Introduction to
High-Performance Computing

Giorgio Amati
Alessandro Ceci

Corso di dottorato in Ingegneria Aeronautica e Spaziale 2025

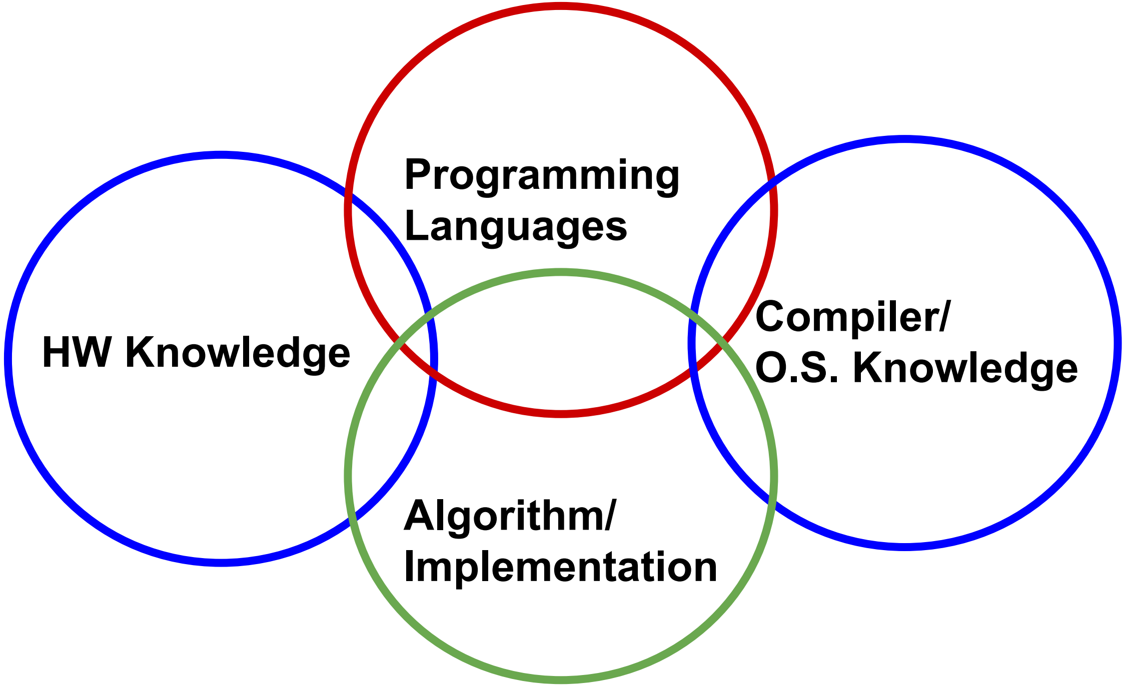
g.amati@cineca.it/g.amaticode@gmail.com
alessandro.ceci@uniroma1.it

Agenda

- ✓ HPC: What is it?
- ✓ Hardware: how it works
- ✓ Algorithm vs. Implementation
- ✓ Compiler + Floating point + I/O
- ✓ HW & Parallel Paradigm
  - Shared Memory parallelization: openmp et al.
  - GPU programming
  - Distributed Memory parallelization: MPI
- ✓ Conclusions & Comments

HPC: what it is?

✓ These are the main skills for an efficient HPC



Example: matrix-matrix multiplication/2

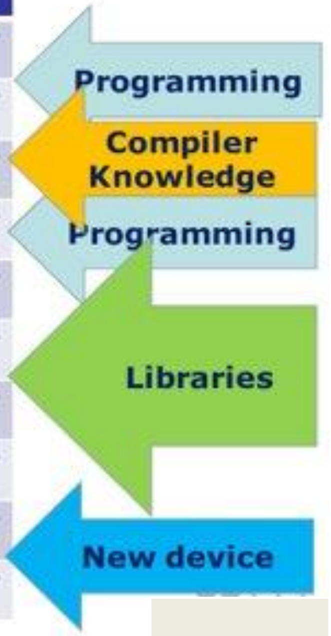
- ✓ Performance can really different, depending on HW, implementation etc….
- ✓ Improvement in performance can be really high….
- ✓ ….or you can easily “depress” performance
- ✓ Performance in Mflops: higher is better

#test	Size	HW	MFlops	Ratio
1-Cache unfriendly	2048	CPU	201	-
2-Cache friendly	2048	CPU	4870	**24x**
3-OpenACC	8192	GPU-V100	361328	**1797x**
4-OpenACC+unrolling	8192	GPU-V100	448923	**2233x**
5-Matmul	16384	GPU-A100	6721790	**33441x**

Performance improvements

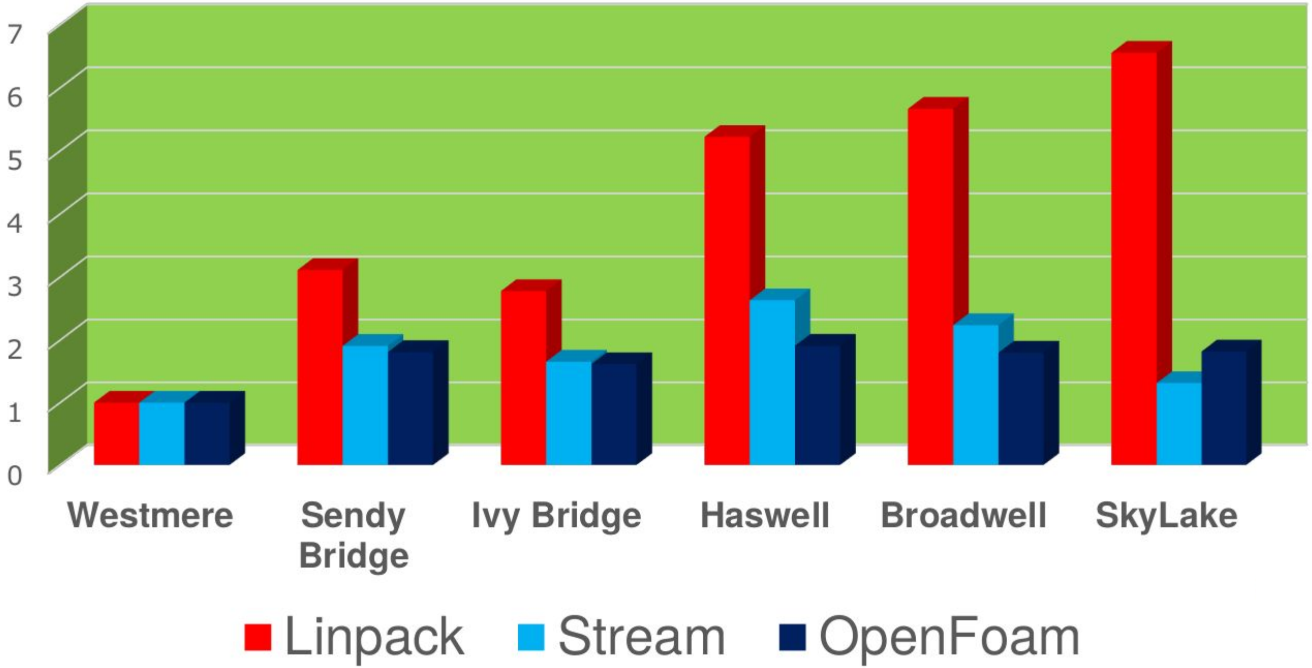
- Matrix-Matrix multiplication (time in seconds)

	Single prec.	Double prec.
Cache un-friendly loop	7500″	7300″
Cache friendly loop	206″	246″
Compiler Optimization	84″	181″
Handmade Optimization	23″	44″
Optimized library (serial)	6.7″	13.2″
Optimized library (OMP, 2 threads)	3.3″	6.7″
Optimized library (OMP, 4 threads)	1.7″	3.5″
Optimized library (OMP, 8 threads)	0.9″	1.8″
PGI accelerator (GPU)	3″	5″
CUBLAs (GPU)	1.6″	3.2″



(Serial) Performance Evolution

Old slide but still valid now!



Be flexible…..

✓ LBM, 3D lid driven cavity, Performance in MLUPs (high is better)

Processor	Par. Paradigm	Task/thread	Implementation	Mlups	Sustained perf.
Intel KNL	OpenMP	64 threads	Fused	1421	10%
Nvidia V100	OpenACC	-	Fused	3419	11%
Nvidia A100	OpenACC	-	Fused	7454	9.5%
Fujitsu A64fx	Hybrid	4 tasks/12 threads	Stream-Collide	1346	9.8%

✓ "Projecting LBM performance on Exascale class Architectures: A tentative outlook." G Amati et al. Journal of Computational Science 55, 101447

comp.lang.fortran (12/03/03)

How to decrease the run time of a code:

1. Improve the algorithm.
2. Same as 1).
3. Tweak the code.
4. Fiddle with compiler switches.
5. Run it on a faster computer :-).
6. Ignore all compiler-provided error checking (a follow-on from [4]).
7. Minimise small subroutines - make them inline.
8. Check array accesses are 1st-index varying fastest.
9. Ensure that the code is working correctly.
10. Ensure that nothing major is accidentally calculated twice.
11. If differential equations: look at alternative solvers, and whether the tolerances are appropriate.
12. If algebraic equations or optimisation - as [11].
13. Minimise saving data to a file - try & do it only at the end, rather than during the course of calculations.
14. Minimise use of automatic arrays, or deallocating arrays.
15. Ensure that the machine is not limited by memory constraints (too much page-swapping).
16. Return to fortran77-style array parameters, rather than fortran90 usage of array descriptors.

Twelves way to fool people about ….

1. Quote only 32-bit performance results, not 64-bit results.
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
3. Quietly employ assembly code and other low-level language constructs.
4. Scale up the problem size with the number of processors, but omit any mention of this fact.
5. Quote performance results projected to a full system.
6. Compare your results against scalar, unoptimized code on Crays.
7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
9. Quote performance in terms of processor utilization, parallel speedups orMFLOPS per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

Different Level of optimization

- ✓ **Single core optimization**
  - Vectorization
  - Data access
  - Serial Optimized libraries
  - Compiler optimization
- ✓ **Single-node optimizations**
  - Intra-node optimization
  - Data access
  - Shared memory Parallelization/Distributed memory Parallelization
  - Offloading
  - Shared Memory Optimized Libraries
- ✓ **Multi-node optimizations**
  - Distributed memory optimization (i.e. load balancing)
  - Parallel Optimized libraries
- ✓ **I/O issues**

You cannot skip one single level of optimization

Some Books

- ✓ Charles Severance; Kevin Dowd “High Performance Computing”, O’Reilly, ISBN 13:9781565923126
- ✓ John L. Hennessy, David A. Patterson , “Computer Architecture: A Quantitative Approach” Morgan Kaufmann; ISBN-10 : 0128119055
- ✓ John L. Hennessy, David A. Patterson , “Computer Organization and Design: The Hardware/Software Interface”, Morgan Kaufmann;
- ✓ D. Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic
- ✓ U. Drepper: “What Every Programmer Should Know About Memory”