements to their total number M^2.

Fig. 6 shows the dependence of the GPU acceleration on the sparseness parameter α in filling up the matrices with dimensions $M = 64$, 128 and 256. As can be seen from the figure, the GPU acceleration is only about 2 (for $M = 64$) at a value of $\alpha \sim 0.01$, which corresponds to the realistic sparseness parameter for the permutation matrix $\mathbb{P}^0$ in the Faddeev kernel.

Thus, to achieve a significant GPU acceleration in calculating the permutation matrix $\mathbb{P}^0$, it is necessary to add one more step to our numerical scheme discussed in Section 2.3 and perform *an initial pre-selection* of nonzero elements of the permutation matrix.

3.2. The GPU algorithm for calculating the permutation matrix for one-channel case

Consider now a calculation of the permutation matrix $\mathbb{P}^0$ entering the kernel for one-channel Faddeev equation (i.e. without spin–angular couplings). There are additional limitations for

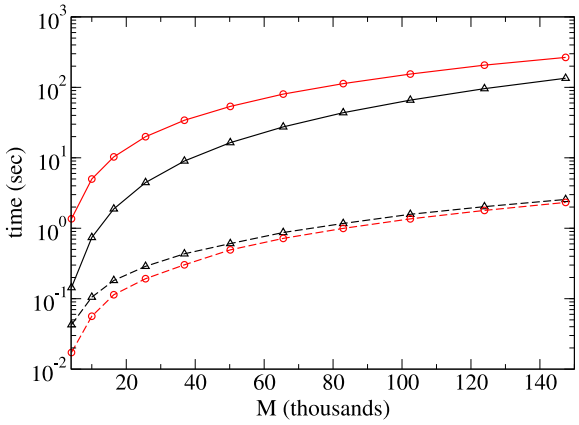

Fig. 7. (Color online) The CPU and GPU computing times (the solid and dashed curves respectively) for pre-selection of the nonzero elements of one-channel permutation matrix $\mathbb{P}^0$ (triangles) and calculation of these elements (circles) depending on the matrix dimension M.

the GPU algorithm for this case compared to simple examples discussed in the previous subsection.

(a) The most serious limitations are a high dimension and also a high sparseness of the permutation matrix, and therefore a special packaging for this matrix is required. Standard packaging for a matrix (we use the packaging on the rows — the so called CSR format) implies, instead of storing the matrix in a single array A with a dimension $M \times M$, the presence of two linear arrays, B and C, with dimensions αM^2, which store the nonzero matrix elements of A and the respective numbers of columns. Also the third linear array W with the dimension M contains addresses of the last nonzero elements (in the array B), corresponding to a given row of the initial matrix A. With such a way of the matrix packaging we get a gain in the memory required for storing the matrix to be equal to $1/(2\alpha)$, i.e. about 50-fold gain for a value of the sparseness 0.01 which is specific for the permutation matrix $\mathbb{P}^0$ in the WP representation. So that, at the specific matrix dimension $M \sim 5 \cdot 10^5$ which is necessary for an accurate calculation of the realistic $3N$ scattering problem, the whole matrix would occupy about 1000 GB of RAM (with single precision), while the same matrix in a compressed form takes about 20 GB RAM only. This is a quite acceptable value for a modern desktop computer.

(b) However, the permutation matrix of such a dimension, even in the packed form, still cannot be placed in the GPU memory which is usually 4–8 GB only. Therefore one needs to subdivide the whole calculation of this matrix into some blocks using an external CPU cycle and then employ the multithread GPU computation for each block.

(c) Another distinction of the calculation of the elements of the matrix $\mathbb{P}^0$ from the simple model example discussed above is the necessity to use a large number of constants: in particular, the mesh points and weights for Gaussian quadratures in a calculation of double integrals and also (in case of a realistic NN interaction with tensor components) algebraic coefficients $g^{l_1\lambda_1 k}_{\gamma\gamma'}$ from Eq. (3) for coupling of different spin–angular channels, values of Legendre polynomials at the nodal points, etc. All these data are stored in the global GPU memory and because of the relatively low access rate of each thread to the global GPU memory, the resulted acceleration is noticeably lower than in the case of the above simple code which does not use a large amount of data from the global GPU memory.

(d) The necessary pre-selection of nonzero elements of the matrix $\mathbb{P}^0$ can be itself quite effectively parallelized with a GPU implementation. Since the runtime for checking the selection criteria for each element is on two orders of magnitude less than the runtime for calculating nonzero element itself, then the degree of GPU acceleration for the stage of a pre-selection turns out less than for the basic calculation. Nevertheless, if we do not employ the GPU at this stage, the computing time for it turns out even larger than the GPU calculation time for all nonzero elements (see below).

After these general observations, we describe the results for the GPU computing of the most tedious step of solving the Faddeev equation in the WP approach – the computation of nonzero elements of the permutation matrix – in a single-channel case. The results attained for a realistic calculation of multichannel nd scattering we leave for the next section.

When the pre-selection of nonzero matrix elements is already done one has the subsidiary arrays C and W containing information about all nonzero elements of $\mathbb{P}^0$ that should be calculated and the number of these nonzero elements is M_t. The parallelization algorithm adopted here assumes that every matrix element is computed by a separate thread. However, the allowable number of threads $N_{\rm thr}$ is restricted by the capacity of the physical GPU memory and is usually less than the total number of nonzero elements M_t. In this case, our algorithm consists of the following steps.

1. The data used in calculation (endpoints of momentum intervals in variables p and q, nodes and weights of Gauss quadratures, algebraic coupling coefficients, etc.) are copied to the GPU memory.

2. The whole set of nonzero elements of the permutation matrix is divided into N_b blocks with $N_{\rm thr}$ elements in each block (except the last one) and the external CPU loop is organized according to the number of such blocks. Inside the loop the following operations are performed:

3. A part of the array C corresponding to the current block is copied to the GPU memory.

4. The CUDA-kernel is launched on GPU in $N_{\rm thr}$ parallel threads each of which calculates only one element (in the case of the one-channel equation) of the permutation matrix.

5. The resulted $N_{\rm thr}$ nonzero elements of the matrix are copied from the GPU memory to the appropriate place of the total array B.

Fig. 7 shows the dependence of the CPU- and GPU-computing time in calculation of the one-channel (s-wave) permutation matrix upon its total dimension $M = N \times \bar{N}$ (for $N = \bar{N}$). In our case, the GPU code was executed in 65536 threads. For the comparison, we display in this figure also the CPU and GPU times which are necessary for a pre-selection of nonzero matrix elements. It is clear from the figure that one needs to use GPU computing not only for the calculation of nonzero elements (that takes most of the time in one-thread CPU computing), but also for the pre-selection of nonzero matrix elements to achieve a high degree of the acceleration.

In Fig. 8, we present the degree of GPU acceleration η for calculating the permutation matrix and also for a complete solution of one-channel Faddeev equation for the s-wave nd scattering problem on the matrix dimension M. It is evident that the runtime for the nonzero elements of the matrix $\mathbb{P}^0$ (which takes the main part of the CPU computing time) is reduced by more than 100 times. The total GPU acceleration in calculating the s-wave partial phase shifts reaches 50. As a result of all these innovations, the total three-body scattering calculation with the matrix dimension $M = 380^2$ takes only about 7 sec on an ordinary PC with GPU.

4. GPU optimization for a realistic 3*N* scattering problem

4.1. GPU-acceleration for multi-channel Faddeev equations

We now turn to the case of three-nucleon scattering problem with the realistic Nijmegen NN potential [14].

Unlike the simple one-channel case discussed above, for realistic interaction we have many coupled spin–angular channels (up to

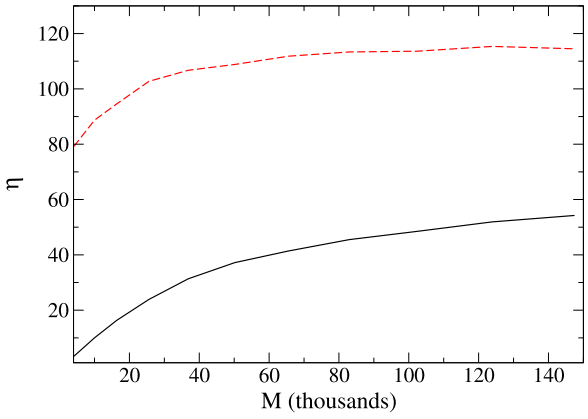

Fig. 8. (Color online) The dependence of the GPU acceleration degree η on the matrix dimension M for calculation of the permutation matrix (dashed curve) and for a complete solution of the *nd* scattering problem (solid curve) in the case of *s*-wave *NN* interaction.

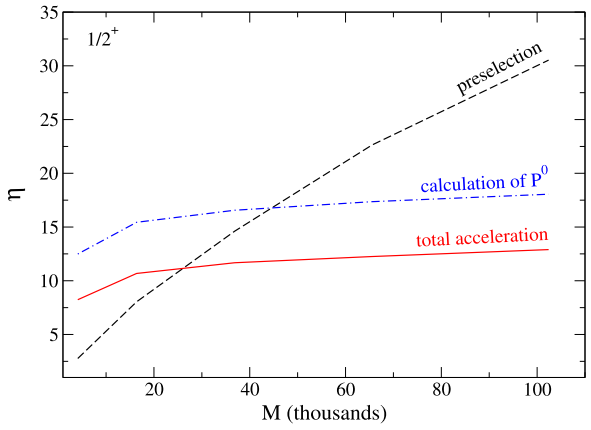

Fig. 9. (Color online) The dependence of the GPU acceleration η on the dimension of the basis $M = N \times \bar{N}$ (for the case $N = \bar{N}$) for the realistic *nd* scattering problem at $J = \frac{1}{2}^+$: dashed line shows the acceleration for the pre-selection of nonzero elements in the permutation matrix $\mathbb{P}^0$, dash-dotted line — for the calculation of these non-zero elements, solid line — the acceleration for the complete solution.

62 channels if the total angular momentum in *NN* pair is restricted as $j \leq 3$). In this case, the calculation of each element of the permutation matrix $\mathbb{P}^0$ comprises the calculation of several tens of double numerical integrals containing the Legendre polynomials. Each matrix element is equal to the sum of such double integrals and the sum includes a large set of algebraic coupling coefficients $g^{l_1\lambda_1 k}_{\gamma\gamma'}$ for the spin–angular channels as in Eq. (3).

Now the GPU-optimized algorithm for the permutation matrix is somewhat different from the single-channel case: because each calculated double integral is used to compute several matrix elements, then each thread now calculates all the matrix elements corresponding to one pair of momentum cells $\{\mathfrak{D}_{ij}, \mathfrak{D}_{i'j'}\}$. These matrix elements belong to different rows of the complete permutation matrix. So that, after the GPU computing for each block of the permutation matrix it is necessary to rearrange and repack (in the single-thread CPU execution) the calculated set of the matrix elements into the arrays B, C and W, representing the complete matrix $\mathbb{P}^0$ in CSR format. All the above moments lead to the fact that the GPU acceleration in calculation of the permutation matrix in the case of realistic nuclear interaction turns out significantly lower than for the single-channel case.

Fig. 9 demonstrates the GPU acceleration η versus the basis dimension $M = N \times \bar{N}$ in the solution of 18-channel Faddeev equation for the partial three-body elastic amplitude with total angular momentum $J = \frac{1}{2}^+$ (solid curve). The dashed and dash-dotted curves show the GPU acceleration for stage of pre-selection of nonzero elements of the permutation matrix $\mathbb{P}^0$ and for calculating of these elements, respectively.

From these results, it is evident that the acceleration degree when calculating the large coupled-channel permutation matrix is about 15 that is considerably less in comparison with the above one-channel case. Nevertheless, the passing from CPU- to GPU-realization *on the same PC* allows to obtain a quite impressive acceleration about 10 in the solution of the 18-channel scattering problem.

In realistic calculation of the observables for elastic three-body scattering, it is necessary to include up to 62 spin–orbital channels. For the current numerical scheme, the efficiency of GPU optimization decreases with increasing number of channels. As an illustration of an accuracy of our approach, we present the results of the complete calculation for realistic elastic *nd* scattering at energy 22.7 MeV. In Fig. 10 we display the differential cross section in comparison with the results obtained in the conventional approach [6] using a supercomputer.

The complete calculation, including up to 62 spin–orbital channels and all states with total angular momentum up to $J_{\max} = 17/2$ took about 30 min on our desk PC. The runtimes for separate steps are given in Table 1.

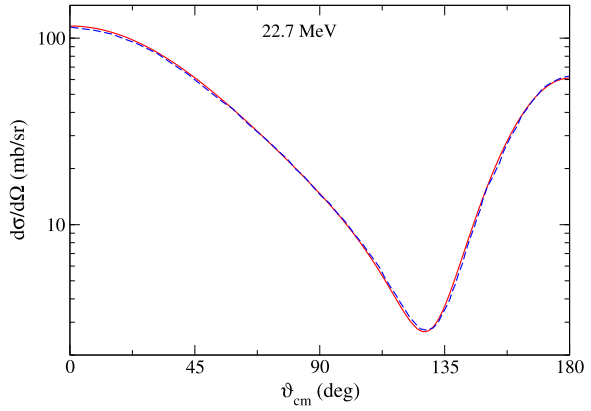

Fig. 10. (Color online) The differential cross section of elastic *nd* scattering at energy 22.7 MeV calculated with realistic *NN* potential in WP formalism with using GPU computing (solid curve) in comparison with the results of the Bochum–Cracow group [6] (dashed curve).

Table 1
Runtime (in s) for separate steps of complete solutions of realistic *nd* scattering problem.

	Step	CPU time	GPU time
1.	Processing input data	30	30
2a.	Pre-selection for $\mathbb{P}^0$	12	1.9
2b.	Calculation of nonzero elements of $\mathbb{P}^0$	4558	524
4.	Iterations and Pade summation	1253	1250
	Total time	5852	1803

As seen from the table, the time of calculation of the permutation matrix elements (steps 2a and 2b) is shortened in ca. 8.7 times due to the GPU optimization. However, for this multichannel Faddeev calculation the major part of computational time is now spent not on calculating the permutation matrix but on the successive iterations of the very large matrix equation, i.e. on multiplication of the kernel matrix by a column representing a current solution. The iteration time takes now about 69% of total solution time. So that, the total acceleration in this multichannel case is only 3.2.

It should be stressed however that the current numerical scheme can be further optimized. Each iteration here includes four matrix multiplications: one multiplication by a diagonal matrix

$\mathbb{G}_1$, two multiplications by block matrices $\mathbb{O}$ and $\tilde{\mathbb{V}}_1$ and one multiplication by sparse matrix $\mathbb{P}^0$, and a multiplication of a sparse matrix by a (dense) vector takes most of the time in the iteration process. It is clear that the algorithm for the iteration can also be parallelized and implemented on the GPU. In this paper, we did not aim to solve this task and focused on the GPU optimization only when calculating the integral kernel of the Faddeev equation. However, for a multiplication of a sparse matrix to a column there are standard procedures, including those implemented on GPU. So that, if to apply the GPU optimization to the iteration step the runtime of complete solution can be reduced further by 2–3 times.

It is also clear that employment of more powerful specialized graphics processors like Tesla K80 would lead even to a considerably greater acceleration of the calculations.

4.2. Further development

It looks to be evident that the described GPU approach will be effective also in the solution of integral equations describing the scattering in systems of four and a larger number of particles (Faddeev–Yakubovsky equations). The main difference between these more complicated problems and the three-body scattering problem considered here is an increase in the number of channels to be included and increase of dimension of integrals defining the kernel matrix elements. As the result, the matrix dimension M and the computational time of each matrix element t_0 will increase. However, a degree of sparseness for the permutation matrices and scheme for calculation of kernel matrix elements will remain the same as in a three-body case. So that, these two factors, i.e. growth of M and t_0, according to our results, will provide even greater GPU acceleration than in a three-body case.

However, when the matrix size M will reach a certain limit, no package will be able to place all nonzero elements in RAM of a computer. In such a case, another strategy should be chosen: one divides the channel space into two parts — the major and minor channels according to their influence on the resulted amplitude. The minor channels would give only a small correction contribution to the solution resulting from the subspace of the major channels. Then, using the convenient projection formalism (such as the known Feshbach formalism), one can account for the minor-channel contribution in a matrix kernel defined in the subspace of the major channels as some additional effective interaction containing the total resolvent in the minor-channel subspace. We have shown previously [10,15] that the basis dimension for the minor channels can be considerably reduced (for a particular problem, it can be reduced in 10 times [15]) without loss in an accuracy of a complete solution.

We hope that such a combined approach together with the multithread GPU computing will lead to the greater progress in the exact numerical solution of quantum few-body scattering problems when using a desktop PC.

5. Conclusion

In the present paper we have checked the applicability of the GPU-computing technique in few-body scattering calculations. For this purpose we have used the wave-packet continuum discretization approach in which a continuous spectrum of the Hamiltonian is approximated by a discrete spectrum of the L_2 normalizable wave-packet states. If to project out all the wave functions and scattering operators onto such a discrete basis we arrive at simple linear matrix equation with non-singular matrix elements instead of the complicated multi-dimensional singular equations in the initial formulation of few-body scattering problem. Moreover, the matrix elements of all the constituents of this equation are calculated independently which make the numerical scheme to be highly parallelized.

The price for this matrix reduction is a high dimension for the matrix kernel. In the case of fully realistic problem the dimension of the kernel matrix turns out so high that such a matrix cannot be placed into RAM of a desktop PC. In addition the calculation of all kernel matrix elements requires a huge computing time in sequential one-thread execution. However, we have developed efficient algorithms of parallelization, which allow to perform basic calculations in the multithread GPU execution and reach a noticeable acceleration of calculations.

It is shown that the acceleration obtained due to GPU-realization depends on the dimension of the basis used and the complexity of the problem. So, in the one-channel three-body scattering problem with a semi-realistic s-wave NN interaction, we obtained as much as 50-fold acceleration for the whole solution while for a separate part of the numerical scheme (most time consuming on CPU) the acceleration achieves more than 100 times. In the multi-channel case corresponding to the fully realistic NN interaction (including up to 62 spin–orbit channels), the acceleration for the permutation matrix calculation is about 8.7 times. A full calculation of the differential cross section is accelerated in this case by 3.2 times. However, the numerical scheme allows a subsequent optimization that will be done in our further investigations. Nevertheless, the present study has shown that the implementation of GPU calculations in few-body scattering problems is very perspective at all and opens new possibilities for a wide area of researches.

It should be stressed, the developed GPU-accelerated discrete approach to solution of quantum scattering problems can be transferred without major changes to other areas of quantum physics, as well as to a number of important areas of classical physics involving solution of multidimensional problems for continuous media studies.

Acknowledgments

This work has been supported partially by Deutsche Forschungsgemeinschaft, grant MU 705/10-1 and the Russian Foundation for Basic Research, Grant No. 16-52-12005.

Appendix. Continuum discretization with stationary wave-packets in few-body scattering problems

Here for the reader's convenience, we outline briefly the method of stationary wave packets. For all details we refer to our previous original papers [11,13] and the recent review [10].

The stationary wave packets (WPs) are introduced for some two-body Hamiltonian $h = h_0 + v$ as integrals of exact scattering wave functions $|\psi_p\rangle$ (normalized to Dirac delta-function on momentum p) over some momentum intervals $\{\Delta_i \equiv [p_{i-1}, p_i]\}_{i=1}^N$:

$$|z_k\rangle = \frac{1}{\sqrt{C_k}} \int_{\Delta_k} w(p)|\psi_p\rangle dp, \quad C_k = \int_{\Delta_k} |w(p)|^2 dp. \tag{A.1}$$

Here $p = \sqrt{2\mu E}$ is relative momenta, μ is the reduced mass of the system, $w(p)$ is a weight function and C_k is the corresponding normalization factor.

The WP states built in this way have a finite normalization as bound states. The set of WP functions together with the possible bound-state wave functions $\{|z_n^b\rangle\}_{n=1}^{N_b}$ of the Hamiltonian h form an orthonormal set and can be employed as a basis similarly to any other L_2 basis functions, which are used to project wave functions and operators [10].

The matrix of Hamiltonian h is diagonal in such a WP basis. The resolvent $g(E) = [E + i0 - h]^{-1}$ for Hamiltonian h has also a diagonal representation in the subspace spanned on the WP basis:

$$g(E) \approx \sum_{n=1}^{N_b} \frac{|z_n\rangle\langle z_n|}{E - \epsilon_n^*} + \sum_{k=N_b+1}^{N} |z_k\rangle g_k(E)\langle z_k|, \tag{A.2}$$

where ϵ_n^* are the bound-state energies and $g_k(E)$ are eigenvalues corresponding to discretized continuum which can be expressed by explicit formulas [10], e.g. for the unit weight $w(p) = 1$:

$$g_k(E) = \frac{\mu}{pd_k}\left\{\ln\left|\frac{p_0 - p_{k-1}}{p_0 - p_k}\right| + \ln\left|\frac{p_0 + p_k}{p_0 + p_{k-1}}\right| - i\pi\theta(p_0 \in \Delta_k)\right\}. \tag{A.3}$$

Here $p_0 = \sqrt{2\mu E}$ is the on-shell momentum, $d_k = p_k - p_{k-1}$ is the width of the momentum interval and θ is the Heaviside-type function defined by the conditions:

$$\theta(p_0 \in \Delta_k) = \begin{cases} 1, & p_0 \in \Delta_k, \\ 0, & p_0 \notin \Delta_k. \end{cases} \tag{A.4}$$

Thus, all the eigenvalues $g_k(E)$ are real, except for the single $g_{k^*}(E)$ corresponding to the interval to which the on-shell momentum value belongs: $p_0 \in \Delta_k*$.

The useful particular case of stationary wave packets is free WP states which are defined for the free-motion Hamiltonian h_0. As in the general case, the continuum of h_0 (in every spin–angular channel α) is divided into non-overlapping intervals $\{\mathfrak{D}_i \equiv [\mathcal{E}_{i-1}, \mathcal{E}_i]\}_{i=1}^{N}$ and two-body free wave-packets are introduced as integrals of exact free-motion wave functions $|p\rangle$ (an index α which marks possible quantum numbers we will omit where is possible):

In particular case of free Hamiltonian h_0 the free WP states are defined as follows:

$$|x_i\rangle = \frac{1}{\sqrt{B_i}}\int_{\mathfrak{D}_i} f(p)|p\rangle dp, \quad B_i = \int_{\mathfrak{D}_i} |f(p)|^2 dp, \tag{A.5}$$

where B_i and $f(p)$ are the normalization factor and weight function respectively.

In such a basis, the free Hamiltonian h_0 has a diagonal finite-dimensional representation as well as the free resolvent $g_0 = [E + i0 - h_0]^{-1}$:

$$g_0(E) \approx \sum_{i=1}^{N} |x_i\rangle g_i(E)\langle x_i|, \tag{A.6}$$

where eigenvalues $g_i(E)$ are defined by Eq. (A.3) (see Ref. [10]). In momentum representation, the states (A.5) take the form of step-like functions:

$$\langle p|x_i\rangle = \frac{f(p)\theta(p \in \mathfrak{D}_i)}{\sqrt{B_i}}. \tag{A.7}$$

In practical calculations, we usually used the free WP with unit weights $f(q) = 1$. The functions of such states are constant inside the chosen momentum intervals. In few-body and multidimensional cases, the WP basis functions are constructed as products of two-body ones, so that the model space can be considered as *a multidimensional lattice*.

The matrix elements of the interaction potential in the free WP representation can be easily calculated using the original momentum representation $v(p, p')$ for the potential:

$$v_{ii'} = \frac{1}{\sqrt{B_i B_{i'}}}\int_{\mathfrak{D}_i}\int_{\mathfrak{D}_{i'}} dpdp' f^*(p) v(p, p') f(p'). \tag{A.8}$$

Moreover, in some rough approximation the potential matrix elements can be found simply as $v_{i,i'} \approx \sqrt{B_i B_{i'}} v(p_i^*, p_{i'}^*)$, where p_i^* and $p_{i'}^*$ are the middle values of momenta over the intervals $\mathfrak{D}_i$ and $\mathfrak{D}_{i'}$ respectively. Further, we will use the above free WP representation for solution of scattering problems.

It was shown [10], that the scattering WPs (A.1) for some total Hamiltonian h can be also approximated in the free WP representation. There is no necessity to find the exact scattering wave functions $|\psi_p\rangle$ in that case. Instead, it is just sufficient to diagonalize the total Hamiltonian matrix in the basis of free WPs. As a result of such direct diagonalization one gets the approximate scattering WPs (and also the functions of bound states if they exist) for Hamiltonian h in the form of expansion into free WP basis:

$$|z_k\rangle \approx \sum_{i=1}^{N} O_{ki}|x_i\rangle, \tag{A.9}$$

where O_{ki} are the matrix elements for rotation from one basis to another. Note that it is not required that the potential v is a short-range one. So that, the same procedure allows to construct wave packets for Hamiltonian including the long-range Coulomb interaction and to get an analytical finite-dimensional representation for the Coulomb resolvent [10].

The Lippmann–Schwinger equation for the transition operator $t(E)$ in momentum representation (for every partial wave l):

$$t_l(p, p'; E) = v_l(p, p') + \frac{1}{4\pi}\int_0^\infty dp'' \frac{v_l(p, p'')t_l(p'', p'; E)}{E + i0 - \frac{(p'')^2}{2m}} \tag{A.10}$$

can be solved in WP representation by projecting equation (A.10) onto free WP basis, the integral equation being reduced to a matrix equation in which all the operators are replaced with their matrices in the WP basis:

$$t_{ii'}^k = v_{ii'} + \sum_{j=1}^{N} v_{ij}[g_0]_j^k t_{ji'}^k, \quad E \in \mathfrak{D}_k \tag{A.11}$$

where v_{ij} are the matrix elements of the interaction operator which are defined by Eq. (A.8). Then the solution of Eq. (A.11) takes the form of histogram representation for the off-shell t-matrix from Eq. (A.10)

$$t_l(p, p'; E) \approx \frac{t_{ii'}^k}{\sqrt{D_i D_{i'}}}, \quad \begin{matrix} p \in \mathfrak{D}_i, \\ p' \in \mathfrak{D}_{i'}, \\ E \in \mathfrak{D}_k, \end{matrix} \tag{A.12}$$

where D_i and $D_{i'}$ are the widths of energy intervals.

The method of continuum discretization described above is directly generalized to the case of three- and few-body system (see the details in Refs. [11,13]). The three-body wave-packets for the *channel Hamiltonian* $H_1 = h_1 \oplus h_0^1$ are defined just as products of two types of wave-packet states for h_1 and h_0^1 subHamiltonians whose spin–angular parts $|\alpha\rangle$ and $|\beta\rangle$ are combined to the respective three-body states having quantum numbers Γ:

$$|Z_{kj}^{\Gamma\alpha\beta}\rangle \equiv |z_k^\alpha, \bar{x}_j^\beta, \alpha, \beta : \Gamma\rangle, \quad \begin{matrix} k = 1, \ldots, N, \\ j = 1, \ldots, \bar{N}. \end{matrix} \tag{A.13}$$

The on-shell elastic amplitude in the WP representation is defined now via the diagonal matrix element of the $\mathbb{U}$-matrix (10) (see details in Ref. [10]):

$$A_{\rm el}^{\Gamma\alpha_0\beta}(q_0) \approx \frac{2m}{3q_0}\frac{[\mathbb{U}]_{1j_0,1j_0}^{\Gamma\alpha_0\beta,\alpha_0\beta}}{\bar{d}_{j_0}}, \tag{A.14}$$

where m is the particle mass, q_0 is the initial two-body momentum and the matrix element is taken between the channel WP states $|Z_{1j_0}^{\Gamma\alpha_0\beta}\rangle = |z_1^{\alpha_0}, \bar{x}_{j_0}^\lambda; \alpha_0, \beta : \Gamma\rangle$ corresponding to the initial and final scattering states. Here $|z_1^{\alpha_0}\rangle$ is the bound state of the pair, the index j_0 denotes the bin $\bar{\mathfrak{D}}_{j_0}$ including the on-shell momentum q_0 and $\bar{d}_{j_0}$ is a momentum width of this bin.

References

[1] B. Block, P. Virnau, T. Preis, Comput. Phys. Comm. 181 (2010) 1549.
[2] M.A. Clark, R. Babich, K. Barrose, R.C. Brower, C. Rebbi, Comput. Phys. Comm. 181 (2010) 1517.
[3] K.A. Wilkinson, P. Sherwood, M.F. Guest, K.J. Naidoo, J. Comput. Chem. 32 (2011) 2313.
[4] https://developer.nvidia.com/cuda-zone.
[5] H. Witała, W. Glöckle, Phys. Rev. C 85 (2012) 064003.
[6] W. Glöckle, H. Witała, D. Hüber, H. Kamada, J. Golack, Phys. Rep. 274 (1996) 107.
[7] J. Carbonell, A. Deltuva, A.C. Fonseca, R. Lazauskas, Prog. Part. Nucl. Phys. 74 (2014) 55.
[8] H. Potter, et al., in: A.M. Shirokov, A.I. Mazur (Eds.), Proceedings of NTSE-2013, Ames, IA, USA, May 13–17, 2013, Khabarovsk, Russia, 2014, p. 263. http://www.ntse-2013.khb.ru/Proc/Sosonkina.pdf.
[9] E. Yarevsky, in: A. Gheorghe, J. Buša, M. Hnatic (Eds.), Mathematical Modeling and Computational Science, in: LNCS, vol. 7125, 2012, p. 290.
[10] O.A. Rubtsova, V.I. Kukulin, V.N. Pomerantsev, Ann. Phys. 360 (2015) 613.
[11] O.A. Rubtsova, V.N. Pomerantsev, V.I. Kukulin, A. Faessler, Phys. Rev. C 86 (2012) 034004.
[12] S.A. Kelvich, V.I. Kukulin, O.A. Rubtsova, Phys. At. 77 (2014) 438.
[13] V.N. Pomerantsev, V.I. Kukulin, O.A. Rubtsova, Phys. Rev. C 89 (2014) 064008.
[14] V.G.J. Stoks, R.A.M. Klomp, C.P.F. Terheggen, J.J de Swart, Phys. Rev. C 49 (1994) 2950.
[15] O.A. Rubtsova, V.I. Kukulin, Phys. At. Nuclei 70 (2007) 2025.