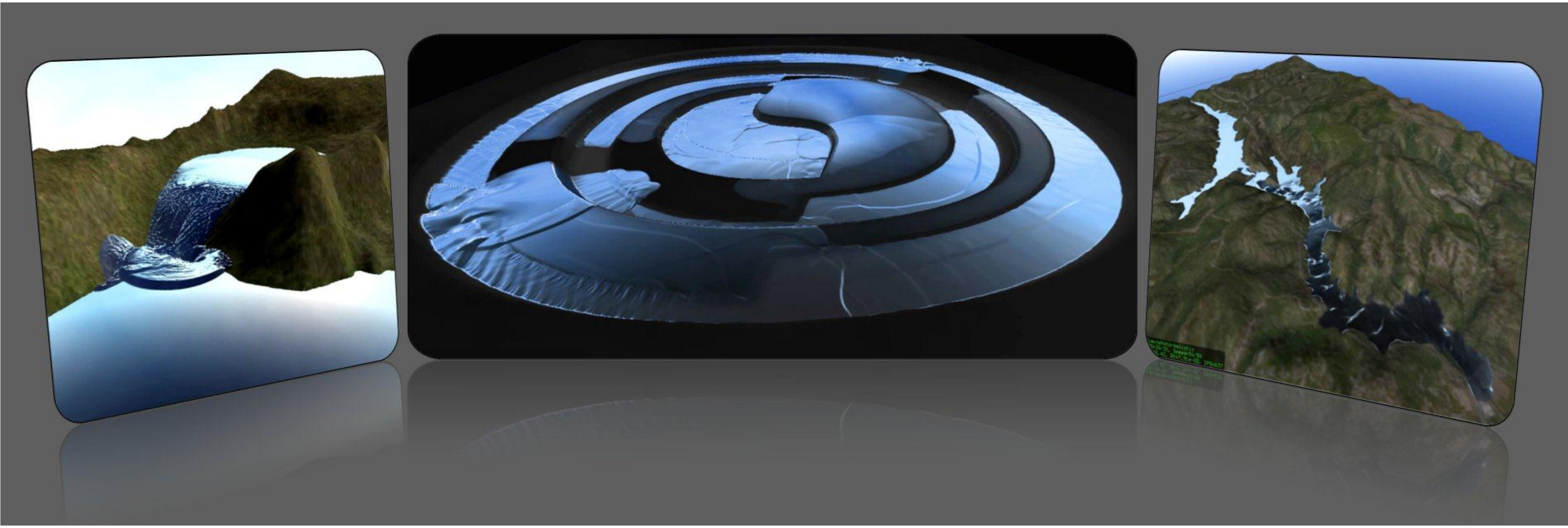
Reporting Performance in the Third Age of GPU Computing

- How to optimize, verify and validate GPU codes



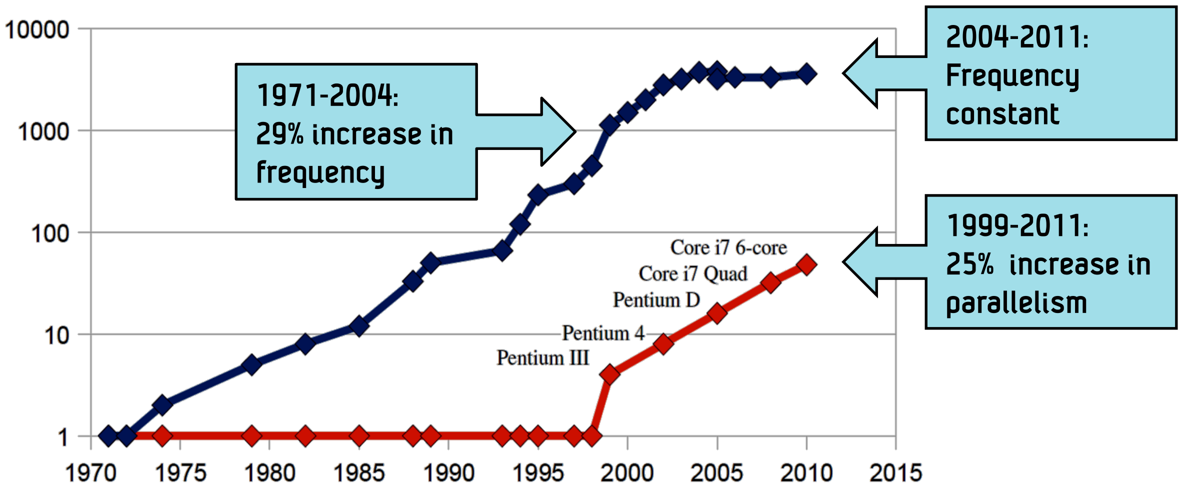
Computational Methods in Water Resources 2012,
University of Illinois at Urbana-Champaign
2012-06-20

André R. Brodtkorb, Ph.D.
SINTEF ICT, Department of Applied Mathematics, Norway
Andre.Brodtkorb@sintef.no

Talk Outline

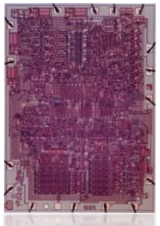
- Short introduction to GPU computing
- Performance assessment, verification, and validation
- Summary

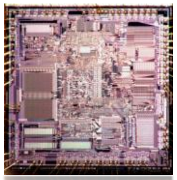
Frequency scaling stops, parallelism begins

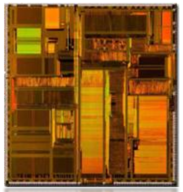


A serial program uses 2% of available resources!

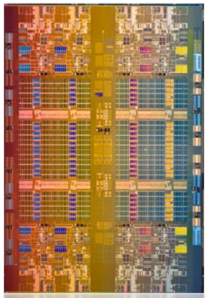
Parallelism technologies:
- Multi-core (8x)
- Hyper threading (2x)
- AVX/SSE/MMX/etc (8x)


1971: Intel 4004,
2300 trans, 740 KHz

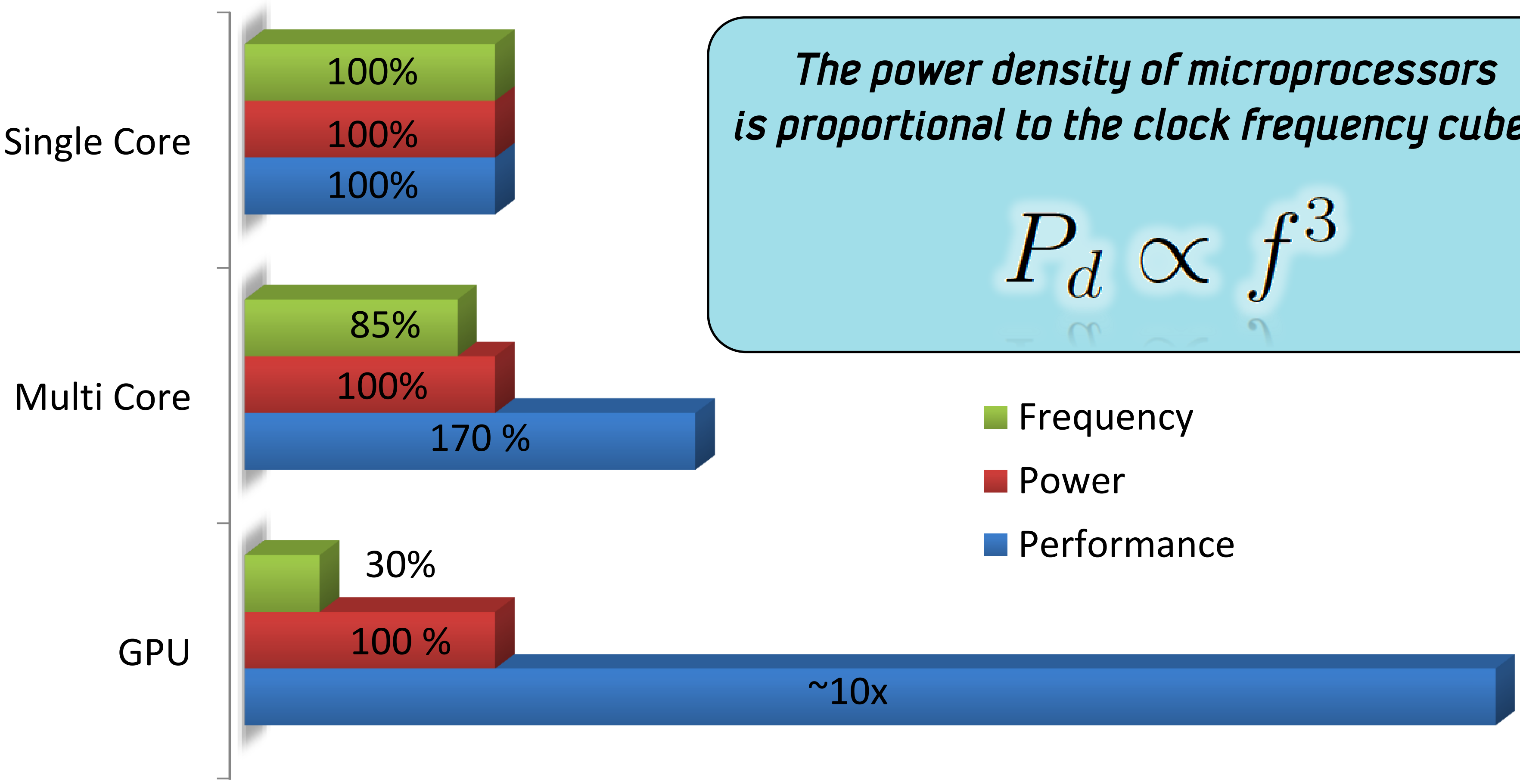

1982: Intel 80286,
134 thousand trans, 8 MHz


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


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


2010: Intel Nehalem,
2.3 bill. trans, 8 X 2.66 GHz

How does parallelism help?

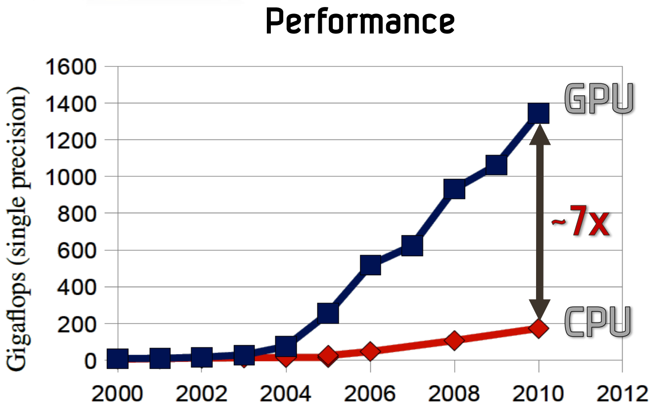


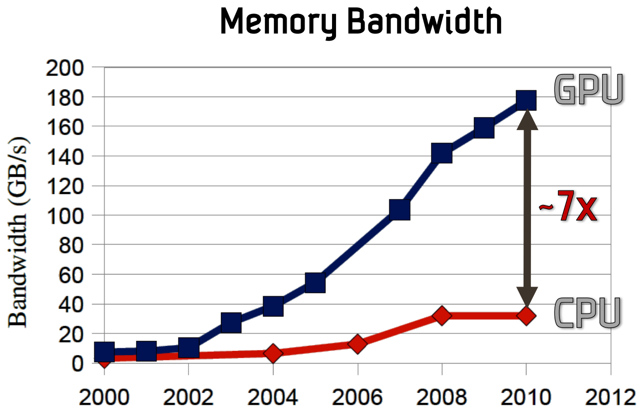
The power density of microprocessors is proportional to the clock frequency cubed:

$$P_d \propto f^3$$

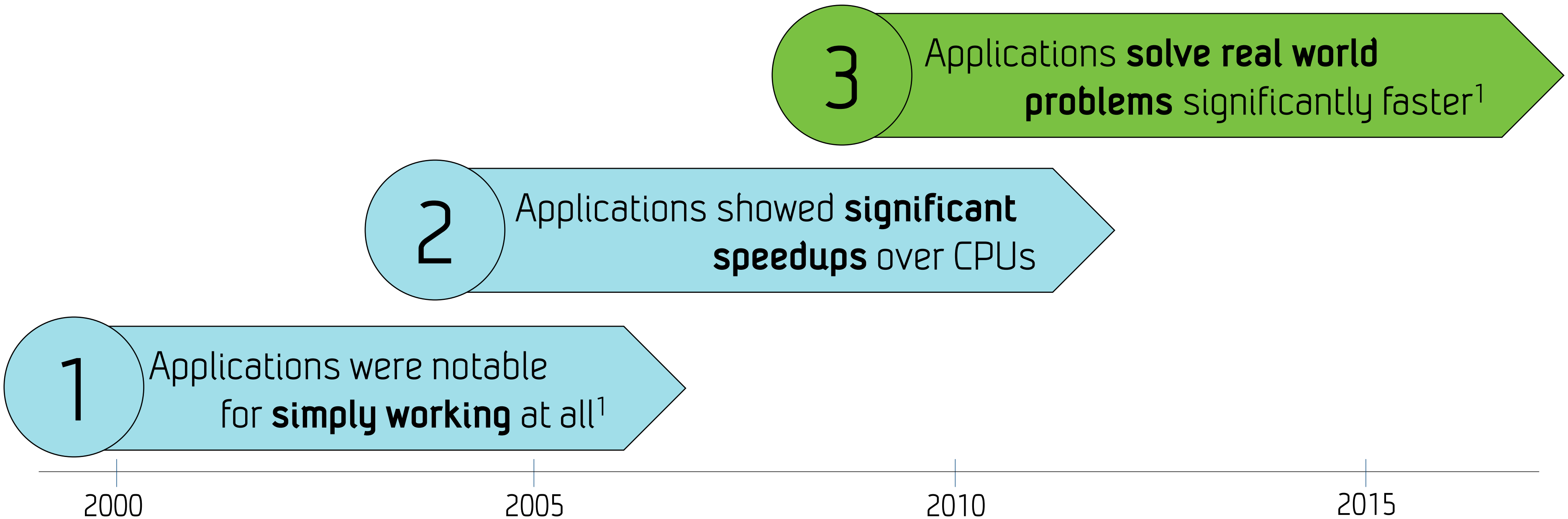
Massive parallelism: The Graphics Processing Unit

	CPU	GPU
Cores	4	16
Float ops / clock	64	1024
Frequency (MHz)	3400	1544
GigaFLOPS	217	1580
Memory (GiB)	32+	3



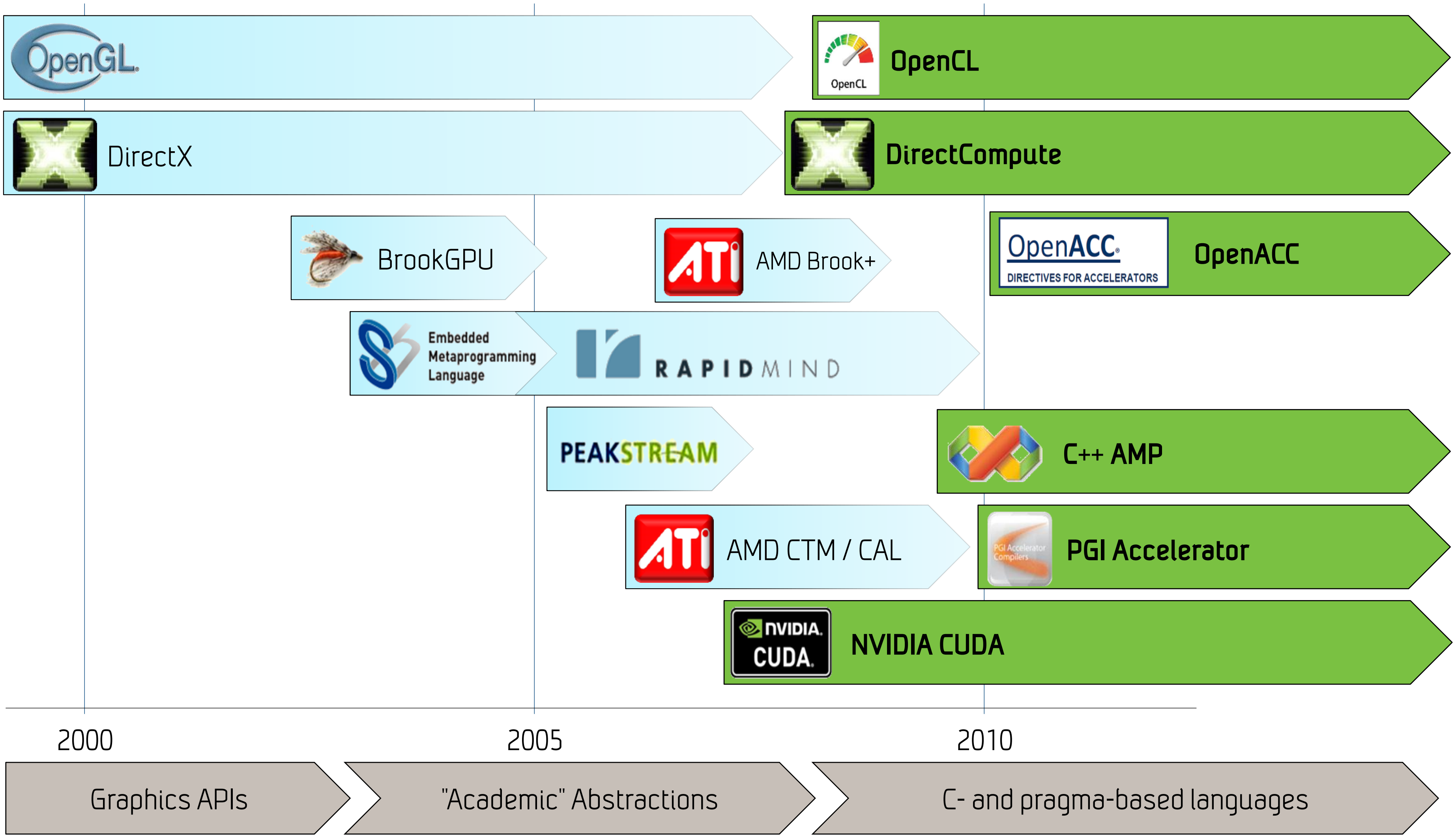


The Ages of GPU Computing

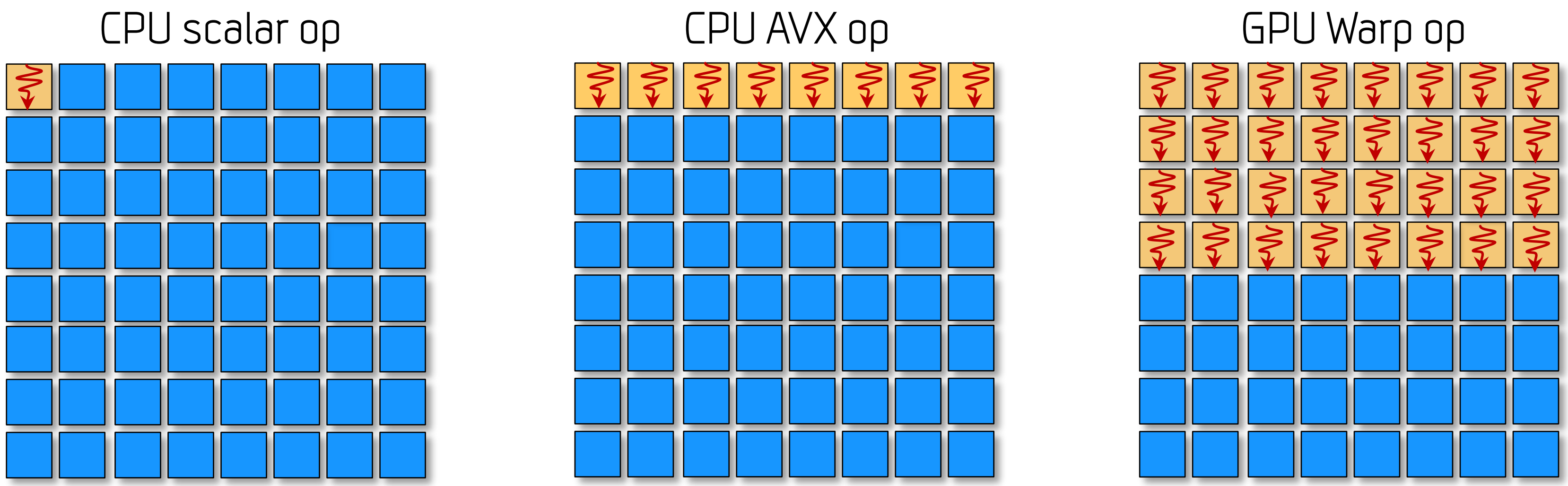


[1] GPU Computing, Owens et al., proceedings of the IEEE, 2008

GPU Programming Languages



GPU Execution model



CPU scalar op	• 1 thread, 1 operand on 1 data element
CPU SSE/AVX op	• 1 thread, 1 operand on 2-8 data elements
GPU Warp op	• 1 **warp** = 32 threads, 32 operands on 32 data elements • Exposed as **individual threads** • Actually runs the **same instruction** • Divergence implies **serialization and masking**

Publishing a GPU Implementation

- **Golden path of least resistance:**
  - Implement whatever you have chosen
  - Demonstrate a 100x speedup over the CPU for a toy problem (and make sure to follow all guidelines in [1,2])
  - Publish in your favorite journal
  - "1964 Buick"

Highlights from [1,2]:

. . .

6. Compare full (or even multiple) GPU performance to a single CPU core.
7. Compare heavily optimized GPU code to unoptimized CPU code.

. . .

[1] David H. Bailey, **Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers**, Supercomputing Review, 1991.
[2] Scott Pakin, **Ten Ways to Fool the Masses When Giving Performance Results on GPUs**, HPC Wire, 2011

Publishing a GPU Implementation

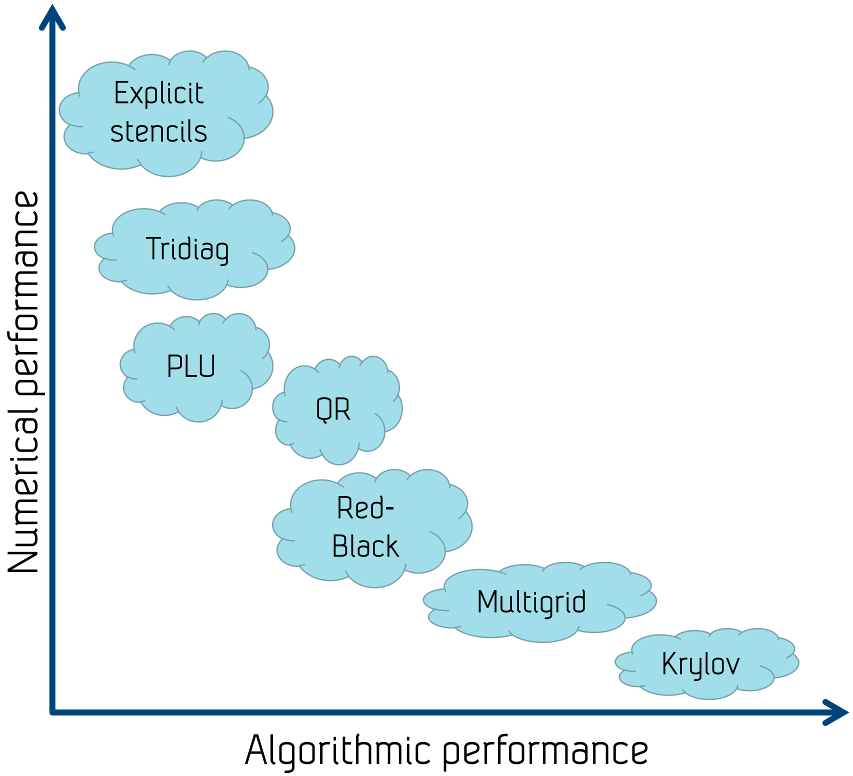
The "right" way:

1. Choose a **real-world problem** to solve
2. Design or choose an **algorithm that fits well with the GPU** execution model
3. Make sure your implementation **gives correct results** and **assess performance fairly**
4. (If you absolutely must, perform a *fair* comparison against a CPU implementation)
5. **Publish** in your favorite journal

> *"In established engineering disciplines a 12 % improvement, easily obtained, is never considered marginal and I believe the same viewpoint should prevail in software engineering"*
>
> --Donald Knuth

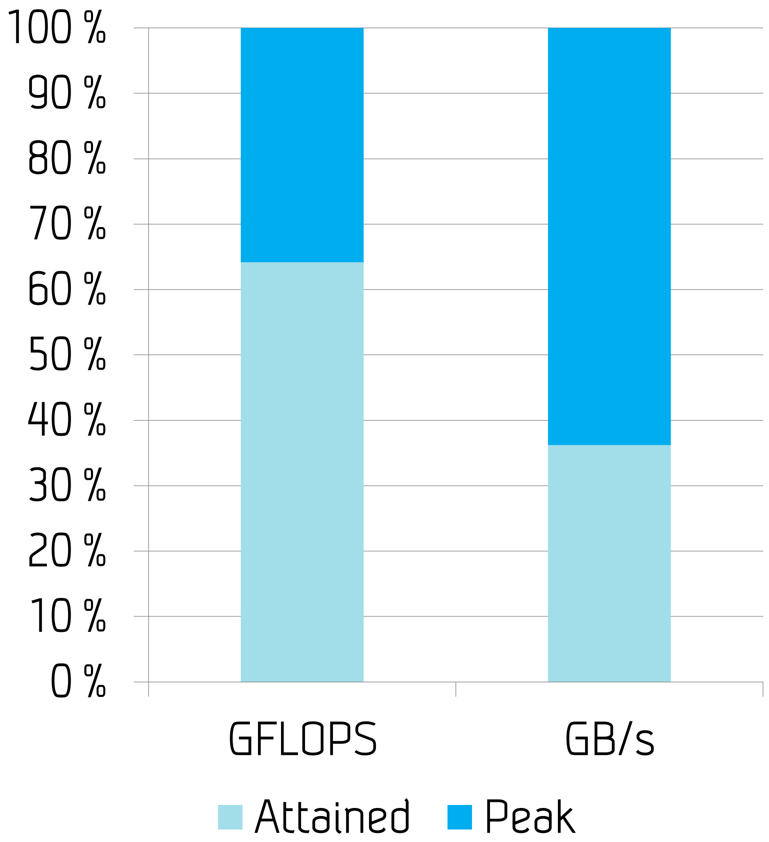
Choosing a solution method for GPUs

- For all problems, the total performance is the product of the algorithmic **and** the numerical performance
  - Your mileage may vary: algorithmic performance is highly problem dependent
- On GPUs, it can sometimes be a good idea to do a "stupid thing many times"
- Sparse linear algebra solvers often have high algorithmic performance
  - But only able to utilize a fraction of the capabilities of CPUs, and **worse on GPUs**
- **Explicit stencil** schemes (compact stencils) often able to **efficiently exploit the GPU execution model**
  - But can have low algorithmic performance



Assessing performance

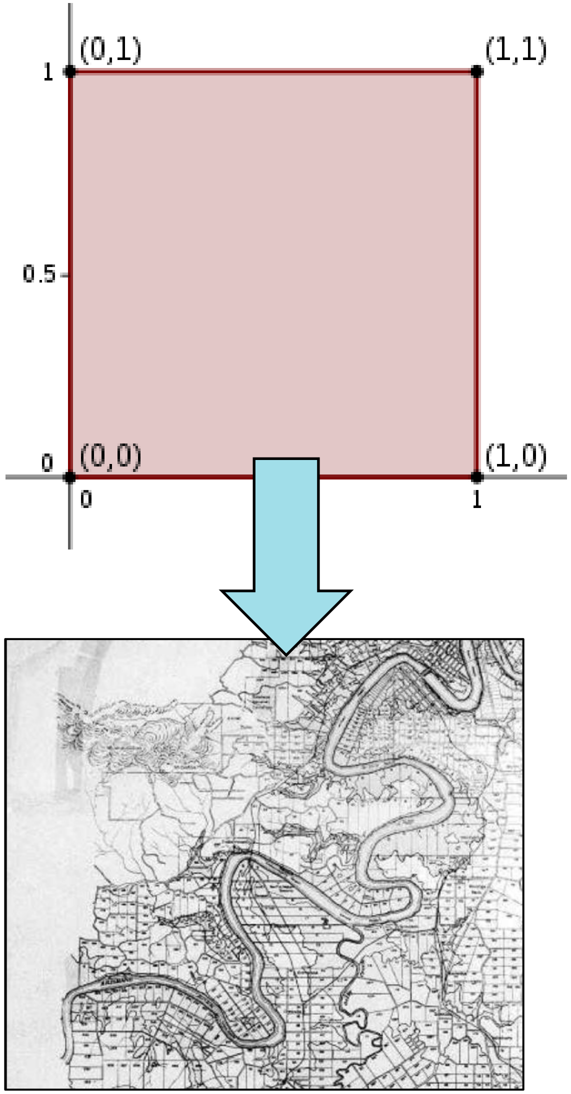
- Different ways of assessing performance
  - Numerical performance does not tell all
  - Number of iterations required, size of time step, and other algorithmic parameters are just as important
- Performance assessment often goes hand in hand with optimization [1]
- Profile your code, and see what percentage of peak performance you attain
  - Use tools such as the CUDA Profiler
  - You should reach near-peak GFLOPS or GB/s, or explain why not
  - Gives an impression of scalability
- Speedups can be dishonest
  - Comparison of apples and pears



[1] P. Micikevicius, Analysis-Driven Optimization, GPU Technology Conference, 2010

Solving real-world problems

- Demonstrating a speedup on the unit square is a proof of concept
- In the third age of GPU Computing, we need to solve real-world problems, and solve them fast!
- Real-world problems increase complexity of problem formulation
- Real-world problems have high requirements to accuracy



Unit square image from Wikipedia, user Magnus Manske

Accuracy and Error

- Garbage in, garbage out

- Simulations have many sources for errors
  - Humans!
  - Model and parameters
    - Friction coefficient estimation
    - "Magic" numerical parameters
    - Choice of boundary conditions
    - Numerical dissipation
    - Handling of wetting and drying
  - Measurement
    - Radar / Lidar / Stereoscopy
    - Low spatial resolution
    - Low vertical accuracy
  - Gridding
    - Can require expert knowledge
  - Computer precision
  - …

Recycle image from recyclereminders.com
Cray computer image from Wikipedia, user David.Monniaux

Single Versus Double Precision

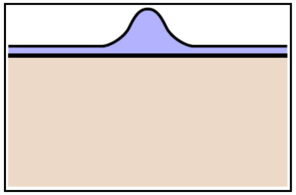
Given erroneous data, double precision calculates a more accurate (but still wrong) answer

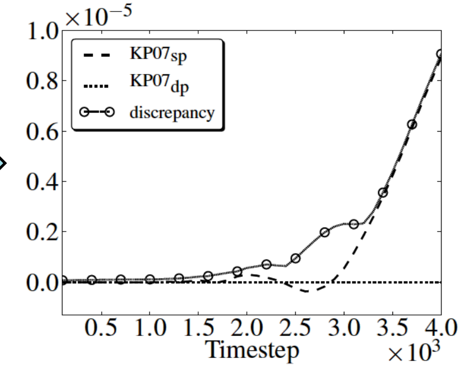
Single precision benefits:

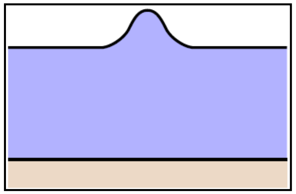
- Uses *half* the storage space
- Uses *half* the bandwidth
- Executes (at least) *twice* as fast

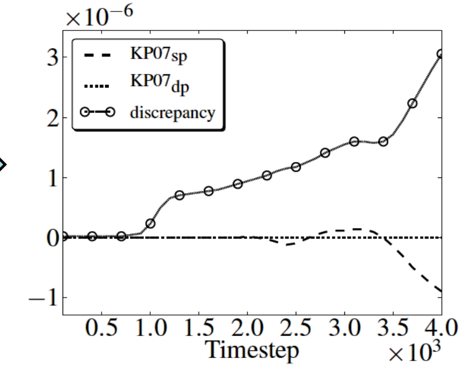
Single Versus Double Precision Example

- Three different test cases
  - Low water depth (wet-wet)
  - High water depth (wet-wet)
  - Synthetic terrain with dam break (wet-dry)

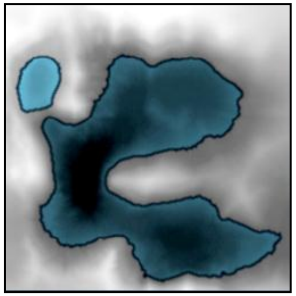


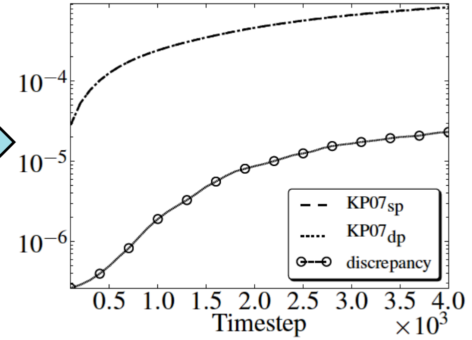




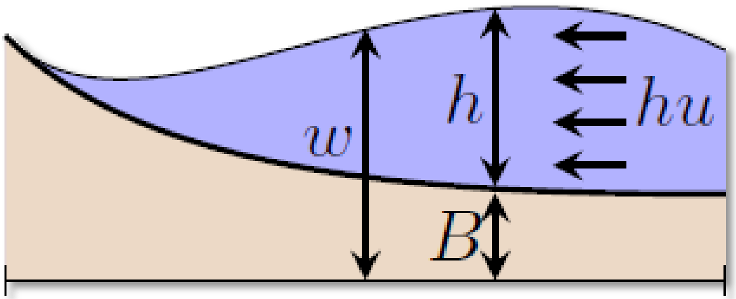


- Conclusions:
  - Loss in conservation on the order of machine epsilon
  - Single precision gives larger error
  - Errors related to the wet-dry front is more than an order of magnitude larger (model error)
  - **Single precision is sufficiently accurate for this scheme**





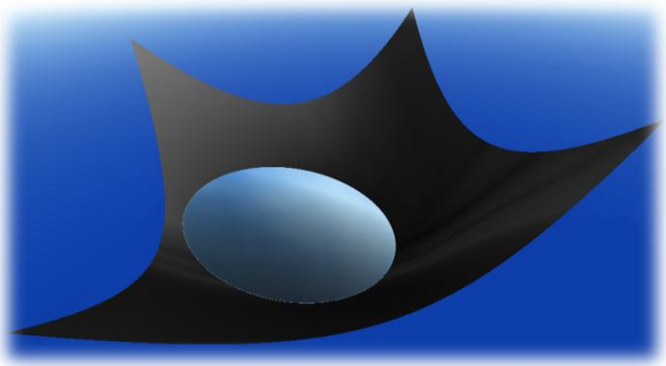
More on Accuracy



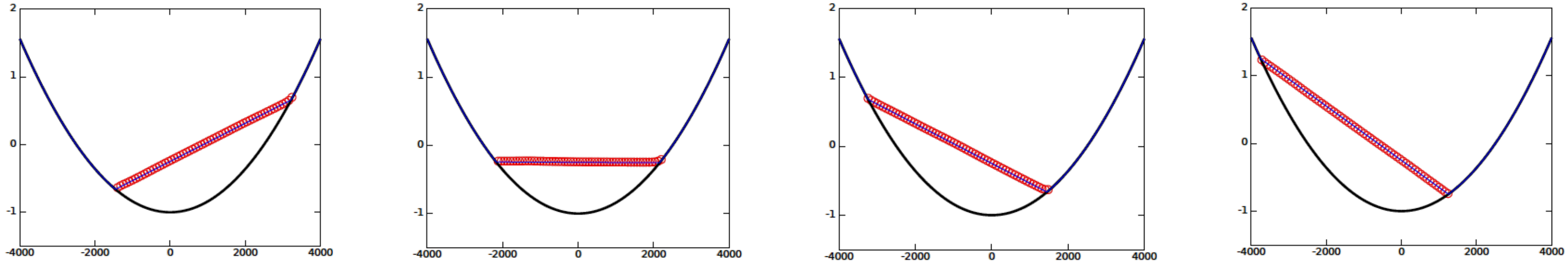
- We were experiencing large errors in conservation of mass for special cases
- The equations is written in terms of w = B+h to preserve "lake at rest"
- Large B, and small h
  - The scale difference gives major floating point errors (h flushed to zero)
  - Even double precision is insufficient
- Solve by storing only h, and reconstruct w when required!
  - Single precision sufficient for most real-world cases
  - Always store the quantity of interest!

2D Verification: Parabolic basin

- Analytical 2D parabolic basin (Thacker)
  - Planar water surface oscillates
  - 100 x 100 cells
  - Horizontal scale: 8 km
  - Vertical scale: 3.3 m

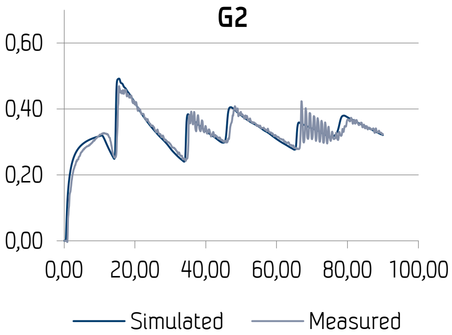


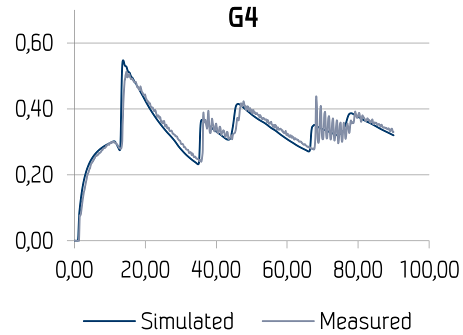
- Simulation and analytical match well
  - But, as most schemes, **growing errors along wet-dry interface** (**model error…**)

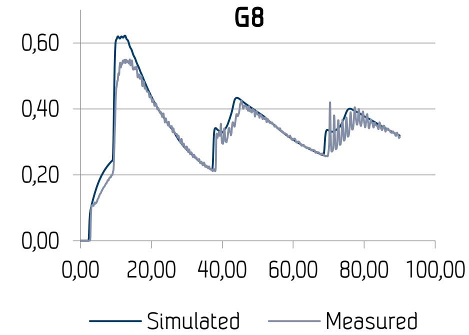


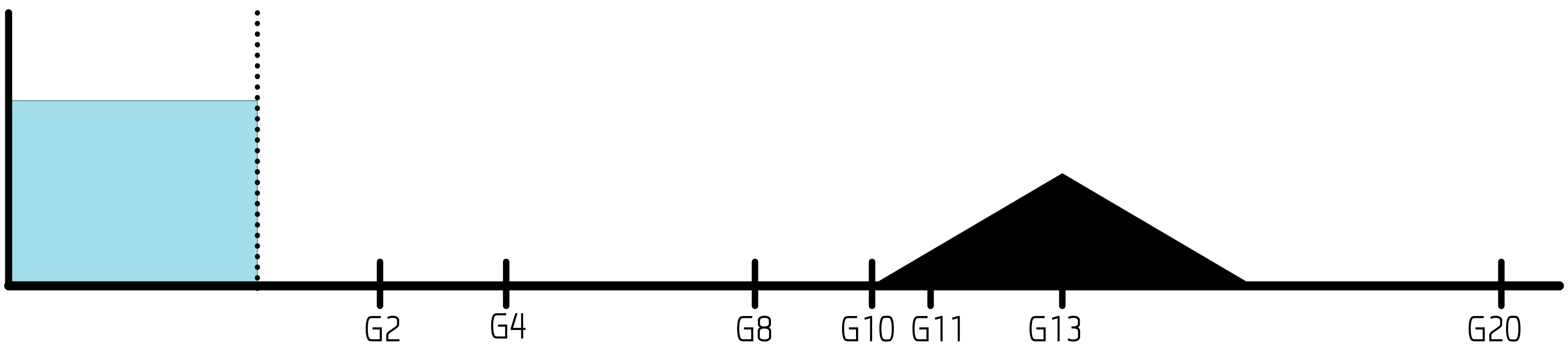
(a) $t \approx 1\pi/8\omega$ (1392.85 s) (b) $t \approx 2\pi/8\omega$ (2787.95 s) (c) $t \approx 3\pi/8\omega$ (4198.64 s) (d) $t \approx 4\pi/8\omega$ (5559.27 s)

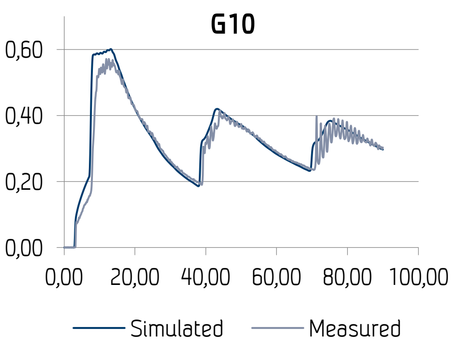
1D Validation: Flow over Triangular bump (90s)

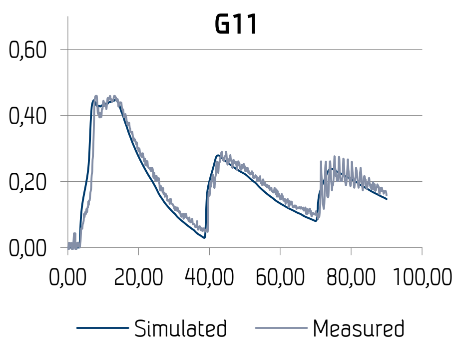


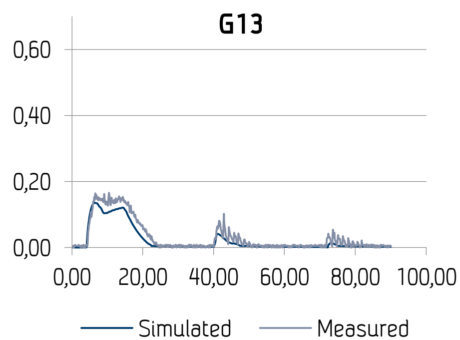


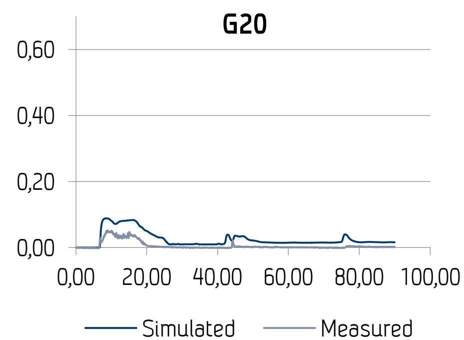






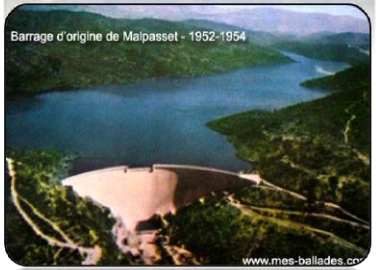


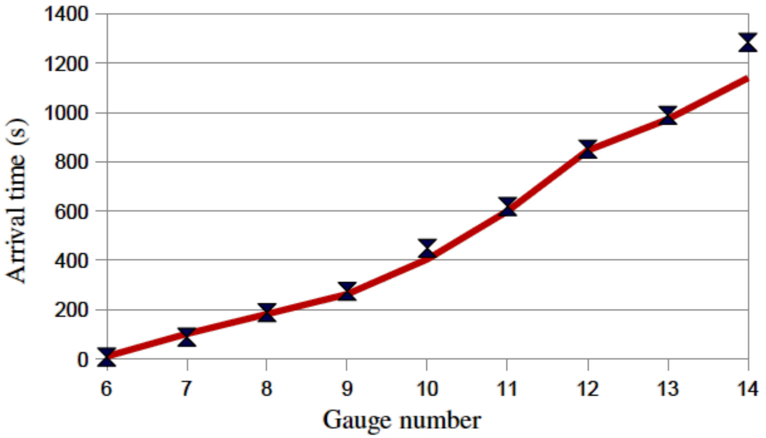




2D Validation: Barrage du Malpasset

- South-east France near Fréjus: Barrage du Malpasset
  - Double curvature dam, 66.5 m high, 220 m crest length, 55 million m^3
  - Bursts at 21:13 December 2nd 1959
    - Reaches Mediterranean in 30 minutes (speeds up-to 70 km/h)
    - 423 casualties, $68 million in damages
  - Validate against experimental data from 1:400 model
    - 482 000 cells (1099 x 439 cells)
    - 15 meter resolution
- **Our results match experimental data very well**
  - Discrepancies at gauges 14 and 9 present in most (all?) published results





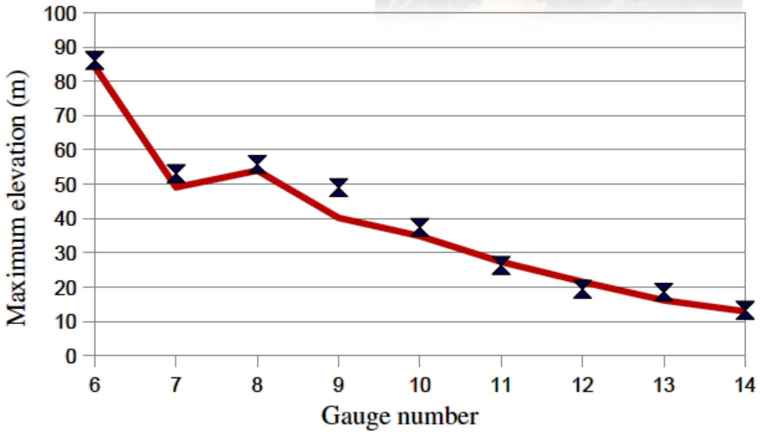
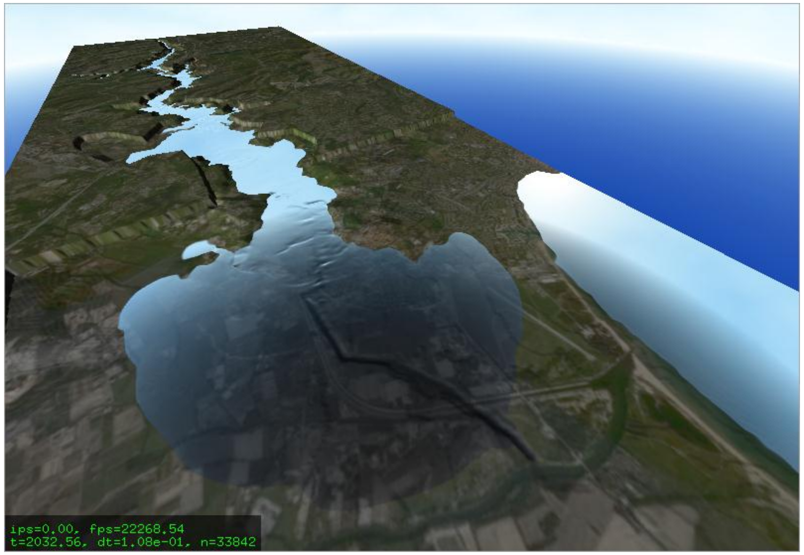
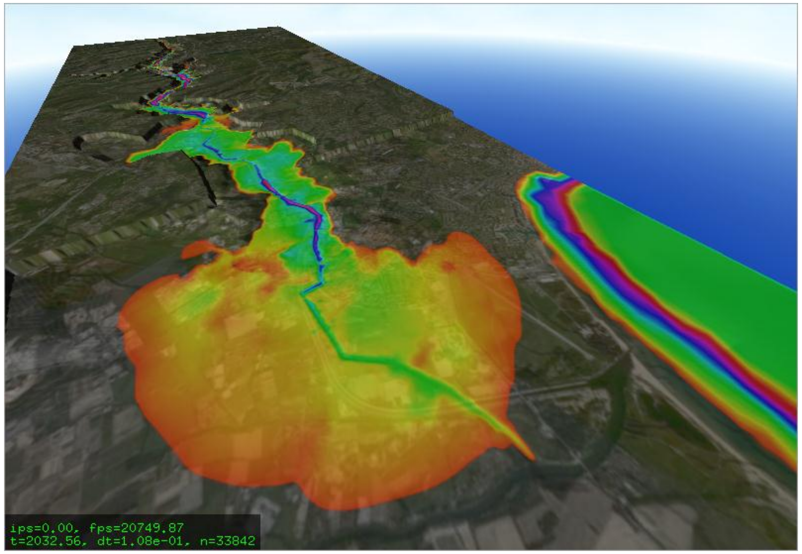


Image from google earth, mes-ballades.com

Video





http://www.youtube.com/watch?v=FbZBR-FjRwY

Thank you for listening!

Talk material based on work on our simulator engine. Some references:

- A. Brodtkorb, M. L. Sætra, Explicit Shallow Water Simulations on GPUs: Guidelines and Best Practices, CMWR Proceedings, 2012
- A. Brodtkorb, M. L. Sætra, M. Altinakar, Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation, Computers & Fuids, 55, (2011), pp 1--12.
- A. R. Brodtkorb, T. R. Hagen, K.-A. Lie and J. R. Natvig, Simulation and Visualization of the Saint-Venant System using GPUs, *Computing and Visualization in Science*, 13(7), (2011), pp. 341--353

Contact:

André R. Brodtkorb

Email: Andre.Brodtkorb@sintef.no

Homepage: http://babrodtk.at.ifi.uio.no/

Youtube: http://youtube.com/babrodtk

SINTEF: http://www.sintef.no/heterocomp