3-1-2021

Transformation of a nucleon-nucleon potential operator into its su(3) tensor form using GPUS

Tomáš Oberhuber
Czech Technical University in Prague

Tomáš Dytrych
Louisiana State University

Kristina D. Launey
Louisiana State University

Daniel Langr
Czech Technical University in Prague

Jerry P. Draayer
Louisiana State University

Recommended Citation

Oberhuber, T., Dytrych, T., Launey, K., Langr, D., & Draayer, J. (2021). Transformation of a nucleon-nucleon potential operator into its su(3) tensor form using GPUS. *Discrete and Continuous Dynamical Systems - Series S, 14* (3), 1111-1122. https://doi.org/10.3934/dcdss.2020383

DISCRETE AND CONTINUOUS
DYNAMICAL SYSTEMS SERIES S
Volume **14**, Number **3**, March **2021**

doi:10.3934/dcdss.2020383

pp. **1111–1122**

TRANSFORMATION OF A NUCLEON-NUCLEON POTENTIAL OPERATOR INTO ITS SU(3) TENSOR FORM USING GPUS

Tomáš Oberhuber*

Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague
Trojanova 13, Praha 2, 120 00, Czech Republic

Tomáš Dytrych

Department of Physics and Astronomy, Louisiana State University
Baton Rouge, LA 70803, USA
Nuclear Physics Institute, Czech Academy of Sciences
Řež 25068, Czech Republic

Kristina D. Launey

Department of Physics and Astronomy, Louisiana State University
Baton Rouge, LA 70803, USA

Daniel Langr

Faculty of Information Technology, Czech Technical University
Prague 16000, Czech Republic
Aerospace Research and Test Establishment
Prague 19905, Czech Republic

Jerry P. Draayer

Department of Physics and Astronomy, Louisiana State University
Baton Rouge, LA 70803, USA

Abstract. Starting from the matrix elements of a nucleon-nucleon potential operator provided in a basis of spherical harmonic oscillator functions, we present an algorithm for expressing a given potential operator in terms of irreducible tensors of the SU(3) and SU(2) groups. Further, we introduce a GPU-based implementation of the latter and investigate its performance compared with a CPU-based version of the same. We find that the CUDA implementation delivers speedups of 2.27x – 5.93x.

1. **Introduction.** A major challenge in computational nuclear physics is to solve the Schrödinger equation for a system of nucleons interacting via realistic nuclear interactions that are consistent with the underlying theory of quantum chromodynamics. Configuration-interaction approaches convert the Schrödinger equation into a matrix eigenvalue problem. A Lanczos algorithm is then applied to find a few of the lowest-lying eigenvalues and eigenstates of the nuclear Hamiltonian matrix, whose elements are expressed in terms of a large many-nucleon basis that span a physically relevant subspace of the Hilbert space, the so-called model space.

2020 *Mathematics Subject Classification.* Primary: 81V35, 81R15; Secondary: 68W10.

Key words and phrases. Nuclear physics, nuclear interaction, irreducible tensors, SU(3) group, SU(2) group, GPU.

* Corresponding author.

The symmetry-adapted no-core shell model [2, 3] (SA-NCSM) is a configuration-interaction approach that employs group-theoretical methods based on the SU(3) and $\mathrm{SU_S}(2)$ symmetry groups (the subscript S denotes spin, see ahead) to reorganize the model space and nuclear interactions with respect to exact and partial symmetries of nuclei. In particular, the symmetry-adapted basis states of the SA-NCSM are classified according to their transformation properties with respect to the following group reduction chain:

$$\begin{array}{ccccccc} \mathrm{SU}(3) & \supset & \begin{array}{c}\mathrm{SO}(3)\\ \times \\ \mathrm{SU_S}(2)\end{array} & \supset & \mathrm{SU}(2) & \supset & U(1) \\ (\lambda\mu) & \kappa & (LS) & & J & & M_J \end{array} \tag{1}$$

Note that the bottom line in (1) displays quantum numbers associated with a given symmetry group. Namely, a generic irreducible representation (irrep) of the SU(3) group is labeled by a pair of non-negative integers $(\lambda\,\mu)$. The total orbital angular momentum L, associated with the SO(3) subgroup of SU(3), is a good quantum number in this scheme. In addition, by coupling the orbital angular momentum L to the complementary spin degree of freedom S, associated with the $\mathrm{SU_S}(2)$ group, one can achieve a classification scheme with good total angular momentum $J = L + S$. The label M_J is the projection of J along the z-axis. The multiplicity label κ reflects the fact that multiple occurrences of L are possible within the $(\lambda\,\mu)$ irrep of SU(3). For a given $(\lambda\,\mu)$ and L (all non-negative integers), the multiplicity index κ runs from 1 to $\mathrm{kmax}(\lambda, \mu, L)$ given by

$$\begin{aligned} \mathrm{kmax}(\lambda,\mu,L) = &\max\left(0, \left[\frac{\lambda+\mu+2-L}{2}\right]\right) - \max\left(0, \left[\frac{\lambda+1-L}{2}\right]\right) \\ &- \max\left(0, \left[\frac{\mu+1-L}{2}\right]\right) \end{aligned} \tag{2}$$

with $[\dots]$ denoting the largest integer.

The essential input for the SA-NCSM calculation, which is unique to this theoretical framework, is a nucleon-nucleon potential operator $\hat{V}$ expanded in terms of SU(3) and $\mathrm{SU_S}(2)$ irreducible tensors. Finding this expansion is a computationally intensive task. The main goal of this article is to introduce a GPU-based implementation of this task.

2. **Expansion formula.** The essential input for the SA-NCSM approach is a nucleon-nucleon potential operator $\hat{V}$ expressed in terms of SU(3) and $\mathrm{SU_S}(2)$ irreducible tensors that are formed from the harmonic oscillator (HO) annihilation and creation operators,

$$\begin{aligned} \hat{V} = &\\ &\frac{1}{4} \sum_{\substack{n_a n_b \\ n_c n_d}} \sum_{(\lambda_f\,\mu_f)\ (\lambda_i\,\mu_i)} \sum_{(\lambda_0\,\mu_0)} \\ &\sum_{\substack{S_f S_i S_0 \\ \kappa_0 \rho_0}} C\left(n_a, n_b, (\lambda_f\,\mu_f), n_c, n_d, (\lambda_i\,\mu_i), (\lambda_0\,\mu_0), S_f, S_i, S_0, \kappa_0, \rho_0\right) \end{aligned}$$

$$\left[\left[a^{\dagger(n_a\,0)}_{\frac{1}{2}} \times a^{\dagger(n_b\,0)}_{\frac{1}{2}}\right]^{(\lambda_f\,\mu_f)}_{S_f} \times \left[\tilde{a}^{(0\,n_d)}_{\frac{1}{2}} \times \tilde{a}^{(0\,n_c)}_{\frac{1}{2}}\right]^{(\lambda_i\,\mu_i)}_{S_i}\right]^{\rho_0(\lambda_0\,\mu_0)}_{\kappa_0(L_0=S_0)J_0=0M_0=0}. \quad (3)$$

Here $a^{\dagger(n\,0)}_{\frac{1}{2}}$ represents an irreducible tensor labeled by $(n\,0)$ and $\frac{1}{2}$ tensor characters with respect to the SU(3) and $\mathrm{SU_S}(2)$ groups. Components of this tensor are fermion creation operators acting in the HO shell n. Similarly, $\tilde{a}^{(0\,n)}_{\frac{1}{2}}$ denotes an irreducible tensor whose components are related to fermion annihilation operators in HO shell n [5]. The symbol $\times$ signifies SU(3) and SU(2) tensorial coupling. The square brackets $[\ldots]$ denote SU(3) and SU(2) irreducible tensor operators [5]. The irreducible tensor operators are labeled by $(\lambda_f\,\mu_f)$, $(\lambda_i\,\mu_i)$, and $(\lambda_0\,\mu_0)$ quantum numbers given by the computationally inexpensive SU(3) coupling rules [6], which determine the decomposition of the Kronecker product of two SU(3) irreps in a direct sum of SU(3) irreps,

$$(n_a\,0)\times(n_b\,0) = \sum_{\oplus}(\lambda_f\,\mu_f) \quad (4)$$

$$(0\,n_d)\times(0\,n_c) = \sum_{\oplus}(\lambda_i\,\mu_i) \quad (5)$$

$$(\lambda_f\,\mu_f)\times(\lambda_i\,\mu_i) = \sum_{\oplus}(\lambda_0\,\mu_0). \quad (6)$$

The symbol ρ_0 in (3) denotes a multiplicity label needed to distinguish between multiple occurrences of a $(\lambda_0\,\mu_0)$ irrep in the Kronecker product $(\lambda_f\,\mu_f)\times(\lambda_i\,\mu_i)$. Irreps $(\lambda_i\,\mu_i)$ and $(\lambda_f\,\mu_f)$ in the Kronecker products (4) and (5) are always unique and no labels ρ are needed. The irreducible tensors in the expansion (3) are also labeled by the spin quantum numbers S_f, S_i, S_0. For a two-body operator, they can take the following values $(S_f, S_i, S_0) \in \{(0,0,0)\,(0,1,1)\,(1,0,1)\,(1,1,0)\,(1,1,1)\,(1,1,2)\}$.

Each term in the expansion (3) contains an irreducible tensor component multiplied by its strength (a real number) that can be computed as

$$\begin{aligned}
&C\,(n_a, n_b, (\lambda_f\,\mu_f), n_c, n_d, (\lambda_i\,\mu_i), (\lambda_0\,\mu_0), S_f, S_i, S_0, \kappa_0, \rho_0)\\
&= (-1)^{S_f+S_0}\sqrt{(2S_i+1)(2S_f+1)(2S_0+1)}\\
&\quad\sum_{\substack{l_a j_a l_b j_b\\ l_d j_d l_c j_c\\ J}} \langle n_a l_a j_a, n_b l_b j_b; J\|\hat{V}\|n_c l_c j_c, n_d l_d j_d; J\rangle\\
&(-1)^{n_d+n_c-j_d-j_c}\sqrt{2J+1}\sqrt{\frac{(2j_a+1)(2j_b+1)}{(\delta_{n_a n_b}\delta_{j_a j_b}\delta_{l_a l_b}\delta_{t_a t_b}+1)}}\sqrt{\frac{(2j_c+1)(2j_d+1)}{(\delta_{n_c n_d}\delta_{j_c j_d}\delta_{l_c l_d}\delta_{t_c t_d}+1)}}\\
&\sum_{L_i L_f}(-1)^{L_i}\sqrt{(2L_i+1)(2L_f+1)}\begin{Bmatrix} l_a & l_b & L_f\\ \frac{1}{2} & \frac{1}{2} & S_f\\ j_a & j_b & J\end{Bmatrix}\begin{Bmatrix} l_d & l_c & L_i\\ \frac{1}{2} & \frac{1}{2} & S_i\\ j_d & j_c & J\end{Bmatrix}\begin{Bmatrix} L_f & L_i & S_0\\ S_i & S_f & J\end{Bmatrix}\\
&\sum_{\kappa_i\kappa_f}\langle(n_a\,0)l_a;(n_b\,0)l_b\|(\lambda_f\,\mu_f)\kappa_f L_f\rangle\langle(0\,n_d)l_d;(0\,n_c)l_c\|(\lambda_i\,\mu_i)\kappa_i L_i\rangle\\
&\langle(\lambda_f\,\mu_f)\kappa_f L_f;(\lambda_i\,\mu_i)\kappa_i L_i\|(\lambda_0\,\mu_0)\kappa_0 S_0\rangle_{\rho_0}.
\end{aligned} \quad (7)$$

It is important to note that the input data for this calculation are provided in form of the matrix elements (real numbers) of a nuclear potential operator $\hat{V}$ in two-nucleon J-coupled HO basis, $\langle n_a l_a j_a, n_b l_b j_b; J \| \hat{V} \| n_c l_c j_c, n_d l_d j_d; J \rangle$. The Kronecker delta $\delta_{t_a t_b}$ and $\delta_{t_c t_d}$ are equal to one for proton-proton and neutron-neutron matrix elements, and vanish for proton-neutron matrix elements. The factors in brackets are the Wigner $9j$ and $6j$ coefficients (real numbers). In the CPU version of the code, we adopt the `WIGXJPF` library [4] for their accurate evaluation. The double-barred factors, denoted by symbol $\langle -; - \| - \rangle_-$, are SU(3)⊃SO(3) Wigner coefficients (real numbers) that are computed by optimized numerical subroutines [1]. Computing coupling coefficients on-the-fly is computationally expensive. To improve the algorithm performance, we store coefficients in fast lookup data structures (`map` from STL library).

3. **Algorithm.** As indicated by the summation order in relation (3), we iterate over combinations of HO shell numbers, n_a, n_b, n_c, n_d (line 2 in Algorithm 1). For each combination, we apply SU(3) coupling rules to find direct sum decompositions $\sum_{\oplus}(\lambda_f\, \mu_f)$ and $\sum_{\oplus}(\lambda_i\, \mu_i)$ (lines 3 and 4). Next, we iterate over SU(3) irreps in each decomposition (line 5 and line 7) and obtain the required SU(3) Wigner coefficients $W^{f}_{(n_a,n_b,\lambda_f,\mu_f)}$ and $W^{i}_{(n_d,n_c,\lambda_i,\mu_i)}$ (lines 6 and 8)

$$\langle (n_a\, 0) l_a; (n_b\, 0) l_b \| (\lambda_f\, \mu_f) \kappa_f\, L_f \rangle \tag{8}$$

$$\langle (0\, n_d) l_d; (0\, n_c) l_c \| (\lambda_i\, \mu_i) \kappa_i\, L_i \rangle. \tag{9}$$

Consequently, we apply SU(3) coupling rules for the Kronecker product $(\lambda_f\, \mu_f) \times (\lambda_i\, \mu_i)$ to obtain a direct sum decomposition $\sum_{\oplus}(\lambda_0\, \mu_0)$ (line 9). For each irrep $(\lambda_0\, \mu_0)$, we calculate the multiplicity $\rho_0^{\max}$ (line 11) and obtain SU(3) Wigner coefficients $W^{0}_{(\lambda_f,\mu_f,\lambda_i,\mu_i,\lambda_0,\mu_0)}$ (line 12)

$$\langle (\lambda_f\, \mu_f) \kappa_f\, L_f; (\lambda_i\, \mu_i) \kappa_i\, L_i \| (\lambda_0\, \mu_0) \kappa_0\, L_0 \rangle_{\rho_0}. \tag{10}$$

Algorithm 1: Determination of irreducible tensors

```
1  transformInteractionToSU3Tensors(inputData)
2  for each (n_a, n_b, n_c, n_d) ∈ inputData do
3      (n_a 0) × (n_b 0) → Σ_⊕(λ_f μ_f)
4      (0 n_d) × (0 n_c) → Σ_⊕(λ_i μ_i)
5      for each (λ_f μ_f) ∈ Σ_⊕(λ_f μ_f) do
6          obtain⟨(n_a 0)l_a; (n_b 0)l_b ‖ (λ_f μ_f)κ_f L_f⟩ ≡ W^f_(n_a,n_b,λ_f,μ_f)
7          for each (λ_i μ_i) ∈ Σ_⊕(λ_i μ_i) do
8              obtain ⟨(0 n_d)l_d; (0 n_c)l_c ‖ (λ_i μ_i)κ_i L_i⟩ ≡ W^i_(n_d,n_c,λ_i,μ_i)
9              (λ_f μ_f) × (λ_i μ_i) → Σ_⊕(λ_0 μ_0)
10             for each (λ_0 μ_0) ∈ Σ_⊕(λ_0 μ_0) do
11                 compute ρ_0^max for SU(3) coupling (λ_f μ_f) × (λ_i μ_i) → (λ_0 μ_0)
12                 obtain
                     ⟨(λ_f μ_f)κ_f L_f; (λ_i μ_i)κ_i L_i ‖ (λ_0 μ_0)κ_0 L_0⟩_ρ0 ≡ W^0_(λ_f,μ_f,λ_i,μ_i,λ_0,μ_0)
13                 C(n_a, n_b, (λ_f μ_f), n_c, n_d, (λ_i μ_i), (λ_0 μ_0), ...) ←
                     tensorStrengthCoeffs(n_a, n_b, n_c, n_d, inputData, W^f_(.), W^i_(.), W^0_(.), ρ_0^max)
14             end
15         end
16     end
17 end
```

The SU(3) Wigner coefficients are utilized in the last step (line 13), where tensor strengths

$$C\left(n_a, n_b, (\lambda_f\, \mu_f), n_c, n_d, (\lambda_i\, \mu_i), (\lambda_0\, \mu_0), S_f, S_i, S_0, \kappa_0, \rho_0\right)$$

are computed using relation (7) for all six allowed spin combinations, S_f, S_i, S_0, as well as multiplicities κ_0 and ρ_0. For details see the Algorithm 2. Note that the first sum in (7) corresponds to the iteration over all two-body matrix elements with the given combination n_a, n_b, n_c, n_d of HO shell numbers (see line 3 in the Algorithm 2).

Algorithm 2: Computation of the tensor strength coefficients.

```
1  tensorStrengthCoeffs(n_a, n_b, n_c, n_d, inputData, W_f, W_i, W_0, ρ_0^max)
2  results ← 0
3  for each V^{n_a n_b n_c n_d}_{l_a j_a l_b j_b l_c j_c l_d j_d J} ∈ inputData(n_a, n_b, n_c, n_d) do
4      for each (S_f, S_i, S_0) do
5          κ_0^max ← kmax(λ_0, μ_0, S_0)
6          for each L_f = |l_a − l_b| ... l_a + l_b do
7              C_f = { l_a  l_b  L_f ; 1/2  1/2  S_f ; j_a  j_b  J }
8              κ_f^max ← kmax(λ_f, μ_f, L_f)
9              for each L_i = |l_c − l_d| ... l_c + l_d do
10                 C_i = { l_d  l_c  L_i ; 1/2  1/2  S_i ; j_d  j_c  J } { L_f  L_i  S_0 ; S_i  S_f  J }
11                 κ_i^max ← kmax(λ_i, μ_i, L_i)
12                 for each κ_0 = 1, ..., κ_0^max do
13                     for each ρ_0 = 1, ..., ρ_0^max do
14                         results(S_f, S_i, S_0, κ_0, ρ_0) += results(S_f, S_i, S_0, κ_0, ρ_0) +
                           V^{n_a n_b n_c n_d}_{l_a j_a l_b j_b l_c j_c l_d j_d J} C_f C_i
                           (−1)^{S_f + S_0 + L_i} sqrt((2S_i + 1)(2S_f + 1)(2S_0 + 1)(2L_i + 1)(2L_f + 1))
                           Σ_{κ_i=1}^{κ_i^max} Σ_{κ_f=1}^{κ_f^max} W^f_{(n_a, n_b, λ_f, μ_f)}(l_a, l_b, κ_f, L_f) W^i_{(n_d, n_c, λ_i, μ_i)}(l_d, l_c, κ_i, L_i)
                           W^0_{(λ_f, μ_f, λ_i, μ_i, λ_0, μ_0)}(κ_f, L_f, κ_i, L_i, κ_0, S_0, ρ_0)
15                     end
16                 end
17             end
18         end
19     end
20 end
```

4. **Implementation on GPUs.** As there is currently no GPU-based code for calculating the required SU(3) Wigner coefficients, these and all additional input required by Algorithm 2 are determined using a CPU-based architecture.

Our first attempt to perform the Algorithm 2 on GPU was only parallelization of the loop over matrix elements (line 3 in Algorithm 2). In this approach, each CUDA thread stores its own private instance for resulting data and the parallel reduction across all threads was performed at the end of the calculation.

Unfortunately, such an approach does not provide enough parallelism to sustain a reasonable occupancy of the GPU. To overcome this we process all irreducible tensors at the same time. For this purpose, we first generate for given HO shell numbers n_a, n_b, n_c, n_d a complete set of all possible SU(3) quantum numbers of irreducible tensors, denoted as $\{(\lambda_f \mu_f)(\lambda_i \mu_i)(\lambda_0 \mu_0)\}$, we also compute the required SU(3) coupling coefficients and store both in a buffer that is optimized for efficient transfer of data from CPU to GPU. In the next subsection we describe it in details.

4.1. **Irreducible tensors triplets buffers.** We preallocate arrays long enough on both CPU and GPU. During the computations on the CPU we treat the array on CPU like stacks while keeping pointers to particular data records. The data layout is depicted on Figure 1.

Record Idx.	Wigner coefficients	Irreps
1	$W^f(n_a, n_b, \lambda_f^1, \mu_f^1)$	(λ_f^1, μ_f^1)
2	$W^i(n_c, n_d, \lambda_i^1, \mu_i^1)$	(λ_i^1, μ_i^1)
3	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^1, \mu_i^1, \lambda_0^1, \mu_0^1)$	(λ_0^1, μ_0^1)
4	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^1, \mu_i^1, \lambda_0^2, \mu_0^2)$	(λ_0^2, μ_0^2)
$\vdots$	$\vdots$	$\vdots$
i_1	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^1, \mu_i^1, \lambda_0^{N_0}, \mu_0^{N_0})$	$(\lambda_0^{N_0}, \mu_0^{N_0})$
i_1+1	$W^i(n_c, n_d, \lambda_i^2, \mu_i^2)$	(λ_i^2, μ_i^2)
i_1+2	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^2, \mu_i^2, \lambda_0^1, \mu_0^1)$	(λ_0^1, μ_0^1)
i_1+3	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^2, \mu_i^2, \lambda_0^2, \mu_0^2)$	(λ_0^2, μ_0^2)
$\vdots$	$\vdots$	$\vdots$
i_2	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^2, \mu_i^2, \lambda_0^{N_0}, \mu_0^{N_0})$	$(\lambda_0^{N_0}, \mu_0^{N_0})$
$\vdots$	$\vdots$	$\vdots$
i_3	$W^i(n_c, n_d, \lambda_i^{N_i}, \mu_i^{N_i})$	$(\lambda_i^{N_i}, \mu_i^{N_i})$
i_3+1	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^{N_i}, \mu_i^{N_i}, \lambda_0^1, \mu_0^1)$	(λ_0^1, μ_0^1)
i_3+2	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^{N_i}, \mu_i^{N_i}, \lambda_0^2, \mu_0^2)$	(λ_0^2, μ_0^2)
$\vdots$	$\vdots$	$\vdots$
i_4	$W^0(\lambda_f^1, \mu_f^1, \lambda_i^{N_i}, \mu_i^{N_i}, \lambda_0^{N_0}, \mu_0^{N_0})$	$(\lambda_0^{N_0}, \mu_0^{N_0})$
i_4+1	$W^f(n_a, n_b, \lambda_f^2, \mu_f^2)$	(λ_f^2, μ_f^2)
i_4+2	$W^i(n_c, n_d, \lambda_i^1, \mu_i^1)$	(λ_i^1, μ_i^1)
i_4+3	$W^0(\lambda_f^2, \mu_f^2, \lambda_i^1, \mu_i^1, \lambda_0^1, \mu_0^1)$	(λ_0^1, μ_0^1)
$\vdots$	$\vdots$	$\vdots$
i_{max}	$W^0(\lambda_f^{N_f}, \mu_f^{N_f}, \lambda_i^{N_i}, \mu_i^{N_i}, \lambda_0^{N_0}, \mu_0^{N_0})$	$(\lambda_0^{N_0}, \mu_0^{N_0})$

FIGURE 1. Data layout of buffer used for transfer of the Wigner coupling coefficients and quantum numbers of irreducible tensors computed on CPU to GPU.

Superscripts N_f, N_i and N_0 denote number of irreps $\sum_{\oplus}(\lambda_f\, \mu_f)$, $\sum_{\oplus}(\lambda_i\, \mu_i)$ and $\sum_{\oplus}(\lambda_0\, \mu_0)$ respectively. Each record has different size which is stored within the

records. When all necessary quantum numbers and Wigner coefficients are computed the buffer is transferred on the GPU. This way, all data representing all irreducible tensors are transferred on GPU at once and not one-by-one. The buffered modification of the Algorithm 1 is Algorithm 3.

Algorithm 3: Buffered determination of irreducible tensors

1 `transformInteractionToSU3TensorsWithBuffer(`*inputData*`)`
2 **for** *each* $(n_a, n_b, n_c, n_d) \in inputData$ **do**
3 allocate buffer B_W for Wigner coefficients and quantum numbers
4 allocate buffer B_{ref} for tuples of references into B_W
5 $recordIdx := 1$
6 $(n_a\, 0) \times (n_b\, 0) \to \sum_{\oplus}(\lambda_f\, \mu_f)$
7 $(0\, n_d) \times (0\, n_c) \to \sum_{\oplus}(\lambda_i\, \mu_i)$
8 **for** *each* $(\lambda_f\, \mu_f) \in \sum_{\oplus}(\lambda_f\, \mu_f)$ **do**
9 obtain$\langle (n_a\, 0) l_a; (n_b\, 0) l_b \| (\lambda_f\, \mu_f) \kappa_f\, L_f \rangle \equiv W^f_{(n_a, n_b, \lambda_f, \mu_f)}$
10 $B_W \leftarrow \left((\lambda_f, \mu_f), W^f_{(n_a, n_b, \lambda_f, \mu_f)} \right)$
11 $W^f_{idx} := recordIdx$
12 $recordIdx++$
13 **for** *each* $(\lambda_i\, \mu_i) \in \sum_{\oplus}(\lambda_i\, \mu_i)$ **do**
14 obtain $\langle (0\, n_d) l_d; (0\, n_c) l_c \| (\lambda_i\, \mu_i) \kappa_i\, L_i \rangle \equiv W^i_{(n_d, n_c, \lambda_i, \mu_i)}$
15 $B_W \leftarrow \left((\lambda_i, \mu_i), W^i_{(n_a, n_b, \lambda_i, \mu_i)} \right)$
16 $W^i_{idx} := recordIdx$
17 $recordIdx++$
18 $(\lambda_f\, \mu_f) \times (\lambda_i\, \mu_i) \to \sum_{\oplus}(\lambda_0\, \mu_0)$
19 **for** *each* $(\lambda_0\, \mu_0) \in \sum_{\oplus}(\lambda_0\, \mu_0)$ **do**
20 compute $\rho_0^{\max}$ for SU(3) coupling $(\lambda_f\, \mu_f) \times (\lambda_i\, \mu_i) \to (\lambda_0\, \mu_0)$
21 obtain $\langle (\lambda_f\, \mu_f) \kappa_f\, L_f; (\lambda_i\, \mu_i) \kappa_i\, L_i \| (\lambda_0\, \mu_0) \kappa_0\, L_0 \rangle_{\rho_0} \equiv$ $W^0_{(\lambda_f, \mu_f, \lambda_i, \mu_i, \lambda_0, \mu_0)}$
22 $B_W \leftarrow \left((\lambda_0\, \mu_0), W^0_{(n_a, n_b, \lambda_0, \mu_0)} \right)$
23 $W^0_{idx} := recordIdx$
24 $recordIdx++$
25 $B_{ref} \leftarrow (W^f_{idx}, W^i_{idx}, W^0_{idx}, \rho_0^{max})$
26 **end**
27 **end**
28 **end**
29 transfer B_W and B_{ref} on GPU
30 `tensorStrengthCoeffsWithBuffer`$(n_a, n_b, n_c, n_d, inputData, B_W, B_{ref})$
31 **end**

Note that on the line 25 we create triplet of references (and ρ_0^{max}) to records holding appropriate combination

$$\left((\lambda_f, \mu_f), W^f_{(n_a, n_b, \lambda_f, \mu_f)}; (\lambda_i, \mu_i), W^f_{(n_a, n_b, \lambda_i, \mu_i)}; (\lambda_0, \mu_0), W^0_{(n_a, n_b, \lambda_0, \mu_0)}, \rho_0^{max} \right)$$

which we insert into a buffer B_{ref}. Each such triplet represents one irreducible tensor as depicted on Figure 2 (here we omit ρ_0^{max}). This buffer is transferred on GPU as well. The size of results $results(S_f, S_i, S_0, \kappa_0, \rho_0)$ computed on the line 14

Record Idx.	Irreducible tensors references triplets
1	$(1, 2, 3)$
2	$(1, 2, 4)$
$\vdots$	$\vdots$
j_1	$(1, 2, i_1)$
$j_1 + 1$	$(1, i_1 + 1, i_1 + 2)$
$j_1 + 2$	$(1, i_1 + 1, i_1 + 3)$
$\vdots$	$\vdots$
j_2	$(1, i_1 + 1, i_2)$
$\vdots$	$\vdots$
j_3	$(1, i_3, i_3 + 1)$
$j_3 + 1$	$(1, i_3, i_3 + 2)$
$\vdots$	$\vdots$
j_4	$(1, i_3, i_4)$
$j_4 + 1$	$(i_4 + 1, i_4 + 2, i_4 + 3)$
$\vdots$	$\vdots$
j_{max}	$(i_4 + 1, i_4 + 2, i_{max})$

FIGURE 2. Data layout of buffer holding records with triplets of references to records in buffer B_W. Each such triplet thus represents one irreducible tensor.

in Algorithm 2 can change for different $(\lambda_0\mu_0)$ since it depends on κ_0^{max} and ρ_0^{max}. Therefore the results are managed in a similar way as B_W. We use preallocated array B_R and for each tensor given by $\{(\lambda_f\mu_f)(\lambda_i\mu_i)(\lambda_0\mu_0)\}$ we store related κ_0, ρ_0 a reference to B_R. Different CUDA blocks processing the same irreducible tensor have their own copies which are reduced on CPU at the end.

4.2. **Tensor strengths computation.** The computation of the tensor strengths is then performed on GPU using Algorithm 4. In this algorithm, each CUDA thread (of one dimensional CUDA blocks) takes one matrix element (of given combination n_a, n_b, n_c, n_d of HO shell numbers, line 12) and one irreducible tensor (lines 13, 15, 19 and 23) and it performs summation over all combinations of quantum numbers $\{(\lambda_f\mu_f)(\lambda_i\mu_i)(\lambda_0\mu_0)\}$ (lines 25–29). These combinations together with related Wigner coefficients are fetched from the buffer B_W (lines 15, 19 and 23) via B_{ref} (line 13). This leads to a much better GPU utilization as the number of CUDA threads needed is equal to the size of the $\{(\lambda_f\mu_f)(\lambda_i\mu_i)(\lambda_0\mu_0)\}$ set multiplied by the number of two-body matrix elements for given values n_a, n_b, n_c, n_d of HO shells. Finaly, on the line 34, we perform parallel reduction inside CUDA blocks to sum-up contributions of all matrix elements mapped to given CUDA block. Intermediate results from particular CUDA blocks are at the end transferred on the host system to be summed-up over all matrix elements on CPU.

Algorithm 4: GPU computation of the tensor strength coefficients.

```
1  tensorStrengthCoeffsWithBuffer(n_a, n_b, n_c, n_d, inputData, B_W, B_ref)
2  results ← 0, matrixElements = |inputData(n_a, n_b, n_c, n_d)|
3  tensors = |B_ref|, matrixElementsBlocks = ⌈matrixElements/blockSize⌉
4  blocks = matrixElementsBlock × tensors
5  create blocksCount blocks of CUDA threads
6  for each CUDA thread t do
7      tensorIdx = blockIdx/matrixElementsBlocks
8      matrixElementsBlockIdx = blockIdx % matrixElementsBlocks
9      matrixElementIdx = matrixElementsBlockIdx × blockDim + threadIdx
10     V^{n_a n_b n_c n_d}_{l_a j_a l_b j_b l_c j_c l_d j_d J} ← inputData(n_a, n_b, n_c, n_d)[matrixElementIdx]
11     (W^f_idx, W^i_idx, W^0_idx, ρ_0^max) ← B_ref[tensorIdx]
12     for each (S_f, S_i, S_0) do
13         ((λ_0 μ_0), W^0_(n_a,n_b,λ_f,μ_f)) ← B_W[W^f_idx]
14         κ_0^max ← kmax(λ_0, μ_0, S_0)
15         for each L_f = |l_a − l_b| ... l_a + l_b do
16             C_f = { l_a  l_b  L_f ; 1/2  1/2  S_f ; j_a  j_b  J }
17             ((λ_f μ_f), W^f_(n_a,n_b,λ_f,μ_f)) ← B_W[W^f_idx]
18             κ_f^max ← kmax(λ_f, μ_f, L_f)
19             for each L_i = |l_c − l_d| ... l_c + l_d do
20                 C_i = { l_d  l_c  L_i ; 1/2  1/2  S_i ; j_d  j_c  J } { L_f  L_i  S_0 ; S_i  S_f  J }
21                 ((λ_i μ_i), W^i_(n_a,n_b,λ_i,μ_i)) ← B_W[W^i_idx]
22                 κ_i^max ← kmax(λ_i, μ_i, L_i)
23                 for each κ_0 = 1, ..., κ_0^max do
24                     for each ρ_0 = 1, ..., ρ_0^max do
25                         (−1)^{S_f+S_0+L_i} √((2S_i+1)(2S_f+1)(2S_0+1)(2L_i+1)(2L_f+1))
                           Σ_{κ_i=1}^{κ_i^max} Σ_{κ_f=1}^{κ_f^max} W^f_(n_a,n_b,λ_f,μ_f)(l_a, l_b, κ_f, L_f) W^i_(n_d,n_c,λ_i,μ_i)(l_d, l_c, κ_i, L_i)
                           W^0_(λ_f,μ_f,λ_i,μ_i,λ_0,μ_0)(κ_f, L_f, κ_i, L_i, κ_0, S_0, ρ_0)
26                     end
27                 end
28             end
29         end
30     end
31     for each κ_0 = 1, ..., κ_0^max and ρ_0 = 1, ..., ρ_0^max do
32         results(S_f, S_i, S_0, κ_0, ρ_0, matrixElementsBlockIdx) =
             reduce(results(S_f, S_i, S_0, κ_0, ρ_0, t))
33     end
34 end
```

4.3. **Multiple GPU parallelization.** To support these computations on distributed memory parallel systems including multiple GPU systems, we implemented a parallel algorithm based on the MPI library. The loop over all n_a, n_b, n_c, n_d values of HO shells is distributed among collaborating MPI processes. As the runtime is

a strongly varying function of HO shell numbers, we use a single master process for distribution of the workload among collaborating processes. If multiple MPI processes run on a single computing node, they share its GPU resources, i.e. more MPI processes may access one GPU. We did not develop hybrid OpenMP/MPI algorithm since the amount of data shared between threads processing different HO shells would not be significant.

5. **Performance results.** To evaluate the performance, we conducted experiments using two different systems. The first one is the Blue Waters system, which is equipped with 4228 Cray XK7 nodes that consist of one 8-core AMD Opteron 6276 Interlagos Processor and one NVIDIA K20 GPU with 2688 CUDA cores, 6 GB of memory and 250 GB/s memory bandwidth. The second system is made up of a single node containing two 16-core IBM Power9 CPUs that are connected by high-speed interface NVLink to a single NVIDIA V100 GPU. This GPU has 5120 CUDA cores, 32 GB of memory and 900 GB/s memory bandwidth.

We measured the performance for different values of $N_{\max}$ parameter that defines the size of two-nucleon Hilbert space in which the interaction matrix elements are provided. This means that for a given $N_{\max}$ cutoff parameter, we are considering only two-nucleon matrix elements $\langle n_a l_a j_a, n_b l_b j_b; J \| \hat{V} \| n_c l_c j_c, n_d l_d j_d; J \rangle$ such that $0 \leq (n_a + n_b) \leq N_{\max}$ and $0 \leq (n_c + n_d) \leq N_{\max}$. In particular, $N_{\max} = 8$ corresponds to $2,300,760$ input two-nucleon matrix elements, $N_{\max} = 10$ corresponds to $8,143,452$ input two-nucleon matrix elements, $N_{\max} = 12$ corresponds to $24,578,196$ input two-nucleon matrix elements, and $N_{\max} = 14$ corresponds to $65,615,742$ input two-nucleon matrix elements.

Table 1 shows the performance results obtained on the Blue Waters system. The computations were carried out using 1, 2, 4, 8, 16, and 64 nodes, each with 8 MPI processes. One of all MPI processes was the master process assigning HO shells to other processes. The fourth column in Table 1 provides efficiency evaluated with respect to the number of nodes (we do not take the master process into account). Our master-slave workload distribution attains a very good scalability in particular for larger values of $N_{\max}$, that is, for larger sets of interaction matrix elements. The last column in Table 1 shows speedups achieved by the GPU accelerated implementation compared to the CPU version of the code. We see that our GPU implementation delivers speedups ranging from 2.44 to 5.93. The speedups tend to decrease for increasing values of $N_{\max}$ parameter. This is caused by the fact that increasing the $N_{\max}$ parameter entails a rapid increase in the number of SU(3) coupling coefficients required. As a result, a more significant portion of the runtime is spent computing on CPUs. (This further suggests that a fresh look into the possibility of developing a parallel SU(3) package could lead to further gains.) Table 2 shows the performance obtained on the Power9 system with 32 MPI processes. These results corroborate the performance behavior observed on the Blue Waters system.

6. **Conclusion.** We presented the algorithm for computing the expansion of a nucleon-nucleon potential operator in terms of the $\mathrm{SU(3)} \times \mathrm{SU_S(2)}$ irreducible tensors. We described its implementation that combines MPI and CUDA parallelization. Enabling GPU using CUDA leads to a performance improvement. In particular, we achieved speedups ranging from 2.44 to 5.93 on Blue Waters, and almost a factor of 3 speedups for the system consisting of the two 16-core Power9 CPUs connected to a single NVIDIA V100 GPU.

$N_{\max}$	MPI procs.	CPU only		CPU+GPU	
		Time [s]	Efficiency	Time [s]	Speed-up
8	7+1	295.3	–	75.3	**3.92**
	15+1	137.6	**1**	36.8	**3.73**
	31+1	66.5	**1**	17.5	**3.79**
	63+1	39.4	**0.83**	9.23	**4.26**
	127+1	32.8	**0.56**	6.35	**5.17**
	255+1	31.0	**0.52**	5.22	**5.94**
10	7+1	2219	–	648	**3.42**
	15+1	1034	**1**	318	**3.24**
	31+1	499	**1**	151	**3.28**
	63+1	248	**0.99**	74	**3.32**
	127+1	165	**0.74**	43	**3.75**
	255+1	138	**0.59**	32	**4.25**
12	7+1	13083	–	4493	**2.91**
	15+1	6097	**1**	2116	**2.88**
	31+1	2943	**1**	1054	**2.79**
	63+1	1447	**1**	515	**2.80**
	127+1	776	**0.92**	269	**2.88**
	255+1	565	**0.68**	169	**3.33**
14	5+1	64865	–	26104	**2.48**
	15+1	30227	**1**	12204	**2.47**
	31+1	14596	**1**	5944	**2.45**
	63+1	7179	**1**	2932	**2.44**
	127+1	3581	**0.99**	1461	**2.45**
	255+1	2142	**0.83**	838	**2.55**

TABLE 1. Performance results obtained on the Blue Waters system. Notation +1 in the column with MPI processes denotes one master process which just assigns HO shells to other MPI processes. The master process is not taken into account for efficiency evaluation.

$N_{\max}$	CPU only	CPU+GPU	
	Time [s]	Time [s]	Speed-up
8	41.65	15.78	**2.63**
10	274.14	97.99	**2.79**
12	1649.7	611.1	**2.69**
14	7761.9	3407.6	**2.27**

TABLE 2. Performance results obtained for 32 MPI processes running on two 16-core IBM Power9 CPUs and a single NVIDIA V100 GPU.

The source code is available on Gitlab: https://gitlab.com/oberhuber.tomas/np-interactions.

Acknowledgments. The work on this paper was supported by OPVVV project no. CZ.02.1.01/0.0/0.0/16_019/0000765: Research Center for Informatics. This

work was also supported in part by the U.S. National Science Foundation (OIA-1738287, ACI-1713690) and SURA. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. We would also like to thank IBM Watson iLab in Czech Republic for providing Power9 CPUs based system.

REFERENCES

[1] Y. Akiyama and J. P. Draayer, A user's guide to Fortran programs for Wigner and Racah coefficients of SU_3, *Comp. Phys. Comm*, **5** (1973), 405–406.

[2] T. Dytrych, K. D. Launey, J. P. Draayer, P. Maris, J. P. Vary, E. Saule, U. Catalyurek, M. Sosonkina, D. Langr and M. A. Caprio, Collective modes in light nuclei from first principles, *Phys. Rev. Lett.*, **111** (2013), 252501.

[3] T. Dytrych, P. Maris, K. D. Launey, J. P. Draayer, J. Vary, D. Langr, E. Saule, M. A. Caprio, U. Catalyurek and M. Sosonkina, Efficacy of the SU(3) scheme for ab initio large-scale calculations beyond the lightest nuclei, *Comp. Phys. Comm.*, **207** (2016), 202–210.

[4] H. T. Johansson and C. Forssén, Fast and accurate evaluation of Wigner $3j$, $6j$, and $9j$ symbols using prime factorization and multiword integer arithmetic, *SIAM J. Sci. Comput.*, **38** (2016), A376–A384.

[5] K. D. Launey, T. Dytrych and J. P. Draayer, Symmetry-guided large-scale shell-model theory, *Prog. Part. Nucl. Phys.*, **89** (2016), 101–136.

[6] M. F. O'Reilly, A closed formula for the product of irreducible representations of SU(3), *J. Math. Phys.*, **23** (1982), 2022–2028.

Received January 2019; revised February 2020.

E-mail address: tomas.oberhuber@fjfi.cvut.cz
E-mail address: daniel.langr@fit.cvut.cz
E-mail address: draayer@sura.org
E-mail address: dytrych@ujf.cas.cz
E-mail address: kristina@phys.lsu.edu