# Solvers: diagonalization, OMM, PEXSI, CheSS. Parallelization issues.

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# Solvers: diagonalization, OMM, PEXSI, CheSS.

# Calculating the density kernel

A typical SCF cycle in DFT codes looks as follows:

1. calculate
   - Hamiltonian matrix $H_{\alpha\beta} = \langle \phi_\alpha | \mathcal{H} | \phi_\beta \rangle$
   - overlap matrix $S_{\alpha\beta} = \langle \phi_\alpha | \phi_\beta \rangle$
2. calculate the density kernel **K** out of **H** and **S**
3. calculate
   - energy $E = Tr(\mathbf{KH})$
   - electronic density $\rho(\mathbf{r}) = \sum_{\alpha,\beta} \phi^*_\alpha(\mathbf{r}) K_{\alpha\beta} \phi_\beta(\mathbf{r})$
4. update the Hamiltonian operator $\mathcal{H}$ according to the new electronic density $\rho$
5. start over again

Limiting factor in most calculations (in particular big ones) with SIESTA: Calculation of the density matrix

# Available solvers

SIESTA offers various solvers to calculate the density kernel:

- Diagonalization (various flavors)
- Orbital minimization method (OMM)
- PEXSI
- CheSS

The choice of the most suited method depends on the specific calculation:

- system size
- sparsity of the matrices
- HOMO-LUMO gap
- used basis set
- etc.

# Diagonalization

Most straightforward approach to calculate the density kernel:

1. solve the generalized eigenvalue problem $\mathbf{H}\mathbf{c}_i = \epsilon_i \mathbf{S}\mathbf{c}_i$
2. calculate the density kernel as $K = \sum_i \mathbf{c}_i \mathbf{c}_i^T$

Advantages:

- exact calculation without any approximations
- universally applicable
- there exist highly optimized libraries

Shortcomings:

- possible sparsity of the matrices cannot be exploited
- cubic scaling with system size
- hard to parallelize

# Various flavors for the diagonalization

There are various libraries to diagonalize a matrix:

- ScaLAPACK: parallel version of LAPACK
  - most popular library for dense general purpose linear algebra
  - various diagonalization algorithms:
    - PDSYEV: based on tridiagonal QR iteration
    - PDSYEVD: based on Divide and Conquer algorithm
    - PDSYEVX: based on Bisection and Inverse Iteration
    - PDSYEVR: based on the parallel MRRR algorithm
  - often limited in parallel performance
- ELPA
  - better performance than ScaLAPACK using the same API
- MAGMA (Matrix Algebra on GPU and Multicore Architectures)
  - linear algebra library for heterogeneous architectures (CPU, Xeon Phi, GPU)
  - Has interfaces to LAPACK and ScaLAPACK routines, so easy to port

In SIESTA:

- ScaLAPACK (various flavors)
- ELPA

# Orbital minimization method

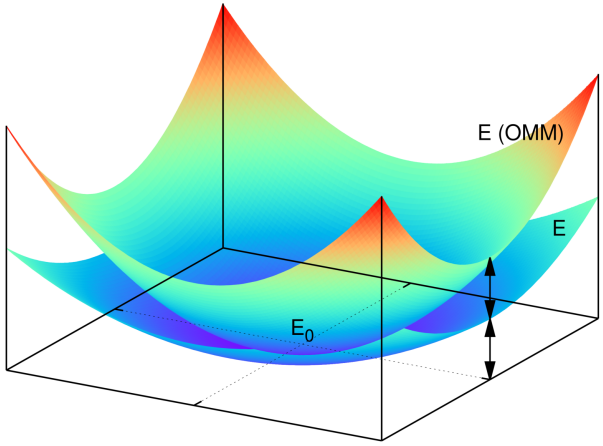
Find $n = N/2$ Wannier functions describing the occupied subspace by direct unconstrained minimization

- The original OMM functional [F. Mauri et al., Phys. Rev. B 47, 9973 (1993); P. Ordejón et al., Phys. Rev. B 48, 14646 (1993)]:

$$\tilde{E} = 2Tr\{[\mathbf{I} + (\mathbf{I} - \mathbf{S})]\mathbf{H}\}, \quad \text{with } S_{ij} = \langle \psi_i | \psi_j \rangle \,, H_{ij} = \langle \psi_i | \mathcal{H} | \psi_j \rangle$$

- For orthonormal set ($\mathbf{S} = \mathbf{I}$): $\tilde{E} = E = Tr(\mathbf{H})$
- For all other cases: $\tilde{E} \geq E$ if $\mathbf{H}$ is negative definite (easily fulfilled by shift)

Unconstrained (i.e. no explicit orthogonalization) global minimum of the function $\tilde{E}$ coincides with $E$

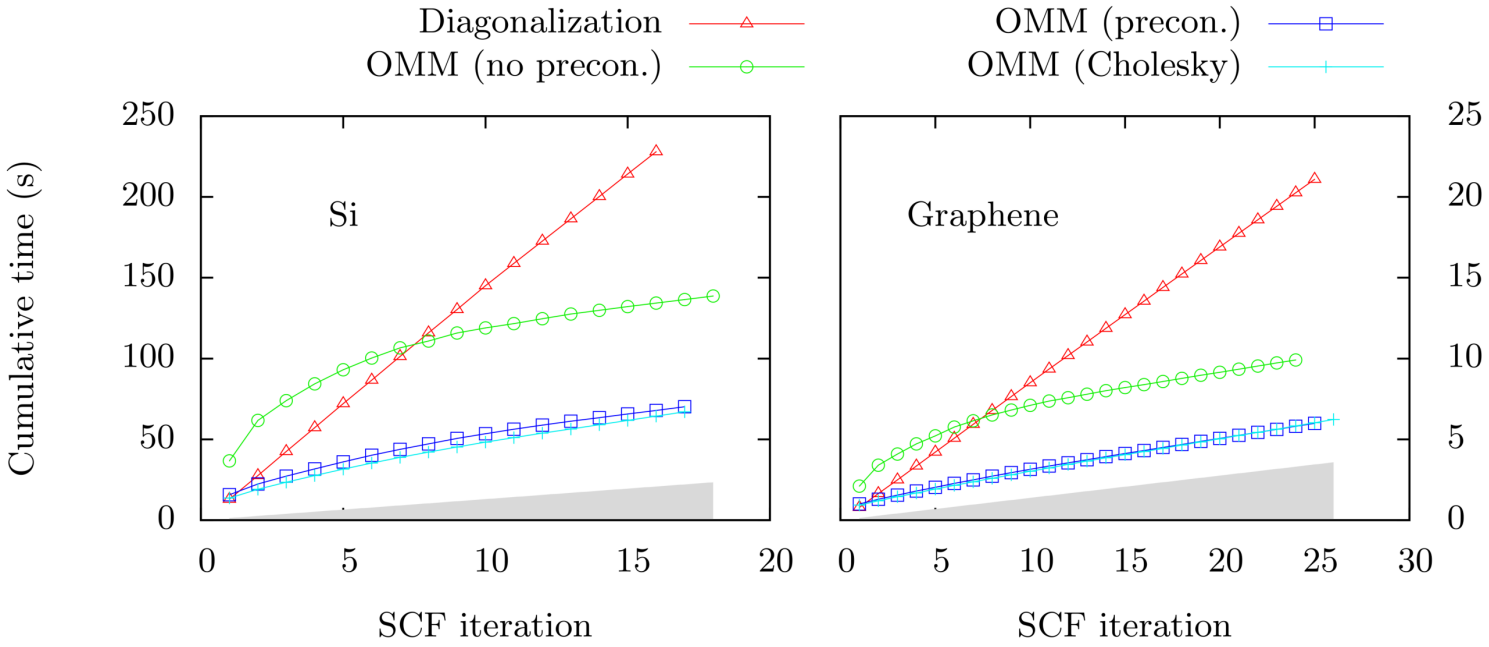


# Orbital minimization method – results

OMM has the potential to be a $\mathcal{O}(N)$ method, even if the version implemented in SIESTA is not (no localization constraints).

Still an interesting alternative to diagonalization

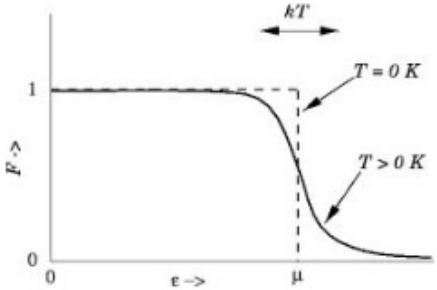
- At each SCF iteration, the results from the previous one can be reused as starting guess
- Especially towards the end of the SCF cycle potentially faster than diagonalization



# Fermi expansion methods

The density matrix can be calculated directly from the Hamiltonian:

$$\mathbf{K} = f(\mathbf{H}), \quad \text{with } f(\epsilon) = \frac{1}{1 + e^{\beta(\epsilon - \mu)}}$$




All we need is a computationally convenient representation of the Fermi function $f$.

Two possibilities:

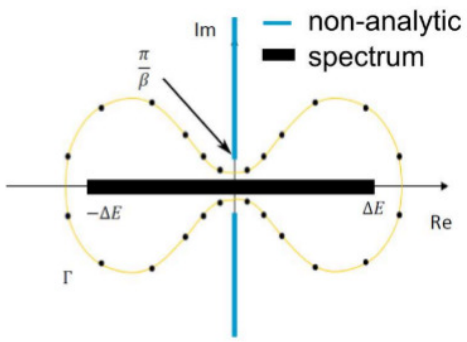
- rational expansion: PEXSI
- Chebyshev expansion: CheSS

# PEXSI

Approximate the Fermi function using a pole expansion in the complex plane:

$$f(\epsilon) \approx Im \sum_{l=1}^{n_p} \frac{w_l}{\epsilon - (z_l + \mu)}$$

$$K \approx Im \sum_{l=1}^{n_p} \frac{w_l}{\mathbf{H} - (z_l + \mu)\mathbf{S}}$$




Advantages:

- Only a small number of poles is required (typically about 40)
- Each pole independent

Most important task: Inversion of the matrices $\mathbf{H} - (z_l + \mu)\mathbf{S}$
$\Longrightarrow$ done with the Selected Inversion algorithm

# PEXSI

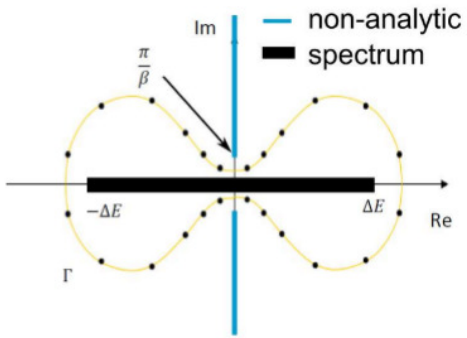
- PEXSI only calculates those elements of the density matrix which are required to calculate physical quantities (charge density, energy, forces, ...)
- Thus PEXSI can exploit the sparsity of the matrices (consequence of the localized character of the basis set $\{\phi_\alpha\}$)

Exact method (no approximations!), but reduced scaling.
For sufficiently big problems:

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

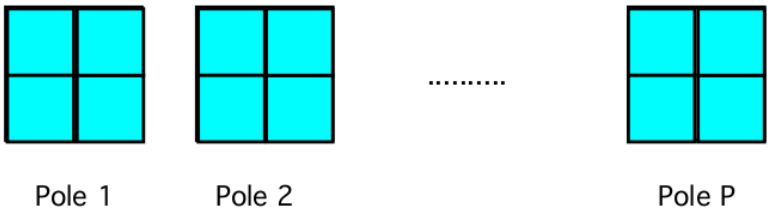
Number of poles depends on the inverse electronic temperature $\beta$ and the spectral width $\Delta E$.

However fast convergence with the number of poles $\Longrightarrow$ still applicable to metals!



# PEXSI – Scaling

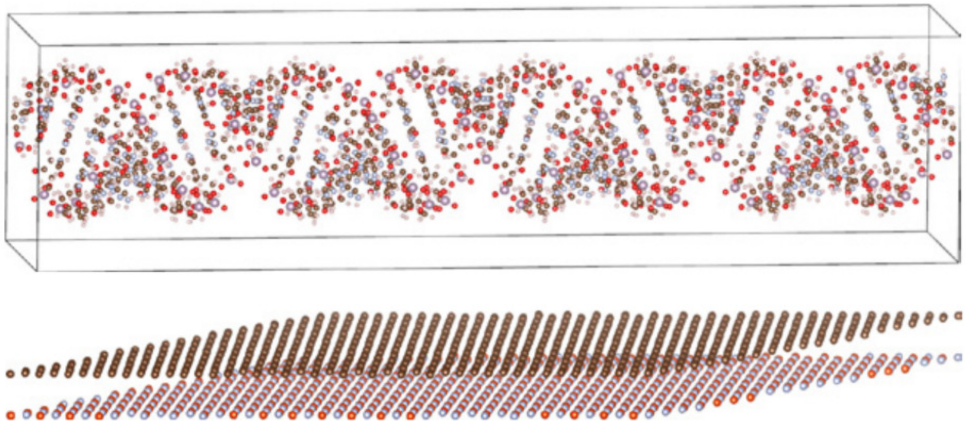
PEXSI exhibits ideal parallel scaling with respect to the poles:

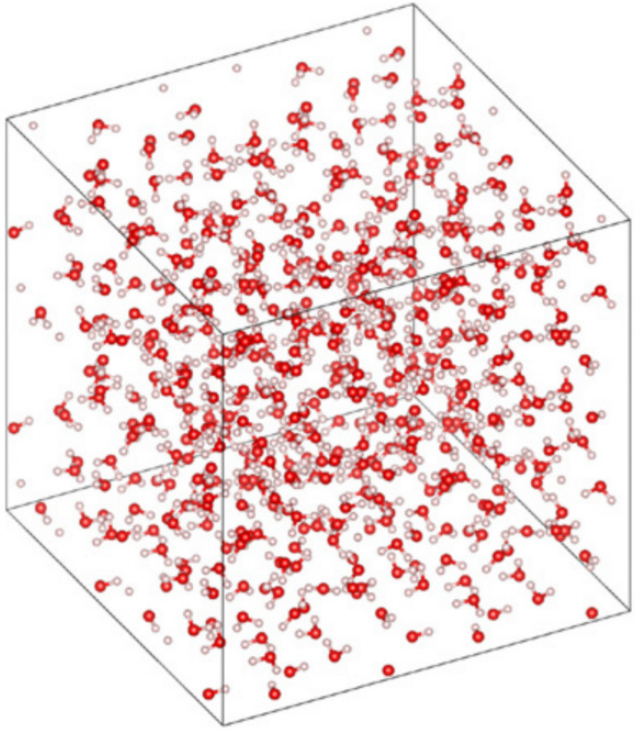



40 processors per pole $\times$ 40 poles:
160 processors

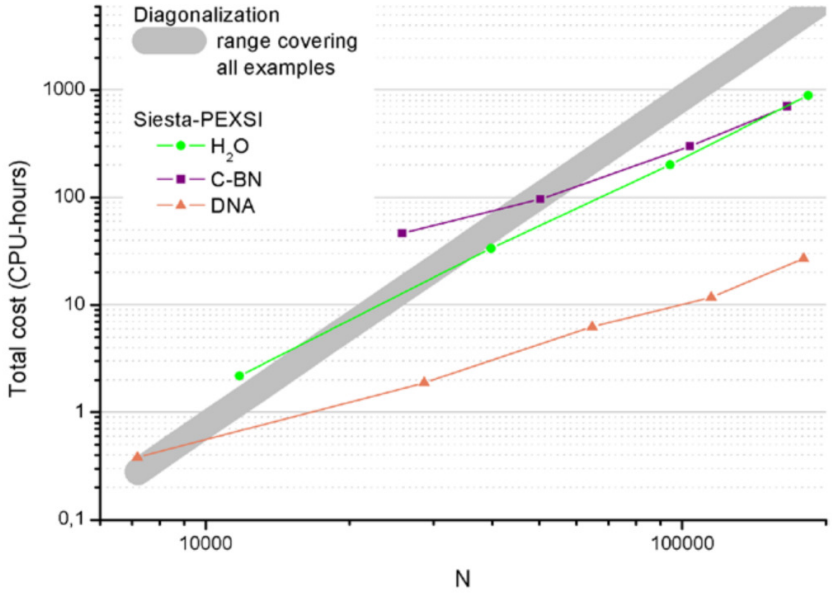
Test systems:

- 1D: DNA (up to 17875 atoms)
- 2D: C−BN (up to 12770 atoms)
- 3D: $H_2O$ (up to 24000 atoms)



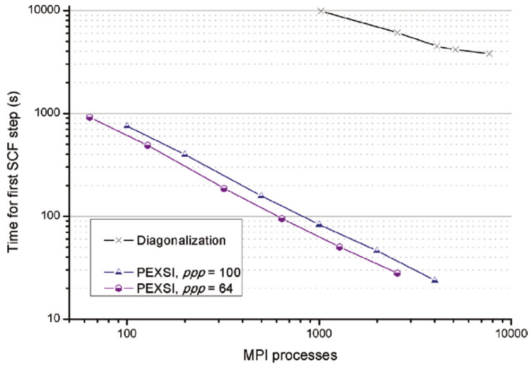


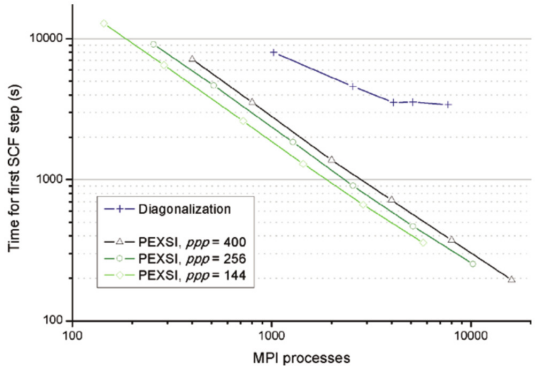
# PEXSI – Scaling

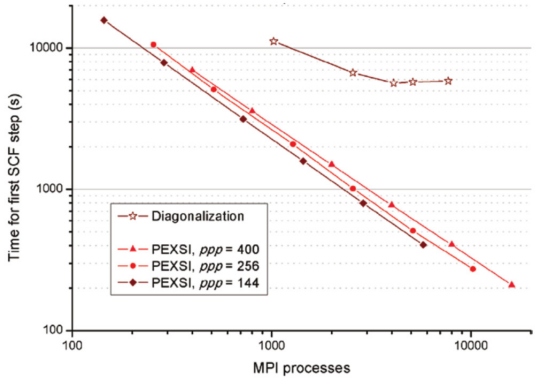



Fitted slopes:

- DNA: 1.3 ; C−BN: 1.7 ; $H_2O$: 2.2
- Deviations from perfect scaling: Due to parallelization issues
- Prefactor: "sparsity" of the system
- For large systems always faster than diagonalization



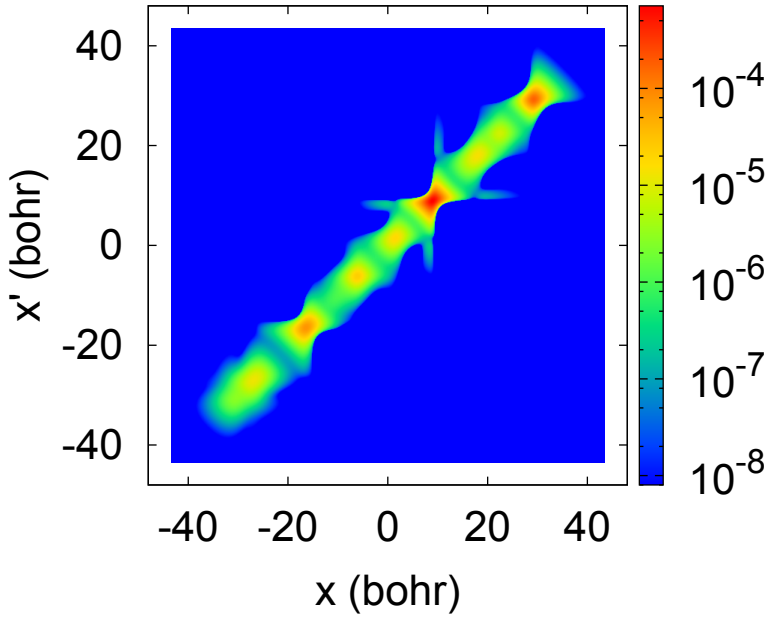
# $\mathcal{O}(N)$ schemes

The key for $\mathcal{O}(N)$ schemes is locality: "nearsightedness" principle (W. Kohn, Phys. Rev. Lett. **76**, 3168 (1996))

Example: Density matrix $F(\mathbf{r}, \mathbf{r}') = \sum_i f_i \psi_i(\mathbf{r}) \psi_i(\mathbf{r}')$

The matrix elements $F(\mathbf{r}, \mathbf{r}')$ decay rapidly with $|\mathbf{r} - \mathbf{r}'|$:

- insulators and metals at finite temperature: exponentially
- metals at zero temperature: algebraically

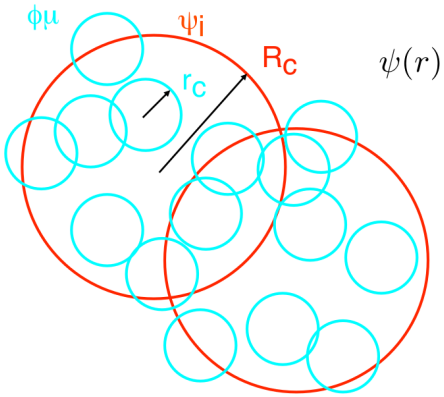



The decay length depends on the HOMO-LUMO gap of the system. $\mathcal{O}(N)$ schemes thus work best for insulators.

# $\mathcal{O}(N)$ schemes

Basic idea: Localization

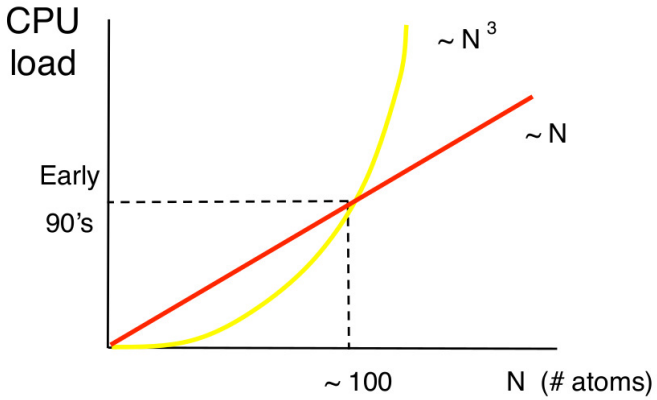
- Confine the orbitals within a sphere with cutoff $R_c$.
- Equivalent: Enforce the density matrix $F$ to be sparse.




$$\psi(r) = \sum_{\mu} c_{i\mu}\phi_{\mu}(r)$$

Justified by the nearsightedness.

$\mathcal{O}(N)$ usually have a larger prefactor than classical $\mathcal{O}(N^3)$ calculations
$\Rightarrow$ crossover point



# CheSS

Basic idea: Approximate the density matrix by a polynomial expansion:

- Use Chebyshev polynomials to avoid instabilities
- Shift and scale $\mathbf{H}$ such that eigenvalues lie in the range $[-1, 1]$ ($\bar{\mathbf{H}}$)
- Calculate the density matrix as

$$\mathbf{K} = f(\mathbf{H}) \approx \frac{c_0}{2}\mathbf{I} + \sum_{i=1}^{n_{pl}} c_i \mathbf{T}^i(\bar{\mathbf{H}})$$

Efficient and flexible approach:

- The coefficients $c_i$ can be cheaply calculated using textbook formulas (also for other expansions than the density matrix, e.g. the inverse)
- The Chebyshev polynomials fulfill the following recursion relation:

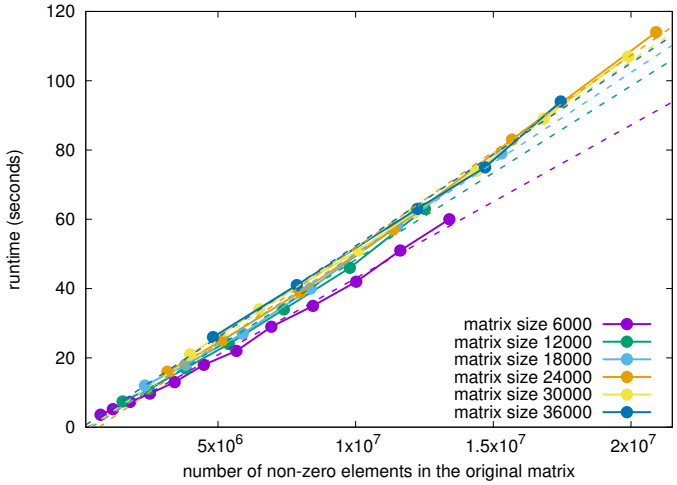
$$\mathbf{T}^0(\bar{\mathbf{H}}) = \mathbf{I} \quad ; \quad \mathbf{T}^1(\bar{\mathbf{H}}) = \bar{\mathbf{H}} \quad ; \quad \mathbf{T}^{j+1}(\bar{\mathbf{H}}) = 2\bar{\mathbf{H}}\mathbf{T}^j(\bar{\mathbf{H}}) - \mathbf{T}^{j-1}(\bar{\mathbf{H}}).$$

From this we see:

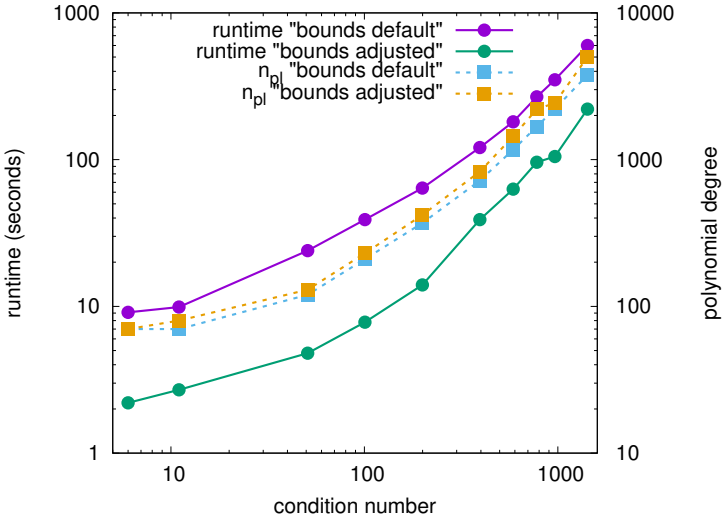
- Only matrix vector multiplications required, easily parallelizable
- $\mathcal{O}(N)$ method by restricting the multiplications to a sparsity pattern

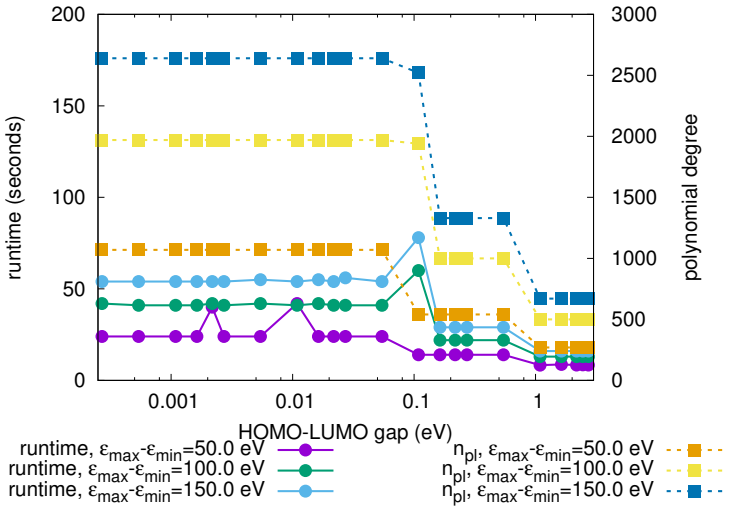
# CheSS

The performance of CheSS only depends on the number of non-zero entries of the matrices



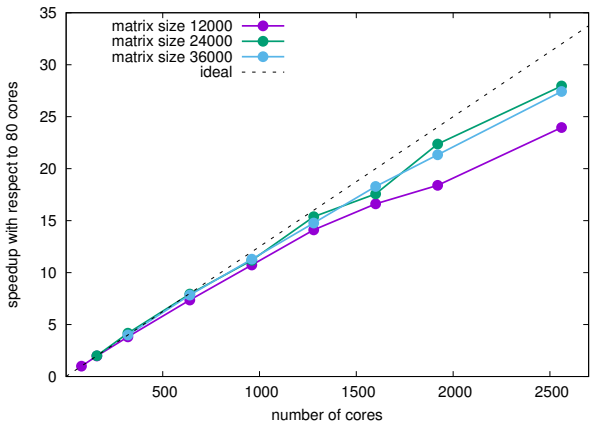

The algorithm works best for matrices with a small eigenvalue spectrum:



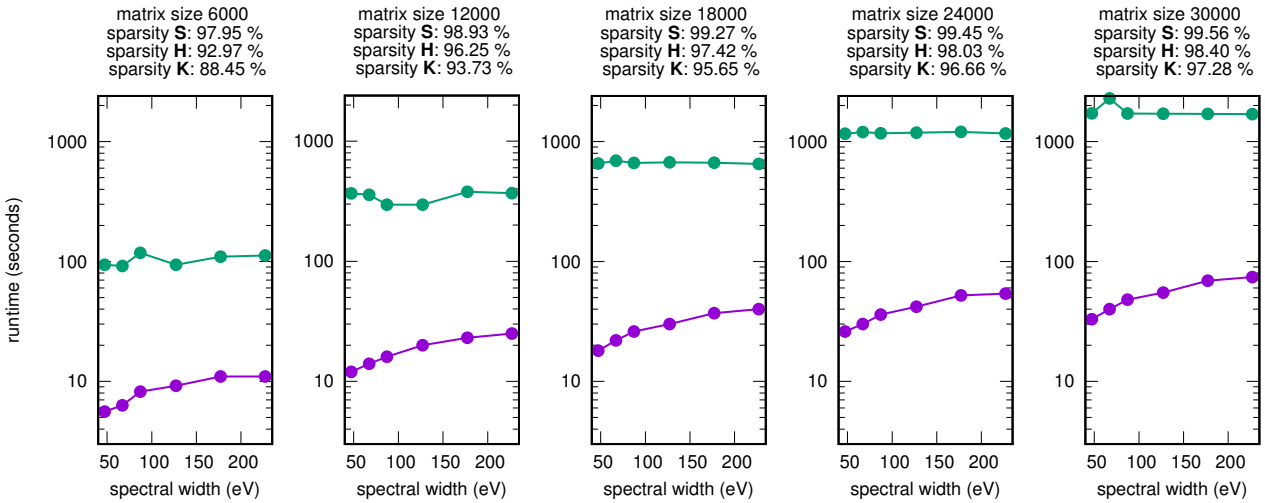



# CheSS

CheSS shows a very good parallel scaling (for both MPI and OpenMP)



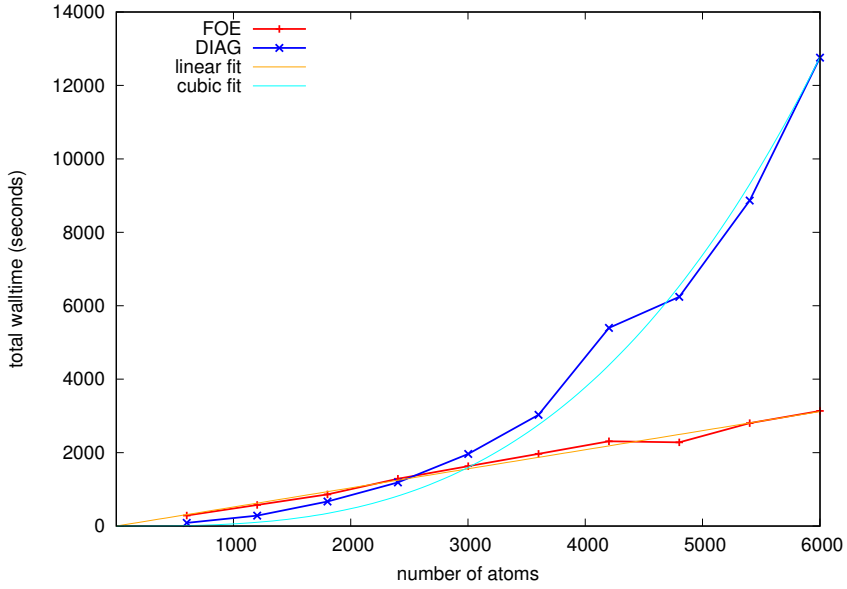

For hybrid MPI/OpenMP setup, CheSS can outperform PEXSI for appropriate systems (small eigenvalue spectrum, non-zero HOMO-LUMO gap)



# CheSS

SIESTA has been interfaced with CheSS (not yet in the official version).

Very simple test system:
alkane chain, DZP basis

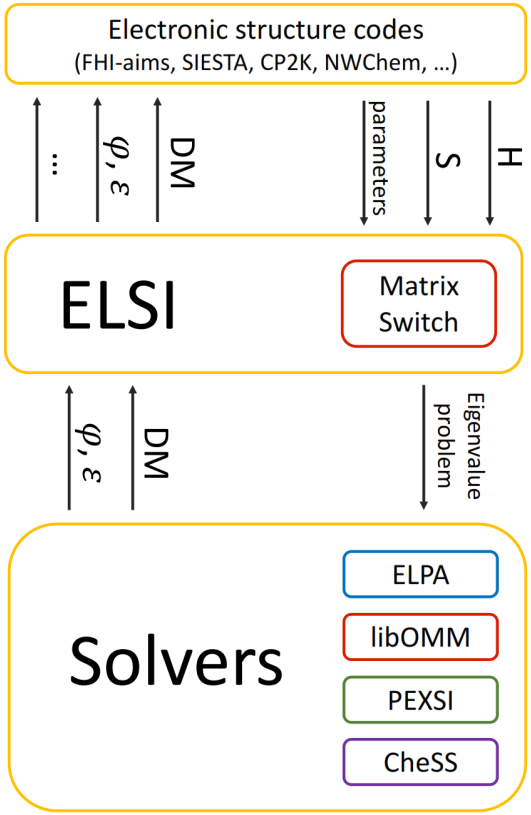



Most important remaining bottleneck: Reduce the spectral width of the SIESTA matrices (sort of contracted basis set?)

# ELSI

The ELSI project (ELectronic Structure Infrastructure) is an effort to unite various of the aforementioned solvers:

- ELPA
- OMM
- PEXSI
- CheSS




Ultimate goal:

- Provide one single interface to all these libraries
- A DFT code interfacing ELSI gets access to all these methods
- Easy case-by-case choice of the solver

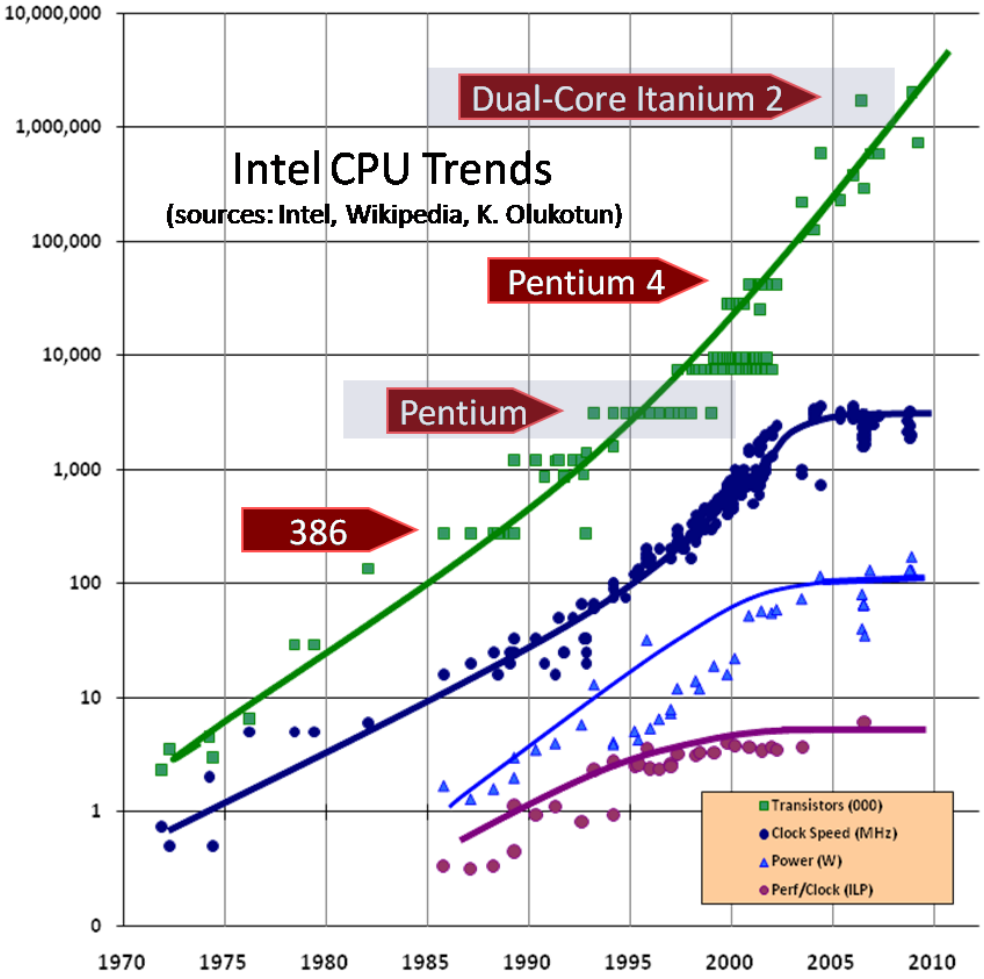
# Parallelization issues

# Need for parallelization

Clock speed of of a single core saturates since about 2005

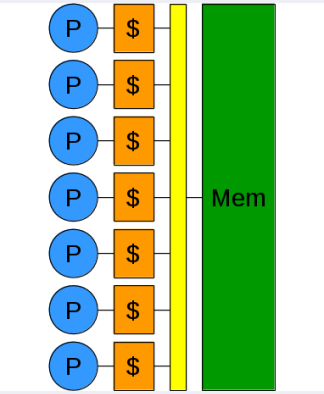
Performance gain only possible by using more cores at the same time

Applications must be highly parallelized

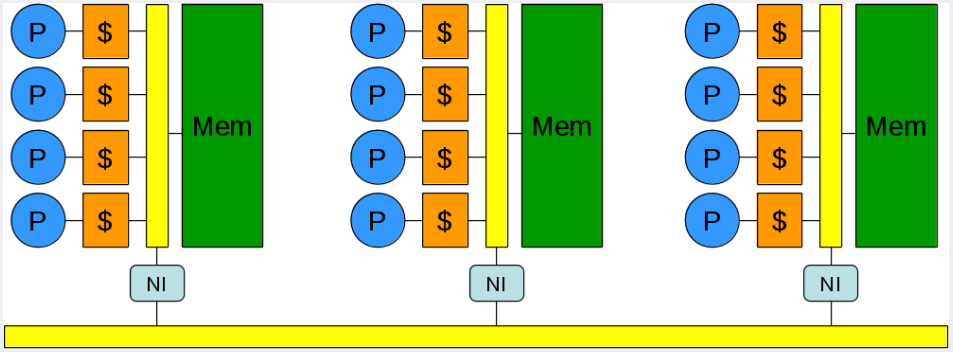


# Types of parallelism

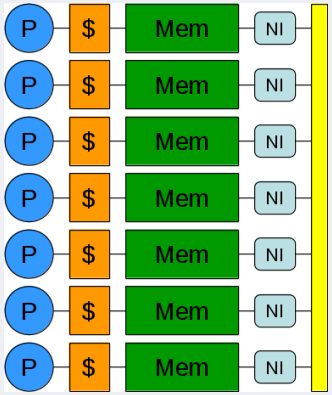
## shared memory



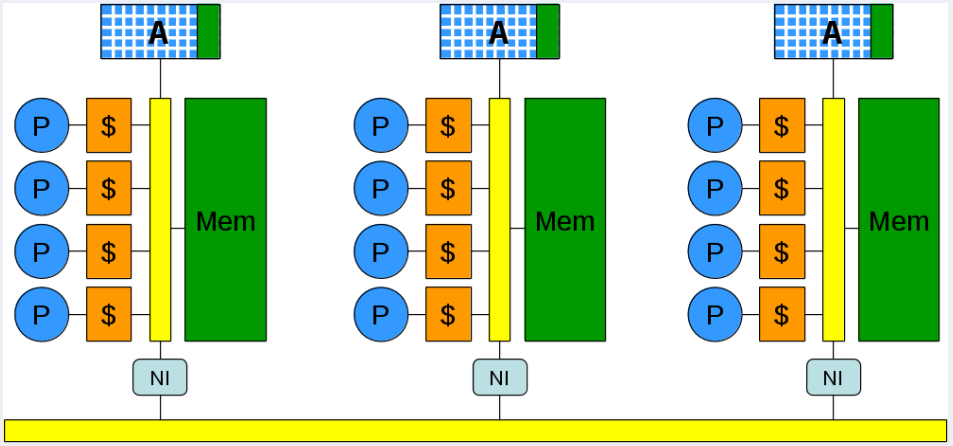

## hybrid distributed/shared memory




## distributed memory




## accelerators



# Characteristics of the parallelization schemes

Shared memory:

- done using OpenMP
- can often be added relatively easily on top of existing code (but getting good performance is not that easy!)
- limiting factor often memory bandwidth
- maximal speedup limited by number of cores per node

Distributed memory:

- distributing data is essential for large applications
- done using MPI (Message Passing Interface)
- introducing MPI often requires a major refacturing of the code
- limiting factor is the communication
- maximal speedup depends on application and architecture, in principle no upper bound

# Characteristics of the parallelization schemes

Hybrid distributed/shared memory:

- combination of MPI and OpenMP
- most complex form of parallelism, careful implementation required
- allows to increase the maximal speedup (multiplicative)
- allows to overcome the aforementioned constraints and limitation (e.g. memory, bandwidth, etc.)

Accelerators:

- often used to accelerate specific intensive parts of the code
- most popular ones: GPUs and MICs
- can be combined with other parallelization schemes
- maximal speedup limited by accelerator
- usually hard work to get good performance

# Parallelization in SIESTA

Parallel resources can be exploited by SIESTA in various ways:

- various levels of distributed memory parallelization using MPI:
  - k-point parallelism
  - Distributing orbitals and gridpoints
- recently shared memory using OpenMP was added
- parallelization of the various external solver:
  - BLAS: OpenMP
  - ScaLAPACK: MPI and OpenMP
  - PEXSI: heavy two-level MPI parallelization, limited OpenMP parallelization
  - CheSS: efficient MPI and OpenMP parallelization

# Parallel compilation

## Compile options:

```
## MPI wrappers to compiler
FC=mpif90

## Compile with OpenMP
FFLAGS= -fopenmp

## Flag activating MPI
FPPFLAGS= -DMPI

## Parallel linar algebra libraries
LIBS = <your_scalapack_lib> <your_blacs_lib>

## additional external libraries
LIBS += <your_chess_lib>

## additional preprocessor flags
FPPFLAGS += -DSIESTA__CHESS

MPI_INTERFACE=libmpi_f90.a
MPI_INCLUDE=.
```

# Parallel execution

## Execution on local workstation:

```
mpirun -n 8 siesta < example.fdf
```

## Execution on cluster

### Submit script (e.g. submit.sh)

```
#!/bin/bash
#BSUB -J Siesta
#BSUB -n 16
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
#BSUB -R "span[ptile=16]"
#BSUB -W 00:20
<load required modules>
mpirun siesta < example.fdf
```

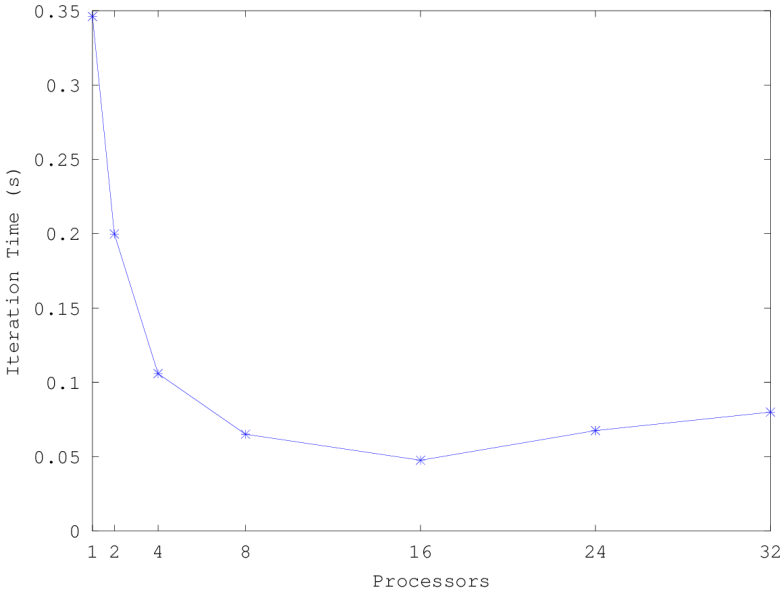
### Submission on cluster

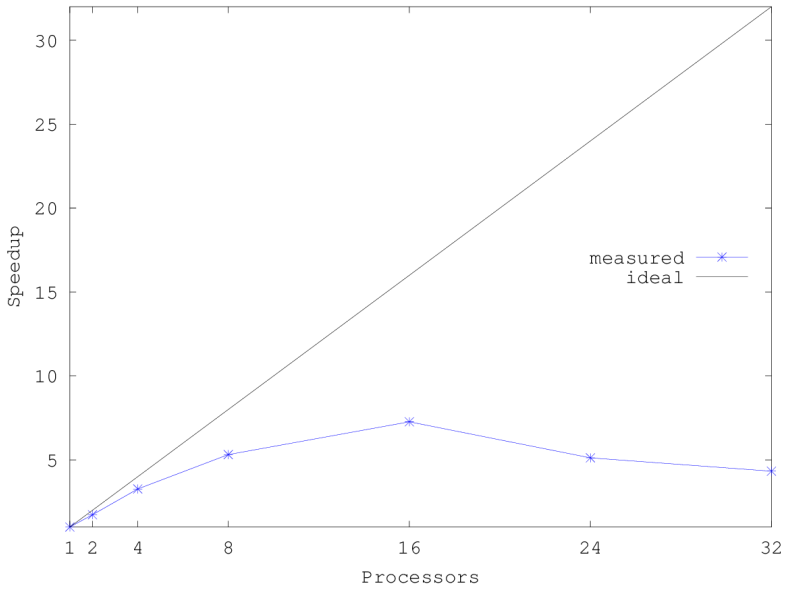
```
bsub < submit.sh
```

# Parallel scaling: Time and speedup

Using more cores does NOT always make a calculation faster (e.g. communication overhead)!

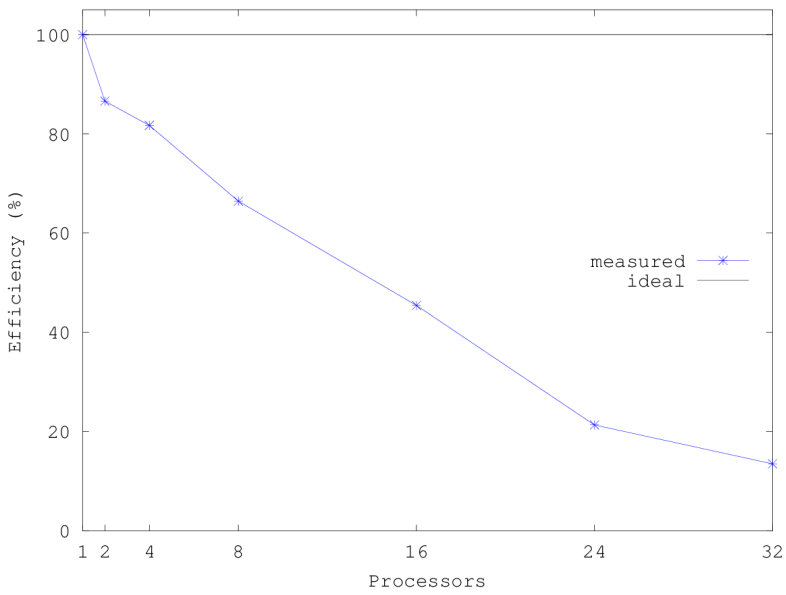
Careful estimation of the parallel resources is requested.

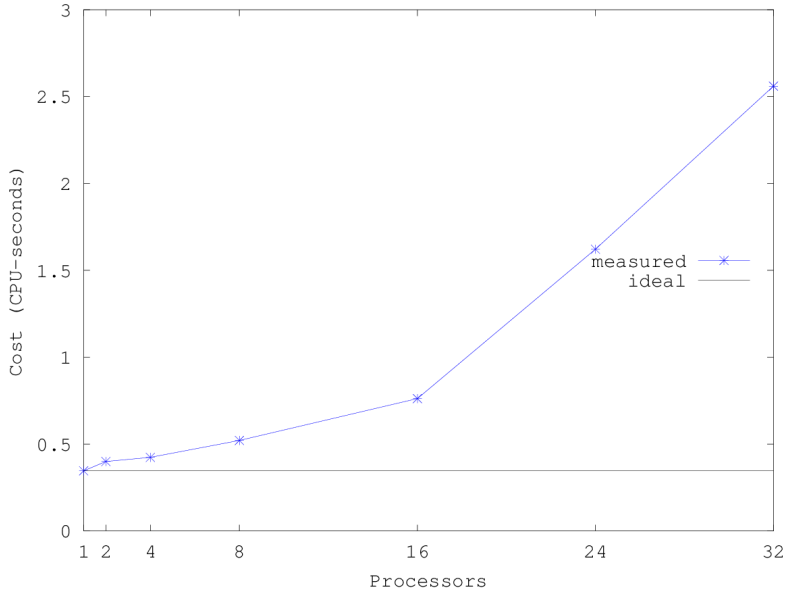


# Parallel scaling: Efficiency and cost

Choosing a bad parallelization scheme strongly affects the efficiency and cost:




To get a rough estimate of the parallel performance, see also

`http://departments.icmab.es/leem/siesta/siestimator/siestimator.php`

# Access to parallel computers

1. Clusters at universities, research institutes, companies, ...
2. National networks
   e.g. in Spain: RES `http://www.res.es/`
3. European networks
   e.g. PRACE: `http://www.prace-ri.eu`
   - "Preparatory access" for testing, benchmarks, etc.
   - "Project access for" production runs

RES and PRACE:

- Regular calls
- Significance of project and scalability of software
- Grants certain amount of CPU-hours (free of charge)

# Thank you for your attention!