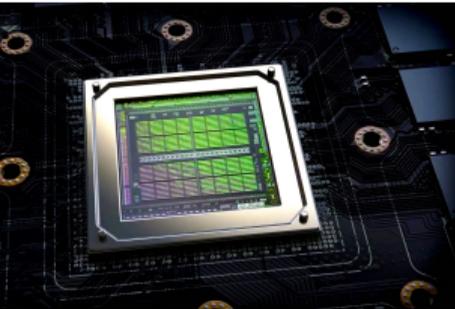




julian.kunkel@gwdg.de

Julian Kunkel

Welcome to the Practical Course on High-Performance Computing



Architecture programming
parallel computing
C++ oneAPI OpenMP syscl
efficiency OpenCL
enjoyful
CUDA ManyCore
Libraries OpenCL Python
supercomputer



Recording!

- This broadcast channel will be recorded via BBB
 - ▶ This includes your video, audio (if shared), and chat messages
 - ▶ We can start/stop video recording if necessary
- Recordings will be available 1-2 days later
- We **may publish** selected trainings on our YouTube channel
 - ▶ Will include video, audio if shared
 - ▶ Use the chat in broadcast in the case of questions
It won't be rendered for the YouTube video

Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing
- 4 Distributed Computing
- 5 Parallel Computing
- 6 Programming
- 7 Conclusions

Learning Objectives of the Module

- Parallelize sequential code using MPI and OpenMP
- Justify performance expectations for code snippets
- Sketch a typical cluster system and the execution of an application
- Characterize the scalability of a parallel app based on its performance
- Analyze parallel app performance using performance analysis tools
- Describe the development and execution models of MPI and OpenMP
- Construct small parallel apps that demonstrate the features of parallel apps
- Demonstrate the application of software engineering concepts
- Demonstrate the usage of an HPC system to load existing software packages and execute parallel apps and workflows

Organization of the Module

■ Attendees

▶ GWDG academy users

- Researchers, PhD students, users of HPC systems in the NHR and local

▶ University students

- Need to develop software after the course to obtain their credits
- Details will be explained at the end of the week

■ Webpage https://hps.vi4io.org/teaching/summer_term_2026/pchpc

▶ Links to Slides, exercise sheets, and more

■ Communication via two BBB channels

▶ Broadcast: you should listen to this one the whole week

- The trainer will present slides, walk through exercises, share suggestions
- Do not share video, note that we record this channel

▶ Breakout: room for group work and general support requests during sessions

■ For university attendees: may use StudIP for asynchronous communication

▶ We use it for announcements

▶ Please use it for any purpose around the topic!

Organization of the Module

- Block course: 1 week of training (this week)
 - ▶ Mix of lecture, hands-on tutorials, and guided exercises
 - ▶ May contain introductory and harder tasks
 - ▶ You can take a break anytime as necessary (particularly during exercises)
- Group work and community (30 min)
 - ▶ Learning in a virtual environment is difficult, therefore, we form groups!
 - ▶ Imagine you sit in a room with 4 people to share ideas and work together
 - ▶ The group *should* stick together in a breakout room the whole week
 - ▶ We will **now** organize teams of 5 attendees
 - 1 Join the Breakout BBB session
 - 2 Room 1-4 are reserved for GWDG-Academy attendees
 - 3 Room 5 are for DLR attendees
 - 4 Room 6+ are for University attendees
 - 5 Join a random room with < 5 attendees - or with peers you know
 - 6 Work on the "Welcome" groupwork (next slide)

Group Work: Welcome

■ Tasks:

- 1 Introduce yourself to your peers
- 2 Describe with one sentence why you joined this course
- 3 Have one of you share the screen of the course

■ Time: 25 min

■ Organization: breakout groups - Please use your mic and chat

Support Structure

- Support request takes place primarily in the Breakout BBB
 - ▶ This channel will never be recorded
 - ▶ Ask questions to colleagues and to us
 - ▶ We will support your learning journey but **YOU** are responsible for it
- Utilize screen sharing (similarly as we would if in the same room)
- L1: Try to resolve issues in your breakout group with your peers
 - ▶ Please use your microphone, share screen and work together (on issues)
 - ▶ It is beneficial for learning
- L2: Ask questions in the global breakout chat
 - ▶ We have trainers that will reply to you, maybe other peers will reply too!
- L3: If breakout chat doesn't help, a trainer will connect your breakout group
- If we realize that the issue should be given to all, the trainer will use the broadcast channel to demonstrate how the issue can be resolved

A Typical Session

- 1** Trainer gives an introduction to the topic
 - ▶ May include some short/small group works (for your breakout group)
- 2** Trainer may give a tutorial to overcome introductory obstacles
 - ▶ Step-by-step walkthrough
 - ▶ We provide an exercise sheet describing the steps and giving an introduction
- 3** Attendees work on tasks individually and in their breakout group
 - ▶ We provide an exercise sheet
 - ▶ Attendees should store their results (e.g. in a Git repository)
- 4** End of the session, volunteers share results on the broadcast channel

Credits

- This course can be taken via the following modules
 - ▶ Modul M.Inf.1829: Praktikum High-Performance Computing (6C)
 - ▶ Modul B.Inf.1803: Fachpraktikum I (5C)
 - ▶ Modul B.Inf.1804: Fachpraktikum II (5C)
 - ▶ Modul B.Inf.1805: Fachpraktikum III (5C)
 - ▶ Modul B.Inf.1833: Fachpraktikum Data Science (9C)
 - ▶ Modul B.Inf.1834: Fachpraktikum Data Science I (klein) (5C)
 - ▶ Modul B.Inf.1835: Fachpraktikum Data Science II (klein) (5C)
- Of these, only M.Inf.1829 is graded
 - ▶ Can be extended via M.Inf.1834 for a total of 9C
 - ▶ Grading is also stricter, see final assignment sheet for details
- Make sure to register for the one you want

Learning Outcomes

After the session, a participant should be able to:

- Characterize distributed, parallel computing and HPC
- Describe how the scientific method relies on HPC
- Sketch generic parallel/distributed system architectures
- Sketch a simple program for vector addition using pseudocode

Outline

- 1 Organization of the Module
- 2 Scientific Method**
- 3 High-Performance Computing
- 4 Distributed Computing
- 5 Parallel Computing
- 6 Programming
- 7 Conclusions

Scientific Method

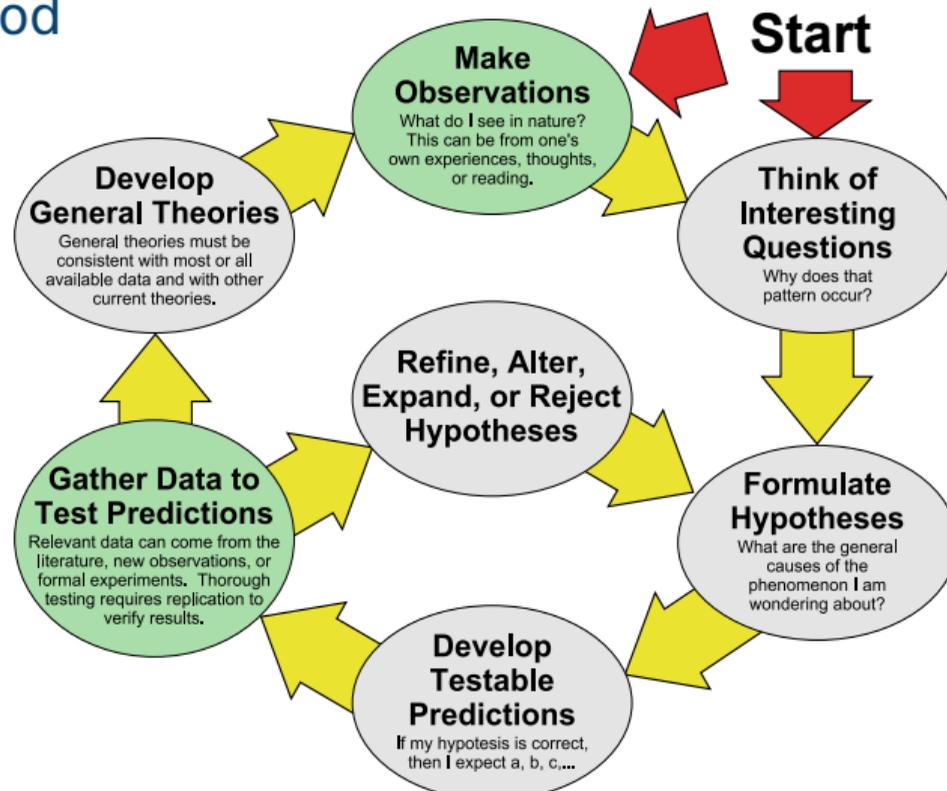
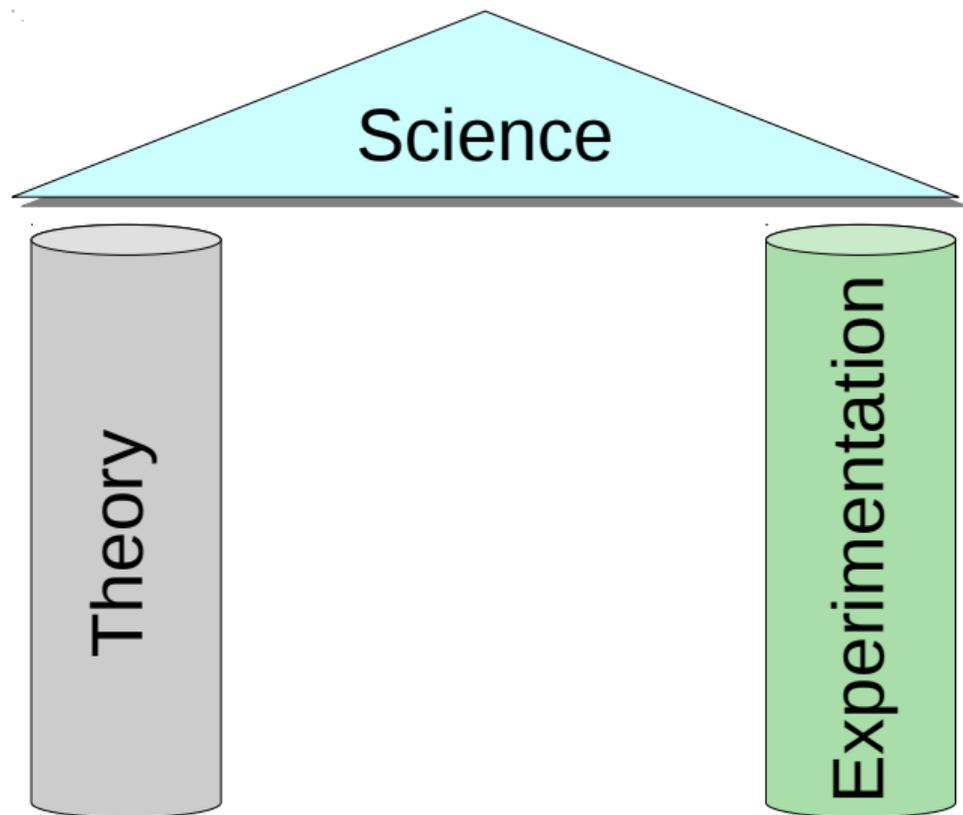
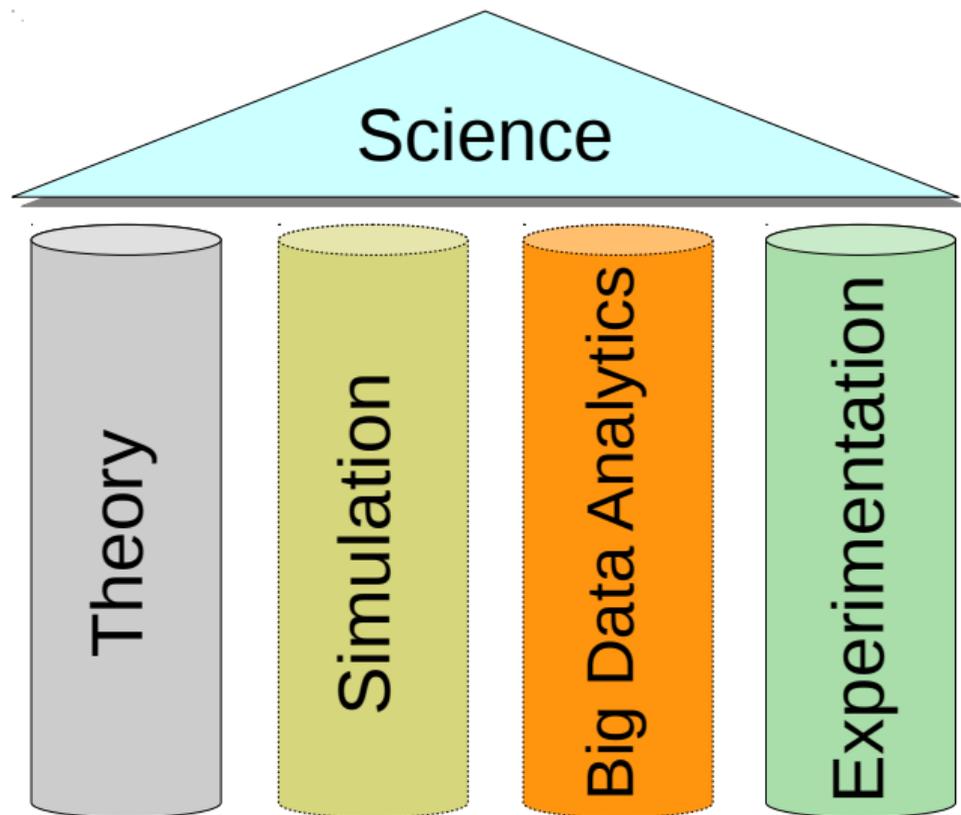


Figure: Based on “The Scientific Method as an Ongoing Process”, ArchonMagnus
https://en.wikipedia.org/wiki/Scientific_method

Pillars of the Scientific Method



Pillars of Science: **Modern Perspective**



Computer-Aided Simulation

Modelling and Simulation of the world replaces traditional experiment

Computer simulation is an instrument empowering scientists with

- Arbitrary temporal and spatial resolutions
- Manipulation of arbitrary (model) parameters
- Reproducibility
- Conducting experiments that are infeasible due to ethics, risks, or costs
 - ▶ Impact of the explosion of nuclear power plant
 - ▶ Impact of poison to humans
 - ▶ Influence of brain neurons
- Prediction of the future
 - ▶ Weather forecast, climate
 - ▶ COVID19 infection progression ...

Simulation is Compute and Memory-Intense

Examples

- Simulation of billions of neurons requires certain memory
- The Modelling of plane engines consists of billions of "elements"
- AI-Models compute with 1000s of GPUs
- Deadline of simulations
 - ▶ Weather prediction demands high resolution and completion within 24 hours.

Simulation is Compute and Memory-Intense

Examples

- Simulation of billions of neurons requires certain memory
- The Modelling of plane engines consists of billions of "elements"
- AI-Models compute with 1000s of GPUs
- Deadline of simulations
 - ▶ Weather prediction demands high resolution and completion within 24 hours.

How can we cope with the huge demand for compute/storage resources?

- A single PC/server/workstation is not able to solve compute task

Simulation is Compute and Memory-Intense

Examples

- Simulation of billions of neurons requires certain memory
- The Modelling of plane engines consists of billions of "elements"
- AI-Models compute with 1000s of GPUs
- Deadline of simulations
 - ▶ Weather prediction demands high resolution and completion within 24 hours.

How can we cope with the huge demand for compute/storage resources?

- A single PC/server/workstation is not able to solve compute task
- We need more performance ... High performance ...

High-Performance Computation

Relation of the Scientific Method to Simulation

Simulation models real systems to gain new insight

- Instrument to make observations, e.g., high-resolution and fast timescale
- Typically used to validate/refine theories, identify new phenomena
- Classical computational science: hard facts (based on models)
- The frontier of science needs massive computing resources
- Data-intensive sciences like climate impose challenges to data handling, too

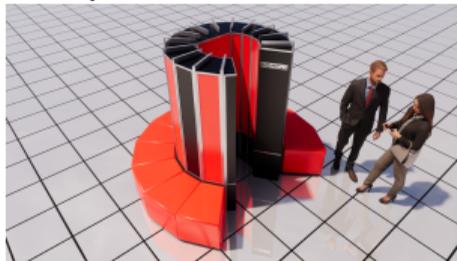
Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing**
- 4 Distributed Computing
- 5 Parallel Computing
- 6 Programming
- 7 Conclusions

High-Performance Computing

Definitions

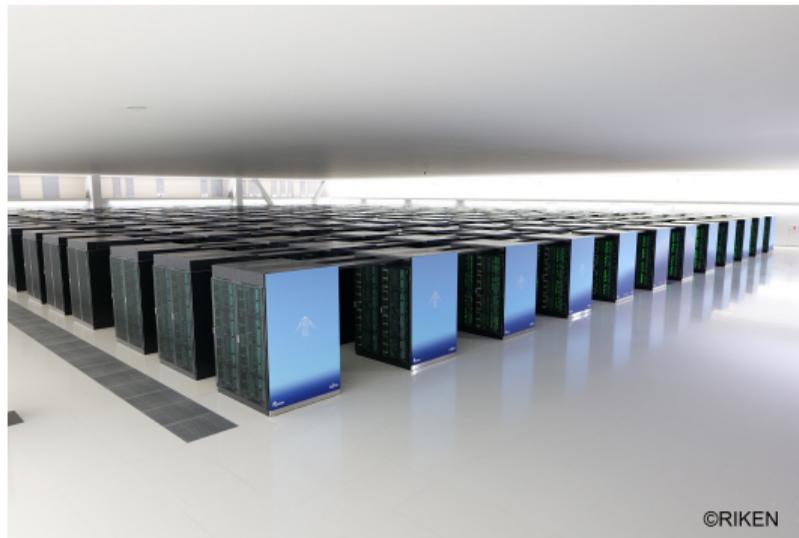
- HPC: Field providing massive compute resources for a computational task
 - ▶ Task needs too much memory or time for a normal computer
 - ⇒ Enabler of complex challenging simulations
- Supercomputer: aggregates power of many compute devices
 - ▶ In the past large monolithic computers such as the Cray
 - ▶ Nowadays: 100-1,000s of servers that are clustered together
 - ▶ Comparison: Car is to Formula-1 like Computer to Supercomputer



Introducing: One of the Fastest Supercomputers of the World

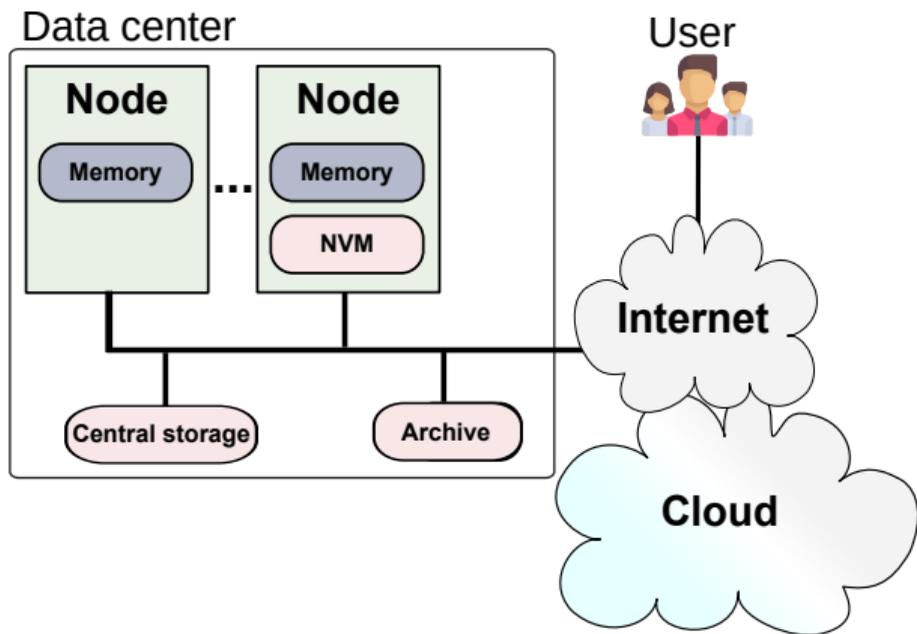
FUGAKU at RIKEN Center for Computational Science

- Nodes/Servers:
 - ▶ 158,978, 7.6 Million CPU Cores
- Compute Peak:
 - ▶ 540 Petaflop/s (10^{15})
- Memory: 5 Petabyte
- Storage: 150 Petabyte HDDs
- Energy Consumption: 30 Megawatt
- Costs: 1 Billion (program) \$



The **Top500** is a list of the most performant supercomputers

Supercomputers & Data Centers



Credits: STFC
 JASMIN Cluster at RAL / STFC
 Used for data analysis of the Centre for
 Environmental Data Analysis (CEDA)

HPC in Göttingen

GWDG: university data center and providing innovative technology solutions

- HPC systems for local scientists, German-wide and for DLR
- Integrates research for HPC systems and services



Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing
- 4 Distributed Computing**
- 5 Parallel Computing
- 6 Programming
- 7 Conclusions

Distributed Computing

Field in computer science that studies **distributed systems**¹

Definition

- System which components² are located on different networked computers
- Components communicate and coordinate actions by passing messages
- Components interact to achieve a common goal
- *Wider sense*: autonomous processes coordinated by passing messages

Characteristics

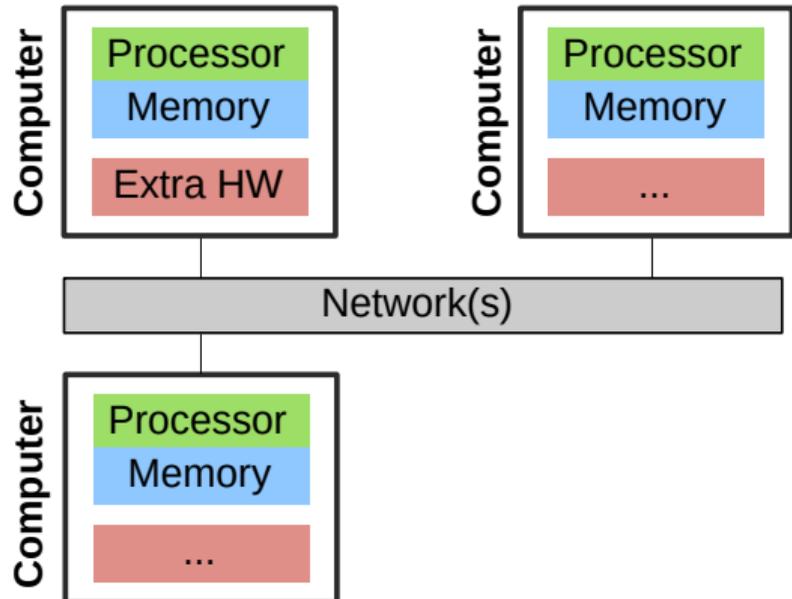
- Distributed memory: components have their own (private) memory
- Concurrency of components: different components compute at the same time
- Lack of a global clock: clocks may diverge
- Independent failure of components, e.g., due to power outage

¹https://en.wikipedia.org/wiki/Distributed_computing

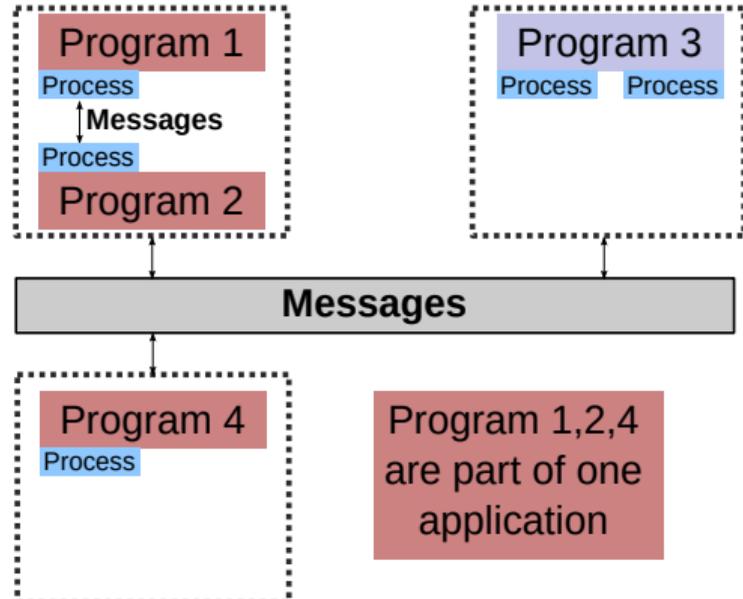
Example Distributed System and Distributed Program

- A **distributed program** (DP) runs on a distributed system
 - ▶ Processes are instances of one program running on one computer
- A **distributed application/algorithm** may involve various DPs/vendors

Hardware perspective



Software perspective (mapped to hw)



Example Distributed Applications and Algorithms

Applications

- The Internet and telecommunication networks
- Cloud computing
- Wireless sensor networks
- The Internet of Things (IoT) – “everything is connected to the Internet”

Algorithms (selection from real-world examples)

- Consensus: reliable agreement on a decision (malicious participants?)
- Leader election
- Reliable broadcast (of a message)
- Replication

Cloud Computing

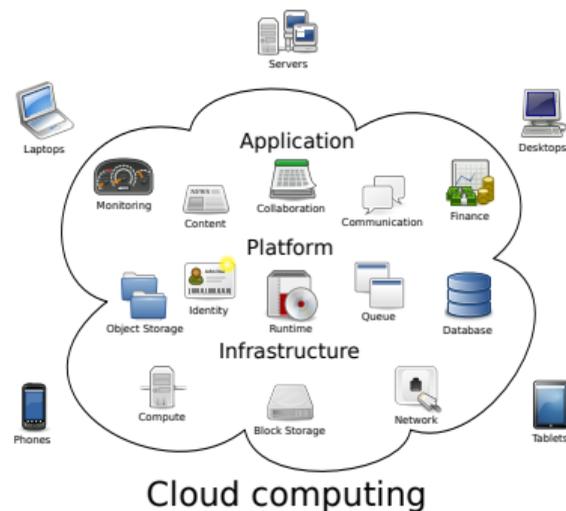
Definition

- On-demand availability of computer system resources (data storage and computing)
 - ▶ Without direct active user management
- Relates to distributed resources
 - ▶ provided by data centers to users over the Internet
- Fog/Edge Computing: brings cloud close to user

Examples

- Applications: Dropbox, Google Mail, Office 365
- Infrastructure: Amazon, Google, Microsoft, Oracle

Image source: Frank, B. Wilson - CloudNINE, https://en.wikipedia.org/wiki/Cloud_computing



Challenges using Distributed Systems

- Programming: concurrency introduces new types of programming mistakes
 - ▶ It is difficult to think about all cases of concurrency
 - ▶ Must coordinate between programs
 - ▶ No global view and debugging
- Resource sharing: system shares resources between all users
- Scalability: the system must be able to grow with the requirements
 - ▶ numbers of users/data volume/compute demand
 - ▶ retain performance level (response time)
 - ▶ requires to add hardware, though
- Fault handling: detect, mask, and recover from failures
 - ▶ Failures are inevitable and the normal mode of operation
- Heterogeneity: system consists of different hardware/software
- Transparency: Users do not care about how/where code/data is
- Security: Availability of services, confidentiality of data

Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing
- 4 Distributed Computing
- 5 Parallel Computing**
- 6 Programming
- 7 Conclusions

Definition: Parallel Computing

Many calculations **or** the execution of processes are carried out simultaneously³

Characteristics

- Goal is to improve performance for an application
 - ▶ Either allowing to solve problems within a deadline or increased accuracy
- Application/System must coordinate independent parallel processing
 - ▶ There are various programming models for parallel applications
- Different architectures speed up computation: **may use** distributed systems

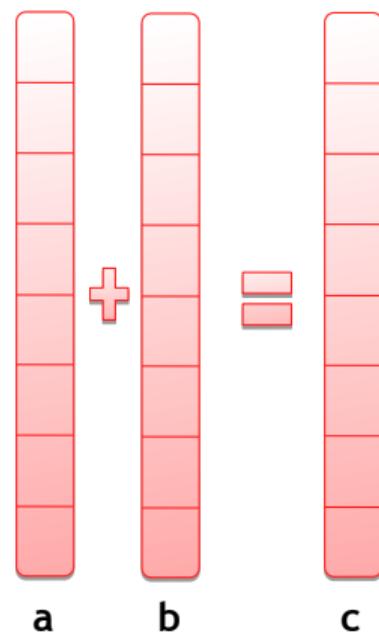
Levels of parallelism (from hardware perspective)

- Bit-level: process multiple bits concurrently (e.g., in an ALU)
- Instruction-level: process multiple instructions concurrently on a CPU
- Data: run the same computation on **different data**
- Task: run **different** computations concurrently

³https://en.wikipedia.org/wiki/Parallel_computing

Bit-Level Parallelism: Vector Parallelism with SIMD

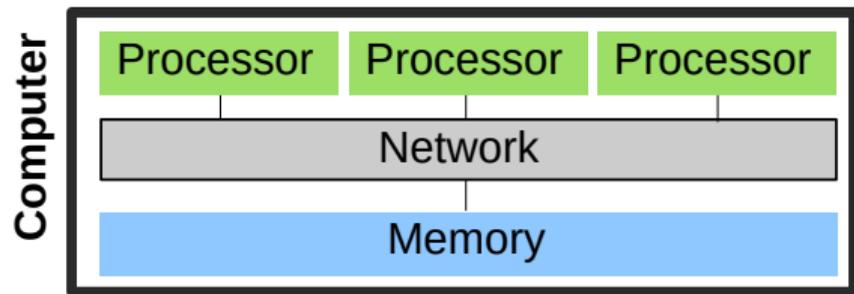
- SIMD = Single instruction multiple data
 - ▶ Apply the same operation on multiple data
- Example: Vector addition: $a = b + c$
 - ▶ $c_i = a_i + b_i$ for all vector elements i
- AVX-512 works on 8x 64-bit elements in parallel
 - ▶ i.e., run same operation on all
 - ▶ The example Xeon can do 8xFP64 FMA ($a = a + (b \cdot c)$) per cycle



Parallel Architectures

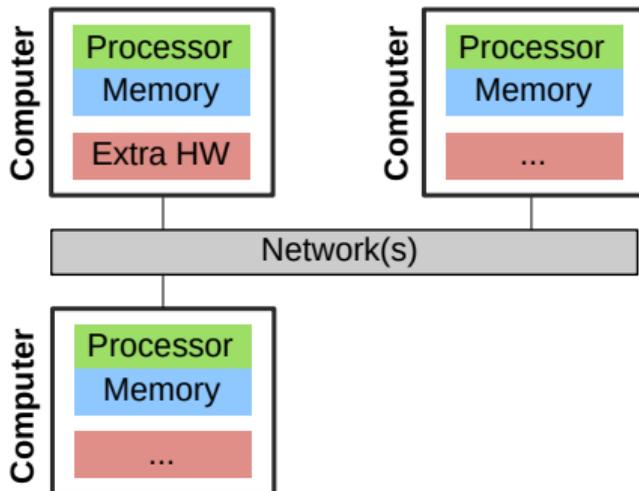
In practice, systems are a mix of two paradigms:

Shared memory



- Processors access joint memory
- Enabled communication/coordination
- Cannot be scaled up to any size
- Expensive to build one big system
- Programming with **OpenMP**

Distributed memory systems (again!)



- Processor only sees own memory
- Performance of the network is key
- Programming with **Message Passing**

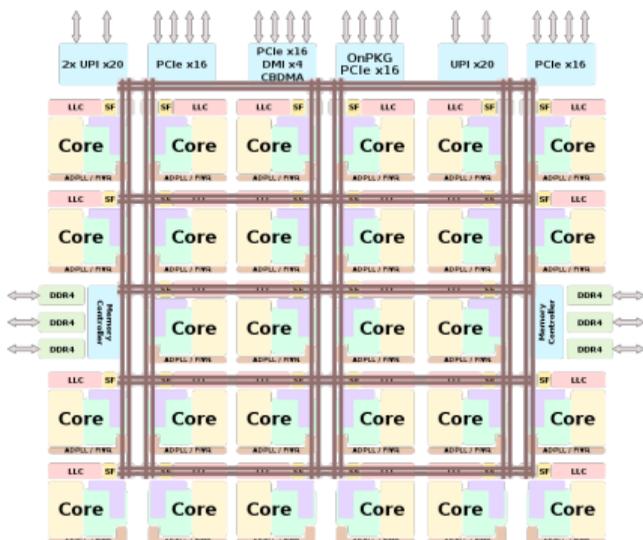
Multicore CPU: Xeon Platinum 8280M Cascade Lake-SP

Performance

- FLOPs: $32 \cdot \text{frequency} \cdot \text{cores}$
 - ▶ 28 cores, 2.7 GHz (1.8 GHz AVX512)
 - ⇒ 2.2 TFLOPs
- 6 Channel DDR4, max 2.933 GHz
 - ▶ Throughput 131 GB/s
- Power: 205 Watt

Architecture

- Each core executes code independently
 - ▶ Feature rich: speculative execution, ...
- Each core has two AVX-512 units
 - ▶ Vector parallelism on 512 bits



- Summary: complex architecture, heavy cores, optimized for latency

Manycore GPU: NVIDIA A100

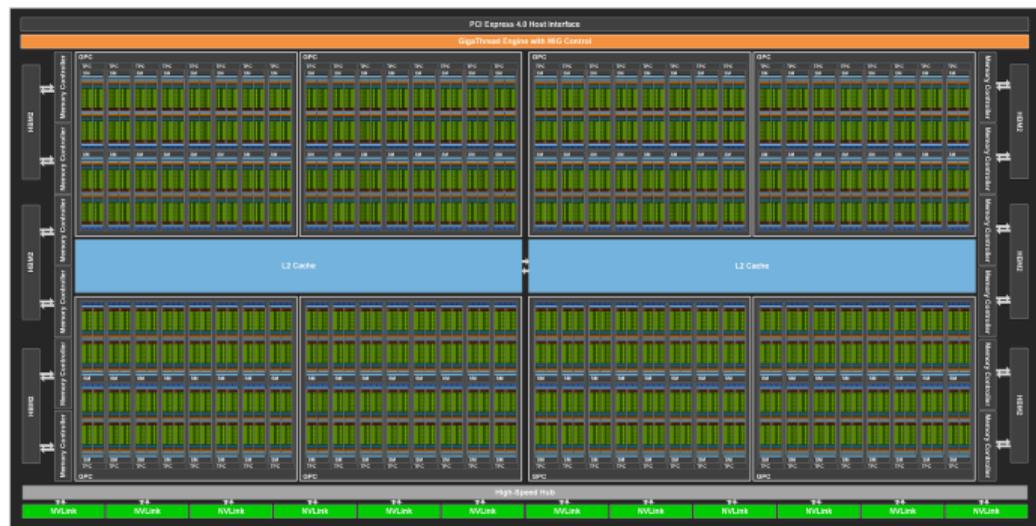
Accelerated computing is outside of this course, but concepts are transferrable

Performance

- FLOPs: 9.7 TFLOPs FP64
 - ▶ 312 TFLOPs Tensor (FP16)
 - ▶ 1.41 GHz
- 40 GByte HBM2 memory
 - ▶ 10 memory channels
 - ▶ Throughput 1600 GB/sec
- Power: 400 Watt

Architecture

- 128 Streaming multiprocessors
 - ▶ Each with 32 FP64 cores
 - ⇒ 4096 cores per GPU
- Summary: Simple cores, optimized for throughput
- Problem: deep pipeline, higher latency, costly startup time of program

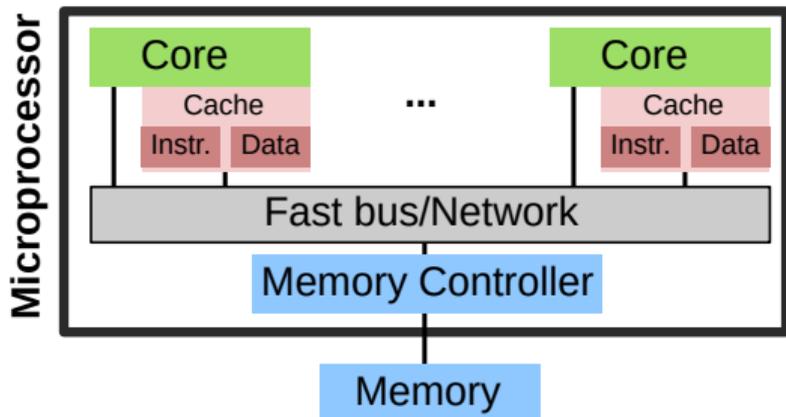


Parallel Programs

- A **parallel program** runs on parallel hardware

In the strict sense: A parallel application coordinates concurrent processing

Schema of a multicore processor



Processor provides all parallelism levels

- Multiple ALU/other units
- Pipelining of processing stages
- SIMD: Single Instruction - Multiple Data
 - ▶ Same operation on multiple data
 - ▶ Instruction set: SSE, AVX
- Multiple cores
 - ▶ Each with own instruction pointer
- May use (GPU) accelerators
 - ▶ CPU in charge of processing

<https://en.wikipedia.org/wiki/Microarchitecture>

Group Work

- Think about an application of parallel computation
 - ▶ Describe the use case briefly
- What computation is performed in parallel?
- Which architecture / hardware presented would you like to use for it?
- Time: 5 min
- Organization: breakout groups - please use your mic or chat

Challenges

- Programming: imports errors from distributed computed +
 - ▶ Low-level APIs and code-optimization to achieve performance
 - ▶ Performance-optimized code is difficult to maintain
 - ▶ Expensive and challenging to debug 1'000 concurrently running processes
 - ▶ Utilizing all compute resources efficiently (load balancing)
 - ▶ Grand challenges are difficult to test, as nobody knows the true answer
- Performance engineering: Optimizing code is main agenda for HPC
 - ▶ Covered in this course
- Scalability: stricter than distributed systems
 - ▶ Strong-scaling: same problem, more parallelism shall improve performance
 - ▶ Weak-scaling: data scales with processors, retain time-to-solution
- Environment: bleeding edge and varying hardware/software systems
 - ▶ Special-purpose hardware (FPGA/ASIC Application-Specific Integrated Circuit)
 - ▶ Limited knowledge to administrate, use, and to compare performance

Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing
- 4 Distributed Computing
- 5 Parallel Computing
- 6 Programming**
- 7 Conclusions

Programming

- Let's investigate how to create a “parallel” program

Abstractions and examples

- Sequential code to compute vector addition
- Automatically parallelizable code for shared memory using OpenMP
 - ▶ Parallelizes code based on user-provided directives
- Manual parallelization for distributed memory using Message passing

Vector Addition: Sequential CPU Code

Compute function

```
1 void vecAdd(int * restrict a, int * restrict b, int * restrict c, int n){
2     for(int i=0; i < n; i++){
3         c[i] = a[i] + b[i];
4     }
5 }
```

Execution

```
1 int a[8];
2 int b[8];
3 int c[8];
4 // fill a and b somehow
5 vecAdd(a, b, c, 8);
```

- Both codes may be placed in the same file \Rightarrow we call this a "single source"

Directive-Based Parallelism using OpenMP: CPU Code

Compute function

```
1 void vecAdd(int * restrict a, int * restrict b, int * restrict c, int n){
2     //Preprocessor directive telling compiler to parallelize for loops
3     #pragma OMP parallel for
4     for(int i=0; i < n; i++){
5         c[i] = a[i] + b[i];
6     }
7 }
```

■ The same code as before, just compile with `-fopenmp...`

Execution

```
1 int a[8];
2 int b[8];
3 int c[8];
4 // fill a and b with values ...
5 vecAdd(a, b, c, 8);
```

Message Passing

Definition

- Message passing is the sending of a message to a process⁴
- What: any data from the memory of the sender
- How: Programmer explicitly requests send/recv

Content of a message

- Header (Sender, receiver, type⁵)
- Data (from memory)



Addressing

- How to define to whom I sent, from whom to receive?
 - ▶ Addressing via "process number": Rank 0 - (N-1)
 - ▶ Processes are enumerated upon start

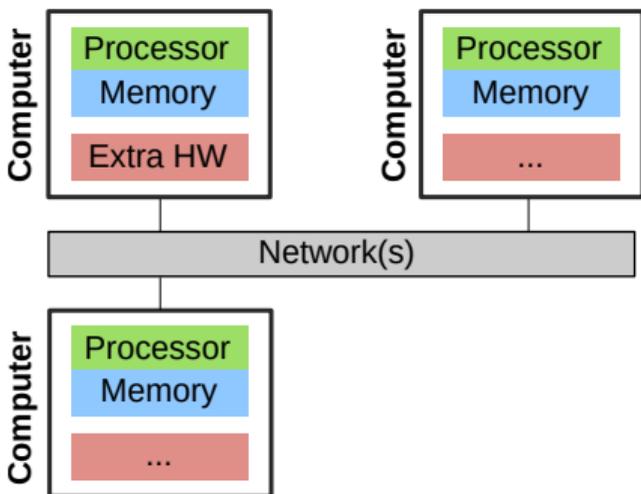
⁴The general definition in distributed systems is more generic

⁵Distinguishes different messages

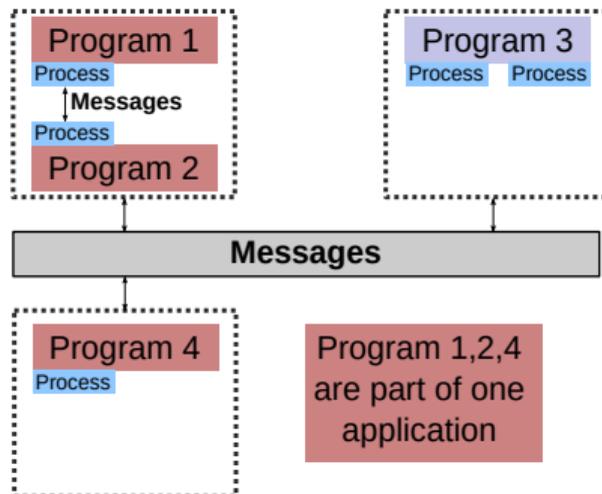
Example Execution of an Message Processing Program

- Processes are instances of an application
 - ▶ Executed on different computers
 - ▶ May execute the same or different code
 - ▶ Addressing via enumeration of the processes
- Different applications can be executed concurrently

Hardware perspective



Software perspective



Programming with Message Passing

- Code of processes of the program define how they cooperate
- Important standard: The Message Passing Interface (MPI)
 - ▶ MPI implementations are a library with communication functions

Single Program Multiple Data (SPMD)

- SPMD: A single binary program created from one source code
- Every process of a program runs on different data

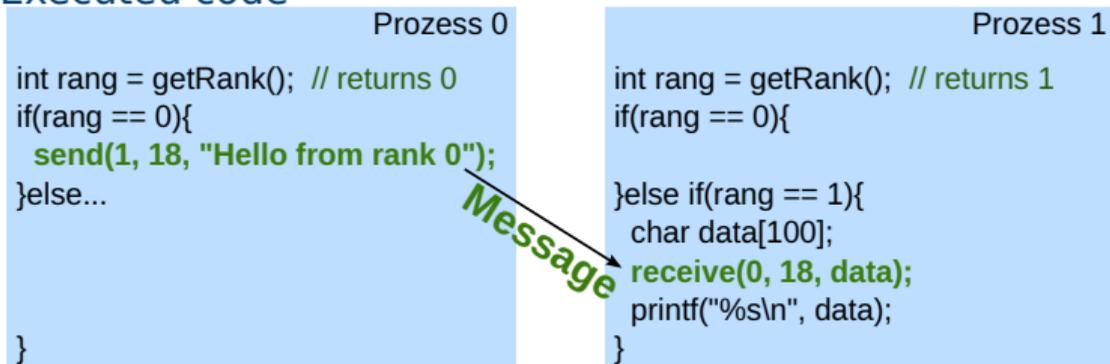
Example message passing

```
1  int Rank = getRank(); // Determine my rank
2  if(Rank == 0){
3      // Send message (18 bytes to Rank 1)
4      send(1, 18, "Hello from rank 0");
5  }else if(Rank == 1){
6      char data[100];
7      // Receive message from Rank 0
8      receive(0, 18, data);
9      printf("%s\n", data);
10 }
```

Concurrent Execution

- Assumption: our example program is executed with two processes
 - ▶ Instructions of both processes are executed concurrently and independently

Executed code



- Semantics of message exchange is defined by operation/function
 - ▶ Receive must block until a suitable message is received
 - ▶ Sending might complete before message is actually received/processed
- Program code is **parallelizable** if any parallel and concurrent execution path leads to the **same solution**

Outline

- 1 Organization of the Module
- 2 Scientific Method
- 3 High-Performance Computing
- 4 Distributed Computing
- 5 Parallel Computing
- 6 Programming
- 7 Conclusions**

Computational Science

- Talking about computer-aided simulation, we mean computational science

Definitions

- Multidisciplinary field using advanced computing capabilities to understand and solve complex problems
 - ▶ Typically using mathematical models and computer simulation
 - ▶ Problems are motivated by industrial or societal challenges
- May utilize single computer, distributed systems, or supercomputers

Examples utilizing distributed computing

- Finding the higgs boson (CERN)
- Bioinformatics applications, e.g., gene sequencing

Examples utilizing high-performance computing

- Computing the weather forecast for tomorrow / next week
- Simulating a tokamak fusion reactor

https://en.wikipedia.org/wiki/Computational_science

Summary

- HPC and supercomputers are enablers for scientific computing
- Supercomputers are relevant for data science
- Parallel computing is the simultaneous calculation/execution
- Shared-memory, distributed-memory and GPU-Architectures differ
- GPUs are accelerating CPUs for massively parallel workloads
- Programming can be challenging
- Programming paradigms
 - ▶ Auto-parallelization with compiler-directives (OpenMP, shared mem)
 - ▶ Parallelization with Message Passing (distributed computing)
- Simple example: Vector addition