

# Ultrasound Signal Processing with GPUs – Introduction to Parallel Programming



---

INTRO



# License / Attribution



- Materials for the short-course „**Ultrasound Signal Processing with GPUs – Introduction to Parallel Programming**” are licensed by us4us Ltd. the IPPT PAN under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).
- Some slides and examples are borrowed from the course „**The GPU Teaching Kit**” that is licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).
  - All the borrowed slides are marked with  



# Short-course Team

- This course is a collaboration effort between:
  - [us4us Ltd.](#) (company)
  - [IPPT PAN](#) (academia)
  - [University of Waterloo](#) (academia)
- A team of people have given their time to help prepare and organize this training:
  - Ziemowit Klimonda,
  - Mateusz Walczak,
  - Piotr Karwat,
  - Damian Cacko,
  - Julia Lewandowska.

**THANKS GUYS!!!**



**Marcin Lewandowski** [marcin@us4us.eu](mailto:marcin@us4us.eu), PhD / **us4us Ltd.** / **IPPT PAN**

Since receiving a Masters degree in Physics followed by a PhD in Electronic Engineering, Marcin Lewandowski has headed many projects in R&D, commercial product design and medical devices development and certification. He has also authored numerous publications in scientific journals on the medical and industrial applications of ultrasound. Over the course of 25+ years working in ultrasound, electronics and software development, Marcin has strived to apply his research expertise in projects with a strong potential for innovation and commercialization. Today, he balances his continued work in science with his role as CEO at us4us Ltd., who produce original ultrasound platforms for research, biomedical and industrial applications.



**Piotr Jarosik** <[piotr.jarosik@us4us.eu](mailto:piotr.jarosik@us4us.eu)> / **us4us Ltd.** / **IPPT PAN**

Piotr Jarosik received his Bachelor's and Masters Degree in Computer Science at Faculty of Electronics and Information Technology, Warsaw University of Technology. Currently, he is a Software Engineer at us4us and a PhD student at IPPT PAN. His research interests include machine learning and ultrasound data processing.

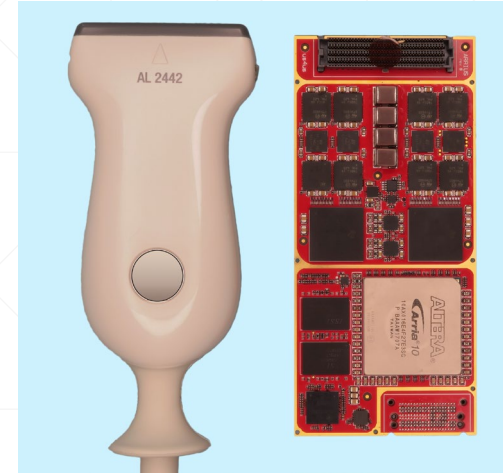


**Billy Yiu** [billy.yiu@uwaterloo.ca](mailto:billy.yiu@uwaterloo.ca), PhD / **University of Waterloo**

Billy Y. S. Yiu received his B.Eng. degree (Hons.) in medical engineering and the M.Phil degree in electrical and electronic engineering from The University of Hong Kong in 2007 and 2010, respectively. He was awarded his Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2019. He was a research staff member with the Biomedical Ultrasound Laboratory in The University of Hong Kong from 2010 to 2016, and currently serves as an associate scientist with the Laboratory on Innovative Technology in Medical Ultrasound (LITMUS), University of Waterloo. Billy's research interests are in advanced ultrasound imaging techniques and systems innovations; GPU implementation of these novel imaging techniques on research platforms for real-time performance has been of his long-standing interest dating back to 2010. Some of his contributions include high-frame-rate DAS beamforming, adaptive beamforming, eigen-based clutter filtering and speckle imaging in which he has published several journal articles based on these works.

# What we do?

- Produce and apply programmable ultrasound research systems (us4R™, us4R-lite™).
- **Ultrasound R&D:**
  - medical imaging and non-destructive testing
  - signal processing and algorithm implementation
- **Electronic Design:**
  - advanced multichannel signal processing electronics
  - high-speed digital design with FPGAs
  - high performance digital signal processing (FPGA, GPU, DSP)
- **Ultrasound Measurements:**
  - electrical/acoustical characterization of ultrasound transducers
  - acoustic output conformance with medical standards verification
- **Product Development / Medical Devices:**
  - design and development of medical devices
  - software development
  - model-making & prototype development



us4OEM™



us4R-lite™



us4R™

# Short-course Goals

- Get a top-level overview of GPU and parallel programming:
  - Understand GPU and NVIDIA CUDA multiprocessors architecture
  - How-to use Python tools for GPU programming
  - How-to apply GPU programming for ultrasound signal processing
- Technical topics:
  - Setup GPU programming environment (CUDA, Python, Numba, Colab)
  - Parallel programming API, tools and techniques
  - GPU memory architecture
  - Thread execution on GPU
  - Performance optimization (data transfer, processing)



# SCHEDULE

SECTION	TYPE (est. TIME)	PRESENTER
INTRO	Lecture (20 min)	Marcin
CUDA Architecture <ul style="list-style-type: none"><li>▪ CUDA Programming Model and Toolkit</li></ul>	Lecture (30 min) Jupyter (60 min)	Marcin Piotr
GPU Memory & Performance <ul style="list-style-type: none"><li>▪ CUDA Memory Model</li><li>▪ Performance guidelines</li></ul>	Lecture (30 min) Jupyter (60 min) Jupyter (30 min)	Marcin Piotr Piotr
US Research Systems <ul style="list-style-type: none"><li>▪ CUDA streams and processing</li></ul>	Lecture (30 min) Jupyter (40 min)	Marcin Piotr
Case-study: Color Doppler processing	Jupyter (30 min)	Piotr
Case-study: Color-encoded speckle imaging platform for real-time complex flow visualization in-vivo	Presentation (30 min)	Billy

# Short-course Materials

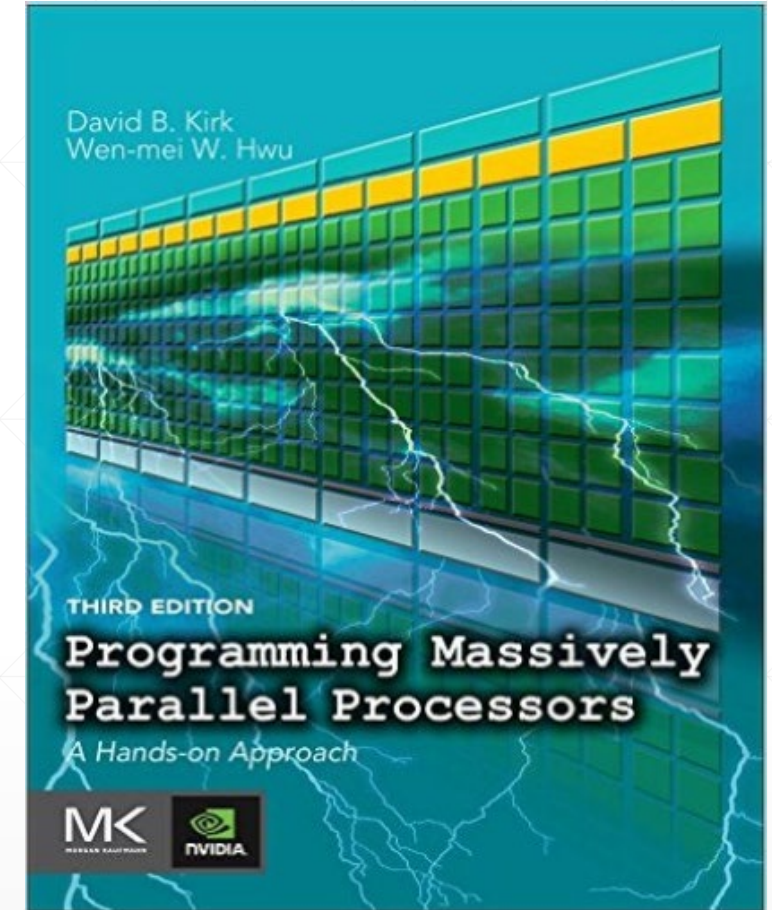
- All the short-course materials are freely available on-line:
  - <https://github.com/Lab4US/gpu-short-course>
  - [YouTube playlist](#) or [Vimeo playlist](#)
- Resources:
  - Lectures, excercises, and case-studies
  - Course slides
  - GPU examples (Jupyter Notebooks)
  - Example RF signal datasets



# The GPU Teaching Kit

## Full-term course – 29 modules training:

- Course Introduction
- Introduction to CUDA C
- CUDA Parallelism Model
- Memory Model and Locality
- Kernel-based Parallel Programming
- Performance Considerations: Memory
- Atomic Operations
- Parallel Computation Patterns (Part 1)
- Parallel Computation Patterns (Part 2)
- Performance Considerations: Parallel Computation Patterns
- Parallel Computation Patterns (Part 3)
- Performance Considerations: Scan Applications
- Advanced CUDA Memory Model
- Floating Point Considerations
- GPU as part of the PC Architecture
- Efficient Host-Device Data Transfer
- Application Case Study: Advanced MRI Reconstruction
- Application Case Study: Electrostatic Potential Calculation
- Computational Thinking For Parallel Programming
- Related Programming Models: MPI
- CUDA Python Using Numba
- Related Programming Models: OpenCL
- Related Programming Models: OpenACC
- Related Programming Models: OpenGL
- Dynamic Parallelism
- Multi-GPU
- Using CUDA Libraries
- Advanced Thrust
- Other GPU Development Platforms: QwickLABS



Book: *Programming Massively Parallel Processors: A Hands-on Approach* by David Kirk and Wen-Mei Hwu, Morgan Kaufmann; 3rd edition, 2016



# INTRO

---

Why we are here!?



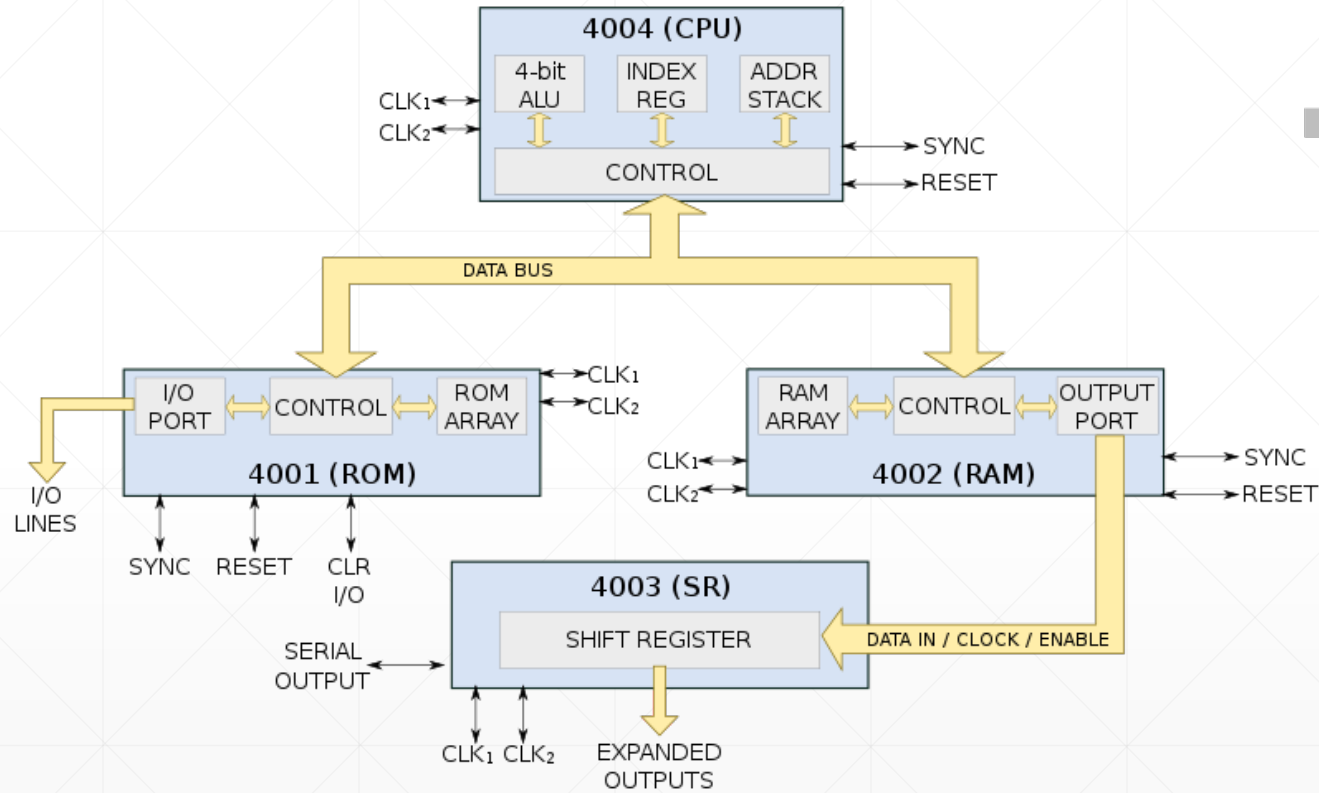
# 50th Anniversary of the first commercial CPU – Intel 4004

- In 1971, the Intel® 4004 processor held **2,300** transistors.
  - clocked at 740 kHz, capable of executing **92,000 instructions per second**.
  - the chip was capable of accessing **4KB** of program memory and **640 bytes of RAM**.
- The 4004 was a 4-bit CPU, designed for use in the Busicom 141-PF printing calculator.
- The 4004 CPU has been produced in **10 microns (10,000 nanometers)** lithography.
  - By comparison, a human hair diameter is ~100 microns



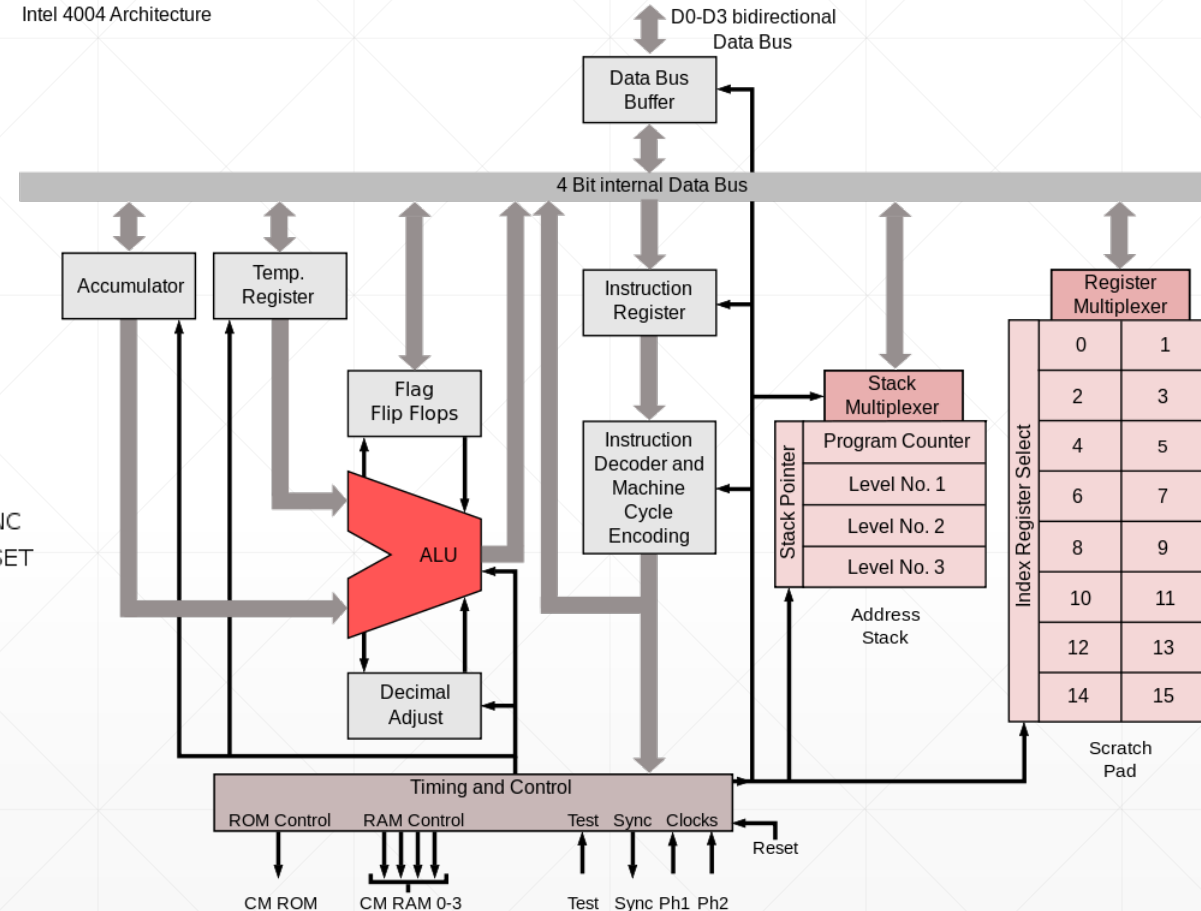
Source: <https://en.wikichip.org/wiki/intel/mcs-4/4004>

# Intel® MCS-4 (Micro-Computer Set-4) and 4004 CPU



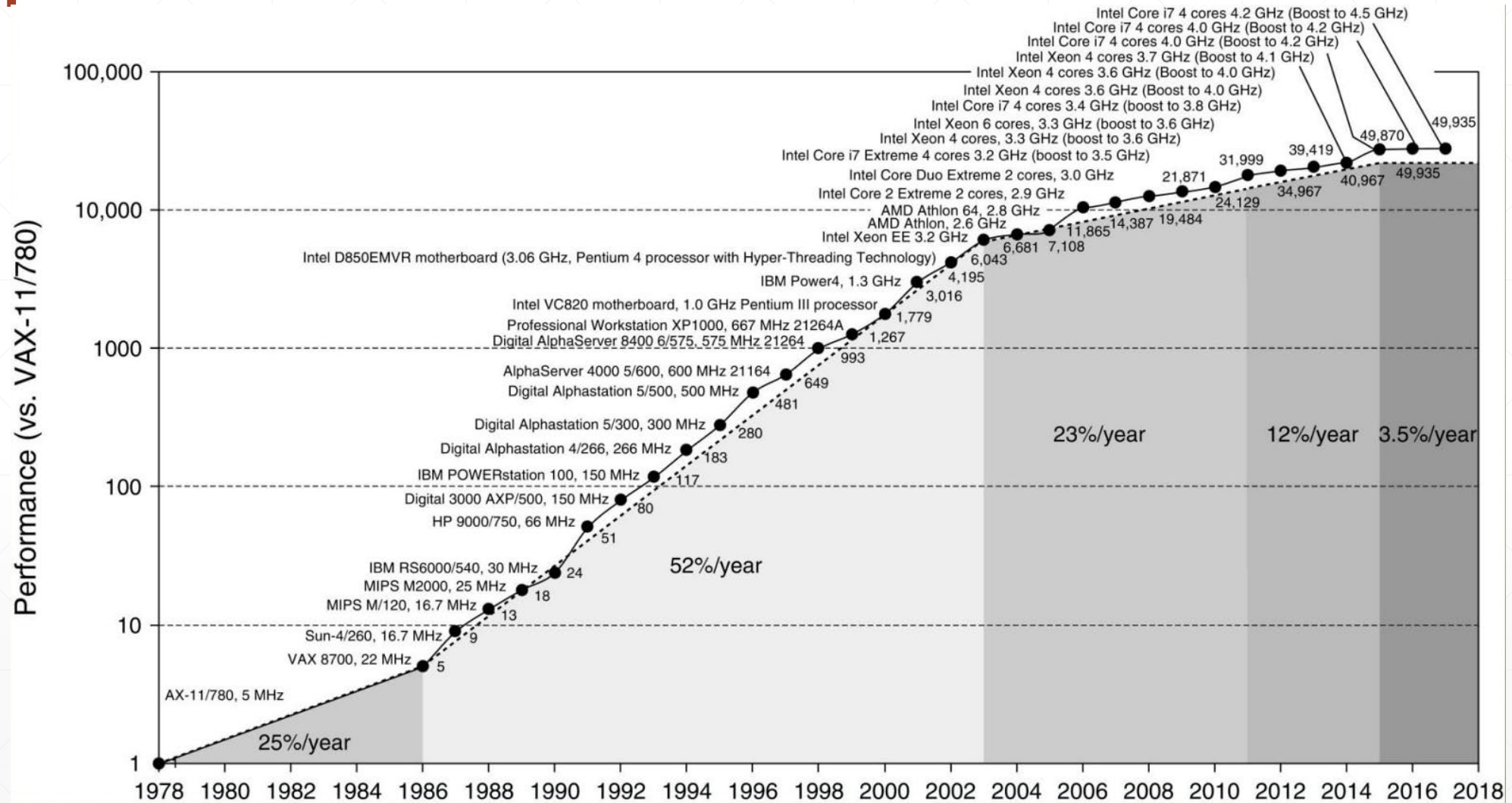
MCS-4 (Micro-Computer Set-4) was a family of 4-bit microprocessor chipsets developed by Intel

Intel 4004 Architecture



Sources: <https://en.wikichip.org/wiki/intel/mcs-4> | [https://pl.melayukini.net/wiki/Intel\\_4004](https://pl.melayukini.net/wiki/Intel_4004)

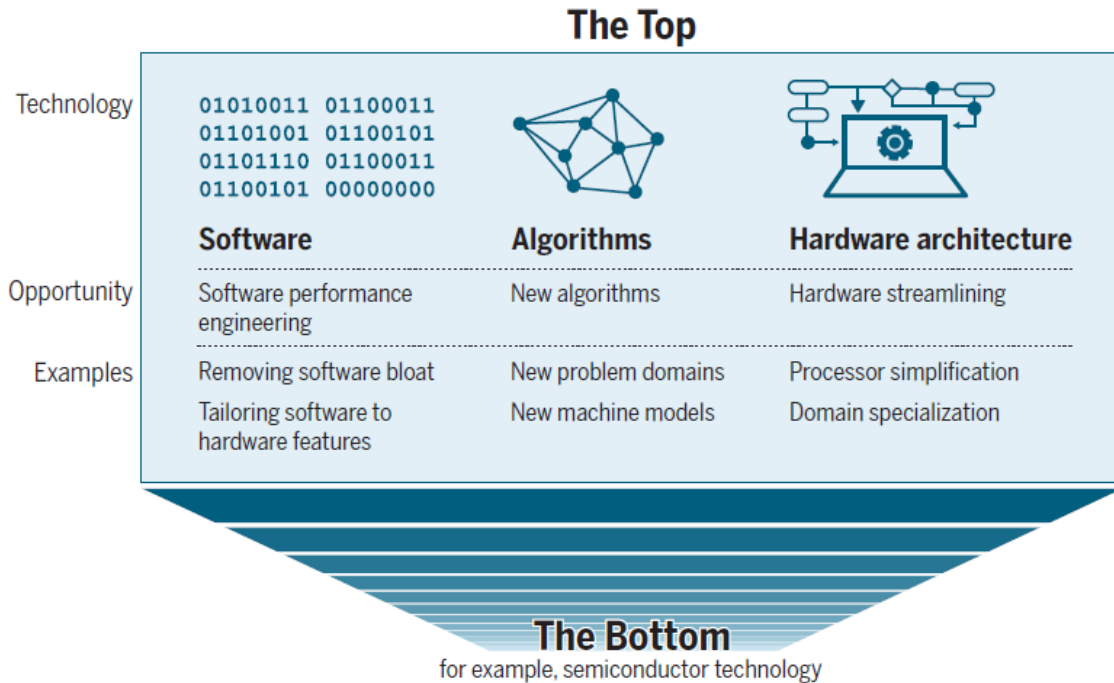
# Uniprocessor Performance



Source: <https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html>



# Where is the Room?



1959

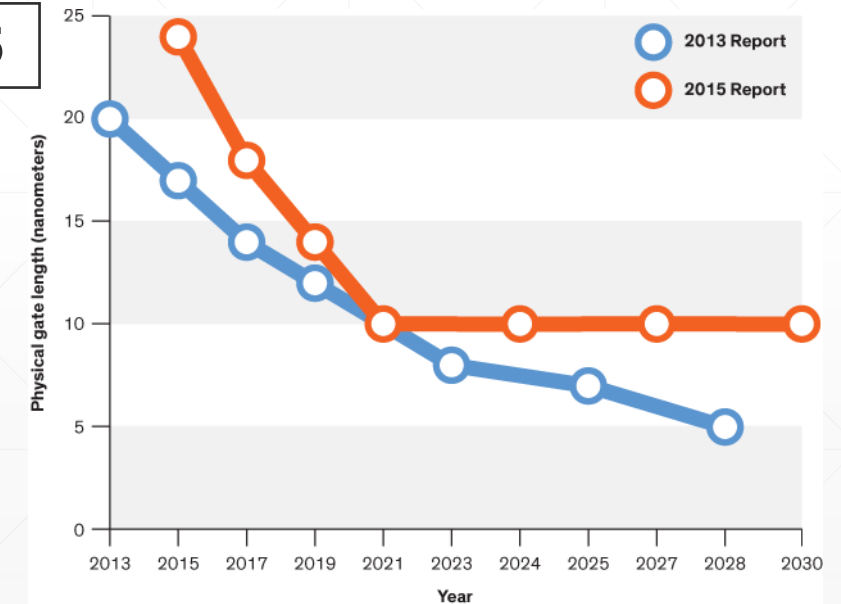
~~There's~~ <sup>was</sup> Plenty of Room  
at the Bottom

*An invitation to enter a new field of physics.*

*by Richard P. Feynman*

**End of The Road (?): International Technology Roadmap for Semiconductors** had previously predicted that the physical gate length of transistors would shrink until at least 2028 [see blue line]

2015

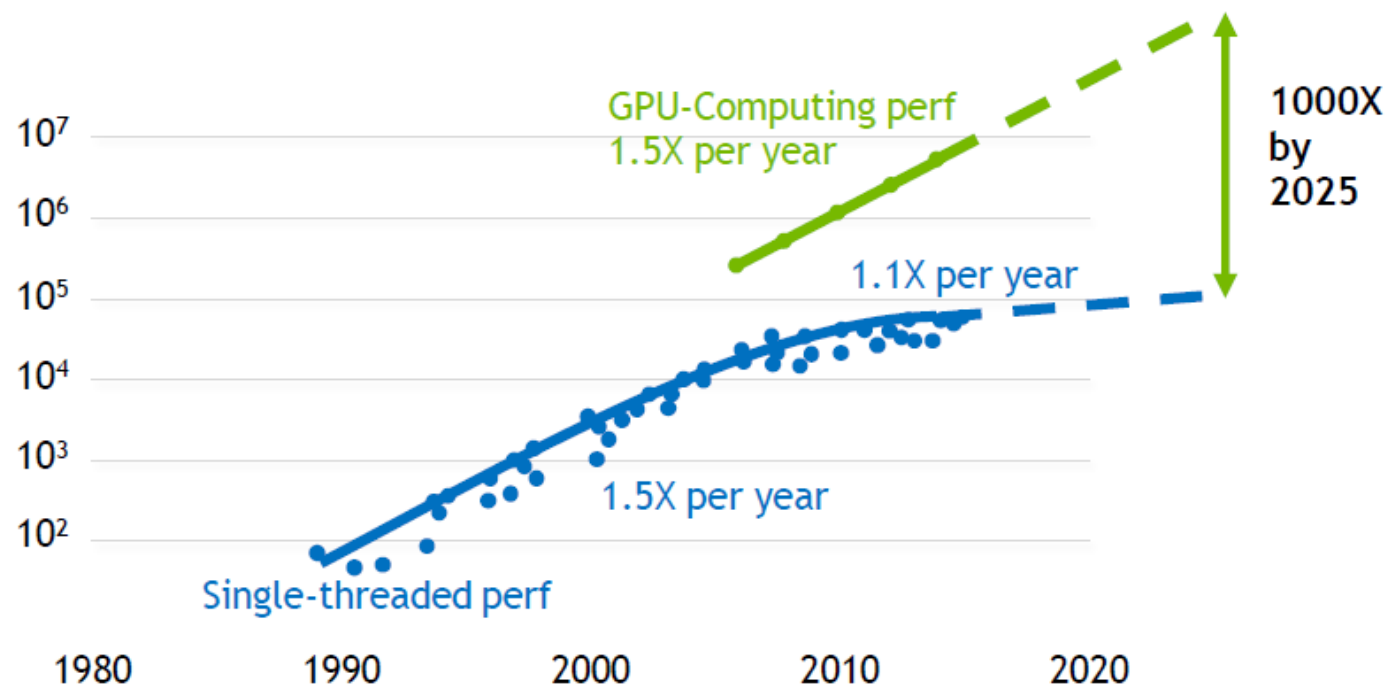
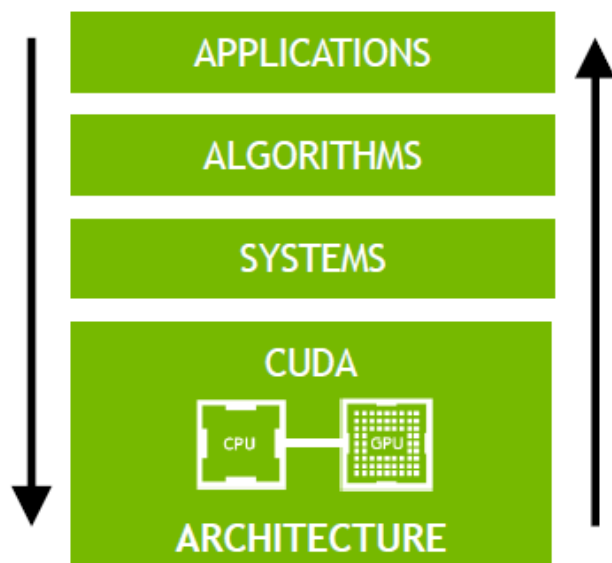


Sources: 1. A transcript of a talk given by Dr. Feynman on December 29, 1959, at the annual meeting of the American Physical Society at Caltech; <http://calteches.library.caltech.edu/47/2/1960Bottom.pdf>  
 2. Leiserson CE, Thompson NC, Emer JS, Kuszmaul BC, Lamson BW, Sanchez D, Schardl TB. There's plenty of room at the Top: What will drive computer performance after Moore's law? Science. 2020 Jun 5;368(6495):eaam9744. doi: 10.1126/science.aam9744. PMID: 32499413. 3. <https://spectrum.ieee.org/semiconductors/devices/transistors-could-stop-shrinking-in-2021>



# Why GPUs?

## RISE OF GPU COMPUTING

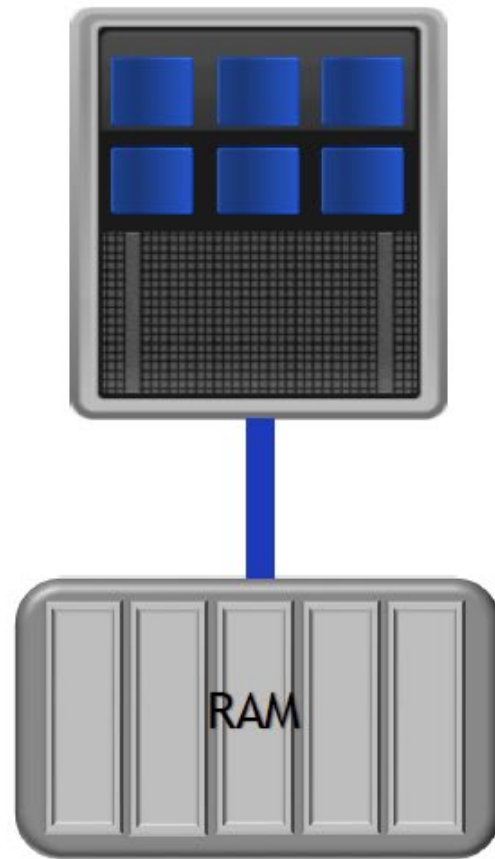


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

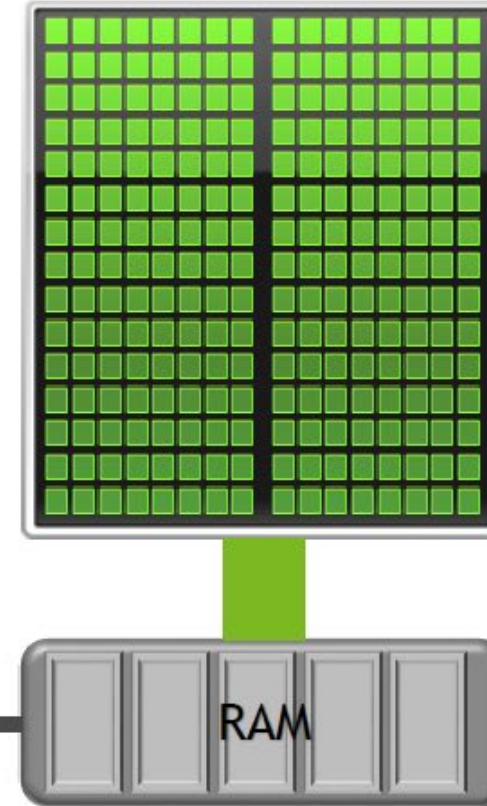


# CPU & GPU – heterogeneous computing

**CPU**  
Optimized for  
Serial Tasks



**GPU**  
Optimized for  
Parallel Tasks

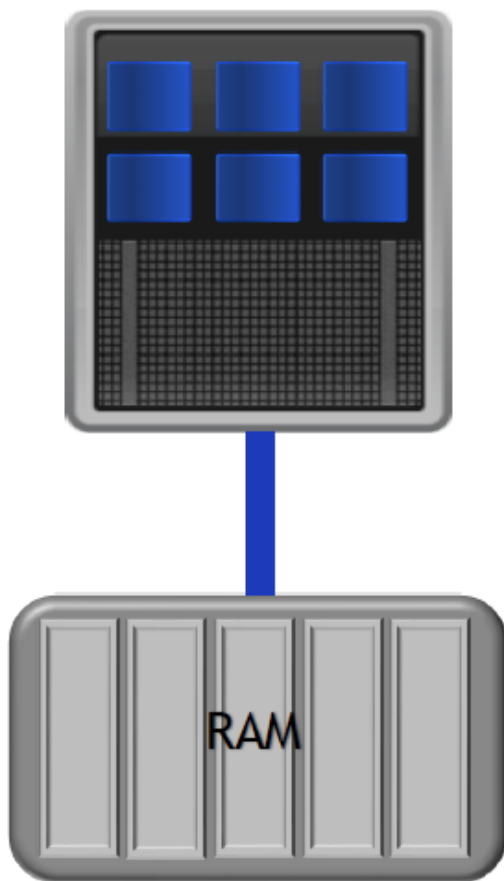


PCIe

Source: NVIDIA, Andreas Hehn, High Throughput with GPUs, 2018

# CPU is good for ...

**CPU**  
Optimized for  
Serial Tasks



## Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- few threads, can run very quickly

## Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt



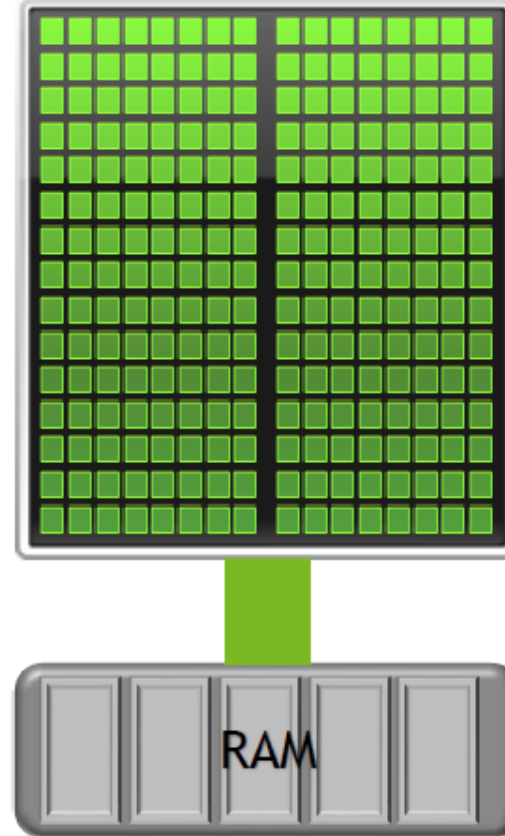
# GPU is good for ...

## Strengths

- High bandwidth main memory
- Latency tolerant via parallelism
- Significantly more compute resources
- High throughput
- High performance/watt

## Weaknesses

- Relatively low memory capacity
- Low per-thread performance



**GPU**  
Optimized for  
Parallel Tasks

# Software: Performance of matrix multiplication

A four-line kernel of the Python 2 code  
for matrix-multiplication

```
for i in xrange(4096):  
    for j in xrange(4096):  
        for k in xrange(4096):  
            C[i][j] += A[i][k] * B[k][j]
```

**Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices.** Each version represents a successive refinement of the original Python code. “Running time” is the running time of the version. “GFLOPS” is the billions of 64-bit floating-point operations per second that the version executes. “Absolute speedup” is time relative to Python, and “relative speedup,” which we show with an additional digit of precision, is time relative to the preceding line. “Fraction of peak” is GFLOPS relative to the computer’s peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

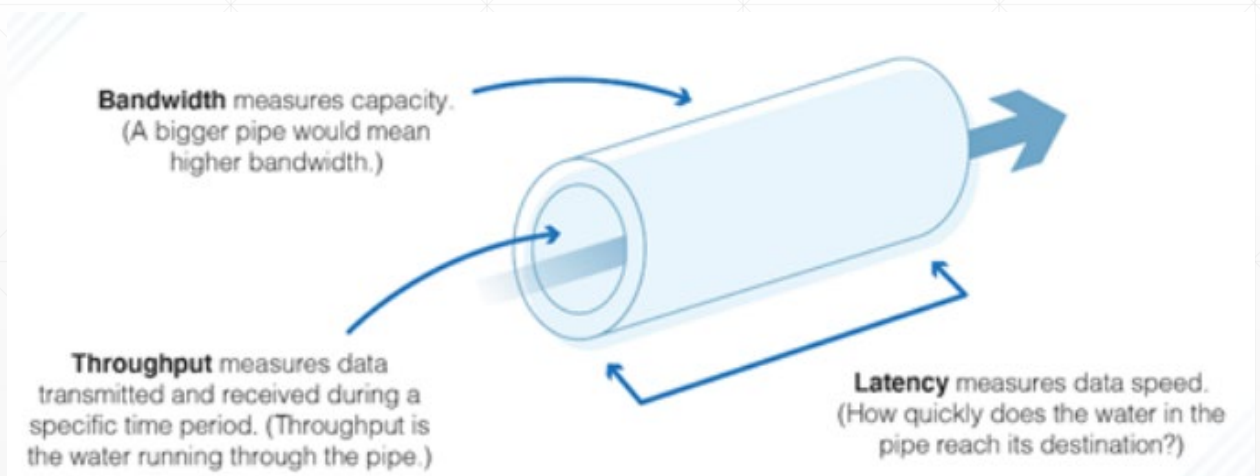
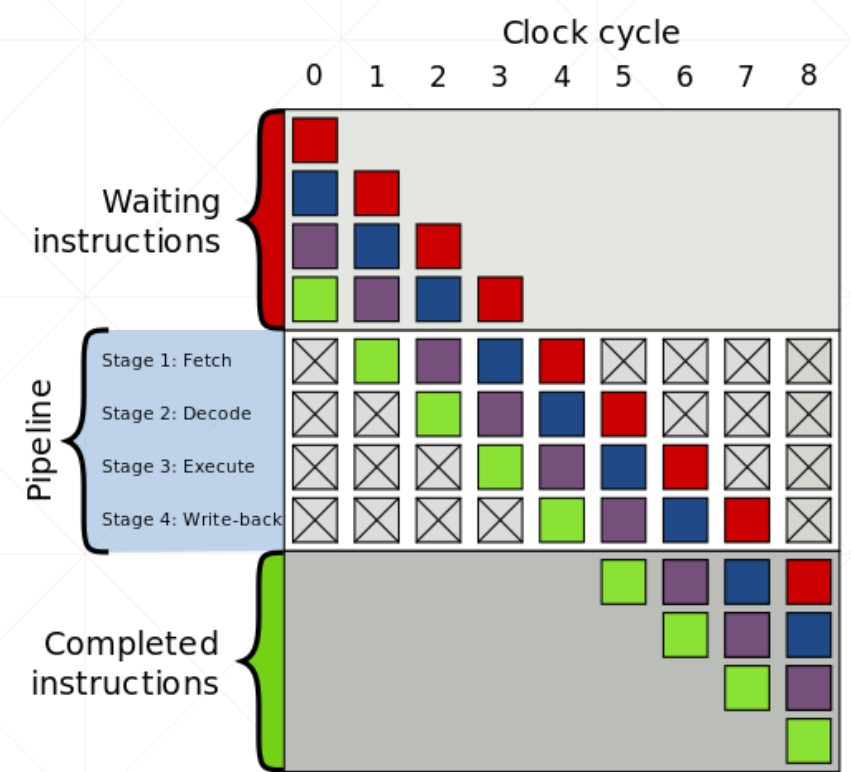
Source: Leiserson CE, Thompson NC, Emer JS, Kuszmaul BC, Lamson BW, Sanchez D, Schardl TB. There's plenty of room at the Top: What will drive computer performance after Moore's law? Science. 2020 Jun 5;368(6495):eaam9744. doi: 10.1126/science.aam9744. PMID: 32499413.





# Latency and Throughput

- **Latency** – time to get a solution
  - metric is time [sec]
    - minimize time, at the expense of power
- **Throughput** – number of operation (tasks) processed per unit of time
  - metric: ops/time [GFLOPS]
    - minimize energy per operation
- CPU: optimized for latency
- GPU: optimized for throughput



Source: <https://www.dnsstuff.com/latency-throughput-bandwidth> | [https://en.wikipedia.org/wiki/Instruction\\_pipelining](https://en.wikipedia.org/wiki/Instruction_pipelining)

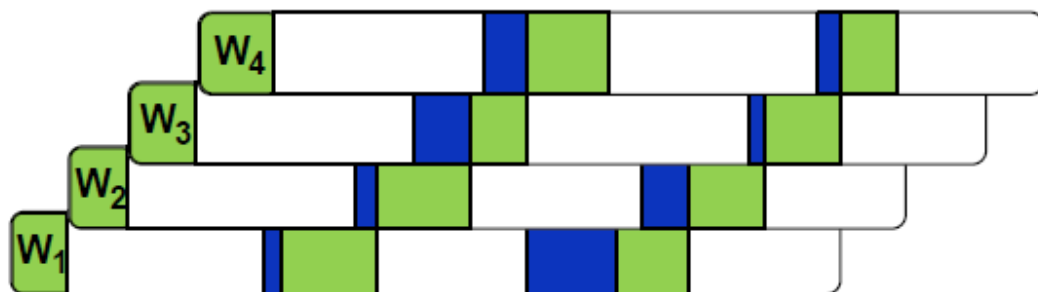
# LOW LATENCY OF HIGH THROUGHPUT?

CPU architecture must **minimize latency** within each thread



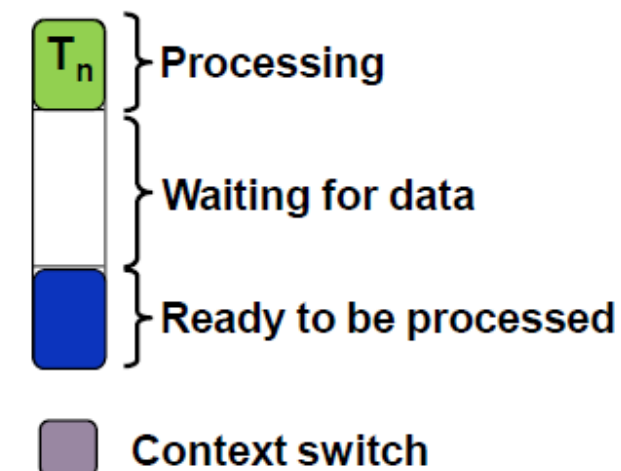
CPU core – Low Latency Processor

GPU architecture **hides latency** with computation from other threads (warps)



GPU Stream Multiprocessor – High Throughput Processor

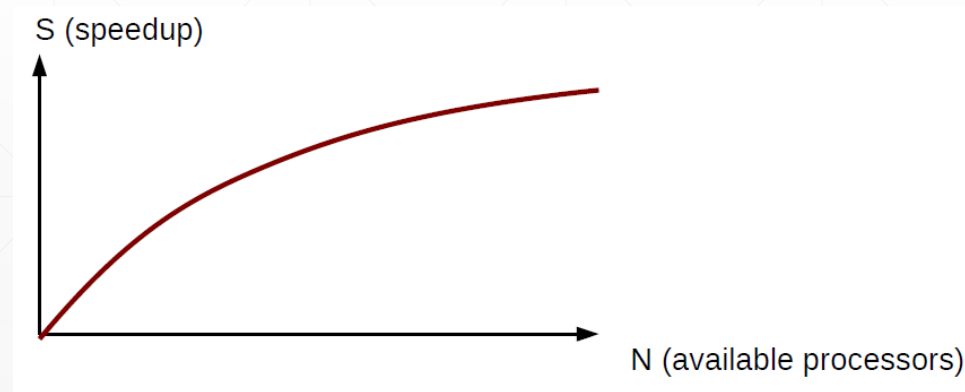
Computation Thread/Warp



# Amdahl's law

- Bounds of speed-up achievable by parallelization:
  - S – Speed-up
  - P – Ratio of parallel portions
  - N – Number of processors.

$$S = \frac{1}{\underbrace{(1-P)}_{\text{Time to run sequential portion}} + \underbrace{\frac{P}{N}}_{\text{Time to run parallel portion}}}$$



Source: [https://en.wikipedia.org/wiki/Amdahl%27s\\_law](https://en.wikipedia.org/wiki/Amdahl%27s_law)

# How much Parallelism?

- Parallelism: independent operations (e.g. memory accesses or computations) which execution can be overlapped
- How much parallelism do I need!?
- J.Little's law in queuing theory tells us:
  - Average customer arrival rate  $\lambda$  << Data Bandwidth [GB/s]
  - Average time spent  $W$  << Latency [s]
  - Average number of customers:  $L = \lambda \times W$  << Number of parallel processors we need

Source: John D. C. Little, 1961. "A Proof for the Queuing Formula:  $L = (\lambda) W$ ," Operations Research, INFORMS, vol. 9(3), pages 383-387, June; <https://doi.org/10.1287/opre.9.3.383>

# And NOW ...

- Make yourself another coffee and ...
- Please continue to the “CUDA Architecture” lecture

