SINTEF

FINITE VOLUMES AND GPU COMPUTING: PAST, PRESENT AND FUTURE

Finite Volume for Complex Applications 8

June 15, 2017, Lille, France

André R. Brodtkorb, Mathematics and Cybernetics, SINTEF Digital

SINTEF

- Established 1950 by the Norwegian Institute of Technology.
- The largest independent research organisation in Scandinavia.
- A non-profit organisation.
- Motto: “Technology for a better society”.
- Key Figures*
  - 2100 Employees from 70 different countries.
  - 73% of employees are researchers.
  - 3 billion NOK in turnover
    (about 360 million EUR / 490 million USD).
  - 9000 projects for 3000 customers.
  - Offices in Norway, USA, Brazil,
    Chile, and Denmark.

Talk outline

- Parallel computing on your desktop
  - Motivation for parallelism
  - Parallel algorithm design

- Lessons learned from working with multi- and many-core processors
  - Ca 2005: OpenGL
  - Ca 2007: CUDA
  - Ca 2014: Jupyter notebooks and pyopencl

- Summary

SINTEF

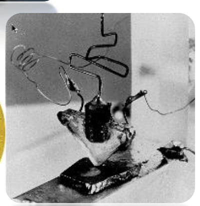
Motivation for going parallel

History lesson: development of the microprocessor 1/2

1942: Digital Electric Computer

(Atanasoff and Berry)

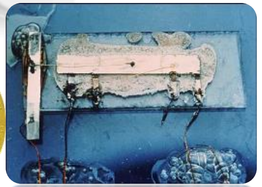
1956



1947: Transistor

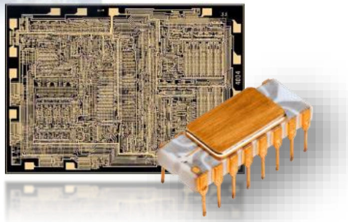
(Shockley, Bardeen, and Brattain)

2000



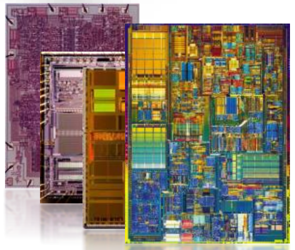
1958: Integrated Circuit

(Kilby)



1971: Microprocessor

(Hoff, Faggin, Mazor)



1971- Exponential growth

(Moore, 1965)

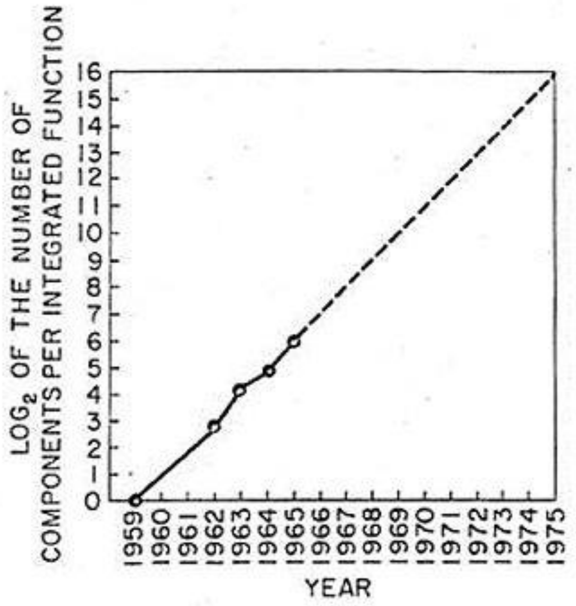
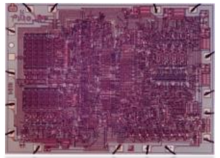


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

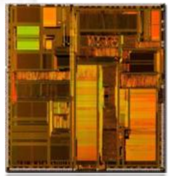
SINTEF

History lesson: development of the microprocessor 2/2

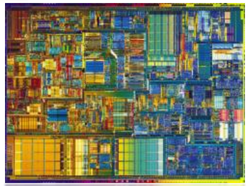


1971: 4004,
2300 trans, 740 KHz

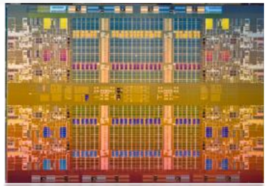
1982: 80286,
134 thousand trans, 8 MHz



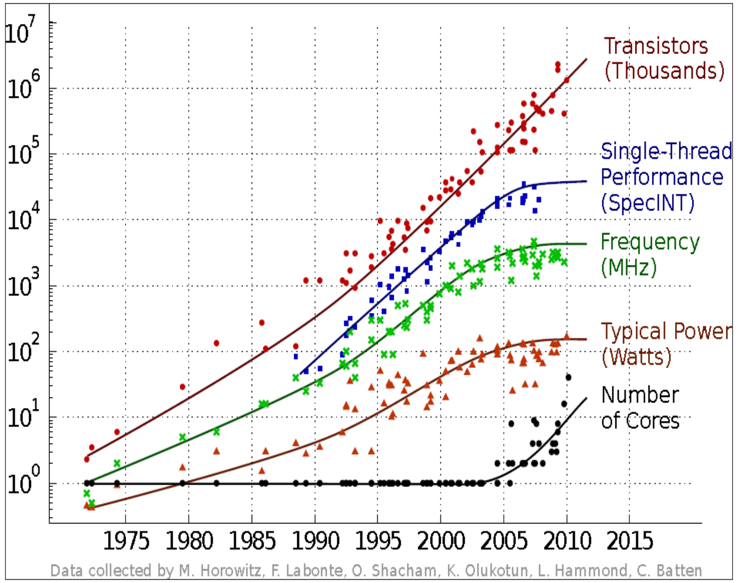
1993: Pentium P5,
1.18 mill. trans, 66 MHz



2000: Pentium 4,
42 mill. trans, 1.5 GHz

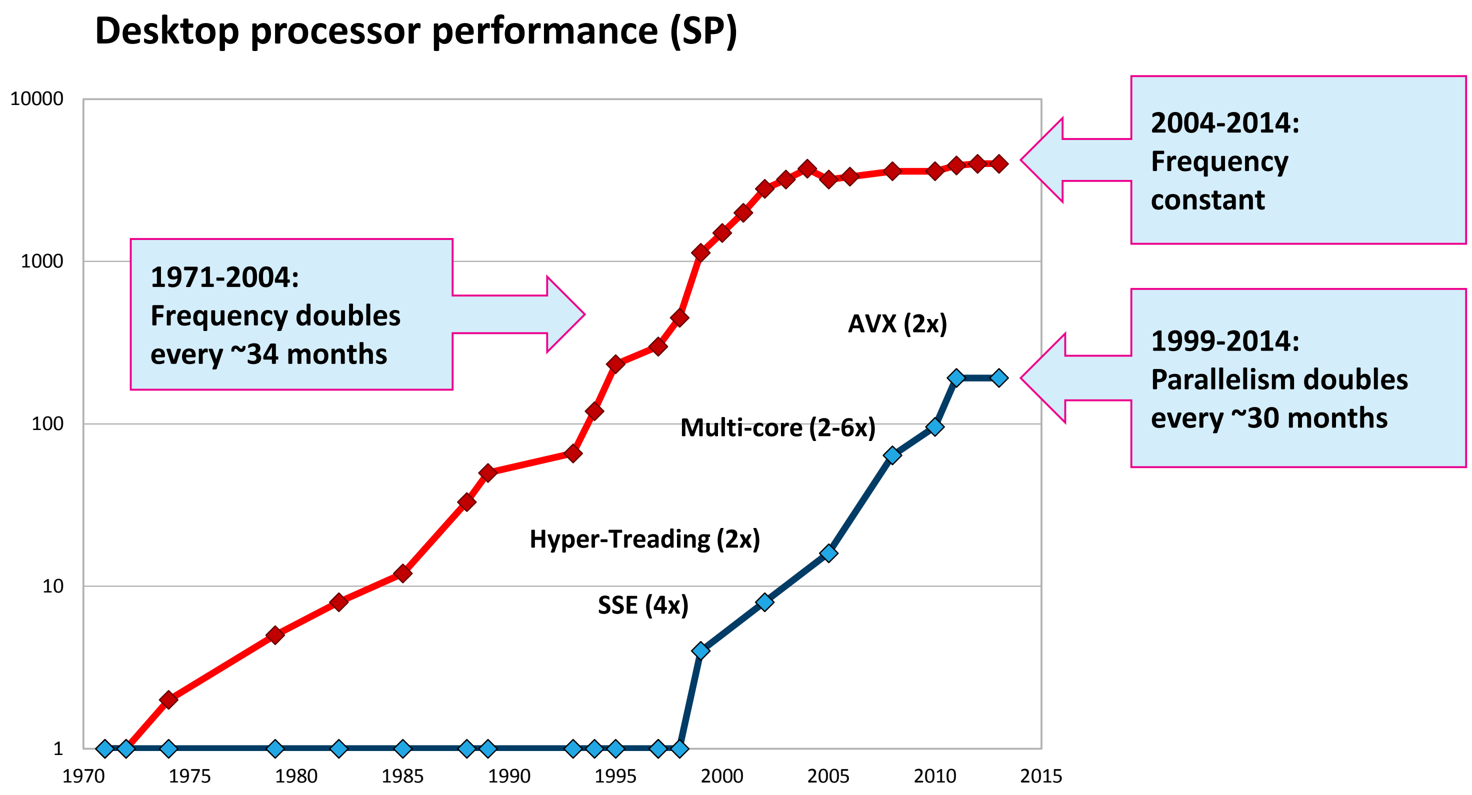


2010: Nehalem
2.3 bill. Trans, 8 cores, 2.66 GHz



SINTEF

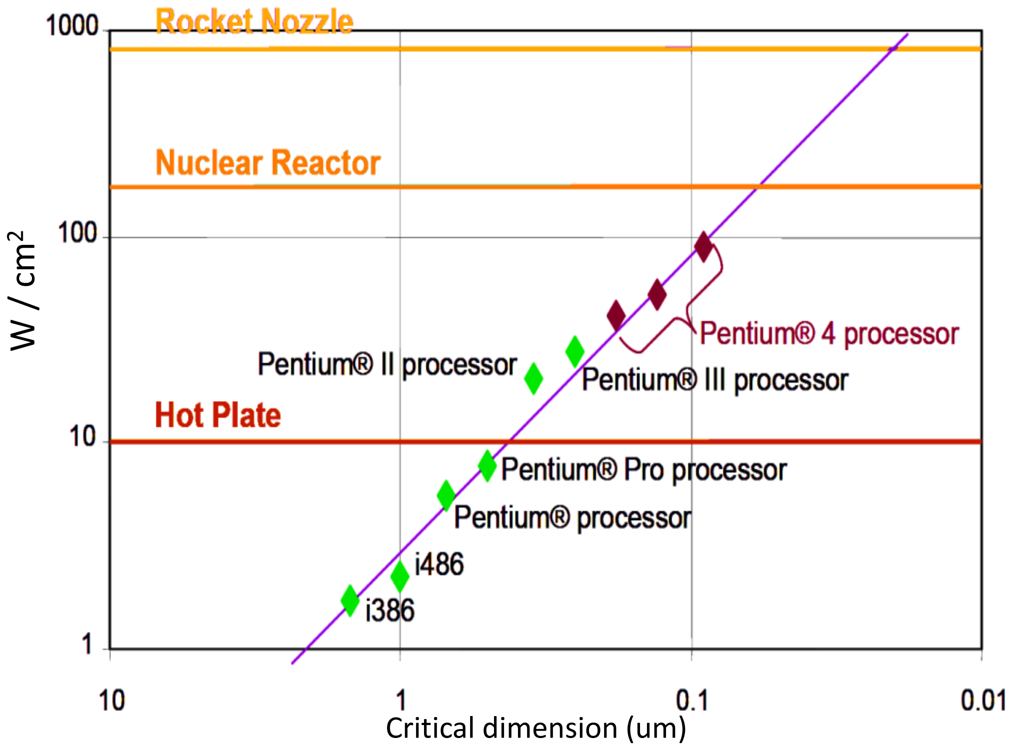
End of frequency scaling



- 1970-2004: Frequency doubles every 34 months (Moore's law for performance)
- 1999-2014: Parallelism doubles every 30 months

What happened in 2004?

- Heat density approaching that of nuclear reactor core: Power wall
  - Traditional cooling solutions (heat sink + fan) insufficient
- Industry solution: multi-core and parallelism!

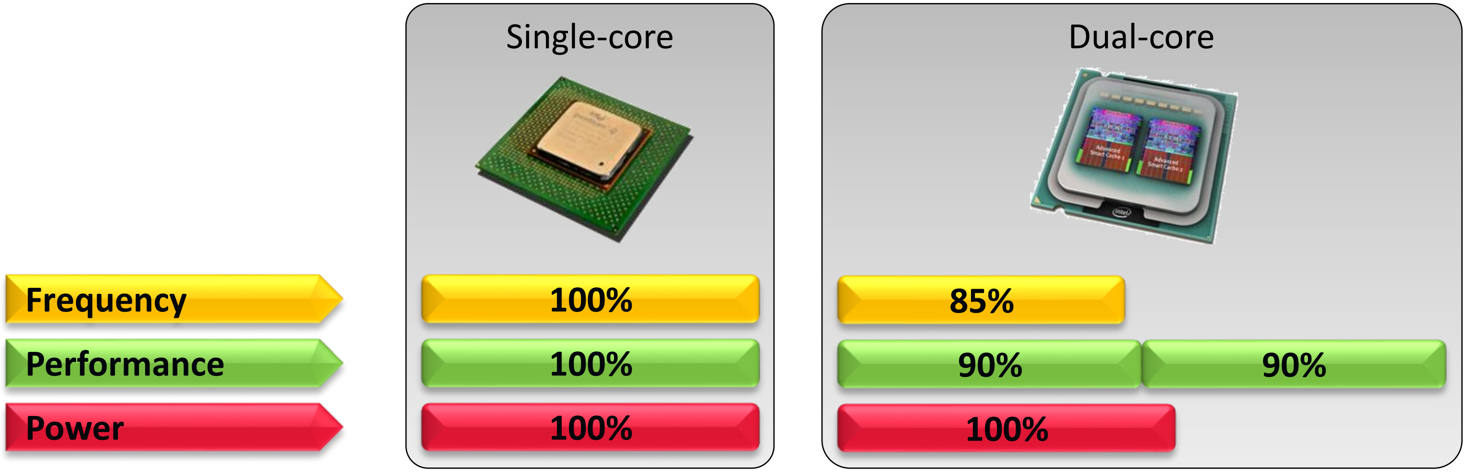


Original graph by G. Taylor, “Energy Efficient Circuit Design and the Future of Power Delivery” EPEPS’09

Why Parallelism?

The power density of microprocessors
is proportional to the clock frequency cubed:[1]

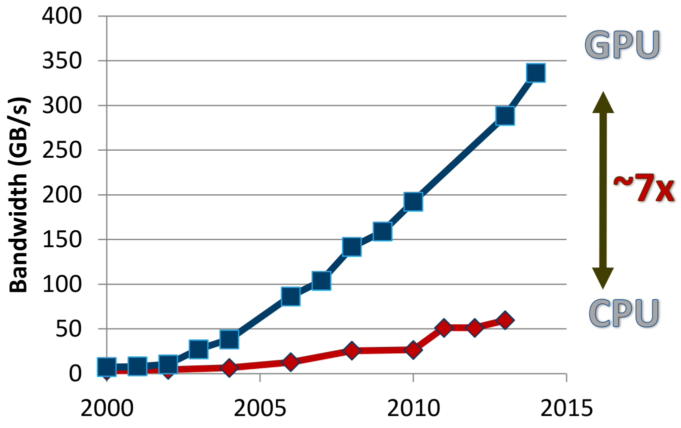
$$P_d \propto f^3$$

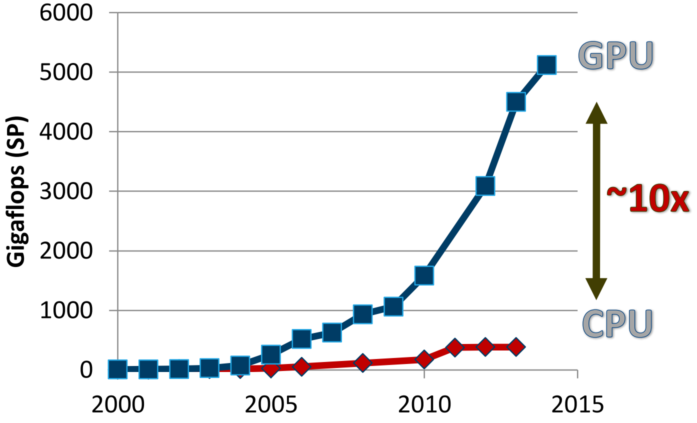


[1] Brodtkorb et al. State-of-the-art in heterogeneous computing, 2010

Massive Parallelism: The Graphics Processing Unit

- Thousands of floating point operations in parallel!
- 5-10 times as power efficient as CPUs!

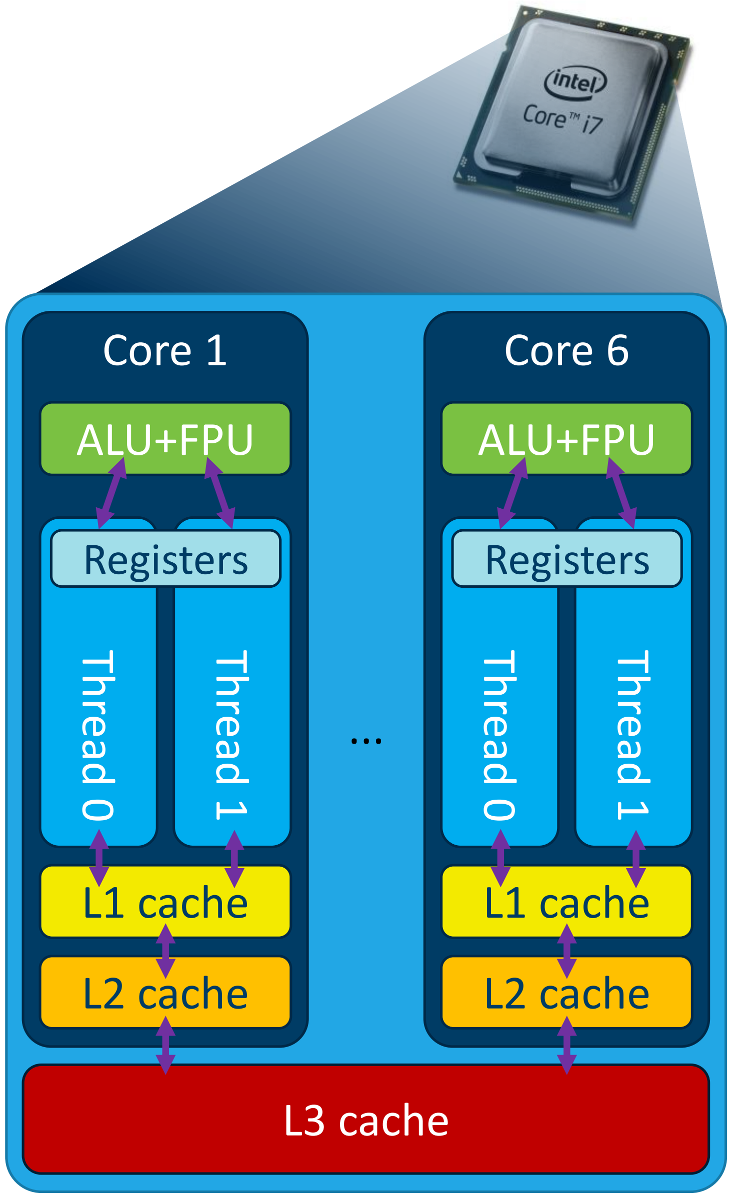




SINTEF

Multi-core CPU architecture

- A single core
  - L1 and L2 caches
  - 8-wide SIMD units (AVX, single precision)
  - 2-way Hyper-threading (hardware threads)
    When thread 0 is waiting for data,
    thread 1 is given access to SIMD units
  - Most transistors used for cache and logic
- Optimal number of FLOPS per clock cycle:
  - 8x: 8-way SIMD
  - 6x: 6 cores
  - 2x: Dual issue (fused mul-add / two ports)
  - Sum: 96!



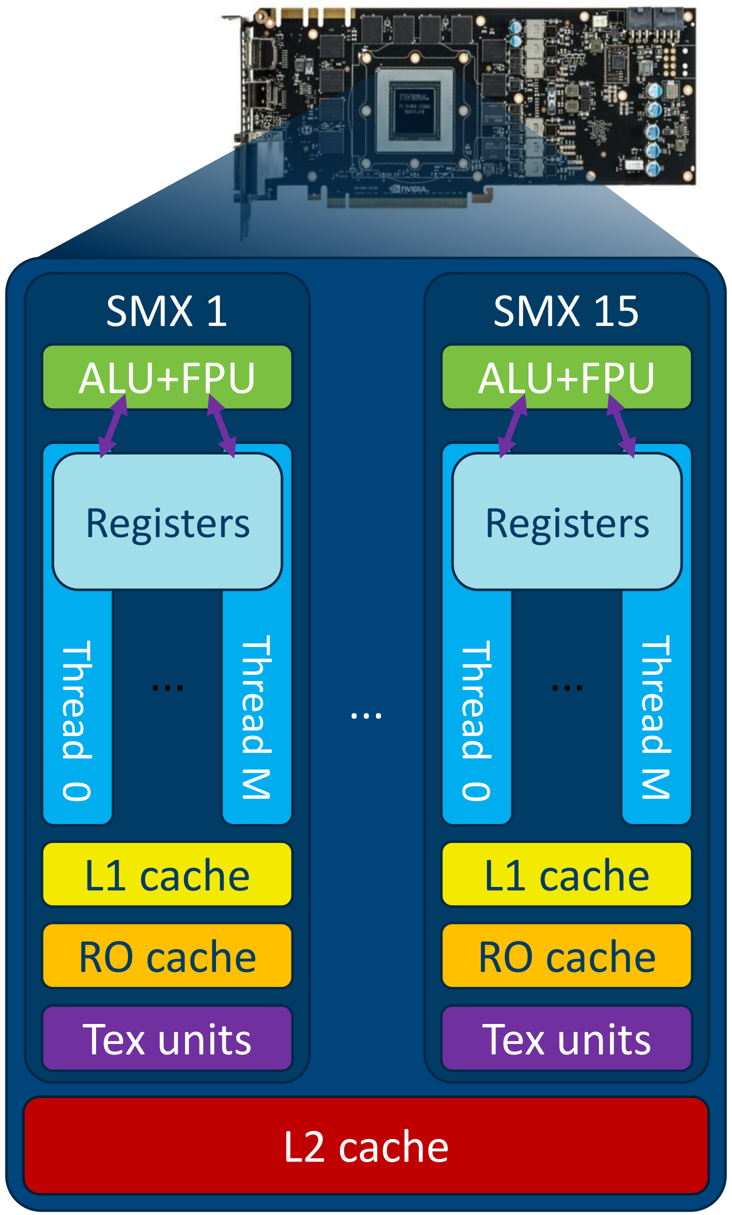
Simplified schematic of CPU design

SINTEF

Many-core GPU architecture

- A single core (Called streaming multiprocessor, SMX)
  - L1 cache, Read only cache, texture units
  - Six 32-wide SIMD units (192 total, single precision)
  - Up-to 64 warps simultaneously (hardware warps)
    Like hyper-threading, but a warp is 32-wide SIMD
  - Most transistors used for floating point operations

- Optimal number of FLOPS per clock cycle:
  - 32x: 32-way SIMD
  - 2x: Fused multiply add
  - 6x: Six SIMD units per core
  - 15x: 15 cores
  - Sum: 5760!

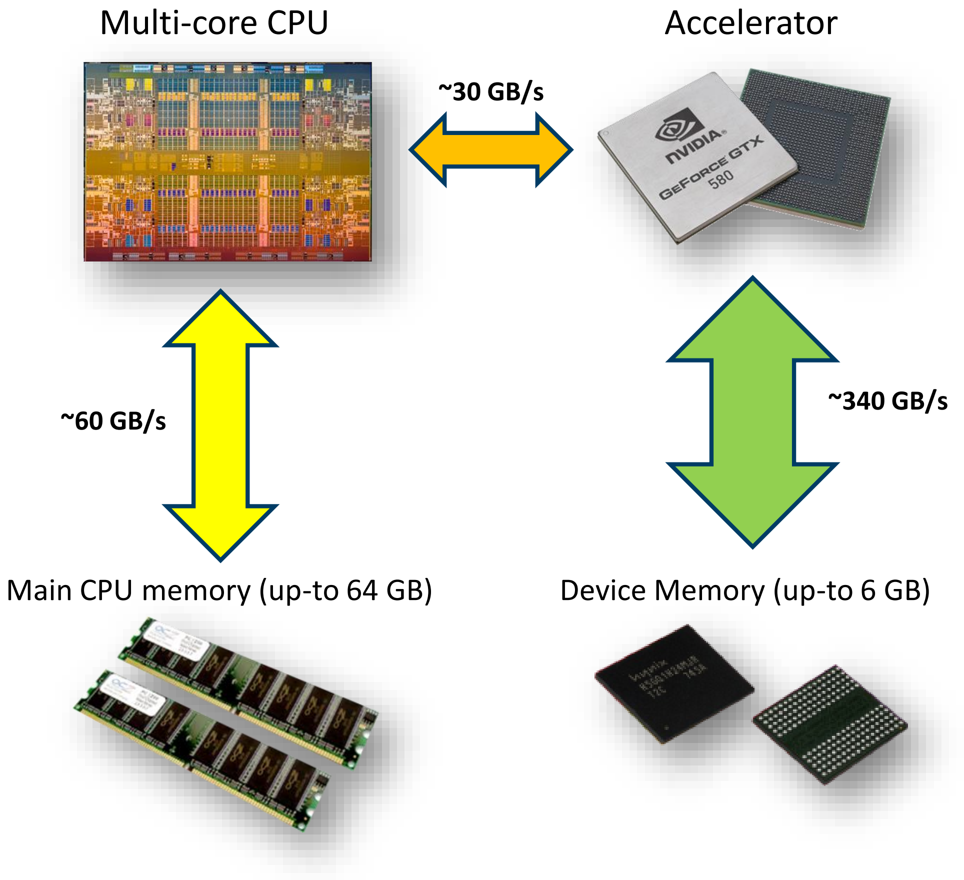


Simplified schematic of GPU design

SINTEF

Memory transfers

- Accelerators are connected to the CPU via the PCI-express bus
  - Slow: 15.75 GB/s each direction
- Accelerator memory is limited but fast
  - Typically on the order of 10 GB
  - Up-to 340 GB/s!
  - Fixed size, and cannot be expanded with new dimm’s (like CPUs)

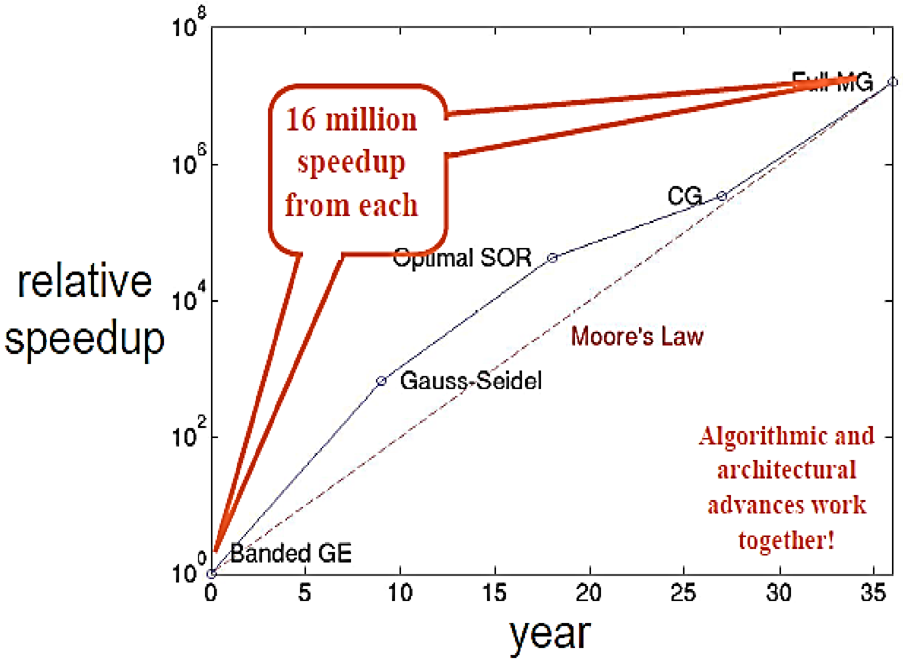


Parallel algorithm design

Why care about computer hardware?

- The key to performance, is to consider the full algorithm and architecture interaction.

- A good knowledge of <u>both</u> the algorithm <u>and</u> the computer architecture is required.



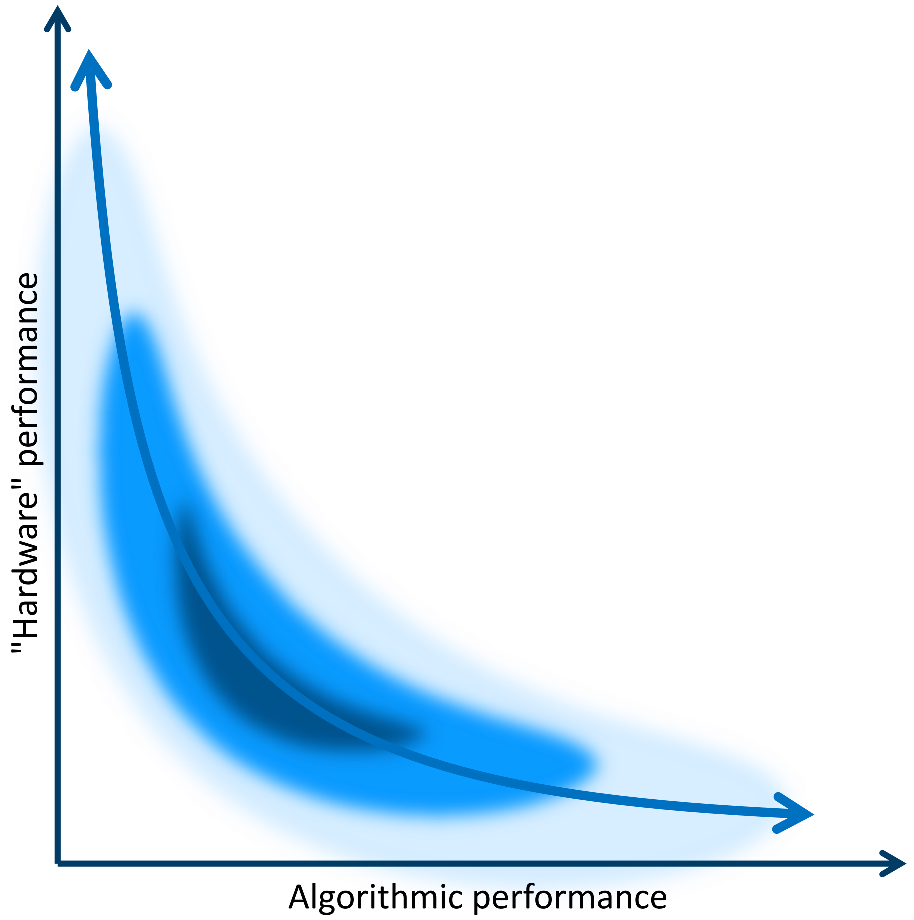
Graph from David Keyes, Scientific Discovery through Advanced Computing, Geilo Winter School, 2008

Algorithmic and numerical performance

- Total performance is the product of algorithmic **and** "hardware" performance
  - Your mileage may vary: algorithmic performance is highly problem dependent

- Many algorithms have low "hardware" performance
  - Only able to utilize a fraction of the capabilities of processors, and often **worse in parallel**

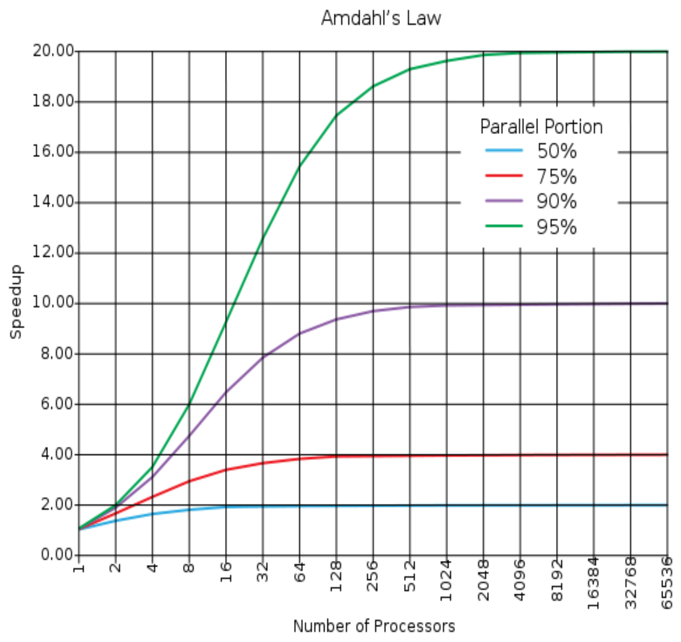
- Need to consider both the algorithm and the architecture for maximum performance



Parallel considerations 1/4

- Most algorithms contains a mixture of work-loads:
  - Some serial parts
  - Some task and / or data parallel parts

- <u>Amdahl's law:</u>
  - There is a limit to speedup offered by parallelism
  - Serial parts become the bottleneck for a massively parallel architecture!
  - Example: 5% of code is serial: maximum speedup is 20 times!

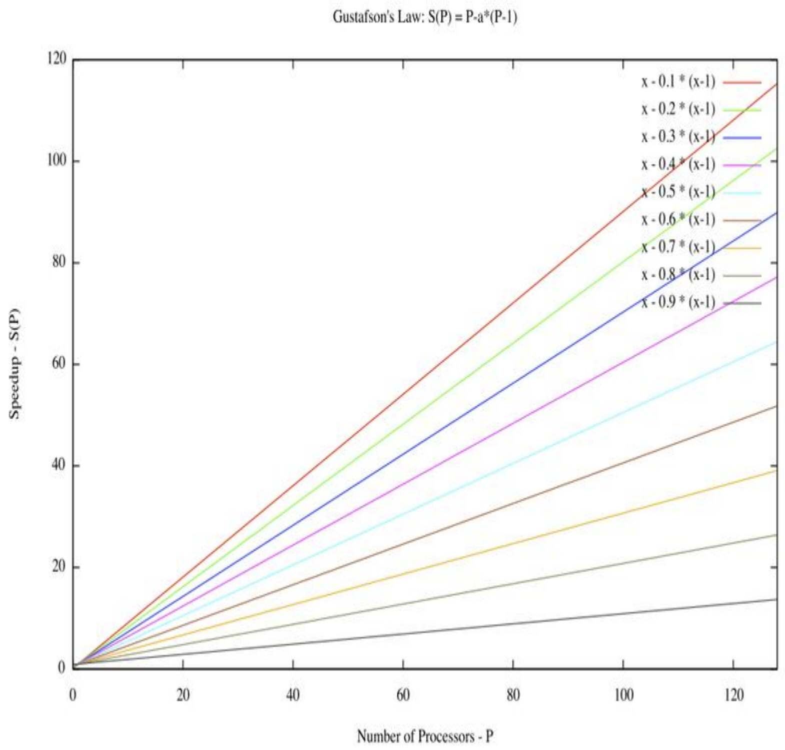


$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

S: Speedup
P: Parallel portion of code
N: Number of processors

Parallel considerations 2/4

- <u>Gustafson's law:</u>
  - If you cannot reduce serial parts of algorithm, make the parallel portion dominate the execution time
  - Essentially: solve a bigger problem to get a bigger speedup!



$$S(P) = P - \alpha \cdot (P - 1).$$

S: Speedup
P: Number of processors
α: Serial portion of code

Graph from Wikipedia, user Peahihawaii, CC-BY-SA 3.0

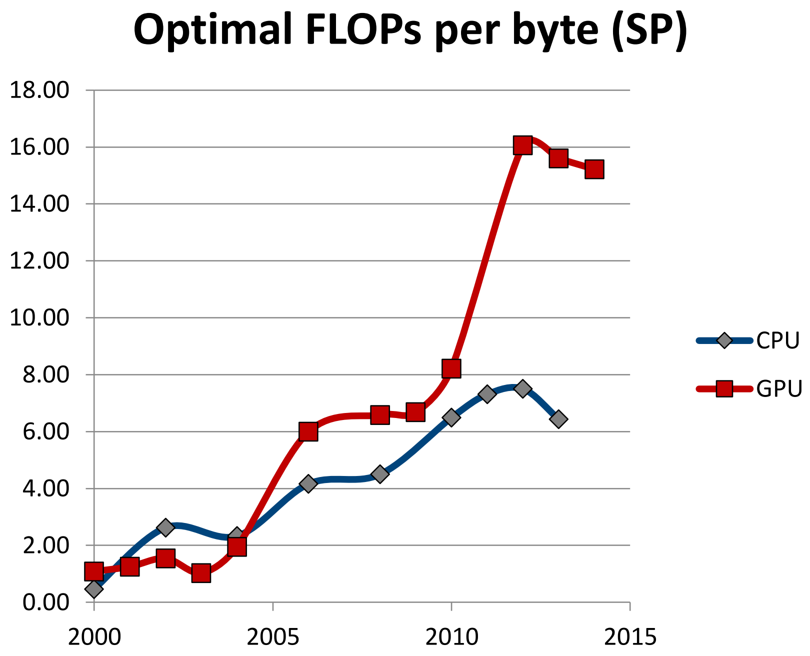
SINTEF

Parallel considerations 3/4

- A single precision number is four bytes
  - You must perform over 60 operations for each float read on a GPU!
  - Over 25 operations on a CPU!

- This groups algorithms into two classes:
  - Memory bound
    Example: Low order finite volume
  - Compute bound
    Example: High order finite volume

- The third limiting factor is latencies
  - Waiting for data
  - Waiting for floating point units
  - Waiting for ...



SINTEF

Parallel considerations 4/4

- Moving data has become the major bottleneck in computing.

- Downloading 1GB from Japan to Switzerland consumes roughly the energy of 1 charcoal briquette[1].
- A FLOP costs less than moving one byte[2].
- Key insight: flops are free, moving data is expensive

[1] Energy content charcoal: 10 MJ / kg, kWh per GB: 0.2 (Coroama et al., 2013), Weight charcoal briquette: ~25 grams

[2] Simon Horst, Why we need Exascale, and why we won't get there by 2020, 2014

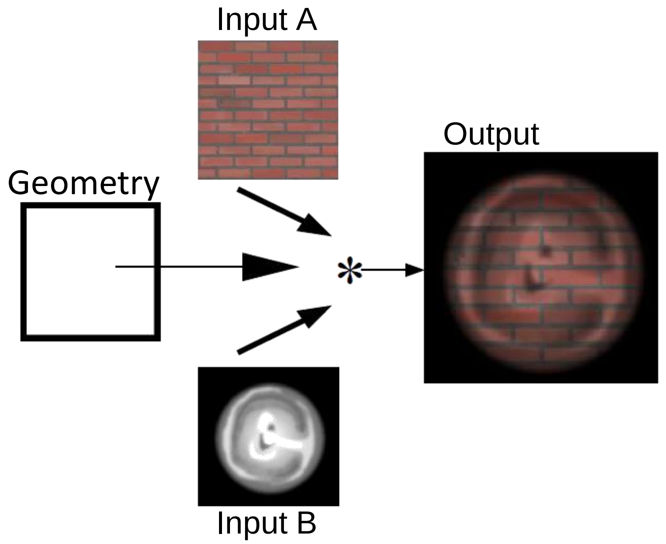
Lessons learned from working with multi- and many-core processors

Ca 2005: GPUs with OpenGL: GPGPU

Early Programming of GPUs

- GPUs were first programmed using OpenGL and other graphics languages
- Mathematics were written as operations on graphical primitives
- Extremely cumbersome and error prone

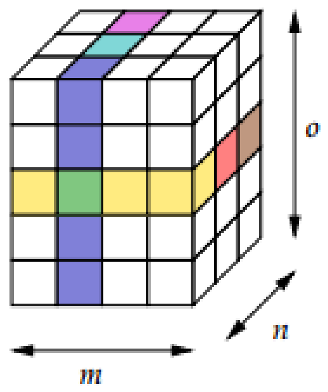
Element-wise matrix multiplication

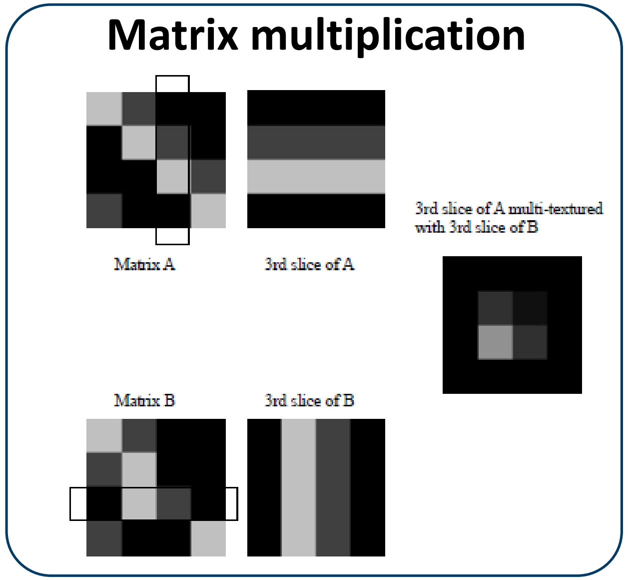


Matrix-matrix multiplication with OpenGL

- Larsen & McAllister demonstrated that using GPUs through non-programmable OpenGL could be faster than using ATLAS [1]

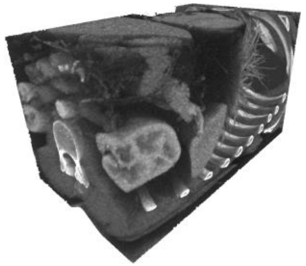
- Their algorithm was "simple"
  - Matrix-matrix multiplication dots
    row i of matrix A with
    column j of matrix B to produce element (i, j)
  - We can formulate this product for a virtual cube of processors
    - Processor (m, n, 0) computes the product A[m, n]*B[n, o]
    - By summing along the o dimension, the matrix product is complete.
  - L&M used textures and blending to implement the virtual cube of processors algorithm



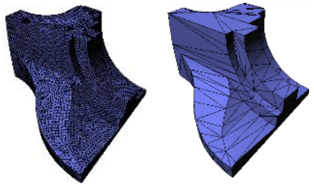


[1] Fast matrix multiplies using graphics hardware, Larsen and McAllister, 2001

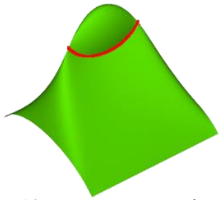
Examples of Early GPU Research at SINTEF



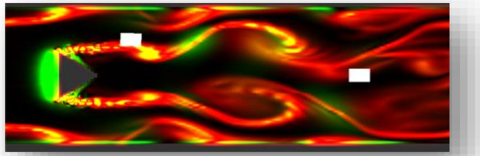
Registration of medical data (~20x)



Preparation for FEM (~5x)

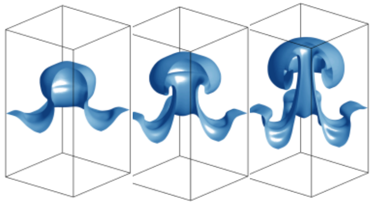


Self-intersection (~10x)

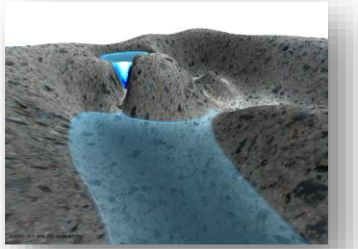


Fluid dynamics and FSI (Navier-Stokes)

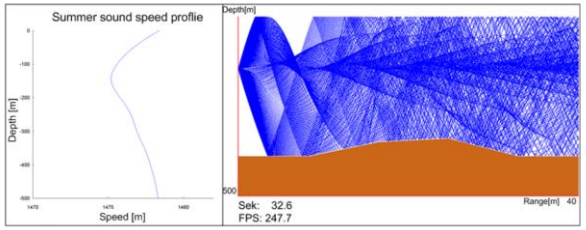
Inpainting (~400x matlab code)



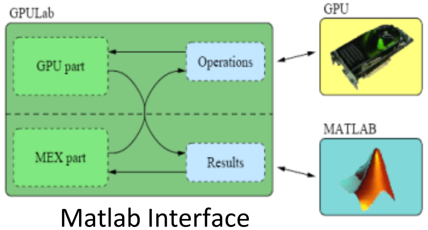
Euler Equations (~25x)



SW Equations (~25x)



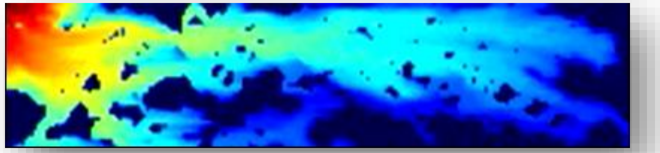
Marine aqoustics (~20x)



Matlab Interface

$$\begin{bmatrix} b_1 - a_{-\frac{1}{2}} & -a_{\frac{1}{2}} & 0 & 0 & 0 & \cdots & 0 \\ -a_{\frac{1}{2}} & b_2 & -a_{\frac{3}{2}} & 0 & 0 & \cdots & 0 \\ 0 & -a_{\frac{3}{2}} & b_3 & -a_{\frac{5}{2}} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -a_{n-\frac{5}{2}} & b_{n-2} & -a_{n-\frac{3}{2}} & 0 \\ 0 & \cdots & 0 & 0 & -a_{n-\frac{3}{2}} & b_{n-1} & -a_{n-\frac{1}{2}} \\ 0 & \cdots & 0 & 0 & 0 & a_{n-\frac{1}{2}} & b_n - a_{n+\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{bmatrix}$$

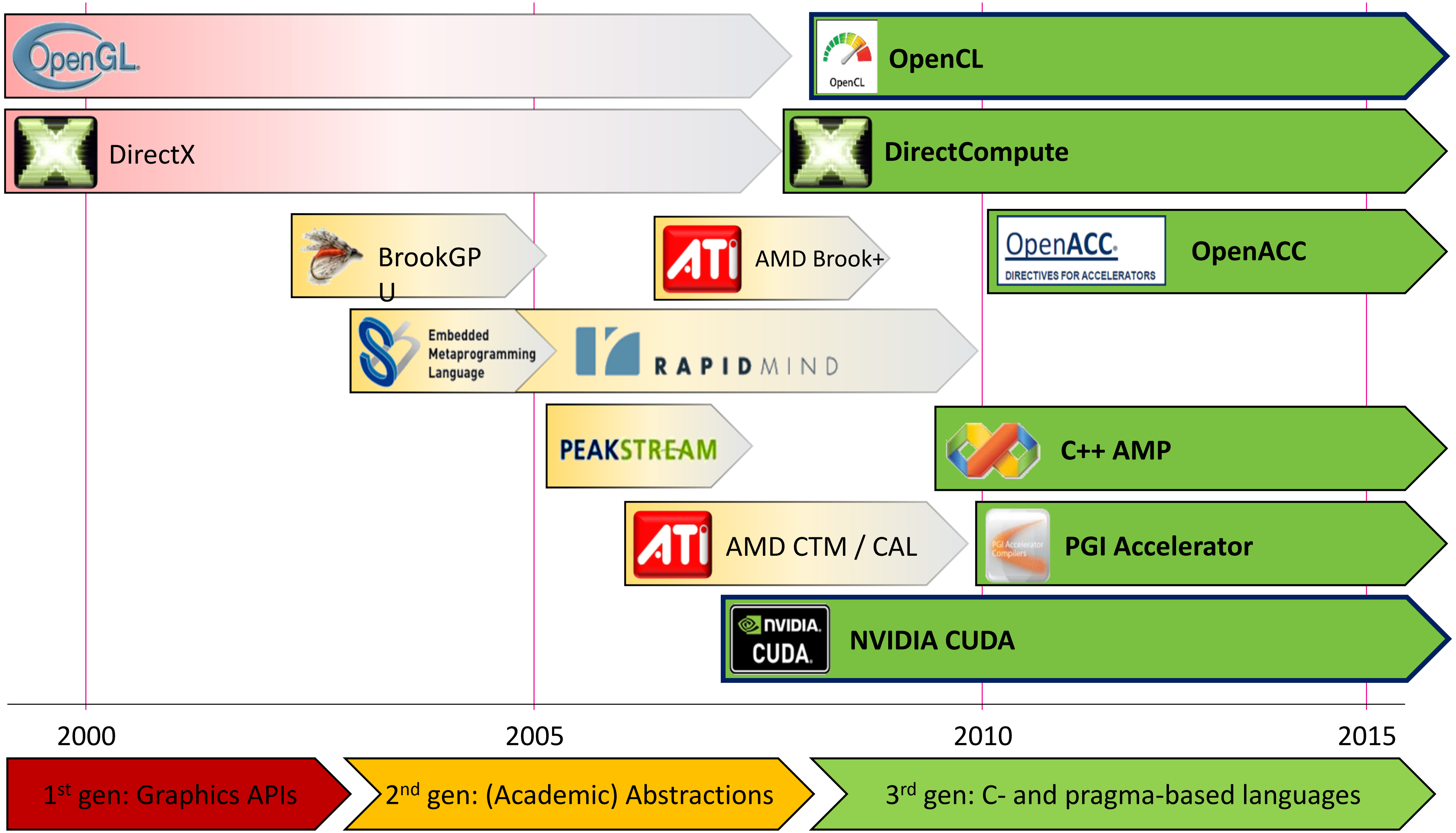
Linear algebra



Water injection in a fluvial reservoir (20x)

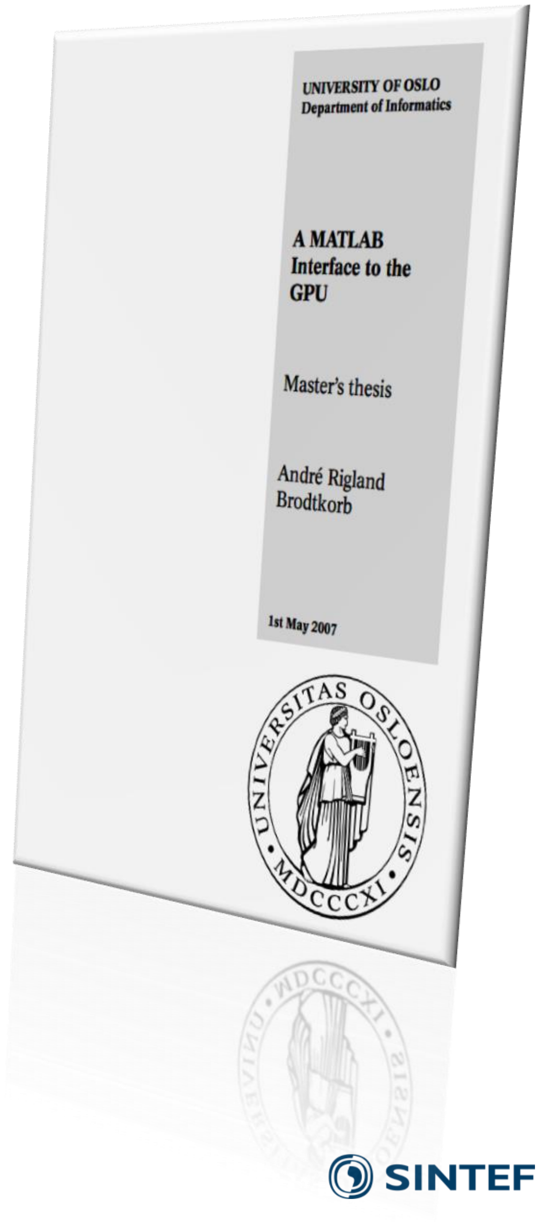
Ca 2007: CUDA and mature programming languages

GPU Programming Languages



My first encounter with CUDA

- May 1st 2007: I handed in my masters thesis.
- June 15th 2007: I hold the oral presentation and receive my grade.
- June 23rd 2007: CUDA was released officially.
  Most of my thesis was officially obsolete.

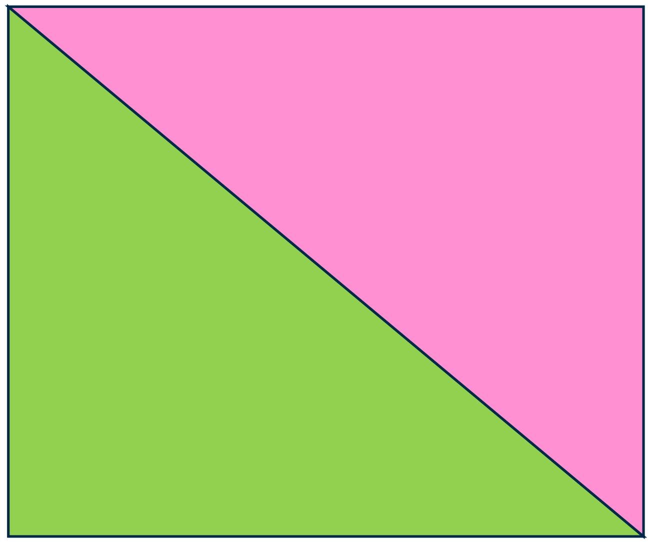


NVIDIA CUDA

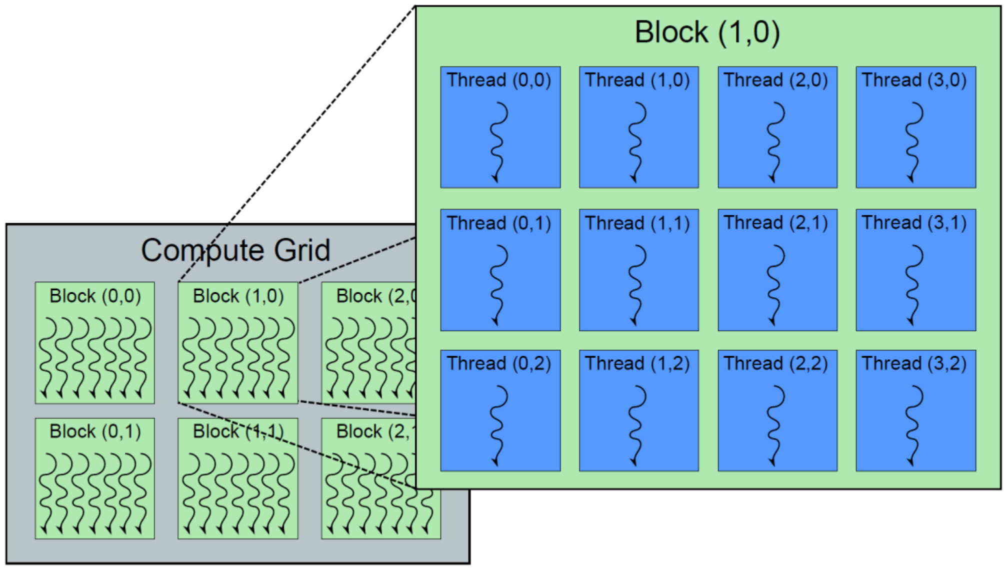
- CUDA solved the major problems with OpenGL
  - Unstable driver and different results on different hardware (it will only run on NVIDIA)
  - Uncomfortable implementation regime
- We could now program in a C-like language
  - Rapid development of a whole range of new applications
  - A huge interest for GPUs emerged
- The first versions, however, had the same amount of compiler bugs as OpenGL

The benefit of CUDA

OpenGL "Kernel launch"



CUDA kernel launch

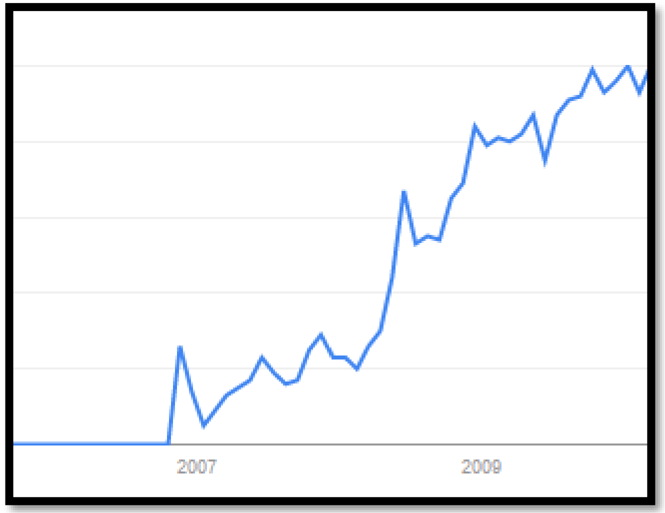


- Render primitives that cover part of the screen that represents your computational domain
- The shader which colours the pixel performs the wanted calculation

- "Pixels" are still the underlying primitive, but now we execute a grid of blocks.
- Each block runs independently, but threads within a block can collaborate

CUDA fever

Google search trends



- CUDA became a superstar in the academic camp "over night".
- Within 2010, there were over 1000 demos, papers, and commercial applications using CUDA on the CUDA Showcase.
- AMD tried countering with Close-to-the-metal (assemly for AMD GPUs) and Brook+, but none recieved any noticeable attention.

Exploring CUDA

- CUDA sparked a whole new range of research articles on GPUs

- Hardware exploration
  - How was texture memory laid out?
  - How large were the different caches?
  - Would it be better to use more registers and less shared memory or not?
  - Is more threads always better?

- Massive focus on memory movement
  - Coalesced reads and writes
  - Cached versus non-cached reads
  - Texture reads versus cached reads

- Low-hanging fruit was rapidly picked
  - More and more articles solved real-world problems
  - Proof-of-concept slowly became less interesting

Ca 2014: High-level GPU with Jupyter and pyopencl

OpenCL

- OpenCL is much like CUDA, but slightly more cumbersome to work with.
  - The benefit is that the same code can run on Intel CPUs, the Xeon Phi, NVIDIA GPUs, AMD GPUs, etc.

- The amount of code needed to do the exact same thing is larger in OpenCL
  - OpenCL is a C API
  - CUDA has C++ bindings, and supports templates

OpenCL

Jupyter and Pyopencl

- OpenCL is a C API, which requires working in C, and possibly long compilation times
- Even the simplest OpenCL example requires a lot of boilerplate code
- Pyopencl solves this, by enabling access to OpenCL through Python
- Jupyter (previously ipython) notebook gives us an interactive shell to try out OpenCL and prototype!

Demo of iPython and Pyopencl

- If time permits

Summary

Summary

- We need to consider parallelism when designing and implementing algorithms
  - We cannot afford to waste most of the true potential
- GPU computing has never been easier
  - Getting good performance still requires knowledge of the architecture
- GPUs have been a success story in many fields
  - Its widespread availability, low cost, and "easy" programming model has made it a success, where other parallel architectures have failed
  - A 10 times performance improvement possible

Thank you for your attention!

André R. Brodtkorb
Email: Andre.Brodtkorb@sintef.no
Homepage: http://babrodtk.at.ifi.uio.no/

SINTEF