CUDA port to GPU of a Boltzmann–Schrödinger–Poisson solver for confined devices

Francesco Vecil, José Miguel Mantas, Pedro Alonso–Jordá

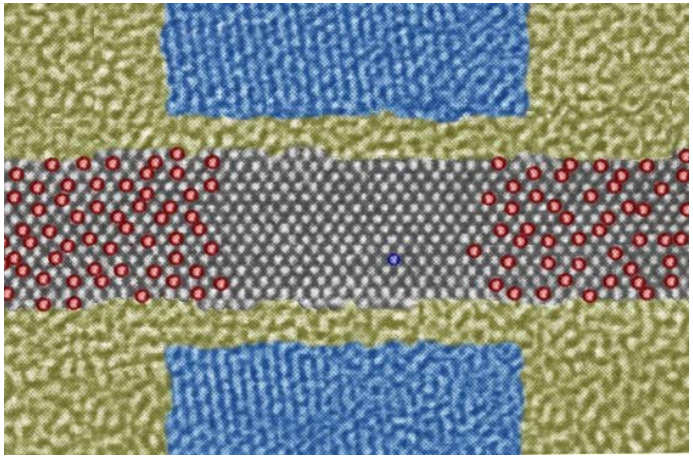
séminaire de l'équipe EDPAN, LMPB, 2023/10/19

Outline

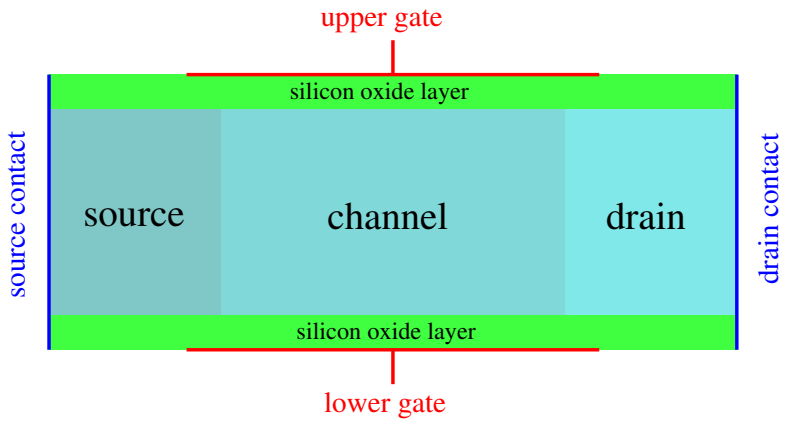
Outline

Geometry

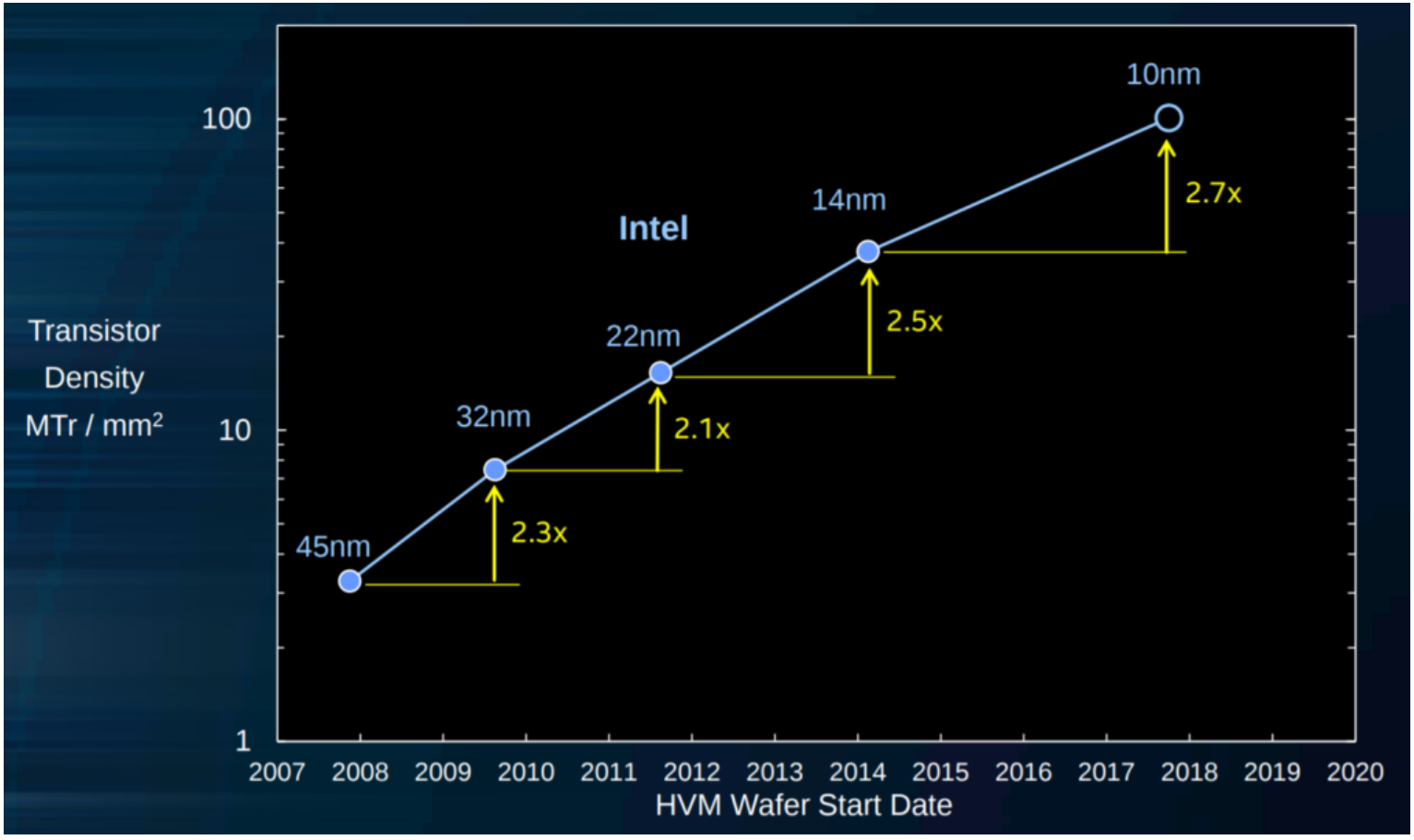
Photoshop impression from TEM images of real interfaces



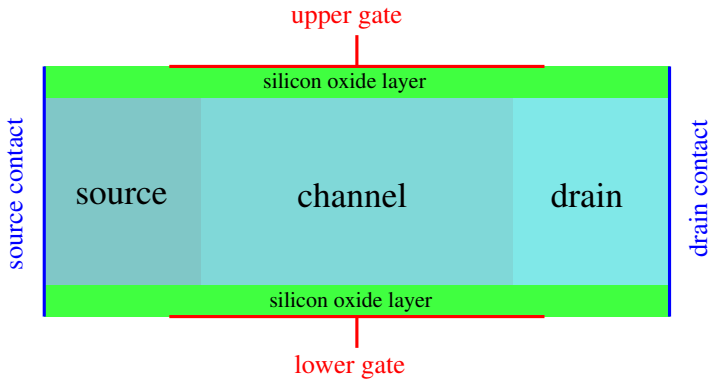
The model



Chronological partial overview



The modeling: transversal dimension



Schödinger–Poisson block

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

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

Longitudinal dimension

The description of the transport from source to drain is outside the scope of this talk.

Outline

Historical perspective

1. 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.
2. 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.
3. 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.
4. Francesco Vecil, José M. Mantas, Pedro Alonso–Jordá *Efficient GPU implementation of a Boltzmann-Schrödinger-Poisson solver for the simulation of nanoscale DG MOSFETs*, The Journal of Supercomputing, 2023, Pages 1–32.

Outline

The iterative scheme

Reminder

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

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

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

Algorithmically

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

Strategy

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

The iterative scheme

Reminder

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

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

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

Algorithmically

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

Strategy

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

The iterative scheme

Reminder

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

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

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

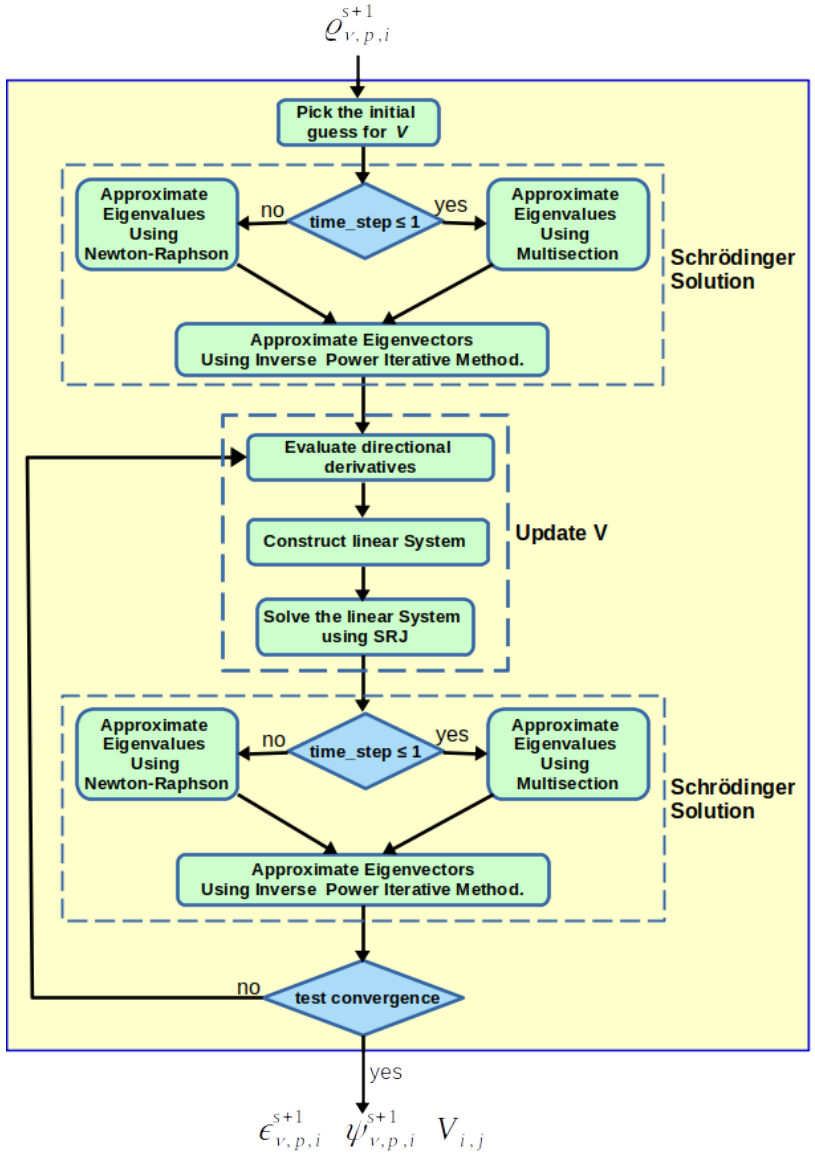
Algorithmically

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

Strategy

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

The iterative scheme



The iterative scheme

The Schrödinger equation

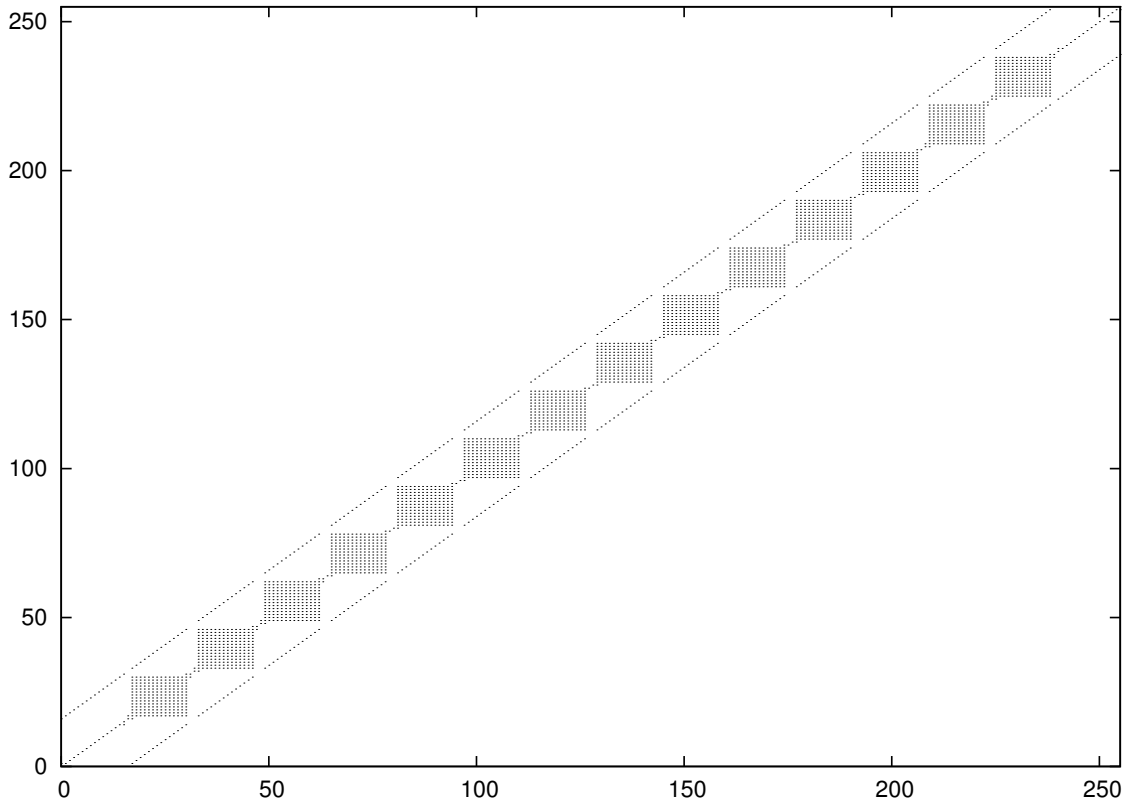
We compute selected eigenvalues and relative eigenvectors of the Schrödinger matrices (one for each valley ν and each longitudinal position i)

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

The iterative scheme

The linear system

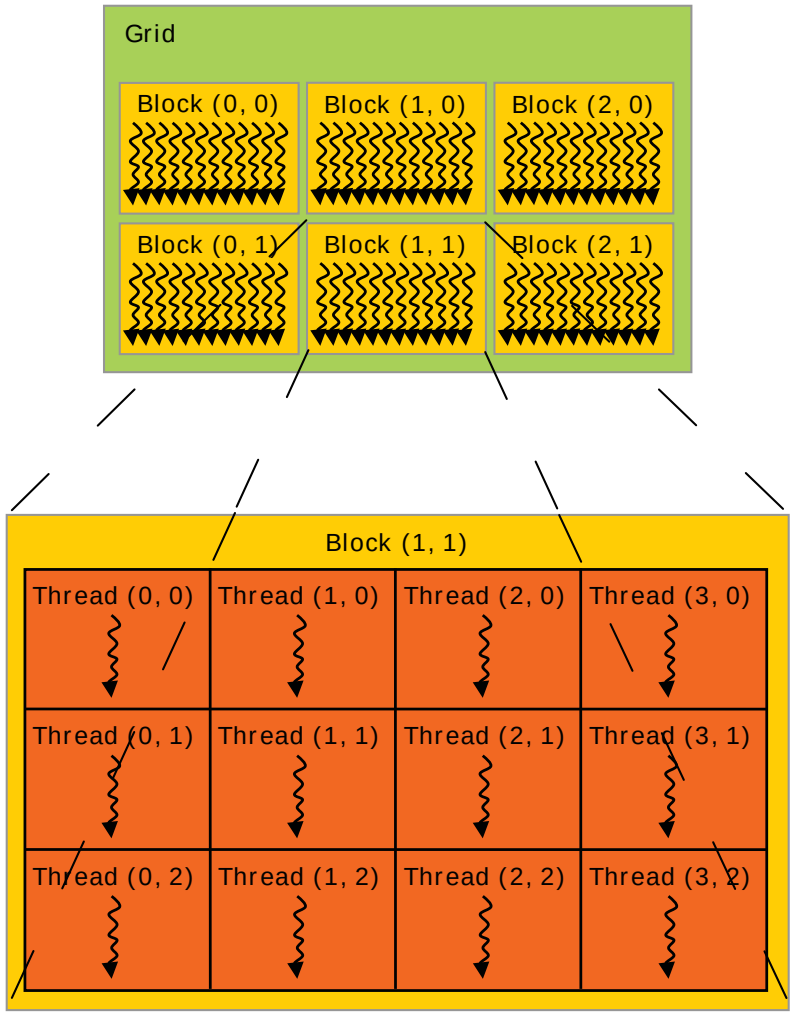
At iteration k, we refine the potential V by $L^{(k)}\ V^{(k+1)} = R^{(k)}$.



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

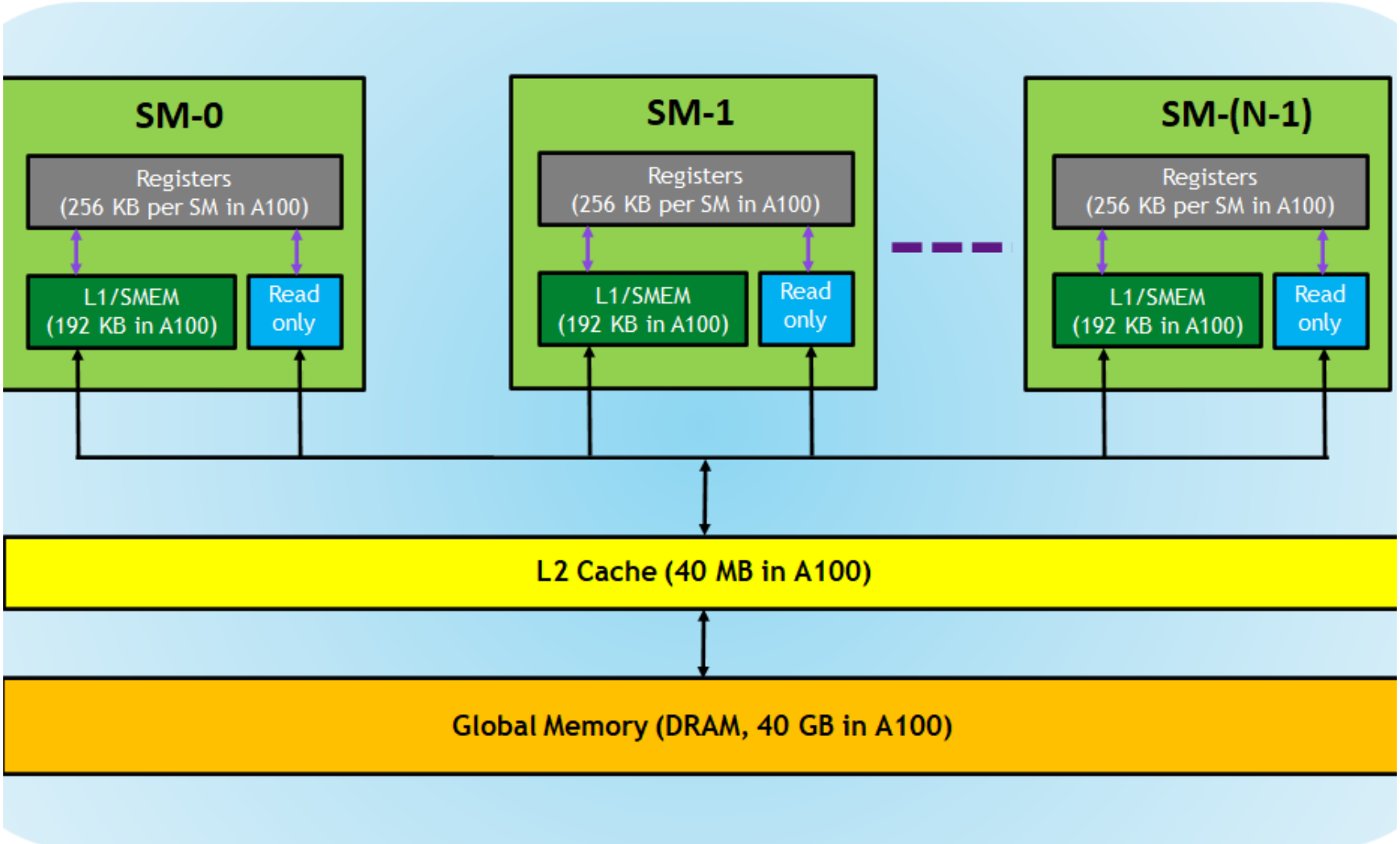
Outline

Cuda programming model on GPU



(from book CUDA C Programming Guide)

Cuda programming model on GPU



(from book CUDA C Programming Guide)

Cuda programming model on GPU: how it looks like

Allocation of the GPU memory

```
cudaMalloc((void **)&_GPU_pdf, _NVALLEYS*NSBN*NX*NW*NPHI*4*sizeof(double));
```

Implementation of a kerel

```
__global__ void cuda_perform_RK_1_3( const discrMeshes *dm, double *_GPU_pdf, const double *_GPU_rhs_pdf, const double DT )
{
  const int NSBN     = dm -> get_NSBN();
  const int NX       = dm -> get_X()   -> get_N();
  const int NW       = dm -> get_W()   -> get_N();
  const int NPHI     = dm -> get_PHI() -> get_N();

  int global_index = blockIdx.x*blockDim.x + threadIdx.x;

  if(global_index < _NVALLEYS*NSBN*NX*NW*NPHI)
    {
      int nu,p,i,l,m;
      GPU_map_1D_to_5D( global_index, &i, NX, &nu, _NVALLEYS, &p, NSBN, &l, NW, &m, NPHI );

      GPU_pdf( nu,p,i,l,m,1 ) = GPU_pdf( nu,p,i,l,m,0 ) + DT*GPU_rhs_pdf( nu,p,i,l,m );
    }
}
#else
```

Call to the kerel

```
  cudaDeviceSetCacheConfig( cfc );
  cuda_perform_RK_1_3 <<< gridSize, blockSize, shmemSize >>> ( device_dm, _GPU_pdf, _GPU_rhs_pdf, DT );
#else
```

Object compilation

```
ptxas info    : Used 15 registers, 368 bytes cmem[0]
nvcc -I/usr/local/cuda-10.2/include -I/home/jmmantas/NVIDIA_CUDA-10.1_Samples/common/inc/ -Ilis-1.7.33/include/ -I/usr/include/openmpi-x86_64/ -Xcompiler -fop
enmp --compiler-options -DUNIX -O3 -m64 -g -gencode arch=compute_70,code=sm_70 -gencode arch=compute_86,code=sm_86 -Xptxas -v -Xptxas -warn-lmem-usage -Xptxas
 -warn-spills -Xptxas -dlcm=ca -lineinfo -use_fast_math  -D CUDA_CODE -c src/cuda_time_integration.cu -o src/cuda_time_integration.o
ptxas info    : 0 bytes gmem, 896 bytes cmem[3]
```

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

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

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

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

Reminder

Three computational phases:

- selected eigenvalues (usually 6) of $\sim$ 1000 matrices
- their relative eigenvectors
- banded linear system of order $\sim$ 4000

The eigenvalues

Newton-Raphson

Iterative method. We give an eigenvalue to each thread. The seeding is by the eigenvalues computed at the previous stage.
The implementation does not use any sophisticated technique worth mentioning.

Seeding

In order to converge to the correct eigenvalues, the algorithm must be initialized not too far from the target value. It is therefore used only after the first step, at which the robust bisection algorithm is used, which only requires an interval.

The eigenvectors

IPIM

The Inverse Power Iterative Method (IPIM) is used.
To approximate eigenvector $\psi_{\nu,p,i,\cdot}$ we iterate, until a certain tolerance parameter is fulfilled:

- $\psi^{(0)}_{\nu,p,i,\cdot} \in \mathbb{R}^{N_z-2}$ is given
- for $k \geq 0$
  - solve $\left(\mathcal{L}_{\nu,i} - \epsilon_{\nu,p,i}\right) \psi^{(k+1)}_{\nu,p,i,\cdot} = \psi^{(k)}_{\nu,p,i,\cdot}$
  - normalize $\psi^{(k+1)}_{\nu,p,i,j} \longleftarrow \dfrac{\psi^{(k+1)}_{\nu,p,i,j}}{\left\| \psi^{(k+1)}_{\nu,p,i,\cdot} \right\|}$

The linear system

In order to solve the linear system, Thomas algorithm is used.

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

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

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

Scheduled-Relaxation 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:

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

Avoiding rounding errors

In practice, we do not use ω_ℓ consecutive steps with parameter q_ℓ. Rather, we "shuffle" the relaxation steps, to avoid **rounding errors**.
The way they follow each other is of fundamental relevance for the stability.

Updating potential

Avoiding rounding errors

For example, the following sequence proves stable

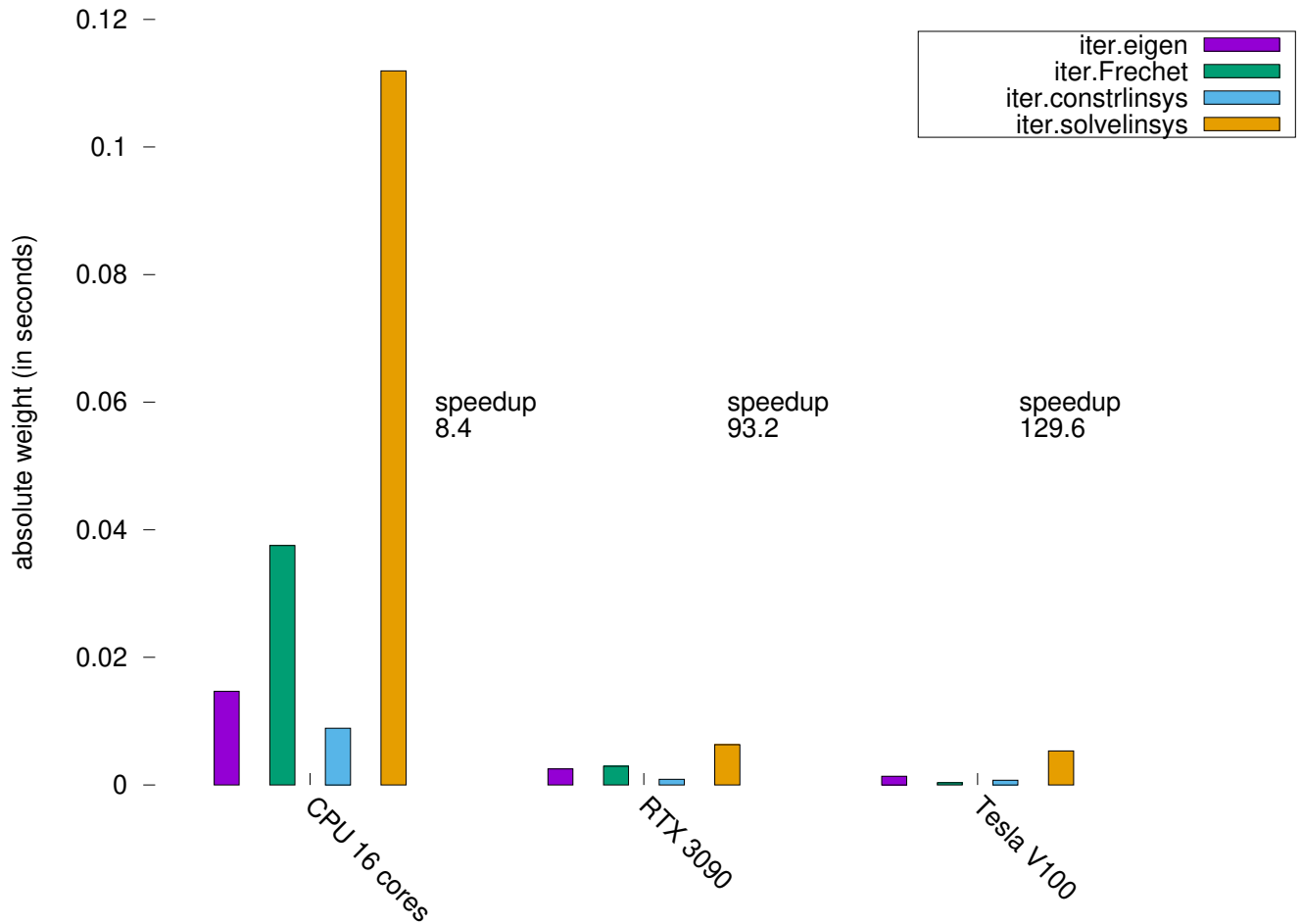
387.38, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 14.127, 0.53448, 0.87254, 1.9628, 0.53448, 1.9628, .53448, 38.971, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 5.1286, 0.87254, 0.87254, 102.42, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 14.127, 0.87254, 0.53448, 0.87254, 1.9628, 0.53448, 14.127, 0.87254, 0.53448, 1.9628, 1.9628, 0.53448, 38.971, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 233.47, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 5.1286, 0.87254, 0.87254, 14.127, 0.53448, 0.87254, 1.9628, 0.53448, 1.9628, 0.53448, 38.971, 0.53448, 0.87254, 1.9628, 0.53448, 5.1286, 0.53448, 0.87254, 5.1286, 0.53448, 0.87254, 102.42, 0.87254, 0.87254, 0.53448, 1.9628, 5.1286, 0.53448, 0.87254, 14.127, 0.53448, 0.87254, 1.9628, 0.53448, 14.127, 0.53448, 0.87254, 1.9628, 0.53448, 1.9628, 1.9628, 0.53448, 0.87254, 0.87254, 0.53448, 0.53448

while, if we choose an unsuitable order, the magnitude of the solution vector may explode and contract even by 14-15 orders, hence leading to rounding errors.

Outline

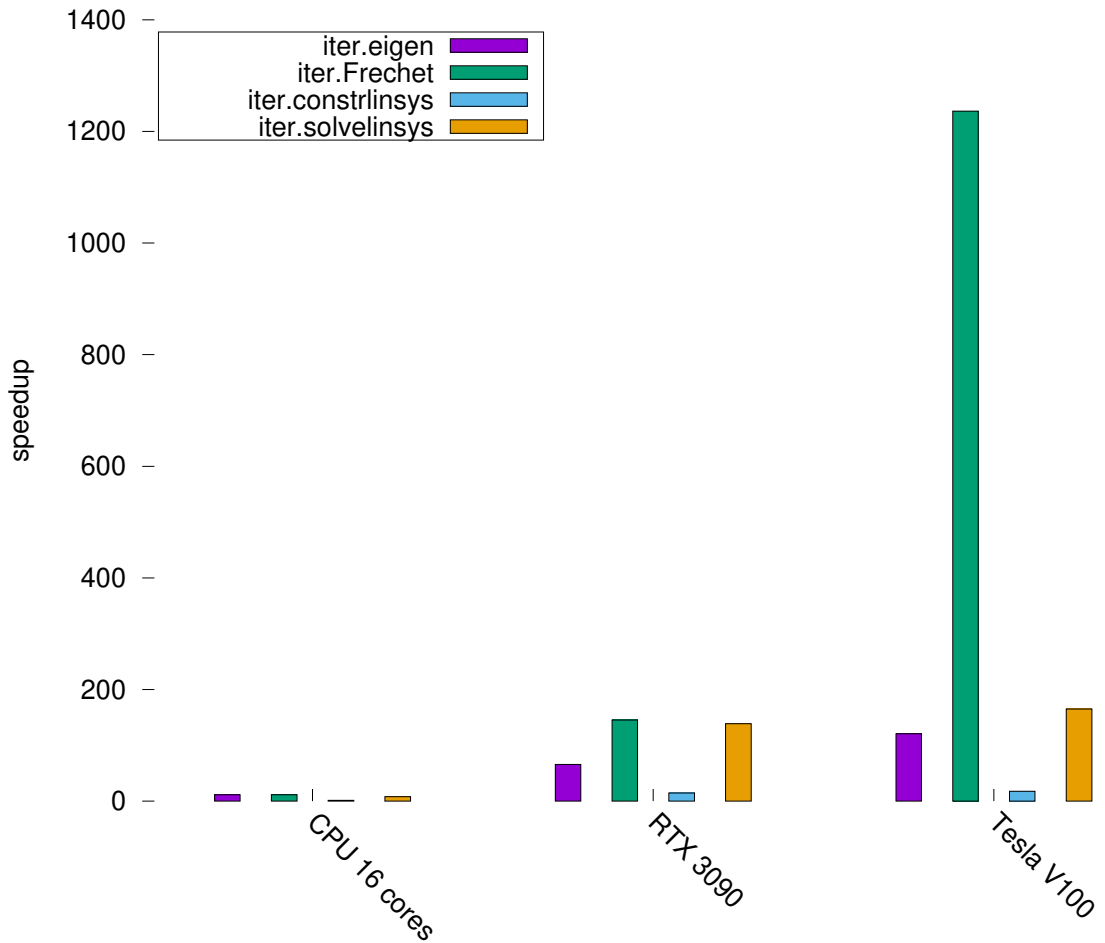
Performances of the code

The Schrödinger-Poisson block



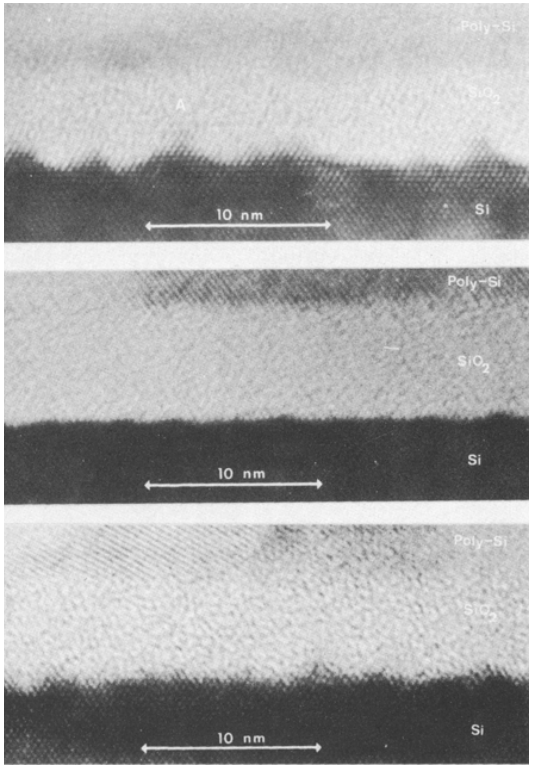
Performances of the code

Phases of the Schrödinger-Poisson block



Outline

Surface roughness



The real Si/SiO_2 interface is not smooth. This affects the properties of the transistor. The impact of the surface roughness is described as an additional scattering phenomenon.

Scattering operator

The usual structure of the scattering operator

$$\mathcal{Q}_{\nu,p}[f] = \int_{\mathbb{R}^2} \left[S_{(\nu,p,\boldsymbol{k}')\to(\nu,p,\boldsymbol{k})} f_{\nu,p}(\boldsymbol{k}') - S_{(\nu,p,\boldsymbol{k})\to(\nu,p,\boldsymbol{k}')} f_{\nu,p}(\boldsymbol{k}) \right] \mathrm{d}\boldsymbol{k}'.$$

The scattering rates are

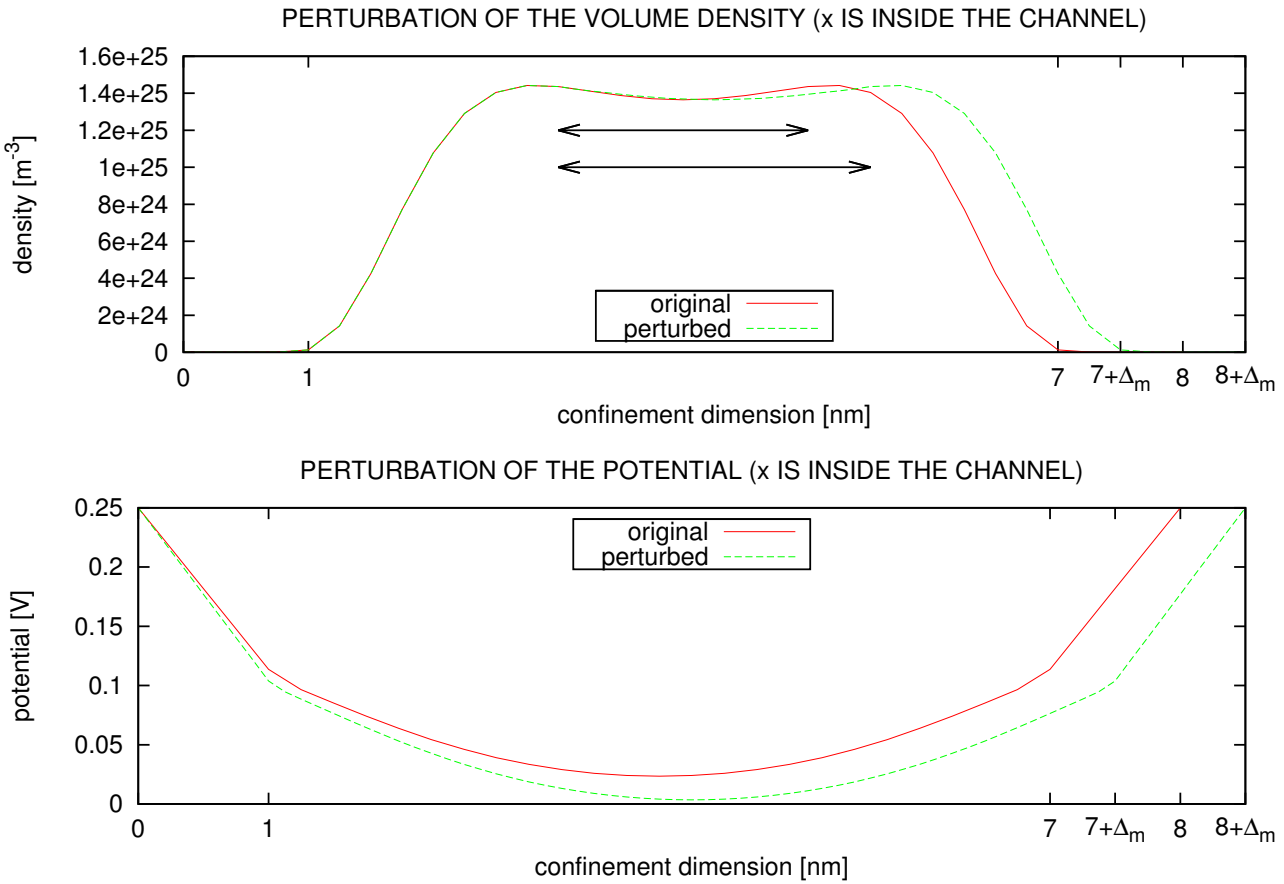
$$S_{\boldsymbol{k}\to\boldsymbol{k}'} = K^{\mathrm{upp}} \cdot \frac{\mathcal{I}^{\mathrm{upp}}_{\nu,p} \cdot \delta\left(\epsilon^{\mathrm{tot}}_{\nu,p}(\boldsymbol{k}) - \epsilon^{\mathrm{tot}}_{\nu,p}(\boldsymbol{k}')\right)}{\left(1 + \frac{|\boldsymbol{k}-\boldsymbol{k}'|^2 (\lambda^{\mathrm{upp}}_{\mathrm{m}})^2}{2}\right)^{3/2}} + K^{\mathrm{low}} \cdot \frac{\mathcal{I}^{\mathrm{low}}_{\nu,p} \cdot \delta\left(\epsilon^{\mathrm{tot}}_{\nu,p}(\boldsymbol{k}) - \epsilon^{\mathrm{tot}}_{\nu,p}(\boldsymbol{k}')\right)}{\left(1 + \frac{|\boldsymbol{k}-\boldsymbol{k}'|^2 (\lambda^{\mathrm{low}}_{\mathrm{m}})^2}{2}\right)^{3/2}}.$$

The overlap integral is replaced by a first-order approximation for the electric field's perturbation:

$$\mathcal{I}^{\mathrm{upp}}_{(\nu,p)}(x) = \left| \int_0^{L_z} |\psi_{\nu,p}(x,z)|^2 \, \frac{\Delta V^{\mathrm{upp}}(x,z)}{\Delta^{\mathrm{upp}}_{\mathrm{m}}} \, \mathrm{d}z \right|^2.$$

All the magnitudes will be adimensionalized.

Scattering operator



Scattering operator

The integrator performs several computational phases:

(i). It stretches by interpolation the total volume density onto the extended domain $\check{z}$

$$\check{N}_{i,\check{\jmath}} = \mathrm{li}\left[N_{i,.}\right]\left(S^{-1}(\check{z}_{\check{\jmath}})\right).$$

(ii). It solves on $\check{z}$ the Poisson equation $\left(D\,\check{V}\right)_{i,\check{\jmath}} = -C_p\left(\check{N}_{i,\check{\jmath}} - (\check{N}_{\mathrm{D}})_{i,\check{\jmath}}\right)$.

(iii). It interpolates $\left(\Delta V^{\mathrm{upp}}\right)_{i,j} = V_{i,j} - \mathrm{LI}\left[\left(\check{V}\right)_{i,.}\right](z_j)$, then $\left(\Delta V^{\mathrm{low}}\right)_{i,j} = V_{i,j} - \mathrm{LI}\left[\left(\check{V}\right)_{i,.}\right]\left(z_j + \frac{\Delta_{\mathrm{m}}}{L_z}\right)$.

(iv). It integrates

$$\mathcal{I}_{\nu,p,i} = \underbrace{\left(\Delta z \sum_{j=1}^{N_z-2} \left(\psi_{\nu,p,i,j}\right)^2 \Delta V^{\mathrm{upp}}_{i,j}\right)^2}_{\mathcal{I}^{\mathrm{upp}}_{\nu,p,i}} + \underbrace{\left(\Delta z \sum_{j=1}^{N_z-2} \left(\psi_{\nu,p,i,j}\right)^2 \Delta V^{\mathrm{low}}_{i,j}\right)^2}_{\mathcal{I}^{\mathrm{low}}_{\nu,p,i}}.$$

Scattering operator

(v). It computes the gain and loss parts of the scattering operator:

$$\mathcal{Q}^{\mathrm{SR,gain}}_{\nu,p,i,\ell,m} = \mathcal{C}\,\mathcal{I}_{\nu,p,i}\,s_{\nu,\ell}\,\Delta\phi \sum_{m'=0}^{N_\phi-1} \Phi_{\nu,p,i,\ell,m'}\,\mathcal{D}_{\nu,\ell,m,m'} \tag{1}$$

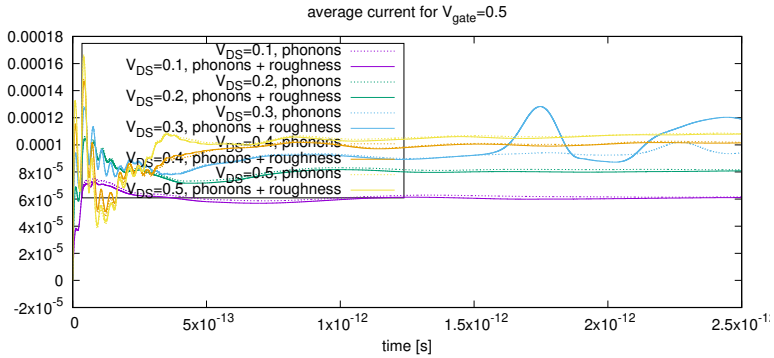
$$\mathcal{Q}^{\mathrm{SR,loss}}_{\nu,p,i,\ell,m} = \mathcal{C}\,\mathcal{I}_{\nu,p,i}\,s_{\nu,\ell}\,\Phi_{\nu,p,i,\ell,m}\,\widetilde{\mathcal{D}}_{\nu,\ell,m}. \tag{2}$$

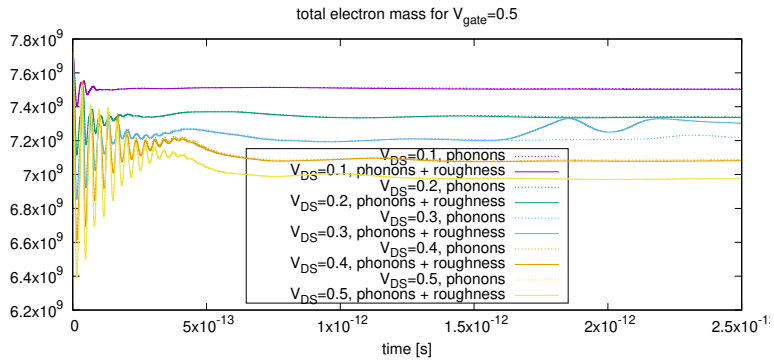
where

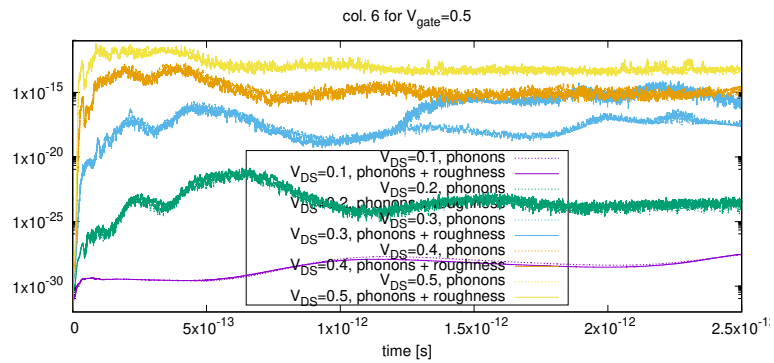
$$\mathcal{D}_{\nu,\ell,m,m'} = \left(1 + 4(k^*)^2\,\lambda_{\mathrm{m}}^2\,w_\ell(1+\alpha_\nu\epsilon^* w_\ell)\sin^2\left(\frac{\phi_m-\phi_{m'}}{2}\right)\right.$$
$$\left.\times\left[m_{x,\nu}\sin^2\left(\frac{\phi_m+\phi_{m'}}{2}\right) + m_{y,\nu}\cos^2\left(\frac{\phi_m+\phi_{m'}}{2}\right)\right]\right)^{-3/2} \tag{3}$$

$$\widetilde{\mathcal{D}}_{\nu,\ell,m} = \Delta\phi \sum_{m'=0}^{N_\phi-1} \mathcal{D}_{\nu,\ell,m,m'}. \tag{4}$$

Preliminary results







Outline

Future work

- Validate and optimize the code including the roughness scattering phenomenon.
- Make the code available.
- Split the computation of the eigenstates from the rest.
- Extend or modify the code to simulate other objects.

GRAZIE!

The authors acknowledge Spanish projects **MTM2011-27739-C04-02**, **MTM2014-52056-P**, **PID2020-117846GB-I00**, and the European Fund for Development.