# SIESTA Solvers in HPC settings

Alberto García (ICMAB-CSIC, Barcelona)

SIESTA advanced workshop, 2-5 June 2025

MaX "*Materials Design at the Exascale*", has received funding from the European Union's Horizon 2020 framework, from Euro-HPC, and from individual countries funding agencies.

# The basic core of SIESTA

$$\psi_i(\boldsymbol{r}) = \sum_{\mu} \phi_\mu(\boldsymbol{r}) c_{\mu i},$$

$$\sum_{\nu\beta} (H^{\alpha\beta}_{\mu\nu} - E_i S_{\mu\nu}\delta^{\alpha\beta}) c^{\beta}_{\nu i} = 0$$

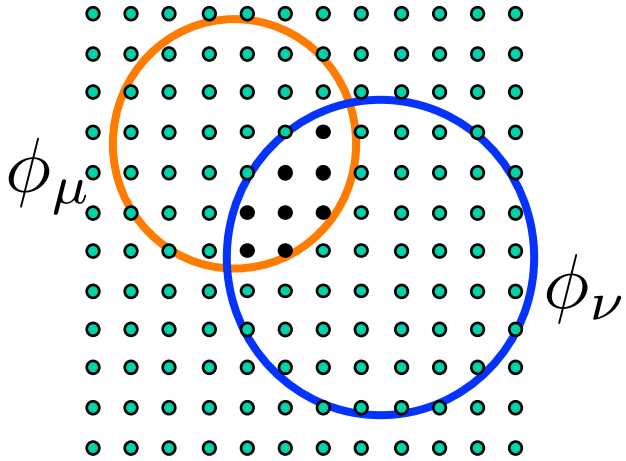
Generalized eigenvalue problem

$$\rho(\boldsymbol{r}) = \sum_{\mu\nu} \rho_{\mu\nu} \phi^*_\nu(\boldsymbol{r}) \phi_\mu(\boldsymbol{r})$$

$$S_{\mu\nu} = \langle \phi_\mu | \phi_\nu \rangle$$

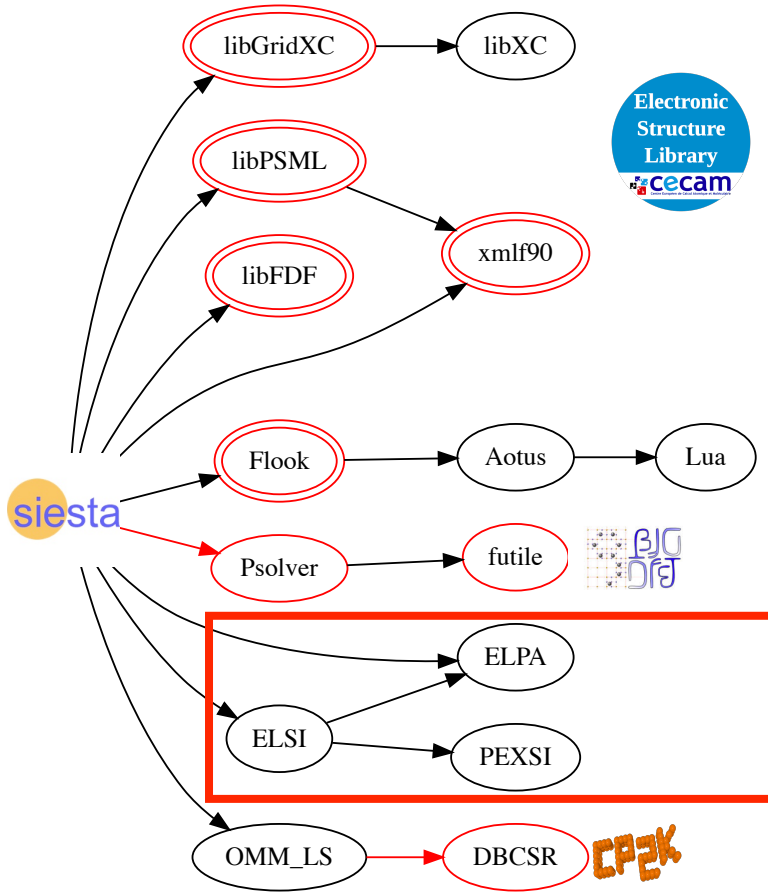
$$\rho_{\mu\nu} = \sum_i c_{\mu i} n_i c_{i\nu}$$

Density matrix




The SOLVER step takes most of the CPU time

# SIESTA: Leveraging external libraries, including solvers




Electronic Structure Library
cecam

Solvers: consolidated interfaces for continuously improved and performance-portable libraries.

# Diagonalization-based solvers

Need to use DIRECT solvers, as the number of desired eigenvectors is a substantial fraction of the matrix size

$$\sum_{\nu\beta}(H_{\mu\nu}^{\alpha\beta} - E_i S_{\mu\nu}\delta^{\alpha\beta})c_{\nu i}^{\beta} = 0$$

SIESTA uses pre-packaged libraries for this pure math problem:

- ScaLaPACK
  - pdsyev, pzheev and related drivers
  - MRRR
- ELPA: Alternative transformation sequence + optimizations
  https://elpa.mpcdf.mpg.de/

- Conversion of H and S to dense form
- Cholesky decomposition to reduce to standard eigenproblem
- Transformation to tri-diagonal form
- Solution of tri-diagonal problem
- Back-transformation

Cubic scaling with matrix size — Quadratic scaling for memory

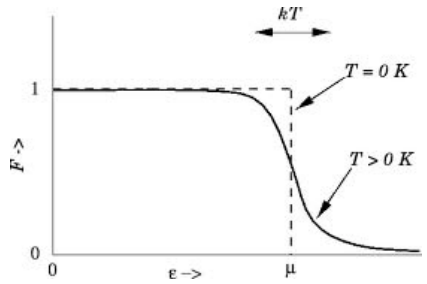
Still competitive for low-cardinality basis sets

# Direct solution for the density matrix

$$\hat{\rho} = f_\beta(\hat{H} - \mu)$$

$$f_\beta(\epsilon_i - \mu) = \frac{2}{1 + e^{\beta(\epsilon_i - \mu)}}$$

Fermi-Dirac function




Fermi Operator Expansion (FOE)

$$p(H) = \frac{c_0}{2} I + \sum_{j=1}^{n_{pl}} c_j T_j(H)$$

Calculation of the DM involves only (sparse) matrix-vector multiplications

CheSS library
(originally in BigDFT)

Linear-scaling

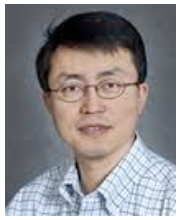
Stephan Mohr (BSC)

- Number of terms in the expansion can be large
- Efficiency increases for contracted basis sets.
- Exploring on-the-fly contraction

# Direct solution for the density matrix

## PEXSI: Pole Expansion plus Selected Inversion

(Lin Lin, Chao Yang, et al., Berkeley)



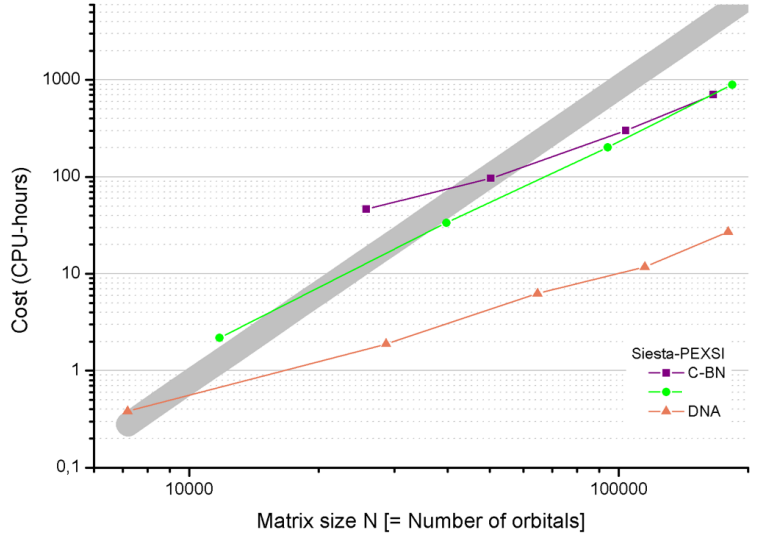
$$\hat{\rho} = Im\left(\sum_{l=1}^{P} \frac{\omega_l}{H-(z_l+\mu)S}\right)$$

For sufficiently big problems

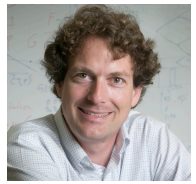
| | |
|---|---|
| (quasi-)1D: | $\mathcal{O}(N)$ |
| (quasi-)2D: | $\mathcal{O}(N^{3/2})$ |
| 3D: | $\mathcal{O}(N^2)$ |

(Due to sparsity of the target density matrix)

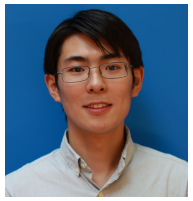
Relatively small number of poles (20-30)
Trivially parallelizable over them



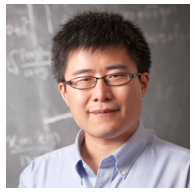
# ELSI initiative to integrate solver libraries



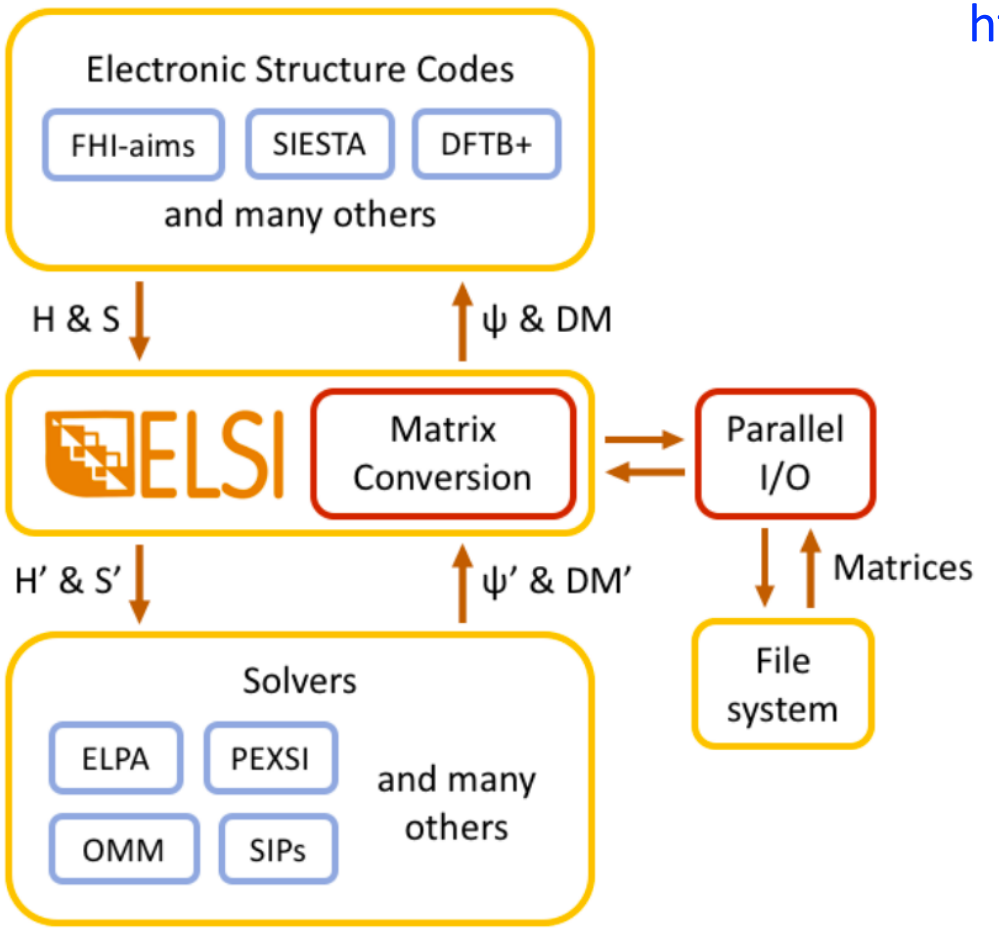
**Volker Blum, Duke**



**Lin Lin, Berkeley**



**Jiangfen Lu, Duke**

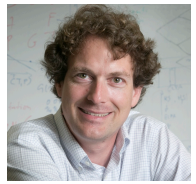



https://elsi-interchange.org

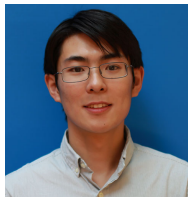
Interface in Siesta:

Collaboration with Victor Yu (Duke)

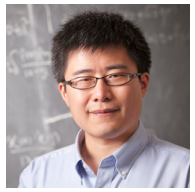
# ELSI initiative to integrate solver libraries



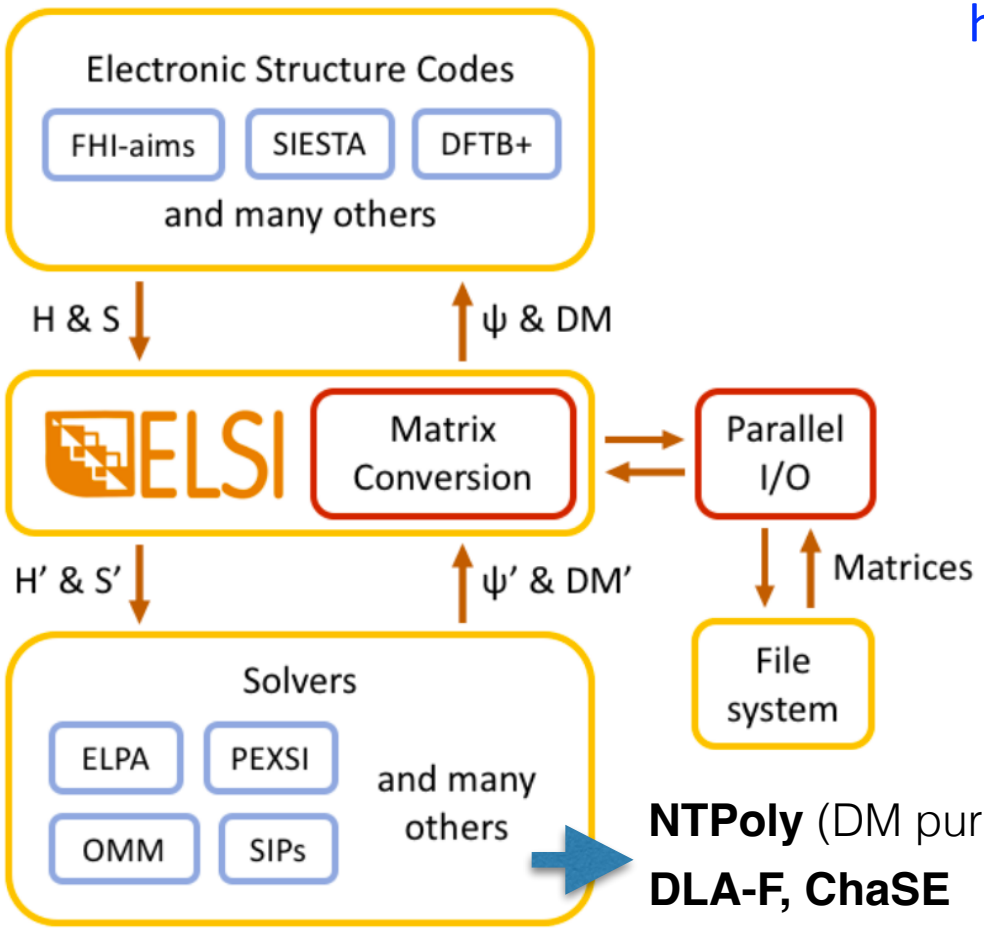
**Volker Blum, Duke**



**Lin Lin, Berkeley**



**Jiangfen Lu, Duke**




https://elsi-interchange.org

Interface in Siesta:

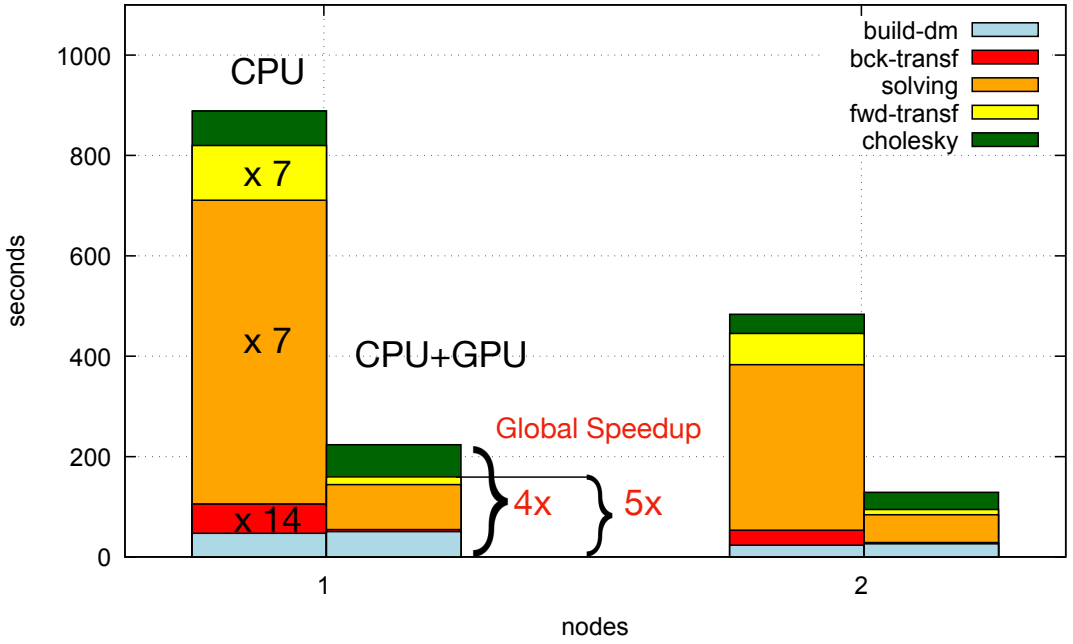
Collaboration with Victor Yu

**NTPoly** (DM purification, O(N))

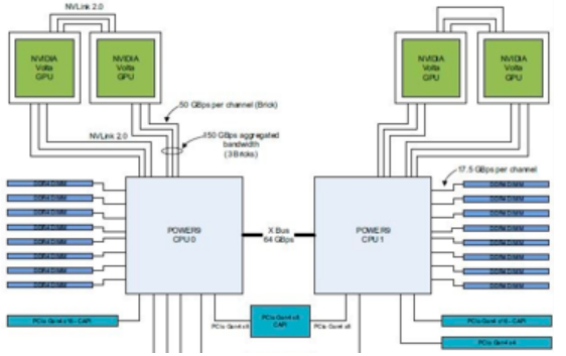
**DLA-F, ChaSE**

# GPU acceleration for diagonalization

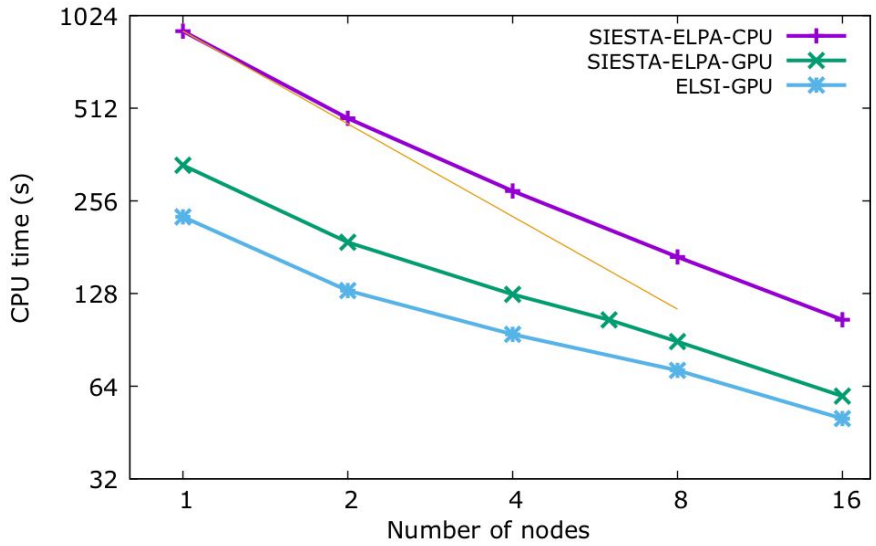
ELSI-ELPA GPU acceleration




Marconi-100 (CINECA): 32 CPUs+ 4 GPUs /node



Proper binding of GPUs to MPI ranks




Future enhancements in ELPA (better kernels) and in ELSI (e.g. build-DM stage) are integrated in SIESTA automatically

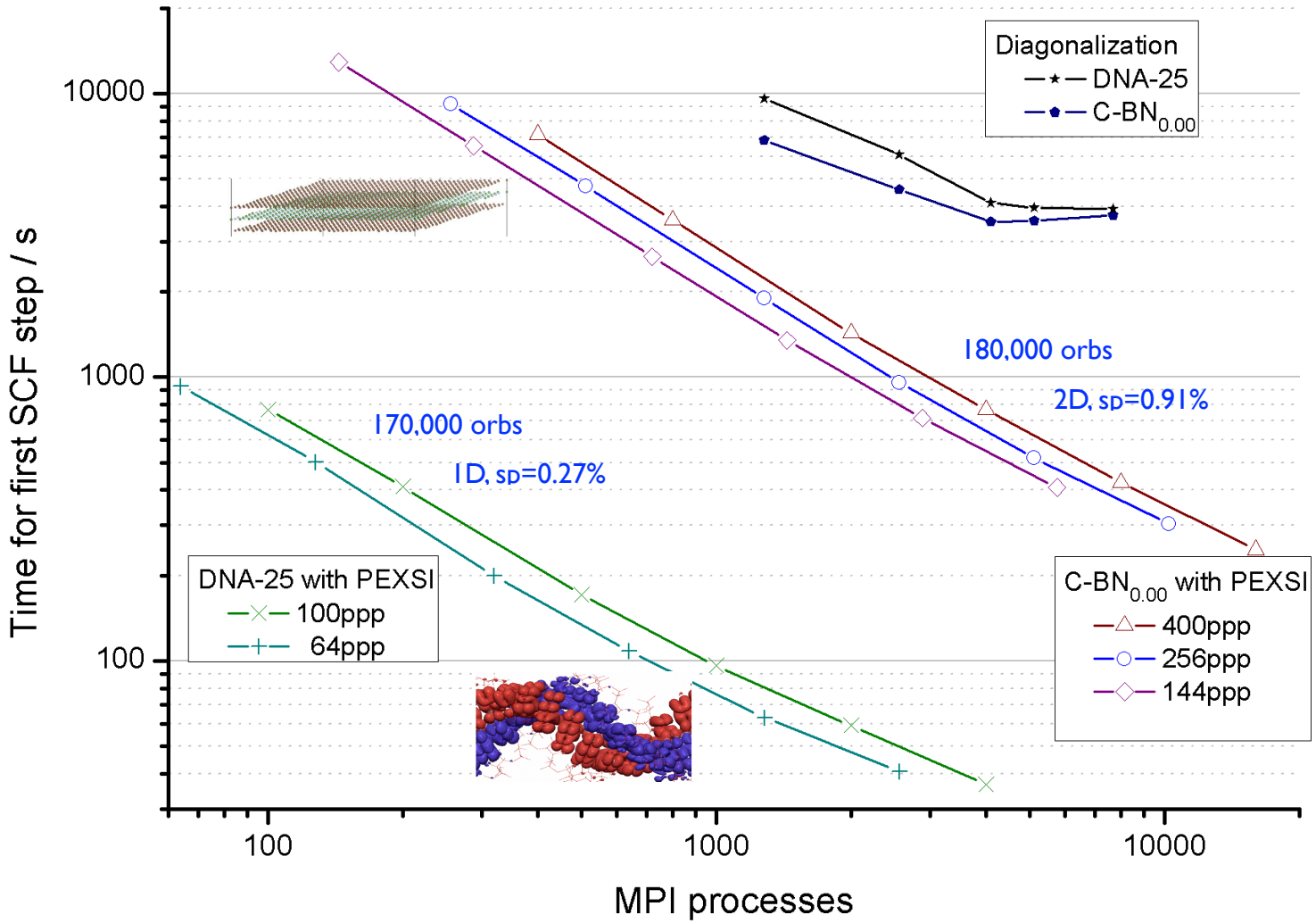
System: Si quantum dot, with approx 35000 orbs

# Massive scalability: PEXSI solver

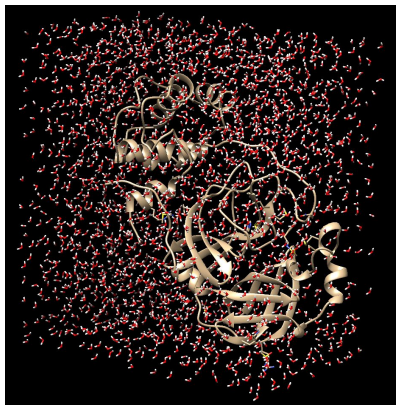
$$\hat{\rho} = \; Im\left(\sum_{l=1}^{P} \frac{\omega_l}{H-(z_l+\mu)S}\right)$$

PEXSI offers:

- Three levels of parallelization (over orbitals, poles, and chemical potential values)
- A reduced memory footprint (only sparse matrices are stored)
- Reduced complexity (maximum O($N^2$) size scaling)

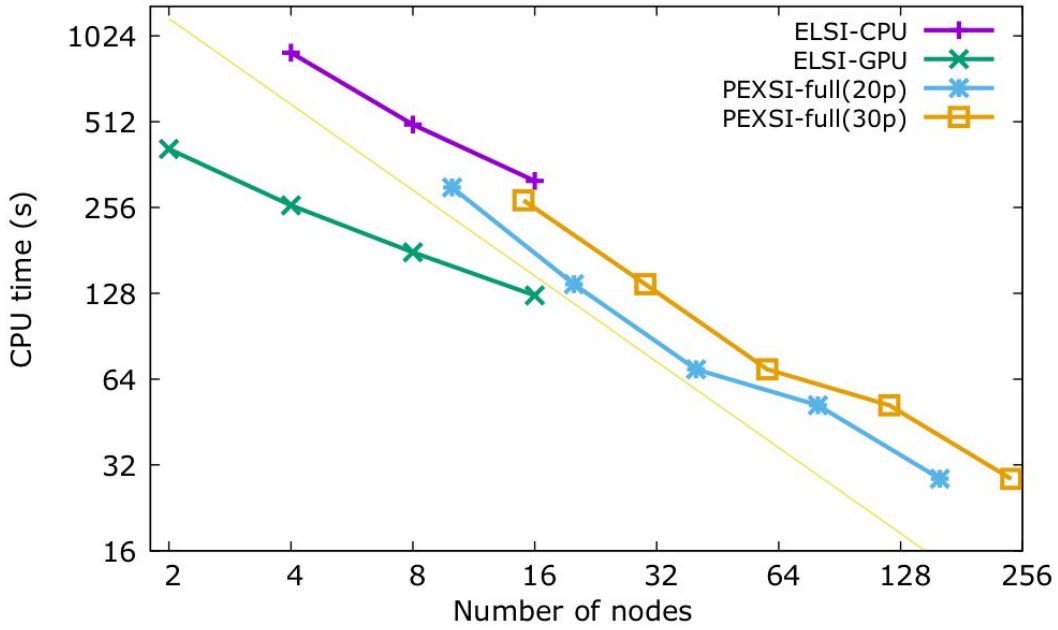


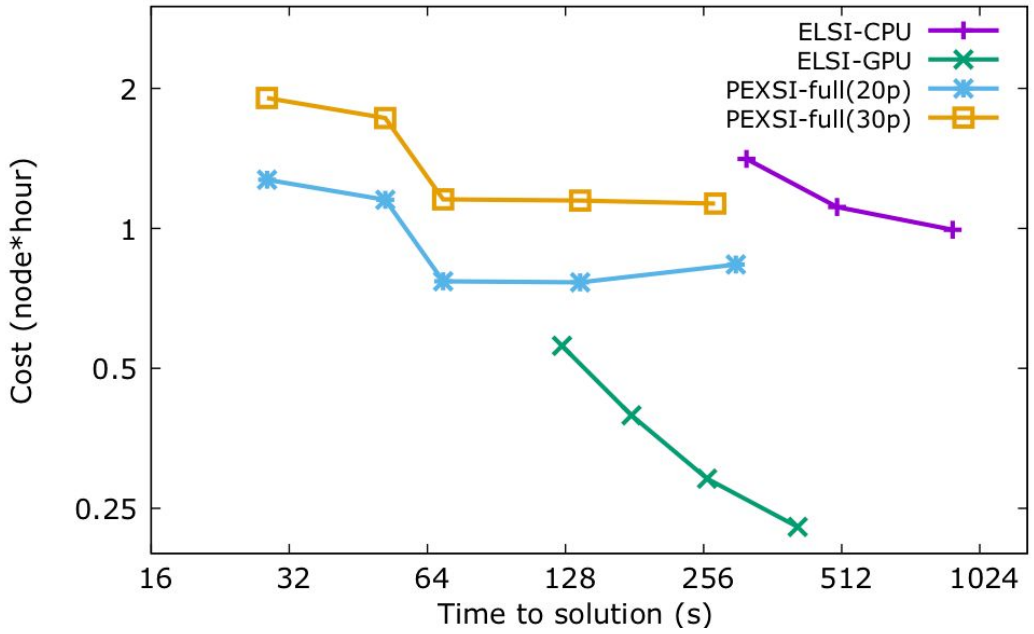
# Comparison of global efficiency of solvers for a very large problem



SARS CoV-2 $M^{pro}$ with solvation water molecules

Approx 8800 atoms; 58000 orbitals



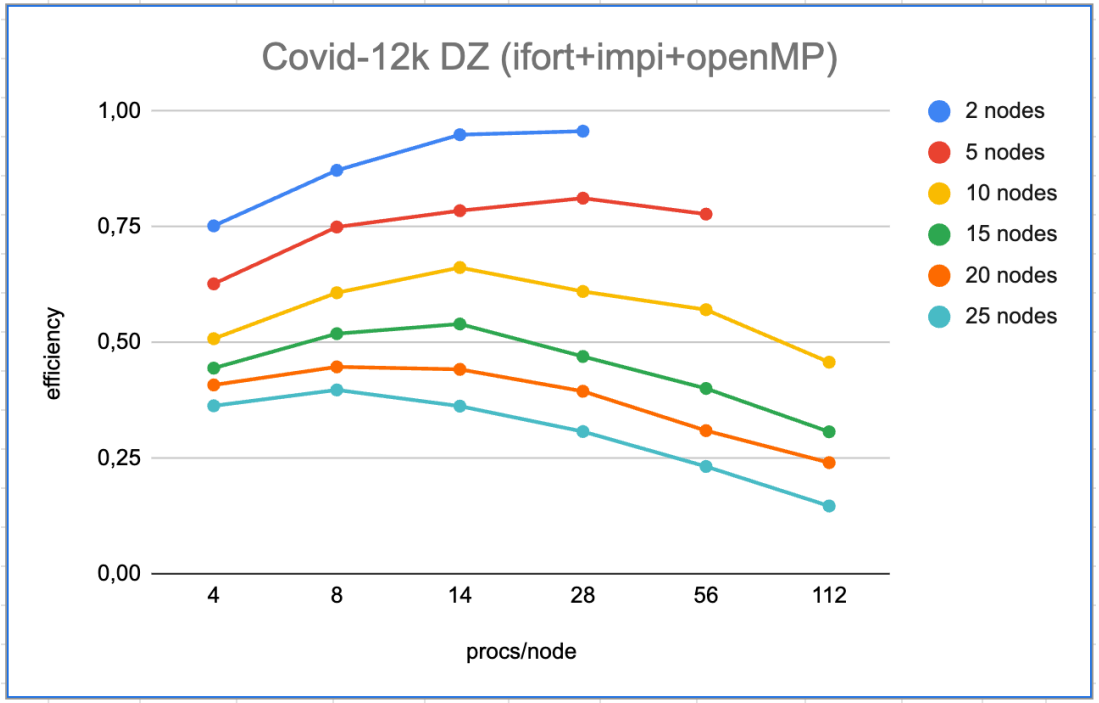



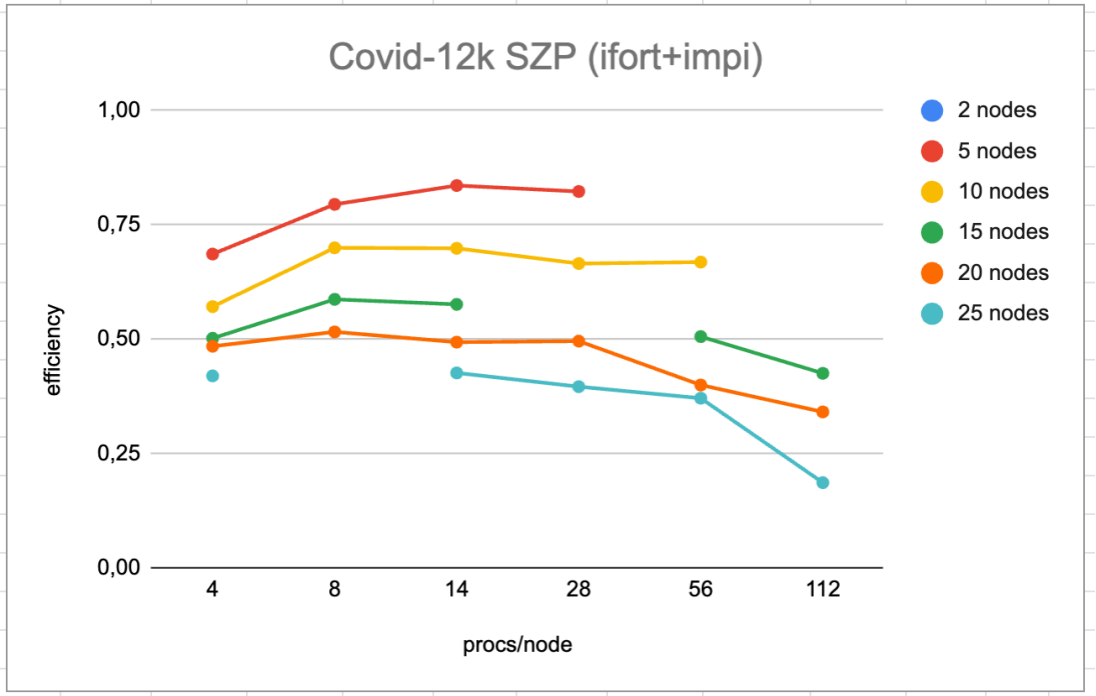

Work on GPU acceleration of PEXSI library is under way

# Scalability improvements with MPI and GPUs in diagonalization

Being tested within MaX at Leonardo (Laura Bellentani) and Marenostrum-5 (Rogeli Grima):

- **MPS** (attractive since we use many MPI ranks elsewhere). It seems to give the best performance.
- **NCCL** (only one MPI rank per GPU?)
- CUDA-Aware MPI (**HPCX-MPI**): recently installed in Leonardo. It needs code changes in ELPA, which might not happen soon.
- We worry about portability…




# MareNostrum 5 with MPS

# Solver selection in Siesta: SolutionMethod keywords

## Solution-Method diagon

ScalaPack solvers and
ELPA with native interface

diag-algorithm:

**divide-and-Conquer**
**expert**
**MRRR**
**ELPA | ELPA-2stage**
**ELPA-1stage**
**...**

## Solution-Method ELSI

Uniform interface for
the ELSI library of solvers:

elsi-solver:

- elpa
- omm
- pexsi
- ntpoly

## Solution-Method PEXSI

Original native interface
Iterative search for μ
instead of parallel interpolation

# Solver selection in Siesta: ELPA solver

Two flavors of the ELPA solver are available:

- ELPA1: One-stage tridiagonalization
- ELPA2: Two-stage tridiagonalization (with specialized kernels)

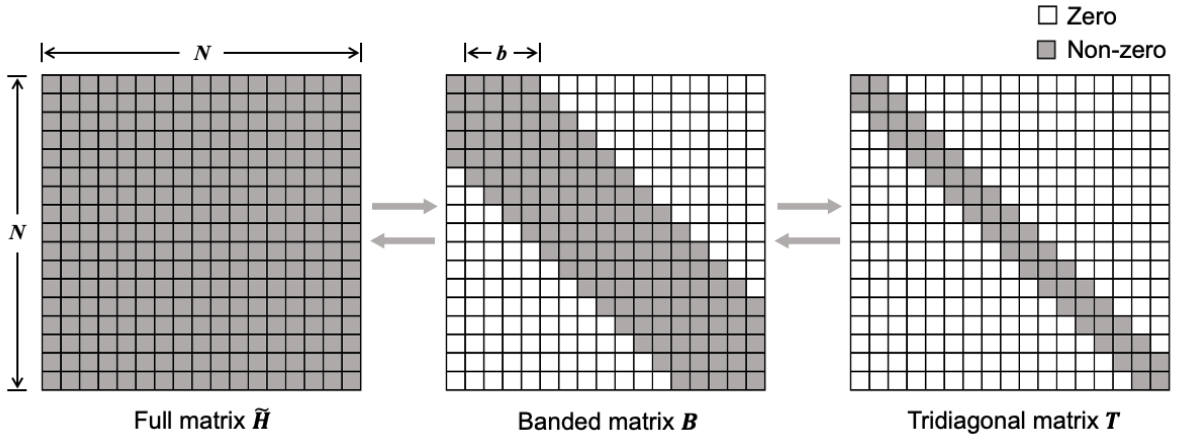
**ELSI.ELPA.Flavor (1 | 2)**
**ELSI.ELPA.GPU (0 | 1)**
**ELSI.ELPA.NSinglePrecision**

ELSI interface (preferred)

**diag-elpa-gpu (F | T)**

(SolutionMethod diagon)

ELSI-ELPA parallelizes over k-points, spins, and orbitals




Two-Stage Tridiagonalization in ELPA2

CPU-ELPA2 outperforms CPU-ELPA1 .
GPU-ELPA1 is marginally faster than GPU-ELPA2 for small node counts.
GPU-ELPA2 becomes faster than GPU-ELPA1 as the node count increases.

(Relative performance depends on archictecture and evolves with new releases…)

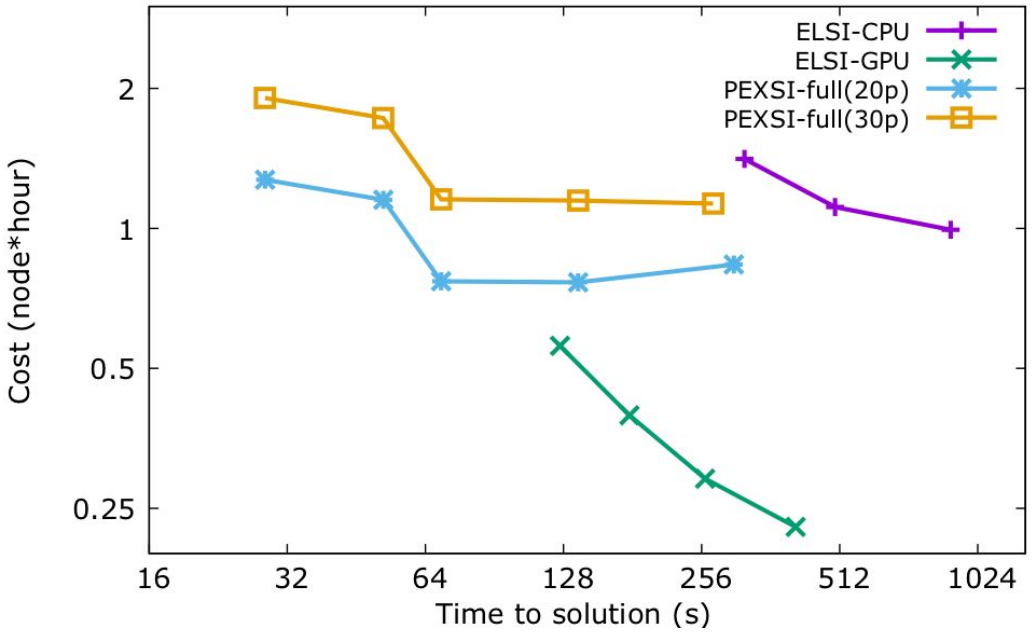
# Solver selection in Siesta: ELSI solvers (PEXSI)

```
solution-method elsi
elsi-solver pexsi
elsi-pexsi-tasks-per-pole 2
elsi-pexsi-number-of-poles 20
elsi-pexsi-number-of-mu-points 2
elsi-output-level 3
```

$$\hat{\rho} = Im\left(\sum_{l=1}^{P} \frac{\omega_l}{H-(z_l+\mu)S}\right)$$

(tpp) Configurable

n_mu=2 is typically appropriate

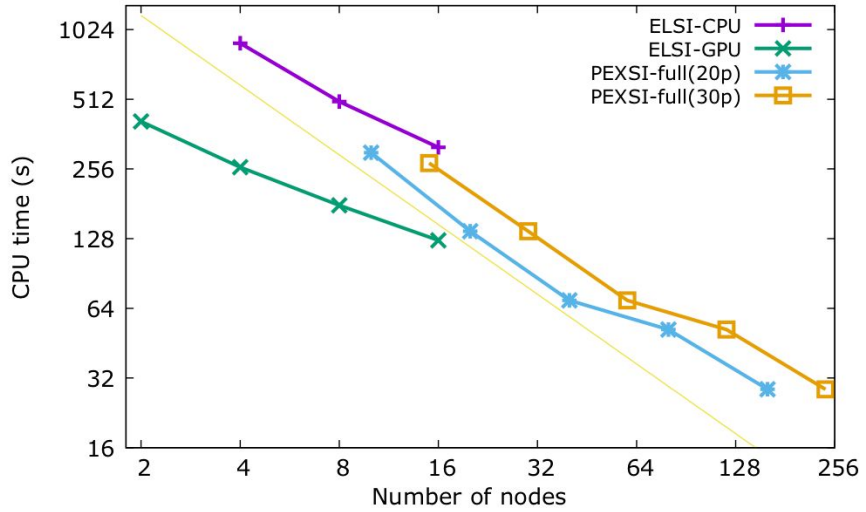



Maximum parallelization levels:

tpp=2: 2*20*2: 80 MPI ranks
tpp=32: 32*20*2: 1280 MPI ranks (40 32-cpu nodes)
tpp=64: 64*20*2: 2560 MPI ranks (80 32-cpu nodes) (**)

# Solvers: Parameters that affect performance

- Use the right algorithm
- Proper number of MPI ranks
- (MPI ranks / number of GPUs)
- Diag.Blocksize
- Use tailored NumberOfEigenStates
- Consider ELSI.ELPA.N.SinglePrecision




+ Architectural and systems issues:
  + Proper building (e.g. external ELPA library)
  + Mapping to underlying hardware

https://gitlab.com/siesta-project/ecosystem/build-tools.git

(+ Lecture on building, deployment, and execution)

# GPU binding

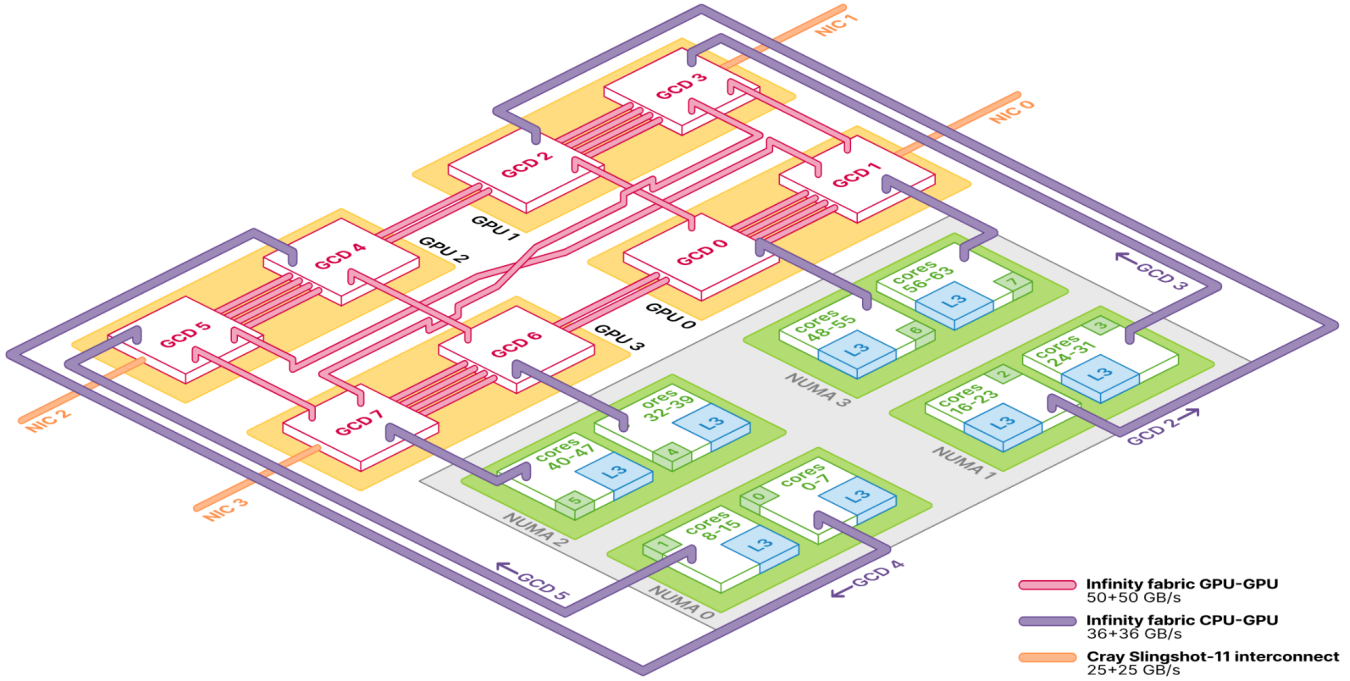
## GPU Binding

Courtesy: Federico Pedron

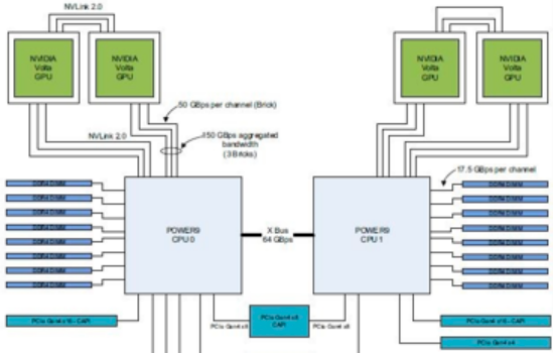
**GPU Binding is extremely important**

By GPU binding, we mean setting up proper CPU bindings and environment variables so that **only CPUs that are physically close to a GPU use that same device**.

Otherwise **performance can degrade pretty fast.**




Source: LUMI Documentation



Source: Marconi100 Docs

# GPU and CPU binding

GPU Binding

```
#!/bin/bash
CPU_ID=$(cat /proc/self/stat | awk '{print $39}')

limit0=40
limit1=48
limit2=104
limit3=112

if [ $CPU_ID -lt $limit0 ]
then
   export CUDA_VISIBLE_DEVICES=0

elif [ $CPU_ID -lt $limit1 ]
then
   export CUDA_VISIBLE_DEVICES=1

elif [ $CPU_ID -lt $limit2 ]
then
   export CUDA_VISIBLE_DEVICES=2

else
   export CUDA_VISIBLE_DEVICES=3

exec $*
```

**CPU ID, not Task ID!**

```
mpirun -np 32 \
       --cpu-list "32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111" \
       --bind-to cpu-list:ordered  deucalion-gpu-bind.sh siesta auwat.fdf > auwat.out
```

# GPU and CPU binding

GPU Binding

## Where can we find this info?

- From the HPC Center **documentation** (with varying degrees of detail).
- Running commands like **$ nvidia-smi topology**, for NVidia cards (on compute nodes).
- Running commands like **$ hwloc** (on compute nodes).

## And even then…

You will still need to search for the right number of MPI tasks and OpenMP threads per GPU.

# Thanks !