Data Processing on Modern Hardware

Introduction

Jana Giceva

What this course is about

- ***Make*** **programs run faster on modern multi-core CPUs using a variety of techniques:**
  - Optimizing the data structures and algorithms for the memory hierarchy
  - Vectorization
  - Parallelizing algorithms
  - Efficient synchronization of data structures

- ***Learn*** **how to best leverage *new* hardware technologies**
  - Accelerators (e.g., FPGAs, modern NICs, etc.)
  - Low-latency high-bandwidth networks
  - Non-volatile memory

- ***Apply*** **the knowledge with hands-on work:**
  - Understand what is happening under the hood, *i.e.*, in the CPU,
  - Do performance analysis and debugging, and
  - Optimize your algorithms and data structures to run well both in isolation and alongside other programs

A motivating example (memory access)

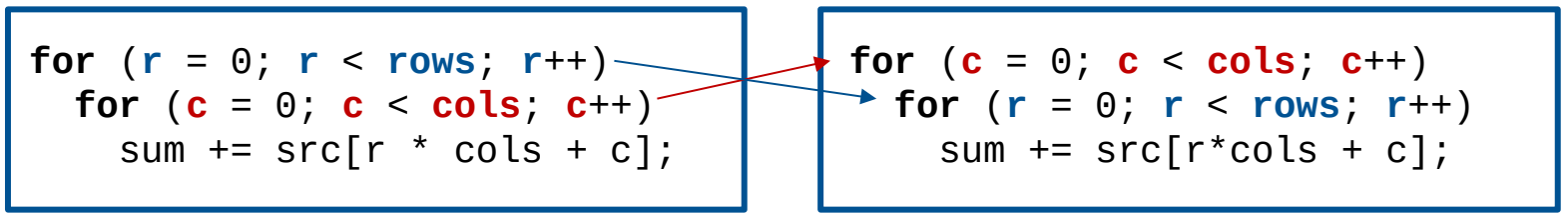
Task: sum up all entries in a two-dimensional array

Alternative 1:

```
for (r = 0; r < rows; r++)
  for (c = 0; c < cols; c++)
    sum += src[r * cols + c];
```

Alternative 2:

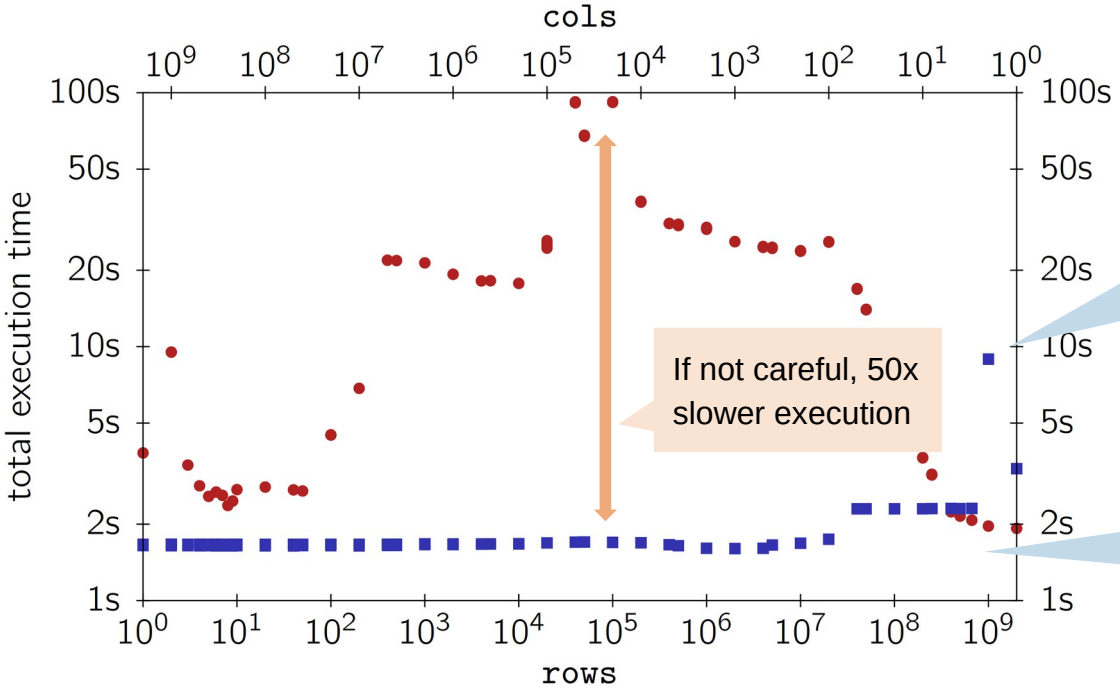
```
for (c = 0; c < cols; c++)
  for (r = 0; r < rows; r++)
    sum += src[r*cols + c];
```



Both alternatives touch the same data, but in different order.

A motivating example (memory access)

Task: sum up all entries in a two-dimensional array



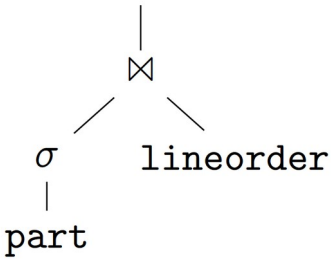
Alternative 2 iterates over the elements column-wise, and its performance is highly dependent on whether the working set size fits in the memory hierarchy.

Alternative 1 iterates over the elements row-wise, which is more friendly to the underlying micro-architectural features.

A motivating example (multi-core)

Task: run multiple parallel instances of the query

```
SELECT SUM(lo_revenue)
FROM   part, lineorder
WHERE  p_partkey = lo_partkey
AND    p_category <= 5
```



To implement the join () use either:

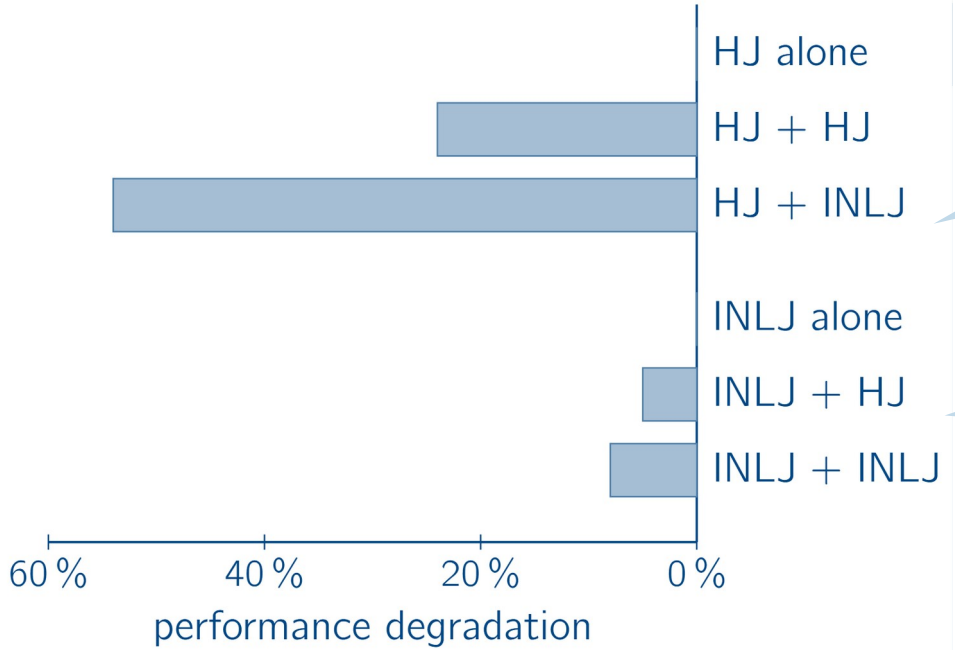
- a **hash join** or
- an **index nested loops join**

Co-execute the independent instances on different CPU cores and compare performance to baseline when they are run in isolation.

Results from “Lee, Ding, Chen, Lu, and Zhang. MCC-DB: Minimizing Cache Conflicts in Multi-core Processors for Databases” *VLDB 2009*

A motivating example (multi-core)

Task: co-run independent join algorithms on different CPU cores



Some algorithms are more sensitive to noisy environments (victims) and their performance can be significantly affected if collocated with a bad neighbor.

One can either design algorithms which are robust, or leverage novel hardware features like Intel's Resource Directory Technology (RDT) for resource allocation and perf. isolation.

Results from "Lee, Ding, Chen, Lu, and Zhang. MCC-DB: Minimizing Cache Conflicts in Multi-core Processors for Databases" *VLDB 2009*

A motivating example (accelerators)

Task: run the following regular-expression queries

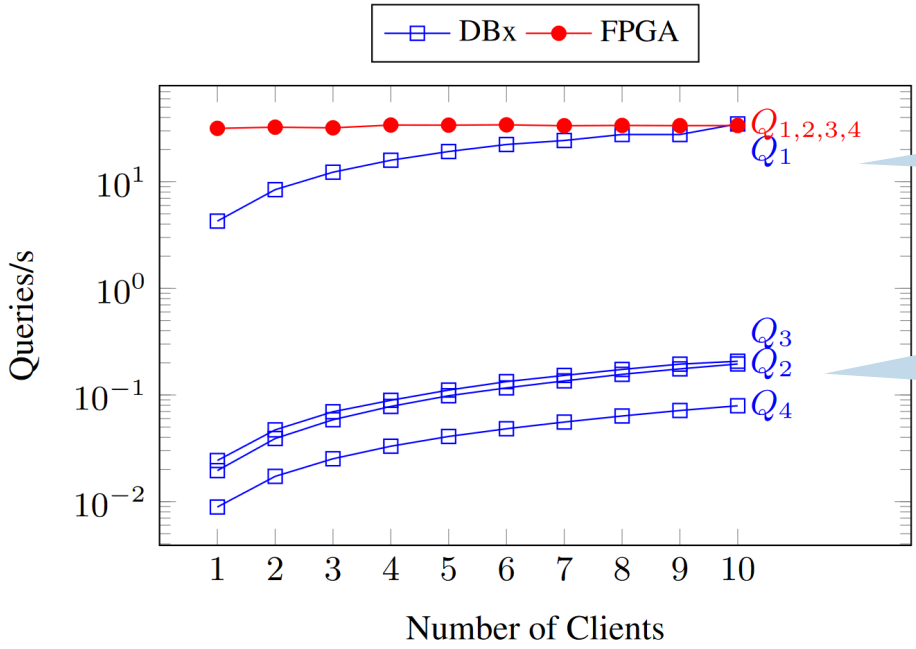
```
Q1: SELECT count(*) FROM address_table
    WHERE address_string LIKE '%Strasse%';
Q2: SELECT count(*) FROM address_table
    WHERE REGEXP_LIKE(address_string, '(Strasse|Str.\.).*(8[0-9]{4})');
Q3: SELECT count(*) FROM address_table
    WHERE REGEXP_LIKE(address_string, '[0-9]+(USD|EUR|GBP)');
Q4: SELECT count(*) FROM address_table
    WHERE REGEXP_LIKE(address_string, '[A-Za-z]{3}\:[0-9]{4}');
```

And compare the performance of CPU-only vs. FPGA-enhanced DBMS.

Results from "Sidler, Istvan, Owaida, Alonso. Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures" *SIGMOD 2017*

A motivating example (accelerators)

Task: run the following regular expression queries



but are a great match for accelerators like FPGAs.

Regular expression matching is notoriously expensive to evaluate on CPUs,

Results from “Sidler, Istvan, Owaida, Alonso. Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures” *SIGMOD 2017*

Course content

- **Cache awareness**
  - How to place data in memory and access it in a way that makes optimal use of memory caches?
- **Query execution**
  - How can we tune our algorithms to fit modern processor architectures?
- **Multi-core architectures**
  - How can we exploit the parallelism provided by multicore architectures?
  - What should we be careful of? Synchronization? NUMA? Interference?
- **Specialized hardware**
  - When should we use accelerators and how to choose the right one for data processing?
- **Working with modern network and storage technologies**
  - Low-latency high-bandwidth network, novel memory fabrics, computational storage?

Course Organization

Lecture:

- Slides uploaded by 12pm. Check the lecture's Moodle.
- In person lectures on Wednesdays (E.126, IMETUM, 5701.EG.026), 13-14.30
- No streaming, no recorded lectures. Recorded lectures from prior years can be found on Moodle.
- Please check Moodle regularly for updates

Tutorials:

- In person on Wednesdays (MI Horsaal 2, 5604.EG.011), 14.30-16 (after the lecture but in a different room)
- First session: today in-person introduction, Q&A session and general set-up
- Slides, homework assignments, etc. will be communicated via Moodle and Gitlab.
- Additional discussions on Mattermost. Please join. Details in Moodle.
- Consider that exercise material is part of the course (and exam) content!

Assessment (Homework + Exam)

- The main goal of the course is doing the exercises and understanding the material, but there will be an exam.
- You will get a grade for the assignments, which improves the grade of the exam.
  - 1 exam point for each homework (max 5)
- There will be a 5-week long project at the end of the semester (no teaching, just guest lectures)
  - Up to 5 exam points for the project
- The Exam: TBD
  - No retake will be offered.
  - Written, 90min.
  - The homework/project bonus is applicable only if you pass the exam.
- The project:

Course set-up

Let’s try to make the course as interactive as possible.

- During the lecture and tutorials, please speak-up, ask questions and discuss!
- Engage in asynchronous discussions on Mattermost
- Send questions you want to be addressed during the tutorial sessions

The material we discuss is relevant in practice:

- We will provide examples and exercises
- You will achieve the maximum fun factor if you try using the techniques on **your machine**

Course material

This is not a standard course – it is state of the art (bleeding edge) systems and research

- There is **no real textbook** for this course, only for the basics

- **Computer architecture basics** are covered in
  - “Computer Architecture: A Quantitative Approach” by Hennessy and Patterson.
  - “Computer Systems: A programmer’s perspective” by Bryant and O’Hallaron

- The **lecture slides** are available **online**

- Most **material** that we are going to cover is **taken out of research papers:**
  - The references to those papers will be given as we go
  - They are all good and easy (and fun!) to read papers.

- We’ll invite **industry speaker**(s) to share their experiences towards the end of the course.

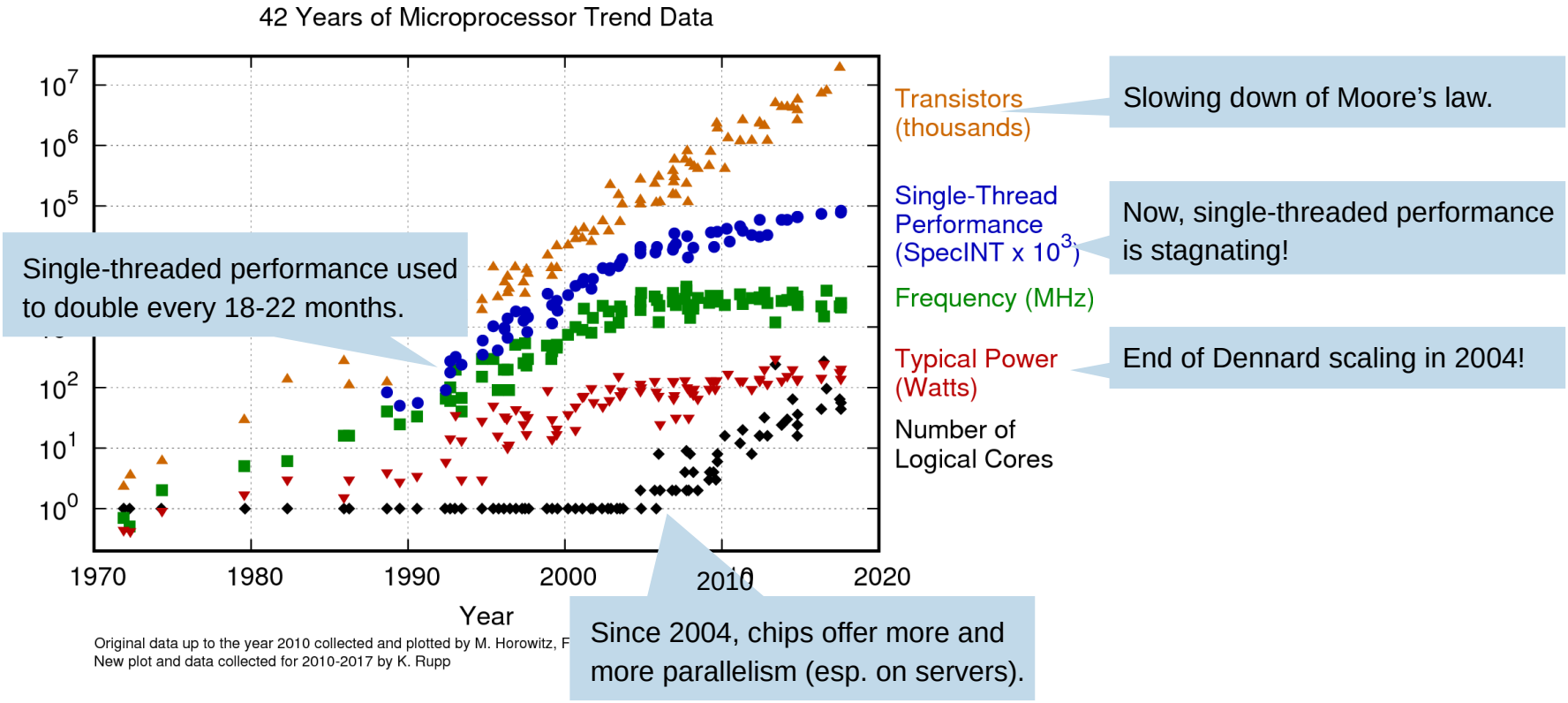
Data Processing on Modern Hardware

Introduction

Jana Giceva

Hardware trends

Hardware trends



Slowing down of Moore's law.

Now, single-threaded performance is stagnating!

End of Dennard scaling in 2004!

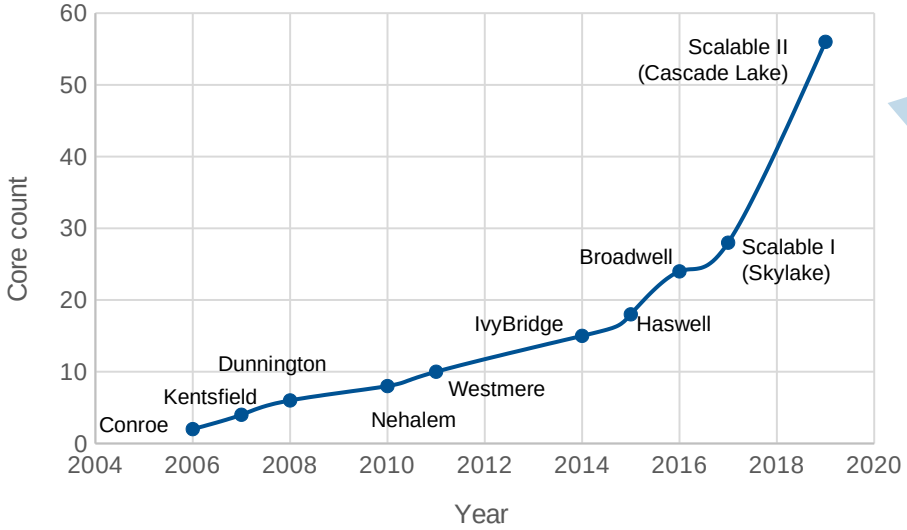
Single-threaded performance used to double every 18-22 months.

Since 2004, chips offer more and more parallelism (esp. on servers).

Effect 1: Increasing parallelism

- Since the ***end of Dennard scaling*** in 2004, historic switch for the microprocessor industry to ***increase core count*** instead of single processor's efficiency

- Go from relying solely on instruction level parallelism (ILP) to ***data-level parallelism*** (DLP) and ***thread-level parallelism*** (TLP).



Major implications for software:

- Previously hardware and compiler could ***implicitly*** exploit ILP without the programmer's attention and help.

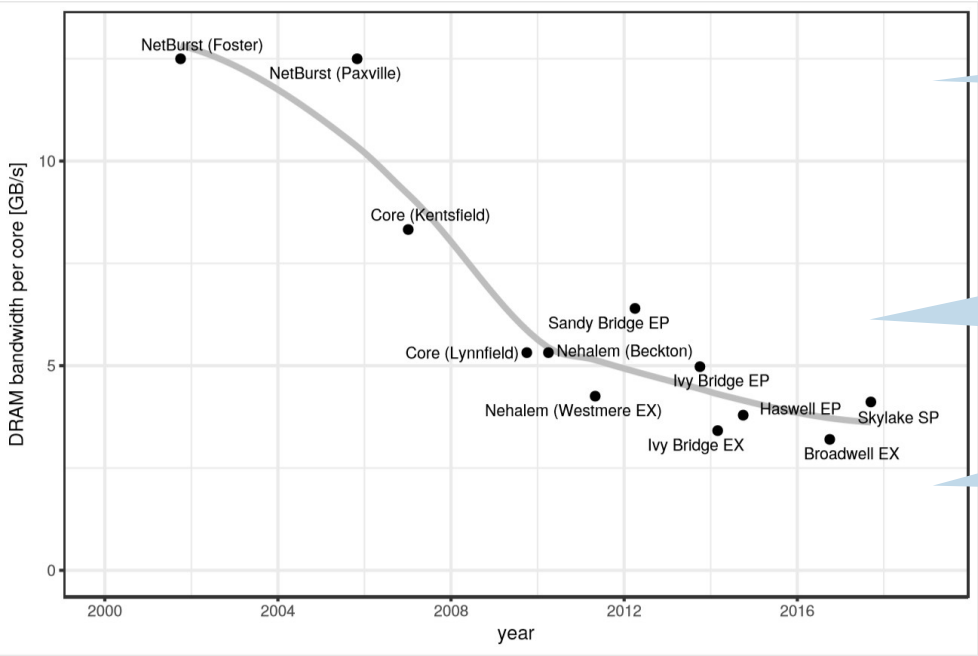
- TLP and DLP, however, are ***explicitly*** parallel and require restructuring of the application so that it can exploit the parallelism

Effect 1: increasing parallelism

- In addition to core count, which exploits thread-level parallelism (TLP)
- The issue width of vectorized instructions has significantly increased to exploit data-level parallelism (DLP) within general purpose processors
- Intel's SIMD (Single Instruction Multiple Data) evolution summarized:
  - 1997: MMX 64-bit (Pentium 1),
  - 1999: SSE 128-bit (Pentium 3),
  - 2011: AVX 256-bit `float` (Sandy Bridge)
  - 2013: AVX2 256-bit `int` (Haswell),
  - 2017: AVX-512 512-bit (Skylake)
  - 2025: AVX-1024 1024-bit (Diamond Rapids)?
- And the increase of popularity and use of Vector and GPU architectures

Effect 2: hitting the Memory wall

- The memory bandwidth does not increase as fast as the core count



Memory has become the new disk

Need for multi-tier cache hierarchy, and

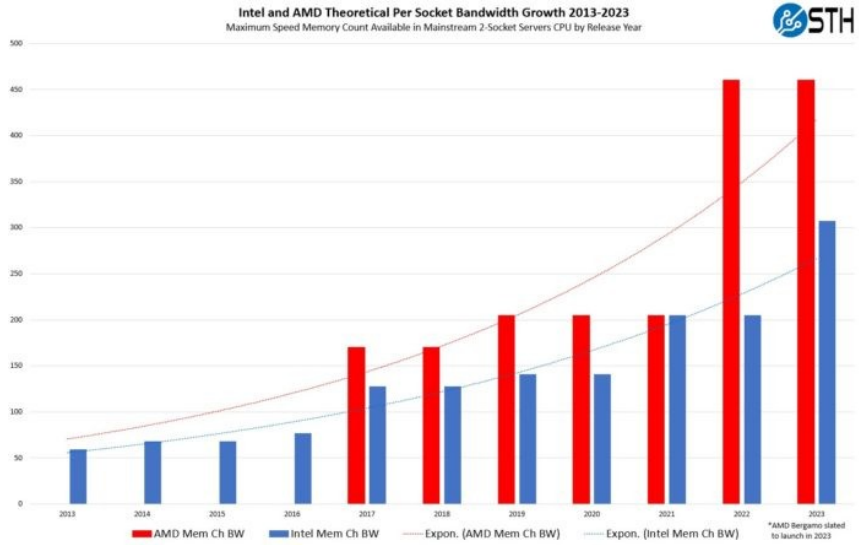
Careful use of registers and the cache hierarchy

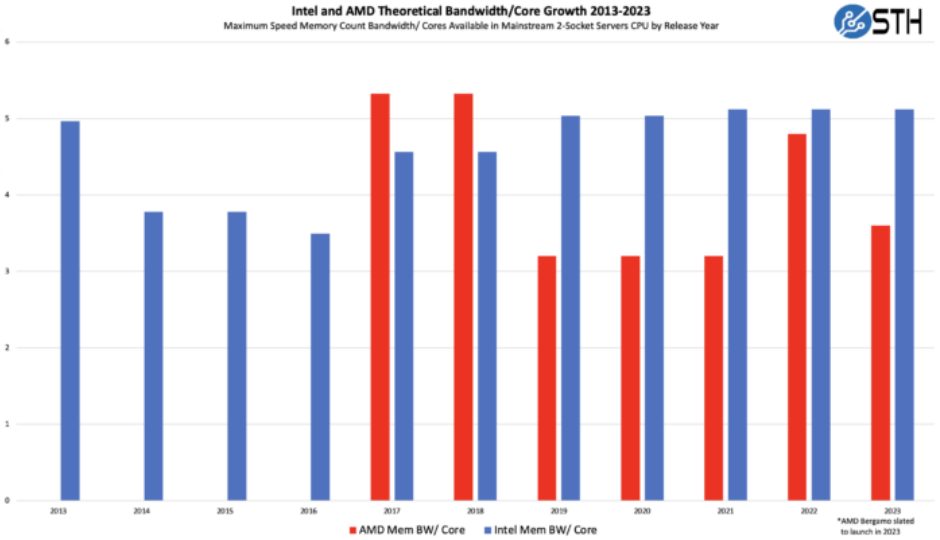
But, caches bring their own benefits and pitfalls:
Pollution, MESI protocol, timing attacks

img src: http://db.in.tum.de/teaching/ws1718/dataprocessing/chapter1.pdf by Dr. Viktor Leis

Effect 2: hitting the Memory wall

- The memory bandwidth does not increase as fast as the core count





img src: http://db.in.tum.de/teaching/ws1718/dataprocessing/chapter1.pdf by Dr. Viktor Leis

Effect 3: towards Hardware Specialization

Dark Silicon:

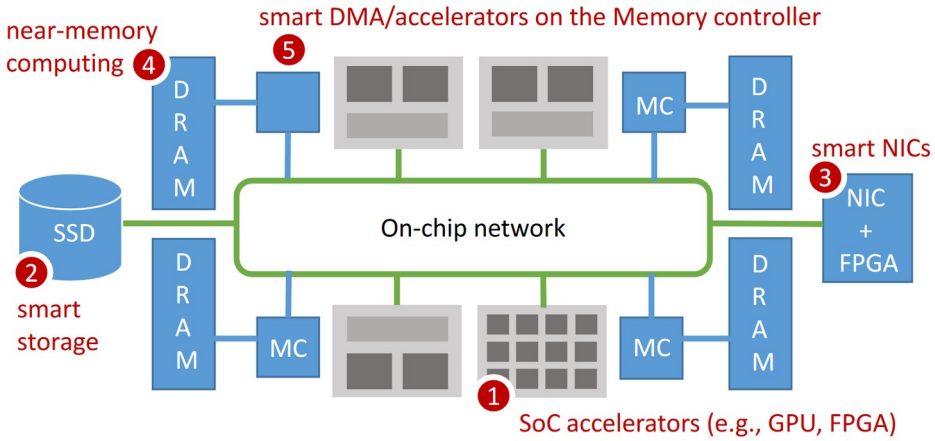
- The end of Dennard scaling
  - Modern CPUs already have close to 10 billion transistors, impossible to power all at the same time
- *Alternative 1*: more cores, but at lower frequency
  - *e.g.*, Intel's Xeon Phi. *But*, Amdahl's law
- *Alternative 2*: many specialized, heterogeneous cores and function units

Domain specific architectures (DSAs)*

- GPUs, TPUs, NPUs, FPGAs, ASICs, etc.

Active hardware:

- smart storage, smart NICs, programmable switches, processing-in-memory, smart DMA engines, programmable memory controllers?

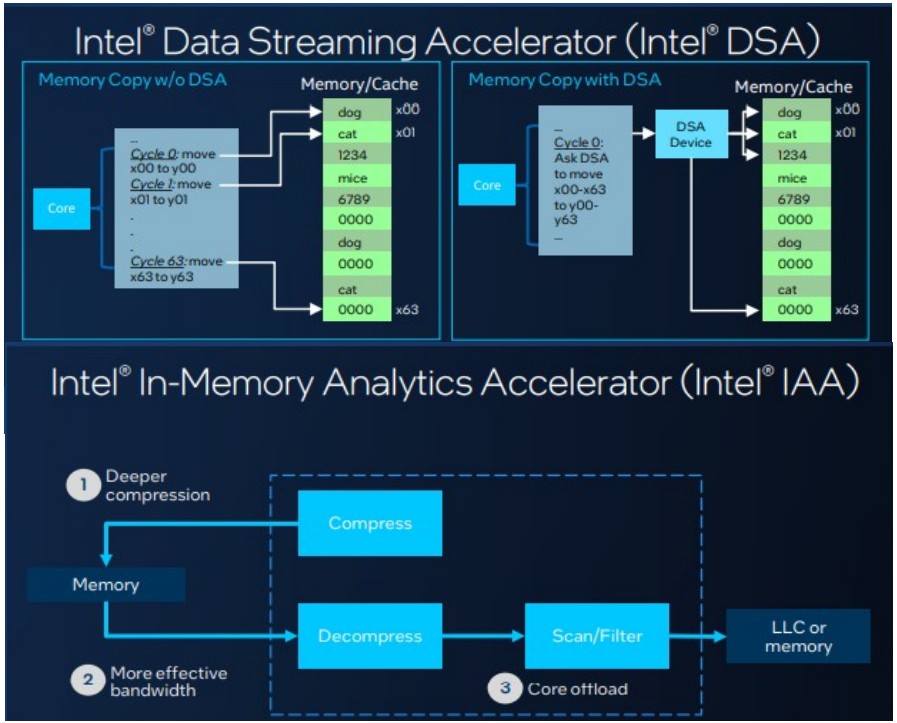


* https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

Example hardware specialization today

- **Historical: Only specialized HW before rise of general purpose**
  - Gamma database machine, Lisp machine

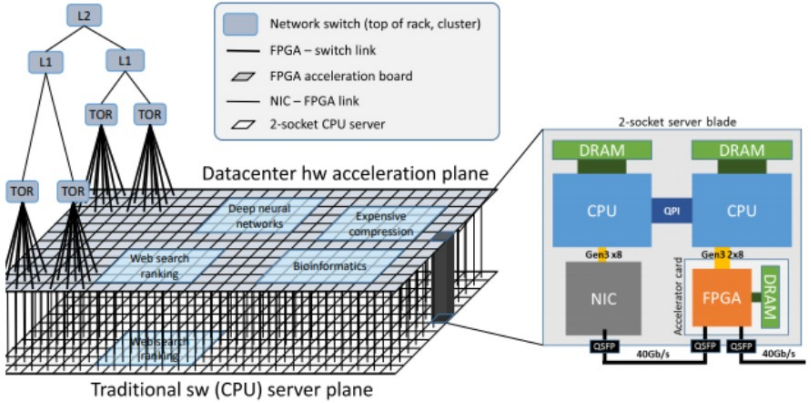
- **Today on the market:**
  - Special instructions: AES encryption, decoders
  - Graphics Processing Units (GPUs)
  - Accelerated Processing Units (APU, by AMD)
  - Intel's Sapphire Rapids built-in accelerators
  - Oracle Sparc T7, M7 (DAX)
  - Google's Tensor Processing Unit (TPU)
  - Field-Programmable Gate Arrays (FPGAs)
  - SSDs with embedded FPGA or ARM cores
  - Programmable switches with P4
  - FPGA enhanced NICs



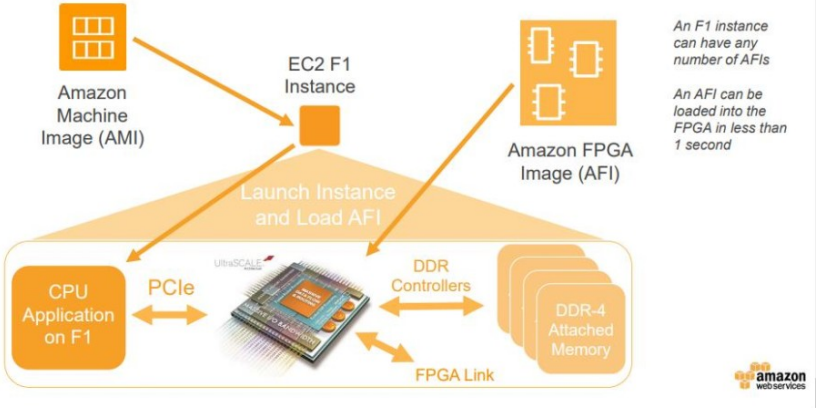
- **Also today:** many ongoing research and industry projects

Accelerators are transforming the Cloud

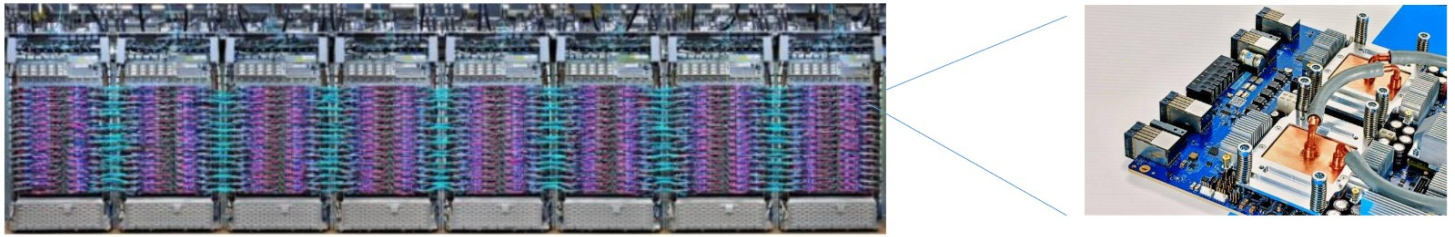
Microsoft's Azure configurable cloud (MICRO'16)



Amazon's FPGA-acceleration using F1 (HotChips'17)



Google;s TPU 4.0 pods (Google I/O 2021)



Implications for (future) system design

- **How do we program them?**
  - What is the interface?
  - Domain specific languages?
- **What is the role of compilers?**
- **How do we decide which computation to offload? Optimizers?**
- **Who manages the accelerators/active hardware (e.g., OS, application, runtime)?**
  - Should they be context-switched?
  - How do we virtualize them?
  - How do they fit in data-center resource disaggregation picture?
  - Good match for serverless functions?
- **What is the failure domain?**

AMD EPYC Genoa

AMD EPYC 9004 Series Processor (9654P):

- Zen 4 Architecture
- CPU clocked at 2.4 GHz (with 96 cores per SoC, 192 threads)
- With up to 6TB DRAM (DDR4800) at 460GB/s using 128 PCIe 5.0 lanes
- Large 384 MB LLC

Advanced technologies:

AMD Infinity Guard Features

- CXL1.1+ memory support (CXL “type 3/Memory”)
- Tiered memory management
- Advanced security (secure encrypted virtualization, multi-key encryption, memory encryption)

https://www.storagereview.com/review/4th-gen-amd-epyc-review-amd-genoa

Check more details at https://www.amd.com/en/campaigns/epyc-9004-architecture

Beyond CPU trends and Accelerators

- **CXL: Compute Express Link**
  - Open standard CC interconnect between CPUs and accelerators, smart NICs, memory
  - Three classes of devices:
    - Type 1: accelerators such as smart NICs (typically lacking local memory), communicate with the host's DDR memory
    - Type 2: GPUs, ASICs, FPGAs (all equipped with DDR or HBM) and can use CXL to share memory
    - Type 3: memory devices can be attached via CXL to provide additional bandwidth and capacity

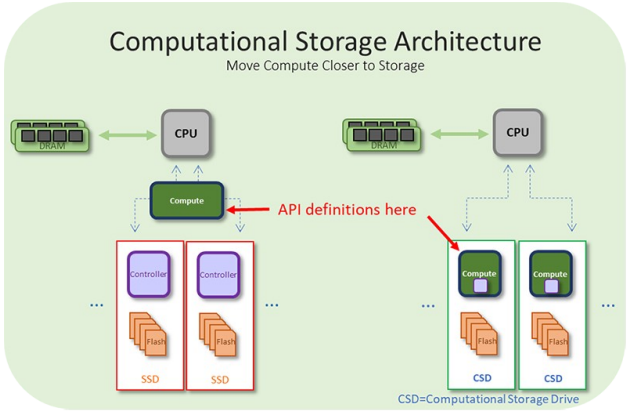
src: https://www.rambus.com/blogs/compute-express-link/

- **Low-latency high-bandwidth networks**
  - InfiniBand (IB) is a networking standard used in HPC that has very high throughput and low latencies
  - Already 400Gbit Ethernet deployed in data-centers

Src: https://osi.rosenberger.com/news-media/400-gigabit-800g-ethernet/

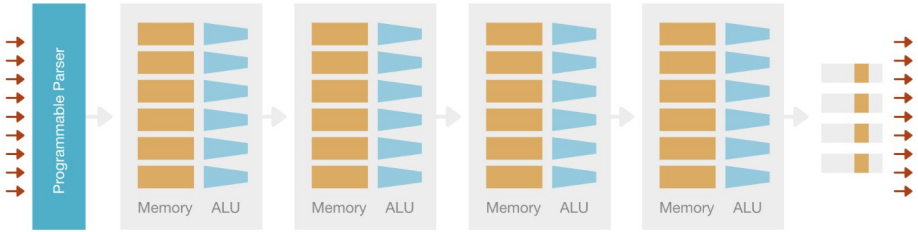
Beyond CPU trends and Accelerators

- **Computational Storage**
  - Architectures that performs computation where data is stored, offloading host processing and reducing data movement.
  - Integration of compute resources, near the storage or between the host and storage



Src: https://www.snia.org/education/what-is-computational-storage

- **In-network computing**
  - via Programmable Switches (e.g., using P4)
  - Industry vendors: Barefoot Tofino, Cavium Xpliant, Cisco Quantum Flow, etc.



src: https://barefootnetworks.com/products/brief-tofino/

References

In addition to cross-references provided in the slides

Some material based on:

- Lecture notes by Prof. Viktor Leis and Prof. Jens Teubner
- Research talks and papers from DaMoN, HotChips, SIGMOD, VLDB, ADMS, MICRO, ISCA, etc.

Interesting videos:

- *A New Golden Age for Computer Architecture*, a Turing-award lecture by Patterson and Hennessy
- *Clouds, catapults and life after the end of Moore's Law* with Dr. Doug Burger – Microsoft Research

Useful material in general for the course at:

- Intel's *Software Developer's Manuals*
- Intel's *Top-Down Micro-architectural Analysis Method*
- Anger Fog's *Software optimization resources*
- Ulrich Drepper's *What every Programmer needs to know about Memory*
- Godbolt – the compiler explorer (https://godbolt.org/)