Hybrid CUDA-OpenMP parallel implementation of a deterministic solver for ultra-short DG-MOSFETs

José M Mantas, Francesco Vecil

Abstract

The simulation of nanoscale 2D DG-MOSFETs and similar semiconductor devices through a deterministic and accurate model can be very useful for the industry but is particularly costly from the computational point of view. In this paper, we develop a hybrid parallel solver: the computing phases which corresponds to the simulation of the Boltzmann transport equation in the longitudinal dimension are performed on the GPU, while that phases related to the modelling of the electrons as waves in the transversal dimension are computed on the multi-core CPUs by using OpenMP. We have adapted the most costly computing phases to GPU in an efficient manner, achieving high performance and reducing drastically the simulation time. We give details about the parallel-design strategy and show the performance results.

Keywords. Semiconductor physics, Subband decomposition, WENO schemes, Quantum-classical dimensional coupling, Deterministic mesoscopic models, Parallel heterogeneous systems, GPU computing, Schrödinger-Poisson system, Parallelization of numerical algorithms.

Hybrid CUDA-OpenMP parallel implementation of a deterministic solver for ultra-short DG-MOSFETs

José M. Mantas*and Francesco Vecil†

1 Introduction

The simulation of semiconductor devices is of very high interest from the technological point of view. In particular, our efforts are oriented towards the nanoscale Double Gate Metal Oxide Semiconductor Field-Effect Transistor, (DG-MOSFET) [Camiola (2013), Suhag (2017)] depicted in Figure 1, which is widely used as logical unit inside electronic devices. The shortest MOSFET used in real integrated circuits is, as of mid 2017, 14 nm long. In this work we aim at simulating a 10 nm one. Downscaling allows for better performances and energy saving [Prasher (2013)]: by one side more logical units fit in the same physical space, on the other side the MOSFET's switching time is reduced .
In order to simulate such objects, we have used a semiclassical approach [Ben Abdallah et al. (2009)]. In the longitudinal dimension, electrons are described as particles: their travel along the device driven by an applied bias is modeled by a set of Boltzmann equations; seven scattering phenomena between electrons and phonons (the silicon crystal lattice's vibrations) are also taken into account. In the transversal dimension, the confinement is modeled through Schrödinger equations thus describing the electrons as waves. Finally, the Poisson equation for the electrostatic field couples the classical and the quantum dimension. In this way, we obtain a fully deterministic solver for the Boltzmann-Schrödinger-Poisson system of equations which exhibits several important advantages in comparison with Monte-Carlo Solvers [Mantas et.al. (2009), Vecil et al. (2014)]. We address the reader to the germ works [Vecil et al. (2014), Ben Abdallah et al. (2009)] for a sound description of the model and the methods.
A sequential implementation of this solver is not practical for realistic simulations because the numerical methods which take part are too time-consuming. In [Vecil et al. (2014)], a parallel implementation of the solver based on the Message Passing Interface (MPI) was described and its performance was studied on a computer cluster. Notwithstanding, the performance is still not competitive with respect to a

*Lenguajes y Sistemas Informáticos, Universidad de Granada, Spain
†Laboratoire de Mathématiques Blaise Pascal, Université Clermont Auvergne, France

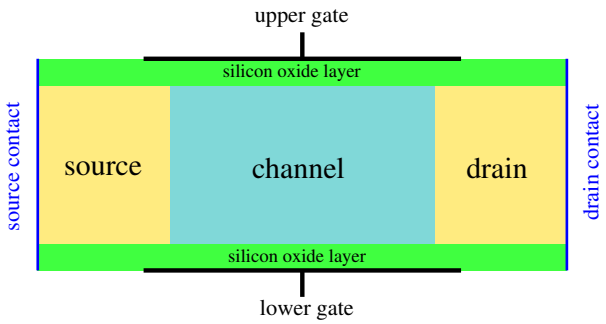

Figure 1: **Geometry of the DG-MOSFET**.

macroscopic, hence lower-dimensional, models which posses an intermediate accuracy depth like: drift-diffusion [Ben et.al. (2004)], energy-transport [Ungel (2010)], moment-like [Ringhofer (2001)], maximum-entropy-principle [Camiola (2013), Mascali et.al. (2012)] or Spherical Harmonics Expansion (SHE) models [Rupp (2011), Hong (2011), Rupp (2012)].
The use of heterogeneous systems equipped with *Graphics Processing Units (GPUs)* has proven to be effective in the acceleration of many numerical computations in science and engineering which exhibit a lot of exploitable fine-grain parallelism. [Owens (2008), Ujaldon (2012), Brodtkorb (2013)]. Nowadays the theorical peak performance of a modern and powerful GPU is around seven times the peak performance of a corresponding CPU [NVIDIA CUDA C (2018)]. In particular, NVIDIA CUDA framework [NVIDIA CUDA Home(2018)] has shown a high efectiveness in the mapping of intensive numerical computations to GPUs, making it possible substantial accelerations of practical simulations over a cutting-edge CPU with a very affordable price and programming effort. As a consequence, there has been a widespread use of CUDA-based platforms to accelerate numerical solvers for Partial Differential Equations. In multiple cases, the main data structures and computations of the numerical scheme are fully ported to the GPU device (see [Asunción (2011), Castro (2011), Asunción (2013), Abdi (2017)] as a sample) and the CPU only monitors the GPU execution by launching the CUDA kernels. However, in many applications it is necessary an hybrid CPU-GPU parallelization because some computing phases can not be efficiently adapted to the CUDA programming model. In this approach, the CUDA programming model can be used for the GPU phases and other parallel programming interfaces can be used to exploit the multiple cores of the available CPUs [DeVries (2013), Ye (2015), Norouzi (2017)].
In the literature, it can be found that other works have derived GPU implementations to simulate semiconductor devices, obtaining noteworthy performance improvements regarding CPU implementations. In [Suzuki (2015)] , a particle-based Monte-Carlo method to approximate the Boltzman transport equation is adapted to a CUDA-enabled GPU in order to simulate a nanostructured MOSFET device. Another Monte-Carlo approach to solve a quantum-kinetic model which describe the electron transport in semiconductors is adapted to a CUDA-based platform in [Karaivanova (2013)]. In [Rupp (2012)], a GPU implementation of the SHE method is derived using the ViennaCL library [Rupp (2016)] to simulate a 2D n^+nn^+ diode. However, we have not found works that had dealt with the adaptation to GPU-based platforms of the fully determinstc numerical solution of the Boltzmann-Schrödinger-Poisson system.
The goal of the present work is the development of an efficient implementation strategy for the Boltzmann-Schrödinger-Poisson solver in [Vecil et al. (2014)] which exploits a heterogeneous parallel environment with several shared-memory multi-core CPUs and one GPU. We make joint use of *OpenMP* [Chapman (2008)] and *CUDA* technologies to reduce considerably the simulation times. The solver, basically, alternatively performs two computational phases: it integrates in time the Boltzmann equations, and it computes the eigenstates. The first part is fully parallelized on the GPU. The second part involves several calculations which are difficult to adapt efficiently to the GPU. Therefore, a mixture of parallel-programming technologies is exploited, depending on which one is empirically more suitable. In this way, we will be able to perform device simulations in reasonable execution times which would promote the use of this highly accurate deterministic simulator for engineering environments.
The paper is structured as follows: Section 2 summarizes the mathematical model and the numerical solver; on Section 3 we detail our implementation strategy; on Section 4 we show our numerical experiments; finally, Section 5 draws some conclusions and sketches our plans for the future.

2 Summary of the mathematical model and the numerical scheme

2.1 Mathematical model

In this paper we focus on efficiently solving the adimensionalized Boltzmann-Schrödinger-Poisson model. We address the reader to [Vecil et al. (2014)] for the mathematical model written with physical units and all the adimensionalization parameters.
From now on, every magnitude will be meant dimensionless. Moreover, some other constants resulting from

the rescaling process will be omitted for the sake of clarity and lighter equations, and can be found in [Vecil et al. (2014)].
The Boltzmann-Schrödinger-Poisson model reads

$$\frac{\partial \Phi_{\nu,p}}{\partial t} + \frac{\partial}{\partial x}\left[a^1_\nu\,\Phi_{\nu,p}\right] + \frac{\partial}{\partial w}\left[a^2_{\nu,p}\,\Phi_{\nu,p}\right] + \frac{\partial}{\partial \phi}\left[a^3_{\nu,p}\,\Phi_{\nu,p}\right] = \mathcal{Q}_{\nu,p}[\Phi]\,s_\nu(w) \tag{1}$$

$$-\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{m_{z,\nu}}\frac{\mathrm{d}\psi_{\nu,p}}{\mathrm{d}z}\right) - (V+V_c)\,\psi_{\nu,p} = \epsilon_{\nu,p}\,\psi_{\nu,p} \tag{2}$$

$$-\nabla\cdot(\varepsilon_\mathrm{R}\,\nabla V) = -\left(\sum_{\nu,p}\varrho_{\nu,p}\cdot|\psi_{\nu,p}|^2 - N_D\right) \tag{3}$$

where z is the electron confinement dimension and x is the electron transport dimension. Here, $\Phi_{\nu,p}(t,x,w,\phi)$ is the probability of finding an electron of the ν^{th} valley, p^{th} subband, at time t, at position x, with energy-angle (w,ϕ) in the 2D impulsion space. The electron-phonon interactions are described by the scattering operator $\mathcal{Q}_{\nu,p}[\Phi]$, with $s_\nu(w)$ a given function due to the cartesian-to-ellipsoidal change of variables in the impulsion space. The eigenstates are the energy levels $\epsilon_{\nu,p}$ and the wave functions $\psi_{\nu,p}$, while the electrostatic potential V and the volume density N are mixed states. The physical constants in here are the silicon's effective masses $m_{z,\nu}$ and dielectric rigidity ε_R, and the MOSFET's confining potential V_c and doping profile N_D.
The presence of several valleys inside the Si band structure, plus the confinement due to the oxide layers, split the total electron population into several: one for each (ν,p)-pair. Hence, we have as many Boltzmann Transport Equations (BTEs) (1) as (ν,p)-pairs; for each BTE, the electrons are advected through the fluxes given by

$$a^1_\nu(w,\phi) = \frac{\sqrt{2w(1+\alpha_\nu w)}\cos(\phi)}{\sqrt{m_{x,\nu}}(1+2\alpha_\nu w)} \tag{4}$$

$$a^2_{\nu,p}(x,w,\phi) = -\frac{\partial \epsilon_{\nu,p}}{\partial x}(x)\;a^1_\nu(w,\phi) \tag{5}$$

$$a^3_{\nu,p}(x,w,\phi) = \frac{\partial \epsilon_{\nu,p}}{\partial x}(x)\;\frac{1}{\sqrt{2w(1+\alpha_\nu w)}}\;\frac{\sin(\phi)}{\sqrt{m_{x,\nu}}} \tag{6}$$

where α_ν is Kane's non-parabolicity factor for the ν^{th} valley. Remark that while a^1 is a constant for the problem, a^2 and a^3 evolve together with the energy levels $\epsilon_{\nu,p}$. Multiple scattering phenomena are taken into account, so that the operator is actually the sum of seven similarly-structured operators.
The Schrödinger equations (2) describe the confinement. Because dimension x acts only as a parameter, we have to solve as many eigenproblems as Si valleys times the discretization points along the x-dimension.
For the Poisson equation (3), its differential operators (the divergence and the gradient) are meant for (x,z), thus for both the transport and the confinement dimensions.
Refer to [Vecil et al. (2014), Ben Abdallah et al. (2009)] and references therein for more details.

2.2 Numerical scheme

2.2.1 The discretization

Globally, the problem spans on a 7-dimensional space:

(i). The **valley** (the silicon band structure)

$$\nu \in \{0,1,2\}.$$

(ii). The **subband** (the energy level's index):

$$p \in \{0, \dots, N_{\text{sbn}} - 1\}.$$

We shall use $N_{\text{sbn}} = 6$ subbands in all our tests.

(iii). The **longitudinal dimension** (unconfined):

$$x_i = i \times \underbrace{\frac{1}{N_x - 1}}_{\Delta x}, \quad i = 0, \dots, N_x - 1.$$

N_x denotes the number of discretization points in this dimension (x).

(iv). The **transversal dimension** (confined):

$$z_j = j \times \underbrace{\frac{1}{N_z - 1}}_{\Delta z}, \quad j = 0, \dots, N_z - 1.$$

N_z denotes the number of discretization points in this dimension (z).

(v). The **energy**:

$$w_\ell = (\ell + 0.5) \times \underbrace{\frac{w_{\max}}{N_w - 1}}_{\Delta w}, \quad \ell = 0, \dots, N_w - 1.$$

N_w is the number of points in the energy discretization and $w_{\max}$ denotes the maximum value of the energy [Vecil et al. (2014)]. The value $w = 0$ is taken out of the meshes because of a singularity at this point.

(vi). The **angle**:

$$\phi_m = m \times \underbrace{\frac{2\pi}{N_\phi}}_{\Delta \phi}, \quad m = 0, \dots, N_\phi - 1, \quad N_\phi \in 2\mathbb{N}.$$

N_ϕ is the number of discretization points for the angle.

(vii). As for the **time** step, it is adapted following a Courant-Friedrichs-Lewy condition, thus:

$$\begin{cases} t^0 = 0 & n = 0 \\ t^{n+1} = t^n + \text{CFL} \times \Delta t^n_{\max} & n \geq 0 \end{cases}$$

with

$$\begin{cases} \Delta t^0_{\max} = 10^{-18} & n = 0 \\ \Delta t^n_{\max} = \left(\frac{\|a^1\|_\infty}{\Delta x} + \frac{\|a^{2,n}\|_\infty}{\Delta w} + \frac{\|a^{3,n}\|_\infty}{\Delta \phi} \right)^{-1} & n \geq 0 \end{cases}$$

where CFL is a parameter in the interval $]0, 1[$, empirically chosen, and the flux coefficients a^i are (4), (5), (6), to which have have added index n to stress their time-dependency.

2.2.2 The magnitudes

From now on, index ν refers to the valley, index p to the subband, index i to point x_i, index j to point z_j, index ℓ to energy point w_ℓ, index m to angle point ϕ_m, index n to time instant t^n, index s to the s^{th} stage of the multi-stage time-integrator.[1]
Moreover, when the indices ν, p, i, j, ℓ, m are not explicitly quantified or specified, we mean by that we take them in their whole rank.
Hence, for instance,

$$\begin{aligned}\Phi^{n,s}_{\nu,p,i,\ell,m} = \Phi^{n,s}_{\nu,p}\left(x_i, w_\ell, \phi_m\right)\\ \forall \nu \in \{0,1,2\}\\ \forall p \in \{0,\ldots,N_{\text{sbn}}-1\}\\ \forall i \in \{0,\ldots,N_x-1\}\\ \forall \ell \in \{0,\ldots,N_w-1\}\\ \forall m \in \{0,\ldots,N_\phi-1\}.\end{aligned}$$

The main magnitudes used for the computations (for a fixed time instant t^n and a particular time-integrator stage s) are:

(a). The **probability distribution function** (pdf) is 5-dimensional:

$$\Phi_{\nu,p,i,\ell,m}.$$

(b). The **energy level** is 3-dimensional:

$$\epsilon_{\nu,p,i}$$

(c). The **surface density** is 3-dimensional:

$$\varrho_{\nu,p,i}$$

(d). The **wave function** is 4-dimensional:

$$\psi_{\nu,p,i,j}$$

(e). The **electrostatic potential** is 2-dimensional:

$$V_{i,j}$$

(f). The **volume density** is 4-dimensional:

$$N_{\nu,p,i,j}$$

Taking into account that the values of all the magnitudes will be stored on a one-dimensional array, the particular ordering of the values for each magnitude is specified in Table 1. In this table, we detail how the N-dimensional index for a magnitude is mapped to a one-dimensional index. For example, if the order is $i > \nu > p > \ell > m$, the mapping is

$$\begin{aligned}(\nu,p,i,\ell,m) \quad \longmapsto \quad & i \times 3 \times N_{\text{sbn}} \times N_w \times N_\phi\\ & +\nu \times N_{\text{sbn}} \times N_w \times N_\phi\\ & +p \times N_w \times N_\phi\\ & +\ell \times N_\phi\\ & +m.\end{aligned}$$

[1] The time-integrator chosen for this work is the third-order Total-Variation Diminishing Runge-Kutta algorithm and will be described in the following.

	dimensions N	1	2	3	4	5
$\Phi_{\nu,p,i,\ell,m}$	5	m	ℓ	p	ν	i
$\epsilon_{\nu,p,i}$	3	p	ν	i		
$\varrho_{\nu,p,i}$	3	p	ν	i		
$\psi_{\nu,p,i,j}$	4	j	p	ν	i	
$V_{i,j}$	2	j	i			
$N_{\nu,p,i,j}$	4	p	ν	j	i	
$\mathcal{W}_{\nu,p,\nu',p',i}$	5	p'	ν'	p	ν	i
$\tilde{\Phi}_{\nu,p,i,\ell}$	4	ℓ	p	ν	i	
$\mathcal{A}_{i,j,j'}$	3	j'	j	i		
$D_{\nu,p,i,\ell,m}$	5	m	ℓ	p	ν	i
$Q_{\nu,p,i,\ell,m}$	5	m	ℓ	p	ν	i
$\mathcal{H}_{\nu,p,i,\ell,m}$	5	m	ℓ	p	ν	i

Table 1: **Ordering of the values for each magnitude.**

2.2.3 General view

In Figure 2 we depict the general idea of how the scheme works. The time integrator used is the (explicit) third-order Total-Variation Diminishing Runge-Kutta (TVD RK-3) [Carrillo et al. (2003)]: we advance in time by performing the following phases:

BTE. We approximate the right hand side (rhs) of the Boltzmann Transport Equations at stage s (omitting the time index n) as:

$$\mathcal{H}^s_{\nu,p,i,\ell,m} := D^s_{\nu,p,i,\ell,m} + Q^s_{\nu,p,i,\ell,m} \tag{7}$$

where

$$\begin{aligned} D^s_{\nu,p,i,\ell,m} := &- \left[\frac{\partial}{\partial x}\left(a^1\,\Phi^s\right)\right]_{\nu,p,i,\ell,m} \\ &- \left[\frac{\partial}{\partial w}\left(a^{2,s}\,\Phi^s\right)\right]_{\nu,p,i,\ell,m} - \left[\frac{\partial}{\partial \phi}\left(a^{3,s}\,\Phi^s\right)\right]_{\nu,p,i,\ell,m} \end{aligned} \tag{8}$$

are the partial derivatives approximated by means of a fifth-order Weighted Essentially NOnoscillatory reconstruction (FD-WENO-5) (refer to [Carrillo et al. (2003)] and references therein about the WENO schemes) and

$$Q_{\nu,p,i,\ell,m}[\mathbf{\Phi}^s] := \sum_{\mu=0}^{6}\left[Q^{\mu,\text{gain}}_{\nu,p,i,\ell} + Q^{\mu,\text{loss}}_{\nu,p,i,\ell,m}\right] \tag{9}$$

is explicitly integrated by some formulae whose details are given later. Here, μ indexes one scattering phenomenon.

Then, we advance to the next Runge-Kutta stage $\mathbf{\Phi}^{n,s} \longrightarrow \mathbf{\Phi}^{n,s+1}$, using the above-mentioned TVD RK-3 scheme:

$$\mathbf{\Phi}^{n,1} = \mathbf{\Phi}^{n,0} + \Delta t\,\boldsymbol{\mathcal{H}}^0 \tag{10}$$

$$\mathbf{\Phi}^{n,2} = \frac{3}{4}\mathbf{\Phi}^{n,0} + \frac{1}{4}\mathbf{\Phi}^{n,1} + \frac{1}{4}\Delta t\,\boldsymbol{\mathcal{H}}^1 \tag{11}$$

$$\mathbf{\Phi}^{n+1,0} = \frac{1}{3}\mathbf{\Phi}^{n,0} + \frac{2}{3}\mathbf{\Phi}^{n,2} + \frac{2}{3}\Delta t\,\boldsymbol{\mathcal{H}}^{\mathbf{2}}. \tag{12}$$

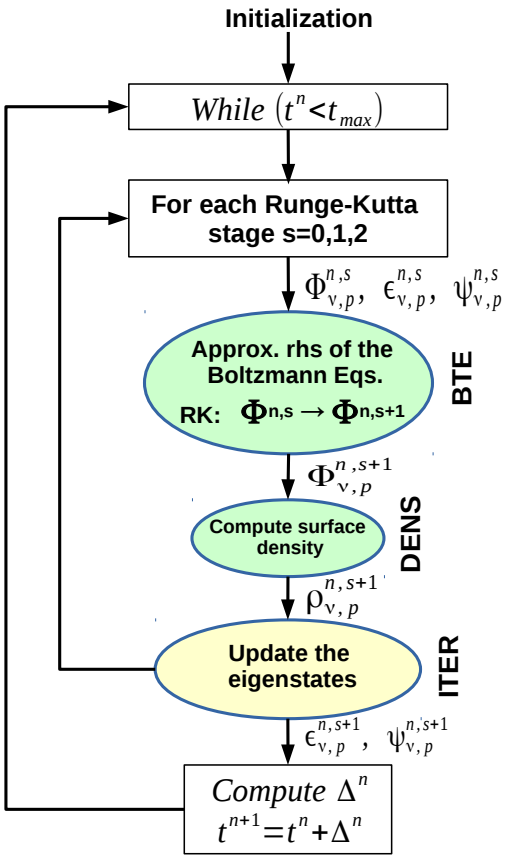

Figure 2: **Overview of the scheme's design.**

Schematically (omitting the time index n),

$$\begin{array}{ll} \left\{\Phi^s_{\nu,p,i,\ell,m}, \epsilon^s_{\nu,p,i}, \psi^s_{\nu,p,i,j}\right\} & \text{input} \\ \downarrow & \\ \left\{\Phi^{s+1}_{\nu,p,i,\ell,m}\right\} & \text{output.} \end{array}$$

DENS. The surface densities, which depend on the pdf, are recomputed in the following way (omitting the indices n and s):

$$\varrho_{\nu,p,i} = \Delta w\, \Delta\phi \sum_{\ell=0}^{N_w-1} \sum_{m=0}^{N_\phi-1} \Phi_{\nu,p,i,\ell,m}.$$

Schematically,

$$\begin{array}{ll} \left\{\Phi^{s+1}_{\nu,p,i,\ell,m}\right\} & \text{input} \\ \downarrow & \\ \left\{\varrho^{s+1}_{\nu,p,i}\right\} & \text{output.} \end{array}$$

ITER. As the eigenstates depend on the surface densities, we need to recompute them, in particular the energy levels $\{\epsilon_{\nu,p}\}$, which are the advection field of the BTEs. To that purpose, we solve the Schrödinger-Poisson block (2)-(3) through a Newton-Raphson iterative method.

Schematically,

$$\begin{array}{ll} \left\{\varrho^{s+1}_{\nu,p,i}\right\} & \text{input} \\ \downarrow & \\ \left\{\epsilon^{s+1}_{\nu,p,i}, \psi^{s+1}_{\nu,p,i,j}\right\} & \text{output.} \end{array}$$

After this phase, we can now chain up to the next RK stage.

In the following, we describe in detail the three main computational phases.

2.3 The *BTE* phase

In Figure 3 we summarize the computations involved in the **BTE** phase.

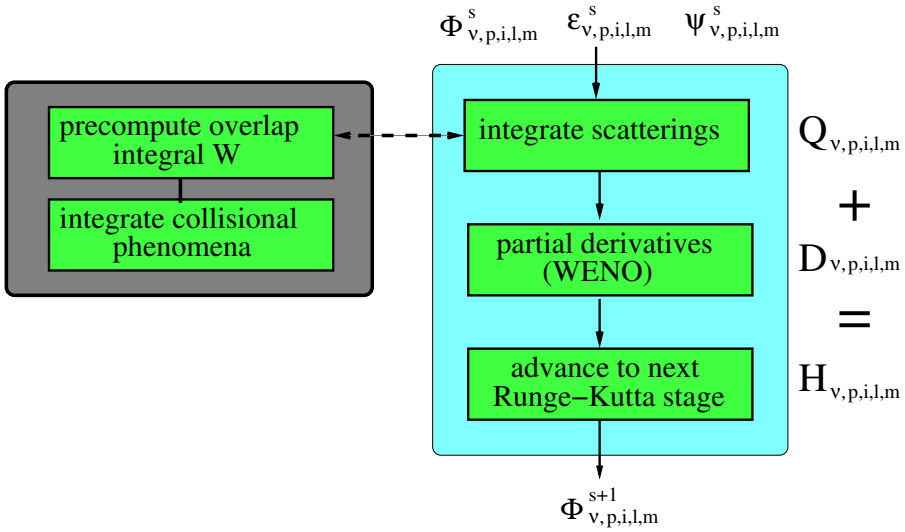

Figure 3: **BTE phase.**

2.3.1 Partial derivatives

This phase computes the three derivatives in (8). For the x-derivative we have

$$\begin{aligned} &\frac{\partial}{\partial x}\left[a^1 \cdot \Phi^s\right]_{\nu,p,i,\ell,m} \\ &\approx \frac{\left(a^1_{\nu,\ell,m}\widehat{\cdot\,\Phi^s_{\nu,p,\cdot,\ell,m}}\right)_{i+\frac{1}{2}} - \left(a^1_{\nu,\ell,m}\widehat{\cdot\,\Phi^s_{\nu,p,\cdot,\ell,m}}\right)_{i-\frac{1}{2}}}{\Delta x}, \end{aligned} \tag{13}$$

for the w-derivative we have

$$\begin{aligned} &\frac{\partial}{\partial w}\left[a^{2,s} \cdot \Phi^s\right]_{\nu,p,i,\ell,m} \\ &\approx \frac{\left(a^{2,s}_{\nu,p,i,\cdot,m}\widehat{\cdot\,\Phi^s_{\nu,p,i,\cdot,m}}\right)_{\ell+\frac{1}{2}} - \left(a^{2,s}_{\nu,p,i,\cdot,m}\widehat{\cdot\,\Phi^s_{\nu,p,i,\cdot,m}}\right)_{\ell-\frac{1}{2}}}{\Delta w}, \end{aligned} \tag{14}$$

for the ϕ-derivative we have

$$\frac{\partial}{\partial\phi}\left[a^{3,s}\cdot\Phi^s\right]_{\nu,p,i,\ell,m} \approx \frac{\left(\widehat{a^{3,s}_{\nu,p,i,\ell,\cdot}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m+\frac{1}{2}} - \left(\widehat{a^{3,s}_{\nu,p,i,\ell,\cdot}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m-\frac{1}{2}}}{\Delta\phi}. \tag{15}$$

Flux reconstruction and boundary conditions. In order to reconstruct the fluxes

$$\begin{aligned}
&\left(\widehat{a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,\cdot,\ell,m}}\right)_{i-\frac{1}{2}} && \text{for } i=0,\dots,N_x\\
&\left(\widehat{a^{2,s}_{\nu,p,i,\cdot,m}\cdot\Phi^s_{\nu,p,i,\cdot,m}}\right)_{\ell-\frac{1}{2}} && \text{for } \ell=0,\dots,N_w\\
&\left(\widehat{a^{3,s}_{\nu,p,i,\ell,\cdot}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m-\frac{1}{2}} && \text{for } m=0,\dots,N_\phi
\end{aligned}$$

using the FD-WENO-5 interpolation technique, we need three ghost points at the left of the first point, and three ghost points at the right of the last point:

$$\begin{aligned}
&\left(a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } i=-3,-2,-1\\
&\left(a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } i=N_x,N_x+1,N_x+2\\
&\left(a^{2,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } \ell=-3,-2,-1\\
&\left(a^{2,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } \ell=N_w,N_w+1,N_w+2\\
&\left(a^{3,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } m=-3,-2,-1\\
&\left(a^{3,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,m}\right) && \text{for } m=N_\phi,N_\phi+1,N_\phi+2,
\end{aligned}$$

These are set depending on the boundary conditions chosen, and are described in detail in [Vecil et al. (2014)]. We recall, anyway, that at the left border for w we substitute

$$\left(\widehat{a^{2,s}_{\cdot,m}\cdot\Phi^s_{\cdot,m}}\right)_{\ell=-\frac{1}{2}} \uparrow \frac{\left(\widehat{a^{2,s}_{\cdot,m}\cdot\Phi^s_{\cdot,m}}\right)_{\ell=-\frac{1}{2}} - \left(\widehat{a^{2,s}_{\cdot,m\pm\frac{N_\phi}{2}}\cdot\Phi^s_{\cdot,m\pm\frac{N_\phi}{2}}}\right)_{\ell=-\frac{1}{2}}}{2}, \tag{16}$$

where indexes ν,p,i have been omitted for the sake of compact notations and $m\pm\frac{N_\phi}{2}$ means the one which is between 0 and $N_\phi-1$. It is important to remark that, for the w-derivative, the flux reconstruction is non-local with respect to index m; rather, we need information coming from two lines

$$\left(a^{2,s}_{\ell,m}\cdot\Phi^s_{\ell,m}\right) \quad \text{for } \ell=-3,\dots,N_w+2$$

and

$$\left(a^{2,s}_{\ell,m\pm\frac{N_\phi}{2}}\cdot\Phi^s_{\ell,m\pm\frac{N_\phi}{2}}\right) \quad \text{for } \ell=-3,\dots,N_w+2$$

to obtain all the

$$\left(\widehat{a^{2,s}_{\cdot,m}\cdot\Phi^s_{\cdot,m}}\right)_{\ell-\frac{1}{2}} \qquad \text{for } \ell=0,\dots,N_w$$

and

$$\left(\widehat{(a^{2,s}_{\cdot,m}\cdot\Phi^s}_{\cdot,m\pm\frac{N_\phi}{2}}\right)_{\ell-\frac{1}{2}} \qquad \text{for } \ell=0,\dots,N_w$$

Upwinding for FD-WENO-5. Let us take as example the x-derivative. In order to approximate the flux

$$\left(\widehat{a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,\cdot,\ell,m}}\right)_{i-\frac{1}{2}}$$

for some $i\in\{0,\dots,N_x\}$, we need five values, namely

$$\left(a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,I,\ell,m}\right)_{I\in\{i-3,i-2,i-1,i,i+1\}}$$

or

$$\left(a^1_{\nu,\ell,m}\cdot\Phi^s_{\nu,p,I,\ell,m}\right)_{I\in\{i-2,i-1,i,i+1,i+2\}}$$

depending on whether the wind blows from the left (i.e. $a^1_{\nu,\ell,m}>0$) or from the right (i.e. $a^1_{\nu,\ell,m}<0$). The w-derivative goes much the same as the x-derivative: for the x-derivative and the w-derivative, the wind direction is constant with respect to x and w respectively; otherwise stated, neither the sign of $a^1_{\nu,\ell,m}$ nor that of $a^{2,s}_{\nu,p,i,\ell,m}$ depend on index i and ℓ respectively.
On the other hand, for flux $a^{3,s}_{\nu,p,i,\ell,m}$ the wind direction is not constant with respect to m, hence a flux splitting is needed:

$$\begin{aligned}&a^{3,s}_{\nu,p,i,\ell,m}\\&=\underbrace{\frac{a^{3,s}_{\nu,p,i,\ell,m}+\left\|a^{3,s}\right\|_\infty}{2}}_{=:a^{3+,s}_{\nu,p,i,\ell,m}\geq 0}+\underbrace{\frac{a^{3,s}_{\nu,p,i,\ell,m}-\left\|a^{3,s}\right\|_\infty}{2}}_{=:a^{3-,s}_{\nu,p,i,\ell,m}\leq 0}.\end{aligned}$$

This way, for $m=0,\dots,N_\phi$,

$$\begin{aligned}&\left(\widehat{a^{3,s}_{\nu,p,i,\ell,\cdot}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m-\frac{1}{2}}\\&=\left(\widehat{a^{3+,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m-\frac{1}{2}}\\&+\left(\widehat{a^{3-,s}_{\nu,p,i,\ell,m}\cdot\Phi^s_{\nu,p,i,\ell,\cdot}}\right)_{m-\frac{1}{2}}.\end{aligned}$$

2.3.2 Scatterings

The computations, sketched in Figure, follow two steps:
(i). Overlap integral. In this step, the overlap integral is precomputed, i.e.

$$\mathcal{W}_{\nu,p,\nu',p',i}=\Delta z\sum_{j=1}^{N_z-2}\left|\psi_{\nu,p,i,j}\right|^2\left|\psi_{\nu',p',i,j}\right|^2, \tag{17}$$

where ν' and p' take values in the same range as ν and p.
(ii). Integration. In this step, the scattering phenomena are integrated. In order to describe the following computations, let us first introduce some notations:

- The boolean function $\mathbb{1}$ is defined as

$$\mathbb{1}\left(\text{condition}\right) = \begin{cases} 1 & \text{if condition is fulfilled} \\ 0 & \text{otherwise} \end{cases}$$

- $\{\tilde{\Phi}_{\nu,p,i,\ell}\}$ means the ϕ-integrated pdf functions, which are computed inside the *DENS* phase. Refer to Section 2.4 for more details.

- In order to evaluate $\{\tilde{\Phi}_{\nu,p,i,\ell}\}$ at points Γ, which do not a priori belong to the w-grid, a linear interpolation is used:

$$\begin{aligned} \mathrm{LI}\left[\tilde{\Phi}_{\nu,p,i,\cdot}\right](\Gamma) := & \frac{\tilde{\Phi}_{\nu,p,i,\ell_\mathrm{u}} - \tilde{\Phi}_{\nu,p,i,\ell_\mathrm{d}}}{\Delta w} \times \Gamma \\ & + \frac{w_{\ell_\mathrm{u}} \cdot \tilde{\Phi}_{\nu,p,i,\ell_\mathrm{d}} - w_{\ell_\mathrm{d}} \cdot \tilde{\Phi}_{\nu,p,i,\ell_\mathrm{u}}}{\Delta w} \\ & \times \mathbb{1}\left(\Gamma \geq 0 \quad \wedge \quad \ell_\mathrm{d} \leq N_w - 2\right) \end{aligned}$$

  with

$$\ell_\mathrm{d} := \left\lfloor \frac{\Gamma}{\Delta w} - \frac{1}{2} \right\rfloor, \qquad \ell_\mathrm{u} := \ell_\mathrm{d} + 1.$$

- The definition of all the physical and numerical constants, appearing in the following, can be found in [Vecil et al. (2014)].

- The values Γ are the energies exchanged at collisional events, their expression being given by

$$\begin{aligned} \Gamma^0_{\nu,p,\nu',p',i,\ell} &= w_\ell + \epsilon_{\nu,p,i} - \epsilon_{\nu',p',i} \\ \Gamma^{\mu,\pm}_{\nu,p,\nu',p',i,\ell} &= w_\ell + \epsilon_{\nu,p,i} - \epsilon_{\nu',p',i} \pm \frac{\hbar\omega^\mu}{\epsilon^*}. \end{aligned}$$

- Magnitude s_ν, ℓ represents the Jacobian of the change of variables

$$s_\nu(w) := \sqrt{m_{\mathrm{x},\nu}\, m_{\mathrm{y},\nu}}(1 + 2\,\tilde{\alpha}_\nu\,\epsilon^*\,w).$$

We distinguish between three kinds of phenomena:

Elastic. One intra-valley phenomenon, without energy exchange:

$$\begin{aligned} Q^{\mu,\mathrm{gain}}_{\nu,p,i,\ell} = \mathcal{C}^\mu & \sum_{p'=0}^{N_\mathrm{sbn}-1} \mathcal{W}_{\nu,p,\nu,p',i} \cdot \mathbb{1}\left(\Gamma^0_{\nu,p,\nu,p',i,\ell} \geq 0\right) \\ & \times s_\nu(w_\ell) \cdot \mathrm{LI}\left[\tilde{\Phi}^s_{\nu,p',i,\cdot}\right]\left(\Gamma^0_{\nu,p,\nu,p',i,\ell}\right) \end{aligned} \tag{18}$$

and

$$\begin{aligned} Q^{\mu,\mathrm{loss}}_{\nu,p,i,\ell,m} = -\mathcal{C}^\mu\, 2\pi \cdot \Phi^s_{\nu,p,i,\ell,m} & \sum_{p'=0}^{N_\mathrm{sbn}-1} \mathcal{W}_{\nu,p,\nu,p',i} \\ \times\, \mathbb{1}\left(\Gamma^0_{\nu,p,\nu,p',i,\ell} \geq 0\right) & \cdot s_\nu\left(\Gamma^0_{\nu,p,\nu,p',i,\ell}\right). \end{aligned} \tag{19}$$

G-type. Three intra-valley phenomena, with energy exchange:

$$
\begin{aligned}
Q^{\mu,\mathrm{gain}}_{\nu,p,i,\ell} &= \mathcal{C}^{\mu}\, s_{\nu}(w_{\ell}) \sum_{p'=0}^{N_{\mathrm{sbn}}-1} \mathcal{W}_{\nu,p,\nu,p',i} \\
&\Big\{ \mathbb{1}\left(\Gamma^{\mu,+}_{\nu,p,\nu,p',i,\ell} \geq 0\right) \cdot (N^{\mu}_{\nu,\nu}+1) \\
&\times \mathrm{LI}\left[\tilde{\Phi}_{\nu,p',i,\cdot}\right]\left(\Gamma^{\mu,+}_{\nu,p,\nu,p',i,\ell}\right) \\
&+ \mathbb{1}\left(\Gamma^{\mu,-}_{\nu,p,\nu,p',i,\ell} \geq 0\right) \cdot N^{\mu}_{\nu,\nu} \\
&\times \mathrm{LI}\left[\tilde{\Phi}_{\nu,p',i,\cdot}\right]\left(\Gamma^{\mu,-}_{\nu,p,\nu,p',i,\ell}\right)\Big\}
\end{aligned} \tag{20}
$$

and

$$
\begin{aligned}
Q^{\mu,\mathrm{loss}}_{\nu,p,i,\ell,m} &= -\mathcal{C}^{\mu}\, 2\pi\, \Phi^{s}_{\nu,p,i,\ell,m} \sum_{p'=0}^{N_{\mathrm{sbn}}-1} \mathcal{W}_{\nu,p,\nu,p',i} \\
&\Big\{ \mathbb{1}\left(\Gamma^{\mu,+}_{\nu,p,\nu,p',i,\ell} \geq 0\right) \\
&\times N^{\mu}_{\nu,\nu} \cdot s_{\nu}\left(\Gamma^{\mu,+}_{\nu,p,\nu,p',i,\ell}\right) \\
&+ \mathbb{1}\left(\Gamma^{\mu,-}_{\nu,p,\nu,p',i,\ell} \geq 0\right) \\
&\times (N^{\mu}_{\nu,\nu}+1) \cdot s_{\nu}\left(\Gamma^{\mu,-}_{\nu,p,\nu,p',i,\ell}\right)\Big\}.
\end{aligned} \tag{21}
$$

F-type. Three inter-valley phenomena, with energy exchange:

$$
\begin{aligned}
Q^{\mu,\mathrm{gain}}_{\nu,p,i,\ell} &= \mathcal{C}^{\mu}\, 2 \sum_{p'=0}^{N_{\mathrm{sbn}}-1} \sum_{\nu'=0}^{2} \mathcal{W}_{\nu,p,\nu',p',i} \\
&\Big\{ \mathbb{1}\left(\Gamma^{\mu,+}_{\nu,p,\nu',p',i,\ell} \geq 0\right) \cdot (N^{\mu}_{\nu',\nu}+1) \\
&\times s_{\nu'}(w_{\ell}) \cdot \mathrm{LI}\left[\tilde{\Phi}_{\nu',p',i,\cdot}\right]\left(\Gamma^{\mu,+}_{\nu,p,\nu',p',i,\ell}\right) \\
&+ \mathbb{1}\left(\Gamma^{\mu,-}_{\nu,p,\nu',p',i,\ell} \geq 0\right) \cdot N^{\mu}_{\nu',\nu} \\
&\times s_{\nu}(w_{\ell}) \cdot \mathrm{LI}\left[\tilde{\Phi}_{\nu',p',i,\cdot}\right]\left(\Gamma^{\mu,-}_{\nu,p,\nu',p',i,\ell}\right)\Big\}
\end{aligned} \tag{22}
$$

and

$$
\begin{aligned}
Q^{\mu,\mathrm{loss}}_{\nu,p,i,\ell,m} &= -\mathcal{C}^{\mu}\, 4\pi\, \Phi^{s}_{\nu,p,i,\ell,m} \sum_{p'=0}^{N_{\mathrm{sbn}}-1} \sum_{\nu'=0}^{2} \mathcal{W}_{\nu,p,\nu',p',i} \\
&\Big\{ \mathbb{1}\left(\Gamma^{\mu,+}_{\nu,p,\nu',p',i,\ell} \geq 0\right) \\
&N^{\mu}_{\nu,\nu'} \cdot s_{\nu'}\left(\Gamma^{\mu,+}_{\nu,p,\nu',p',i,\ell}\right) \\
&+ \mathbb{1}\left(\Gamma^{\mu,-}_{\nu,p,\nu',p',i,\ell} \geq 0\right) \\
&+ (N^{\mu}_{\nu,\nu'}+1) \cdot s_{\nu'}\left(\Gamma^{\mu,-}_{\nu,p,\nu',p',i,\ell}\right)\Big\}.
\end{aligned} \tag{23}
$$

2.4 The *DENS* phase

This phase performes two computations, that are sketched in Figure 4.

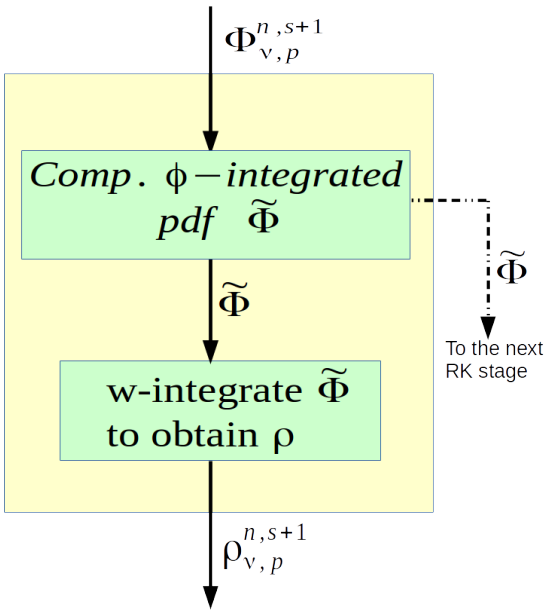

Figure 4: **DENS phase.** Performs reductions on the pdf $\Phi^{n,s}_{\nu,p,i,\ell,m}$.

(i). Angle-averaged pdf. In this step, the ϕ-integrated distribution function, noted $\tilde{\Phi}$, is computed:

$$\tilde{\Phi}_{\nu,p,i,\ell} := \Delta\phi \sum_{m=0}^{N_\phi - 1} \Phi^s_{\nu,p,i,\ell,m}. \tag{24}$$

Remark. *These computations are needed for the integration of the scattering operator in the next Runge-Kutta stage.*

(ii). Surface densities. Once $\tilde{\Phi}$ have been computed, their w-integral yields the surface densities $\varrho_{\nu,p,i}$:

$$\varrho_{\nu,p,i} = \Delta w \sum_{\ell=0}^{N_w - 1} \tilde{\Phi}_{\nu,p,i,\ell}. \tag{25}$$

2.5 The *ITER* phase

The idea is to solve the Poisson equation by seeking for the zero of this functional:

$$P[V] := -\nabla^2 V + \sum_{\nu,p} \varrho_{\nu,p}(x) \cdot |\psi_{\nu,p}[V]|^2 \tag{26}$$

via a Newton-Raphson iterative scheme. Here, we have stressed the direct dependency of the wave functions $\{\psi_{\nu,p}\}$ on the electrostatic potential V through the Schrödinger equation.
Let us describe how these phases work.

2.5.1 The Newton-Raphson scheme

The iterative scheme on (26) can be restated as

$$P\left[V^{(k)}\right] + \mathrm{d}P\left(V^{(k)}, V^{(k+1)} - V^{(k)}\right) = 0 \tag{27}$$

$V^{(0)}$ is given.

Obviously, stage $k+1$ is a refinement of the previous stage k. The derivative is meant in a directional sense. Details of the computations can be found in [Ben Abdallah et al. (2009)].

2.5.2 Schrödinger diagonalization

We can rewrite the steady-state Schrödinger equation in terms of the V-dependent linear operator

$$S[V](\Psi) := -\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{m_{z,\nu}}\frac{\mathrm{d}\Psi}{\mathrm{d}z}\right) - V\cdot\Psi.$$

We wish to compute the first N_{sbn} eigenvalues and relative eigenvectors, called *energy levels* and *wave functions*.
Remark that for this equation, both the valley ν and the position x are parameters, we shall have then $3\times N_x$ completely independent eigenproblems to solve.
In order to do this, we discretize the operator (for $\nu = 0,1,2$ and $i = 0,\ldots,N_x-1$) using finite differences thus obtaining a symmetric tridiagonal matrix of order N_z-2, being:

$$\left(\frac{\frac{1/4}{m_{\mathrm{z},\nu,i,j-1}} + \frac{1/2}{m_{\mathrm{z},\nu,i,j}} + \frac{1/4}{m_{\mathrm{z},\nu,i,j+1}}}{\Delta z^2} - V_{i,j}\right)_{j=1,\ldots,N_z-2}$$

the elements in the diagonal, and

$$\left(-\frac{\frac{1/4}{m_{\mathrm{z},\nu,i,j}} + \frac{1/4}{m_{\mathrm{z},\nu,i,j+1}}}{\Delta z^2}\right)_{j=1,N_z-3}$$

the elements in the sub-diagonal (and the super-diagonal).
The values of the effective masses $m_{\mathrm{z},\nu}$, for the particular case of the DG-MOSFET device, depend on the material:

$$m_{\mathrm{z},\nu,i,j} = \begin{cases} 0.5 & \text{if } (i,j) \text{ is in the } SiO_2 \text{ region} \\ 0.19 & \text{if } \nu < 2 \text{ and } (i,j) \text{ is in the Si region} \\ 0.98 & \text{if } \nu = 2 \text{ and } (i,j) \text{ is in the Si region} \end{cases}$$

From this matrix we extract by some method the first (lowest) N_{sbn} eigenvalues $\{\epsilon_{\nu,p,i}\}_{p\in\{0,\ldots,N_{\mathrm{sbn}}-1\}}$ and relative eigenvectors $\{\psi_{\nu,p,i,j}\}_{(p,j)\in\{0,\ldots,N_{\mathrm{sbn}}-1\}\times\{0,\ldots,N_z-1\}}$.
We take into account the boundary condition

$$\psi_{\nu,p,i,0} = \psi_{\nu,p,i,N_z-1} = 0$$

and the normalization of the eigenvectors

$$\left(\psi_{\nu,p,i,j} \longleftarrow \frac{\psi_{\nu,p,i,j}}{\sqrt{\Delta z \sum_{j'=1}^{N_z-2} |\psi_{\nu,p,i,j'}|^2}}\right)_{j=1,\ldots,N_z-2}.$$

2.5.3 Updating the potential V (construction and solution of the linear system)

One stage of the Newton-Raphson scheme on (26) translates into solving (27). (More details about the derivation can be found in [Ben Abdallah et al. (2009)].) This scheme boils down to the linear system on $V^{(k+1)}$

$$L^{(k)}\, V^{(k+1)} = R^{(k)}, \tag{28}$$

where

$$L^{(k)}\,V^{(k+1)} = -\,\mathrm{div}\left[\varepsilon_{\mathrm{R}}\,\nabla V^{(k+1)}\right] + \int \mathcal{A}^{(k)}(x,z,\zeta)\,V^{(k+1)}(x,\zeta)\,\mathrm{d}\zeta$$

$$R^{(k)} = -\,N^{(k)}(x,z) + \int \mathcal{A}^{(k)}(x,z,\zeta)\,V^{(k)}(x,\zeta)\,\mathrm{d}\zeta,$$

being $\mathcal{A}^{(k)}(x,z,\zeta) := \mathcal{A}[V^{(k)}](x,z,\zeta)$ basically the directional derivative of the density $N^{(k)} := N[V^{(k)}]$. The scheme is depicted in Figure 5.

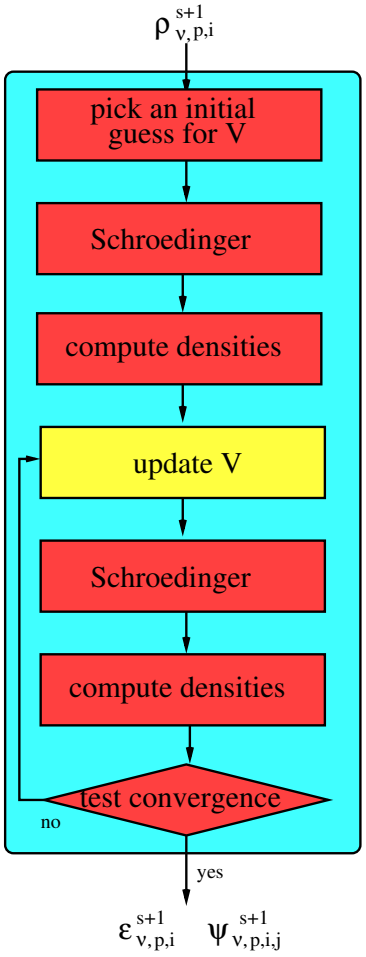

Figure 5: **ITER phase.**

The Laplacian in the linear system (28) reads

$$\mathrm{div}\left[\varepsilon_{\mathrm{R}}\,\nabla V^{(k+1)}\right] = \frac{\partial}{\partial x}\left(\varepsilon_{\mathrm{R}}\,\frac{\partial V^{(k+1)}}{\partial x}\right) + \frac{\partial}{\partial z}\left(\varepsilon_{\mathrm{R}}\,\frac{\partial V^{(k+1)}}{\partial z}\right)$$

and is discretized using finite differences

$$
\begin{aligned}
&\left(\operatorname{div}\left[\varepsilon_{\mathrm{R}}\,\nabla V^{(k+1)}\right]\right)_{i,j}\\
&=\left(\frac{\frac{1}{2}(\varepsilon_R)_{i-1,j}+\frac{1}{2}(\varepsilon_R)_{i,j}}{\Delta x^2}\right)V_{i-1,j}^{(k+1)}\\
&+\left(\frac{\frac{1}{2}(\varepsilon_R)_{i,j-1}+\frac{1}{2}(\varepsilon_R)_{i,j}}{\Delta z^2}\right)V_{i,j-1}^{(k+1)}\\
&-\left(\frac{\frac{1}{2}(\varepsilon_R)_{i-1,j}+(\varepsilon_R)_{i,j}+\frac{1}{2}(\varepsilon_R)_{i+1,j}}{\Delta x^2}\right.\\
&\left.+\frac{\frac{1}{2}(\varepsilon_R)_{i,j-1}+(\varepsilon_R)_{i,j}+\frac{1}{2}(\varepsilon_R)_{i,j+1}}{\Delta z^2}\right)V_{i,j}^{(k+1)}\\
&+\left(\frac{\frac{1}{2}(\varepsilon_R)_{i,j}+\frac{1}{2}(\varepsilon_R)_{i,j+1}}{\Delta z^2}\right)V_{i,j+1}^{(k+1)}\\
&+\left(\frac{\frac{1}{2}(\varepsilon_R)_{i,j}+\frac{1}{2}(\varepsilon_R)_{i+1,j}}{\Delta x^2}\right)V_{i+1,j}^{(k+1)}.
\end{aligned}
$$

The integral is discretized by means of trapezoid rule

$$
\begin{aligned}
&\left(\int \mathcal{A}^{(k)}(x,z,\zeta)\,V^{(k+1)}(x,\zeta)\,\mathrm{d}\zeta\right)_{i,j}\\
=&\frac{\Delta z}{2}\cdot\left[\sum_{j'=0}^{N_z-2}\mathcal{A}_{i,j,j'}^{(k)}\,V_{i,j'}^{(k+1)}+\sum_{j'=1}^{N_z-1}\mathcal{A}_{i,j,j'}^{(k)}\,V_{i,j'}^{(k+1)}\right],
\end{aligned}
\tag{29}
$$

where

$$
\begin{aligned}
\mathcal{A}_{i,j,j'}^{(k)}=2\sum_{\nu,p}\sum_{p'\neq p}&\frac{\varrho_{\nu,p,i}^{s+1}-\varrho_{\nu,p',i}^{s+1}}{\epsilon_{\nu,p',i}^{(k)}-\epsilon_{\nu,p,i}^{(k)}}\\
&\times\psi_{\nu,p,i,j'}^{(k)}\,\psi_{\nu,p',i,j'}^{(k)}\,\psi_{\nu,p',i,j}^{(k)}\,\psi_{\nu,p,i,j}^{(k)}
\end{aligned}
\tag{30}
$$

For the right hand side $R^{(k)}$, the integral is computed in a similar way to (29), and the density is simply

$$
N_{i,j}^{(k)}=2\sum_{\nu,p}\sum_{p'\neq p}\varrho_{\nu,p,i}^{s+1}\left|\psi_{\nu,p,i,j}^{(k)}\right|^2.
\tag{31}
$$

As for the boundary conditions, Dirichlet is imposed at metallic contacts (source, drain and the two gates), while homogeneous Neumann is taken elsewhere.
The matrix $L^{(k)}$ representing this linear system (28) is of order $N_x\times N_Z$, and contains N_x square blocks of size N_z on the diagonal. In Figure 6 we depict it, together with a zoom around one block. Remark that five diagonals are due to the Laplacian, the filling of the blocks are due to the integral term; the remaining non-zero values take into account the boundary conditions.

3 Implementation strategies

In this section we give the relevant details about the non-trivial parts of the parallel hybrid implementation of the numerical solver, taking into account the different computing phases (see Figure 2).

3.1 The *BTE* phase

We remind that this phase consists of three main computations:

- the integration of the scattering operator, which includes the computation of the overlap integral,
- the computation of the partial derivatives through FD-WENO-5, and
- the sums and multiplications required to implement the Runge-Kutta steps.

3.1.1 Implementation of the scattering operator

The rhs of the Boltzmann equation (see Eq. 7) is stored with the same ordering as the pdf (see Table 1 for the ordering of the arrays).

Overlap integral. The overlap integral is 5-dimensional

$$\mathcal{W}_{\nu,p,\nu',p',i}.$$

Each CUDA thread computes one point in the output array, by performing the summation in (17) for the corresponding values of (ν, p, ν', p', i). The results are stored as an array in the global *device* memory (see Table 1).

Constant magnitudes. After the precomputations, several magnitudes (the effective masses $m_{\{\mathrm{x,y,z}\},\nu}$, the Kane factors $\tilde{\alpha}_\nu$, the occupation numbers $N_{\mu,\nu\to\nu'}$, and several parameters) are computed in arrays which are moved to the device's constant memory (in total, about 200 double-precision values).

Integration of the gain part. In formula (9) we have stressed that the gain part of the collisional operator does not actually depend on m, it is thus 4-dimensional. Then, we create one thread for each (ν, p, i, ℓ) point and store them in a 4D array ordered as $i > \nu > p > \ell$. Each thread performs the following computations: once (18), three times (20) and three times (22).

The kernel is implemented in such a way to minimize accesses to global memory. In particular, this requires a reordering of the *for*-loops. Formulae (9) together with (18), (20), (22), written for human comprehension, suggest that, for any (ν, p, i, ℓ) given, the ordering be

```
1    for μ = 0, ..., 6
2        for p' = 0, ..., N_sbn − 1
3            for ν' = 0, 1, 2
4                [compute]
```

which is not a ggod idea because the magnitudes $\mathcal{W}_{\nu',p',\nu,p,i}$, $\mathcal{W}_{\nu,p',\nu,p,i}$, $\epsilon_{\nu',p',i}$, $\epsilon_{\nu,p,i}$, reside in global device

memory and we want to read them as few times as possible. To improve this, a better order is

```
1   read ϵ_{ν,p,i}
2   for p' = 0, ..., N_sbn − 1
3       perform F-type integration
4       for ν' = 0, 1, 2
5           if ν' ≠ ν
6               read W_{ν',p',ν,p,i} and ϵ_{ν',p',i}
7               for μ = 4, ..., 6
8                   [compute]
9       perform elastic integration
10      read W_{ν,p',ν,p,i} and ϵ_{ν,p',i}
11          [compute]
12      perform G-type integration
13      for μ = 1, ..., 3
14          [compute]
```

which minimizes the reading of these data from global memory.

Integration of the loss part. This part of the code has to compute once (19), three times (21) and three times (23). In order to perform that, we create one thread for each (ν, p, i, ℓ, m) point.

In order to minimize uncoalesced access to global memory for this computation, we store in the shared memory the $\epsilon_{\cdot,\cdot,i}$ and the $\mathcal{W}_{\nu,p,\cdot,\cdot,i}$ needed by a CUDA thread block. For each block, we shall allocate $3 \times N_{\text{sbn}} \times 3$ *doubles* of shared memory, the factor being due to the fact that inside one thread block, at most two different values for i are possible (remember that i comes first in the ordering, see Table 1).

3.1.2 Implementation of FD-WENO-5

.

We have tested several implementations for each of the three partial derivatives

$$\frac{\partial}{\partial x}\left[a^{1}_{\nu,\ell,\cdot}\Phi^{s}_{\nu,p,\cdot,\ell,m}\right]_i$$
$$\frac{\partial}{\partial w}\left[a^{2,s}_{\nu,p,i,\cdot,m}\Phi^{s}_{\nu,p,i,\cdot,m}\right]_\ell$$
$$\frac{\partial}{\partial \phi}\left[a^{3,s}_{\nu,p,i,\ell,\cdot}\Phi^{s}_{\nu,p,i,\ell,\cdot}\right]_m$$

and kept the most efficient one based on empirical evidence.

The x-derivative. Each thread takes care of one whole line in the x-dimension. More clearly, we create $3 \times N_{\text{sbn}} \times N_w \times N_\phi$ threads, each of them computing

$$\frac{\partial}{\partial x}\left[a^{1}_{\nu,\ell,\cdot}\cdot\Phi^{s}_{\nu,p,\cdot,\ell,m}\right]_i \quad \forall i = 0, \ldots, N_x - 1.$$

The w-derivative. The strategy is similar to that for the x-derivative, with the main difference that each

thread takes care of two lines instead of just one: for $m = 0, \dots, \frac{N_\phi}{2} - 1$, it computes

$$\frac{\partial}{\partial w}\left[a^{2,s}_{\nu,p,i,\cdot,m}\Phi^s_{\nu,p,i,\cdot,m}\right]_\ell \quad \forall \ell = 0, \dots, N_w - 1$$

and

$$\frac{\partial}{\partial w}\left[a^{2,s}_{\nu,p,i,\cdot,m+\frac{N_\phi}{2}}\Phi^s_{\nu,p,i,\cdot,m+\frac{N_\phi}{2}}\right]_\ell \quad \forall \ell = 0, \dots, N_w - 1$$

in order to perform locally on the thread the averaging described in (16).
We exploit shared memory to load from global memory the $\frac{\partial \epsilon_{\nu,p}}{\partial x}$ hence reducing costly loads.
The ϕ-derivative. Unlike the x- and w-derivatives, each (ν, p, i, ℓ, m)-point $\frac{\partial}{\partial \phi}\left[a^{3,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m$ of the partial derivative is computed by a thread. We remind that a flux splitting is needed, thus

$$\begin{aligned}&\frac{\partial}{\partial \phi}\left[a^{3,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m \\ = \underbrace{\frac{\partial}{\partial \phi}\left[a^{3+,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m}_{\text{information from the left}} &+ \underbrace{\frac{\partial}{\partial \phi}\left[a^{3-,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m}_{\text{information from the right}}.\end{aligned}$$

Hence, for fixed (ν, p, i, ℓ), the partial derivative at point ϕ_m depends on the seven points $\{\phi_{m-3}, \phi_{m-2}, \phi_{m-1}, \phi_m, \phi_{m+1}, \phi_{m+2},$ six for $\frac{\partial}{\partial \phi}\left[a^{3+,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m$ and six for $\frac{\partial}{\partial \phi}\left[a^{3-,s}_{\nu,p,i,\ell,\cdot}\Phi^s_{\nu,p,i,\ell,\cdot}\right]_m$, as sketched in Figure 7.
If we use N_{TPB} threads per block, this means that we need $N_{\text{B}} := \left\lceil \frac{3 \times N_{\text{sbn}} \times N_x \times N_w \times N_\phi}{N_{\text{TPB}}} \right\rceil$ blocks in total.
Each block will take care of N_{TPB} points, thus involving at most N_{S} different (ν, p, i, ℓ)-points where:

$$N_{\text{S}} = \left\lceil \frac{N_{\text{TPB}} - 1}{N_\phi} \right\rceil + 1.$$

N_{S} represents the number of ϕ-lines or sections, partially or totally included to compute the output of a thread block. Therefore, we declare a shared memory vector *sm* with

$$2 \times \underbrace{N_{\text{S}} \times (N_\phi + 6)}_{\text{sm\_sz}}$$

doubles. The factor $2\times$ is required to take into account the flux splitting

$$a^{3,s}_{\nu,p,i,\ell,m} = a^{3+,s}_{\nu,p,i,\ell,m} + a^{3-,s}_{\nu,p,i,\ell,m}$$

in such a way that if

$$\text{sm}\,[k] \longleftrightarrow a^{3+,s}_{\nu,p,i,\ell,m} \cdot \Phi^s_{\nu,p,i,\ell,m},$$

then

$$\text{sm}\,[k + \text{sm\_sz}] \longleftrightarrow a^{3-,s}_{\nu,p,i,\ell,m} \cdot \Phi^s_{\nu,p,i,\ell,m}.$$

Moreover, the $+6$ takes into account the boundary conditions (FD-WENO-5 requires 3 ghost points at each border).
The mapping from the global memory to the shared memory is sketched in Figure 8, for just the positive flux.
We proceed in the following manner for each block:

1. Compute information for the particular block:

- the first index in the block:
$$\text{fi} = \text{blockIdx} \times N_{\text{TPB}},$$
- the number of sections in the block:
$$N_{\text{S}} = \left\lceil \frac{N_{\text{TPB}} - 1}{N_\phi} \right\rceil + 1,$$
- the number of points per section:
$$N_{\text{tp}} = N_{\text{S}} \times N_\phi,$$
- the block's displacement with respect to the beginning of the first line
$$\text{gap} = \text{fi} \mod N_\phi,$$
- the 1D-global index associated with the thread:
$$\text{gi} = \text{blockIdx} \times N_{\text{TPB}} + \text{threadIdx},$$
- the thread's section index
$$\text{si} = \left\lfloor \frac{\text{gap} + \text{threadIdx}}{N_\phi} \right\rfloor.$$

2. Compute the 5D indices of the point that the thread will compute:
$$\text{gi} \mapsto (\nu, p, i, \ell, m)\,.$$
3. Load data from global memory to the shared memory vector *sm*:
$$\begin{aligned}&\text{for } M = 0, \ldots, N_{\text{tp}} - 1\\ &\quad \text{sm}_{3+6\times\left\lfloor \frac{M}{N_\phi} \right\rfloor + M} = \Phi[\text{fi} - \text{gap} + M]\\ &\quad \text{(same for the negative flux)}\end{aligned}$$
4. Introduce boundary conditions into the shared memory vector *sm*:
$$\begin{aligned}&\text{for } s = 0, \ldots, N_{\text{S}} - 1\\ &\quad \text{for } M = 1, 2, 3\\ &\quad\quad sm_{3+6\times s - M} \longleftarrow \text{bound. cond.}\\ &\quad\quad \text{(same for the negative flux)}\\ &\quad\quad sm_{3+6\times s + N_\phi - 1 + M} \longleftarrow \text{bound. cond.}\\ &\quad\quad \text{(same for the negative flux)}\end{aligned}$$
5. Synchronize the threads in the block
6. Compute the partial derivative using the data in *sm*:
$$\begin{aligned}&\text{for } M = 0, \ldots, N_{\text{TPB}} - 1\\ &\quad \text{compute } \left(\frac{\partial \Phi^s}{\partial \phi}\right)_{\nu,p,i,\ell,m}.\end{aligned}$$

3.1.3 The Runge-Kutta calculations

Each thread takes care of one (ν, p, i, ℓ, m) point. Computations are structured so as to have coalescence in reading and writing from and to the global device memory.

3.2 The *DENS* phase

We remind that this phase performs the two computations (24) and (25).
Angle-averaged pdf. We create $3 \times N_{\text{sbn}} \times N_x \times N_w$ threads, each of them performing the calculation (24) to compute a particular $\tilde{\Phi}_{\nu,p,i,\ell}$.
In order to exploit the on-chip memory to perform the local computations, we employ a shared memory vector to load the pdf $\Phi^s_{\nu,p,i,\ell,m}$ from the global memory.
Hence, in order to fully exploit the shared-memory bandwidth and the global memory coalescent reading, we perform the global–to–shared-memory load by an external *for*-loop where consecutive threads in a warp read consecutive values in the DRAM that are written in contiguous positions of the shared memory (see Figure 9).
Once the data have been loaded, integration must be performed. Each thread wants to sum N_ϕ consecutive values of the sm shared memory vector. In particular, the thread indexed I wants to compute:

$$\sum_{m=0}^{N_\phi - 1} \text{sm}\left[I \times N_\phi + m\right].$$

We have to be careful to avoid bank conflicts, the shared memory is arranged by banks, as sketched in Figure 9. As a consequence, although 32 consecutive *doubles* can be accessed simultaneously in one clock cycle, if the 32 threads of a warp try to access 32 different positions that are congruent modulo 32 (belonging to the same bank), this operation might require up to 32 cycles (might be fewer depending on the capabilities of the GPU).
Considering that as soon as $N_\phi > 32$ bank conflicts are inevitable, we can at least reach the minimum number of conflicts by shifting the starting point "diagonally" and summing "circularly". More clearly, this means that thread which computes the I-th element of the output vector Φ^s performs the following sum instead:

$$\sum_{m=I}^{I+N_\phi - 1} \text{sm}\left[I \times N_\phi + (m \mod N_\phi)\right].$$

Surface densities. For each (ν, p, i) we have to perform the integration in (25). Each CUDA thread takes care of one point. As in the pdf vector $\tilde{\Phi}_{\nu,p,i,\ell}$ the ℓ-index comes last in the ordering, this guarantees a good coalescence in reading.

3.3 The *ITER* phase

This phase is currently computed entirely on the CPU, and is, at the state of the art, the more complex part of the parallel code.

3.3.1 Schrödinger diagonalization

The computations described in 2.5.2 are performed on the host (CPUs) using a parallel description based on OpenMP directives. The $3 \times N_x$ eigenvalue problems are evenly divided into the available threads. Each thread solves the assigned eigenvalue problems by using the LAPACK routine `dsgetr` [Anderson et al. (1999)].

	ILUT (1 core)	**ILUT** (16 cores)
BiCG	0.764657	0.196927 (3.8x)
BiCGSTAB	0.659210	**0.148999** (4.4x)
GPBiCG	0.686384	0.199920 (3.4x)
BiCGSafe	0.660219	0.176699 (3.7x)
IDR	**0.645309**	0.165456 (3.9x)
BiCR	0.772417	0.207286 (3.7x)
CRS	0.689740	0.158914 (4.3x)
BiCRSTAB	0.675243	0.158104 (4.2x)
GPBiCR	0.705240	0.207514 (3.3x)
BiCRSafe	0.667985	0.184428 (3.8x)

Table 2: **Linear system.** The cost-per-step of solving the linear system, in LIS implementation using OpenMP, for different preconditioner/solver couplings, using 16 cores, with the standard meshes $N_x = N_z = 65$. Tolerance parameter is set 10^{-8}.

3.3.2 Newton-Raphson steps

The magnitude (30)

$$\mathcal{A}^{(k)}_{i,j,j'} = 2\left\{\sum_{\nu=0}^{2}\left[\sum_{p=0}^{N_{\text{sbn}}-1}\left(\sum_{p'}^{N_{\text{sbn}}-1}\frac{\varrho^{s}_{\nu,p,i}-\varrho^{s}_{\nu,p',i}}{\epsilon^{(k)}_{\nu,p',i}-\epsilon^{(k)}_{\nu,p,i}}\right.\right.\right.$$
$$\left.\times\psi^{(k)}_{\nu,p',i,j'}\cdot\psi^{(k)}_{\nu,p',i,j}\cdot\mathbb{I}\left[p'\neq p\right]\right)$$
$$\left.\left.\times\psi^{(k)}_{\nu,p,i,j'}\cdot\psi^{(k)}_{\nu,p,i,j}\right]\right\}. \tag{32}$$

is 3-dimensional. The computations are performed on the CPU with an OpenMP parallel environment.
Preconditioning and solution of the linear system The linear system is of order $N_x \times N_Z$, thus the order is 4225 in our experiments with standard meshes ($N_x = N_z = 65$), the matrix is banded and out of $17,850,625$ entries only $\approx 250,000$ are non-zero on 129 diagonals; the system is, therefore, more than 98 % sparse.
We have performed an extensive battery of tests coupling several preconditioners and solvers as implemented in the LIS (Library of Iterative Solvers) [Nishida (2010), Kotakemori (2005)], which allows for multi-core building using OpenMP. We have empirically chosen the strategy that fits best the problem for standard meshes. From the tests, it is clear that the only suitable preconditioners are those of the Incomplete LU factorization (ILU) family [Saad (2003)], the others being unstable or slow.
In Table 2, we only report the times obtained through the ILUT preconditioner [Saad (2003)], because it performs better than ILU. We see that while on a sequential execution ILUT-IDR is the best strategy, the coupling of ILUT with the BiCGSTAB solver [Saad (2003)] exploits better a 16-core configuration being faster than a 16-core execution of the ILUT preconditioner in conjunction with an IDR (Induced Dimension Reduction) solver [Sonneveld (2008)].

3.4 Overlapping the computations

In order to take advantage of the CPU and the GPU operating in parallel on independent calculations, it is best to re-organize the computations by decomposing the BTE phase (in particular, the computation of the partial derivatives though FD-WENO-5) in order to make it partially overlap with the other phases.
As the a^1-flux in the x-derivative

$$\frac{\partial}{\partial x}\left[a^{1}_{\nu,\ell,m}\cdot\Phi^{s+1}_{\nu,p,\cdot,\ell,m}\right]$$

is constant, this computation can be performed for the next Runge-Kutta stage $(s+1)$ while the the computation of the the eigenstates $\left\{\epsilon^{s+1}_{\nu,p,i}, \psi^{s+1}_{\nu,p,i,j}, V^{s+1}_{i,j}\right\}$ (the *ITER* phase corresponding to the s-th Runge-Kutta stage) is performed on the *host*.
Moreover, in order to optimize the data transfer between the *host* to the *device* DRAM, the memory for $\varrho_{\nu,p,i}$ and the eigenstates $\left\{\epsilon_{\nu,p,i}, \left(\frac{\partial \epsilon}{\partial x}_{\nu,p,i}\right), \psi_{\nu,p,i,j}\right\}$ is allocated as *page-locked* (aka *pinned*), though the copies cannot be made asynchronous.
This optimized scheme is summarized in Figure 10.

4 Numerical experiments

In the following we analyze the performances of our hybrid parallel code.

4.1 Heterogeneous platform

All experiments were performed on a server with dual Intel Xeon ES-2630 CPUs (16 cores total) with 64 GB RAM, and a 1 TB solid state hard drive. The system includes one Nvidia Tesla K40(c) GPU [NVIDIA(2012)] with 2880 CUDA cores and 12 GB of GDDR5 memory. The operating system is Linux Centos 7.3 with GCC version 4.8.5 and the CUDA 7.5 runtime.

4.2 Source code structure

We distinguish the CUDA code to be executed on GPU from the OpenMP code which run on CPU:

- **GPU code.** The code contains 16 CUDA kernels, 12 of which listed in Table 3, ordered by their decreasing computational weight. The remaining 4 are either only executed once for initialization, or seldom just for computing some macroscopic data to be stored in files for visualization.
- **OpenMP code.** Inside the *ITER* block, three functions exploit *OpenMP* for shared-memory parallel environment, they are listed in Table 3 in their execution order.

4.3 Overview

In Figure 11 we sketch the absolute (in seconds) and relative weight (in percentage) of the three main computational phases (BTE, ITER and DENS). This study has been obtained by executing one time step of three versions of the solver for an standard mesh ($N_{\text{sbn}} = 6$, $N_x = 65$, $N_z = 65$, $N_w = 300$, $N_\phi = 48$) on the above-mentioned platform:

- **sequential**: a sequential version using one CPU thread.
- **omp**: an OpenMP parallel version of the full solver using 16 CPU threads.
- **omp+cuda**: the parallel hybrid solver using 16 CPU threads.

From now on, we will assume this configuration for the runtime experiments (one time step with the standard mesh and 16 CPU threads) unless we indicate explicitly variations in this configuration. As can be seen, the per-step cost of the hybrid solver is dominated by the *ITER* phase, followed by the *BTE* with a lesser but comparable cost. Although the BTE phase is the costliest part in the sequential and OpenMP solvers, the complete adaptation to GPU of this phase in the hybrid solver allows to reduce considerably its weight in the runtime. As a consequence, in the hybrid parallel solver the *ITER* phase is the costliest part because this block involves complex computations which despite of having been parallelized on CPU, they have not been adapted to the GPU. On the other hand, the weight of *DENS* phase is not very relevant in any version of the solver.

rank	kernel/function name	phase	computation	times/step
1	GPU_integrate_PHONONS_loss	BTE	(19), (21), (23)	3
2	GPU_approx_partf_PHI	BTE	(15)	3
3	GPU_approx_partf_W	BTE	(14)	3
4	GPU_approx_partf_X	BTE	(13)	3
5	GPU_set_fluxes_a3	BTE	(6)	3
6	GPU_compute_integrated_pdf_energy	DENS	(24)	3
7	GPU_integrate_PHONONS_gain	BTE	(18), (20), (22)	3
8	GPU_perform_RK_2_3	BTE	(11)	1
9	GPU_perform_RK_3_3	BTE	(12)	1
10	GPU_perform_RK_1_3	BTE	(10)	1
11	GPU_compute_Wm1	BTE	(17)	3
12	GPU_integrated_phitilde	DENS	(25)	3
13	compute_eigenstates	ITER	(2)	?
14	compute_kernel	ITER	(30)	?
15	lis_solve	ITER	(28)	?

Table 3: **Hybrid GPU-CPU parallelization.** CUDA kernels are listed 1-12, OpenMP-based methods are 13-15. The Library for Iterative Solvers (LIS) is built using OpenMP. Right column: the number of calls inside one time step; symbol *?* is used to mean unpredictability.

4.4 The CUDA kernels (*BTE* and *DENS* phases)

In Figure 12 we sketch the absolute weight of the main computational steps (in seconds) for the *BTE* phase and also the relative weight (regarding the total execution time obtained with each solver) for these steps. The integration of the loss part is clearly the most weighted computing phase in all the solvers (strongly in the sequential solver) but it is also the phase which benefits to a greater extent from an efficient parallelization (strongly in the omp+cuda solver). On the other hand, the effect of the parallelization of the remaining phases is very similar.
Table 4 provides data to evaluate the quality of the implementation for the most important kernels. These data have been obtained by using the `nvprof` utility [NVIDIA CUDA(2018)] when the parallel CUDA-OpenMP solver completed one time step.
Efficiency measures are split into two big categories: the *computational throughput* measures the throughput of FLoating-point OPerations per Second (FLOPS) they perform, while the *data throughput* measures the throughput of data between the SM's and the various memories (registers, local, shared, L1 and L2 cache, global). A good behavior in both these aspects is needed.

- **Computational throughput.** As shown in columns ii and iii, the heaviest kernels (kernels 1-4) achieve a good performance (GFlops per second) in comparison with the peak double-precision floating point performance of the device (1.43 TeraFlops per second).
- **Data throughput.** We recall that the GPU possesses several memory regions, accessing which is as much costlier as the memory region is physically distant from the ALU's. Thus, in increasing order of distance, we have: registers, shared memory/L1 cache L2 cache and finally the off chip DRAM memory. Obviously, the closer the data exploited for computation, the faster the access to them. Follow some comments on the data in Table 4.
  - Column iv-v. The L2 cache is used to cache access to global (and local) memory. Kernels with large L2 hit are those reading several times from the DRAM the same values of some variable: in particular, in kernel 5 $N_w \times N_\phi$ threads read $\epsilon_{\nu,p,i}$, in kernel 7 reads N_ϕ threads read $\tilde{\Phi}_{\nu,p,i,\ell}$, in kernel 11, values of $\psi_{\nu,p,i,j}$ are recycled several times. Empirical tests show that for these kernels it is best to avoid loading these vectors from global memory to shared memory, and rather let the L2 cache operate.

i	ii	iii	iv	v	vi	vii	viii
	avg. time	GFlops/s	L2 hit	L2 RT	GMLE	RGLT/peak	LMEM
1	32.8 ms	539	11.15 %	8.75 GB/s	27.86 %	4.8 %	0 B
2	10.7 ms	598	1.01 %	35.77 GB/s	74.99 %	13.4 %	0 B
3	10.6 ms	284	1.56 %	69.18 GB/s	66.59 %	8 %	7304 B
4	6.66 ms	389	3.83 %	79.55 GB/s	94.52 %	13.3 %	1144 B
5	2.92 ms	226	93.42 %	47.22 MB/s	6.25 %	0.4 %	0 B
6	1.81 ms	9	0 %	69.75 GB/s	100 %	24.2 %	0 B
7	2.47 ms	275	94.72 %	26.02 GB/s	61.00 %	60 %	0 B
8	3.59 ms	28	0 %	104.53 GB/s	100 %	36.2 %	0 B
9	3.59 ms	28	0 %	104.56 GB/s	100 %	36.3 %	0 B
10	2.90 ms	11	0 %	86.45 GB/s	100 %	30 %	0 B
11	.297 ms	16	96.74 %	221.72 GB/s	6.17 %	7.6 %	0 B
12	.160 ms	2	63.21 %	42.96 GB/s	6.22 %	5.5 %	0 B

Table 4: **Measures of the CUDA kernels' efficiency.** The kernels are listed in decreasing computational weight.

Column i: The kernels' numbering, as in Table 3.

Column ii-iii: Computational throughput. Column ii expresses the total per-step computational time needed and column iii expresses the achieved gigaflops per second (peak is 1430 GFlops/s).

Columns iv-vii: Data throughput. Here, *L2 RT* stands for *L2 Read Throughput*, *GMLE* stands for *Global Memory Load Efficiency*, *RGLT* stands for *Requested Global Load Throughput* (peak is 288 GB/s).

Column viii: Spilling to *Local Memory* (*LMEM*).

- Column vi. GMLE measures the how much bandwidth is wasted for a non-coalescent reading from the global memory. The Tesla K40(c) uses a 384-bit-wide memory interface; this means that at least 6 double-precision floating-point values are accessed in DRAM each time a read request is sent, even if just one value were wanted. Our code achieves reasonable performances from this point of view.
- Column viii. The maximum register size (255) is never exceeded, nevertheless LMEM (Local MEMory) is used to allocate arrays, that cannot reside on on-chip memory. Also, this affects the exploitability of the L2 cache for "hits".

4.5 The OpenMP methods (*ITER* phase)

In Figure 13, we sketch the absolute weight of the main computational steps (in seconds) for the *ITER* phase and also the relative weight (regarding the total execution time for each solver) for these steps. We can observe that the omp+cuda solver obtains slightly worst execution times than the the omp solver because of the effect of the data transfers between the host and the device. The Figure remarks again the relevant relative weight of the ITER phase in the omp+cuda solver.

Schrödinger. In Figure 15 (lower right corner), we can see the speedup obtained in the Schrödinger diagonalization step by the omp solver with respect to the sequential solver for the standard meshes $N_x = N_z = 65$ with several number of threads. As can be seen, the CPU-parallel part scales properly, though it does not attain the theoretical speedup of 16x (for 16 cores), rather close to 11x.

4.6 Scaling with respect to the meshes

In Figure 14, we sketch the weight of each computational phase for 1D sweeps with respect to the meshes taken as reference $N_{\text{sbn}} = 6$, $N_x = 65$, $N_z = 65$, $N_w = 300$, $N_\phi = 48$.

Version	step	BTE	DENS	ITER	FD-WENO-5	scatt.	lin. sys.	Schröd.	NR kernel
sequential	19.82	18.56	0.044	1.22	3.69	13.70	0.59	0.23	0.37
OMP 2-core	10.05	9.35	0.024	0.67	1.87	6.89	0.34	0.12	0.19
OMP 4-core	5.60	5.18	0.0146	0.40	1.03	3.82	0.207	0.066	0.106
OMP 6-core	4.05	3.72	0.011	0.317	0.74	2.74	0.167	0.048	0.076
OMP 8-core	3.32	3.04	0.0103	0.27	0.61	2.24	0.145	0.04	0.063
OMP 10-core	2.83	2.58	0.0086	0.24	0.618	1.8	0.136	0.032	0.051
OMP 12-core	2.39	2.16	0.0075	0.21	0.53	1.5	0.119	0.026	0.042
OMP 14-core	2.07	1.85	0.0068	0.208	0.46	1.28	0.121	0.022	0.036
OMP 16-core	1.73	1.53	0.0062	0.199	0.31	1.12	0.119	0.02	0.032
OMP 16-core/GPU	0.47	0.21	0.00618	0.26	0.087	0.105	0.15	0.03	0.05

Table 5: **Execution time.** Execution time of each parallel computational phase for one time step using a standard mesh.

Multiplying by 2 the number of points in either dimension produces a different result:

- For N_{sbn}, it increases the relative weight of the scatterings: this is because N_{sbn} has quadratic influence on the integration formula. Moreover, it increases the cost of the Newton-Raphson kernel (30) and is a parameter for all the transport part.
- For N_{x}, not only is it a parameter for the transport part and the Schrödinger equation, it increases the order of the linear system.
- For N_{z}, it does not affect the transport part at all, but it increases the order of both the linear system and the eigenvalue problem

$$-\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{m_{z,\nu}}\frac{\mathrm{d}\psi_{\nu,p}}{\mathrm{d}z}\right) - V\,\psi_{\nu,p} = \epsilon_{\nu,p}\,\psi_{\nu,p}.$$

- N_w and N_ϕ are essentially parameters which affect the execution time of the FS-WENO-5 computation in the BTE phase.

4.7 Speedup

In Figure 15 we examine the speedup achieved by our strategies. We compare the performances of the CUDA/OpenMP hybrid code to those of a pure OpenMP code. The details are given in Table 5.
In order to evaluate the benefits of using the GPU, we compare the last two lines in Table 5, i.e. a pure OpenMP 16-core parallelization (without GPU) and a OpenMP 16-core/GPU parallelization. We remind that, in our hybrid code, *BTE* and *DENS* phases are parallelized on the GPU, while for *ITER* nothing changes.
For the average cost of one time step, we achieve a speedup of 3.6x, due to the *BTE* phase achieving a speedup of 7.1x. Inside that phase, the most beneficial parallelization is the one referring the integration of the scattering operator, with a factor 10.6x, while the FD-WENO-5 computation achieves a factor 3.5x. This is somehow expected, because the scatterings require much more computations and hence the GPU can be exploited better.
As for the *DENS* and the *ITER* parts, they do not enjoy benefits by sending the *BTE* computations to the GPU, because some data transfer between the host and the device memories slow them down.

5 Conclusions and perspectives

We have proposed a successful implementation for an extremely heavy code from the computational point of view. While a purely sequential code on CPU takes many days, the computations can now be performed within some hours, and the code can be used to obtain reference results for other macroscopic solvers.

The computation of the eigenstates (the *ITER* block) is, at the state of the art, what is most time-consuming in the code. Any effort of improvement should, therefore, be focused on implementing it fully on the GPU. Moreover, this would would also avoid any memory transfer between the host and the device memories.
In particular, the construction, preconditioning and solution of the linear system should be the first part to port to GPU computing. The second task is the port to GPU of the eigenvalue problem.
From the modelling point of view, an eighth scattering phenomenon should be added, namely the *surface roughness*, aiming at taking into account the irregularities in the silicon thickness of a real device. These deviations make more charge appear near the interfaces, and at such small size can significantly modify the device's behavior.

Acknowledgements. Francesco Vecil and J. M. Mantas acknowledges the project MTM2014-52056-P funded by the Spanish Ministerio de Economía y Competitividad (MINECO) and the European Regional Development Fund (ERDF/FEDER).

References

[Vecil et al. (2014)] F. Vecil and J. M. Mantas and M. .J Cáceres and C. Sampedro and A. Godoy and F. Gámiz (2014) *A parallel deterministic solver for the Schrödinger–Poisson–Boltzmann system in ultra-short DG-MOSFETs: Comparison with Monte-Carlo*, Computers and Mathematics with Applications 67 (9) 1703–1721. doi:http://dx.doi.org/10.1016/j.camwa.2014.02.021.

[Ben Abdallah et al. (2009)] Ben Abdallah N, Cáceres MJ, Carrillo JA, Vecil F (2009) *A deterministic solver for a hybrid quantum-classical transport model in nanoMOSFETs*, Journal of Computational Physics 228 (17) 6553–6571. doi:10.1016/j.jcp.2009.06.001.

[Mantas et.al. (2009)] J. M. Mantas, M. J. Cáceres(2009), Efficient deterministic parallel simulation of 2D semiconductor devices based on WENO-Boltzmann schemes, Computer Methods in Applied Mechanics and Engineering 198 (5–8) 693 – 704.

[Carrillo et al. (2003)] J. A. Carrillo, I. M. Gamba, A. Majorana, C.-W. Shu, *A WENO-solver for the transients of Boltzmann-Poisson system for semiconductor devices: performance and comparisons with Monte Carlo methods*, Journal of Computational Physics 184 (2) (2003) 498–525. doi:10.1016/S0021-9991(02)00032-3.

[Suzuki (2015)] A. Suzuki, T. Kamioka, Y. Kamakura, T. Watanabe (2015), *Particle-based Semiconductor Device Simulation Accelerated by GPU computing*, Journal of Advanced Simulation in Science and Engineering 2 (1)1, 211–224.

[Li (2005)] Y. Li, S-M. Yu (2005) *A parallel adaptive finite volume method for nanoscale double-gate MOSFETs simulation*, Journal of Computational and Applied Mathematics 175 , 87–99.

[Li (2003)] Y. Li, T.-S. Chao, S.M. Sze (2003) *A Novel Parallel Approach for Quantum Effect Simulation in Semiconductor Devices*, International Journal of Modelling and Simulation 23:2 94–102.

[Kumar (2016)] G. Kumar, M. Singh, A. Bulusu, G. Trivedi (2016) *A Framework to Simulate Semiconductor Devices Using Parallel Computer Architecture*, Journal of Physics: Conference Series 759:1.

[Anderson et al. (1999)] E. Anderson, Z. Bai et al (1999), *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics. Third Edition.

[Camiola (2013)] V. D. Camiola, G. Mascali, V. Romano (2013), *Simulation of a double-gate MOSFET by a non-parabolic energy-transport subband model for semiconductors based on the maximum entropy principle*, Mathematical and Computer Modelling 58 (1-2) 321–343.

[Suhag (2017)] Ashok Kumar Suhag, Rakesh Sharma (2017), *Design and Simulation of Nanoscale Double Gate MOSFET using high K Material and Ballistic Transport Method* , Materials Today: Proceedings 4 10412–10416.

[Prasher (2013)] Rakesh Prasher, Devi Dass, Rakesh Vaid (2013), *Performance of a double gate nanoscale MOSFET (DG-MOSFET) based on novel channel materials* , Journal of Nano- and Electronic Physics 5(1).

[Mascali et.al. (2012)] G. Mascali, V. Romano (2012), *A non parabolic hydrodynamical subband model for semiconductors based on the maximum entropy principle*, Mathematical and Computer Modelling 55 (3-4) 1003–1020.

[Ben et.al. (2004)] N. Ben Abdallah, F. Méhats, N. Vauchelet (2004), *A note on the long time behavior for the drift-diffusion-Poisson system*, C. R. Math. Acad. Sci. Paris 339 (10) 683–688.

[Ungel (2010)] A. Ungel (2010), *Energy transport in semiconductor devices*, Mathematical and Computer Modelling of Dynamical Systems 16(1).

[Ringhofer (2001)] C. Ringhofer, C. Schmeiser, A. Zwirchmayr (2001) *Moment Methods for the Semiconductor Boltzmann Equation on Bounded Position Domains*, SIAM Journal on Numerical Analysis , 39:3, 1078-1095,

[Rupp (2011)] K. Rupp, T. Grasser, A. Jungel (2011), *On the feasibility of spherical harmonics expansions of the Boltzmann transport equation for three-dimensional device geometries*, in: Electron Devices Meeting (IEDM), 2011 IEEE International, 2011, 34.1.1–34.1.4.

[Hong (2011)] S.-M. Hong, A. T. Phòam, C. Jungemann (2011), *Deterministic solvers for the Boltzmann transport equation*, Springer.

[Brodtkorb (2013)] A.R. Brodtkorb, T.R. Hagen, M.L. Sætra (2013), *Graphics processing unit (GPU) programming strategies and trends in GPU computing.* J. Parallel Distrib. Comput. 73(1), 4–13.

[Ujaldon (2012)] M. Ujaldon (2012), *High performance computing and simulations on the GPU using CUDA.* In: 2012 International Conference on High Performance Computing & Simulation, HPCS 2012, Madrid, Spain, July 2-6, 1–7

[Owens (2008)] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone., J.C. Phillips (2008) *GPU computing.* Proceedings of the IEEE 96(5), 879–899.

[NVIDIA CUDA Home(2018)] NVIDIA: *CUDA Zone.* https://developer.nvidia.com/cuda-zone (accessed March 2018).

[NVIDIA CUDA(2018)] NVIDIA: *CUDA Toolkit Documentation. Profiler User's Guide..* https://docs.nvidia.com/cuda/profiler-users-guide/ (accessed March 2018).

[NVIDIA CUDA C (2018)] NVIDIA: *CUDA C Programming Guide.* http://docs.nvidia.com/cuda/cuda-c-programming-guide/ (accessed March 2018).

[Asunción (2011)] M. de la Asunción, J.M. Mantas, M.J. Castro (2011) *Simulation of one-layer shallow water systems on multicore and CUDA architectures.* Journal of Supercomputing 58(2), 206–214

[Castro (2011)] M.J. Castro, S. Ortega, M. de la Asunción, J.M. Mantas, J.M. Gallardo (2011) *GPU computing for shallow water flow simulation based on finite volume schemes.* Comptes Rendus Mécanique 339(2–3), 165–184.

[Asunción (2013)] M. de la Asunción, M.J. Castro, E.D. Fernández-Nieto, J.M. Mantas, S. Ortega, J.M. González (2013) *Efficient GPU implementation of a two waves TVD-WAF method for the two-dimensional one layer shallow water system on structured meshes.* Computers & Fluids 80, 441 – 452.

[Abdi (2017)] D.S. Abdi, L. C. Wilcox, T. C. Warburton, F. X. Giraldo (2017) *A GPU-accelerated continuous and discontinuous Galerkin non-hydrostatic atmospheric model.* The International Journal of High Performance Computing Applications 1 – 29.

[Norouzi (2017)] Y. Ye, K. Li, Y. Wang, T. Dengz (2017) *New hybrid CPU-GPU solver for CFD-DEM simulation of fluidized beds.* Powder Technology 316, 233–244.

[DeVries (2013)] B. DeVries, J. Iannelli, C. Trefftz, K. A. O'Hearn, G. Wolffez (2013) *Parallel implementations of FGMRES for solving large, sparse non-symmetric linear systems.* Procedia Computer Science 18 491–500.

[Ye (2015)] Y. Ye, K. Li, Y. Wang, T. Dengz (2015) *Parallel computation of Entropic Lattice Boltzmann method on hybrid CPU–GPU accelerated system.* Computers & Fluids 110 114–121.

[Rupp (2012)] K. Rupp, A. Jüngel , T. Grasser (2012) *A GPU-Accelerated Parallel Preconditioner for the Solution of the Boltzmann Transport Equation for Semiconductors.* Lecture Notes in Computer Science, vol 7174, 147–157. Springer, Berlin, Heidelberg

[Karaivanova (2013)] A. Karaivanova, E. Atanassov, T. Gurov (2013) *Monte Carlo Simulation of Ultrafast Carrier Transport: Scalability Study.* Procedia Computer Science, 18 2298–2306.

[Rupp (2016)] K. Rupp, Ph. Tillet, F. Rudolf, J. Weinbub, A. Morhammer, T. Grasser, A. Jüngel, S. Selberherr (2016) *ViennaCL - Linear Algebra Library for Multi- and Many-Core Architectures* SIAM Journal on Scientific Computing, 38 (5), S412 – S439.

[Chapman (2008)] B. Chapman, G. Jost, R. van der Pas (2008) *Using OpenMP: Portable Shared Memory Parallel Programming* The MIT Press.

[NVIDIA(2012)] NVIDIA. *NVIDIA's Next Generation CUDA TM Compute Architecture: Kepler GK110.* https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf (accessed March 2018).

[Nishida (2010)] A. Nishida (2010) *Experience in Developing an Open Source Scalable Software Infrastructure in Japan.* Lecture Notes in Computer Science, 6017, 448–462.

[Kotakemori (2005)] H. Kotakemori, H. Hasegawa, A. Nishida (2005) *Performance Evaluation of a Parallel Iterative Method Library using OpenMP* Proceedings of the 8th International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2005). Beijing: IEEE, 432–436.

[Saad (2003)] Y. Saad. (2003) *Iterative Methods for Sparse Linear Systems.* SIAM.

[Sonneveld (2008)] P. Sonneveld and M. B. van Gijzen. (2008) *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems.* SIAM J. Sci. Comput., 31(2), p.1035—1062.

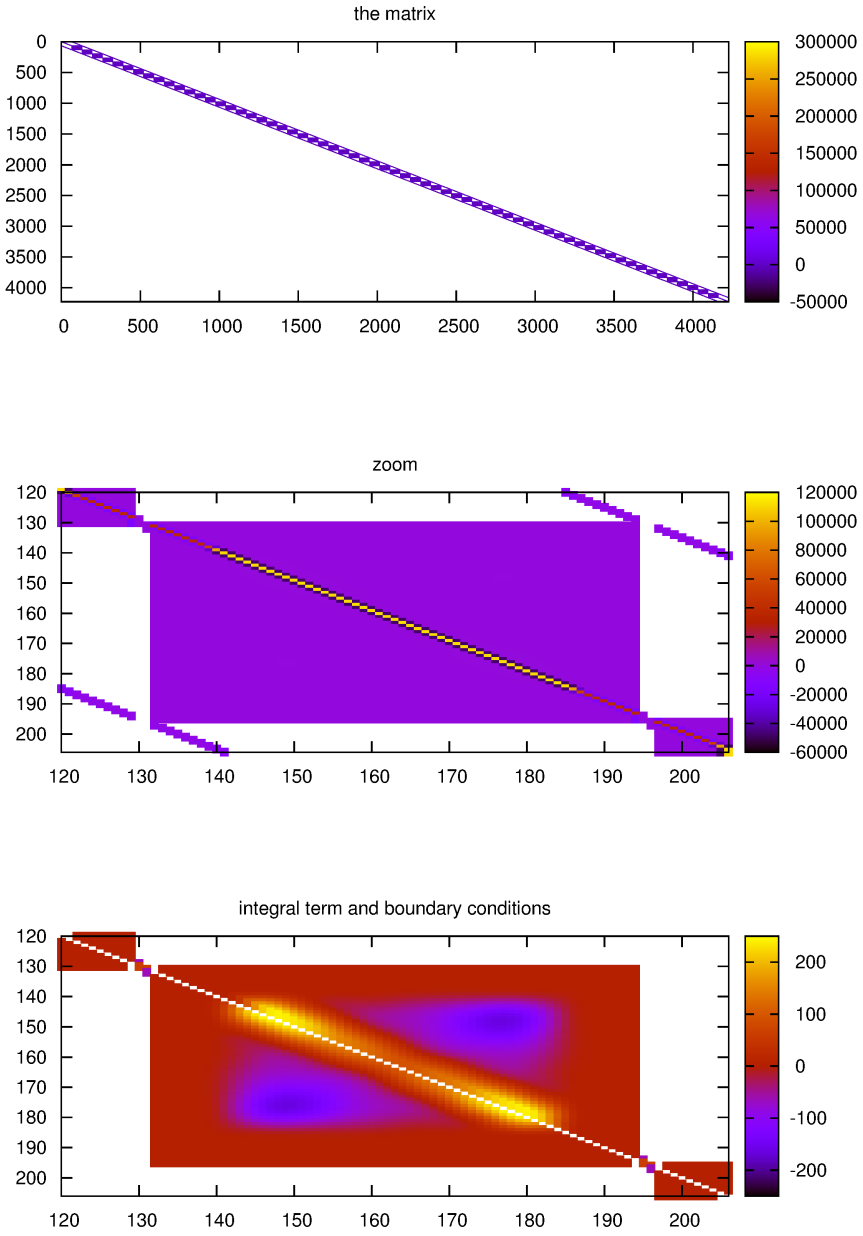

Figure 6: **The ITER block.** The matrix $L^{(k)}$ of the linear system matrix (28).

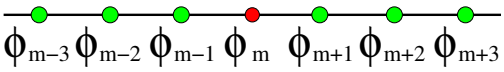

Figure 7: **Point stencil in FD-WENO-5 to compute the ϕ-derivative.**

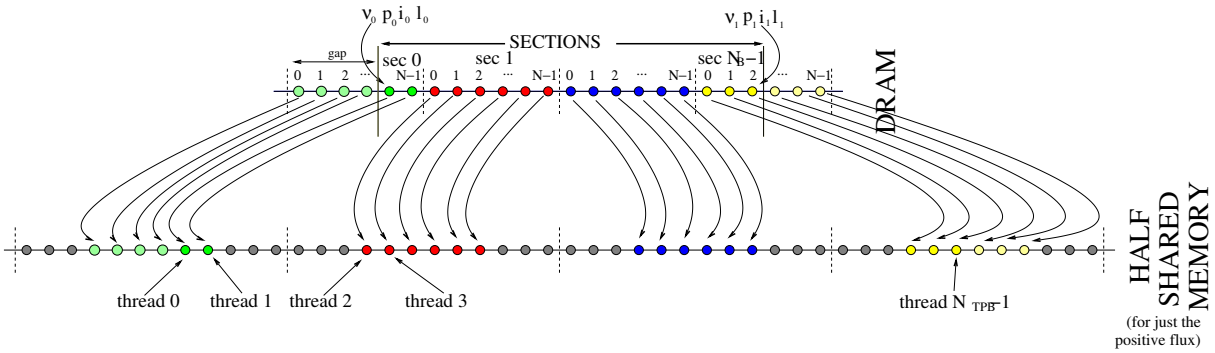

Figure 8: **mapping from global to shared memory to compute the ϕ-derivative**

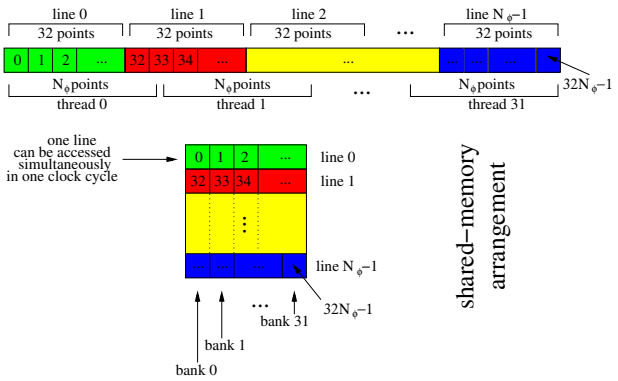

Figure 9: **Shared memory and banks.**

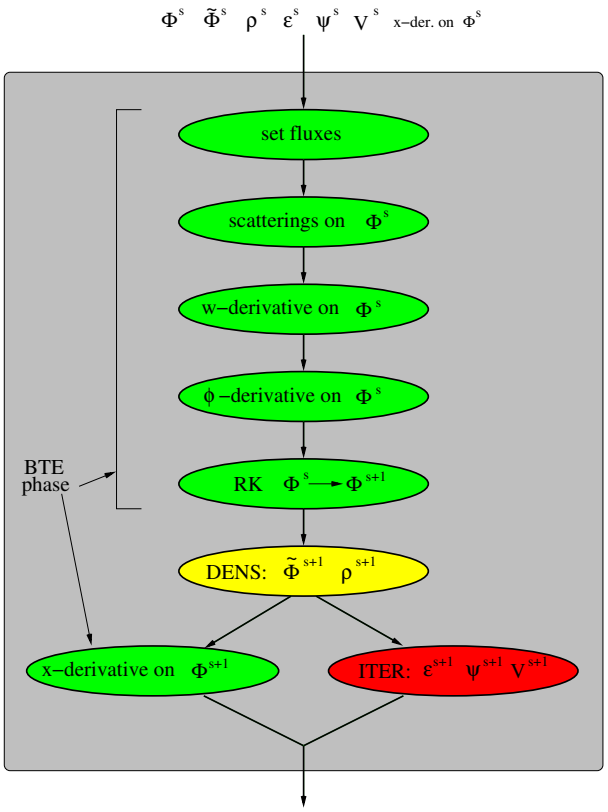

Figure 10: **Overlapping the computations.** The computation of the partial derivative $\frac{\partial(a^1 \cdot \Phi^{s+1})}{\partial x}$ (on the GPU) can be overlapped to the ITER phase (on the dual CPU *host*).

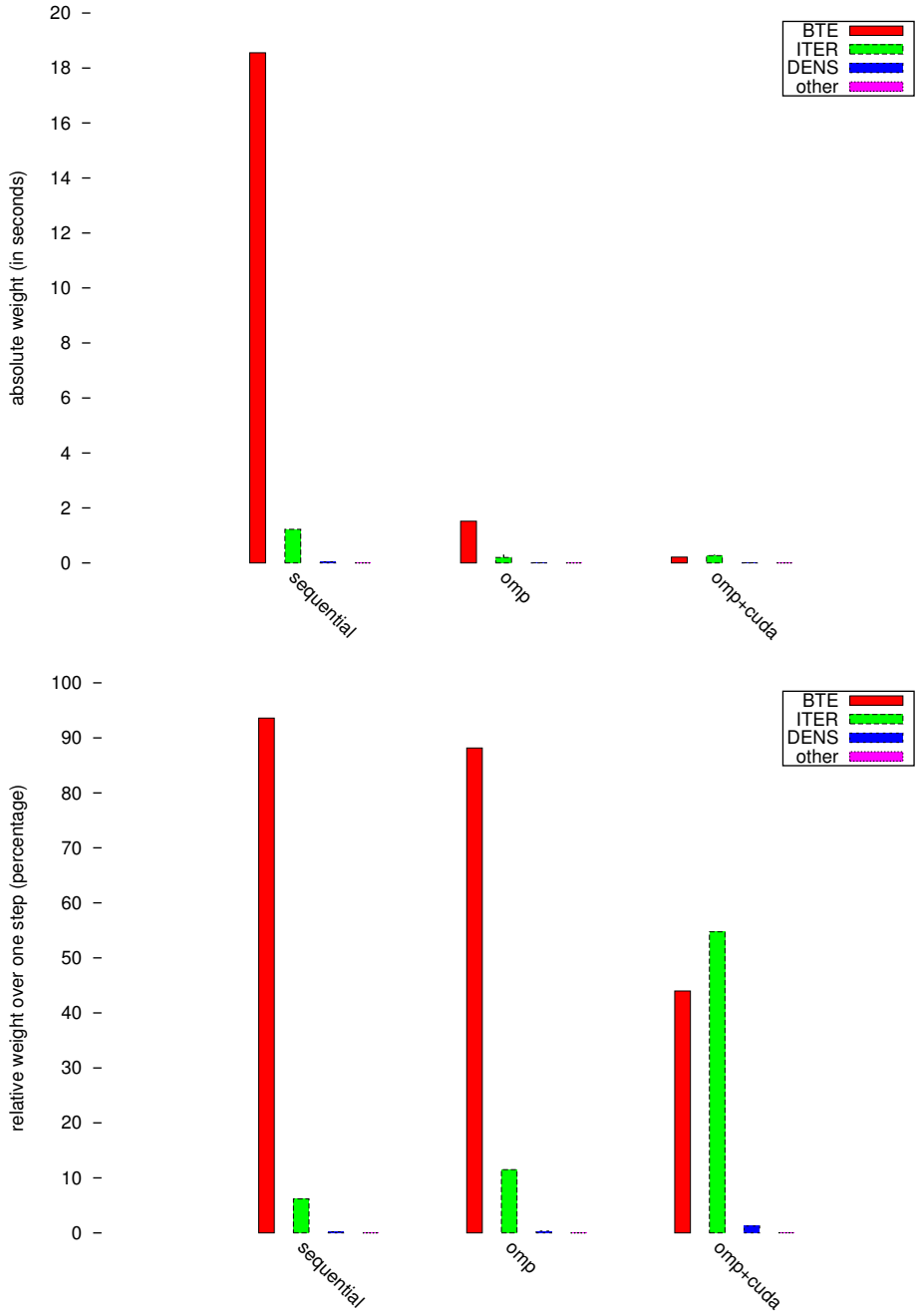

Figure 11: **Absolute (above) and relative (below) weight of the main computational phases.** We draw the relative weights of the phases consistent with the scheme in Figure 2.

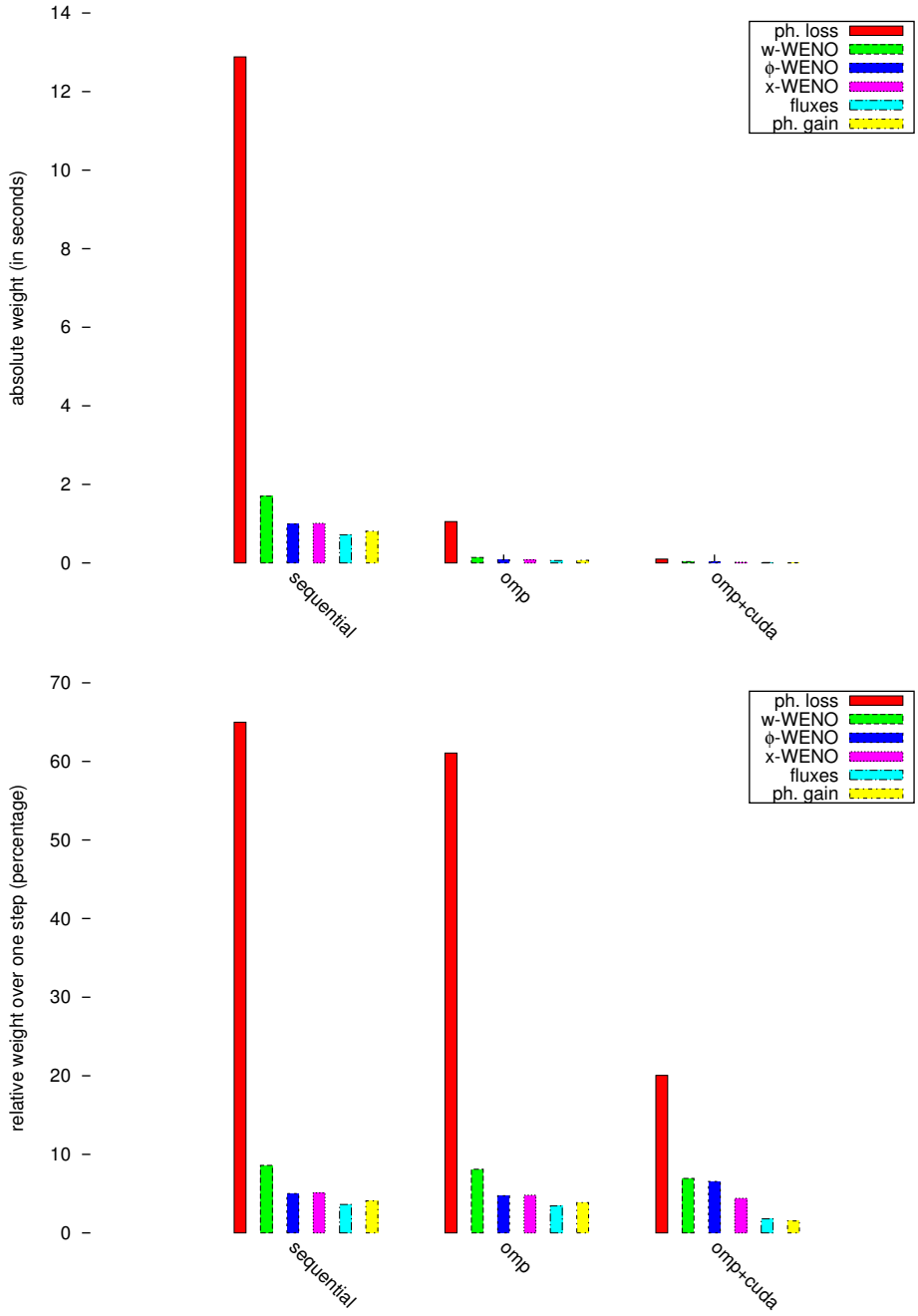

Figure 12: **Absolute (above) and relative (below) weight of the main kernels in the BTE phase.** The blocks are represented ordered as for the computational weight.

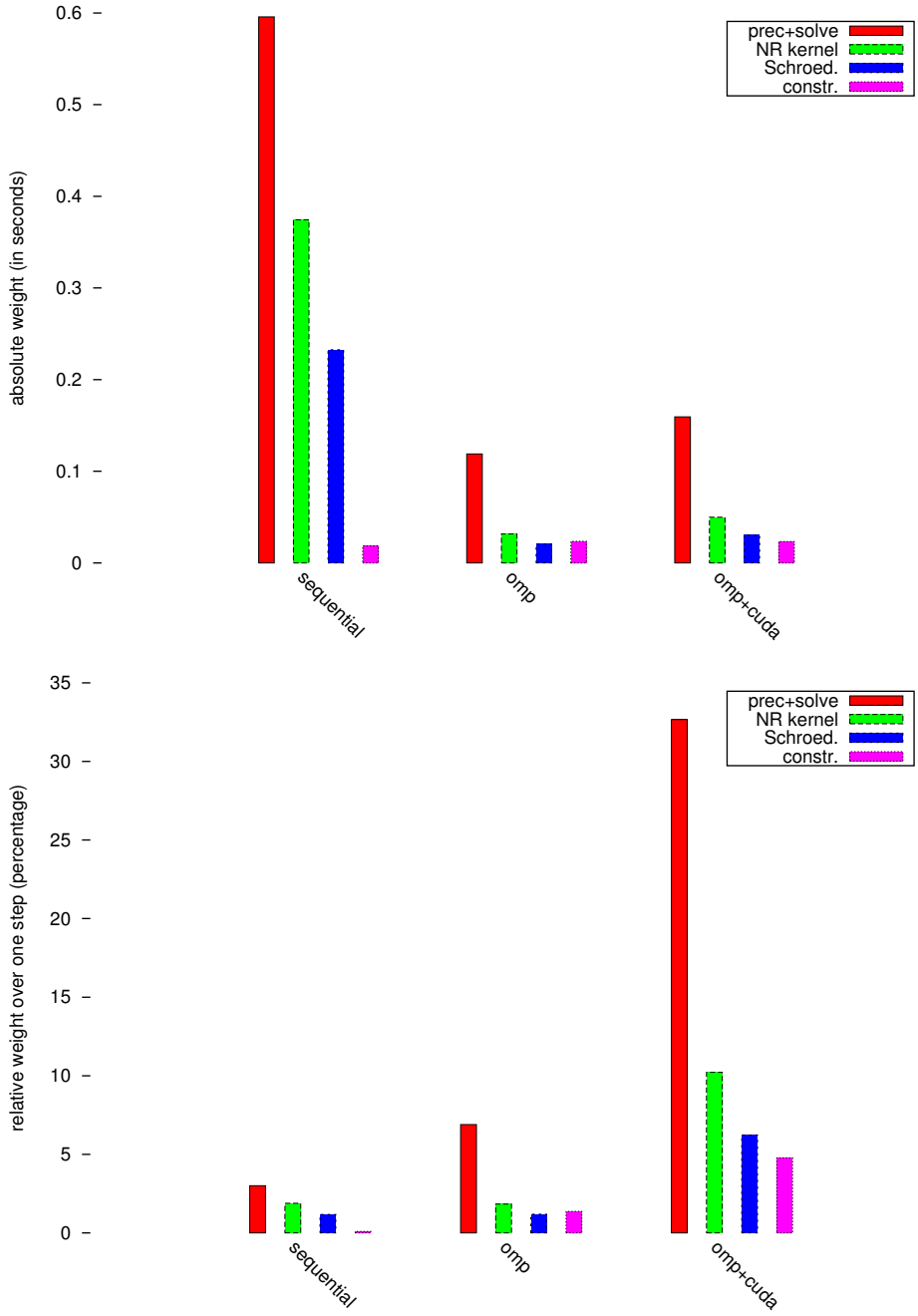

Figure 13: **Absolute (above) and relative (below) weight of the main computing steps in the ITER phase.** The blocks are represented ordered as for the computational weight.

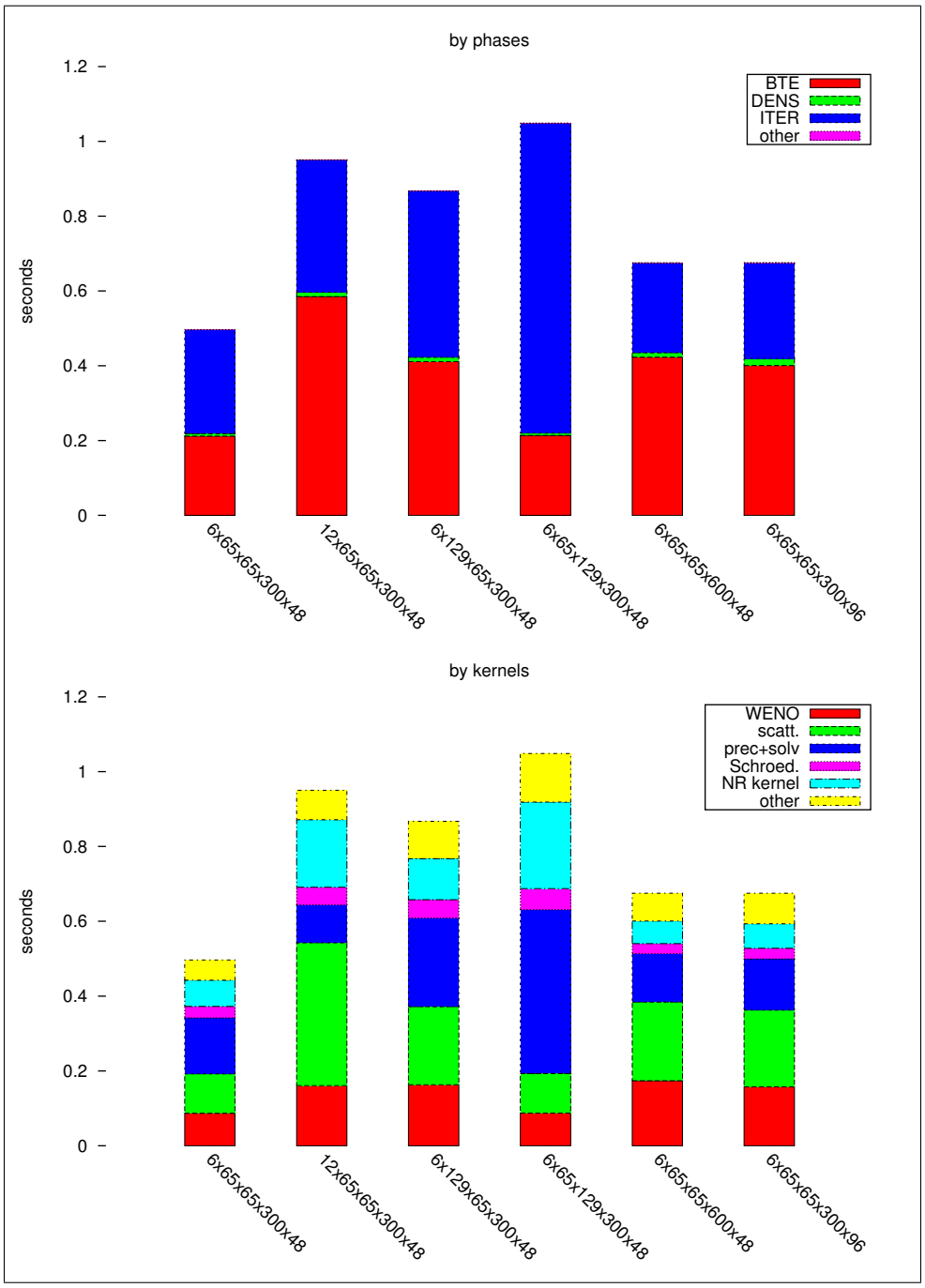

Figure 14: **Computational load of the hybrid parallel code.** The cost of each computational phase, for different meshes. The reference meshes are $N_{\mathrm{sbn}} \times N_x \times N_z \times N_w \times N_\phi = 6 \times 65 \times 65 \times 150 \times 48$.

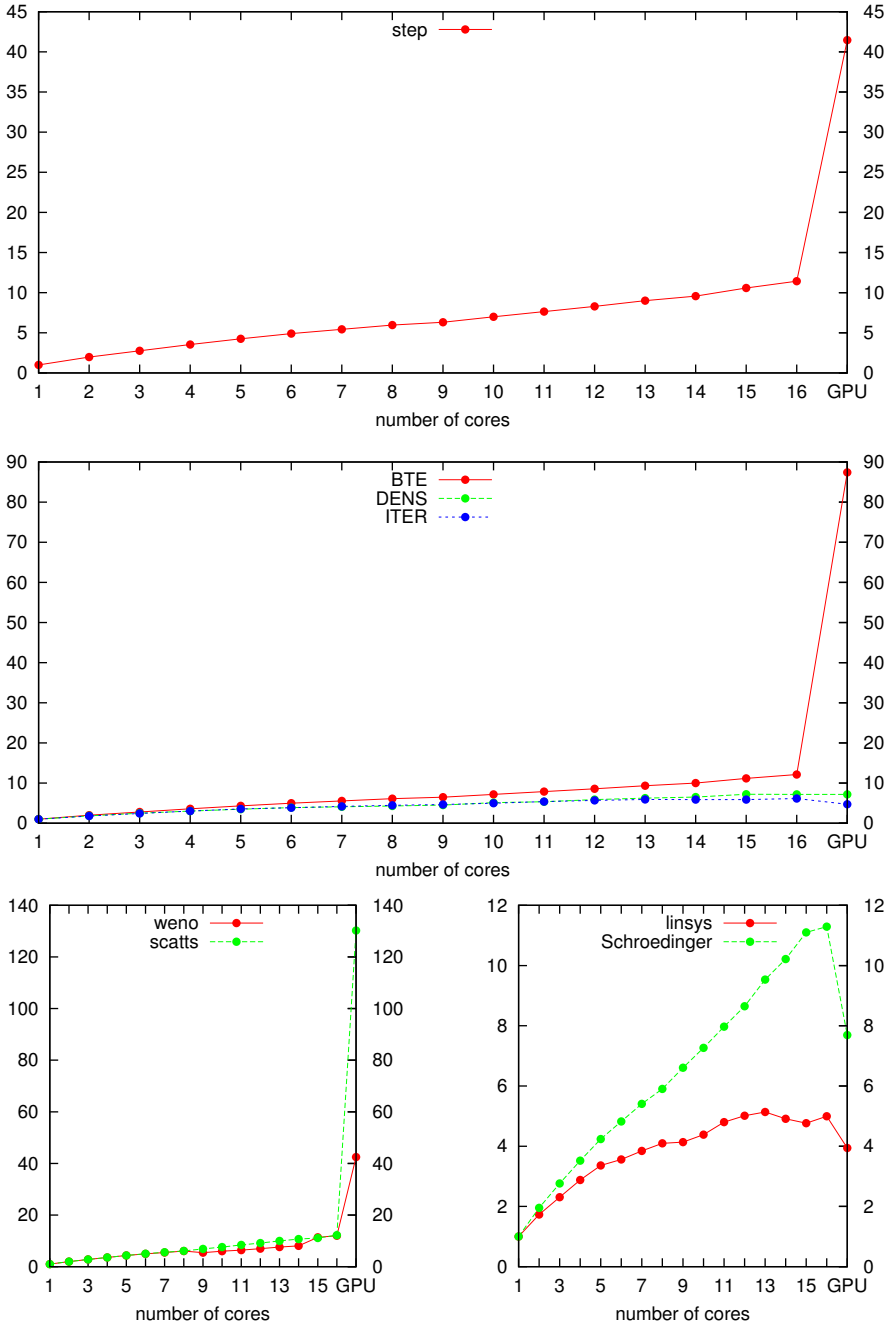

Figure 15: **Speedups.** In these figures the speedups are meant with respect to a purely sequential code executed by one CPU thread on the execution platform.