GPU implementation of a Schrödinger–Poisson solver for a nanoscaled DG MOSFET

Francesco Vecil, José Miguel Mantas

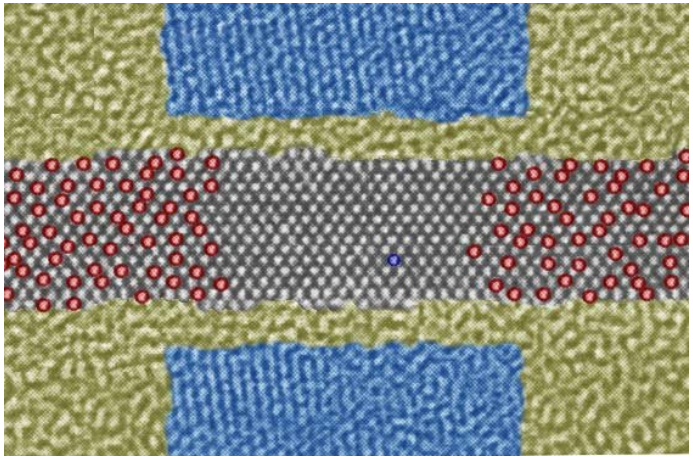
ECMI 2021, online, 2021/04/15

Outline

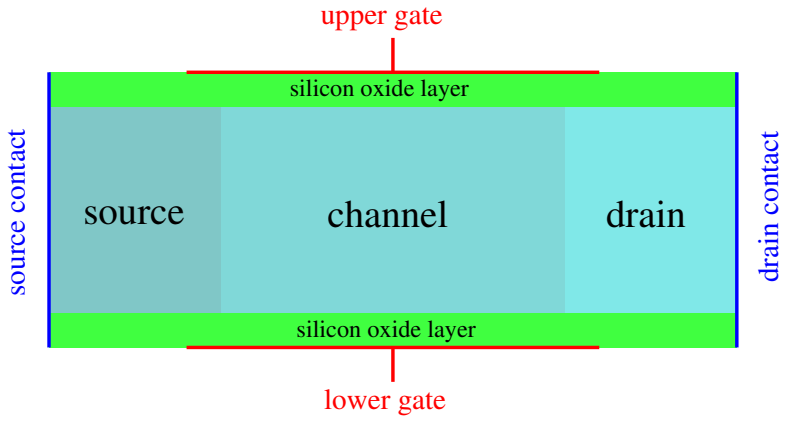
Outline

Geometry

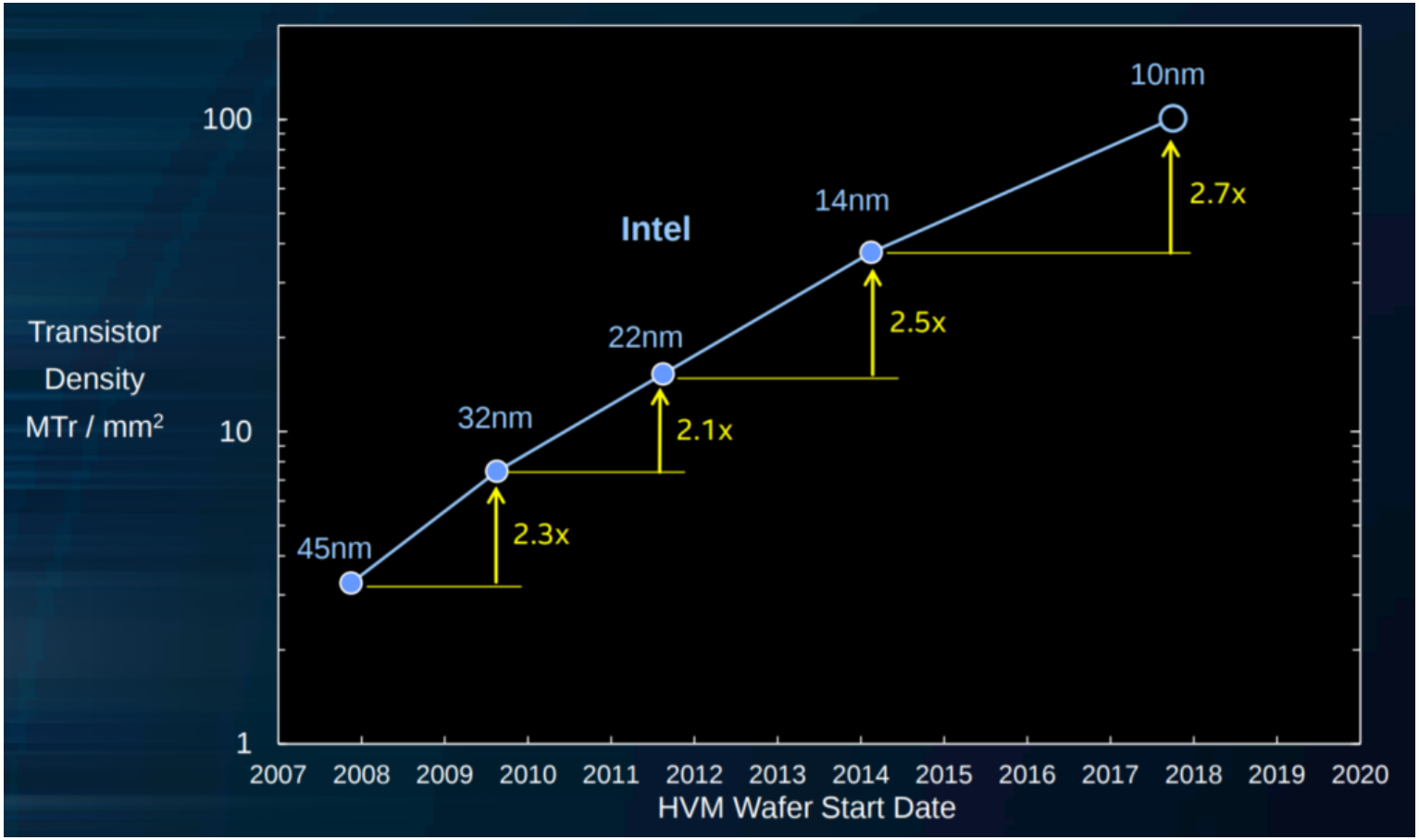
Photoshop impression from TEM images of real interfaces



The model

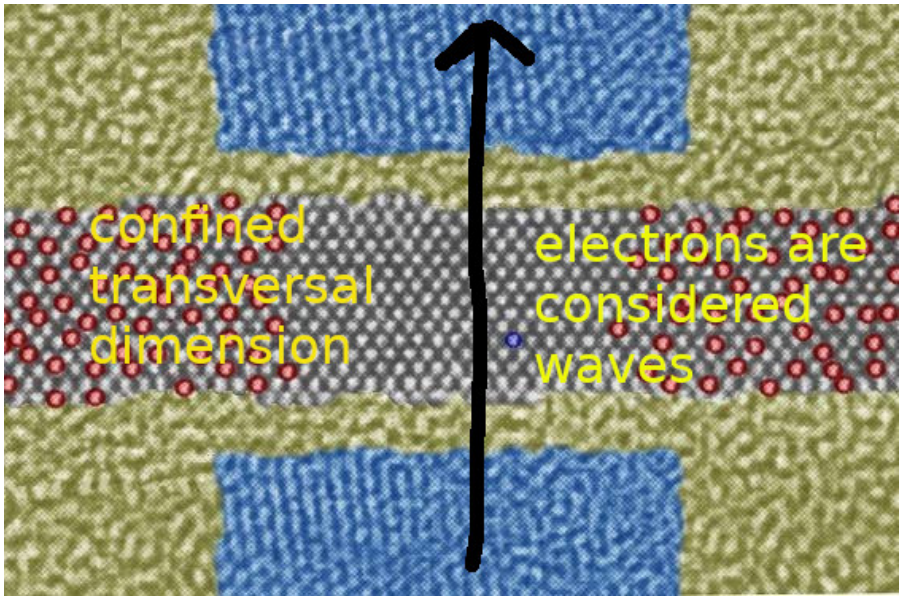


Chronological partial overview



The modeling

Transversal dimension



Schödinger–Poisson block

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

$$-\mathrm{div}_{x,z}\left[\varepsilon_R\nabla_{x,z}V\right]=-\frac{q}{\varepsilon_0}\left(2\sum_{\nu,p}\varrho_{\nu,p}\left|\psi_{\nu,p}[V]\right|^2-N_D\right).$$

Outline

Historical perspective

- Naoufel Ben Abdallah, María J. Cáceres, José Antonio Carrillo F. Vecil, *A deterministic solver for a hybrid quantum-classical transport model in nanoMOSFETs*, Journal of Computational Physics Volume 228, Issue 17, 2009, Pages 6553–6571.
- José M. Mantas, Mará J. Cáceres, Carlos Sampedro, Andrés Godoy, Francisco Gámiz, *A parallel deterministic solver for the Schrödinger-Poisson-Boltzmann system in ultra-short DG-MOSFETs: Comparison with Monte-Carlo*, Computers and Mathematics with Applications, Volume 67, Issue 9, 2014, Pages 1703–1721.
- José M. Mantas, Francesco Vecil, *Hybrid OpenMP-CUDA parallel implementation of a deterministic solver for ultrashort DG-MOSFETs*, The International Journal of High Performance Computing Applications, Volume 34, Issue 1, 2020, Pages 81–102.

Outline

The Newton scheme

Reminder

We recall here the Schrödinger–Poisson block for the computation of the advection field:

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

$$-\mathrm{div}_{x,z}\left[\varepsilon_R \nabla_{x,z} V\right] = -\frac{q}{\varepsilon_0}\left(2\sum_{\nu,p}\varrho_{\nu,p}\left|\psi_{\nu,p}[V]\right|^2 - N_D\right).$$

Blackbox

$$\varrho_{\nu,p}(x) \longrightarrow \boxed{\text{Schrödinger–Poisson block}} \longrightarrow \epsilon_{\nu,p}(x),\ \psi_{\nu,p}(x,z),\ V(x,z).$$

Strategy

Using an iterative method: the Newton-Raphson scheme. This leads to iteratively solving a linear system and an eigenvalue/eigenvector problem.

The Newton scheme

Reminder

We recall here the Schrödinger–Poisson block for the computation of the advection field:

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

$$-\mathrm{div}_{x,z}\left[\varepsilon_R \nabla_{x,z} V\right] = -\frac{q}{\varepsilon_0}\left(2\sum_{\nu,p}\varrho_{\nu,p}\left|\psi_{\nu,p}[V]\right|^2 - N_D\right).$$

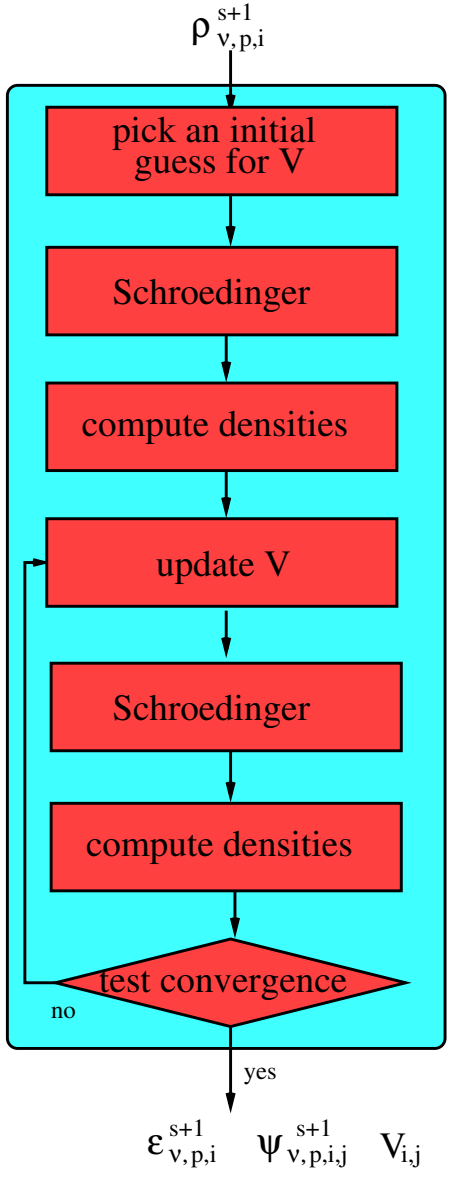
Blackbox

$$\varrho_{\nu,p}(x) \longrightarrow \boxed{\text{Schrödinger–Poisson block}} \longrightarrow \epsilon_{\nu,p}(x),\ \psi_{\nu,p}(x,z),\ V(x,z).$$

Strategy

Using an iterative method: the Newton-Raphson scheme. This leads to iteratively solving a linear system and an eigenvalue/eigenvector problem.

The Newton scheme



The Newton scheme

The Schrödinger equation

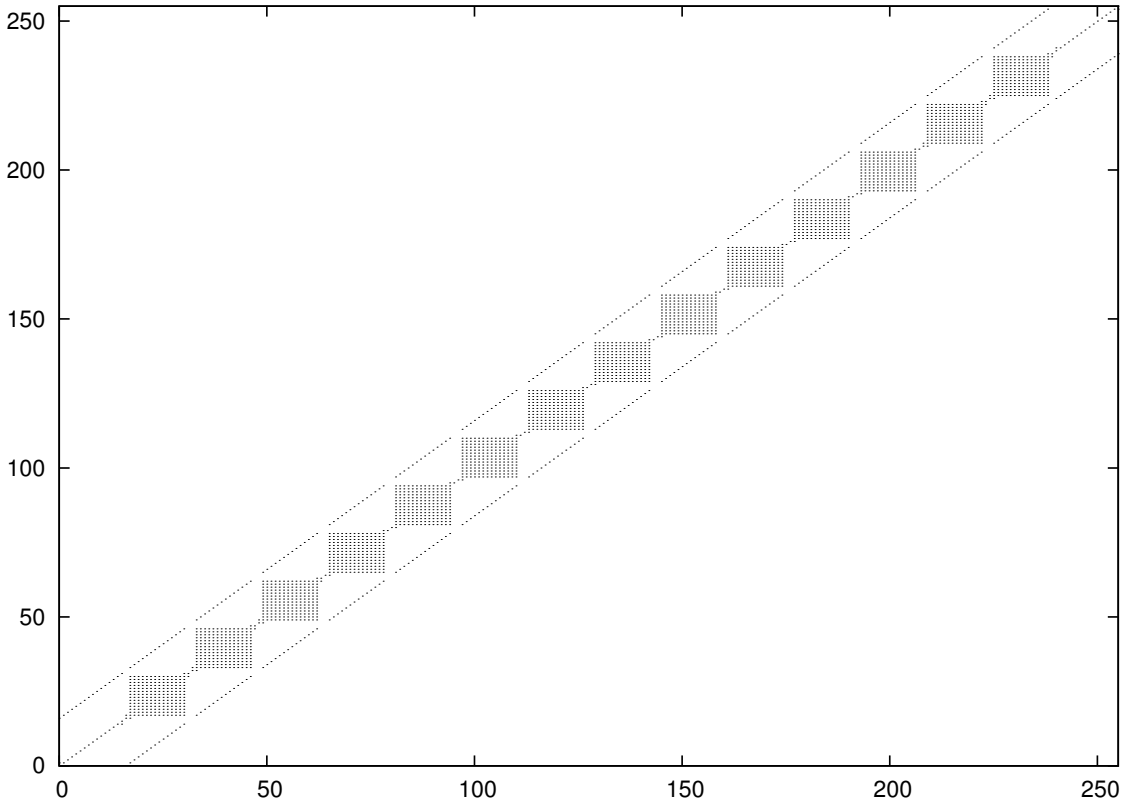
We compute selected eigenvalues and relative eigenvectors of the Schrödinger matrices (one for each valley ν and each longitudinal position i)

$$\mathcal{L}_{\nu,i} = \begin{pmatrix} d_0 & e_0 & & & & \\ e_0 & d_1 & e_1 & & & \\ & e_1 & d_2 & e_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & e_{n-3} & d_{n-2} & e_{n-2} \\ & & & & e_{n-2} & d_{n-1} \end{pmatrix}$$

The Newton scheme

The linear system

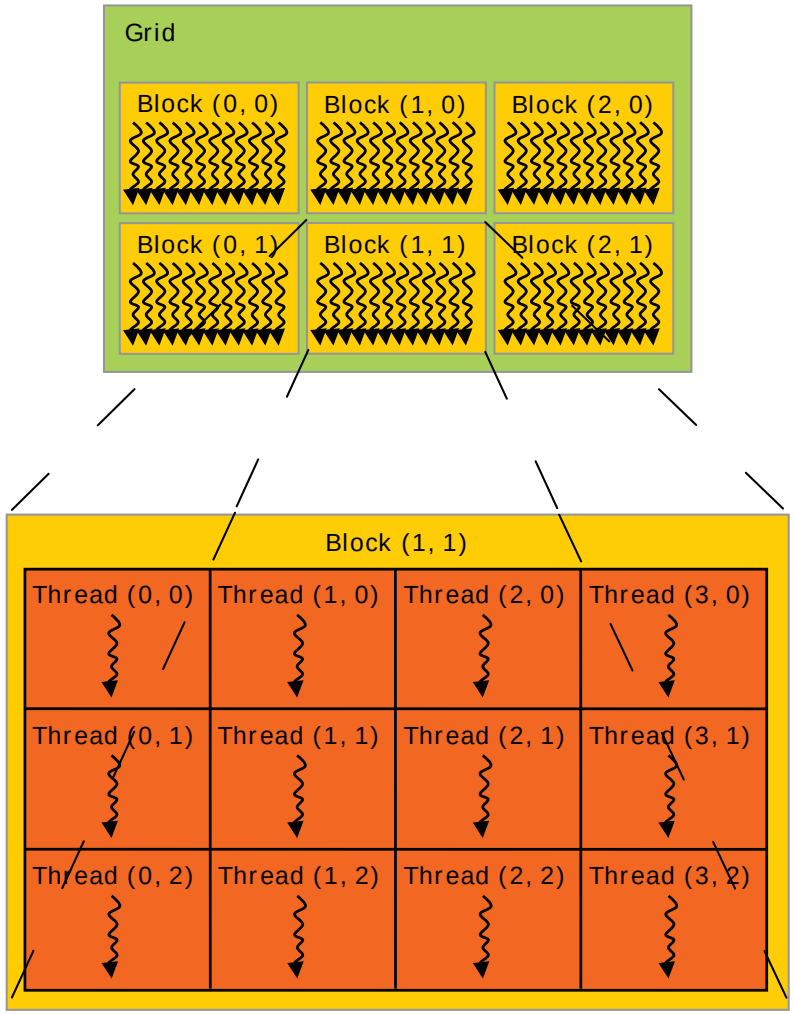
At iteration k, we refine the potential V by $L^{(k)}\ V^{(k+1)} = R^{(k)}$.



The system has bandwidth $2\,N_z + 1$, and contains non-local terms.

Outline

Cuda programming model on GPU



(from book CUDA C Programming Guide)

Some remarks on the Cuda implementation

Fine grain

To be exploited as much as possible: many threads, each of them with a light weight.

Shared memory

Use of block's shared memory to minimize reads from DRAM or to load data from DRAM in a coalescent manner.

Avoid transfer of information

The amount of information being copied between the RAM of the CPU and the GPU should be kept as small as possible.

Exploit warps

Warps are groups of 32 threads. They are physically executed concurrently at hardware level and can exchange information in the fastest way.

Some remarks on the Cuda implementation

Fine grain

To be exploited as much as possible: many threads, each of them with a light weight.

Shared memory

Use of block's shared memory to minimize reads from DRAM or to load data from DRAM in a coalescent manner.

Avoid transfer of information

The amount of information being copied between the RAM of the CPU and the GPU should be kept as small as possible.

Exploit warps

Warps are groups of 32 threads. They are physically executed concurrently at hardware level and can exchange information in the fastest way.

Some remarks on the Cuda implementation

Fine grain

To be exploited as much as possible: many threads, each of them with a light weight.

Shared memory

Use of block's shared memory to minimize reads from DRAM or to load data from DRAM in a coalescent manner.

Avoid transfer of information

The amount of information being copied between the RAM of the CPU and the GPU should be kept as small as possible.

Exploit warps

Warps are groups of 32 threads. They are physically executed concurrently at hardware level and can exchange information in the fastest way.

Some remarks on the Cuda implementation

Fine grain

To be exploited as much as possible: many threads, each of them with a light weight.

Shared memory

Use of block's shared memory to minimize reads from DRAM or to load data from DRAM in a coalescent manner.

Avoid transfer of information

The amount of information being copied between the RAM of the CPU and the GPU should be kept as small as possible.

Exploit warps

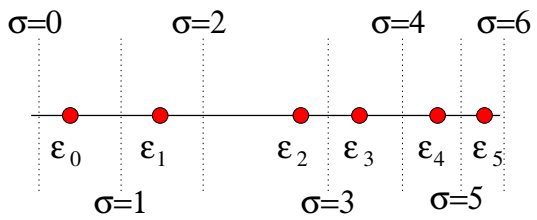
Warps are groups of 32 threads. They are physically executed concurrently at hardware level and can exchange information in the fastest way.

Outline

The eigenvalues

Bisection/Multisection

- Iterative method: selected eigenvalue lie inside shrinking intervals.
- We assign one eigenvalue to each warp.



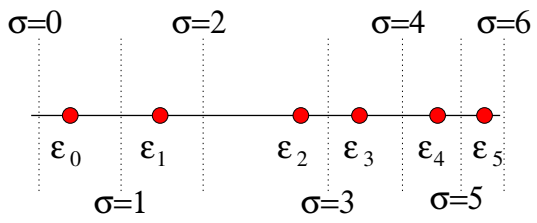
Some more details

- Shared memory is exploited in order to read in a coalescent way from the matrices $\mathcal{L}_{\nu,i}$.
- Shuffle functions are used to perform reductions at warp level.
- Some data are declared as `volatile` in order to prevent the compiler from trying any optimization and introduce rush conflicts.

The eigenvalues

Bisection/Multisection

- Iterative method: selected eigenvalue lie inside shrinking intervals.
- We assign one eigenvalue to each warp.



Some more details

- Shared memory is exploited in order to read in a coalescent way from the matrices $\mathcal{L}_{\nu,i}$.
- Shuffle functions are used to perform reductions at warp level.
- Some data are declared as `volatile` in order to prevent the compiler from trying any optimization and introduce rush conflicts.

The eigenvectors

Parallel Cyclic Reduction

Once the eigenvalues have been computed, in order to obtain the eigenvectors we solve tridiagonal (non-symmetric) linear systems by the PCR algorithm

$$\left(\begin{array}{ccccccc|c|c}
\tilde{d}_0 & e_0 & & & & & & & \\
e_0 & \tilde{d}_1 & e_1 & & & & & & \\
 & e_1 & \tilde{d}_2 & e_2 & & & & & \\
 & & \ddots & \ddots & \ddots & & & & \\
 & & & e_{j-2} & \tilde{d}_{j-1} & e_{j-1} & & & \\
\hline
 & & & & 0 & 1 & 0 & & \\
\hline
 & & & & & e_j & \tilde{d}_{j+1} & e_{j+1} & \\
 & & & & & & \ddots & \ddots & \ddots \\
 & & & & & & & e_{n-2} & \tilde{d}_{n-1}
\end{array}\right)
\left(\begin{array}{c}
\psi_0 \\ \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{j-1} \\ \hline \psi_j \\ \hline \psi_{j+1} \\ \vdots \\ \psi_{n-1}
\end{array}\right)
=
\left(\begin{array}{c}
0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \hline 1 \\ \hline 0 \\ \vdots \\ 0
\end{array}\right).$$

Some remarks

- PCR is not very efficient but very parallel.
- One block per linear system, of size multiple of 32.

Updating potential

Jacobi scheme

Jacobi iterative algorithm is very parallel but also particularly inefficient.

Relaxed Jacobi scheme

Suppose we are solving (for the sake of lighter notations) linear system $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$.
If we decompose matrix $\boldsymbol{A}$ as $\boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$, the relaxed Jacobi iteration of parameter $\omega > 0$ reads

$$\mathcal{L}_\omega \boldsymbol{x} := \boldsymbol{M}_\omega^{-1} \boldsymbol{N}_\omega \boldsymbol{x} + \boldsymbol{M}_\omega^{-1} \boldsymbol{b}.$$

where

$$\boldsymbol{M}_\omega := \frac{1}{\omega} \boldsymbol{D}, \qquad \boldsymbol{N}_\omega := \frac{1-\omega}{\omega} \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}.$$

Updating potential

Successive Relaxed Jacobi (SRJ) scheme

The SRJ consists in defining sequences of relaxed Jacobi steps:

$$\mathcal{L} := \underbrace{\mathcal{L}_{\omega_P} \circ \ldots \circ \mathcal{L}_{\omega_P}}_{q_P \text{ times}} \circ \cdots \circ \underbrace{\mathcal{L}_{\omega_2} \circ \ldots \circ \mathcal{L}_{\omega_2}}_{q_2 \text{ times}} \circ \underbrace{\mathcal{L}_{\omega_1} \circ \ldots \circ \mathcal{L}_{\omega_1}}_{q_1 \text{ times}}$$

and updating the guess for the solution of system $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$ using these:

$$\boldsymbol{x}^{(\ell+1)} = \mathcal{L}\boldsymbol{x}^{(\ell)}.$$

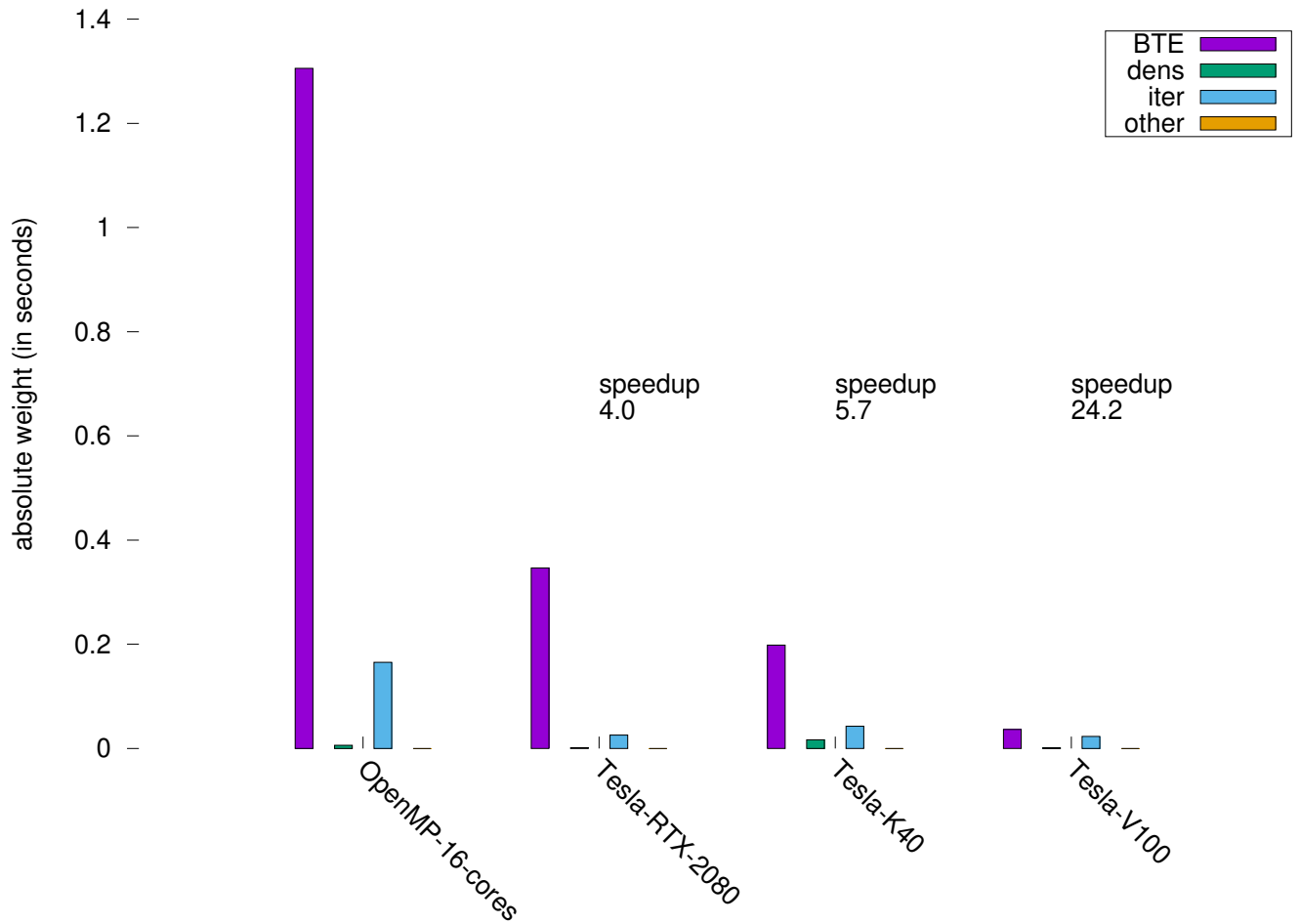
We shall use the following parameters: $P = 7$ for the number of SRJ "blocks", $q_1 = q_2 = \ldots = q_7 = 93$ for the iterations inside each "block", and as relaxation parameters

$$\omega_1 = 370.035, \quad \omega_2 = 167.331, \quad \omega_3 = 51.1952, \quad \omega_4 = 13.9321$$
$$\omega_5 = 3.80777, \quad \omega_6 = 1.18727, \quad \omega_7 = 0.556551.$$

Outline

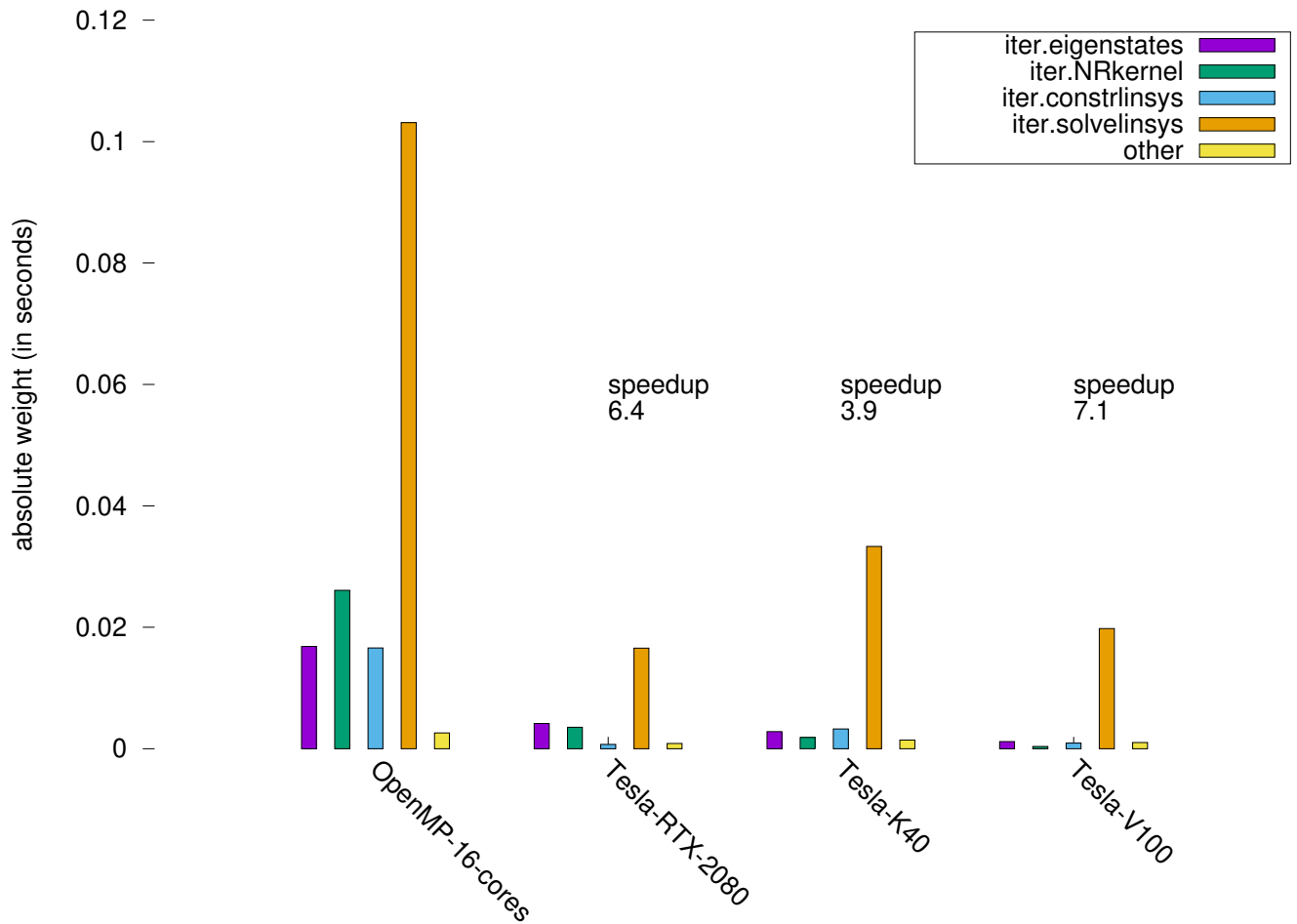
Performances of the code

Overview



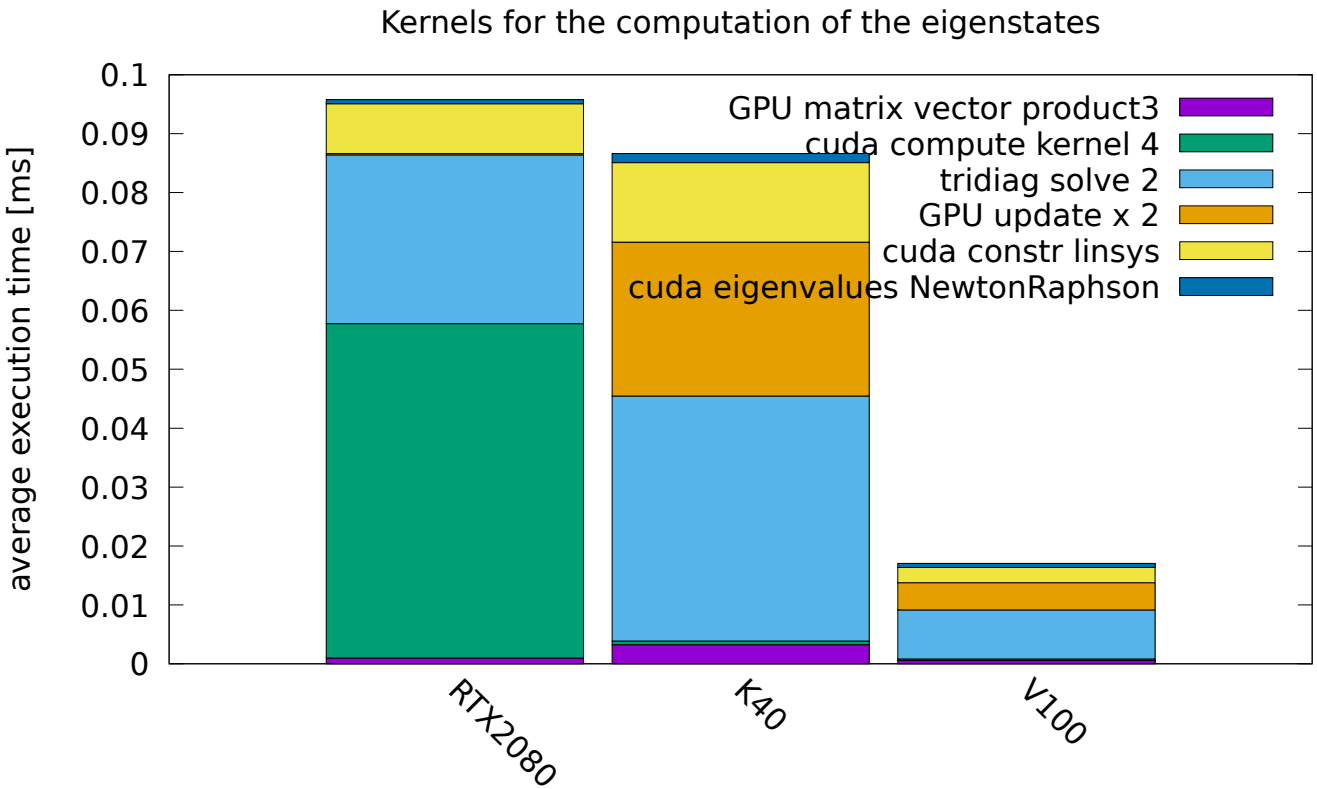
Performances of the code

The Schrödinger-Poisson block



Performances of the code

CUDA kernels of the Schrödinger-Poisson block



GRAZIE!

The authors acknowledge Spanish projects **MTM2011-27739-C04-02** and **MTM2014-52056-P** and the European Fund for Development.