CS 610: Compiler Challenges for Parallel Architectures

Swarnendu Biswas

Semester 2020-2021-I

CSE, IIT Kanpur

Content influenced by many excellent references, see References slide for acknowledgements.

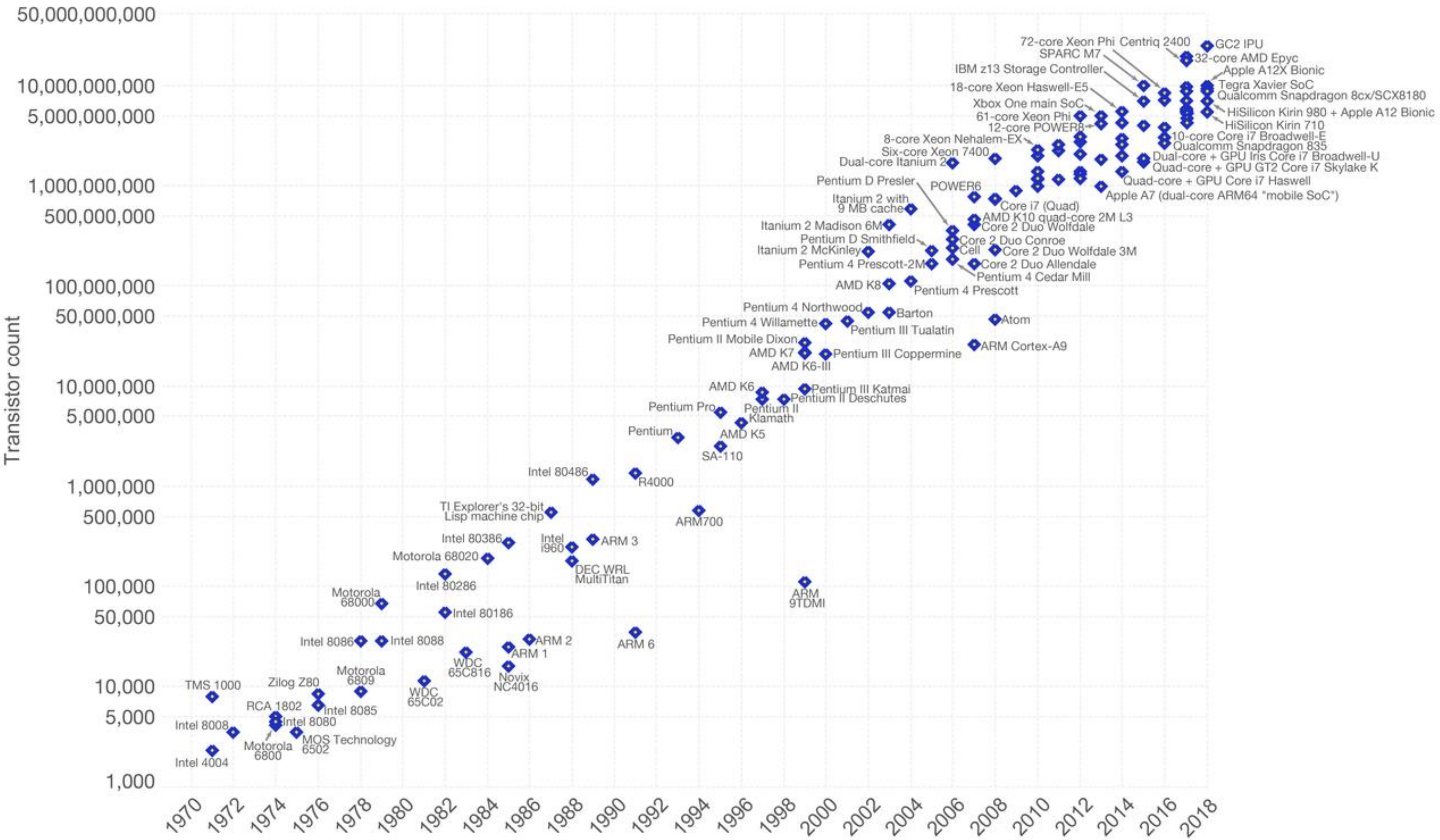
Copyright Information

Improvements in Computing Capabilities

- Last few decades have been exciting for the parallel computing community
- Improvements in computing capabilities
  i. Improvement in underlying technology (aka Moore's law)

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

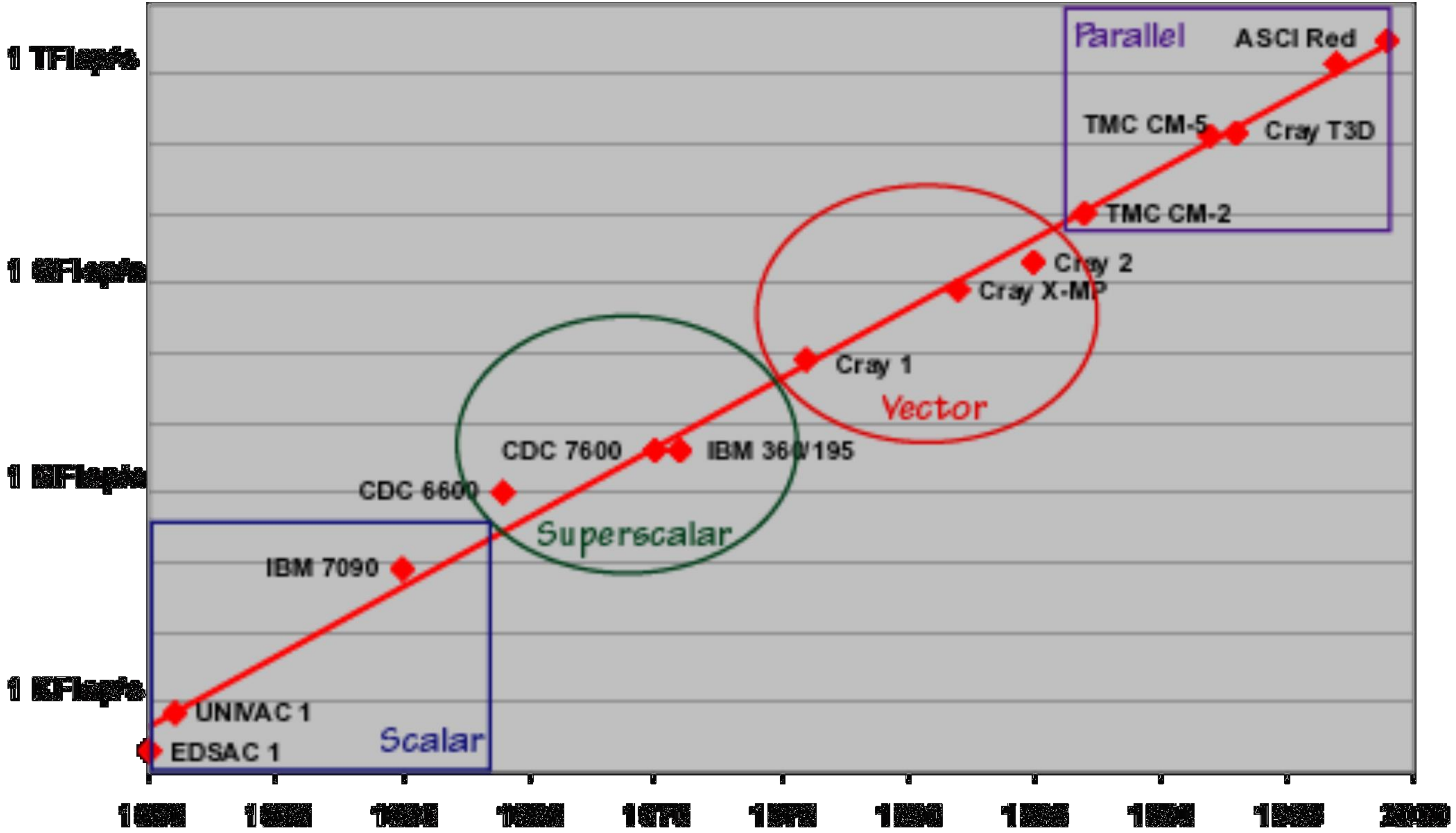
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

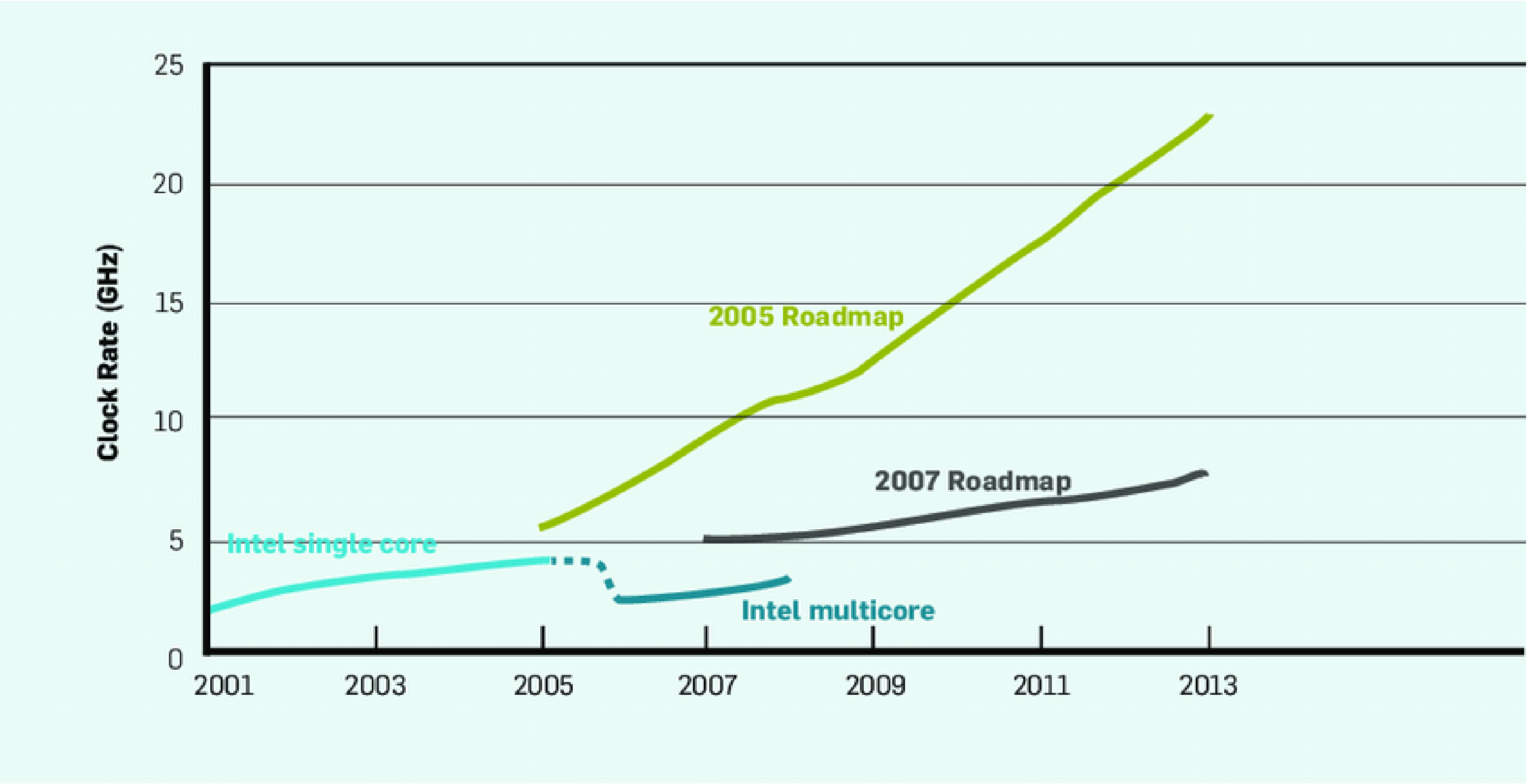
Improvements in Computing Capabilities

- Last few decades have been exciting for the parallel computing community
- Improvements in computing capabilities
  i. Improvement in underlying technology (aka Moore's law)
  ii. Advances in computer architecture
    - Instruction level parallelism (pipelining)
    - Multiple execution units
    - Vector operations
    - VLIW and Superscalar instruction issue
    - Deeper and sophisticated cache hierarchies



Optimizing Compilers for Modern Architectures – R. Allen and K. Kennedy

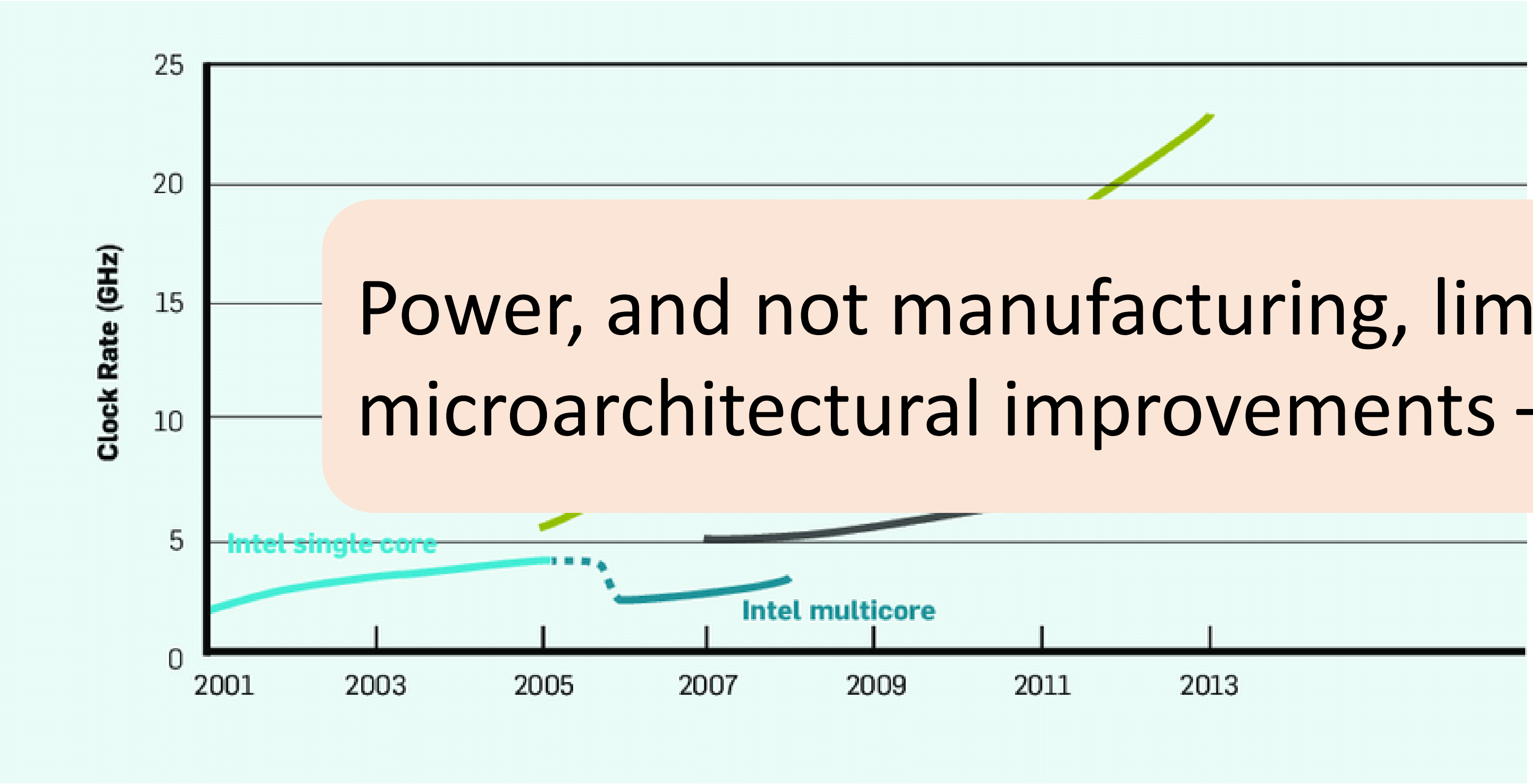
Challenges to Growth in Performance



Clock speeds are not increasing any more

K. Asanovic et al. A View of the Parallel Computing Landscape. CACM, Oct 2009.

Challenges to Growth in Performance

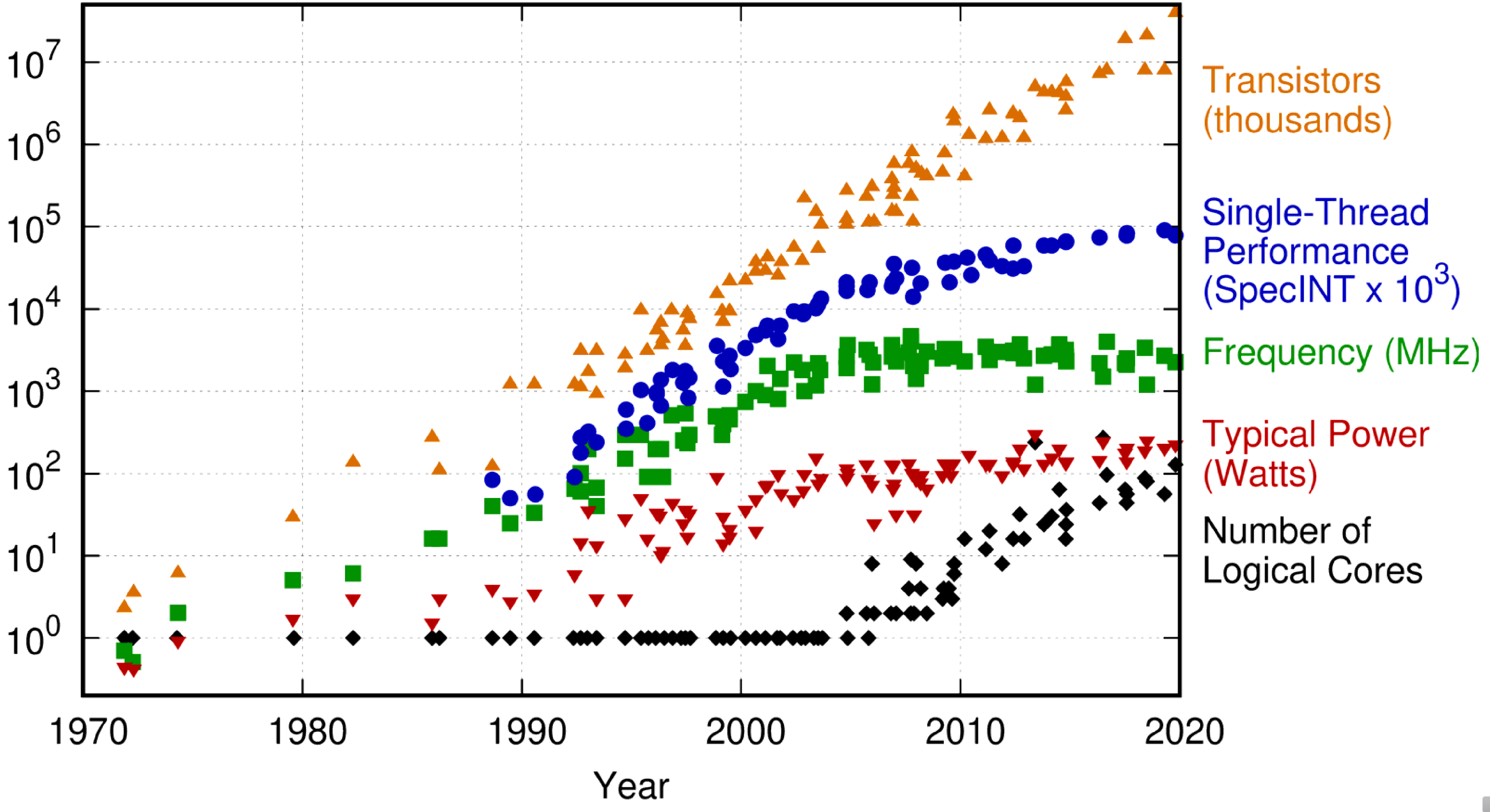


Power, and not manufacturing, limits microarchitectural improvements – F. Pollack

Clock speeds are
easing
ore

K. Asanovic et al. A View of the Parallel Computing Landscape. CACM, Oct 2009.

48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Hardware Trends in the Last Ten Years!

- 2005 – 2018
  - Single core performance increase is ~20%
- **Programs do not run any faster by themselves**

Programs Do Not Run Any Faster by Themselves!

- Microarchitectural techniques
  - Multiple functional units, superscalar architecture, VLIW, more cache structures (e.g., L4 caches), deeper pipelines

Programs Do Not Run Any Faster by Themselves!

- Microarchitectural techniques
  - Multiple functional units, superscalar architecture, VLIW, more cache st

Law of diminishing returns!

There is little or no more hidden parallelism (ILP) to be found

Programs Do Not Run Any Faster by Themselves!

- Microarchitectural techniques
  - Multiple functional units, superscalar architecture, VLIW, more cache structures (e.g., L4 caches), deeper pipelines

- Complex systems are more difficult to program efficiently
  - Systems programmers now need to be aware of memory hierarchies and other architectural features to fully exploit the potential of the hardware

Programs Do Not Run Any Faster by Themselves!

- Microarchitectural techniques
  - Multiple functional units, superscalar architecture, VLIW, more cache structures (e.g., L4 caches), deeper pipelines

Have you heard of ninja programmers?

> Intel MKL, Intel MKL-DNN, cuDNN and several other popular libraries are hand-optimized for best performance

- Co…
  - …ies and …hardware

What is the software side of the story?

Develop Parallel Programs

From my perspective, parallelism is the biggest challenge since high-level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful.

…

Industry is building parallel hardware, assuming people can use it. And I think there's a chance they'll fail since the software is not necessarily in place. So this is a gigantic challenge facing the computer science community.

– David Patterson, ACM Queue, 2006.

Develop Parallel Programs

To save the IT industry, researchers must demonstrate greater end-user value of from an increasing number of cores –
A View of Parallel Computing Landscape, CACM 2009.

New Challenges in Software Development

- Adapt to the changing hardware landscape
- Most applications are single-threaded

How can we develop software that makes effective use of the extra hardware?

Compilers to the rescue!

- A compiler is a system software that **translates** a program in a source language to an **equivalent** program in a target language

Compilers to the rescue!

- A compiler is a system software that **translates** a program in a source language to an **equivalent** program in a target language

Role of a compiler

- Generate correct code
- Improve the code according to some metric

Compilers to the rescue!

- Compiler technology has become more important as machines have become more complex
- Success of computer architecture innovations depends on the ability of compilers to provide efficient language implementations on that architecture

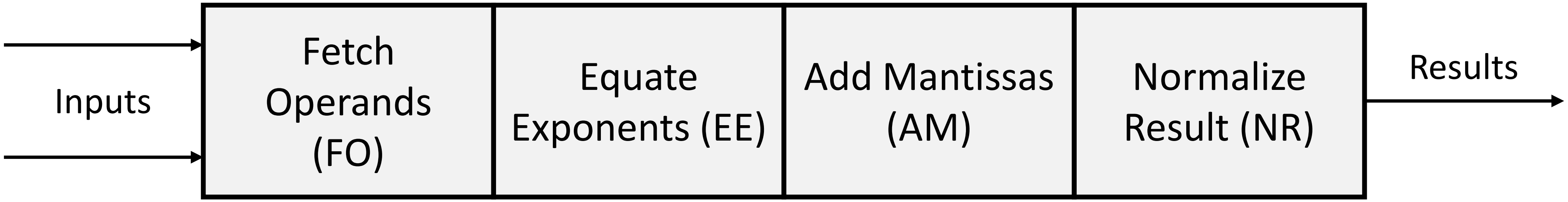
Compiling for Scalar Pipelines

- Pipelining subdivides a complex operation into independent microoperations so that, if the different microoperations use different resources, the microoperations can be overlapped by starting an operation as soon as its predecessor has completed the first microoperation

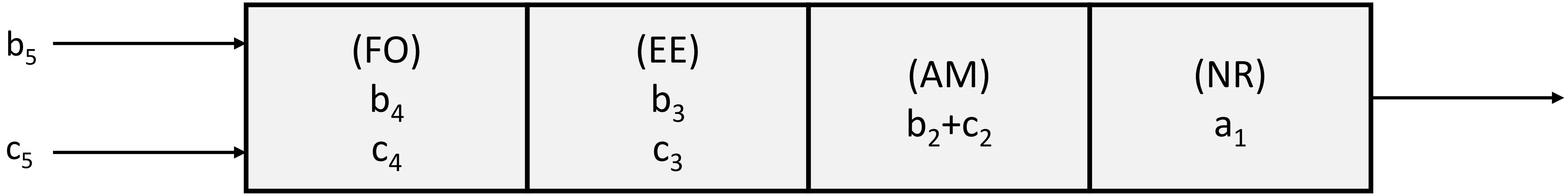
- A pipelined functional unit is effective only when the pipe is kept full; that is, only when there are operands available for operation on each segment clock cycle.

Compiling for Scalar Pipelines

Floating-point Adder

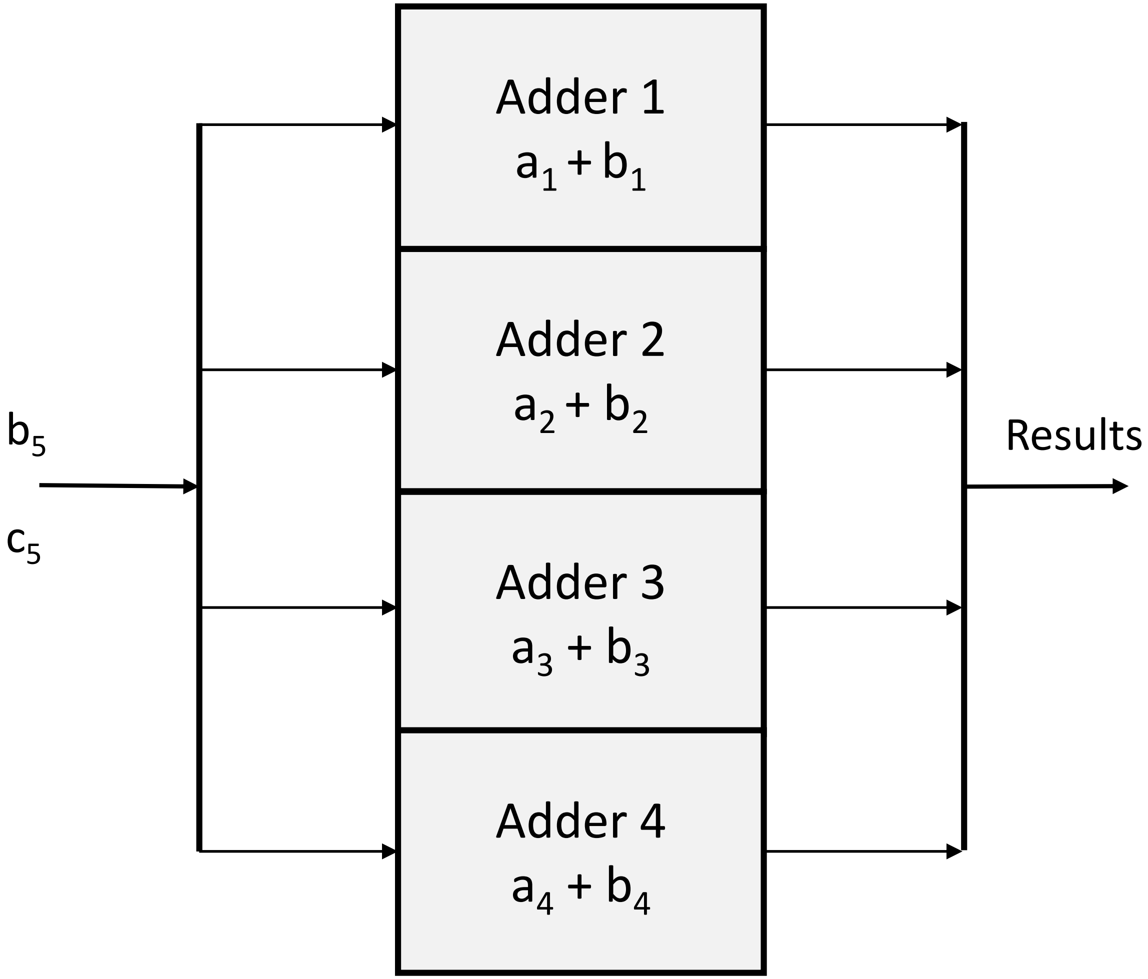


A pipelined execution unit computing $a_i = b_i + c_i$



Compiling for Scalar Pipelines

- Multiple functional units
- Can issue n/m operations per cycle
  - Assuming n units and m cycles for an operation to complete
- Also called fine-grained parallelism



Compiler Challenges with Pipelining

- The key performance barrier is pipeline stalls
  - A stall occurs when a new set of inputs cannot be injected into the pipeline because of a hazard
- Structural hazards – Available machine resources do not support instruction overlap
  - For example, if a machine has only one port to memory, it cannot overlap the fetch of instructions with the fetch of data
  - Such a hazard cannot be avoided through compiler strategies

Compiler Challenges with Pipelining

- Data hazards – Result produced by one instruction is needed by a later one

```
LW   R1, 0(R2)
ADD  R3, R1, R4
```

  - Compiler can schedule an instruction that does not use R1
- Control hazards – Occurs during processing of branch instructions

Vector Instructions

- Apply same operation to different positions of one or more arrays
  - Goal: keep pipelines of execution units full

```
VLOAD   V1,A
VLOAD   V2,B
VADD    V3,V1,V2
VSTORE  V3,C
```

→

```
C(1:64) = A(1:64) + B(1:64)
```

- Challenges
  - Increases processor state for the vector registers
  - Increases the cost of processor context switching
  - Expanded the instruct set complicating instruction decode
  - Can pollute the cache hierarchy

Compiler Challenges with Vector Instructions

```
DO I = 1, 64
  C(I) = A(I) * B(I)
ENDDO
```

```
DO I = 1, 64
  A(I+1) = A(I) + B(I)
ENDDO
```

Superscalar and VLIW Processors

- Goal is to issue multiple instructions on the same cycle
- Superscalar – looks ahead in the instruction stream and issues instructions that are ready to execute
- VLIW – executes a wide instruction per cycle
  - Usually one instruction slot per functional unit

- Challenges
  - Finding enough parallel instructions
  - Require more memory bandwidth

Compiling for Multiple-Issue Processors

- Compiler must recognize when operations are not related by dependence
  - Solution: vectorization
- Compiler must schedule instructions so that it requires as few total cycles as possible
  - Solution: instruction scheduling

Importance of Instruction Scheduling

- Assume a 2 cycle delay for loads from cache and for floating-point addition

```
LD      R1,A
LD      R2,B
FADD    R3,R1,R2
STD     X,R3
LD      R4,C
FADD    R5,R3,R4
STD     Y,R5
```

How many cycles?

Importance of Instruction Scheduling

- Assume a 2 cycle delay for loads from cache and for floating-point addition

```
LD      R1,A
LD      R2,B
FADD    R3,R1,R2
STD     X,R3
LD      R4,C
FADD    R5,R3,R4
STD     Y,R5
```

How many cycles?

```
LD      R1,A
LD      R2,B
LD      R4,C
FADD    R3,R1,R2
FADD    R5,R3,R4
STD     X,R3
STD     Y,R5
```

Scheduling in VLIW

```
LD      R1,A
LD      R2,B
FADD    R3,R1,R2
STD     X,R3
LD      R4,C
LD      R5,D
FADD    R6,R4,R5
STD     Y,R6
```

LD R1, A	LD R4, C
LD R2, B	LD R5, D
delay	delay
FADD R3, R1, R2	FADD R, R4, R5
STD X,R3	STD Y, R6

Processor Parallelism

- Synchronous parallelism
  - Replicate processors, with each processor executing the same program on different data
  - Data Parallelism – same task on different data
- Asynchronous parallelism
  - Replicate processors, but each processor can execute different programs
  - Requires explicit synchronization
  - Task Parallelism – independent tasks on same or different data

Compiling for Asynchronous Parallelism

```
PARALLEL DO I = 1, N
  A(I+1) = A(I) + B(I)
ENDDO
```

```
PARALLEL DO I = 1, N
  A(I-1) = A(I) + B(I)
ENDDO
```

Bernstein’s Conditions

- When is it safe to run two tasks R1 and R2 in parallel?

- If none of the following holds
  1. R1 writes into a memory location that R2 reads
  2. R2 writes into a memory location that R1 reads
  3. Both R1 and R2 write to the same memory location

Granularity of Parallelism

Vectorization

- Parallelism is finer-grained
- Synchronization overhead is small

Asynchronous Parallelism

- Parallelism is coarser-grained
- Larger start-up and synchronization overheads

Granularity of Parallelism

Vectorization

- Parallelism is finer-grained
- Synchronization overhead is small

Asynchronous Parallelism

- Parallelism is coarser-grained
- Larger start-up and synchronization overheads

Compilers should parallelize the outer loops and vectorize the inner ones

References

- R. Allen and K. Kennedy – Optimizing Compilers for Multicore Architectures, Chap 1.