Parallelization strategies
in
PWSCF
(and other QE codes)

MPI vs Open MP

MPI – Message Passing Interface

distributed memory, explicit communications

Open MP – open Multi Processing

shared data multiprocessing

MPI vs Open MP

MPI – Message Passing Interface

distributed memory, explicit communications

multicore processors (workstations,laptops), cluster of multicore processors interconnected by a fast communication network.

Open MP – open Multi Processing

shared data multiprocessing

parallelization inside a multicore processor, standalone or part of a larger network.

MPI – Message Passing Interface

distributed memory, explicit communications

```
call mp_bcast ( data, root, grp_comm )

call mp_sum ( data, grp_comm )

call mp_alltoall ( sndbuff, rcvbuff, grp_comm )

call mp_barrier ( grp_comm )
```

Open MP – open Multi Processing

shared data multiprocessing

```
!$omp parallel do
   DO j = 1, n
      a(j) =  a(j) + b(j)
   ENDDO
!$omp end parallel do
```

MPI – Message Passing Interface

distributed memory, explicit communications

Data distribution is a big constraint, usually made at the beginning and kept across the calculation.
It is pervasive and rather rigid: a change in data distribution or a new parallelization level impacts the whole code.
Computation should mostly involve local data.
Communications should be minimized.
Data distribution reduces memory footprint.

Open MP – open Multi Processing

shared data multiprocessing

Can be used inside a memory-sharing multi-processor node. Cannot be used across nodes.
Scalability with the number of threads may vary.
Its implementation can be incremental.

Network important factors

Bandwidth and Latency

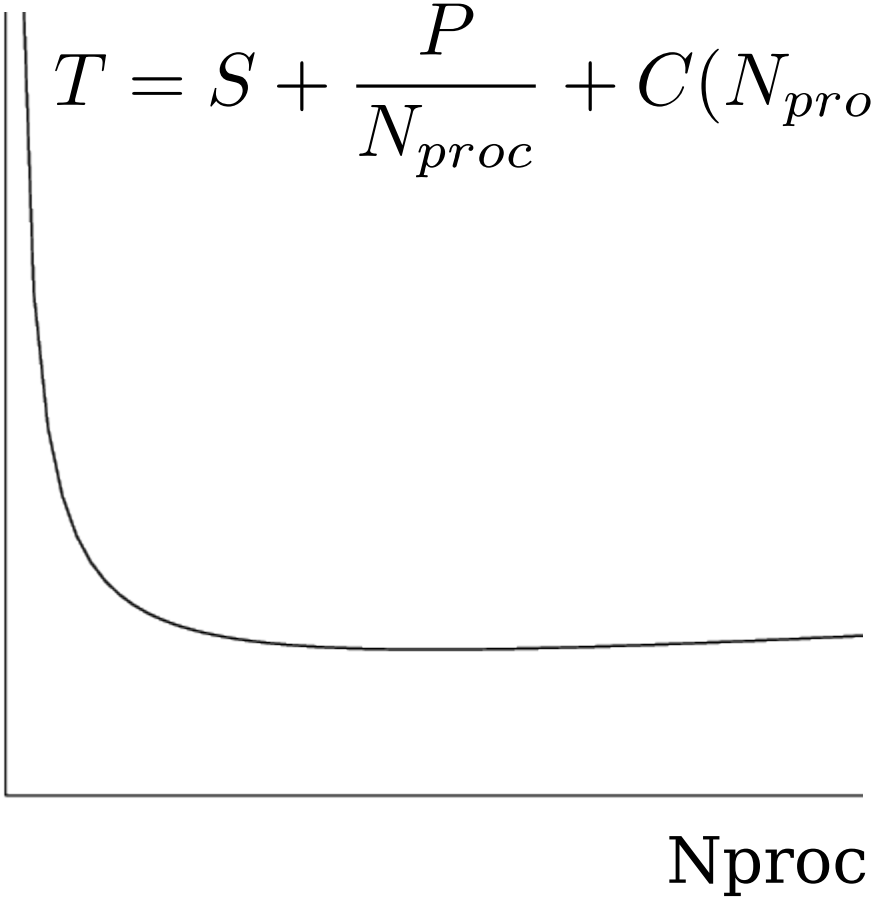
A fast interconnection is important but how often and how much one communicates is also very important.

Blocking vs non blocking communications may also play a role.

Disk I/O is typically bad and should be avoided.
On parallel machines even more so.
If RAM allows for it keep things in memory.

Amdahl's law

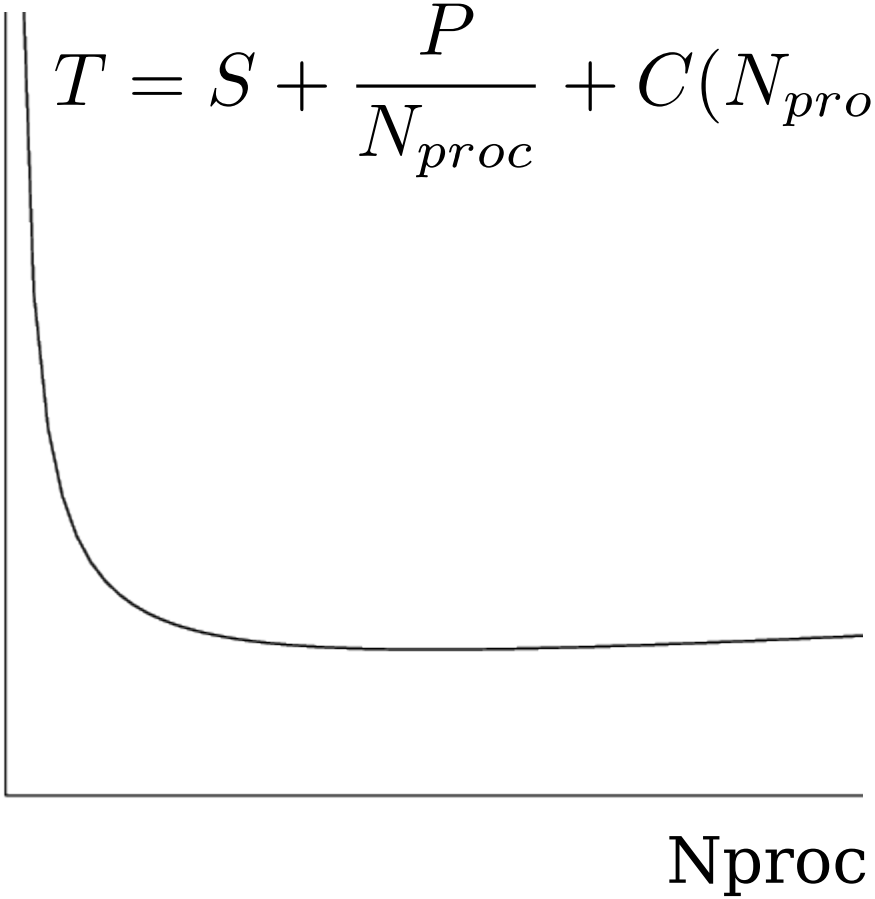
$$T = S + \frac{P}{N_{proc}} + C(N_{proc})$$



No matter how well you parallelize your code on the long run the scalar fraction dominates.

Amdahl's law

$$T = S + \frac{P}{N_{proc}} + C(N_{proc})$$



No matter how well you parallelize your code on the long run the scalar fraction dominates.
Even before communication becomes an issue.

Strong Scaling vs Weak Scaling

Strong Scaling: scaling when system size remains fixed
Weak Scaling: scaling when system size also grows

Strong Scaling is much more difficult to achieve than Weak Scaling.

Computer centers are OK with Weak Scaling because they can use it to justify their existence, but they really push for Strong Scaling.

Many times one does not need to perform huge calculations but rather many medium/large calculations. Extreme parallelization would not be needed but queue scheduler constraints enforce the use of many cores.

MPI – Message Passing Interface

distributed memory, explicit communications

MPI – Message Passing Interface

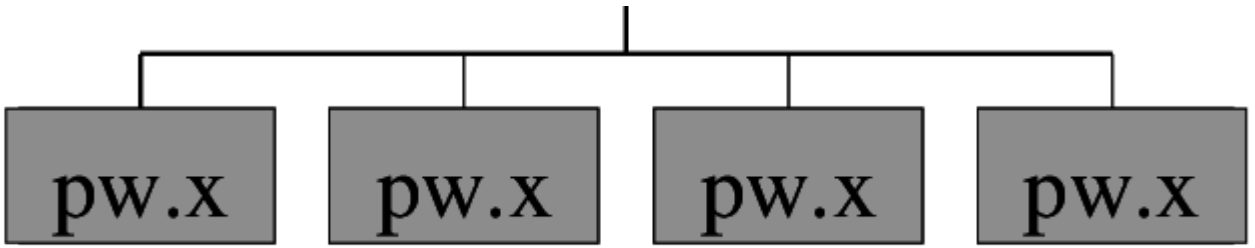
multiple processes, multiple data, single program

if MPI library is linked and invoked

```
  CALL MPI_Init(ierr)
```

it is possible to start several copies of the code
on different cores/processors of the machine/cluster

```
prompt> mpirun -np 4 pw.x < pw.in > pw.out
```

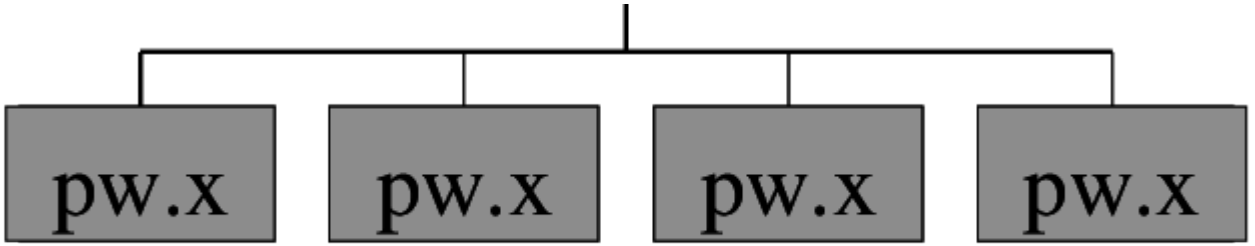


each core executes the code starting at the beginning
and following the flow and computation instructions
as determined by the information available ***locally***
to that core/processor.

MPI – Message Passing Interface

multiple processes, multiple data, single program

```
prompt> mpirun -np 4 pw.x < pw.in > pw.out
```



it may be useful to know how many cores are running

```
   CALL mpi_comm_size(comm_world,numtask,ierr)
```

and my id-number in the group

```
   CALL mpi_comm_rank(comm_world,taskid,ierr)
```

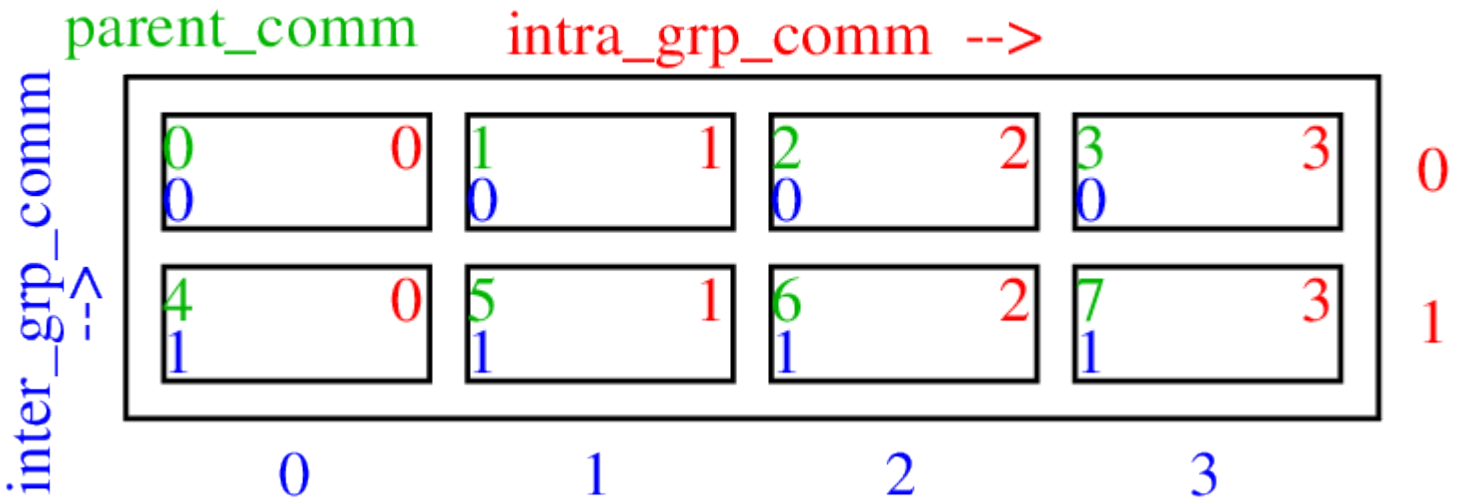
comm_world is the global default communicator
defined on MPI initialization.

MPI – Message Passing Interface

using a hierarchy of parallelization levels

communication groups can be further split as needed/desired

```
my_grp_id = parent_mype / nproc_grp          ! the sub grp I belong to
me_grp    = MOD( parent_mype, nproc_pgrp )   ! my place in the sub grp
!
! ... an intra_grp_comm communicator is created (to talk within the grp)
!
CALL mp_comm_split( parent_comm, my_grp_id, parent_mype, intra_grp_comm )
!
! ... an inter_grp_comm communicator is created (to talk across grps)
!
CALL mp_comm_split( parent_comm, me_grp, parent_mype, inter_grp_comm )
```



MPI – Message Passing Interface

basic communication operations

```
call mp_barrier ( grp_comm )

call mp_bcast ( data, root, grp_comm )

call mp_sum ( data, grp_comm )

call mp_alltoall ( sndbuff, rcvbuff, grp_comm )
```

MPI – Message Passing Interface

a simple example

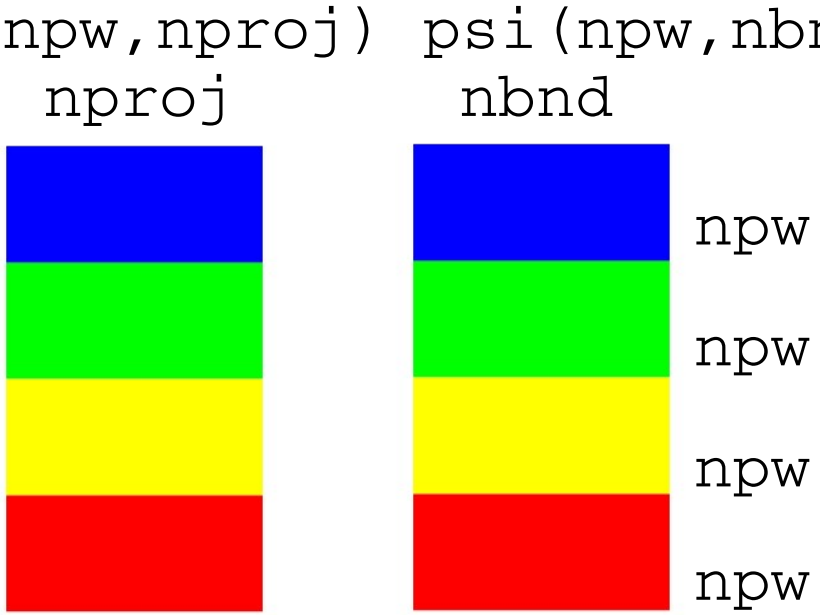
$$\langle \beta_i | \psi_j \rangle = \sum_{k+G} \beta_i^*(k+G)\psi_j(k+G)$$

MPI – Message Passing Interface

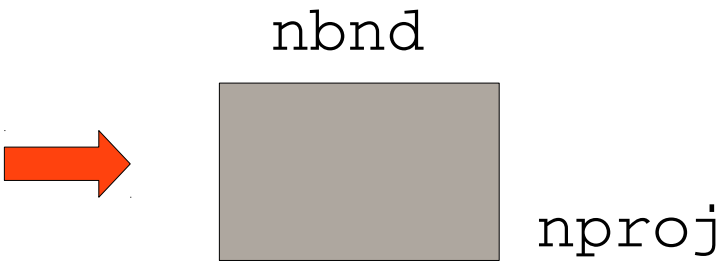
a simple example

$$\langle \beta_i | \psi_j \rangle = \sum_{k+G} \beta_i^*(k+G)\psi_j(k+G)$$

beta(npw,nproj) psi(npw,nbnd)



betapsi(nproj,nbnd)



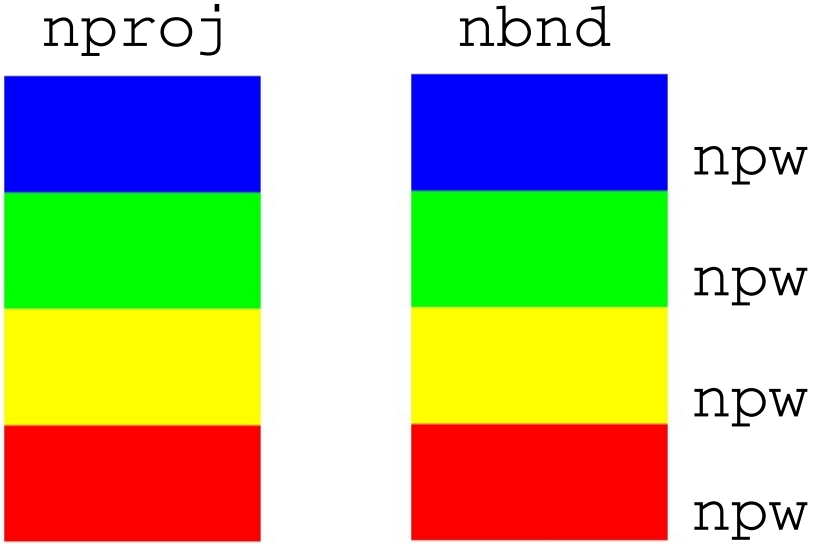
how one gets betapsi?

MPI – Message Passing Interface

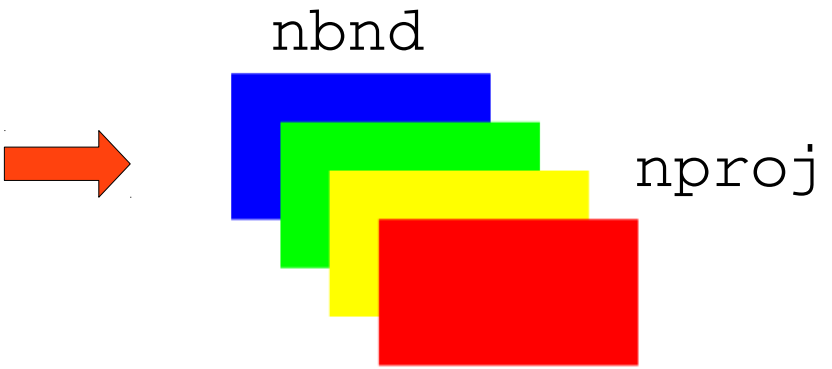
a simple example

$$\langle \beta_i | \psi_j \rangle = \sum_{k+G} \beta_i^*(k+G)\psi_j(k+G)$$

beta(npw,nproj) psi(npw,nbnd)



betapsi(nproj,nbnd)




```
CALL ZGEMM( 'C', 'N', nproj, nbnd, npw, (1.0_DP,0.0_DP), &
                  beta, npwx, psi, npwx, (0.0_DP,0.0_DP), &
                  betapsi, nprojx )
```

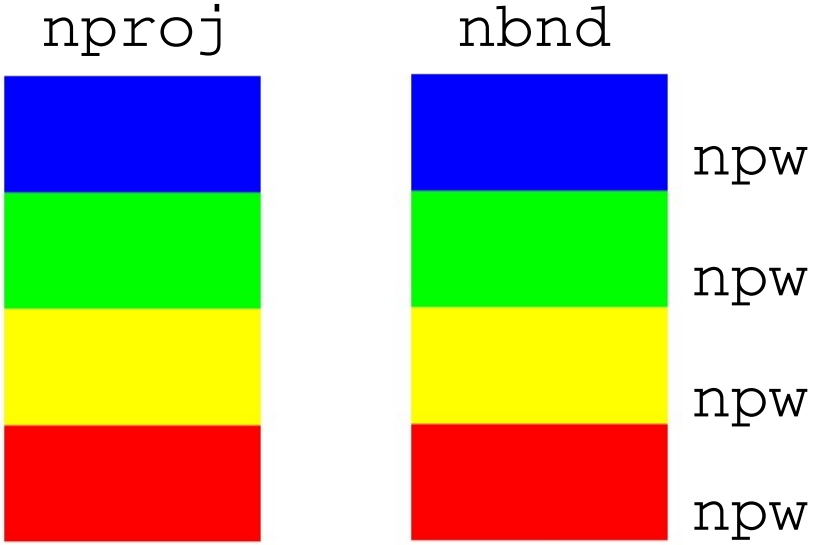
each processor has a partially summed betapsi

MPI – Message Passing Interface

a simple example

$$\langle \beta_i | \psi_j \rangle = \sum_{k+G} \beta_i^*(k+G)\psi_j(k+G)$$

beta(npw,nproj) psi(npw,nbnd)

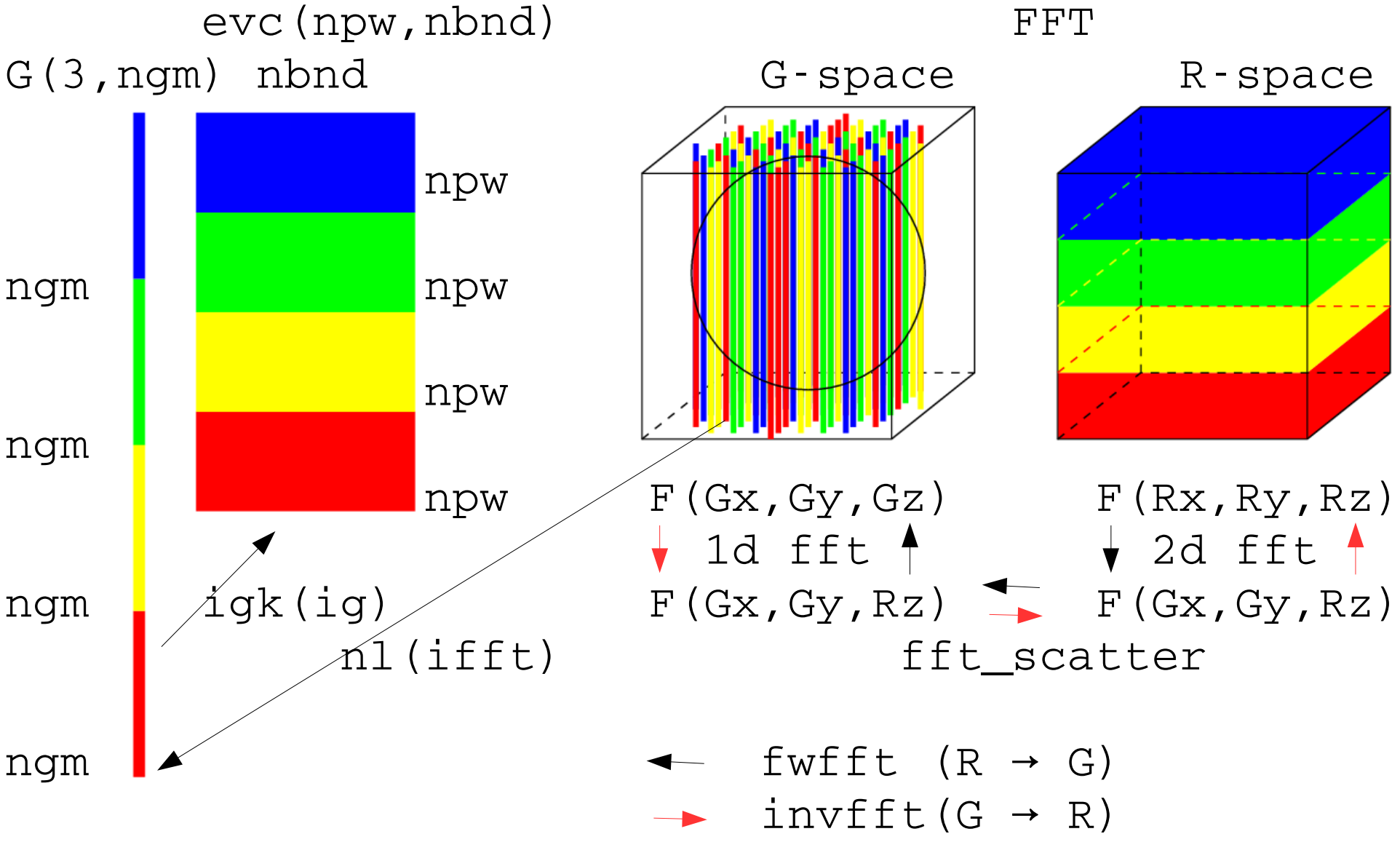


betapsi(nproj,nbnd)


```
CALL ZGEMM( 'C', 'N', nproj, nbnd, npw, (1.0_DP,0.0_DP), &
                  beta, npwx, psi, npwx, (0.0_DP,0.0_DP), &
                  betapsi, nprojx )
CALL mp_sum( betapsi, intra_bgrp_comm )
```

at the end each processor has the complete betapsi !

R & G parallelization



MPI – Message Passing Interface

hierarchy of parallelization in PW

```
call mp_comm_split ( parent_comm, subgrp_id,
                     parend_grp_id, subgrp_comm )
```

```
mpirun -np $N pw.x -nk $NK -nb $NB -nt $NT -nd $ND
                                   < pw.in > pw.out
```

```
-nk (-npool, -npools)    # of pools
-ni (-nimage, -nimages) # of images for NEB or PH
-nb (-nband, -nbgrp, -nband_group) # of band groups
-nt (-ntg, -ntask_groups) # of FFT task groups
-nd (-ndiag, -northo,-nproc_diag, -nproc_ortho)
                         # of linear algebra groups
```

$N = $NI x $NK x $NB

MPI – Message Passing Interface

hierarchy of parallelization in PW

- R & G space parallelization
  data are distributed, communication of results is frequent. High communication needs, reduction of processor memory footprint.

- K-point parallelization
  different k-points are completely independent during most operations. Needs to collect contributions from all k-points from time to time. Mild communication needs, no lowering of the processor memory footprint, unless all k-point are kept in memory...

- Image parallelization
  different NEB images or different irreps in PH are practically independent. Low communication needs, no lowering of the processor memory footprint

MPI – Message Passing Interface

additional levels of parallelization

- Band parallelization
  different processors deal with different subset of the bands. Computational load distributed, no memory footprint reduction for now.

- Task group parallelization
  FFT data are redistributed to perform multiply FFT at the same time. Needed when number of processors is large compared with FFT dimension (nrx3).

- linear algebra parallelization
  diagonalization routines are parallelized

MPI/openMP scalability issues

-openMP can be used inside a memory-sharing multi-processor node. Cannot be used across nodes. Scalability with the number of threads may vary.

-whenever possible use image and k-point parallelism as they involve low communication. Beware of the granularity of the load distribution and the size of the individual subgroups.

-R & G space distribution really distributes memory ! It is communication intensive. FFT dimension limited.

To extend scalability to large number of processors -task_groups, -band_group and/or -ndiag are needed.