Concurrent Systems (ComS/CprE 527)

Ali Jannesari

Department of Computer Science

Iowa State University, Spring 2023

INTRODUCTION

Outline

- System programming (C/C++)
- Parallel programming
- Connection between AI (deep learning) and parallel computing
- Types of parallelism
- Obstacles to parallelism
- What developing parallel software means

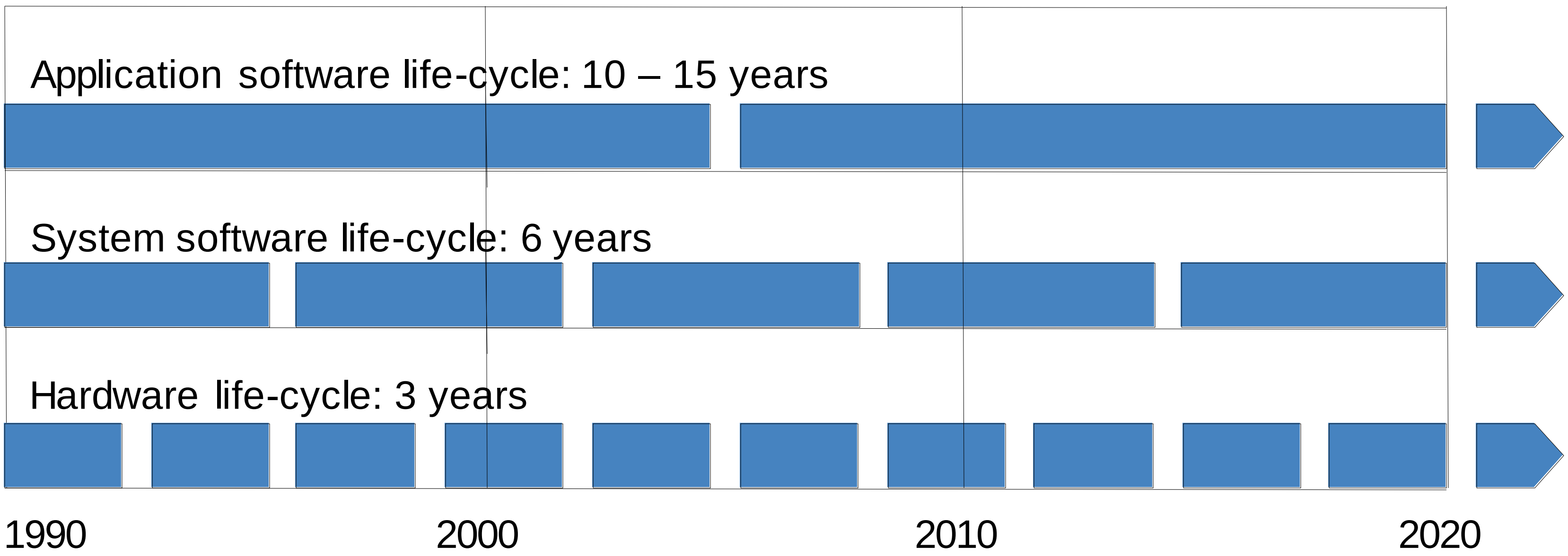
System programming

- Focus on system software as opposed to applications
  - Services provided to other programs rather to end user
- Focus on performance / resource constraints
  - Often exploitation of specific hardware / platform properties
  - Common in high-performance computing

one or both

- Use of low-level programming language suitable for resource-constrained environment
  - Lightweight runtime library with less error checking
  - Runtime garbage collection rare
  - Integrates well with assembler

System software life-cycle



C

- General-purpose programming language
- Closely associated with UNIX
  - System and many programs running on it written in C
  - Nevertheless, independent of any OS or platform
- Often called a “system programming language”
  - Useful for writing of, e.g., compilers or operating systems
  - Examples: gcc and Python
- Also suitable to write end-user applications (e.g., game engines)
  - However, increasingly superseded by C++ (mostly superset of C)
  - Most parallel programming interfaces provide C binding

C++

C++ is a language for developing and using elegant and efficient abstractions

Bjarne Stroustrup

- General purpose
- Bias towards systems programming
- Programming styles
  - Procedural programming
  - Data abstraction
  - Object-oriented programming
  - Generic programming

Emphasis on their effective combination

Parallelism in a computer system

Thread-level

- Software-managed parallelism
- Uses parallel threads or processes running on different cores
- Ability to utilize large numbers of cores is called scalability

Instruction-level

- Pipelining overlaps the execution of consecutive instructions
- Multiple issue allows instruction execution rate to exceed the clock rate
- Vector instructions apply the same operation to multiple elements of an array (compiler has to “vectorize” code)

Low-level digital design

- Set-associative caches use multiple banks of memory
- Carry-lookahead or prefix adder exploit parallelism to speed up the calculation of sums

Parallel programming

- Defined as programming using thread-level parallelism
- Why? Two questions:

Why *parallel* computing?

Why parallel *programming*?

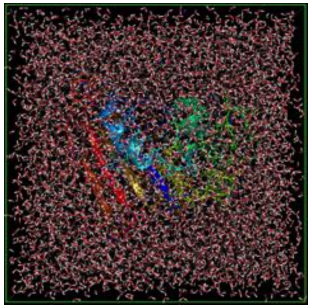
Why parallel computing?

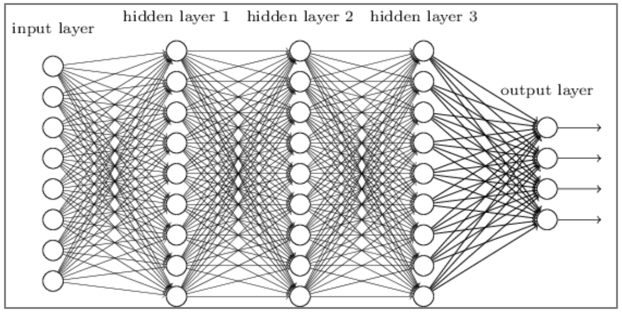
- Problems that cannot be solved fast enough sequentially (with today's processor technology)
- Example: simulation of physical phenomena
- Two dimensions
  - Time to solution
  - Size of the problem
- Sometimes real-time constraints (e.g., weather forecast, autonomous driving)

Hurricane Katrina

Further examples of compute-intensive application

- Simulations
  - Natural sciences: molecular dynamics, materials science
  - Engineering: crash, aerodynamics, fluid dynamics, combustion
- Big Data
  - Graph analysis, sorting
  - Deep learning
- Multimedia
  - Stream processing
  - Games
- Finance
  - Valuation of assets

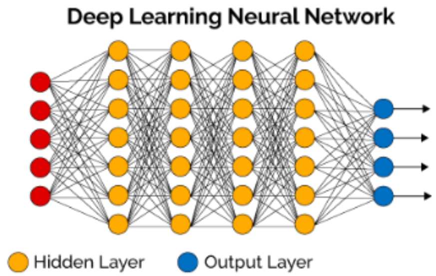


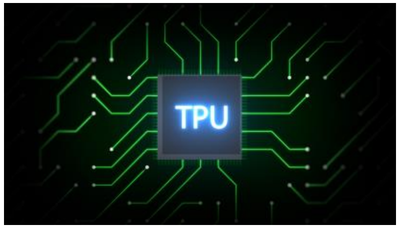


Deep learning and parallel computing

- Connection between AI (deep learning) and parallel computing
- A neural network
  - is modeled to process information in parallel,
  - learns patterns, and then
  - Is used to recognize things
- Faster training via parallelism
  - Data/model parallelism
  - Distributed training
- Faster inference (e.g. DNN pruning, compression techniques)

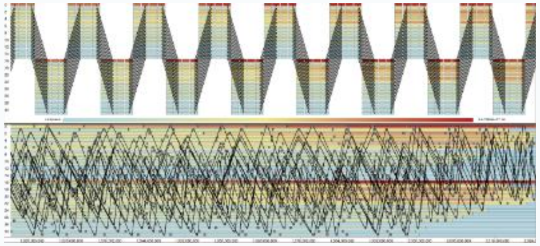
Artificial Intelligence



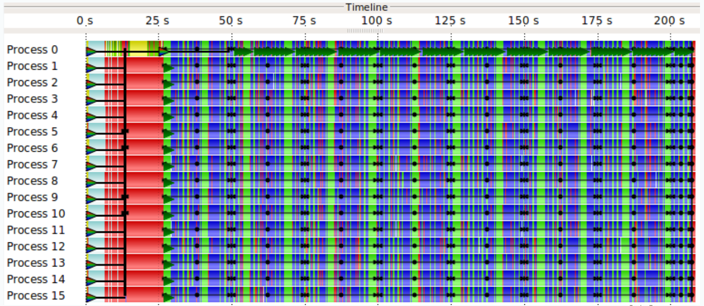


Performance

ORNL - Summit



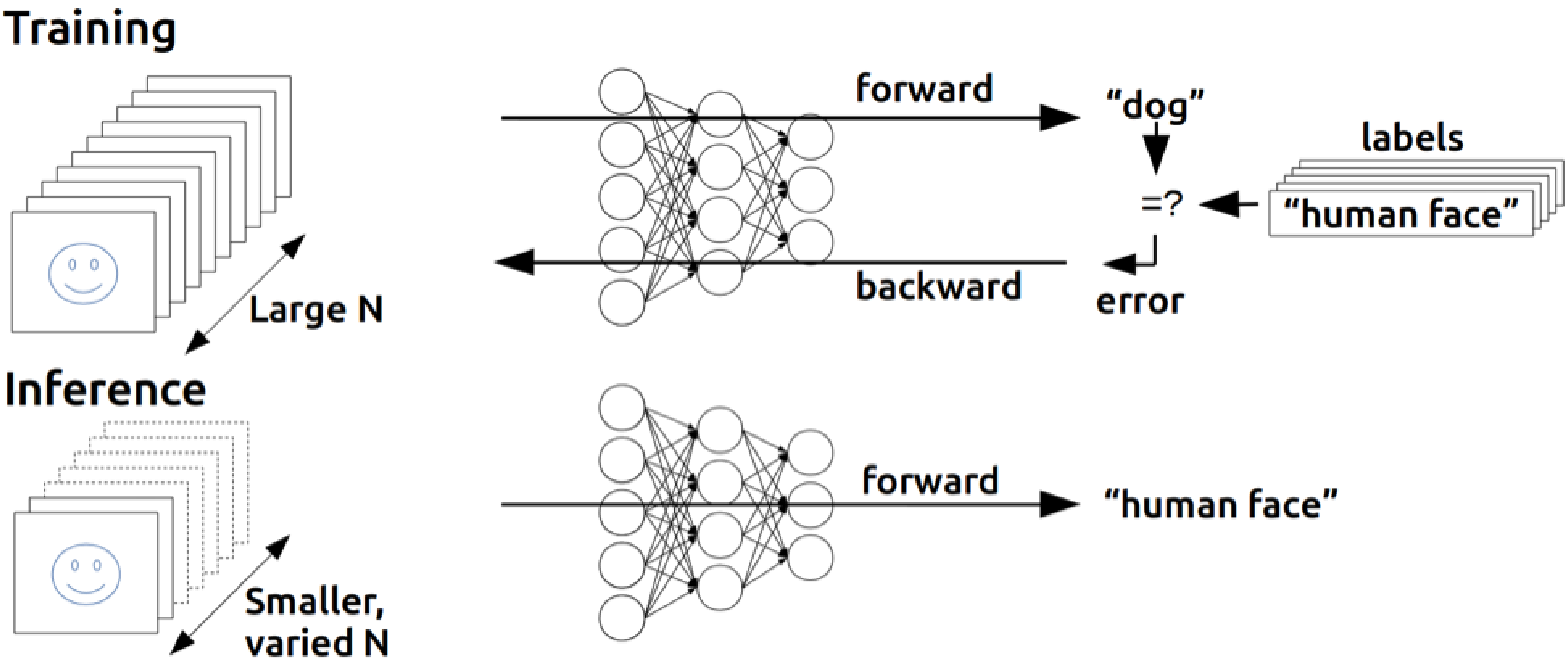
LLNL - Ravel



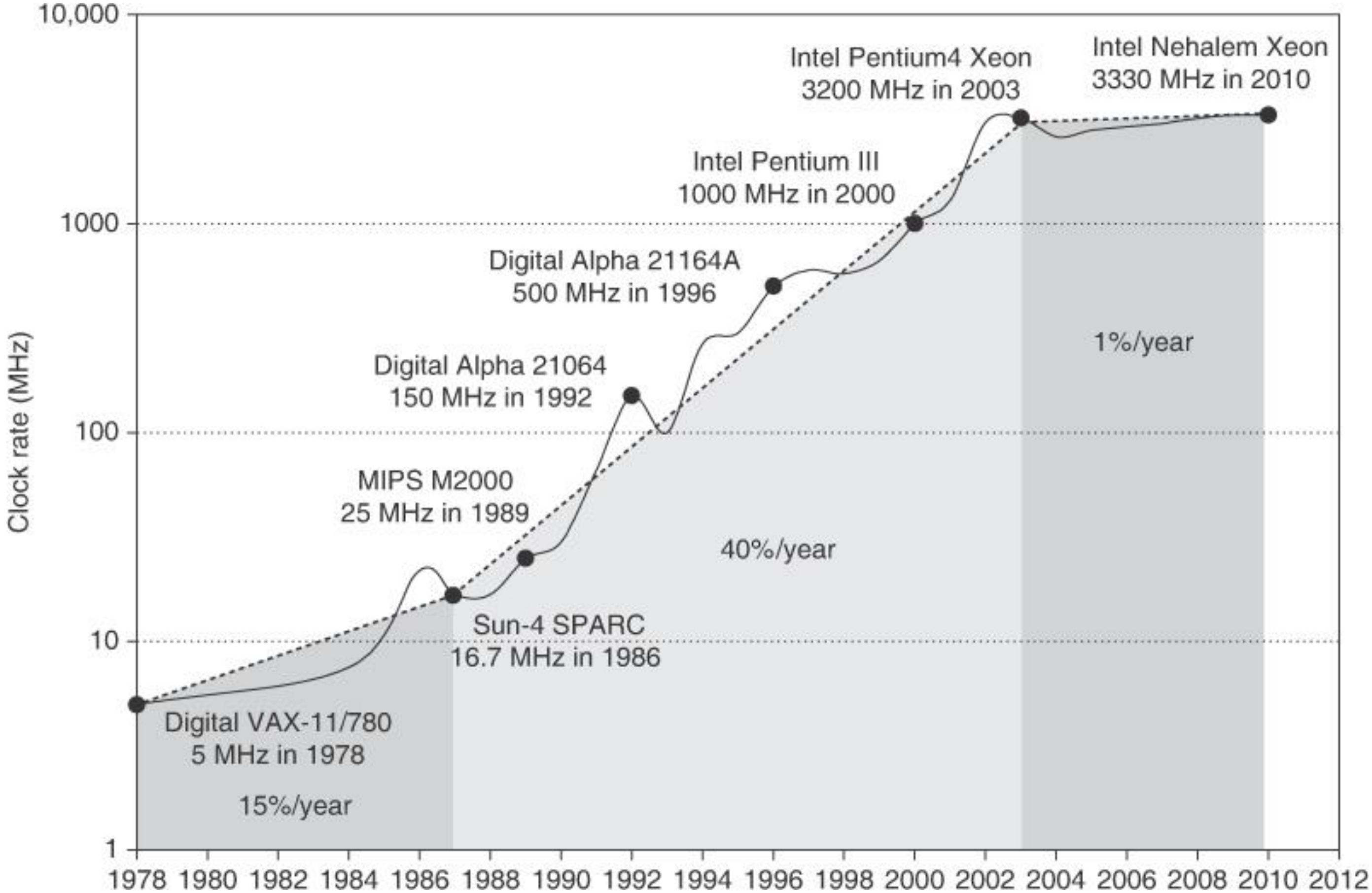
Vampir

Deep learning and parallel computing

- Faster training and inference (**high-performance deep learning**)



Why can't we just create faster uni-processors?

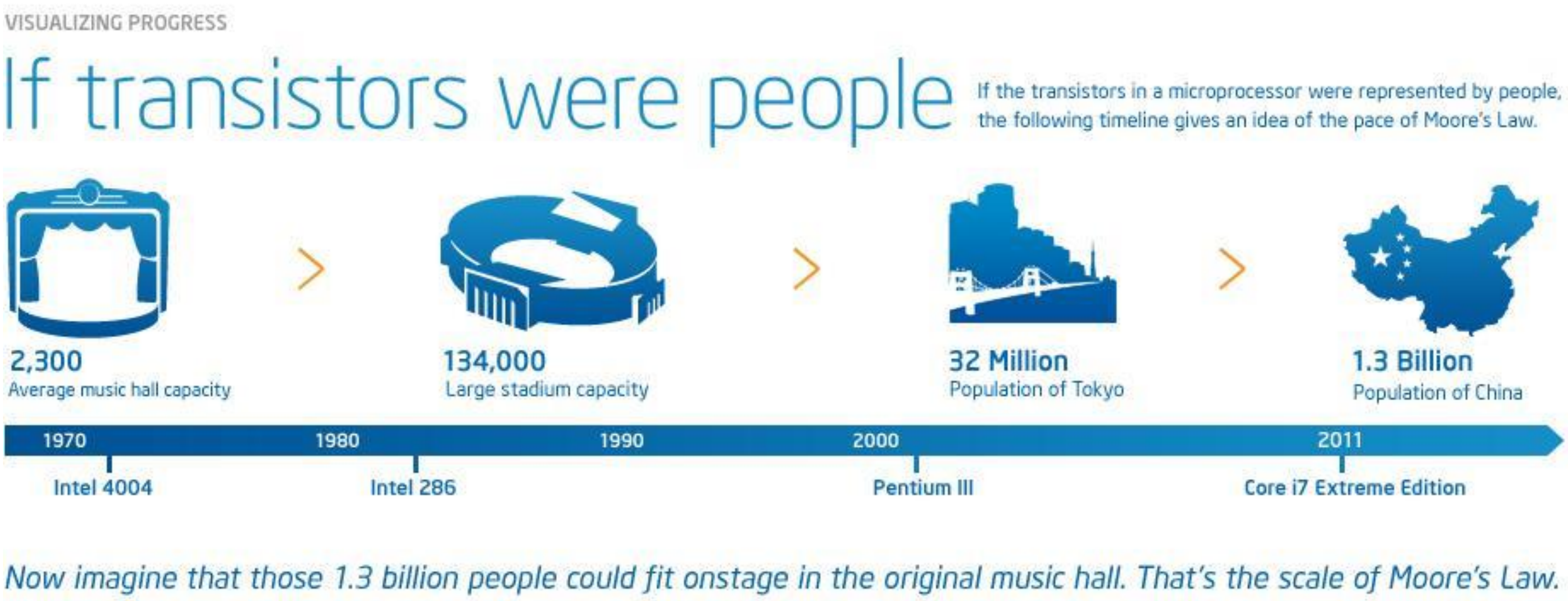


Source: Hennessy, Patterson: Computer Architecture, 5th edition, Morgan Kaufmann

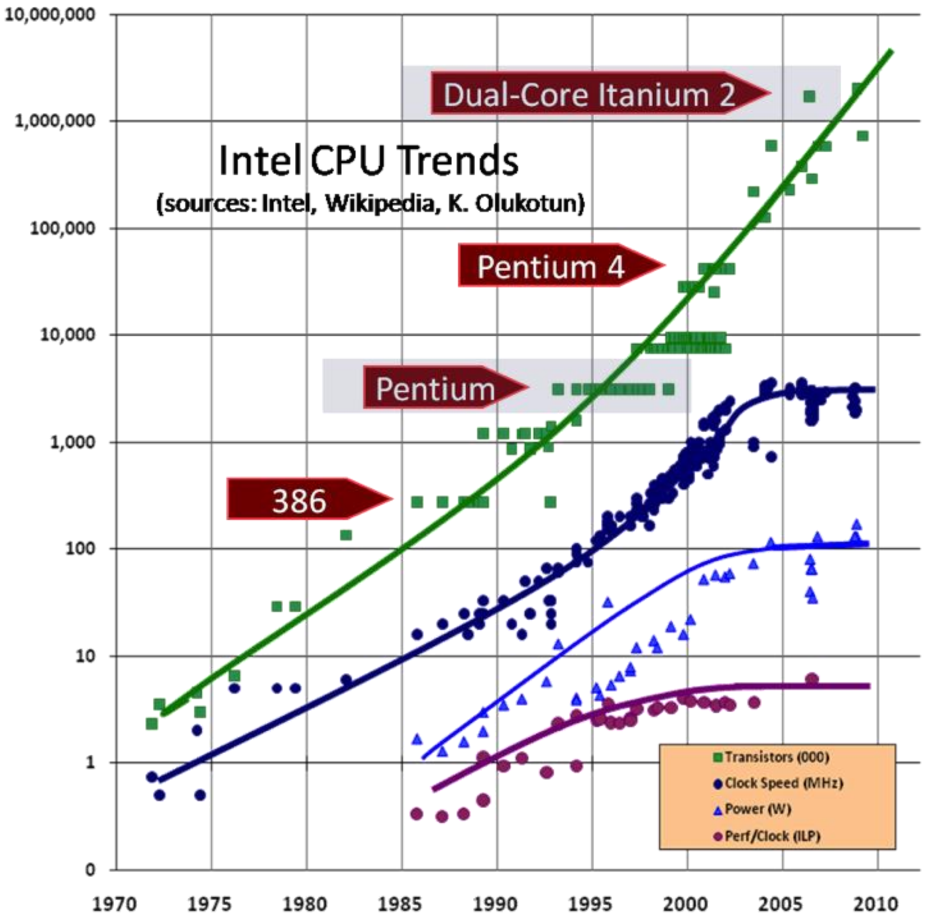
Moore’s law

The number of transistors per chip doubles roughly every two years

Exponential growth of processor performance



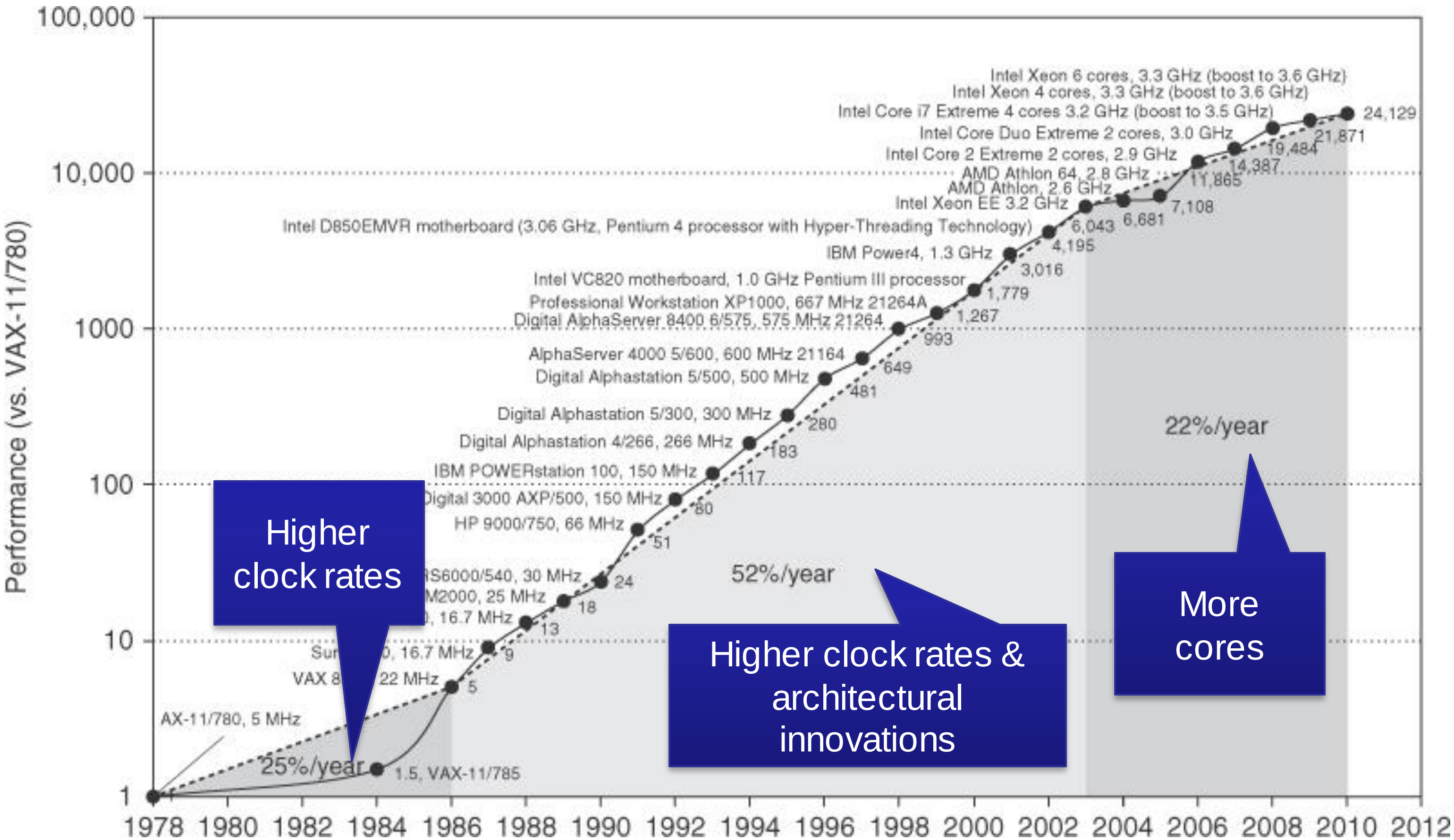
Moore’s law (2)



- Reduction of feature size won’t last forever
- May continue up to 5-7 nm
- End maybe ~ mid 2020s

Source: Herb Sutter: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, 2009.

Growth in processor performance



Source: Hennessy, Patterson: Computer Architecture, 5th edition, Morgan Kaufmann

Dennard scaling

- Why haven’t clock speeds increased, even though transistors have continued to shrink?
- Dennard (1974) observed that voltage and current should be proportional to the linear dimensions of a transistor
  - Thus, as transistors shrank, so did necessary voltage and current; power is proportional to the area of the transistor.

Courtesy of Bill Gropp

Dennard scaling

Dynamic power = $\alpha * CFV^2$

- α = percent time switched
- C = capacitance
- F = frequency
- V = voltage

Capacitance is related to area

- So, as the size of the transistors shrunk, and the voltage was reduced, circuits could operate at higher frequencies at the same power

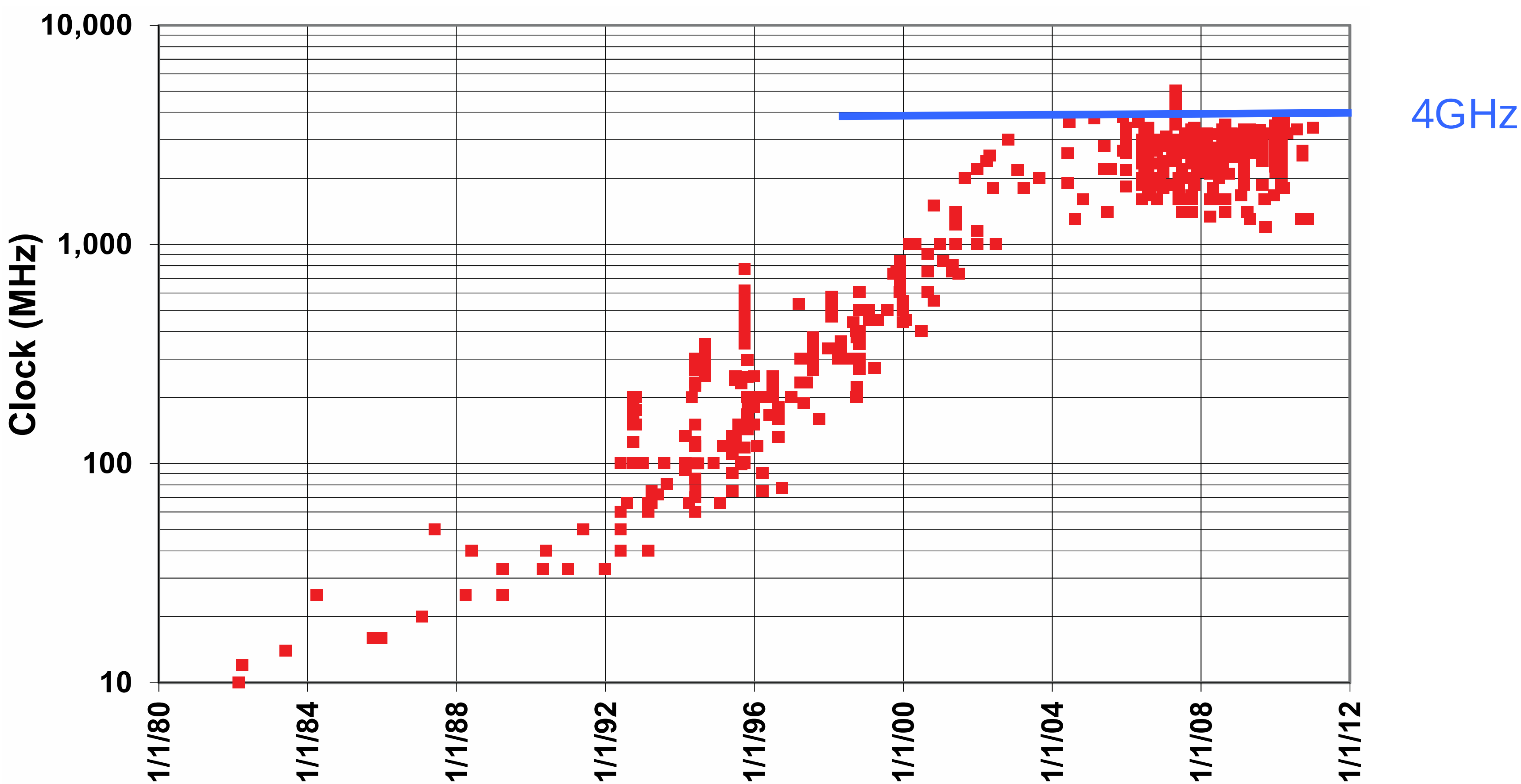
Courtesy of Bill Gropp

End of Dennard scaling

- Dennard scaling ignored the “leakage current” and “threshold voltage”, which establish a baseline of power per transistor
- As transistors get smaller, power density increases because these don’t scale with size
- These created a “Power Wall” that has limited practical processor frequency to around 4 GHz since 2006

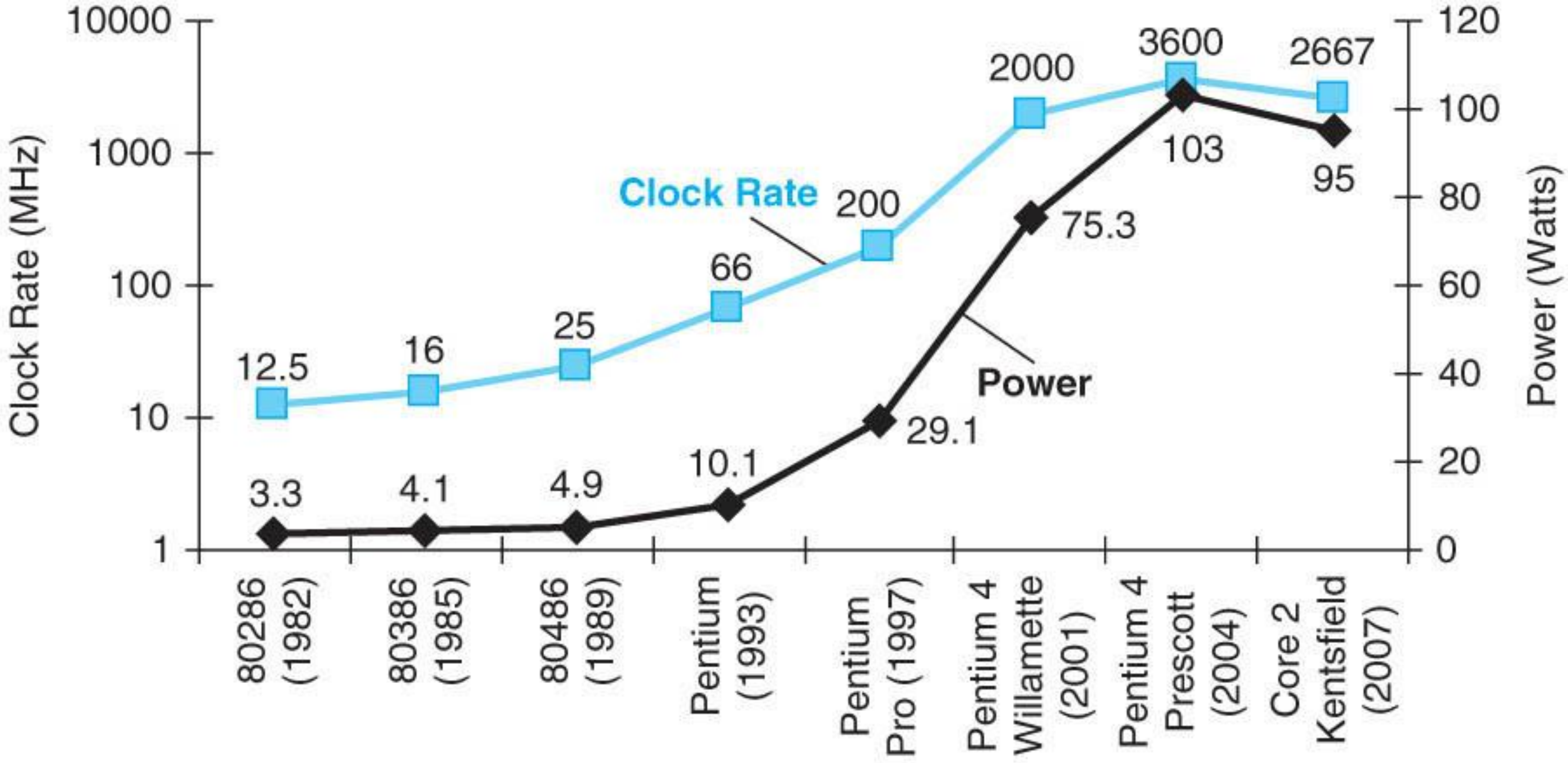
Courtesy of Bill Gropp

Historical clock rates



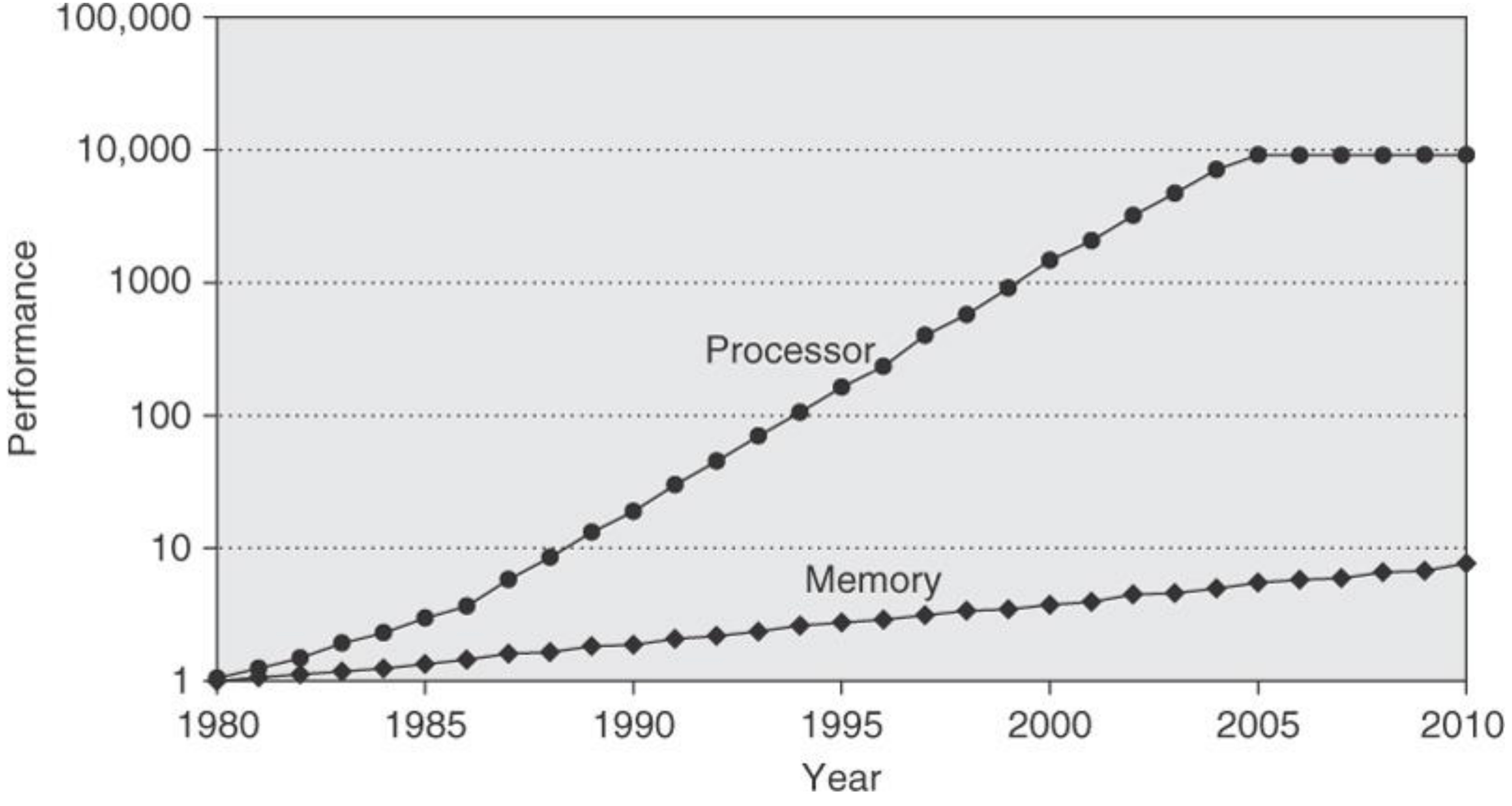
Courtesy of Bill Gropp

Clock rate vs. power



Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann

CPU and memory performance



Source: Hennessy, Patterson: Computer Architecture, 5th edition, Morgan Kaufmann

Bandwidth and latency

Bandwidth or throughput

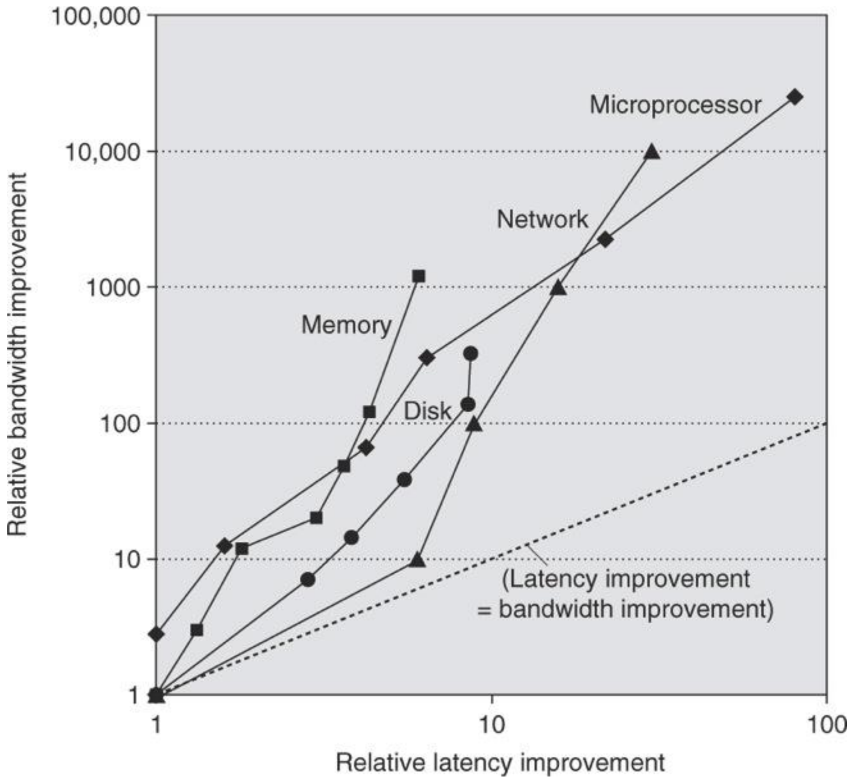
- Total amount of work done in a given time (e.g., disk transfer rate)

Latency or response time

- Time between start and completion of an event (e.g., disk access time)

Rule of thumb

- Bandwidth grows by at least the square of the improvement in latency



Source: Hennessy, Patterson: Computer Architecture, 5th edition, Morgan Kaufmann

Instruction level parallelism

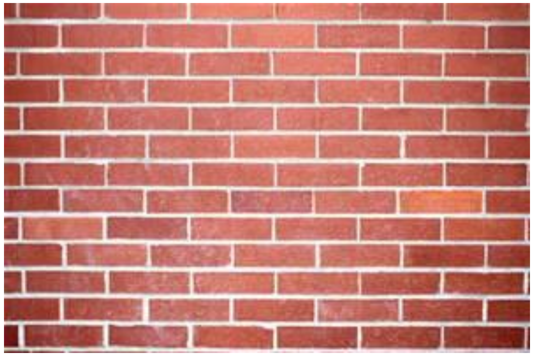
- Parallelism on the level of individual machine instructions
  - Pipelining
  - Branch prediction
  - Dynamic scheduling
  - Multiple issue
  - Speculation
- In the past, exploitation of ILP was a main vehicle of processor performance improvement
- Now, diminishing returns on finding more ILP in programs

Summary

Power wall

Memory wall

ILP wall



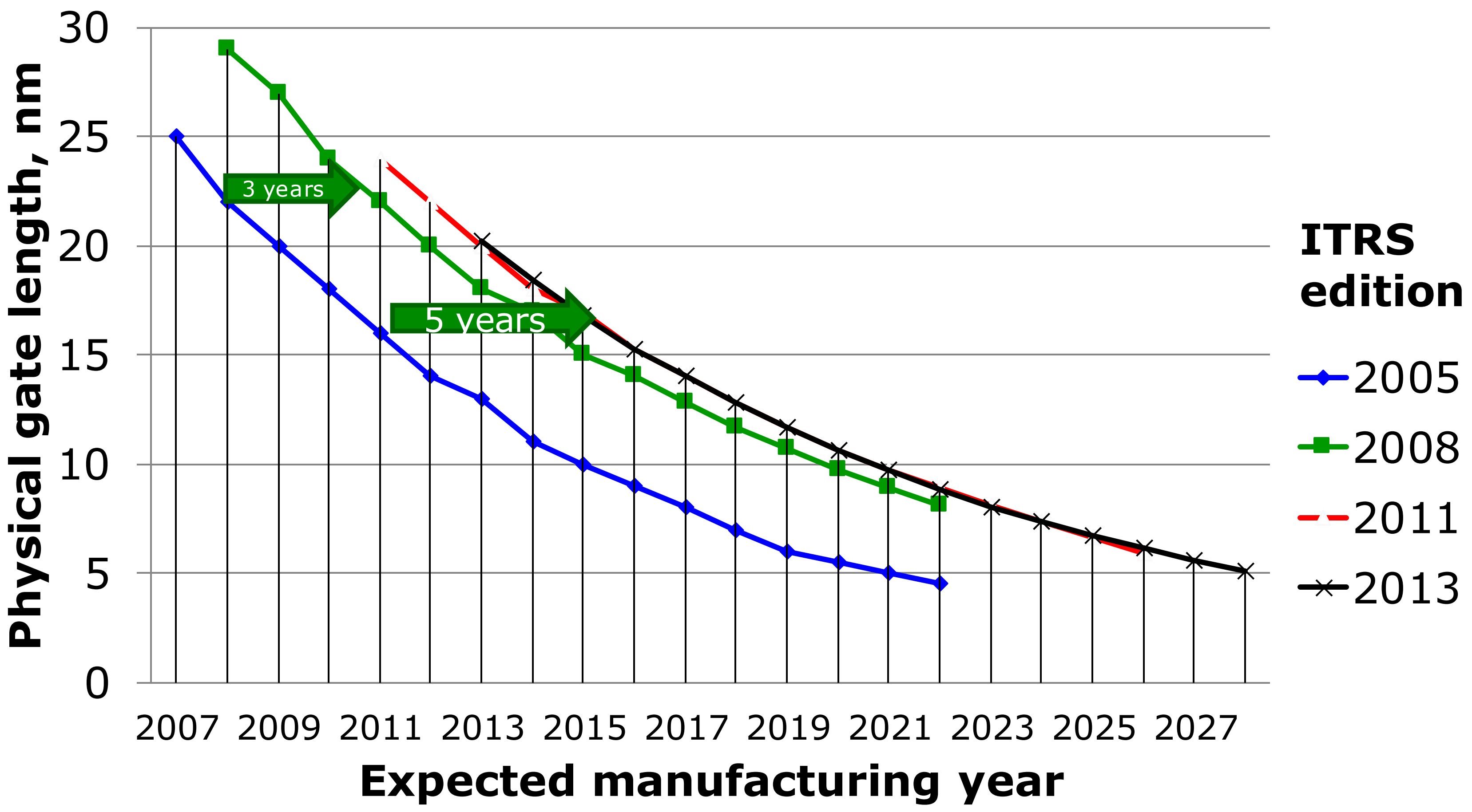
Brick wall

Road maps

- The Semiconductor industry has produced a roadmap of future trends and requirements
- Semiconductor Industry Association (~1977, roadmaps from early ’90s)
- International Technology Roadmap for Semiconductors (~1998)
  - http://www.itrs.net/
- Emerging hardware and technology for machine learning/AI

Courtesy of Bill Gropp

ITRS projections for gate lengths (nm) for 2005, 2008 and 2011 editions

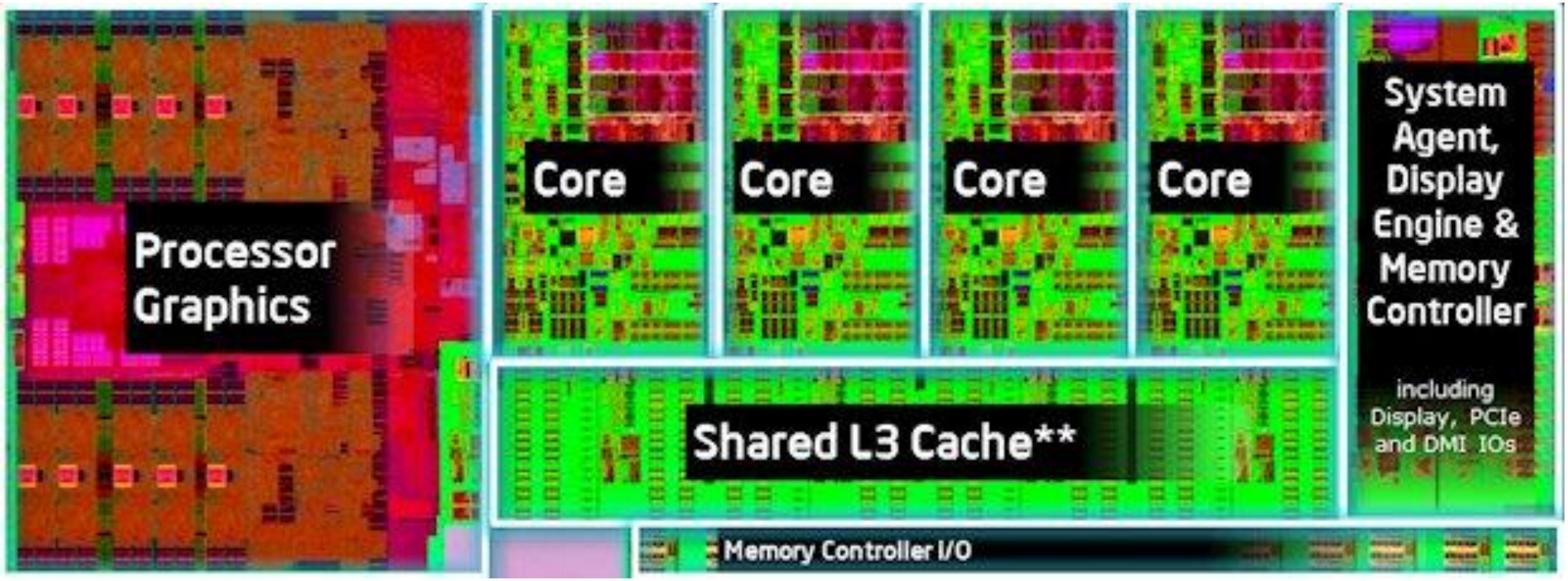


Courtesy of Bill Gropp

Multicore

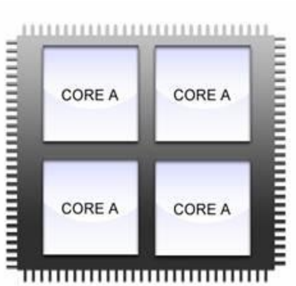
- Since 2002, uni-processor performance improvement has dropped
  - Power dissipation
  - Almost unchanged memory latency
  - Little instruction-level parallelism left to exploit efficiently
- Further performance improvements by placing multiple processors on a single die (multi-core architecture)
  - Initially called on-chip or single-chip multiprocessing
  - Cores often share resources (e.g., L2, L3 cache, I/O bus)
  - Limited by memory bandwidth
- Leverages design investment by replicating it

Intel Haswell

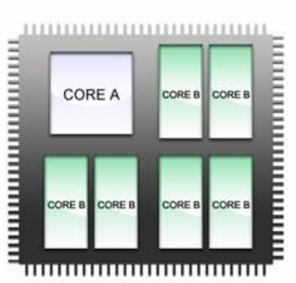


Manycore

- New version of Moore's law
  - The number of cores will double every two years (not exactly true)
  - Recall that today's GPUs feature 100s and 1000s of cores
- Heterogeneity
  - Not all cores necessarily uniform
  - Cores for specific functions
    - Control vs. computation
    - Video
    - Graphics
    - Cryptography
    - Digital signal processing
    - Vector processing
    - Deep learning (DNN)



Homogeneous design



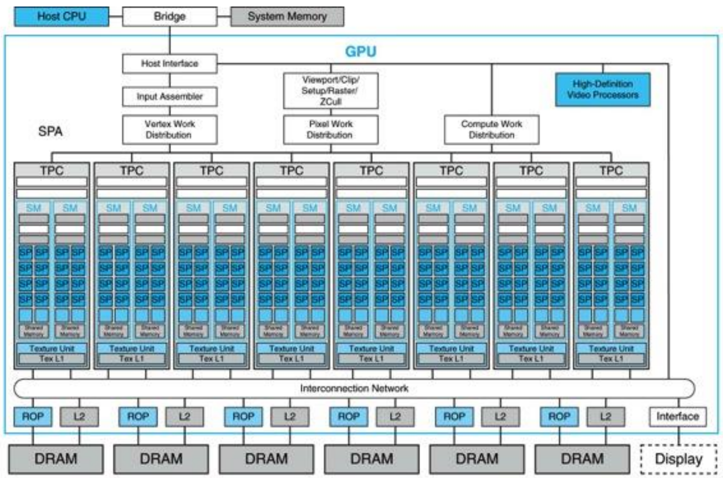
Heterogeneous design

Graphics processing units (GPUs)

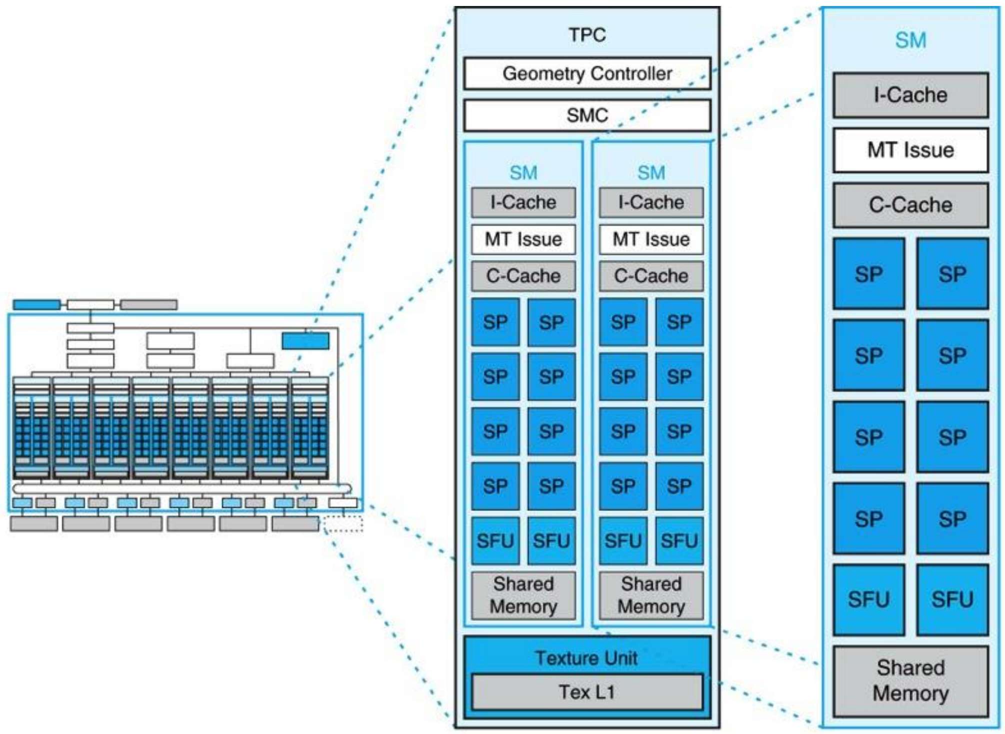
- Processors optimized for 2D and 3D graphics and video
- Became more programmable over time
  - Dedicated logic replaced by programmable processors
- New paradigm at the intersection of graphics processing and parallel computing: visual computing
  - Enables new graphics algorithms
- GPU computing
  - Using a GPU for computing via a parallel programming language and API (e.g., CUDA, OpenCL, Vulkan)

Example: NVIDIA G80

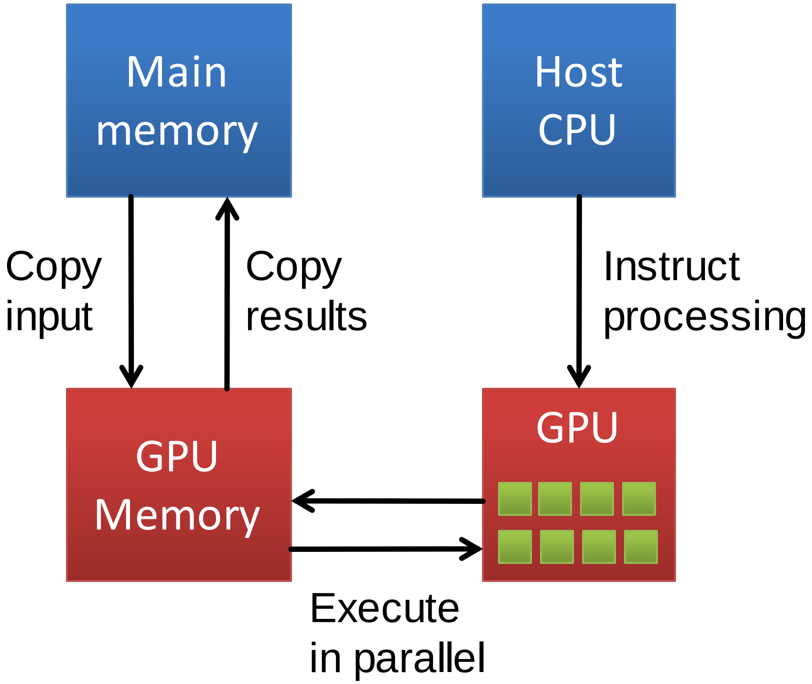
Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann

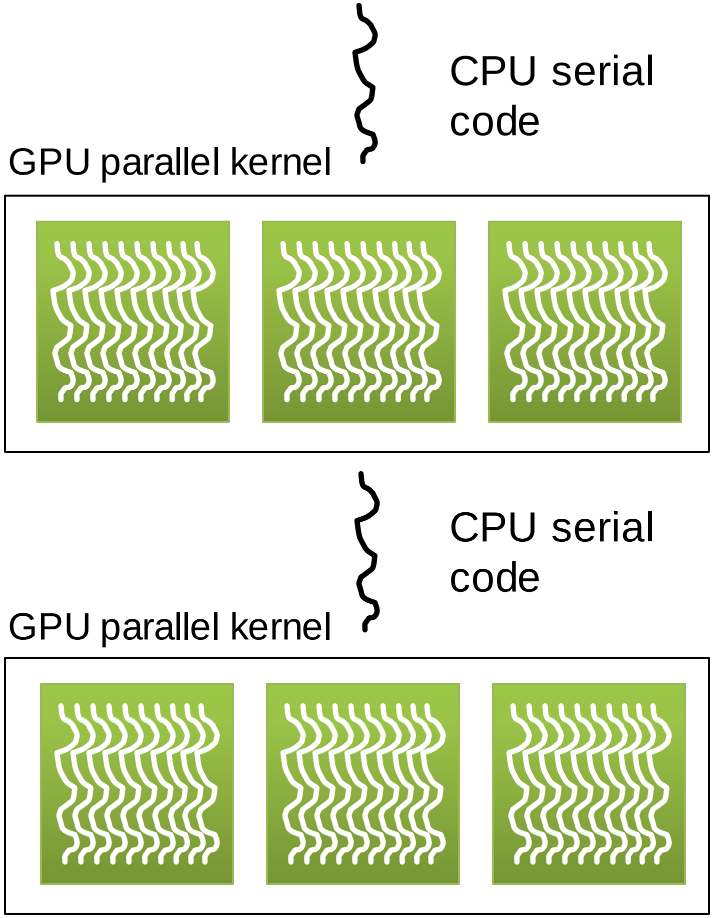


GPU forms heterogeneous system with general-purpose CPU

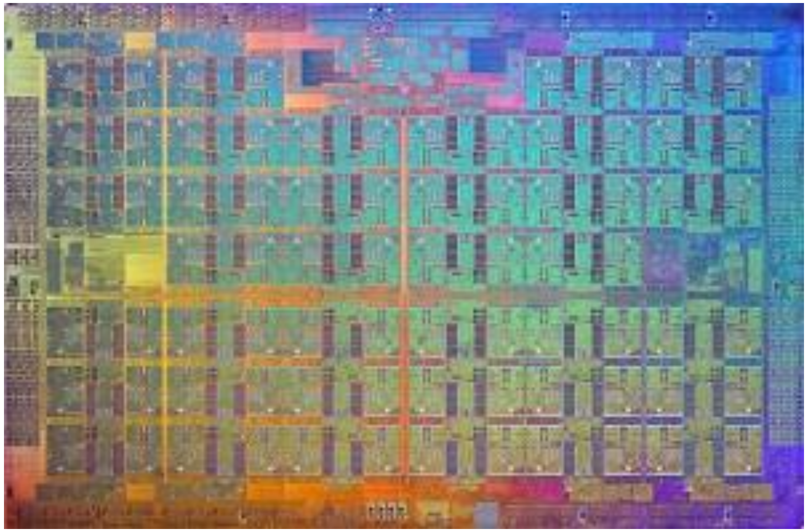


GPU computing





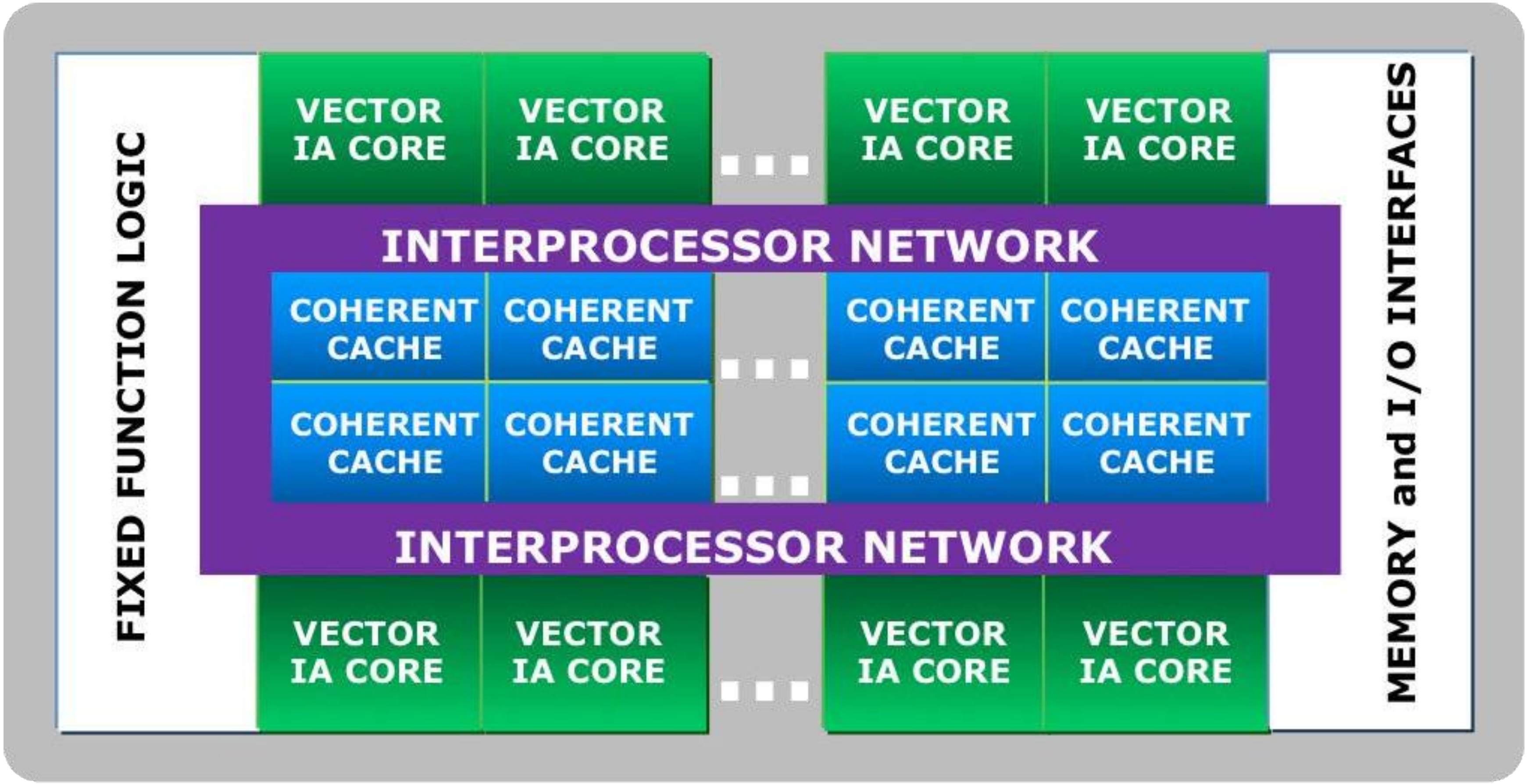
Intel Xeon Phi – Knights Landing



Model 7290

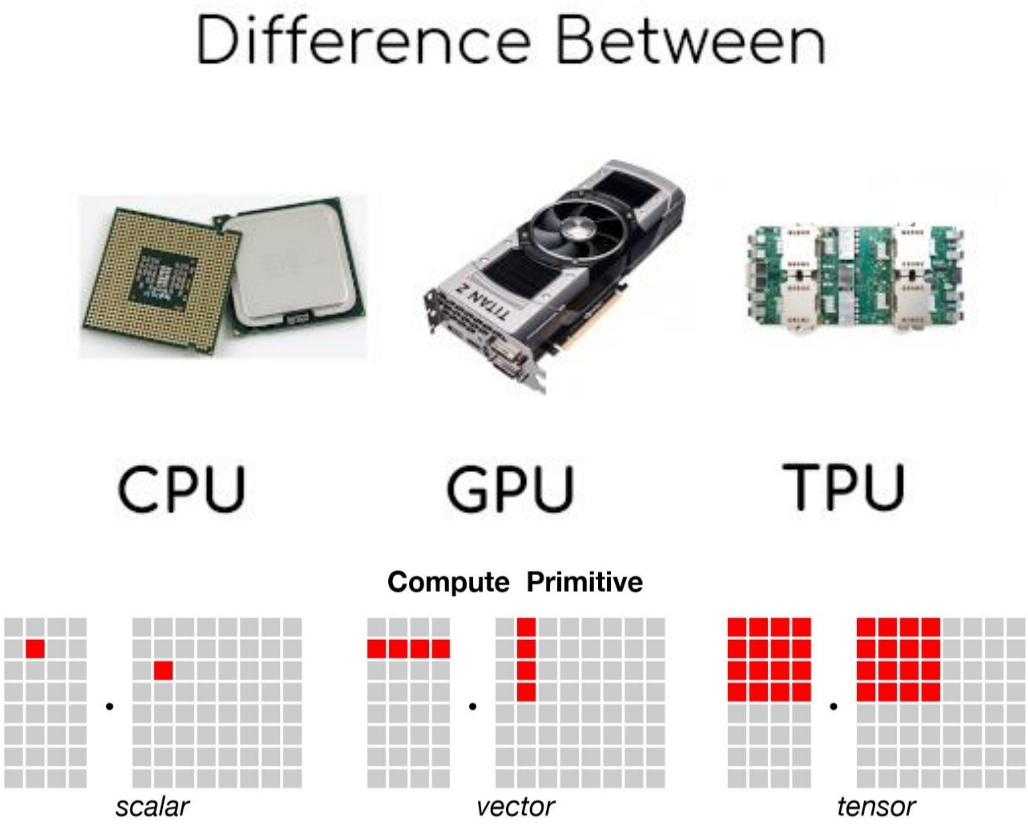
- 72 x86-based cores
- Core frequency 1.5 GHz
- 4 hardware threads per core
- Cache coherence across entire processor
- 288 hardware threads in total
- 512-bit wide SIMD instructions
- 16 GB memory

Intel Xeon Phi Architecture

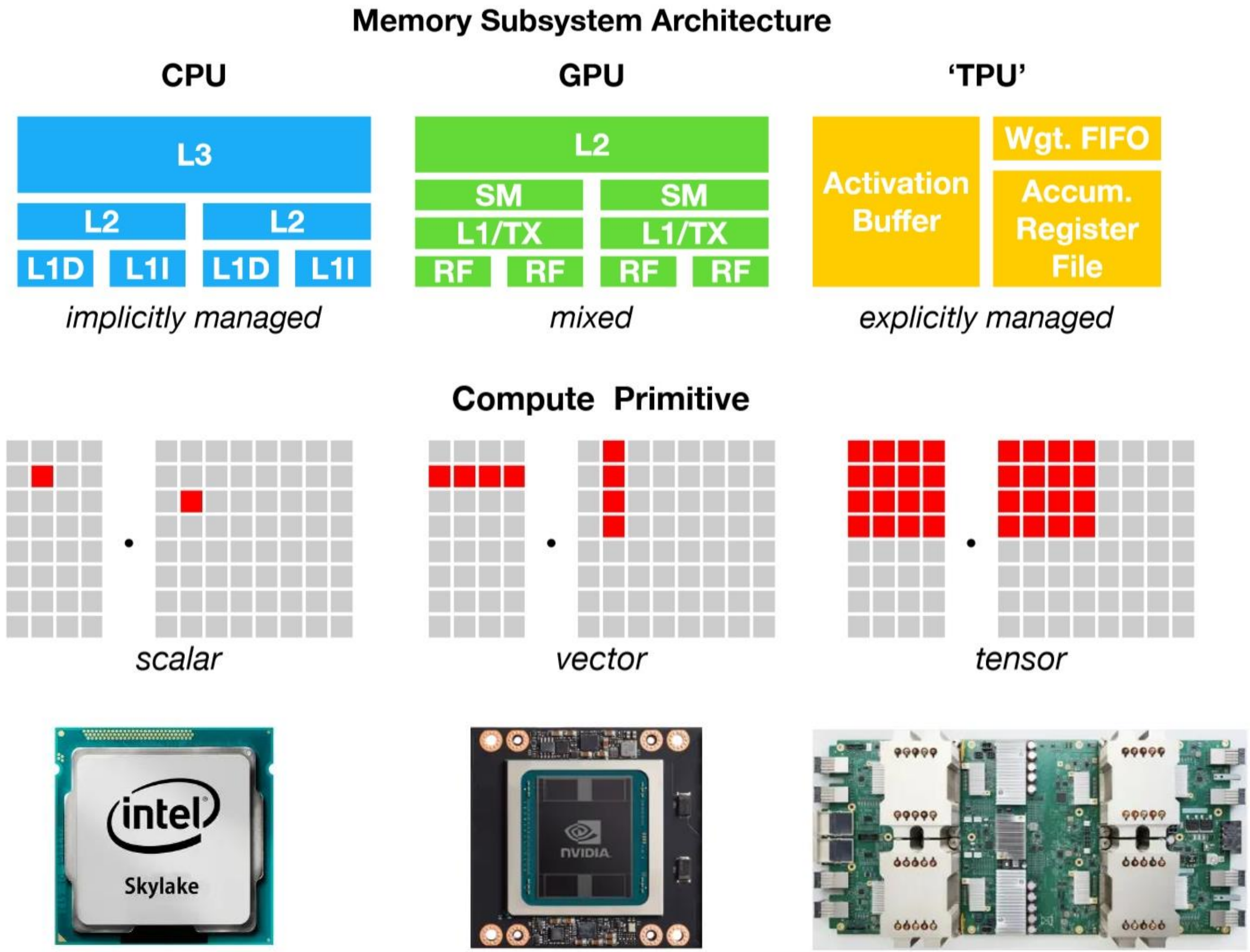


TPU - Tensor processing unit

- Tensor processing unit (TPU)
- AI accelerator application-specific integrated circuit developed by Google
- Specifically for neural network machine learning
- Compatible Google's own TensorFlow software



TPU - Tensor processing unit



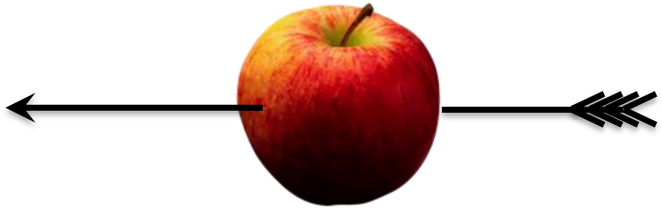
Why parallel computing?

Limited single-processor performance leaves thread-level parallelism (in combination with sequential optimization) as the only option to speed up individual programs

Friedrich Schiller, Wilhelm Tell, 1. Chapter, 3. Section

“The weak is getting powerful when connected.”

Werner Stauffacher



Why parallel computing? (2)

- Dynamic power = $\alpha * CFV^2$
- Scaling up the frequency on a given CPU requires also higher voltage
- For a given circuit in a given technology, power increases at a rate proportional to approx. F^3

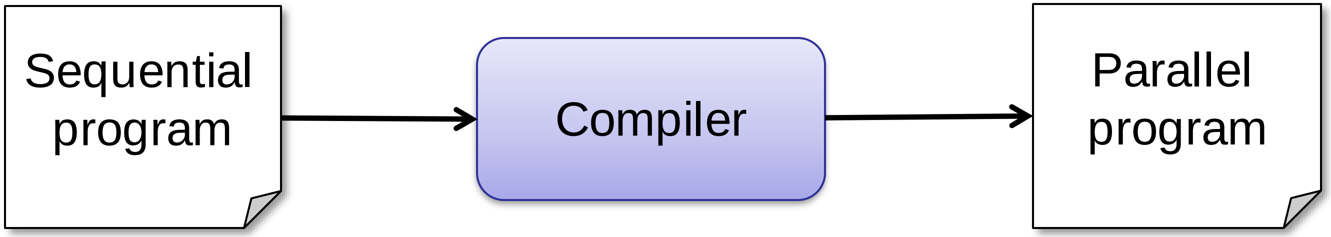
Lower frequency can result in over-proportional power savings

Parallel computing has the potential to solve the same problem in the same time using less energy

Green Computing

Why parallel programming?

Why don’t we have auto-parallelizing compilers?



- Not all parallelization opportunities statically visible
  - Would result in very conservative parallelization
- Practical only for certain pieces of code
  - Loop nests via polyhedral model (affine loop bounds and array accesses)
  - Small sequences of instructions (vectorization) ➔ fine granular parallelization

Parallelism vs. concurrency
(often used interchangeably)

Parallelism

- Parallel processing of subtasks for the purpose of speedup
- Requires parallel hardware

Concurrency

- Overlapping but potentially unrelated activities
- Access to shared resources possible
- Does not require parallel hardware

Concurrent programming

Separation of concerns

- Group related code
- Keep unrelated code apart
- Examples
  - Waiting for input vs. processing input in interactive applications
  - Waiting for requests vs. processing requests in server
  - Background tasks such as monitoring the file system for changes in desktop applications
- Beneficial even when using a single CPU

Web browser

DVD player

Types of parallelism

Functional parallelism

- Views problem as a stream of instructions that can be broken down into functions to be executed simultaneously
- Each processing element performs a different function
- Sometimes also called task parallelism

Data parallelism

- Views problem as an amount of data that can be broken down into chunks to be independently operated upon (e.g., array)
- Each processing element performs the same function but on different pieces of data

Example

Several tutors grade a test

- The task sheet contains several tasks

Functional parallelism	Data parallelism
Each tutor grades a different subset of the tasks	Each tutor grades a subset of the students

Another example

Functional parallelism

Construction workers

Data parallelism

Hollow square formation

Comparison

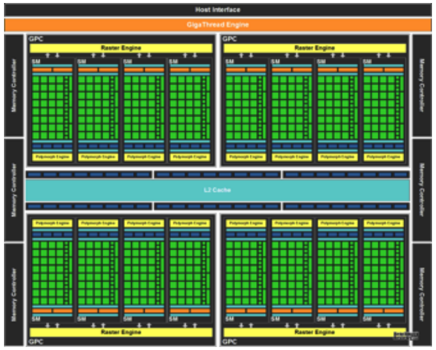
Functional parallelism

- Often exploited via pipeline

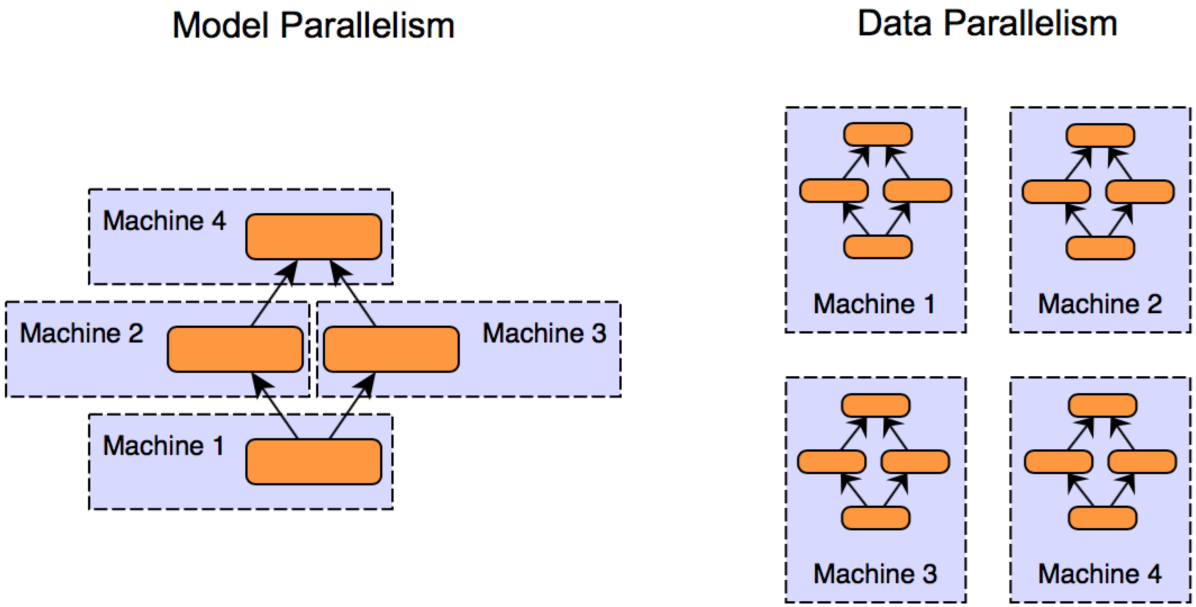
- Limited scalability – algorithms do not contain arbitrary number of functions
- “Too many cooks spoil the broth”

Data parallelism

- Scales with the amount of data
- Little control per computation required ➔ energy efficient
- Suitable for manycore architectures ➔ high-performance machine/deep learning

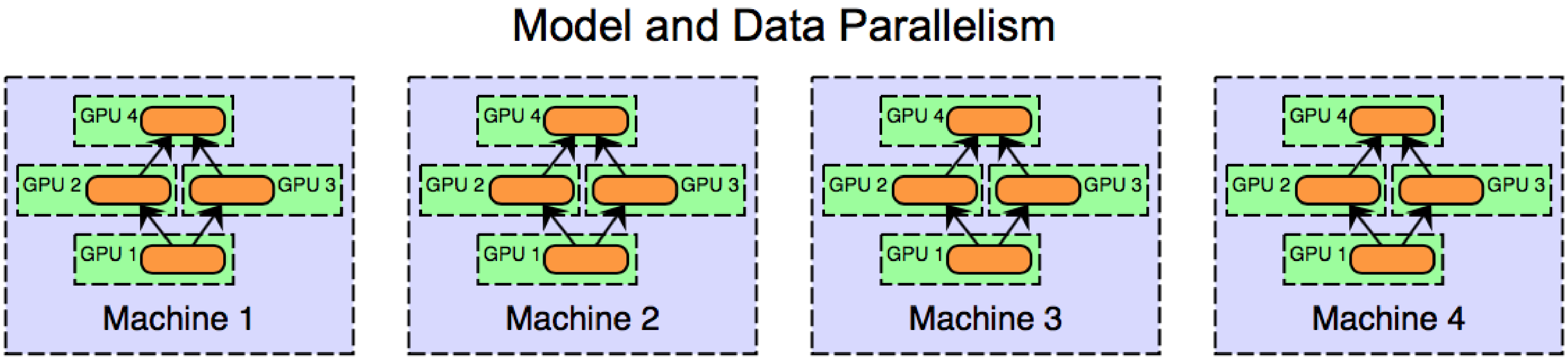


Data parallelism vs. model parallelism in distributed deep learning training



- **Data parallelism:** Different machines have a complete copy of the model
- **Model parallelism:** Different machines in the distributed system are responsible for the computations in different parts of a single network

Data parallelism vs. model parallelism in distributed deep learning training (2)



- **Hybrid parallelism:** Use model parallelism (model split across GPUs) for each machine, and data parallelism between machines
- A cluster of **multi-GPU systems**

Developing parallel software

Three scenarios:

- Writing parallel code from scratch
- Parallelizing a sequential program
- Modifying a parallel program

Modifying existing software

Redesign is normal, and software design “de novo” is the exception

Ralph E. Johnson

Cost and benefits of parallelization

Benefit

- Speedup
- Sometimes cleaner design (separation of concerns)
- More aggregate memory (distributed memory parallelization)
- Potentially lower energy consumption

Cost

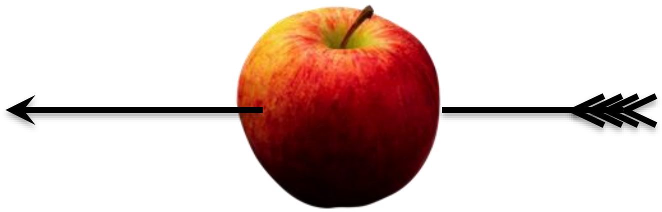
- Programming effort
- Program complexity
- Overhead (communication & synchronization)
- Bugs
- Potentially non-determinism
- Extra dependencies (library, compiler)

Again Wilhelm Tell

Friedrich Schiller, Wilhelm Tell, 1. Chapter, 3. Section

“The weak is getting powerful when connected.”

Werner Stauffacher



„The strong one is the most powerful alone“

Wilhelm Tell

Obstacles to parallelism – dependences

Two types

- Control dependences

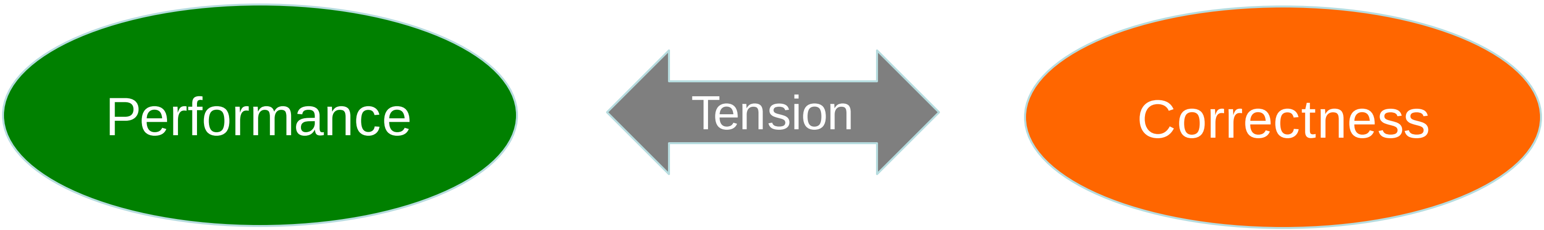
```
if (condition) then
   do_work();
```

- Data dependences

```
for ( i = 1; i <= 2; i++ )
   a[i] = a[i] + a[i-1];
```

Dependences may prevent parallelization

Designing for concurrency



Parallelization strategy

Sequential program with 150 loops

- Where to look for potential parallelism?
- Which loops should be parallelized?
- Which loops cannot be parallelized?

Summary

- No more improvements of scalar performance
  - Power wall
  - Memory wall
  - ILP wall

  Brick wall
- Connection between deep learning and parallel computing
- Data parallelism usually more scalable than functional parallelism
- Development of parallel software occurs rarely from scratch
- When parallelizing a program, pay attention to
  - Correctness
  - Performance
  - Cost