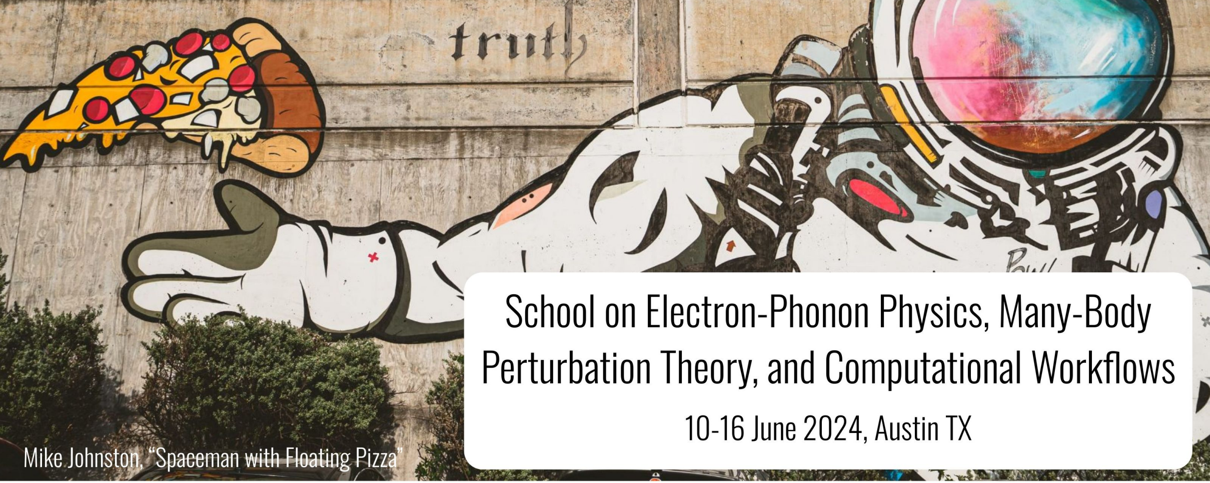


School on Electron-Phonon Physics, Many-Body Perturbation Theory, and Computational Workflows

10-16 June 2024, Austin TX

U.S. DEPARTMENT OF ENERGY

NSF

TACC TEXAS ADVANCED COMPUTING CENTER

ODEN

Hackathon Sat.1

QUANTUM ESPRESSO & GPU survival guide

Paolo Giannozzi

Dept. Mathematics, Computer Science, Physics, University of Udine, Italy
and CNR-IOM, Trieste, Italy

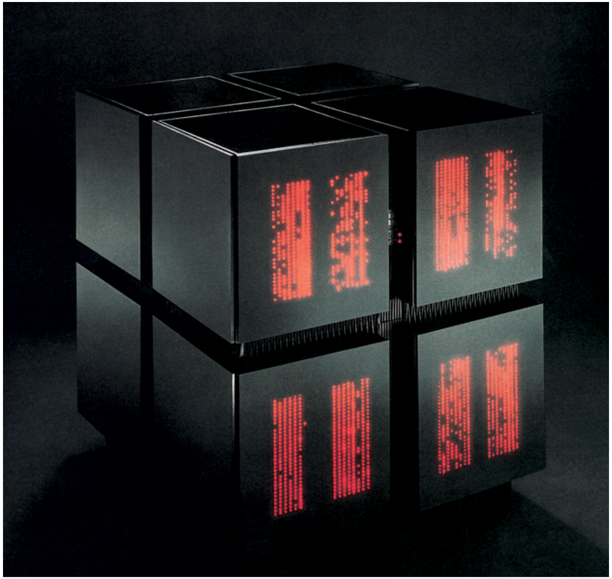
Once upon a time and now

One of the first (1985) commercial massively parallel machines, the Connection Machine CM-1, now on display in a museum.

It was an innovative advanced machine, expensive and really fast, and really hard to program.

For a few thousands $$ now you may have the modern equivalent of a CM under your desk (not recommended: it is hot and noisy).

The hard part: programming it...!



GPU basics (hardware)

- Accelerated hybrid architectures, aka GPUs (graphical processing units), are the current "big thing" in high-performance computing (HPC). The main reason: *more Flops per Watt* than conventional architectures.
- GPUs are a specialized piece of hardware that work like FPUs (floating-point units) "on steroids": they are physically connected to a conventional node and perform specific mathematical operations, exploiting a massive *internal parallelism* of the GPU.
- In a typical configuration, a few GPUs (say 1 to 4) are connected to a multi-core node with a few tens of cores (say 32 to 128). Big machines for HPC have many multi-core nodes, part or all of which have one or more GPUs.
- The GPUs have an *internal fast memory* that may be quite large (tens of Gb). The memory is fixed: you cannot buy a memory upgrade.
- GPUs do not directly access data on disk or other peripherals, they do that *via the CPU*. Depending upon the configuration, the GPUs of a single node may (or may not) directly communicate between them, e.g., via MPI calls.

Current GPU brands: NVidia, AMD, Intel

GPU basics (software)

- The code (or a single MPI process) runs on the CPU, transfers data to/from the GPU, instructs the GPU to make specific computations on those data.
- If your code/MPI process runs out of GPU memory, you are out of luck.
- GPUs are really fast when performing many operations in parallel, thus exploiting the internal parallelism. In this respect, they behave like old vector or parallel machines.
- GPUs are really fast if data is already on GPU, but moving data from CPU to GPU and vice versa is slow and *must be kept to the strict minimum*.
- Currently there is no way to write code for GPU using machine-independent portable coding, and no automatic acceleration either. Each GPU brand comes with its own set of compilers, libraries, supported languages.
  - NVidia: CUDA software stack, CUDA Fortran extensions, OpenACC directives
  - AMD: ROCm software stack, OpenMP v.5 directives
  - Intel: oneAPI software stack, OpenMP v.5 directives

MaX goals and philosophy

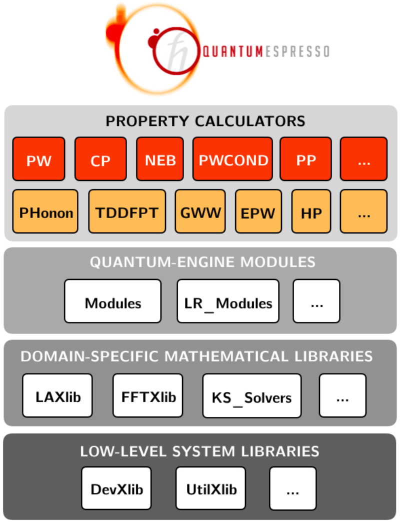
A software infrastructure for

- *Sustainable software development*
- *Performance portability*

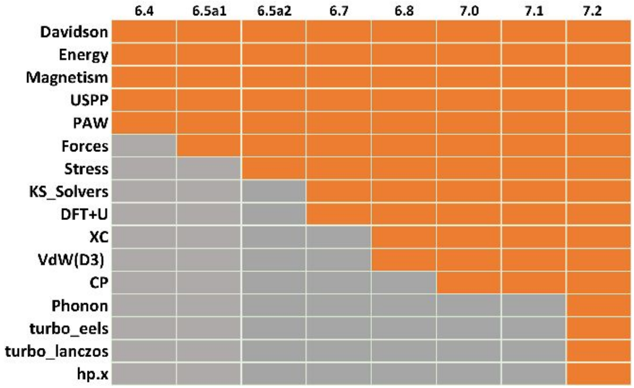
for exascale applications(*), via:

- *Separation of concerns*:
  scientists work on science (top layers in figure), IT people on computers (bottom layers)
- *Co-design*:
  scientists and IT people work together with hardware vendors

(*) suitable for exascale machines, that is, capable of 10^{18} flops



Porting to heterogeneous architectures



	6.4	6.5a1	6.5a2	6.7	6.8	7.0	7.1	7.2
Davidson	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Energy	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Magnetism	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
USPP	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
PAW	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Forces	⬜	🟧	🟧	🟧	🟧	🟧	🟧	🟧
Stress	⬜	⬜	🟧	🟧	🟧	🟧	🟧	🟧
KS_Solvers	⬜	⬜	⬜	🟧	🟧	🟧	🟧	🟧
DFT+U	⬜	⬜	⬜	🟧	🟧	🟧	🟧	🟧
XC	⬜	⬜	⬜	⬜	🟧	🟧	🟧	🟧
VdW(D3)	⬜	⬜	⬜	⬜	🟧	🟧	🟧	🟧
CP	⬜	⬜	⬜	⬜	⬜	🟧	🟧	🟧
Phonon	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟧
turbo_eels	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟧
turbo_lanczos	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟧
hp.x	⬜	⬜	⬜	⬜	⬜	⬜	⬜	🟧

The QUANTUM ESPRESSO suite has been accelerated using a mixed CUDA Fortran/OpenACC scheme. A version based on OpenMP offloading is under heavy development, in order to enhance portability to hardware from different vendors.

A rather dumb example of CUDA Fortran

```
USE cudafor
attributes( device ) :: h_d, s_d, e_d, psi_d
...
!$cuf kernel do(3) <<<*,*>>>
do ipol=1,npol
   do k = 1, m
      do i = 1, n
         denm = h_d (i,ipol) - e_d (k) * s_d (i,ipol)
         if (abs (denm) < eps) denm = sign (eps, denm)
         psi_d (i, ipol, k) = psi_d (i, ipol, k) / denm
      enddo
   enddo
enddo
psi = psi_d
...
```

Arrays with attribute `DEVICE` are on device (GPU), all others are on host (CPU)

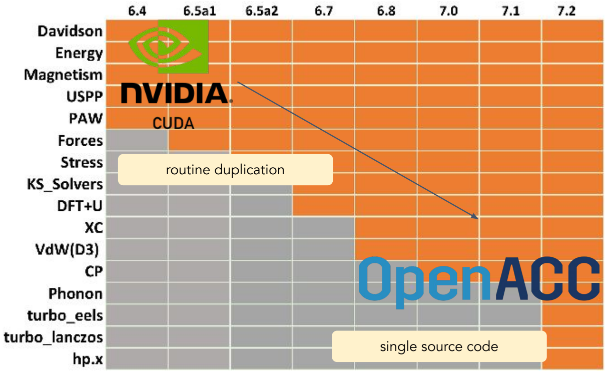
An equally dumb example with OpenACC

```
...
!$acc present( h, s, e, psi)
...
!$acc parallel loop collapse(3)
do ipol=1,npol
   do k = 1, m
      do i = 1, n
         denm = h (i,ipol) - e_d (k) * s (i,ipol)
         if (abs (denm) < eps) denm = sign (eps, denm)
         psi (i, ipol, k) = psi (i, ipol, k) / denm
      enddo
   enddo
enddo
!$acc update host(psi)
...
```

Arrays can copied from host (CPU) to device (GPU) and vice versa

Towards a portable GPU version

The transition from CUDA to Openacc



DIRECTIVE-BASED PROGRAMMING MODELS

MAINTAINABLE

PORTABLE

SINGLE SOURCE CODE

GPU and QUANTUM ESPRESSO: state of the art

Currently, the development of QUANTUM ESPRESSO for GPUs relies on

- OpenACC for NVidia GPUs.
  CUDA Fortran is being slowly phased out (with a few exceptions)
  Work still ongoing (and will always be) but basically production-ready
- OpenMP v.5 for AMD and Intel GPUs
  Experimental, approaching production-ready state for AMD (LUMI)
  Available as a branch of the development `git` repository

Running on GPUs, in practice

In general: most GPUs, notably ALL cheap ones, do not have double-precision floating-point operations as main target. Do not expect spectacular performances from those GPUs.

Relevant parameters affecting performances are

1. Floating-point GPU performances (increases for increasing $$$)
2. Available GPU memory (the more, the better, but of course more $$$)
3. How well one can distribute the load (see below)

In general: use GPU wisely, not massively. Quality instead of quantity!

1. *Run one MPI process per GPU*. Oversubscription, i.e., running multiple MPI processing on a GPU, is seldom a good idea.
2. *Prefer low-communication parallelization levels*: k-points for scf calculations, wave-vectors and irreps for phonon calculation, whatever is available.
3. Use plane-wave parallelization *only if you need to distribute memory*. Each MPI process has to fit into the memory of the connected GPU. Plane-wave parallelization works well but it involves significant inter-GPU communications. Use GPU MPI if available.

Some references

1. *Quantum ESPRESSO toward the exascale*, P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, J. Chem. Phys. **152**, 154105 (2020). DOI: https://doi.org/10.1063/5.0005082
2. *Quantum ESPRESSO: one further step towards the exascale*, I. Carnimeo, F. Affinito, S. Baroni, O. Baseggio, L. Bellentani, R. Bertossa, P. Delugas, F. Ferrari Ruffino, S. Orlandini, F. Spiga, and P. Giannozzi, J. Chem. Theory Comput. 19, 6992 (2023)
3. *Quantum ESPRESSO towards performance portability: GPU offload with OpenMP*, F. Ferrari Ruffino, L.Bellentani, G. Rossi, F. Affinito, S. Baroni, O. Baseggio, P. Delugas, P. Giannozzi, J. Kurzak, Ye Luo, O. O'Reilly, S. Orlandini, I. Carnimeo, Proceedings of the First EuroHPC user day, Procedia Computer Science (2024) in press.