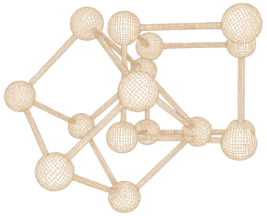


Advanced School on
QUANTUM MODELLING
of Materials with CRYSTAL

UNIVERSITÀ
DI TORINO

Parallel Computing in the CRYSTAL Code

Giacomo Ambrogio

Volta Redonda, State of Rio de Janeiro
Brazil

giacomo.ambrogio@unito.it

QMMC 2026

Outline

- **Why parallel?**
  - The serial problem
  - Parallel **hardware** architectures
- **When to use parallel computing**
  - The **overhead**
  - Parallel Implementations in Crystal
    - The algorithm (SCF)
    - **PCRYSTAL**
    - **MPPCRYSTAL**
    - Exmples
    - **OpenMP**
- Using GPUs: **GCRYSTAL**

Why Parallel?

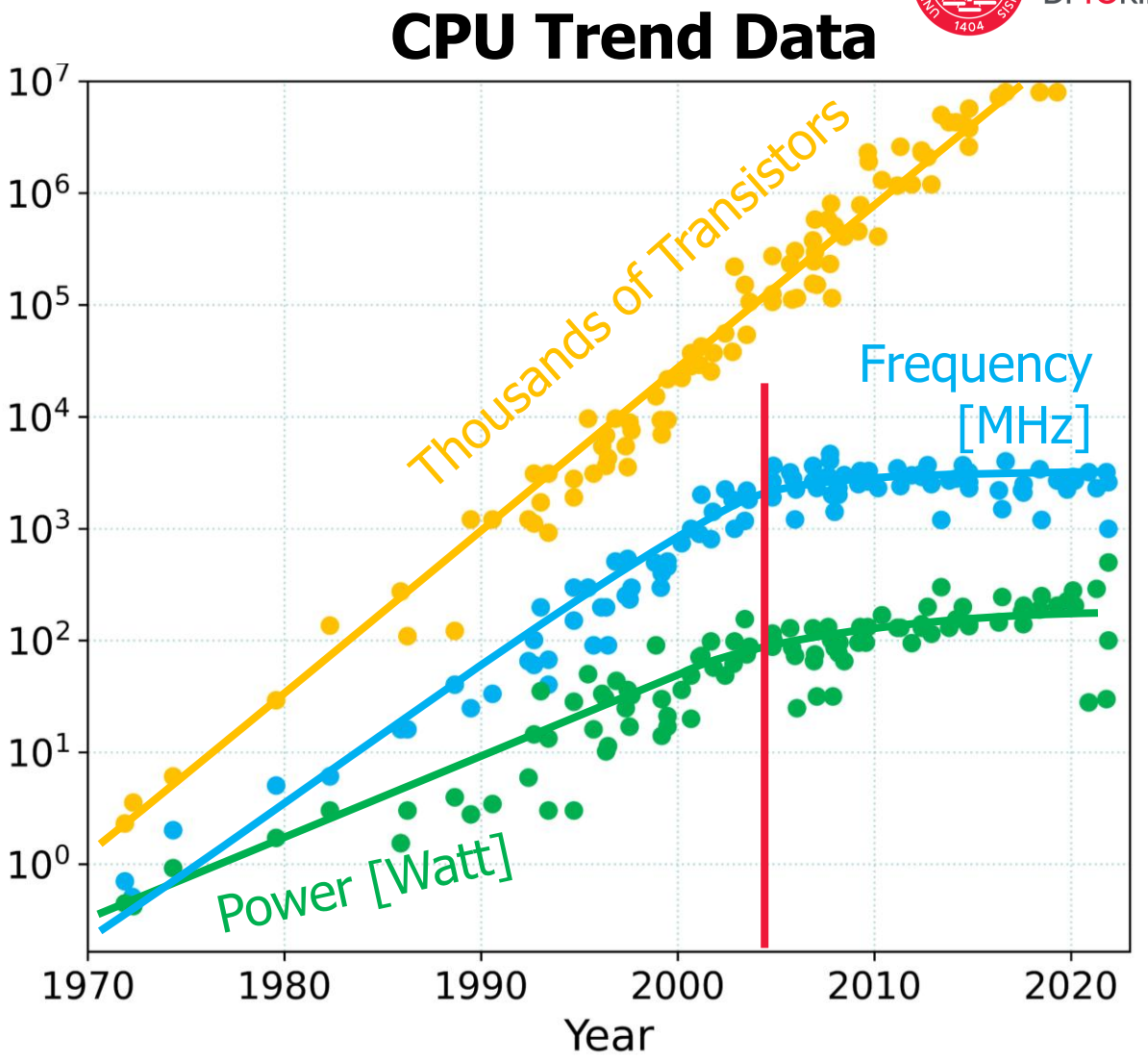
- Why do we need parallel computers?
- Why should you be interested in using parallel computers?

The Serial Problem

Moore's Law: number of **transistors** in microchips *doubles* roughly every two years

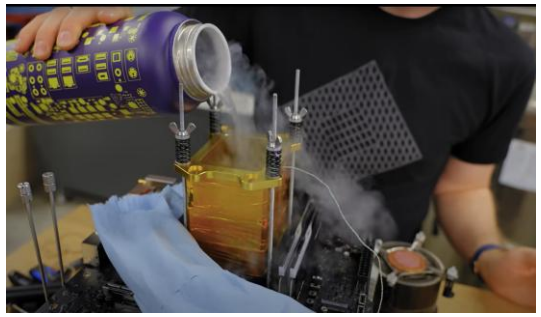
At first, we were consistently able to increase the **frequency** (operations performed per second).

Around 2002, hardware reached a **physical limitation**
increasing power → silicon chip would melt



The Serial Problem

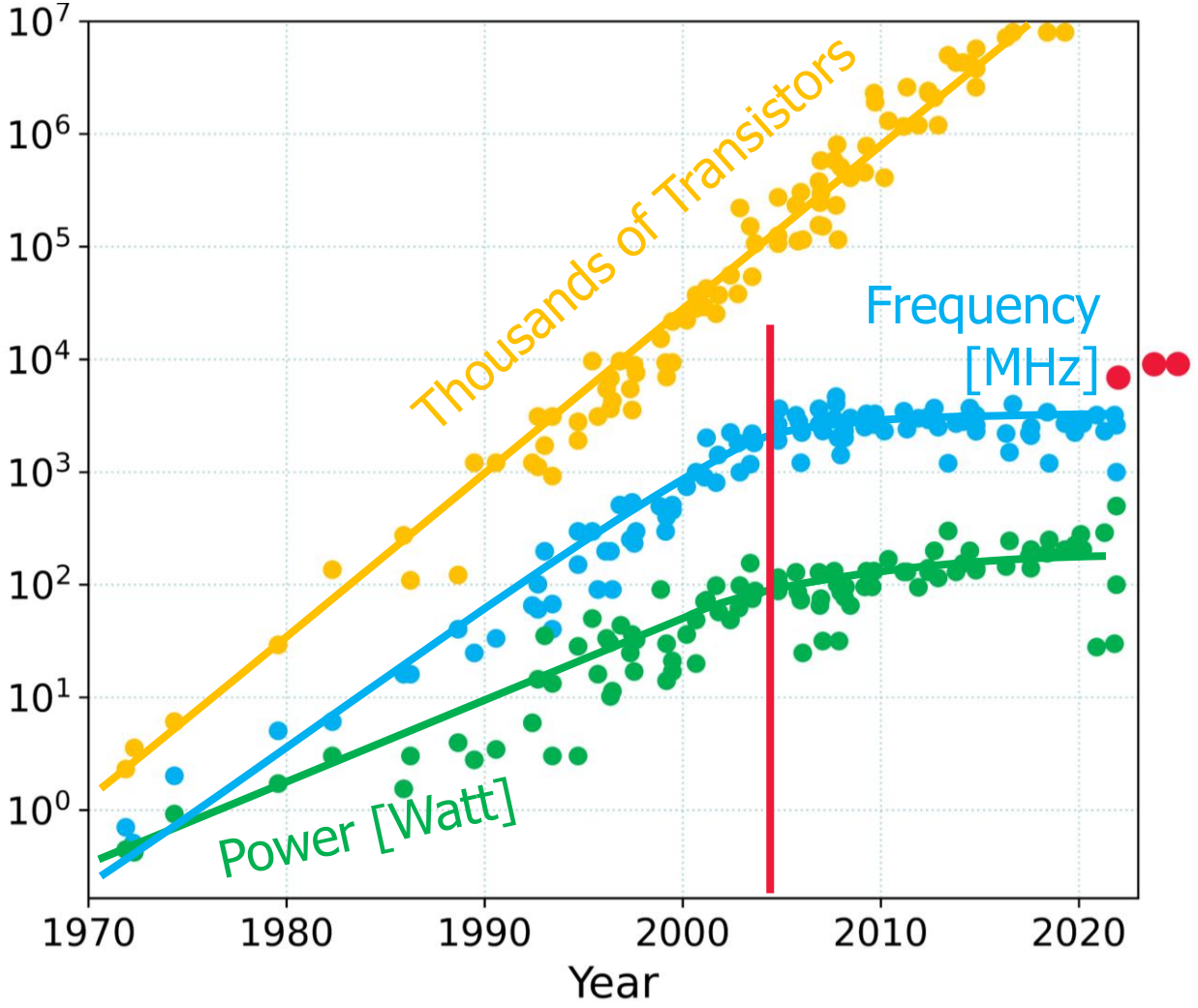
Some crazy attempts to keep low temperature: **Liquid Nitrogen**



Intel
i9-12900K
6.9 GHz
2022

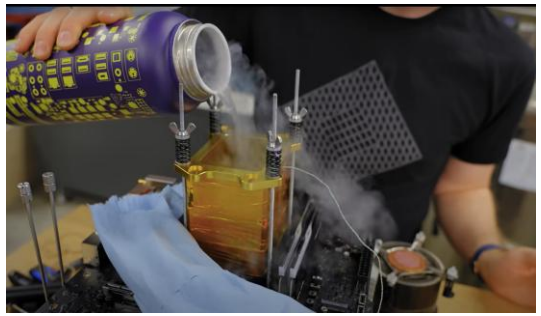
Intel
i9-14900K
9.04 GHz
2023

Intel
i9-14900KS
9.12 GHz
2025



The Serial Problem

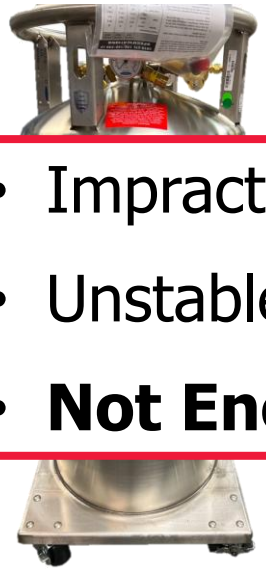
Some crazy attempts to keep low temperature: **Liquid Nitrogen**



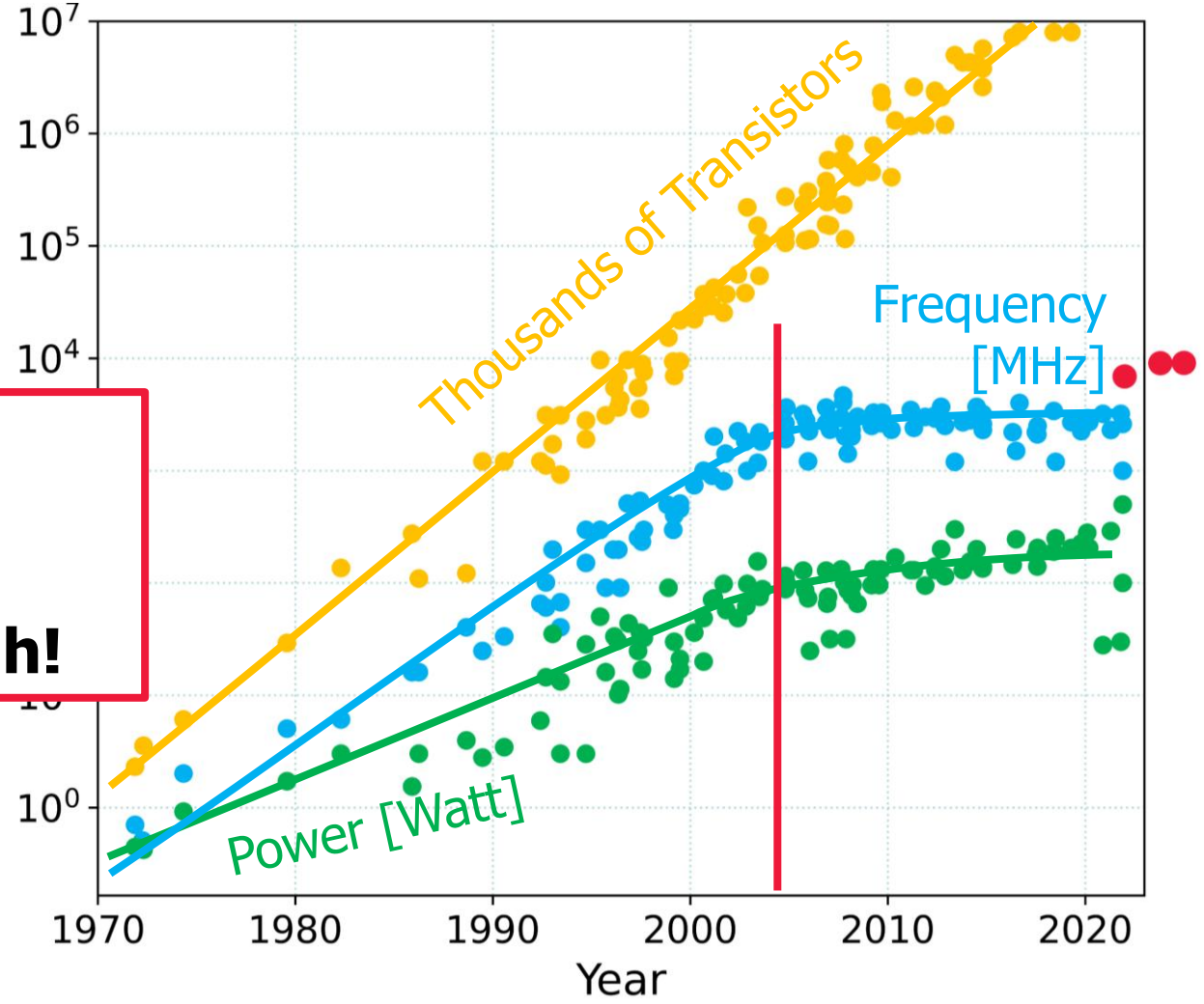
Intel
i9-12900K
6.9 GHz
2022

Intel
i9-14900K
9.04 GHz
2023

Intel
i9-14900KS
9.12 GHz
2025



- Impractical
- Unstable
- **Not Enough!**



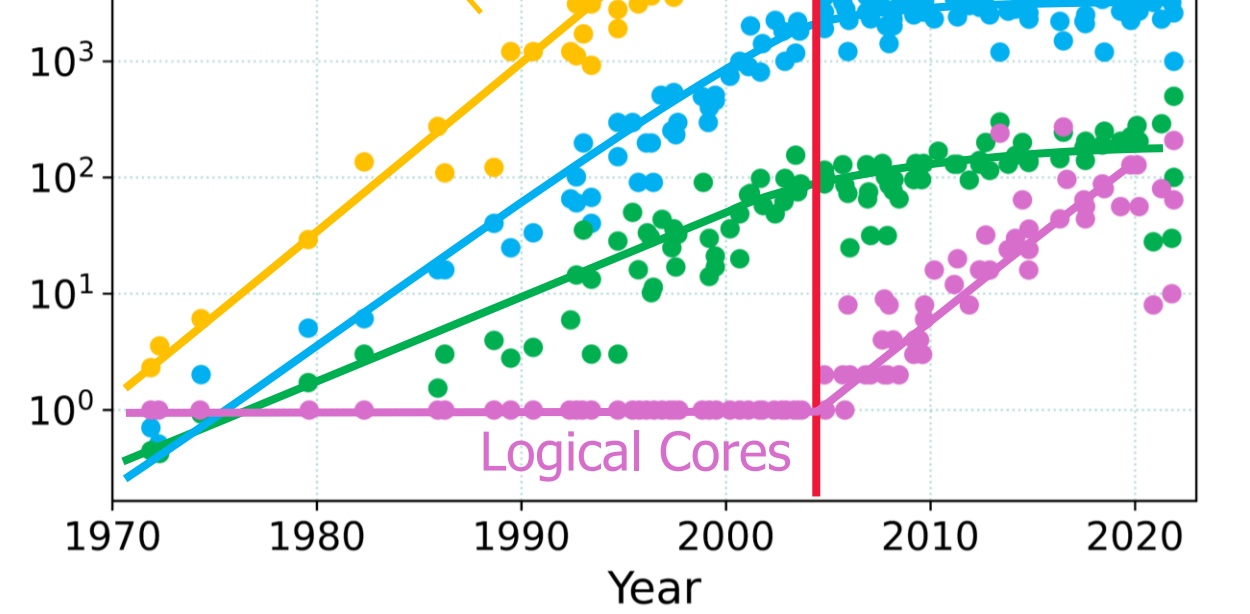
The Serial Problem

The solution is to go **Parallel**

More computing units
(in the same chip)
→ More results in the same time

CORES

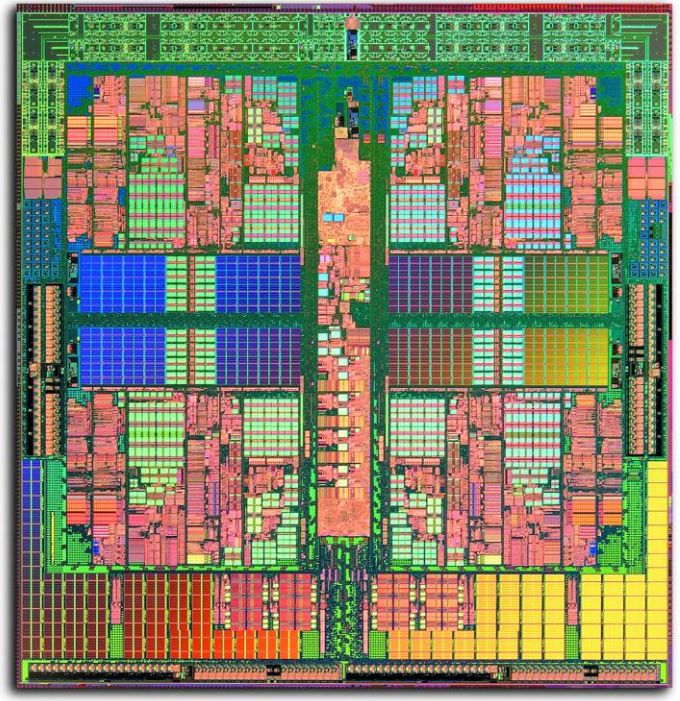
All modern computers and electronic devices are parallel:



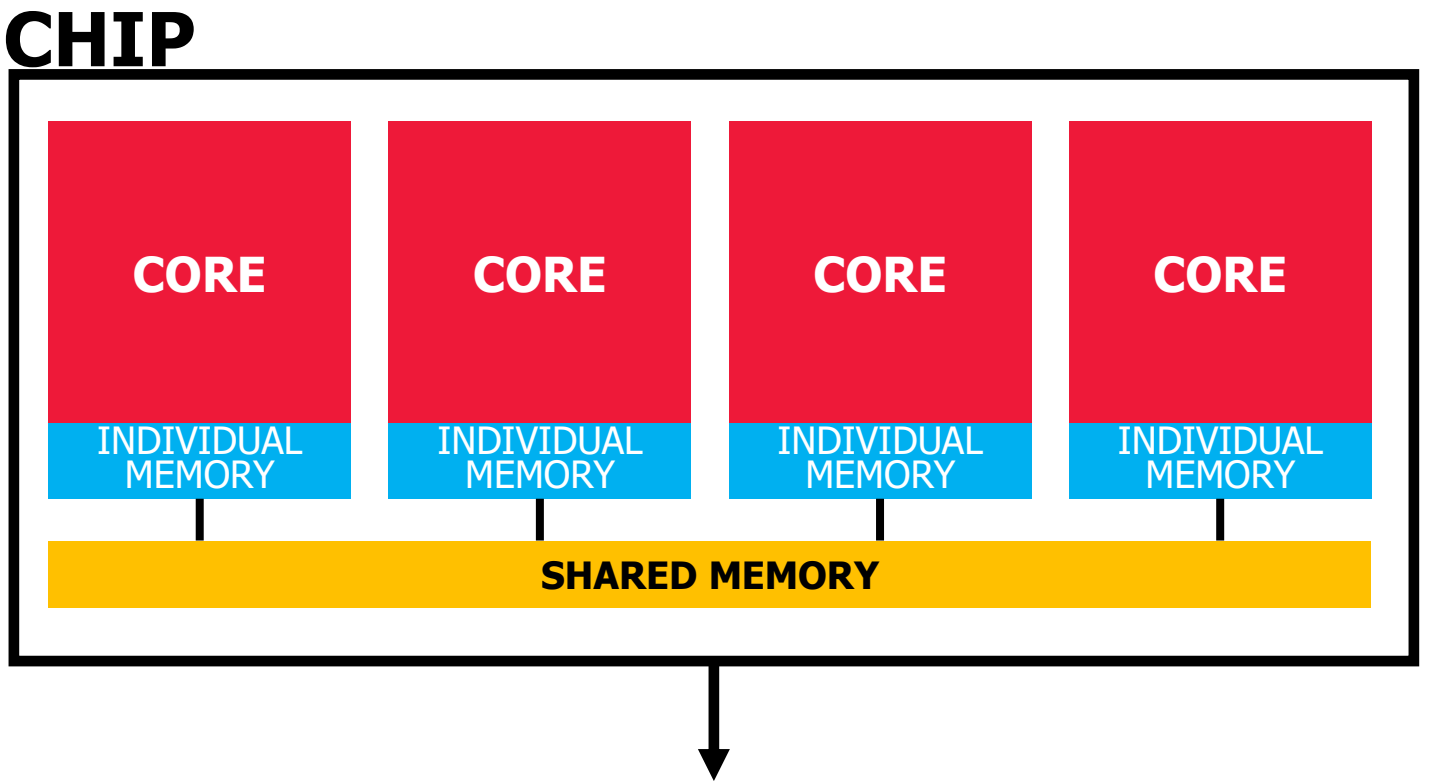
Inside Parallel Computers

The main component is the
Multicore CPU

→ From 2 to **192** cores per Chip!



Quad-Core AMD
Opteron
2007



Inside Large Parallel Computers

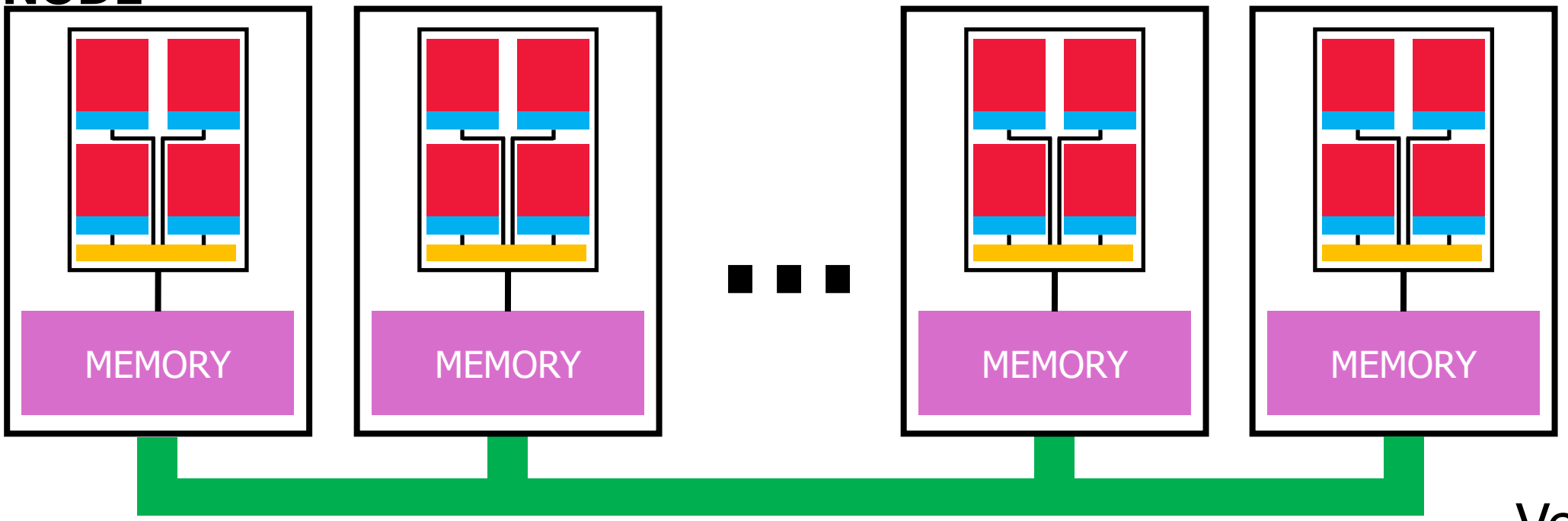
Multicore processors are clustered together with an **interconnect**

→ ethernet, InfiniBand

Supercomputer

High Performance Computing

NODE



Same components as in small systems

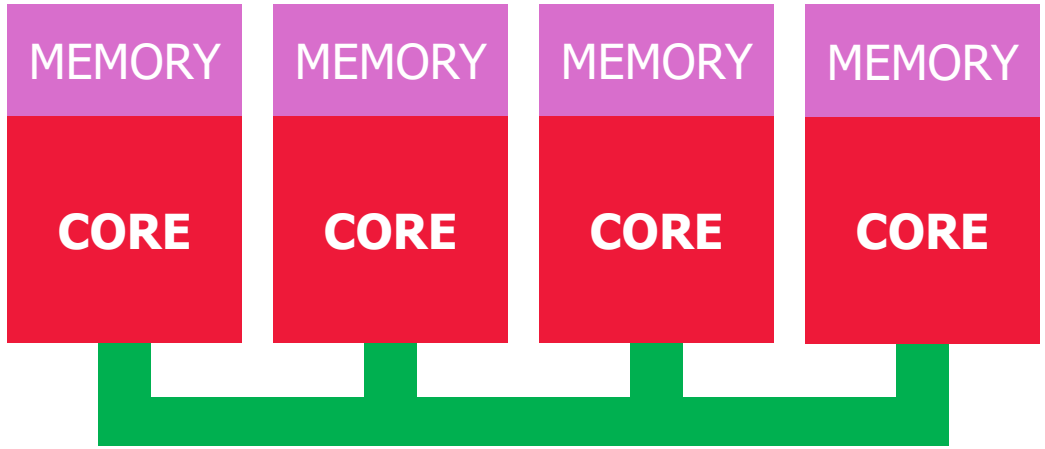
- More cores
- More memory

Very expensive!

Simplify the Model (for now)

Consider a simplified view: a number of **cores** connected via an **interconnect**, each with its own **private memory**

- Ignore multicore
- Ignore limits of interconnection



Why Parallel?

- Faster **time to solution**
  Jobs that takes days or weeks → hours
- More **Memory** so larger jobs
  32 nodes means 32 times more memory
  Hopefully, one can run a **larger jobs**
- **BUT** more complex **to run**
- Depends on **local setup**
- And more **difficult** to get the best out of the program

→ I will try to give some **tips** on how to do this!

Always Parallel?

You *probably* always want to run parallel. However,

- **10 people** do not necessarily get a **job done 10 times quicker**

→ You need **enough work** to keep 10 people busy!

- Similarly, **10 cores** does not mean the computation will run **10 times quicker**

→ You need to have enough work to **keep 10 cores busy**!

Do not always **use all** available **cores**

- There might be sufficient work for 10 cores, but not for 100
- **Overhead**: using more cores might make your program SLOW DOWN

The Overhead

Work

$$\sim O\left(\frac{N}{P}\right)$$

Overhead

$$\sim O\left(P\right)$$

P Number of cores

Reasons for overhead in parallel programs include:

- **Amdahl's law**: not all the program is parallelised
- **Load balance:** not all tasks are equal
- The **interconnect** (necessary, but it is slow)
- Memory **bandwidth**
- **I/O**
- O/S jitter
- Phase of the moon
- Black magic
- The computer hates you

The Overhead

Work

$$\sim O\left(\frac{N}{P}\right)$$

Overhead

$$\sim O\left(P\right)$$

P Number of cores

As N increases the overhead becomes **less important**

Parallel Computing Is **Best** For Solving **Large Systems**

Trying To Quantify It

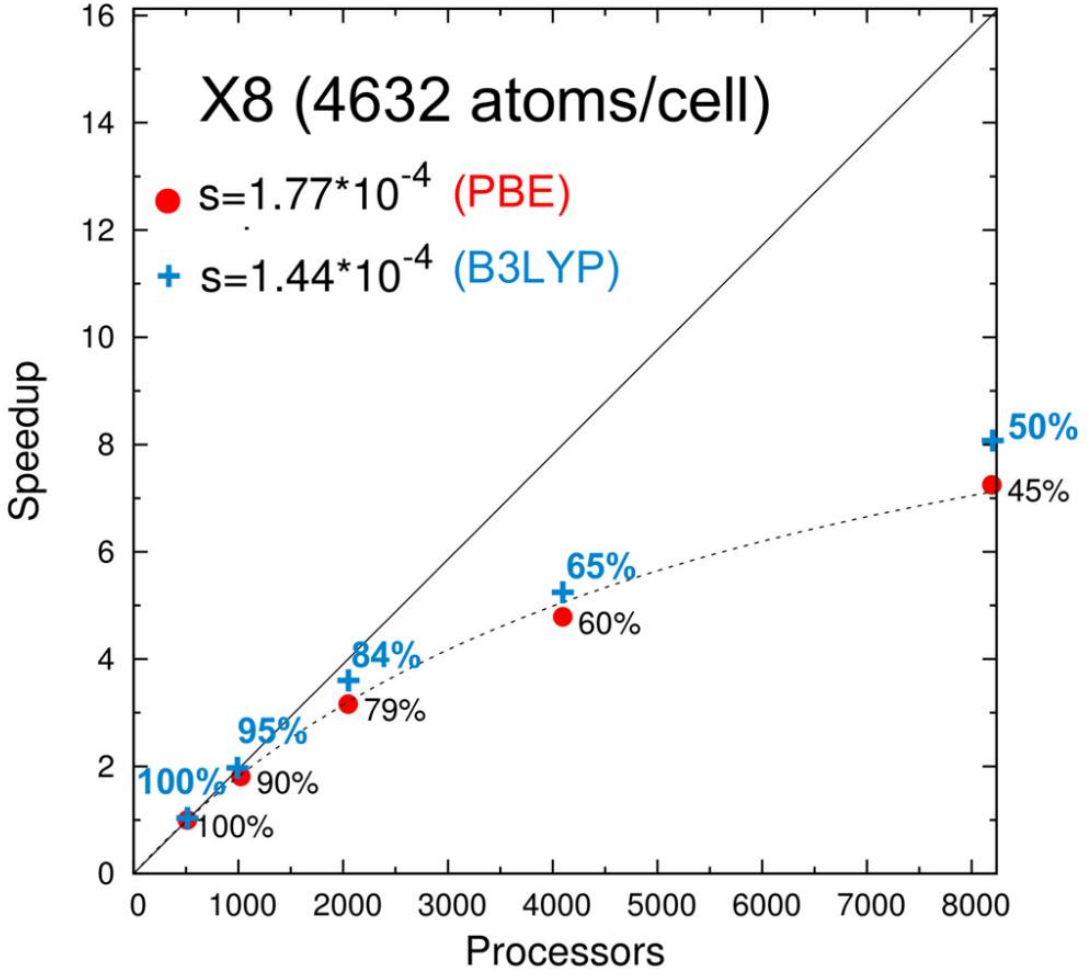
Each job have its own **characteristics**

→ may be best to run on a **different number of cores**

What do we mean by "best"?

Need to measure something!

- **Speed up**: applies to any code
- **CRYSTAL**: SCF cycles per unit time



Parallel Implementations in CRYSTAL

UNIVERSITÀ
DI TORINO

PCRYSTAL

- **Replicated data**
- All of CRYSTAL implemented
- Good for **small-medium systems** on **small-medium core counts**
- *Reference Implementation*

MPPCRYSTAL

- **Distributed data**
- Most of CRYSTAL implemented
- Good for **medium-large problems** on **medium-very large core counts**

OpenMP

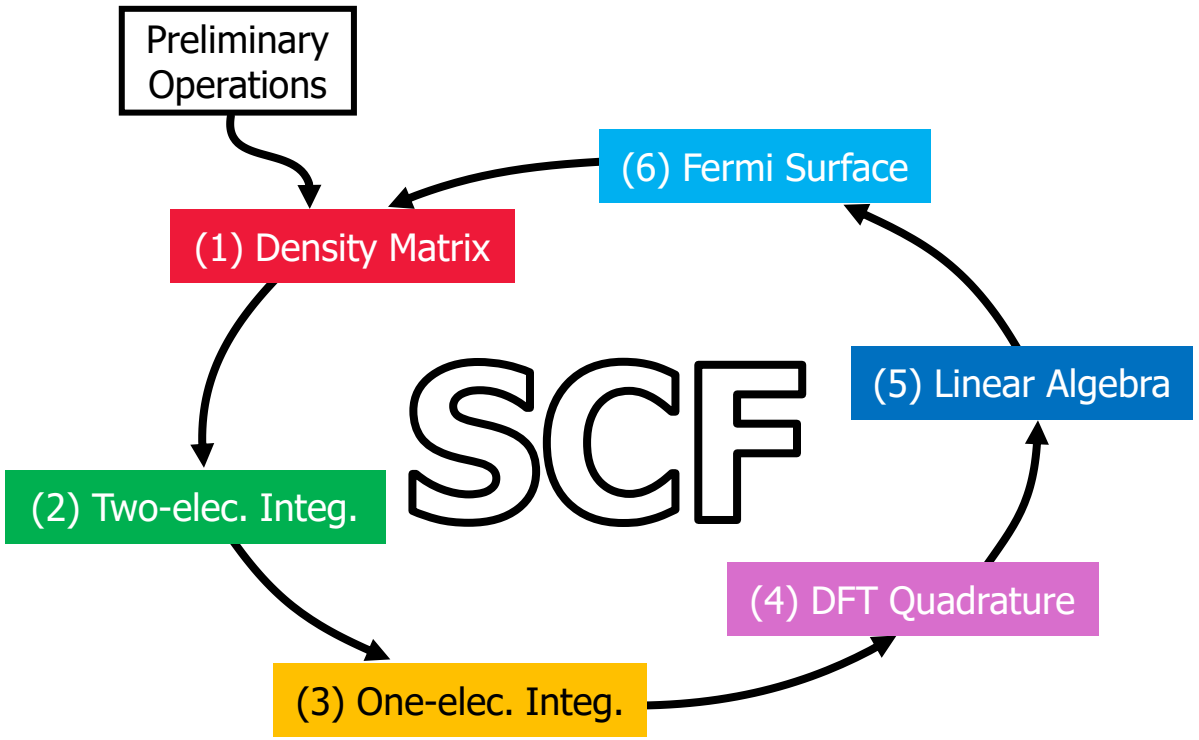
- New in CRYSTAL23
- More later

GCRYSTAL

- In **development** for **next release**
- More later

Basic Algorithm

Single point **energy**
(and **wavefunction**) calculation:



Repeat until **convergence** on energy

1. Start with guess of **Density matrix**
2./3. The **Hamiltonian** matrix $\mathbf{H_g}$ is obtained as a sum of **independent integrals**
4. **DFT terms** are added to $\mathbf{H_g}$
5. $\mathbf{H_g}$ is converted in reciprocal space: set of $\mathbf{H_k}$ matrices obtained. $\mathbf{H_k}$ is orthogonalized.
   Solve for $\mathbf{H_k C_k = S_k C_k E_k}$
6. Compute **Fermi Surface**
1. Form new **Density Matrix** from $\mathbf{C_k}$

PCRYSTAL

Based on MPI (4.x.x) standard for parallelism

Should compile and work **everywhere**

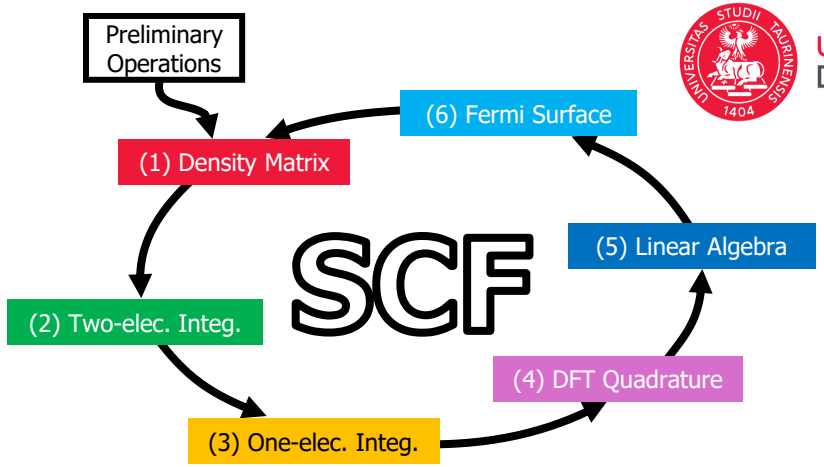
Replicated Data

- Each processor has a **complete copy** of "all" objects used by the code
- Make **implementation simple**
- But memory limits size of system that can be studied

PCRYSTAL

Based on MPI (4.x.x) standard for parallelism

Should compile and work **everywhere**



Replicated Data

- Each processor has a **complete copy** of "all" objects used by the code
- Make **implementation simple**
- But memory limits size of system that can be studied

Generating the Hamiltonian $\mathbf{H}_g$ matrix

- Each integral is **independent**
  → give a subset to each one of the cores
- DFT: **integration** over a **grid** of independent points
  → each core computes a different set of points

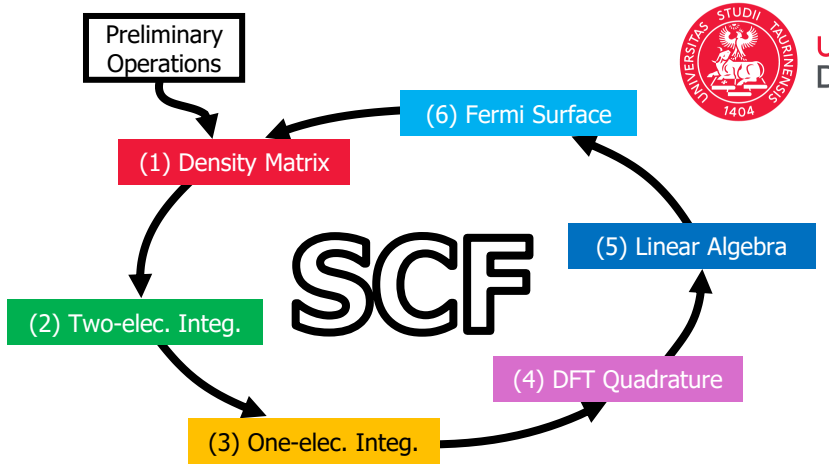
Almost perfectly parallel!

→ Limit on scaling is load balancing

PCRYSTAL

Based on MPI (4.x.x) standard for parallelism

Should compile and work **everywhere**



Replicated Data

- Each processor has a **complete copy** of "all" objects used by the code
- Make **implementation simple**
- But memory limits size of system that can be studied

Solving Secular Equation $\mathbf{H_k C_k = S_k C_k E_k}$

- Each **k** point is **independent**
  → each core can deal with a subset

But number of **k** points **limits parallelism**

Large systems require few **k** points (only 1 in the limit)

Each core **allocate one full matrix** in memory

→ **Limit in size** despite **number of cores used**

PCRYSTAL – How to run

Ultimately depends on your **local setup**

General command line is:

```
>> mpirun -np 4 /path/to/Pcrystal
```

Usually, you will need a **batch script** if running on a cluster

Strictly **NO changes** required to INPUT file

But it *must* be named INPUT, and be located in same directory where you launch the code

PCRYSTAL – Summary

In general **scales well** where integrals dominate
(small-medium systems)

k points **limit scaling** in linear algebra section

Limit on size due to **memory/core**
(same as serial CRYSTAL)

→ Some tricks can be used to partially overcome this

MPPCRYSTAL

Also based on MPI (4.x.x) standard for parallelism

Exploit **ScaLAPACK library** for linear algebra
www.netlib.org/scalapack/ (requirement)

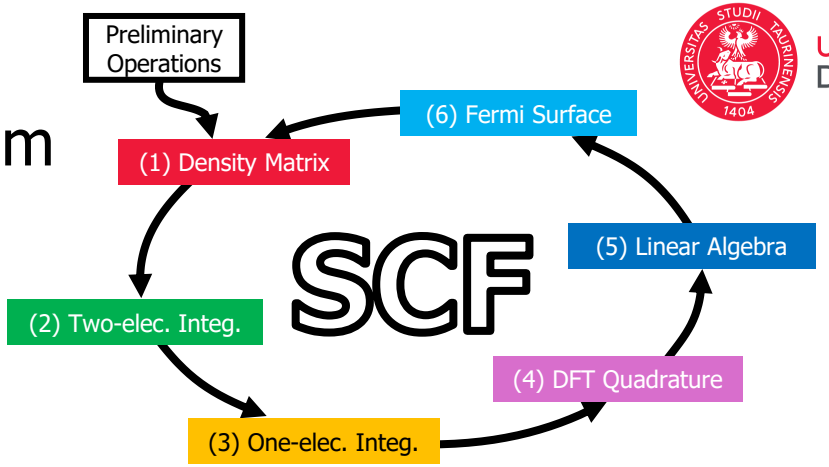
Distributed Data

- Each core holds only a **part** of large data structures
- More **complex to implement**
- More cores and memory → **Larger jobs**

MPPCRYSTAL

Also based on MPI (4.x.x) standard for parallelism

Exploit **ScaLAPACK library** for linear algebra
www.netlib.org/scalapack/ (requirement)



Distributed Data

- Each core holds only a **part** of large data structures
- More **complex to implement**
- More cores and memory → **Larger jobs**

Construction of $\mathbf{H}_g$ matrix same as PCRYSTAL

Linear Algebra

- Exploit **k** point **parallelism**
- Each **k** point exploit **further levels of parallelism**
- More communication → **More overhead**
  Work scales $O(N^3)$, communications scale $O(N^2)$
- **No I/O** (just store everything in memory)

MPPCRYSTAL – Summary

Good for **larger systems** and **large core counts**

Overhead can dominate in smaller systems

Rough estimate: Use MPP if $\frac{\text{spins} \times \mathbf{k} \text{ points} \times \text{AO}}{\text{cores}} \geq 20$

(Very machine and case dependent)

Distributed data means that given enough cores can solve *any* system

Not all of CRYSTAL is **implemented**

(Will fail quickly and cleanly if the requested feature is not available)

Running is just the same as PCRYSTAL

Properties

Both PCRYSTAL and MPPCRYSTAL write **final output** files (fort.9 i.e. wavefunction) that are in **exactly the same** form as the serial code

- **Run PROPERTIES just as before**
- But might take a long time and take a lot of memory

→ **PPROPERTIES**, *replicated* data like PCRYSTAL

→ **MPPPROPERTIES**, *Distributed* data like MPPCRYSTAL

- Supports `NEWK`, `BAND`, `DOS`, `ECH3`, `POT3`

How To Get The Best Out Of Them

Basic procedure is:

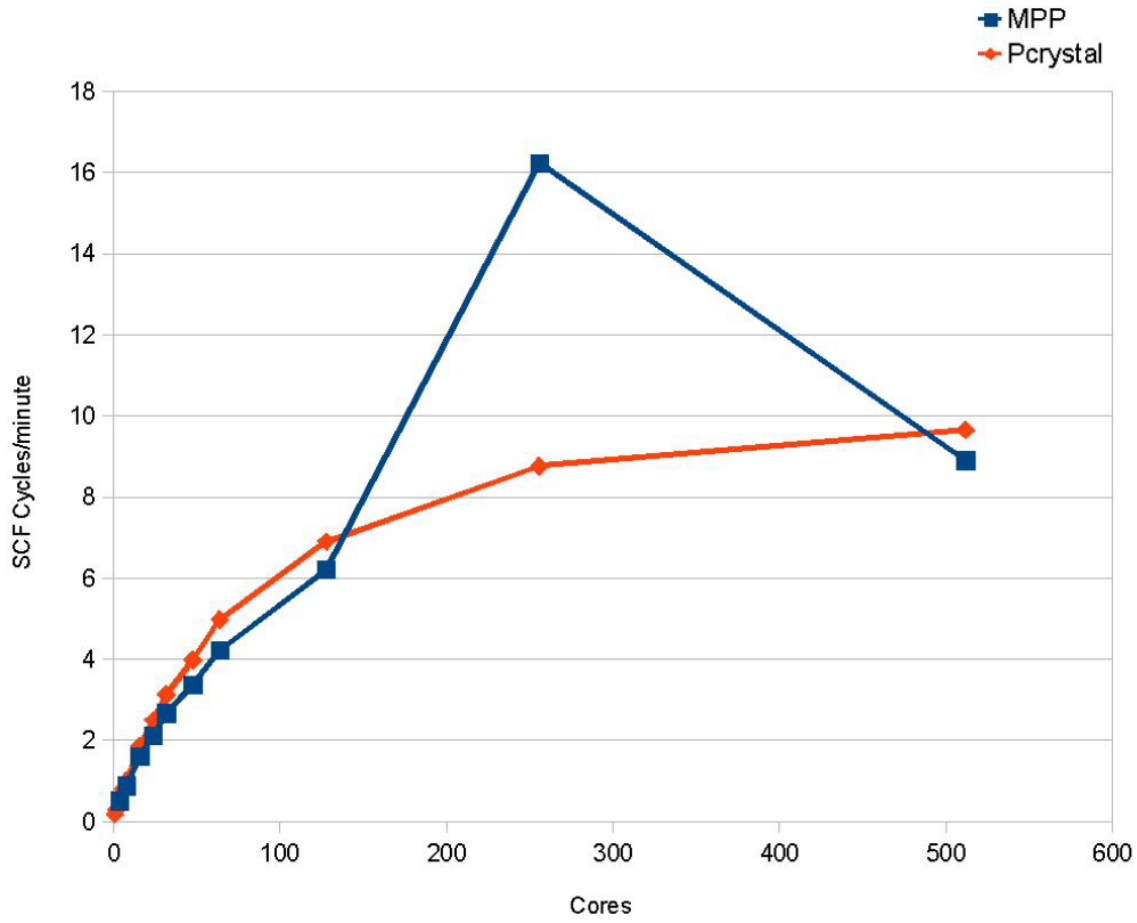
- A series of **short runs** (small number of SCF cycles) on different number of cores to look at the scaling
- Look at the output to see if you can work around any **warnings***
- **New** set of **short runs** with **improved input**

From that data **select PCRYSTAL or MPPCRYSTAL** and the number of cores to be used

***Coulomb and Exchange buffers** ***MPP allk/splitk** (see later)

Very *Old* Example

- Organic crystal used in a drug study
- **624 basis functions**
- **8** irreducible **k points**
- Calculation: **geometry optimisation**
  → For performance tests just run the first SCF
- 16 cycles



Very *Old* Example

So far **PCRYSTAL** looks better

→ System **too small** for diagonalisation in MPPCRYSTAL to work well

EXCEPT at 256 cores. What's going on?

At **128** cores we see in the output:

```
WARNING **** MPP **** ALL K POINTS DONE BY ALL PROCESSORS
```

ALLK

At **256** we see:

```
INFORMATION **** MPP **** MPP running in k point/spin parallel mode
```

SPLITK

→ **Extra performance** due to deciding a *good* number of cores for **k** point parallelism (SPLITK)

The Art of Gentle Persuasion

How to force it to go k-point parallelism?

- In the last section put:

```
CMPLXFAC
2
```

This means you estimate your complex **k** points are **twice as expensive** as purely real ones

- Then look at the nature of your k-points

```
*** K POINTS COORDINATES (OBLIQUE COORDINATES IN UNITS OF IS = 3)
1-R( 0 0 0) 2-C( 1 0 0) 3-C( 0 1 0) 4-C( 1 1 0)
5-C( 0 0 1) 6-C( 1 0 1) 7-C( 0 1 1) 8-C( 1 1 1)
```

we have **1 real k** point and **7 complex**: the **unit cost** is $2 \times 7 + 1 = 15$

- So run on **multiples** of **15 cores** and the code should decide to go **SPLITK**

A Bit of Help

If running in **ALLK**, the code print out a **message** similar to the below to help you *(n.b. Not for the example discussed above)*:

```
**************************************************************************
Please, consider that for this calculation a double level of parallelism
can be exploited by using a number of processors given by:

     N_proc = 44 x a_non_prime_integer (as 4, or 6, or 8, or 9, ...)

For instance, N_proc =   176, or    264, or    352, or     396, or ...
              N_proc =   2816, or   5632, or   11264, or   22528, or ...
              N_proc =   45056, or  90112, or ...
**************************************************************************
```

New Performance

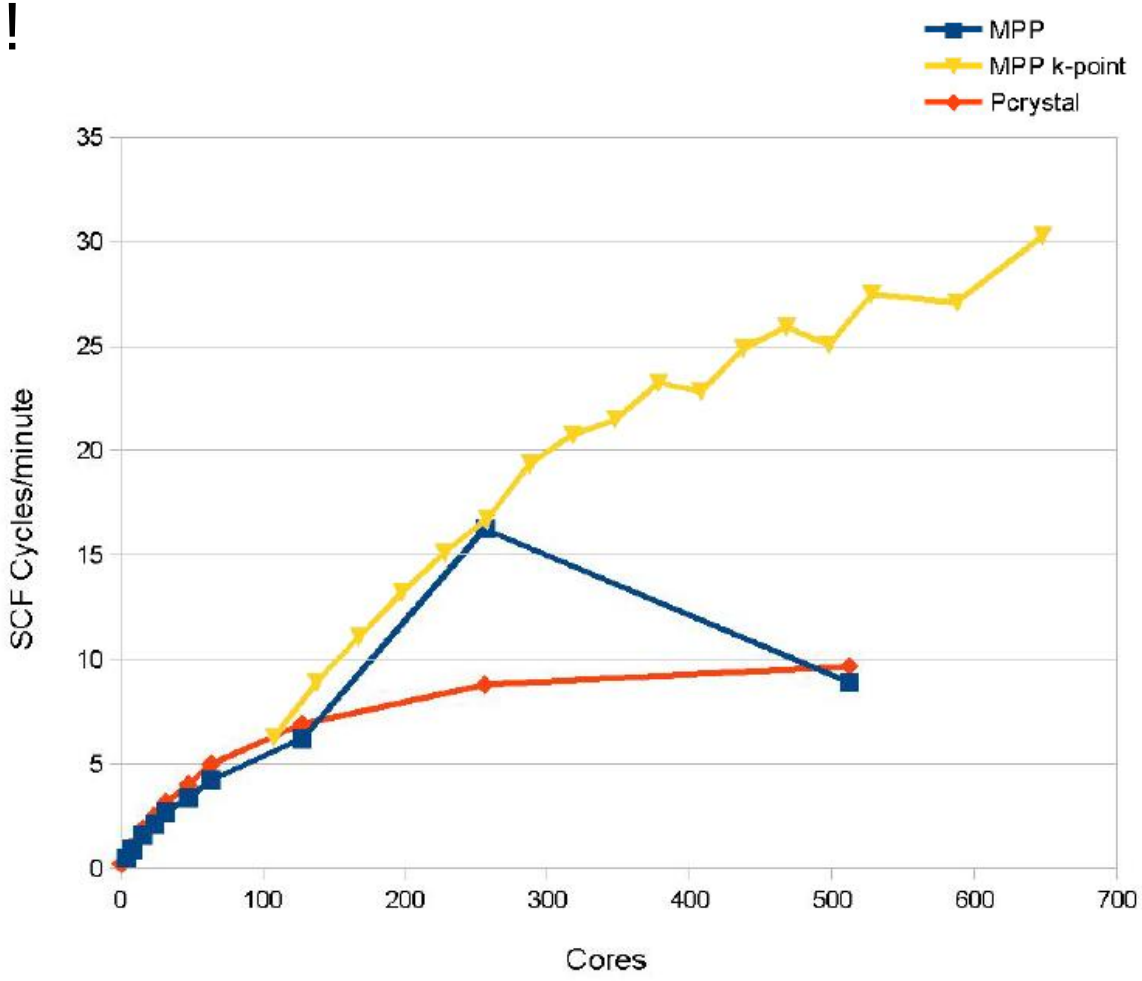
With a bit of work, we have increased our number of SCF cycles from 10 per minute to **around 30**!

- And the best code is now MPPcrystal

Of course **scaling is not perfect**, up to you to make decision where is the best for you, especially if **paying for the time**

Rough estimate above gives 250 cores for MPP

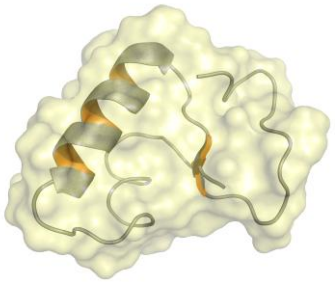
- Not too bad!

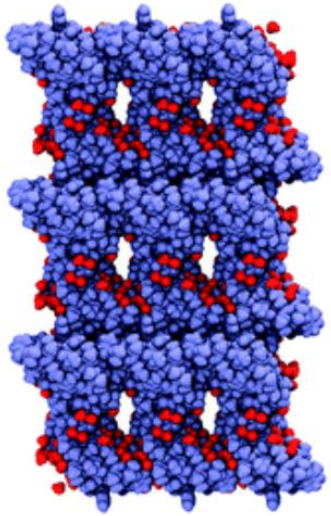


More Examples

Crambin

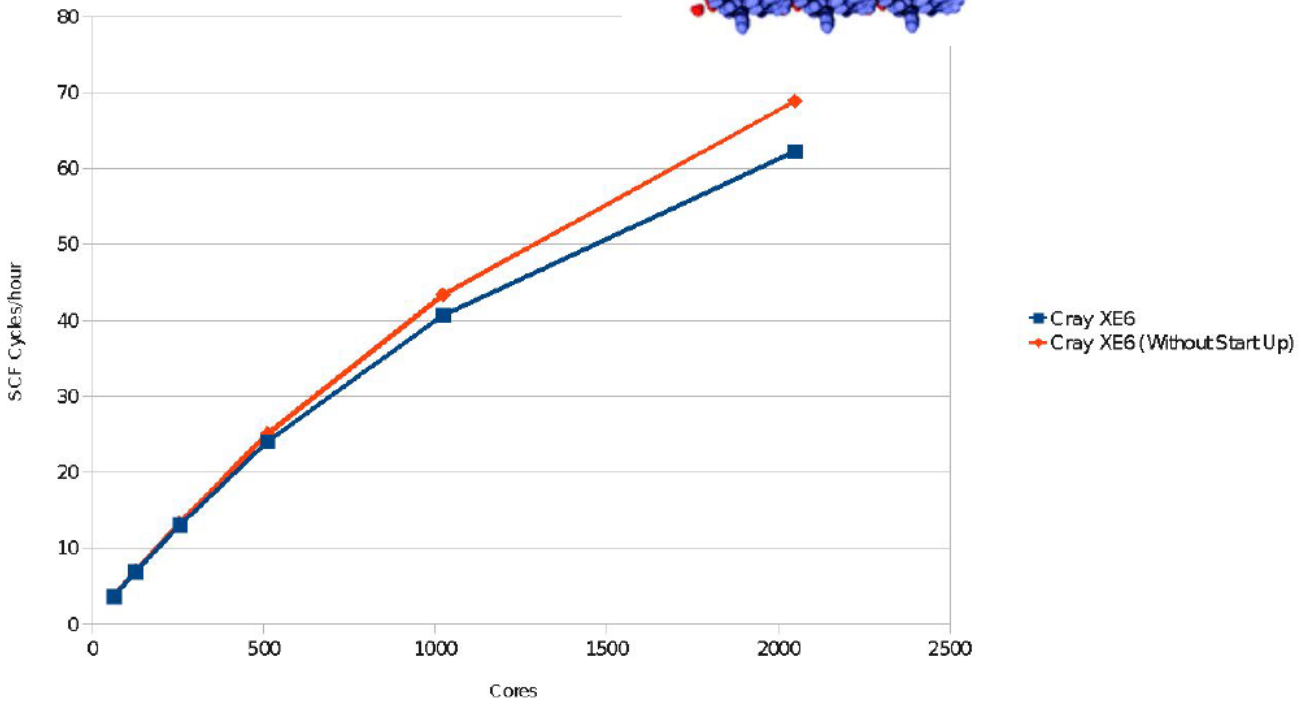
- Small protein (46 residues)
- Crystal structure determined to very high resolution



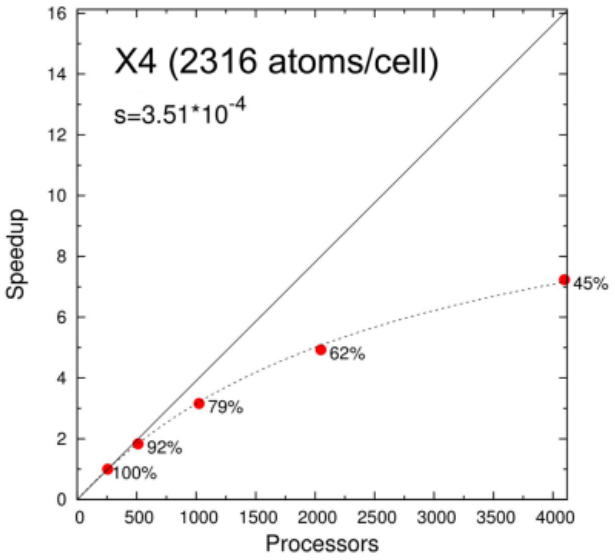


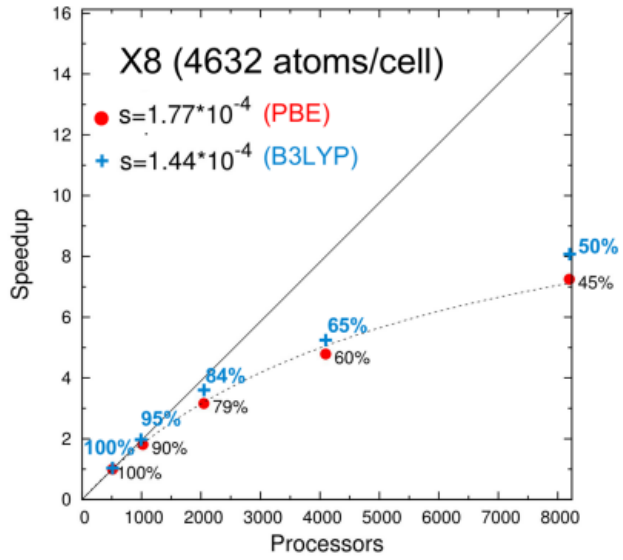
- **1284 atoms**
- **6-31G** basis set**
- **12354 functions**
- All calculations **B3LYP**
- Latest calculations include a full structural optimisation including water of crystallisatio

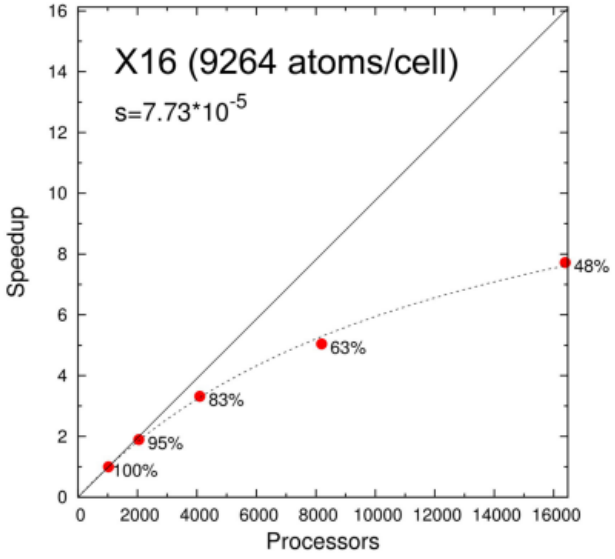
Piero Ugliengo et al. Chem. Sci. 2016 7(2)

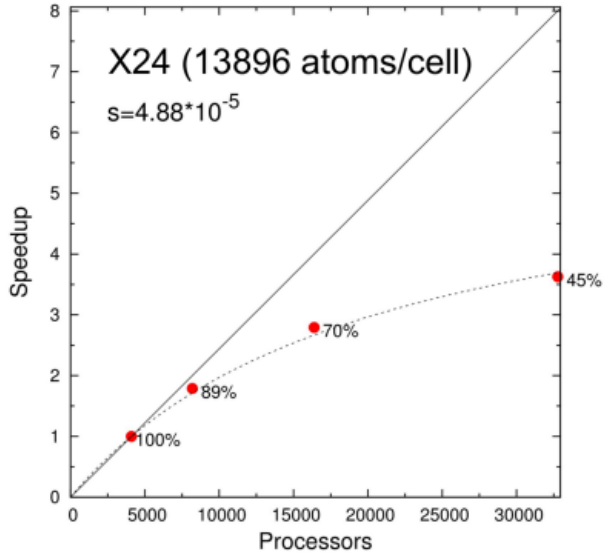


More Examples





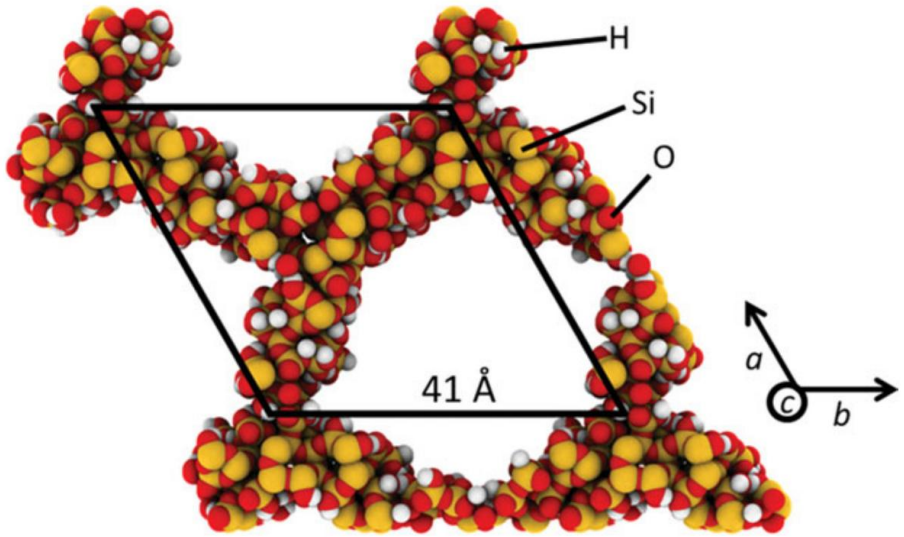




MCM-41

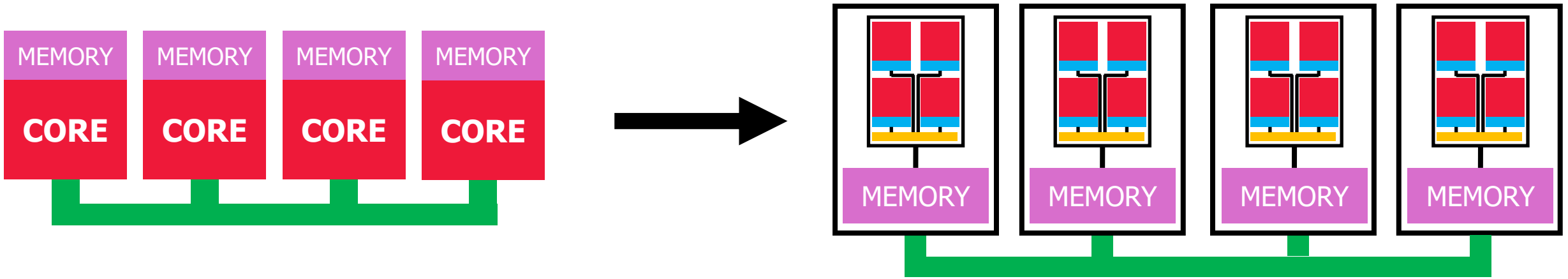
- **Amorphous** (at short range) **silica**
- Long range order of pores
- Model developed by Piero Ugliengo at et al. at the University of Turin

- **579 atoms, 7790 basis functions**
- Also looked at supercells



Introduction to OpenMP

Let's go back to the **real hardware** scheme:



Limits on **system size** are posed by **memory/core** available

- PCRYSTAL hard limit
- MPPCRYSTAL can use more processes to reduce requirement of each process

The ***trick*** can be to use **less cores for** each **node** so that **memory/core ratio** is **increased**

→ But in this way, we are **wasting resources**

Introduction to OpenMP

The solution is to use **threading:**

- In specific **regions** of the code, **threads** are generated so to perform **operations in parallel** on **same *shared* memory**

  → **No replication of data**

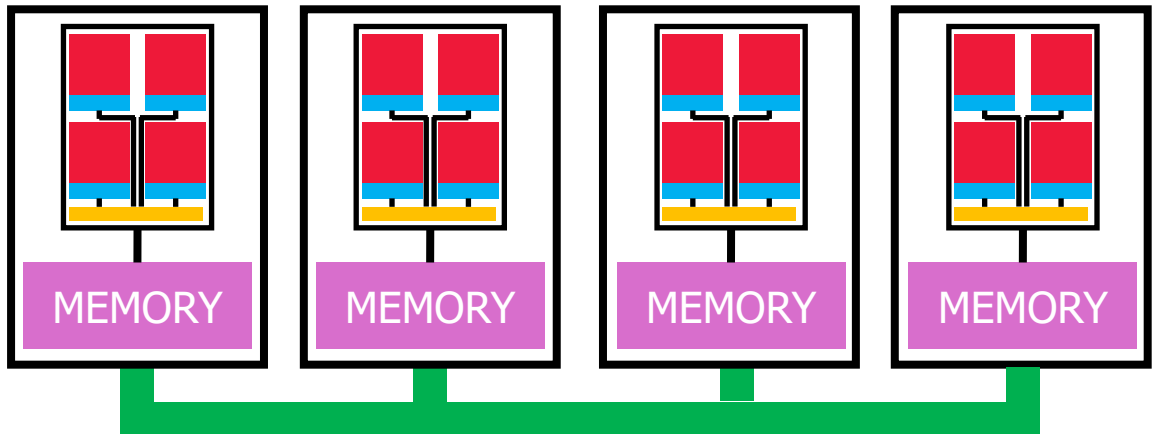
- **This is possible with OpenMP**

  → New feature in Crystal23

- Can be used only in the same Multicore CPU
- When submitting jobs use:

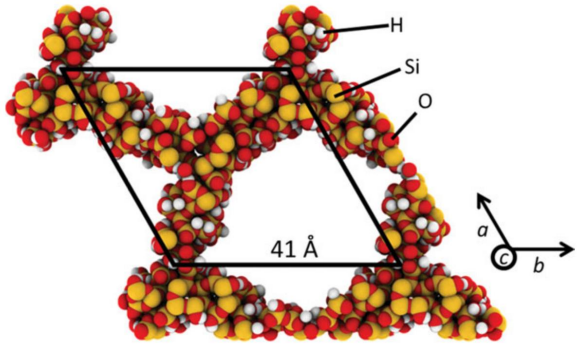
```
>> export OMP_NUM_THREADS=4
```

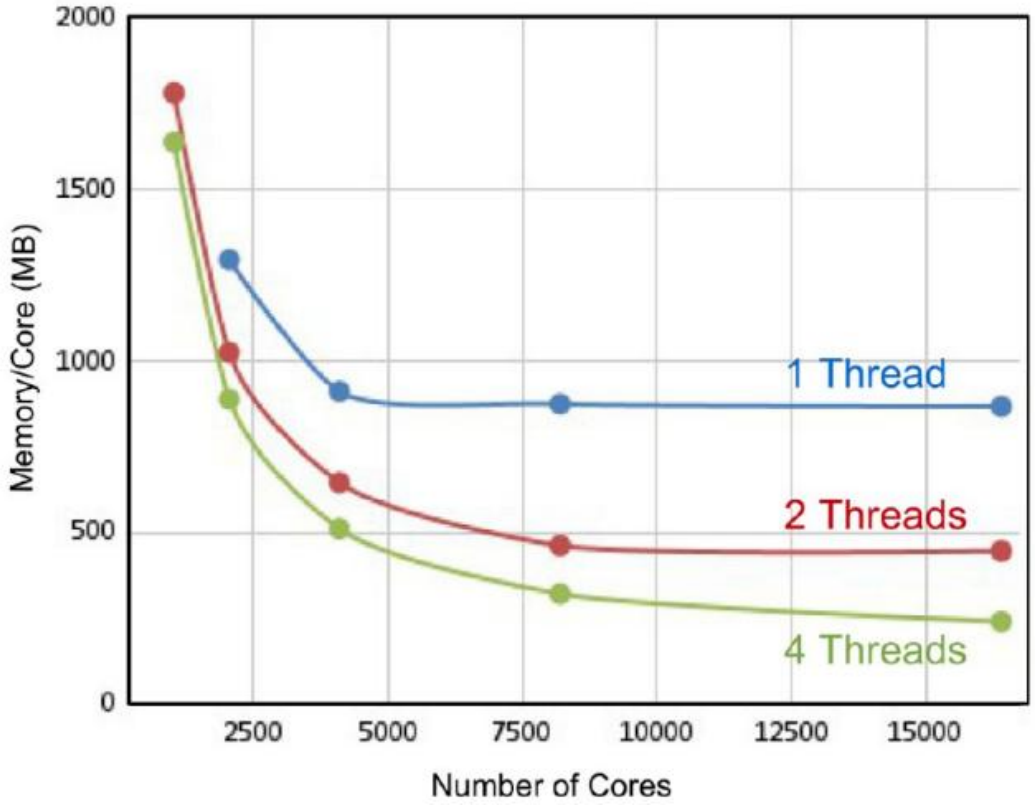
- Available for **CRYSTAL**, **PCRYSTAL** and **MPPCRYSTAL**

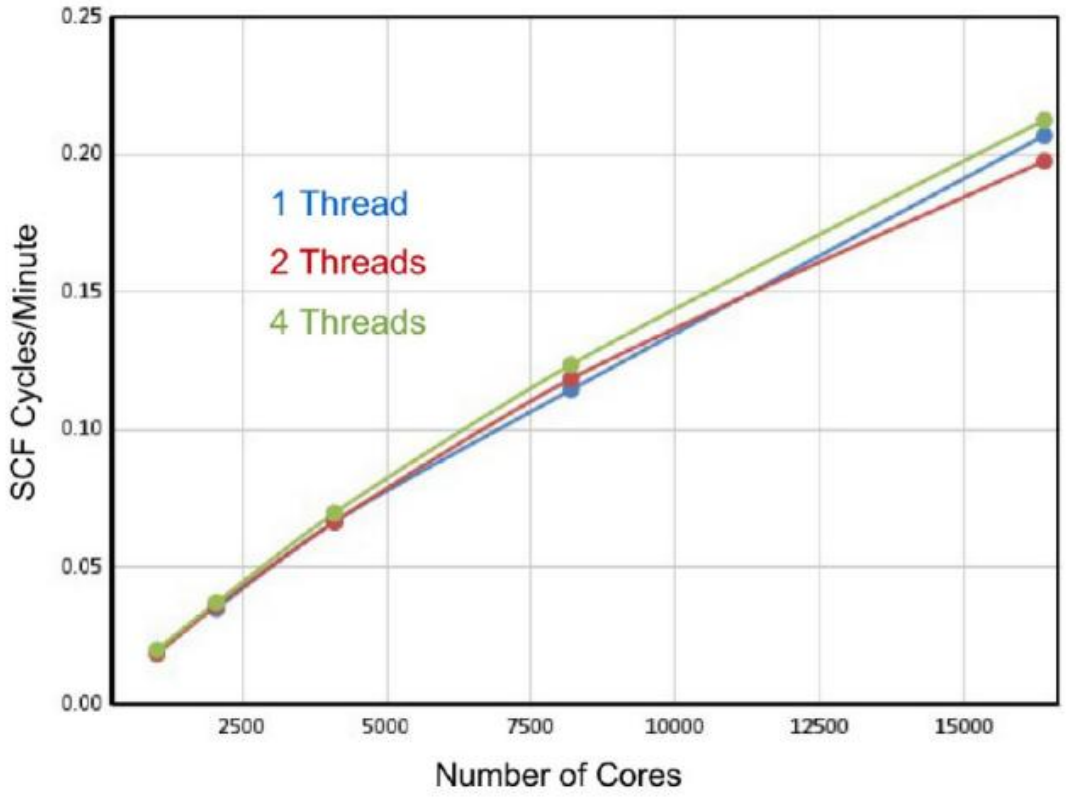


Example: MCM41x16 - PBE

- MPPCRYSTAL + openMP
- 9264 Atoms
- 124640 Basis Functions

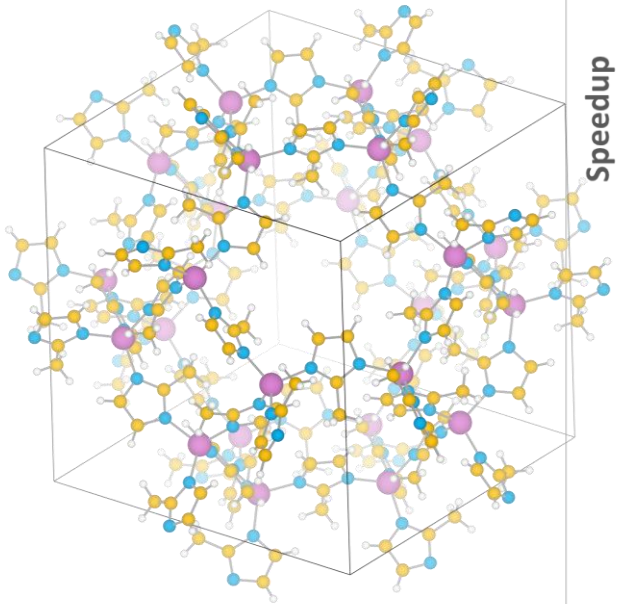


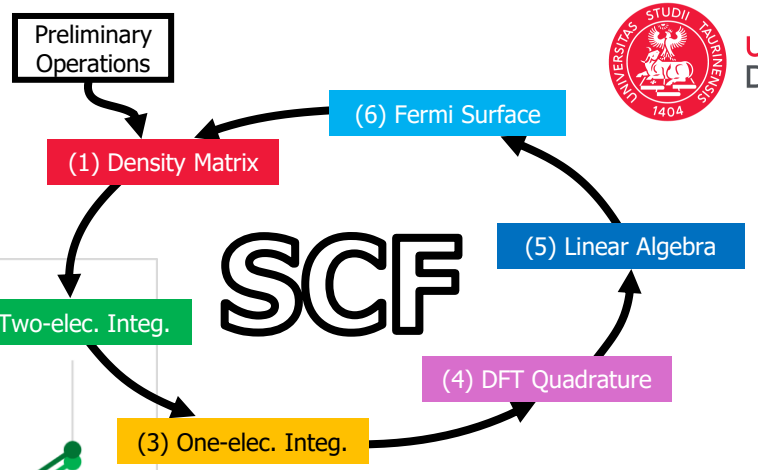


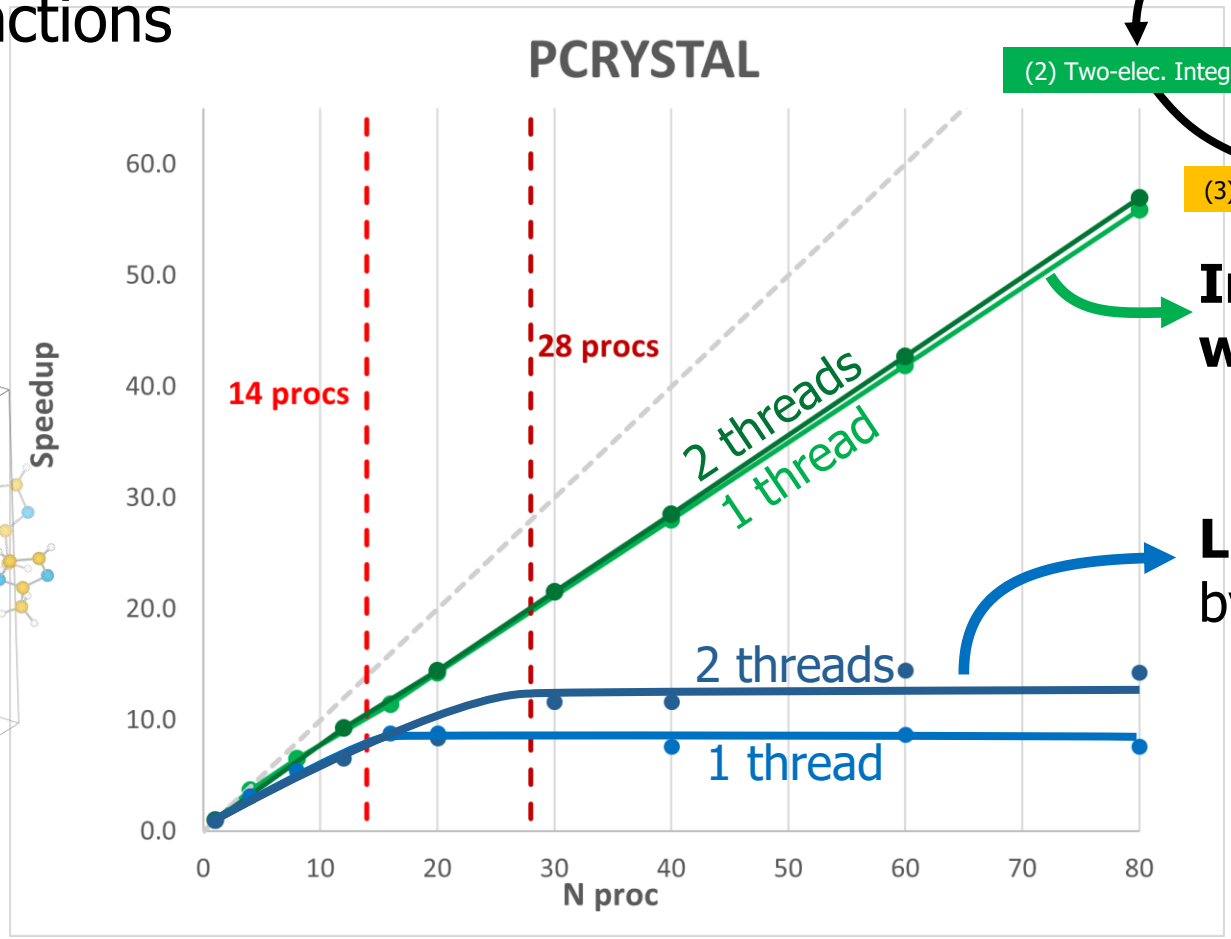


Example: ZIF-8 with PCRYSTAL

- PCRYSTAL + openMP
- 276 Atoms
- 2772 Basis Functions
- 14 **k** points







Integrals scale efficiently **with** and **without** openMP

Linear algebra is **limited** by number of **k** points

Using OpenMP, it is possible to **exploit more cores**

GCRYSTAL

- Another option to solve **linear algebra**
- Use **GPUs** to perform **large matrix** diagonalizations
  → full matrix in **GPU memory**
- GPUs are **co-processors**:
  → Used *on top* of CPUs just for some **heavy tasks**
- Gain **access** to **HPC** based on GPU resources
  → Usually required to demonstrate the capability to run on GPUs
- Alternative to solve **very large systems** with **smaller HW setup**
- Ongoing development, probably available in **next release!**

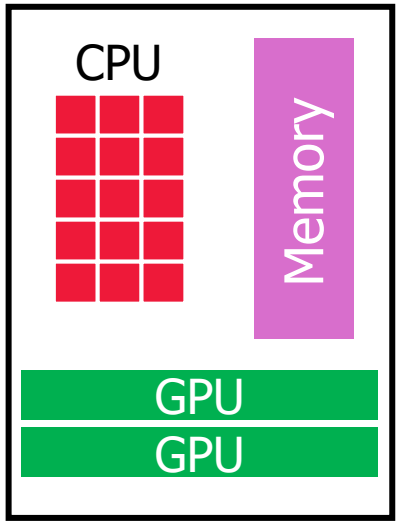
Just for NVIDIA GPUs for now

If want to try alpha version, lets get in touch

GCRYSTAL - Logic and Approach

Common Architectures

- Many CPU cores (**e.g. 128**)
- Few GPUs (**e.g. 4**)



Hybrid CPU/GPU execution

- Use as may MPI processes as needed **to saturate CPU cores**
- Just bind **few processes** to **GPUs**
- Reorganize **k point distribution**:

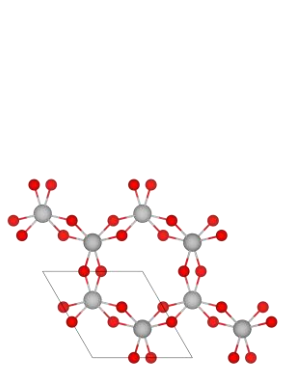
→ only those processes get **k** points

It is **more efficient** to perform operations in **series** on **GPU** rather than to **parallelize** on **CPU cores**

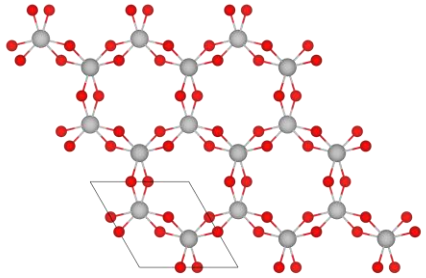
Example α-quartz

Conditions:

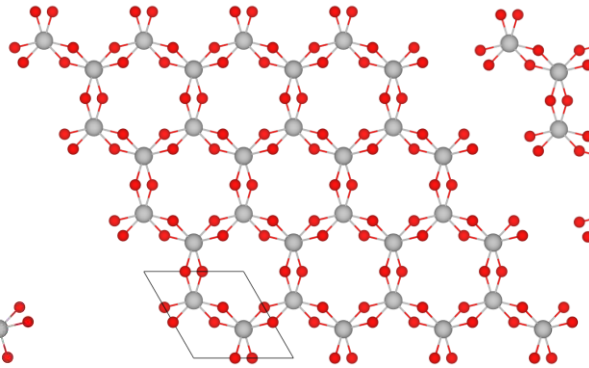
- PBE
- 128 procs
- 0/8 GPUs
- 14 **k** points



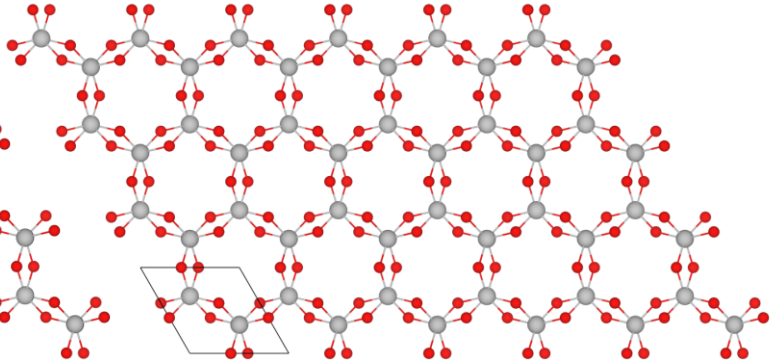
- 12 Atoms
- 248 BFs



- 96 Atoms
- 1984 BFs



- 324 Atoms
- 6696 BFs



- 648 Atoms
- 13392 BFs

→ **LA** time in SCF:

PCRYSTAL	0.07	15.4	606	4454
GCRYSTAL 1 GPU	0.31	2.4 (-84%)	37 (-94%)	228 (-95%)

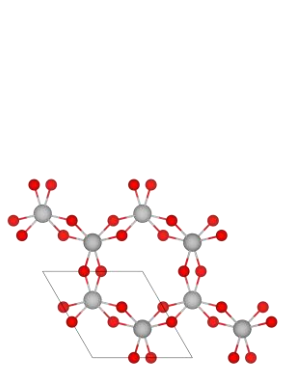
14 diagonalizations **in series**
(but more efficient)

G. Ambrogio, et al. "Accelerated linear algebra for large scale DFT calculations of materials on CPU/GPU architectures with CRYSTAL" *JCP* 162.8 (2025)

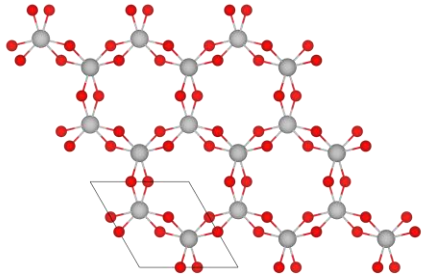
Example α-quartz

Conditions:

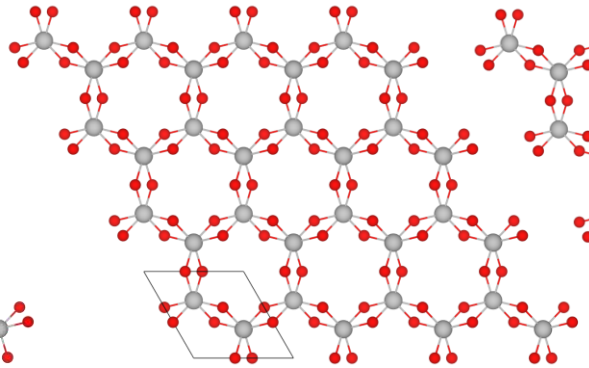
- PBE
- 128 procs
- 0/8 GPUs
- 14 **k** points



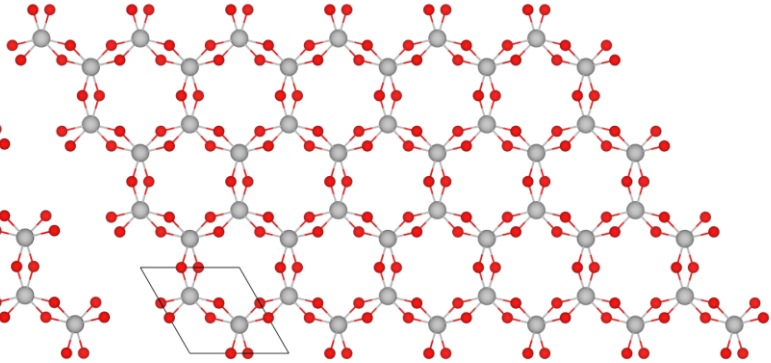
- 12 Atoms
- 248 BFs



- 96 Atoms
- 1984 BFs



- 324 Atoms
- 6696 BFs



- 648 Atoms
- 13392 BFs

→ **LA** time in SCF:

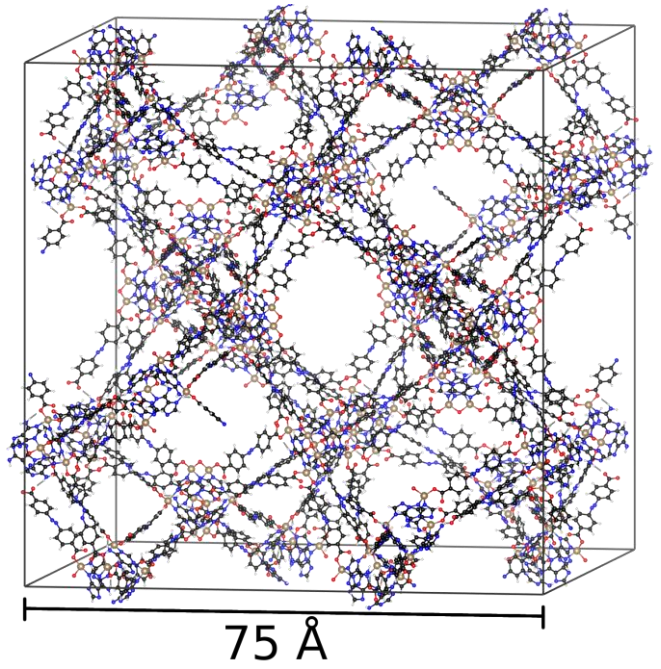
PCRYSTAL	0.07	15.4	606	4454
GCRYSTAL 1 GPU	0.31	2.4 (-84%)	37 (-94%)	228 (-95%)
GCRYSTAL 2 GPUs	0.14	1.4 (-91%)	20 (-97%)	116 (-97%)
GCRYSTAL 4 GPUs	0.12	1.0 (-93%)	14 (-98%)	74 (-98%)
GCRYSTAL 8 GPUs	0.11	0.8 (-94%)	10 (-98%)	51 (-99%)

G. Ambrogio, et al. "Accelerated linear algebra for large scale DFT calculations of materials on CPU/GPU architectures with CRYSTAL" *JCP* 162.8 (2025)

Example bio-MOF-102

Conditions:

- 2808 Atoms/Cell
- 33672 Basis Functions
- PBEsol0
- 1 k Point
- 0-1 GPU



→ **LA** time in SCF:

MPPCRYSTAL	1 Node (128 cores)	538
MPPCRYSTAL	2 Nodes (256 cores)	288
MPPCRYSTAL	4 Nodes (512 cores)	147
MPPCRYSTAL	8 Nodes (1024 cores)	60
GCRYSTAL	1 Node (128 cores + 1 GPU)	87

G. Ambrogio, et al. "Accelerated linear algebra for large scale DFT calculations of materials on CPU/GPU architectures with CRYSTAL" *JCP* 162.8 (2025)

Final Remarks

2 parallelisation strategies currently available:

- **PCRYSTAL**
  - Good for **medium size systems**
  - **Scalability limited** by **k** points
  - Can't study systems bigger than serial CRYSTAL
- **MPPCRYSTAL**
  - Good for **large systems**
  - Can **scale well** for very big problems

OpenMP can be used with both

→ Reduces memory required

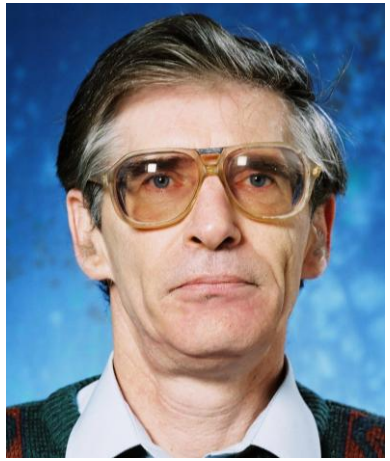
→ Improve efficiency just for PCRYSTAL, if limited by **k** points

In all cases, a little bit of attention to the **input** can **improve the performance**

- **GCRYSTAL** *(coming soon)*
  - Use **GPUs** for Linear Algebra
  - Hopefully will be extended to other sections of the code

Acknowledgment

Advanced School on
QUANTUM MODELLING
of Materials with CRYSTAL



***Vic* Saunders**

Dr. Ian J. Bush

Dr. Barry Searle

UNIVERSITÀ DI TORINO

chimica

2023 - 2027
DIPARTIMENTO DI ECCELLENZA
Ministero dell'Università e della Ricerca

Part of the material presented has been adapted from slides kindly provided by Dr. **Ian Bush**

UK Research and Innovation

COMPUTATIONAL TOOL FOR SOLID STATE CHEMISTRY AND PHYSICS
CRYSTAL

SCF in CRYSTAL

P

MPP

(2) Two-elec. Integ.

$$E_{\text{coul}} = \frac{1}{2} \sum_{\mu\nu} \sum_{\mathbf{g}} P^{\mathbf{g}}_{\mu\nu} \sum_{\sigma\rho} \sum_{\mathbf{q}} P^{\mathbf{q}}_{\sigma\rho} \sum_{\mathbf{s}} (\chi^{\mathbf{0}}_{\mu} \chi^{\mathbf{g}}_{\nu} | \chi^{\mathbf{s}}_{\sigma} \chi^{\mathbf{s+q}}_{\rho})$$

$$E_{\text{exx}} = -\frac{1}{2} \sum_{\mu\nu} \sum_{\mathbf{g}} P^{\mathbf{g}}_{\mu\nu} \sum_{\sigma\rho} \sum_{\mathbf{q}} P^{\mathbf{q}}_{\sigma\rho} \sum_{\mathbf{s}} (\chi^{\mathbf{0}}_{\mu} \chi^{\mathbf{s}}_{\sigma} | \chi^{\mathbf{g}}_{\nu} \chi^{\mathbf{s+q}}_{\rho})$$

(3) One-elec. Integ.

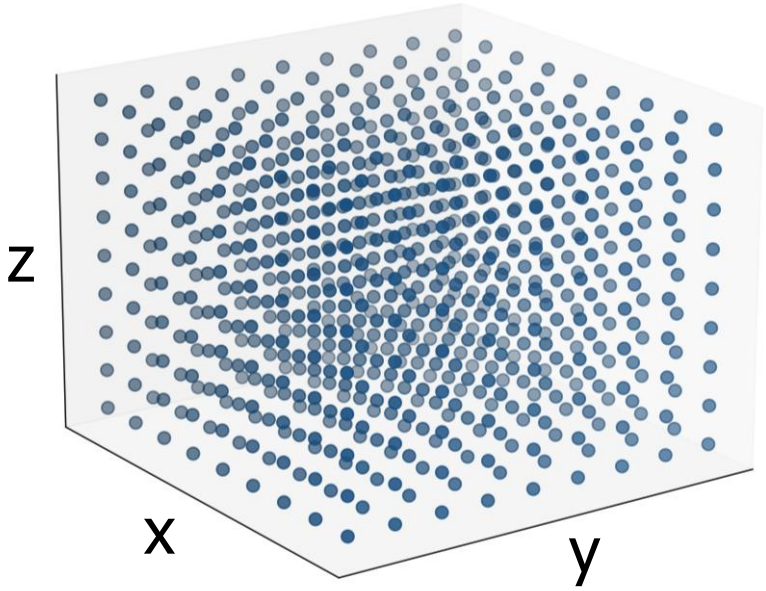
$$E_{\text{kin}} = \sum_{\mu\nu} \sum_{\mathbf{g}} P^{\mathbf{g}}_{\mu\nu} \langle \chi^{\mathbf{0}}_{\mu} | -\frac{\nabla^2}{2} | \chi^{\mathbf{g}}_{\nu} \rangle$$

$$E_{\text{e-nucl}} = -\sum_{\mu\nu} \sum_{\mathbf{g}} P^{\mathbf{g}}_{\mu\nu} \sum_{\lambda} Z_{\lambda} \sum_{\mathbf{h}} \left\langle \chi^{\mathbf{0}}_{\mu} \left| \frac{1}{|\mathbf{r} - \mathbf{h} - \mathbf{s}_Z|} \right| \chi^{\mathbf{g}}_{\nu} \right\rangle$$

SCF in CRYSTAL

(4) DFT Quadrature

$$E_{\mathrm{xc}}[\rho(\mathbf{r})] = \int d\mathbf{r}\varepsilon_{\mathrm{c}}[\rho(\mathbf{r})] + \int d\mathbf{r}\varepsilon_{\mathrm{x}}[\rho(\mathbf{r})]$$



Batches of Points Distributed to Processes

Computed on a 3D Grid of Points

Each Point is Independent from Others

SCF in CRYSTAL

(5) Linear Algebra

$$H^{\mathbf{k}}_{\mu\nu} = \sum_{\mathbf{g}} H^{\mathbf{g}}_{\mu\nu} e^{i\mathbf{k}\cdot\mathbf{g}}$$

For each **k**:

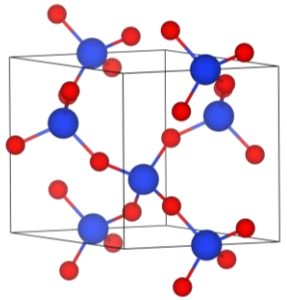
$$\mathbf{H}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}} = \mathbf{S}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}}\mathbf{E}^{\mathbf{k}}$$

n AOs

n AOs

$\mathbf{H}^{\mathbf{k}}$

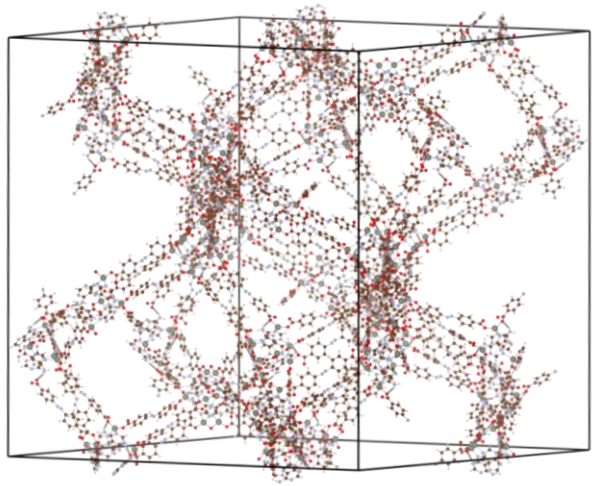
Small Unit Cell



- Few Basis Functions
- Many **k** Points

Efficient Parallelization

Large Unit Cell



- Many Basis Functions
- Few **k** Points

Pour Parallelization

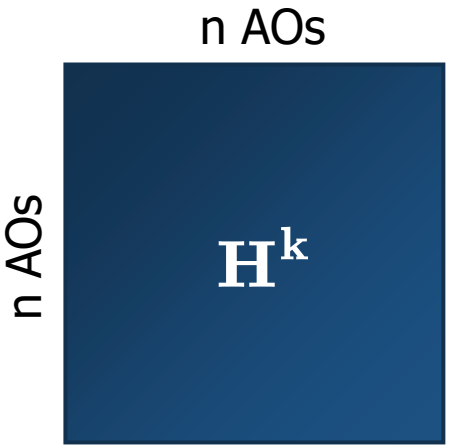
SCF in CRYSTAL

(5) Linear Algebra

$$H^{\mathbf{k}}_{\mu\nu} = \sum_{\mathbf{g}} H^{\mathbf{g}}_{\mu\nu} e^{i\mathbf{k}\cdot\mathbf{g}}$$

For each **k**:

$$\mathbf{H}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}} = \mathbf{S}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}}\mathbf{E}^{\mathbf{k}}$$



General Rule

$$n\ \mathbf{k}\ \text{points} \geq n\ \text{processes}$$

Large Systems

Large-Scale Architectures

SCF in CRYSTAL

P
MPP

Advanced School on
QUANTUM MODELLING
of Materials with CRYSTAL

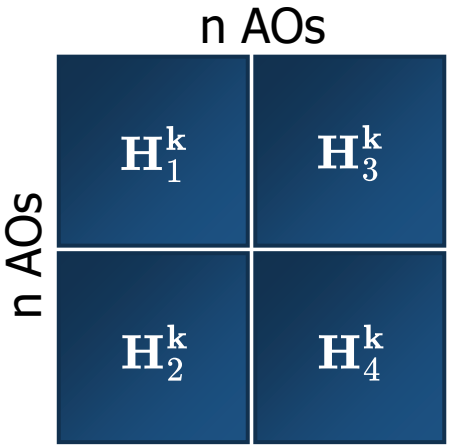
UNIVERSITÀ
DI TORINO

(5) Linear Algebra

$$H^{\mathbf{k}}_{\mu\nu} = \sum_{\mathbf{g}} H^{\mathbf{g}}_{\mu\nu} e^{i\mathbf{k}\cdot\mathbf{g}}$$

For each **k**:

$$\mathbf{H}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}} = \mathbf{S}^{\mathbf{k}}\mathbf{A}^{\mathbf{k}}\mathbf{E}^{\mathbf{k}}$$



General Rule

$$n\ \mathbf{k}\ \text{points} \geq n\ \text{processes}$$

Large Systems

Large-Scale Architectures

MPPCRYSTAL → ScaLAPACK

- Allocation of Large Matrices Across Multiple Processes
- Linear Algebra Operations Executed by Multiple Processes (using MPI)

R. Orlando, M. Delle Piane,I. J. Bush, P. Ugliengo, M. Ferrabone, and R. Dovesi, A new massively parallel version of CRYSTAL for large systems on high performance computing architectures, *J. Comput. Chem.*, **2012**, *33.28*, 2276-2284

SCF in CRYSTAL

P
MPP

Advanced School on
QUANTUM MODELLING
of Materials with CRYSTAL

UNIVERSITÀ
DI TORINO

(6) Fermi Surface

$$\mathrm{n\ elect.} = \frac{1}{\Omega}\int_{\mathrm{FBZ}} d\mathbf{k} \sum_i \theta[\epsilon_F - \varepsilon_i^{\mathbf{k}}]$$

Not parallelized

(1) Density Matrix

$$P_{\sigma\rho}^{\mathbf{g}} = \frac{1}{\Omega}\sum_i f_i \sum_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{g}} a_{\sigma,i}^{\mathbf{k}} [a_{\rho,i}^{\mathbf{k}}]^* \theta[\epsilon_F - \varepsilon_i^{\mathbf{k}}]$$

Parallelized over **k** points

Not expensive