Research Paper

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2021, Vol. 35(1) 60–77

DOI: 10.1177/1094342020959423
journals.sagepub.com/home/hpc
SAGE

A domain-specific language and matrix-free stencil code for investigating electronic properties of Dirac and topological materials

Andreas Pieper[1,2], Georg Hager[2] and Holger Fehske[1]

Abstract

We introduce PVSC-DTM (Parallel Vectorized Stencil Code for Dirac and Topological Materials), a library and code generator based on a domain-specific language tailored to implement the specific stencil-like algorithms that can describe Dirac and topological materials such as graphene and topological insulators in a matrix-free way. The generated hybrid-parallel (MPI+OpenMP) code is fully vectorized using Single Instruction Multiple Data (SIMD) extensions. It is significantly faster than matrix-based approaches on the node level and performs in accordance with the roofline model. We demonstrate the chip-level performance and distributed-memory scalability of basic building blocks such as sparse matrix-(multiple-) vector multiplication on modern multicore CPUs. As an application example, we use the PVSC-DTM scheme to (i) explore the scattering of a Dirac wave on an array of gate-defined quantum dots, to (ii) calculate a bunch of interior eigenvalues for strong topological insulators, and to (iii) discuss the photoemission spectra of a disordered Weyl semimetal.

Keywords

Stencil code, domain-specific language, topological quantum matter

1. Introduction and related work

Dirac-type semimetals and topological insulators are new materials with an enormous application potential in fields ranging from nano-electronics, plasmonics and optics to quantum information and computation. Their striking electronic, spectroscopic, and transport properties result from spin-polarized (chiral), (semi)metallic surface states, which are located in the middle of the spectrum and show linear dispersion to a good approximation. The discovery of such massless Dirac fermions in graphene by Castro Neto et al. (2009), on the surface of topological insulators by Hasan and Kane (2010), and in Weyl semimetals by Xu et al. (2015) has triggered the investigation of Dirac physics. Realizing that certain transport, magnetic and optical properties of solid state systems can be expressed by topological invariants that are insensitive to local perturbations, has largely changed the focus and direction of current condensed matter research from strong correlation to topological aspects (see Chamon et al., 2014).

Whether a material develops distinct topological phases is dictated by the dimension, the lattice structure and associated electronic band structure including the boundary states, and the relevant interactions, all reflected in the system's Hamilton operator and its symmetries. Therefore it is of great interest to determine and analyze the ground-state and spectral properties of paradigmatic model Hamiltonians for topological matter. This can be achieved by means of unbiased numerical approaches.

PVSC-DTM is a highly parallel, vectorized (matrix-free) stencil code for investigating the properties of two-dimensional (2D) graphene and graphene-nanoribbons (GNRs), three-dimensional (3D) topological insulators as well as Weyl semimetals, including also disorder effects, by using modern numerical methods based on matrix polynomials. Due to the complexity of the problem, a considerable amount of computation is required. Thus, one of the design goals of PVSC-DTM was to build highly parallel software that supports the architectural features of modern computer systems, notably SIMD (Single Instruction

[1] Universität Greifswald, Greifswald, Germany
[2] Erlangen Regional Computing Center (RRZE), Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

Corresponding author:
Georg Hager, Erlangen Regional Computing Center (RRZE), Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstr. 1, 91058 Erlangen, Germany.
Email: georg.hager@fau.de

Multiple Data) parallelism, shared-memory thread parallelism, and massively parallel, distributed-memory parallelism. On the compute node level, the development process was guided by performance models to ensure that the relevant bottleneck is saturated. The major methodological advantage compared to existing software packages for similar purposes is that all matrix operations are performed without an explicitly stored matrix, thereby greatly reducing the pressure on the memory interface and opening possibilities for advanced optimizations developed for stencil-type algorithms.

In order to ease the burden on users and still provide the flexibility to adapt the code to different physical setups, a domain-specific language (DSL) was developed that allows for a formulation of the problem without any reference to a specific implementation, let alone optimization. The actual code is generated automatically, including parallelization and blocking optimizations. Although several stencil DSLs have been developed (see, e.g. Schmitt et al., 2014; Tang et al., 2011; Zhang et al., 2017), some even with specific application fields in mind such as in Ragan-Kelley et al. (2013), there is to date no domain-specific approach to generating efficient stencil code for algorithms describing the specific quantum systems mentioned above from a high-level representation. Optimal blocking factors and other optimization strategies are traditionally determined using auto-tuning, which was extensively analyzed in the past by, e.g. Datta et al. (2008, 2009), Kamil et al. (2010), and Basu et al. (2013). Here we calculate optimal blocking factors automatically from machine properties, which makes performance tuning (automatically or manually) on the generated code or within the code generation phase obsolete. The touchstone for performance optimality is whether the sparse matrix-vector multiplication (spMVM) loop can achieve minimal code balance and still utilize a large fraction of the memory bandwidth. In all application cases investigated so far, this was not observed as a restriction or disadvantage.

Temporal blocking strategies have been a subject of intense research over the last two decades (Bandishti et al., 2012; Malas et al., 2017; Wonnacott, 2000). They perform multiple stencil sweeps of in-cache tiles in order to (ideally) decouple from the main memory bottleneck. In unmodified form, these approaches are unsuitable for the applications covered here because spMVM is only a part (albeit an important one) of the whole algorithm. However, blocking optimizations do exist for, e.g. filter diagonalization methods, and can have a similar effect as temporal blocking for pure stencil codes, as shown by Kreutzer et al. (2015, 2018).

This report gives an overview of the physical motivation and describes in detail the implementation of the framework, including the DSL. Performance models are developed to confirm the optimal resource utilization on the chip level and assess the potential of code optimizations, such as spatial blocking and on-the-fly random number generation. Performance comparisons on the node and the highly parallel level with matrix-bound techniques (using the GHOST library) show the benefit of a matrix-free formulation. The code is freely available for download at `http://tiny.cc/PVSC-DTM`.

For the benchmark tests we used two different compute nodes: A dual-socket Intel Xeon E5-2660v2 "Ivy Bridge" (IVB) node with 10 cores per socket and 2.2 GHz of nominal clock speed, and an Intel Xeon E5-2697v4 "Broadwell" (BDW) node with 18 cores per socket and 2.3 GHz of nominal clock speed. In all cases the clock frequency was fixed to the nominal value (i.e., Turbo Boost was not used). The "cluster on die" (CoD) mode was switched off on BDW, so both systems ran with two ccNUMA domains. The maximum achievable per-socket memory bandwidth was 40 Gbyte/sec on IVB and 61 Gbyte/sec on BDW. The Intel C/C++ compiler in version 16.0 was used for compiling the source code.

For all distributed-memory benchmarks we employed the "Emmy" cluster at RRZE (Erlangen Regional Computing Center). This cluster comprises over 500 of the IVB nodes described above, each equipped with 64 GB of RAM and connected via a full nonblocking fat-tree InfiniBand network.

This paper is organized as follows. Section 2 provides typical lattice-model Hamiltonians for graphene nanoribbons with imprinted quantum dots, strong topological insulators, and disordered Weyl semimetals. A matrix-free method and code for the calculation of electronic properties of these topological systems is described in Sec. 3, with a focus on a domain-specific language that serves as an input to a code generator. To validate and benchmark the performance of the numerical approach, the proposed PVSC-DTM scheme is executed for several test cases. Section 4 describes the matrix-polynomial algorithms used for some physical "real-world" applications. Finally, we conclude in Sec. 5.

2. Model Hamiltonians

In this section we specify the microscopic model Hamiltonians under consideration, in a form best suitable for the application of the PVSC-DTM stencil code. The emergence of Dirac-cone physics is demonstrated.

2.1. Graphene

Graphene consists of carbon atoms arranged in a 2D honeycomb lattice structure (see the review by Castro Neto et al., 2009). The honeycomb lattice is not a Bravais lattice, because two neighboring sites are inequivalent from a crystallographic point of view, but can be viewed as a triangular lattice with a two-atom basis, as shown in Goerbig (2011).

Taking into account only nearest-neighbor hopping processes on the honeycomb lattice, the resulting *two-band* structure of pure graphene,

$$\varepsilon_{\pm}(\mathbf{k}) = \pm\left[3 + 2\cos\left(\sqrt{3}k_y a\right) + 4\cos\left(\frac{\sqrt{3}}{2}k_y a\right)\cos\left(\frac{3}{2}k_x a\right)\right]^{1/2}, \quad (1)$$

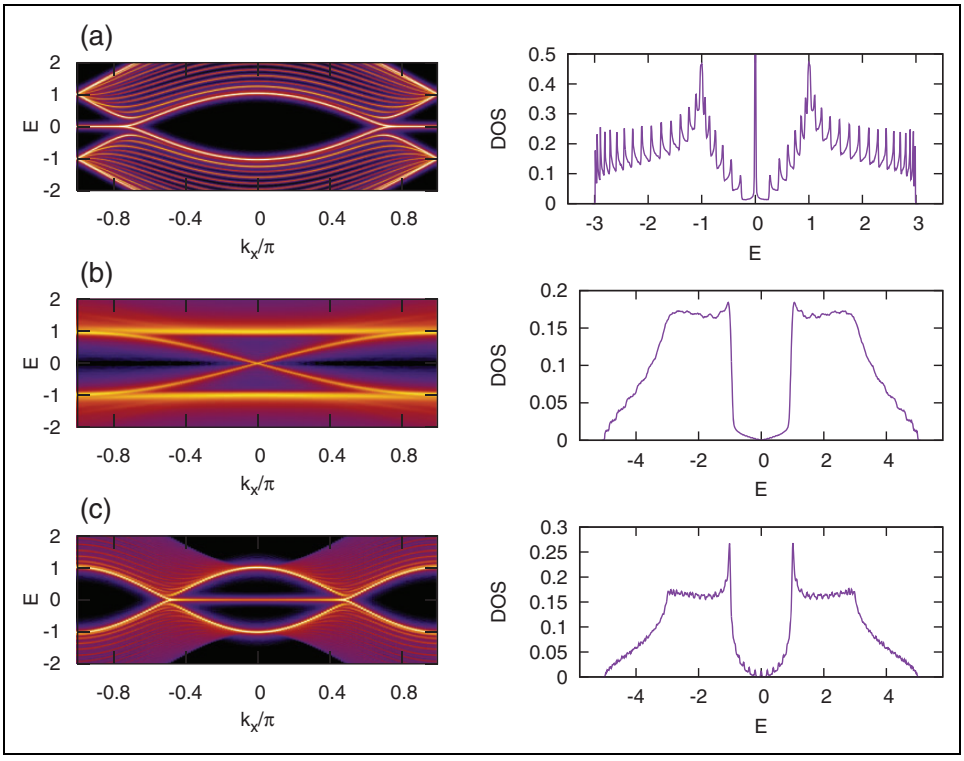

Figure 1. Band dispersion along k_x deduced from the single-particle spectral function $A(\mathbf{k}, E)$ [left panels] and density of states (DOS) [right panels] for the model Hamiltonians (2), (4), and (8). (a) Zigzag GNR with $v_{n,j} = 0$ having 16 "rows" and open boundary conditions (BCs) in y direction (periodic BCs in x direction). (b) Strong TI with $m = 2$, $\Delta_{1/2} = V_n = 0$ on a cuboid with $512 \times 64 \times 8$ sites and periodic BCs. Here the Dirac cone (linear dispersion) near $E = 0$ is due to the surface states. (c) Weyl semimetal on a cuboid with $256 \times 32 \times 32$ sites and open BCs in z direction (periodic BCs in x and y directions).

exhibits an upper (+) anti-bonding π^* band and a lower (−) bonding π band, which touch each other at so-called Dirac points; next to any of those the dispersion becomes linear (see Castro Neto et al., 2009). In the following, we set the lattice constant $a = 1$. The corresponding graphene tight-binding Hamiltonian respects time-inversion symmetry, which implies $\varepsilon(-\mathbf{k}) = \varepsilon(\mathbf{k})$, and if $\mathbf{k}^D$ is the solution for $\varepsilon(\mathbf{k}) = 0$ [which is the Fermi energy E_F for intrinsic (undoped) graphene], so is $-\mathbf{k}^D$, i.e. the Dirac points occur in pairs.

Compared to the band structure of an infinite 2D graphene sheet, the DOS of finite GNRs is characterized by a multitude of Van Hove singularities, as shown by Castro Neto et al. (2009) and Schubert et al. (2009). For zigzag GNRs, the strong signature at $E = 0$ indicates the high degeneracy of edge states, as shown in Figure 1(a). By contrast, armchair GNRs are gapped around $E = 0$; this finite-size gap vanishes when the width of the ribbon tends to infinity.

Particularly with regard to the implementation of the PVSC-DTM, the effective tight-binding Hamiltonian for graphene's π-electrons is brought into the form:

$$H = \sum_{n=1}^{N/4} \left(\Psi^\dagger_{n+\hat{\mathbf{e}}_x} T_x \Psi_n + \Psi^\dagger_{n+\hat{\mathbf{e}}_y} T_y \Psi_n + \text{H.c.} \right) + \sum_{n=1}^{N/4} \Psi^\dagger_n (T_n + V_n) \Psi_n, \tag{2}$$

where Ψ_n is a four-component spinor at site n. Here and in what follows we use units such that $\hbar = 1$ and measure the energy in terms of the carbon-carbon electron transfer integral t; N is the number of lattice sites. Then, in Eq. (2), the first term describes the particle transfer $T_{x,y}$ between neighboring cells (containing now *four* atoms each) in x and y direction, while the second term gives the transfer T_n within the cells. To include the case of (on-site) disorder, we are allowing the potentials $v_{n,j}$ to vary within the cells and from cell to cell. Then 4×4 matrices are

$$T_x = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \; T_y = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$
$$T_n = \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix}, \tag{3}$$
$$V_n = \begin{pmatrix} v_{n,0} & 0 & 0 & 0 \\ 0 & v_{n,1} & 0 & 0 \\ 0 & 0 & v_{n,2} & 0 \\ 0 & 0 & 0 & v_{n,3} \end{pmatrix}.$$

2.2. Topological insulators

The remarkable properties of 3D topological insulators (TIs) result from a particular topology of their band structure, which exhibits gapped (i.e., insulating) bulk and gapless (i.e., metallic) linearly dispersed Dirac surface states (see reviews by Hasan and Kane (2010) and Qi and Zhang (2011)). Bulk-surface correspondence implies, as shown by Fu et al. (2007), that so-called weak TIs (which are less robust against the influence of non-magnetic impurities) feature none or an even number of helical Dirac cones while strong (largely robust) $\mathbb{Z}_2$ TIs have a single Dirac cone.

As a minimal theoretical model for a 3D TI with cubic lattice structure we consider—inspired by the orbitals of strained 3D HgTe or the insulators of the Bi_2Se_3 family, as studied in Sitte et al. (2012)—the following *four-band* Hamiltonian:

$$\begin{aligned} H = &\sum_{n=1}^{N} \Psi_n^\dagger \left(m\Gamma^1 + \Delta_1\Gamma^5 + \Delta_2\Gamma^{15} + V_n\Gamma^0\right)\Psi_n \\ &-\sum_{n=1}^{N}\sum_{j=1}^{3}\left(\Psi_{n+\hat{\mathbf{e}}_j}^\dagger \frac{\Gamma^1 - \mathrm{i}\Gamma^{j+1}}{2}\Psi_n + \mathrm{H.c.}\right), \end{aligned} \tag{4}$$

where Ψ_n is a four-component spinor at site n. The Hamiltonian is expressed in terms of the five Dirac matrices Γ^a, $\Gamma^{(1,2,3,4,5)} = (1\otimes s_z, -\sigma_y\otimes s_x, \sigma_x\otimes s_x, -1\otimes s_y, \sigma_z\otimes s_x)$, and their 10 commutators $\Gamma^{ab} = [\Gamma^a, \Gamma^b]/2i$, which satisfy the Clifford algebra, $\{\Gamma^a, \Gamma^b\} = 2\delta_{a,b}\Gamma^0$, with Γ^0 being the identity 1_4 and s_i (σ_i) the Pauli matrices referring to orbital (spin) space (see Pieper and Fehske, 2016; Schubert et al., 2012). Hence, H constitutes a complex, sparse, banded matrix with seven subdiagonals of small dense blocks of size 4×4. The corresponding tight-binding 4×4 band matrix reads:

$$\begin{aligned} \varepsilon(\mathbf{k}) = &-\sum_{j=1}^{3}\left(\Gamma^1 - \mathrm{i}\Gamma^{j+1}\right)\cos k_j \\ &+ m\Gamma^1 + \Delta_1\Gamma^5 + \Delta_2\Gamma^{15}. \end{aligned} \tag{5}$$

The parameter m can be used to tune the band structure: For $|m| < 1$, a weak TI with two Dirac cones per surface arises, whereas for $1 < |m| < 3$, a strong TI results, with a single Dirac cone per surface (see Figure 1(b)). In the case that $|m| > 3$ we have a conventional band insulator. External magnetic fields cause finite Δ_1 and Δ_2, which will break the inversion symmetry. Δ_1, in addition, breaks the time-inversion symmetry.

We now describe for the TI problem how the density of states (DOS) and the single-particle spectral function $A(\mathbf{k}, E)$ depicted in Figure 1 is obtained using state-of-the art exact diagonalization and kernel polynomial methods that were described in Weiße and Fehske (2008) and Weiße et al. (2006), respectively. For a given sample geometry (and disorder realization), we can calculate $\{|l\rangle\}$, the two-fold Kramers degenerate eigenstates of H (5). Those can be visualized in momentum and energy space, via the momentum-resolved spectral function

$$A(\mathbf{k}, E) = \sum_{\nu=1}^{4}\sum_{l=1}^{4N} |\langle l|\psi(\mathbf{k},\nu)\rangle|^2 \delta(E - E_l) \tag{6}$$

and the density of states (DOS)

$$\begin{aligned} \mathrm{DOS}(E) &= \sum_{l=1}^{4N}\delta(E - E_l) \\ &= \frac{1}{N}\sum_{n}\sum_{\nu=1}^{4}\sum_{l=1}^{4N} |\langle l|\psi(\mathbf{r}_n,\nu)\rangle|^2\delta(E - E_l), \end{aligned} \tag{7}$$

even in the case of disorder. Here, $\langle \mathbf{e}_n^{(p)}\otimes\mathbf{e}_\nu^{(b)}|\psi(\mathbf{k},\,\nu')\rangle = \exp(\mathrm{i}\langle\mathbf{k}|\mathbf{e}_n^{(p)}\rangle)\delta_{\nu\nu'}$ is a Bloch state and $\langle|\mathbf{e}_n^{(p)}\otimes\mathbf{e}_\nu^{(b)}|\psi(\mathbf{r}_{n'},\,\nu')\rangle = \delta_{nn'}\delta_{\nu\nu'}$ is a Wannier state, where $|\mathbf{e}_n^{(p)}\rangle$ and $|\mathbf{e}_\nu^{(b)}\rangle$ denote the canonical basis vectors of position and band index space, respectively (see Schubert et al., 2012).

For the model (4) with $m = 2$ (and $V_n = 0$, $\Delta_{1/2} = 0$), bulk states occur for energies $|E| \geq 1$. Moreover, subgap surface states develop, forming a Dirac cone located at the surface momentum $\mathbf{k}^D = (0, 0)$, as shown in Figure 1(b). The latter states determine the striking electronic properties of TIs.

2.3. Weyl semimetals

The Weyl semimetallic phase, which can be observed, e.g. in TaAS (see Xu et al., 2015), is characterized by a set of linear-dispersive band touching points of two adjacent bands, the so-called Weyl nodes. The real-space Weyl points are associated with chiral fermions, which behave in momentum space like magnetic monopoles. Unlike the 2D Dirac points in graphene, the 3D Weyl nodes are protected by the symmetry of the band structure and, as long as there is no translational-symmetry-breaking intervalley-mixing between different Weyl nodes, the Weyl semimetal is robust against perturbations, as was shown by Yang et al. (2011). In this way a Weyl semimetal hosts, like a TI, metallic topological surface states (arising from bulk topological invariants). However, while the topological surface states of TIs give rise to a closed Fermi surface (in momentum space), the surface-state band structure of Weyl semimetals is more exotic; it forms open curves, the so-called Fermi arcs, which terminate on the bulk Weyl points (see Wan et al., 2011).

The minimal theoretical models for topological Weyl semimetals have been reviewed quite recently by McCormick et al. (2017). Here we consider the following 3D lattice *two-band* Hamiltonian,

$$\begin{aligned} H = &\sum_{n=1}^{N}\left(\Psi_{n+\hat{\mathbf{e}}_x}^\dagger\frac{\sigma_x}{2}\Psi_n + \sum_{j=y,z}\Psi_{n+\hat{\mathbf{e}}_j}^\dagger\frac{\sigma_x + \mathrm{i}\sigma_j}{2}\Psi_n + \mathrm{H.c.}\right) \\ &+\sum_{n=1}^{N}\Psi_n^\dagger[V_n - \sigma_x(2 + \cos k_0)]\Psi_n, \end{aligned} \tag{8}$$

where Ψ_n is now a two-component spinor and σ_j are the Pauli matrices (again, the lattice constant is set to unity, just as the transfer element). In momentum space $[\mathbf{k} = (k_x, k_y, k_z)]$, the (2×2) band matrix takes the form $(V_n = 0)$

$$\begin{aligned}\varepsilon(\mathbf{k}) = \sigma_x(\cos k_x - \cos k_0 + \cos k_y + \cos k_z - 2) \\ +\sigma_y \sin k_y + \sigma_z \sin k_z,\end{aligned} \tag{9}$$

developing two Weyl nodes at momenta $\mathbf{k}_{\pm}^{W} = (\pm k_0, 0, 0)$ with $k_0 = \pi/2$, as seen in Figure 1(c) (Hasan et al., 2017).

3. Matrix-free code for topological systems with Dirac cones

In general, topological materials have a rather complex lattice structure, although it is so regular that a matrix-free formulation of spMVM and similar kernels is feasible due to the stencil-like neighbor relations. The lattice is always periodic (apart from disorder effects), but particle transfer integrals or interactions vary widely among materials. In other words, the resulting stencil geometry depends strongly on the physics, and in a way that makes it impossible to set up optimal code for all possible situations in advance although the core algorithm is always a stencil-like update scheme. The required blocking strategies for optimal code performance also vary with the stencil shape. Consequently, it is worthwhile to generate the code for a particular physical setup. This allows to hard-code performance-relevant features and takes out a lot of uncertainty about compiler optimizations. In this section we describe some of the design goals and implementation details of our matrix-free code, including the DSL, representative benchmark cases, and performance results.

3.1. Preliminary considerations

Many numerical algorithms that can describe quantum systems, such as eigenvalue solvers or methods for computing spectral properties, such as the Kernel Polynomial Method (KPM) (reviewed in Weiße et al., 2006), require the multiplication of large sparse matrices with one or more RHS vectors as a time-consuming component. If the matrix is stored explicitly in memory and special structures such as symmetry and dense subblocks are not exploited, The data transfer between the CPU and the main memory is the performance-limiting bottleneck. An upper limit for the performance of a typical linear algebra building block such as spMVM can thus be easily calculated by means of the naive roofline model, which was popularized by Williams et al. (2009) and applied in many different contexts, including stencils (Datta et al., 2009), fluid dynamics (Randles et al., 2013), and sparse linear algebra (Gropp et al., 2000), among many others:

$$P \leq \min\left(P_{\text{peak}}, b_{\text{S}}/B_{\text{c}}\right) . \tag{10}$$

This model assumes that the performance of a loop is either limited by the computational peak performance of the CPU (P_{peak}) or by the maximum performance allowed by memory data transfers ($b_{\text{S}}/B_{\text{c}}$), whichever is more stringent. In case of spMVM and similar algorithms on any modern multicore CPU, the former is much larger than the latter, so we can safely ignore it here. b_{S} is the achievable main memory bandwidth in bytes/s; it can be determined by a suitable benchmark, such as STREAM by McCalpin (1991–2007). B_{c} is the *code balance*, i.e. the ratio of the required data volume through the memory interface (in bytes) and the amount of work (usually floating-point operations, but any valid "work" metric will do). Clearly, $b_{\text{S}}/B_{\text{c}}$ is then an upper limit for the expected performance of the loop. In practice one can determine the code balance by code inspection and an analysis of data access locality. Whenever the data traffic cannot be calculated accurately, e.g. because some indirect and unpredictable access is involved, it is often possible to give at least a lower limit for B_{c} and thus an absolute upper limit for the performance. A "good" code in the context of the roofline model is a code whose performance is near the limit. Once this has been achieved, any optimization that lowers B_{c} will increase the performance accordingly. Many refinements of the model have been developed to make it more accurate in situations where the bottleneck is not so clearly identified, e.g. by Ilic et al. (2014) and Stengel et al. (2015).

Depending on the processor architecture, SIMD vectorization, i.e. using data-parallel instructions to carry out multiple operations in parallel on short vectors, may be required to achieve memory bandwidth saturation even with a code that has a rather low code balance. This happens when the single core is too slow with scalar, i.e. non-SIMD code, so that even the combined demand of all cores for data does not exert enough "pressure" on the memory interface. Hence, SIMD vectorization was given special attention in our framework. See below for details.

It was first shown by Gropp et al. (2000) that the minimal code balance of spMVM for double precision, real matrices in CRS format and a 32-bit index is $6\ \text{bytes}/\text{flop}$, leading to memory-bound execution if the matrix does not fit into a cache. A matrix-free formulation can greatly reduce the demand for data and leads, in case of many topological materials, to stencil-like update schemes. Even if some of the terms in the operators require variable coefficients, getting rid of the matrix data still has a notable effect. Although the resulting code is limited by memory bandwidth as well, the code balance can be very low depending on the particular stencil shape and on whether layer conditions (LCs) are satisfied. The concept of LCs was conceived by Rivera and Tseng (2000) and applied in the context of advanced analytic performance models by Stengel et al. (2015). In the following we briefly describe the optimizations that were taken into account, using a simple five-point stencil as an example.

Listing 1 shows one update sweep of this code, i.e. one complete update of one LHS vector. In a matrix-bound

Listing 1. Two-dimensional five-point stencil sweep with one RHS and one LHS vector. The highlighted elements must come from cache for optimal code balance.

```
double *x, *y; // RHS/LHS vector data
int imax,kmax; // grid size [0:imax]x[0:kmax]

for(int k=1; k<kmax; ++k)
 for(int i=1; i<imax; ++i)
  y[i+k*(imax+1)] =
    c1*x[(i+1)+k*(imax+1)] + c2*x[(i-1)+k*(imax+1)]
  + c3*x[i+(k+1)*(imax+1)] + c4*x[i+(k-1)*(imax+1)]
  + noise[i+k*(imax+1)];
```

formulation the coefficients `c1,...,c4` would be stored in memory as separate, explicit arrays. In addition to the RHS vector the array `noise[]` is read, implementing a random potential. As is customary with stencil algorithms we use the lattice-site update (LUP) as the principal unit of work, which allows us to decouple the analysis from the actual number of flops executed in the loop body.

The minimum code balance of this loop nest for data in memory is $B_c = (16 + 8 + 8)$ bytes/LUP $= 32$ bytes/LUP, because each LHS element must be updated in memory (16 bytes), and each RHS and noise element must be loaded (eight bytes each). If nontemporal stores can be used for `y[]`, the code balance reduces to 24 bytes/LUP because the write-allocate transfers do not occur and data is written to memory directly. The minimum code balance can only be attained, however, if the highlighted RHS elements do not have to be loaded from memory. This LC is satisfied if at least three successive rows of `x[]` fit into the cache. Assuming that the `noise[]` array and the LHS also require one row of cache space, the condition reads:

$$5 \times \text{imax} \times 8 \text{ bytes} < C \,, \tag{11}$$

where C is the available cache size in bytes per thread. In multi-threaded execution with outer loop parallelization via OpenMP, each thread must have its LC fulfilled. If the condition is broken, the inner loop can be blocked with a block size of `ib` and the condition will be satisfied if

$$\text{ib} < \frac{C}{40\,\text{bytes}} \,. \tag{12}$$

If the blocking is done for a cache level that is shared among the threads in the team, the LC gets more and more stringent the more threads are used. For best single-threaded performance it is advisable to block for an inner cache, i.e. L1 or L2. See Stengel et al. (2015) for more details. Our production code determines the optimal block size according to (12).

The `noise[]` array is a significant contribution to the code balance. However, its contents are static and can be generated on the fly, trading arithmetic effort for memory traffic. The generation of random numbers should certainly be fast and vectorizable, so as to not cause too much overhead. See the section on random number generation below for details.

Listing 2. Two-dimensional five-point stencil sweep with n_b RHS and LHS vectors. SIMD vectorization across RHS and LHS vectors is possible and efficient if the vector storage order can be chosen as shown.

```
double *x, *y; // RHS/LHS vector data
int imax,kmax; // grid size [0:imax]x[0:kmax]

for(int k=1; k<kmax; ++k)
 for(int i=1; i<imax; ++i)
  for(int s=0; s<nb; ++s)
   y[s+nb*i+nb*k*(imax+1)] =
     c1*x[s+nb*(i+1)+nb*k*(imax+1)]
   + c2*x[s+nb*(i-1)+nb*k*(imax+1)]
   + c3*x[s+nb*i+nb*(k+1)*(imax+1)]
   + c4*x[s+nb*i+nb*(k-1)*(imax+1)]
   + noise[s+nb*i+nb*k*(imax+1)];
```

Some algorithmic variants require the concurrent, independent execution of stencil updates on multiple source and target vectors. Although SIMD vectorization is easily possible even with a single update by leveraging the data parallelism along the inner dimension, a more efficient option exists for multiple concurrent updates: If the vectors can be stored in an interleaved way, i.e. with the leading dimension going across vectors, vectorization along this dimension is straightforward if the number of vectors is large compared to the SIMD width. As opposed to the traditional scheme, perfect data alignment can be achieved (if this is required) and no shuffling of data in SIMD registers is necessary for optimal register reuse. See Listing 2 for an example using a simple five-point stencil. The considerations about LCs do not change apart from the fact that now each RHS vector needs to have its own LC fulfilled. Condition (12) is thus modified to

$$\text{ib} < \frac{C}{5 \times n_b \times 8\,\text{bytes}} \tag{13}$$

if `n_b` is the number of concurrent updates and the `noise[]` arrays have to be loaded from memory.

In summary, code generation in our framework has two goals: produce a spMVM routine with minimal code balance by cache-adapted loop blocking, and produce SIMD-vectorized code in order to be able to address the memory bandwidth bottleneck in all relevant cases.

3.2. Domain-specific language

In order to provide a maximum amount of flexibility to users and still guarantee optimal code, a DSL was constructed which is used to define the physical problem at hand. A precompiler written in Python then generates OpenMP-parallel C code for the sparse matrix-vector multiplication (a "lattice sweep"), which can be handled by a standard compiler. In the following we describe the DSL in detail by example.

The source code for the DSL program resides in a text file. The code begins with a specification of the problem dimensionality (2D/3D) and the basis size:

```
dim 2
size 4
```

The stencil coefficients can take various forms: *constant*, *variable*, or *random*. The number of coefficients of each kind is set by the keywords `n_coeff_*`, where "*" is one of the three options. For example, in case of four variable coefficients:

```
n_coeff_variable 4
```

The command `nn` with two or three arguments (depending on the `dim` parameter) and the following `size` lines define a sparse coefficient matrix to a neighboring lattice block at the offset defined by the arguments of nn. Multiple entries in a line are separated by ";". Optionally the first entry begins with "`l`" followed by the row index. A single block entry is written as "`{column index}|{value}`" for a fixed value, or as "`{column index}| {type}|{type index or value}`" for a different type. This is a simple example for a coefficient matrix one lattice position to the left of the current position (set by `nn -1 0`) and a fixed entry of value -1 at position (0,1) and another entry of value -1 at position (3,2):

```
nn -1 0
l0;      1|-1
l1;
l2;
l3;              2|-1
```

Note that all indexing is zero-based. The following coefficient types are allowed:

`f` Fixed coefficient, hard-coded (default type). This means that an entry of "`1|-1`" can also be written as "`1|f|-1`". It will be hard-coded into the generated C code.

`c` Constant coefficient per lattice site, read from an array of length `n_coeff_const`. For example, "`1|c|2`" means that the coefficient used will be `coeff_c[2]`. The coefficient array can be changed at runtime if required, or preset in the DSL source code. For example, the line

```
coeff_const_default 1 2 -0.5
```

will initialize the array `coeff_c[]` with the specified values.

`v` Variable coefficient per lattice site, read from an array of length `n_coeff_variable` per lattice site.

`r` Random coefficient per lattice site, read from an array of length `n_coeff_rand` per lattice site.

Listing 3. DSL source for a graphene stencil.

```
dim 2
size 4
n_coeff_variable 4

nn -1 0
l0;    1|-1
l1;
l2;
l3;          2|-1

nn 0 -1
l0;                 3|-1
l1;
l2;
l3;
```

```
nn 0 0
l0; 0|v|0; 1|-1
l1; 0|-1;  1|v|1; 2|-1
l2; 1|-1;  2|v|2; 3|-1
l3;        2|-1;  3|v|3

nn 0 1
l0;
l1;
l2;
l3; 0|-1

nn 1 0
l0;
l1; 0|-1
l2;                 3|-1
l3;
```

Listing 4. Generated matrix-vector multiplication code for the graphene stencil (shortened).

```
#pragma omp for schedule(static,1) nowait
for(int i2=sys->b_y; i2<sys->n_y+sys->b_y; i2++){
    // y loop
 for(int i1=i1_0; i1<i1_end; i1++){ // z loop
  int j = i1 +  ldz * ( i2 + ldy * i3);
  int i  = 4 * j;
  #pragma vector aligned
  for(int k=0; k<4; k++) {  // vector block loop
   y[(i+0)*4+k] =  scale_z[k] * y[(i+0)*4+k] +
       scale_h[k]
    * ( -shift_h[k] * x[ (i+0)*4+k ]
    -1. * x[(i+1+4*(0+ldz*(-1)))*4+k]
    -1. * x[(i+3+4*(-1+ldz*(0)))*4+k]
    -1. * x[(i+1+4*(0+ldz*(0))*4+k]
    +coeff_v[j*ldv+0] * x[(i+0+4*(0+ldz*(0)))*4+k
        ]);
   y[(i+1)*4+k] = ...;
   y[(i+2)*4+k] = ...;
   y[(i+3)*4+k] = ...;
  }}
  if( p->dot_xx || p->dot_xy || p->dot_yy ) {
  for(int i1=i1_0; i1<i1_end; i1++) {
   int j = i1 +  ldz * ( i2 + ldy * i3);
   int i  = 4 * j;
   #pragma  vector aligned
   for(int k=0; k<4; k++) {
    xx[k]+=x[(i+0)*4+k]*x[(i+0)*4+k]+x[(i+1)*4+k]*
        x[(i+1)*4+k]+x[(i+2)*4+k]*x[(i+2)*4+k]+x[(
        i+3)*4+k]*x[(i+3)*4+k];
    xy[k]+=...;
    yy[k]+=...;
}}}}
```

In Listing 3 we show a complete example for a 2D graphene stencil with variable coefficients on the diagonal, while Listing 4 shows the generated C source code for the spMVM. Note that spatial blocking is the only explicit optimization done by the code generator. We rely on the compiler to produce SIMD-vectorized code from the C source, which is usually possible by giving it sufficient information, particularly about non-aliasing of pointers.[1] Coefficient arrays and execution parameters (such as, e.g. the grid size) can be configured and changed at runtime. The code repository[2] contains numerous examples that

demonstrate the DSL usage and how the generated source code can be embedded into algorithms.

3.3. Random number generator

In some physically relevant cases, the Hamiltonian matrix has a diagonal, random component. These random numbers are usually stored in a constant array to be loaded during the matrix-vector multiplication step. At double precision this leads to an increase in code balance by 8 bytes/LUP, which can be entirely saved by generating the random numbers on the fly using a fast random number generator (RNG). Considering that the stencil update schemes studied here can run at several billions of lattice site updates per second on modern server processors, a suitable RNG must be able to produce random numbers at a comparable rate. This is not possible with standard library-based implementations such as, e.g. `drand48()`, but faster and (in terms of quality) better options do exist. The RNG code should be inlined with the actual spMVM or at least be available as a function that generates long sequences of random numbers in order to avoid hazardous call overhead.

The standard type of RNG used in scientific computing is the linear congruential generator (LCG), which calculates $x_{i+1} = (ax_i + b) \bmod m$ and casts the result to a floating-point number. The numbers a, b, and m parameterize the generator; for efficiency reasons one can choose m to be a power of two (e.g., $m = 2^{48}$ in `drand48()`), but such simple methods fail the statistical tests of the popular TESTU01 suite devised by L'Ecuyer and Simard (2007). However, if there are no particular quality requirements (i.e., if only "some randomness" is asked for), they may still be of value. Despite the nonresolvable dependency of x_{i+1} on x_i, which appears to rule out SIMD vectorization, LCGs can be vectorized if a number of independent random number sequences is needed and if the SIMD instruction set of the hardware allows for the relevant operations (e.g., SIMD-parallel addition, multiplication, and modulo on unsigned integer operands with the required number of bits).

A more modern and similarly fast approach to RNGs are the *xorshift* generators by Marsaglia (2003). In the simplest case they work by a sequence of XOR mask operations of the seed with a bit-shifted version of itself: $x = x \ll a$; $x = x \gg b$; $x = x \ll c$. Improved versions like the *xorshift128+* by Vigna (2017) pass all statistical tests of the "SmallCrush" suite in TESTU01. Table 1 shows a performance comparison of different RNGs on one socket of the IVB and BDW systems, respectively.

The speedup between IVB and BDW is particularly large for the xorshift generators because the AVX2 instruction set on BDW supports SIMD-parallel bit shifting operations, which are not available in AVX. For reference, we have included a SIMD-vectorized Mersenne Twister RNG (SFMT19937), which is available in Intel's Math Kernel Library (MKL). Note that the purpose of Table 1 is not to compare the performance of different RNG algorithms but to give an impression of how expensive random number generation is compared to the pure spMVM operation. To this end, the GRN/sec numbers should be compared to the GLUP/s stencil performance numbers in the following sections.

Table 1. Performance comparison of the LCG32 RNG, four xorshift generators of different sophistication, and the SFMT19937 generator available in the Intel MKL. Performance numbers are given in billions of random numbers per second (GRN/sec) on one socket (10 or 18 cores, respectively). The benchmark consists in the (repeated) computation of 2000 double-precision random numbers with uniform distribution. For each generator, the second column lists the failed SmallCrush tests of the TESTU01 suite. These particular test results were taken with the Intel C compiler version 17.0 update 5. The benchmarks are available in the code repository.

RNG (loop unrolling)	Failed SmallCrush	Perf. IVB [GRN/sec]	Perf. BDW [GRN/sec]
lgc32 (32)	1,2,3,4,5,6,7,8,9,10	19.3	28.8
xorshift32 (32)	1,3,4,8,9,10	10.1	28.8
xorshift128 (16)	6	8.29	25.4
xorshift64_long (16)	8	8.26	31.7
xorshift128plus_long (16)	—	6.34	20.7
Intel MKL SFMT19937	—	7.72	22.0

Listing 5. Using an "out-of-band" fast RNG to save memory data traffic.

```
for(int k=1; k<kmax; ++k) {
  for(int i=1; i<imax; ++i)
    random[i] = ...; // fast RNG
  for(int i=1; i<imax; ++i)
    y[i+k*(imax+1)] =
      c1*x[(i+1)+k*(imax+1)] + c2*x[(i-1)+k*(imax+1)]
    + c3*x[i+(k+1)*(imax+1)] + c4*x[i+(k-1)*(imax+1)]
    + random[i];
}
```

Whether a particular RNG impacts the performance of the matrix-free spMVM step depends on the application. In the cases we investigate here, a whole row of random numbers can be generated in advance before each inner loop traversal (see Listing 5) without a significant performance degradation in the bandwidth-saturated case. Fusing the RNG with the inner loop is a possible optimization that would, however, not change the roofline limits but at best lead to faster bandwidth saturation as the number of cores goes up.

3.4. Geometry

The current implementation of PVSC-DTM supports cuboid domains of arbitrary size (only limited by memory capacity) with 3D domain decomposition using MPI, and OpenMP-based multithreading on the subdomain level. For each spatial dimension, periodic BCs can be configured separately as needed. Variable-coefficient arrays and initial vectors can be preset via a user-defined callback function with the following interface:

```
double (*fnc)(long *r, int k, \
void *parm, void *seed);
```

This function must expect the following input parameters:

```
{x=r[0], y=r[1], z=r[2], \
basis_place=r[3], vector_block_index=k}
```

It returns the respective vector entry as a double-precision number. The pointer parm is handed down from the vector initial call and allows for configuring specific options. The pointer seed is a reference to a 128-bit process- and thread-local random seed.

Finally a vector block will be initialized by calling the function:

```
pvsc_vector_from_func(pvsc_vector *vec, \
pvsc_vector_func_ptr *fnc, void *parm);
```

This mechanism lets the user define a generalized initial function with optionally free parameters. In addition, a thread-local random seed for optional random functions is available in the initialization function, which enables a fully parallelized initialization of vectors.

3.5. Benchmarks

In order to validate the performance claims of our matrix-free implementation and optimization of random number generation we ran several test cases on the benchmark systems described in the introduction. Performance is quantified in billions of lattice site updates per second (GLUPs/sec). For all stencil update kernels (spMVMs) with constant coefficients studied here, the minimal code balance is 24 bytes/LUP with on-the-fly RNGs (see Listing 5) and 32 bytes/LUP with random numbers stored as constant arrays. The roofline model thus predicts bandwidth-bound per-socket upper performance limits of 1.67 GLUPs/sec on IVB and 2.5 GLUPs/sec on BDW.

Figures 2 and 3 show the performance scaling of the spMVM with a 3D 7-point stencil on one socket of the benchmark systems. On IVB, the "fallback" kernel, which uses explicitly stored random numbers, saturates the memory bandwidth with eight cores at about 95% of the achievable bandwidth (black solid line). The kernel without random numbers (labeled "const. coeff.") marks a practical upper performance limit. It also saturates at about the same bandwidth (and thus at 33% higher performance), with a very slight additional speedup from SMT (Hyper-Threading). As expected, the versions with on-the-fly RNGs are somewhat slower on the core level due to the increased amount of work, which, in case of the xorshift variants, leads to a lower performance than for the fallback variant up to seven cores, and non-saturation when only physical cores are used. SMT can close this gap by filling pipeline bubbles

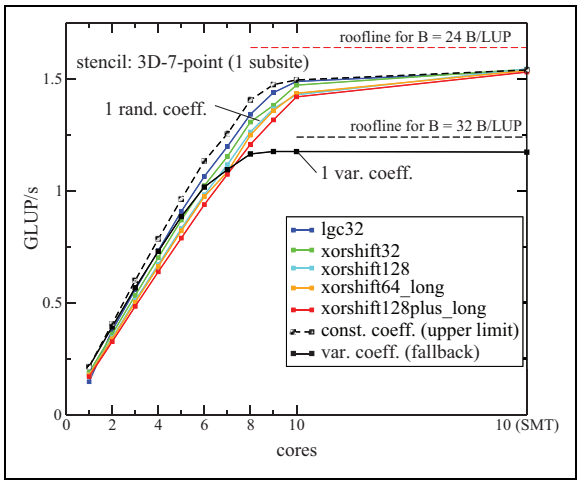

Figure 2. Performance scaling of a spMVM kernel using a constant-coefficient 3D 7-point stencil problem with (as in Listing 5) and without on-the-fly RNG on one IVB socket (10 cores) and with SMT (2 threads per core). The dashed line shows the performance without a random potential, whereas the filled black squares (fallback) show the result with random numbers read from memory. All other data sets were obtained with different on-the-fly RNGs ($n_b = 1$, system size 512^3).

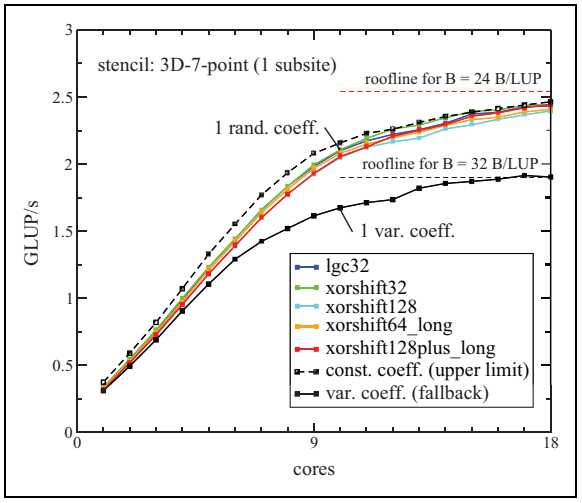

Figure 3. Performance scaling as in Figure 2 but on one BDW socket (18 cores).

on the core level, and all RNG versions end up at exactly the same performance with 20 SMT threads. On BDW the full-socket situation is similar, but all versions come closer to the practical bandwidth limit than on IVB, and the fallback variant is slower than all RNG versions at all core counts.

The bottom line is that even the most "expensive" on-the-fly RNG allows memory bandwidth saturation on both architectures, that the roofline model predictions are quite accurate, that the automatic spatial blocking in PVSC-DTM works as intended and yields the optimal in-memory code balance, and that the elimination of the stored random number stream causes the expected speedup even with high-quality RNGs.

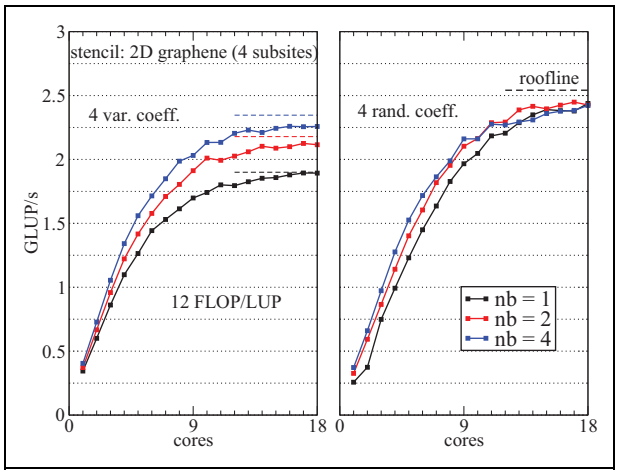

Figure 4. Performance scaling of spMVM with graphene stencil kernels on blocks of vectors of size $n_b = 1$, 2, and 4 with variable coefficients (left) and on-the-fly RNGs (right) on BDW (system size $8000 \times 8000/n_b$).

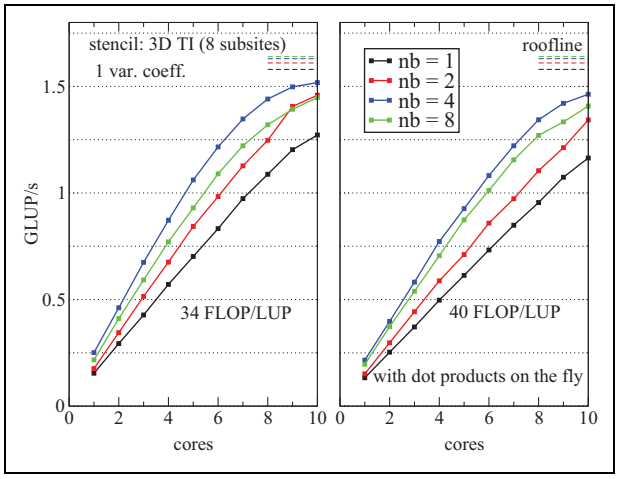

Figure 5. Performance scaling of the stencil spMVM kernel of a 3D TI model without (left) and with (right) on-the-fly dot products for $n_b = 1$, 2, 4, and 8 on IVB (system size $256^2 \times 256/n_b$).

Figure 4 shows a performance comparison of stored random numbers and on-the-fly RNG for a 2D graphene application with four subsites, a block vector size n_b of 1, 2, and 4, and four variable coefficients. The code balance goes down from 32 bytes/LUP to 28 bytes/LUP and finally to 26 bytes/LUP when going from $n_b = 1$ to 2 and 4, approaching the limit of 24 bytes/LUP at $n_b \to \infty$. With on-the-fly RNGs substituting the variable-coefficient arrays this balance is achieved for any n_b, which is shown in the right panel of Figure 4. It can also be observed in the data that the improved SIMD vectorization with $n_b > 1$ speeds up the code measurably in the nonsaturated regime, but this advantage vanishes close to saturation because the data transfer becomes the only bottleneck.

Figures 5 and 6 show the performance of the stencil kernels of a 3D TI model with different n_b on IVB and BDW. Two versions are shown for each architecture: The

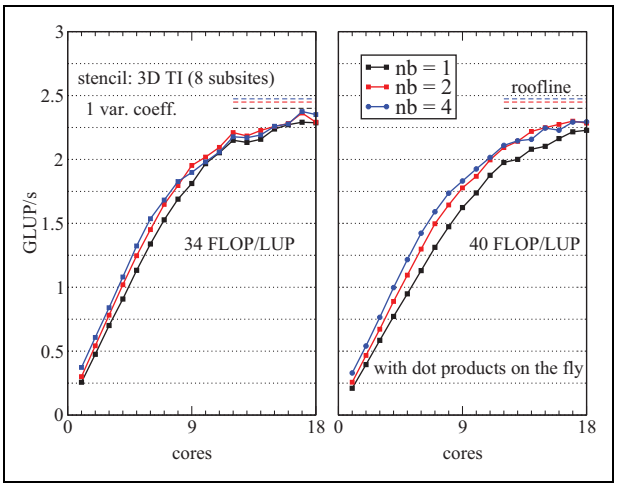

Figure 6. Performance scaling of the stencil spMVM kernel of a 3D TI model without (left) and with (right) on-the-fly dot products for $n_b = 1$, 2, and 4 on BDW (system size $256^2 \times 256/n_b$).

standard one and an optimized version with dot products fused into the stencil kernel, increasing the number of flops per update by six. The code balance for TI stencils is lower than for graphene or the 7-point stencil, hence more cores are required for bandwidth saturation.

At larger n_b the loop body becomes more complicated, and the benefit of SIMD vectorization may be compensated by a more inefficient in-core execution due to register shortage and less effective out-of-order execution. This can be seen on IVB at $n_b = 8$, where the available number of cores is too small to reach saturation, as opposed to $n_b = 4$, where the SIMD width matches the number of block vectors.

Calculating dot products on the fly has a rather small impact on performance (less that 15%), which on BDW vanishes at saturation because of its generally lower machine balance. Still, overall the roofline model provides a good estimate of the expected socket-level performance of our matrix-free codes even for topological insulators. Note, however, that the saturation properties depend on many factors, such as the number of cores per socket, the memory bandwidth, the clock speed, and the SIMD width. An accurate prediction of speedup versus the number of cores would require a more advanced performance model, such as the ECM model described in Stengel et al. (2015).

In Figures 7 and 8 we compare PVSC-DTM with the GHOST library developed by Kreutzer et al. (2017) using a strong scaling TI test case on the Emmy cluster. One MPI process was bound to each socket, with OpenMP parallelization (10 threads) across the cores. Both PVSC-DTM and GHOST were used in "vector mode," i.e. without overlap between communication and computation. GHOST always uses explicitly stored matrices, which is why PVSC-DTM not only has the expected performance advantage due to its matrix-free algorithms but also requires less hardware to handle a given problem size. The maximum number of nodes was chosen such that a maximum

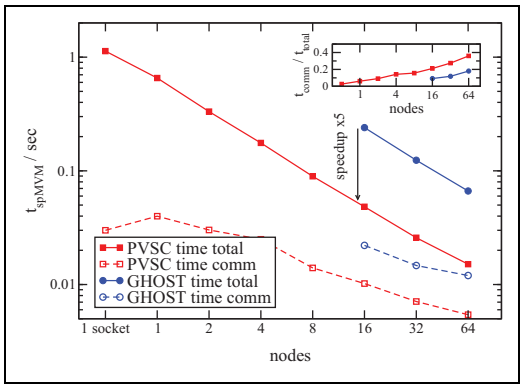

Figure 7. Runtime of spMVM ($n_b = 1$) for a strong scaling test case of the TI model (system size 608^3) with 200 iterations on the Emmy cluster, comparing PVSC-DTM with the GHOST library. The dashed lines show the communication time only. Inset: ratio of communication time vs. total runtime. All codes were run with one MPI process per socket (10 cores) and 10 OpenMP threads per process.

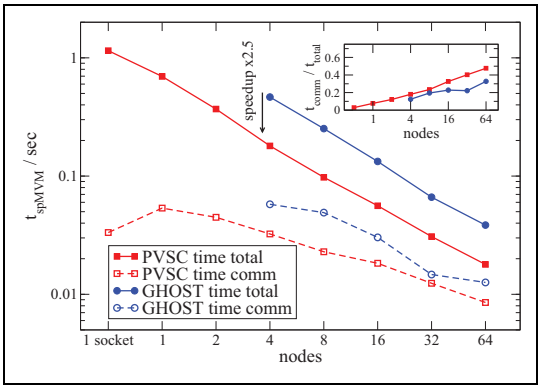

Figure 8. Runtime of spMVM ($n_b = 4$) for a strong scaling test case of the TI model (system size 384^3) with 200 iterations on the Emmy cluster, comparing PVSC-DTM with the GHOST library. The dashed lines show the communication time only. Inset: ratio of communication time vs. total runtime. All codes were run with one MPI process per socket (10 cores) and 10 OpenMP threads per process.

communication overhead of about 40–50% (see insets) can be observed for PVSC-DTM, which is a reasonable upper limit in production runs for resource efficiency reasons. Note that GHOST exhibits a larger communication time than PVSC-DTM because it assumes a general matrix and cuts it into horizontal blocks, resulting in sub-optimal communication behavior for stencil-based patterns. Our generated code, on the other hand, can exploit the regular next-neighbor exchange pattern.

In the test case in Figure 7, GHOST requires at least 16 nodes for storing the matrix and two vectors. With the same resources, PVSC-DTM is about 5× faster, and can outperform GHOST already with four nodes. The ratio of communication to computation time is naturally larger with PVSC-DTM due to the faster code execution. Although this particular test case cannot be run on a single node with GHOST, the performance comparison at 16 nodes also reflects quite accurately the per-node (or per-socket, i.e. pure OpenMP) performance ratio between PVSC-DTM and GHOST, since at this point the communication overhead is still only 10–20%.

For $n_b > 1$ the memory traffic caused by the matrix becomes less significant and the speedup of PVSC-DTM versus GHOST gets smaller. In the smaller $n_b = 4$ test case shown in Figure 8, GHOST requires at least four nodes and is still about 2.5× slower than PVSC-DTM at that point. Again, this is also the expected per-socket speedup if it were possible to run the test case on a single socket with GHOST.

4. Algorithms and application examples

So far we have set the stage for the quantum physics context of possible applications of our framework. We have also described the performance properties of generated code and shown that it achieves near-optimal performance for the sparse matrix-vector multiplication (as given by the memory bandwidth limitation and the minimal code balance) on two different processor architectures for operators relevant for real applications. In the following sections we give some examples for typical applications in the field of Dirac and topological materials that utilize spMVM as a major numerical component.

In large-scale simulations of any kind, avoiding global synchronization points is crucial for scalability. This challenge can be met by modern matrix polynomial methods. The kernel polynomial method (KPM), the Chebyshev time propagation approach described in Weiße and Fehske (2008) and Alvermann and Fehske (2008), and the high-performance Chebyshev filter diagonalization technique (ChebFD) implementation introduced in Pieper et al. (2016) are already available in PVSC-DTM . These algorithms benefit from partial dot products, vector blocking, and loop fusion. The high-order commutator-free exponential time-propagation algorithm introduced by Alvermann and Fehske (2011) for driven quantum systems will be implemented in the near future.

4.1. Time propagation

The time evolution of a quantum state $|\psi\rangle$ is described by the Schrödinger equation. If the Hamilton operator H does not explicitly depend on the time t we can formally integrate this equation and express the dynamics in terms of the time evolution operator $U(t, t_0)$ as $|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle$ with $U(t, t_0) = e^{-iH(t-t_0)}$ ($\hbar = 1$). Expanding the time evolution operator into a finite series of first-kind Chebyshev polynomials of order k, $T_k(x) = \cos(k \arccos(x))$, we obtain (Fehske et al., 2009; Tal-Ezer and Kosloff, 1984)

$$U(\Delta t) = e^{-ib\Delta t}[c_0(a\Delta t) + 2\sum_{k=1}^{M} c_k(a\Delta t)T_k(\tilde{H})]. \qquad (14)$$

Prior to the expansion the Hamiltonian has to be shifted and rescaled such that the spectrum of $\tilde{H} = (H - b)/a$ is within the definition interval of the Chebyshev polynomials, $[-1, 1]$, where a and b are calculated from the extreme eigenvalues of H: $b = \frac{1}{2}(E_{\max} + E_{\min})$ and $a = \frac{1}{2}(E_{\max} - E_{\min} + \varepsilon)$. The expansion coefficients c_k are given by

$$c_k(a\Delta t) = \int_{-1}^{1} \frac{T_k(x) e^{-\mathrm{i}xa\Delta t}}{\pi\sqrt{1-x^2}} dx = (-\mathrm{i})^k J_k(a\Delta t) \qquad (15)$$

(J_k denotes the k-th order Bessel function of the first kind).

Calculating the evolution of a state $|\psi(t_0)\rangle$ from one time grid point to the adjacent one, $|\psi(t)\rangle = U(\Delta t)|\psi(t_0)\rangle$, we have to accumulate the c_k-weighted vectors $|w_k\rangle = T_k(\tilde{H})|\psi(t_0)\rangle$. Since the coefficients $c_k(a\Delta t)$ depend on the time step but not on time explicitly, we need to calculate them only once. The vectors $|w_k\rangle$ can be computed iteratively, exploiting the recurrence relation of the Chebyshev polynomials, $|w_{k+1}\rangle = 2\tilde{H}|w_k\rangle - |w_{k-1}\rangle$, with $|w_1\rangle = \tilde{H}|w_0\rangle$ and $|w_0\rangle = |\psi(t_0)\rangle$. Evolving the wave function from one time step to the next then requires M spMVMs of a given complex vector with the (sparse) Hamilton matrix of dimension N and the summation of the resulting vectors after an appropriate rescaling. Thus, for time-independent H, arbitrary large time steps are in principle possible at the expense of increasing M. We may choose M such that for $k > M$ the modulus of all expansion coefficients $|c_k(a\Delta t)| \sim J_k(a\Delta t)$ is smaller than a desired accuracy cutoff. This is facilitated by the fast asymptotic decay of the Bessel functions, $J_k(a\Delta t) \sim \frac{1}{\sqrt{2\pi k}}\left(\frac{ea\Delta t}{2k}\right)^k$ for $k \to \infty$. Thus, for large M, the Chebyshev expansion can be considered as quasi-exact. Besides the high accuracy of the method, the linear scaling of computation time with both time step and Hilbert space dimension are promising in view of potential applications to more complex systems. In our cases almost all computation time is spent in spMVMs, which can be efficiently parallelized, allowing for a good speedup on highly parallel computers. This also means that any significant speedup that can be achieved for the spMVM, such as by our matrix-free formulation, will have a corresponding effect on the runtime of the overall algorithm. The actual speedup is a function of the memory traffic reduction; for instance, a sparse matrix stored in CRS format that describes a stencil-like neighborhood relation with eight neighbors will (in double precision) cause a minimum data traffic of approximately 7.6 bytes/flop when acting on a RHS vector. In a matrix-free formulation this balance reduces to 1.3 bytes/flop, leading to a performance improvement of 5.7× if the memory bandwidth can be saturated in both cases.

As an example, we apply the Chebyshev time evolution scheme to the propagation and scattering of a Dirac electron wave packet on a graphene sheet with an imprinted gate-defined quantum dot array (Fehske et al., 2015; Pieper et al., 2013). This is a timely issue of of high experimental relevance (Caridad et al., 2016; Tang et al., 2016; Walls and Hadad, 2015). We mimic the quantum dot array by implementing the potential V_n in (2) as

$$V(\mathbf{r}) = \sum_{l=1,k=1}^{L,K} V_k \Theta(R_{\mathrm{dot}} - |\mathbf{R}_{l,k} - \mathbf{r}|) \qquad (16)$$

with varying amplitude $V_k = V_0 + \Delta V|k - K/2|$ in y direction. In (16), R_{dot} (D_{dot}) is the radius of a single quantum dot (the nearest-neighbor distance between dots) and $\mathbf{R}_{\mathbf{l},\mathbf{k}} = (x_0 + lD_{\mathrm{dot}}, y_0 + kD_{\mathrm{dot}})$ gives the dot's position [l (k) counts in x (y) direction]. The quantum dot lattice can be created by applying spatially confined top gate voltages. The gradient change of the dot potentials mimics spatially varying effective refraction indices for the Dirac electron waves.

Figure 9 illustrates the scattering and temporary particle confinement by the quantum dot array. It has been demonstrated by Heinisch et al. (2013) and Pieper et al. (2013) that the normal modes of an isolated quantum dot lead to sharp resonances in the scattering efficiency. Appearing for particular values of R_{dot}, V, and E, they allow the "trapping" even of Dirac electrons. Of course, for the scattering setup considered here, only quasi-bound states can appear, which may have an exceptionally long lifetime, however. Thereby the energy of the wave is fed into vortices inside the dot. For a periodic array of dots the normal modes at neighboring dots can couple, leading to coherence effects (such inter-dot particle transfer takes place on a reduced energy scale compared to pure graphene (Fehske et al., 2015)). The situation becomes more complicated—but also more interesting—when the dot potentials are modulated spatially or energetically. In this case, a direction-dependent transmission (cascaded Mie scattering (Caridad et al., 2016)) or even the focusing of the electron beam outside the dots can be observed (Tang et al., 2016). Similar phenomena are demonstrated by Figure 9. For this simulation the electron is created by a Gaussian wave packet

$$\psi(\mathbf{r}, t = 0) = \exp\left(-\frac{(x - x_0)^2}{4\Delta x^2}\right)\psi_{\mathbf{K},\mathbf{x}}(\mathbf{r}) \quad , \qquad (17)$$

where $\psi_{\mathbf{K},\mathbf{x}}(\mathbf{r})$ is Dirac electron with momentum in x direction. When the wavefront hits the dot region, the wave is partly trapped by the quantum dots, whereby—for the parameters used—the resonance conditions are better fulfilled near the lower/upper end of the dot array (here the particle wave is best captured). The other way around, the transmission (and also the reflection, i.e. the backscattering) is strongest in the central region, leading to a curved wavefront. For larger time values a second pulse (wavefront) emerges (note that we have reflections and repeated scattering events in our finite GNR, but the largest time considered, $t = 3t_1$, is much shorter than the pass-through time of an unperturbed Dirac wave). In any case, one

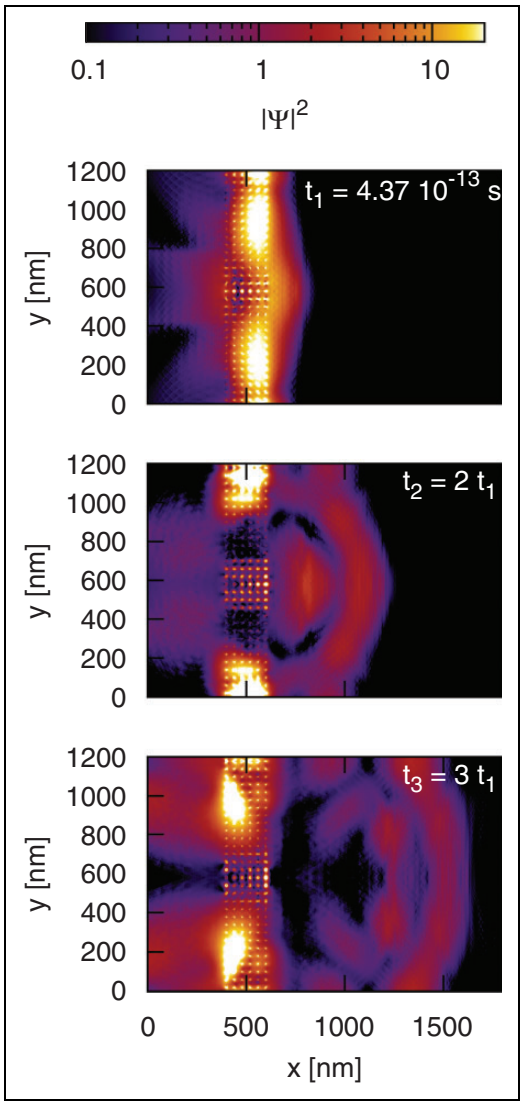

Figure 9. Time evolution of a Dirac electron wave impinging on a graphene quantum dot array (visible by the bright spots). We consider a GNR (periodic BCs in y direction) with an imprinted quantum dot lattice ($L = 6$). The radii of the quantum dots $R_{\text{dot}} = 10$ nm, their (midpoint) distance $D_{\text{dot}} = 40$ nm, and the dot potentials (parameterized by $V_0 = 0.1$, $\Delta V = 0.002$) vary along the y direction between a minimum and maximum value of $y = 600$ nm and $y = 0$ nm or 1200 nm, respectively. A (Gaussian) wave packet with momentum in x direction was created at $(x, y) = (200\,\text{nm}, 600\,\text{nm})$ with $(x, \Delta x) = (200\,\text{nm}, 300\,\text{nm})$ at time $t = 0$. The panels give the (color coded) squared amplitude of the wave function $|\psi(\mathbf{r}, t)|^2$ at times t_1, $2t_1$, and $3t_1$ with $t_1 = 4.37 \times 10^{-13}$ sec (from top to bottom).

observes a strongly time- and direction-dependent emission pattern for the considered graphene-based nanostructure, which can be exploited to manipulate electron beams. Particularly interesting in this respect would be focusing of the electron beam with large focal length, such that the focal spot lies outside the array. Then the structure can be used as a coupler to other electronic units. Achieving this by tuning the gradient of the gate potential appears to be a very efficient way, which is more easily realized in practice than modifying the geometrical properties of the array such as the lattice gradient or the layer number (Tang et al., 2016).

4.2. Interior eigenvalues of topological insulators

Since the electronic properties of TIs are mainly determined by the (topologically nontrival) surface states located in the bulk-state gap, an efficient calculation of electron states at or close to the center of a spectrum is of vital importance. This can be done by Chebyshev filter diagonalization (ChebFD), a straightforward scheme for interior eigenvalue computation, which is based on polynomial filter functions and therefore has much in common with the KPM. ChebFD applies a matrix polynomial filter that is suitable for the target interval to a block of vectors. In each iteration, the search space is checked for convergence using a Rayleigh-Ritz procedure. ChebFD has already proven its practical suitability: Parallelized and implemented on the "SuperMUC" supercomputer at LRZ Garching, 10^2 central eigenvalues of a 10^9-dimensional sparse matrix have been calculated at 40 Tflop/s sustained performance by Pieper et al. (2016).

Figure 10 shows the DOS of a strong TI. The focus is on the (pseudo-) gap region of the DOS. Implementing the effect of nonmagnetic impurities by uniformly distributed random on-site potentials V_n, we see how disorder fills the gap that exists in the DOS of system with a finite number of sites (see the red curves in the upper panels). Introducing a finite Δ_1, which mimics, e.g. the effect of an external magnetic field, the midgap Dirac cone formed by the surface states is broken up. Again, disorder will induce electronic states in the gap region generated by Δ_1. This is demonstrated by the lower panel of Figure 10, showing the DOS at the band center ($E = 0$) in the Δ_1-γ plane. As the disorder strength increases, more and more states pop up at $E = 0$ until the DOS saturates when γ reaches the order of magnitude of the bulk band gap. For a more detailed investigation of disordered (weak and strong) TIs we refer the reader to Kobayashi et al. (2013) where, besides the phase diagram, also the DOS was calculated using the KPM (see supplementary material in that paper). Compared to KPM, our ChebFD approach yields a better resolution at the same computational cost in the target interval (band center), which is important regarding the scientific applications.

The ChebFD algorithm is robust and scalable, but algorithmically sub-optimal. In PVSC-DTM we have also implemented the trLanczosFD (thick-restart Lanczos with polynomial filters) algorithm by Li et al. (2016). This algorithm benefits a little more from a matrix-free formulation because it uses smaller block vectors: Smaller blocks increase the impact of the data transfer for the matrix elements as shown, e.g. in Kreutzer et al. (2018). However, its actual advantage is improved convergence. A thorough description would exceed the scope of this paper; in Table 2 we show runtime data and the maximum residual of the inner Ritz eigenvalues for trLanczosFD on two TI test cases in comparison with ChebFD. TrLanczosFD outperforms

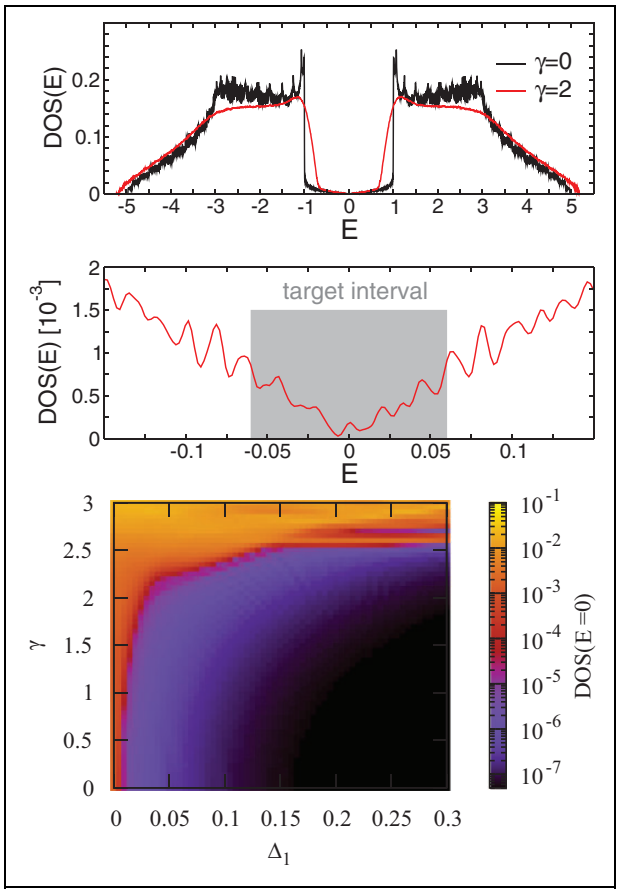

Figure 10. Density of states for a strong TI described by the Hamiltonian (4) with $m = 2$, $\Delta_2 = 0$, and open (periodic) BCs in z (x and y) direction. Top panel: DOS without ($V_n = 0$; black curve) and with [$V_n \in [-\gamma/2, \gamma/2]$; red curve] disorder, where $\Delta_1 = 0$. Data obtained by KPM with stochastic trace evaluation for a cuboid with $256 \times 256 \times 10$ sites. Middle panel: Zoom-in of the central part of the spectrum with the target interval used for the ChebFD calculations (Pieper et al., 2016). Bottom panel: DOS at the band center ($E = 0$) in dependence on the gap parameter Δ_1 and the disorder strength γ (Pieper and Fehske, 2016). Applying the KPM, 2048 Chebyshev moments were used for a system with $512 \times 512 \times 10$ sites. Note that the finite DOS at $\Delta_1 = 0$ is a finite-size effect and due to the finite KPM resolution (variance $\sigma = 0.01$).

Table 2. Test cases for the filter diagonalization method, using a matrix for TI with all eigenvalues in the range [−5.5,5.5]. Runtime and residuum data for runs with eight nodes on the Emmy cluster are shown for the ChebFD and the trLanczosFD algorithm, respectively.

	Test case 1	Test case 2
Size	480×480×6	240×240×6
Eigenpairs	72	40
Emmy nodes	8	8
ChebFD:		
Runtime [s]	2852	642
Max res.	7.3×10^{-12}	4.9×10^{-15}
TrLanczosFD:		
Runtime [s]	760	142
Max res.	3.5×10^{-15}	1.7×10^{-11}

ChebFD by a factor of almost four (using PVSC-DTM for both).

4.3. Disorder effects in Weyl semimetals

The Weyl nodes in the gapless topological Weyl semimetals are believed to be robust against perturbations unless, e.g. the translation or charge conservation symmetry is broken. Showing the stability of a single or a pair of Weyl nodal points against disorder has been the subject of intense research (Chen et al., 2015; Liu et al., 2016; McCormick et al., 2017; Pixley et al., 2015; Shapourian and Hughes, 2016; Zhao and Wang, 2015). Due to the vanishing DOS at the nodal points, disorder effects can be expected to be particularly pronounced. Since analytic methods fail widely in their quantitative predictions, even in the case of weak disorder, we use a purely numerical, KPM-based approach to analyze the spectral properties of Weyl semimetals with real-space quenched potential disorder.

Figure 11 displays the momentum-resolved spectral function $A(\mathbf{k}, E)$ of a disordered Weyl metal along different paths in the bulk Brillouin zone. The photoemission spectra shown were calculated for the model (8) with random potentials V_n drawn from a uniform box distribution of strength γ, i.e. $V_n \in [-\gamma/2, \gamma/2]$. The presented data should be compared with the results for the clean case provided by Figure 1 (c). Most notably, the electronic states at the Fermi arc (connecting the nodal points) and its immediate vicinity are hardly influenced by weak and even intermediate disorder. This does not apply for states further away from the Fermi surface. Here, the spectral signatures (band dispersion) are rapidly washed out, even for weak disorder. Of course, strong disorder will also affect the Fermi arc and the nodal points: Above a certain disorder strength they will be smeared out in both energy and momentum space and, as a result, the Weyl semimetal will transform into a diffusive metal with a finite DOS at the nodal points. A more detailed investigation of the spectral properties would be desirable in order to confirm the very recent evidence found by Su et al. (2017) for an intermediate Chern insulator state between the disordered Weyl semimetallic and diffusive metallic phases. At even stronger disorder, the distribution of the local density of states significantly broadens (just as in the case of strongly disordered strong TIs (Schubert et al., 2012) or disordered GNR (Schubert et al., 2009; Schubert and Fehske, 2012)) and Anderson localization sets in (Pixley et al., 2015).

5. Conclusion and outlook

The PVSC-DTM DSL and library have been demonstrated to be powerful tools for generating high-performance code to investigate ground-state, spectral, and dynamic properties of graphene, topological insulators, and other materials whose physics is governed by short-range interactions that lead to stencil-like numerical kernels. In particular, by calculating the time evolution and scattering of wave packets

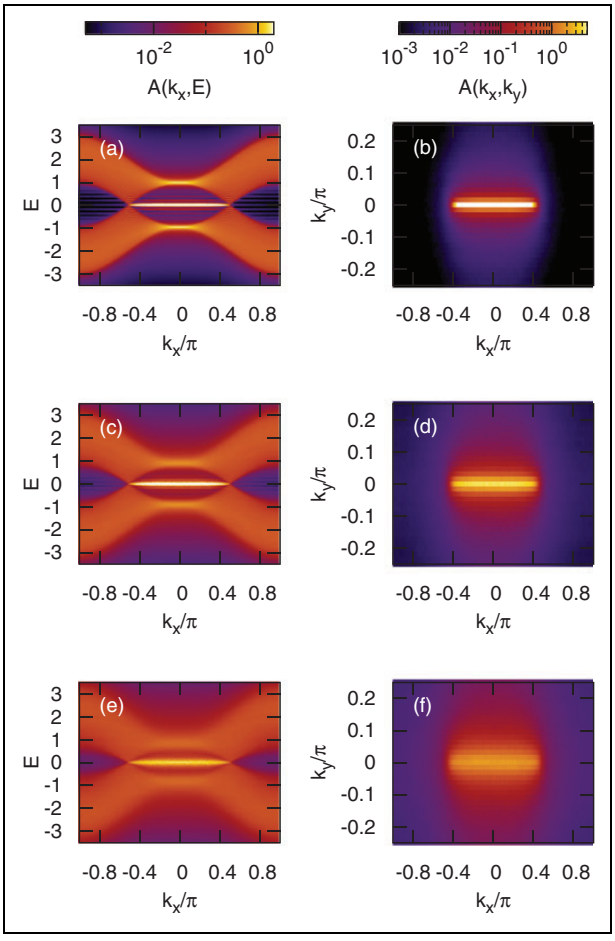

Figure 11. Spectral function $A(\mathbf{k}, E)$ for a disordered Weyl semimetal with Fermi arcs, as obtained from the respective equation (6) for a 3D system with $256 \times 32 \times 32$ sites and periodic (open) BCs in x, y (z) directions. The test wave function is initialized only on one surface. Left: $A(\mathbf{k}, E)$ along the k_x direction for $k_y = k_z = 0$. Right: $A(\mathbf{k}, E)$ in the k_x-k_y plane ($k_z = 0$) at $E = 0$. The disorder strength $\gamma = 1$ (top row), $\gamma = 2$ (middle row), and $\gamma = 3$ (bottom row).

in graphene-based nanostructures, determining the interior eigenvalues related to protected surface states in topological insulators, and treating the effects of random impurities in Weyl semimetals, we exemplarily showed that the proposed PVSC-DTM scheme can easily be combined with other numerical algorithms based on an efficient matrix-vector multiplication and used for studying very diverse aspects in the highly topical field of functional quantum matter.

Due to is matrix-free design, PVSC-DTM outperforms matrix-based libraries such as GHOST. It also implements effective SIMD vectorization and fast on-the-fly random number generation and yields optimal memory-bound chip-level performance as shown by the roofline model. Spatial blocking of the iteration loops is fully automatic and based on layer conditions.

Several improvements to the DSL library are left for future work: A better integration of the random number generator with the inner update loop would increase the non-saturated and sequential performance. Overlapping computation with communication would improve the distributed-memory parallel efficiency. Both optimizations are prerequisites for a possible integration with advanced blocking algorithms. Blocked ChebFD, which we actually used in this work, is one example, but more complex schemes exist. Exploiting the symmetry of the stencil shape or coefficients is not currently implemented in the DSL but could be useful to make writing the code easier. The system geometry is currently limited to rectangular and cuboid domains, which is a restriction that may be lifted to support other physical setups, e.g. ring-shaped structures to study the Aharonov-Bohm effect or boundary-related (topological) states.

Finally we plan to implement more algorithms in order to make the library more versatile beyond the showcases described here.

Acknowledgements

We thank Andreas Alvermann, Rafael L. Heinisch, and Gerhard Wellein for fruitful discussions.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: We are grateful for financial support by KONWIHR, the Bavarian Competence Network for High Performance Computing (projects PVSC-TM and PEigFex). We are also indebted to the LRZ Garching for providing access to HLRB-II.

ORCID iD

Georg Hager https://orcid.org/0000-0002-8723-2781

Notes

1. This can either be achieved by using the restrict keyword on pointer declarations, by compiler directives such as ivdep, or via global options such as `-fno-alias`.
2. http://tiny.cc/PVSC-DTM

References

Alvermann A and Fehske H (2008) Chebyshev approach to quantum systems coupled to a bath. *Physical Review B* 77: 045125.

Alvermann A and Fehske H (2011) High-order commutator-free exponential time-propagation of driven quantum systems. *Journal of Computational Physics* 230(15): 5930–5956.

Bandishti V, Pananilath I and Bondhugula U (2012) Tiling stencil computations to maximize parallelism. In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*, Salt Lake City, UT, USA, 10–16 November 2012, pp. 1–11.

Basu P, Venkat A, Hall M, et al. (2013) Compiler generation and autotuning of communication-avoiding operators for geometric multigrid. In: *20th Annual international conference on high performance computing*, Bangalore, India, 18–21 December 2013, pp. 452–461.

Caridad JM, Connaughton S, Ott C, et al. (2016) An electrical analogy to Mie scatterings. *Nature Communications* 7: 12894.

Castro Neto AH, Guinea F, Peres NMR, et al. (2009) The electronic properties of graphene. *Reviews of Modern Physics* 81: 109–162.

Chamon C, Goerbig MO, Moessner R, et al. (eds) (2014) *Topological Aspects of Condensed Matter Physics*, *Lecture Notes of the Les Houches Summer School*, vol. 103. Oxford: Oxford University Press.

Chen CZ, Song J, Jiang H, et al. (2015) Disorder and metal-insulator transitions in Weyl semimetals. *Physical Review Letters* 115: 246603.

Datta K, Kamil S, Williams S, et al. (2009) Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Review* 51(1): 129–159.

Datta K, Murphy M, Volkov V, et al. (2008) Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: *Proceedings of the 2008 ACM/IEEE conference on supercomputing*, SC '08. Piscataway, NJ, USA: IEEE Press. ISBN 978-1-4244-2835-9, pp. 4:1–4:12. Available at: http://dl.acm.org/citation.cfm?id=1413370.1413375 (accessed 12 September 2020).

Fehske H, Hager G and Pieper A (2015) Electron confinement in graphene with gate-defined quantum dots. *Physica Status Solidi (B)* 252(8): 1868–1871.

Fehske H, Schleede J, Schubert G, et al. (2009) Numerical approaches to time evolution of complex quantum systems. *Physics Letters A* 373: 2182–2188.

Fu L, Kane CL and Mele EJ (2007) Topological insulators in three dimensions. *Physical Review Letters* 98: 106803.

Goerbig MO (2011) Electronic properties of graphene in a strong magnetic field. *Reviews of Modern Physics* 83: 1193–1243.

Gropp WD, Kaushik DK, Keyes DE, et al. (2000) Towards realistic performance bounds for implicit CFD codes. In: Keyes D, Periaux J, Ecer A, Satofuka N and Fox P (eds) *Parallel Computational Fluid Dynamics 1999*. Amsterdam: Elsevier. ISBN 978-0-444-82851-4, pp. 241–248. DOI: 10.1016/B978-044482851-4.50030-X.

Hasan MZ and Kane CL (2010) Colloquium: topological insulators. *Reviews of Modern Physics* 82: 3045–3067.

Hasan MZ, Xu SY, Belopolski I, et al. (2017) Discovery of Weyl fermion semimetals and topological Fermi arc states. *Annual Review of Condensed Matter Physics* 8(1): 289–309.

Heinisch RL, Bronold FX and Fehske H (2013) Mie scattering analog in graphene: lensing, particle confinement, and depletion of Klein tunneling. *Physical Review B* 87: 155409.

Ilic A, Pratas F and Sousa L (2014) Cache-aware roofline model: upgrading the loft. *IEEE Computer Architecture Letters* 13(1): 21–24.

Kamil S, Chan C, Oliker L, et al. (2010) An auto-tuning framework for parallel multicore stencil computations. In: *2010 IEEE international symposium on parallel distributed processing (IPDPS)*, Atlanta, GA, USA, 19–23 April 2010, pp. 1–12.

Kobayashi K, Ohtsuki T and Imura KI (2013) Disordered weak and strong topological insulators. *Physical Review Letters* 110: 236803.

Kreutzer M, Ernst D, Bishop AR, et al. (2018) Chebyshev filter diagonalization on modern manycore processors and GPGPUs. In: R Yokota, M Weiland, D Keyes, and C Trinitis (eds) *High Performance Computing*. Cham: Springer International Publishing. ISBN 978-3-319-92040-5, pp. 329–349.

Kreutzer M, Pieper A, Hager G, et al. (2015) Performance engineering of the kernel polynomial method on large-scale CPU-GPU systems. In: *2015 IEEE international parallel and distributed processing symposium*, Hyderabad, India, 25–29 May 2015, pp. 417–426.

Kreutzer M, Thies J, Röhrig-Zöllner M, et al. (2017) GHOST: building blocks for high performance sparse linear algebra on heterogeneous systems. *International Journal of Parallel Programming* 45(5): 1046–1072.

L'Ecuyer P and Simard R (2007) TestU01: a C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software* 33(4): 22:1–22:40.

Li R, Xi Y, Vecharynski E, et al. (2016) A thick-restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems. *SIAM Journal on Scientific Computing* 38(4): A2512–A2534.

Liu S, Ohtsuki T and Shindou R (2016) Effect of disorder in a three-dimensional layered Chern insulator. *Physical Review Letters* 116: 066401.

Malas TM, Hager G, Ltaief H, et al. (2017) Multidimensional intratile parallelization for memory-starved stencil computations. *ACM Transactions on Parallel Computing* 4(3): 12: 1–12:32.

Marsaglia G (2003) Xorshift RNGs. *Journal of Statistical Software* 8(1): 1–6.

McCalpin JD (1991–2007) STREAM: sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, VA. Available at: http://www.cs.virginia.edu/stream/. A continually updated technical report (accessed 12 September 2020).

McCormick TM, Kimchi I and Trivedi N (2017) Minimal models for topological Weyl semimetals. *Physical Review B* 95: 075133.

Pieper A and Fehske H (2016) Topological insulators in random potentials. *Physical Review B* 93: 035123.

Pieper A, Heinisch RL and Fehske H (2013) Electron dynamics in graphene with gate-defined quantum dots. *EPL (Europhysics Letters)* 104(4): 47010.

Pieper A, Kreutzer M, Alvermann A, et al. (2016) High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. *Journal of Computational Physics* 325: 226–243.

Pixley JH, Goswami P and Das Sarma S (2015) Anderson localization and the quantum phase diagram of three dimensional disordered Dirac semimetals. *Physical Review Letters* 115: 076601.

Qi XL and Zhang SC (2011) Topological insulators and superconductors. *Reviews of Modern Physics* 83: 1057–1110.

Ragan-Kelley J, Barnes C, Adams A, et al. (2013) Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: *Proceedings of the 34th ACM SIGPLAN conference on programming language design and implementation*, PLDI '13. New York, NY, USA: ACM. ISBN 978-1-4503-2014-6, pp. 519–530.

Randles AP, Kale V, Hammond J, et al. (2013) Performance analysis of the lattice Boltzmann model beyond Navier-Stokes. In: *2013 IEEE 27th international symposium on parallel and distributed processing*, Boston, MA, USA, 20–24 May 2013, pp. 1063–1074.

Rivera G and Tseng CW (2000) Tiling optimizations for 3D scientific computations. In: *Supercomputing, ACM/IEEE 2000 conference*, Dallas, TX, USA, USA, 4–10 November 2000, pp. 32–32.

Schmitt C, Kuckuk S, Hannig F, et al. (2014) ExaSlang: a domain-specific language for highly scalable multigrid solvers. In: *2014 Fourth international workshop on domain-specific languages and high-level frameworks for high performance computing*, pp. 42–51. DOI: 10.1109/WOLFHPC.2014.11.

Schubert G and Fehske H (2012) Metal-to-insulator transition and electron-hole puddle formation in disordered graphene nanoribbons. *Physical Review Letters* 108: 066402.

Schubert G, Fehske H, Fritz L, et al. (2012) Fate of topological-insulator surface states under strong disorder. *Physical Review B* 85: 201105.

Schubert G, Schleede J and Fehske H (2009) Anderson disorder in graphene nanoribbons: a local distribution approach. *Physical Review B* 79: 235116.

Shapourian H and Hughes TL (2016) Phase diagrams of disordered Weyl semimetals. *Physical Review B* 93: 075108.

Sitte M, Rosch A, Altman E, et al. (2012) Topological insulators in magnetic fields: Quantum Hall effect and edge channels with a nonquantized θ term. *Physical Review Letters* 108: 126807.

Stengel H, Treibig J, Hager G, et al. (2015) Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model. In: *Proceedings of the 29th ACM international conference on supercomputing*, ICS '15. New York, NY, USA: ACM. DOI: 10.1145/2751205.2751240.

Su Y, Wang XS and Wang SR (2017) A generic phase between disordered Weyl semimetal and diffusive metal. *Scientific Reports* 7: 14382.

Tal-Ezer H and Kosloff R (1984) An accurate and efficient scheme for propagating the time dependent Schrödinger equation. *The Journal of Chemical Physics* 81(9): 3967–3971.

Tang Y, Cao X, Guo R, et al. (2016) Flat-lens focusing of electron beams in graphene. *Scientific Reports* 6: 33522.

Tang Y, Chowdhury RA, Kuszmaul BC, et al. (2011) The Pochoir stencil compiler. In: *Proceedings of the twenty-third annual ACM symposium on parallelism in algorithms and architectures, SPAA '11*. New York, NY, USA: ACM. ISBN 978-1-4503-0743-7, pp. 117–128.

Vigna S (2017) Further scramblings of Marsaglia's xorshift generators. *Journal of Computational and Applied Mathematics* 315: 175–181.

Walls J and Hadad D (2015) Suppressing Klein tunneling in graphene using a one-dimensional array of localized scatterers. *Scientific Reports* 5: 8435.

Wan X, Turner AM, Vishwanath A, et al. (2011) Topological semimetal and Fermi-arc surface states in the electronic structure of pyrochlore iridates. *Physical Review B* 83: 205101.

Weiße A and Fehske H (2008) Exact diagonalization techniques. In: Fehske H, Schneider R and Weiße A (eds) *Computational Many-Particle Physics*, *Lecture Notes in Physics*, vol. 739, p. 529. DOI: 10.1007/978-3-540-74686-7_18.

Weiße A, Wellein G, Alvermann A, et al. (2006) The kernel polynomial method. *Reviews of Modern Physics* 78: 275–306.

Williams S, Waterman A and Patterson D (2009) Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM* 52(4): 65–76.

Wonnacott DG (2000) Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In: *International parallel and distributed processing symposium*, Cancun, Mexico, 1–5 May 2000, pp. 171–180.

Xu SY, Belopolski I, Alidoust N, et al. (2015) Discovery of a Weyl fermion semimetal and topological Fermi arcs. *Science* 349(6248): 613–617.

Yang KY, Lu YM and Ran Y (2011) Quantum Hall effects in a Weyl semimetal: possible application in pyrochlore iridates. *Physical Review B* 84: 075129.

Zhang N, Driscoll M, Markley C, et al. (2017) Snowflake: a lightweight portable stencil DSL. In: *2017 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, Lake Buena Vista, FL, USA, 29 May–2 June 2017, pp. 795–804.

Zhao YX and Wang ZD (2015) Disordered Weyl semimetals and their topological family. *Physical Review Letters* 114: 206602.

Author biographies

Andreas Pieper holds a Master degree in Physics from the University of Greifswald. He worked as a research associate at the Universities of Greifswald and Erlangen, and was supported by the Bavarian KONWIHR project for high-performance computing. Using innovative numerical techniques, he mainly studied transport in graphene-based nanostructures and topological quantum matter.

Georg Hager holds a PhD and a Habilitation degree in Computational Physics from the University of Greifswald. He is a senior researcher in the HPC group at Erlangen Regional Computing Center and an associate lecturer at the Institute of Physics of the University of Greifswald. Recent research includes architecture-specific optimization strategies for current microprocessors, performance engineering of scientific codes on chip and system levels, and structure formation in large-scale parallel codes.

Holger Fehske received a PhD in Physics from the University of Leipzig, and the Habilitation and Venia Legendi in Theoretical Physics from the University of Bayreuth. In 2002 he became a full professor in Greifswald. Since then he holds the chair for complex quantum systems and works in the fields of solid state theory, quantum statistical physics, light-matter interaction, quantum informatics, plasma physics, and computational physics. He is also a long-standing member of the steering committee of the High Performance Computing Center Stuttgart.