Introduction to Numerical Simulations and High Performance Computing: From Materials Science to Biochemistry

Mauro Boero

Institut de Physique et Chimie des Matériaux de Strasbourg

University of Strasbourg - CNRS, F-67034 Strasbourg, France

Computational Materials Science Initiative

計算物質科学イニシアティブ

@ Dept. of Applied Physics, The University of Tokyo,

7-3-1 Hongo, Tokyo 113-8656, Japan

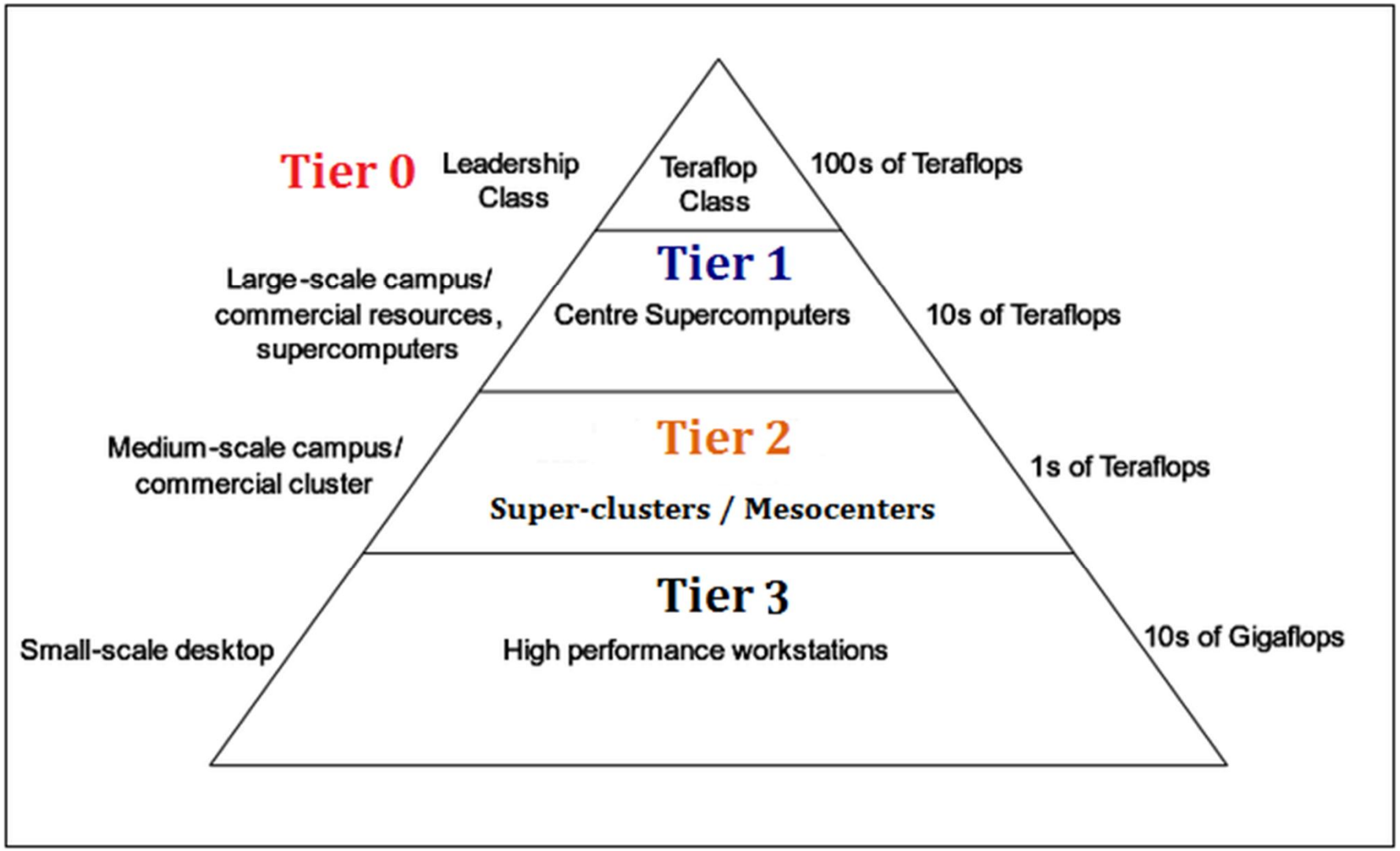
Part 7:

Brief overview of HPC architectures and parallel programming

Outline

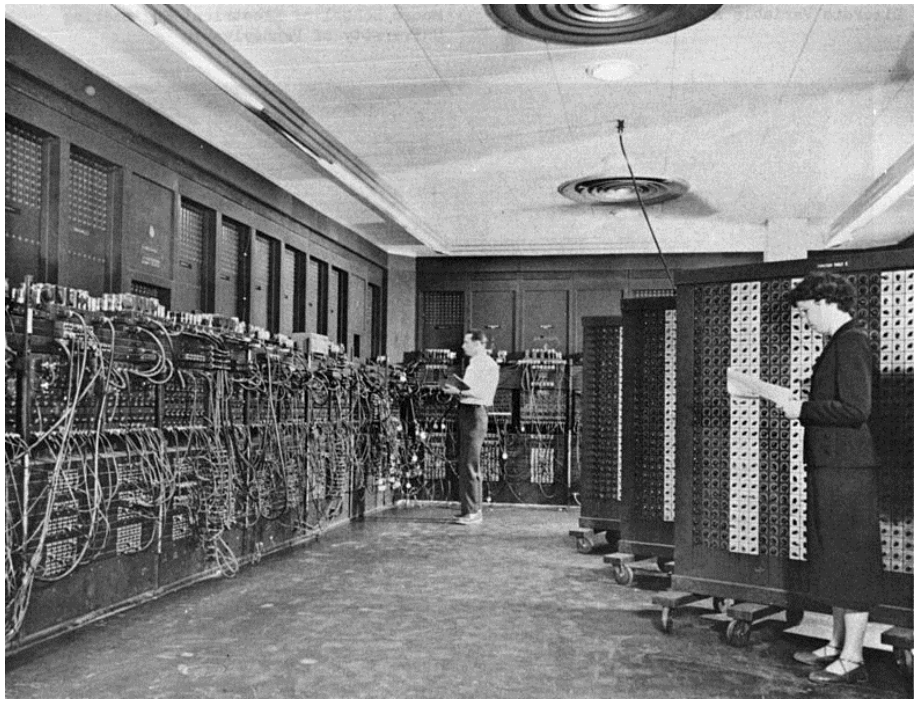
- High Performance Computing (HPC): Massively parallel machines and CPU/GPU achitectures
- Parallelization strategies: MPI/OMP & Co.
- Practical example of implementation: numerical scheme, basis set, direct space and Fourier transform & Co.
- Example performance and speedup (QM & QM/MM)

High Performance Computing (HPC) nowadays



High Performance Computing (HPC): Brief History

• John von Neumann (*Manhattan Project*) programmed the first algorithm on **ENIAC** (**Electronic Numerical Integrator And Computer)**
• ENIAC was designed to calculate artillery firing tables for US Army's ballistic research (1946)



Nowadays
ENIAC@IPCMS

High Performance Computing(HPC): Brief History

Ken Wilson, a Nobel Laureate physicist at Cornell, wrote a white paper in the early 1980's on the need of the scientific community for supercomputing

a) He described machines that would be able to "think"

(1) The "Gibbs keyboard" - 1 key, "read my mind"

b) The physics community got behind this proposal and "pushed"

Source: http://www.mrynet.com/cray/docs.html

1985 Cray-2

Source: http://www.bobndenise.com/computers/computer.htm

IBM 3090

Example: of modern Machines:

HPC @ IPCMS (ENIAC)

- 896 Cores network architecture with Infiniband interconnection
- Front Quad Xeon 8 GB RMA + 64 dual processor compute nodes Quad Xeon - 32 GB RAM
- Scalable network infrastructure to Gigabit Ethernet, storage > 20 TB

Equip@Meso

- Hybdrid CPU/GPU NEC HPC1812Rd-2/GPS12G4Rd-2
- 145 nodes = 2320 Computing cores (Mellanox Infiniband)

Other available HPC resources in France:

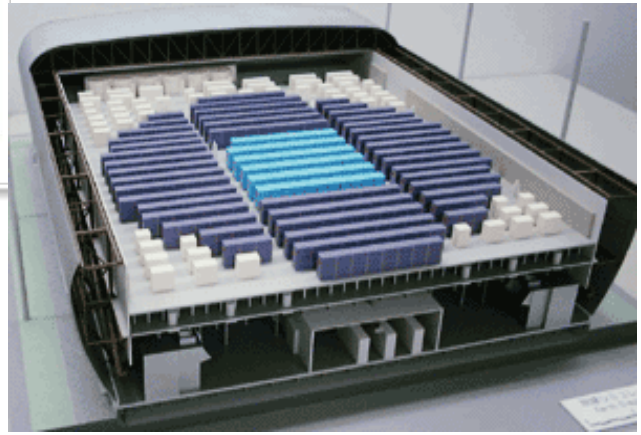
- IBM Blue Gene / Q & IBM x3750 @ IDRIS

- BULL Cluster @ INES

- CURIE cluster @ TGCC

Earth Simulator & K-Computer (Japan)

EARTH SIMULATOR



About *parallel* programming:

MPI (Message Passing Interface)
https://www.open-mpi.org/
https://www.mpich.org/
https://software.intel.com/en-us/intel-mpi-library

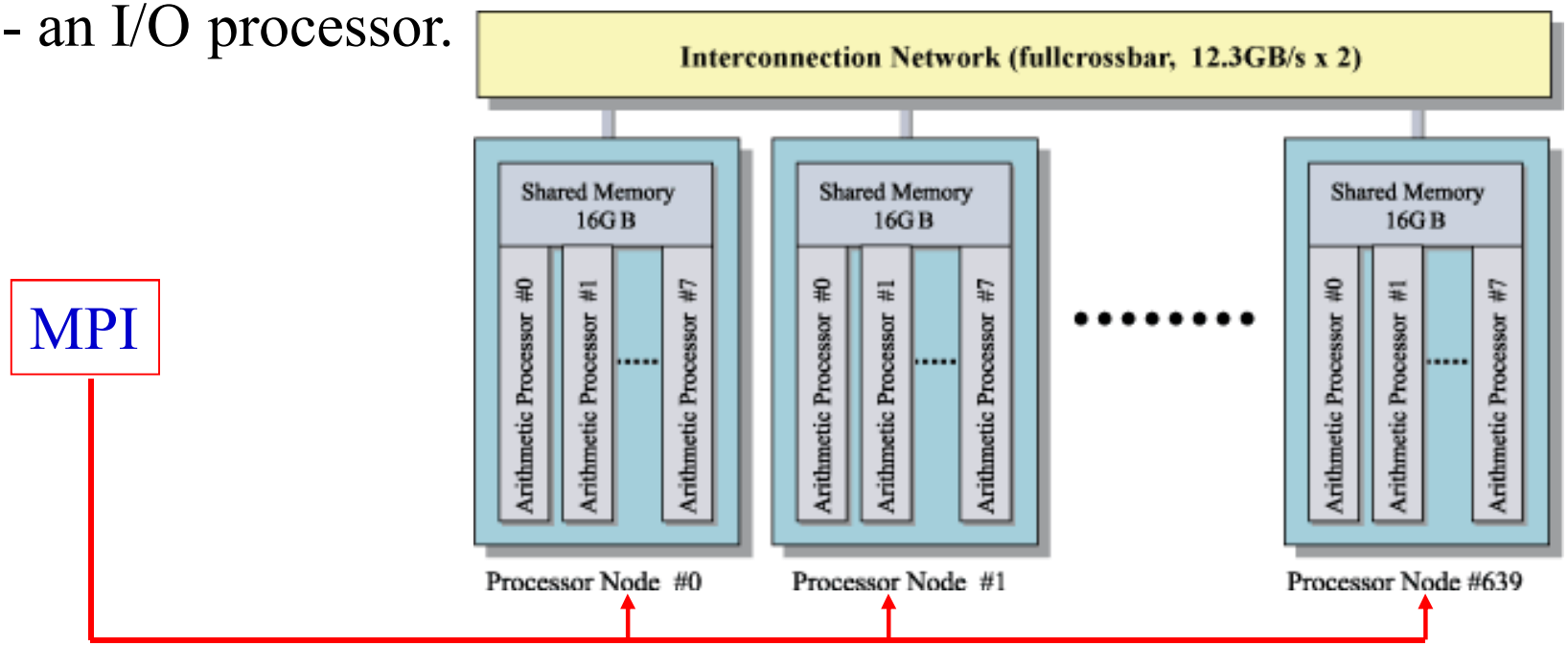
and

OpenMP (http://openmp.org/)

are the major tools to parallelize a computer code

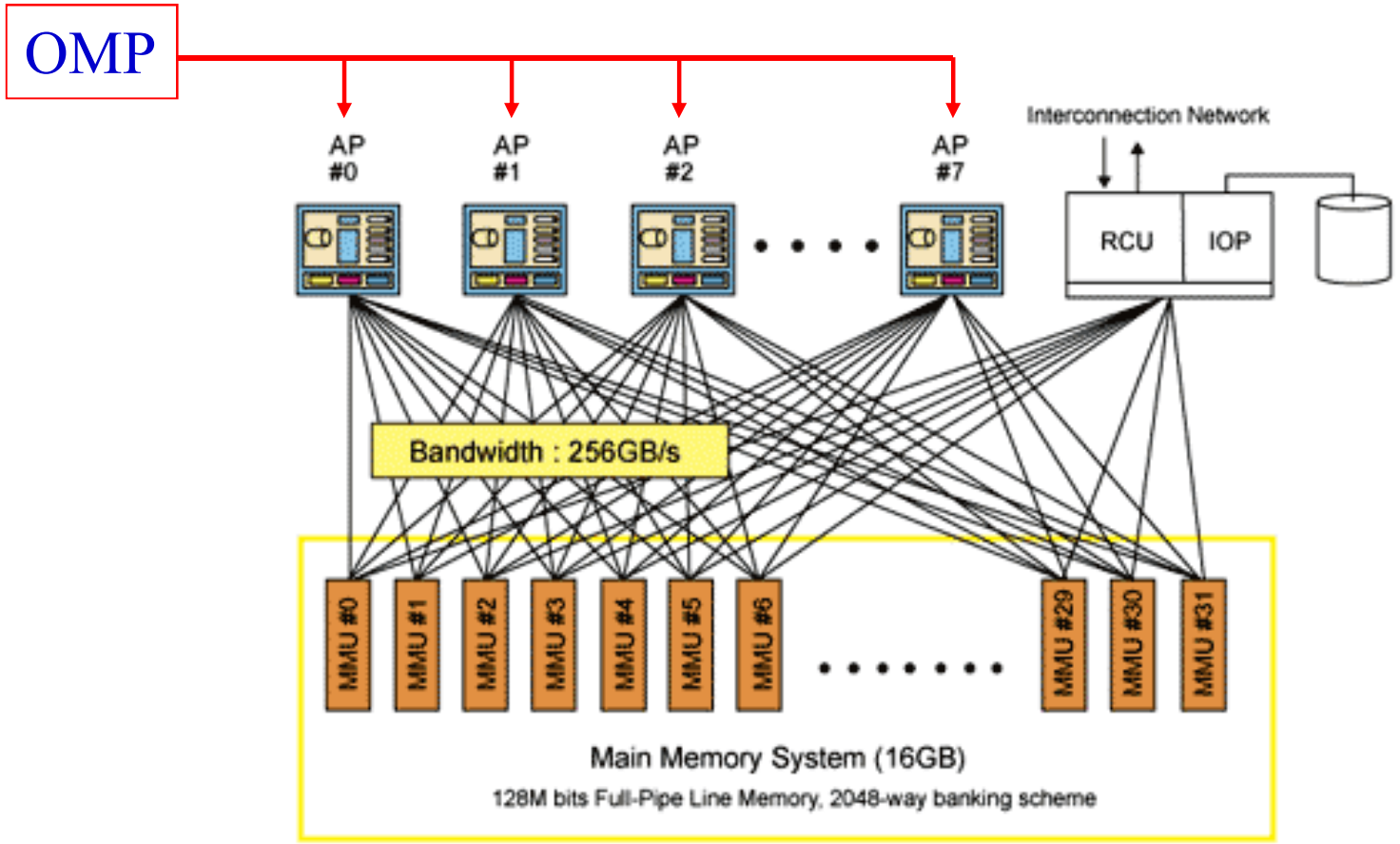
ES system configuration

Parallel vector supercomputer system with 640 processor nodes (PNs) connected by 640x640 single-stage crossbar switches.
Each PN is a system with a shared memory, consisting of
- 8 vector-type arithmetic processors (APs): total=5120 AP
- a 16-GB main memory system (MS)
- a remote access control unit (RCU)
- an I/O processor.



ES system : single processor node (PN)

- The overall MS is divided into 2048 banks
- The sequence of bank numbers corresponds to increasing addresses of locations in memory.



About parallel programming:

What is MPI ?

- Is a message-passing interface (library) specification (*basically calls in a computer code*)
- Is NOT a programming language or computer specification
- Is NOT a specific implementation or (commercial) product
- Is a package/wrapper to be used within a specific language
- Is intended for parallel computers, clusters and heterogeneous networks
- Is designed to provide access to advanced parallel hardware for
  - programmers/developers of codes AND languages
  - end users
  - library writers

About parallel programming:

Why MPI ?

- MPI provides a “not-so-hard-to-handle” efficient and portable way to parallelize programs
- It has been explicitly designed to enable libraries…
- …which may eliminate the actual need for users to learn MPI
- Yet, ***it is better to learn MPI if you are a developer*** (or you are doomed/fired)

About parallel programming: the Environment

Two major questions arise immediately in a parallel code:

- How many processes (cores/CPUs) participate to this computation?
- …and on which one am I (now)?

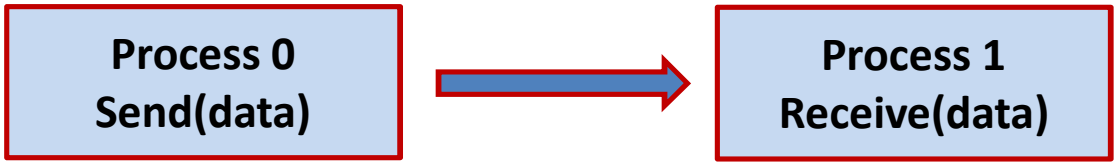
MPI has two functions designed to answer these questions:

mpi_comm_size reports the number of processes

mpi_comm_rank reports the rank, a number between 0 and Nproc-1 identifying the calling process

About parallel programming: MPI send/receive

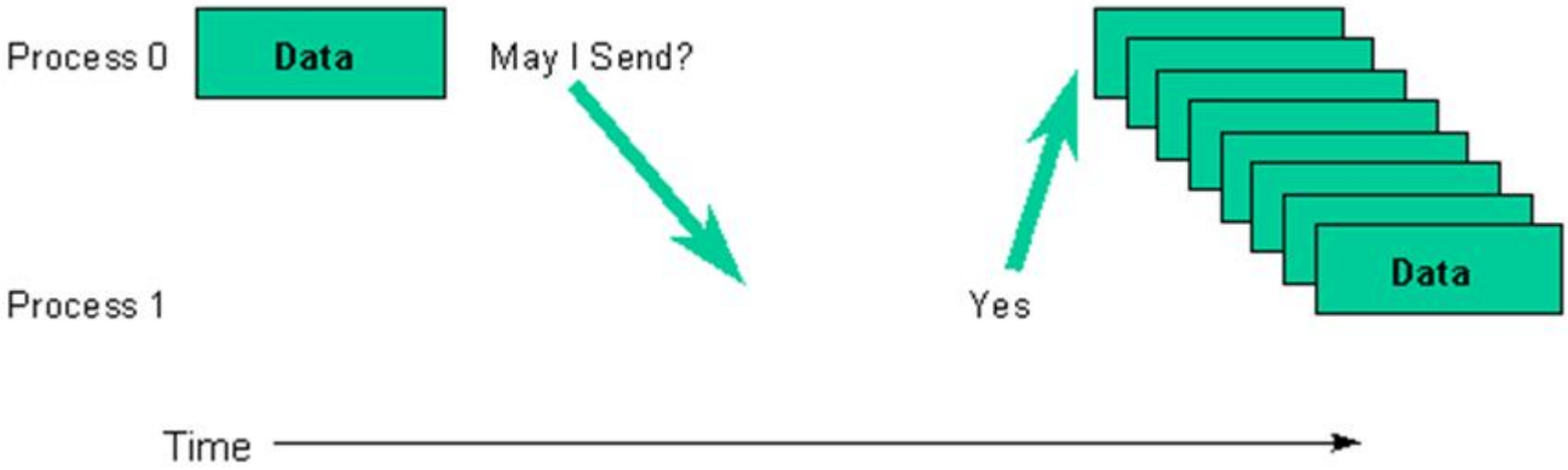
We need to supply the details in



And we need to specify:

- How **data** will be described
- How processes will be identified
- How the receiver will recognize/screen messages
- What it will mean for these operations to complete
- …and what to do after receving&computing is over

About parallel programming: How does it work?



- **Data** transfer requires synchronization
- Requires cooperation between sender and receiver
- …and this is *not always apparent in the source code*

About parallel programming: Main tasks are simple (well… somehow)

- A parallel code uses mainly six basic MPI functions used as call MPI_...(…)

MPI_INIT	…start mpi procedure
MPI_FINALIZE	…end mpi procedure
MPI_COMM_SIZE	…determines the size of the group associated with a communicator
MPI_COMM_RANK	…determines the rank of the calling process in the communicator
MPI_SEND	…send
MPI_RECV	…receive

- Point-to-point (send/recv) is the only way…

About parallel programming: Introduciton to collective operations in MPI

- Collective operations are called by all processes in a communicator
- MPI_BCAST distributes data from one process (the parent or root) to all others in a communicator
- MPI_REDUCE combines data from all processes in a communicator and returns it to one process (the parent or root)
- SEND/RECEIVE can be replaced by BCAST/REDUCE improving simplicity and efficiency
- **All-to-All** (any process communicating with any other one) is the heaviest task/workload (of course)

About GPUs and code(s) performance

Looking back at early ‘90s:

Fastest machines were CRAY's...

– Vector machine

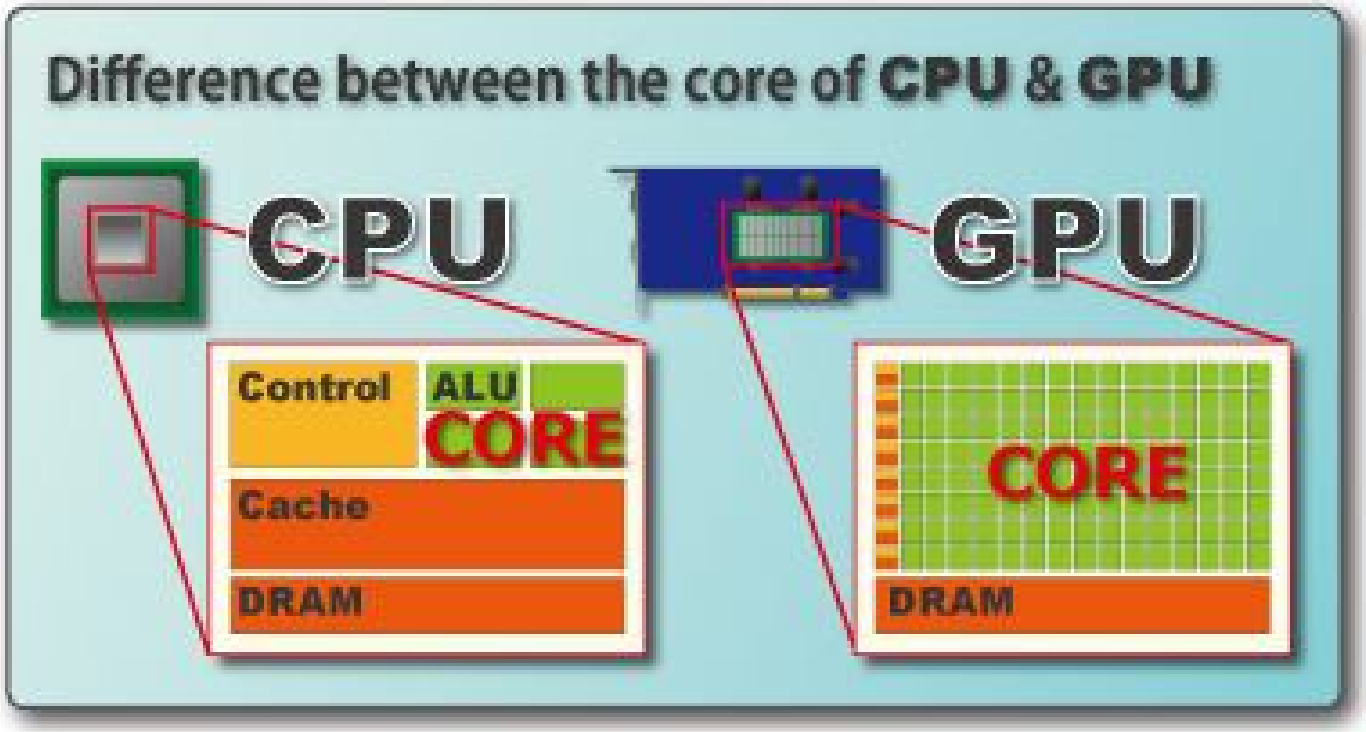
– Fast memory streams vectors through a very fast Processor

...and Connection Machines

– Massively parallel architecture

– Very many slower processors each compute on one element of the result vector.

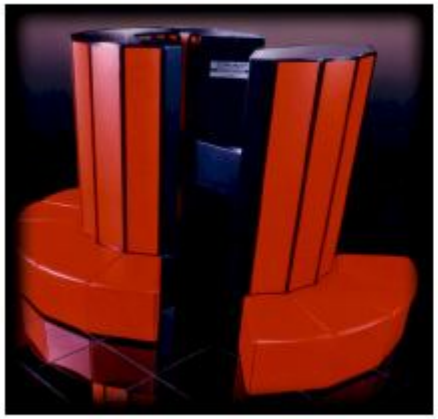
CPU ??? GPU ???



iP-rocess **iP-ut in order**

About GPUs and code(s) performance

The vector machine strikes back



vector computer CM GPU Nvidia®

A modern GPU is both a vector machine and a massively parallel architecture

About GPUs and code(s) performance

Pros	Cons
Fast	Specialized
Cheap	Hard to program (efficiently)
Low-power	Bandwidth problems
Compact	Rapidly changing (need for continuous recoding)

Future is streaming anyway?.... Or not?

Example of GPU programming on NVIDIA

- GeForce Quadro, Tesla, Mobile : Tegra, Cloud : GeForce GRID VGX
- 2011–Tegra2, 2012-Tegra3, 2013-Tegra4, 2014-TegraK1 Logan, 2015-Parker
- GPU generation: 2008-Tesla, 2010-Fermi, 2012-Kepler, 2014-Maxwell (next: Volta)
  Doubling the computing power at each new generation.
  Purpose: Power efficiency, Ease of programming, large application coverage.
  More calculations per consumed Watt

Programming: NVIDIA @ Portland (PGI-Fortan & C++)

Good points: vector-matrix & matrix-matrix operations, pointers handling

Weak points: FFT and communication speed for massively parallel applications

OpenACC: Generalization of OpenMP. (***Not sure if better than OMP 4.0***)

OpenMP example

```
!$OMP parallel do private (i,j)
do i=1,…
 do j=1,…
   Anew(i,j)=…A(i,j)
 enddo
enddo
!$OMP parallel end
```

OpenACC example

```
!$acc data copy(A), create(Anew)
!$acc kernels
 do i=1,…
   do j=…
      Anew(i,j)=…A(i,j)
   enddo
enddo
!$acc end kernels
!$acc end data
```

Which calculations?

- Standard example: Schrödinger/Kohn-Sham/Dirac/whatever quantum mechanics formulation (non *t*-dependent)

$$\hat{\mathrm{H}}\,\psi_i(\mathbf{x}) = E_i\psi_i(\mathbf{x})$$

- Simple basis set case: orbitals in plane waves

$$\psi_i(\mathbf{x}) = \sum_{\mathbf{G}} c_i(\mathbf{G})\, e^{i\mathbf{G}\mathbf{x}}$$

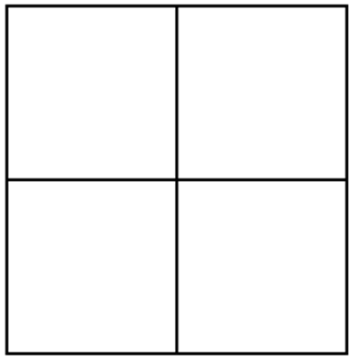
- **G** are the reciprocal space vectors. The Hilbert space spanned by **PWs** is truncated to a suitable cut-off E^{cut} such that

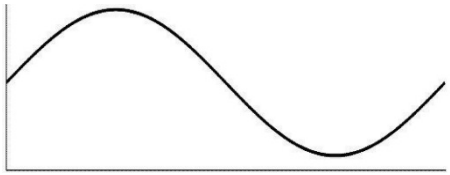
$$\mathbf{G}^2/2 < E^{cut}$$

Plane wave expansion: $\psi_i(\mathbf{x}) = \sum_{\mathbf{G}} c_i(\mathbf{G})\, e^{i\mathbf{G}\mathbf{x}}$

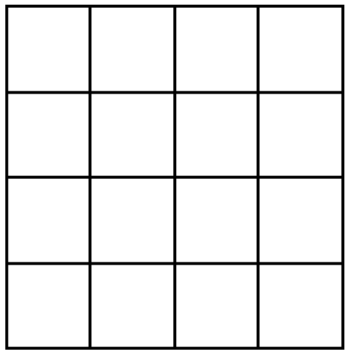
For each electron $i=1,\ldots,N$, $\mathbf{G}=1,\ldots,M$ are the reciprocal space vectors. The Hilbert space spanned by **PWs** is truncated to a cut-off $\mathbf{G}_{cut}^2/2 < E^{cut}$

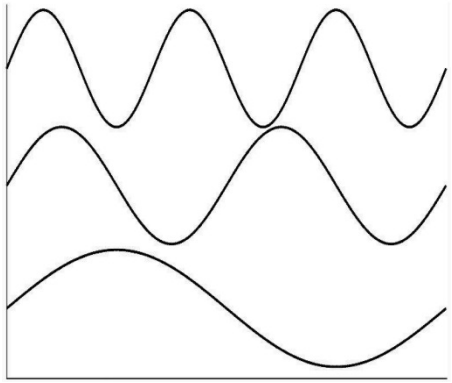
R space $\mapsto$ **G** space





$\mathbf{E}_1^{cut}$





$\mathbf{E}_2^{cut} > \mathbf{E}_1^{cut}$

Plane wave expansion: $\psi_i(\mathbf{x}) = \sum_{\mathbf{G}} c_i(\mathbf{G})\, e^{i\mathbf{G}\mathbf{x}}$

R space $\mapsto$ **G** space

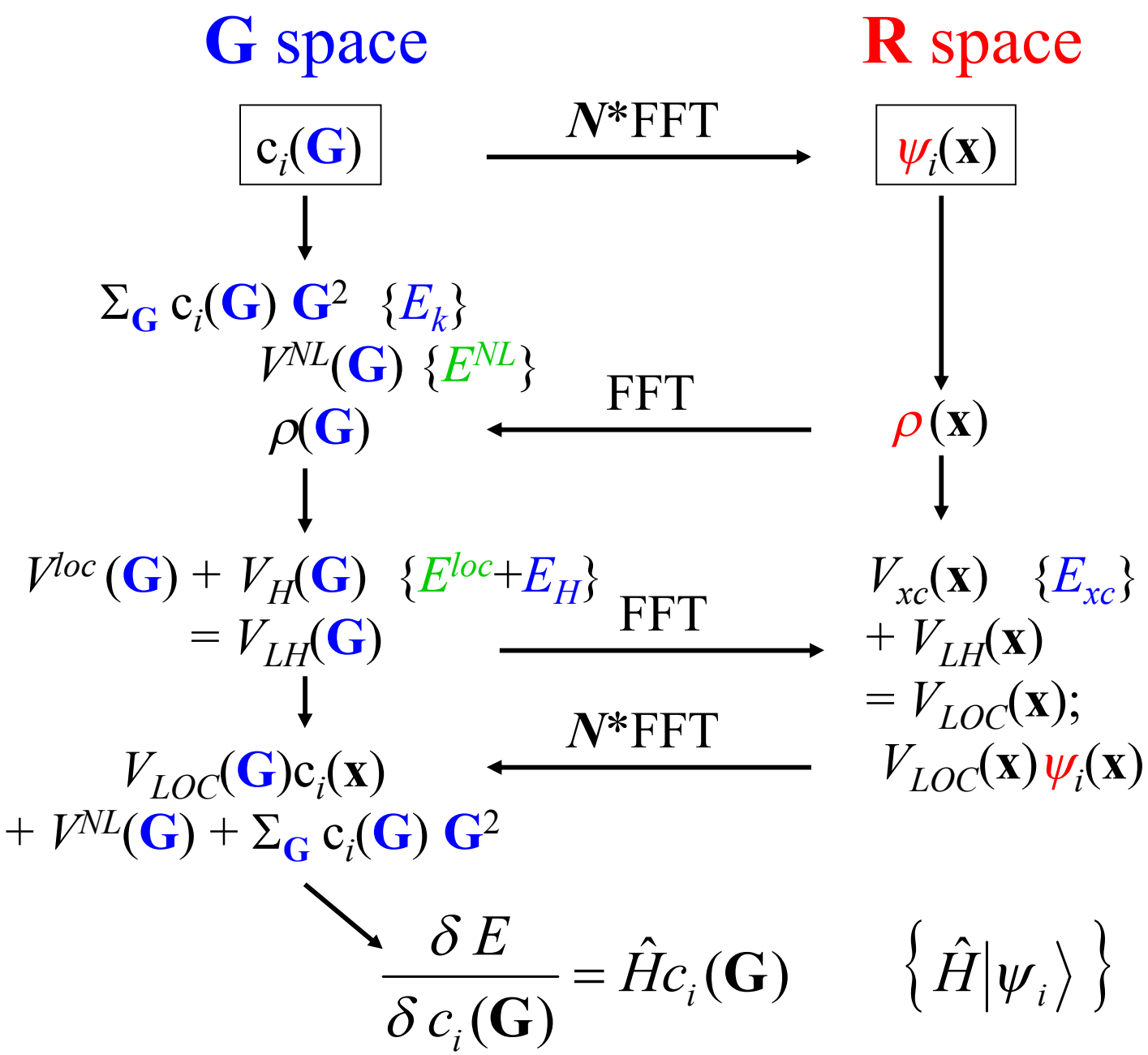
$$\psi_i(\mathbf{x}) = \begin{pmatrix} \psi_i(0,0,0) \\ \psi_i(1,0,0) \\ \dots \\ \dots \\ \psi_i(N_x, N_y, N_z) \end{pmatrix} + c.c. \qquad c_i(\mathbf{g}) = \begin{pmatrix} c_i(0,0,0) \\ c_i(1,0,0) \\ \dots \\ \dots \\ c_i(NG_x, NG_y, NG_z) \end{pmatrix} + c.c.$$

Double-precision floating-point format

From Wikipedia, the free encyclopedia

Double-precision floating-point format is a computer number format that occupies 8 bytes (64 bits) in computer memory and represents a wide, dynamic range of values by using a floating point.

Typical calculation with ~500 atoms: N_x x N_y x N_z = 500000

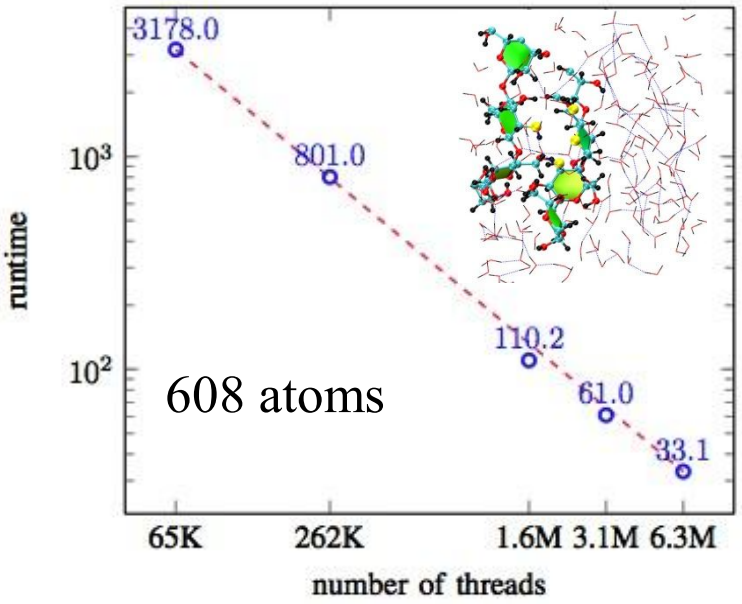

G space
R space
$c_i(\mathbf{G})$
N*FFT
$\psi_i(\mathbf{x})$
$\Sigma_{\mathbf{G}}\, c_i(\mathbf{G})\, \mathbf{G}^2 \quad \{E_k\}$
$V^{NL}(\mathbf{G})\ \{E^{NL}\}$
$\rho(\mathbf{G})$
FFT
$\rho(\mathbf{x})$
$V^{loc}(\mathbf{G}) + V_H(\mathbf{G}) \quad \{E^{loc}+E_H\}$
$= V_{LH}(\mathbf{G})$
FFT
$V_{xc}(\mathbf{x}) \quad \{E_{xc}\}$
$+ V_{LH}(\mathbf{x})$
$= V_{LOC}(\mathbf{x});$
N*FFT
$V_{LOC}(\mathbf{G})c_i(\mathbf{x})$
$+ V^{NL}(\mathbf{G}) + \Sigma_{\mathbf{G}}\, c_i(\mathbf{G})\, \mathbf{G}^2$
$V_{LOC}(\mathbf{x})\,\psi_i(\mathbf{x})$
$\frac{\delta E}{\delta c_i(\mathbf{G})} = \hat{H}c_i(\mathbf{G})$
$\left\{ \hat{H}\left|\psi_i\right\rangle \right\}$

Practical implementation

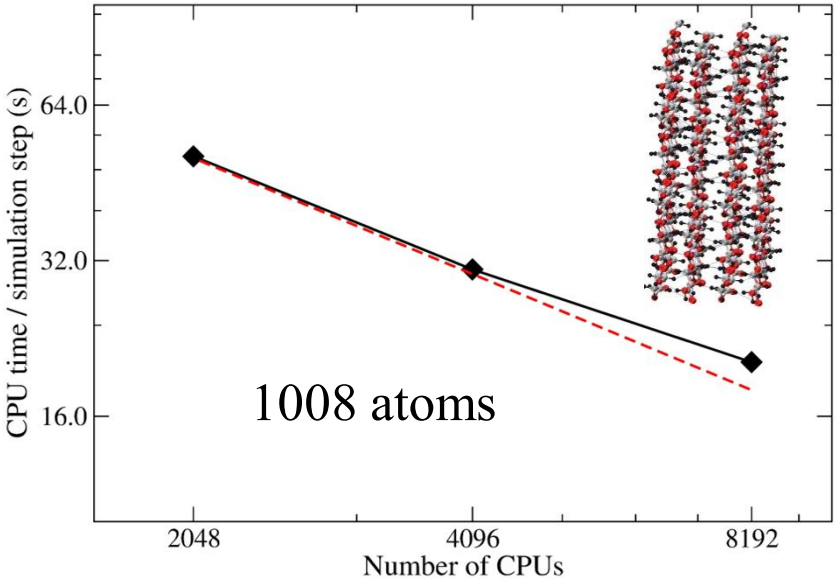
- **G=1,…,*M*** (*loop on reciprocal vectors*) distributed in a parallel processing in bunches of ***M*/(*nproc*)** or via MPI or hybrid MPI+OMP
- ***i* =1,…,*N*** (*loop on **electrons***) distributed (MPI / OMP)
- **k = 1,…,N_{kpt}** (loop on **k-points**) distributed (MPI / OMP)
- ***I* = 1,…,N_{Atoms}** (*loop on **atoms***) distributed (MPI / OMP), particularly useful in QM/MM simulations where MM ~ *O*(*N*)
- Parallel FFT: your own or libraries, e.g. fftw3 as in http://www.fftw.org/

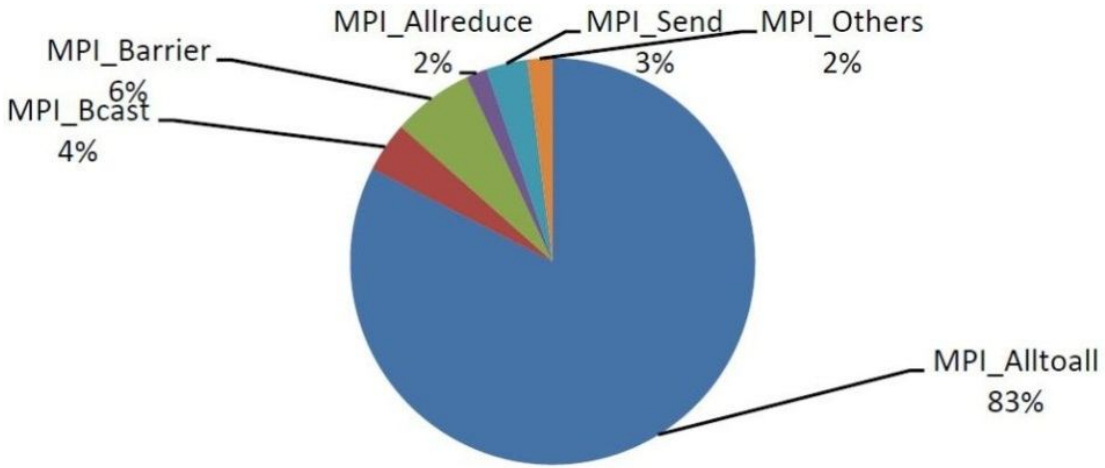
MPI workload distribution for CPMD

Performance on M of threads

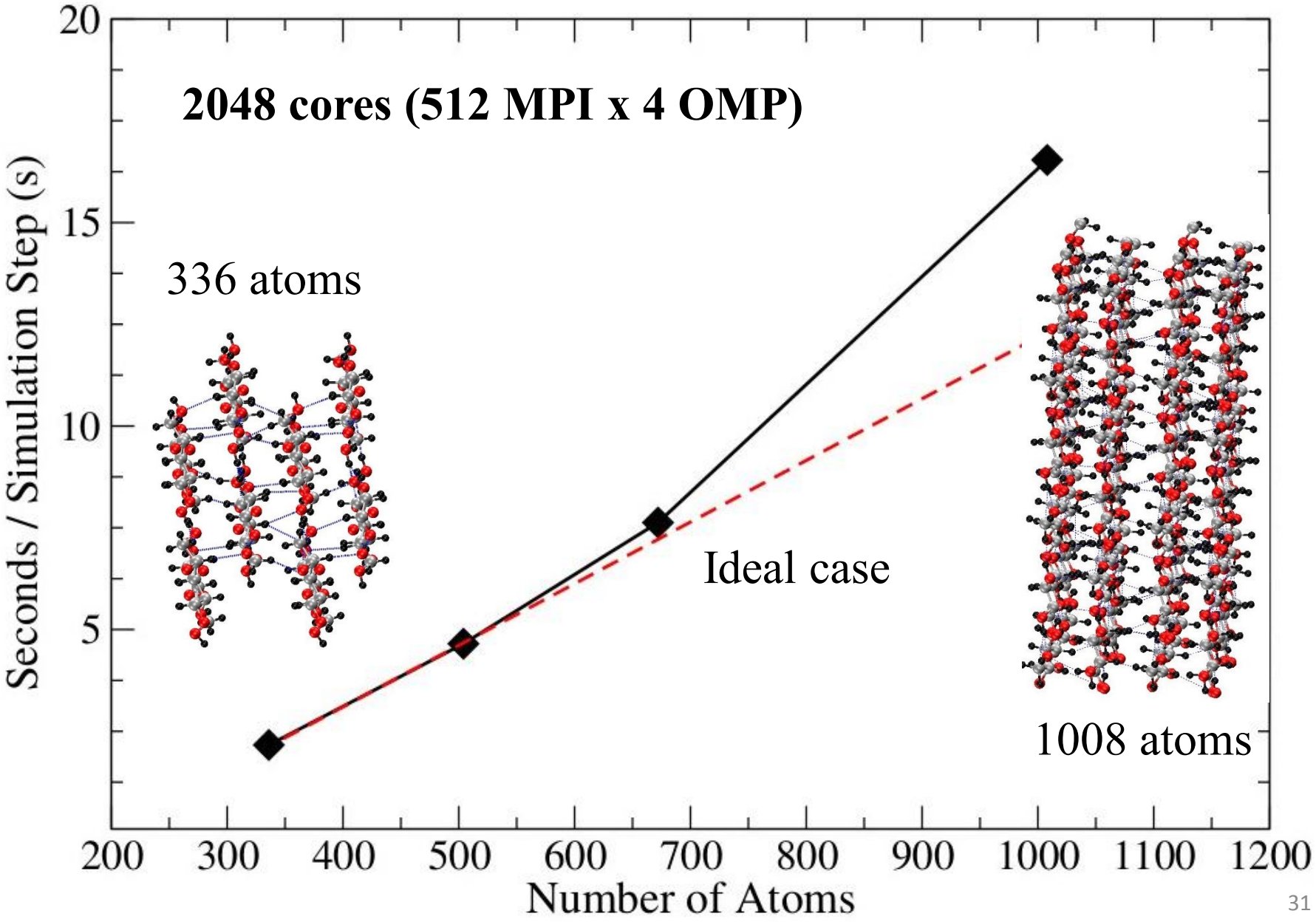


Performance on IBM BG/Q

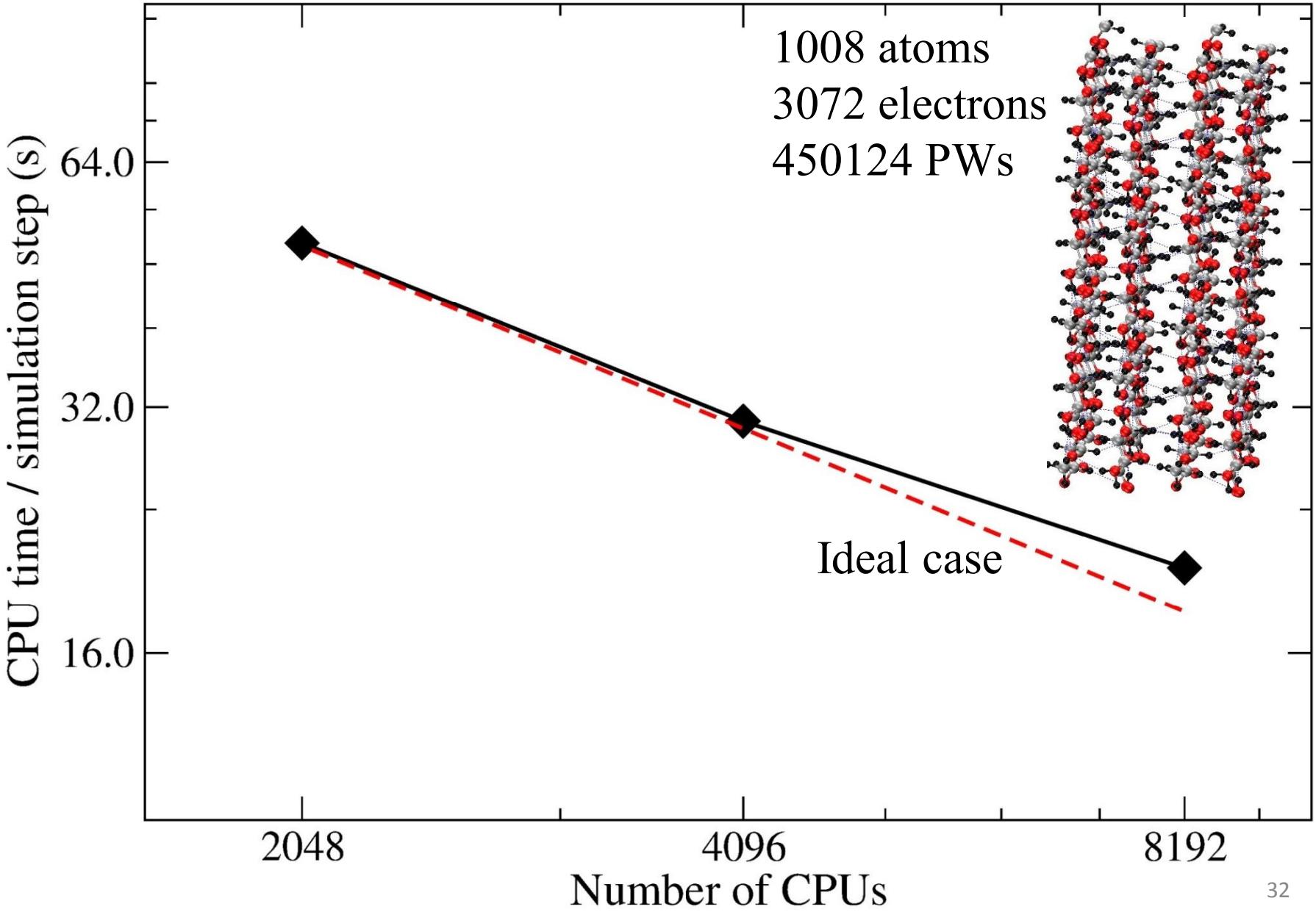




Scaling for a full QM system (β-cellulose) @ IBM-BG/Q



Scaling for a full QM system (β-cellulose) @ IBM-BG/Q



Thank you for your attention

ご清聴ありがとうございます

Vielen Dank für Ihre Aufmerksamkeit

Merci de votre attention

Grazie della vostra attenzione

Ευχαριστώ για την προσοχή σας

관심에 감사드립니다