

Data Processing on Modern Hardware

Introduction

Jana Giceva



About me



Jana Giceva

Chair for Database Systems

Boltzmannstr. 3, Office: 02.11.043

jana.giceva@in.tum.de

Academic Background:

- 2006 – 2009 BSc in Computer Science at *Jacobs University Bremen*
- 2009 – 2011 MSc in Computer Science at *ETH Zurich*
- 2011 – 2017 PhD in Computer Science at *ETH Zurich* (topic: DB/OS co-design)
- 2017 – 2019 Lecturer in Department of Computing at *Imperial College London*
- Since 2020 Assistant Professor for Database Systems at *TUM*

Connections with Industry:

- Held roles with *Oracle Labs* and *Microsoft Research* in the USA in 2013 and 2014
- PhD Fellowship from *Google* in 2014
- Early Career Faculty Award from *VMware* in 2019



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];
```

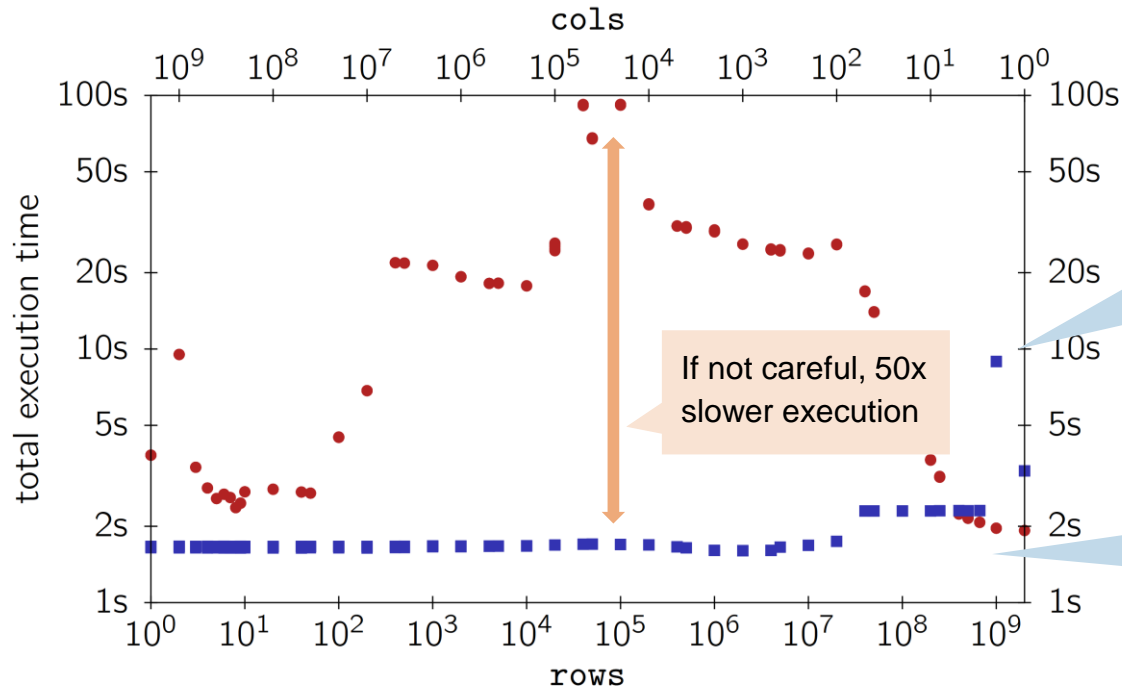
Alternative 2:

```
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



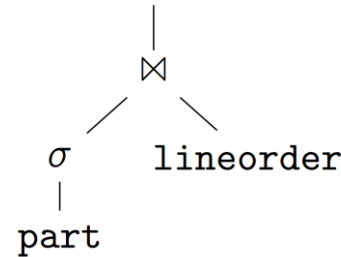
Alternative 2 iterates over the elements column-wise, and its performance is highly dependent on whether the working set size fits in the memory hierarchy.

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
```



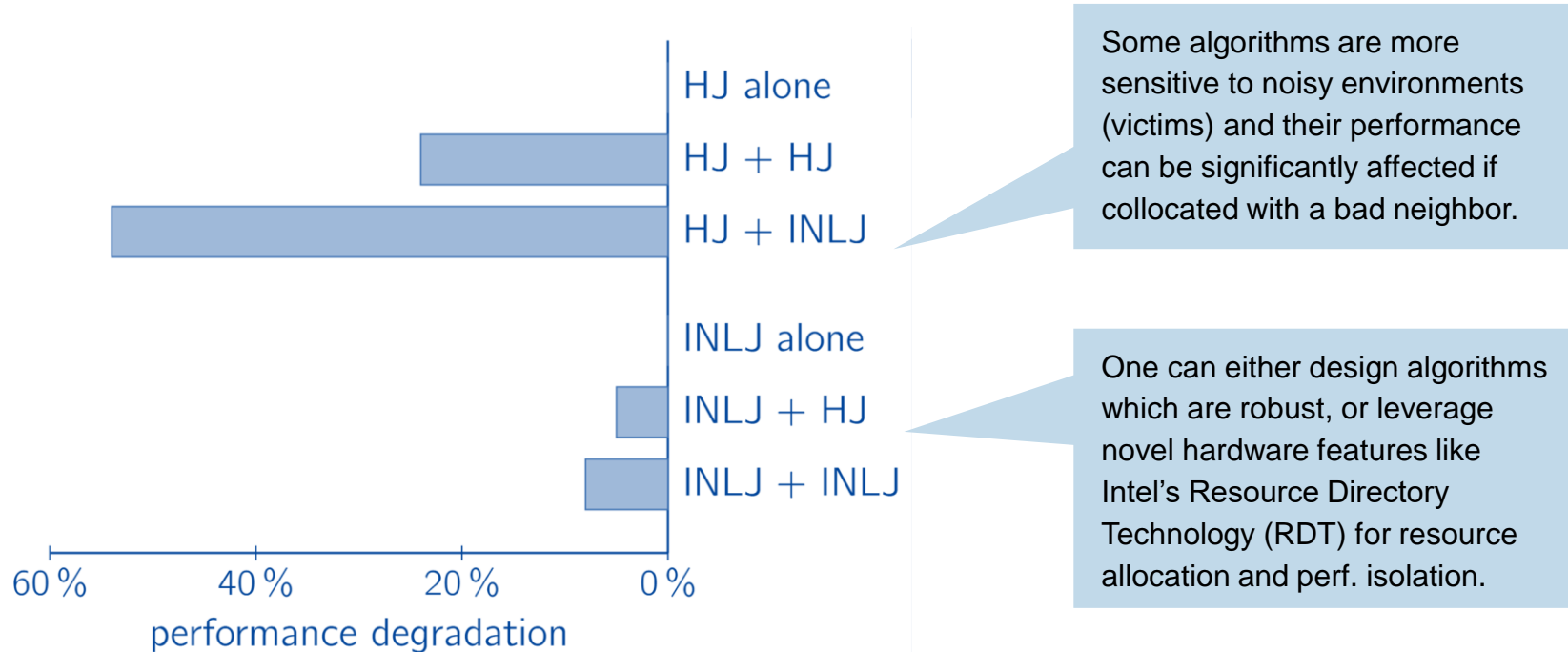
To implement the join (\bowtie) use either:

- 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.

A motivating example (multi-core)

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



A motivating example (accelerators)

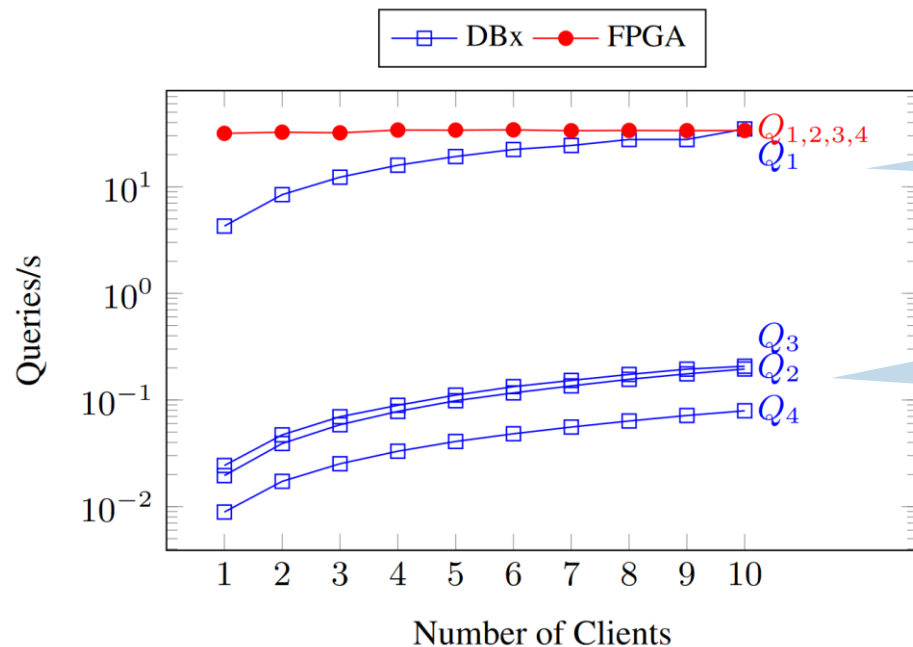
Task: run the following regular-expression queries

```
Q1: SELECT count(*) FROM address_table
     WHERE address_string LIKE '%Strasse%';
Q2: SELECT count(*) FROM address_table
     WHERE REGEXP_LIKE(address_string, '(Strasse|Str.\.)*([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.

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,

- **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**

- InfiniBand / RDMA and Persistent Memory / NVRAM

Course Organization



Lecture:

- Recorded videos uploaded by 8am. Check the lecture's Moodle [webpage](#).
- Wednesdays, 9-11h
- Course website: <http://db.in.tum.de/teaching/ss20/dataprocessingonmodernhardware>
- Please check regularly for updates

Tutorials:

- Interactive video web-conference at: <https://bbb.rbg.tum.de/jan-tk9-mzh>
- Wednesdays, 11-12h (after the lecture)
- First session: today in-person introduction, Q&A session and general set-up
- Consider that exercise material is part of the course 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 h , which improves the grade of the exam e
 - $\min(e, 0.6e + 0.4h)$
- The Exam: TBD
 - Depending on the situation, it may be online and could be oral.

Let's try to make the course as interactive as possible given the circumstances and TUM's regulations.

- During the tutorials, please speak-up, ask questions and discuss!
- Engage in asynchronous discussions on Moodle
- Send me questions you want to be addressed during the tutorial sessions
- If tutorials work well, we could try to shift also the lecture to be live and recorded.

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**

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

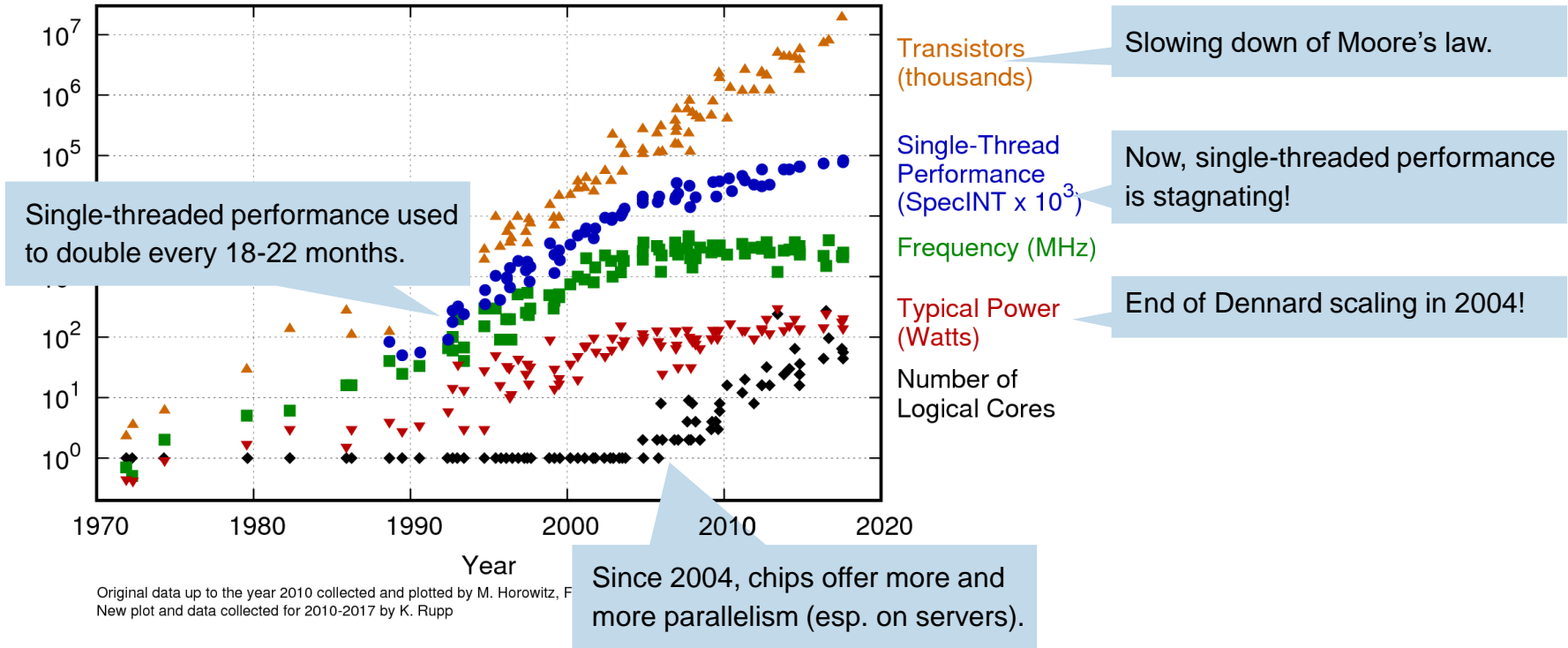
Jana Giceva



Hardware trends

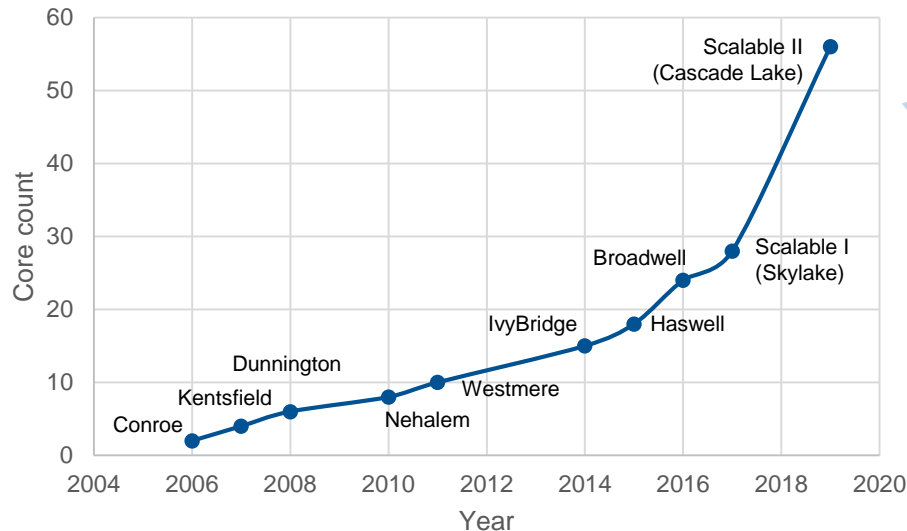
Hardware trends

42 Years of Microprocessor Trend Data



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

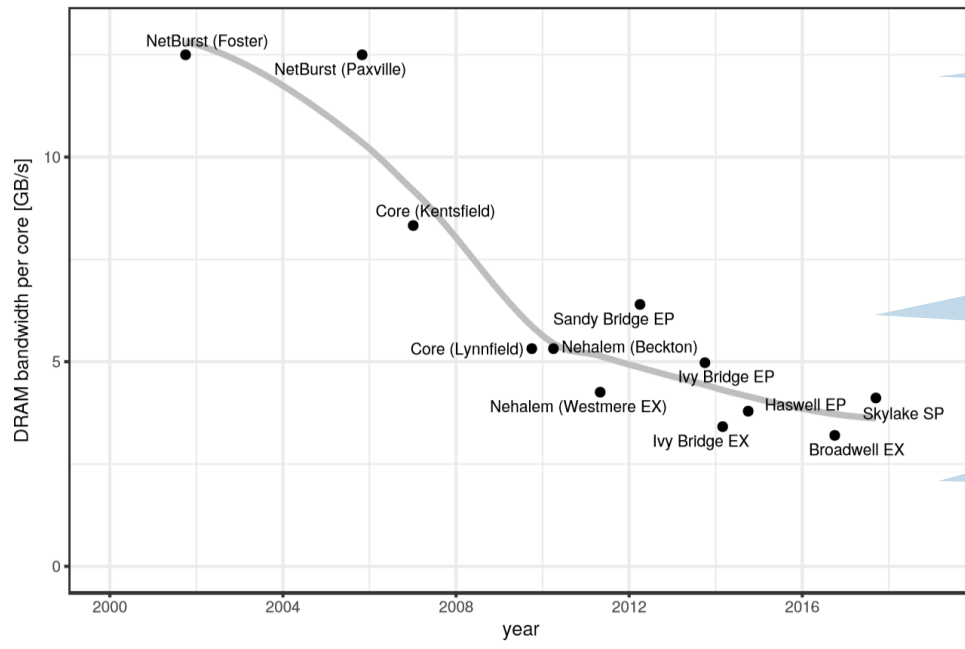
Effect 1: increasing parallelism



- 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)
- 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

Effect 3: towards Hardware Specialization

Dark Silicon:

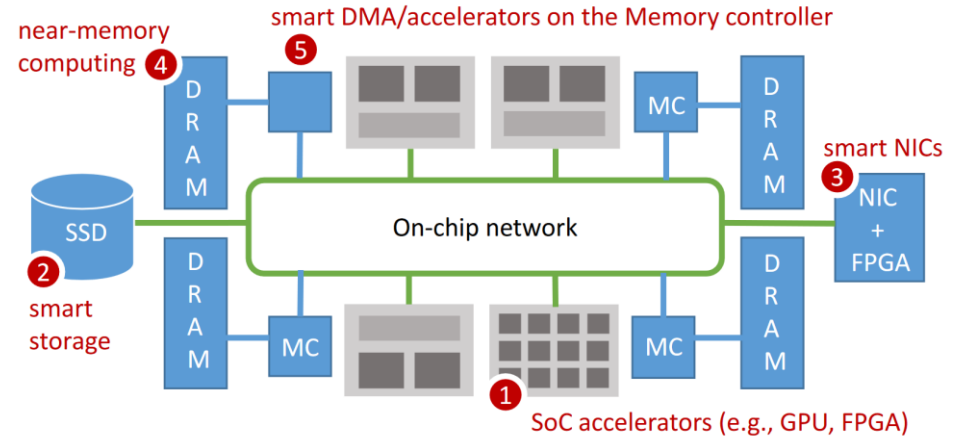
- The end of Dennard scaling
 - Modern CPUs already have close to 10 billion transistors, impossible to power all at the same time
- *Alternative 1*: more cores, but at lower frequency
 - e.g., Intel's Xeon Phi. *But*, Amdahl's law
- *Alternative 2*: many specialized, heterogeneous cores and function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

Domain specific architectures (DSAs)*

- GPUs, TPUs, NPU, 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**

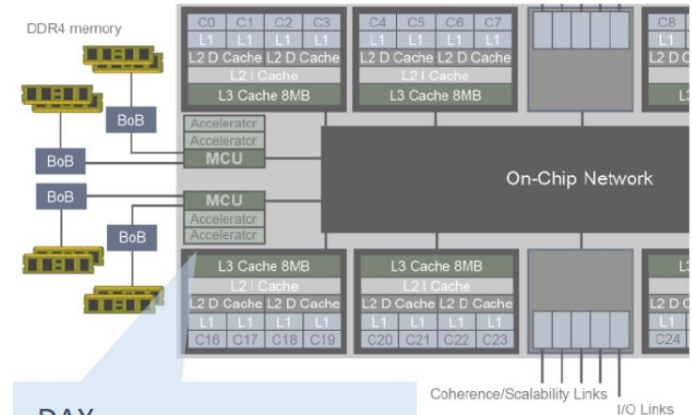
- Gamma database machine, Lisp machine

- **Today on the market:**

- Special instructions: AES encryption, video decoding
- Graphics Processing Units (GPUs)
- Accelerated Processing Units (APU, by AMD)
- Oracle Sparc T7, M7 (data analytics accelerators 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

Oracle's SQL in Silicon – SPARC M7



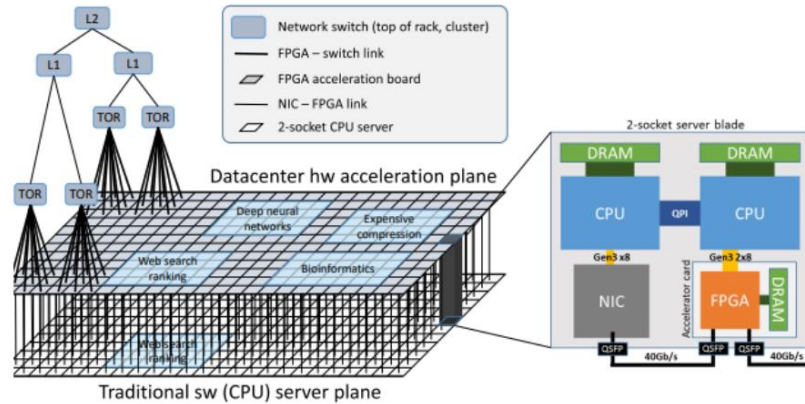
DAX

- In-line compression, decompression
- Bloom-filter
- Predicate evaluation
- Filtering by bit-vector
- Encryption

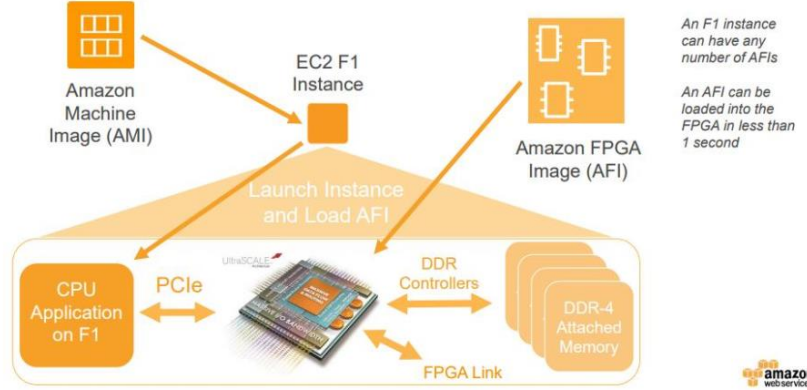
src: White Paper,
August 2016

Accelerators are transforming the Cloud

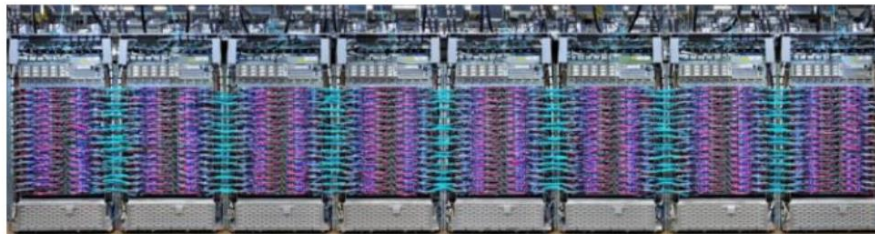
Microsoft's Azure configurable cloud (MICRO'16)



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



Google's TPU 3.0 pods (Google I/O 2018)



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?**

Intel Scalable 2nd generation Processors



In our group we have a new generation Scalable Processor **Intel Xeon-Gold 6212U** (formerly Cascade Lake)

Basic specifications:

- CPU clocked at 2.4 GHz with 24-cores (48 threads)
- with 192GB DRAM (6 x 32GB DDR4 at 2933 MHz), non-inclusive LLC, UPI interconnect
- and 768GB of PMEM (6 x 128GB Intel Optane DC Persistent Memory)

Advanced technologies:

- Vectorization:
 - SSE4.2, AVX, AVX2, AVX-512
- Transactional memory:
 - Synchronization Extensions (TSX)
- Resource allocation and management:
 - Resource Director Technology (RDT)
- Security extensions:
 - Fast and secure data encryption/decryption (AES), Page and Memory Protection Keys (MPX),
Trusted Execution Technology (SGX)

<https://ark.intel.com/content/www/us/en/ark/products/192453/intel-xeon-gold-6212u-processor-35-75m-cache-2-40-ghz.html>

<https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview>

AMD EPYC Infinity Architecture Processors



AMD EPYC 7002 Series Processor 7H12:

- based on AMD Infinity Architecture
- CPU clocked at 2.6 GHz (with 64 cores per SoC, 128 threads)
- With up to 4TB DRAM at 204GB/s using 128 PCIe 4.0 lanes
- Large 256 MB LLC

Advanced technologies:

- AMD Infinity Guard
 - Secure Memory Encryption (SME)
 - Secure Encrypted Virtualization (SEV)

src: <https://www.amd.com/system/files/documents/TIRIAS-White-Paper-AMD-Infinity-Architecture.pdf>

Check more details at “A Deep Dive Into AMD’s ROME EPYC Architecture” article by Timothy Morgan

<https://www.nextplatform.com/2019/08/15/a-deep-dive-into-amds-rome-epyc-architecture/>

Beyond CPU trends and Accelerators

■ Persistent Memory (PMEM)

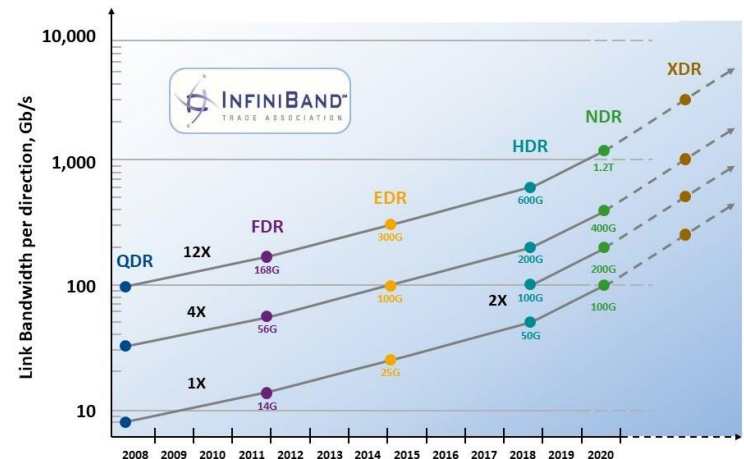
- byte-addressable memory device that resides on the memory bus
- can have DRAM-like access to data (similar latency as DRAM)
- while non-volatile like NAND flash.
- Examples: NVDIMM and Intel's 3D Xpoint DIMMs (Optane DC persistent memory modules)

src: <https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html>

■ Low-latency high-bandwidth networks

- InfiniBand (IB) is a networking standard used in HPC that has very high throughput and low latencies
- HDR already offers 200-600 Gb/s
- Blurring the boundaries between a single machine and a cluster

src: <https://www.mellanox.com/products/infiniband-switches/QM8700>

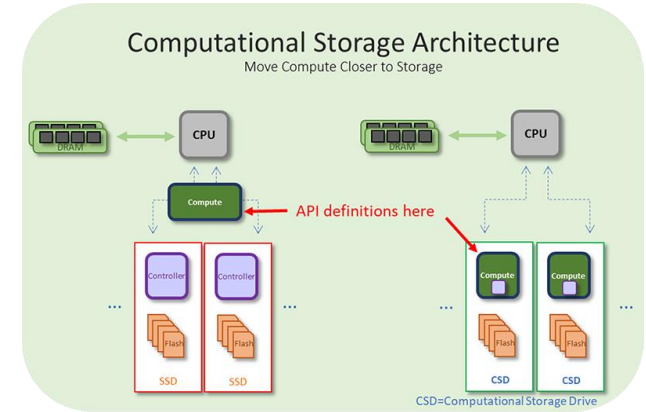


Beyond CPU trends and Accelerators

■ 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

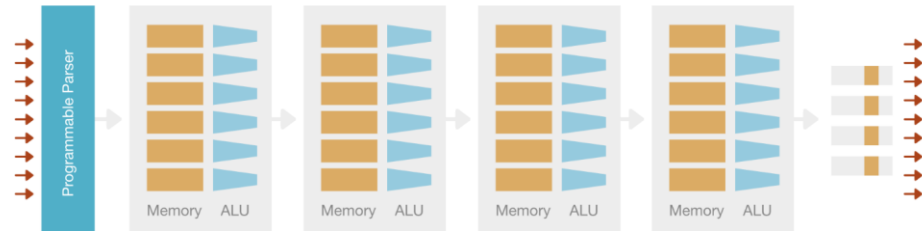
Src: <https://www.snia.org/education/what-is-computational-storage>



■ 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/>



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/>)