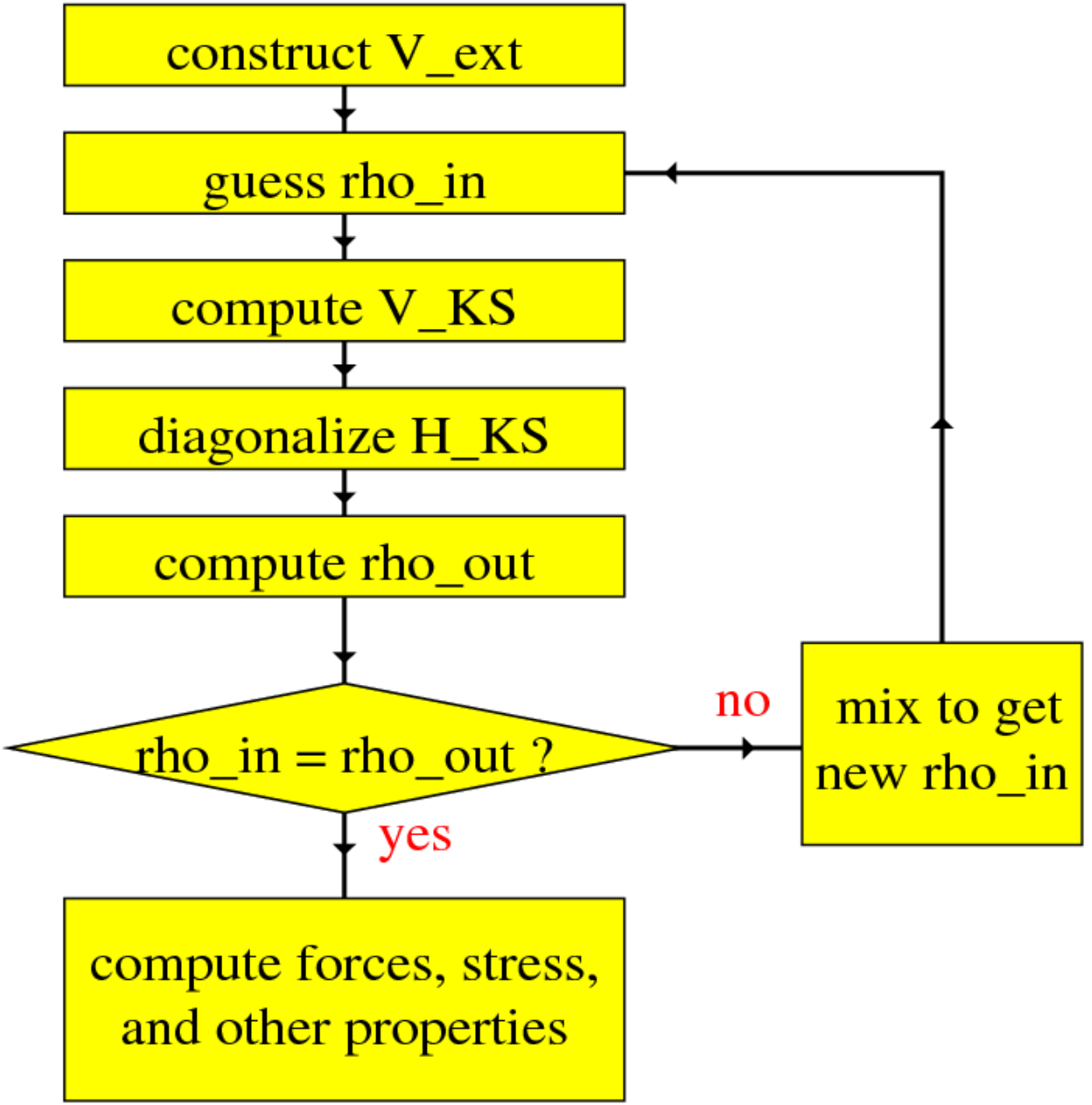
PWSCF

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

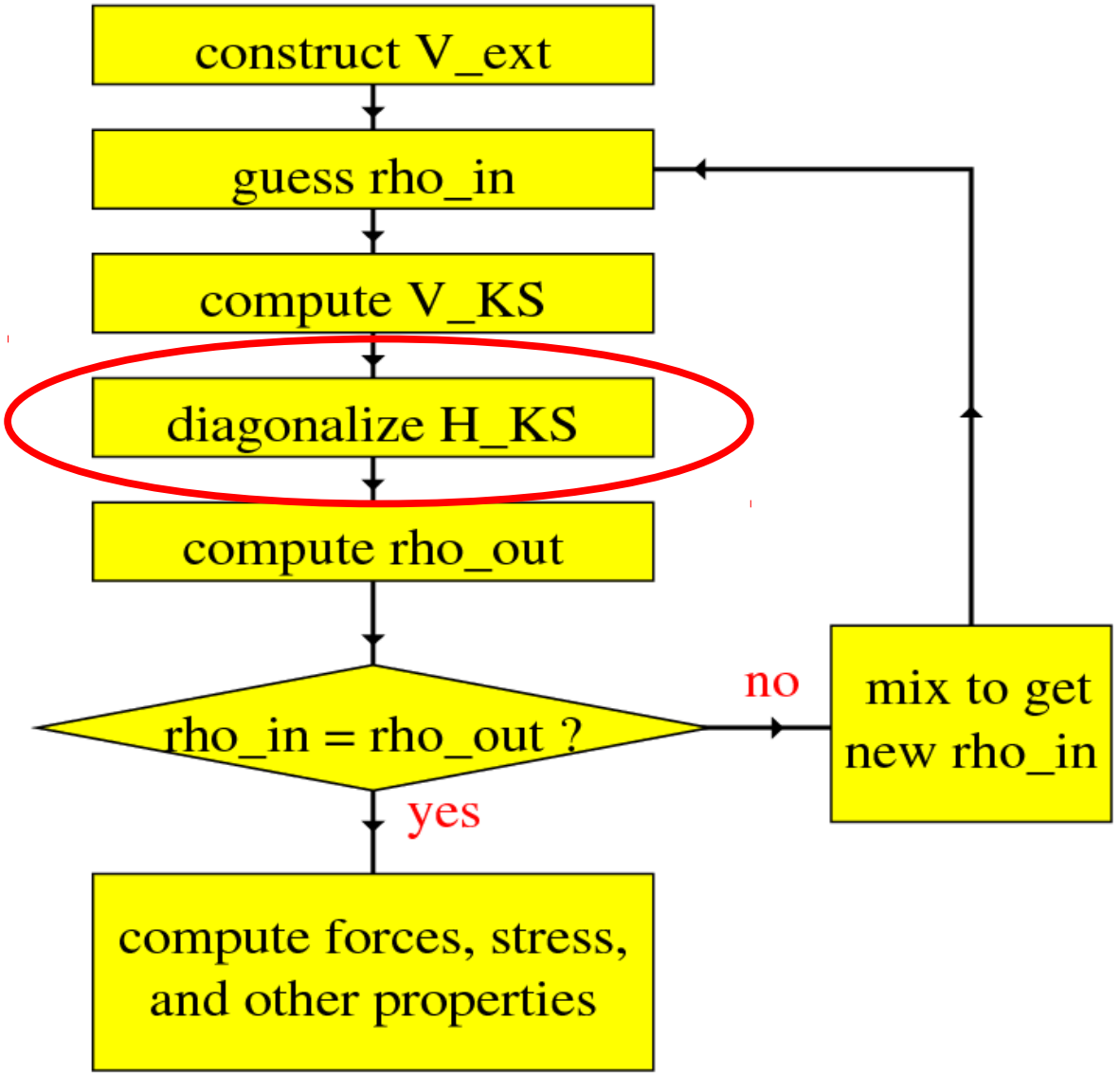
```
do ik = 1, nks
   call get_buffer     (evc)
   call init_us_2      (vkb)
   call diag_bands    -->   DIAG_BANDS
   call save_buffer
end do ik
```

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



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> $2nbnd \times 2nbnd$ 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*

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

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

.... 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

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 $\tilde{V}_{n+1} = \text{span}\{u_1^{(n+1/2)}, u_2^{(n+1/2)}, \cdots, u_N^{(n+1/2)}\}$, $U^{(n)} = (u_1^{(n)}, u_2^{(n)}, \cdots, u_N^{(n)})$, $U^{(n+1/2)} = (u_1^{(n+1/2)}, u_2^{(n+1/2)}, \cdots, u_N^{(n+1/2)})$, *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 <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

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> $2nbnd \times 2nbnd$ 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 solution at fixed Hamiltonian

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

A variant of ParO method (3)

- 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 $|\tilde{\phi}_i^{(n)}\rangle = |\phi_i^{(n)}\rangle + |\tilde{\phi}_i^{(n)}\rangle$

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

- 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

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.

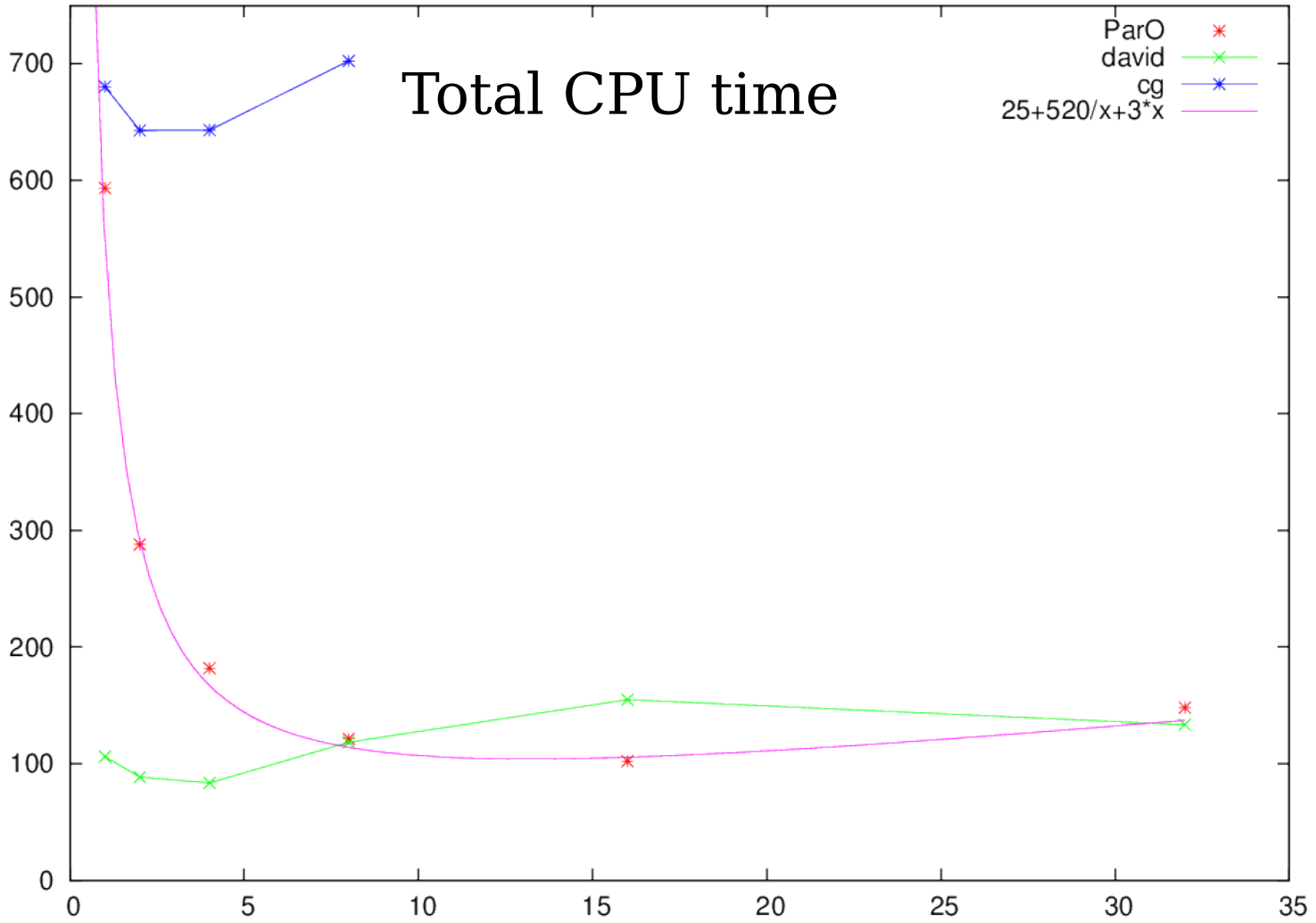
- Could be possible to reduce this memory and/or the number of h_psi involved by playing with the algorithm.

Comparison with the other methods

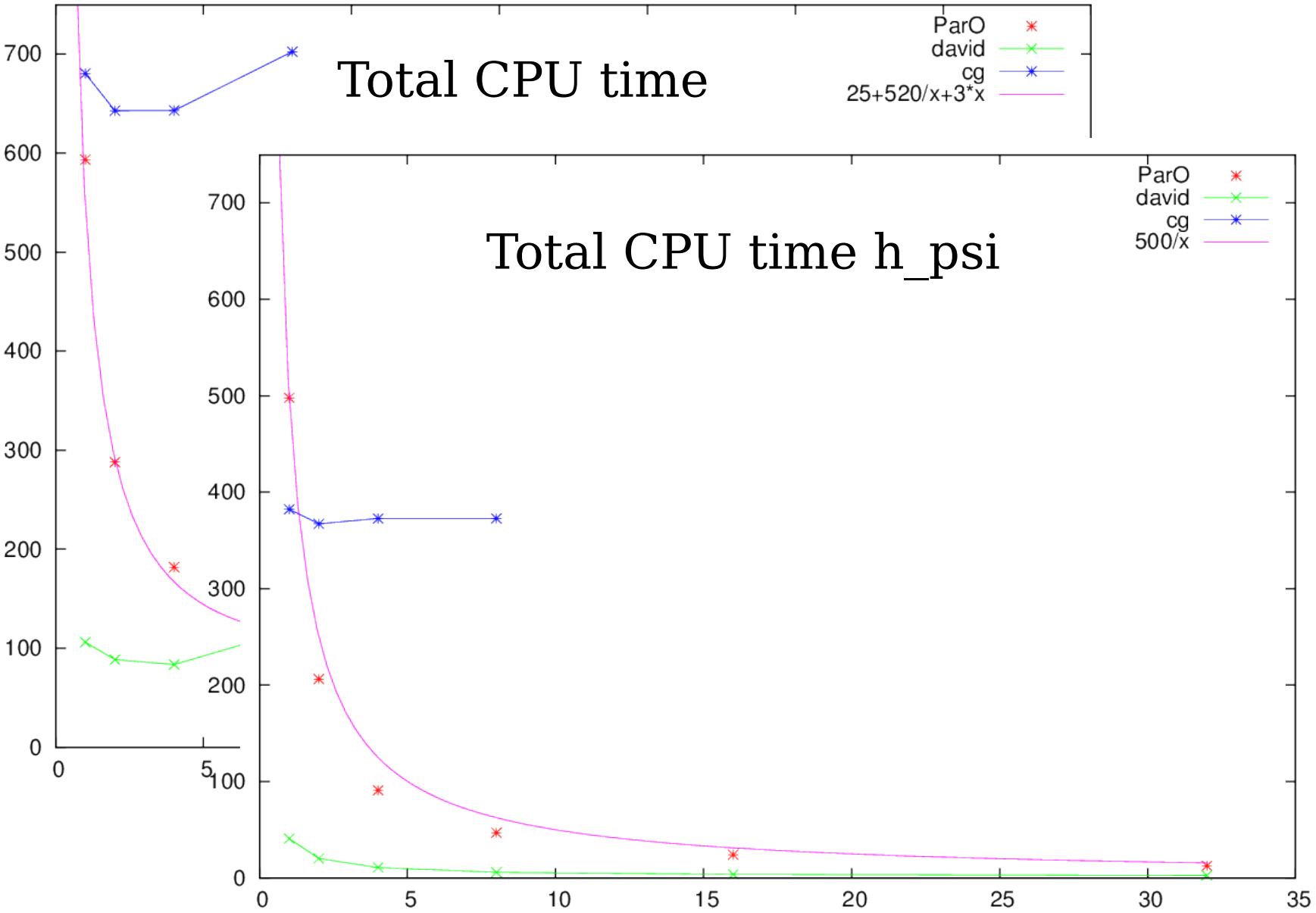
- NOT competitive with Davidson at the moment

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

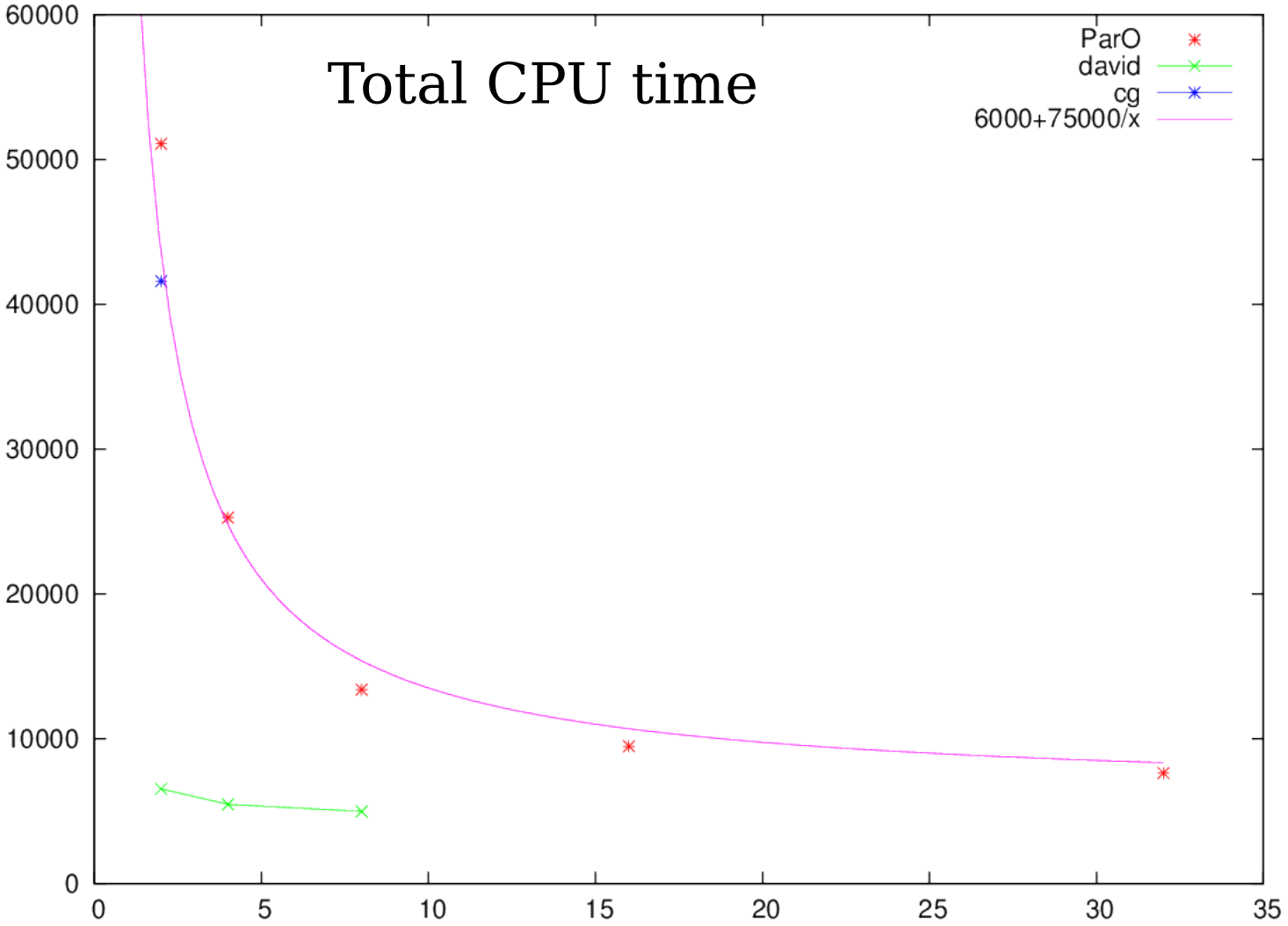
216 Si atoms in a SC cell : Timing



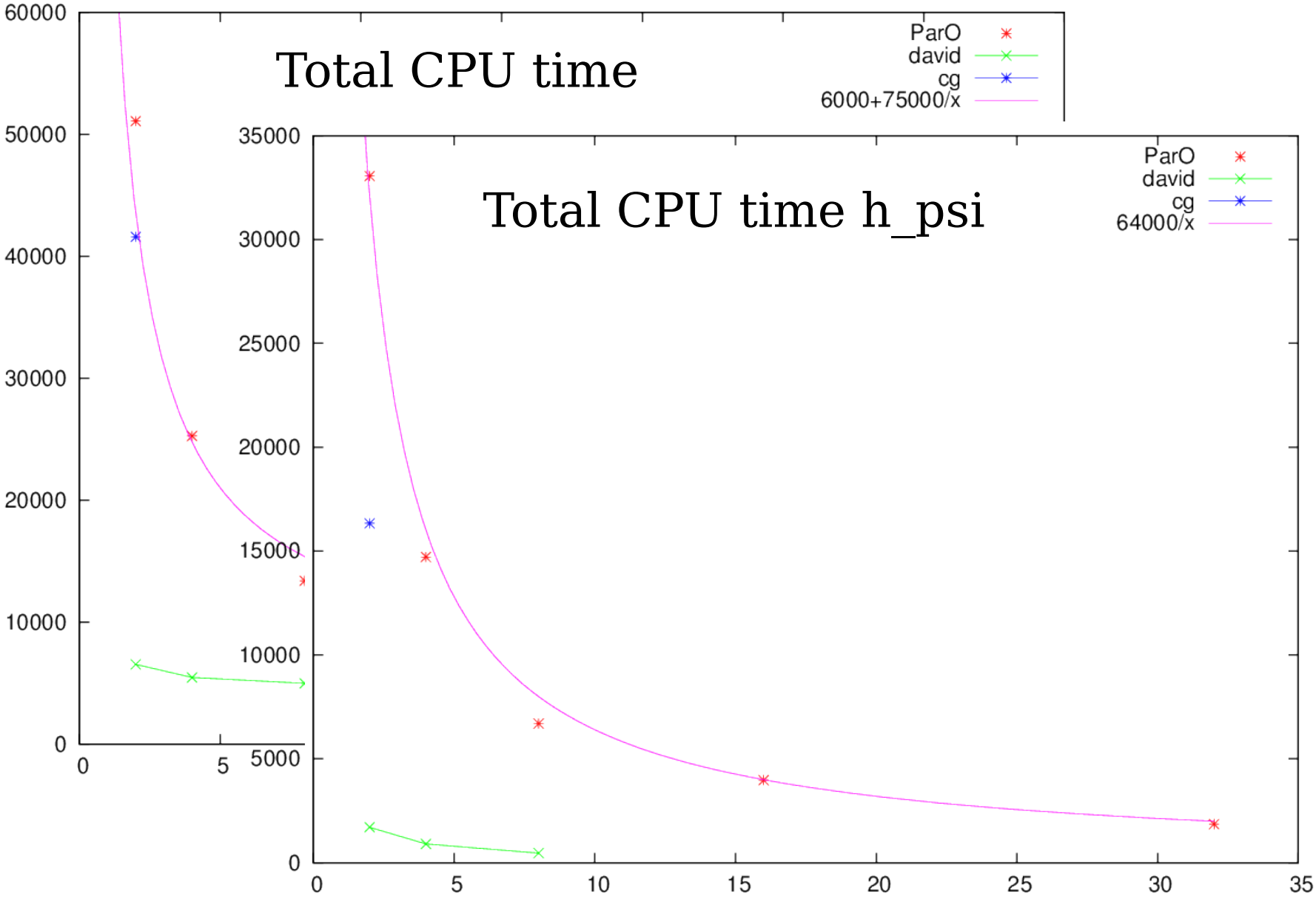
216 Si atoms in a SC cell : Timing



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