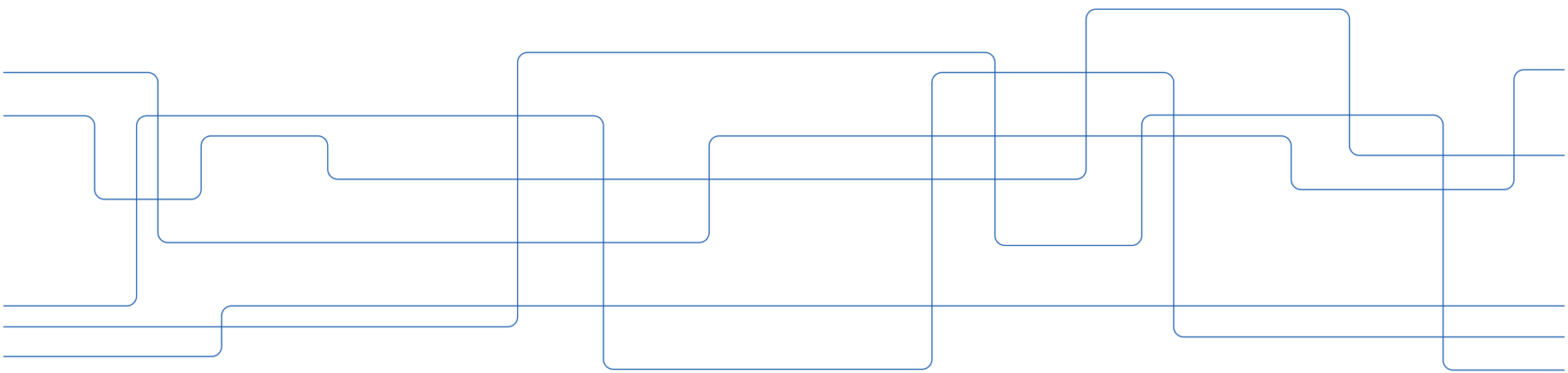
Performance Engineering on Modern Processor Architectures

Dirk Pleiter

KTH, EECS, PDC and CST

2021-10-05

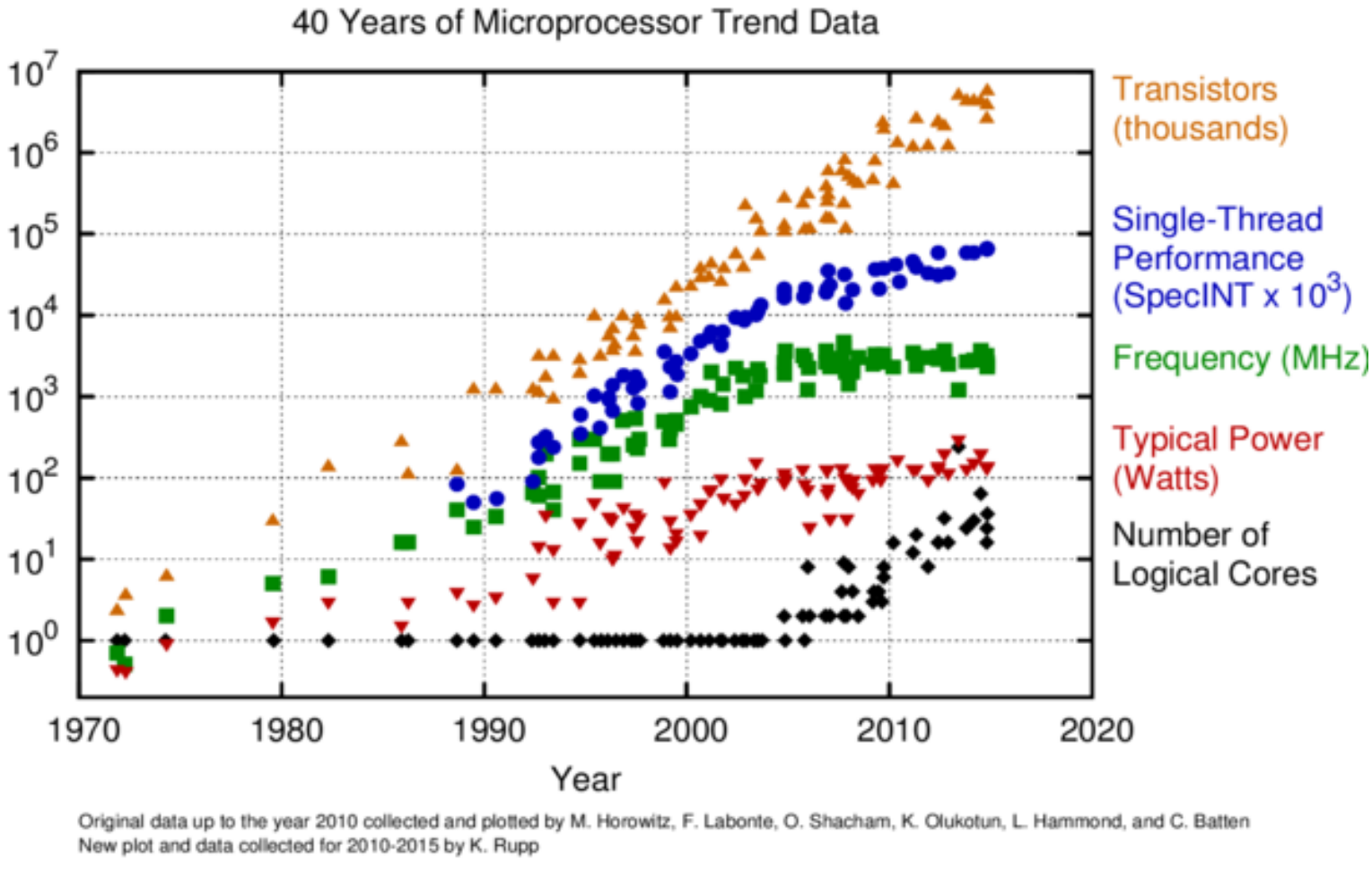


Overview

Content

Uni-Processor Performance Limits

[Karl Rupp, 2015]



Rent's Rule

- **Rent's rule**:

$$T = k\, G^p$$

  - G ... Number of logic elements (gates)
  - T ... Number of edge connections (terminals)
  - k ... Rent's coefficient
  - p ... Rent's exponent
- Problem: typically $p \ll 1$

  ☞ Difficult to ballance communication and compute
- Strategy for problem mitigating: **Memory hierarchy**
  - Fast but small on-chip memory (cache)
  - Slower but larger off-chip memory
- **Trend towards deeper memory hierarchies**
  - DRAM main memory + high-bandwidth memory

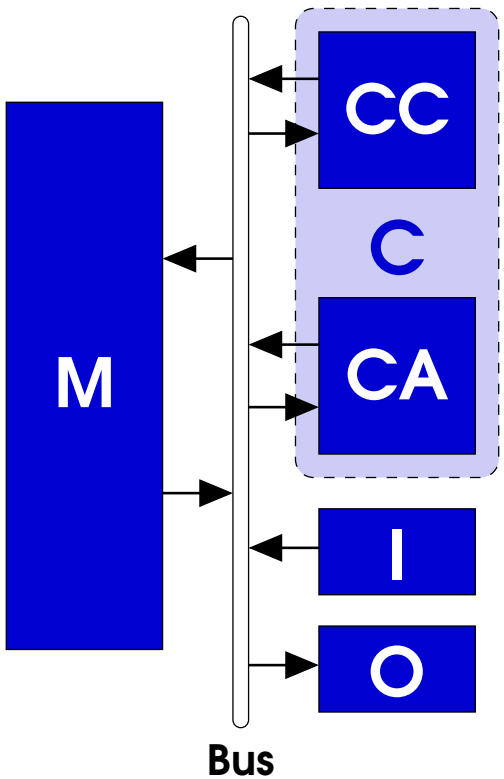
Key Challenges of Performance Engineering

- Exploit parallelism at all levels
  - Pipeline parallelism
    - Example: instruction-level parallelism (ILP)
  - Instruction-level data parallelism
    - Example: SIMD instructions
  - Core- and node-level parallelism
    - Example: multi-threading
- Optimise data transport
  - Mitigate memory bandwidth limitations
  - Hide memory access latencies

Content

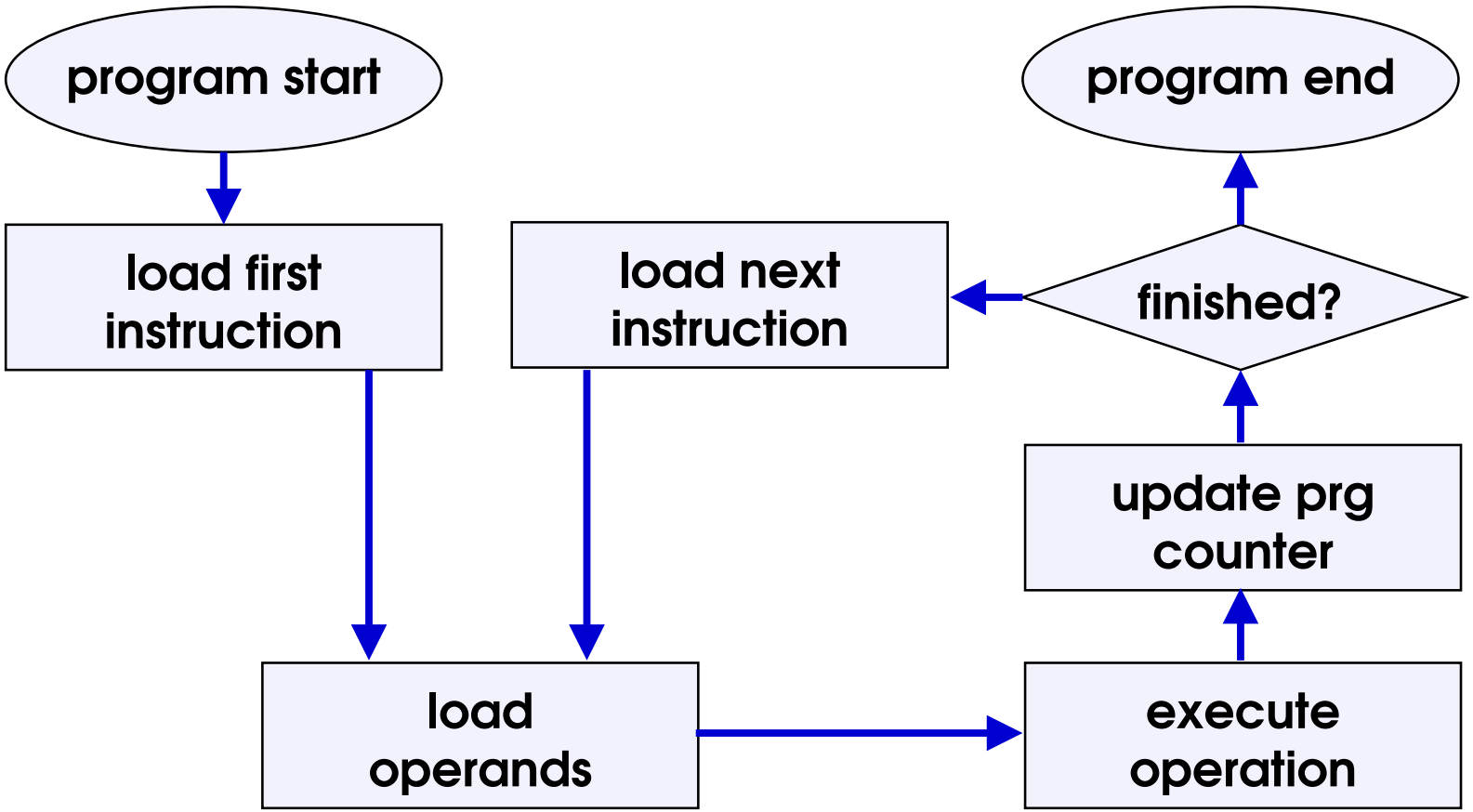
Von Neumann Architecture

[Neumann, 1945]

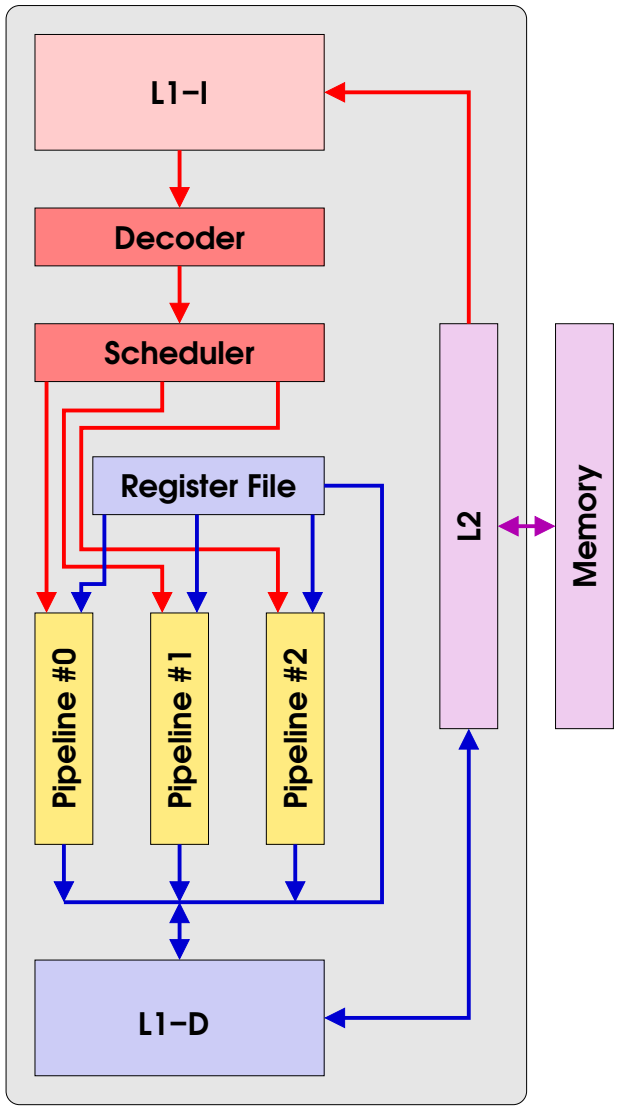


- Components defined by v. Neumann:
  - Central arithmetic CA
  - Central control CC
  - Memory M
  - Input I
  - Output O
- Simplified modern view:
  - Central Processing Unit
  - Memory
  - I/O chipset
  - Bus
- Memory used for instructions and data ☞ Self-modifying code

V. Neumann Architecture: Instruction Processing



Simple Processor Architecture

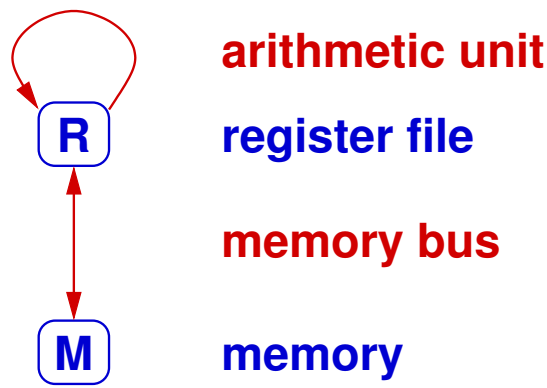


- Components
  - Instruction decoder and scheduler
  - Execution pipelines
  - Register file
  - Caches (L1-I, L1-D, L2)
  - External memory
- Memory architecture
  - From outside: von Neumann
    - General purpose memory
    - Common L2 cache
  - From inside: Harvard
    - Data L1 cache (L1-D)
    - Instruction L1 cache (L1-I)

Content

Information Exchange: Architectural Model

- **Machine** = Set of interconnected devices
  - Storage devices
  - Processing/transport devices
- **Storage devices**
  - Examples: Memory, register file, cache
  - Parameters: Storage size σ
- **Processing/transport devices**
  - Examples: Arithmetic pipeline, bus
  - Parameters: bandwidth/throughput β, startup latency λ
- **Graphical representation**
  - Vertices = Storage devices
  - Edges = Processing/transport devices



Information Exchange Function

- A computation implies that information is transferred from a storage device x to a storage device y.
- **Information Exchange Function**:

$$I^k_{x,y}(W) \quad = \quad \text{data transferred between computer sub-systems for specific computation } k$$

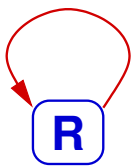
x ... source storage device (e.g. memory)
y ... destination storage device (e.g. register file)
W ... **problem size/work-load**

Information Exchange Model: Latency Predictions

Ansatz to predict latency:

$$\Delta t^k_{x,y} \simeq \lambda_{x,y} + I^k_{x,y}/\beta_{x,y}$$

Example:



arithmetic unit

register file

- $x,\ y = \mathrm{R}$
- $\beta_{R,R}$ = throughput arithmetic unit
- $I^k_{x,y}$ = number of input (or output) operands

Beware of limitations of this ansatz:

- Transfer mechanism may depend on task size N
- Bandwidth changes due to resource congestion is ignored
- ...

Information Exchange: BLAS1 (1/2)

- BLAS1 operation `sscal`: $x_i \leftarrow \alpha \cdot x_i$ $(i = 1, \ldots, N)$
  - x ... vector of single precision floating-point numbers
  - α ... single-precision floating-point number
- Information exchange:

multiply α and x	$I_{\text{fp}} = N \cdot 1\,\text{Flop}$
load x, store x	$I_{\text{mem}} = N \cdot (4+4)\,\text{Bytes}$

- Assume the following hardware parameters:

Floating-point unit throughput	$\beta_{\text{fp}} = 1\,\text{Flop/clock cycle}$
Memory bandwidth	$\beta_{\text{mem}} = 1\,\text{Byte/clock cycle}$

- Latency predictions (ignoring start-up latency):

Δt_{fp}	$N \cdot 1\,\text{clock cycles}$
Δt_{mem}	$N \cdot 8\,\text{clock cycles}$

Information Exchange: BLAS1 (2/2)

Latency for full operation:

- No overlap of memory load/store and arithmetic operations:

$$\Delta t(N) = \Delta t_{\mathrm{fp}} + \Delta t_{\mathrm{mem}} = N \cdot 9\,\mathrm{cycle}$$

- Perfect overlap of memory load/store and arithmetic operations:

$$\Delta t(N) = \max\left(\Delta t_{\mathrm{fp}}, \Delta t_{\mathrm{mem}}\right) = N \cdot 8\,\mathrm{cycle}$$

☞ **Memory bandwidth limited problem**

Information Exchange Model: BLAS3

- BLAS3 operation (DGEMM):

$$W_{ij} \leftarrow \sigma \cdot \sum_{k=1}^{N} U_{ik} \cdot V_{kj} + \rho \cdot W_{ij} \qquad (i, j = 1, \ldots, N)$$

U, V, W ... matrices of double precision floating-point numbers
σ, ρ ... double-precision floating-point numbers

- Information exchange

additions, multiplications	$I_{\mathrm{fp}} = (2 \cdot N^3 + 3 \cdot N^2)\,\mathrm{Flop}$
load $\{U, V, W\}$, store W	$I_{\mathrm{mem}} = (2 \cdot N^3 + 2 \cdot N^2) \cdot 8\,\mathrm{Bytes}$

Arithmetic Intensity

- **Arithmetic Intensity** $=$ [H. Harris, 2005]

$$\mathrm{AI} = \frac{\text{Number of floating-point operations}}{\text{Amount of transferred data}} = \frac{I_{\mathrm{fp}}}{I_{\mathrm{mem}}}$$

- Example: $x_i \leftarrow \alpha \cdot x_i$

$$\mathrm{AI} = \frac{1\,\text{Flop}}{8\,\text{Bytes}} \qquad \text{single-precision (SSCAL)}$$

$$\mathrm{AI} = \frac{1\,\text{Flop}}{16\,\text{Bytes}} \qquad \text{double-precision (DSCAL)}$$

Roofline Model (1/2)

[S. Williams et al., 2009]

- Floating-point and memory performance limits:

$$b_{\text{fp}} \leq B_{\text{fp}}, \quad b_{\text{mem}} \leq B_{\text{mem}}$$

  where $b = I/\Delta t$ is the observed performance and B the peak performance
- Upper limit for latency assuming perfect overlap of memory and arithmetic operations assuming the latency-bandwidth model to hold with zero start-up latencies:

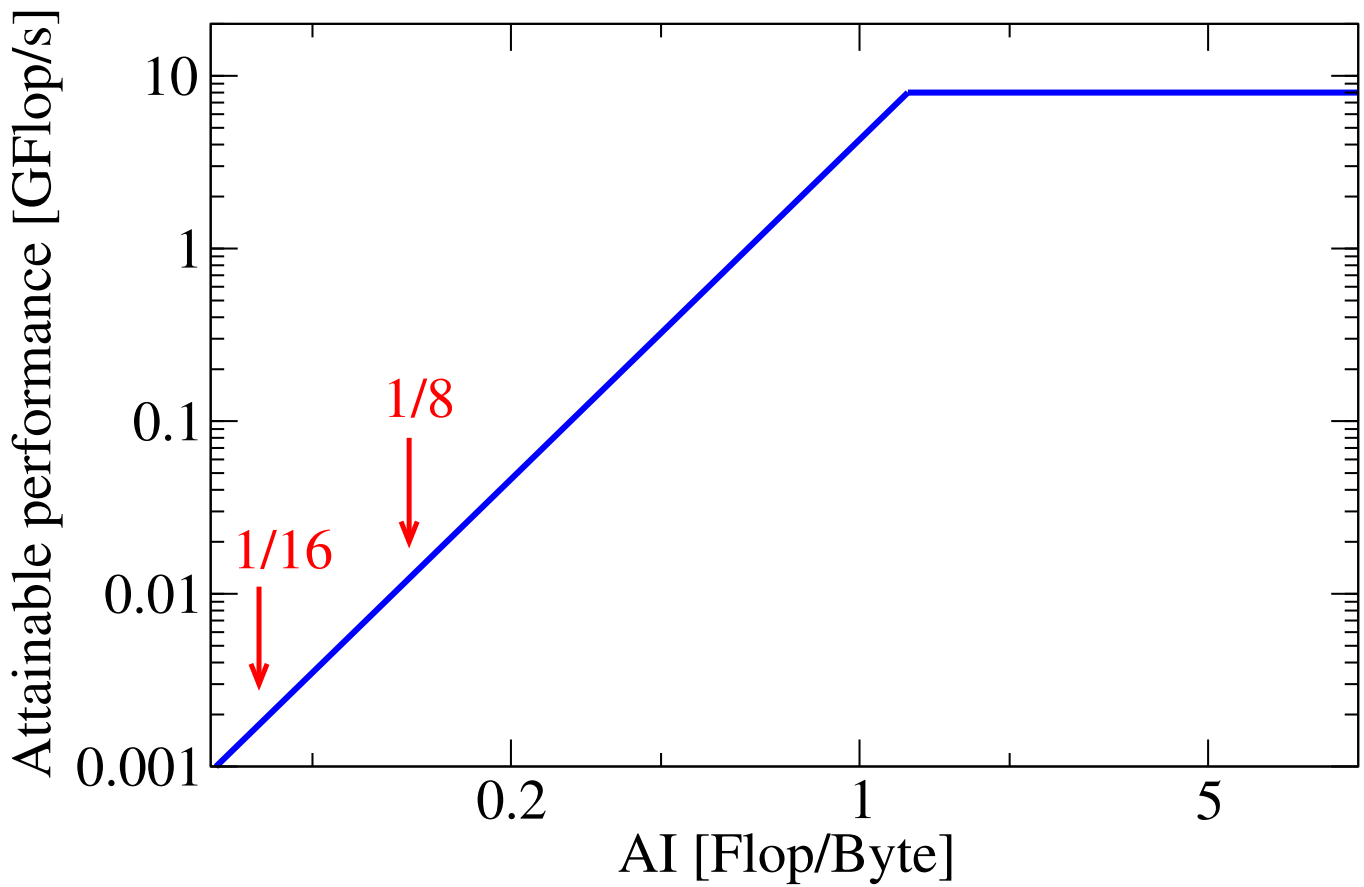
$$\Delta t = \max\left(\frac{I_{\text{fp}}}{b_{\text{fp}}}, \frac{I_{\text{mem}}}{b_{\text{mem}}}\right) \geq \max\left(\frac{I_{\text{fp}}}{B_{\text{fp}}}, \frac{I_{\text{mem}}}{B_{\text{mem}}}\right)$$

- Upper limit (=roof) for floating-point performance ("attainable performance")

$$b_{\text{fp}} = \frac{I_{\text{fp}}}{\Delta t} \leq \min\left(B_{\text{fp}}, \frac{I_{\text{fp}}}{I_{\text{mem}}} B_{\text{mem}}\right) = \min\left(B_{\text{fp}}, \text{AI} \cdot B_{\text{mem}}\right)$$

Roofline Model (2/2)

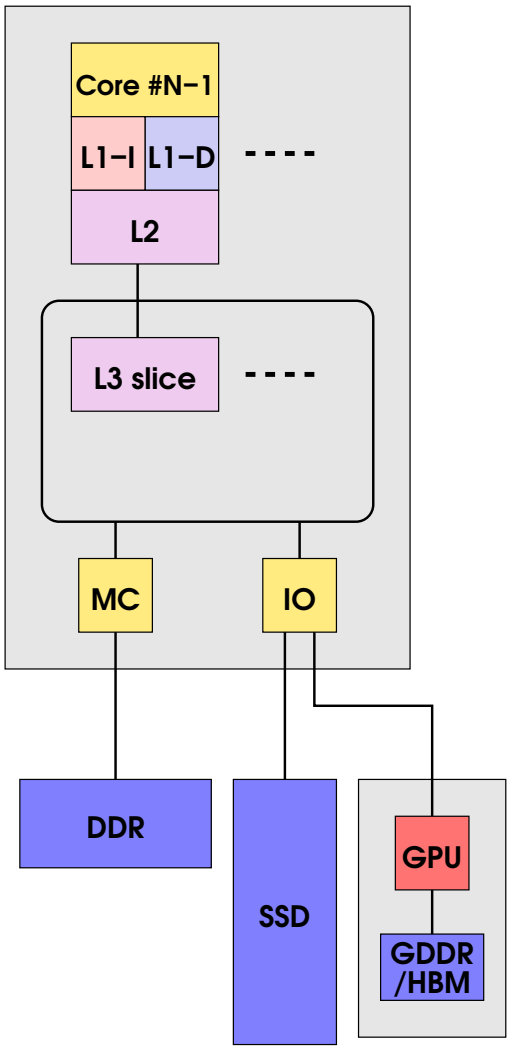
Example: $x_i \leftarrow \alpha \cdot x_i$



Content

Memory in Modern Node Architectures

- **Today's compute nodes: different memory types and technologies**
  - **Volatile memories**
    - Main memory (DDR3 or DDR4)
    - Caches (SRAM)
    - Accelerator memory (GDDR or HBM)
  - **Non-volatile memories**
    - SSD (NAND flash, 3D XPoint)
- **Different capabilities**
  - Bandwidth
  - Latency for single transfer
    - Access time
    - Cycle time
  - Capacity



Memory Access Locality

- Empirical observation: Programs tend to reuse data and instructions they have used recently
- Observation can be exploited to
  - Improve performance
  - Optimize use of more/less expensive memory
- Types of localities
  - **Temporal locality**: Recently accessed items are likely to be accessed in the near future
  - **Spatial locality**: Items whose addresses are near one another tend to be referenced close together in time

Memory Access Locality: Example

```
double a[N][N], b[N][N];

for (i=0; i<N; i++)
  for (j=1; j<N; j++)
    a[i][j] = b[j-1][0] + b[j][0];
```

- Assume right-most index being fastest running index
- Temporal locality: `b[j][0]`
- Spatial locality: `a[i][j]` and `a[i][j+1]` ($j + 1 < N$)

Memory: Cache

Cache miss = Word not found in cache

- Word has to be fetched from next memory level
- Typically a full **cache line** (=multiple words) is fetched

Cache organisation: **Set associative cache**

- Match line onto a set and then place line within the set
  - Choice of set address: (address) mod (number of sets)
- Types:
  - **Direct-mapped cache**: 1 cache line per set
    - Each line has only one place it can appear in the cache
  - **Fully associative cache**: 1 set only
    - A line can be placed anywhere in the cache
  - **n-way set associative cache**: n cache lines per set
    - A line can be placed in a restricted set of places (1 out of n different places) in the cache

Cache misses

- Categories of cache misses

  Compulsory: Occurs when cache line is accessed the first time

  Capacity: Re-fetch of cache lines due to lack of cache capacity

  Conflict: Cache line was discharged due to lack of associativity

- **Average memory access time** $=$

$$\text{hit time} + \text{miss rate} \cdot \text{miss penalty}$$

Hit time ... Time to hit in the cache

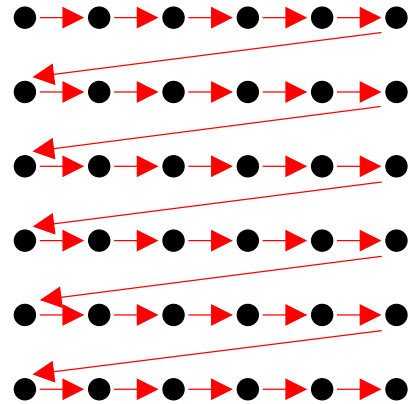
Miss penalty ... Time to load cache line from memory

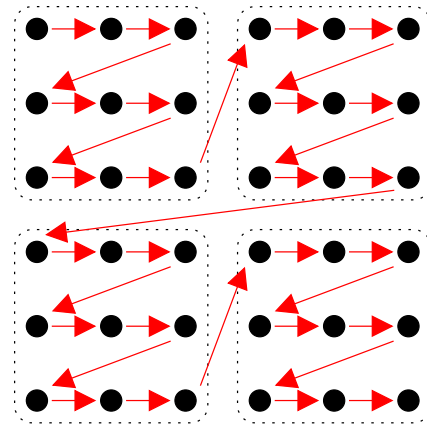
Exploiting Caches: Cache Blocking

- **Cache blocking** is a strategy where data structures or memory access patterns are changed such that capacity cache misses are reduced
  - Optimisation strategy sometimes also called **loop tiling** or **loop blocking**
- Example: DGEMM

$$W_{ij} \leftarrow \sigma \cdot \sum_{k=1}^{N} U_{ik} \cdot V_{kj} + \rho \cdot W_{ij}$$

  - Advantage: U and V needs to be loaded only once per block
  - Disadvantage: Non-linear read of U, V





Cache-blocked DGEMM

- Let B be the block size with $B \leq N$ and N being a multiple of B
- Information exchange analysis without blocking:

$$\begin{aligned} I_{\text{fp}}(N) &= (2 \cdot N^3 + 3 \cdot N^2)\,\text{Flop} \\ I_{\text{mem}}(N) &= \left(2 \cdot N^3 + 2 \cdot N^2\right)\,8\,\text{Byte} \\ AI &= I_{\text{fp}}/I_{\text{mem}} \simeq (1/8)\,\text{Flop/Byte} \end{aligned}$$

- Information exchange analysis with blocking:

$$\begin{aligned} I_{\text{fp}}(N,B) &= (2 \cdot N^3 + 3 \cdot N^2)\,\text{Flop} \\ I_{\text{mem}}(N,B) &= \left[2 \cdot \left(\frac{N}{B}\right)^2 \cdot \left(\frac{N}{B}\right) \cdot B^2 + 2 \cdot N^2\right] \cdot 8\,\text{Byte} \\ &= \left[2 \cdot (N^3/B) + 2 \cdot N^2\right] \cdot 8\,\text{Byte} \\ AI &= I_{\text{fp}}/I_{\text{mem}} \simeq (B/8)\,\text{Flop/Byte} \end{aligned}$$

Content

Instruction Set Architecture

Typically supported instruction can be categorised as follows:

Operator type	Examples
Data transfer	Load and store operations
Arithmetic and logic	Integer arithmetics, bitwise operations, compare operations
Control	Branch, jump, procedure call and return
Floating point Mathemetical functions	Floating point arithmetics Inverse, square root

Instruction Level Parallelism (ILP)

- **ILP** = Parallelism among instructions from small code areas which are independent of one another
- Exploitation of ILP
  - Overlapping of instructions in a pipeline
  - Parallel execution of instructions in multiple functional units
- Modern processors support issuing of several instructions per cycle
  - Example: superscalar processors
- Optimisation strategy: Reduce number of instructions by performing multiple operations per instruction

SIMD Parallelism

- Single Instruction Multiple Data (SIMD) instructions exploit data-level parallelism by operating on data items in parallel
  - E.g., SIMD add

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

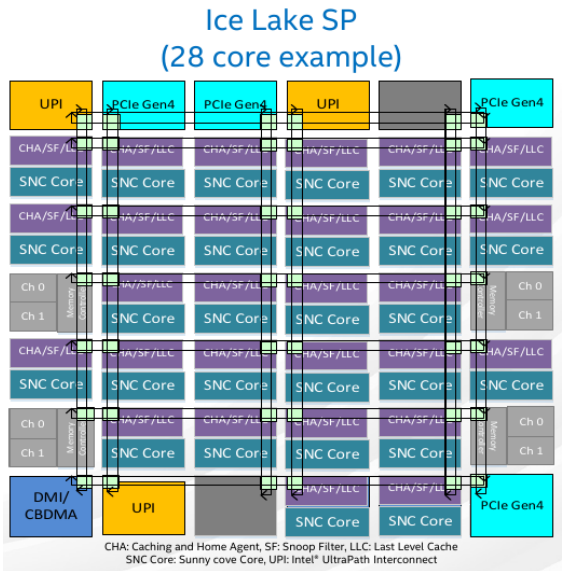
- Example ISA:
  - Intel Streaming SIMD Extensions (SSE)
  - POWER ISA (VMX/AltiVec, VSX)
  - ARMv7 NEON, ARMv8, ARM SVE
  - Intel Advanced Vector Extensions:
    - AVX/AVX2: 256 bit
    - AVX512: 512 bit

Content

x86 Architecture: Intel Xeon Platinum (1/2)

Hardware parameters of Intel Platinum 8358 (Ice Lake)

Number of cores	32
Base core clock	2.6 GHz
Max. core clock	3.4 GHz
L1 cache (data+instr)	48+32 kBytes/core
L2 cache	1280 kBytes/core
L3 cache	48 MBytes
Memory technology	DDR4
Number of channels	8
Max. memory bandwidth	205 GBytes/s
Peak DP performance	32 Flop/cycle/core
Max. TDP	250 Watt
Lithography	10 nm
Release date	Q1/2021

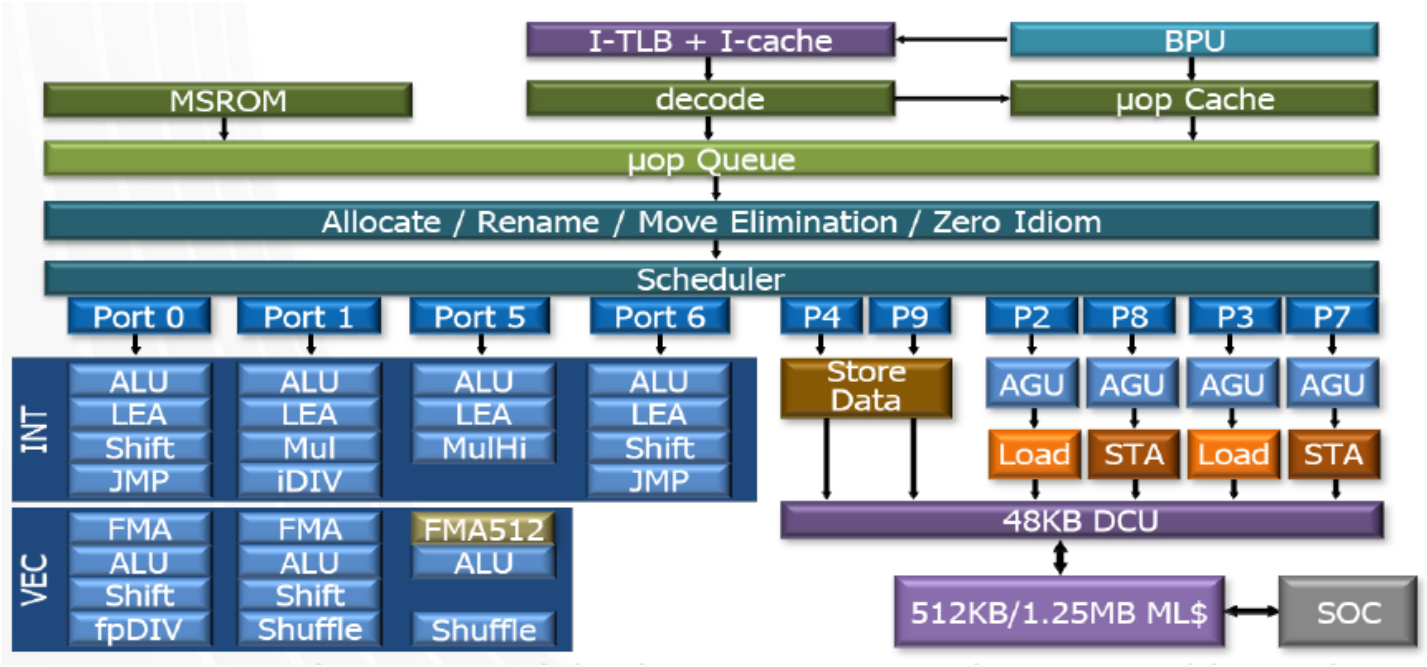


[Intel, Hotchips, 2020]

Nominal performance numbers as specified by manufacturer

x86 Architecture: Intel Xeon Platinum (2/2)

Intel Platinum 8358 (Ice Lake) micro-architecture:

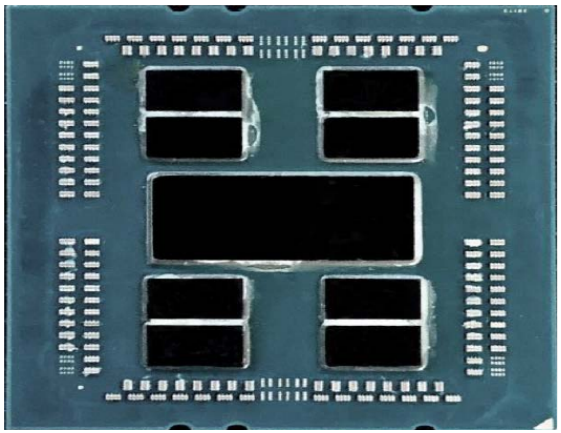


[Intel, Hotchips, 2020]

x86 Architecture: AMD EPYC

Hardware parameters of AMD EPYC 7742 (Rome)

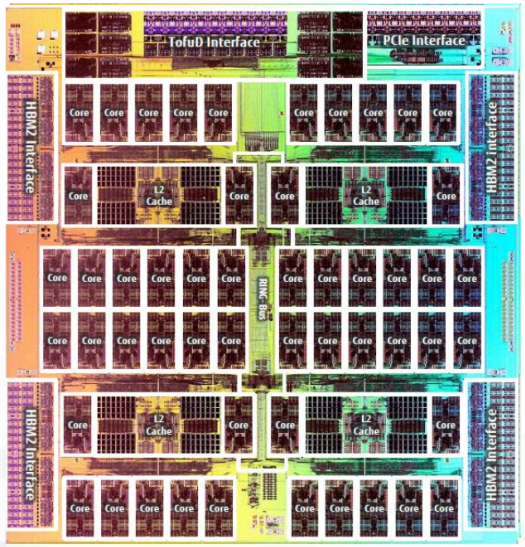
Number of cores	64
Base core clock	2.25 GHz
Max. core clock	3.4 GHz
L1 cache (data+instr)	32+32 kBytes/core
L2 cache	512 kBytes/core
L3 cache	256 MBytes
Memory technology	DDR4
Number of channels	8
Max. memory bandwidth	205 GBytes/s
Peak DP performance	16 Flop/cycle/core
Max. TDP	225 Watt
Lithography	7/14 nm
Release date	2019

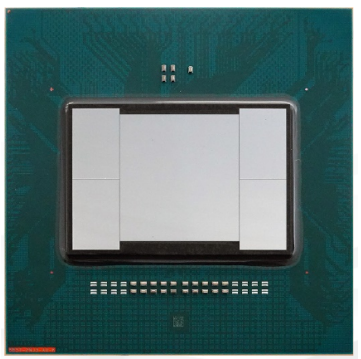


[AMD]

Fujitsu's A64FX Processor

- ARMv8 + Scalable Vector Extension (SVE)
  - SVE width: 512 bit
- Number of cores: $4 \times (12 + 1)$
- Normal/boost frequency:
  2.0 GHz/2.2 GHz
- Cache parameters
  - L1-D/L1-I: 64/64 kiByte per core, 4-way
  - L2: 8 MiByte per (12+1) cores, 16-way
  - Cache line size: 256 Byte
- Performance figures
  - Floating-point performance:
    2×16 FP64/cycle/core
  - HBM memory bandwidth: 1024 GByte/s



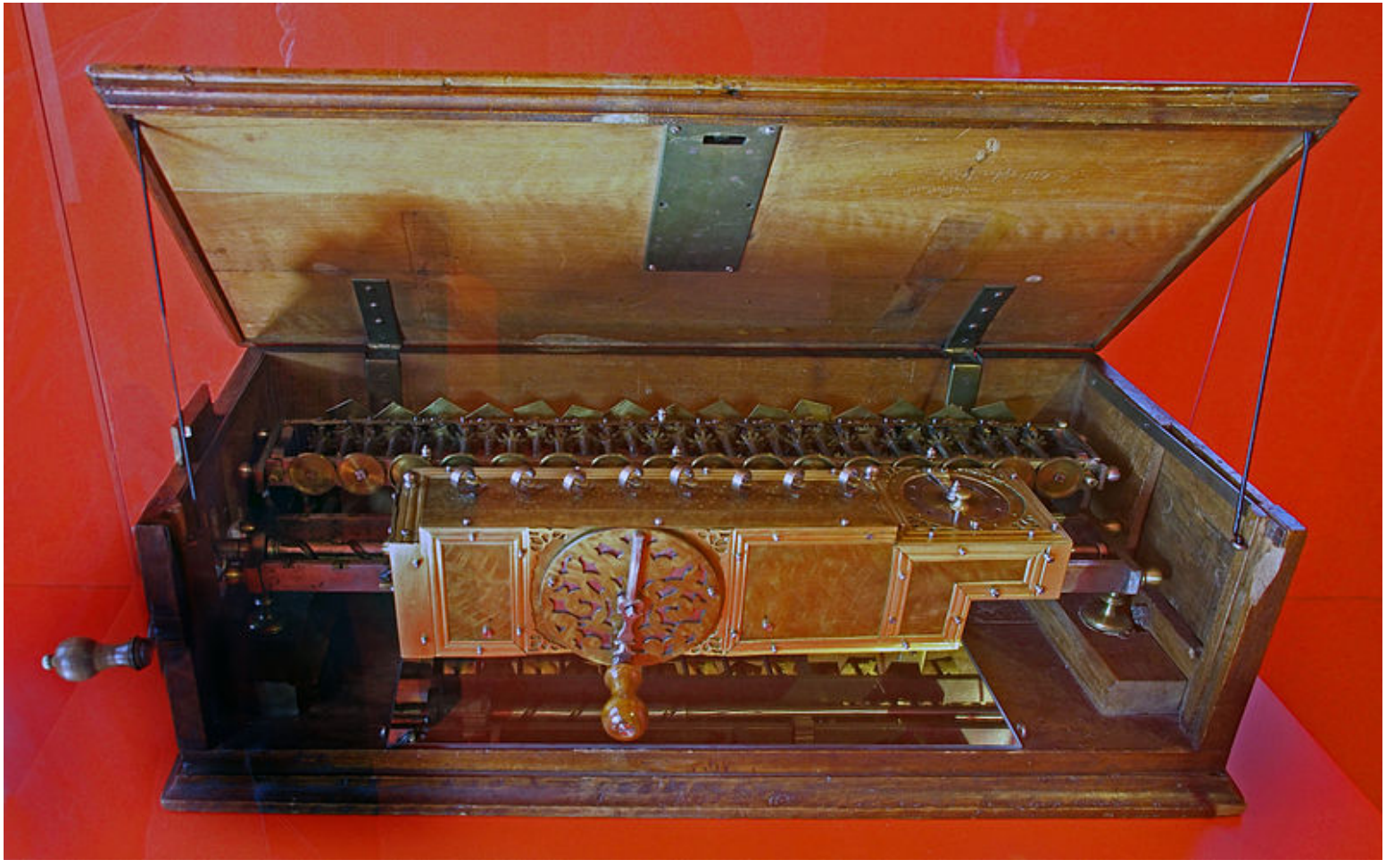


Content

Summary

- Modern processors feature a significant amount of parallelism
  - Examples:
    - 32 . . . 64 cores
    - SIMD instructions with operands of 256 . . . 512 bits
  - Need suitable programming models to exploit parallelism
    - Multi-threading programming models like OpenMP
    - Auto-vectorising compilers, SIMD intrinsics, . . .
- Compute capabilities versus data transport capabilities are typically relatively large
  - Methodologies: information exchange analysis and roofline model
  - Need algorithms and suitable data layouts that allow to exploit data locality using available caches

Finish with a Simple Architecture: Leibniz' Reckoner



[Museum Schloss Herrenhausen]