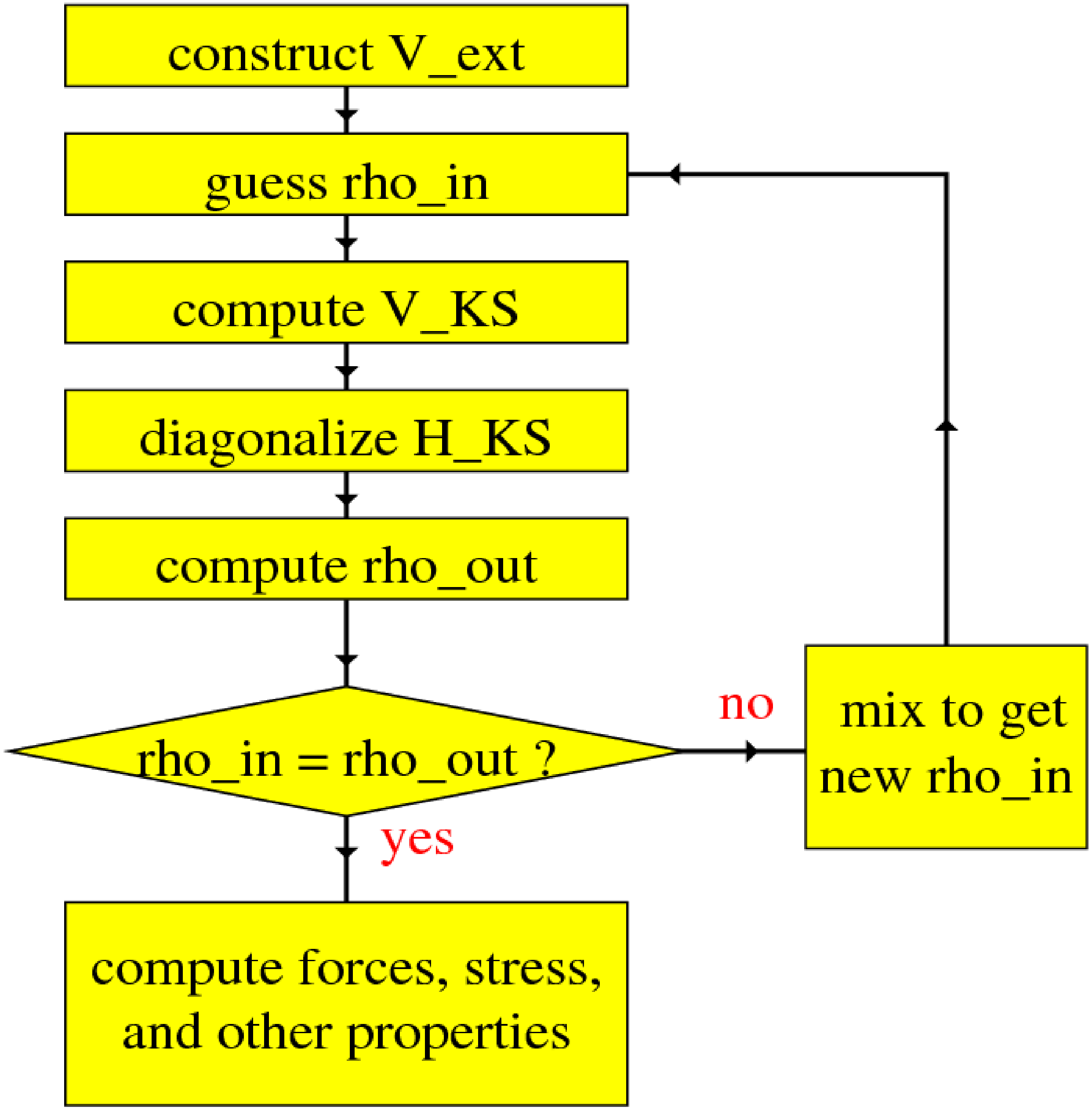
Electronic Structure: from BlackBoard to Source Code

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

PWSCF

and

diagonalization



construct V_ext

guess rho_in

compute V_KS

diagonalize H_KS

compute rho_out

rho_in = rho_out ?

no

mix to get new rho_in

yes

compute forces, stress, and other properties

ELECTRONS

```
call electron_scf
     do iter = 1, niter
        call c_bands    -->   C_BANDS
        call sum_band   -->   SUM_BAND
        call mix_rho
        call v_of_rho
     end do iter
```

PWSCF

```
call read_input_file   (input.f90)

call run_pwscf

     call setup         --> SETUP
     call init_run      --> INIT_RUN
     do
        call electrons  --> ELECTRONS
        call forces
        call stress
        call move_ions
        call update_pot
        call hinit1
     end do
```

SETUP

defines grid and other dimensions, no system specific calculations yet

INIT_RUN

```
call pre_init
call allocate_fft
call ggen
call allocate_nlpot
call allocate_paw_integrals
call paw_one_center
call allocate_locpot
call allocate wfc
call openfile
call hinit0
call potinit
call newd
call wfctinit
```

ELECTRONS

```
call electron_scf
     do iter = 1, niter
        call c_bands      -->   C_BANDS
        call sum_band     -->   SUM_BAND
        call mix_rho
        call v_of_rho
     end do iter
```

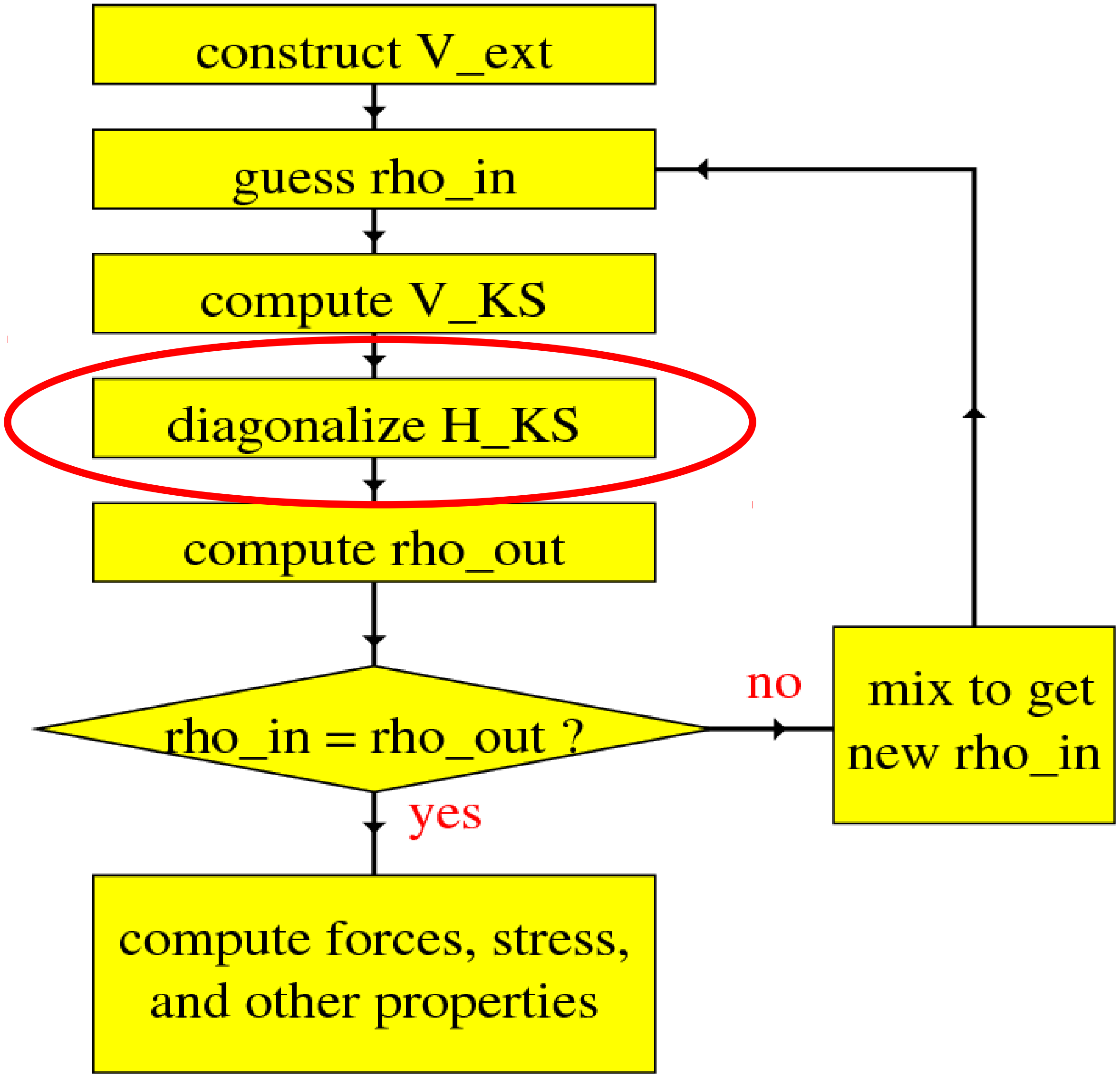
C_BANDS

```
do ik = 1, nks
   call get_buffer       (evc)
   call init_us_2        (vkb)
   call diag_bands      -->   DIAG_BANDS
   call save_buffer
end do ik
```

DIAG_BANDS

```
DAVIDSON (isolve=0)
   hdiag = g2 + vloc_avg + Vnl_avg
   call cegterg or pcegterg

CG (isolve=1)
   hdiag = 1 + g2 + sqrt(1+(g2-1)**2)
   call rotate_wfc
   call ccgdiagg
```

Step 4 : diagonalization



construct V_ext

guess rho_in

compute V_KS

diagonalize H_KS

compute rho_out

rho_in = rho_out ?

no

mix to get new rho_in

yes

compute forces, stress, and other properties

Diagonalization of H_{KS} is a major step in the scf solution of any system.

In pw.x 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.

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> 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 the solution

→ *3nbnd x* 3*nbnd* → *4nbnd x* 4*nbnd …* → <u>*nbnd x 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*

Conjugate Gradient

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

- Minimize the single particle energy

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

by (pre-conditioned) CG method

subject to the constraints

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

.... see attached documents for more details

- 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*

  - h_1psi, s_1psi

    * preconditioning

Making optimized codes available to the community and exploit novel architectures:

the QE experience

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

ESLW_Drivers

10-21 July 2017

Electronic Structure Library Workshop:
a cecam initiative predating e-cam

Volker Blum   - ELSI

Viktor Yu - ELSI

William Huhn - ELSI

David Lopez - Siesta

Yann Pouillon - Abinit

Micael Oliveira – Octopus & Abinit

Fabiano Corsetti – Siesta & Onetep

Paolo Giannozzi – QE

Anoop Chandran - QE

Pietro Delugas - QE

Ivan Carnimeo - QE

Emine Kucukbenli - QE

Layla Martin-Samos - QE

Stefano de Gironcoli - QE

Diagonalization of H_{KS} is a major step in the scf solution of any system.

In pw.x in QE 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.

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	*so far empty*
/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*
/PPCG	*PPCG 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	

- 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*

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

  - h_psi, s_psi, g_psi *code specific*

- 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*

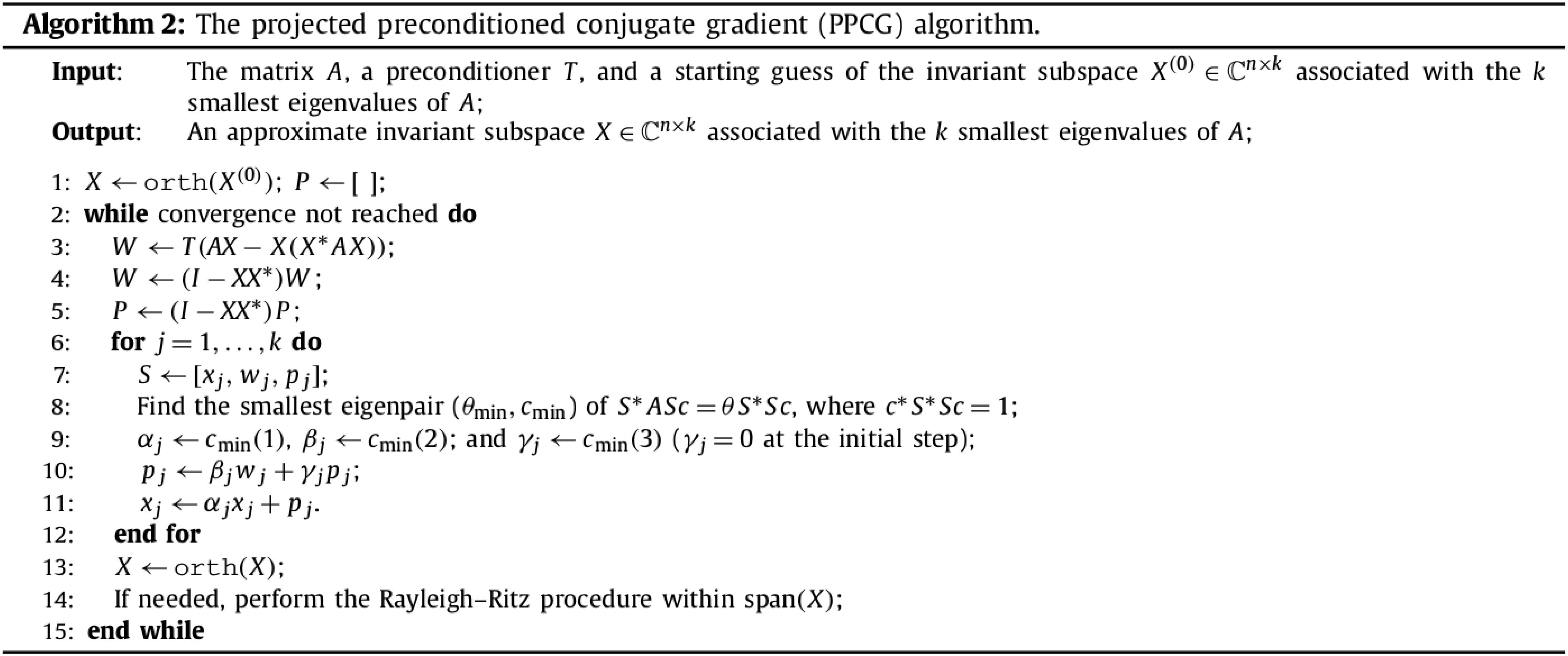
  - rotate_wfc_gamma, rotate_wfc_k *real/cmplx initial diag*

  - h_1psi, s_1psi *code specific*

    * preconditioning

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;

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

- -memory friendly: bands are dealt with a small block at a time.
- -global calls to h_psi give opportunities for band parallelization.
- -each block can be dealt with independently (parallelization)
- -most operations on arrays use efficient BLAS3 calls (DGEMM)

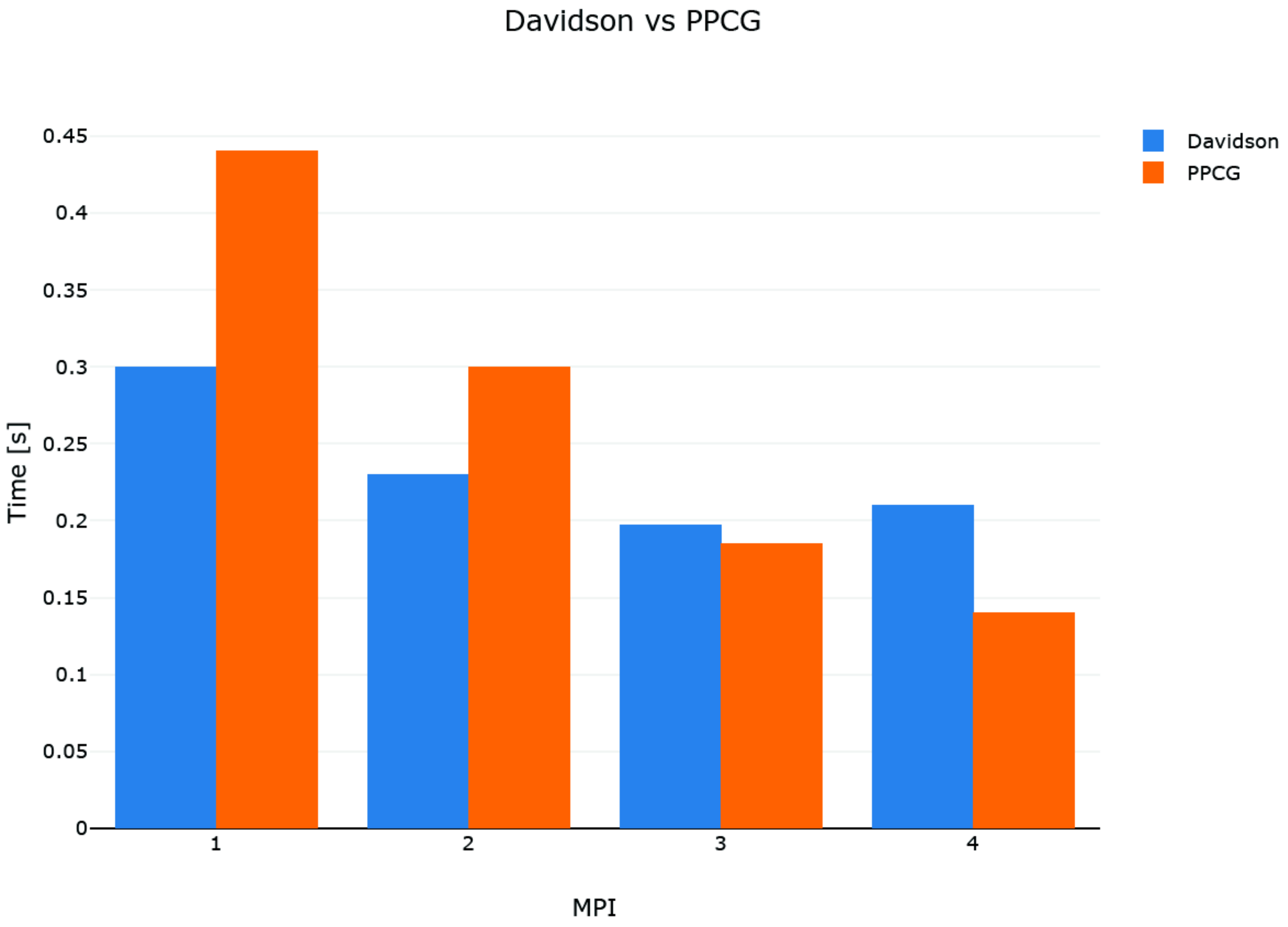
- routines

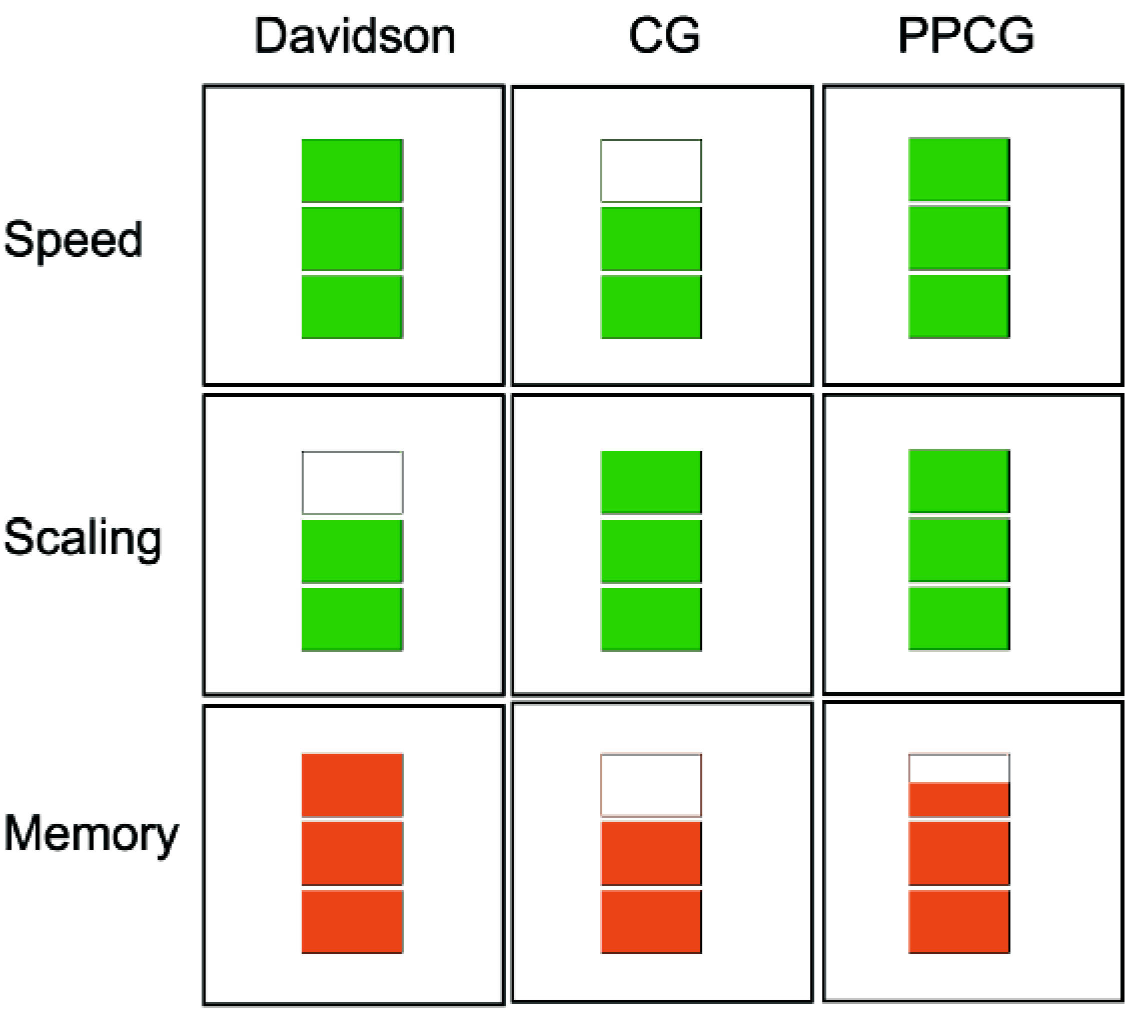
 - ppcg,   *real PPCG,* *cmplx version not yet available*

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

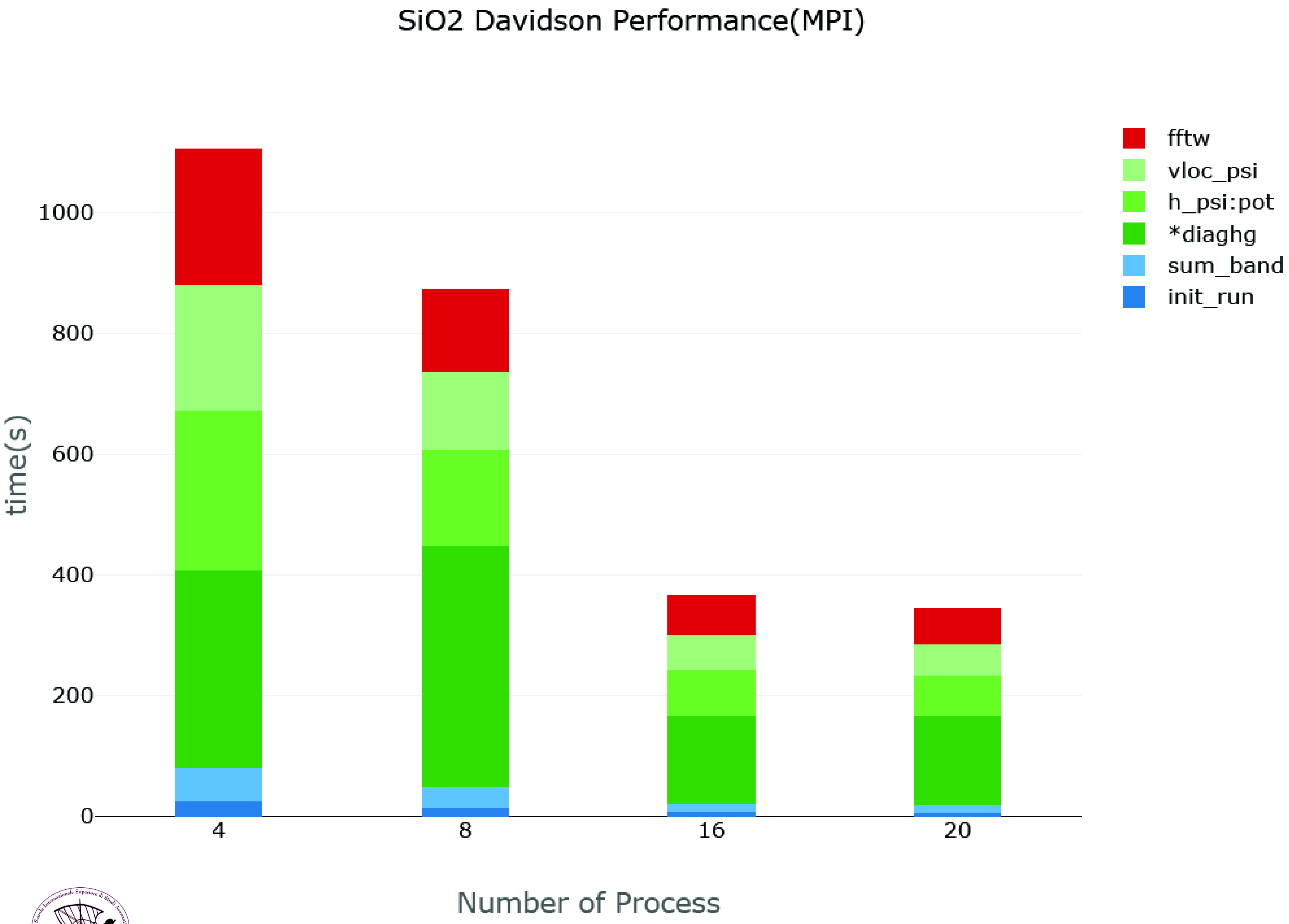
 - h_psi, s_psi   *code specific*

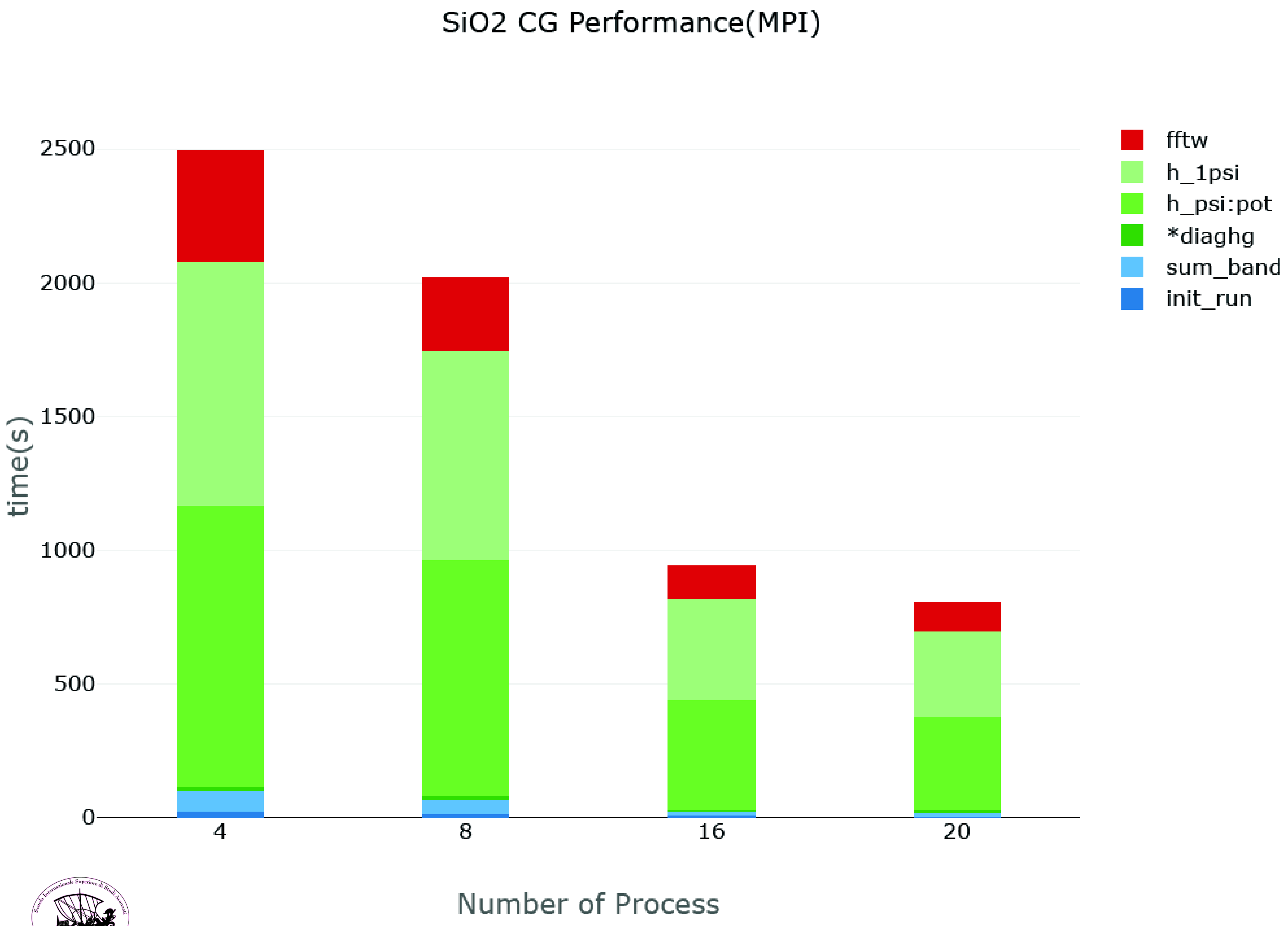
   * preconditioning


Davidson vs 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
PPCG



	Davidson	CG	PPCG
Speed	🟩🟩🟩	⬜🟩🟩	🟩🟩🟩
Scaling	⬜🟩🟩	🟩🟩🟩	🟩🟩🟩
Memory	🟥🟥🟥	⬜🟥🟥	⬒🟥🟥


SiO2 Davidson Performance(MPI)
time(s)
1000
800
600
400
200
0
4
8
16
20
Number of Process
fftw
vloc_psi
h_psi:pot
*diaghg
sum_band
init_run


SiO2 CG Performance(MPI)
time(s)
0
500
1000
1500
2000
2500
4
8
16
20
Number of Process
fftw
h_1psi
h_psi:pot
*diaghg
sum_band
init_run

Parallel Orbital update method

and

some thoughts about

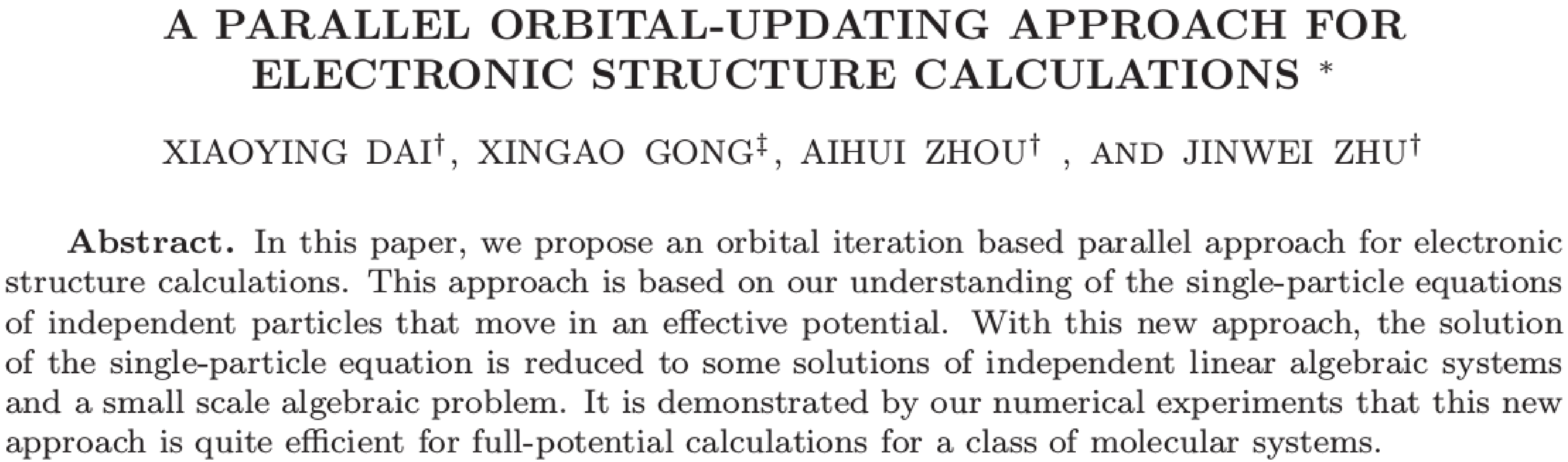
-bgrp parallelization

-ortho parallelization

-task parallelization

in pw.x

Some recent work on an alternative iterative methods

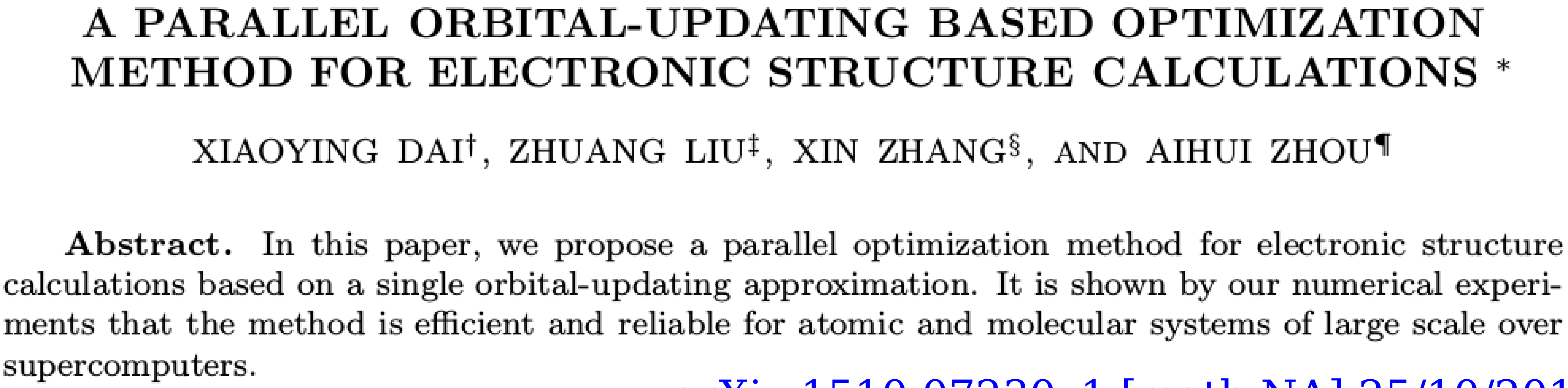


A PARALLEL ORBITAL-UPDATING APPROACH FOR ELECTRONIC STRUCTURE CALCULATIONS *

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



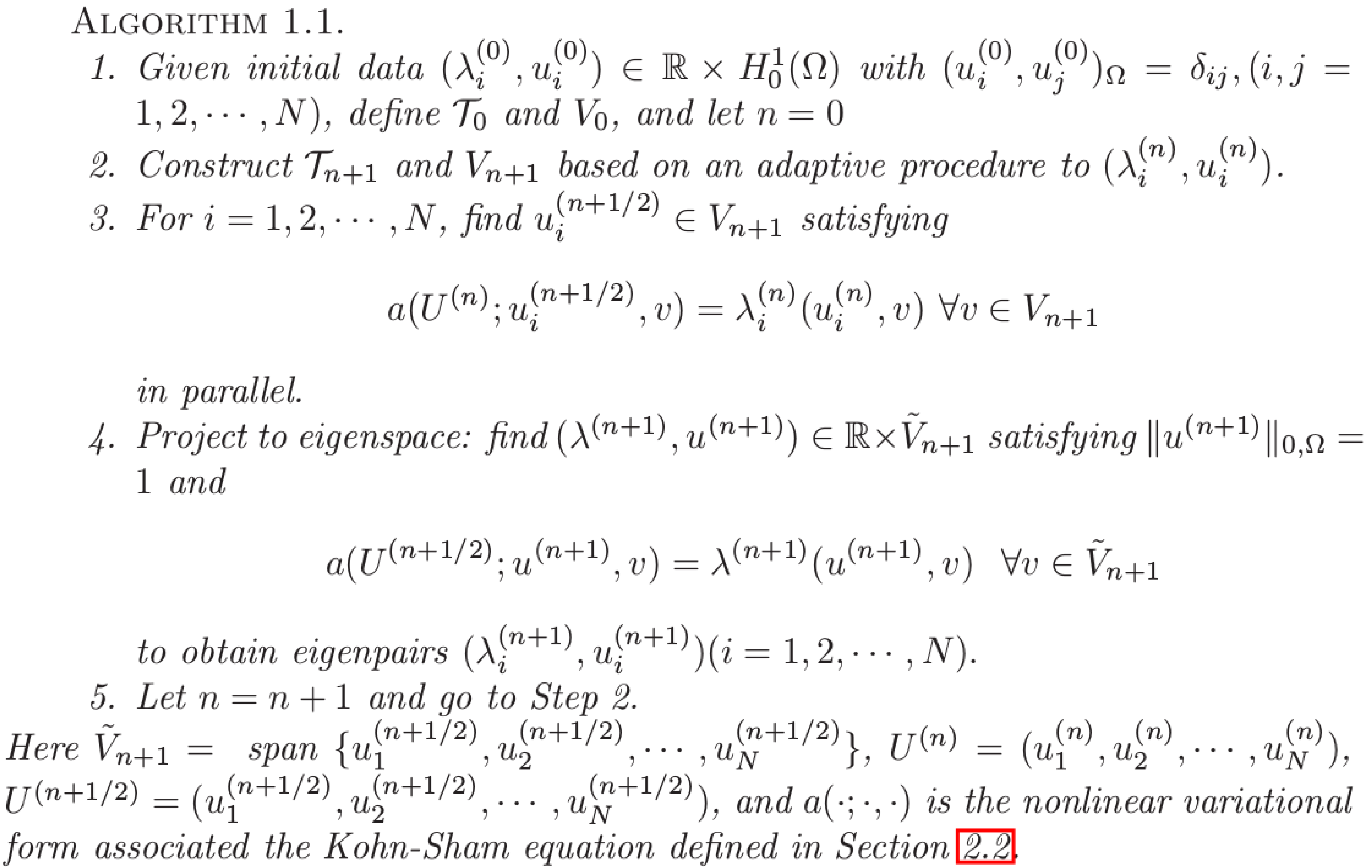
A PARALLEL ORBITAL-UPDATING BASED OPTIMIZATION METHOD FOR ELECTRONIC STRUCTURE CALCULATIONS *

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 in a nutshell



Algorithm 1.1.

1. *Given initial data* $(\lambda_i^{(0)}, u_i^{(0)}) \in \mathbb{R} \times H_0^1(\Omega)$ *with* $(u_i^{(0)}, u_j^{(0)})_\Omega = \delta_{ij}, (i,j = 1, 2, \cdots, N)$, *define* $\mathcal{T}_0$ *and* V_0, *and let* $n = 0$
2. *Construct* $\mathcal{T}_{n+1}$ *and* V_{n+1} *based on an adaptive procedure to* $(\lambda_i^{(n)}, u_i^{(n)})$.
3. *For* $i = 1, 2, \cdots, N$, *find* $u_i^{(n+1/2)} \in V_{n+1}$ *satisfying*

$$a(U^{(n)}; u_i^{(n+1/2)}, v) = \lambda_i^{(n)}(u_i^{(n)}, v) \ \forall v \in V_{n+1}$$

in parallel.

4. *Project to eigenspace: find* $(\lambda^{(n+1)}, u^{(n+1)}) \in \mathbb{R} \times \tilde{V}_{n+1}$ *satisfying* $\|u^{(n+1)}\|_{0,\Omega} = 1$ *and*

$$a(U^{(n+1/2)}; u^{(n+1)}, v) = \lambda^{(n+1)}(u^{(n+1)}, v) \quad \forall v \in \tilde{V}_{n+1}$$

to obtain eigenpairs $(\lambda_i^{(n+1)}, u_i^{(n+1)})(i = 1, 2, \cdots, N)$.

5. *Let* $n = n + 1$ *and go to Step 2.*

Here $\tilde{V}_{n+1} =$ *span* $\{u_1^{(n+1/2)}, u_2^{(n+1/2)}, \cdots, u_N^{(n+1/2)}\}$, $U^{(n)} = (u_1^{(n)}, u_2^{(n)}, \cdots, u_N^{(n)})$, $U^{(n+1/2)} = (u_1^{(n+1/2)}, u_2^{(n+1/2)}, \cdots, u_N^{(n+1/2)})$, *and* $a(\cdot;\cdot,\cdot)$ *is the nonlinear variational form associated the Kohn-Sham equation defined in Section 2.2.*

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

ParO as I understand it

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$

- Solve in parallel 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 small *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

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

A variant of ParO method (2)

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$

- Solve in parallel 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 small 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

A variant of ParO method (3)

- Given trial eigenpairs: $\{|\phi_i^{(n)}\rangle, \varepsilon_i^{(n)}\}$
- Solve in parallel 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 $|\tilde{\tilde{\phi}}_i^{(n)}\rangle = |\phi_i^{(n)}\rangle + |\tilde{\phi}_i^{(n)}\rangle$

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

- Diagonalize the small *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

Memory requirements for ParO method

- Memory required is nbnd * npwx + [nbnd*npwx] in the original ParO method or when $|\tilde{\phi}_i^{(n)}\rangle$ are used.

- Memory required is 3 * nbnd * npwx + [2*nbnd*npwx] if both $|\tilde{\phi}_i^{(n)}\rangle$ & $|\phi_i^{(n)}\rangle$ are used.

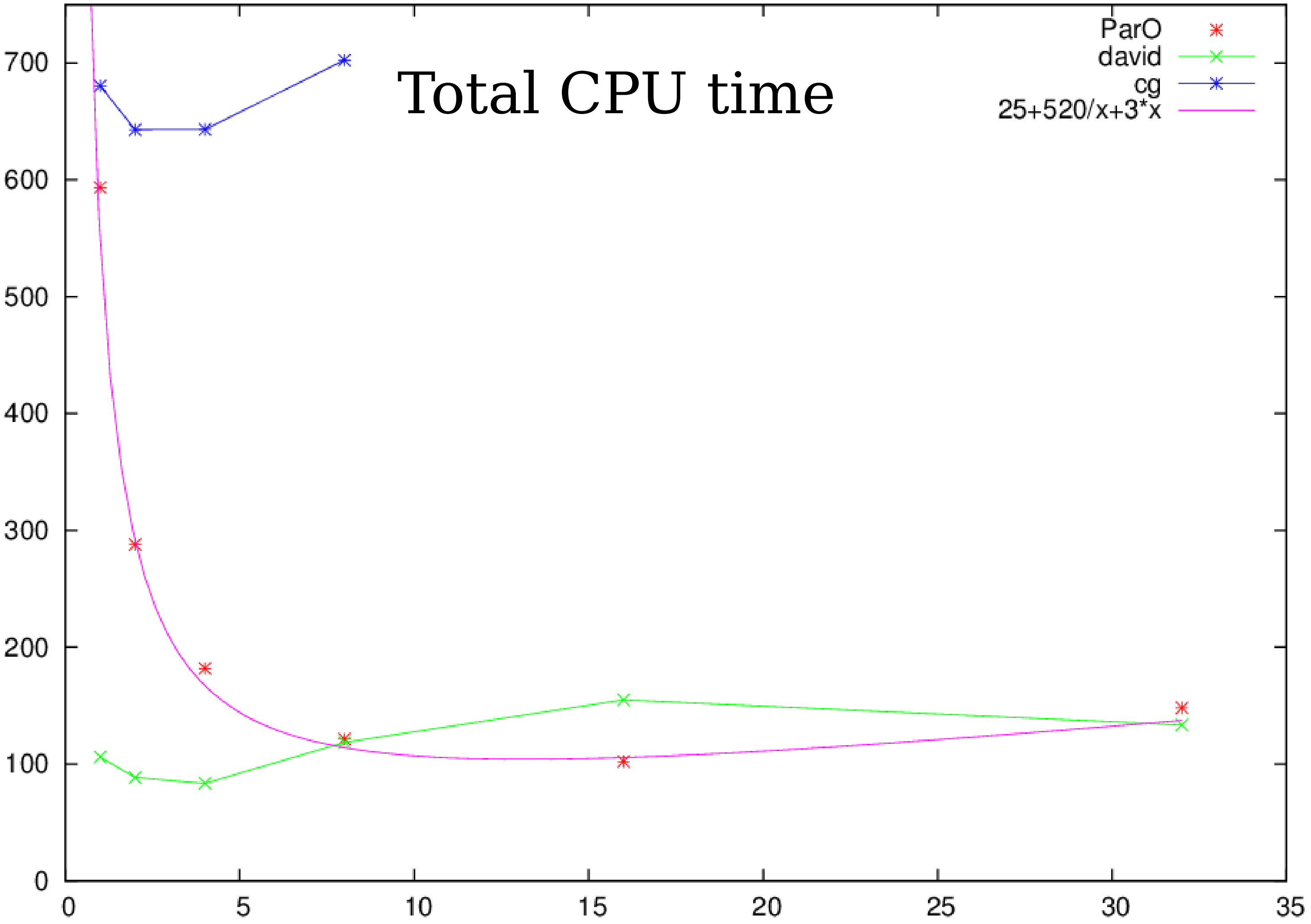
- Could be possible to reduce this memory and/or the number of h_psi involved by playing with the algorithm.

Comparison with the other methods

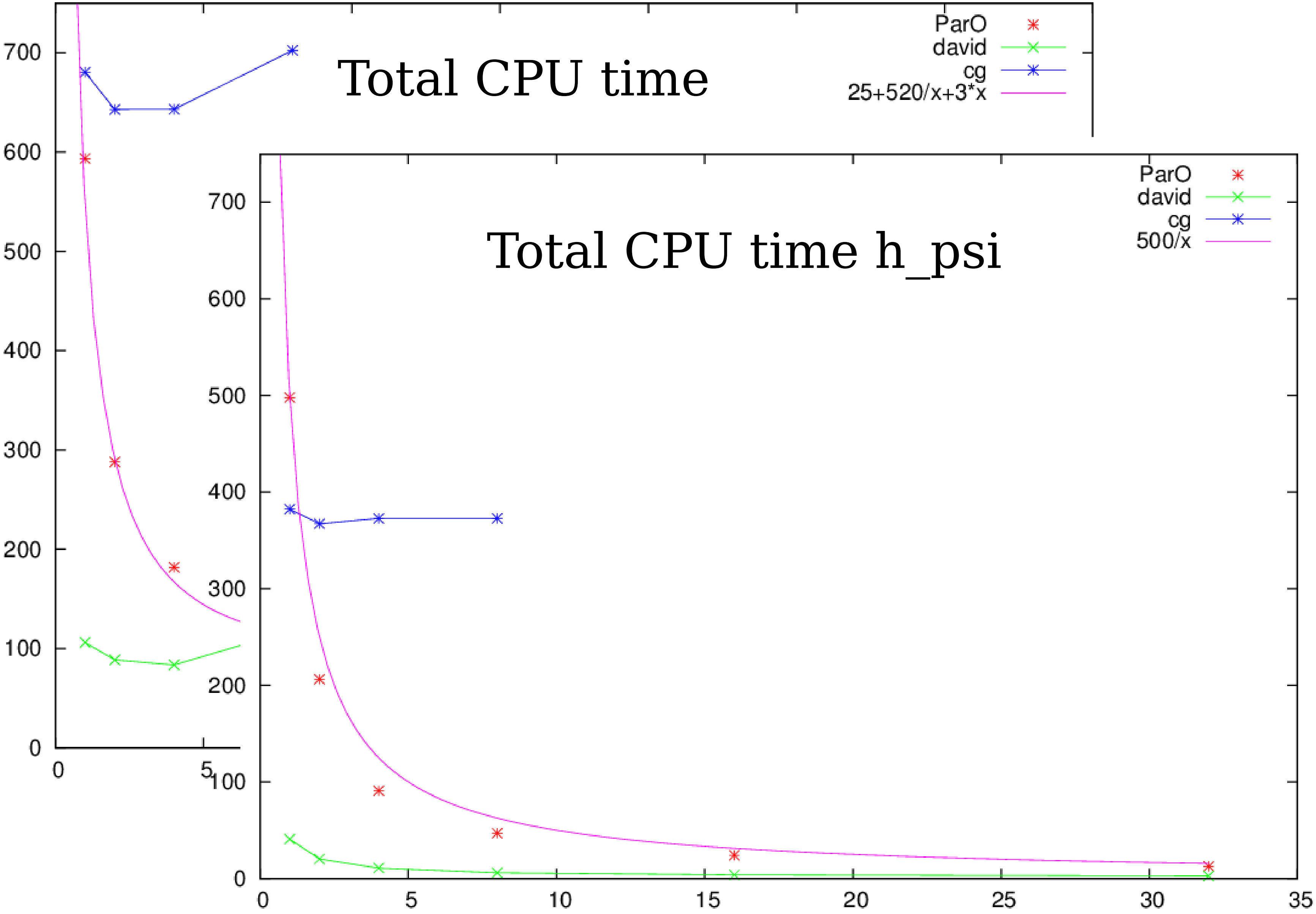
- NOT competitive with Davidson at the moment

- Timing and number of h_psi calls similar to cg on a single bgrp basis. It scales !

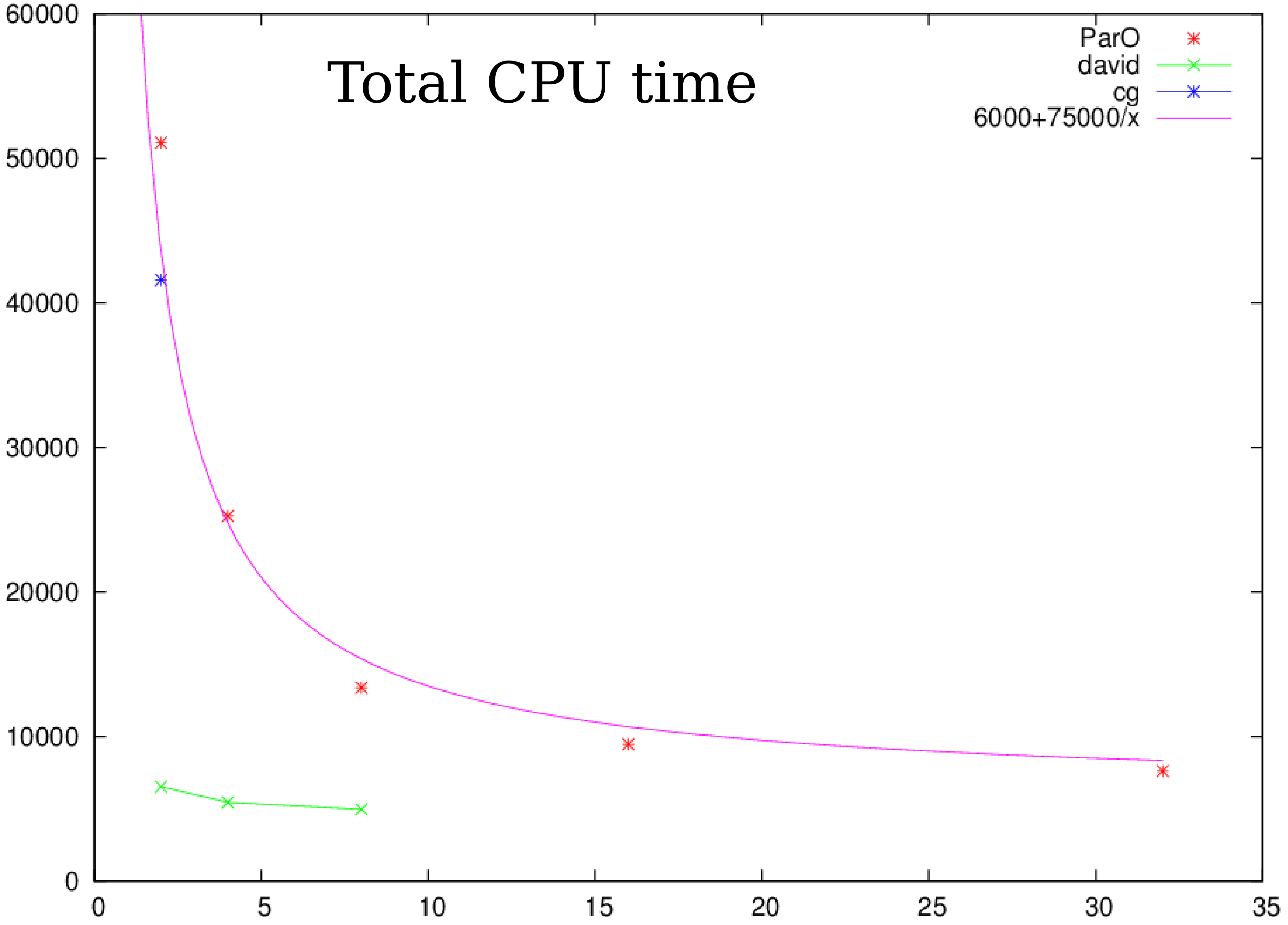
216 Si atoms in a SC cell : Timing



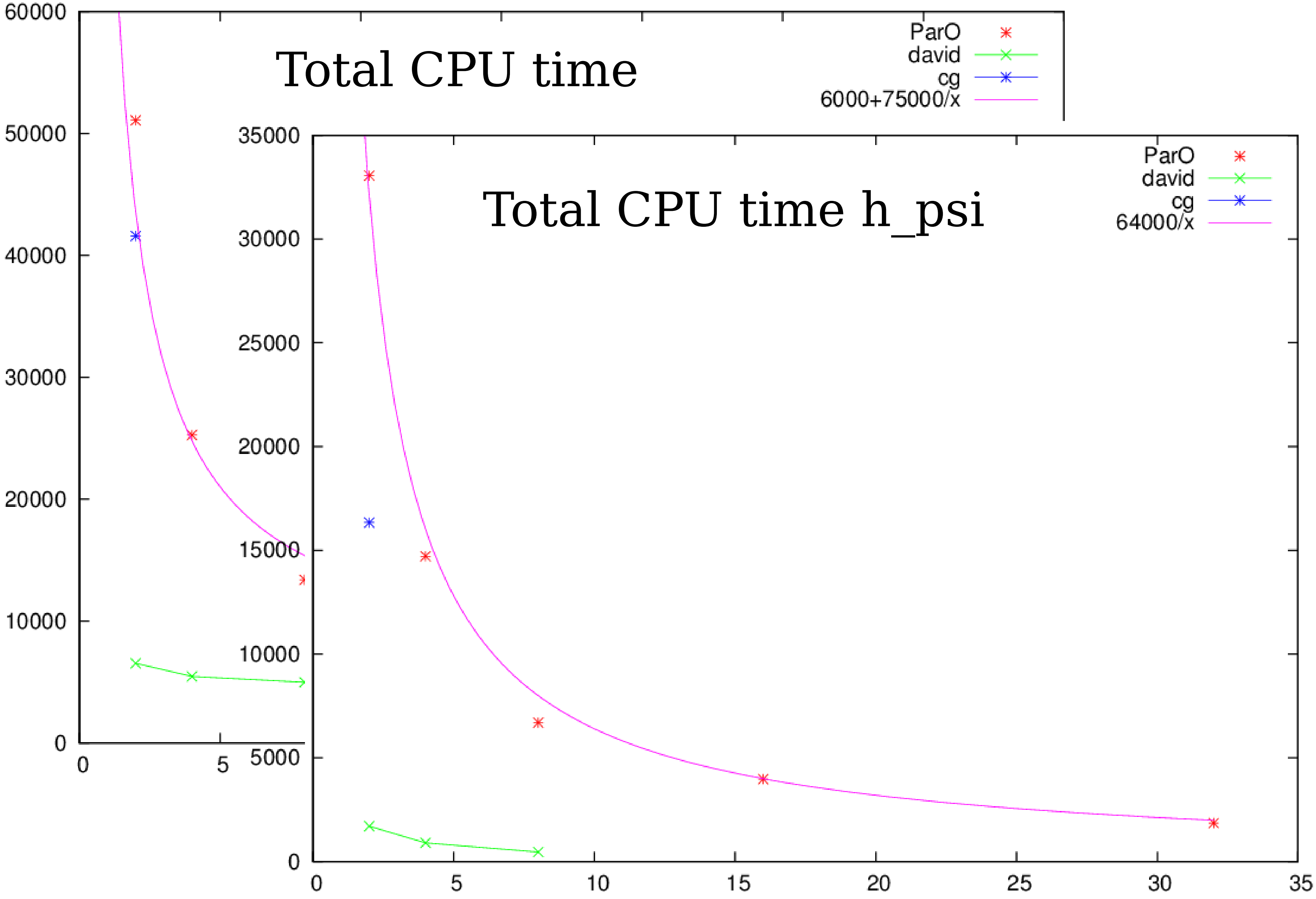
216 Si atoms in a SC cell : Timing



Not only Silicon: BaTiO3 320 atms, 2560 el



Not only Silicon: BaTiO3 320 atms, 2560 el



Comparison with the other methods

- NOT competitive with Davidson at the moment

- Timing and number of h_psi calls similar to CG on a single bgrp basis. It scales well with bgrp parallelization!

TO DO LIST

- Profiling of a few relevant test cases

- Extend band parallelization to other parts

- Understand why h_psi is so much more efficient in the Davidson method.

- See if number of h_psi can be reduced

- bgrp parallelization
- We should use bgrp parallelization more extensively distributing work w/o distributing data (we have R&G parallelization for that) so as to scale up to more processors.
- We can distribute different loops in different routines (nats, nkb, ngm, nrxx, …). Only local effects: incremental!
- A careful profiling of the code is required.
- ortho/diag parallelization
- It should be a sub comm of the pool comm (k-points) not of the bgrp comm.
- Does it give any gain ? Except for some memory reduction I saw no gain (w/o scalapack).
- task parallelization
- Only needed for very large/anisotropic systems, intrinsically requiring many more processors than planes.
- Is not a method to scale up the number of processors for a “small” calculation (should use bgrp parallelization for that).
- Should be activated also when m < dffts%nogrp

Porting MaX community codes to novel architectures using CUDA Fortran: the QE experience

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

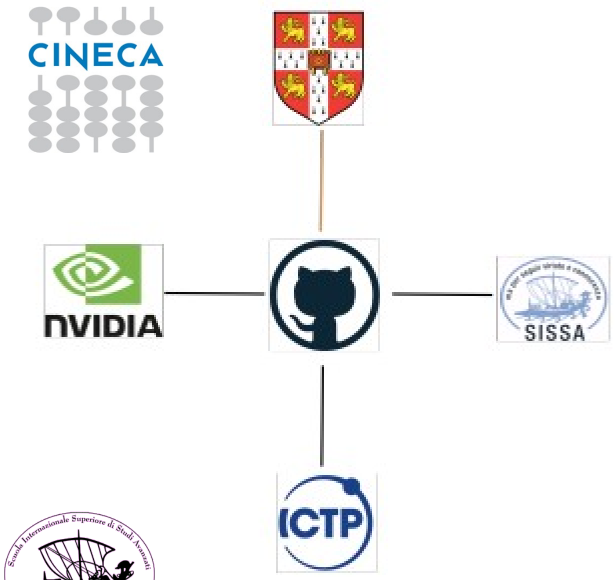
success has many fathers, failure is an orphan

Tacitus Agricola (98), Galeazzo Ciano (1942), JFK (1961), ...

In February 2017 Massimiliano Fatica (nvidia) came to Trieste to present CUDA Fortran and their GPU work on QE.

A number of QE developers were present as well as Anoop Chandran (SISSA/ICTP MHPC student supported by QEF)

Interest from nvidia to keep supporting development in QE has been confirmed recently.



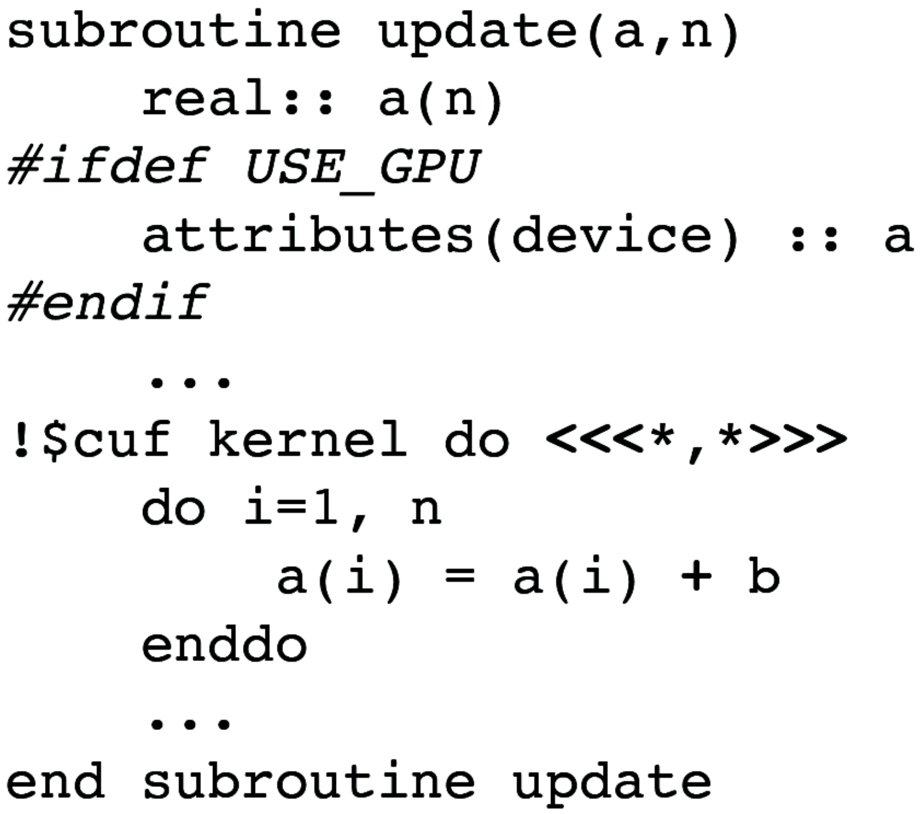
CUDA Fortran is basically Fortran

```
module m
    ...
    real :: a(n)
    ...
end module

subroutine update
    use m, only: a
    ...
    do i=1, n
        a(i) = a(i) + b
    enddo
    ...
end subroutine update
```

CUDA Fortran is basically Fortran

```
module m
    ...
    real :: a(n)
    real,device :: a_d(n)
    ...
end module

subroutine update
#ifdef USE_GPU
    use m, only: a => a_d
#else
    use m, only: a
#endif
    ...
!$cuf kernel do
    do i=1, n
        a(i) = a(i) + b
    enddo
    ...
end subroutine update
```

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.

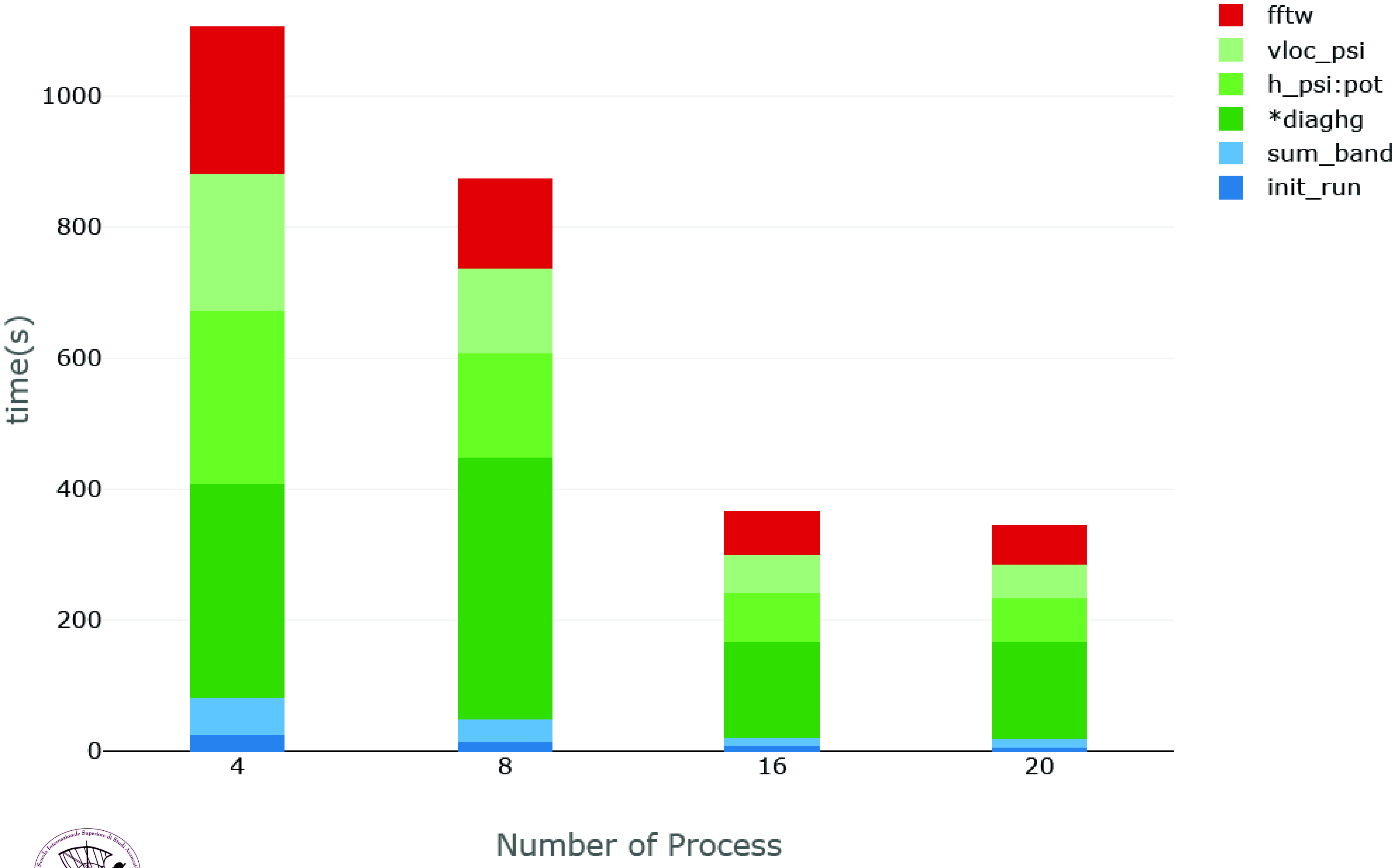
Diagonalization of H_{KS} is a major step in the scf solution of any system.

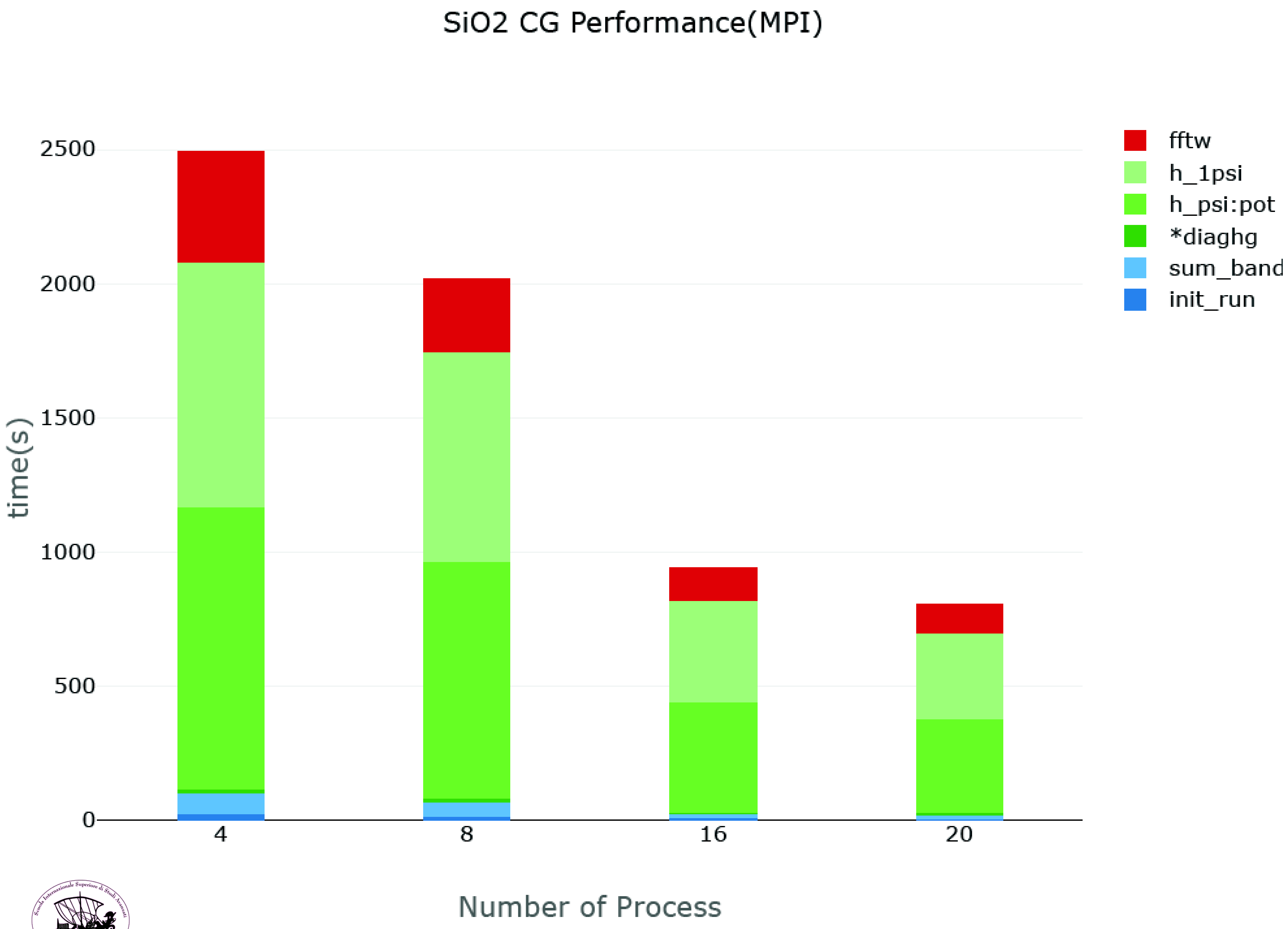
In pw.x in QE two methods are implemented:

- 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.

SiO2 Davidson Performance(MPI)




SiO2 CG Performance(MPI)
time(s)
2500
2000
1500
1000
500
0
4
8
16
20
Number of Process
🟥 fftw
🟩 h_1psi
🟩 h_psi:pot
🟩 *diaghg
🟦 sum_band
🟦 init_run

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)

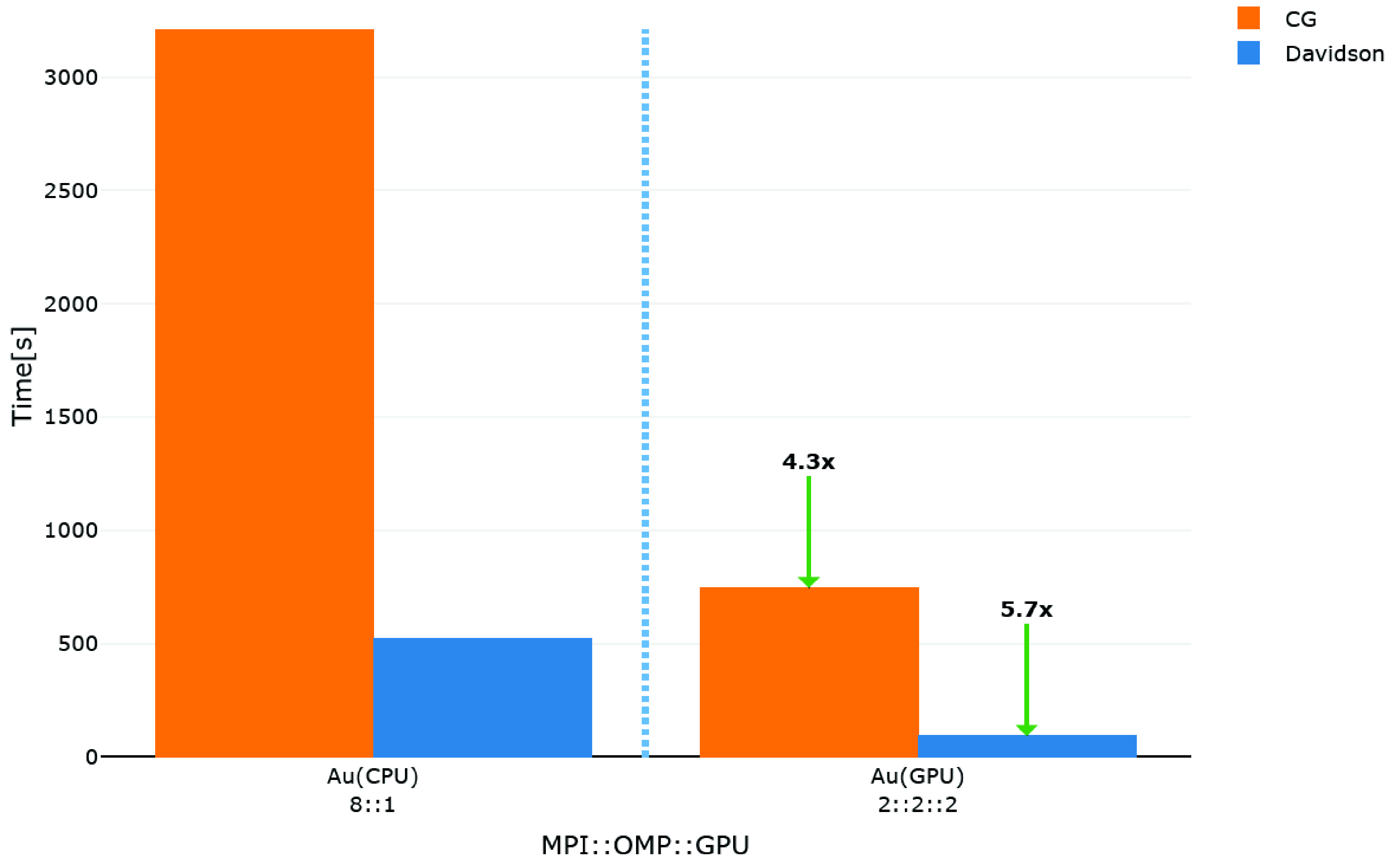
comparison depends on the selected architecture.

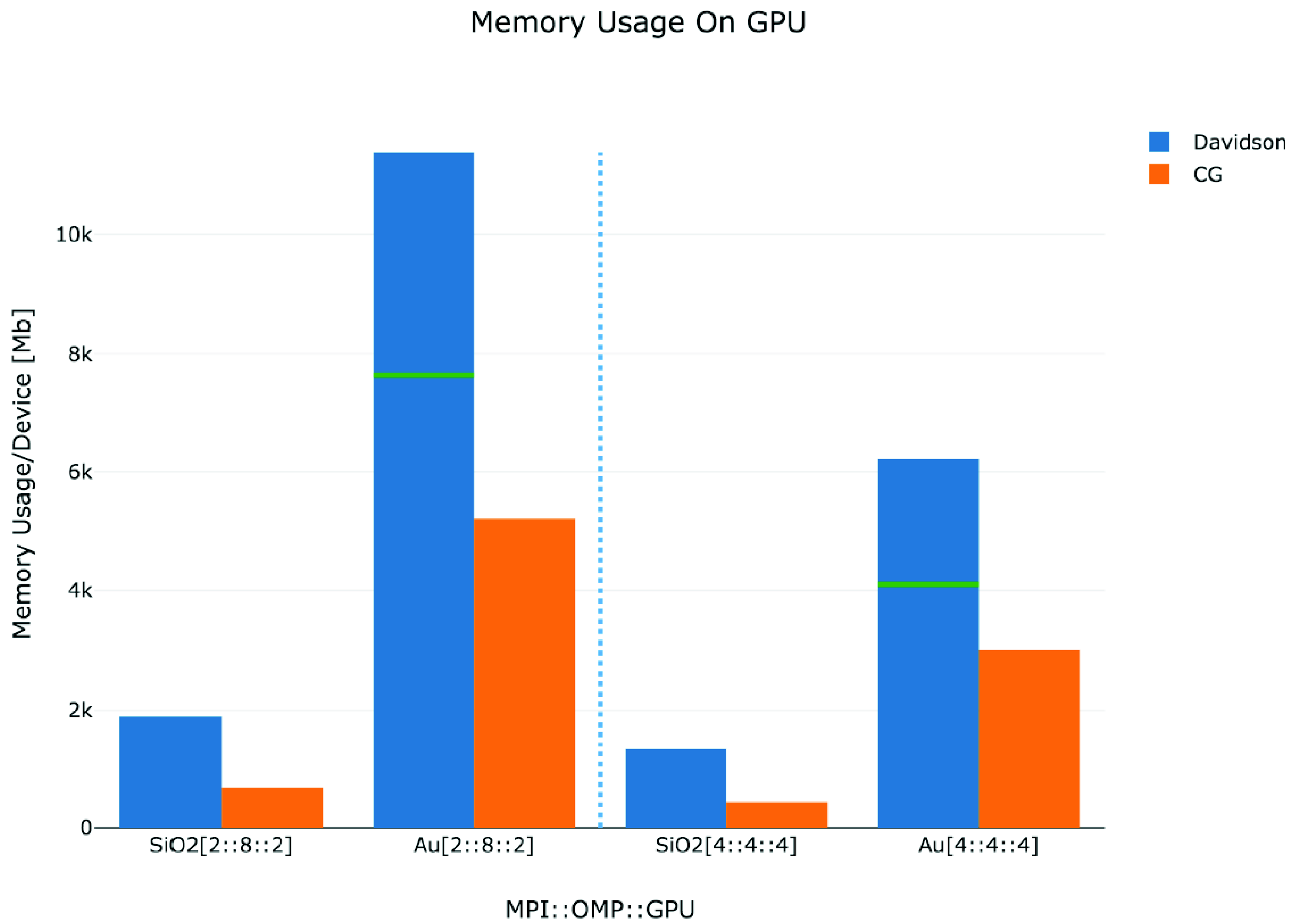
a reliable performance modeling would be very useful to make rational choices when buying hardware for and allocating resources to a user community.

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

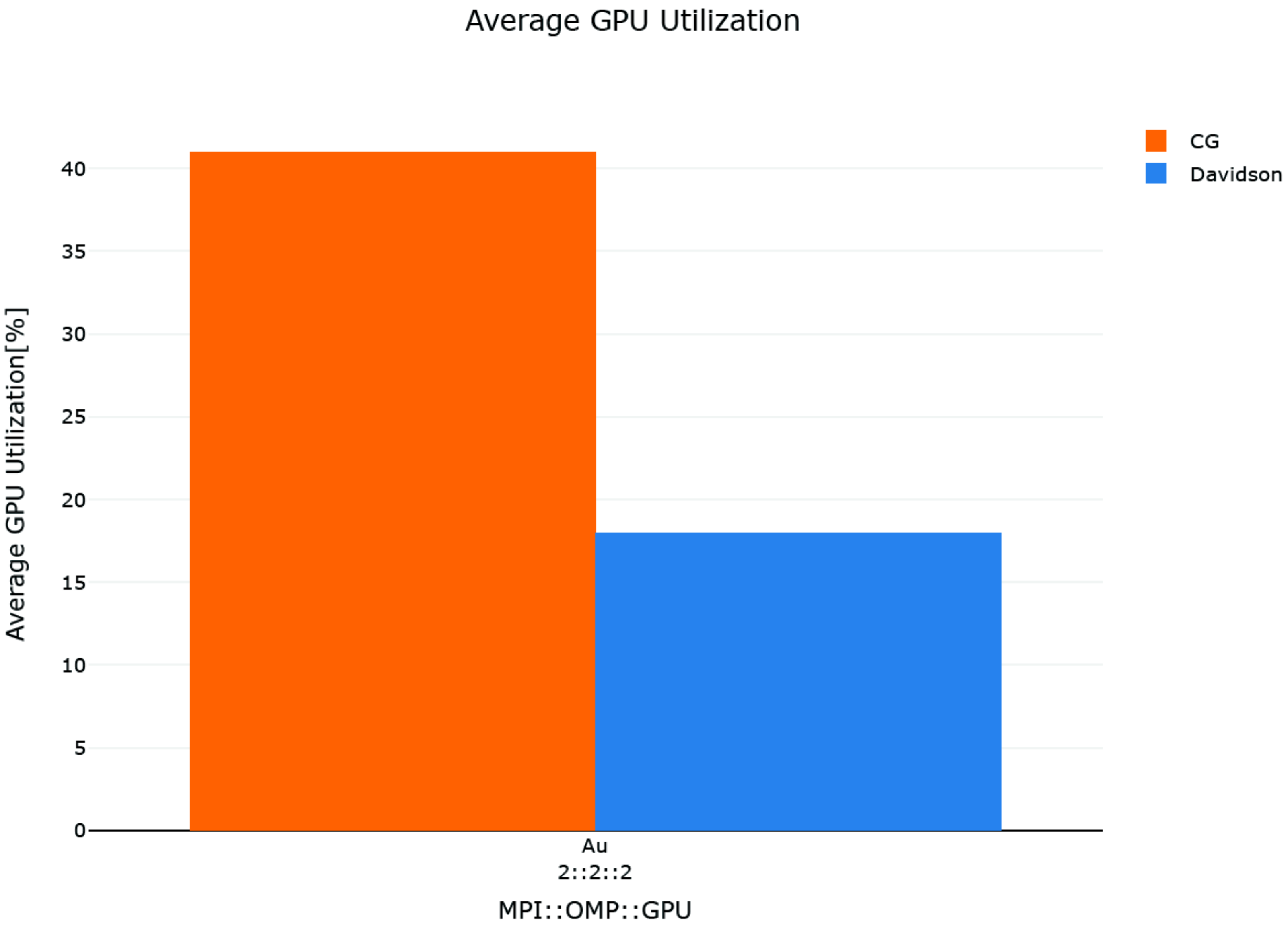
-Davidson/CG solvers, more recently Force computation

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