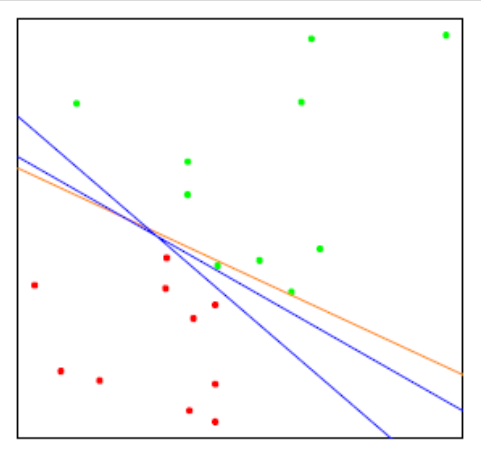


Parallel & Scalable Machine Learning

Introduction to Machine Learning Algorithms

Dr. –Ing. Gabriele Cavallaro

Postdoctoral Researcher

High Productivity Data Processing Group

Juelich Supercomputing Centre

Lecture 11 - 19/02/2020

CLUSTERING – WITH HPC

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

PRACE PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE TRAINING PORTAL

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

DEEP Projects

COURSE OUTLINE

- Parallel and Scalable Machine Learning Driven by HPC
- Introduction to Machine Learning Fundamentals
- Supervised Learning with a Simple Learning Model
- Artificial Neural Networks (ANNs)
- Introduction to Statistical Learning Theory
- Validation and Regularization
- Pattern Recognition Systems
- Parallel and Distributed Training of ANN
- Supervised Learning with Deep Learning
- Unsupervised Learning – Clustering
- Clustering with HPC
- Introduction to Deep Reinforcement Learning

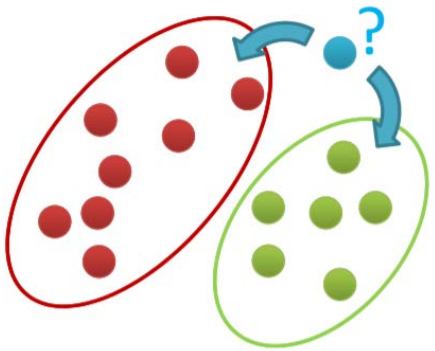
OUTLINE

- The Free Lunch is Over
  - Moore's Law
  - Work Harder and work smarter
  - Get Help: Many-Core Era
- Hardware Levels of Parallelism
  - In-core, In-Processor, Single and Multiple Computers
  - Graphics Processing Units (GPUS)
- Shared-Memory Systems
- Distribute-Memory Systems
  - MPI
- HPDBSCAN

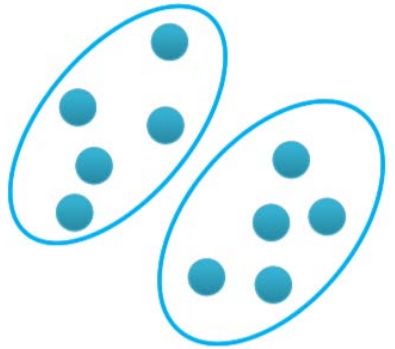
MACHINE LEARNING

Form of Learning

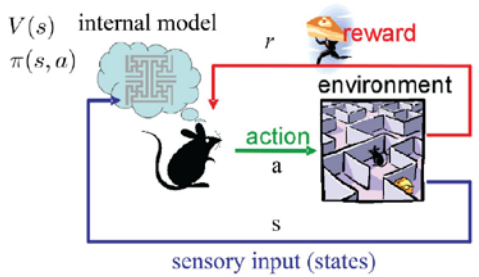
- **Supervised learning:** correct responses for input data are given
  - "teacher" signal, correct "outcomes", "labels" for the data
  - Classic frameworks: **classification**, **regression**



- **Unsupervised learning:** only data are given
  - Find "hidden" structure, patterns
  - Classical frameworks: **clustering**, **dimensionality reduction**
    THIS LECTURE



- **Reinforcement learning:** data including (sparse) **reward** $\mathrm{r}(\boldsymbol{X})$
  - Discover actions $\boldsymbol{a}$ that minimize total future reward R
  - **Active** learning: experience depends on choice of $\boldsymbol{a}$



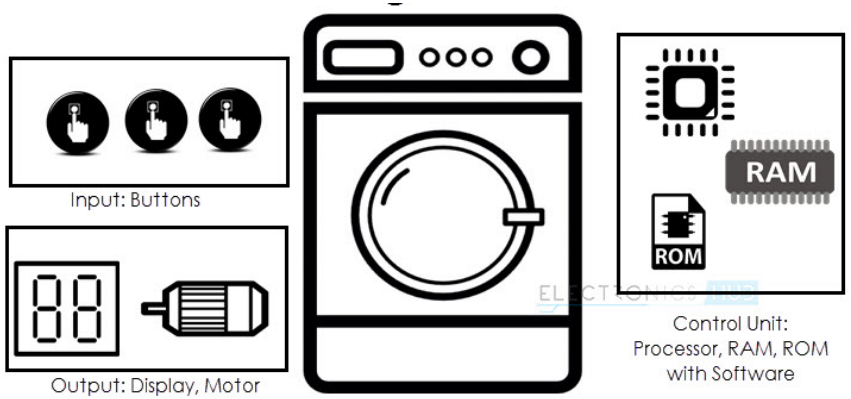
COMPUTER MARKET

IT Environments

- **Embedded system**
  - System that you **don't use directly** (not a computer by itself, with a display and input devices)
  - Combination of a computer processor, memory, and input/output peripheral devices
  - Has a **dedicated function** within a larger mechanical or electrical system (e.g., washing machine)

- **Embedded Computing:** real time behavior
  - The system **guarantees to end** at a defined point (task is done in a specific amount of time)
  - Not interested in fast computing
  - **Power consumption** and **price** as major issue

[1] Embedded system



[2] Embedded system example

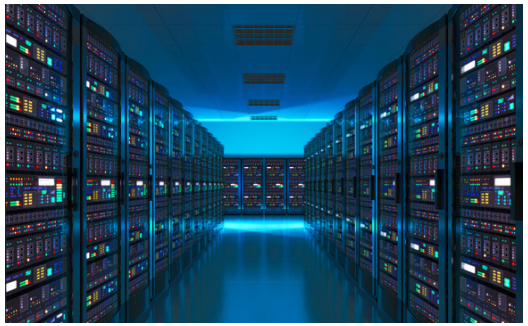
COMPUTER MARKET

IT Environments

- Desktop Computing
  - Home computers
  - **Performance / price** ration as major issue

- Servers
  - The key: maximum **performance** and **maintainability reliability**
  - Business service provisioning and major goals
  - Web servers, banking back-end, order processing, …

[3] Desktop Computer



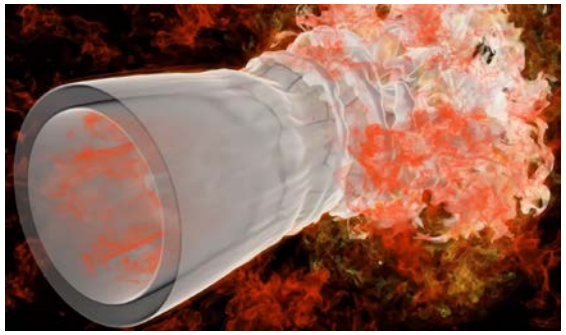
[4] Microsoft opens UK data centres

APPLICATIONS

Some Problems always benefit from faster processing

- Simulation and modeling (climate, earthquakes, airplane design, car design, vehicle traffic patterns, …)
- **Machine (deep) learning with big data**
- Web search, social networks
- Modern computer games
- Next-generation medicine
  - DNA sequencing, simulation of drug effects
- Business data processing
- Graphic effects on consumer devices

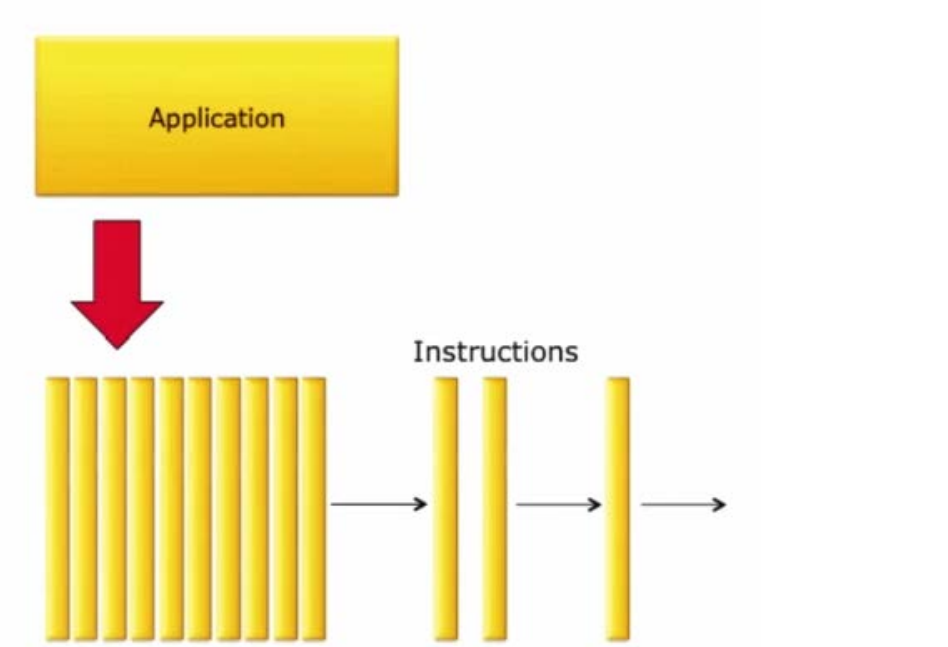
[5] HPC and Supercomputing



RUNNING APPLICATIONS

How can you become faster?

- Split the application into **instructions** (generated by the compiler)
  - These instructions are then given to the system
  - We want to **execute** these instructions **as fast as possible**



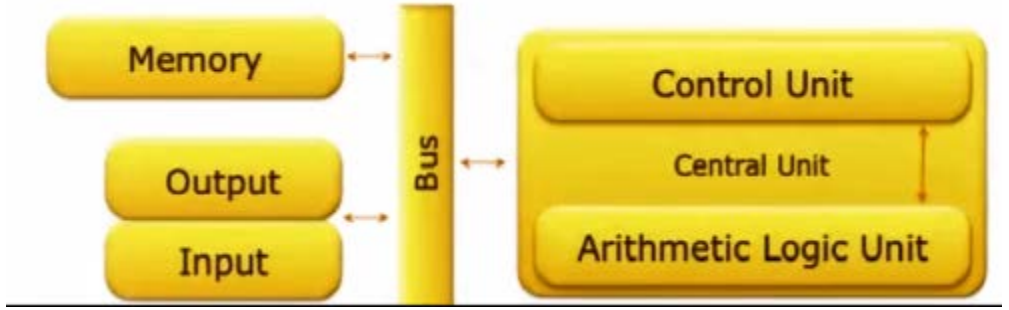
[6] WT 2013/14

MACHINE MODEL

Von Neumann Architecture

- First computers had **fixed programs** (electronic calculator)
- **Von Neumann Architecture** (1945, for EDVAC project)
  - Radical idea: load code into memory (i.e., **code becomes data**)
  - Use one machine for doing several things (**general purpose computer**)
- Von Neumann bottleneck (Bus): **access** the memory at least **3 times**
  - (1) Read the instruction, (2) read the input data and compute, (3) write the result

[6] WT 2013/14



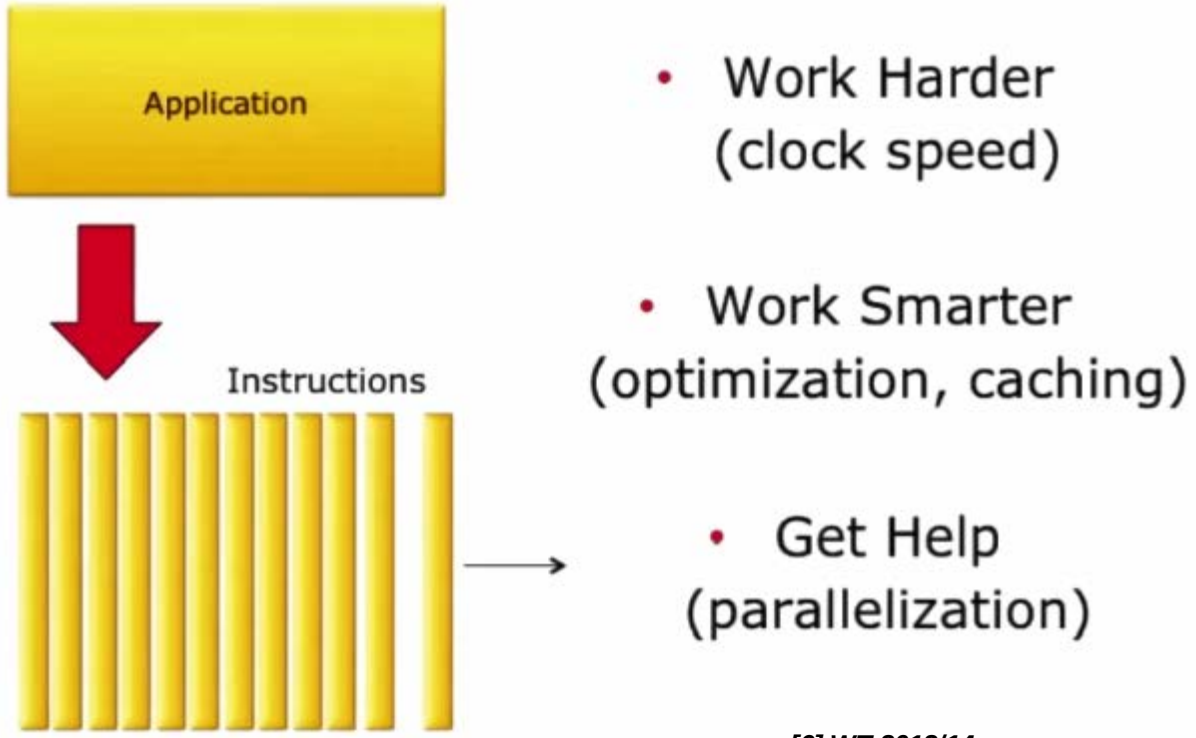
- Instruction set for control flows stored in memory
- Program is treated as data, which allows the exchange of code during runtime and self modification

  - Today machines rely on **optimizations tricks** (caching, buffering etc..) to deal with this problem
  - However this **issue** is getting again relevant today with **multi-core** and **many-core systems**

HOW TO DO ANYTHING FASTER

Three Ways

- E.g., Washing your car, doing homework, processing instructions in a computer
- In **Hardware terminology:**



[6] WT 2013/14

Increase the **frequency speed** of the hardware

Optimize the computing time per instruction (Instruction Level Parallelism (ILP), code optimization, caching)

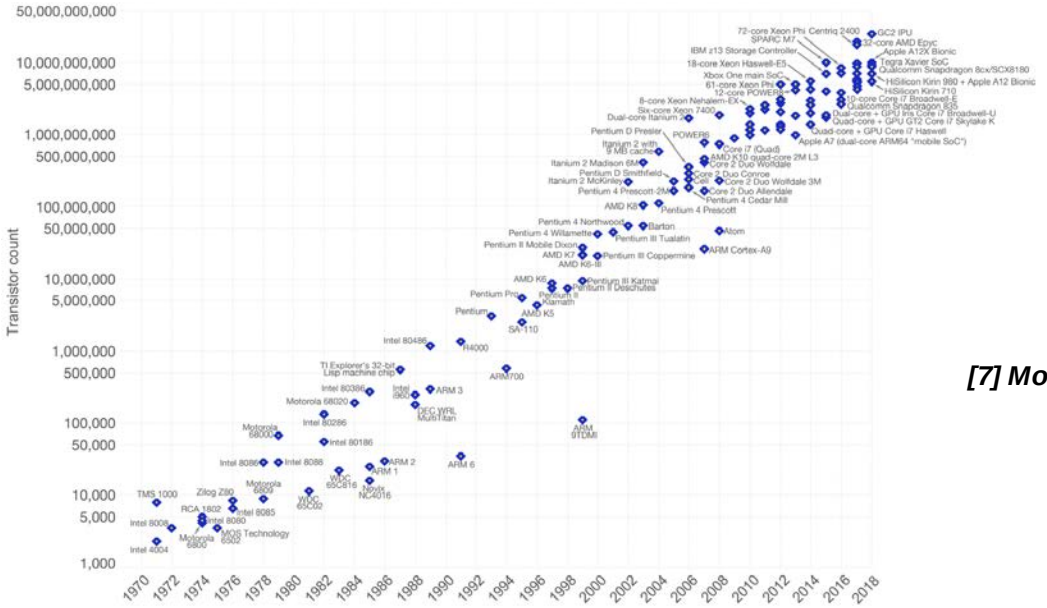
Main solution: from 1950 to 2000

Use more than one process unit (option that was never really exploited until lately) The first two options were good enough

Nowadays

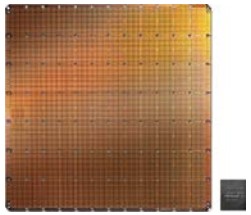
MOORE'S LAW

- "… the **number of transistors** that can be inexpensively **placed on integrated circuits** is increasing exponentially, **doubling** approximately **every two years**. …" (Gordon Moore, 1965)
  - Rule of exponential growth
  - Applied to many IT hardware developments
  - Sometimes misinterpreted as performance indication
  - Meanwhile a self-fulfilling prophecy
  - May still hold for the next 10-20 year



[7] Moore's law

- It is only about the **number of transistors** that can be places **over a die**
  - Die: small block of semiconducting material on which a given functional circuit is fabricated
  - Transistors have reached the size of few nanometers



E.g., 1.2 trillion transistors on a die over a area of 46,000mm^2

[8] Cerebras Wafer Scale Engine

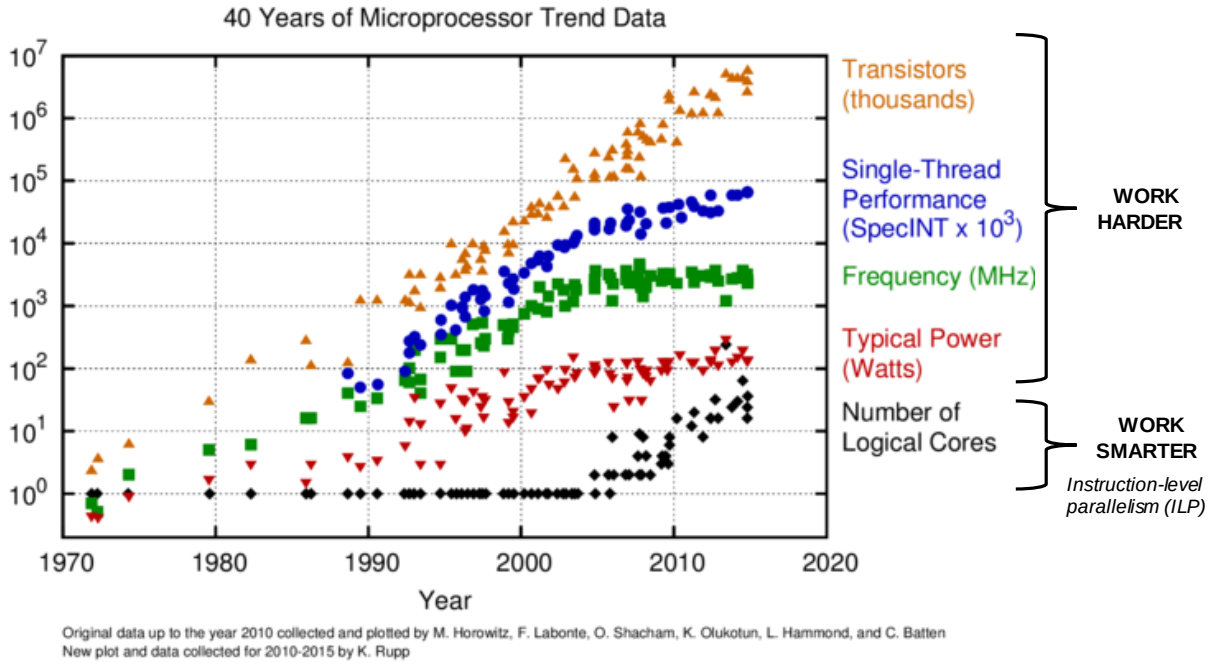
THE INDUSTRY PROBLEM

Processor Speed Development

- Companies have been shrinking the technology to try to **follow Moore's Law**
  - By doubling the density of semiconductor, they were getting performance and power improvements

- After a while the improvements started plateauing
  - Silicon clock **frequency** is flattening out
  - **Power** is flattening out with silicon
  - **Performance** is flattening out with silicon

- Investment costs to keep up with performance demands
  - Increasingly high
  - Only a few fab vendors can keep up

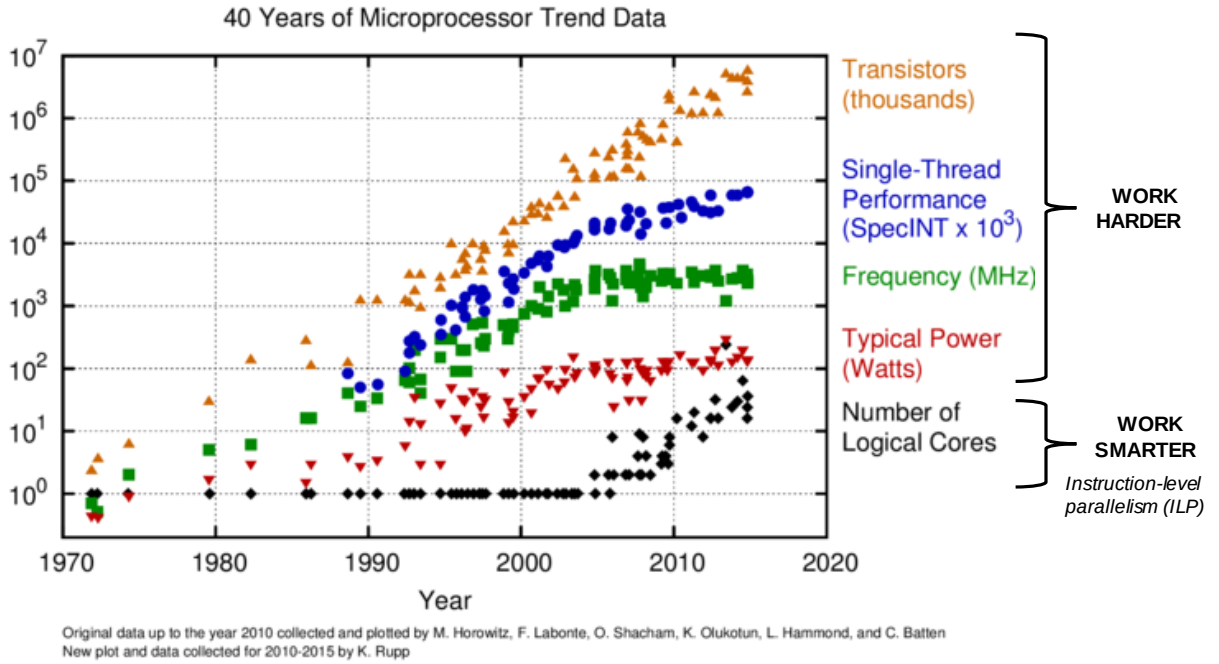


[9] Herb Sutter

THE FREE LUNCH IS OVER

Paradigm Shift

- Clock speed curve flattened in 2003 (constraint in physics)
  - **Power Wall**
  - Instruction Level Parallelism **(ILP) wall**
  - **Memory wall**



[9] Herb Sutter

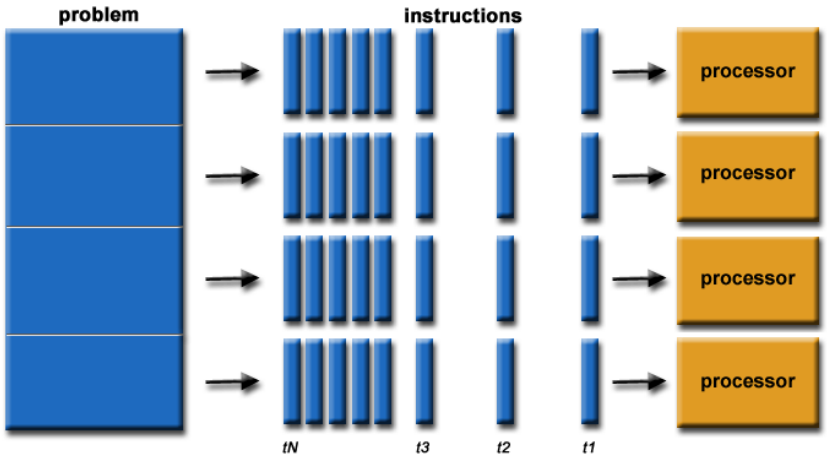
- Speeding up the serial instruction execution through clock speed improvements no longer works
- We stumbled into the **Many-Core Era**
  - Use the additional transistors to **build more cores**
  - Get more performance (**speed up**) with many core
  - The task of reaching better performance is a responsibility of **software developers**

PARALLEL COMPUTING

Concepts

- Simultaneous use of **multiple compute resources** to solve a computational problem:
  - Break the problem into **discrete parts** that can be **solved concurrently**
  - Each part is further broken down to a **series of instructions**
  - Instructions from each part **execute simultaneously** on different processors

An overall control/coordination mechanism is employed



‘**compute elements**’ (e.g. cores) solve a problem in a **cooperative way**

[10] Introduction to Parallel Computing

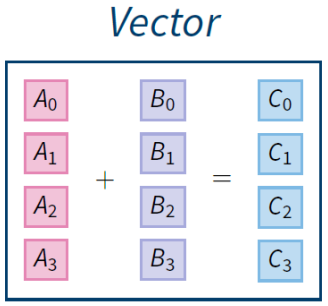
- The problem should be solved in less time than with a single compute resource

HARDWARE LEVELS OF PARALLELISM

In-core parallelism

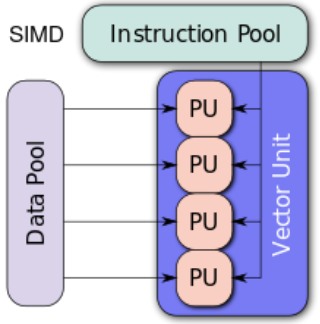
- **Single Instruction Multiple Data (SIMD)**
  - **Each CPU core** has its own independent **SIMD execution units**
  - Exploit **data level parallelism**, but **not concurrency**
  - Help to improve the performance of multimedia use
    - E.g., For same operation to every pixel of an image

Elements of vectors are processed in parallel



Parallel execution units into each CPU core

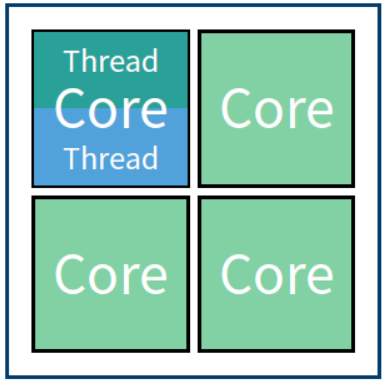
simultaneous (parallel) computations, but only a single process (instruction) at a given moment



Register sizes (e.g. 64, 128, 256 and 512 bits)

[11] Andreas Herten

- **Simultaneous Multithreading (SMT)**
  - Process of a CPU splitting each of its physical cores into **virtual cores (threads)**
  - Multiple **threads with multiple tasks** are executed simultaneously on one CPU core
  - Although running on the same core, they are **completely separated** from each other
  - Similar in concept to preemptive multitasking but is implemented at the thread level
  - Intel branded this process as hyper-threading



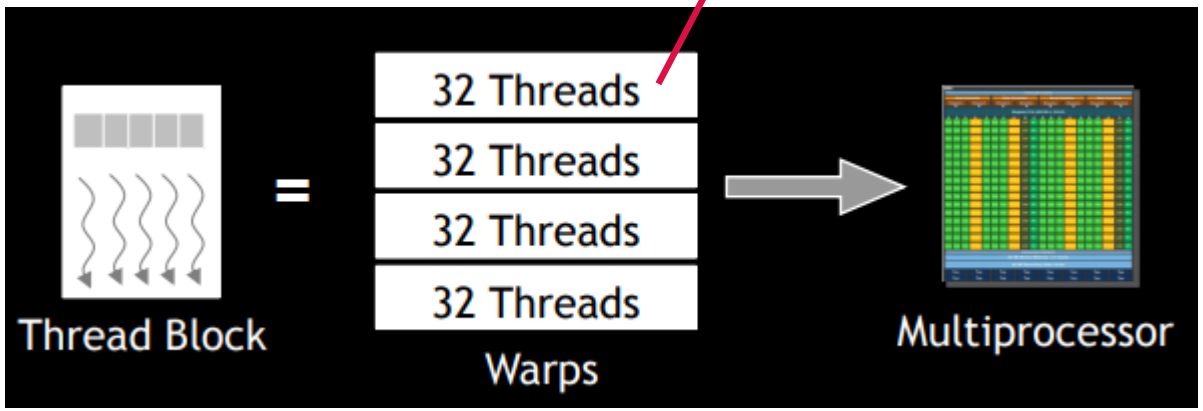
HARDWARE LEVELS OF PARALLELISM

In-processor parallelism

- **Single Instruction Multiple Threads (SIMT= SIMD + SMT)**
  - SIMD is combined with multithreading (introduced by **Nvidia**)
  - The hardware groups **threads** that execute the same instruction into **warps**
  - Several warps constitute a **thread block**

WARP
(working unit)

- Executed physically in parallel on multiprocessor
- Its threads issue instructions in lock-step (as with SIMD)



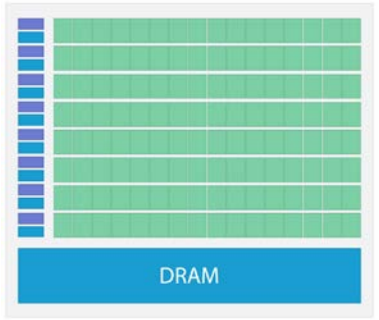
[12] Peter Messmer

GRAPHICS PROCESSING UNITS (GPUS)

Processors

- Use of **many simple cores** executing threads rather slowly
  - **High throughput computing**-oriented architecture
  - Use massive parallelism by executing a lot of **concurrent threads**
  - Handle an ever increasing amount of **multiple instruction threads**

- Great for **data and task parallelism**
  - Applications that use vector/matrix multiplication (e.g., deep learning algorithms)



[11] Andreas Herten

CPU VS. GPU

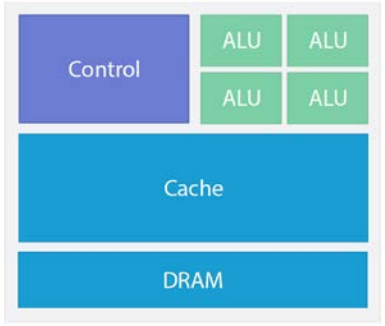
Overview

- A matter of specialities

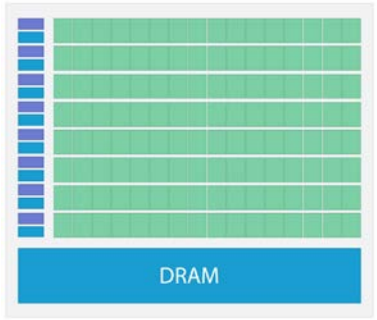
TRANSPORTING ONE

TRANSPORTING MANY

- Chip



CPU



GPU

[11] Andreas Herten

HARDWARE LEVELS OF PARALLELISM

Single Computer and Multiple Computers

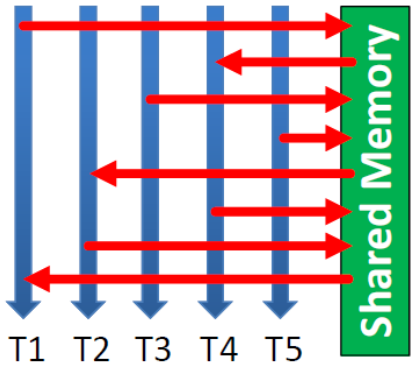
- Exposed to the **programmer** in the form of **multi-core** systems

(A) Single-machine (**shared memory**) (B) Multi-machine (**distributed memory**)

Parallel programming :'**Programming models**'

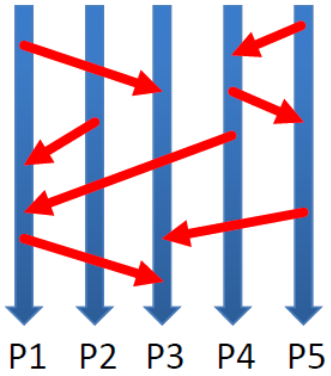
OpenMP **Message Passage Interface (MPI)**

Multiple threads (the programmer work on the **parallelism**, leaving the data shuffling to the hardware system



[13] OpenMP API Specification

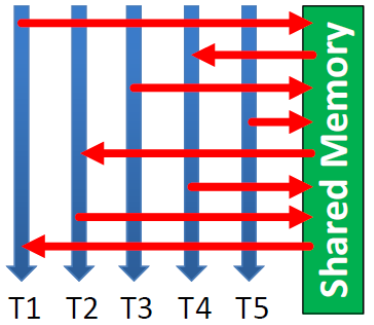
Multiple processes (forces the programmer to consider the **distribution of the data** as a first-class concern)



[14] MPI Standard

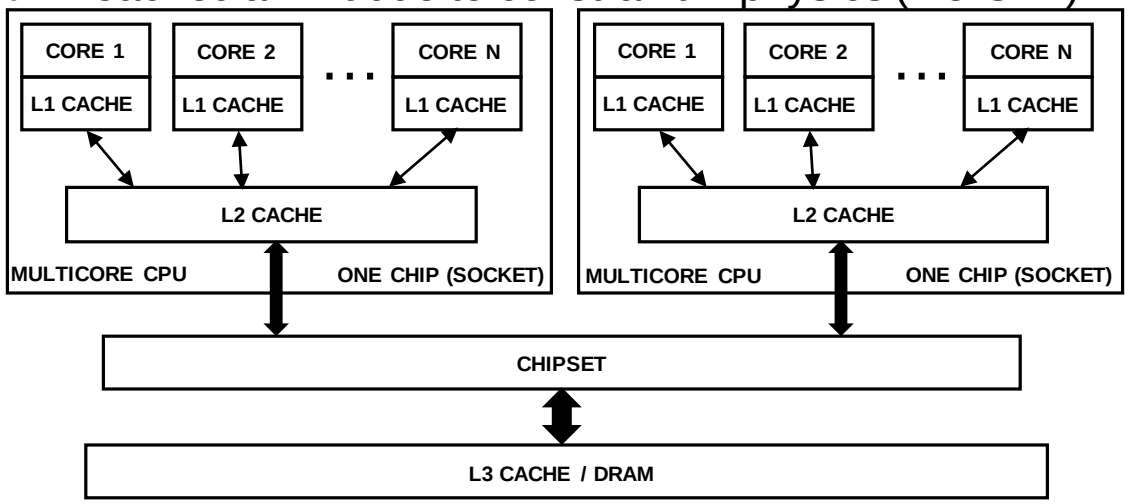
SHARED MEMORY SYSTEM

Single Computer



[13] OpenMP API Specification

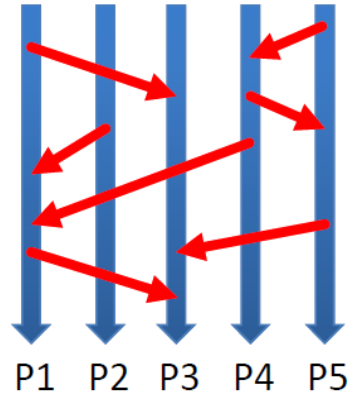
- System where a number of CPUs work on a common, **shared physical address space**
- Programming using **OpenMP** (set of compiler directives to **'mark parallel regions**')
  - Immediate access to all data by all processors without explicit communication
- Significant advances in CPU (or microprocessor chips)
  - **Multi-core CPU** chips have quad, six, or n processing cores on one chip
  - Clock-rate for single processors increased from 10 MHz (Intel 286) to 4 GHz (Pentium 4) in 30 years
    - Reached a limit due to constraint in physics (~ 5 GHz)



Hierarchy of caches (on/off chip)
L1 cache is private to each core; on-chip
L2 cache is shared; on-chip
L3 cache or Dynamic random access memory (DRAM); off-chip

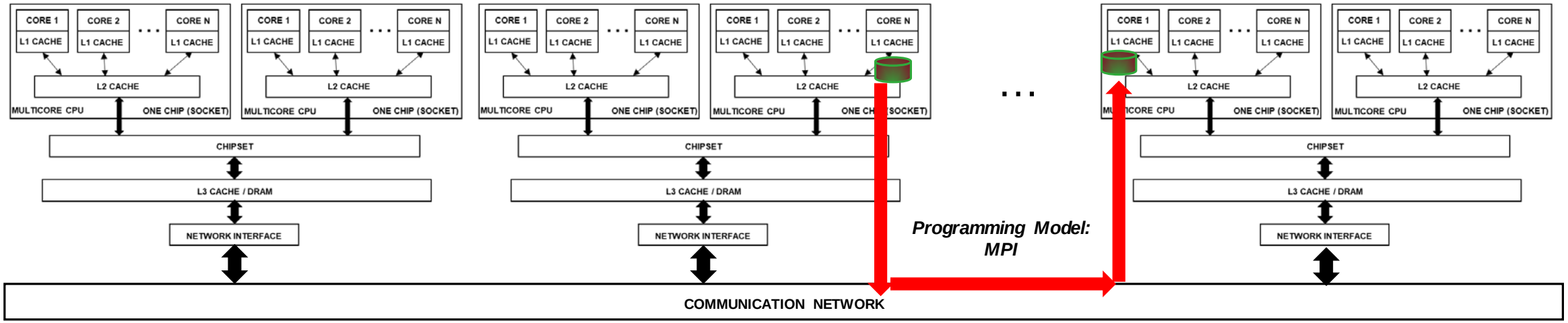
DISTRIBUTED-MEMORY SYSTEM

Multiple Computers



[14] MPI Standard

- **No remote memory access** on distributed-memory systems
  - Process cannot access another process' memory directly
- Enables **explicit message passing** as communication between processors
  - Require to '**send messages**' back and forth between processes
- Programming is tedious & complicated, but most flexible method



- Processors communicate via Network Interfaces (NI)
  - NI mediates the connection to a Communication network

HIGH PERFORMANCE COMPUTING

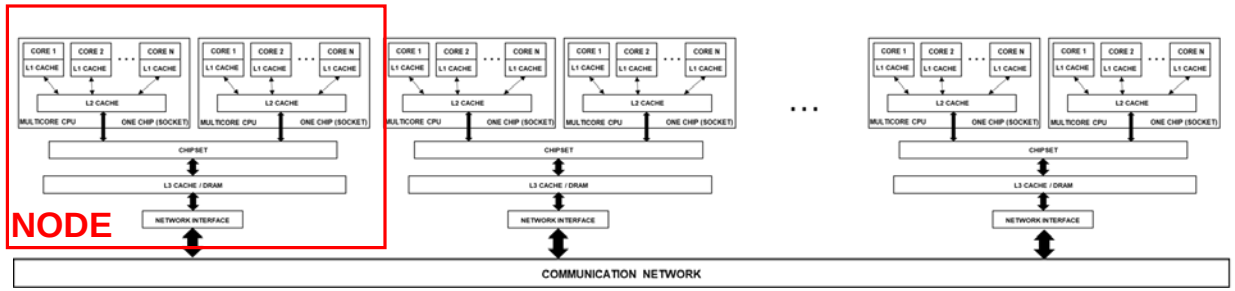
Mixture of shared-memory and distributed-memory systems

- Wikipedia: ‘redirects from HPC to **Supercomputer**’
  - Computer at the frontline of contemporary processing capacity with particularly **high speed of calculation**

[15] Wikipedia ‘Supercomputer’ Online

- **HPC** includes work on ‘four basic **building blocks**’:
  - **Theory** (numerical laws, physical models, speed-up performance, etc.)
  - **Technology** (multi-core, supercomputers, networks, storages, etc.)
  - **Architecture** (shared-memory, distributed-memory, interconnects, etc.)
  - **Software** (libraries, schedulers, monitoring, applications, etc.)

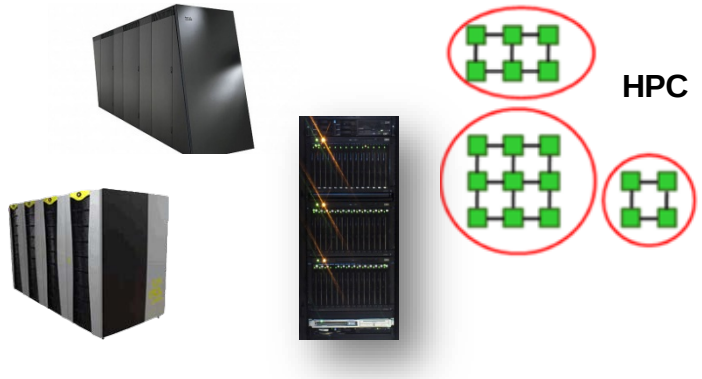
- Architecture: Shared-memory building blocks interconnected with a fast network (e.g., **InfiniBand**)



HIGH PERFORMANCE COMPUTING

Definition

- **High Performance Computing (HPC)** is based on computing resources that enable the efficient use of parallel computing techniques through specific support with dedicated hardware such as high performance cpu/core interconnections

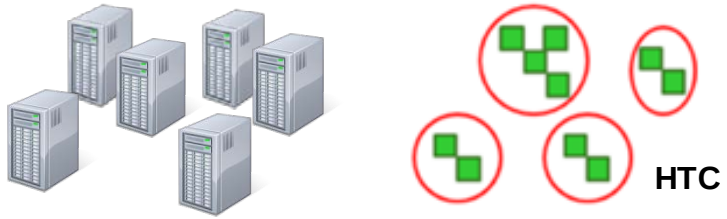


HPC

network interconnection important

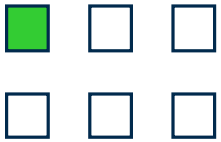
focus in this lecture

- **High Throughput Computing (HTC)** is based on commonly available computing resources such as commodity PCs and small clusters that enable the execution of 'farming jobs' without providing a high performance interconnection between the cpu/cores



HTC

network interconnection less important!



TOP 500 LIST

November 2019

- The measure of speed in HPC matters
  - Common measure for parallel computers established by **TOP500 list**
  - Based on LINPACK **benchmark** for ranking the **best 500** computers worldwide

[16] TOP 500 supercomputing sites

- LINPACK solves a dense system of linear equations of unspecified size. It covers only a single architectural aspect ('**critics exist**') ***[17] LINPACK Benchmark implementation***

- Alternatives realistic applications, benchmark suites and criteria exist

[18] HPC Challenge Benchmark Suite [19] JUBE Benchmark Suite [20] The GREEN500

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482

...

31	Forschungszentrum Juelich (FZJ) Germany	**JUWELS Module 1** - Bull Sequana X1000, Xeon Platinum 8168 24C 2.7GHz, Mellanox EDR InfiniBand/ParTec ParaStation ClusterSuite Atos	114,480	6,177.7	9,891.1	1,361

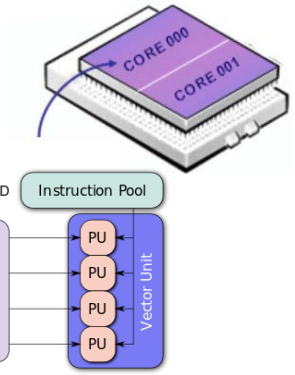
HARDWARE LEVELS OF PARALLELISM

Summary

- Best performance is achieved with a combination of them!

SIMD **SIMT** **SMT** **MPI** **MSA**

In-core parallelism



[11] Andreas Herten

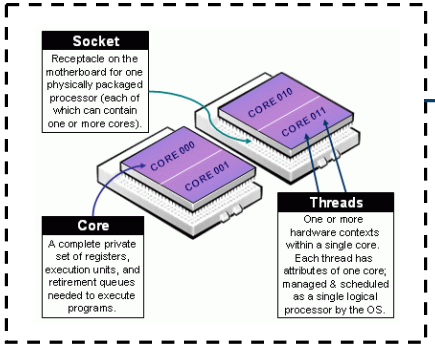
In-processor parallelism

Many threads on many cores

[12] Peter Messmer

Single Computer

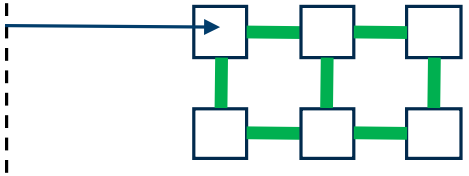
Simultaneous Multithreading
Cross-core, Cross-socket
OpenMP, pthreads



[21] SLURM

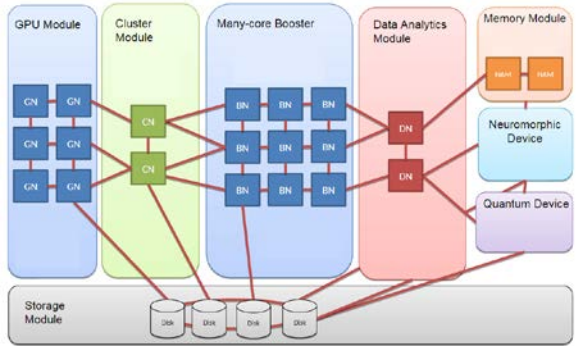
Multiple ''Computers''

Tightly-coupled Supercomputing



Multiple HPC Systems

Tightly-coupled Heterogeneous Hardware



[22] The DEEP projects

WHAT IS MESSAGE PASSING INTERFACE (MPI)?

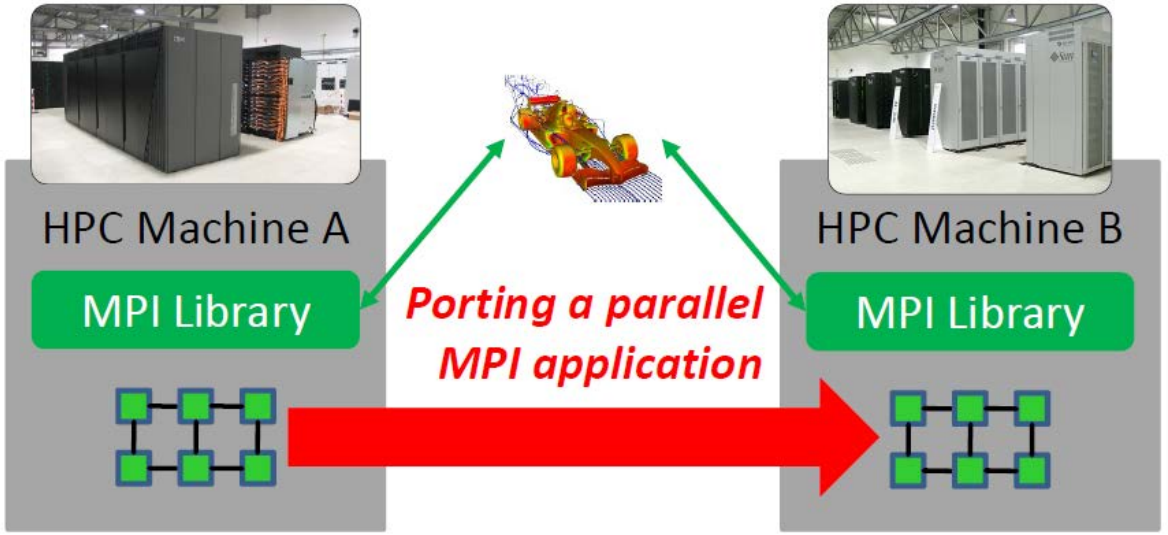
- '***Communication library***' abstracting from low-level network view
  - Offers 500+ available functions to communicate between computing nodes
  - Practice reveals: parallel applications often require just ~12 (!) functions
  - Includes routines for efficient 'parallel I/O' (using underlying hardware)
- Supports '**different ways of communication**'
  - 'Point-to-point communication' between two computing nodes (P⇔P)
  - Collective functions involve 'N computing nodes in useful communication'
- Deployment on Supercomputers
  - Installed on (almost) all parallel computers
  - Different languages: C, Fortran, Python, R, etc.
  - Careful: different versions exist

Recall 'computing nodes' are independent computing processors (that may also have N cores each) and that are all part of one big parallel computer

MPI

Key Keatures

- Simplify programming in parallel programming, focus on applications
- It is not designed to handle any communication in computer networks
- Designed for **performance** within **large parallel computers** (e.g. no security)
- Several open-source well-tested implementations of MPI
- It enables **portability** of parallel applications



START ‘THINKING’ PARALLEL

- **Parallel MPI programs know about the existence of other processes** of it and what their own role is in the bigger picture

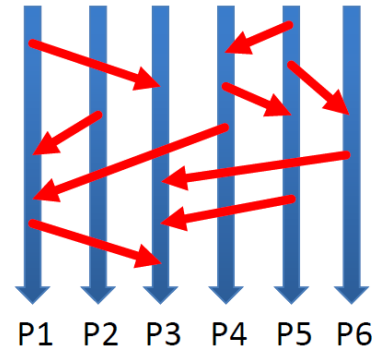
- MPI programs are **written in a sequential** programming language, **but executed in parallel**
  - Same MPI program runs on all processes (**Single Program Multiple Data**)

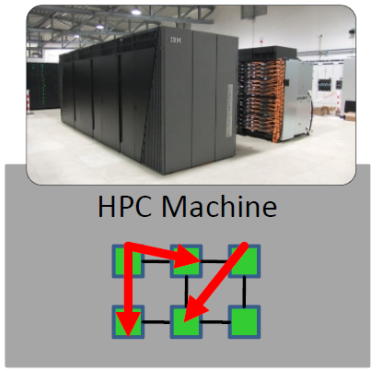
- **Data exchange** is key for design of applications
  - Sending/receiving data **at specific times** in the program
  - **No shared memory** for sharing variables with other remote processes
  - Messages can be simple variables (e.g. a word) or complex structures

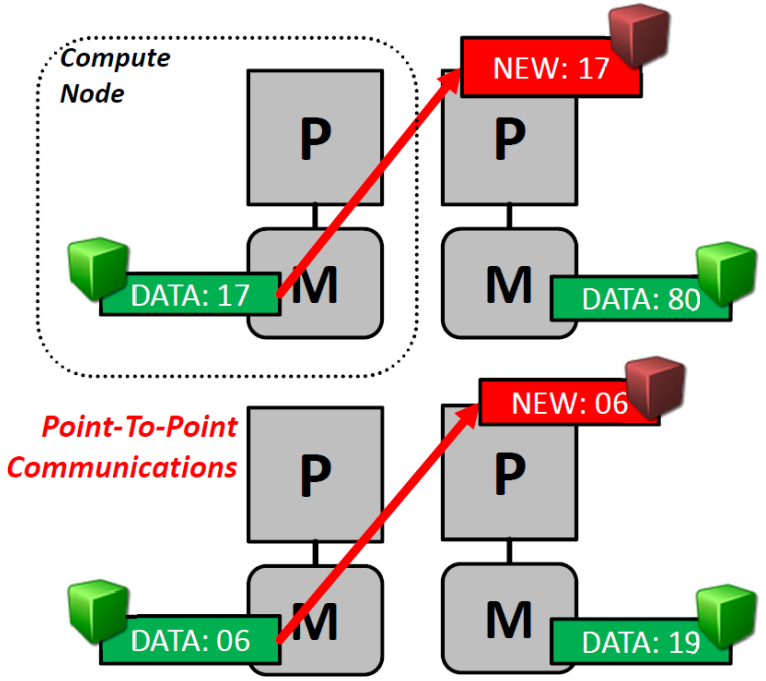
- Start with the basic building blocks using MPI
  - Building up the ‘parallel computing environment’

MESSAGE PASSING: EXCHANGING DATA

- Each processor has its own data and memory that cannot be accessed by other processors

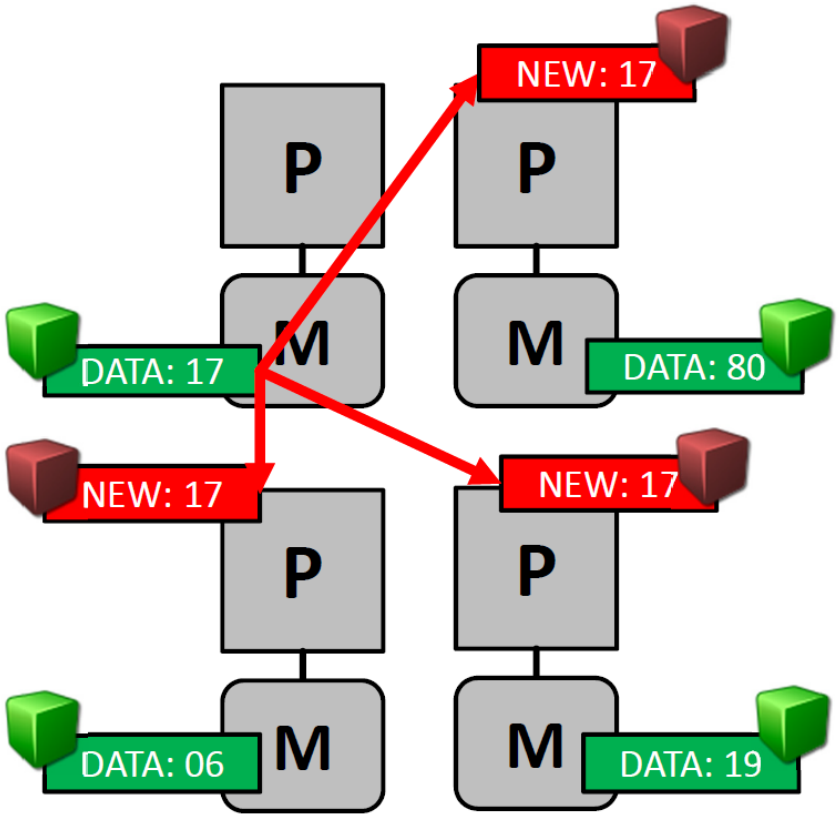






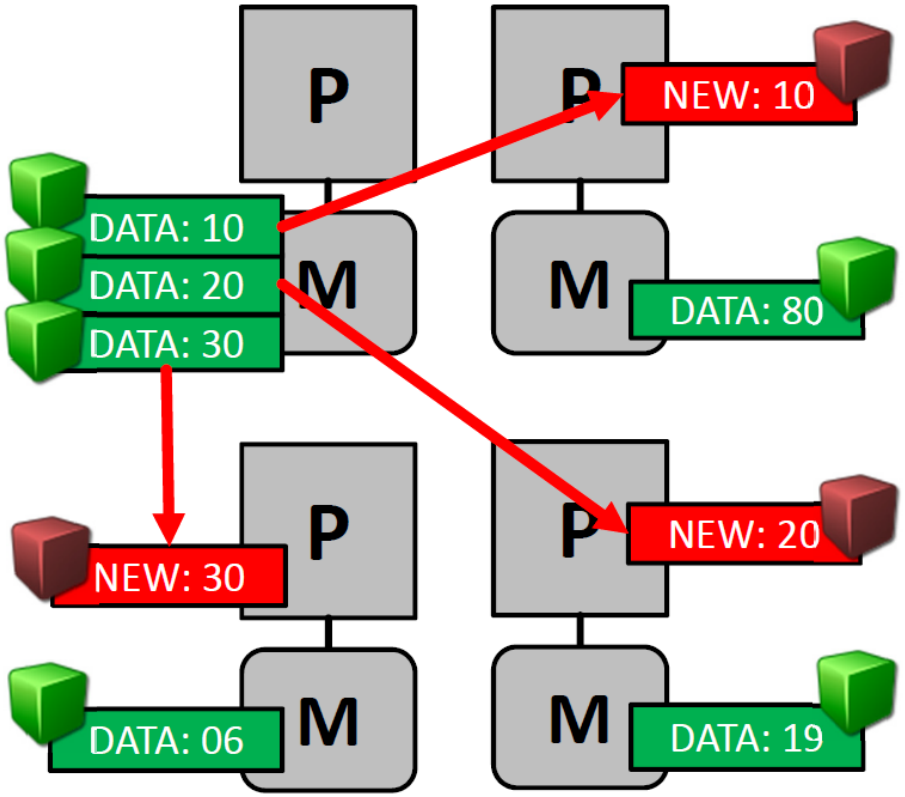
COLLECTIVE FUNCTIONS: BROADCAST (ONE-TO-MANY)

- Broadcast distributes the **same** data to many or even all other processors



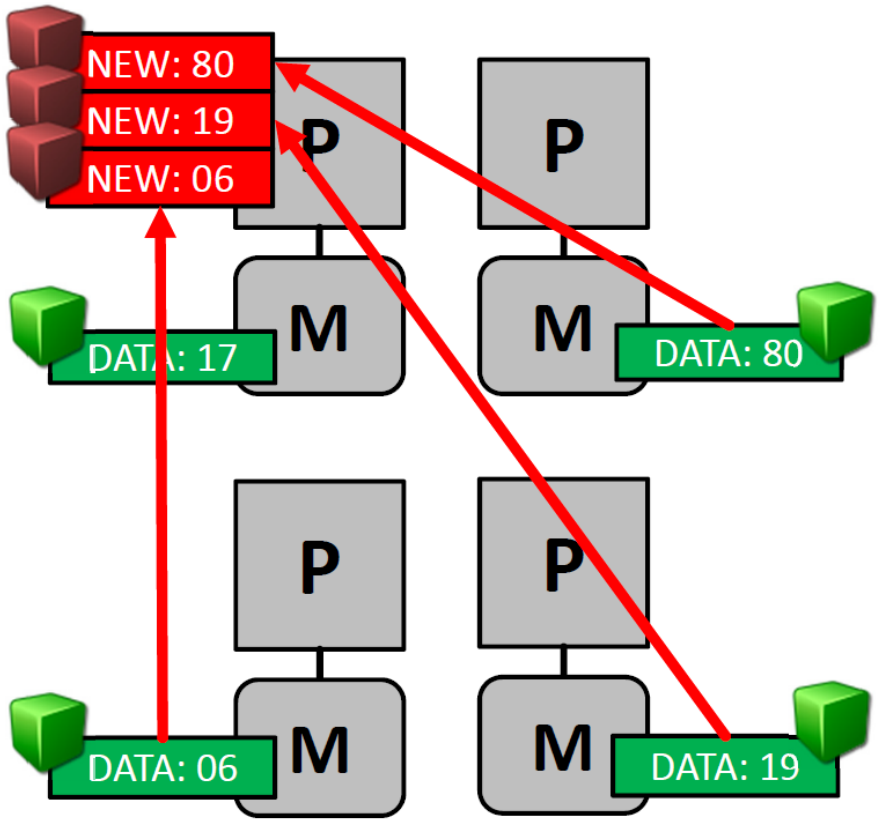
COLLECTIVE FUNCTIONS: SCATTER (ONE-TO-MANY)

- Scatter distributes different data to many or even all other processors



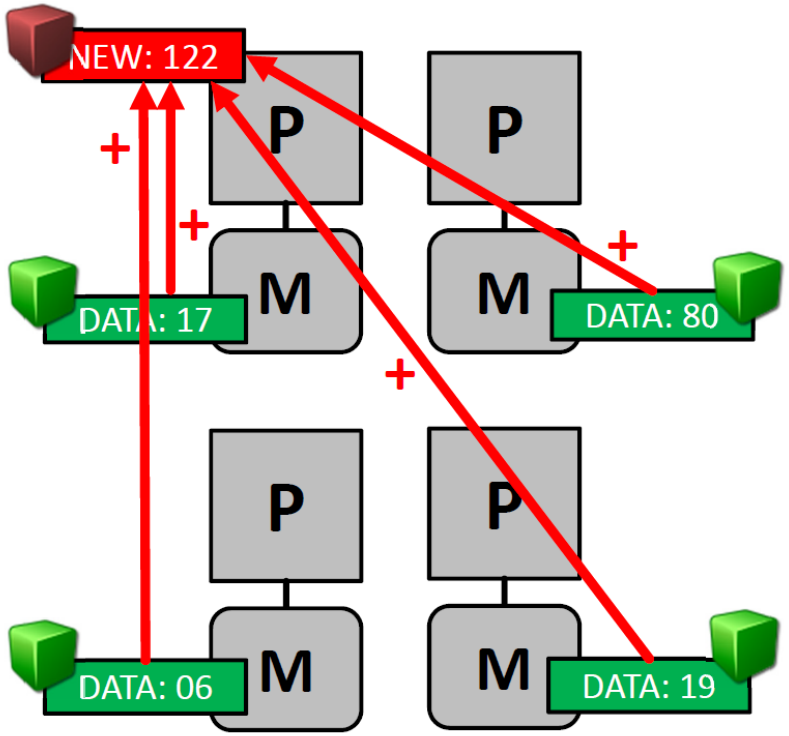
COLLECTIVE FUNCTIONS: GATHER (MANY-TO-ONE)

- Gather **collects** data from many or even all other processors to one specific processor



COLLECTIVE FUNCTIONS: REDUCE (MANY-TO-ONE)

- Each Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a **global minimum or maximum, sum, or product** of the different data located at different processors



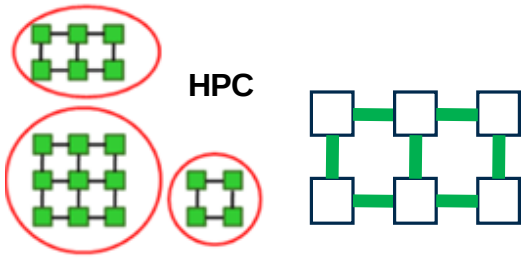
+ global sum as example

JURECA – CLUSTER MODULE

HPC System at JSC

- 1797 compute nodes
  - Two Intel Xeon E5-2680 v3 Haswell CPUs per node: 2 x 12 cores, 2.5 GhZ
- 75 compute nodes
  - Two NVIDIA K80 GPUs (2 x 4992 CUDA cores)
- Architecture & Network
  - Based on T-Platforms V-class server architecture
  - Mellanox EDR InfiniBand high-speed network with non-blocking fat tree topology
  - 100 GiB per second storage connection to JUST
- 45,216 CPU cores
- 1.8 (CPU) + 0.44 (GPU) Petaflop/s peak performance

[23] JURECA



DBSCAN

Parallel Implementation

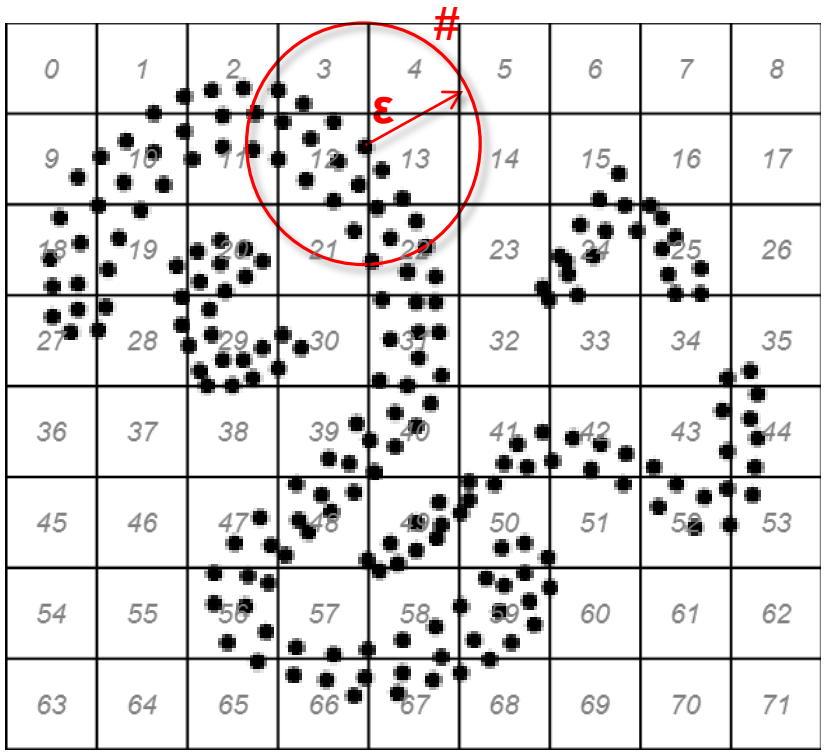
Technology	Platform Approach	Analysis
HPDBSCAN (authors implementation)	C; MPI; OpenMP	Parallel, hybrid, DBSCAN
Apache Mahout	Java; Hadoop	K-means variants, spectral, no DBSCAN
Apache Spark/MLlib	Java; Spark	Only k-means clustering, No DBSCAN
scikit-learn	Python	No parallelization strategy for DBSCAN
Northwestern University PDSDBSCAN-D	C++; MPI; OpenMP	Parallel DBSCAN

[23] M.Goetz

HPDBSCAN

Algorithm Details

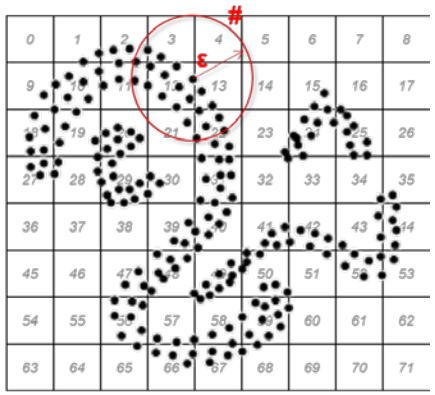
- Parallelization Strategy
  - Smart ‘Big Data‘ Preprocessing into Spatial Cells (‘indexed‘)
  - OpenMP and HDF5 parallel I/O
  - MPI (+ optional OpenMP hybrid)
- Preprocessing Step
  - Spatial indexing and redistribution according to the point localities
  - Data density based chunking of computations
- Computational Optimizations
  - Caching of point neighborhood searches
  - Cluster merging based on comparisons instead of zone reclustering



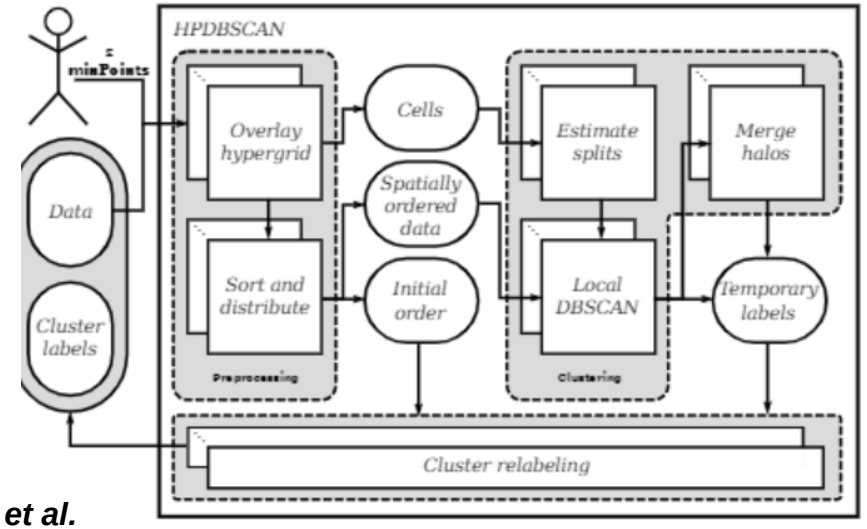
[23] M.Goetz

HPDBSCAN

Algorithm Details



1. **Load** the entire dataset in **equal-sized chunks** by all processors **in parallel**
2. Assign the data to a **unique spatial cell** related to their location within the data space,
   - With respect to the given distance function
3. Perform the **local clustering**: assign a temporary cluster label to each of the data points
4. Generate cluster relabeling rules that decide if the **temporary label assigned** by a processing unit disagrees with the ones in the halo areas of the neighboring processors
   - The rules are **broadcasted (MPI)** and applied locally

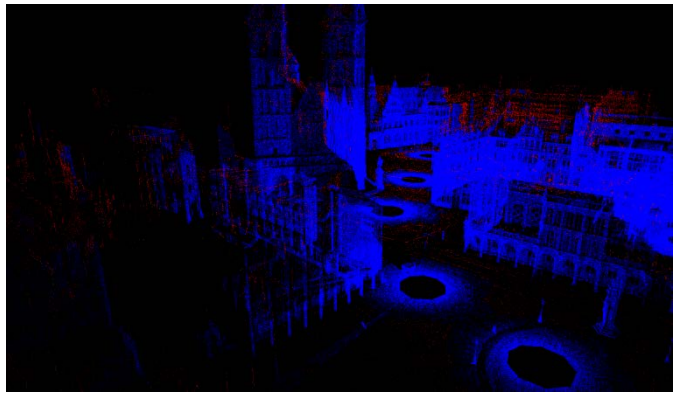


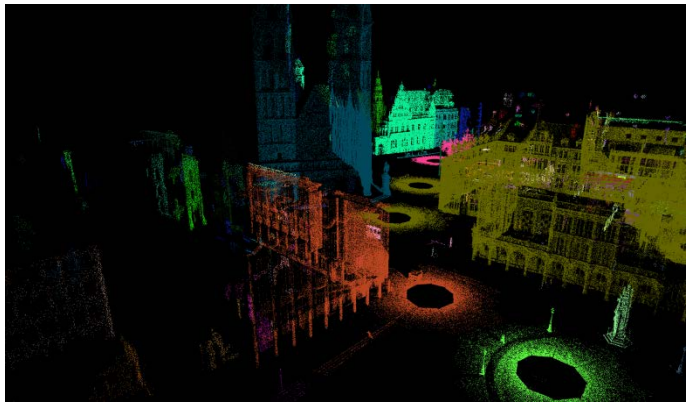
[23] M.Goetz, et al.

PRACTICAL

Bremen Dataset

- Different clusterings of the inner city of Bremen
  - Using smart visualizations of the point cloud library (PCL)
  - Big Bremen (81 mio points) & sub sampled Small Bremen (3 mio points)





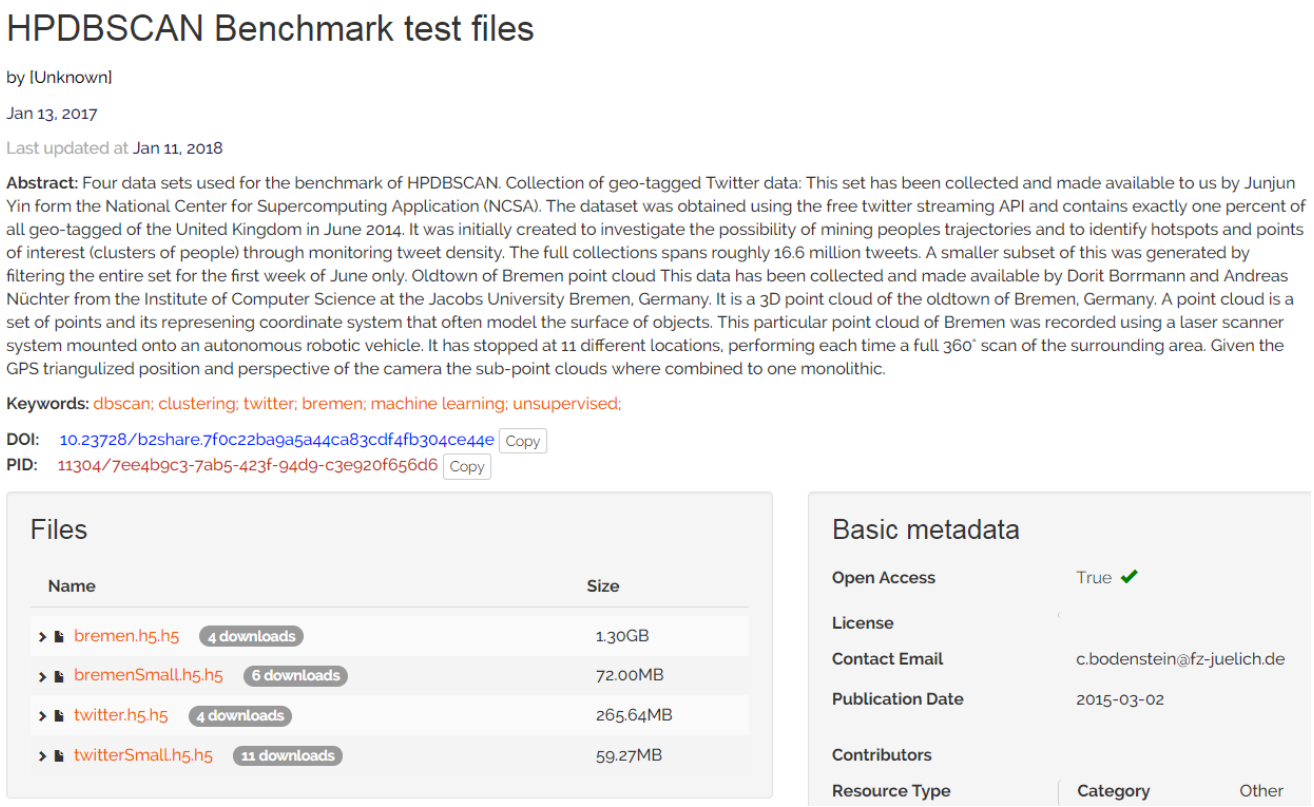
- **The Bremen Dataset is encoded in the HDF5 format (binary)**
- **You need your own copy of the file**

[25] Bremen Dataset

PRACTICAL

Bremen Dataset - Available in B2SHARE

- Available at https://b2share.eudat.eu/
  - Search for HPDBSCAN



HPDBSCAN Benchmark test files

by [Unknown]

Jan 13, 2017

Last updated at Jan 11, 2018

Abstract: Four data sets used for the benchmark of HPDBSCAN. Collection of geo-tagged Twitter data: This set has been collected and made available to us by Junjun Yin form the National Center for Supercomputing Application (NCSA). The dataset was obtained using the free twitter streaming API and contains exactly one percent of all geo-tagged of the United Kingdom in June 2014. It was initially created to investigate the possibility of mining peoples trajectories and to identify hotspots and points of interest (clusters of people) through monitoring tweet density. The full collections spans roughly 16.6 million tweets. A smaller subset of this was generated by filtering the entire set for the first week of June only. Oldtown of Bremen point cloud This data has been collected and made available by Dorit Borrmann and Andreas Nüchter from the Institute of Computer Science at the Jacobs University Bremen, Germany. It is a 3D point cloud of the oldtown of Bremen, Germany. A point cloud is a set of points and its represening coordinate system that often model the surface of objects. This particular point cloud of Bremen was recorded using a laser scanner system mounted onto an autonomous robotic vehicle. It has stopped at 11 different locations, performing each time a full 360° scan of the surrounding area. Given the GPS triangulized position and perspective of the camera the sub-point clouds where combined to one monolithic.

Keywords: dbscan; clustering; twitter; bremen; machine learning; unsupervised;

DOI: 10.23728/b2share.7f0c22ba9a5a44ca83cdf4fb304ce44e Copy

PID: 11304/7ee4b9c3-7ab5-423f-94d9-c3e920f656d6 Copy

Files

Name	Size
bremen.h5.h5 4 downloads	1.30GB
bremenSmall.h5.h5 6 downloads	72.00MB
twitter.h5.h5 4 downloads	265.64MB
twitterSmall.h5.h5 11 downloads	59.27MB

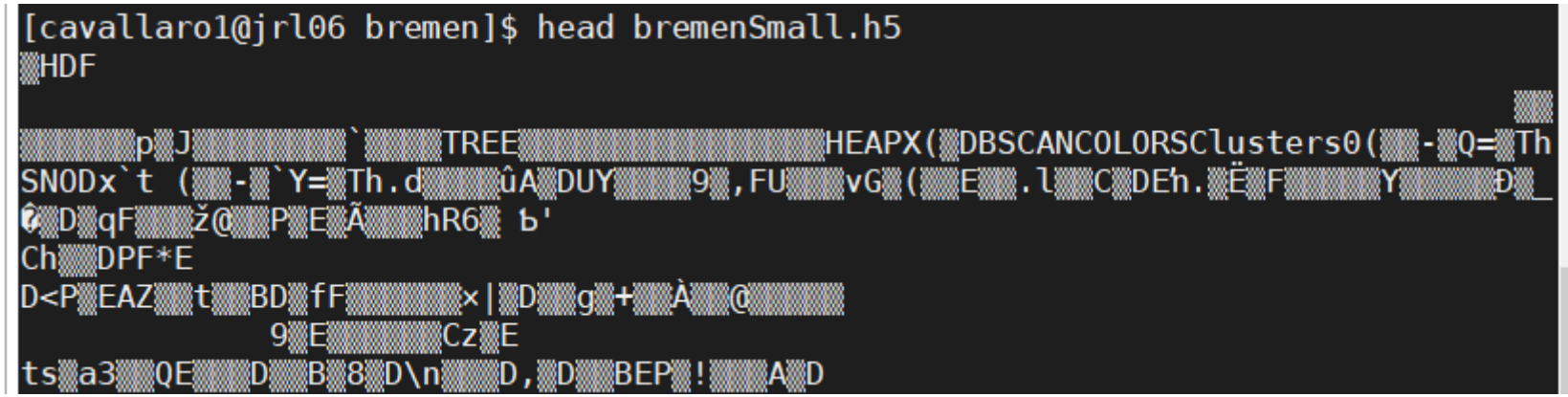
Basic metadata

Open Access	True ✔
License	
Contact Email	c.bodenstein@fz-juelich.de
Publication Date	2015-03-02
Contributors	
Resource Type	Category Other

[25] Bremen Dataset

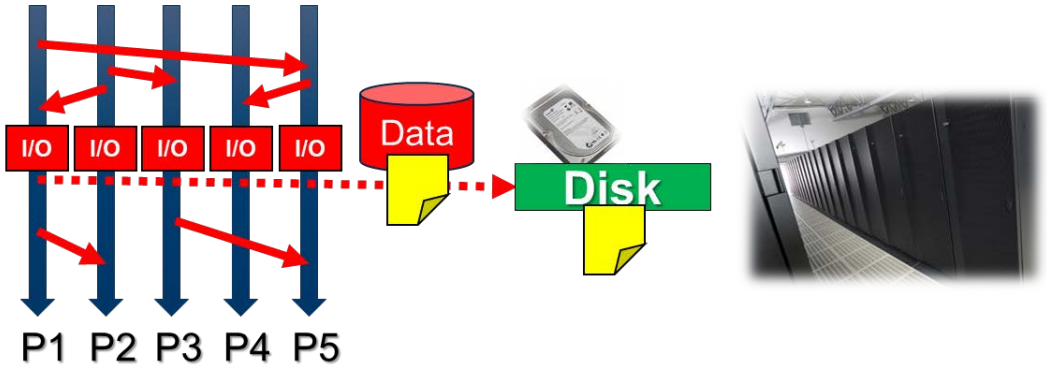
EXERCISES – EXPLORE BREMEN HDF5 DATASETS (BINARY)

- Notice binary content
- $ head /some_path/bremenSmall.h5
-



HDF5 – PARALLEL I/O

Shared File



- Each process performs I/O to a single file
  - The file access is 'shared' across all processors involved
  - E.g. MPI/IO functions represent 'collective operations'
- Scalability and Performance
  - 'Data layout' within the shared file is crucial to the performance
  - High number of processors can still create 'contention' for file systems

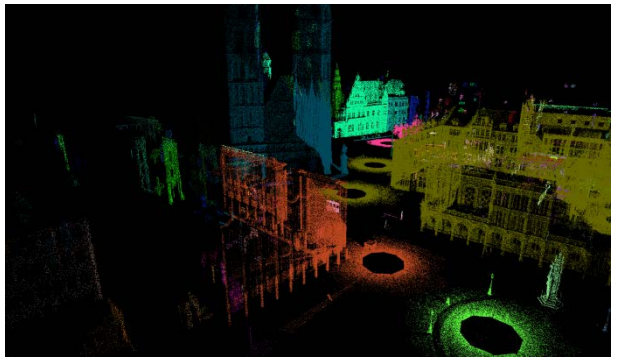
- **Parallel I/O: shared file means that processes can access their 'own portion' of a single file**
- **Parallel I/O with a shared file like MPI/IO is a scalable and even standardized solution**

JURECA HPC SYSTEM – HPDBSCAN CHECK OUTCOME

```
Calculating Cell Space...
        Computing Dimensions... [OK] in 0.001823
        Computing Cells...      [OK] in 0.018203
        Sorting Points...       [OK] in 0.214498
        Distributing Points...  [OK] in 0.126684
DBSCAN...
        Local Scan...           [OK] in 102.478086
        Merging Neighbors...    [OK] in 0.006435
        Adjust Labels ...       [OK] in 0.006754
        Rec. Init. Order ...    [OK] in 0.939040
        Writing File ...        [OK] in 0.019757
Result...
        21      Clusters
        2974394 Cluster Points
        25606   Noise Points
        2949094 Core Points
Took: 104.192540s
```

```
[cavallaro3@jrl12 ~]$ ls
HPDBSCAN-6794501.err  HPDBSCAN-6794501.out
[cavallaro3@jrl12 ~]$
```

- **The outcome of the clustering process is written directly into the HDF5 file using cluster IDs and noise IDs**



REFERENCES

- [1] Embedded system

  Online: https://en.wikipedia.org/wiki/Embedded_system
- [2] Embedded system example

  Online: https://www.electronicshub.org/basics-of-embedded-c-program/
- [3] Desktop Computer

  Online: https://www.datamate.org/buddy/
- [4] Microsoft opens UK data centres

  Online: https://www.theinquirer.net/inquirer/news/2470021/microsoft-opens-three-uk-data-centres-for-azure-and-office-365
- [5] HPC and Supercomputing at GTC 2017

  Online: https://www.youtube.com/watch?v=Usl_TCUTWD8
- [6] Parallel Programming Concepts (WT 2013/14): Introduction

  Online: https://www.tele-task.de/lecture/video/4135/
- [7] Moore's law

  Online: https://en.wikipedia.org/wiki/Moore%27s_law
- [8] Cerebras Wafer Scale Engine

  Online: https://www.cerebras.net/
- [9] Herb Sutter, ‘’The Free Lunch Is Over’’

  Online: http://www.gotw.ca/publications/concurrency-ddj.htm
- [10] Introduction to Parallel Computing

  Online: https://computing.llnl.gov/tutorials/parallel_comp/

REFERENCES

- [11] Andreas Herten, ''GPU Accelerators at JSC''

  Online: https://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/supercomputer-ressources-2019-11/12-sc-gpu.html?nn=2363978
- [12] Peter Messmer, CUDA Overview, New Features and Optimization
- Online: file:///C:/Users/caval/Downloads/Nvidia_CUDAIntro.pdf
- [13] The OpenMP API specification for parallel programming

  Online:http://openmp.org/wp/openmp-specifications/
- [14] The MPI Standard,

  Online: http://www.mpi-forum.org/docs/
- [15] Wikipedia 'Supercomputer'

  Online: http://en.wikipedia.org/wiki/Supercomputer
- [16] TOP500 Supercomputing Sites

  Online: http://www.top500.org/
- [17] LINPACK Benchmark

  Online: http://www.netlib.org/benchmark/hpl/
- [18] HPC Challenge Benchmark Suite

  Online: http://icl.cs.utk.edu/hpcc/
- [19] JUBE Benchmark Suite

  Online: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/JUBE/_node.html
- [20] The GREEN500

  Online: https://www.top500.org/green500/

REFERENCES

- [21] SLURM: Support for Multi-core/Multi-thread Architectures

  Online: https://slurm.schedmd.com/mc_support.html
- [22] The DEEP projects

  Online: https://www.deep-projects.eu/
- [23] JURECA

  Online: https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
- [24] M.Goetz, C. Bodenstein , M. Riedel, ''HPDBSCAN - Highly parallel DBSCAN'', Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments – MLHPC, 2015.
- [25] Bremen, B2SHARE, 'HPDBSCAN Benchmark test files',
  Online: https://b2share.eudat.eu/records/7f0c22ba9a5a44ca83cdf4fb304ce44e