High Performance Computing
and
High Performance humans in Computing

Maria Grazia Pia
INFN Genova

mariagrazia.pia@ge.infn.it
maria.grazia.pia@cern.ch

IAEA (Virtual) Workshop on
Computational Nuclear Science and Engineering
12-16 July 2021
MGP

- Does not like writing bios
- Physicist at INFN (Istituto Nazionale di Fisica Nucleare), Section of Genova, Italy
  - Pre-covid: large fraction of time at CERN, looking forward to post-covid era
- Associate Editor of IEEE Transactions on Nuclear Science
- Moderator of arXiv Computational Physics (physics.comp-ph)
- High Energy Physics background
  - CERN, FNAL, SLAC
- Monte Carlo development (Geant4) and applications, physics data libraries validation
- Statistical data analysis, scientometrics, epistemology
- No Facebook, Twitter, LinkedIn, Instagram… but google me, and you will find track of my research activity
What is HPC?

10^6 CPU cores
1 exabyte storage
>170 sites
42 countries

Summit supercomputer

>27000 NVIDIA Volta GPUs
> 9000 IBM Power9 CPUs

exploit workload parallelism to achieve scalability

manage data to achieve efficiency of operation

HP
HPC systems

HPC computers are networks of processors
- Very fast memories
- Low-latency, high-bandwidth communication systems
  • between the processors
  • between the processors and the associated memories

Homogeneous: only CPUs
Hybrid: CPUs + GPUs
- GPUs can handle millions of threads simultaneously, are more energy efficient, have faster memories, require less data transfer
- CPUs oversee the computation

Key features of HPC architecture
- speed of operation
- parallelism to perform multiple operations
- efficient use of critical components
- electrical power that it consumes
- reliability
- how easy to program

Maria Grazia Pia, INFN Genova
Top supercomputers

<table>
<thead>
<tr>
<th>Rmax PFLOPS</th>
<th>Location</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>442.010</td>
<td>RIKEN, Japan</td>
<td>Fujitsu</td>
</tr>
<tr>
<td>148.600</td>
<td>ORNL, USA</td>
<td>IBM</td>
</tr>
<tr>
<td>94.640</td>
<td>LLNL, USA</td>
<td>IBM</td>
</tr>
<tr>
<td>93.015</td>
<td>Natl. Supercomp. Centre, China</td>
<td>NRCPC</td>
</tr>
<tr>
<td>64.590</td>
<td>NERSC, USA</td>
<td>Nvidia</td>
</tr>
</tbody>
</table>

Performance

<table>
<thead>
<tr>
<th>Country</th>
<th>PFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>669</td>
</tr>
<tr>
<td>Japan</td>
<td>594</td>
</tr>
<tr>
<td>China</td>
<td>564</td>
</tr>
</tbody>
</table>

FLOPS: floating-point operations per second

Supercomputer performance

System share by application area

- Research: 43.8%
- Weather and Climate Research: 12.5%
- Energy: 12.5%
- Benchmarking: 12.5%
- Aerospace: 12.5%
- Information Service: 6.3%
- Semiconductor: 12.5%

CC BY-SA 3.0 Al.Graphics
Commodity clusters

- A group of integrated computer systems
  - Standalone components (COTS), capable of independent operation
  - Integration network is separately developed
  - Off the shelf mass storage
  - Interfaces adhere to industry standards

- Very successful
  - First one in 1997, ~50% of Top 500 systems in 2005, ~ 85% today

- Parallel programming modalities
  - **Throughput computing**: efficiently run a large number of jobs that are independent or require minimal communication
  - **Message-passing**: requires a significant amount of communication and coordination within the application
    - communicating sequential processes model, exemplified by the MPI
  - **Shared-memory multiple-thread applications**
    - exemplified by the OpenMP (open multiprocessing) programming model
LHC data are currently handled by the Worldwide LHC Computing Grid (WLCG). WLCG is a massive distributed computing infrastructure.

CMS estimated resources required HL-LHC era


Data volumes in HL-LHC at the multi-Exabyte scale

LHC experiments are exploring GPUs for accelerated event reconstruction and simulation.
Parallel algorithms

Physical

- SIMD: single-instruction multiple data parallelism
- MIMD: multiple-instruction multiple data
  - shared memory parallelism
  - distributed memory parallelism

Multiple parts of the workload are performed concurrently to reduce the time to achieve the solution

Several parallel algorithms are used in scientific computing
- fork–join
- divide and conquer
- halo exchange
- permutation
- embarrassingly parallel
- manager–worker
- task dataflow

Emerging numerical methods in supercomputing applications:
- graph traversal
- finite state machines
- combinational logic
- statistical machine learning

Some algorithms are better suited for one kind of physical parallelism versus another
Chances are that you would deal with HPC and parallel computing in Monte Carlo simulation in particle/nuclear physics

- **Embarrassingly parallel**
  - Parallelism with essentially no inter-task communication
  - Highly partitionable workload with minimal overhead
  - Concurrency is trivially extracted from the workflow
  - Often require gathering the results at the end into a manager process

- **Monte Carlo transport is naturally suitable to event-level parallelism**

- **Methods for embarrassing parallel simulation documented in most popular Monte Carlo particle transport codes**
  - **Threading**: OpenMP threading on a single multicore computer or on a single node of a server or cluster
  - **Message-passing** between nodes on a cluster using MPI environment
  - Used separately or together
Multi-threaded simulations

Ability to exploit hardware multi-threading capabilities

Goal: reduce the memory footprint of parallel applications, while preserving the linear speedup as a function of the number of physical cores

Based on a master–worker model

- master thread prepares geometry and physics setups
- worker threads compete for the next events to be simulated

an application example

Min Cheol Han et al, Multi-threading performance of Geant4, MCNP6, and PHITS Monte Carlo codes for tetrahedral-mesh geometry, 2018 Phys. Med. Biol. 63 09NT02

Maria Grazia Pia, INFN Genova
Quantum computing is a computing paradigm that exploits quantum mechanical properties (superposition, entanglement, interference...) of matter to do calculations.

Quantum computing

Quantum supremacy using a programmable superconducting processor


https://doi.org/10.1038/s41586-019-1666-5

Received: 22 July 2019
Accepted: 20 September 2019
Published online: 23 October 2019

In quantum circuits:
- data = qubits
- operations = quantum gates
- results = measurements

The Sycamore processor can run a test computation in 200 seconds that would take the world’s biggest supercomputers 10000 years to complete.

IBM researchers claim Google's challenge would take a classical computer just two and half days.

China beats Google to claim the world's most powerful quantum computer

5 July 2021

By Matthew Sparkes
High Performance humans in Computing

Technology Methodologies
Producing results

Measurements over 10 years
Publicly accessible data
Fred P. Brooks,

“No Silver Bullet - Essence and Accidents of Software Engineering”

*IEEE Computer*, vol. 20, no. 4, pp.10-19, April 1987

As we look to the horizon of a decade hence, we see *no silver bullet*. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.

... 

Although we see no startling breakthroughs - and indeed, I believe such to be inconsistent with the nature of software - many encouraging innovations are under way. A *disciplined, consistent effort* to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

**Introduction to concepts and methods**

📌 Pills of wisdom
📌 Food for thought
📌 Curiosity
📌 Background for further learning
Software development **methods** and **techniques** are seldom part of academic programs for physics and engineering degrees.

**Cowboy programming**

**Emphasis on ingenious artistry**

- Galloping off on one's own without a prior plan
- Brute-force programming
- Uncertain design requirements, code rewrite
- Quick and dirty: code and fix later
- Lack of comments, documentation, reviews
- Reinventing the wheel

The results are often spotty and difficult to duplicate.

Inexperienced developers are unfamiliar with **technologies** and **methodologies** that support producing quality software effectively.
Much more than just hacking code…

These complex disciplines include activities, generate products and involve responsibilities in various roles articulated over the software life-cycle: get-go, elaboration, construction, use, maintenance…

Built on best practices derived from experience

Software development methodologies are conceptual frameworks to structure, plan and control the process of developing the software

Highly prescriptive Adaptable to the context

Wide variety

Small projects Large scale projects
Old, risky... and most common

Cascade of phases: the output of one is input to the next

Emphasis on **planning**

Difficult to accommodate **change**

Risk of discovering problems at a late stage of the project
Variant of waterfall: V-model

Emphasis on testing at all levels of software development

Each development phase is associated with a testing phase
Spiral development

Grasp the nettle: non-linear view of the software life cycle

Set objectives

Assess and mitigate risks

Emphasizes risk management

- Identify risks
- Assign priorities to risks
- Develop a series of prototypes for the identified risks
- Use a waterfall model for each development loop

loop in the spiral = phase of software development


Maria Grazia Pia, INFN Genova
Unified Process (UP, USDP, RUP)

Iterative, incremental process, with emphasis on modeling

Complex, but highly customizable

Best practices

- Develop software iteratively
  - High priority features developed first
- Manage requirements
  - Document requirements
  - Keep track of changes
  - Analyze the impact of changes before accepting them
- Use component-based architectures
  - Structure the system into components
- Visually model software
  - UML (Unified Modeling Language)
- Verify software quality
  - Test (and more)
- Control changes to software
  - Change management system
  - Configuration management and tools
Kent Beck et al. (2001)

Manifesto for Agile Software Development
https://agilemanifesto.org/

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Emphasis on
- Effective communication among all stakeholders
- Adaptive response to change
- Rapid, incremental delivery of software

Agile Methods

B. Boehm, “Get Ready for Agile Methods, With Care”, IEEE Computer, 2002, http://dx.doi.org/10.1109/2.976920
B. Meyer, Agile!: The Good, the Hype and the Ugly, Springer, 2014
R. C. Martin, Clean Agile: Back to Basics, Prentice Hall, 2019
Scrum

Project management for agile (incremental) development

Short, daily meetings to review progress, reprioritize

Starting point for planning

Sprint

2-4 weeks fixed length

Potentially shippable product increment

Scrum master is a facilitator

- arranges daily meetings
- tracks the backlog of work to be done
- records decisions
- measures progress against the backlog
- communicates with customers and management

Starting point for planning

S. Ockerman and S. Reindl, Mastering Professional Scrum: A Practitioners Guide to Overcoming Challenges and Maximizing the Benefits of Agility, Addison-Wesley, 2019
Many different approaches are possible

- Positive and negative sides in any of them
- **Good or bad** often depends on the **context**
- Small/large scale project, short/long lifetime etc.

Process frameworks may (**should**) be adapted and extended

- A good software process is **tailored to the project**

Grain of salt

- Commercial vs. scientific environment
- …let’s not forget that often we are not only the developers, but also the customers!
How to improve the way we develop software?

Improvement requires **measurement**: quantify *before/after*

Software Capability Maturity Model

**CMM, CMMI**

- **Initial**
  - Processes unpredictable, poorly controlled and reactive
- **Managed**
  - Processes characterized for projects and is often reactive.
- **Defined**
  - Processes characterized for the organization and is proactive. (Projects tailor their processes from organization’s standards)
- **Quantitatively Managed**
  - Processes measured and controlled
- **Optimizing**
  - Focus on process improvement


Helpful guidance towards adopting good practices


For singles

What if I work at a project where I am the only software developer?

The benefits of sound methodologies are not restricted to large scale projects or sizeable teams.

https://doi.org/10.1184/R1/6585197.v1
Further learning

Get a mentor!

Maria Grazia Pia, INFN Genova
Technology

- No time for an extensive overview and in-depth analysis
- A few highlights on key technologies and tools

Modelling  Testing  Tools

Dealing with legacy code

Basic techniques of programming hygiene

“agile software craftsmanship”

“If you have been a programmer for more than two or three years, you have probably been significantly slowed down by someone else’s messy code.” R. C. Martin, Clean Code

[Your own?]
# Best practices for clean code

## Names
- Choose descriptive and unambiguous names
- Make meaningful distinction
- Use pronounceable names
- Use searchable names
- Replace magic numbers with named constants
- Avoid encodings. Don’t append prefixes or type information

## Source code structure
- Separate concepts vertically
- Related code should appear vertically dense
- Declare variables close to their usage
- Dependent functions should be close
- Similar functions should be close
- Place functions in the downward direction
- Keep lines short
- Don’t use horizontal alignment
- Use white space to associate related things and disassociate weakly related
- Don’t break indentation

## Tests
- One assert per test
- Readable
- Fast
- Independent
- Repeatable

## Code smells
- Rigidly. The software is difficult to change. A small change causes a cascade of subsequent changes
- Fragility. The software breaks in many places due to a single change

## Design rules
- Keep configurable data at high levels
- Prefer polymorphism to if/else or switch/case
- Separate multi-threading code
- Prevent over-configurability
- Use dependency injection
- Follow Law of Demeter A class should know only its direct dependencies

## Functions
- Small
- Do one thing
- Use descriptive names
- Prefer fewer arguments
- Have no side effects
- Don’t use flag arguments. Split method into several independent methods that can be called from the client without the flag

## Understandability
- Be consistent. If you do something a certain way, do all similar things the same way
- Use explanatory variables
- Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place
- Prefer dedicated value objects to primitive type
- Avoid logical dependency. Don’t write methods which works correctly depending on something else in the same class
- Avoid negative conditionals

## Comments
- Always try to explain yourself in code
- Don’t be redundant
- Don’t add obvious noise
- Don’t use closing brace comments
- Don’t comment out code. Just remove

## Object and data structure
- Hide internal structure
- Prefer data structures
- Avoid hybrids structures (half object and half data)
- Should be small
- Do one thing
[Visual] Modeling

- Enormously helpful
  - **Think before typing** implementation code on the keyboard
  - Bird’s-eye view of the software (and of the hardware)
  - Visualize relationships among the players, which may not be easy to catch just by looking at source code

- **UML** (Unified Modelling Language)
  - Standard, lingua franca for the communication of models
  - Plenty of educational material and tools, from simple to very powerful
  - Umbrello (https://umbrello.kde.org/) ➤ ... Sparx Enterprise Architect™

- **Diagrams**: class, package, component, deployment, object, state, activity, sequence, interaction, use case...

- **Design patterns**
  - elements of reusable object-oriented software
Model of the a package for the management of cross sections data for electron-photon interactions

Atomic data libraries are used by all major Monte Carlo codes for particle transport.
High level requirements of a software system for the verification of atomic data libraries
Validation test workflow

UML activity diagram
Workflow of a test for the validation of atomic data libraries
**ArchiMate** diagram of the computational environment of a software system for the verification test of atomic data libraries
How do you trust the software you use?

"I'm just doing what the other ones are doing"

“Testing shows the presence, not the absence, of bugs.” In other words, a program can be proven incorrect by a test, but it cannot be proven correct. All that tests can do, after sufficient testing effort, is allow us to deem a program to be correct enough for our purposes.

Edsger Wybe Dijkstra
Test

Levels of testing
- Unit
- Integration
- System
- Acceptance

Regression testing
Performance testing
Stress testing
Configuration testing
Security testing

Verification test
Functional/non-functional testing
Black/white-box testing

Validation test
Test harness
Test automation

Test coverage
Test planning
Test frameworks

Test cases

Validation test... etc.
Test-driven development (TDD)

- Essential: an automated testing environment
- Write a test that fails (before writing the code that implements new functionality)
- Write code that makes the test pass
- Improve the design of the software
- Suitable to small-size projects

Facilitates regression testing
Discover problems early during the software development

Limited to unit testing,
still need system testing, performance, reliability testing etc.

“If it ain’t broken, don’t fix it”

A piece of software can be broken in many ways

- **Functional**
  - it no longer delivers the function it is designed to perform

- **Maintenance**
  - it can no longer be maintained

- Obsolete or no **documentation**
- Missing **tests**
- Original **developers** or users have left
- **Inside knowledge** about the system has disappeared
- Limited understanding of the **entire system**
- Too long to turn things over to **production**
- Too much time to make **simple changes**
- Need for constant **bug fixes**
- **Big** **build times**
- Difficulties **separating** products
- **Duplicated** code
- Code **smells**

**Warnings you are heading into trouble**

*usually do not occur isolated*

*S. Demeyer, S. Ducasse, O. Nierstrasz, Object Oriented Reengineering Patterns*
In the ideal world there would be hardly any need for refactoring.

In the real world most software needs to be refactored.

By learning refactoring you also learn writing code that minimizes the need to be refactored.

Refactoring is a disciplined technique for improving the design of an existing code.
Further learning

Get a mentor!
The End

HPC = Technology + Methodology

Master them, so that you can choose what is most appropriate to your research problem

This 40’ introduction only scratches the surface, more to follow next year

Feel free to contact me for further information and for suggestions for next year

Maria Grazia Pia, INFN Genova