Comparing the Efficiency of Iterative Eigenvalue Solvers: the Quantum ESPRESSO experience

Stefano de Gironcoli
Scuola Internazionale Superiore di Studi Avanzati
Trieste-Italy

Diagonalization of the Kohn-Sham hamiltonian is a major step in the scf solution of any electronic sctructure system

A number of methods to perform this task are currently used and new ones may be suggested that have advantages in terms of stability, scalability, memory efficiency, ...

- Davidson
- band-by-band Conjugate Gradient
- Projected Preconditioned CG
- Parallel Orbital-update
- ...

Efficiency of the solver depends on implementation details

it is not easy to develop new methods without good knowledge of the underlying code and its datastructure.

QUANTUMESPRESSO

HOME ::PROJECT ::WHAT CAN QE DO ::DOWNLOAD ::LEARN ::PSEUDO ::TOOLS ::
QE WIKI ::CONTACTS ::QUOTE ::LOGOS ::

25 May 2011 Version 4.3.1 of Quantum ESPRESSO is available for download.

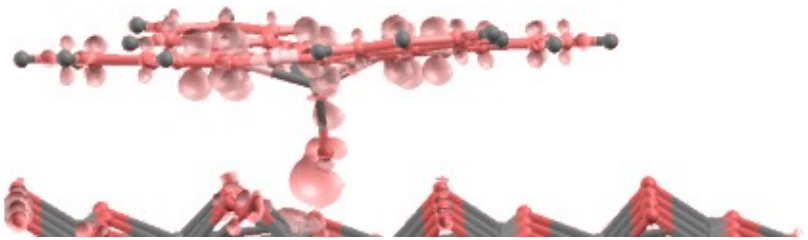
05 May 2011
The first GPU-enabled beta release of Quantum ESPRESSO is available for download.

01 April 2011
The new release, v.4.3, of the Quantum ESPRESSO distribution is available for download.

13 July 2010
Bugfix release v.4.2.1 of the Quantum ESPRESSO distribution is available for download.

10 May 2010
A new version, v.4.2, of the

Quantum ESPRESSO is an integrated suite of computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials (both norm-conserving and ultrasoft).



http://www.quantum-espresso.org/

In pw.x of Quantum ESPRESSO two methods are implemented:

- Davidson diagonalization
  - efficient in terms of number of Hpsi required
  - memory intensive: requires a work space up to
    (1+3**david*) * nbnd * npwx
    and diagonalization of matrices up to
    *david**nbnd x *david**nbnd
    where *david* is by default 4, but can be reduced to 2

- Conjugate Gradient
  - memory friendly: bands are dealt with one at a time.
  - the need to orthogonalize to lower states makes it intrinsically sequential and not efficient for large systems.

www.flapw.de

fleur

CASTEP

Abinit

siesta

ESLW_Drivers

10-21 July 2017 @ICTP

Volker Blum - ELSI

William Huhn – ELSI

Yann Pouillon – Abinit

Fabiano Corsetti – Siesta & Onetep

Anoop Chandran - QE

Ivan Carnimeo – QE

Layla Martin-Samos – QE

Viktor Yu - ELSI

David Lopez - Siesta

Micael Oliveira – Octopus & Abinit

Paolo Giannozzi – QE

Pietro Delugas - QE

Emine Kucukbenli - QE

Stefano de Gironcoli - QE

The two main iterative eigensolvers employed in the *pw.x* code of the *Quantum ESPRESSO* distribution were completely disentangled from the rest of the code. The solvers make use of the Linear Algebra domain-specific library LAXlib, developed within the MaX CoE, which is interfaced with ELPA and ScalaPack.

Solvers exploit MPI parallelization and in addition to basis-set component distribution, a parallelization over target states is possible, as well as a specific parallelization for the dense linear algebra.

Generic k-point as well as Gamma specific versions of the solvers are included. The Reverse Communication Interface (RCI) paradigm, allowing for a complete abstraction from the basis type and the interface used to perform the matrix-vector operations, has also been implemented for one of the solvers.

A toy code implementing the Cohen-Bergstresser empirical pseudopotential method is included to exemplify the use of the solvers and allow a test of their functionalities. It uses FFTXlib from MaX CoE.

The software developed during the Workshop is hosted by the e-cam gitlab server in Lausanne as a public sub-project of the ESL initiative (gitlab.e-cam2020/esl/ESLW_Drivers).

https://gitlab.e-cam2020.eu/esl/ESLW_Drivers

CB_toy_code/Doc	*Phys.Rev. **141**, 789 (1966)*
/examples	*contains inputs and ref. outputs*
/src	*contains simple code mains*
FFTXlib	*fft library used by CB_toy_code*
KS_Solvers/CG	*band-by-band CG*
/Davidson	*Davidson iterative diagonalization*
LAXlib	*linear algebra library (int w ELPA)*
UtilXlib	*basic utilities (error,timinig,para)*
archive	*library archive (lapack source)*
clib	*c timing routine*
include	
install	*configure, makedeps*
Makefile	
configure	

https://gitlab.e-cam2020.eu/esl/ESLW_Drivers

CB_toy_code/Doc	*Phys.Rev. **141**, 789 (1966)*
/examples	*contains inputs and ref. outputs*
/src	*contains simple code mains*
FFTXlib	*fft library used by CB_toy_code*
KS_Solvers/CG	*band-by-band CG*
/Davidson	*Davidson iterative diagonalization*
/Davidson_RCI	*Reverse Comm Interf version*
/ParO	*Parallel Orbital-updating*
/PPCG	*Projected Preconditioned CG*
LAXlib	*linear algebra library (int w ELPA)*
UtilXlib	*basic utilities (error,timinig,para)*
archive	*library archive (lapack source)*
clib	*c timing routine*
include	
install	*configure, makedeps*
Makefile	
configure	

Conjugate Gradient

- For each band, given a trial eigenpair: $\{|\phi_i^{(n)}\rangle, \varepsilon_i\}$

- Minimize the single particle energy

$$E(|\phi_i\rangle) = \langle\phi_i|H_{KS}|\phi_j\rangle$$

by (pre-conditioned) CG method

subject to the constraints

$$\langle\phi_i|S|\phi_j\rangle = \delta_{ij}, \quad \forall j \leq i$$

- Repeat for next band until completed

•Conjugate gradient

-memory friendly: bands are dealt with one at a time.

-the need to orthogonalize to lower states makes it intrinsically sequential and not efficient for large systems.

•routines

- rcgdiagg , ccgdiagg *real/cmplx CG diagonalization generalize*

- rotate_wfc_gamma, rotate_wfc_k *real/cmplx initial diag*

- h_1psi, s_1psi

  * preconditioning

Davidson Diagoalization

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$
- Eigenpairs of the reduced Hamiltonian

$$\tilde{H}_{ij} = \langle\phi_i^{(n)}|H_{KS}|\phi_j^{(n)}\rangle, \qquad \tilde{S}_{ij} = \langle\phi_i^{(n)}|S|\phi_j^{(n)}\rangle$$

- Build the correction vectors $|\tilde{\phi}_i^{(n)}\rangle$

$$|\tilde{\phi}_i^{(n)}\rangle = (H_{diag} - \varepsilon_i S_{diag})^{-1}(H_{KS} - \varepsilon_i S)|\phi_i^{(n)}\rangle$$

- Build an extended reduced Hamiltonian

$$\tilde{H}_{ij} = \langle\phi_i^{(n)}/\tilde{\phi}^{(n)}|H_{KS}|\phi_j^{(n)}/\tilde{\phi}_j^{(n)}\rangle, \quad \tilde{S}_{ij} = \langle\phi_i^{(n)}/\tilde{\phi}_i^{(n)}|S|\phi_j^{(n)}/\tilde{\phi}_j^{(n)}\rangle$$

- Diagonalize the <u>small</u> $2nbnd \times 2nbnd$ reduced Hamiltonian to get the new estimate for the eigenpairs

$$(\tilde{H} - \varepsilon\tilde{S})v = 0 \quad \longrightarrow \quad \{|\phi_i^{(n+1)}\rangle, \varepsilon_i^{(n+1)}\}$$

- Repeat if needed in order to improve the solution

$\rightarrow 3nbnd \times 3nbnd \rightarrow 4nbnd \times 4nbnd \ldots \rightarrow$ <u>$nbnd \times nbnd$</u>

•Davidson diagonalization

-efficient in terms of number of Hpsi required

-memory intensive: requires a work space up to

(1+3*david) * nbnd * npwx

and diagonalization of matrices up to

*david**nbnd x *david**nbnd

where *david* is by default 4, but can be reduced to 2

•routines

- regterg , cegterg *real/cmplx eigen iterative generalized*

- h_psi, s_psi, g_psi

- rdiaghg, cdiaghg *real/cmplx diagonalization H generalized*

PPCG – Projected Preconditioned Conjugate Gradient

E. Vecharynski, C. Yang, J.E. Pask, *J. Comp.Phys.* ***290***,73 (2015)

Algorithm 2: The projected preconditioned conjugate gradient (PPCG) algorithm.

Input: The matrix A, a preconditioner T, and a starting guess of the invariant subspace $X^{(0)} \in \mathbb{C}^{n\times k}$ associated with the k smallest eigenvalues of A;

Output: An approximate invariant subspace $X \in \mathbb{C}^{n\times k}$ associated with the k smallest eigenvalues of A;

1: $X \leftarrow \mathtt{orth}(X^{(0)})$; $P \leftarrow [\ ]$;
2: **while** convergence not reached **do**
3: $W \leftarrow T(AX - X(X^*AX))$;
4: $W \leftarrow (I - XX^*)W$;
5: $P \leftarrow (I - XX^*)P$;
6: **for** $j = 1, \ldots, k$ **do**
7: $S \leftarrow [x_j, w_j, p_j]$;
8: Find the smallest eigenpair $(\theta_{\min}, c_{\min})$ of $S^*ASc = \theta S^*Sc$, where $c^*S^*Sc = 1$;
9: $\alpha_j \leftarrow c_{\min}(1)$, $\beta_j \leftarrow c_{\min}(2)$; and $\gamma_j \leftarrow c_{\min}(3)$ ($\gamma_j = 0$ at the initial step);
10: $p_j \leftarrow \beta_j w_j + \gamma_j p_j$;
11: $x_j \leftarrow \alpha_j x_j + p_j$.
12: **end for**
13: $X \leftarrow \mathtt{orth}(X)$;
14: If needed, perform the Rayleigh–Ritz procedure within span(X);
15: **end while**

each band (or small group of bands) is updated by diagonalizing a small 3*blksize x 3*blksize matrix built from the current X, the orthogonal residual and the orthogonal conjugate direction

• PPCG work in progress

•-memory friendly: bands are dealt with a small block at a time.
•-global calls to h_psi give opportunity for band parallelization
(not working properly yet)
•-each block can be dealt with independently (parallelization)
•-most operations on arrays use efficient BLAS3 calls (DGEMM)

•routines

- ppcg, *real PPCG, cmplx version presently not available*

- rotate_wfc_gamma, *real initial diag (the same as CG)*

- h_psi, (s_psi) *generalized algorithm not available yet*

* preconditioning

Some recent work on an alternative iterative methods

A PARALLEL ORBITAL-UPDATING APPROACH FOR ELECTRONIC STRUCTURE CALCULATIONS *

XIAOYING DAI†, XINGAO GONG‡, AIHUI ZHOU† , AND JINWEI ZHU†

Abstract. In this paper, we propose an orbital iteration based parallel approach for electronic structure calculations. This approach is based on our understanding of the single-particle equations of independent particles that move in an effective potential. With this new approach, the solution of the single-particle equation is reduced to some solutions of independent linear algebraic systems and a small scale algebraic problem. It is demonstrated by our numerical experiments that this new approach is quite efficient for full-potential calculations for a class of molecular systems.

arXiv:1405.0260v2 [math.NA] 20/11/2014

A PARALLEL ORBITAL-UPDATING BASED OPTIMIZATION METHOD FOR ELECTRONIC STRUCTURE CALCULATIONS *

XIAOYING DAI†, ZHUANG LIU‡, XIN ZHANG§, AND AIHUI ZHOU¶

Abstract. In this paper, we propose a parallel optimization method for electronic structure calculations based on a single orbital-updating approximation. It is shown by our numerical experiments that the method is efficient and reliable for atomic and molecular systems of large scale over supercomputers.

arXiv:1510.07230v1 [math.NA] 25/10/2015

ParO : Parallel Orbital-updating method in a nutshell

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$
- Solve <u>in parallel</u> the *nbnd* linear systems

$$(H_{KS} + \lambda S)|\tilde{\phi}_i^{(n)}\rangle = (\varepsilon_i^{(n)} + \lambda) S|\phi_i^{(n)}\rangle$$

- Build the reduced Hamiltonian

$$\tilde{H}_{ij} = \langle\tilde{\phi}_i^{(n)}|H_{KS}|\tilde{\phi}_j^{(n)}\rangle, \quad \tilde{S}_{ij} = \langle\tilde{\phi}_i^{(n)}|S|\tilde{\phi}_j^{(n)}\rangle$$

- Diagonalize the <u>small</u> *nbnd* x *nbnd* reduced Hamiltonian to get the new estimate for the eigenpairs

$$(\tilde{H} - \varepsilon\tilde{S})v = 0 \quad \longrightarrow \quad \{|\phi_i^{(n+1)}\rangle, \varepsilon_i^{(n+1)}\}$$

- Repeat if needed in order to improve solution at fixed Hamiltonian

Y Pan, XY Dai, XG Gong, S de Gironcoli, GM Rignanese, and AH Zhou,
J. Comp. Phys. ***348****, 482-492 (2017)*

A variant of ParO method

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$
- Solve <u>in parallel</u> the *nbnd* linear systems

$$(H_{KS} + \lambda S)|\tilde{\phi}_i^{(n)}\rangle = (\varepsilon_i^{(n)} + \lambda) S|\phi_i^{(n)}\rangle$$

- Build the reduced Hamiltonian from both $|\tilde{\phi}_i^{(n)}\rangle$ & $|\phi_i^{(n)}\rangle$

$$\tilde{H}_{ij} = \langle \tilde{\phi}_i^{(n)} / \phi_i^{(n)} | H_{KS} | \tilde{\phi}_j^{(n)} / \phi_j^{(n)} \rangle, \quad \tilde{S}_{ij} = \langle \tilde{\phi}_i^{(n)} / \phi_i^{(n)} | S | \tilde{\phi}_j^{(n)} / \phi_j^{(n)} \rangle$$

- Diagonalize the <u>small</u> 2*nbnd* x 2*nbnd* reduced Hamiltonian to get the new estimate for the eigenpairs

$$(\tilde{H} - \varepsilon \tilde{S}) v = 0 \quad \longrightarrow \quad \{|\phi_i^{(n+1)}\rangle, \varepsilon_i^{(n+1)}\}$$

- Repeat if needed in order to improve solution at fixed Hamiltonian

Y Pan, XY Dai, XG Gong, S de Gironcoli, GM Rignanese, and AH Zhou,
J. Comp. Phys. ***348****, 482-492 (2017)*

A variant of ParO method (2)

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$
- Solve <u>in parallel</u> the *nbnd* linear systems

$$\left(H_{KS} - \varepsilon_i^{(n)} S + \alpha S|\phi_i^{(n)}\rangle\langle\phi_i^{(n)}|S\right)|\tilde{\phi}_i^{(n)}\rangle = -(H_{KS} - \varepsilon_i^{(n)} S)|\phi_i^{(n)}\rangle$$

- Build the reduced Hamiltonian from both $|\tilde{\phi}_i^{(n)}\rangle$ & $|\phi_i^{(n)}\rangle$

$$\tilde{H}_{ij} = \langle\tilde{\phi}_i^{(n)}/\phi_i^{(n)}|H_{KS}|\tilde{\phi}_j^{(n)}/\phi_j^{(n)}\rangle, \quad \tilde{S}_{ij} = \langle\tilde{\phi}_i^{(n)}/\phi_i^{(n)}|S|\tilde{\phi}_j^{(n)}/\phi_j^{(n)}\rangle$$

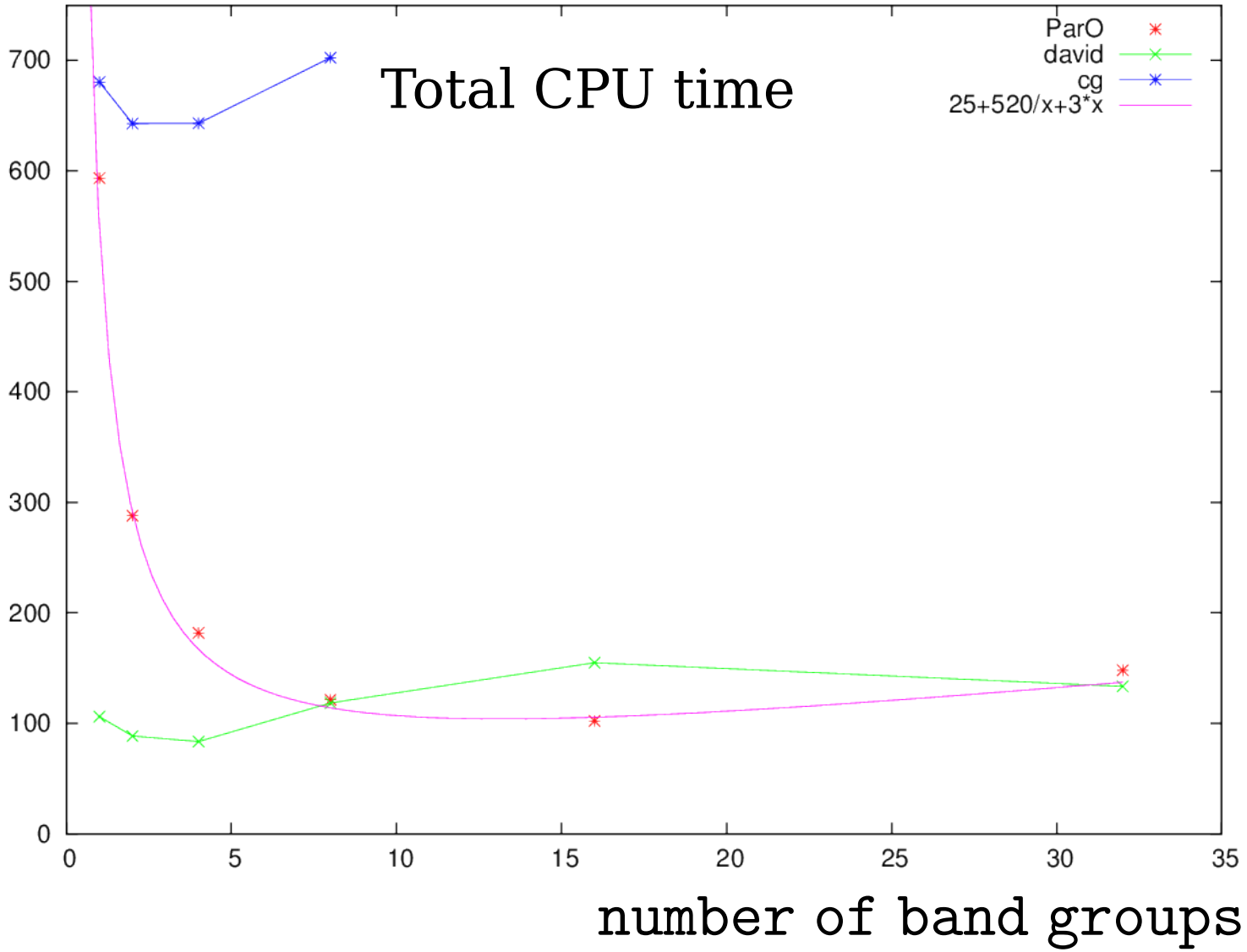
- Diagonalize the <u>small</u> 2*nbnd* x 2*nbnd* reduced Hamiltonian to get the new estimate for the eigenpairs

$$(\tilde{H} - \varepsilon\tilde{S})v = 0 \quad \longrightarrow \quad \{|\phi_i^{(n+1)}\rangle, \varepsilon_i^{(n+1)}\}$$

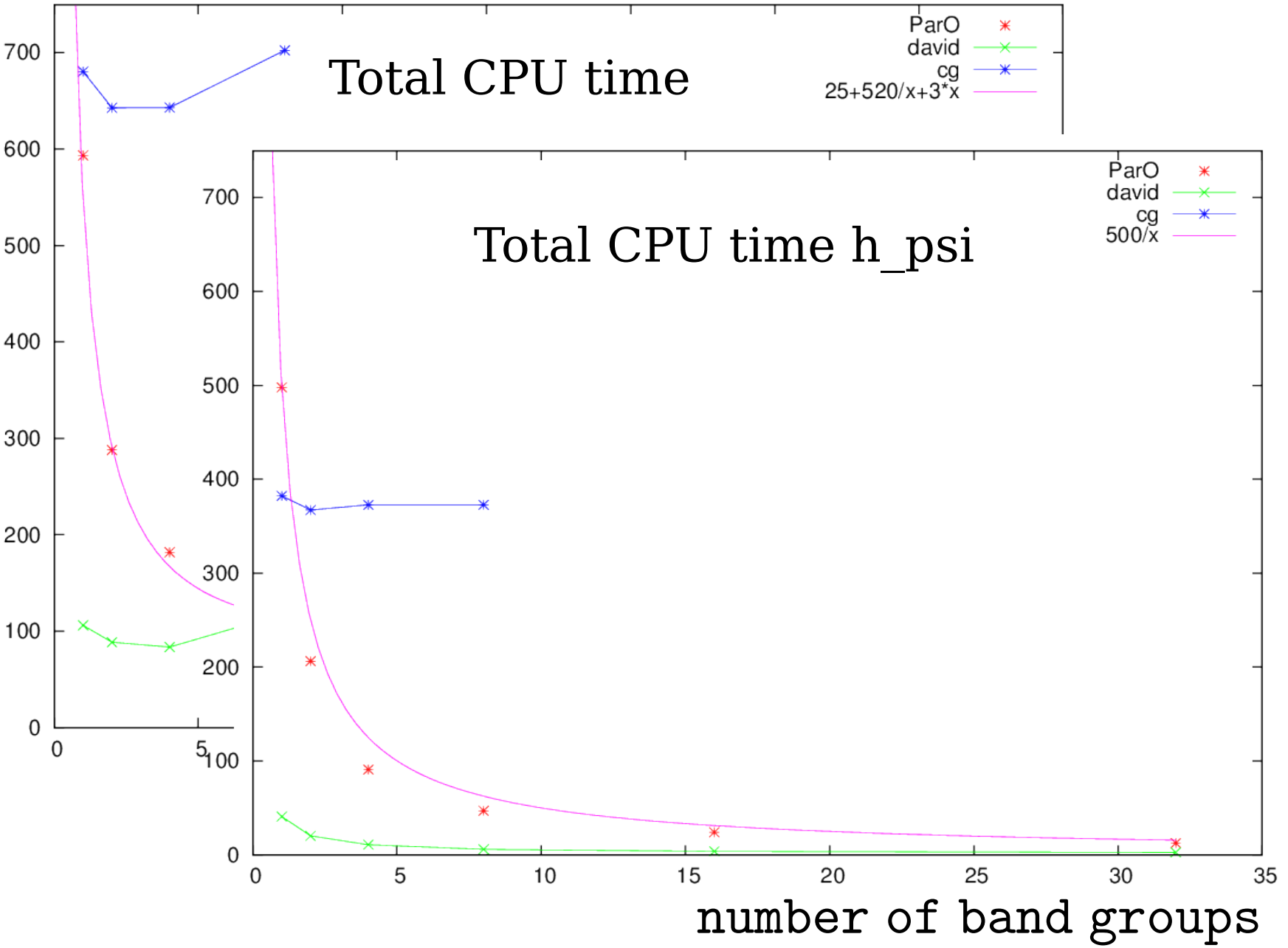
- Repeat if needed in order to improve solution at fixed Hamiltonian

Y Pan, XY Dai, XG Gong, S de Gironcoli, GM Rignanese, and AH Zhou,
J. Comp. Phys. ***348****, 482-492 (2017)*

216 Si atoms in a SC cell : Timing



216 Si atoms in a SC cell : Timing



Many Factors contribute to Resulting Efficiency

- Domain decomposition parallelization
- Basis set components are distributed
- Memory is distributed

- Band group parallelization
- Operations on dynamically defined band groups are distributed
- Memory is NOT distributed

- Parallel dense diagonalization
- A dedicated communicator is present (interface with ScaLapack and ELPA)

Many Options to explore

- OpenMP/MPI parallelization
  Use OpenMP inside a node MPI across nodes ?
  For given resource allocation which distribution is best ?

- CPU/GPU hybrid
  How to maintain source code unity ?
  To what extent is this possible/desirable ?

CUDA Fortran is basically Fortran

```
subroutine update(a,n)
    real:: a(n)
#ifdef USE_GPU
    attributes(device) :: a
#endif
    ...
!$cuf kernel do <<<*,*>>>
    do i=1, n
        a(i) = a(i) + b
    enddo
    ...
end subroutine update
```

It is possible, *with some limited effort,* to integrate GPU-aware sections in a single source. Similarly to MPI/OpenMP cases. Encapsulation/modularization of the more architecture-specific bits will help readability and maintainability.

Adding GPUs: a range of different machines

Ulysses @ SISSA	16 nodes: 20 cores - 2 Gpus
Drake @ CNR	1 nodes: 16 cores - 4 Gpus (k80)
DAVIDE @ CINECA	45 nodes: 16 cores - 4 Gpus (p100)

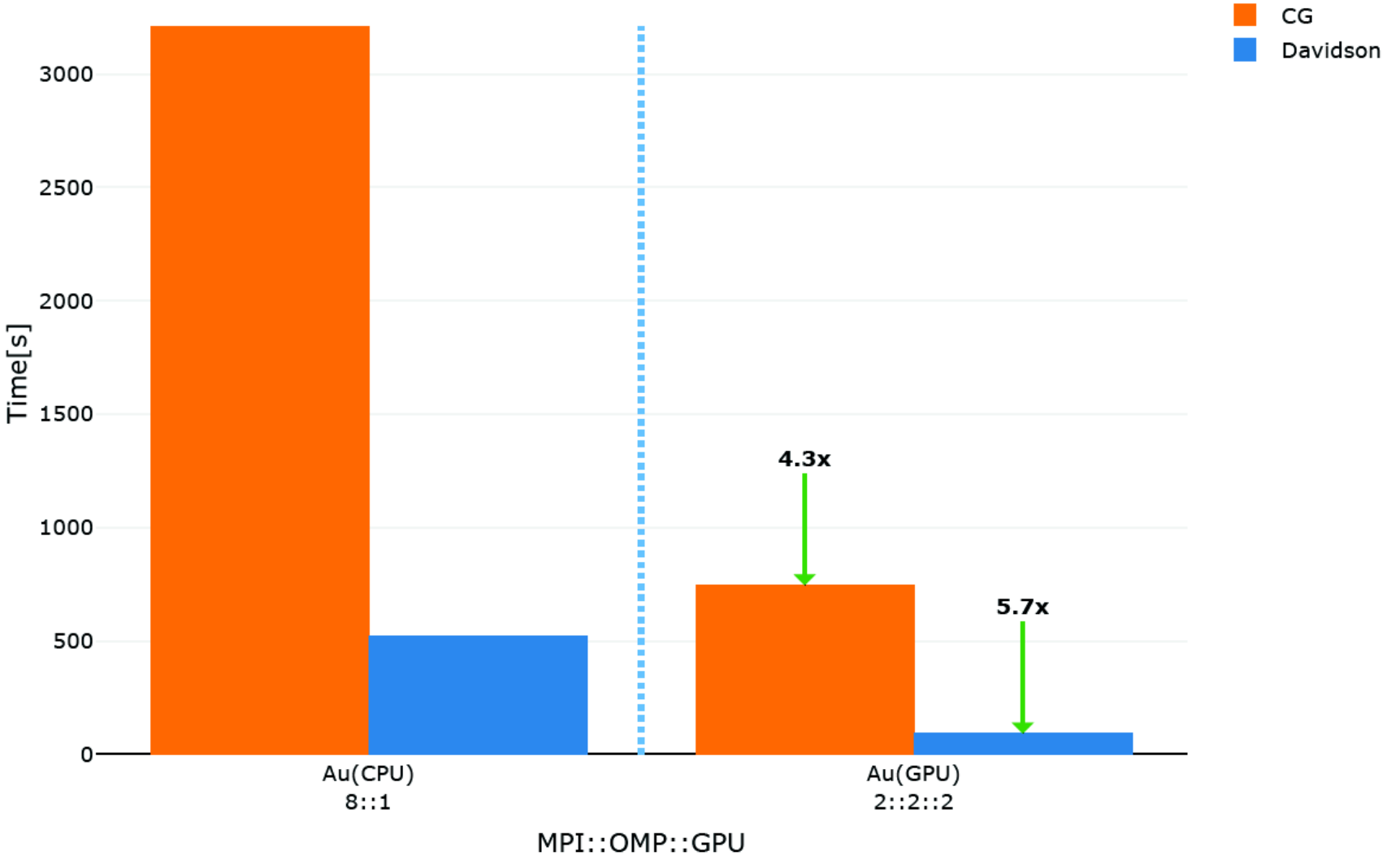
comparison depends on the selected architecture.

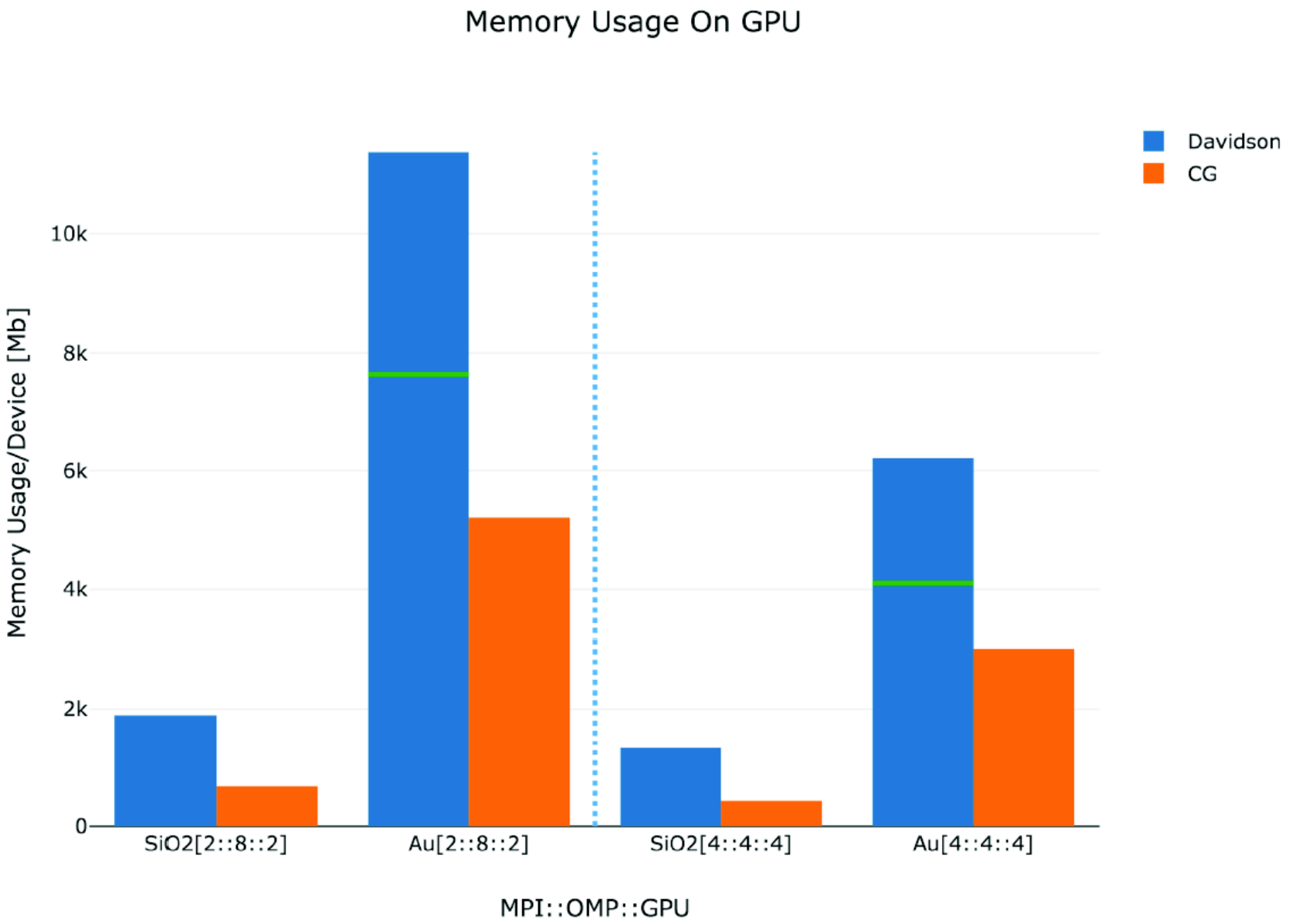
a reliable performance modeling would be very useful to make rational choices when buying hardware for and allocating resources to a user community.

so far the focus of the effort has been more on enabling the use of the new architecture rather than optimizing performance.

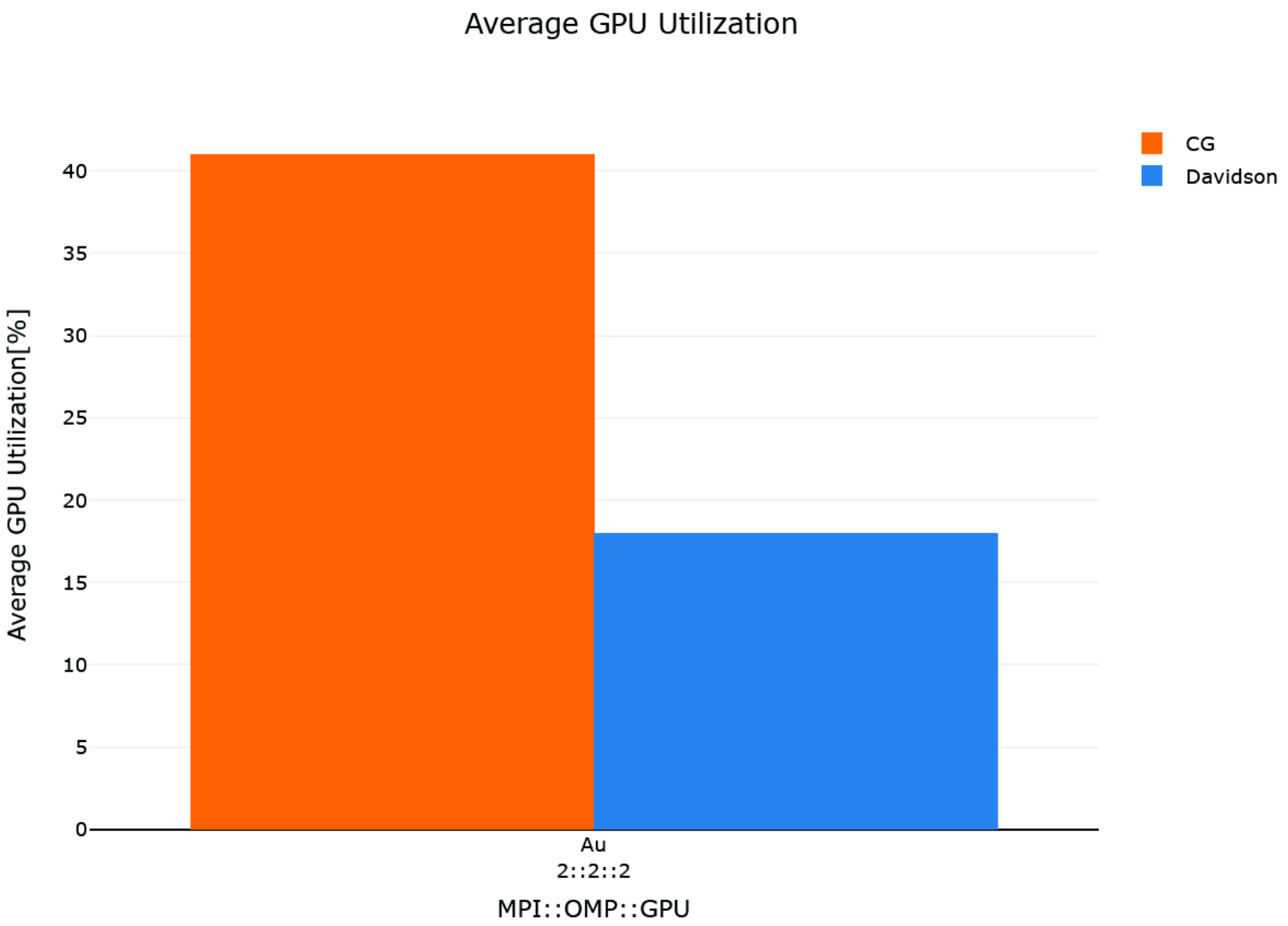
-Davidson/CG solvers

Performance On **CPU vs GPU** [P100]

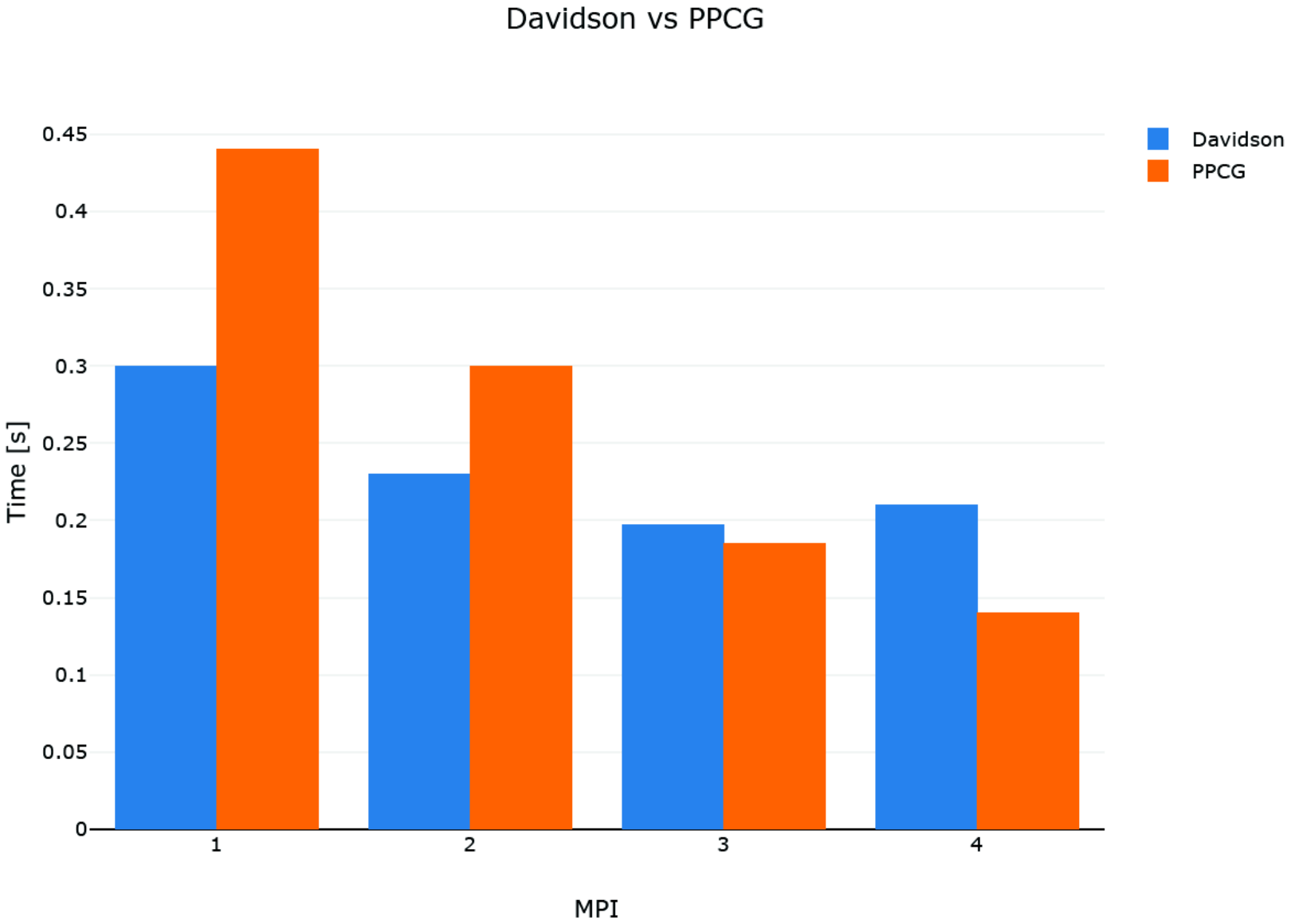




#MPI should be = #GPU => OMP parallelism on CPU is important as core/gpu ratio may be significant



CG uses devices more efficiently
Time-to-solution favours Davidson

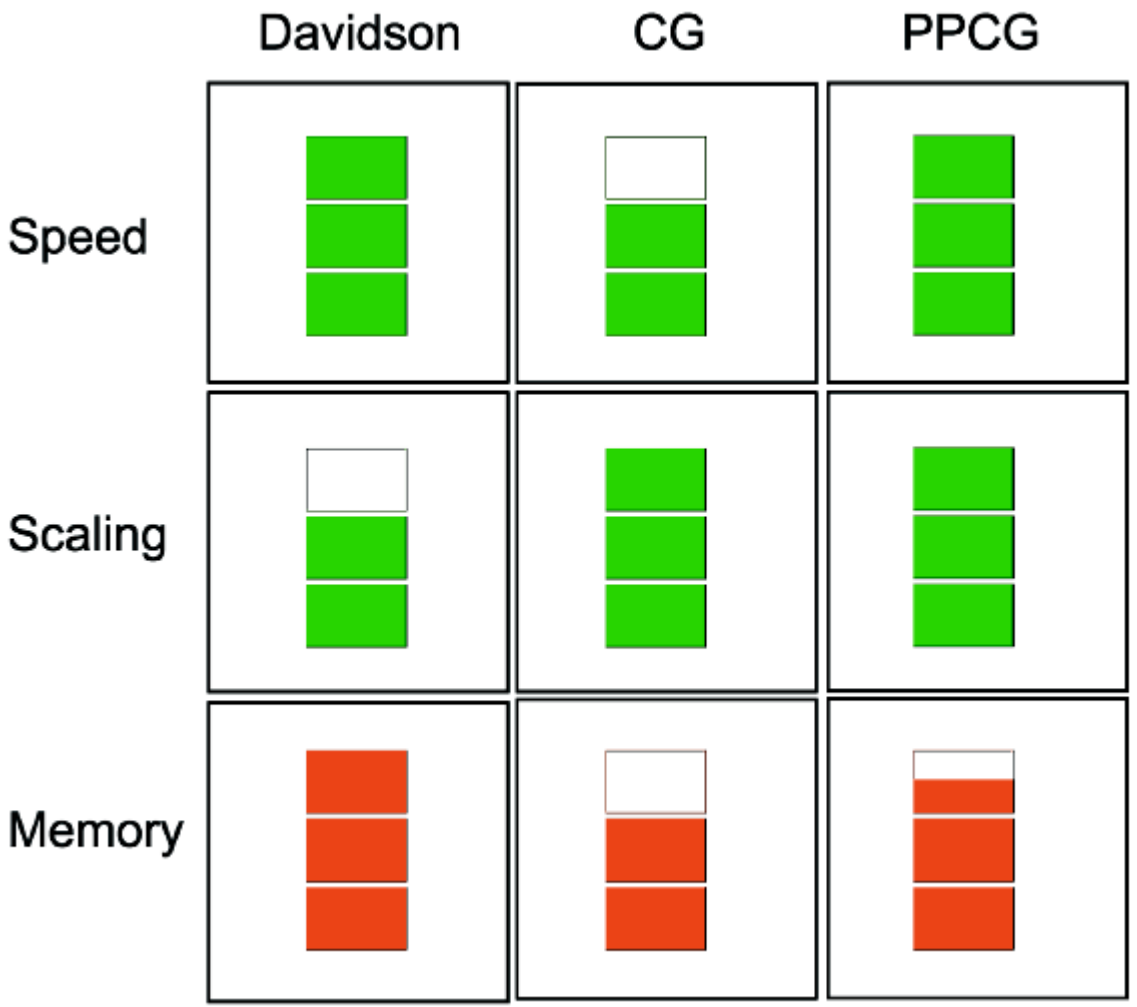

Davidson vs PPCG
Davidson
PPCG
Time [s]
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
MPI

Davidson Diagonalization with Scalapack

MPI x omp	1 node	2 nodes	4 nodes	8 nodes
36 x 1	922.02	494.34	267.29	308.76
18 x 2	**907.99**	**381.05**	217.41	176.18
12 x 3	949.99	421.89	**213.70**	165.17
9 x 4	969.37	446.02	253.20	167.27
6 x 6	951.11	431.43	233.96	**155.91**
4 x 9	1037.23	465.58	328.12	176.94
3 x 12	1337.31	633.09	359.37	236.59
2 x 18	1357.39	603.59	395.83	244.41
1 x 36	2214.57	1276.03	633.39	398.98

PPCG Diagonalization with Scalapack

MPI x omp	1 node	2 nodes	4 nodes	8 nodes
36 x 1	1696.00	871.79	558.25	836.72
18 x 2	**1559.21**	**708.68**	414.20	383.61
12 x 3	1847.92	793.36	416.11	301.33
9 x 4	1899.39	853.77	446.09	298.12
6 x 6	1876.67	812.19	**391.78**	**262.13**
4 x 9	1985.95	824.50	442.81	268.55
3 x 12	2363.67	1166.58	646.18	349.64
2 x 18	2652.92	1125.26	657.53	354.51
1 x 36	3972.33	2443.66	1163.25	621.99


Davidson
CG
PPCG
Speed
Scaling
Memory

Conclusions

- Being code agnostic: ESLW_Drivers is a playground that may be useful to experiment in an (almost) realistic system without the need to be fully embedded in a given code.

- Hybrid CPU/GPU: source code unity is an issue.
  No easy solution. Confine, template, encapsulate...

- OpenMP/MPI: OpenMP used to be very bad.
  It still is but is improving and may be usefull in the hybrid CPU/GPU case to reach better scalability