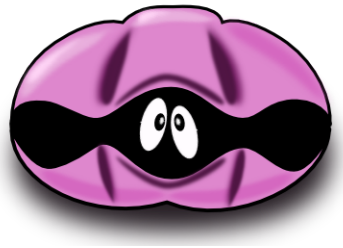
CNR
Istituto di Struttura
della Materia

CNR-ISM, Division of Ultrafast
Processes in Materials (FLASHit),
Area della Ricerca di Roma 1,
Monterotondo Scalo, Italy

# Yambo parallelization: strategies & performance

D. Sangalli
& the Yambo team

Yambo©



MAX

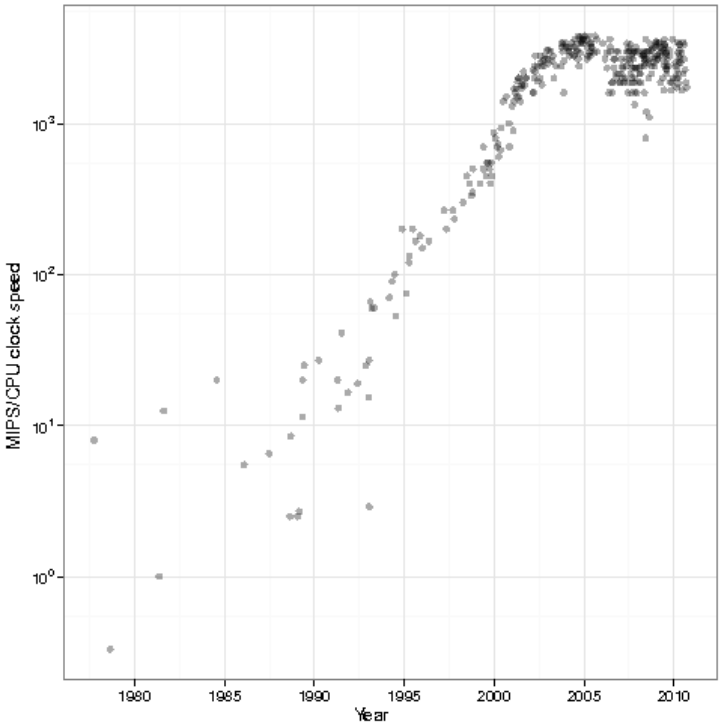
Bologna
4th Dec 2017

# outline

- motivations: **HPC, exascale**
- paradigms: **MPI, OpenMP**, and more
- Yambo **parallelization strategies**
- **Performance**

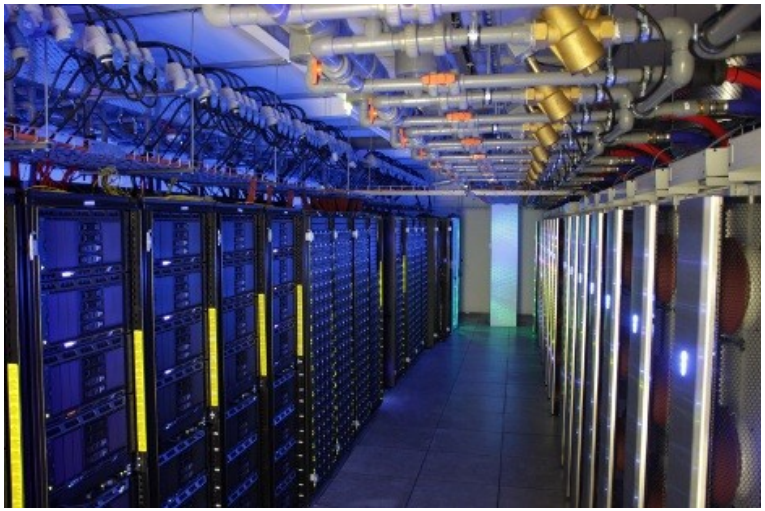
# outline

- motivations: **HPC, exascale**
- paradigms: **MPI, OpenMP**, and more
- Yambo **parallelization strategies**
- **Performance**




Clock frequency did not
improve any more since 2005

from single core
to multi core
to HPC centers



# HPC & exascale

**The White House**
Office of the Press Secretary

For Immediate Release

July 29, 2015

## Executive Order -- Creating a National Strategic Computing Initiative

EXECUTIVE ORDER

- - - - - - -

CREATING A NATIONAL STRATEGIC COMPUTING INITIATIVE

# HPC & exascale

## within the EU HPC ecosystem

**EINFRA-5-2015 – Centres of Excellence for computing applications**

Specific challenge: Establishing a limited number of Centres of Excellence (CoE) is necessary to ensure EU competitiveness in the application of HPC for addressing scientific, industrial or societal challenges. CoEs will be user-focused, develop a culture of excellence, both scientific and industrial, placing computational science and the harnessing of 'big data' at the centre of scientific discovery and industrial competitiveness. […]

ETP 4 HPC
THE EUROPEAN TECHNOLOGY PLATFORM FOR HIGH PERFORMANCE COMPUTING

MAX

### Keywords

HPC
exascale
big data
scientific SW
users
communities

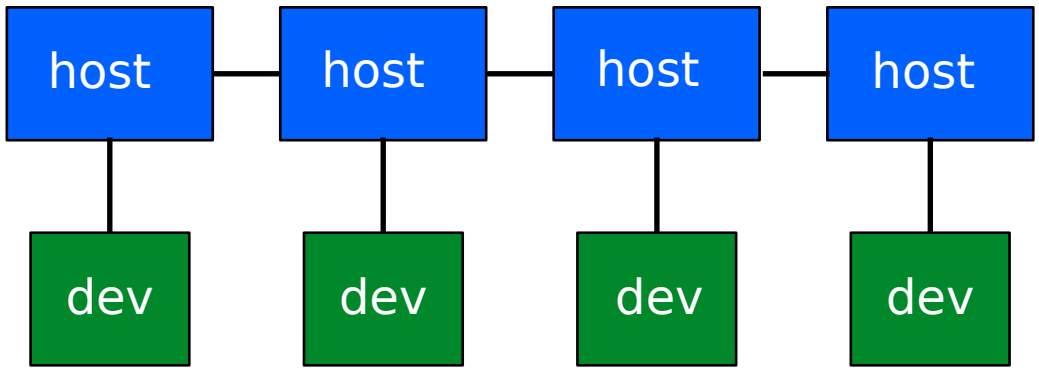
- 9 CoEs established
- **4 in Materials Science** (and electronic structure)
- + weather, medicine, biochem, data management, HPC profiling
- **MaX** - Materials design at the exascale

# HPC & exascale

**the exascale challenge**
in high performance computing

- 10^18 flops/s
- 10^18 Bytes
- abrupt technology changes
- action is needed for full exploitation

**The architecture may be non trivial**



# so what?

- A lot of **computational resources**. Will be **even more so** in the future
- YAMBO **parallel structure** extensively developed in the past years
- **data and automation**: interfacing with yambopy and AiiDA

Yambo ©

- QP vs KS energies
- BSE vs IP absorption

**Higher accuracy**

**High throughput**

AiiDA
Automated Interactive Infrastructure and Database for Computational Science

**yambopy**

non equilibrium physics
(not yet GPL)

Yambo ©

**New physics**

**Larger systems**

Surfaces, interfaces, nanostructures ab-initio

Extend the limits of system size with both DFT (5000 atoms?) and MBPT (200 atoms?)

# outline

- motivations: **HPC, exascale**
- paradigms: **MPI, OpenMP**, and more
- Yambo **parallelization strategies**
- **Performance**

# MPI

- MPI: **message passing** interface (protocol)
- one the most **popular standards** for parallel computing (versions: MPI-2, MPI-3)
- multiple processes (**MPI tasks**) are started at the same time, and communicate by means of MPI calls


## Opportunities

- **memory** and **computation** can be distributed
- widely supported and available everywhere
- arch optimized
- both synchronous and asynchronous communication

## Threats

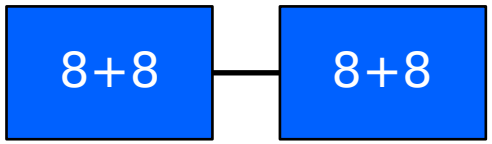
- communication overheads
- **load unbalance**
- memory duplication

# MPI

## Control


- running over 16 MPI tasks is achieved by:
  mpirun -np 16  yambo  -F file.in
- safer to set
  export OMP_NUM_THREADS=1  (see next slides)

## Typical usage




scheme of nodes
with two octacore CPUs

- 2 nodes, each with 16 cores
- **32 cores** in total
- => run with **32 MPI tasks** (memory permitting)

# OpenMP


- OpenMP: open multi processing (API)
- support **share memory** paradigm
- **multiple threads** are forked/joined during execution to run in parallel (over the same chunk of memory) specific portions of the code
- standard and widely supported (v3 and v4)

## Opportunities

- exploits computing power keeping memory under control (no memory duplication)
- can be combined with MPI

## Threats

- works **within the node**
- fork/join processes lead to sensible time overheads
- scalability to large number of threads not so easy

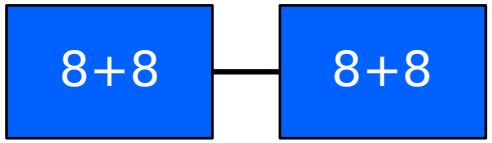
# OpenMP

## Control


- running over 8 OpenMP threads is achieved by setting:

  export OMP_NUM_THREADS=8
- if using MKL, also set:
  export MKL_NUM_THREADS=8

## Typical usage




scheme of nodes
with two octacore CPUs

- 2 nodes, each 16 cores
- **OMP can only be used within the node**
- => run with 4 or 8 threads ok
  => run with 16 threads possible but less recommended (two sockets present)

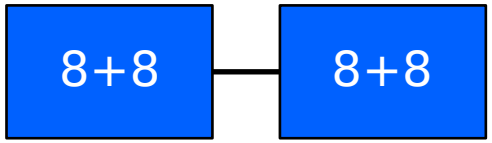
# hybrid MPI+OpenMP

## Control

- export OMP_NUM_THREADS=4
- mpirun -np 8 yambo -F file.in



## Typical usage



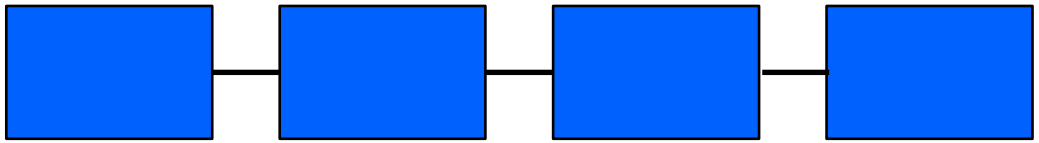

**Yambo TIP:**

- use as many MPI tasks as possible
- when memory is over use omg threads to fill the node

- 2 nodes, each with 16 cores
- **OMP within the node, MPI intra and inter node**
- => run with 8 MPI tasks, each using up to 4 OpenMP threads

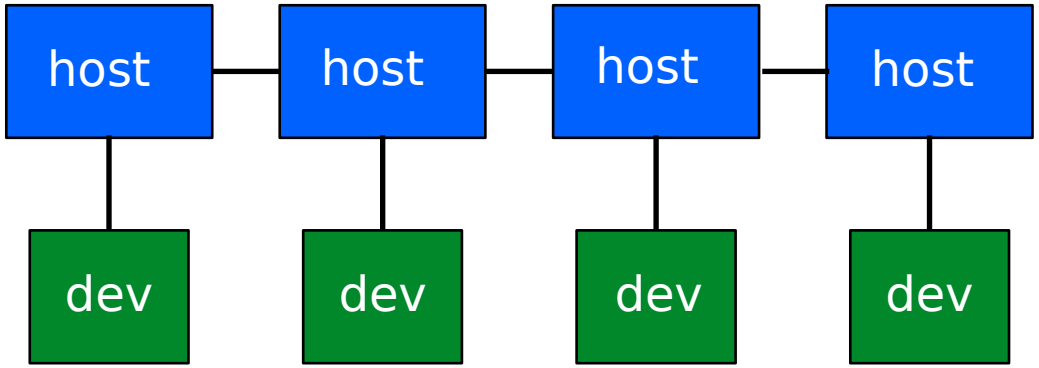
# machine architectures

- **homogeneous arch**



- collections of nodes with a given number of cores
- ex: most local clusters
- ex: Marconi A2 (Intel KNL), #12 top500

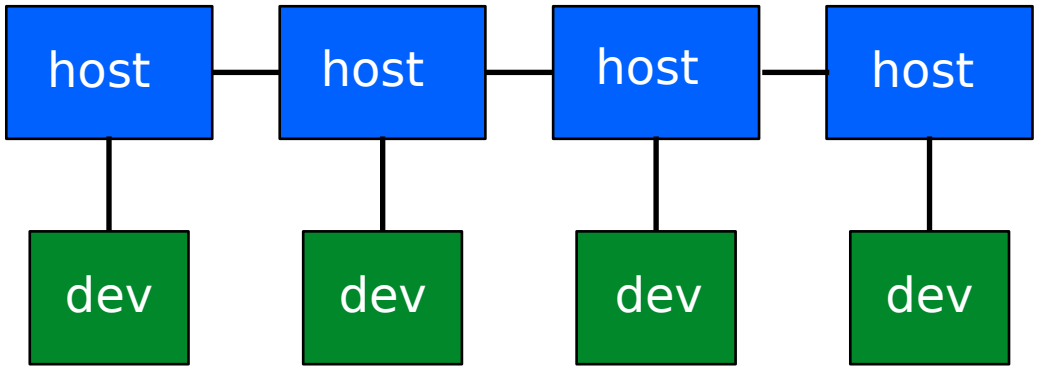
- **heterogeneous arch**




- ex: Piz-Daint (CSCS, CH), #8 top500 (Intel Xeon + NVIDIA Tesla P100)
- “Heterogeneity is there to stay”

# how to use GPUs?

- usually, 1 MPI task per accelerator
- need to handle the transfer of data from host to device(s)
- with nvidia GPUs, CUDA and CUDA-Fortran can be used
- some support is there, not as spread as MPI/OpenMP,
- **solutions designed for specific hardware, not as universal as MPI/OpenMP**
- GPUs support under development in YAMBO ... stay tuned



# outline

- motivations: **HPC, exascale**
- paradigms: **MPI, OpenMP**, and more
- Yambo **parallelization strategies**
- **Performance**

# yambo: parallelism

- yambo implements a **hybrid MPI+OpenMP** paradigm
- **MPI** works over several (3 to 5) **different levels**, according to the runlevel
- **OpenMP** usually (not always) works at a **lower level**, reaching very different levels of efficiency
- **parallel linear algebra** is supported (ScaLapack, SLEPC, PETSC)
- overall, yambo is quite **parallel oriented**


+

# yambo: parallelism

## Fermi blue gene Q

How Carlo explained it to Andrea

**Andrea**

**“You can run yambo on 32'768 cores”**

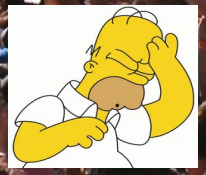
**Carlo**

CINECA

# yambo: parallelism

## Fermi blue gene Q

## How Andrea got it

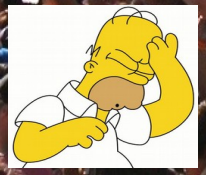


call mpi_allreduce(x)
among almost 50'000 cores...
Are you sure you want to do it?

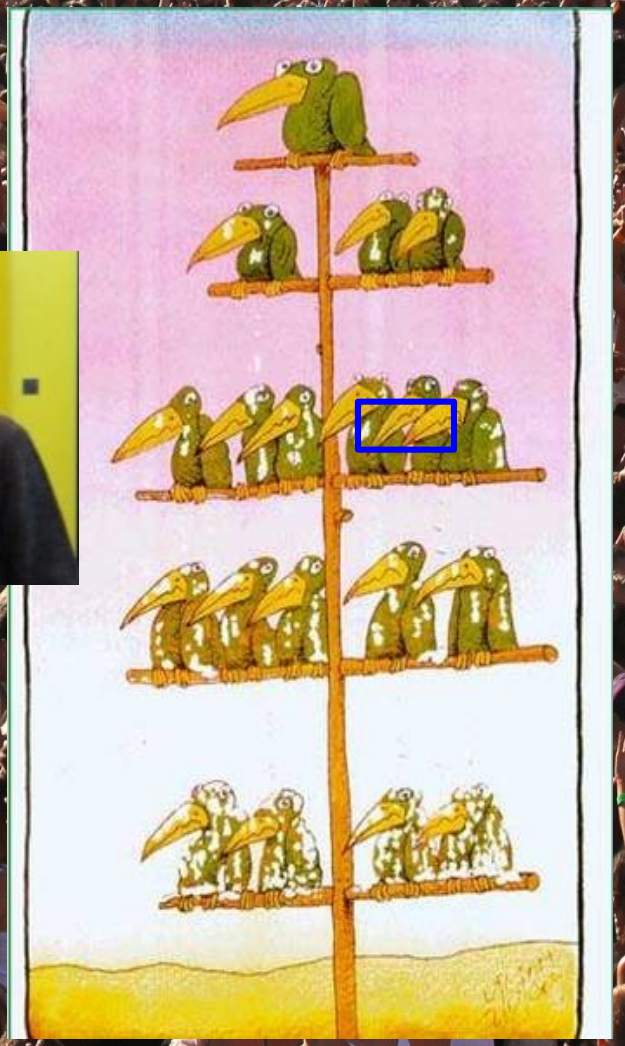
# yambo: parallelism

## Fermi blue gene Q

## How Andrea got it



call mpi_allreduce(x)
among almost 50'000 cores...
Are you sure you want to do it?



## Action to do:

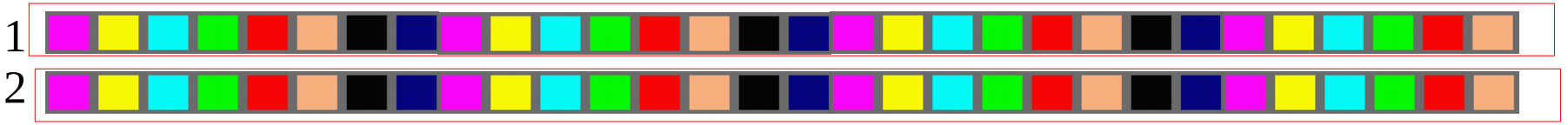
**We do need a hierarchy among the Cores**

# yambo: parallelism

60 MPI tasks

# yambo: parallelism

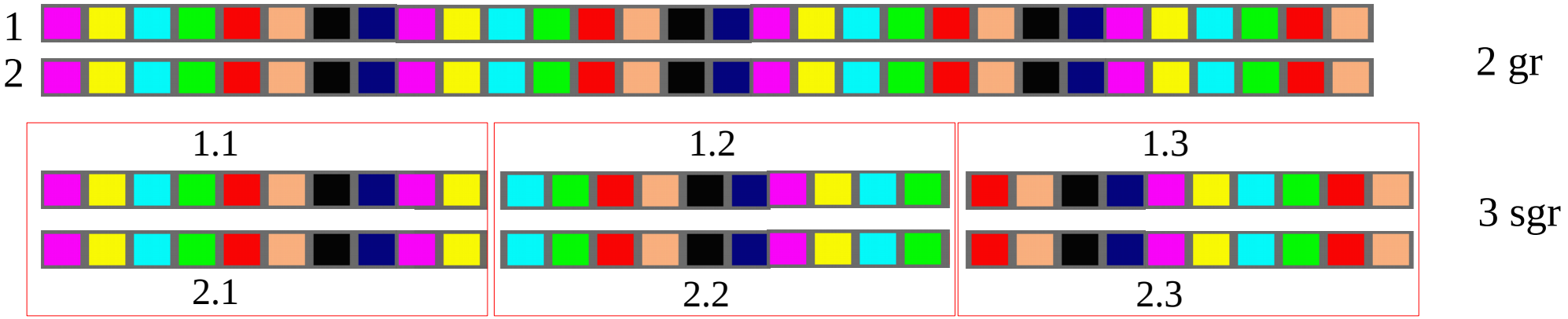
60 MPI tasks with a 2




2 gr

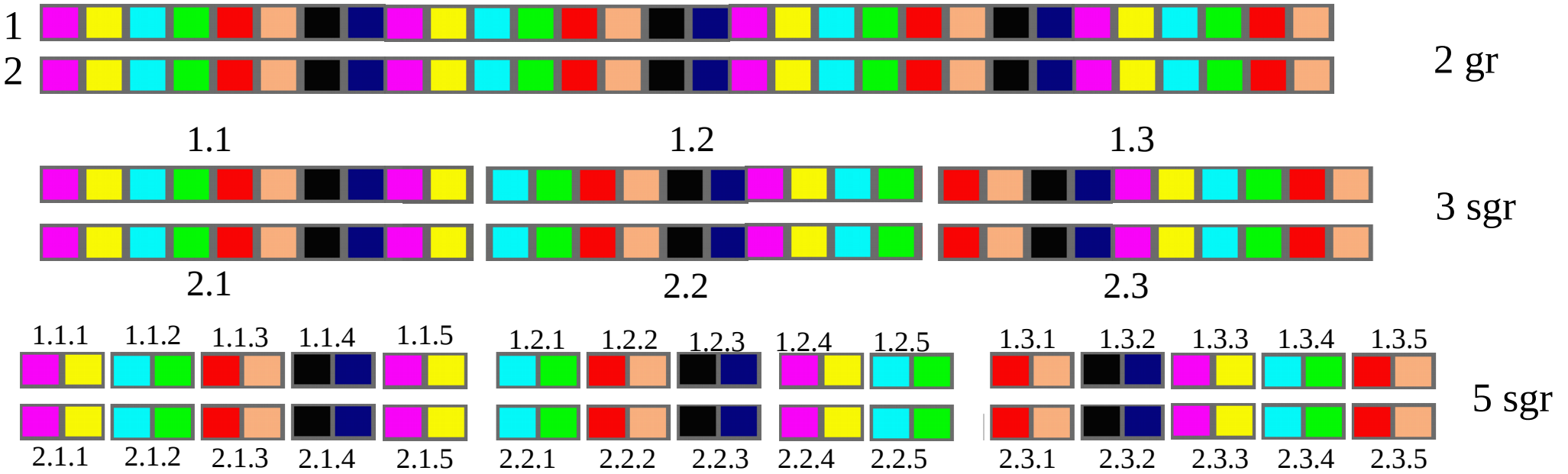
# yambo: parallelism

60 MPI tasks with a 2.3



# yambo: parallelism

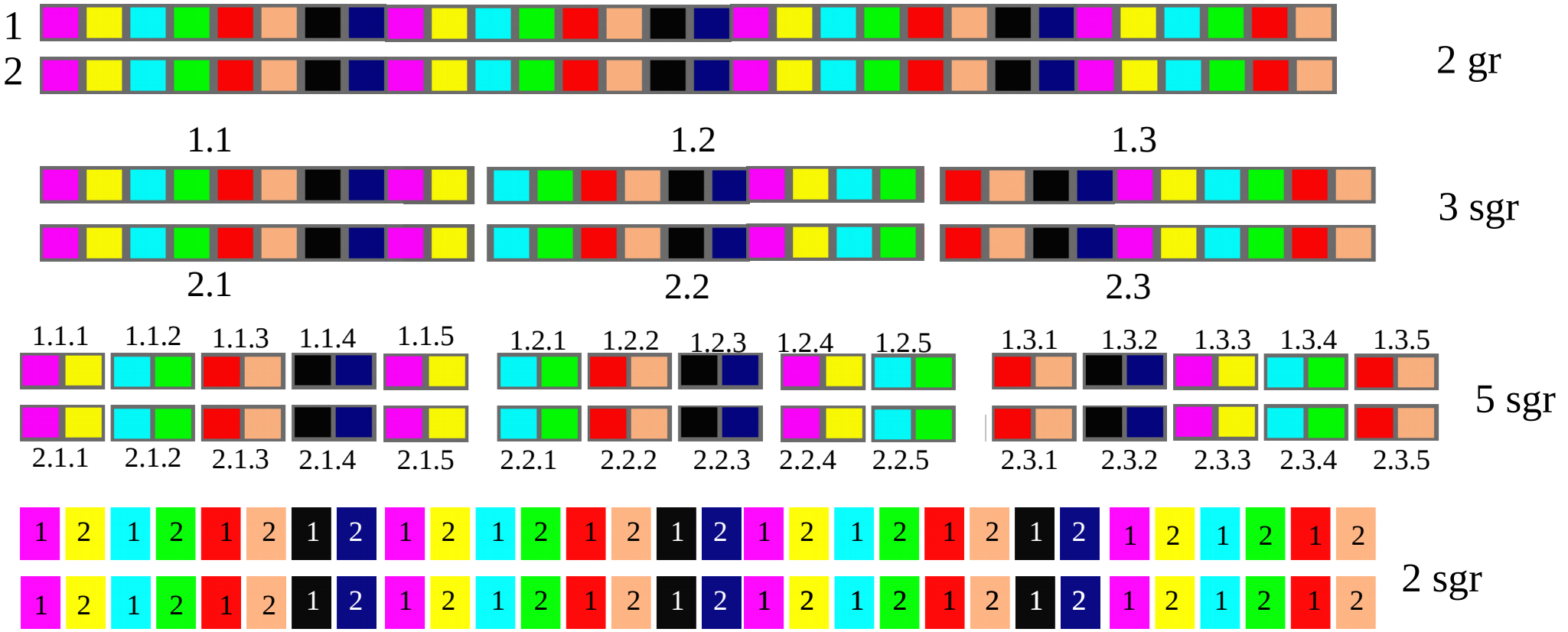
60 MPI tasks with a 2.3.5



# yambo: parallelism

60 MPI tasks with a 2.3.5.2 scheme 2*3*5*2=60

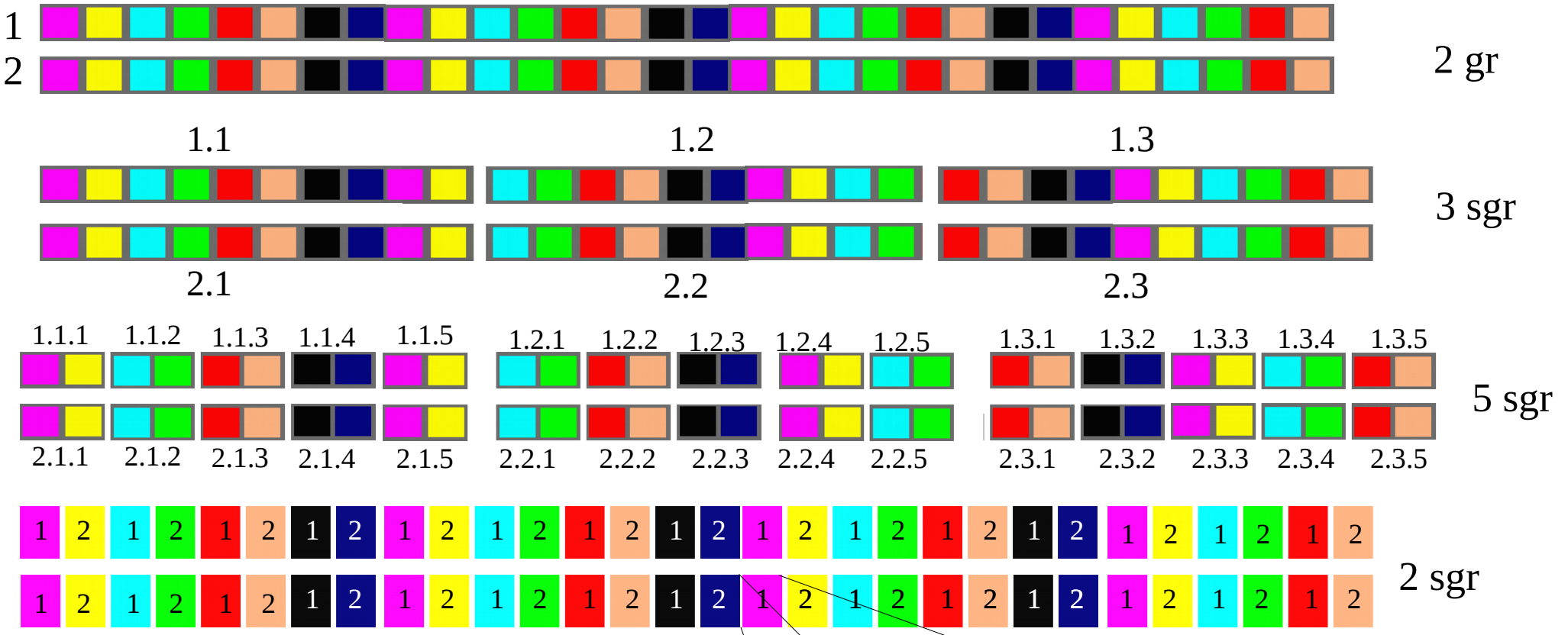
Each number in the hierarchy will be distributed on specific variables



# yambo: parallelism

60 MPI tasks with a 2.3.5.2 scheme  2*3*5*2*8=480

Each number in the hierarchy will be distributed on specific variables

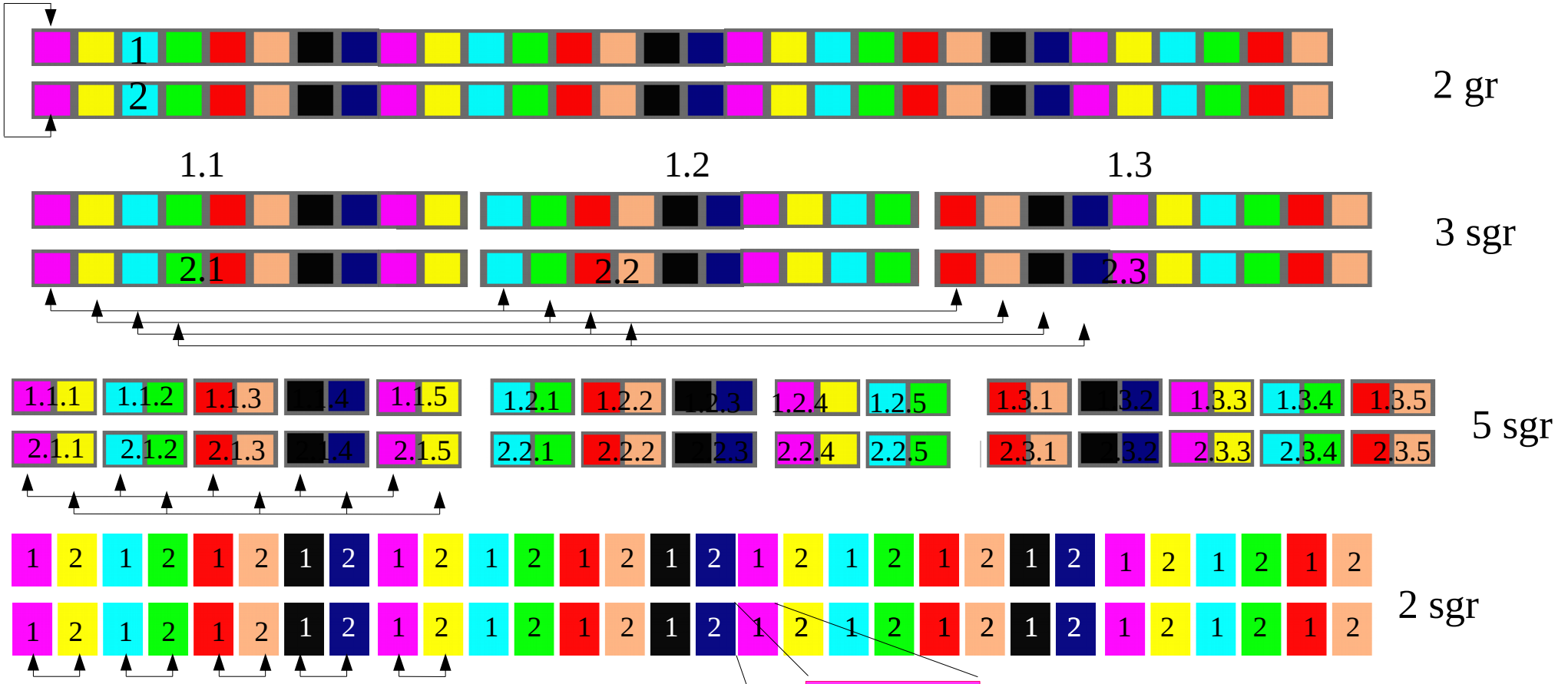



We are using 480 cores in total
a) thanks to OpenMP only 60 MPI tasks
b) yambo can exploit parallelism
on different variables

# yambo: parallelism

60 MPI tasks with a 2.3.5.2 scheme  2*3*5*2*8=480

Each number in the hierarchy will be distributed on specific variables




We are using 480 cores in total
a) thanks to OpenMP only 60 MPI tasks
b) yambo can exploit parallelism
on different variables
c) the number of communications is limited

This enables to overcome limits of MPI communications which do not work up to any number of cores

# Linear Response

$$\chi^0_{\mathbf{GG'}}(\mathbf{q},\omega) = 2\sum_{c,v}\int_{BZ}\frac{d\mathbf{k}}{(2\pi)^3}\rho^*_{cv\mathbf{k}}(\mathbf{q},\mathbf{G})\rho_{cv\mathbf{k}}(\mathbf{q},\mathbf{G'})\, f_{v\mathbf{k}-\mathbf{q}}(1-f_{c\mathbf{k}})\times$$
$$\times\left[\frac{1}{\omega+\epsilon_{v\mathbf{k}-\mathbf{q}}-\epsilon_{c\mathbf{k}}+i0^+}-\frac{1}{\omega+\epsilon_{c\mathbf{k}}-\epsilon_{v\mathbf{k}-\mathbf{q}}-i0^+}\right]$$

q transferred momenta (MPI q)

Xo bands (MPI c,v)

k momenta (MPI k)

space variables

```
X_all_q_ROLEs= "q k c v"     # CPUs roles (q,k,c,v)
X_all_q_CPU  = "1 2 4 2"     # CPUs for each role
X_Threads = 4                # num threads
```

MPI-cv best memory distribution
MPI-k as efficient, some mem dupl
MPI-q may lead to load unbalance, and memory duplication
OpenMP efficient, need extra mem

# Linear Response

$$\chi^0_{\mathbf{GG'}}(\mathbf{q},\omega) = 2\sum_{c,v}\int_{BZ}\frac{d\mathbf{k}}{(2\pi)^3}\rho^*_{cv\mathbf{k}}(\mathbf{q},\mathbf{G})\rho_{cv\mathbf{k}}(\mathbf{q},\mathbf{G'})\,f_{v\mathbf{k}-\mathbf{q}}(1-f_{c\mathbf{k}})\times$$
$$\times\left[\frac{1}{\omega+\epsilon_{v\mathbf{k}-\mathbf{q}}-\epsilon_{c\mathbf{k}}+i0^+}-\frac{1}{\omega+\epsilon_{c\mathbf{k}}-\epsilon_{v\mathbf{k}-\mathbf{q}}-i0^+}\right]$$

q transferred momenta (MPI q)

Xo bands (MPI c,v)

k momenta (MPI k)

space variables

MPI-cv best memory distribution
MPI-k as efficient, some mem dupl
MPI-q may lead to load unbalance, and memory duplication
OpenMP efficient, need extra mem

```
X_all_q_ROLEs= "q k c v"     # CPUs roles (q,k,c,v)
X_all_q_CPU  = "1 2 4 2"     # CPUs for each role
X_Threads = 4                # num threads
```

$$\chi(\mathbf{q},\omega) = [I-\chi_0(\mathbf{q},\omega)v(\mathbf{q})]^{-1}\chi_0(\mathbf{q},\omega)$$

```
X_all_q_LinAlg_INV = 64      # CPUs for Linear Alg
```

# Linear Response

$$\chi^0_{\mathbf{GG'}}(\mathbf{q},\omega) = 2\sum_{c,v}\int_{BZ}\frac{d\mathbf{k}}{(2\pi)^3}\rho^*_{cv\mathbf{k}}(\mathbf{q},\mathbf{G})\rho_{cv\mathbf{k}}(\mathbf{q},\mathbf{G'})\,f_{v\mathbf{k}-\mathbf{q}}(1-f_{c\mathbf{k}})\times$$
$$\times\left[\frac{1}{\omega+\epsilon_{v\mathbf{k}-\mathbf{q}}-\epsilon_{c\mathbf{k}}+i0^+}-\frac{1}{\omega+\epsilon_{c\mathbf{k}}-\epsilon_{v\mathbf{k}-\mathbf{q}}-i0^+}\right]$$

q transferred momenta (MPI q) | Xo bands (MPI c,v) | k momenta (MPI k) | space variables

```
X_all_q_ROLEs= "q k c v"     # CPUs roles (q,k,c,v)
X_all_q_CPU  = "1 2 4 2"     # CPUs for each role
X_Threads = 4                # num threads
```

MPI-cv best memory distribution
MPI-k as efficient, some mem dupl
MPI-q may lead to load unbalance, and memory duplication
OpenMP efficient, need extra mem

$$\chi(\mathbf{q},\omega) = [I-\chi_0(\mathbf{q},\omega)v(\mathbf{q})]^{-1}\chi_0(\mathbf{q},\omega)$$

```
X_all_q_LinAlg_INV = 64      # CPUs for Linear Alg
```

Recent changes: OpenMP at higher level → better performance at price of memory

To come soon: MPI also at lower level → better memory distribution

# Exchange self-energy

$$\Sigma^x_{n\mathbf{k}} = \langle n\mathbf{k}|\Sigma^x|n\mathbf{k}\rangle = -\sum_m \int_{BZ} \frac{d\mathbf{q}}{(2\pi)^3} \sum_{\mathbf{G}} v(\mathbf{q}+\mathbf{G})|\rho_{nm}(\mathbf{k},\mathbf{q},\mathbf{G})|^2 f_{m(\mathbf{k}-\mathbf{q})}$$

QP states (MPI qp)

G bands (MPI b)

q transferred momenta (MPI q)

space variables (OMP SE_T)

SE_ROLEs= “q qp b”
SE_CPU = “1 2 8”
SE_Threads = 4

# CPUs roles (q,qp,b)
# CPUs for each role
# num threads for self-energy calc

| | |
|---|---|
| MPI-b | best memory distribution VALENCE |
| MPI-qp | no communication, mem repl |
| MPI-q | usually leads to load unbalance |
| OpenMP | very efficient up to large number of threads |

# GW (corr) self-energy

$$\Sigma^c_{n\mathbf{k}}(\omega) = \langle n\mathbf{k}|\Sigma^c|n\mathbf{k}\rangle = i\sum_m \int_{BZ} \frac{d\mathbf{q}}{(2\pi)^3} \sum_{\mathbf{G},\mathbf{G}'} \frac{4\pi}{|\mathbf{q}+\mathbf{G}|^2} \rho_{nm}(\mathbf{k},\mathbf{q},\mathbf{G})\rho^*_{nm}(\mathbf{k},\mathbf{q},\mathbf{G}')$$
$$\times \int d\omega' G^0_{m\mathbf{k}-\mathbf{q}}(\omega-\omega')\epsilon^{-1}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega')$$

QP states (MPI qp)

G bands (MPI b)

q transferred momenta (MPI q)

space variables (OMP SE_T)

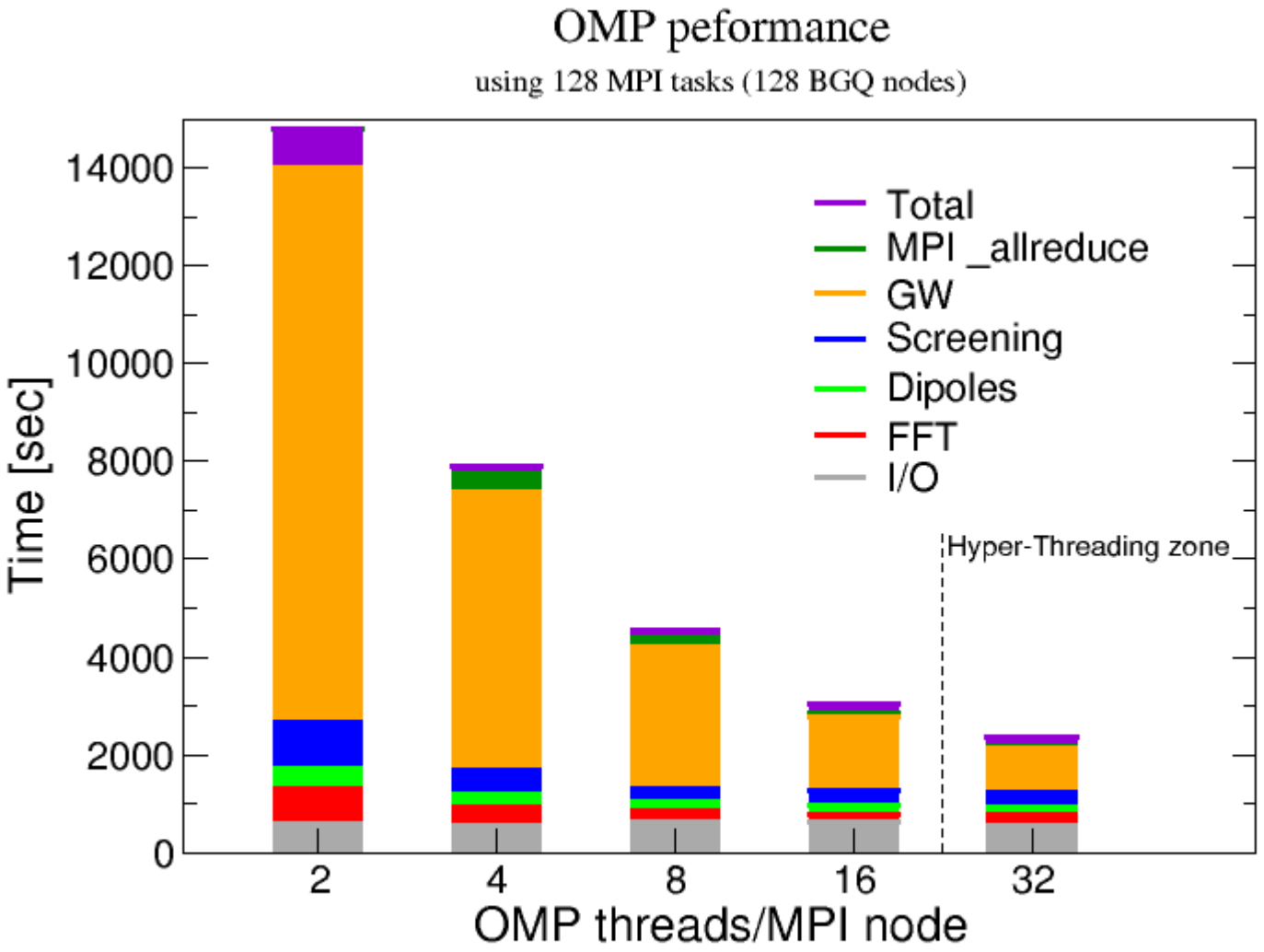
SE_ROLEs= “q qp b” # CPUs roles (q,qp,b)
SE_CPU = “1 2 8” # CPUs for each role
SE_Threads = 4 # num threads for self-energy calc

| | |
|---|---|
| MPI-b | best memory distribution |
| MPI-qp | no communication, mem repl |
| MPI-q | usually leads to load unbalance |
| OpenMP | very efficient up to large number of threads |

# outline

- motivations: **HPC, exascale**
- paradigms: **MPI, OpenMP**, and more
- Yambo **parallelization strategies**
- **Performance**

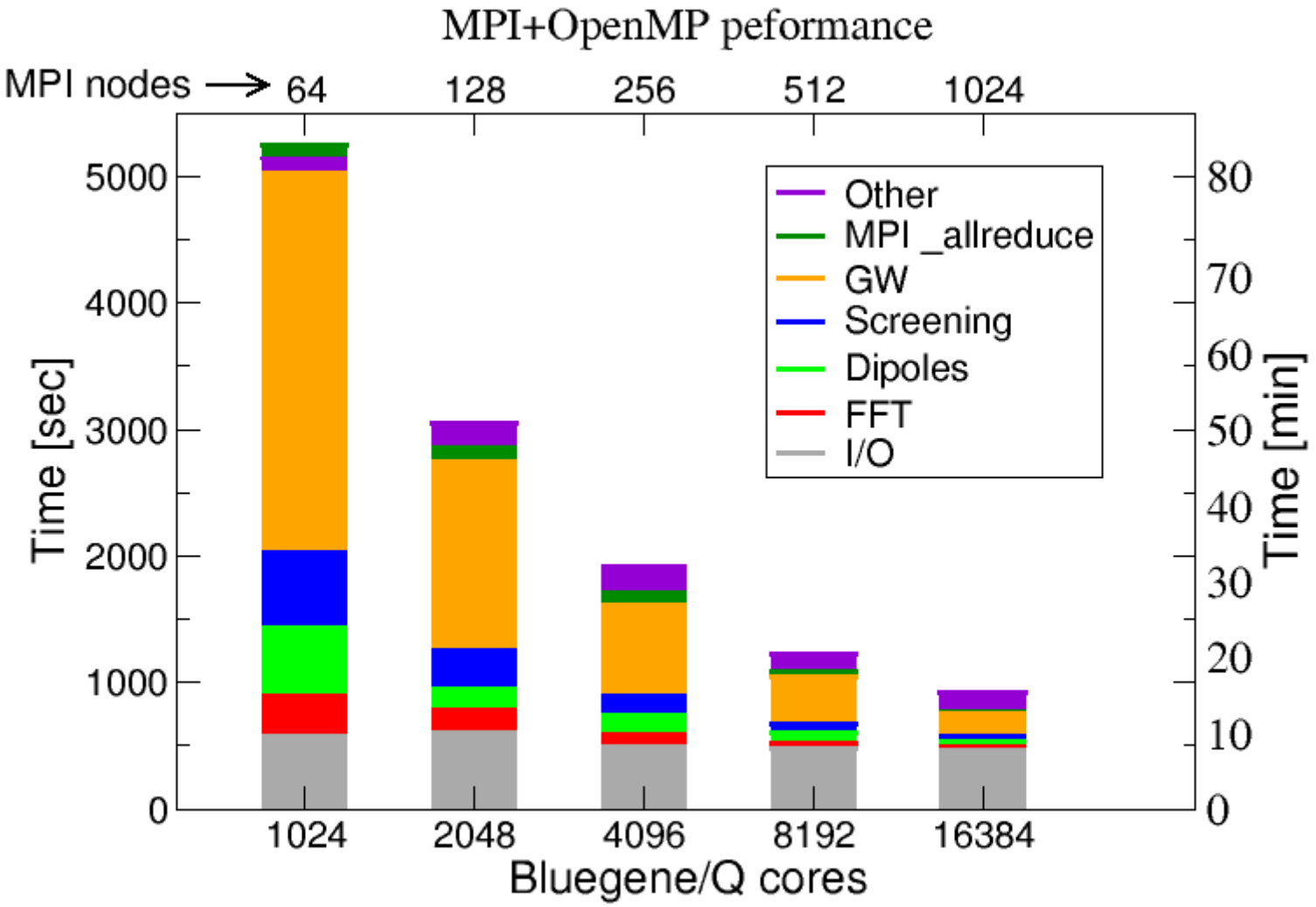
# yambo scaling on BGQ





Very first scaling tests … now some years old

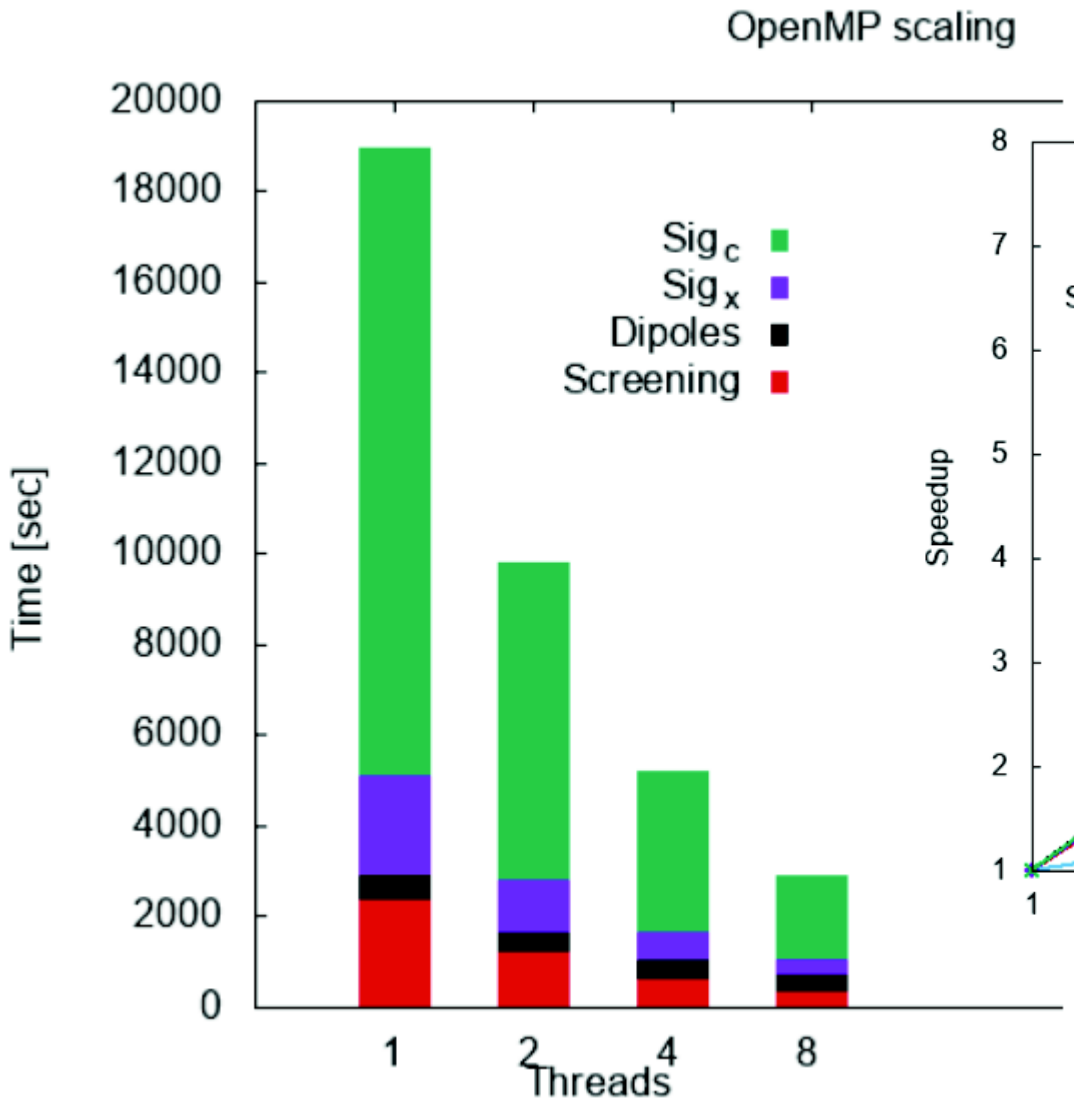
# yambo scaling on BGQ

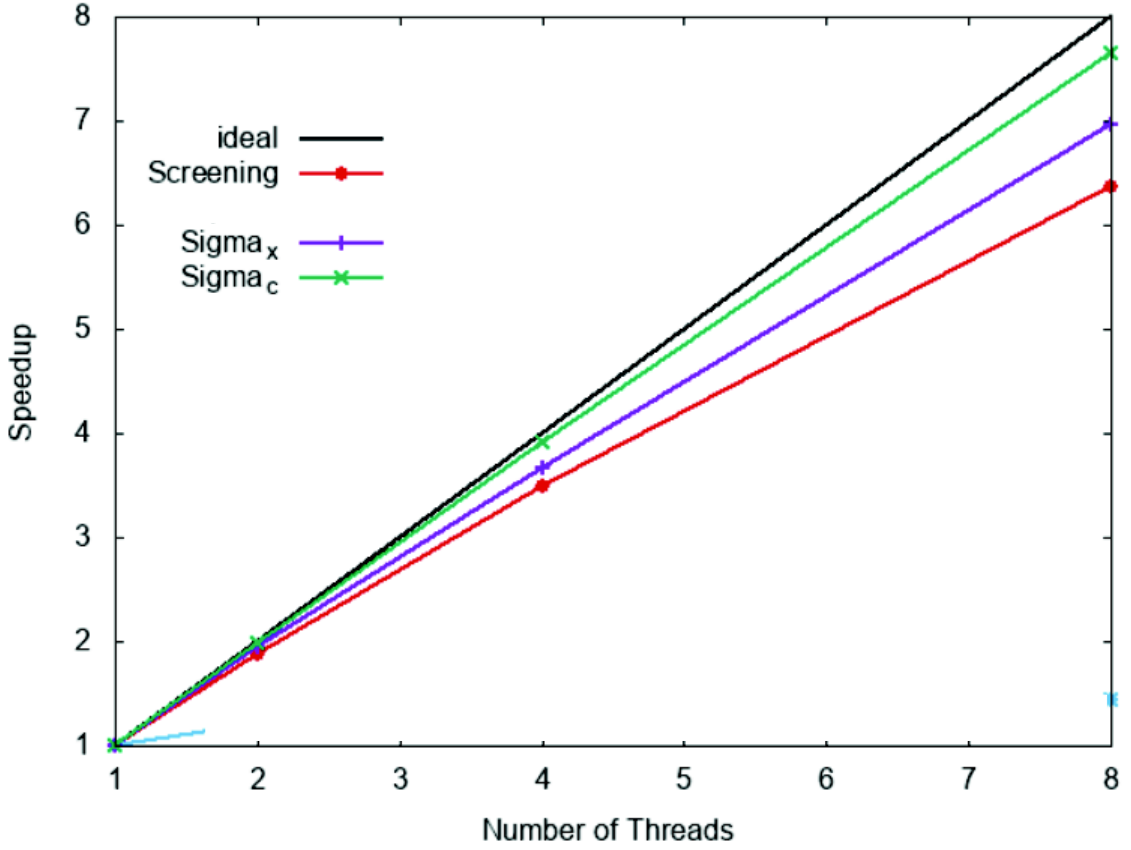




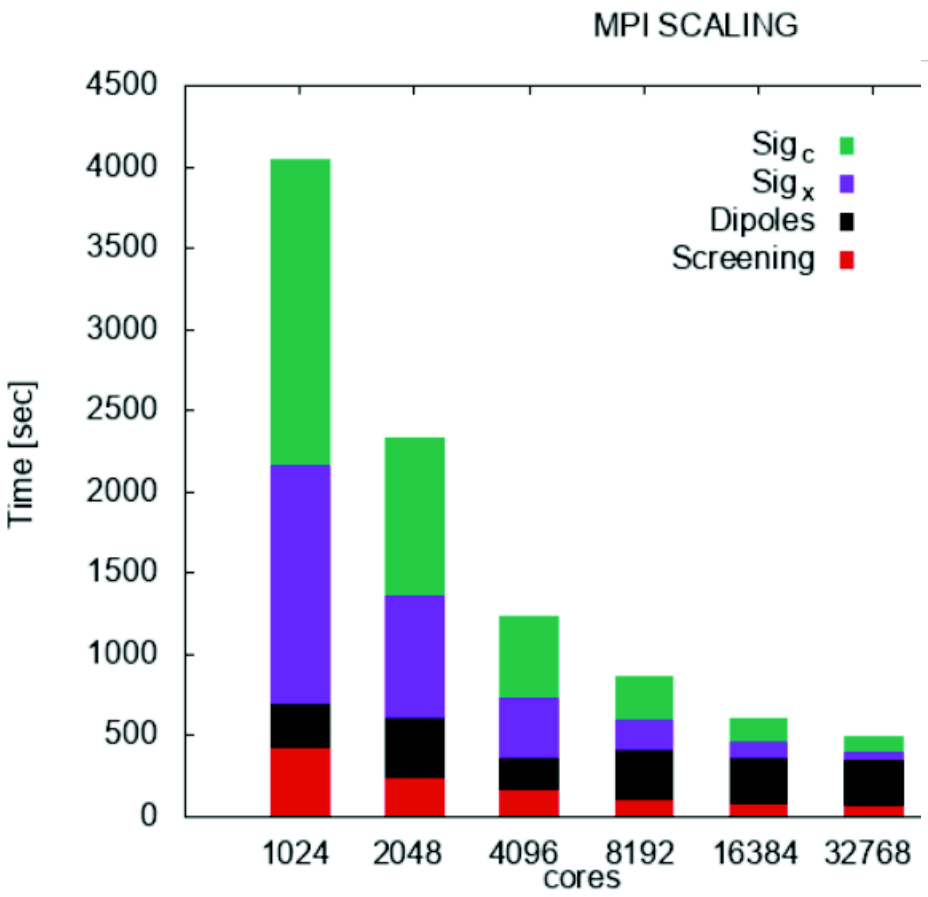
Very first scaling tests … now some years old

# yambo scaling on BGQ

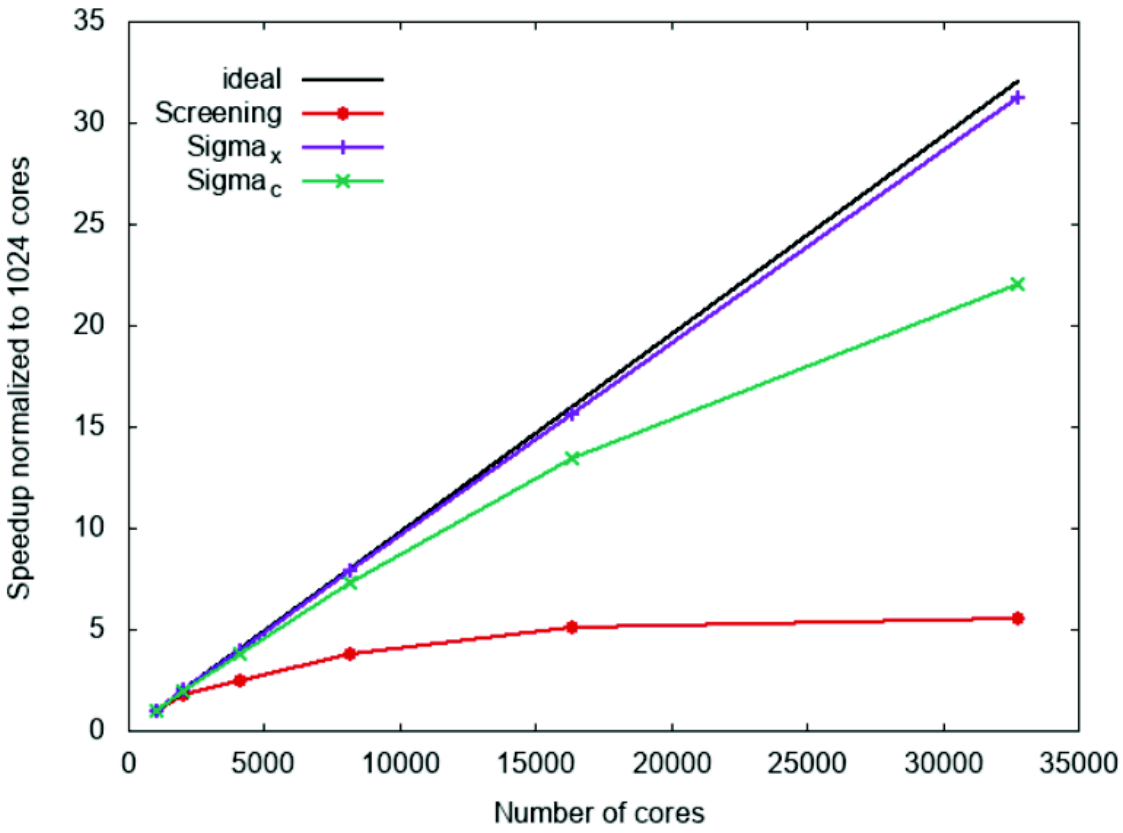


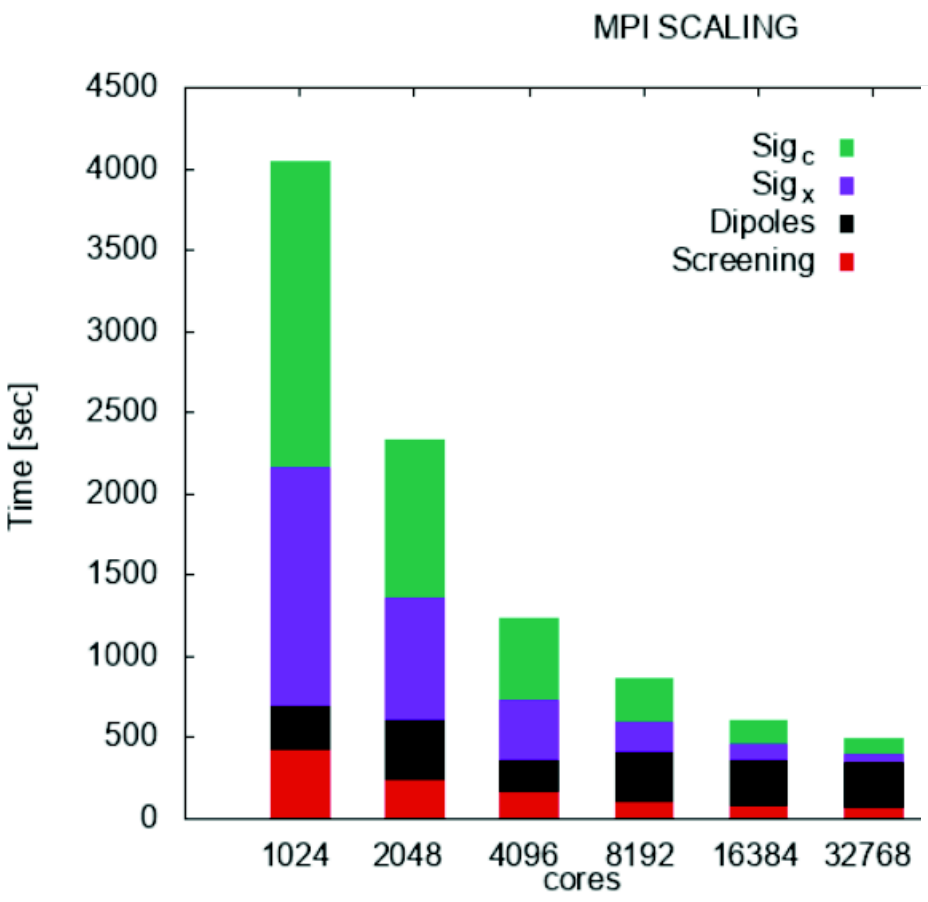
# yambo scaling on BGQ



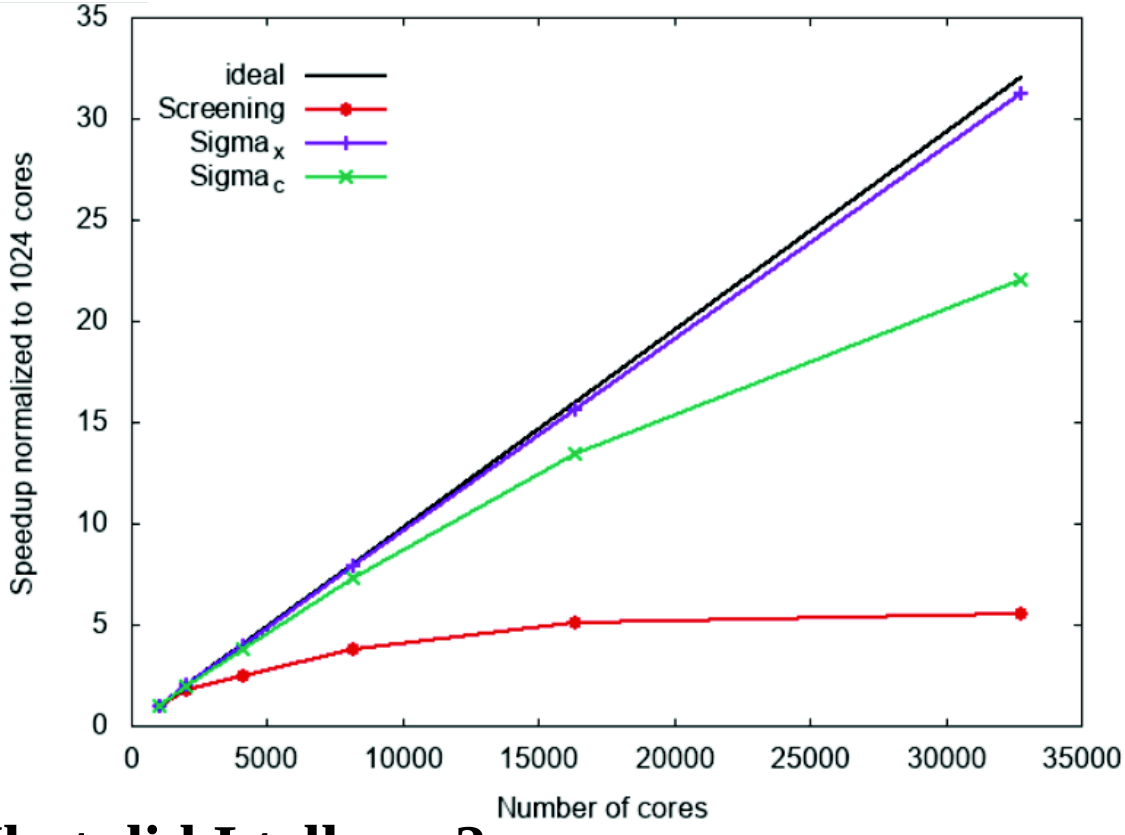

Using 16 threads per node



# yambo scaling on BGQ




Using 16 threads per node




**“What did I tell you? You can run yambo on 32'768 cores”**

**Andrea**

**Oh yeah, we made it! 2 racks of BGQ**

**Carlo**

CINECA

# yambo: scalapack

- LinRes functions

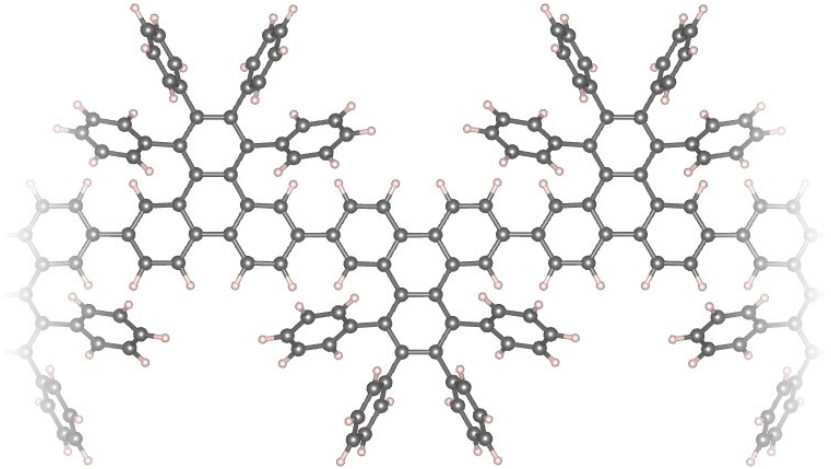
$$\begin{aligned} \chi &= \chi_0 + \chi_0 v \chi \\ &= [I - \chi_0 v]^{-1} \chi_0 \end{aligned}$$

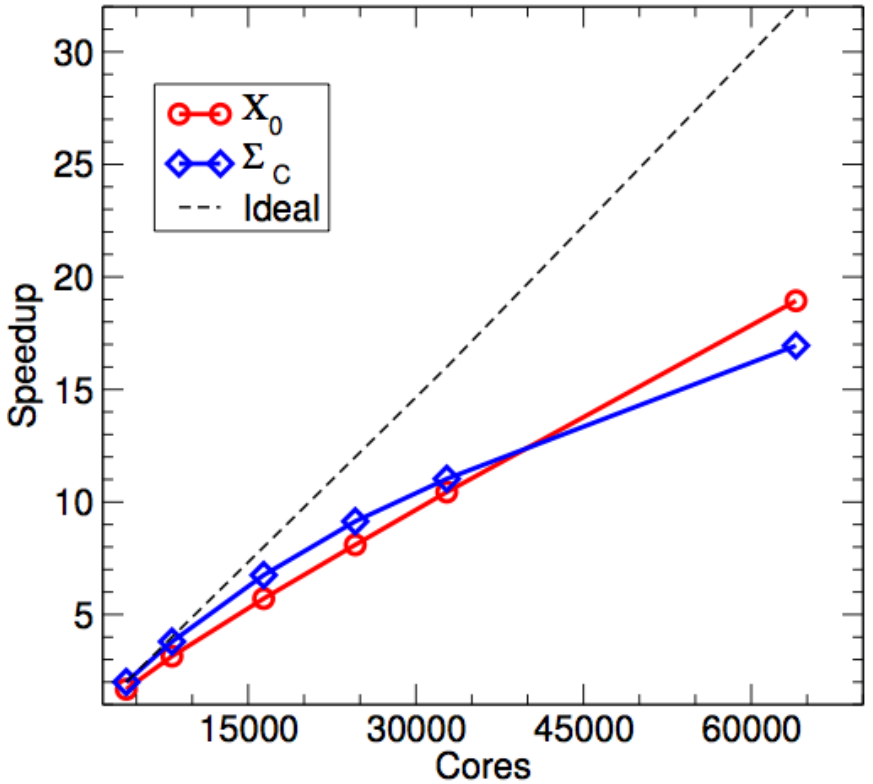
| SLK grid | Lin. system [sec] | X_s routine [sec] | Lin. system [sec] | X_s routine [sec] |
|---|---|---|---|---|
| | N = 3791 | | N = 10443 | |
| $1 \times 1$ | 7 | 14 | 75 | 304 |
| $2 \times 2$ | 4.87 | 10 | 88 | 182 |
| $3 \times 3$ | 2.99 | 5 | 49 | 104 |
| $4 \times 4$ | 2.02 | 4 | 32 | 92 |
| $5 \times 5$ | 1.75 | 4 | 23 | 52 |
| $6 \times 6$ | 1.56 | 3 | 19 | 44 |
| $8 \times 8$ | 1.17 | 3 | 13 | 32 |

Table 7: Timing to compute the reducible response function $\chi$ given the irreducible response $\chi_0$, by solving the linear system $(1 - v\chi_0)\chi = \chi_0$. Two different sizes of the involved matrices (N=3791 and 10443, respectively) have been considered. Calculations have been performed for single-precision complex algebra using scalapack on a Intel-Xeon (E5-2640 v3 @ 2.60GHz) architecture (4 nodes with 16 cores per node).

# a real life example

- Yambo **single GW calculation** scaling up to 1000 KNL nodes (~ 3 Pfl/s, 65’536 cores)
- hybrid **MPI+OpenMP+scaLapack**
- Calculations relevant for an **active research field** (graphene nanoribbons)
- Performed on a brand new architecture (**Intel KNL** @ Marconi)



CNR
Istituto di Struttura
della Materia

# Thank you for your attention

*www.yambo-code.org*

*en.wikipedia.org/wiki/YAMBO_code*

*www.facebook.com/yambocode*

*plus.google.com/+YambocodeOrgPage*

*github.com/yambo-code*
*github.com/henriquemiranda/yambopy*