GPU COMPUTING
LECTURE 01 - INTRODUCTION

Kazem Shekofteh
kazem.shekofteh@ziti.uni-heidelberg.de
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg

ABOUT ME

- Post-Doc at Heidelberg University, Germany
  - PhD from Ferdowsi University (GPU scheduling)
  - Visiting scientist, Heidelberg University (2016-2017)
- Performance, scalability
  - High-performance computing in medicine and bioinformatics (sequencing algorithms)
- This course: why GPUs
  - Walking faster …
  - Future applications

OBJECTIVES & PREREQUISITES

Objectives: The students …

- know the architecture of GPUs
- are able to create simple GPU programs in CUDA
- understand the factors that determine the performance of GPU programs
- are able to optimize GPU programs for better performance

Methodology

- Lectures focus on current architectures, trends, technology constraints
- Exercises: Practical hands-on, reading
- Excessive programming & some paper reviewing

Prerequisites

- Required: C++, Linux, computer architecture basics, OS basics
- Recommended: parallel courses “Parallel Computer Architecture”, “Introduction to HPC”, & “Advanced Parallel Computing”

ORGANIZATION

Lectures – 2 hours/week - kazem.shekofteh@ziti.uni-heidelberg.de

- Tuesday, 14:00

Exercises – 2 hours/week - Christian Alles (christian.alles@stud.uni-heidelberg.de)

- Tuesday, 16:00
- Groups of up to three students allowed – individual work must be visible
- Mixture of reading/exercises/programming/experiments

One final oral or written exam

- Prerequisite: 75% of all exercise points, bonus points for “willingness to present” (best case: 50% of all exercise points)

ASSIGNMENTS

Practical exercises: usually coding & experiments

Reading & feedback based on paper review

- Ideal review here is 3 sentences for each of the following:
  1. Primary contribution
  2. Key insight of the contribution
  3. Your opinion/reaction to the content
- Review: rating relative to all other papers (of this venue)
  - Strong reject, weak reject, weak accept, accept
- Old papers: optionally include some comments on how right this paper was

Provide review (summary) using Moodle until next Monday 09:00

- Discussion round as part of the exercise

Agenda - Course “GPU Computing” winter 2022/23

		Lecture	Exercise	
18.10.22	1	Intro		Reading
25.10.22	2	CUDA	Kernel launch, data movement	
01.11.22	-	-		
08.11.22	3	Basic Architecture		Global memory
15.11.22	4	MMULT	Shared memory	
22.11.22	5	Parallel Computing		MMULT GPU + Optimizations (Bonus)
29.11.22	6	Profiling		
06.12.22	7	Scheduling Optimizations	Reduction	
13.12.22	8	N-Body Methods		
20.12.22	9	Host Optimizations		n-Body + Optimizations (Bonus)
27.12.22	xmas			
03.01.23	xmas			
10.01.23	10	Productivity		
17.01.23	11	Stencil Codes	Stencil in OpenACC	
24.01.23	12	GPU Programming Models		
31.01.23	13	Consistency & Coherence		
14.02.23	Exam			

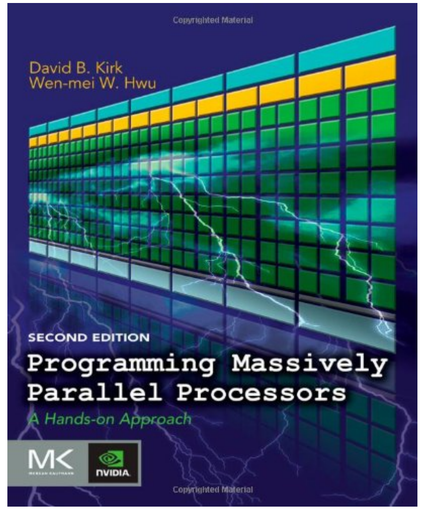
ADDITIONAL READING

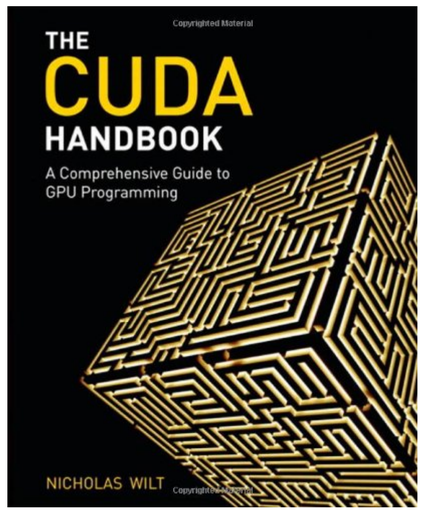
Books

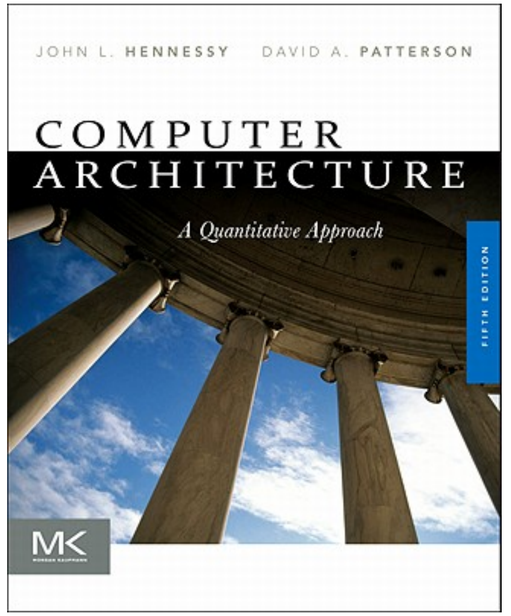
Kirk, Hwu: Programming Massively Parallel Processors, Elsevier, 2012

Wilt: CUDA Handbook, Addison-Wesley, 2013

Hennessy/Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann







Publications/Conferences

ISCA, HDCA, ASPLOS, PACT, IPDPS, ICPP, ISPASS, HPDC, ...

See ACM/IEEE websites (or author‘s web site for limited copies)

http://developer.nvidia.com/category/zone/cuda-zone

http://www.gputechconf.com

QUESTIONNAIRE

SISD vs. SIMD

Pthread/OpenMP

Data-parallel language

TLB

Cache misses capacity vs. conflict

Double precision

GPU OVERVIEW

MOTIVATION

Let’s go camping together!

GPU BACKGROUND

Primary use in gaming

- Each console has a (powerful) GPU
- Meantime photorealistic

Graphics: big, multi-dimensional floating-point operations in parallel

- Programmable

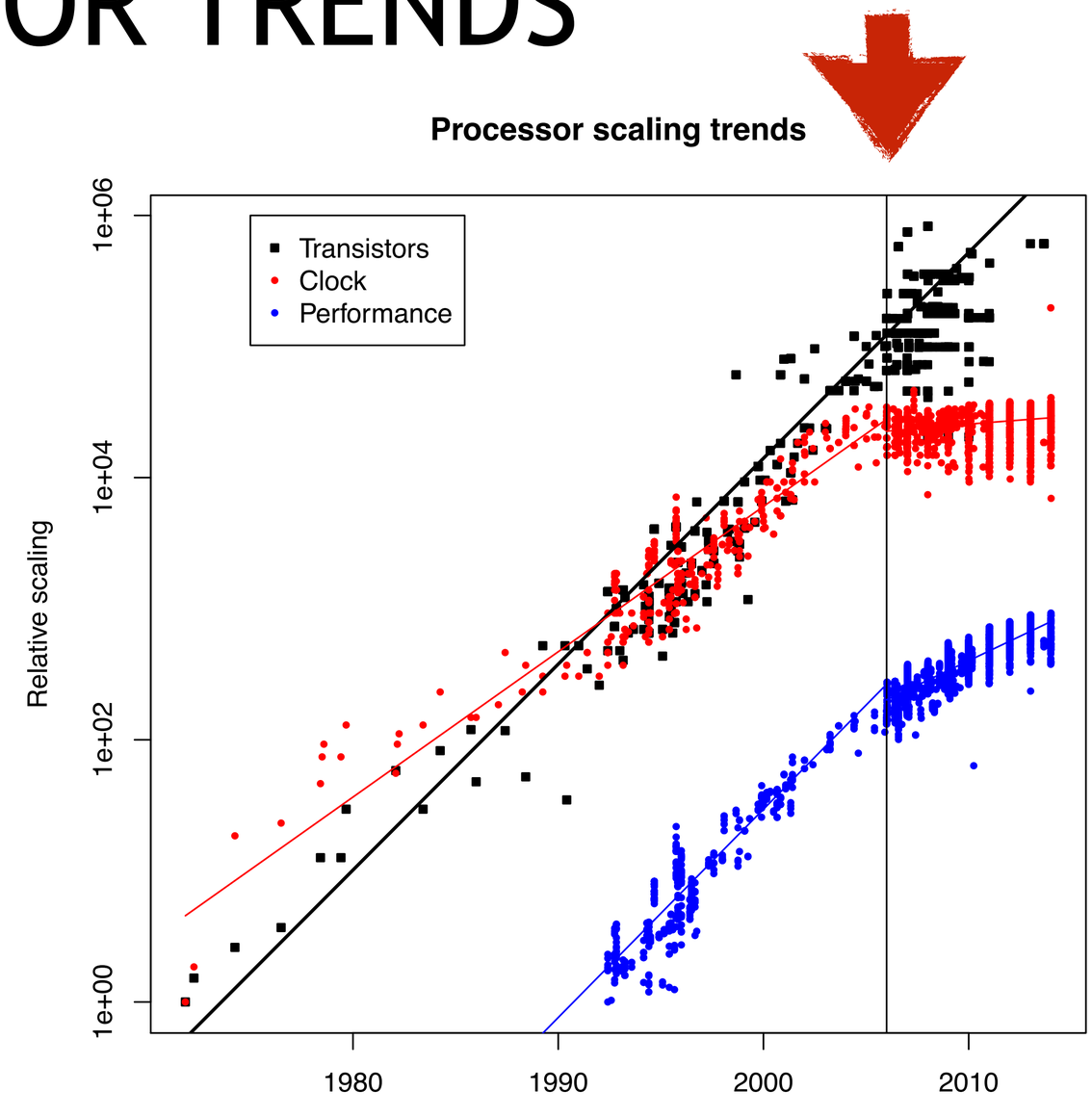
Since ~2007 used for general-purpose computing

- CUDA

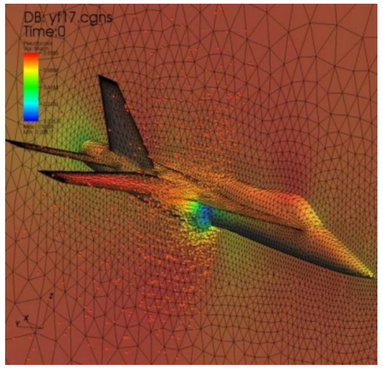
NVIDIA

https://80.lv/articles/ue4arch-interview-about-realiscit-environments-in-ue4/

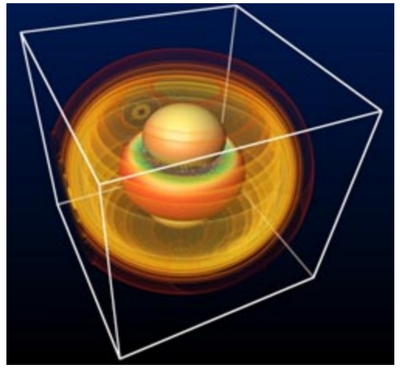
PROCESSOR TRENDS



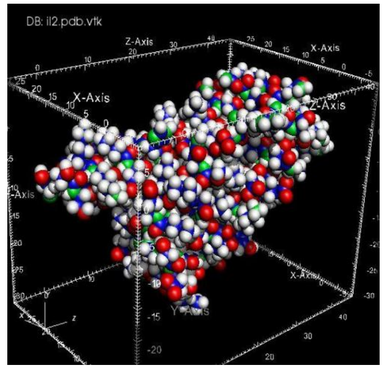
data from: http://cpudb.stanford.edu



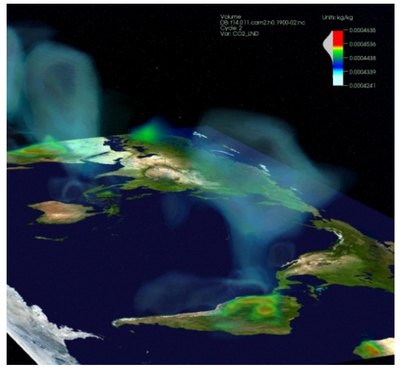
CFD



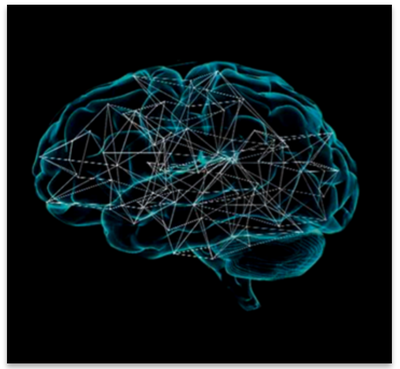
Cosmology



Molecular Dynamics



Weather/climate research

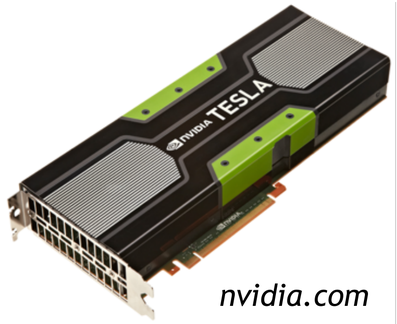


Machine Learning

Tera-FLOP/s
September 1997

15 years

Tera-FLOP/s
November 2012



nvidia.com

A cluster of GPUs:

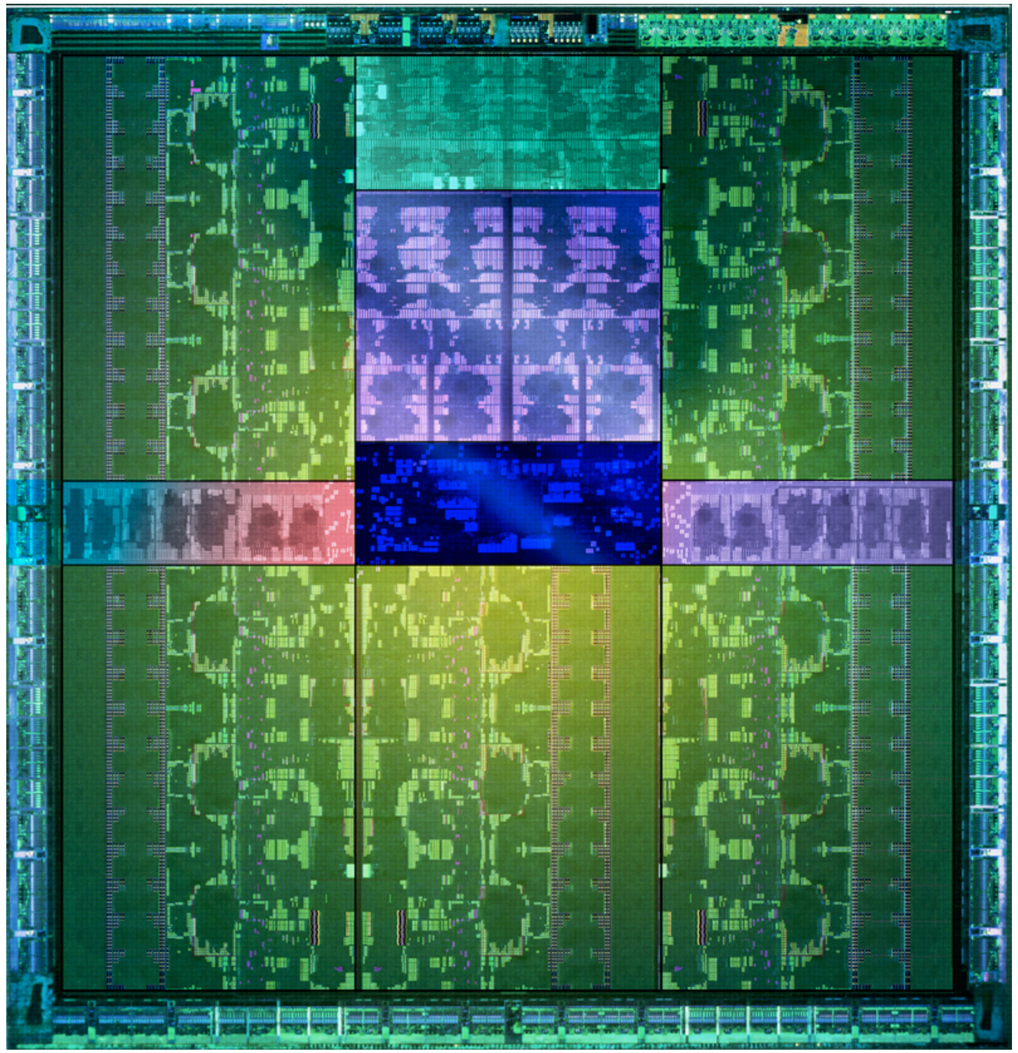
Peta-FLOP/s
November 2012

?

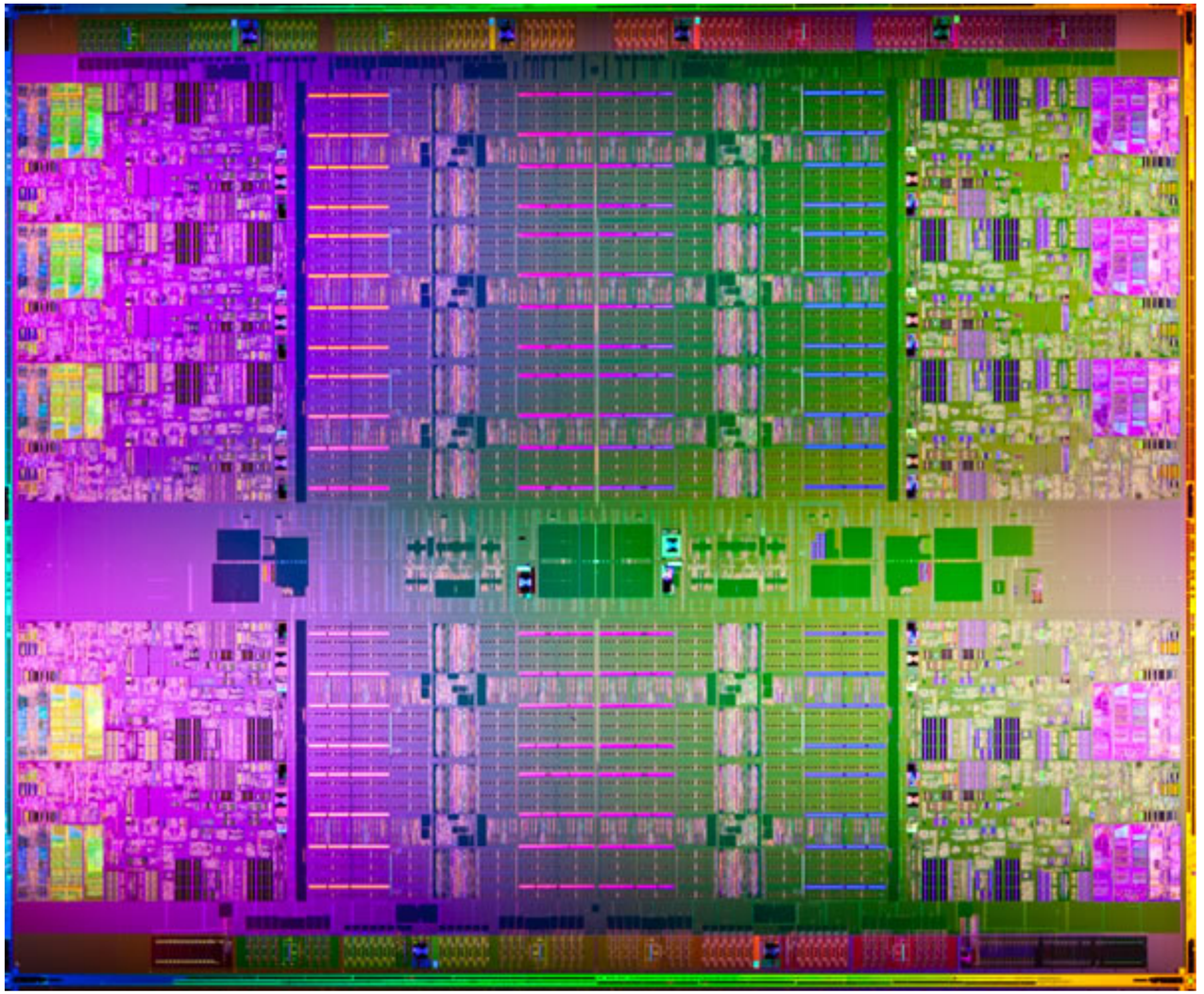
Exa-FLOP/s
2018?

?

DIE SHOTS - CPU OR GPU?



NVIDIA Kepler- GK110



Intel Xeon E7 - Westmere-EX

	Xeon E5-2699v4 (Broadwell, 2016)	Tesla K20 (GK110) (Kepler, 2012)	NVIDIA P100 (GP100) (Pascal, 2016)	NVIDIA V100 (GV100) (Volta, 2017)
		~3-4	~1.5	
Core count	22 cores 2 FP-ALUs/core	13 SMs 64/SM(DP), 192/SM(SP)	56 SMs 32/SM(DP), 64/SM(SP)	84 SMs 32/SM(DP), 64/SM(SP)
Frequency	2.2-3.6GHz	0.7GHz	1.328-1.480GHz	1.455GHz
Effective vector width	256bit (SP/DP) AVX 2.0	1024bit (SP), 2048bit (DP) static grouping		1024bit (SP), 2048bit (DP) dynamic grouping
Peak Perf.	633.6 GF/s (DP)	1,165 GF/s (DP), SP x3	5.3 TF/s (DP), SP x2	7.5 TF/s (DP), SP x2
Use mode	latency-oriented		throughput-oriented	
Latency	minimization		toleration	
Programming	10s of threads		10,000s+ of threads	
Memory bandwidth	76.8 GB/s 128bit DDR4-2400	250 GB/s 384-bit GDDR-5	720 GB/s 4096-bit HBM2	
Memory	1.54TB	5 GB	16G	32G
Die size	456 mm^2	550mm^2	610mm^2	815mm^2
Transistor	7.2 billion	7.1 billion	15.3 billion	21.1 billion
Technology	14nm	28nm	16nm FinFET	12 nm FFN
Power	145W	250W	300W	300W
Power efficiency	4.37 GF/Watt (DP) 8.74 GF/Watt (SP)	4.66 GF/Watt (DP) 14 GF/Watt (SP)	17.66 GF/Watt (DP) 35 GF/Watt (SP)	25 GF/Watt (DP) 50 GF/Watt (SP)

BULK-SYNCHRONOUS PARALLEL

In 1990, Valiant already described GPU computing pretty well

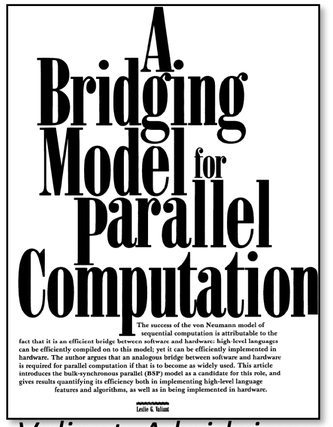
Superstep

- Compute, communicate, synchronize

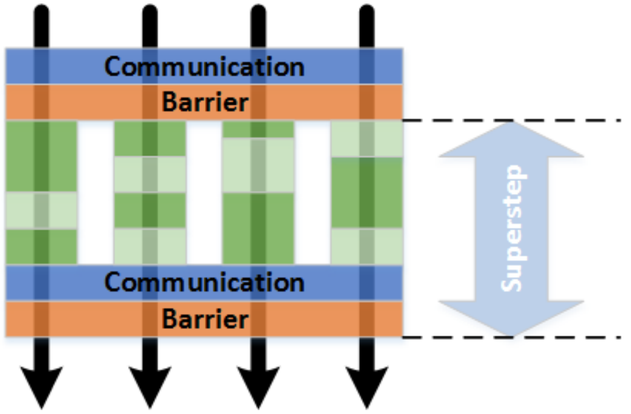
Parallel slackness: # of virtual processors v, physical processors p

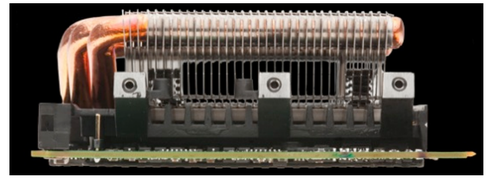
- v = 1: not viable
- v = p: unpromising wrt optimality
- v >> p: leverage slack to schedule and pipeline computation and communication efficiently

Extremely scalable, bad for unstructured parallelism



Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990





OUR VIEW OF A GPU

Software view: a programmable many-core scalar architecture

- Huge amount of scalar threads to exploit parallel slackness, operates in lock-step
- SIMT: single instruction, multiple threads

IT'S A (ALMOST) PERFECT INCARNATION OF THE BSP MODEL

Hardware view: a programmable multi-core vector architecture

- SIMD: single instruction, multiple data
- Illusion of scalar threads: hardware packs them into compound units

IT'S A VECTOR ARCHITECTURE THAT HIDES ITS VECTOR UNITS

SCALING RULES

Moore vs. Amdahl

MOORE'S LAW

Gordon Moore

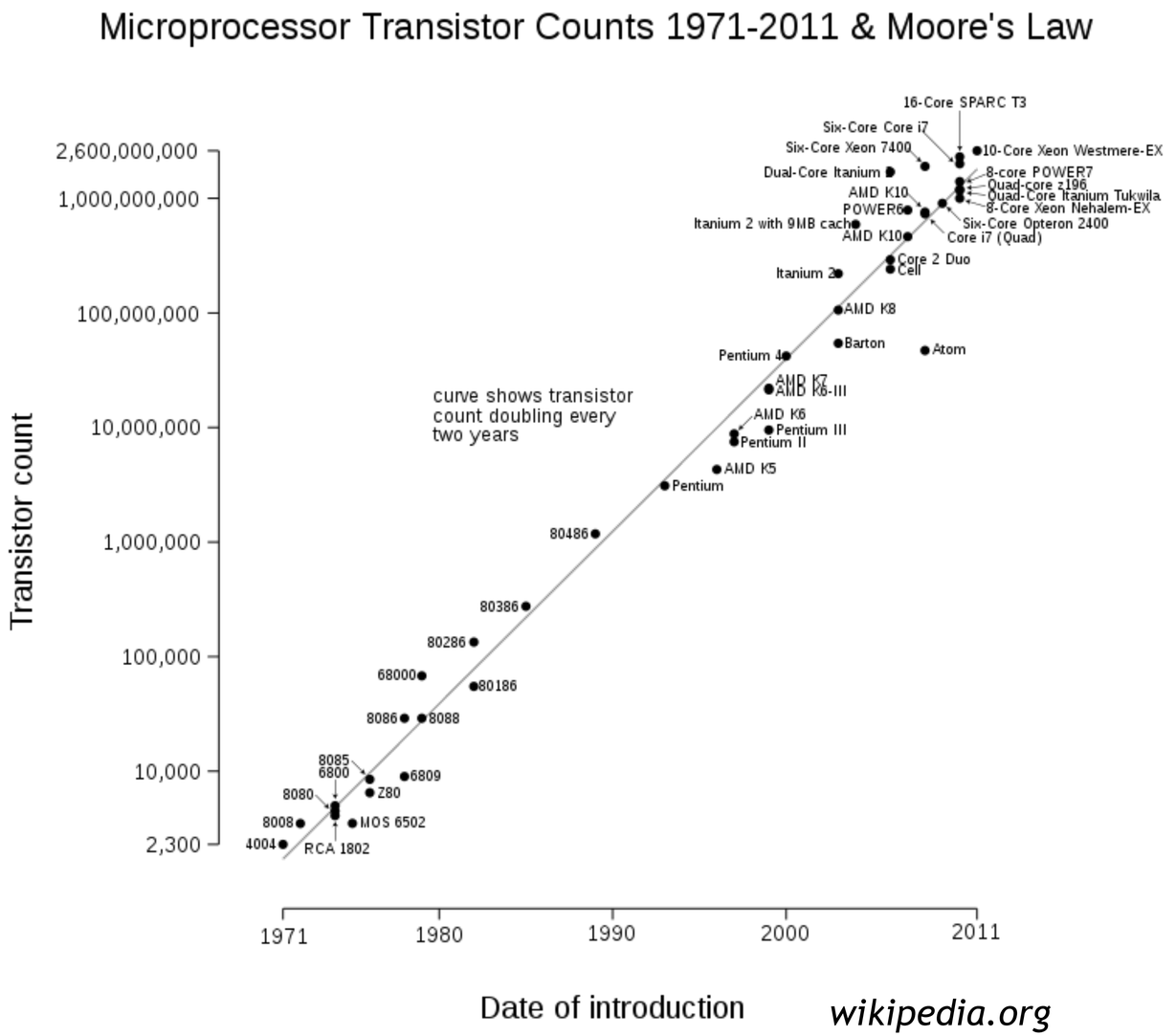
- 1965: Doubling each year
- 1975: Transistor count of ICs doubling every two years

Derived "laws"

- CPU performance doubling every 18 months
- Memory size four times every three years
- Memory performance doubling every 10 years
- At the same costs double performance every two years

intel.com

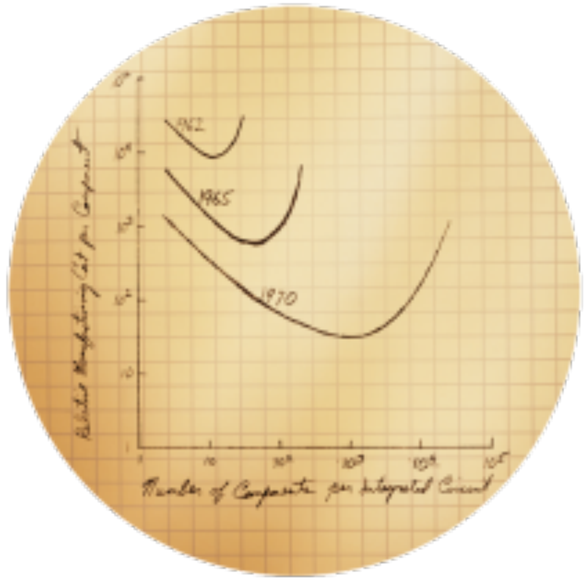
MOORE’S LAW - HISTORY



wikipedia.org

MOORE’S LAW - FUTURE

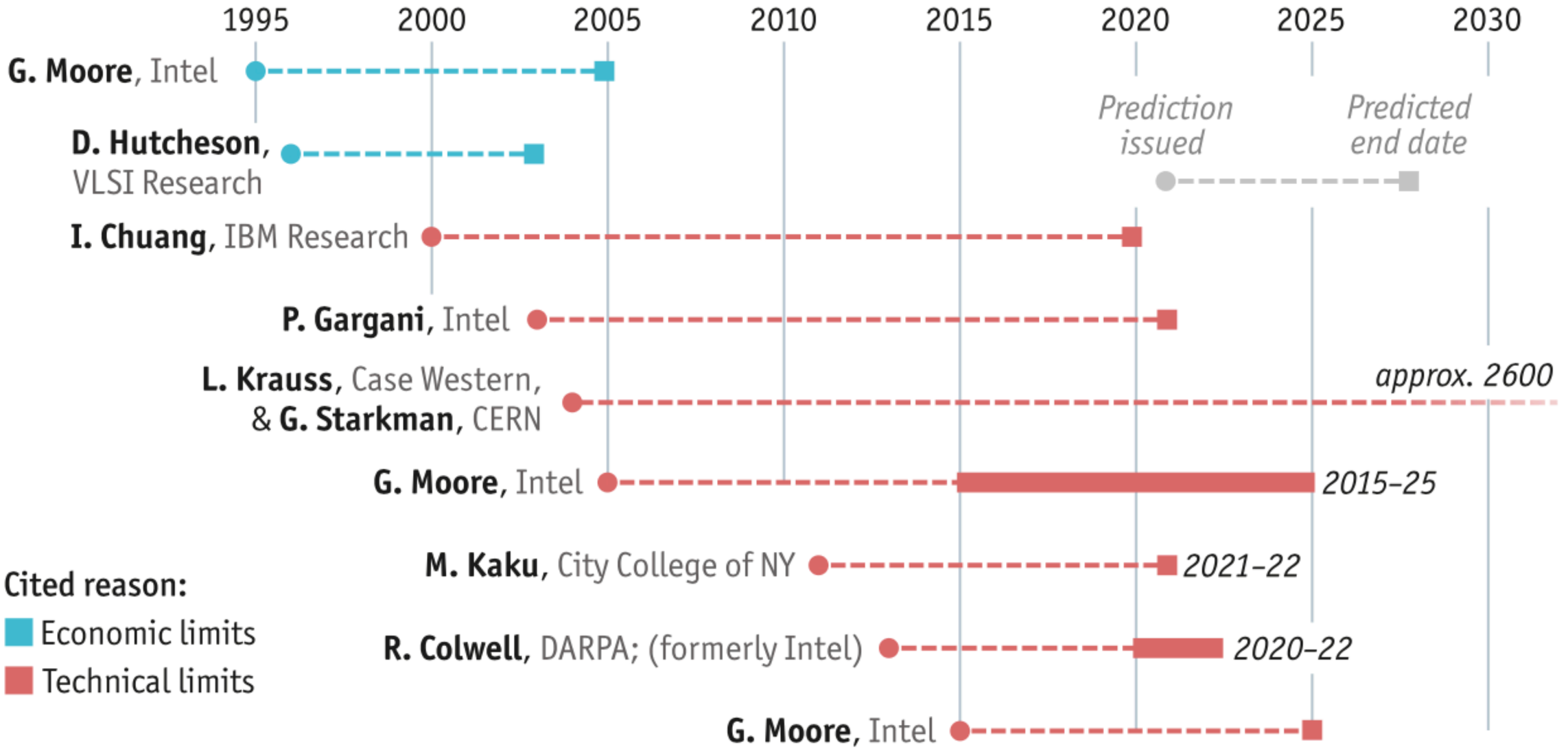
- Industry is trying to keep the pace
  - Self-fulfilling prophecy
  - “positive feedback between belief and behavior”
- Atoms as fundamental lower bound
  - Even then, increase of die size can maintain the law
  - Intel’s statements about end of Moore’s law
    - 2003: 2013-2018
    - 2005: until 2015
    - 2008: until 2029
- Bernie Meyerson (IBM): 7-9nm is the limit
  - Quantum mechanics effects



https://www.economist.com/technology-quarterly/2016-03-12/after-moores-law

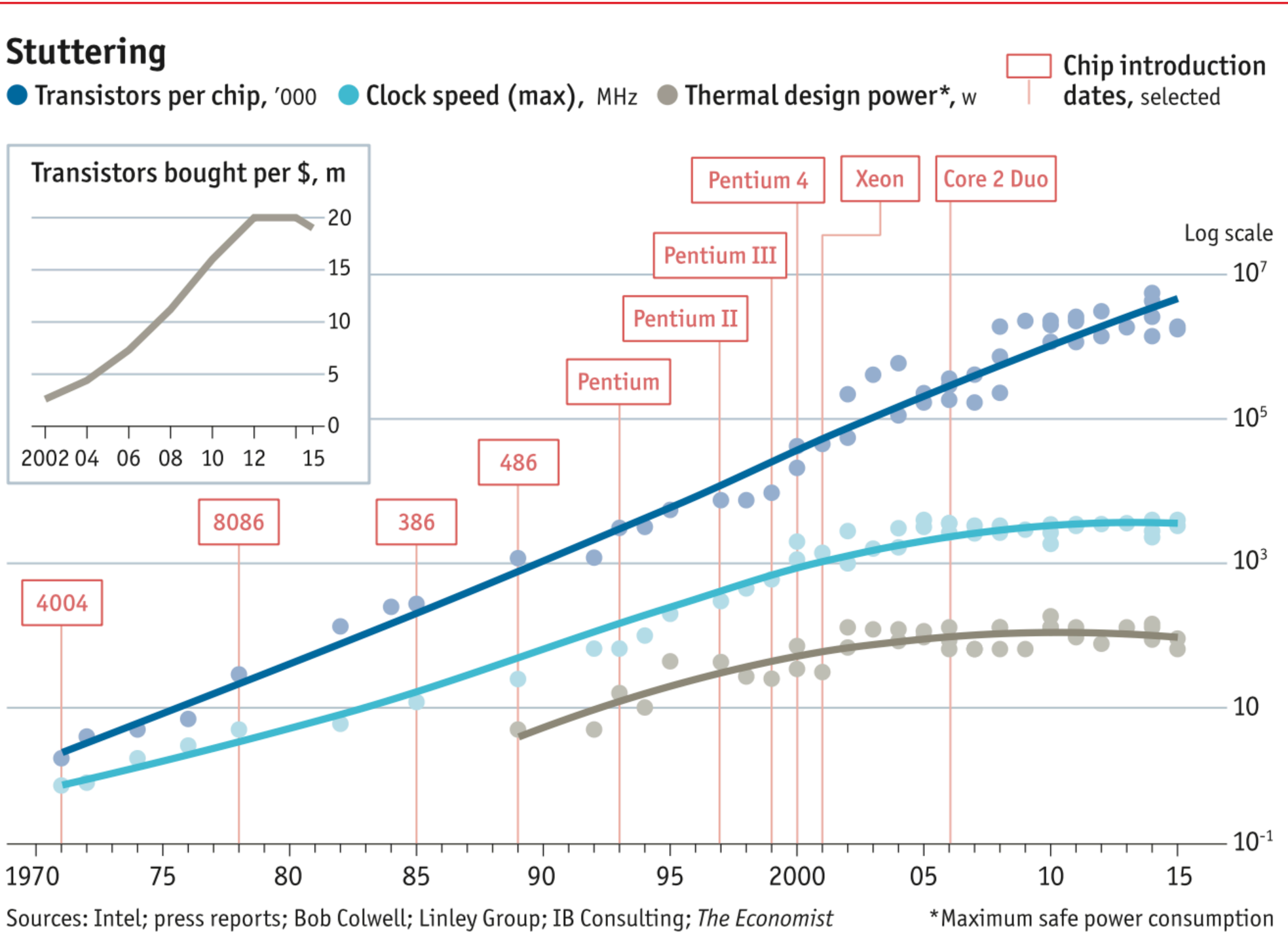
Faith no Moore

Selected predictions for the end of Moore's law

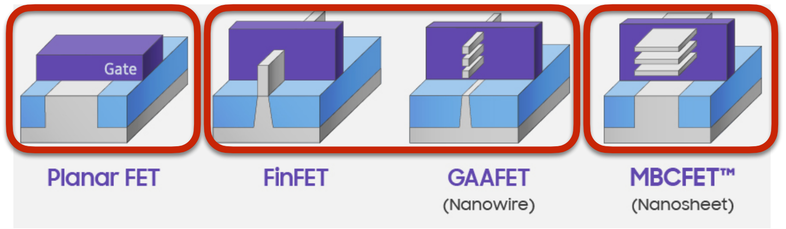


Sources: Intel; press reports; *The Economist*

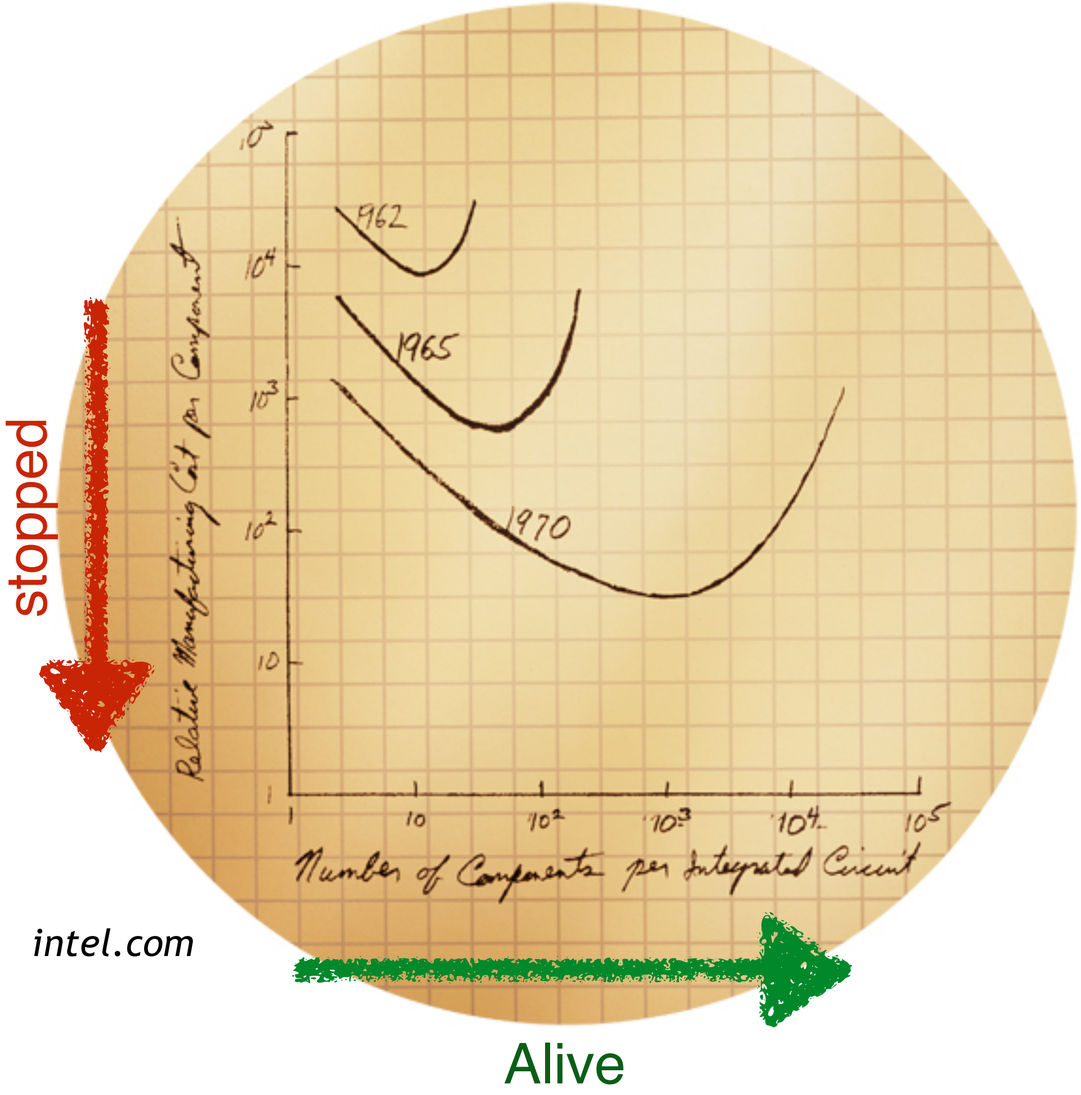
https://www.economist.com/technology-quarterly/2016-03-12/after-moores-law



MOORE'S LAW



samsung.com



intel.com

Transistors still getting smaller

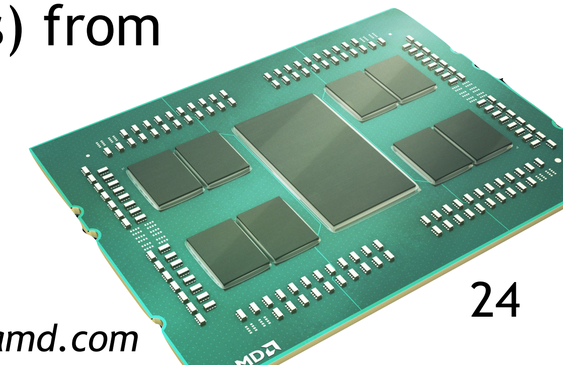
- Albeit end in sight (probably 3nm)
- Careful: prototyping != mass production
- New devices: TFET, FEFET (Thomas Theis, 2017)

Chips get larger

- Size limited by reticle (some NVIDIA GPUs are already at the max)

Chiplets (lego-like silicon bricks)

- See Multi-Chip Modules (MCMs) from the 80s/90s
- 2nd gen AMD Epyc: 8 chiplets



amd.com

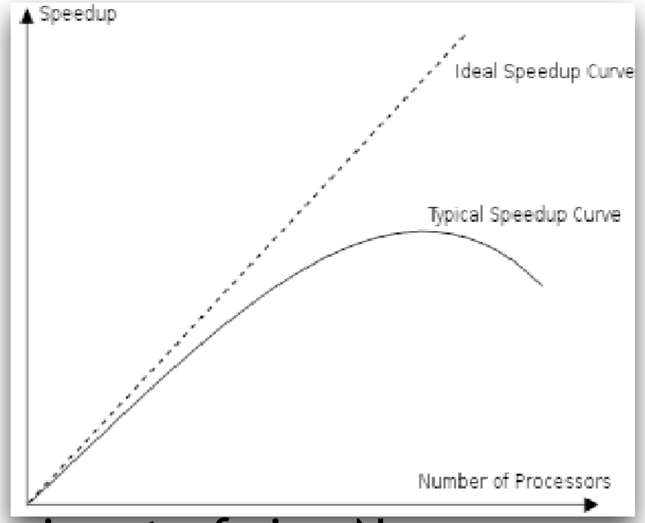
SPEED-UP

Speed-up: “How much faster can one program be executed”

- Assumption: instead of one resource, N identical resources are available
- Naive: P resources yield an execution time of 1/P
- No overhead assumed

Reality: significant loss

- Break-even point when execution time starts to increase again



For a given algorithm:

- t_{old} = SerTime(N) = time of the best serial implementation for an input of size N
- t_{new} = ParTime(N,P) = time of the parallel implementation, using P parallel units

Speedup(P,N) = SerTime(N) / ParTime(N,P)

Efficiency(P,N) = SerTime(N) / (P * ParTime(N,P)) ***usually expressed in percentage***

SPEED-UP - REGIMES

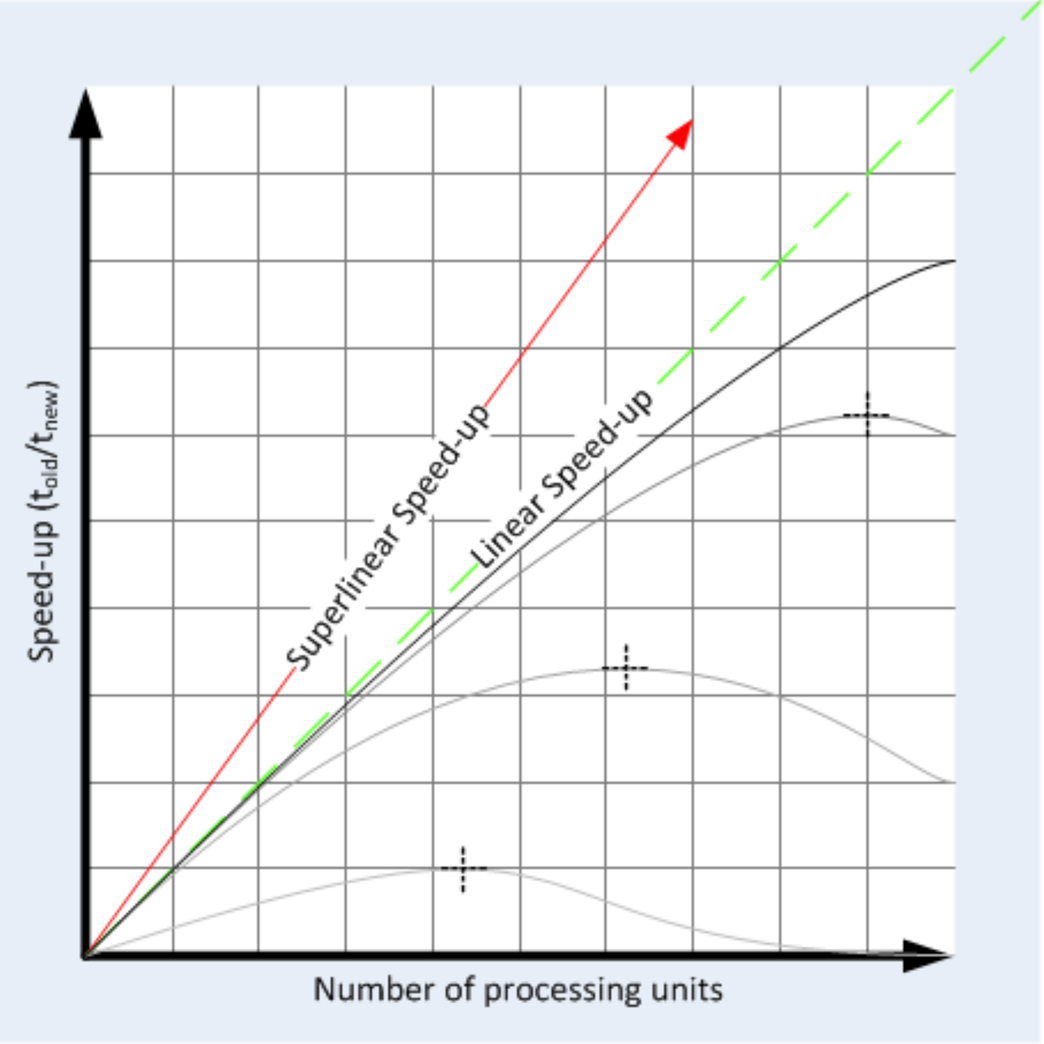
1 <= Speedup(p) <= p

0 <= Efficiency(p) <= 1

Linear speed-up: Speedup(p) = p

Superlinear speed-up: Speedup(p) > p

Usually not possible



AMDAHL'S LAW

Model to find the maximum improvement in terms of performance

Assumption: only a fraction of the execution time can be parallelized (parallel fraction P).

Assumption: the other fraction is the serial one: serial fraction S

Then: P + S = 1

As fraction P is processed in parallel, this fraction of time is reduced (N parallel execution units)

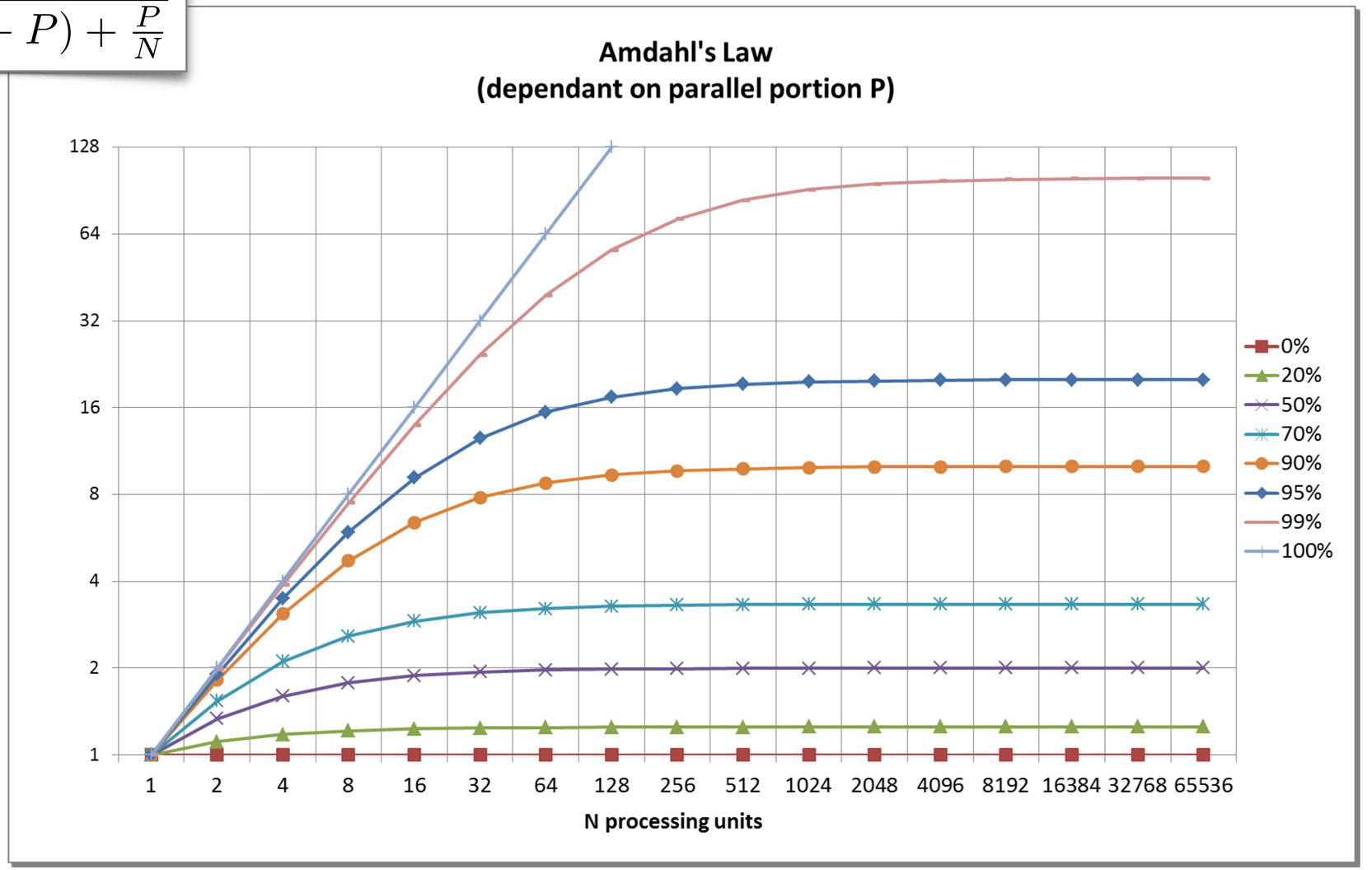
$$Speedup = \frac{1}{(1-P) + \frac{P}{N}}$$

Notes

Speed-up has an upper limit dependent on S, not on N!

AMDAHL'S LAW

$$Speedup = \frac{1}{(1-P) + \frac{P}{N}}$$



NOTES ON AMDAHL AND HIS LAW

Amdahl himself …

1. …. wanted to claim that parallel computing is not viable

   "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities“, AFIPS Conference Proceedings, 1967.

2…. was an optimist

   Extra work is required for parallelization

   Synchronization, communication, management, …

   In this regard his law is too optimistic

3…. was a pessimist

   We can (have to?) scale the problem size with N

   Gustafson‘s law – superlinear speedup (1988)

   Parallel algorithms exist that reduce fraction S

   Superlinear speed-up due to caching effects

SUMMARY

GPU Computing is using GPUs for non-graphical computations

- More performance (compute, memory)
- Better energy-efficiency (how the term picoJoule is getting more and more attention)

Key differences to a CPU

- Much (many much’es) more parallelism
- Latency is not minimized, but tolerated
- Offload compute model
- No general-purpose programming (yet?)
- Memory capacity is small
- Single-thread performance is a nightmare

More reading

- https://www.economist.com/technology-quarterly/2016-03-12/after-moores-law