HPC: Introduction

LECTURER: UROŠ LOTRIČ

ASSISTANT: DAVOR SLUGA

Course goals

Understanding of parallel computer design

Programming of parallel computer systems

Pattern-based parallel programming

Practical experience using a parallel cluster

Background on parallel performance modelling

Course plan

Introduction

- Architecture
- Performance models and analysis

Parallel programming patterns

- Parallel programming from a point of view of parallel computation patterns
- Map, Collectives, Data reorganization, Stencil and recurrence, Fork-Join, Pipeline

Shared memory programming

- OpenMP, OpenCL

Distributed memory programming

- OpenMPI

Algorithms

- Detailed analysis of some interesting algorithms

Course data

web:

- https://ucilnica.fri.uni-lj.si/hpc

e-mail:

- uros.lotric@fri.uni-lj.si
- davor.sluga@fri.uni-lj.si

Contact hours:

- Uroš Lotrič: before lectures or scheduled by e-mail

Course Assignments

Parallel programming lab

- Exercises for parallel programming patterns
- Programming approaches: Embarrassingly parallel, OpenMP, OpenCL, MPI

Team term project

- Programming, presentation, paper
- Team project presentations at the end of semester (last week of May)

(Oral) exam

- Midterm, early May or end of semeser?

Grading

- 1/3 Labs & homework + 1/3 Project + 1/3 Exam

Course tools

Tools:

- C language
- OpenMP, OpenCL, OpenMPI
- OS Linux

Course literature

M. McCool, A. Robinson, J. Reinders: Structured Parallel Programming: Patterns for Efficient Computation, Morgan Kaufmann, 2013

R. Trobec, B. Slivnik, P. Bulić, B. Robič: Introduction to Parallel Computing, Springer, 2018

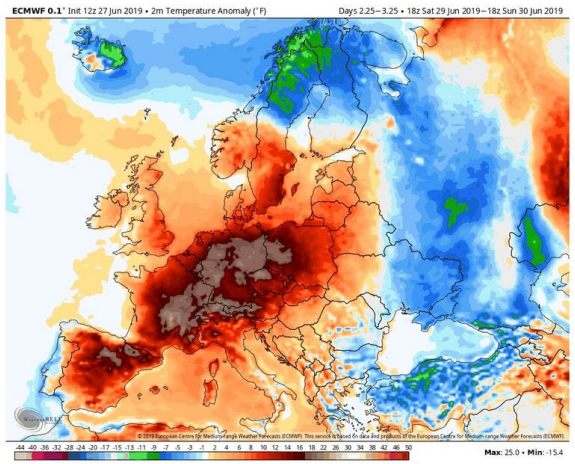
P. S. Pacheco: An introduction to parallel programming, Morgan Kaufmann, 2011

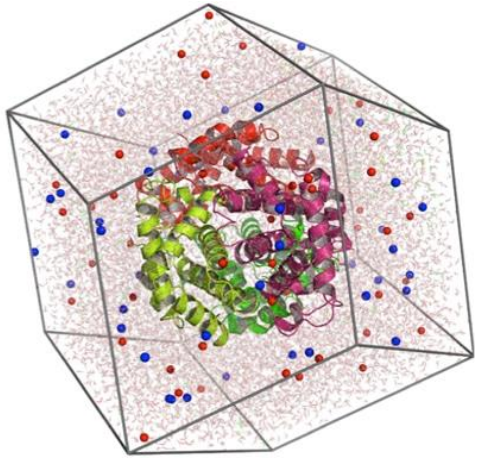
M.J. Quinn: Parallel programming in C with MPI and OpenMPI, Mc Graw Hill, Boston, 2003

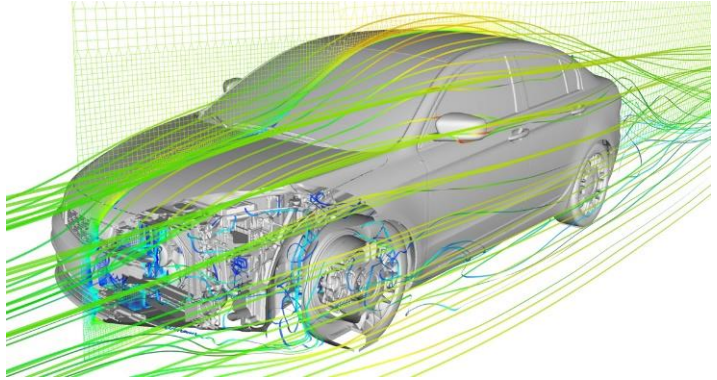
Motivation

Natural sciences

- Everlasting pursuit of realism
- Simulations help us better understand the nature
- Are much more cost and time effective compared to real-world experiments
- Applications
  - Weather forecasting, material sciences, nuclear physics,
  - Chemistry, lattice QCD, biochemistry, life sciences, genomics



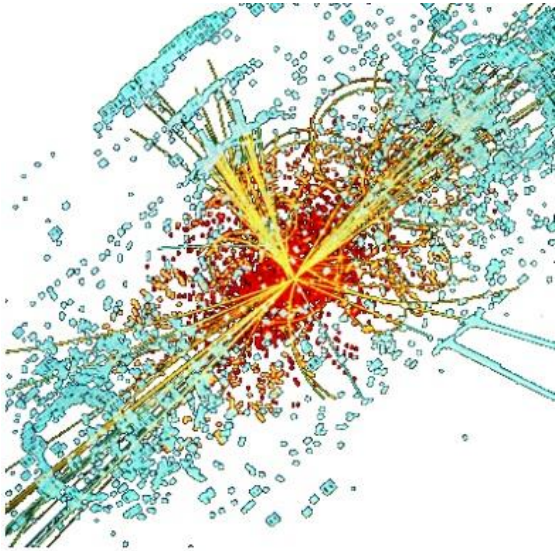


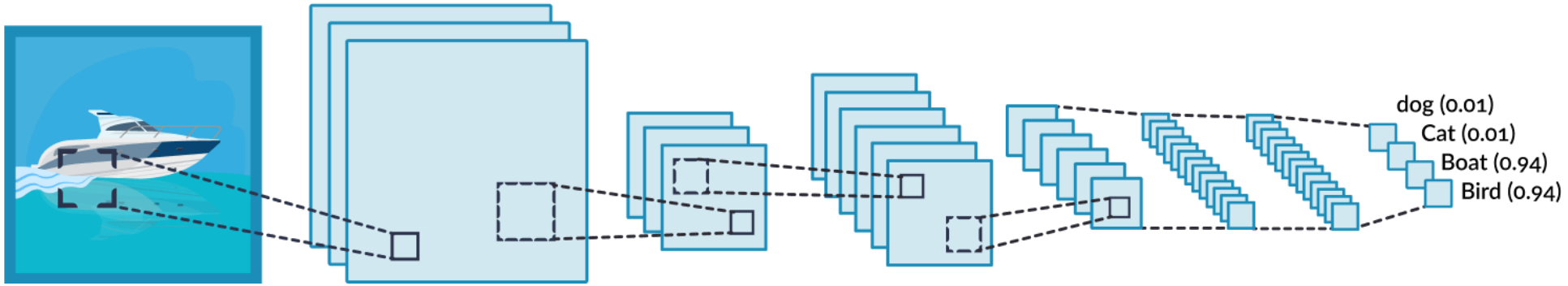


Motivation

Data analytics

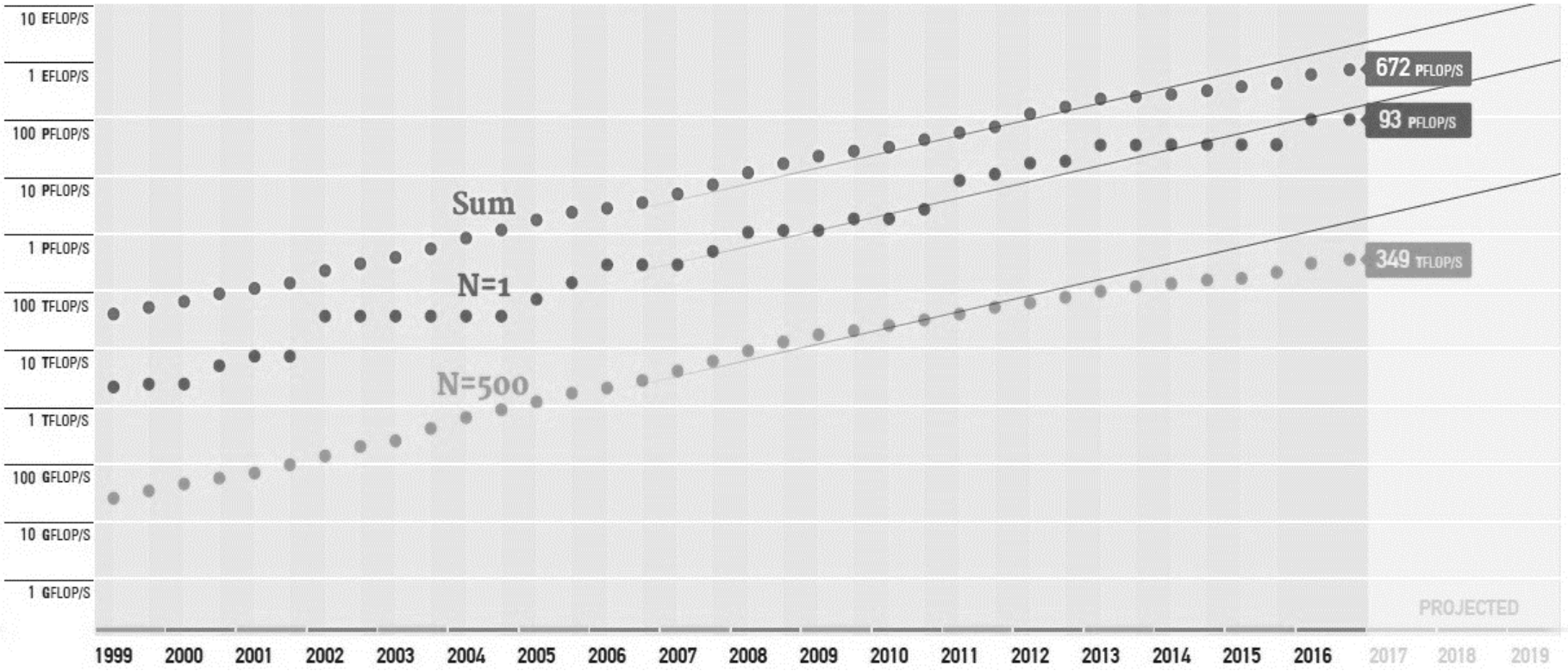
- Large quantities of data are collected
- Retrieving knowledge from data
- Confirmation of models
- Statistical and artificial intelligence modelling
- Applications
  - High-Energy Physics, Astrophysics,
  - Artificial Intelligence, Deep Learning
  - Image and signal processing, robotics





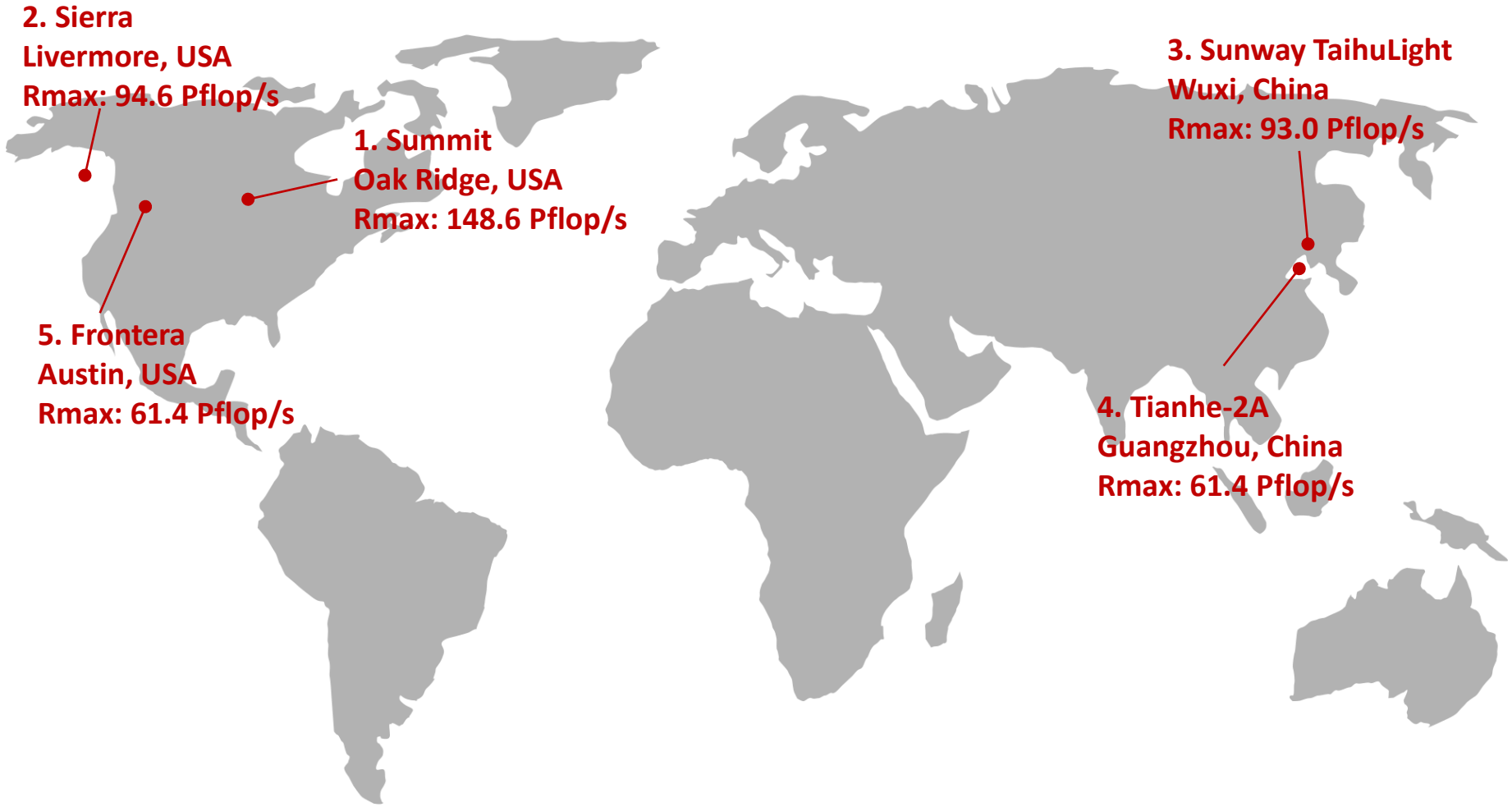
Motivation

Ever growing need for enormous computing and data processing resources



Source: A CLOSER LOOK AT 2016 TOP 500 SUPERCOMPUTER RANKINGS, https://www.nextplatform.com/2016/11/14/closer-look-2016-top-500-supercomputer-rankings/

Motivation: huge computer resources



Source: top500.org, November 2019

Motivation: huge computer resources

Europe is awakening

EuroHPC initiative

- Three pre-exascale systems by 2020/21 (>200 Petaflop/s)
- Five petascale systems by 2020/21 (~10 Petaflop/s)
- Two exascale systems in 2022/23
- Post-exascale system afterwards

Motivation: EuroHPC JU – systems

Finland
- LUMI
- Pre-exascale (>200 Petaflop/s)
- Location: Kajaani

Italy
- Leonardo
- Pre-exascale (>270 Petaflop/s)
- Location: Bologna

Spain
- MareNostrum 5
- Pre-exascale (>200 Petaflop/s)
- Location: Barcelona

Motivation: EuroHPC JU – systems

Bulgaria

- PetaSC
- Petascale (~5 Petaflop/s)
- Location: Sofia

Czech Republic

- EURO_IT4I
- Petascale (~13 Petaflop/s)
- Location: Ostrava

Luxembourg

- MeluXina
- Petascale (~10 Petaflop/s)
- Location: Bissen

Portugal

- Deucalion
- Petascale (~10 Petaflop/s)
- Location: Minho

Motivation: EuroHPC JU – systems

Slovenia is going Petascale!

VEGA

- Performance: ~10 Petaflop/s
- Location: Maribor
- Lead organization: Institute of Information Sciences Maribor
- Total budget: ~30 million €
- Collaborators: IZUM, UM, FIŠ, SLING

Goals

- Upgrade existing HPC capabilities
- Provide infrastructure for open research data
- Data storage for Slovenian R&D
- Offer computational capacities to industry
- International cooperation

HPC RIVR

IZUM
Institute of Information Science, Maribor, Slovenia

Motivation: HPC Maister@UM

Prototype system

Testing in progress

Configuration

- ~220 Tflop/s
- 76 CPU compute nodes with 4864 cores
  - 2 x AMD EPYC 7501, 512GB RAM, 2TB SSD
  - 48 + 28 (connected with Infiniband)
- 6 GPU compute nodes with 24 GPUs
  - 2 x Intel Xeon Gold 6128, 512GB RAM, 2TB SSD, 4x nVidia V100
- 3 general purpose servers with 192 cores
- 3 SSD servers with 140 TB of fast storage
- Infiniband HDR100 and Ethernet 100Gb/s

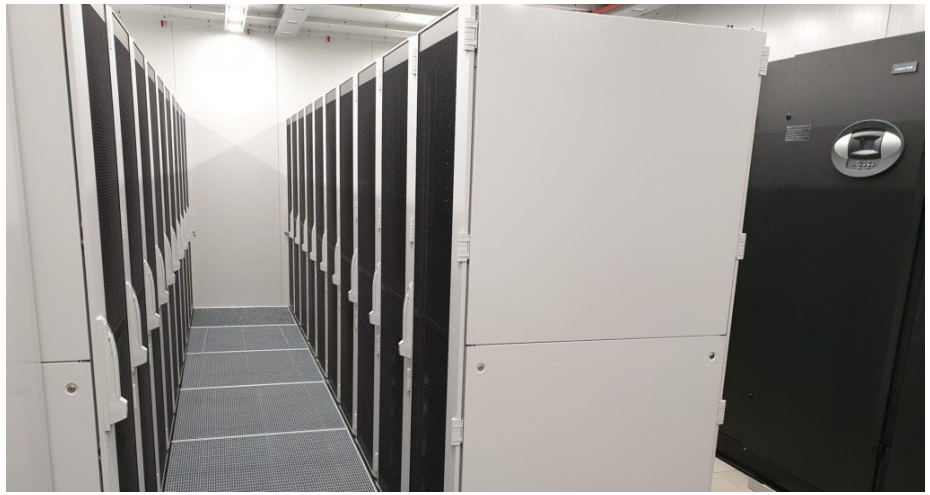
HPC Trdina@FIŠ

- Small system for educational purposes

Motivation: HPC Vega@IZUM

Full system (~10 Pflop/s)

- Deployed in April 2021
- Energy consumption ~1 MW total
- https://doc.vega.izum.si/architecture/
- Computational power
  CPU: GPU = 50: 50
  - 768 (0,25TB)+ 128 CPU (1TB) compute nodes with 122.880 cores
  - 60 GPU compute nodes with 240 Nvidia Tesla A100
- ~30 general purpose servers
  - Login nodes, front ends, databases, data transfer, virtual machines, …
- 1PB of SSD storage and 18PB of HDD storage
- Network: WAN 500 Gb/s, LAN 100 Gb/s low latency



Motivation: NSC@IJS

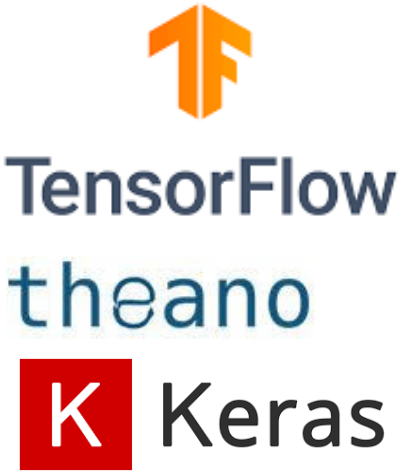
Hardware

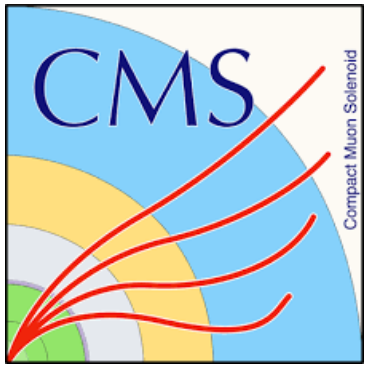
- 1984 cores
  - Nodes: 8 x 32 cores + 27 x 64 cores
- 16 GPUs NVidia Tesla Kepler 40
- 9216 GB RAM
- Some new nodes
- LAN
  - 1 Gb/s, 10 Gb/s for storage
  - InfiniBand 56 Gb/s
- WAN
  - 10 Gb/s

Software

- Middleware NorduGrid ARC
- Batch system: SLURM
- OS Centos
  - Specific SW in runtime environments
  - Light virtualization (Singularity HPC)

Motivation: HPC applications





Motivation: Slovenia HPC Fan Club

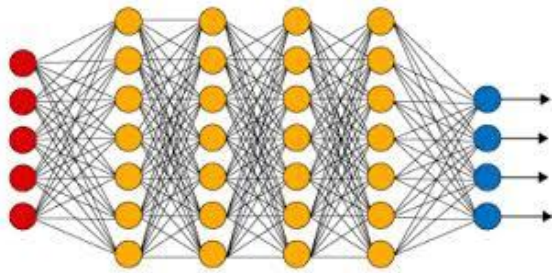
Slovenian Environment Agency

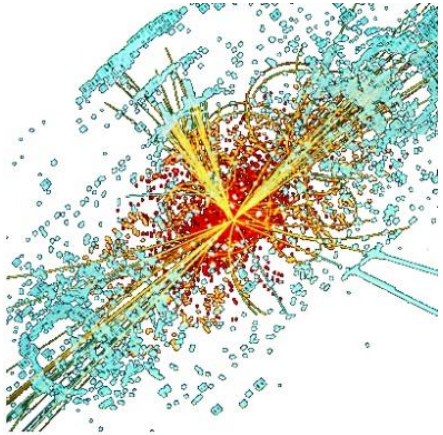
Institute Jožef Stefan

National Institute of Chemistry

Faculty of Mechanical Engineering

Turboinstitute





Motivation: HPC applications

Two primary reasons

- Faster time to solution (response time)
- Solve bigger computing problems (in the same time)

Other factors that motivate parallel processing

- Effective use of machine resources
- Cost efficiencies
- Overcoming memory constraints

Parallelism = concurrency + parallel HW + performance

Motivation: HPC applications

How does my application scale?

I have a specific problem, none of the packages fits.
How to solve it more efficiently?

How to develop such nice tools and packages?

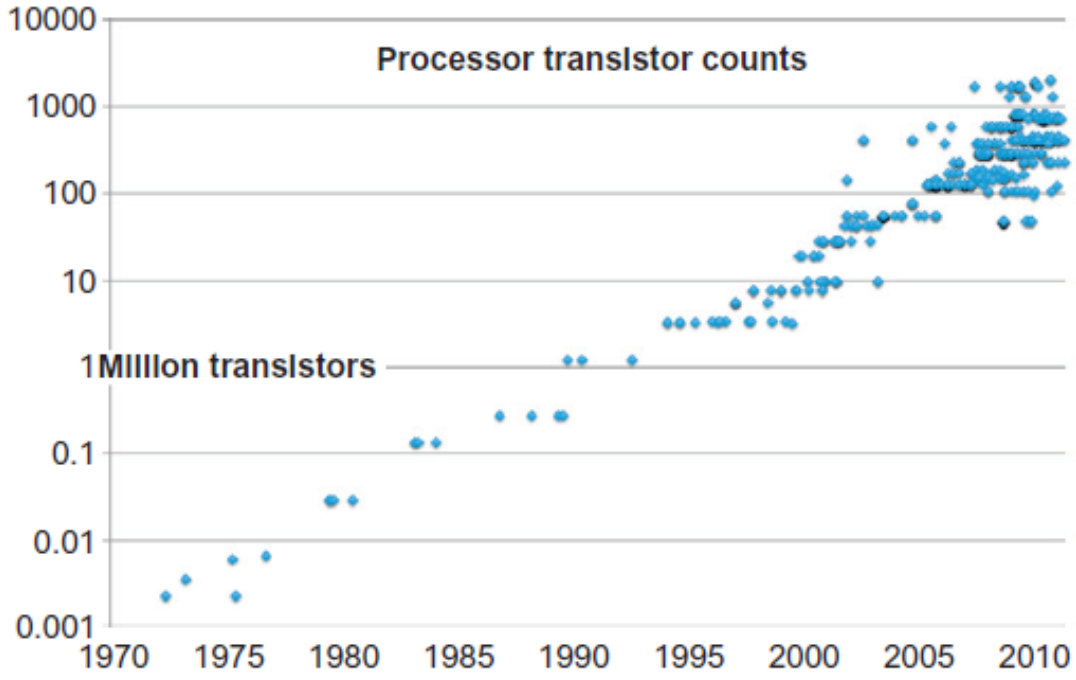
There is a new technology available.

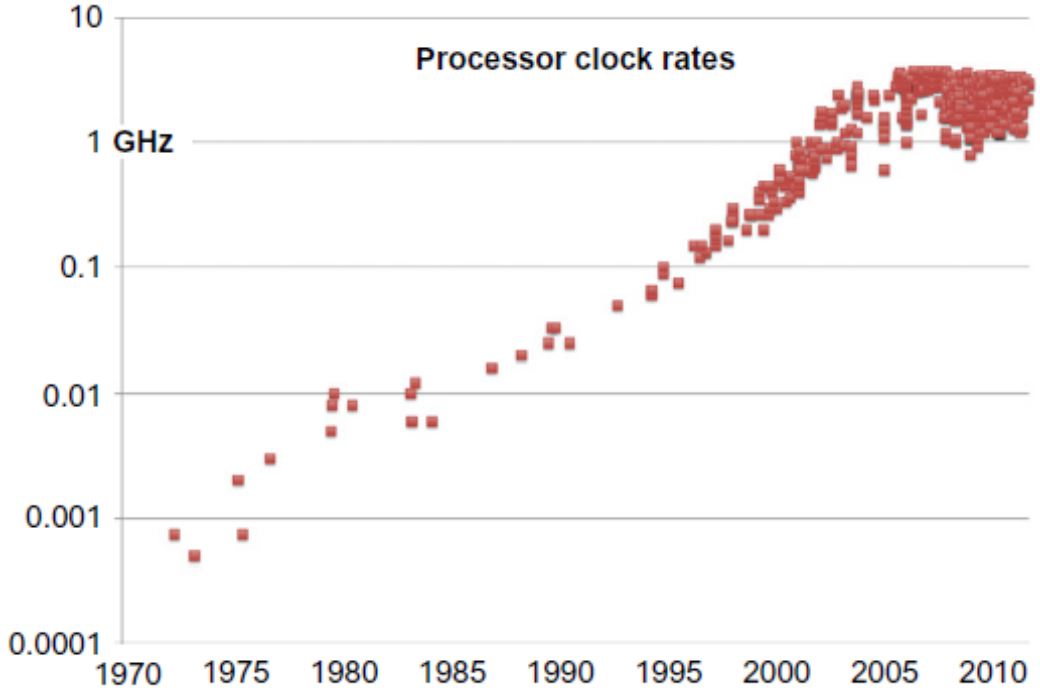
- Can my problem make use of it?
- How to adapt my program for it?

Motivation: Pervasive parallelism

Hardware trends are encouraging parallelism

- Moore's Law
  - Number of transistors doubles every 18 months,
  - Still stands, last years more diversity
- Till 2004: smaller components, increasing clock rates, hardware changes

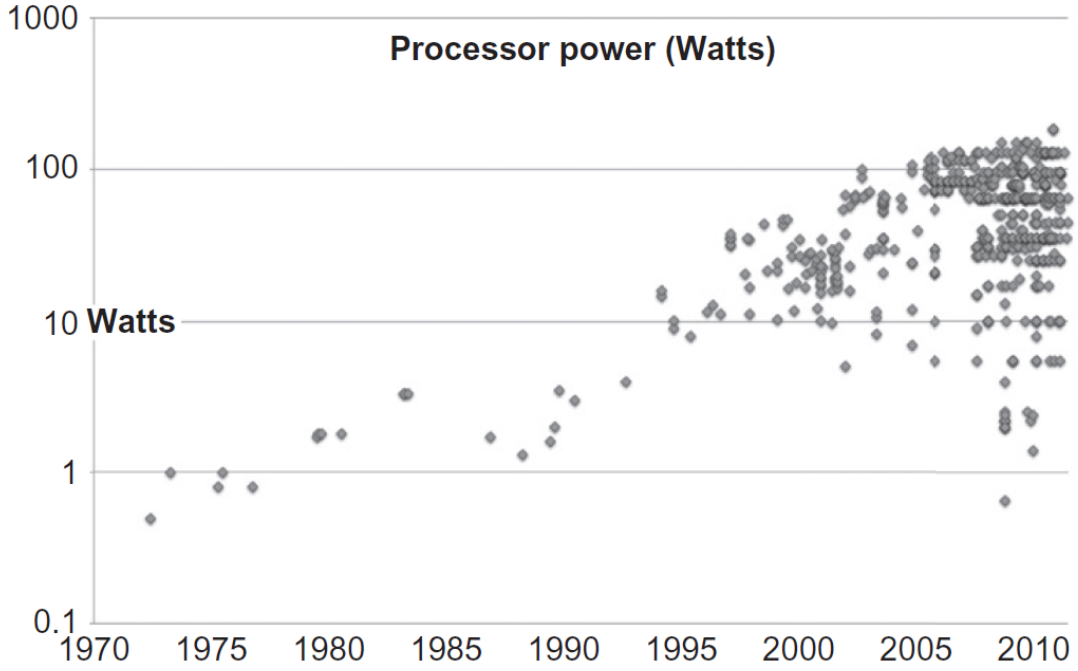


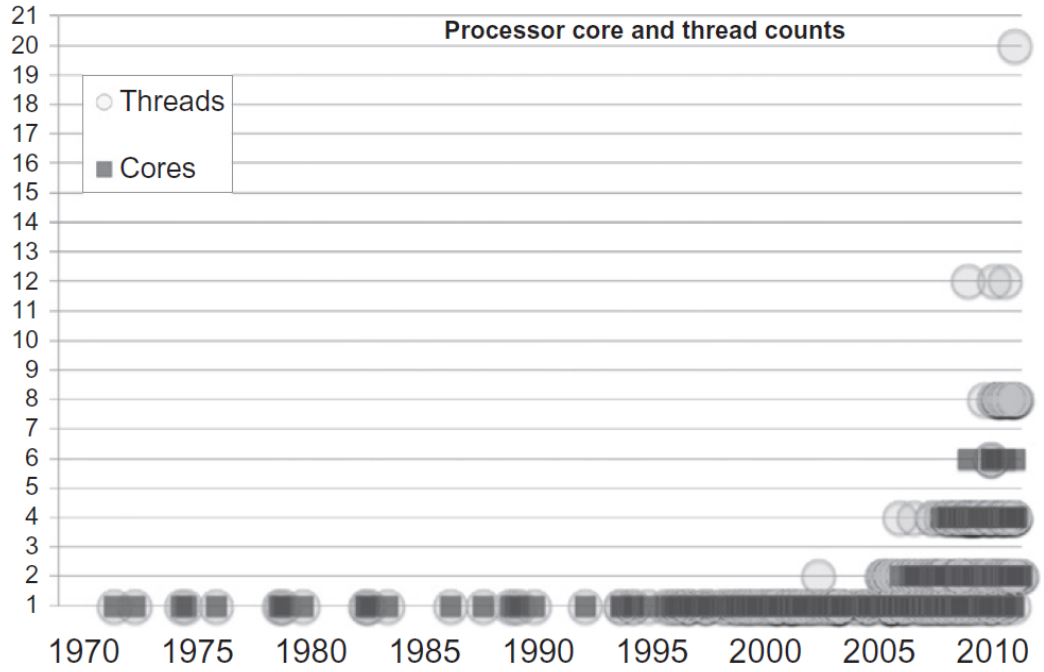


Motivation: Pervasive parallelism

Hardware trends are encouraging parallelism

- Dynamic power increase to 130 W,
- After 2004: Greater efficiencies, lower power solutions

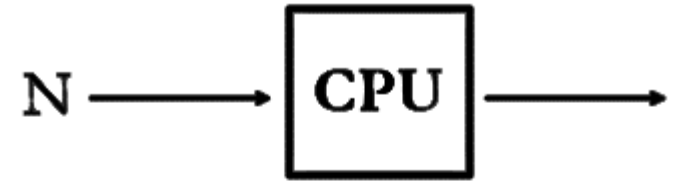




Motivation: Pervasive parallelism

Hardware trends are encouraging parallelism

- Two is better than one



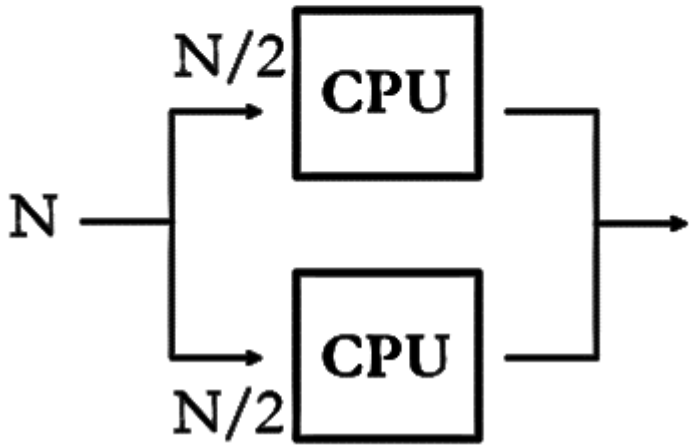
f_1
U_1
C_1
$P_1 = k\, C_1\, f_1\, U_1^2$

$$P = U_0 I = U_0 \frac{de}{dt} = U_0 C \frac{dU}{dt}$$

$$U = U_0 \sin(2\pi f\ t)$$

$$\frac{dU}{dt} = 2\pi f U_0 \cos(2\pi f\ t)$$

$$P = U_0 C \frac{dU}{dt} = 2\pi C f U_0^2$$



$f_2 = 0.5\, f_1$
$C_2 = 1.2\, C_1$
$U_2 = 0.6\, U_1$
$P_2 = 2\, k\, C_2\, f_2\, U_2^2 = 0.43\, P_1$

Motivation: Pervasive parallelism

Three walls in 2005

- Power wall
  - unacceptable growth in power usage with clock rate
- Instruction Level Parallelism wall
  - limits on hardware parallelism
    - Superscalar instructions (simultaneous execution of unrelated instructions)
    - Very Large Instruction Word (compiler analysis)
    - Pipelining
    - Speculative techniques: branch prediction and prefetching (cache)
- Memory wall
  - huge discrepancy of processor speeds relative to memory speeds
  - off-chip communication is slow and power hungry
  - cache helps, but is more complicated than on a single CPU
  - latency and bandwidth
  - problems with scalability

Motivation: Pervasive parallelism

Three walls in 2005

- Need to write explicitly parallel programs
- New processor designs provide multiple mechanisms for explicit parallel programming
- You must use them and use them well for good scalability
- If you write a program for scalable parallelism, it will continue to scale on new processors with more and more parallelism available

Historical trends

Hardware is naturally parallel

- Parallelism in hardware has been present since the earliest computers
- Great sophistication in mainframe and vector supercomputers (late 1980)

Miniaturization

- Intel 4004 4-bit microprocessor with 2300 transistors
- Today million times more transistors, a lot of potential
- Improvements over the years:
  - word sizes,
  - superscalar capabilities, vector instructions, out-of-order execution, deep pipelines, parallel arithmetic units, multithreading,
  - caches, cache prefetching, virtual memory controllers, page table walking, memory access controllers,
  - graphics processing units

Historical trends

Term supercomputer is first used during development of Cray-1, 1976, 10 mio. USD, Los Alamos National Laboratory

At the end of seventies they are introduced to petrol and automotive industry

In eighties they come to business world

Why?

- Faster computations gives competitive advantage
- Less experiments means cheaper development
- Faster development of new products

Historical trends

First supercomputers were vector computers

- High price, slow development

Today supercomputers are distributed computer systems

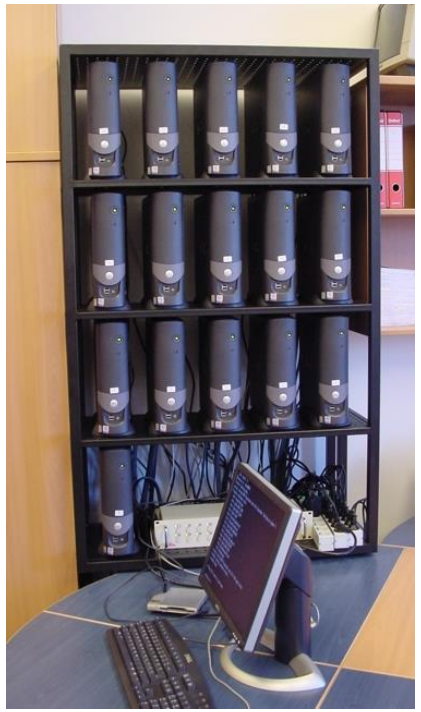
- Massive production, faster development
- Cheaper systems, step-by-step upgrade

Historical trends

Supercomputing in Slovenia

- CONVEX SPP1000/XA-64
  - 64 processors, 6.19 Gflop/s
  - 8 mio. USD, IJS, 1992

- Grid of workstations
  - Today each CPU core is capable of ~10 Gflop/s



Historical trends

Modern parallel computers

- Standard computer in institutes: DEC VAX 11/780, 1 MFLOPS
- 1981: Cosmic Cube:
  - 64 Intel 8086 processors (XT), 5 – 10 MFLOPS, za 1/2 VAX price
- 1986: Connection Machine, Thinking Machine Corporation,
  - 1 CPU, many ALUs
- 1994: Beowulf, NASA,
  - 16 standard Intel DX processors connected to 10 Mbit Ethernet, Linux
- 1996: cluster for less than 50k USD is capable of 1 Gflop/s
- 2019:
  - Summit (1st) and Sierra (2nd) equipped with nVidia Tesla V100 GPUs
  - TaihuLight (3rd) only processors
  - Tianhe-2A (4^{th}) processors and Matrix-2000 accelerators (replacement for Xeon Phi due to USA ban)
  - Trinitry (7^{th}) equipped with Xeon Phi processors

HPC: Parallel hardware

LECTURER: UROŠ LOTRIČ

ASSISTANT: DAVOR SLUGA

Recall: von Neumann architecture

Instructions and data are stored in memory

Machine cycle

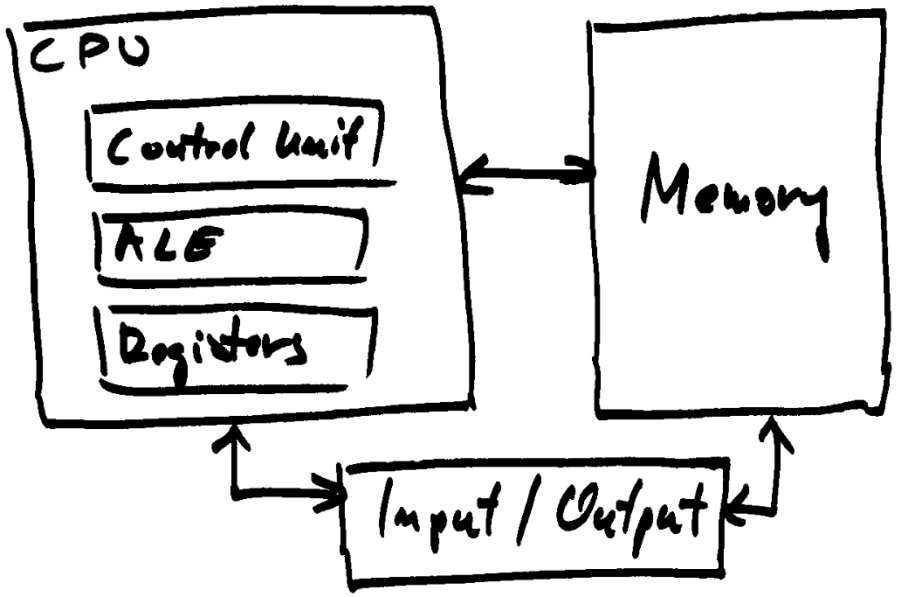
- Fetch
- Decode
- Execute
- Store

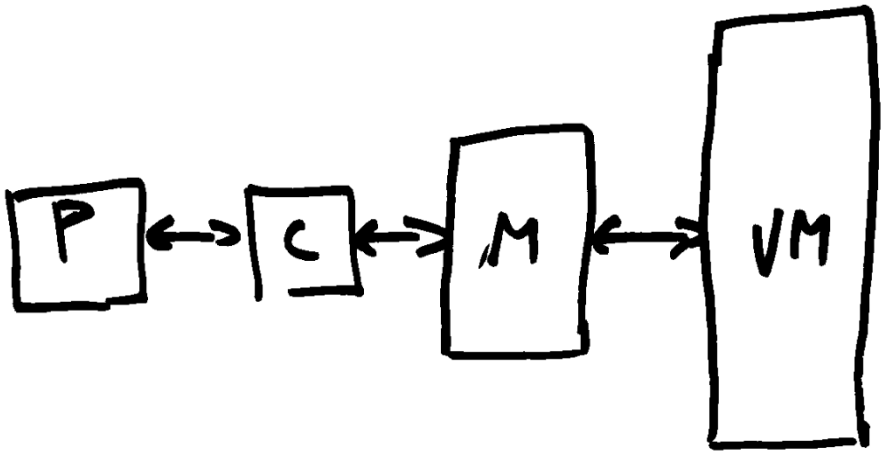
Single instruction, single data

Memory bottleneck

Improvements

- Memory hierarchy
  - Registers, cache, main memory, virtual memory
  - Temporal and spatial locality
- Parallelism in hardware
  - Pipeline, Superscalar execution, Speculative execution,
  - Multiple functional units, Hardware multithreading





Recall: Memory

Memory hierarchy

- Registers, caches, main memory, virtual memory
  - Access to main memory is two orders of magnitude slower than to the cache
- Cache
  - Organized in cache lines (256B)
  - Coherent memory access
  - Latency (crucial) and bandwidth

Recall: Cache

Block of memory is always transferred

- Make use of temporal and spatial locality
- If data is missing, processor still waits
  - Less frequent than without cache
- example

```
float z[1000];
sum = 0;
for(i=0; i<1000; i++)
    sum += z[i];
```

- Multi-stage memory
  - Caches L1, L2, L3
  - Main memory
  - Virtual memory

Recall: Cache

Cache hit

Cache miss

- Copying from the main memory
- Where to copy?
  - Delete the oldest block
  - Associative: all blocks are equivalent
  - Set-associative: some possible locations
  - Direct: exact mapping of main memory block to cache locations

Reading and writing

- Coherence
  - Write-through: when writing to cache, data is also written to main memory
  - Write-back: dirty bit is set in cache when writing, when block is about to be overwritten, it’s content is first written to the main memory

Recall: Cache

Example of cache misses

- Matrix A is stored row-wise
- Cache size is 4 doubles

```
double A[MAX][MAX], x[MAX], y[MAX];

for(i=0; i<MAX; i++)
    for(j=0; j<MAX; j++)
        y[i] += A[i][j]*x[j];

for(j=0; j<MAX; j++)
    for(i=0; i<MAX; i++)
        y[i] += A[i][j]*x[j];
```

Recall: Virtual memory

Extends main memory

Page size 4 MB

Page fault

- disk access latency much higher (1.000.000) than cache miss, takes to resolve
- Importance of data locality
- Rapid virtual to physical memory address mapping (TLB)

TLB (Translation Lookaside Buffer)

- Specialized cache for address translation (up to 128 entries on 1st level)
- TLB miss:
  - access to a page not in TLB, walking over the whole page table to find translation
  - Example: stencil @ large 3D array: row, column, page access
    - Element by element walk, number of pages can be large
    - High TLB miss rate when accessing neighbouring elements even when number of page faults is low
    - If the 3D array is big, a high page fault rate may also result

Recall: Instruction level parallelism

Pipeline

- Functional units are arranged in stages
- Stages should be of similar complexity
- Summation of two floating point numbers
  - 1,23e4 + 6,54e3
  - reading, exponent comparison, alignment of exponents, summation, normalization, rounding, storing
  - Number of cycles for 1 and for 100 summations?

Recall: Instruction level parallelism

Vector instructions

- Explicit usage of multiple functional units specified by operation on small amount of data
- Special instructions

Parallel arithmetic units

- Parallelism in single thread
- Two units:
  - one is summing odd and the other even indices

```
for(i=0; i<1000; i++)
    z[i] = x[i] + y[i]
```

Recall: Instruction level parallelism

Superscalar

- Parallel execution of non-dependable instructions
- Allocation of functional units is dynamic, at runtime (for non-superscalar processors is static, at compile time)
- Example: parallel arithmetic units with dynamic allocation
- Example: speculative execution

```
if(x > 0)
{
    z = x + y;
    w = x;
}
else
    w = y;
```

  - **`if(x > 0)`** and assignment **`z = x + y`** are performed in parallel

Recall: Instruction level parallelism

Hardware multithreading

- No need to search for parallelism in sequential thread
  - In many cases does not exist
- Simultaneous multithreading
  - instructions from multiple threads feed to superscalar scheduler (dynamic, at runtime)
- Switch-on event multithreading
  - Latency hiding
  - when one thread stops due to memory or IO access, another can be processed
  - Rapid switching of threads at long-latency operations

Serial illusion

Developments in hardware have led to long-sustained serial illusion

Serial traps: Serial assumptions incorporated in tools and thinking

- Language design

```
1  void
2  addme(int n, double a[n], double b[n], double c[n]) {
3     int i;
4     for (i = 0; i < n; ++i)
5        a[i] = b[i] + c[i];
6  }
```

```
1  double a[10];
2  a[0] = 1;
3  addme(9, a+1, a, a); // pointer arithmetic causing aliasing
```

- Compilers are not reliable at discovering parallelism
  - Loops, memory access through pointers in C