QUANTUM ESPRESSO FOUNDATION

QUANTUMESPRESSO

MaX School on Advanced Materials and Molecular Modelling with QUANTUM ESPRESSO

Quantum ESPRESSO on HPC and GPU systems: parallelization and hybrid architectures

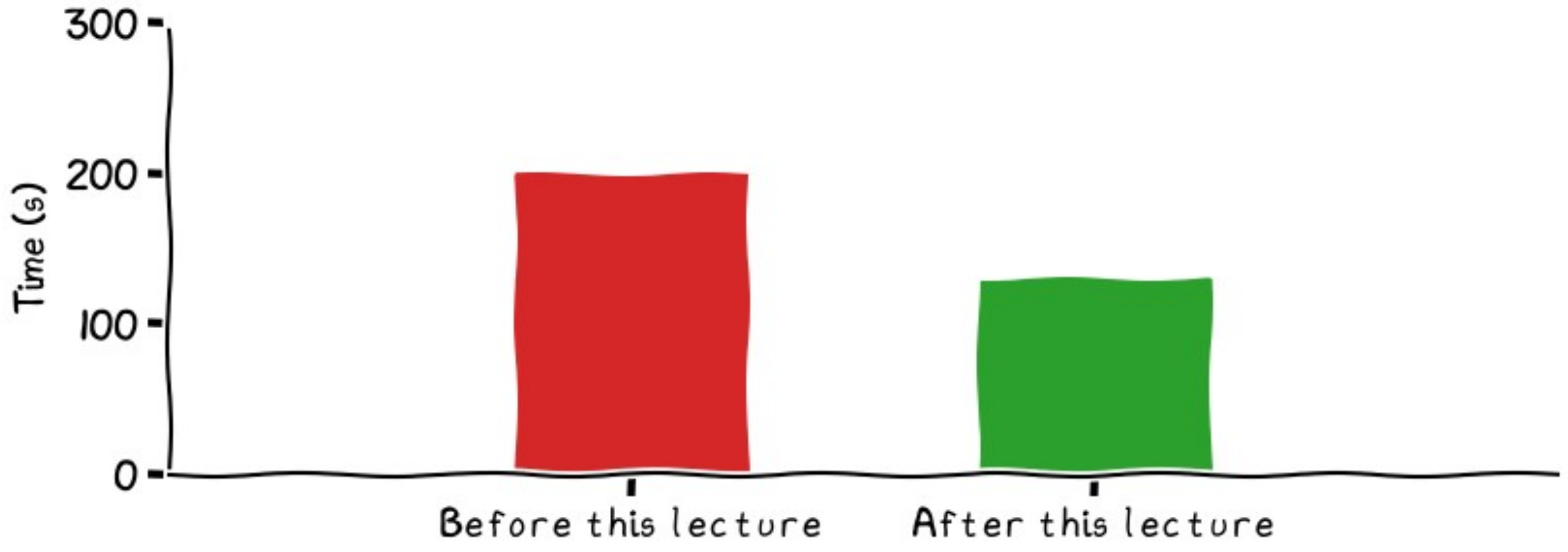
Pietro Bonfà
Università di Parma and CNR-Nano, Italy

MaX School on Advanced Materials and Molecular Modellingwith Quantum ESPRESSO
May 17-28 2021,
ICTP Virtual Meeting

MAX · ICTP The Abdus Salam International Centre for Theoretical Physics · CINECA · 上海大学 SHANGHAI UNIVERSITY · Jožef Stefan Institute Ljubljana, Slovenia · cecam · SISSA

Running on a laptop

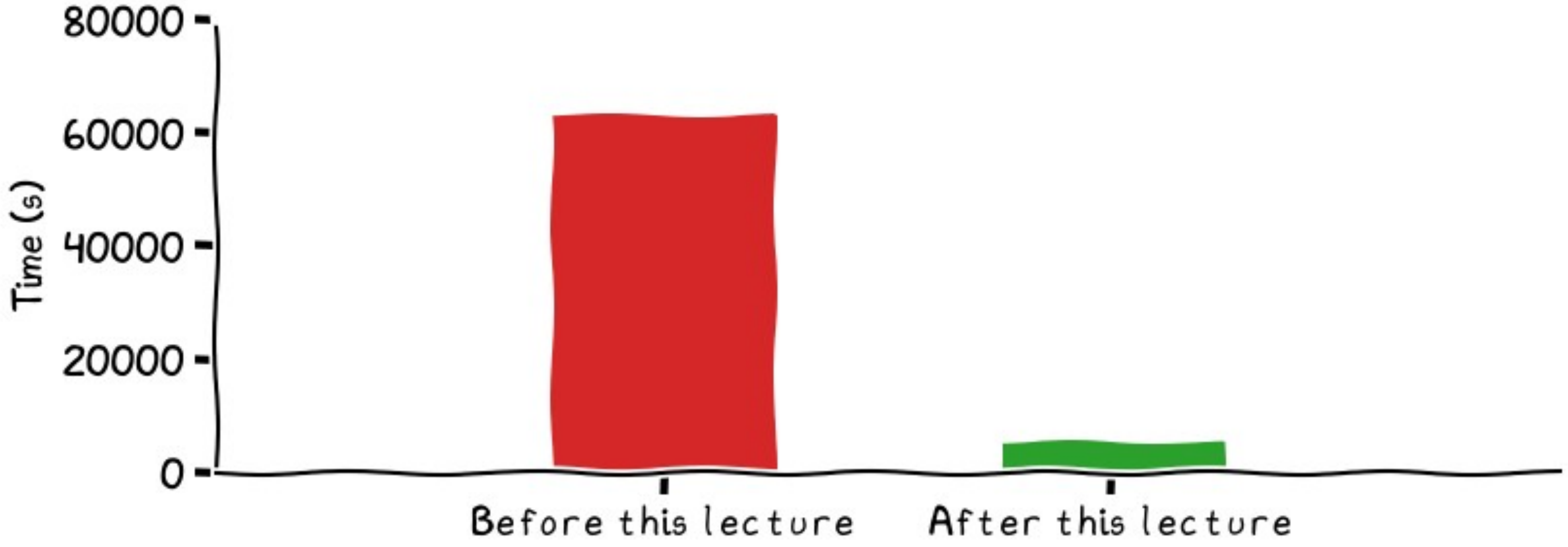
- You go home and type
  ./configure && make pw



On a real HPC node...

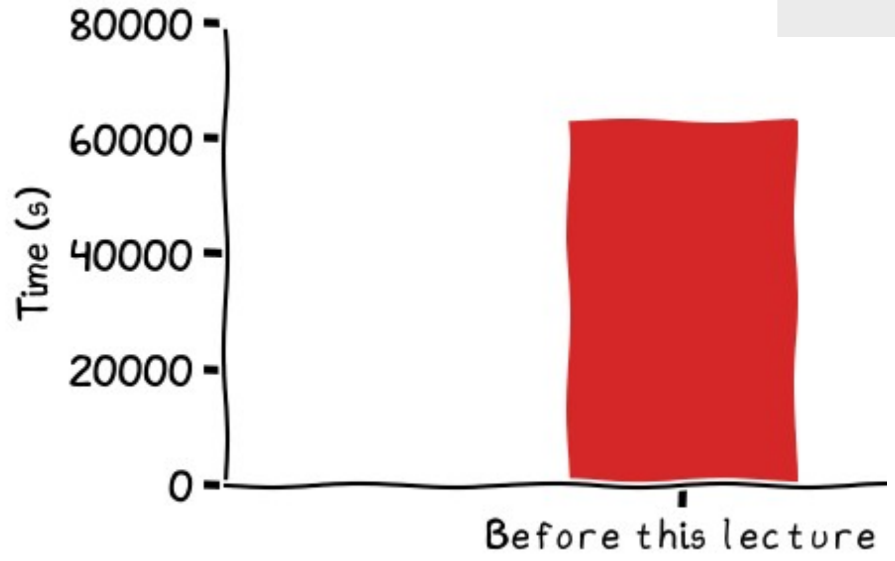
2 x AMD Epyc7352 + 4 x NVIDIA A100

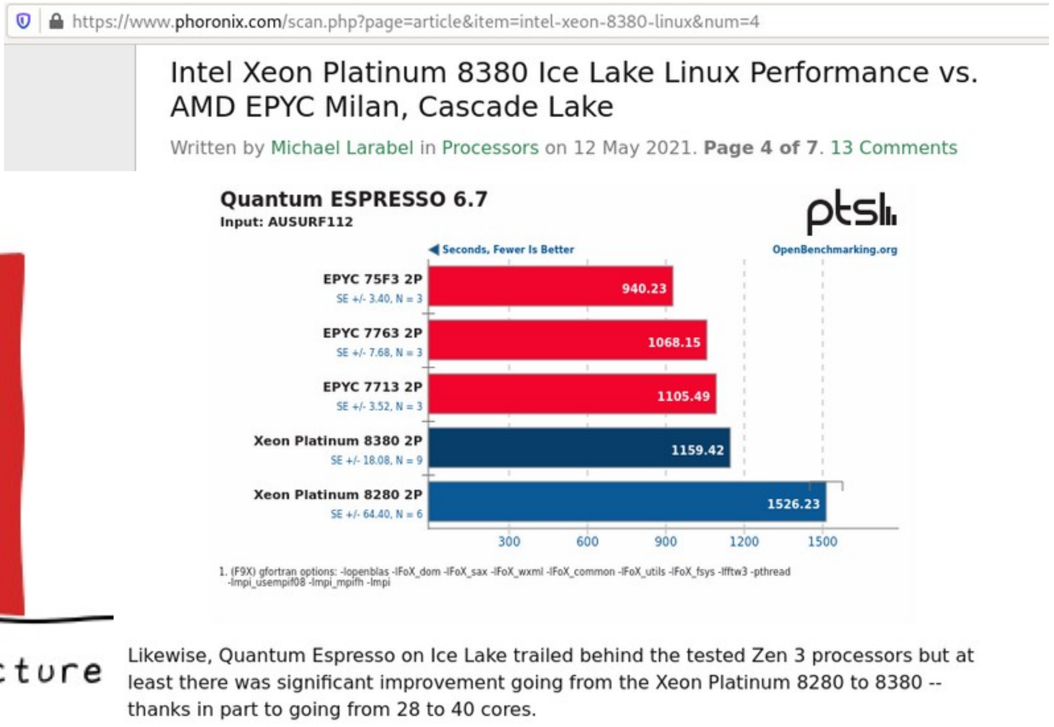
```
./configure && make pw
```



On a real HPC node...

2 x AMD Epyc7352 + 4 x NVIDIA A100





https://www.phoronix.com/scan.php?page=article&item=intel-xeon-8380-linux&num=4

Intel Xeon Platinum 8380 Ice Lake Linux Performance vs. AMD EPYC Milan, Cascade Lake

Written by Michael Larabel in Processors on 12 May 2021. **Page 4 of 7**. 13 Comments

Quantum ESPRESSO 6.7
Input: AUSURF112

Seconds, Fewer Is Better

	Seconds
EPYC 75F3 2P (SE +/- 3.40, N = 3)	940.23
EPYC 7763 2P (SE +/- 7.68, N = 3)	1068.15
EPYC 7713 2P (SE +/- 3.52, N = 3)	1105.49
Xeon Platinum 8380 2P (SE +/- 18.08, N = 9)	1159.42
Xeon Platinum 8280 2P (SE +/- 64.40, N = 6)	1526.23

1. (F9X) gfortran options: -lopenblas -lFoX_dom -lFoX_sax -lFoX_wxml -lFoX_common -lFoX_utils -lFoX_fsys -lfftw3 -pthread -lmpi_usempif08 -lmpi_mpifh -lmpi

Likewise, Quantum Espresso on Ice Lake trailed behind the tested Zen 3 processors but at least there was significant improvement going from the Xeon Platinum 8280 to 8380 -- thanks in part to going from 28 to 40 cores.

QE running on HPC systems

- Parallel – Message Passing Interface
- Parallel – OpenMP
- Hierarchical levels of parallelization for fine grain performance tuning
- Accelerated – CUDA Fortran for NVIDIA GPUs

Parallel computing
(a concise introduction)

Amdahl's law

- A task takes the time **T** to run.
- A portion **p** of **T** may benefit from parallel execution. <u>That portion</u> becomes **s** times faster.
- The original task now takes

$$T'(p, s) = (1 - p)T + \frac{p}{s}T$$

Amdahl's law

The speedup of the whole task is

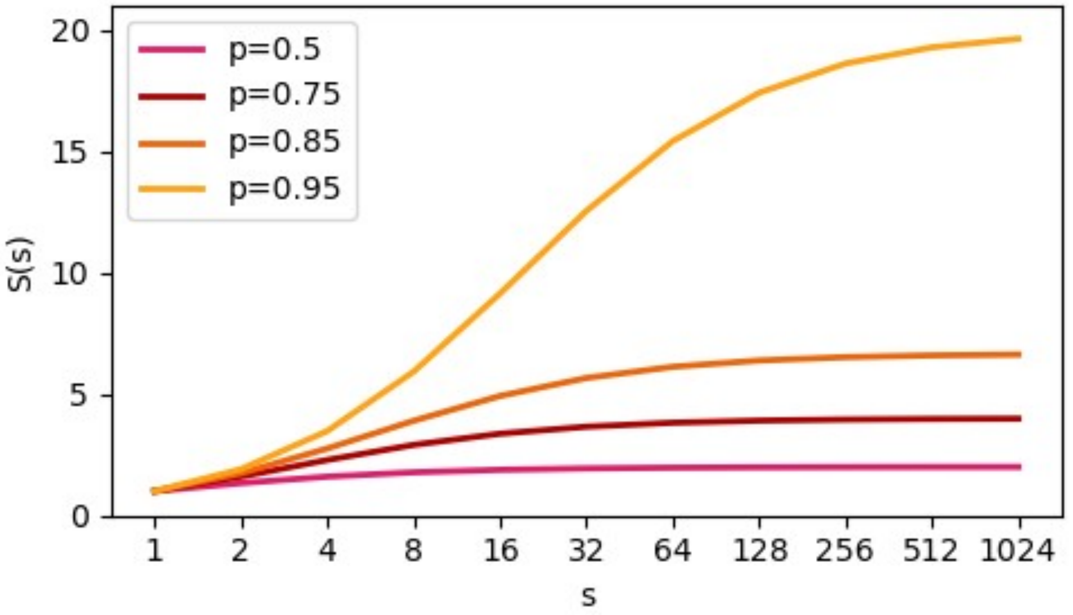
$$S = \frac{T}{T'}$$

and from

$$T'(p,s) = \left[(1-p) + \frac{p}{s}\right] T$$

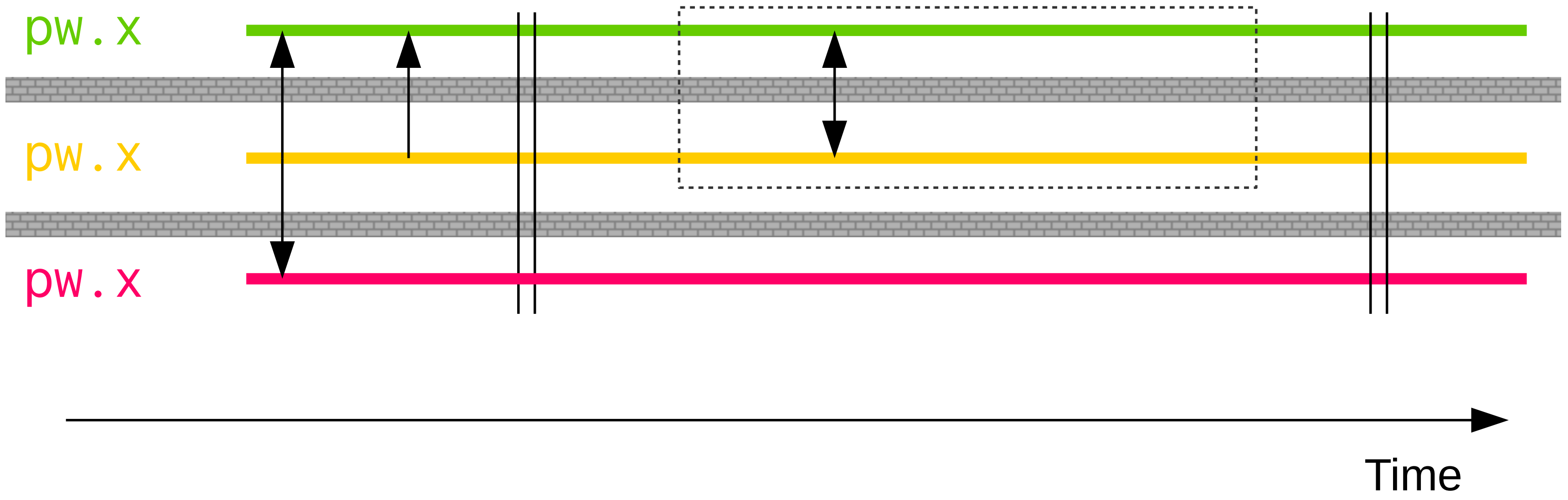
it is easily obtained

$$S(p,s) = \frac{1}{(1-p) + \frac{p}{s}}$$



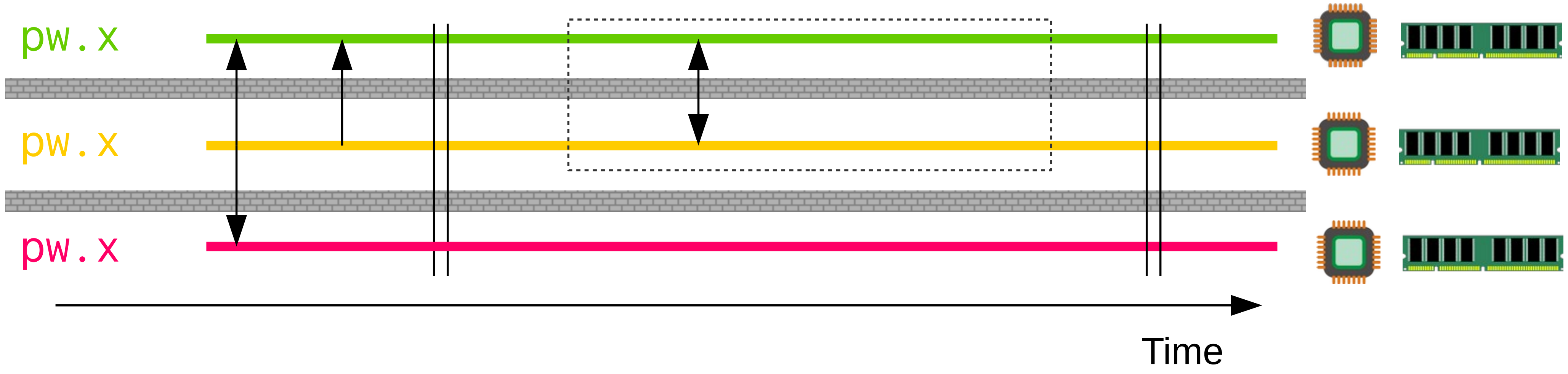
Message Passing

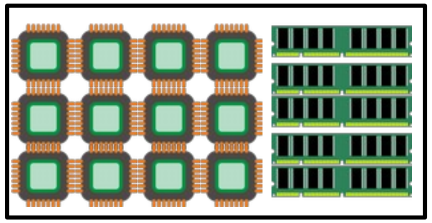
mpirun -np 3 pw.x



Message Passing

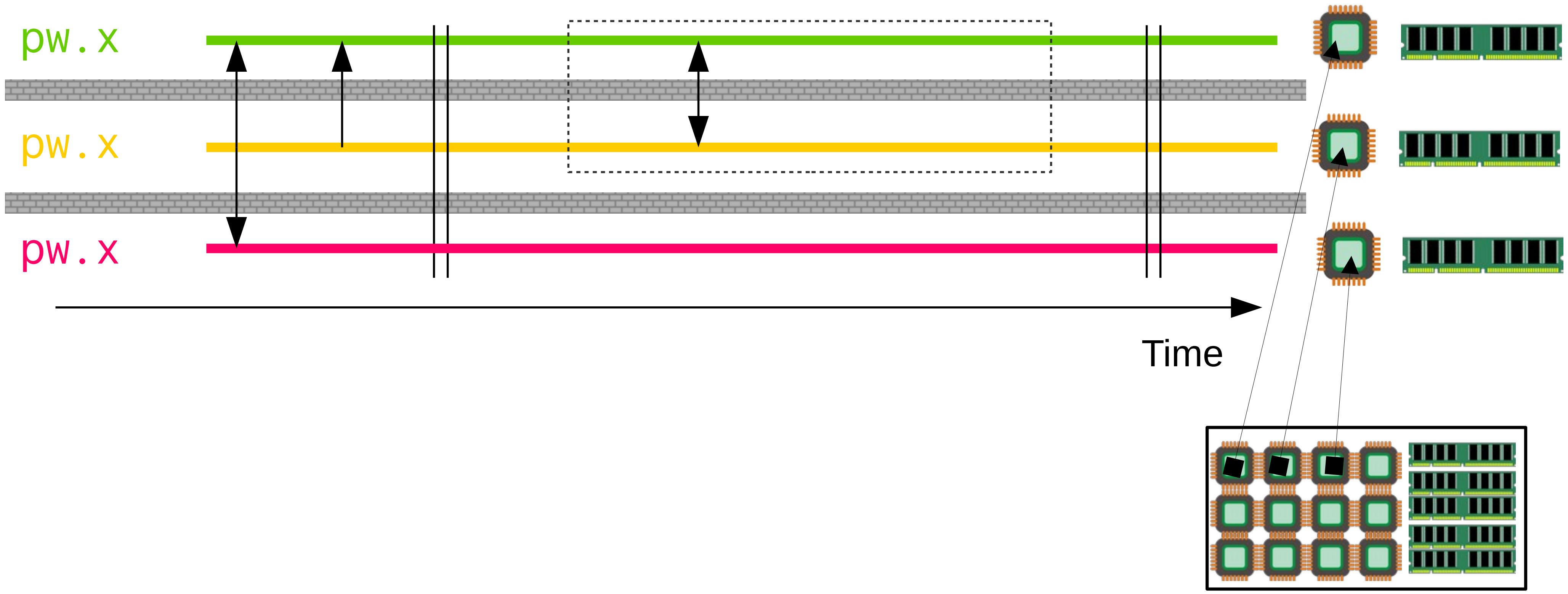
```
mpirun -np 3 pw.x
```





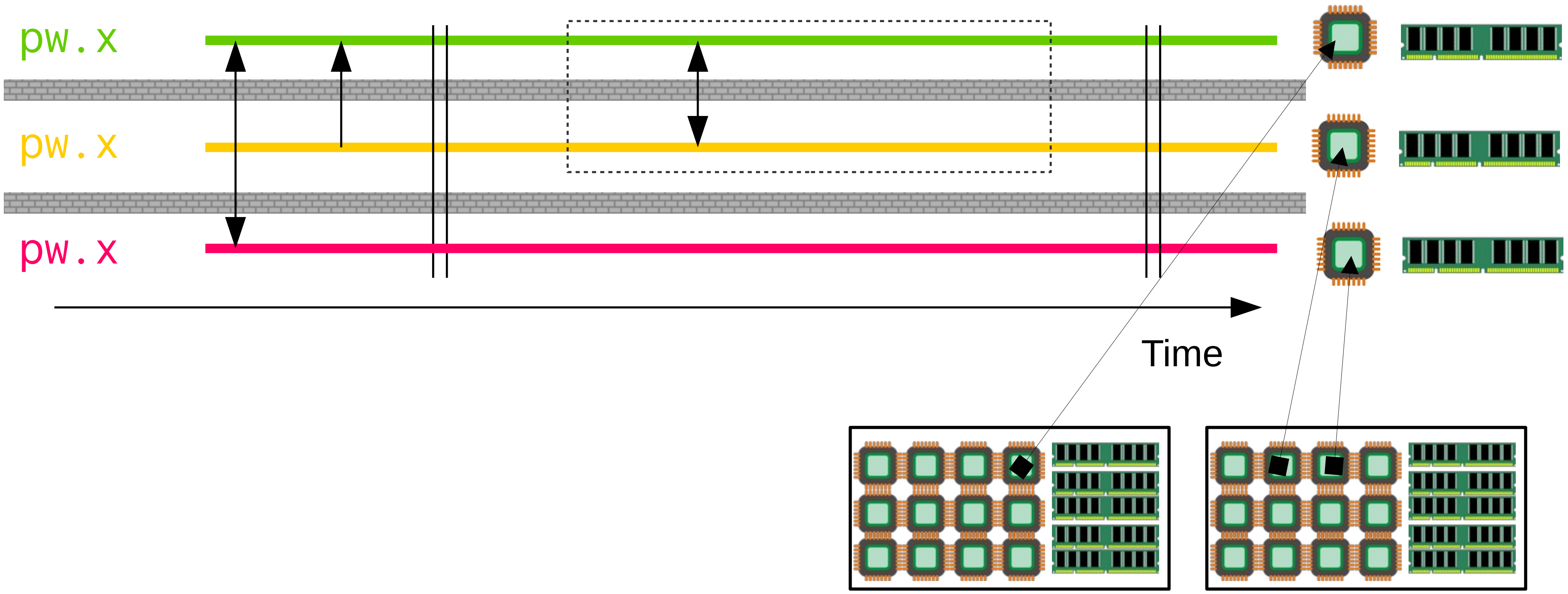
Message Passing

```
mpirun -np 3 pw.x
```



Message Passing

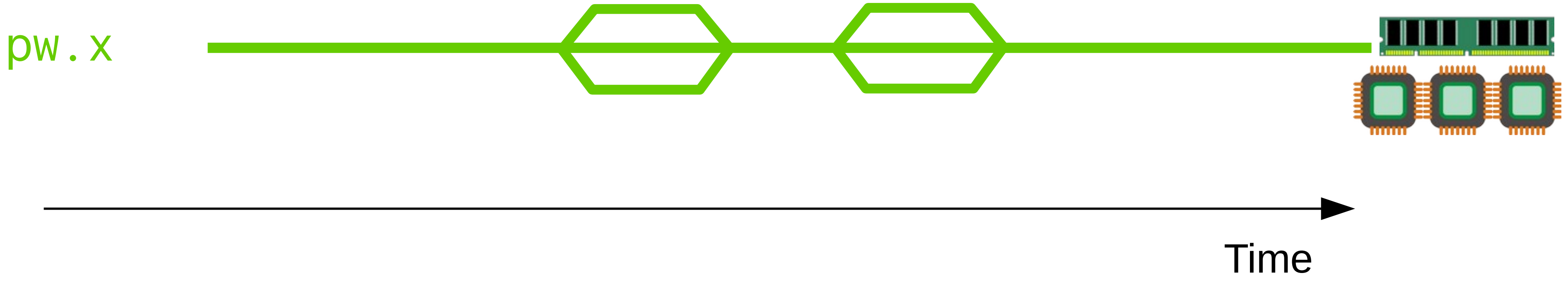
mpirun -np 3 pw.x



Message Passing & OpenMP

Export OMP_NUM_THREADS=3

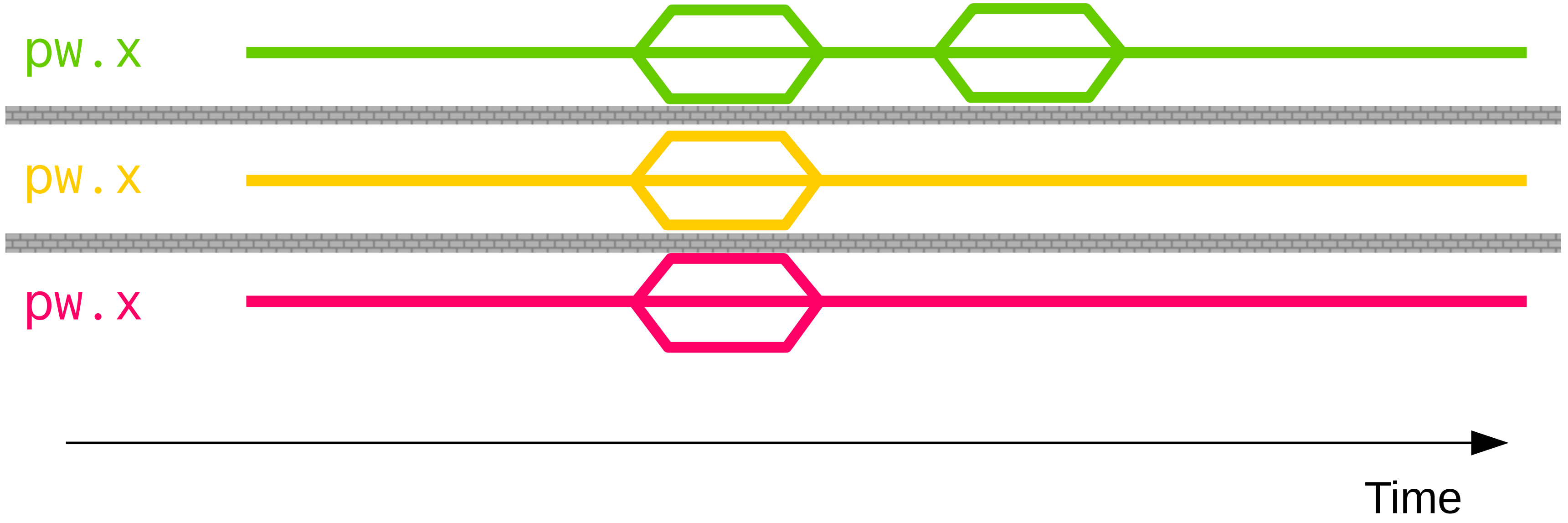
./pw.x



Message Passing & OpenMP

Export OMP_NUM_THREADS=3

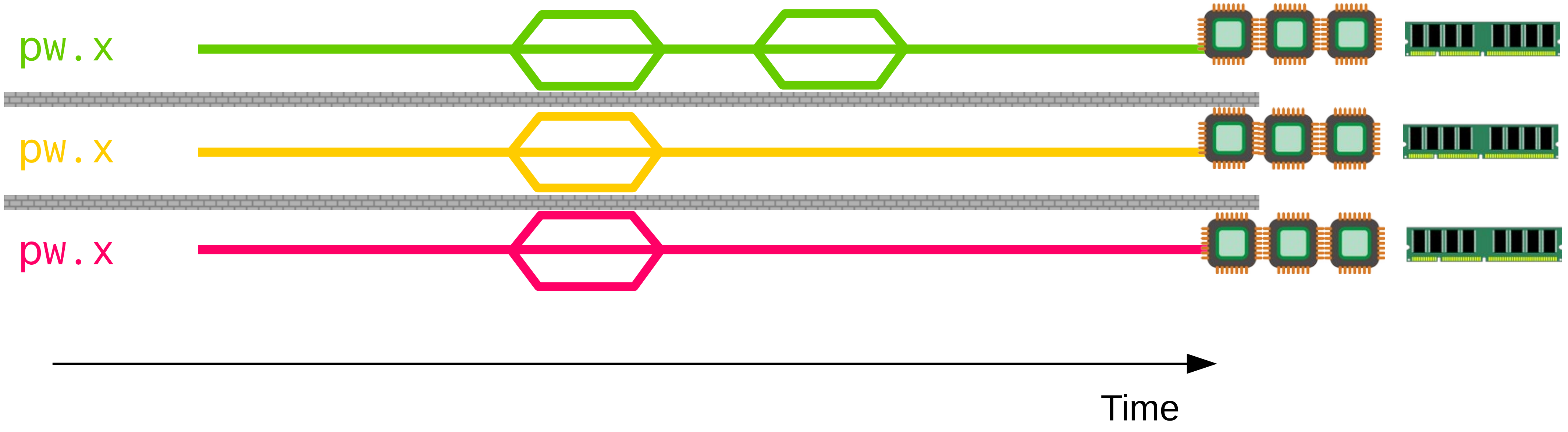
mpirun -np 3 pw.x



Message Passing & OpenMP

Export OMP_NUM_THREADS=3

mpirun -np 3 pw.x



Data decomposition

- From P. Giannozzi, day 2

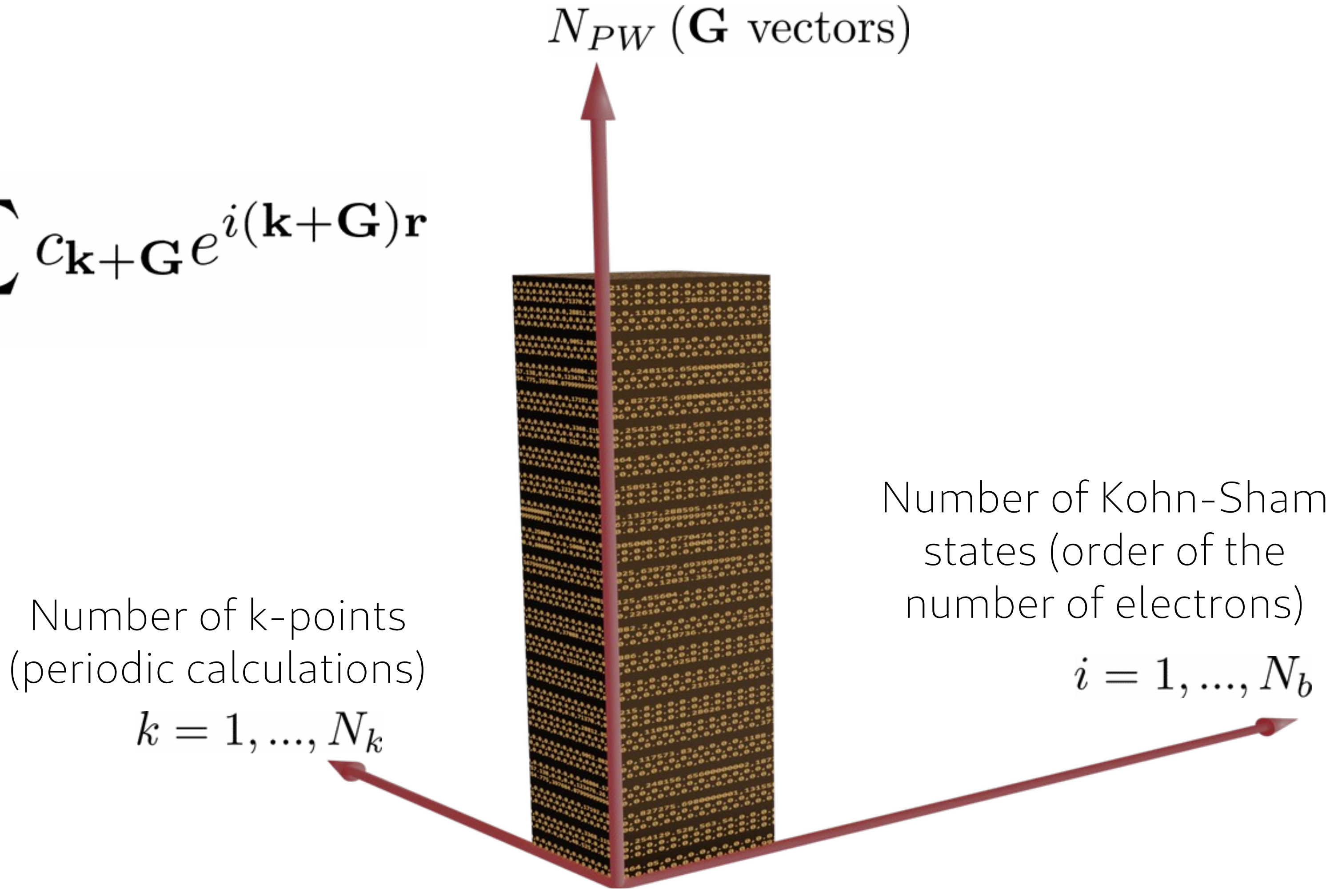
$$\psi_i(\mathbf{r}) = \frac{1}{\sqrt{V}} \sum_{\mathbf{G}} c_{\mathbf{k}+\mathbf{G}} e^{i(\mathbf{k}+\mathbf{G})\cdot\mathbf{r}}, \qquad \frac{\hbar^2}{2m}|\mathbf{k}+\mathbf{G}|^2 \le E_{cut}$$

The code computes an unsymmetrized charge density

$$\tilde{n}(\mathbf{r}) = \sum_{\mathbf{k}\in IBZ} \sum_{v} w_{\mathbf{k}} \left|\psi_{\mathbf{k},v}(\mathbf{r})\right|^2$$

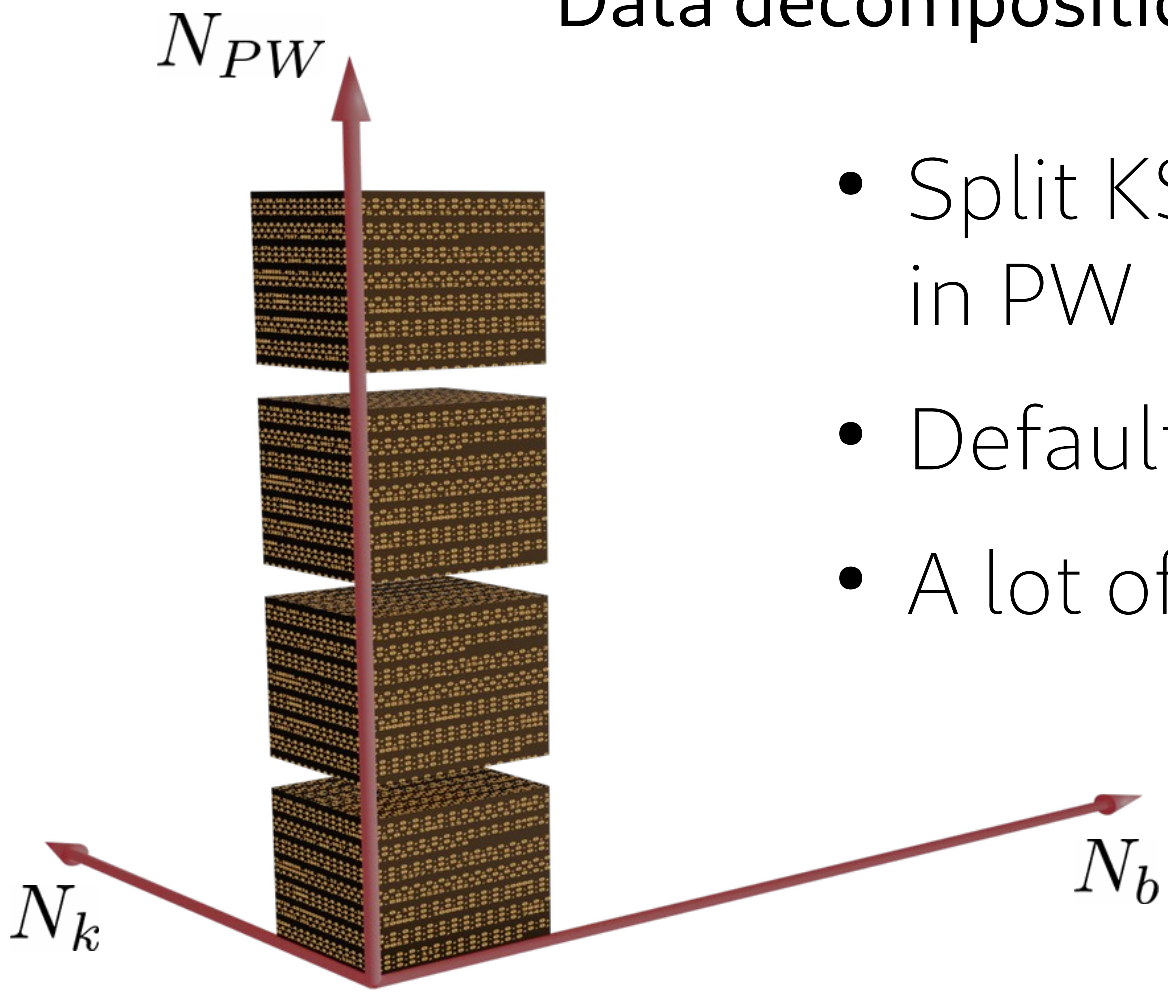
Data decomposition

$$\psi_{ik}(\mathbf{r}) = \frac{1}{\sqrt{V}} \sum_{\mathbf{G}} c_{\mathbf{k}+\mathbf{G}} e^{i(\mathbf{k}+\mathbf{G})\mathbf{r}}$$



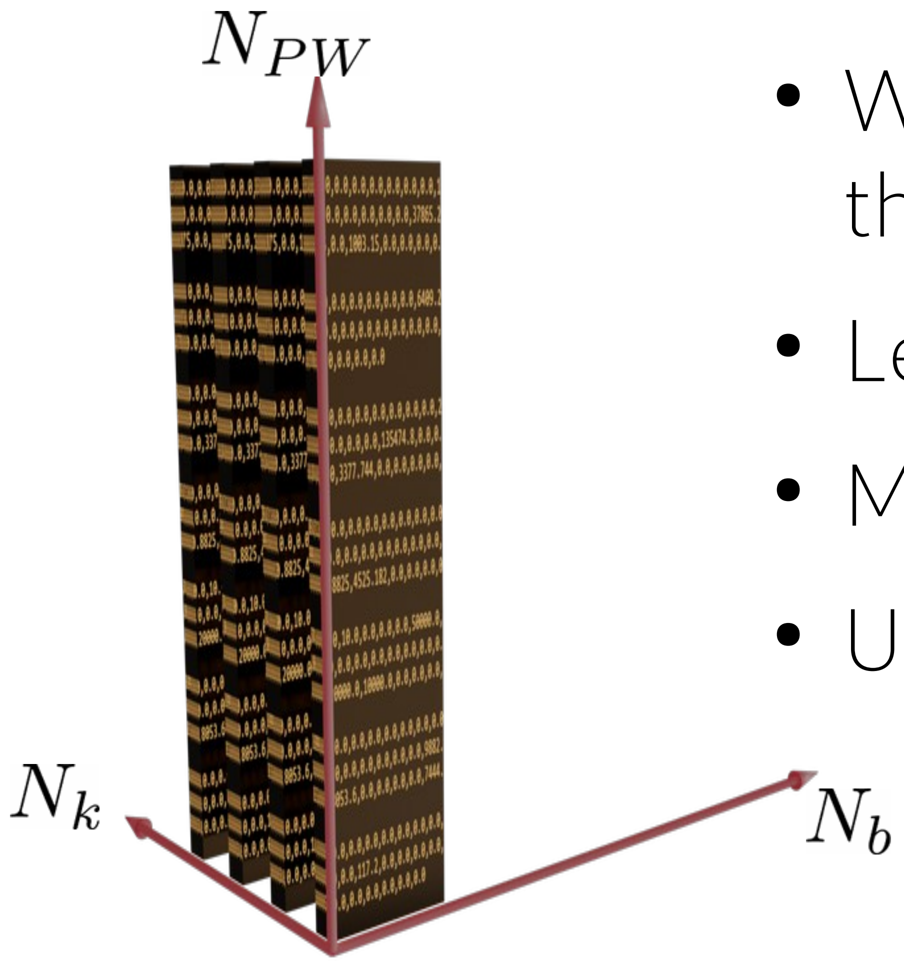
N_{PW} ($\mathbf{G}$ vectors)
Number of k-points (periodic calculations)
$k = 1, ..., N_k$
Number of Kohn-Sham states (order of the number of electrons)
$i = 1, ..., N_b$

Data decomposition



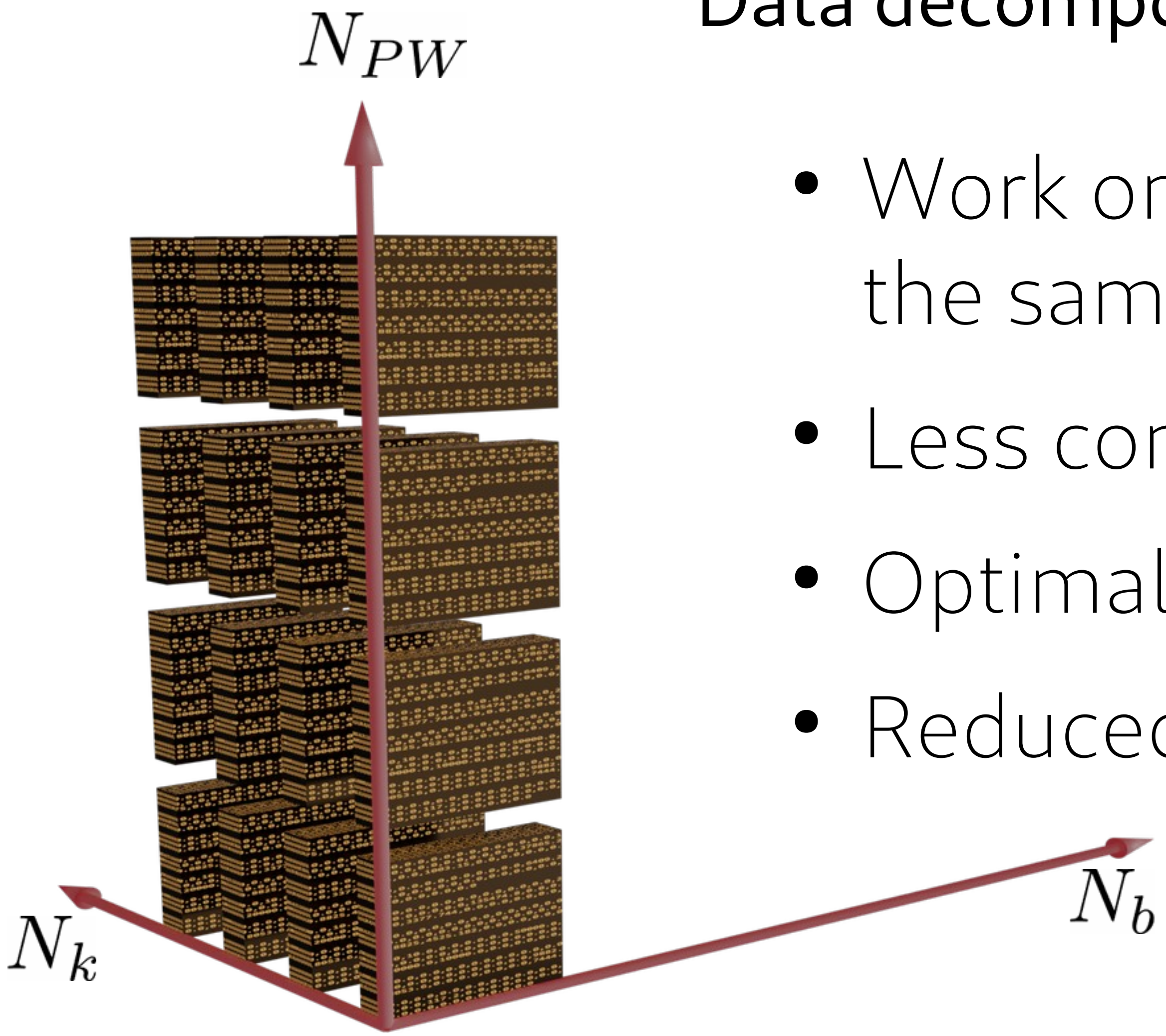
- Split KS states expansion in PW
- Default
- A lot of messages

Data decomposition



- Work on multiple k-points at the same time
- Less communication
- More memory
- Unbalance

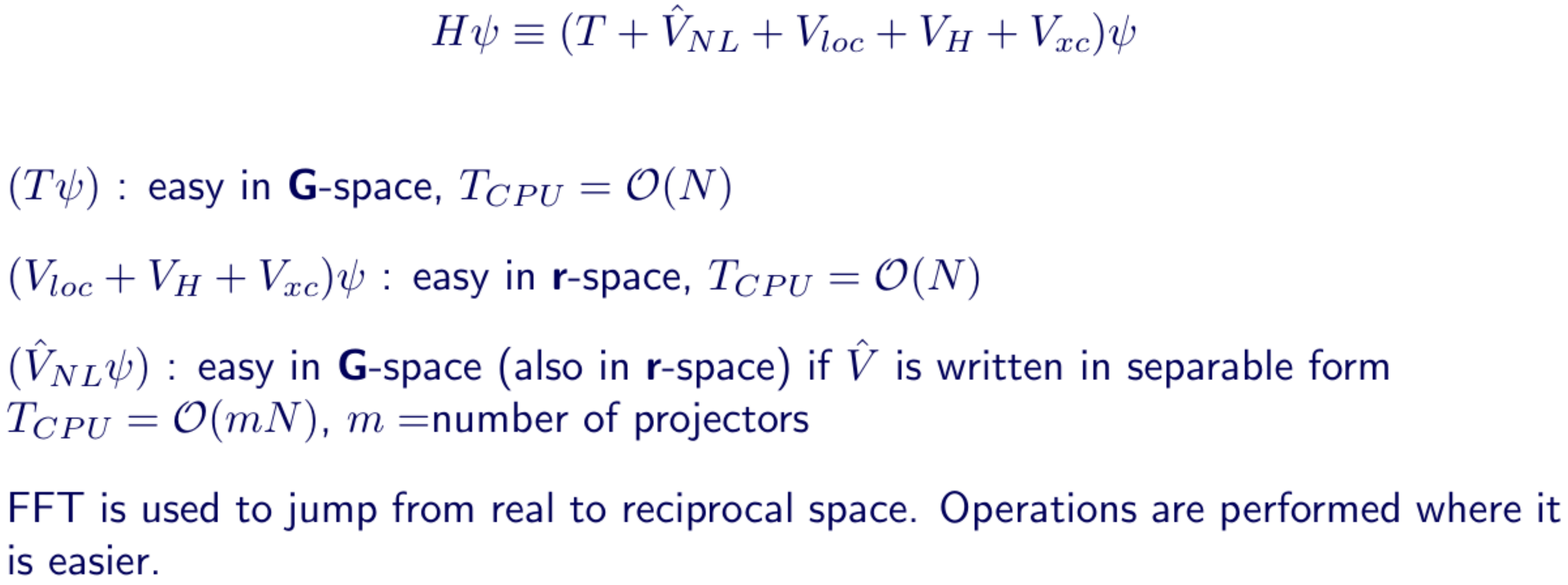
Data decomposition



- Work on multiple k-points at the same time and split WFs.
- Less communication
- Optimal memory usage
- Reduced unbalance

Workload decomposition

Again from P. Giannozzi, day 2



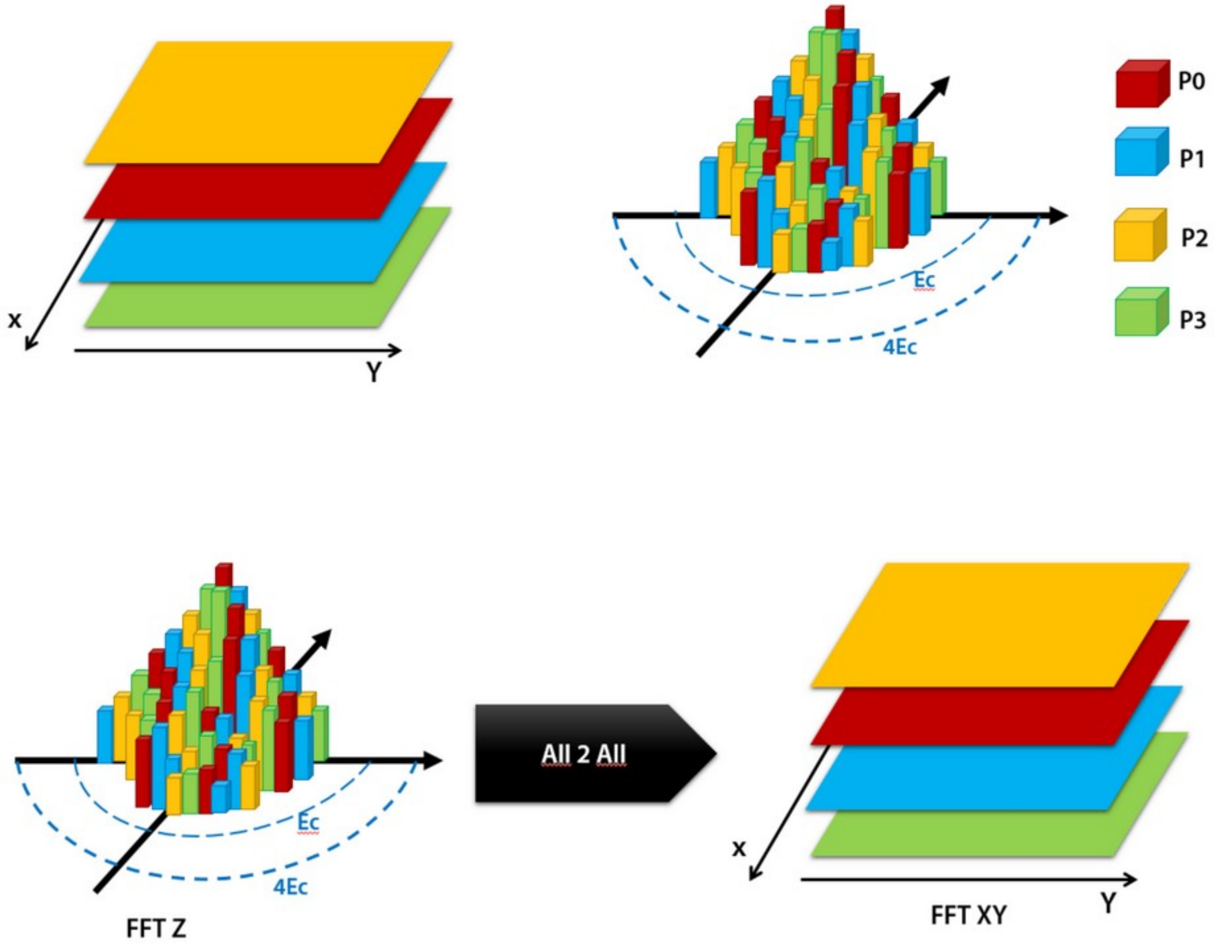
$$H\psi \equiv (T + \hat{V}_{NL} + V_{loc} + V_H + V_{xc})\psi$$

$(T\psi)$: easy in **G**-space, $T_{CPU} = \mathcal{O}(N)$

$(V_{loc} + V_H + V_{xc})\psi$: easy in **r**-space, $T_{CPU} = \mathcal{O}(N)$

$(\hat{V}_{NL}\psi)$: easy in **G**-space (also in **r**-space) if $\hat{V}$ is written in separable form
$T_{CPU} = \mathcal{O}(mN)$, m =number of projectors

FFT is used to jump from real to reciprocal space. Operations are performed where it is easier.



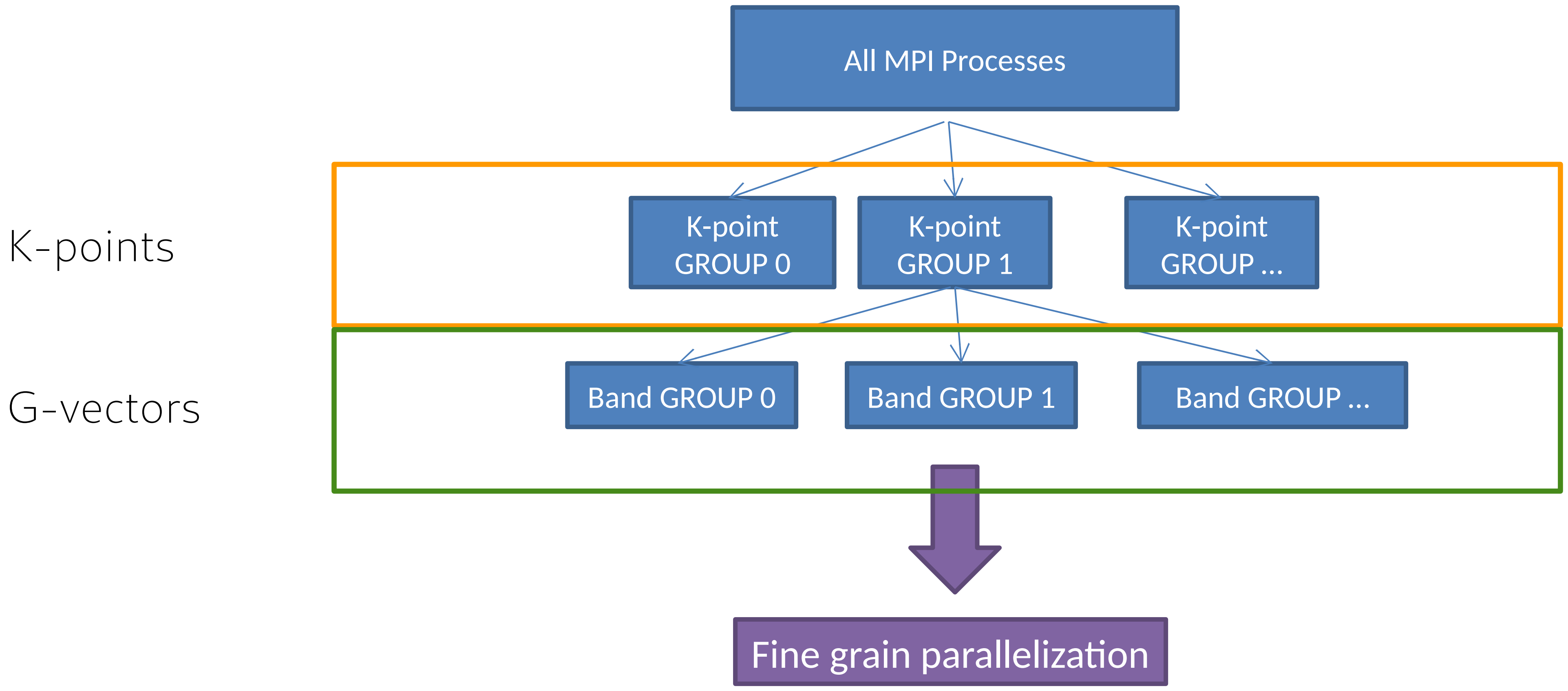
Eric Pascolo, master thesis

$$\begin{tikzcd}
F(Gx, Gy, Gz) \arrow[d, shift right] \arrow[d, phantom, "\text{1D FFT}" right, xshift=2em] \\
F(Gx, Gy, Rz) \arrow[u, shift right] \arrow[d, shift right] \arrow[d, phantom, "\text{Many MPI messages}" right, xshift=2em] \\
F(Gx, Gy, Rz) \arrow[u, shift right] \arrow[d, shift right] \arrow[d, phantom, "\text{2D FFT}" right, xshift=2em] \\
F(Rx, Ry, Rz) \arrow[u, shift right]
\end{tikzcd}$$

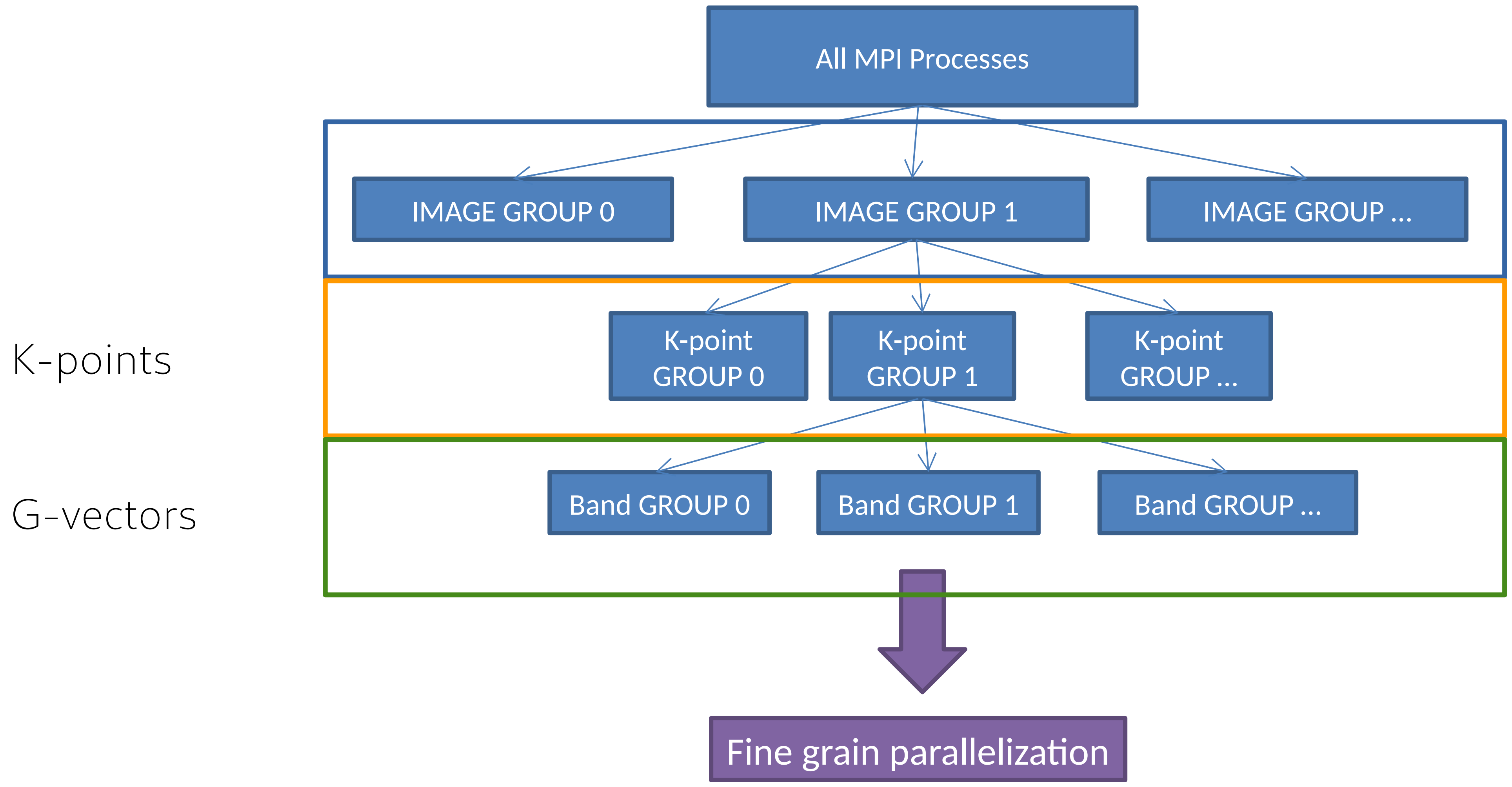
An example: BCC Fe

```
Parallelization info
--------------------
sticks:   dense  smooth     PW     G-vecs:    dense   smooth      PW
Min         111      37     13                 2974      573     129
Max         112      38     14                 2976      575     132
Sum        1781     599    219                47597     9189    2093

Using Slab Decomposition

Dense  grid:    47597 G-vectors     FFT dimensions: (  50,   50,   50)
Smooth grid:     9189 G-vectors     FFT dimensions: (  30,   30,   30)
```

Parallel levels



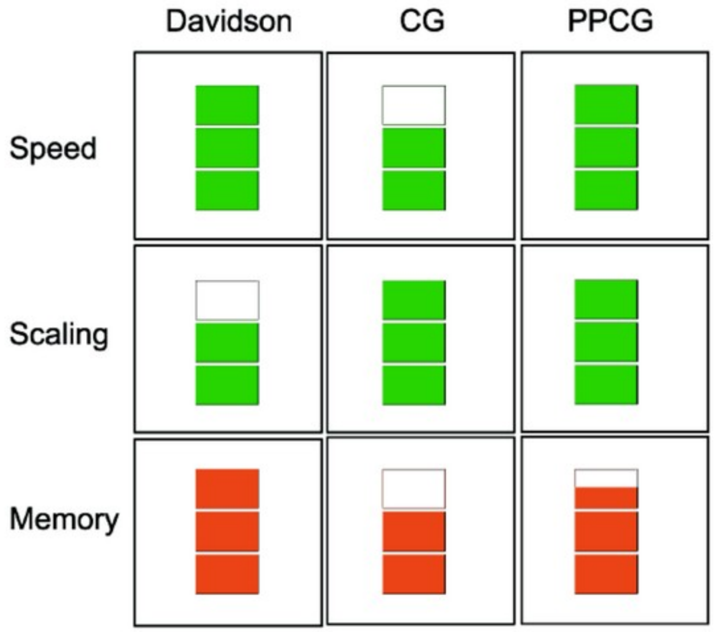
Parallel levels



Parallel Diagonalization

- Diagonalization options
  - Davidson
  - CG
  - PPCG
  - ParO
  - …

$$H_{KS}\psi_j$$



Picture from Anoop Chandran

Parallel Diagonalization

- Diagonalization options
  - <u>Davidson</u>
  - CG
  - PPCG
  - ParO
  - ...

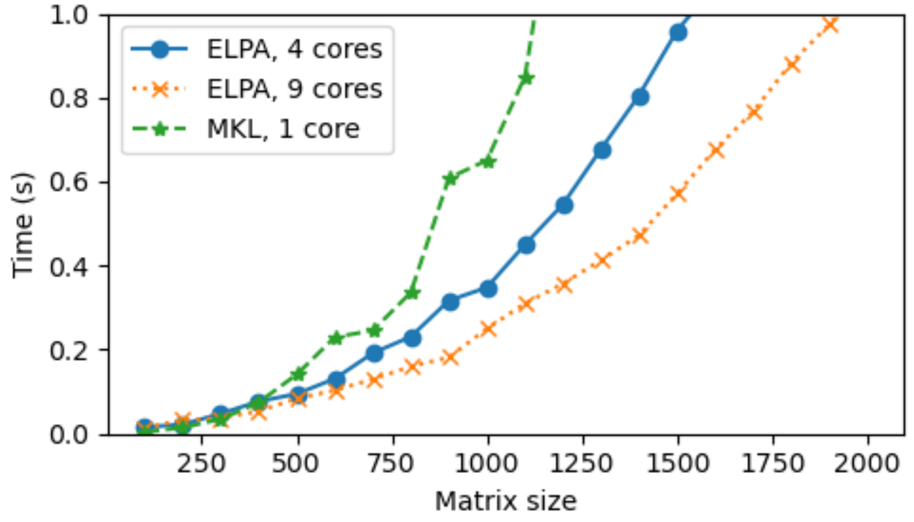
$$\left\{\left|\psi_i^{(n)}\right\rangle, \varepsilon_i^{(n)}\right\}$$

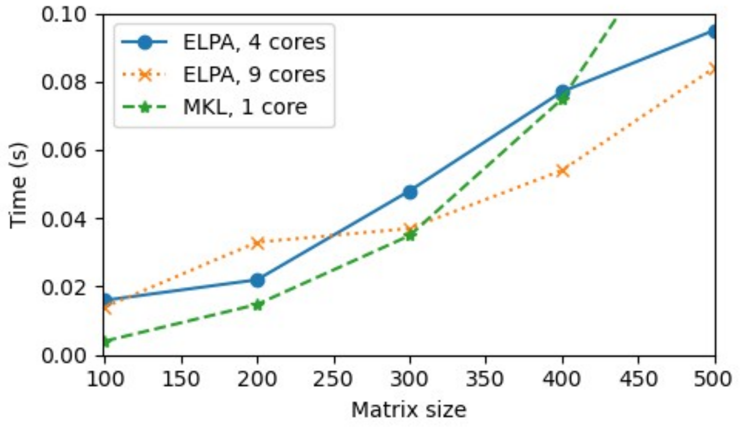
$$\tilde{H}_{ij} = \left\langle \psi_i^{(n)} \left| H_{KS} \right| \psi_j^{(n)} \right\rangle, \quad \tilde{S}_{ij} = \left\langle \psi_i^{(n)} | S | \psi_j^{(n)} \right\rangle$$

$$\left|\tilde{\psi}_i^{(n)}\right\rangle = \left(H_{diag} - \varepsilon_i S_{diag}\right)^{-1} \left(H_{KS} - \varepsilon_i S\right) \left|\psi_i^{(n)}\right\rangle$$

$$\mathbf{H}\mathbf{v} = \varepsilon \mathbf{S}\mathbf{v}$$

Trend of Parallel Diagonalization





- Results for all eigen-states.
- Additional communications in parallel Davidson.

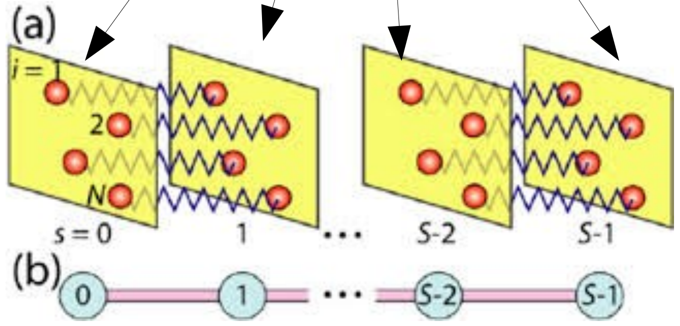
Fine Grained Parallelization

- OpenMP
  - In the code and in the libraries.
  - Only when MPI is saturated.
  - Generally no more than 8 threads.
  - Don’t forget about it on HPC systems!
- Multithreading
  - Generally not useful

Image parallelism

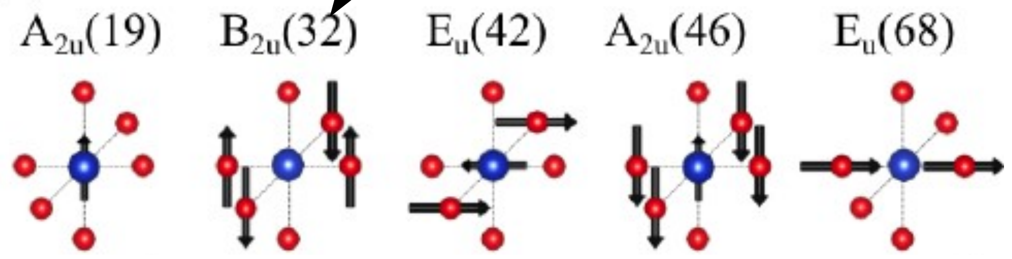
Nudged Elastic Band

Images



PHonon (linear response)

Irreducible modes



Images from
DOI: 10.1016/j.cpc.2007.09.011
DOI: 10.1103/PhysRevB.93.174419

Image parallelism

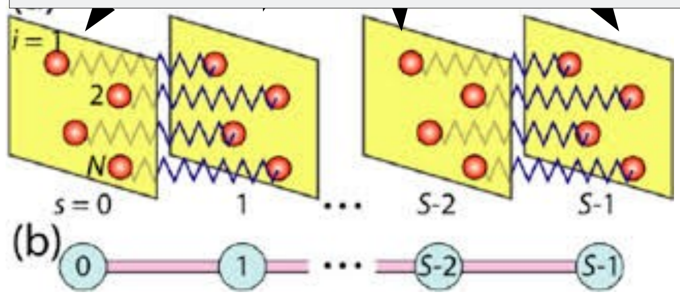
Nudged Elastic Band

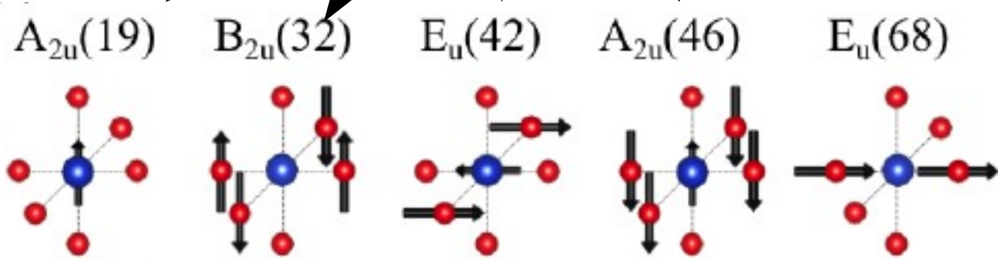
PHonon (linear response)

Images

Irreducible modes

Abstract layer for embarrassingly parallel tasks





Images from
DOI: 10.1016/j.cpc.2007.09.011
DOI: 10.1103/PhysRevB.93.174419

Image parallelism

Distribution of images

```
mpirun neb.x -nimage I -inp neb.in > neb.out
```

Output:

```
     path-images division:  nimage    =    I
```

Max value: total number of images in the simulation.

Constraints:

- Depend on code using this “abstract” parallelism

Tentative optimal value: `nimage` = max possible value

K-points aka Pools

```
mpirun pw.x -npool X -inp pw.in > pw.out
```

Output:

```
     K-points division:     npool     =       X
```

Distribute k points among X pools of MPI procs.
Max value: n(k)

Constraints:
- at least 1 k point per pool
- Must be a divisor of the total number of processes

Tentative optimal value: npool = max(n(k))

Parallel diagonalization

```
mpirun pw.x -npool X -ndiag Y -inp pw.in > pw.out
```

Distribute and parallelize matrix diagonalization and matrix-matrix multiplications needed in iterative diagonalization (pw.x) or orthonormalization(cp.x).

Max value: n(MPI)/X
Constraints:
- Must be square
- Must be smaller than band-group size

Tentative optimal value:
- Use it for inputs with more than 100 KS;
- depends on many architectural parameters

Output

```
Subspace diagonalization (size of sub-
group: sqrt(Y)*sqrt(Y) procs)
```

Finding the right balance

- Pools:
  - Very effective, low communication
  - Memory hungry!
- G vectors:
  - Lower memory footprint
  - More communication
- OpenMP:
  - Practically no memory duplication
  - When MPI is saturated
- Diagonalization method:
  - Davidson: faster, more memory
  - CG: slower, less memory

Libraries

- A few libraries you may need

OpenBLAS

FFTW

ScaLAPACK

ELPA

Intel® oneAPI Math Kernel Library

AMD Optimizing CPU Libraries (AOCL)

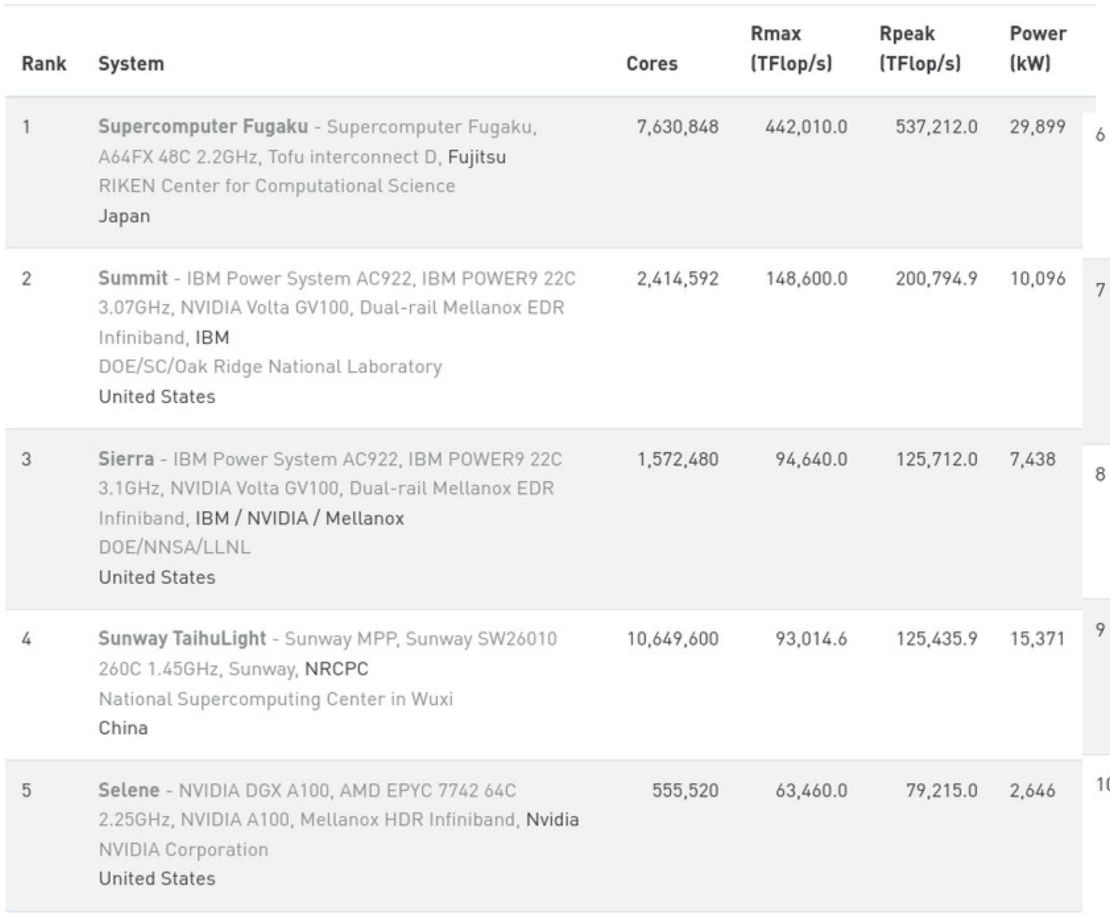
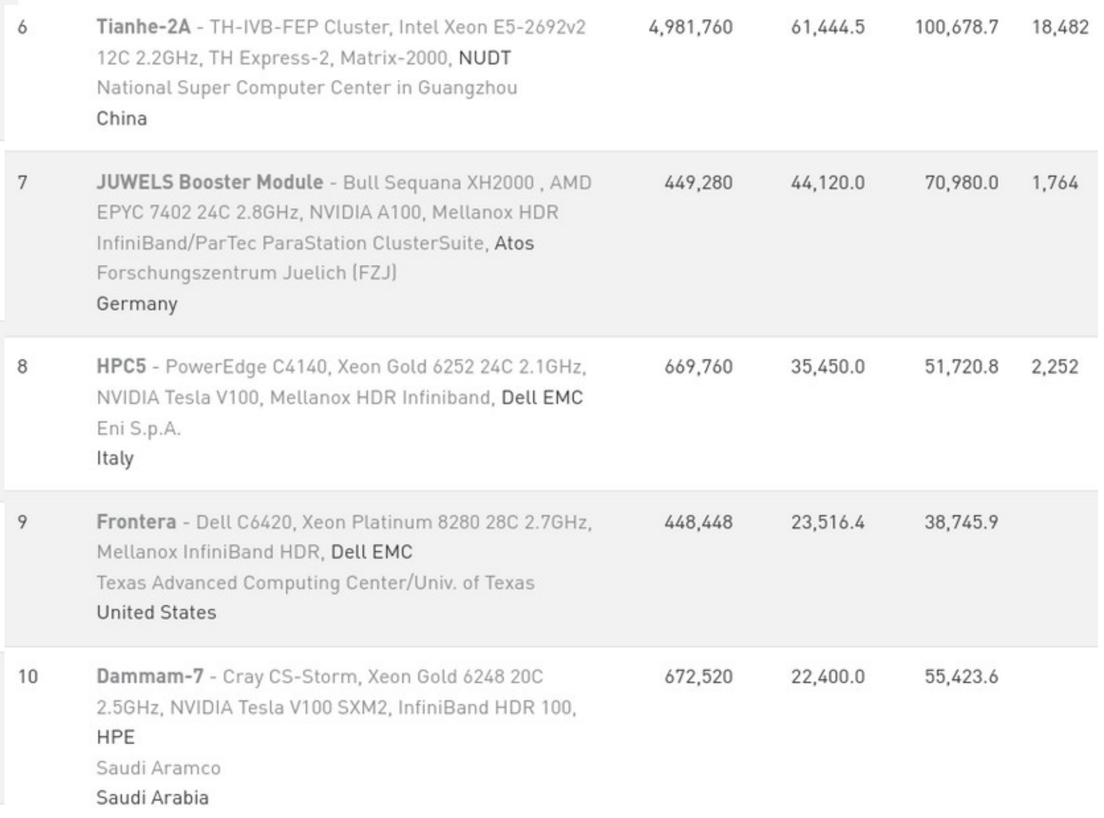
Input/Output

A parallel filesystem is essentially a parallel application.

- Different performance have different cost:
  - Home directory
  - Long term storage
  - Scratch space

Always set the `outdir` folder in input to a fast scratch space.

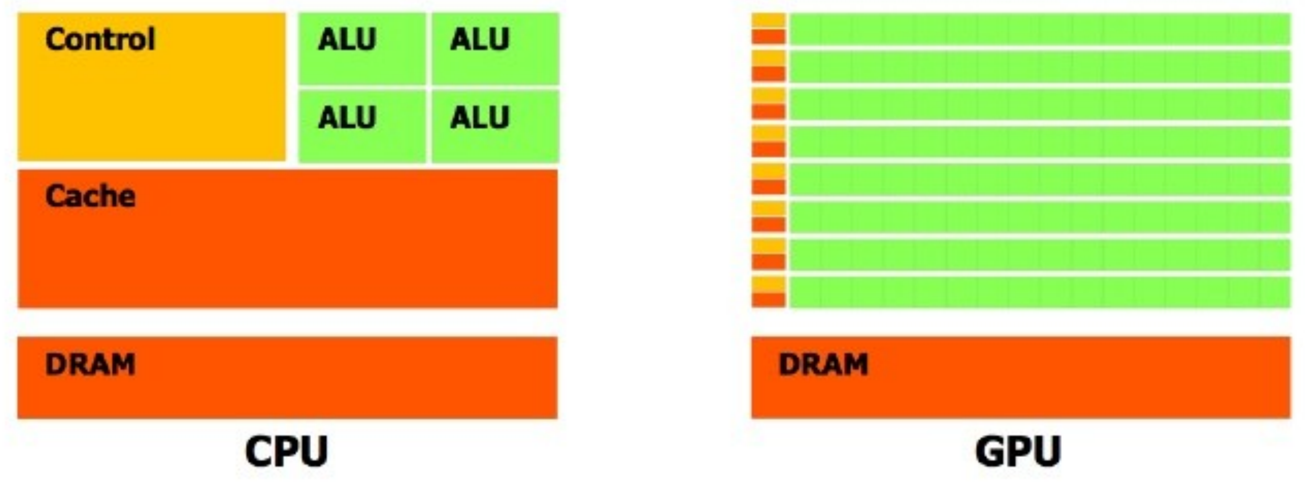
GPU Acceleration

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
6	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
7	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764
8	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC Eni S.p.A. Italy	669,760	35,450.0	51,720.8	2,252
9	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	
10	**Dammam-7** - Cray CS-Storm, Xeon Gold 6248 20C 2.5GHz, NVIDIA Tesla V100 SXM2, InfiniBand HDR 100, HPE Saudi Aramco Saudi Arabia	672,520	22,400.0	55,423.6	

https://top500.org/lists/top500/2020/11/

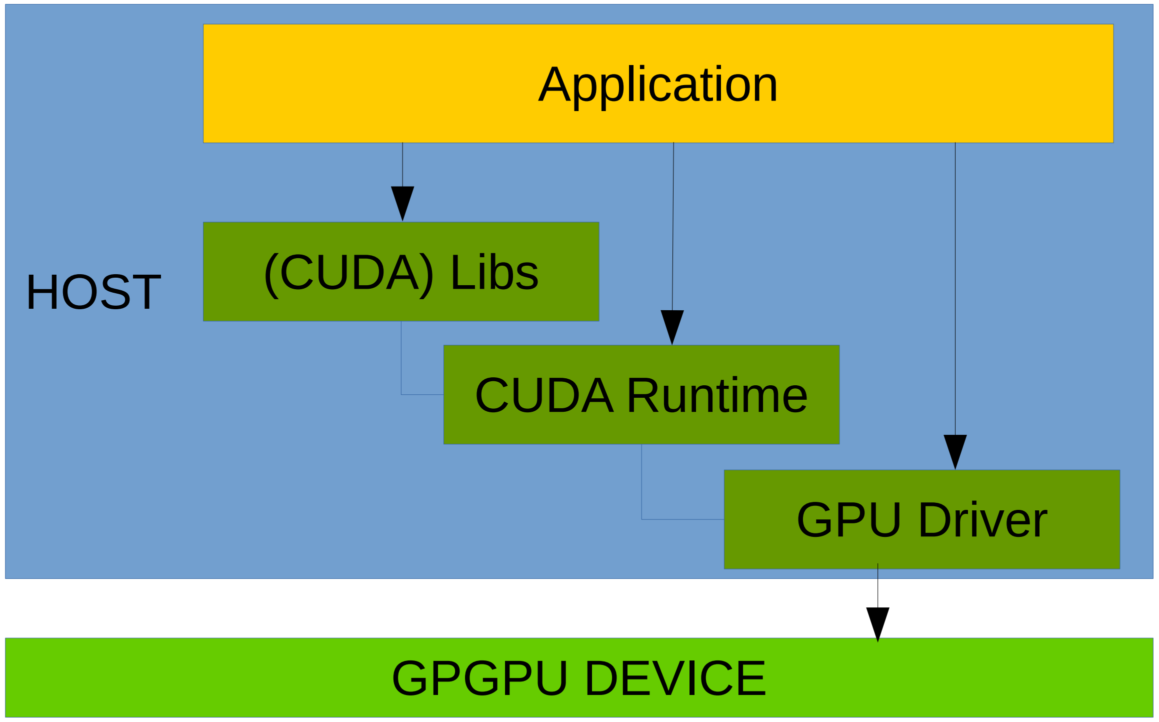
Different architecture



More transistors devoted to data processing
(but less optimized memory access and speculative execution)

CUDA Programming

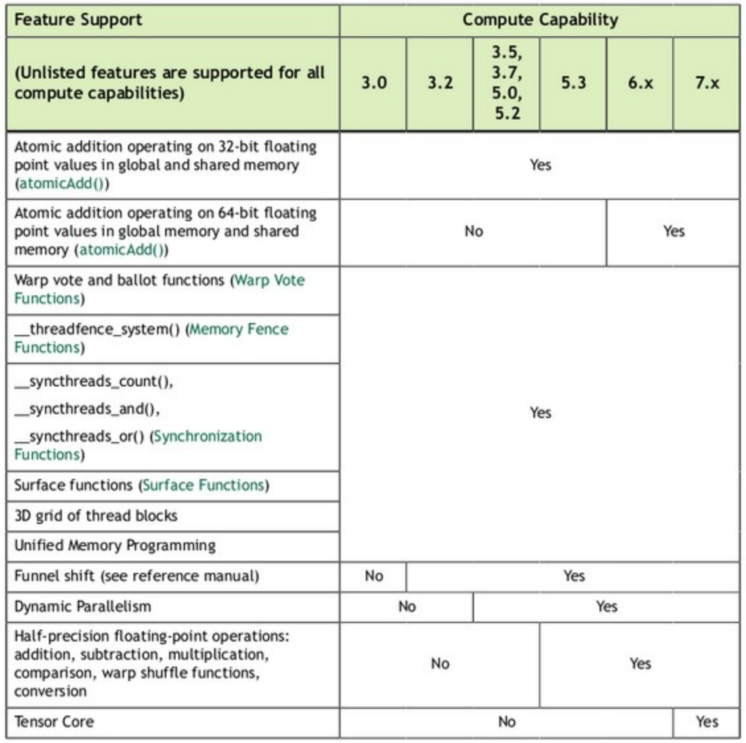
In order to compile QE-GPU you'll need to know a bit about *CUDA* ...



CUDA Programming

The *compute capabilities* codify the features and specifications of the target device.

- Tesla K40: 3.5
- Tesla K80: 3.7
- Tesla P100: 6.0
- Tesla V100: 7.0
- Tesla A100: 8.0



Feature Support	Compute Capability					
(Unlisted features are supported for all compute capabilities)	3.0	3.2	3.5, 3.7, 5.0, 5.2	5.3	6.x	7.x
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())	Yes	Yes	Yes	Yes	Yes	Yes
Atomic addition operating on 64-bit floating point values in global memory and shared memory (atomicAdd())	No	No	No	No	Yes	Yes
Warp vote and ballot functions (Warp Vote Functions)	Yes	Yes	Yes	Yes	Yes	Yes
__threadfence_system() (Memory Fence Functions)	Yes	Yes	Yes	Yes	Yes	Yes
__syncthreads_count(), __syncthreads_and(), __syncthreads_or() (Synchronization Functions)	Yes	Yes	Yes	Yes	Yes	Yes
Surface functions (Surface Functions)	Yes	Yes	Yes	Yes	Yes	Yes
3D grid of thread blocks	Yes	Yes	Yes	Yes	Yes	Yes
Unified Memory Programming	Yes	Yes	Yes	Yes	Yes	Yes
Funnel shift (see reference manual)	No	Yes	Yes	Yes	Yes	Yes
Dynamic Parallelism	No	No	Yes	Yes	Yes	Yes
Half-precision floating-point operations: addition, subtraction, multiplication, comparison, warp shuffle functions, conversion	No	No	No	Yes	Yes	Yes
Tensor Core	No	No	No	No	No	Yes

From CUDA C Programming guide v10

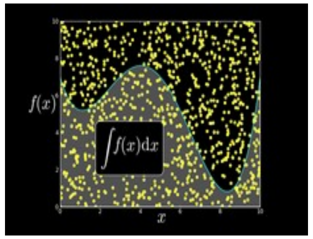
Compilers

The accelerated version requires
NVIDIA HPC SDK

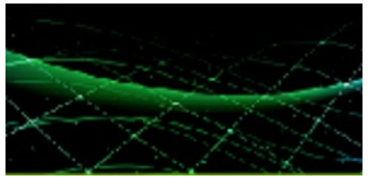
CUDA FORTRAN

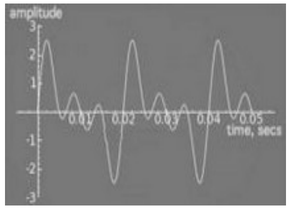
Freely available at
https://developer.nvidia.com/hpc-sdk

NVIDIA cuBLAS



NVIDIA cuRAND


NVIDIA cuSolver

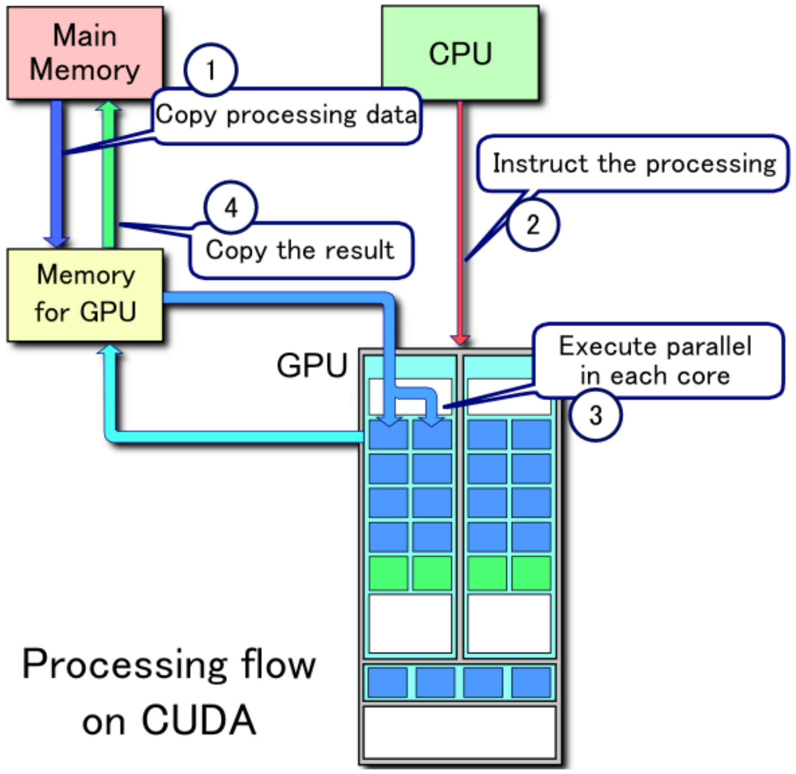


NVIDIA cuFFT

Execution flow

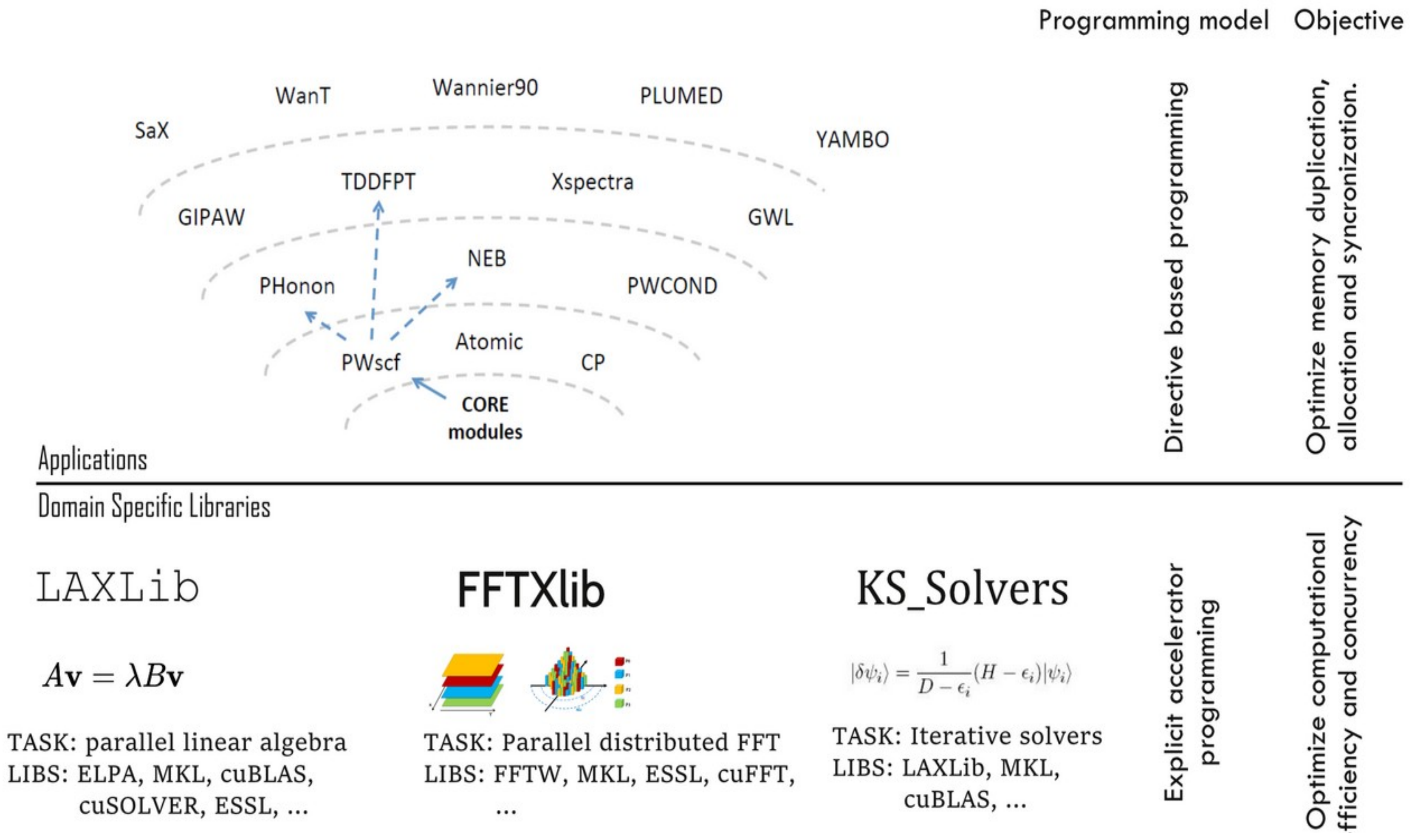
Typical code progression

1) Memory allocated on host and device
2) Data is transferred **from** the *Host* **to** the *Device*
3) Kernel is lunched by the Host on the Device
4) Data is transferred **from** the *Device* **to** the *Host*.
5) Memory is deallocated.



From https://commons.wikimedia.org/wiki/File:CUDA_processing_flow_(En).PNG

Quantum ESPRESSO


Programming model
Objective
SaX
WanT
Wannier90
PLUMED
YAMBO
TDDFPT
Xspectra
GIPAW
GWL
NEB
PHonon
PWCOND
Atomic
PWscf
CP
CORE modules
Applications
Domain Specific Libraries
Directive based programming
Optimize memory duplication, allocation and syncronization.
Explicit accelerator programming
Optimize computational fficiency and concurrency
LAXLib
FFTXlib
KS_Solvers

$$A\mathbf{v} = \lambda B\mathbf{v}$$

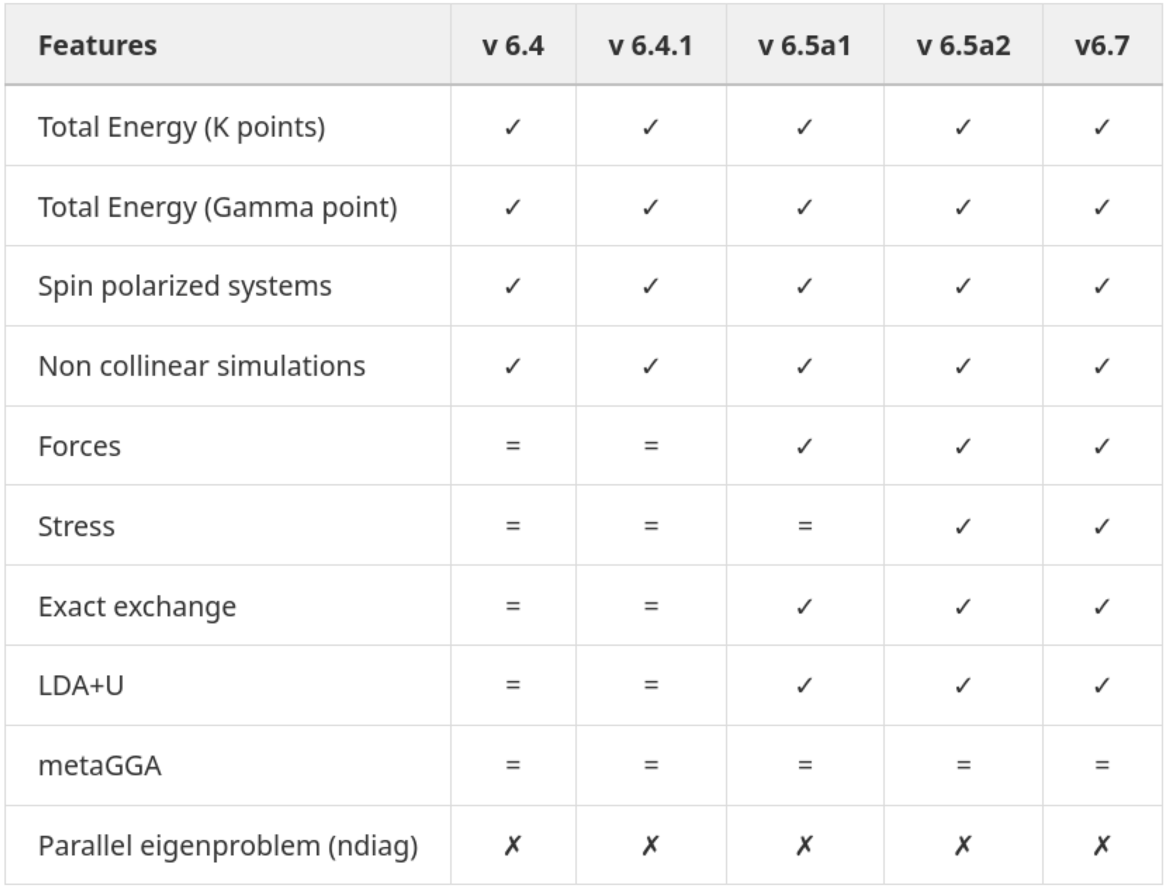
TASK: parallel linear algebra
LIBS: ELPA, MKL, cuBLAS, cuSOLVER, ESSL, ...

TASK: Parallel distributed FFT
LIBS: FFTW, MKL, ESSL, cuFFT, ...

$$|\delta\psi_i\rangle = \frac{1}{D-\epsilon_i}\langle H-\epsilon_i\rangle|\psi_i\rangle$$

TASK: Iterative solvers
LIBS: LAXLib, MKL, cuBLAS, ...

Quantum ESPRESSO



Features	v 6.4	v 6.4.1	v 6.5a1	v 6.5a2	v6.7
Total Energy (K points)	✓	✓	✓	✓	✓
Total Energy (Gamma point)	✓	✓	✓	✓	✓
Spin polarized systems	✓	✓	✓	✓	✓
Non collinear simulations	✓	✓	✓	✓	✓
Forces	=	=	✓	✓	✓
Stress	=	=	=	✓	✓
Exact exchange	=	=	✓	✓	✓
LDA+U	=	=	✓	✓	✓
metaGGA	=	=	=	=	=
Parallel eigenproblem (ndiag)	✗	✗	✗	✗	✗

The GPUs of Marconi100

NVIDIA Tesla V100

16GB memory per GPU

5120 CUDA Cores

Running at 1200-1300 MHz

Performance Ratio

- Let’s consider Marconi100@CINECA (ranked 11)

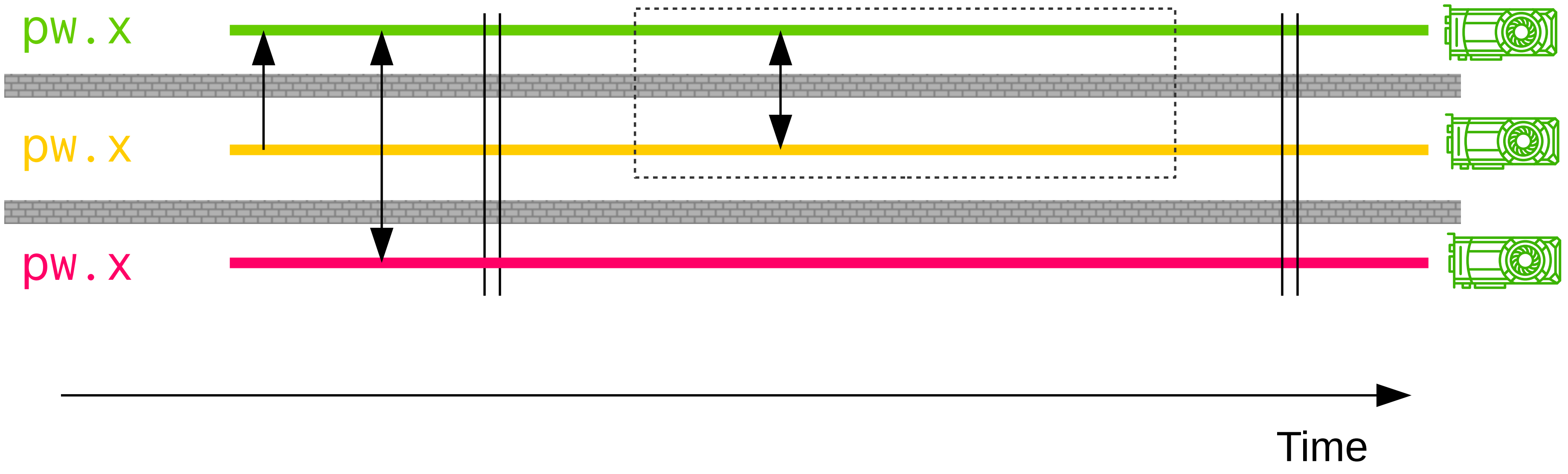
~700 GFlops

~31.2 TFlops

Icons By Misha Petrishchev, RU
and iconsmind.com, GB

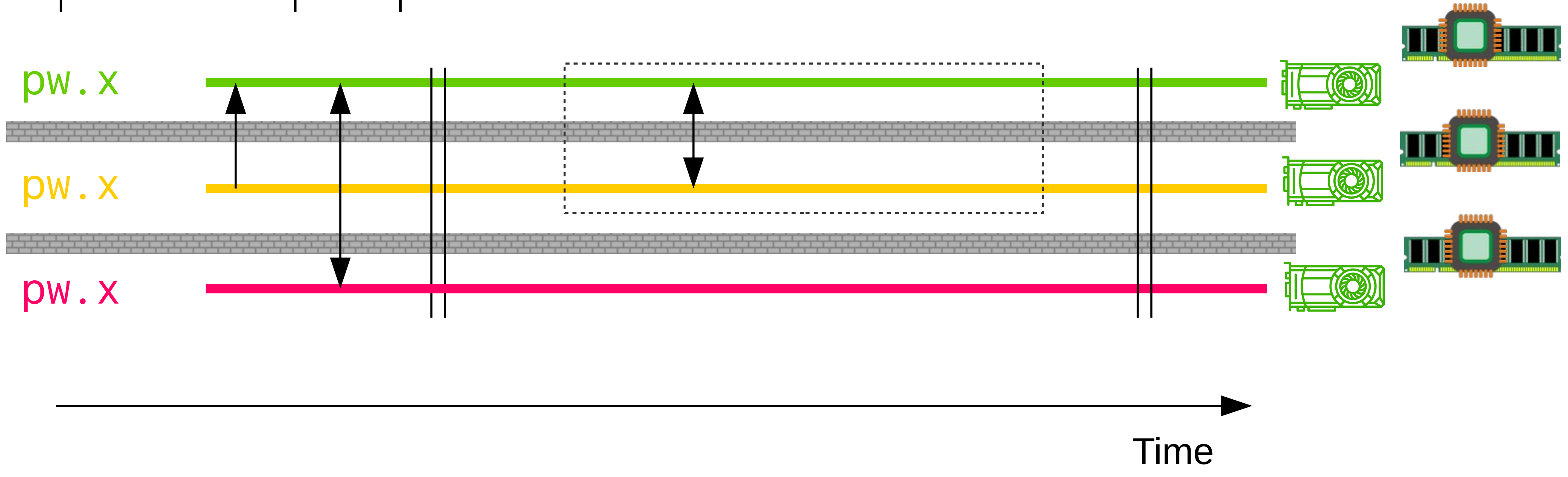
QE with GPU acceleration

```
mpirun -np 3 pw.x
```



QE with GPU acceleration

mpirun -np 3 pw.x



How to run the GPU code

1 GPU <-> 1 MPI

Fill the CPU with OpenMP threads

No parallel eigensolver (-ndiag 1) yet

K-point pools are great, but device memory is limited.

A few practical advices

Configure options:

```
--enable-openmp          compile for openmp execution if possible (default: no)
--enable-parallel        compile for parallel execution if possible (default: yes)

--enable-cuda-env-check=yes
                         The configure script will check CUDA installation
                         and report problems [default=no]
--with-cuda=PATH         prefix where CUDA is installed [default=no]
--with-cuda-cc=VAL       GPU architecture (Kepler: 35, Pascal: 60, Volta: 70) [default=35]
--with-cuda-runtime=VAL CUDA runtime (Pascal: 8+, Volta: 9+) [default=10.1]

--with-scalapack         (yes|no|intel) Use scalapack if available. Set to
                         "intel" to use Intel MPI and blacs (default: use openMPI)

--with-elpa-include      Specify full path ELPA include and modules headers (default: no)
--with-elpa-lib          Specify full path ELPA static or dynamic library (default: no)
--with-elpa-version      Specify ELPA API version (2015 for ELPA releases
                         2015.x and 2016.05; 2016 for ELPA releases 2016.11,
                         2017.x and 2018.05; default 2018 for ELPA releases
                         2018.11 and beyond)
```

A few practical advices

- Checking compilation options
  - Example for an Intel based platform…

```
MANUAL_DFLAGS  =
DFLAGS         =  -D__DFTI -D__MPI -D__SCALAPACK -D__ELPA_2016
FDFLAGS        = $(DFLAGS) $(MANUAL_DFLAGS)

[…]

MPIF90         = mpiifort
F90            = ifort
CC             = icc
F77            = ifort
[…]

BLAS_LIBS      =   -lmkl_intel_lp64  -lmkl_intel_thread -lmkl_core
```

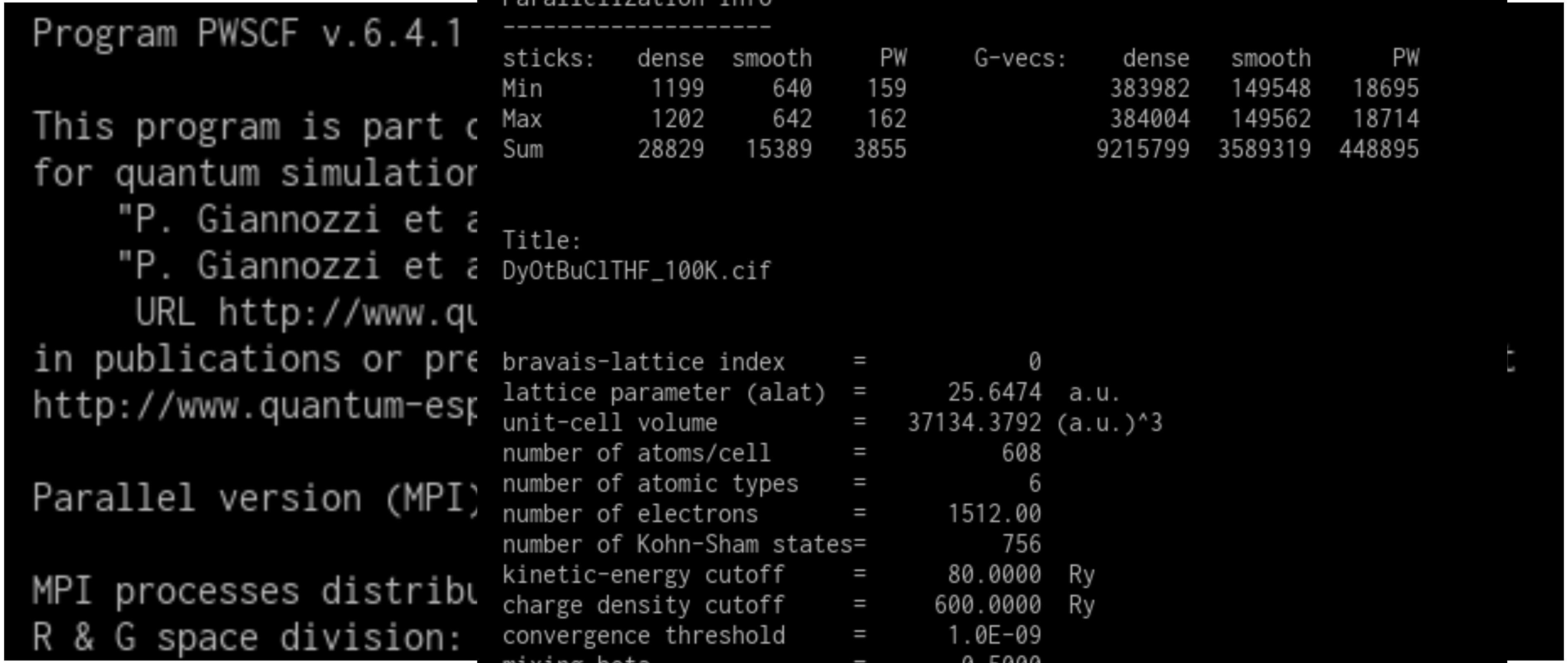
A few practical advices

- Checking compilation options
  - Example for an Intel based platform…
  - Example for a (NVIDIA) GPU platform

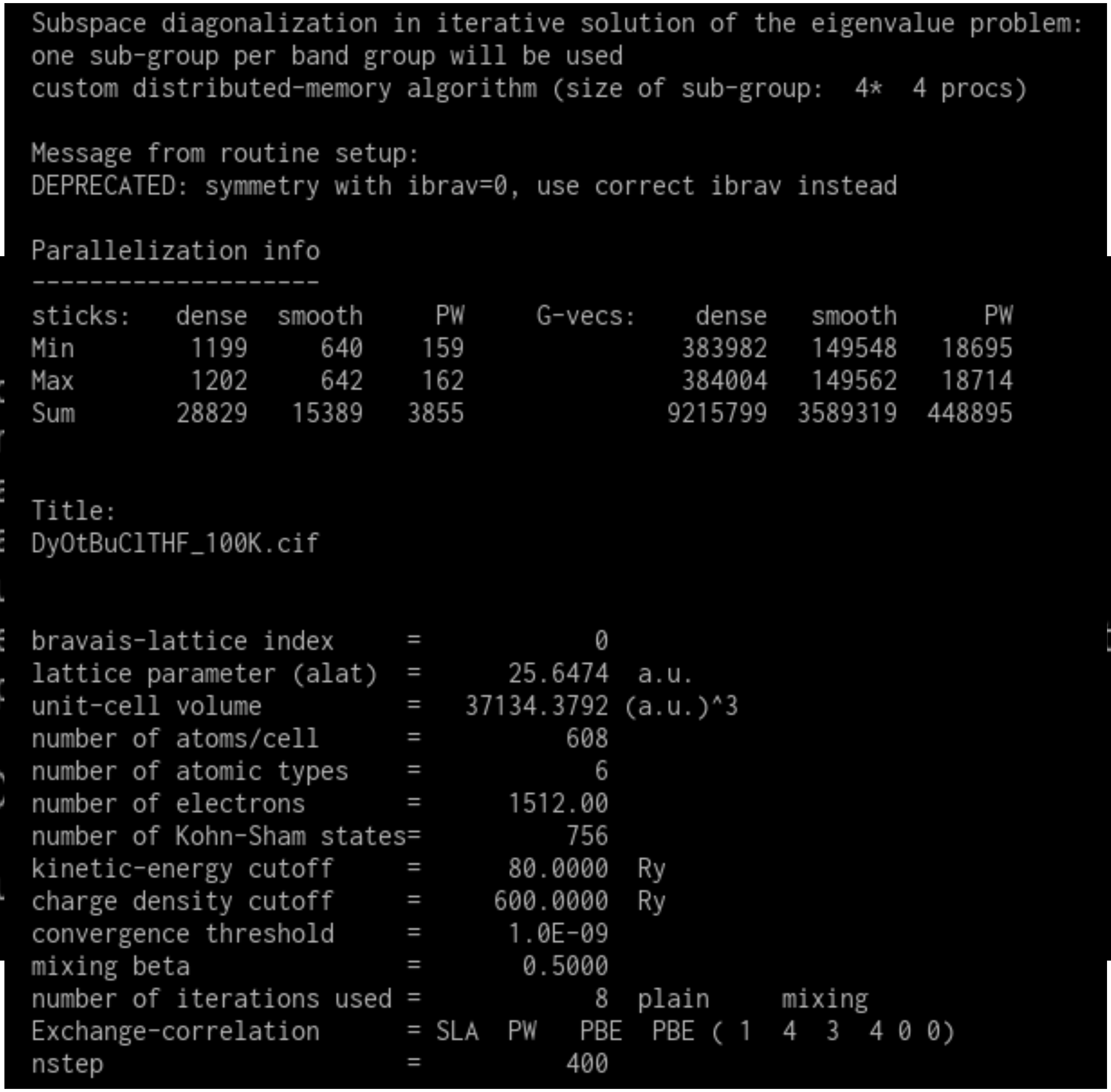
```
MANUAL_DFLAGS  =
DFLAGS         =  -D__PGI -D__CUDA -D__USE_CUSOLVER -D__FFTW -D__MPI
FDFLAGS        = $(DFLAGS) $(MANUAL_DFLAGS)

[…]

# GPU architecture (Kepler: 35, Pascal: 60, Volta: 70 )
GPU_ARCH=70

# CUDA runtime (Pascal: 8.0, Volta: 9.0)
CUDA_RUNTIME=11.0

# CUDA F90 Flags
CUDA_F90FLAGS=-Mcuda=cc70,cuda11.0 [...]
```

Something very bad...

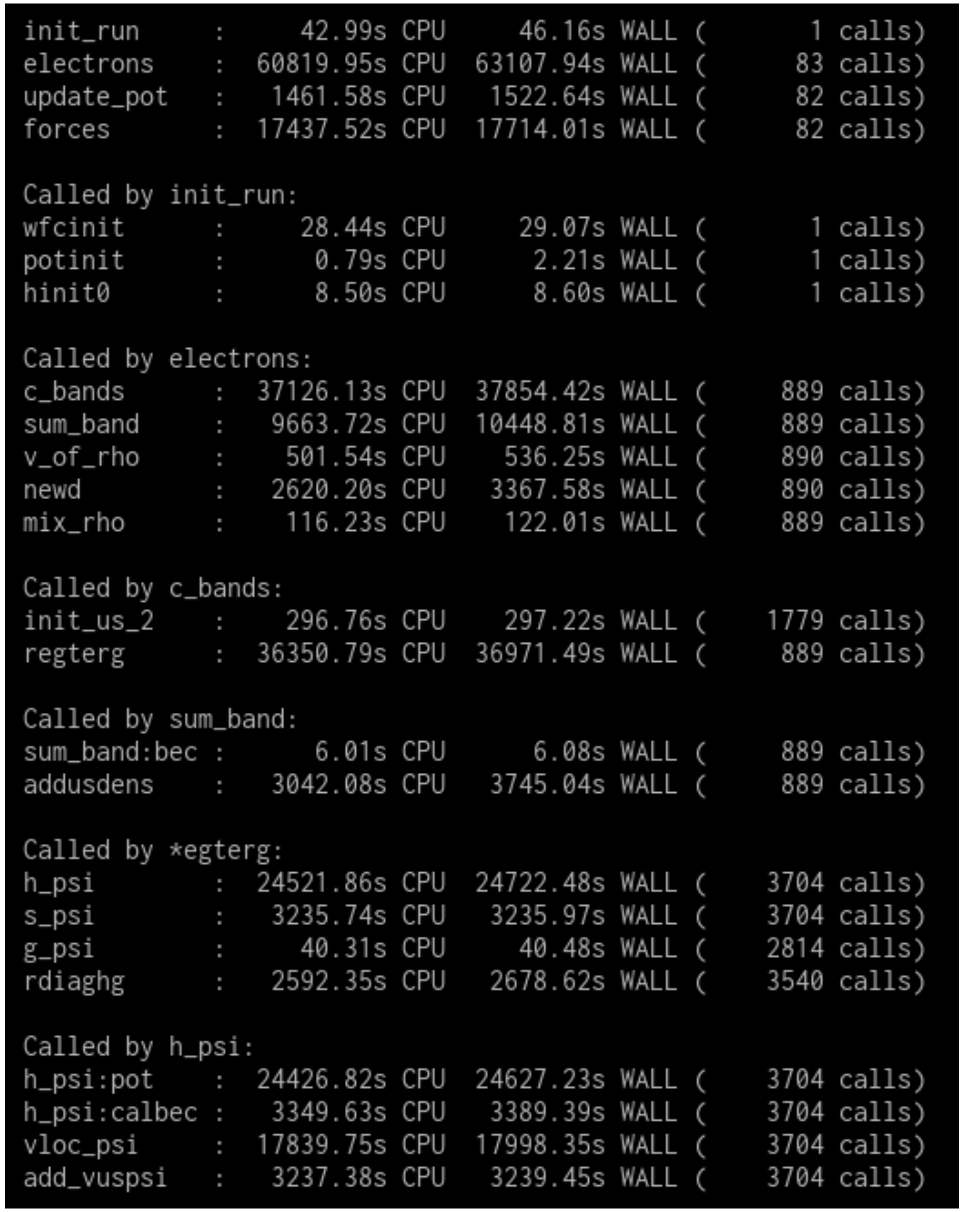
```
Program PWSCF v.6.2 starts on 29Nov2017 at 15:22:59

This program is part of the open-source Quantum ESPRESSO suite
for quantum simulation of materials; please cite
    "P. Giannozzi et al., J. Phys.:Condens. Matter 21 395502 (2009);
    "P. Giannozzi et al., J. Phys.:Condens. Matter 29 465901 (2017);
     URL http://www.quantum-espresso.org",
in publications or presentations arising from this work. More details at
http://www.quantum-espresso.org/quote

Parallel version (MPI & OpenMP), running on   15552 processor cores
Number of MPI processes:                     432
Threads/MPI process:                          36

MPI processes distributed on    12 nodes
K-points division:     npool     =       2
R & G space division:  proc/nbgrp/npool/nimage =     216
```



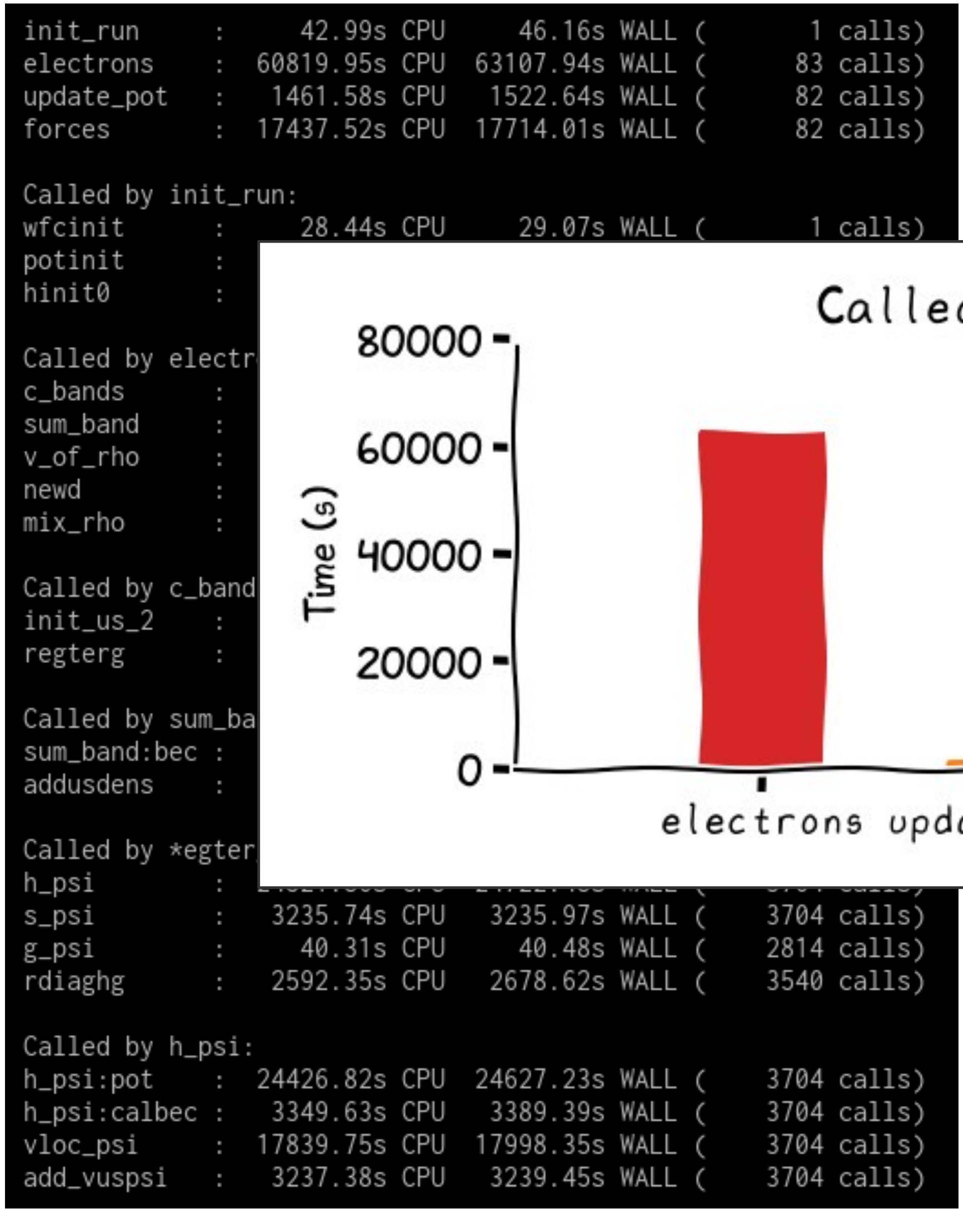
```
Program PWSCF v.6.4.1

This program is part c
for quantum simulatior
    "P. Giannozzi et a
    "P. Giannozzi et a
     URL http://www.qu
in publications or pre
http://www.quantum-esp

Parallel version (MPI)

MPI processes distribu
R & G space division:
```



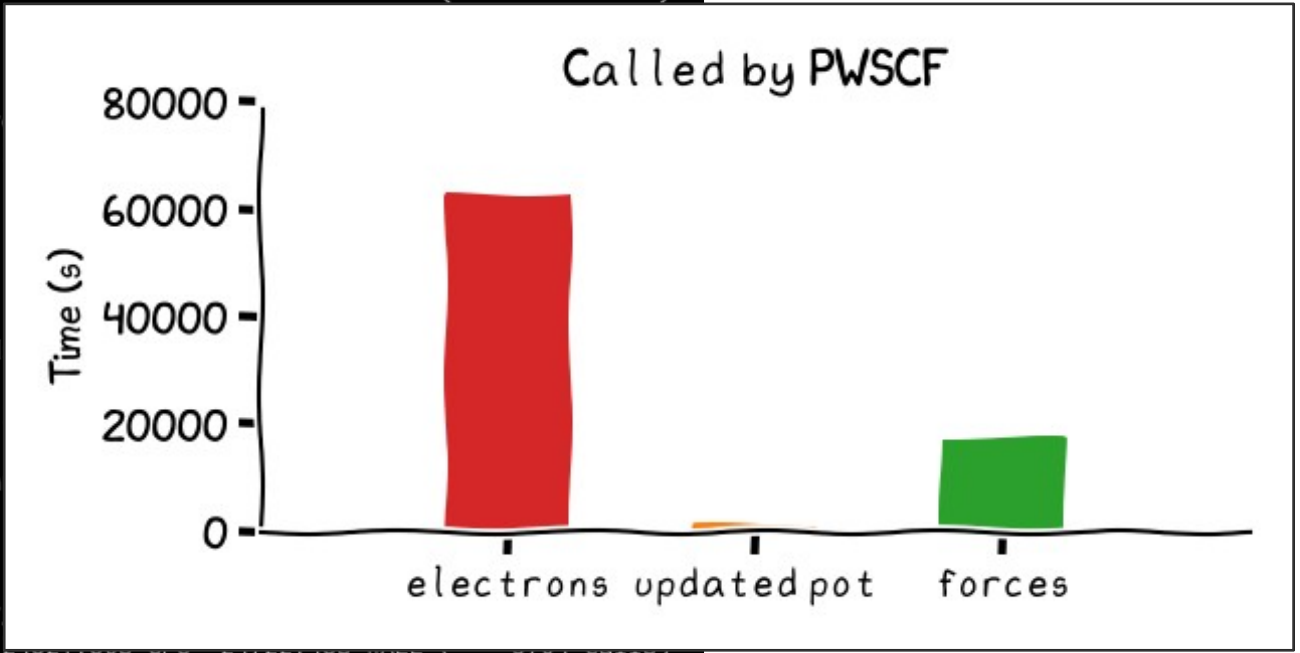
```
Subspace diagonalization in iterative solution of the eigenvalue problem:
one sub-group per band group will be used
custom distributed-memory algorithm (size of sub-group:  4*  4 procs)

Message from routine setup:
DEPRECATED: symmetry with ibrav=0, use correct ibrav instead

Parallelization info
--------------------
sticks:   dense  smooth     PW     G-vecs:    dense   smooth      PW
Min        1199     640    159               383982   149548   18695
Max        1202     642    162               384004   149562   18714
Sum       28829   15389   3855              9215799  3589319  448895


Title:
DyOtBuClTHF_100K.cif


bravais-lattice index     =            0
lattice parameter (alat)  =      25.6474  a.u.
unit-cell volume          =   37134.3792 (a.u.)^3
number of atoms/cell      =          608
number of atomic types    =            6
number of electrons       =      1512.00
number of Kohn-Sham states=          756
kinetic-energy cutoff     =      80.0000  Ry
charge density cutoff     =     600.0000  Ry
convergence threshold     =      1.0E-09
mixing beta               =       0.5000
number of iterations used =            8  plain     mixing
Exchange-correlation      = SLA  PW   PBE  PBE ( 1  4  3  4 0 0)
nstep                     =          400
```

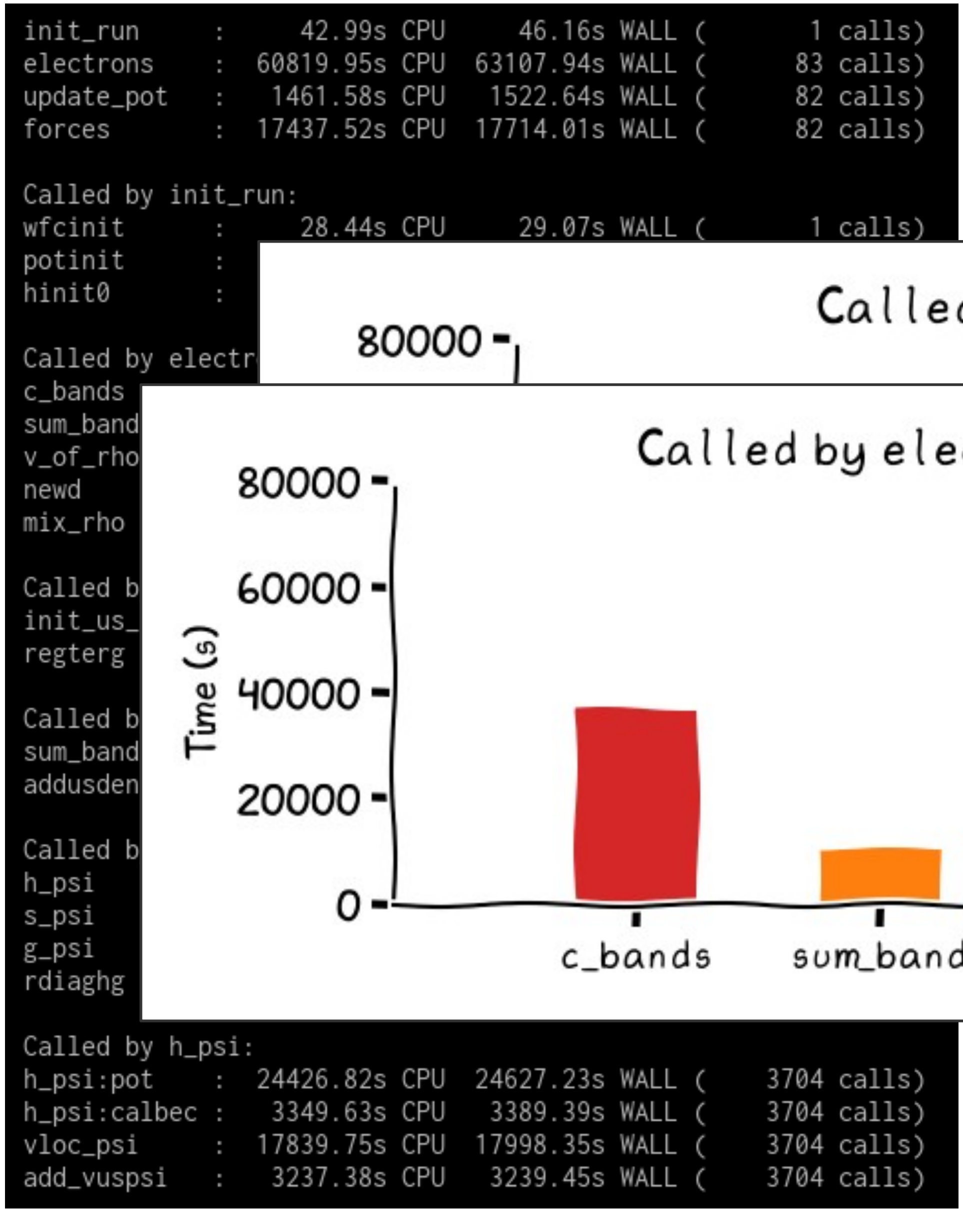


```
init_run     :     42.99s CPU     46.16s WALL (       1 calls)
electrons    :  60819.95s CPU  63107.94s WALL (      83 calls)
update_pot   :   1461.58s CPU   1522.64s WALL (      82 calls)
forces       :  17437.52s CPU  17714.01s WALL (      82 calls)

Called by init_run:
wfcinit      :     28.44s CPU     29.07s WALL (       1 calls)
potinit      :      0.79s CPU      2.21s WALL (       1 calls)
hinit0       :      8.50s CPU      8.60s WALL (       1 calls)

Called by electrons:
c_bands      :  37126.13s CPU  37854.42s WALL (     889 calls)
sum_band     :   9663.72s CPU  10448.81s WALL (     889 calls)
v_of_rho     :    501.54s CPU    536.25s WALL (     890 calls)
newd         :   2620.20s CPU   3367.58s WALL (     890 calls)
mix_rho      :    116.23s CPU    122.01s WALL (     889 calls)

Called by c_bands:
init_us_2    :    296.76s CPU    297.22s WALL (    1779 calls)
regterg      :  36350.79s CPU  36971.49s WALL (     889 calls)

Called by sum_band:
sum_band:bec :      6.01s CPU      6.08s WALL (     889 calls)
addusdens    :   3042.08s CPU   3745.04s WALL (     889 calls)

Called by *egterg:
h_psi        :  24521.86s CPU  24722.48s WALL (    3704 calls)
s_psi        :   3235.74s CPU   3235.97s WALL (    3704 calls)
g_psi        :     40.31s CPU     40.48s WALL (    2814 calls)
rdiaghg      :   2592.35s CPU   2678.62s WALL (    3540 calls)

Called by h_psi:
h_psi:pot    :  24426.82s CPU  24627.23s WALL (    3704 calls)
h_psi:calbec :   3349.63s CPU   3389.39s WALL (    3704 calls)
vloc_psi     :  17839.75s CPU  17998.35s WALL (    3704 calls)
add_vuspsi   :   3237.38s CPU   3239.45s WALL (    3704 calls)
```

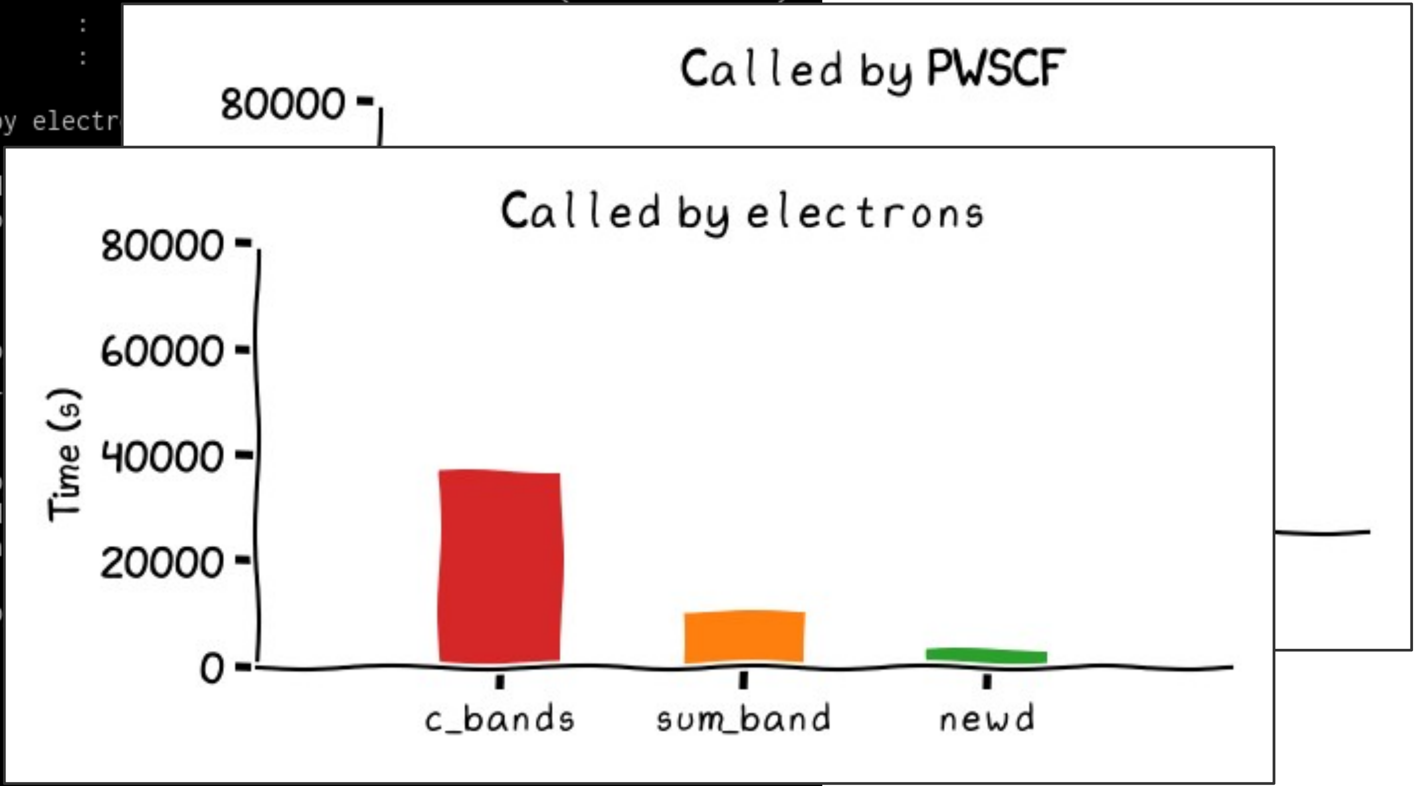


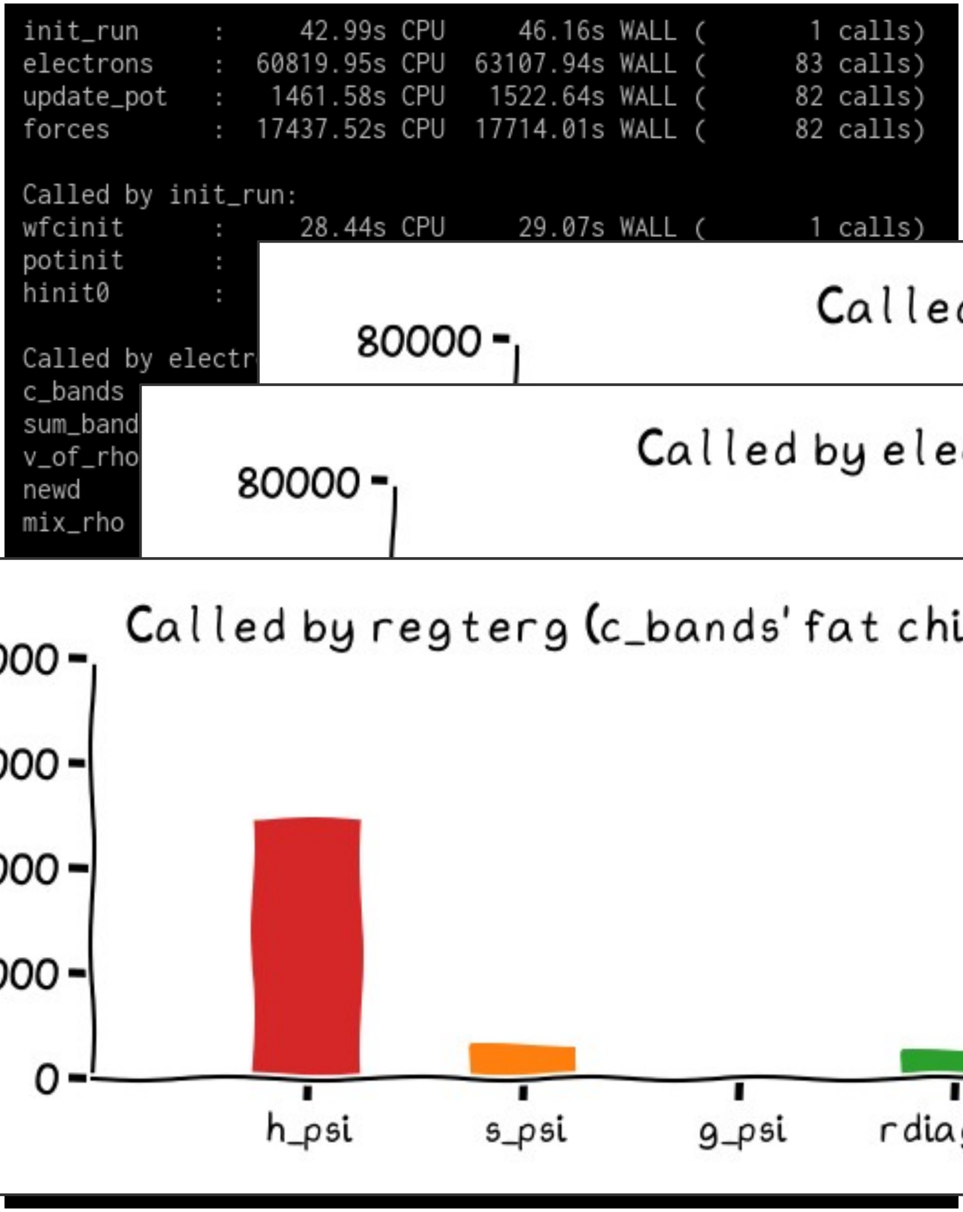
```
init_run      :     42.99s CPU     46.16s WALL (       1 calls)
electrons     :  60819.95s CPU  63107.94s WALL (      83 calls)
update_pot    :   1461.58s CPU   1522.64s WALL (      82 calls)
forces        :  17437.52s CPU  17714.01s WALL (      82 calls)

Called by init_run:
wfcinit       :     28.44s CPU     29.07s WALL (       1 calls)
potinit       :
hinit0        :

Called by electr
c_bands       :
sum_band      :
v_of_rho      :
newd          :
mix_rho       :

Called by c_band
init_us_2     :
regterg       :

Called by sum_ba
sum_band:bec  :
addusdens     :

Called by *egter
h_psi         :
s_psi         :   3235.74s CPU   3235.97s WALL (    3704 calls)
g_psi         :     40.31s CPU     40.48s WALL (    2814 calls)
rdiaghg       :   2592.35s CPU   2678.62s WALL (    3540 calls)

Called by h_psi:
h_psi:pot     :  24426.82s CPU  24627.23s WALL (    3704 calls)
h_psi:calbec  :   3349.63s CPU   3389.39s WALL (    3704 calls)
vloc_psi      :  17839.75s CPU  17998.35s WALL (    3704 calls)
add_vuspsi    :   3237.38s CPU   3239.45s WALL (    3704 calls)
```



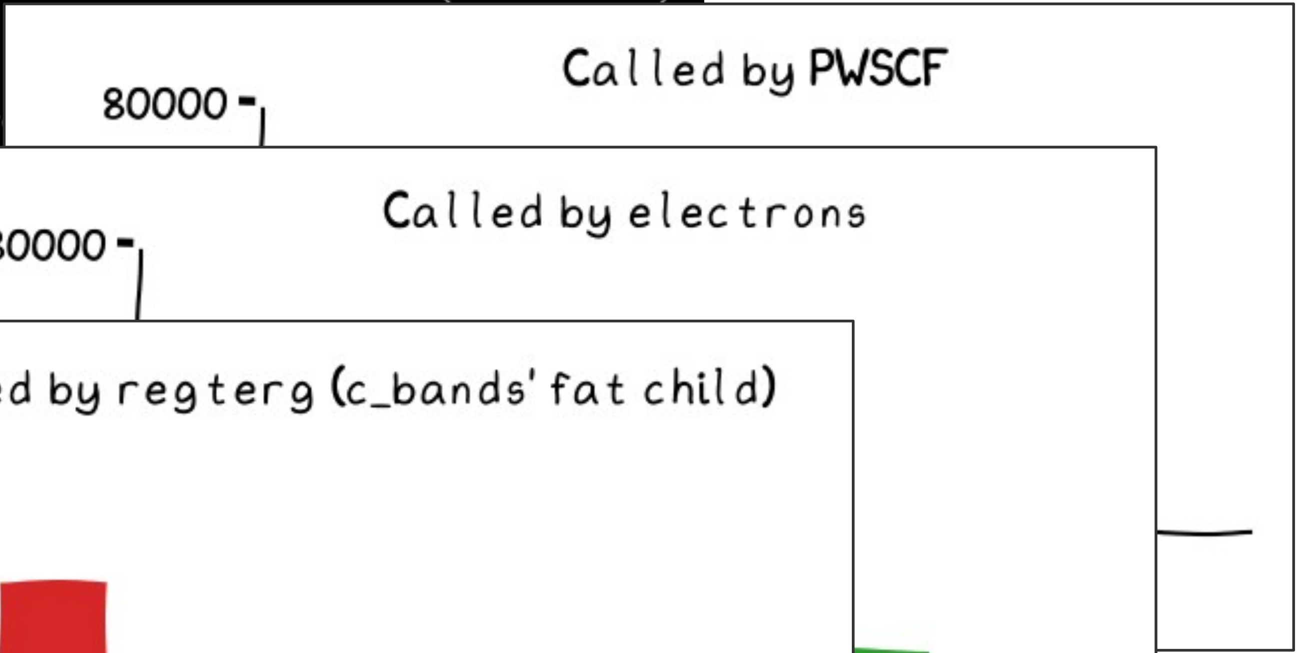


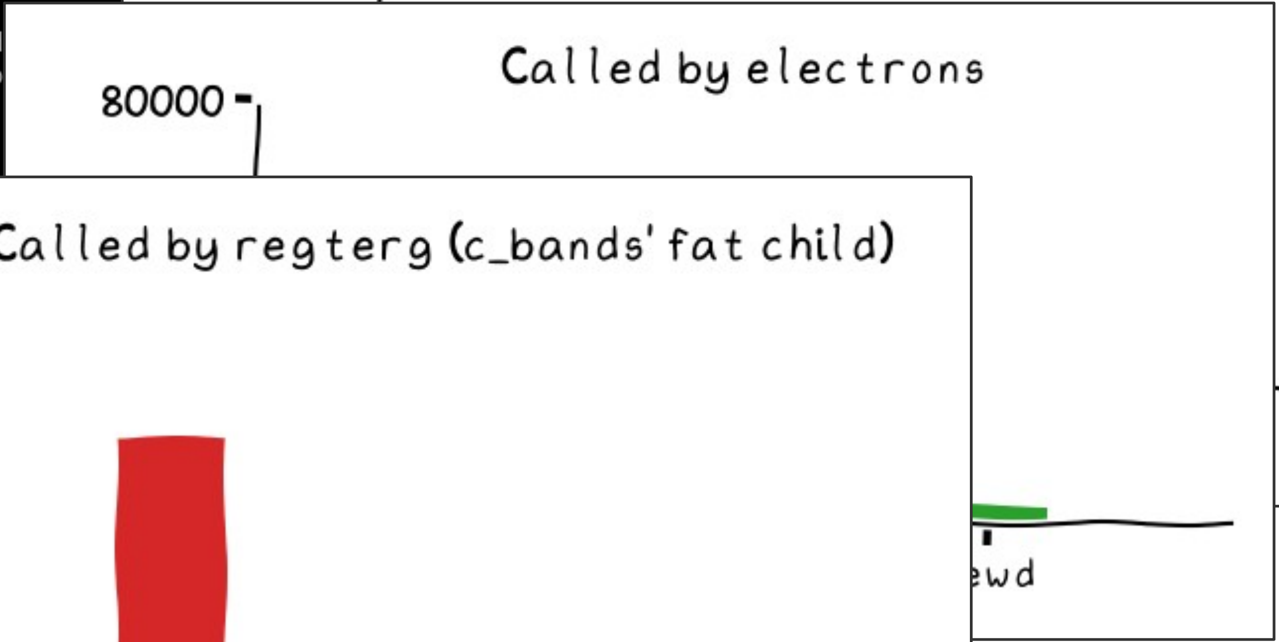
```
init_run      :     42.99s CPU     46.16s WALL (       1 calls)
electrons     :  60819.95s CPU  63107.94s WALL (      83 calls)
update_pot    :   1461.58s CPU   1522.64s WALL (      82 calls)
forces        :  17437.52s CPU  17714.01s WALL (      82 calls)

Called by init_run:
wfcinit       :     28.44s CPU     29.07s WALL (       1 calls)
potinit       :
hinit0        :

Called by electr
c_bands
sum_band
v_of_rho
newd
mix_rho

Called b
init_us_
regterg

Called b
sum_band
addusden

Called b
h_psi
s_psi
g_psi
rdiaghg

Called by h_psi:
h_psi:pot     :  24426.82s CPU  24627.23s WALL (    3704 calls)
h_psi:calbec  :   3349.63s CPU   3389.39s WALL (    3704 calls)
vloc_psi      :  17839.75s CPU  17998.35s WALL (    3704 calls)
add_vuspsi    :   3237.38s CPU   3239.45s WALL (    3704 calls)
```

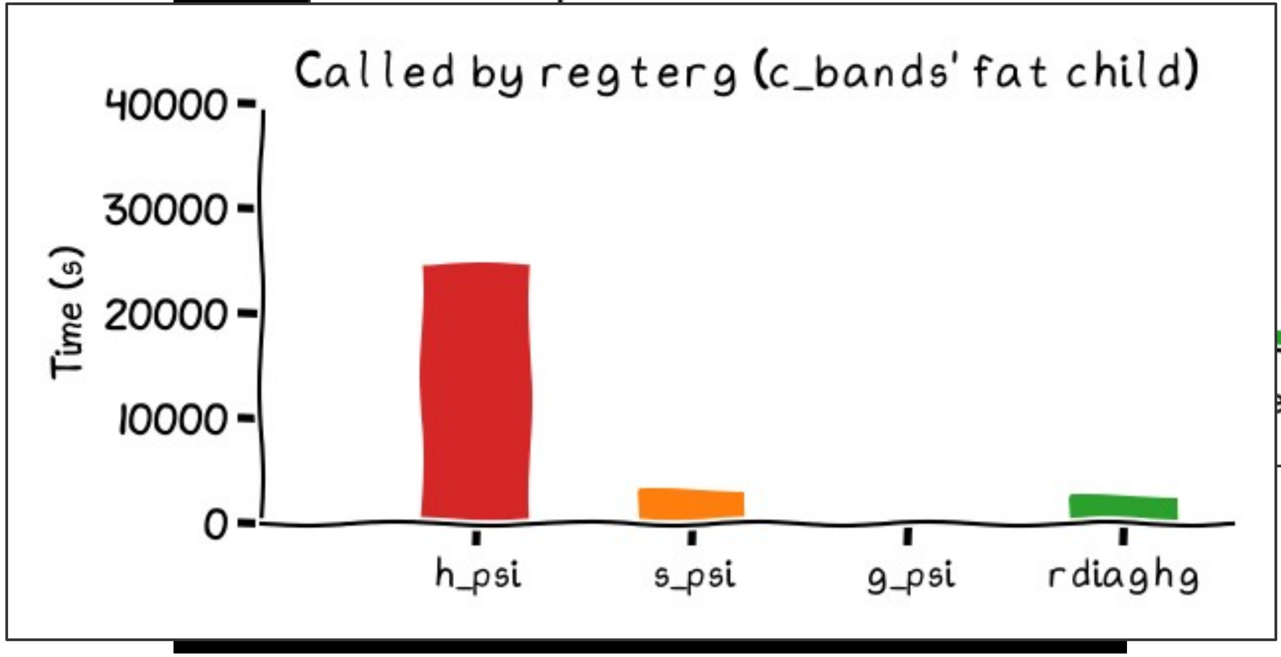




```
init_run      :     42.99s CPU     46.16s WALL (       1 calls)
electrons     :  60819.95s CPU  63107.94s WALL (      83 calls)
update_pot    :   1461.58s CPU   1522.64s WALL (      82 calls)
forces        :  17437.52s CPU  17714.01s WALL (      82 calls)

Called by init_run:
wfcinit       :     28.44s CPU     29.07s WALL (       1 calls)
potinit       :
hinit0        :

Called by electr
c_bands
sum_band
v_of_rho
newd
mix_rho
```






Called by regterg (c_bands' fat child)
Time (s)
h_psi
s_psi
g_psi
rdiaghg