Computer Physics Communications 278 (2022) 108427

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

GPU-accelerated time integration of Gross-Pitaevskii equation with discrete exterior calculus ☆

Markus Kivioja *, Sanna Mönkölä, Tuomo Rossi

University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35, FI-40014 University of Jyvaskyla, Finland

ARTICLE INFO

Article history:
Received 18 November 2021
Received in revised form 5 May 2022
Accepted 19 May 2022
Available online 24 May 2022

Keywords:
Gross-Pitaevskii
Discrete exterior calculus
GPGPU

ABSTRACT

The quantized vortices in superfluids are modeled by the Gross-Pitaevskii equation whose numerical time integration is instrumental in the physics studies of such systems. In this paper, we present a reliable numerical method and its efficient GPU-accelerated implementation for the time integration of the three-dimensional Gross-Pitaevskii equation. The method is based on discrete exterior calculus which allows us the usage of more versatile spatial discretization than traditional finite difference and spectral methods are applicable to. We discretize the problem using six different natural crystal structures and observe the correct choices of spatial tiling to decrease the truncation error and increase the reliability compared to Cartesian grids. We pay attention to the computational performance optimizations of the GPU implementation and measure speedups of up to 152-fold when compared to a reference CPU implementation. We parallelize the implementation further to multiple GPUs and show that 92% of the computation time can fully utilize the additional resources.

1. Introduction

Gross [1] and Pitaevskii [2] introduced a mathematical model for the physics of quantized vortices in superfluids. The model is expressed as the *Gross-Pitaevskii* equation (GPE), also known as the *non-linear Schrödinger* equation, which in the case of dilute gases of bosonic atoms [3] reads

$$i\hbar\partial_t\Psi(\mathbf{r},t) = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}) + g|\Psi(\mathbf{r},t)|^2\right]\Psi(\mathbf{r},t), \tag{1}$$

where Ψ is the complex-valued wave function, i is the imaginary unit, $\hbar$ is the reduced Planck constant, m is the atom mass, V is the external potential, and g is the effective interaction strength. The wave function is normalized so that $\int |\Psi(\mathbf{r},t)|^2 \mathrm{d}^3\mathbf{r} = N$, where N is the number of atoms.

The numerical time integration of the GPE is of great interest in physics e.g. for simulating and studying the behavior of Bose-Einstein condensates as shown by recent studies [4–6]. The numerical solutions require the spatial domain to be discretized and a comprehensive survey of GPE time integration methods [7] shows that it is widely done by using evenly spaced square and cubic grids, in 2D and 3D domains, respectively. In addition to the finite difference method, recently, the two mostly used discretization methods for the GPE have been the Fourier spectral method [8,9] and the finite element method which has enabled the usage of axial symmetric [10] and triangle [11] grids as well. It has been observed however that in the case of Maxwell's equations the choice of more diverse discretization grids can improve the accuracy of the solution [12,13]. In this paper, we will show that this is the case with the three-dimensional GPE as well, and that the accuracy of the time integration and the reliability of numerical simulations can be improved by increasing the isotropy of the spatial discretization. We measure the error of the time integration by using six different natural crystal structures for the discretization and discover the cubic grid to produce the largest error of them all.

We enable the usage of more sophisticated grids than what has previously been used with the GPE, by applying the concepts of differential geometry and exterior calculus. Particularly the employment of differential forms plays an important role by removing the metric from the differential operators and allowing their exact presentation also at the discrete level. The discrete extension to differential forms is given by the discrete exterior calculus (DEC) [14,15] which we will concentrate on in this paper. In DEC the only source of discretization error is the discrete Hodge star operator, which defines the relations between primal and dual discrete differential forms, and thus is determined by the relation of the primal and dual spatial grids. DEC has previously been successfully utilized in elastodynamics [16], fluid dynamics [17], electromag-

☆ The review of this paper was arranged by Prof. Hazel Andrew.

* Corresponding author.
E-mail addresses: markus.i.kivioja@student.jyu.fi (M. Kivioja), sanna.monkola@jyu.fi (S. Mönkölä), tuomo.j.rossi@jyu.fi (T. Rossi).

https://doi.org/10.1016/j.cpc.2022.108427

netism [18], and in quantum mechanics in the form of the standard linear Schrödinger equation [19]. Our work is the first one to apply DEC to the non-linear Schrödinger equation.

The computational performance of the time integration is of high importance, as the computation time determines how fine the discretization, and how large the spatial and time domains can be in practice. Those factors may determine the set of physics research problems the time integrator is applicable to, as the physical properties of the simulated systems can compose requirements for the accuracy and domain size [20]. The independent nature of a time integration step of an individual grid vertex, makes the method ideal for the massively parallel computing model [21]. In [22], [23], [24], and [25], general-purpose GPU computing (GPGPU) [26] has been utilized for solving the non-linear Schrödinger equation with promising results. Though this has only been done using Cartesian spatial discretization and in many cases by utilizing a ready-made linear algebra library.

We implement our DEC-based method on multiple GPUs by giving attention to the lower-level details and common performance bottlenecks [27] in the modern GPU hardware architectures. We note that the memory utilization and access patterns in particular play an important role in the performance optimizations of the GPU implementation, and we concentrate on them especially. The implementation shows up to 152-fold performance improvement when benchmarked against a reference CPU version, and good scalability over the number of GPUs. There is no previous research on the topic of using GPUs in DEC. Esqueda et al. [28] mention the usage of a single GPU in their numerical experiments, but give no description of the implementation nor its computing time performance, therefore this paper aims to fill those gaps.

The rest of the paper is organized as follows. In section 2 we describe the method for discretizing the spatial domain of the time integration and the six different grid types used in our experiments. Section 3 presents the application of the concepts of discrete exterior calculus to the GPE. We give a detailed description of our implementation and performance optimizations of the method on multiple GPUs in section 4. The numerical behavior and accuracy, as well as the execution time performance and its scalability, are examined in section 5. Last, we summarize the paper in section 6.

2. Spatial discretization

The three-dimensional spatial domain is discretized by using cell complexes consisting of linear and convex oriented k-cells, where $k \in \{0, 1, 2, 3\}$. Hence in our case, 0-cells are defined as points $\mathbf{x} \in \mathbb{R}^3$, oriented 1-cells as line segments between two 0-cells, oriented 2-cells as convex two-dimensional linear surfaces enclosed by finite sets of 1-cells, and oriented 3-cells as convex three-dimensional volumes enclosed by finite sets of 2-cells. The $(k-1)$-cells enclosing a k-cell are called boundary cells and the orientations of k-cells are determined by the order of their boundary cells.

We use sets of two cell complexes which consist of a primary complex and a dual complex. The dual complex is constructed by assigning a dual $(3-k)$-cell for each primary k-cell and by setting the positions of the dual 0-cells to be the circumcenters of their corresponding primary 3-cells, which ensures that the dual 1-cells are always orthogonal to the primary 2-cells and vice versa. We choose the circumcentric positioning of the dual cells, since it doesn't yield significant losses in the accuracy [29], but is computing timewise an optimal option, as we will show in section 3.

Since the quantized vortices modeled by the GP equation can advance in all directions, we make a hypothesis that the best accuracy for the time integration is achieved by maximizing the spatial isotropy of the cell complex. For this purpose, we construct the cell complexes using the six different natural crystal structures that were applied to the time integration of Maxwell's equations by Räbinä [30]. These structured grids are the cubic crystal systems: cubic, face-centered cubic (FCC), and body-centered cubic (BCC), and the tetrahedrally close-packed (TCP) structures: A15, C15, and Z.

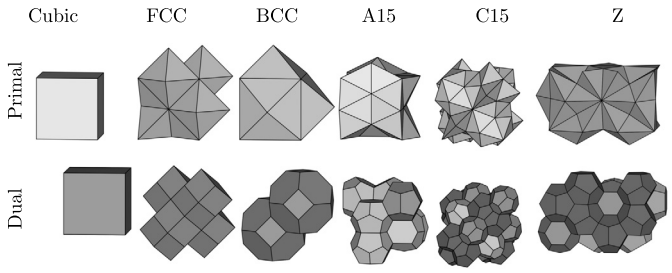


Fig. 1. The grid types used for the spatial discretization [13,32].

The standard cubic tiling acts as the basis for the other cubic crystal systems. The FCC and BCC grids are constructed by adding vertices at the center of the faces and bodies of the cubic grid, respectively. The FCC grid-based tiling of the spatial domain consists of alternating bodies of regular octrahedra and tetrahedra, and the dual bodies are Kepler's rhombic dodecahedra. In the BCC grid tiling, the primary bodies are congruent tetrahedra, whose faces are isosceles triangles with the relation of $2:\sqrt{3}$ between the bottom and side edges. The dual bodies of the BCC tiling are truncated octahedra with equal length edges.

The motivation for the TCP structures is to achieve small dihedral angles, which is shown to be a preferred quality of a grid in the time integration of Maxwell's equations [31]. The A15 and C15 grids are constructed by adding vertices to the BCC and FCC grids, respectively. The dual bodies of the A15 grid are irregular dodecahedra, centered at the BCC grid vertices, and tetrakaidecahedra surrounding the other primal vertices. The dual C15 grid consists of 12-hedra and 16-hedra. Unlike the other structures, the Z grid is asymmetric in that its z-direction is divergent from the x-, and y-directions. Though, the xy-plane is symmetric in 60-degree increments. Its dual grid is composed of 12-hedra, 14-hedra, and 15-hedra.

The BCC and C15 grids are expected to perform well since C15 has the smallest dihedral angle and BCC least variance in the edge lengths. Additionally, both grid types are previously shown to be the most numerically isotropic [32]. All of the aforementioned cubic crystal systems and TCP structures are visualized in Fig. 1.

With each of the grid types, we construct the full spatial tiling by repeating a basis replicable structure, and during the rest of this paper we call their separate occurrences *replicable structure instances*.

3. Computational model

We consider the more general dimensionless form of the time-dependent GPE (see [33])

$$i\partial_{\bar{t}}\bar{\Psi}(\bar{\mathbf{r}},\bar{t}) = \left[-\tfrac{1}{2}\nabla^2 + \bar{V}(\bar{\mathbf{r}}) + \bar{g}|\bar{\Psi}(\bar{\mathbf{r}},\bar{t})|^2\right]\bar{\Psi}(\bar{\mathbf{r}},\bar{t}), \tag{2}$$

where $\bar{\Psi}$, $\bar{V}$, $\bar{g}$, $\bar{\mathbf{r}}$, and $\bar{t}$ are suitably scaled versions of their barless counterparts, and $\int |\Psi(\bar{\mathbf{r}},\bar{t})|^2 d^3\bar{\mathbf{r}} = 1$. In this paper, we concentrate only on the time integration of the GPE and omit the computation of the initial value, in which case the exact scales can be ignored and $\bar{V}$ and $\bar{g}$ considered as some adjustable real-valued scalar parameters that are constant w.r.t. time.

In order to apply discrete exterior calculus, we first transform the equation in smooth setting into a more suitable format, which

assumes familiarity with differential geometry and exterior algebra.

We can regard the scalar wave function $\bar{\Psi}$ as a smooth differential 0-form $\hat{\Psi}$, which generalizes the equation to smooth differentiable manifolds of arbitrary dimensions

$$i\partial_{\bar{t}}\hat{\Psi} = \left[\tfrac{1}{2}\delta d + \bar{V} + \bar{g}|\hat{\Psi}|^2\right]\hat{\Psi},$$

where d is the exterior derivative, and $\delta = -\star^{-1} d\star$ is the codifferential. Here $\star$ is the Hodge star operator which, in n-dimensional space, maps k-forms to $(n-k)$-forms and is defined by the relation $\alpha \wedge (\star\beta) = \langle\alpha, \beta\rangle\, e_1 \wedge \cdots \wedge e_n$ for all pairs of k-forms α, β, when $e_1 \cdots e_n$ are orthonormal basis forms and $\wedge$ the wedge product.

We introduce a differential 1-form $\hat{u}$ and write the above equation as a pair of equations

$$\begin{aligned} i\partial_{\bar{t}}\hat{\Psi} &= \tfrac{1}{2}\delta\hat{u} + \bar{V}\hat{\Psi} + g|\hat{\Psi}|^2\hat{\Psi}, \\ \hat{u} &= d\hat{\Psi}. \end{aligned}$$

Considering the pair in 3-dimensional space, then by applying the Hodge star operator on both sides of the upper equation, integrating the upper and lower equations over a 3-manifold V and a 1-manifold C, respectively and applying the generalized Stokes' theorem

$$\int_{\Omega} d\omega = \int_{\partial\Omega} \omega$$

to both equations, the system takes an integral form of

$$\begin{aligned} i\partial_{\bar{t}} \int_V \star\hat{\Psi} &= \tfrac{1}{2}\int_{\partial V} \star\hat{u} + \int_V \left[V + g|\hat{\Psi}|^2\right] \star \hat{\Psi}, \\ \int_C \hat{u} &= \int_{\partial C} \hat{\Psi}, \end{aligned} \tag{3}$$

where $\partial\Omega$ denotes the boundary of a manifold Ω.

Hirani et al. [14] defined the *discrete differential k-form* on a k-cell c^k by the de Rham map

$$\tilde{\alpha} = \langle\alpha, c^k\rangle = \int_{c^k} \alpha,$$

where α is a smooth differential k-form, and $\tilde{\alpha}$ its corresponding discrete counterpart. Furthermore the *diagonal discrete Hodge star* operator was introduced and defined by the relation

$$\tfrac{1}{|*c^k|}\langle\star\alpha, *c^k\rangle = \tfrac{1}{|c^k|}\langle\alpha, c^k\rangle, \tag{4}$$

where $*c^k$ is the circumcentric dual cell of c^k, and $|c^k|$ denotes the k-volume of the cell. These definitions apply naturally to the integral form eq. (3) of the GPE when its smooth manifolds are replaced by oriented complex cells and the smooth Hodge star by the discrete version.

More precisely, if $\mathcal{B}$ is an oriented 3-cell and its boundary $\partial\mathcal{B}$ consists of oriented 2-cells $\mathcal{F}_i$, and we denote their dual 1-cells by $\mathcal{E}_i^*$ and further their boundaries $\partial\mathcal{E}_i^*$ by dual 0-cells $\mathcal{N}_{ij}^*$, we see that

$$\begin{aligned} \int_{\partial\mathcal{B}} \star\hat{u} &= \sum_i \int_{\mathcal{F}_i} \star\hat{u} = \sum_i \langle\star\hat{u}, \mathcal{F}_i\rangle = \sum_i \tfrac{|\mathcal{F}_i|}{|\mathcal{E}_i^*|}\langle\hat{u}, \mathcal{E}_i^*\rangle, \\ \langle\hat{u}, \mathcal{E}_i^*\rangle &= \int_{\mathcal{E}_i^*} \hat{u} = \int_{\partial\mathcal{E}_i^*} \hat{\Psi} = \sum_j \int_{\mathcal{N}_{ij}^*} \hat{\Psi} = \sum_j \langle\hat{\Psi}, \mathcal{N}_{ij}^*\rangle. \end{aligned}$$

Then by noting that $\int_{\mathcal{B}} \star\hat{\Psi} = \langle\star\hat{\Psi}, \mathcal{B}\rangle = \frac{|\mathcal{B}|}{|\mathcal{B}^*|}\langle\hat{\Psi}, \mathcal{B}^*\rangle$ and that the 0-volume $|\mathcal{B}^*| = 1$, we get

$$i\partial_{\bar{t}}\langle\hat{\Psi}, \mathcal{B}^*\rangle = \tfrac{1}{2}\tfrac{1}{|\mathcal{B}|}\sum_i \tfrac{|\mathcal{F}_i|}{|\mathcal{E}_i^*|}\sum_j \langle\hat{\Psi}, \mathcal{N}_{ij}^*\rangle + [\bar{V} + g|\hat{\Psi}|^2]\langle\hat{\Psi}, \mathcal{B}^*\rangle. \tag{5}$$

Note that here $\langle\hat{\Psi}, \mathcal{B}^*\rangle$ is the value of $\bar{\Psi}$ at the circumcenter of $\mathcal{B}$, and $\langle\hat{\Psi}, \mathcal{N}_{ij}^*\rangle$ are the signed values at the positions of $\mathcal{N}_{ij}^*$, where the signs are defined by the orientations of $\mathcal{E}_i^*$.

When the eq. (5) is applied to every dual 0-cell of a cell complex consisting of $m \in \mathbb{N}$ 3-cells, the resulting linear system of equations can be presented in a matrix form of

$$i\partial_{\bar{t}}\psi = \left[\tfrac{1}{2} \star_3 \mathrm{d}_2 \star_2^{-1} \mathrm{d}_2^T + U\right]\psi, \tag{6}$$

where $\mathrm{d}_2 \in \mathbb{Z}^{m\times m_{\mathcal{F}}}$ is a sparse incidence matrix whose elements are defined as $(\mathrm{d}_2)_{i,j} = \pm 1$ if the jth 2-cell is a boundary cell of the ith 3-cell and 0 otherwise. The sign is defined by the orientation of the boundary 2-cell w.r.t. the orientation of the 3-cell. Here $m_{\mathcal{F}} = \sum_i^m \#\partial\mathcal{B}_i$, where $\#\partial\mathcal{B}_i$ denotes the cardinality of the set of boundary 2-cells of the ith 3-cell. $\psi \in \mathbb{C}^m$ is a column vector consisting of the wave function values at the positions of the dual 0-cells, and the diagonal matrices $\star_2 \in \mathbb{R}^{m_{\mathcal{F}}\times m_{\mathcal{F}}}$ and $\star_3 \in \mathbb{R}^{m\times m}$ defined as

$$\begin{aligned} \star_2 &= \mathrm{diag}\left(\tfrac{|\mathcal{E}_1^*|}{|\mathcal{F}_1|}, \tfrac{|\mathcal{E}_2^*|}{|\mathcal{F}_2|}, \ldots, \tfrac{|\mathcal{E}_{\#\partial\mathcal{B}_m}^*|}{|\mathcal{F}_{\#\partial\mathcal{B}_m}|}\right), \\ \star_3 &= \mathrm{diag}\left(\tfrac{1}{|\mathcal{B}_1|}, \tfrac{1}{|\mathcal{B}_2|}, \ldots, \tfrac{1}{|\mathcal{B}_m|}\right). \end{aligned}$$

$U \in \mathbb{R}^{m\times m}$ is a diagonal matrix with elements $U_{i,i} = \bar{V}_i + \bar{g}|\psi_i|^2$, where $\bar{V}_i$ denotes the value of $\bar{V}(\bar{\mathbf{r}})$ at the position of the ith dual 0-cell. The diagonality of the matrices $\star_k$ is a direct consequence of the definition eq. (4) which only applies to circumcentric dual complexes.

The truncation error of eq. (6) is fully contained in the matrices $\star_k$ and is numerically shown to be $\mathcal{O}(|\mathcal{E}^*|^2)$ [34].

3.1. Time discretization

The time domain is discretized by using the standard central-difference method [35] which, when applied to the eq. (6), yields

$$i\tfrac{\psi^{(n+1)} - \psi^{(n-1)}}{2\Delta\bar{t}} = \left[\tfrac{1}{2} \star_3 \mathrm{d}_2 \star_2^{-1} \mathrm{d}_2^T + U^{(n)}\right]\psi^{(n)},$$

and furthermore gives us the formula for the time integration steps:

$$\psi^{(n+1)} = \psi^{(n-1)} - 2i\Delta\bar{t}M^{(n)}\psi^{(n)}, \tag{7}$$

where $\Delta\bar{t}$ is the length of the time step, $\psi^{(n)}$ and $U^{(n)}$ the wave function value vector and its corresponding matrix U, respectively, at a time instant $n\Delta\bar{t}$ so that $n \in \mathbb{Z}$, and

$$M^{(n)} = \tfrac{1}{2} \star_3 \mathrm{d}_2 \star_2^{-1} \mathrm{d}_2^T + U^{(n)} \in \mathbb{R}^{m\times m}.$$

In a general case, when the edge lengths of the dual grid vary, as is the case with the FCC, A15, C15, and Z grids, the ith row of the eq. (7) is written out as

$$\begin{aligned} \psi_i^{(n+1)} = \psi_i^{(n-1)} &- i\Delta\bar{t} \\ &\times \left[\tfrac{1}{|\mathcal{B}_i|}\sum_j \tfrac{|\mathcal{F}_{i,j}|}{|\mathcal{E}_{i,j}^*|}(\psi_i^{(n)} - \psi_{i,j}^{(n)}) + (\bar{V} + g|\psi_i^{(n)}|^2)\psi_i^{(n)}\right], \end{aligned} \tag{8}$$

where $\mathcal{F}_{i,j}$ denotes the jth boundary cell of the ith 3-cell and $\mathcal{E}^*_{i,j}$ its dual cell. $\psi_{i,j}$ denotes the element corresponding to the dual 0-cell that is a boundary cell of $\mathcal{E}^*_{i,j}$ but is not the ith dual 0-cell. The summation is over all boundary 2-cells of the ith 3-cell.

The cubic and BCC grids are special cases in that their dual edge lengths and the numbers of boundary faces of dual bodies don't vary, which allows us to simplify the eq. (8) to

$$\psi_i^{(n+1)} = \psi_i^{(n-1)} - i\Delta\bar{t} \times \left[\frac{|\mathcal{F}|}{|\mathcal{E}^*||\mathcal{B}|} (\#\partial\mathcal{B}\psi_i^{(n)} - \sum_j \psi_{i,j}^{(n)}) + (\bar{V} + g|\psi_i^{(n)}|^2)\psi_i^{(n)} \right].$$

The time discretization related truncation error of eq. (7) is the well known $\mathcal{O}(\Delta\bar{t}^2)$ error of the central-difference method.

The time discretization method is conditionally stable, in that there is a $|\mathcal{E}^*|$ dependent upper limit for the time step size and when the limit is exceeded the method becomes unstable. The instability is generally caused by a real or imaginary part of an element of ψ changing its sign on every time step which will eventually cause the absolute value to increase without a limit as shown in [32].

We try to avoid this by limiting the maximum absolute difference of changes of $\psi_R := \mathfrak{Re}(\psi)$ (or, equivalently, of $\psi_I := \mathfrak{Im}(\psi)$) on subsequent time steps with

$$|(\psi_{R_i}^{(n+2)} - \psi_{R_i}^{(n)}) - (\psi_{R_i}^{(n)} - \psi_{R_i}^{(n-2)})| \leq C\psi_{R_i}^{\max}, \tag{9}$$

for some constant C. Here $\psi_R{}^{\max}$ is a vector so that $|\psi_{R_i}^{(n)}| \leq \psi_{R_i}^{\max}$ for all i and n. We take a linear approximation of the eq. (7) by using a matrix $M := M^{(0)}$ and with that we get

$$(\psi_R^{(n+2)} - \psi_R^{(n)}) - (\psi_R^{(n)} - \psi_R^{(n-2)}) = 2\Delta\bar{t}M\psi_I^{(n+1)} - 2\Delta\bar{t}M\psi_I^{(n-1)}$$
$$= 2\Delta\bar{t}M(\psi_I^{(n+1)} - \psi_I^{(n-1)}) = -4\Delta\bar{t}^2M^2\psi_R^{(n)},$$

which when applied to eq. (9), with the definition of $\psi_R{}^{\max}$, gives us

$$\Delta\bar{t}_i \leq \sqrt{\frac{C\psi_{R_i}^{\max}}{|4M^2{}_{i,*}\psi_R^{\max}|}},$$

where $M^2{}_{i,*}$ denotes the ith row of the matrix M^2. We note that $M_{i,i} >> M_{i,j}$ when $j \neq i$ and hence approximate M by considering only its diagonal elements which reduces the above condition to

$$\Delta\bar{t}_i \leq \frac{\sqrt{C}}{2M_{i,i}}. \tag{10}$$

For a global $\Delta\bar{t}$ which holds for all i, the greatest matrix diagonal element must be picked. Since eq. (10) is an approximation we used numerical experiments to find a sufficient value for the constant C. With the grid types introduced in section 2, the time integration was found to be stable when $C = 1$ which gives us the final condition

$$\Delta\bar{t} < (2 \max_{1\leq i\leq m} M_{i,i})^{-1}. \tag{11}$$

4. GPU implementation

Since the matrix M in the time integration step eq. (7) is usually large, the vector multiplication may be infeasible slow, when computed in a serial manner. We, therefore, take advantage of parallel computing and utilize GPGPU. This is done by implementing the DEC-based GPE time integrator using Nvidia's CUDA API [36] and targeting hardware with CUDA compute capability of 5.2 and above. In this and the following sections, we will also use the terminology introduced with CUDA.

We assign one GPU thread for each dual 0-cell in the cell complex so that the ith thread will effectively compute the value of the expression

$$\psi_i^{(n+1)} = \psi_i^{(n-1)} - 2i\Delta\bar{t}M_{i,*}^{(n)}\psi^{(n)}, \tag{12}$$

where $M_{i,*}$ denotes the ith row of the matrix M. Since M is sparse we precompute, for each row i, a set $J^i = \{j \in \mathbb{N} : M_{i,j} \neq 0\}$ of the column indices of the non-zero elements, and only include the columns $j \in J^i$ to the computation of eq. (12). From the derivation of the matrix M in section 3, it follows that $\#J^i = \#\partial\mathcal{B}^i + 1$, where $\mathcal{B}^i$ is the 3-cell enclosing the ith dual 0-cell.

Since the full spatial grid is constructed by duplicating a replicable structure, we also generate the sets of the columns of the non-zero matrix elements only for one structure and reuse them with the other structure instances. This is achieved by replacing the sets J^i by defining new sets

$$\hat{J}^i = \left\{ \hat{j} \in \mathbb{Z}^4 : \hat{j} = \left(c_l^j, s_x^j - s_x^i, s_y^j - s_y^i, s_z^j - s_z^i\right), M_{i,j} \neq 0, i \neq j \right\},$$

where c_l^j is the local index of the jth dual 0-cell in the replicable structure, and s_x^j, s_y^j, s_z^j denote the x-, y-, and z-directional indices of the cell's structure instance, respectively. In other words, we express the columns of the non-zero matrix elements using four-component vectors consisting of the local index in the replicable structure and the x-, y-, and z-directional offsets of the indices of the structure instance w.r.t. the indices of the structure instance of the ith dual 0-cell. With this approach, the number of required different sets $\hat{J}^i$ is decreased to m_s, where m_s denotes the number of the dual 0-cells in a single replicable structure instance. We know that every set J^i holds an element with a value of i, so we can drop the cases of $i = j$ from the sets $\hat{J}^i$ and thereby reduce their cardinality to $\#\hat{J}^i = \#\partial\mathcal{B}^i$.

Likewise the matrix M is compressed by storing only the non-zero elements corresponding to one structure instance so that the stored element count is $\sum_{i=0}^{m_s} \#\hat{J}^i$. The matrix M and the sets $\hat{J}^i$ depend on the grid type and are computed only once in advance for each of the grids. Apart from the optimizations mentioned later in the paper, M and $\hat{J}^i$ are the only grid-dependent components of the implementation and hence the runtime part of the integrator does not require changes between the different spatial discretizations.

The solution vector ψ is double buffered in the global GPU memory so that one of the buffers holds the values for the time instants $t^{(n)}$ of odd n, and the other for the time instants of even n. The GPU kernel function treats one of the buffers as the vector $\psi^{(n)}$ in eq. (12) and the other one as both $\psi^{(n+1)}$ and $\psi^{(n-1)}$ by updating its value cumulatively. The same kernel is invoked for every time step and the two value buffers are swapped in between the invocations.

The GPU kernel function is described in pseudocode in Algorithm 1, where S is a three component vector, holding the domain size, measured in replicable structure instance counts, Js an array of all unique sets $\hat{J}$, $\star$s a dense matrix holding the absolute values of the non-zero elements of $\star_3 \mathrm{d}_2 \star_2^{-1} \mathrm{d}_2^T$ corresponding to the indices in Js, and dt the time step size.

4.1. Memory access pattern

The thread group dimensions of the kernel launches are configured in a way that the global y- and z-directional indices of the threads match with the indices of the replicable structure instances, whose value elements the threads are assigned to compute. The global thread count in the x-direction is multiplied by

Algorithm 1 GPU kernel for computing one time step.

```
1: procedure TimeStep(ψ^(n+1), ψ^(n−1), ψ^(n), ⋆s, Js, V, g, dt)
2:     c_l ← (blockIdx.x × blockDim.x + threadIdx.x) mod m_s
3:     s_x ← (blockIdx.x × blockDim.x + threadIdx.x)/m_s
4:     s_y ← (blockIdx.y × blockDim.y + threadIdx.y)
5:     s_z ← (blockIdx.z × blockDim.z + threadIdx.z)
6:     i ← m_s × (s_z × S.x × S.y + s_y × S.x + s_x) + c_l
7:     J ← Js[c_l]
8:     ⋆ ← ⋆s[c_l]                          ▷ Take one row from the matrix ⋆s
9:     Δ ← (0 + 0i) ∈ ℂ
10:    for ∀f ∈ {0, 1, ..., #J − 1} do
11:        j ← J[f]                          ▷ j is a 4 element array
12:        j ← m_s × (j[3] × S.x × S.y + j[2] × S.x + j[1]) + j[0]
13:        Δ ← Δ + ⋆[f] × (ψ^(n)[i] − ψ^(n)[j])
14:    ψ^(n+1)[i] ← ψ^(n−1)[i] − dt × i × Δ + (V[i] + g|ψ^(n)[i]|^2)ψ^(n)[i]
```

m_s so that the corresponding x-directional structure index is determined by $s_x = \left\lfloor \frac{t_x}{m_s} \right\rfloor$, where $t_x \in \mathbb{N}$ denotes the global x-directional thread index. The local index of the dual 0-cell in the structure is then given by $c_l = t_x \mod m_s$.

The memory layout of the vectors ψ is such that the elements corresponding to the dual 0-cells in the same replicable structure instance are stored in a contiguous interval, and the structure instances themselves in row-major order, where a row is considered to coincide with the x-direction. Therefore by structuring the thread groups in an aforementioned way, we maximize the coalescing of the global memory load and store access patterns. We use double-precision floating-point numbers for all of the data and arithmetics, so a complex-valued element of ψ will occupy 16 bytes of global memory. Hence on our target hardware with L2-cached global memory accesses with 32-byte cache lines, every pair of two subsequent elements can be accessed with the same transaction. Therefore when all of the threads access the element of their assigned dual 0-cells simultaneously, a warp of 32 threads makes 16 transactions per load/store.

A set $\hat{J}^i$ contains duplicates of elements of other sets $\hat{J}^l$, with $l \neq i$, in which case every element of $\psi^{(n)}$ will be read multiple ($\#J^i$) times during the computation of one time step. We avoid increasing the number of global memory accesses by utilizing the local on-chip memory on modern GPU hardware, called shared memory. This is accomplished by making every thread first load its corresponding element of $\psi^{(n)}$ from the global memory into the shared memory so that during the computation of $\psi^{(n+1)}$ most of the elements of $\psi^{(n)}$ can be loaded from the shared memory with low latency. With the current GPU hardware architectures the shared memory can be shared only between the threads inside the same thread group, and because some of the needed elements of $\psi^{(n)}$ might be loaded into the shared memory by threads living in a different thread group than the one computing the element of $\psi^{(n+1)}$, those still need to be read from the global memory. The number of elements needed to be read from the global memory during the computation can be reduced by maximizing the size of the thread groups, so that when the threads are associated with the dual 0-cells, the area of the borders between thread groups is minimized.

In the GPU hardware the shared memory is divided into banks so that each bank can be accessed simultaneously. However when multiple threads in the same warp make a request to the same bank the accesses conflict and are serialized. We avoid this by properly permuting the elements of the sets $\hat{J}^i$ and by adding memory padding in between the elements of the vectors ψ. We optimize the method for hardware with 32 banks and a bank width of 32 bits. Therefore each double-precision complex-valued element of 128 bits will occupy 4 consecutive banks in the shared memory.

Since one bank can only transfer 32 bits per transaction, a load/store of 128 bits per thread will always need at a minimum of four transactions. For this reason, GPUs with CUDA compute capability of 5.2 and above will access 128 bits per thread for 8 threads at a time so that the whole warp access creates exactly four transactions and causes no bank conflicts. In consequence, we can design our shared memory layout and accesses for eliminating bank conflicts by considering an artificial system with a warp size of 8 threads and a shared memory with 8 banks of 128 bits per bank.

From this, it follows that the element with the local dual 0-cell index of c_l is stored in the same bank as all the elements with an index $(c_l + 8k) \mod m_s$, where $k \in \mathbb{N}_+$. Consequently, the number of elements of different dual 0-cell indices stored in the same bank is $\frac{\mathrm{lcm}(m_s, 8)}{8}$, where $\mathrm{lcm}(\cdot, \cdot)$ denotes the least common multiple. When m_s is even, the previous expression can have values of $\frac{m_s}{2}$, $\frac{m_s}{4}$, or $\frac{m_s}{8}$. When m_s is odd every bank would hold elements of all the different dual 0-cell indices, but in that case we add a 128 bit-sized padding in between the last and first elements of different replicable structure instances, which reduces the case back to the even m_s. The elements of the sets $\hat{J}^i$ are strove to be permuted so that for any fixed $l \in \{0, 1, \cdots, \#\hat{J}^i\}$, it holds that

$$\begin{aligned} &\hat{J}^i_{l,0} \neq (\hat{J}^j_{l,0} + 8k) \mod m_s, \\ &\forall i, j \in \{0, 1, \cdots, m_s\} : i \neq j, \lfloor \frac{i}{32} \rfloor = \lfloor \frac{j}{32} \rfloor, \end{aligned} \tag{13}$$

where $\hat{J}^i_{l,0}$ denotes the first component of the lth element in the set $\hat{J}^i$. The equality of the integer parts of the divisions by 32 relaxes the requirement to apply only inside the same warp. When $m_s > 32$ there might be no possible permutations for the elements so that the inequality eq. (13) would hold, without also permuting the order of the sets $\hat{J}^i$ themselves. We avoid doing that since it would decrease the number of coalesced accesses to the global memory, which has a greater performance cost. For that reason with some grid types, with large m_s, we settle for only minimizing the number of pairs (i, j) breaking the rule. A heuristic iterative algorithm is used for the generation of the permutations.

We are able to allow for $\hat{J}^i_{l,0} = \hat{J}^j_{l,0}$, because of the shared memory broadcasting capability offered by our target GPU hardware, which permits multiple threads in the same warp to access the same 32-bit word without causing bank conflicts. However it is not always the case that the target elements of the loads made by the ith and jth thread are the same, even though $\hat{J}^i_{l,0} = \hat{J}^j_{l,0}$. This happens when loads are made across the borders of structure instances, and when $m_s \mod 32 \neq 0$, so that different threads in the same warp may not be assigned with dual 0-cells within the same structure instance. We eliminate the bank conflicts caused by these cases by first squeezing the thread groups to extend only in the x-direction so that one warp is able to access at a maximum of three structure instances, which are also guaranteed to be consecutive. Then by adding memory padding in between all the elements of ψ, if necessary, to increase the number of structure instances in between the two closest elements which have the same local dual 0-cell index and share the same bank, to be greater than two.

Fig. 2 presents the layout of the shared memory after 64-bit padding has been added, in the case of the BCC and FCC grids with $m_s = 12$. A matrix cell in the figure represents a combination of two consecutive 32-bit banks, and the numbers inside the cells are the local dual 0-cell indices corresponding to the elements stored in the banks. From the figure, it can be seen how there are $4m_s$ elements in between the cases where two elements of the same c_l are stored in the same bank. Without the padding, the interval would be only $2m_s$, and the cases of $\hat{J}^i_{l,0} = \hat{J}^j_{l,0}$ able to cause bank conflicts.

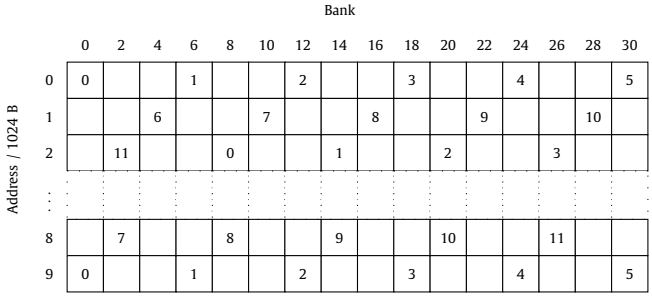


Fig. 2. The layout of ψ in the shared memory banks after the padding has been added, in the case of the BCC and FCC grids.

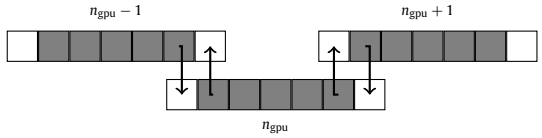


Fig. 3. The stored elements of ψ by different GPUs, and the memory copies between them, presented in the units of z-slices. The gray cells denote the z-slices the GPUs are assigned to compute.

4.2. Multiple GPUs

The implementation is further parallelized to multiple GPUs by using domain decomposition [37]. The buffers for the vectors ψ are laid out in the memory in a way that the elements corresponding to the replicable structure instances on the same xy-plane (z-slice) will be in a contiguous memory area. Therefore we distribute the work across multiple GPUs by splitting the spatial domain across the z-axis, to minimize the number of data copy operations between different GPUs, and hence the overhead caused by them. The splitting is done as evenly as possible so that the maximum difference in the number of assigned z-slices for different GPUs is one.

The computation domains assigned to different GPUs are separate, though two consecutive GPUs store elements of ψ in a way that the subdomains covered by the stored elements overlap over two z-slices. Hence if we denote the total number of z-slices with N_z, and the number of GPUs with N_{gpu} (and we assume that $N_z \mod N_{\text{gpu}} = 0$), the GPU of index $n_{\text{gpu}} \in \mathbb{N}$ will be assigned to compute the elements of the z-slices with indices from $n_{\text{gpu}} \frac{N_z}{N_{\text{gpu}}}$ to $\left(n_{\text{gpu}} + 1\right) \frac{N_z}{N_{\text{gpu}}} - 1$, inclusive, but store the elements from $n_{\text{gpu}} \frac{N_z}{N_{\text{gpu}}} - 1$ to $\left(n_{\text{gpu}} + 1\right) \frac{N_z}{N_{\text{gpu}}}$. After each time step the n_{gpu}th GPU will copy the elements of its second z-slice to the memory of the last z-slice of the $(n_{\text{gpu}} - 1)$th GPU. Similarly the second to last z-slice will be copied to the first z-slice of the $(i_{\text{gpu}} + 1)$th GPU, as shown in Fig. 3. Each array cell in the figure represents a whole z-slice and the cells colored with gray denote the assigned computational domains of the GPUs.

To avoid unnecessary synchronizations between the CPU and GPUs, and between and within individual GPUs, we utilize the concepts of streams, events, and asynchronous memory copies and kernel launches introduced by the CUDA API. For every GPU there are three streams and events created, one for the kernel execution and one for each of the two memory copies. The two memory copies of a single GPU are independent of each other, and therefore can be initiated concurrently, which furthermore is achieved by placing them in separate streams. The kernel invocations are placed in their own stream, with an event after each invocation, to signal the memory copy streams when they can start their work. An event is also added to each of the memory copy streams, to signal the kernel streams of the neighboring GPUs of the next time step. The CPU is not blocked by any of the GPU work, up until the command buffers of the GPUs are fully occupied, or we need to access the time integration results on the CPU side.

Table 1
The lengths of dual 1-cells after the scales, for equalizing the operation counts, have been applied.

Cubic	FCC	BCC	A15	C15	Z
1	1.09	0.80	0.93	0.68	0.99

5. Numerical experiments

We test the method by time integrating the dimensionless GPE eq. (2) and initializing the wave functions using stationary vortex solutions $\bar{\Psi}_{\lambda,\bar{g},\kappa}$, arising from the dynamics of Bose-Einstein condensates [38]. The initial values take a form of

$$\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}},\bar{t}) = f(\bar{\rho},\bar{z})e^{i\kappa\bar{\phi}-i\bar{\mu}\bar{t}},$$

where f is a real-valued function satisfying the time-independent equation

$$\left[\frac{1}{2}\left(\frac{\kappa^2}{\bar{\rho}^2} - \partial_{\bar{\rho}}^2 - \frac{1}{\bar{\rho}}\partial_{\bar{\rho}} - \partial_{\bar{z}}^2\right) + \bar{V} + \bar{g}f^2\right] f = \bar{\mu} f. \tag{14}$$

Here $\bar{\mathbf{r}} = (\bar{\rho},\bar{\phi},\bar{\mathbf{z}})$ is the dimensionless position vector presented in cylindrical coordinates, κ the so-called winding number, and $\bar{\mu}$ the dimensionless chemical potential. The eq. (14) can be solved with e.g. a relaxation method [39]. An example of an initial value is visualized in Fig. 4.

We initialize the vectors of the discretized wave functions with

$$\psi_i^{(n)} = \bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}}_i, n\Delta\bar{t}), n \in \{-1, 0\}.$$

The computational domain is the smallest cuboid so that

$$|\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}},0)| > 10^{-5} \max_{\bar{\mathbf{r}}} |\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}},0)|, \tag{15}$$

for all $\bar{\mathbf{r}}$ inside the cuboid. As the boundary condition, we use $\bar{\Psi}(\bar{\mathbf{r}}) = 0$, for all $\bar{\mathbf{r}}$ outside of the cuboid.

All of the source codes used in the measurements of this section are available in a public repository [40].

5.1. Accuracy

We compare the accuracy of different spatial discretizations by scaling the grids in a way that the lengths of the dual 1-cells are less than 5% of the effective wavelength of $\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}},0)$, and that the number of arithmetic operations per integration of one time unit $\sum_{i=0}^{m} \#J^i \frac{1}{\Delta t}$, stays constant between the different grid types, and is $6{\times}10^9$. The resulting distances of two adjacent dual 0-cells w.r.t. the cubic grid are presented in Table 1.

The accuracies are estimated by comparing the time integration results to a stationary vortex state $\bar{\Psi}_{\lambda,\bar{g},\kappa}$. We use a stationary vortex state with $\lambda = 1, \bar{g} = 300, \kappa = 10$, that can be shown by the Bogoliubov equations (see [41]), to be dynamically unstable. This causes the vortex to split up into multiple smaller parts, by even a small perturbation to the stationary state. Hence, we first measure the amount of time $\bar{t}$ the equation can be integrated before the perturbation caused by the numerical error initiates the splitting. We do this by measuring an error defined as $E(\bar{t}) = 1 - |\int \bar{\Psi}^*_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}},0)\bar{\Psi}(\bar{\mathbf{r}},\bar{t})d^3\bar{\mathbf{r}}|$, and considering the split to happen when $E(\bar{t}) > 0.01$.

From Fig. 5a it can be perceived that the simulated vortex stays intact for the longest when the BCC and C15 grids are used. The cubic grid not only causes the quickest vortex splitting but also

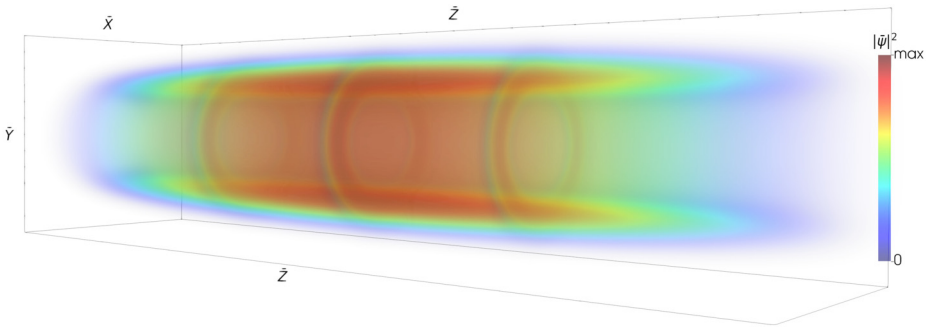
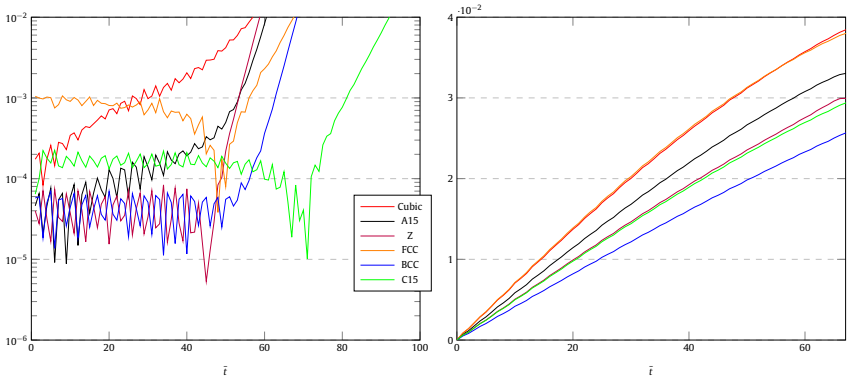


Fig. 4. A visualization of an initial value $\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}}, 0)$, with $\lambda = 0.1$, $\bar{g} = 5000$, and $\kappa = 10$. Three cross sections are highlighted at $z = -10, 0, 10$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) $E(\bar{t})$ (b) $RMSE(\bar{t})$

Fig. 5. Errors $E(\bar{t})$ and $RMSE(\bar{t})$ produced by the numerical time integration with different spatial discretizations.

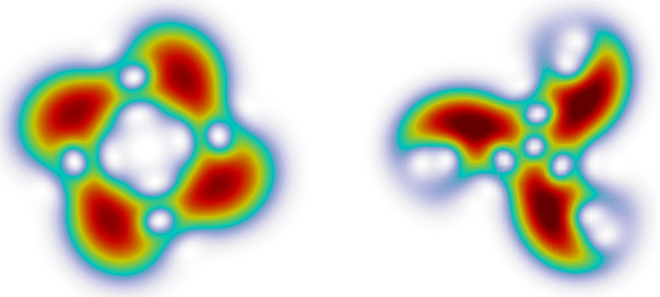
results in increasing deviation from the stationary state right from the beginning, whereas almost all of the other grids keep the error relatively constant at first. A15 is the only other grid that shows similar deviation behavior as the cubic grid.

Since $E(\bar{t})$ only takes the magnitude of the wave function into account, we also measure the root mean square error as

$$RMSE(\bar{t}) = \sqrt{\frac{1}{m}\sum_{i=0}^{m} |\bar{\Psi}_{\lambda,\bar{g},\kappa}(\bar{\mathbf{r}}_i, n\Delta\bar{t}) - \psi_i^{(n)}|^2},$$

which considers the phase as well. The measurement outcomes are presented in Fig. 5b, which shows similar results as were observed earlier, as in the cubic grid being the least accurate and C15 and BCC the most. In this measurement, the FCC grid produces similar accuracy with the cubic grid at first, but closer to the end starts to deviate, by showing a slightly slower increase in the error. Interestingly, when the phase is taken into account, the error increases slower with the BCC grid than with C15, even though C15 maintained the vortex shape the longest.

The Bogoliubov equations are also able to predict the splitting symmetries of the dynamically unstable stationary states, as in how many parts they split up into, when perturbed. With the stationary state used in our accuracy measurements, the prediction is that a 3-fold splitting should occur. Fig. 6 shows visualizations of a cross-section of the wave function at $\bar{z} = 0$, after time integrated with the cubic and BCC grids, until $E(\bar{t}) > 0.01$. The figure clearly shows how the splitting symmetry produced by the cubic grid doesn't match with the prediction of the Bogoliubov equations but is 4-fold instead. With the BCC grid, a 3-fold splitting can be observed, which matches the prediction.



(a) Cubic (b) BCC

Fig. 6. Visualizations of a cross-section of $|\bar{\Psi}|^2$ at $\bar{z} = 0$, after time integrations with the cubic and BCC grids, until $E(\bar{t}) > 0.01$.

5.2. Computational performance

At first, we test the computational performance of the GPU implementation by using the same stationary vortex state as in the accuracy measurements, but by increasing the dimensions of the computational domain, so that it becomes a cube, still satisfying the requirement eq. (15). We vary the problem size by scaling the spatial grid and monitoring its effect on the speedups the GPU provides compared to a serial CPU implementation. This experiment was performed with the BCC grid, and using an NVIDIA® GeForce® RTX 2080 Ti GPU, and an AMD Ryzen™ 9 3900 CPU. The length of an edge of the computation grid cube was varied from 14 to 270 when measured in replicable structure instance counts. Hence, the number of degrees of freedom varied between $14^3 \times 12$ and $270^3 \times 12$, inclusive.

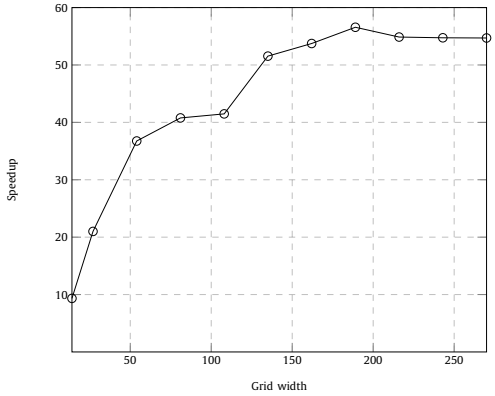


Fig. 7. The computational performance improvements provided by the GPU against the CPU, as a function of problem size.

The results are presented in Fig. 7, which shows a rapid increase in the speedups at first, when the problem size increases, but then reaches the maximum of 57-fold speedup, at the problem size of $189^3 \times 12$ degrees of freedom, and stays approximately on that level from thereon. From that, we predict that this is the maximum reachable speedup with the used GPU and CPU.

We look more deeply into the reasons behind this behavior by using the Roofline model [42]. For that, we pick four cases from the previous experiment, the grid widths of 14, 27, 54, and 189, and examine their computational performances in relation to their arithmetic intensities. In Fig. 8a the aforementioned four cases are numbered from 1 to 4, respectively. The horizontal ceiling is the maximum peak performance of the double-precision pipeline of the used GPU, of 368 $\frac{\text{GFLOP}}{\text{s}}$. The sloped ceiling is produced by the maximum memory bandwidth of 648 $\frac{\text{GB}}{\text{s}}$. It can be observed that with the smallest problem sizes the GPU is not fully utilized and the performance falls significantly behind the possible maximum. When the problem size increases the performance approaches the ceiling, but at the same time, the arithmetic intensity decreases, making the performance memory bandwidth bound. There is an anomaly when it comes to the arithmetic intensity of the problem size number three, which might be caused by a lucky alignment of the borders of thread groups, minimizing the number of the required compute time global memory loads, mentioned in section 4.1.

We repeat the performance measurements with the other most accurate grid type C15 and examine the results likewise using the Roofline model, shown in Fig. 8b. With the C15 grid, the distribution of the arithmetic intensities between different problem sizes is smaller than with BCC, but still indicating similar behavior of the largest problem size moving the performance from being compute-bound to bandwidth bound. Also as with BCC, the same anomaly in arithmetic intensities in the case of the problem size number three is observable. With the BCC grid, we were able to eliminate all of the shared memory bank conflicts, but couldn't achieve that with C15, leaving some conflicts to the load transactions. This is likely one of the reasons why the C15 grid only achieves 89% of the performance of the BCC grid.

Next we examine the computational performance on multiple GPUs. The measurements were performed using up to 8 NVIDIA® Tesla® P100 GPUs. As an initial value, we used a stationary vortex solution with $\lambda = 0.1$, $\bar{g} = 5000$, and $\kappa = 20$. For the discretization, the BCC grid was used, and a scale which produces $87 \times 87 \times 302 \times 12$ degrees of freedom. We examine the speedups provided by the additional GPUs compared to the performance on a single GPU, by first retaining the problem size constant and measuring $\frac{T_1}{T_{N_{\text{gpu}}}}$, where $T_{N_{\text{gpu}}}$ is the computation time on N_{gpu} GPUs. The results are shown in Fig. 9a, where we also present the function from Amdahl's law [43]

$$S_t(N_{\text{gpu}}) = \frac{1}{(1 - p_t) + \frac{p_t}{N_{\text{gpu}}}}, \tag{16}$$

fitted to our observed data, yielding $p_t = 0.92$. $S_t(N_{\text{gpu}})$ is the speedup, and p_t the proportion of the computation time that the part benefiting from the additional GPUs originally occupied. With this, we can then calculate the maximum achievable speedup to be $\lim_{N_{\text{gpu}} \to \infty} S_t(N_{\text{gpu}}) = \frac{1}{1 - p_t} = 12.5$. The majority of the 8% of the computation time that doesn't gain from the added GPUs, is likely caused by the latency of the memory copies between different GPUs.

We then measure the speedups by fixing the computation time to T_1 on all of the considered GPU counts and adjusting the problem sizes accordingly. In this case, the speedups are defined as $\frac{W_{N_{\text{gpu}}}}{W_1}$, where $W_{N_{\text{gpu}}}$ denotes the computation workload on N_{gpu} GPUs. In turn, to these measurements, we fit Gustafson's law [44]

$$S_w(N_{\text{gpu}}) = 1 - p_w + N_{\text{gpu}} p_w, \tag{17}$$

where p_w is the proportion of the workload that the part benefiting from the additional GPUs originally occupied. The fitting produces $p_w = 0.78$ and both the measured and fitted speedups are presented in Fig. 9b. The most significant cause for p_w being smaller than p_t is that we scale the problem size equally in all spatial dimensions and if we denote this scale by s_w, the amount of transferred data between the GPUs increases in $\mathcal{O}(s_w^{\frac{2}{3}})$.

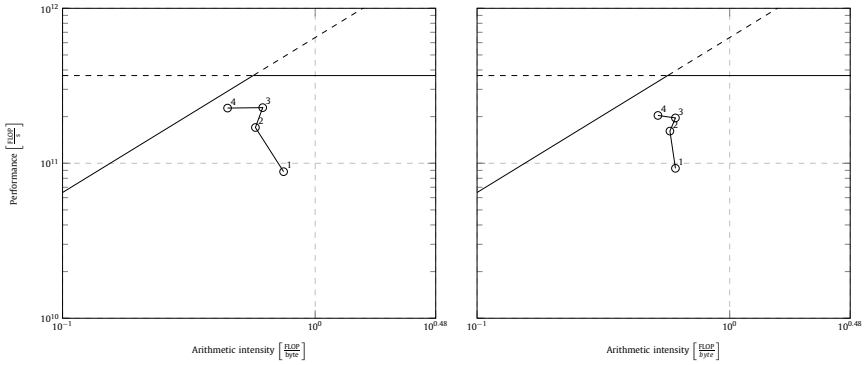
Last, we compare the multi-GPU performance against a parallelized CPU implementation. The parallelization was done with domain decomposition by utilizing the MPI communication protocol, and the performance was measured using up to 48 12-core Intel® Xeon® E5-2690 v3 CPUs. The same GPUs, initial value, and a number of degrees of freedom were used as in the measurements presented in Fig. 9a. This time, we measure the computation speed in how many time units of $\bar{t}$ can be integrated in one second, and present both the CPU and the GPU speeds in Fig. 10. It is observed that one GPU outperforms 144 CPU cores, and in fact, provides 152-fold speedup against one CPU core. The scalability of the CPU implementation exceeds the scalability provided by the GPUs, which we believe to be caused by the lower computational performance of the CPUs, making the computation occupy a larger part of the total time and workload, i.e. causing greater p_t and p_w in eq. (16) and eq. (17), respectively, than on GPUs.

6. Conclusions

We presented a discrete exterior calculus-based numerical time integration method for the three-dimensional Gross-Pitaevskii equation. The spatial domain was discretized with six different natural crystal structures and the method implemented on multiple GPUs. We measured the accuracy provided by the different discretization grids and the computation time performance of the GPU implementation.

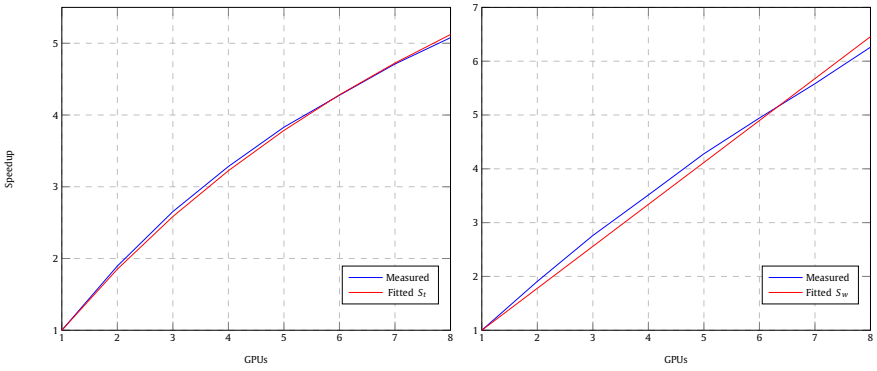
All of the discretization grid types showed to provide better accuracy than the cubic grid. Especially the C15 and BCC grids were proved to be the most accurate ones in all of the experiments, which coincides with previous work showing them to be most numerically isotropic. The cubic grid also provided unreliable simulation results that didn't agree with the predictions yielded by the physics theory, whereas with all of the other grids the results did match the predictions.

The GPU implementation of the method was described with the emphasis on the performance optimizations. In which, we paid particular attention to the memory utilization, as we then showed



(a) BCC (b) C15

Fig. 8. The Roofline models of the computations of four different problem sizes, using the BCC and C15 grids.



(a) Amdahl's law (b) Gustafson's law

Fig. 9. The speedups provided by additional GPUs, measured in (a) $\frac{T_1}{T_{N_{\text{gpu}}}}$ and (b) $\frac{W_{N_{\text{gpu}}}}{W_1}$, including fitted Amdahl's and Gustafson's laws with $p_t = 0.92$ and $p_w = 0.78$, respectively.

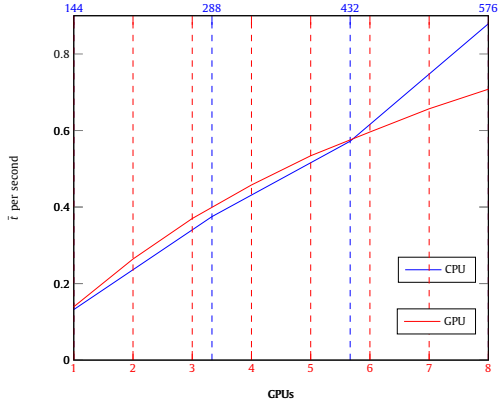


Fig. 10. Computation speeds in $\tilde{t}$ time units integrated per second, as functions of the CPU core and GPU counts.

later in the paper, that the performance is bound by the maximum memory bandwidth of the hardware when the problem size gets large enough. The GPU implementation was able to provide up to 57- and 152-fold speedups against a CPU, depending on hardware. The multi-GPU implementation showed that 92% of the computation time and 78% of the workload has perfect scalability over the number of GPUs.

Most parts of the presented GPU implementation, and the performance optimizations, in particular, are independent of the equation to be integrated. Hence future studies could utilize these parts to other problems as well when the equations are discretized with DEC.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The CPU cluster resources were provided by the CSC - IT Center for Science, owned by the Finnish Ministry of Education and Culture. We would also like to acknowledge the assistance of Dr. Jukka Räbinä in providing the reference CPU implementation.

References

[1] E.P. Gross, Nuovo Cimento 20 (1960) 454–477, https://doi.org/10.1007/BF02731494, https://cds.cern.ch/record/343403.
[2] L.P. Pitaevskii, Sov. Phys. JETP 13 (2) (1961) 451–454.
[3] F. Dalfovo, S. Giorgini, L.P. Pitaevskii, S. Stringari, Rev. Mod. Phys. 71 (3) (1999) 463–512, https://doi.org/10.1103/revmodphys.71.463.
[4] A. Andriati, L. Brito, L. Tomio, A. Gammal, Stability of a Bose condensed mixture on a bubble trap, arXiv:2107.04130, 2021.
[5] A. Geelmuyden, S. Erne, S. Patrick, C. Barenghi, S. Weinfurtner, The sound-ring radiation of expanding vortex clusters, arXiv:2105.11509, 2021.
[6] V. Shukla, S. Nazarenko, Non-equilibrium Bose-Einstein condensation, arXiv:2105.07274, 2021.
[7] P. Bader, S. Blanes, F. Casas, M. Thalhammer, Efficient time integration methods for Gross–Pitaevskii equations with rotation term, arXiv:1910.12097, 2019.

[8] J. Gaidamour, Q. Tang, X. Antoine, Comput. Phys. Commun. 265 (2021) 108007, https://doi.org/10.1016/j.cpc.2021.108007, https://www.sciencedirect.com/science/article/pii/S0010465521001193.
[9] J. Cui, Y. Wang, C. Jiang, Comput. Phys. Commun. 261 (2021) 107767, https://doi.org/10.1016/j.cpc.2020.107767, https://www.sciencedirect.com/science/article/pii/S0010465520303830.
[10] Želimir Marojević, E. Göklü, C. Lämmerzahl, Comput. Phys. Commun. 202 (2016) 216–232, https://doi.org/10.1016/j.cpc.2015.12.004, https://www.sciencedirect.com/science/article/pii/S0010465515004427.
[11] G. Vergez, I. Danaila, S. Auliac, F. Hecht, Comput. Phys. Commun. 209 (2016) 144–162, https://doi.org/10.1016/j.cpc.2016.07.034, https://www.sciencedirect.com/science/article/pii/S0010465516302508.
[12] J. Keranen, E. Koljonen, T. Tarhasaari, L. Kettunen, IEEE Trans. Magn. 40 (2004) 1452–1455.
[13] J. Räbinä, S. Mönkölä, T. Rossi, SIAM J. Sci. Comput. 37 (6) (2015) B834–B854, https://doi.org/10.1137/140988759.
[14] M. Desbrun, A.N. Hirani, M. Leok, J.E. Marsden, Discrete exterior calculus, arXiv:math/0508341, 2005.
[15] M. Desbrun, E. Kanso, Y. Tong, Discrete Differential Forms for Computational Modeling, Birkhäuser, Basel, 2008, pp. 287–324.
[16] P.D. Boom, O. Kosmas, L. Margetts, A. Jivkov, A geometric formulation of linear elasticity based on discrete exterior calculus, arXiv:2104.06000, 2021.
[17] P. Jagad, A. Abukhwejah, M. Mohamed, R. Samtaney, Phys. Fluids 33 (1) (2021) 017114, https://doi.org/10.1063/5.0035981.
[18] M. Salmasi, M. Potter, J. Comput. Phys. 364 (2018) 298–313, https://doi.org/10.1016/j.jcp.2018.03.019, https://www.sciencedirect.com/science/article/pii/S0021999118301773.
[19] L. Silva, C. Batista, I. Roa González, A. Macêdo, W. de Oliveira, S. Melo, J. Comput. Theor. Nanosci. 16 (2019) 3670–3682, https://doi.org/10.1166/jctn.2019.8364.
[20] S. Marburg, in: Computational Acoustics of Noise Propagation in Fluids-Finite and Boundary Element Methods, 2008, pp. 309–332.
[21] W.D. Hillis, Daedalus 121 (1) (1992) 1–15, http://www.jstor.org/stable/20025415.
[22] V. Lončar, A. Balaž, A. Bogojević, S. Škrbić, P. Muruganandam, S.K. Adhikari, Comput. Phys. Commun. 200 (2016) 406–410.
[23] J.R. Schloss, L.J. O'Riordan, J. Open Sour. Softw. 3 (32) (2018) 1037.
[24] B.D. Smith, L.W. Cooke, L.J. LeBlanc, GPU-accelerated solutions of the nonlinear Schrödinger equation, arXiv:2010.15069, 2020.
[25] J.P. Wilson, Comput. Phys. Commun. 235 (2019) 279–292, https://doi.org/10.1016/j.cpc.2018.02.013, https://www.sciencedirect.com/science/article/pii/S0010465518300419.
[26] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, T.J. Purcell, Comput. Graph. Forum 26 (1) (2007) 80–113, https://doi.org/10.1111/j.1467-8659.2007.01012.x, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01012.x.
[27] M.S. Serpa, F.B. Moreira, P.O.A. Navaux, E.H.M. Cruz, M. Diener, D. Griebler, L.G. Fernandes, in: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2019, pp. 233–236.
[28] H. Esqueda, R. Herrera, S. Botello, M.A. Moreles, A geometric description of Discrete Exterior Calculus for general triangulations, arXiv:1802.01158, 2018.
[29] M.S. Mohamed, A.N. Hirani, R. Samtaney, Numerical convergence of Discrete Exterior Calculus on arbitrary surface meshes, arXiv:1802.04506, 2018.
[30] J. Räbinä, On a numerical solution of the Maxwell equations by discrete exterior calculus, 2014.
[31] A. Bossavit, in: W.H.A. Schilders, E.J.W. ter Maten, S.H.M.J. Houben (Eds.), Scientific Computing in Electrical Engineering, Springer, Berlin, Heidelberg, 2004, pp. 128–136.
[32] J. Räbinä, L. Kettunen, S. Mönkölä, T. Rossi, ESAIM: M2AN 52 (3) (2018) 1195–1218, https://doi.org/10.1051/m2an/2018017.
[33] J. Räbinä, P. Kuopanportti, M. Kivioja, M. Möttönen, T. Rossi, Phys. Rev. A 98 (2018), https://doi.org/10.1103/PhysRevA.98.023624.
[34] E. Schulz, G. Tsogtgerel, Discrete Comput. Geom. 63 (2) (2019) 346–376, https://doi.org/10.1007/s00454-019-00159-x.
[35] P.-b. Zhou, Finite Difference Method, Springer, Berlin, Heidelberg, 1993, pp. 63–94.
[36] P. Vingelmann, F.H. Fitzek, CUDA, release: 11.4.2, https://developer.nvidia.com/cuda-toolkit, 2021.
[37] V. Dolean, P. Jolivet, F. Nataf, An introduction to domain decomposition methods - algorithms, theory, and parallel implementation, 2015.
[38] P. Kuopanportti, M. Möttönen, Phys. Rev. A 81 (3) (Mar. 2010), https://doi.org/10.1103/physreva.81.033627.
[39] J. Ortega, W. Rheinboldt, in: J. Ortega, W. Rheinboldt (Eds.), Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, 1970, pp. 181–239, https://www.sciencedirect.com/science/article/pii/B9780125285506500181.
[40] M. Kivioja, J. Räbinä, GPU-accelerated DEC-based Gross-Pitaevskii solver, https://doi.org/10.5281/zenodo.5700296, 2021, https://github.com/markus-kivioja/GpuDecGpe.
[41] M. Möttönen, T. Mizushima, T. Isoshima, M.M. Salomaa, K. Machida, Phys. Rev. A 68 (2003) 023611, https://doi.org/10.1103/PhysRevA.68.023611.
[42] S. Williams, A. Waterman, D.A. Patterson, Roofline: an insightful visual performance model for multicore architectures, https://doi.org/10.1145/1498765.1498785, Apr. 2009.
[43] C. Chraibi, Int. J. Comput. Sci. Info. Technol. 6 (5) (2015) 4412–4416.
[44] J. Gustafson, Commun. ACM 31 (1988) 532–533, https://doi.org/10.1145/42411.42415.