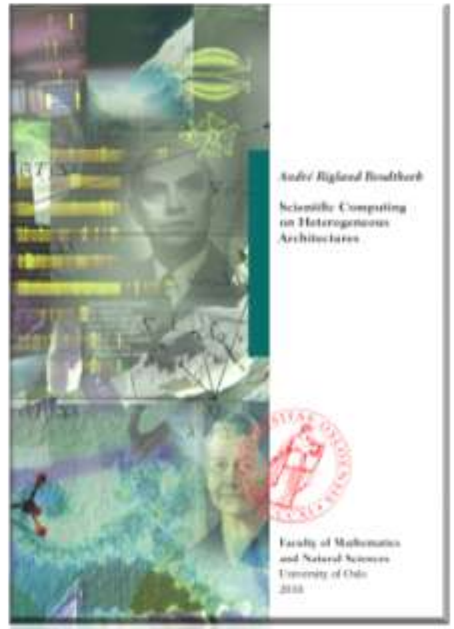


# SCIENTIFIC COMPUTING ON HETEROGENEOUS ARCHITECTURES

Ph.D. Thesis Presentation
André Rigland Brodtkorb
2010-12-17

# Outline

- Introduction
  - The advent of heterogeneous architectures
  - Overview of research

- Research topics
  - Heterogeneous Architectures
  - Linear Algebra on the GPU through MATLAB
  - Shallow Water Simulations on the GPU

- Summary

# Brief History of the Microprocessor

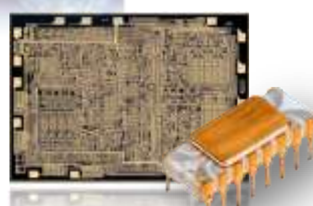
**1942: Digital Electric Computer**
(Atanasoff and Berry)

1956

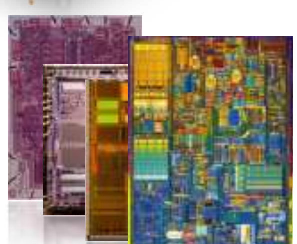
**1947: Transistor**
(Shockley, Bardeen, and Brattain)

2000

**1958: Integrated Circuit**
(Kilby)

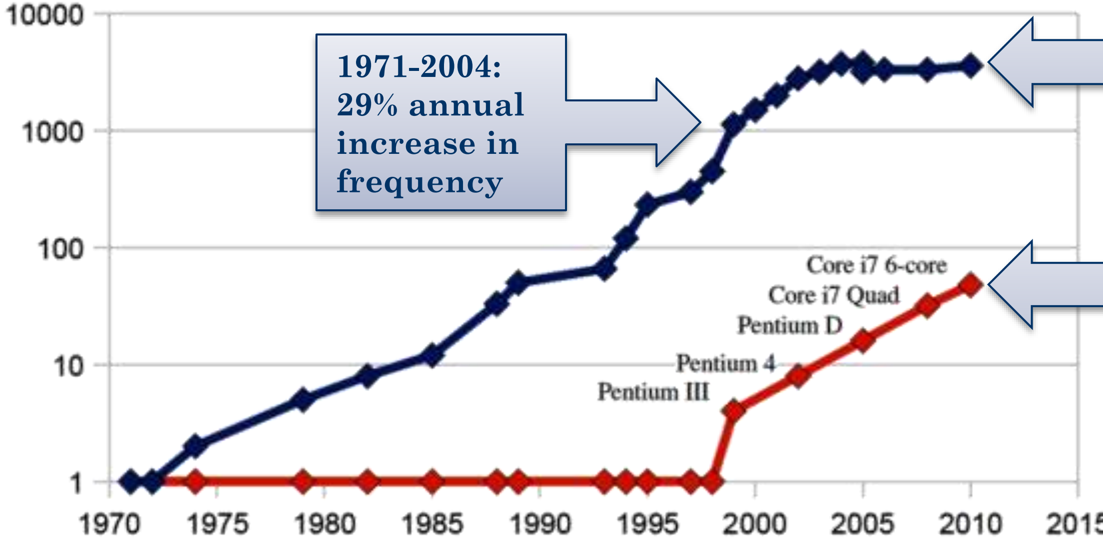


**1971: Microprocessor**
(Hoff, Faggin, Mazor)



**1971- More transistors**
(Moore, 1965)

# From Frequency to Parallelism




**1971-2004: 29% annual increase in frequency**

**2004-2010: Frequency remains constant**

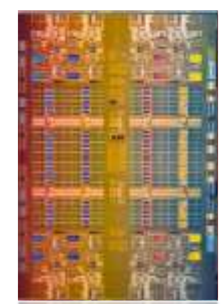
**1999-2010: 25% annual increase in parallelism**

**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**

# Why Parallelism?

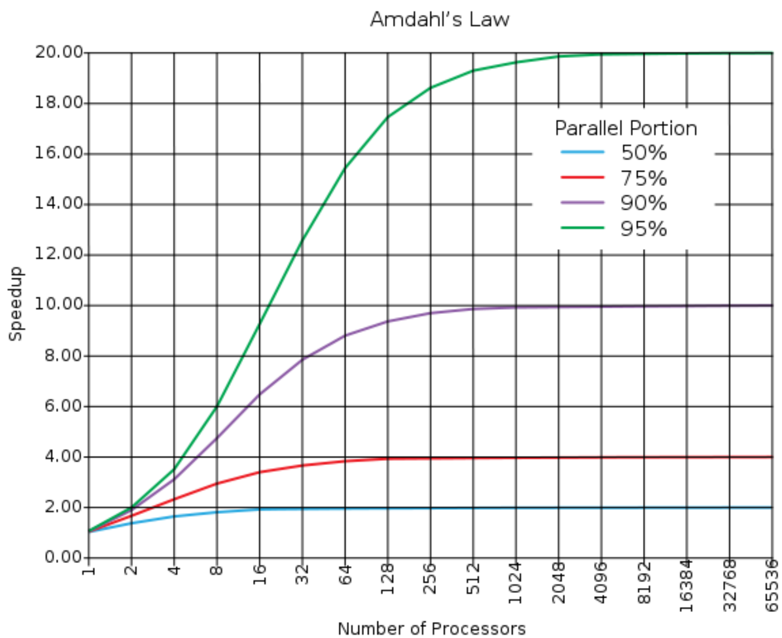
***The power density of microprocessors is proportional to the clock frequency cubed:***

$$P_d \propto f^3$$

| | Single-core | Dual-core | |
|---|---|---|---|
| Frequency | 100% | 85% | |
| Performance | 100% | 90% | 90% |
| Power | 100% | 100% | |

# Heterogeneous Architectures

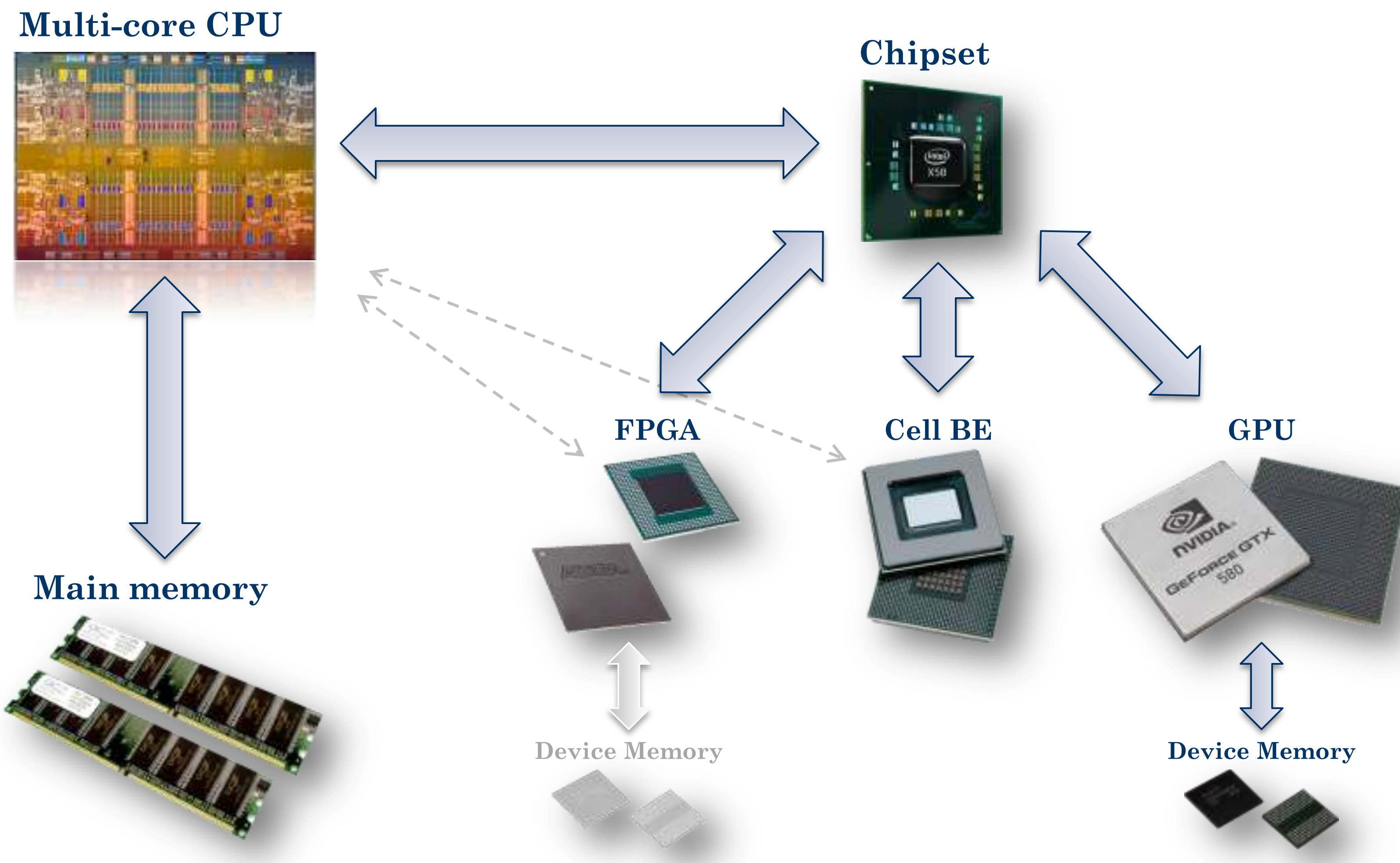
- Amdahl's law:
  - There is a limit to speedup offered by parallelism
  - Serial parts become the bottleneck
- Heterogeneous architectures use both serial and parallel resources
  - Efficient serial CPUs
  - Efficient parallel accelerators (GPUs, FPGAs, Cell BEs, etc.)




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

Graph from Wikipedia: Amdahls law

# Todays Heterogeneous Architectures



# Focus #1: Linear Algebra

$$\begin{aligned} x + y &= 10 \\ x - y &= 4 \end{aligned} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \end{bmatrix} \Rightarrow \begin{aligned} x &= 7 \\ y &= 3 \end{aligned}$$

- A fundamental toolbox in scientific computing
  - The study of vectors and matrices
  - A classical problem is to solve Ax = b

# Example - The Heat Equation

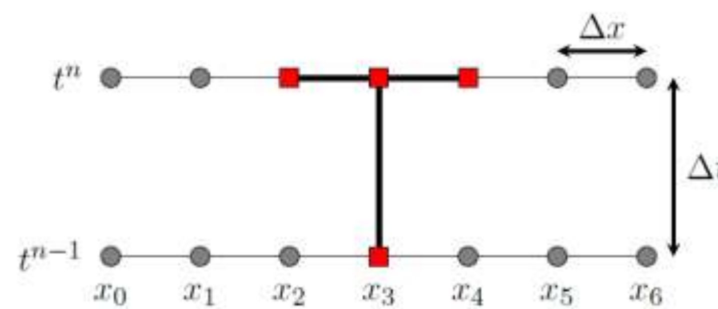
- Describes diffusive heat conduction
- Prototypical partial differential equation

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

- u is the temperature, kappa is the diffusion coefficient, t is time, and x is space.

1. Replace continuous derivatives with discrete derivatives

$$\frac{1}{\Delta t}(u_i^n - u_i^{n-1}) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$



# Example - The Heat Equation

2. Gather the unknowns into an algebraic equation per cell

$$-ru_{i-1}^{n} + (1+2r)u_{i}^{n} - ru_{i+1}^{n} = u_{i}^{n-1}, \qquad r = \frac{\kappa \Delta t}{\Delta x^2}$$

3. Write as a system of linear equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -r & 1+2r & -r & 0 & 0 & 0 & 0 \\ 0 & -r & 1+2r & -r & 0 & 0 & 0 \\ 0 & 0 & -r & 1+2r & -r & 0 & 0 \\ 0 & 0 & 0 & -r & 1+2r & -r & 0 \\ 0 & 0 & 0 & 0 & -r & 1+2r & -r \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0^n \\ u_1^n \\ u_2^n \\ u_3^n \\ u_4^n \\ u_5^n \\ u_6^n \end{bmatrix} = \begin{bmatrix} u_0^{n-1} \\ u_1^{n-1} \\ u_2^{n-1} \\ u_3^{n-1} \\ u_4^{n-1} \\ u_5^{n-1} \\ u_6^{n-1} \end{bmatrix}$$

4. Solve Ax=b using standard methods

Linear algebra is computationally demanding
Use the graphics card to accelerate

# Focus #2: Stencil Computations

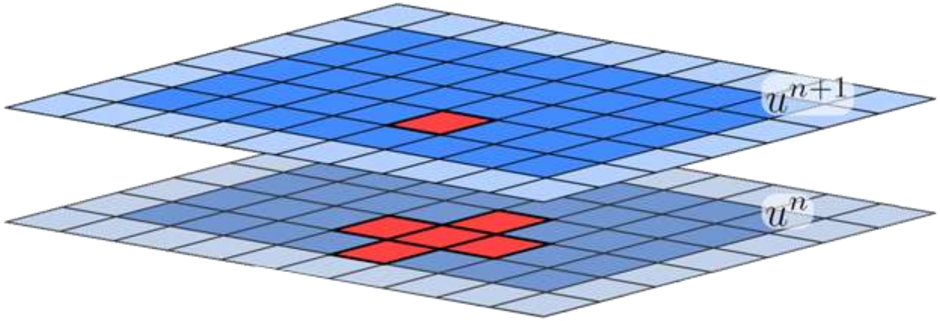
- The heat equation can also lead to an explicit scheme

$$\frac{1}{\Delta t}(u_i^{\boxed{n}} - u_i^{\boxed{n-1}}) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

$$\Downarrow$$

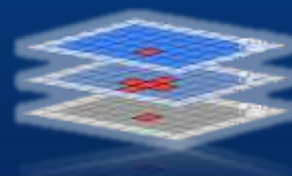
$$\frac{1}{\Delta t}(u_i^{\boxed{n+1}} - u_i^{\boxed{n}}) = \frac{\kappa}{\Delta x^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

- Calculate a new value as a weighted sum of neighbours
  - Often memory bound
  - Embarassingly parallel

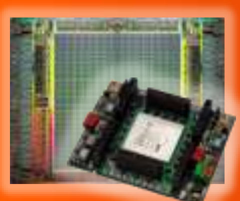



Stencil computations are embarassingly parallel
Use the graphics card to accelerate

## Scientific Computing



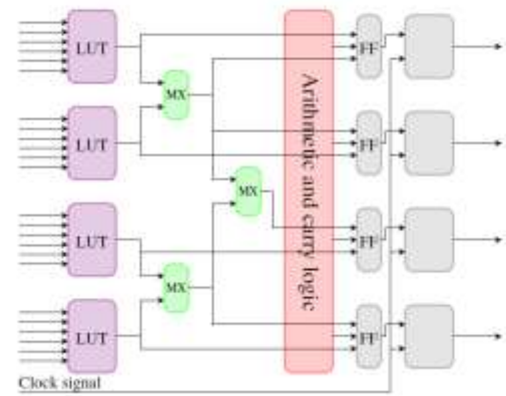
**Heterogeneous Architectures**



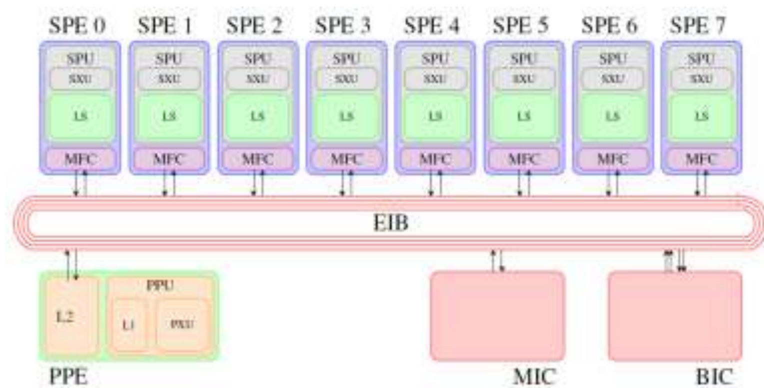
**Main Focus**

# FPGAs and the Cell BE

- FPGAs
  - Used in specialized and embedded systems
  - Tens of thousands of *configurable logic blocks*
  - Advanced routing network
  - Extremely power efficient

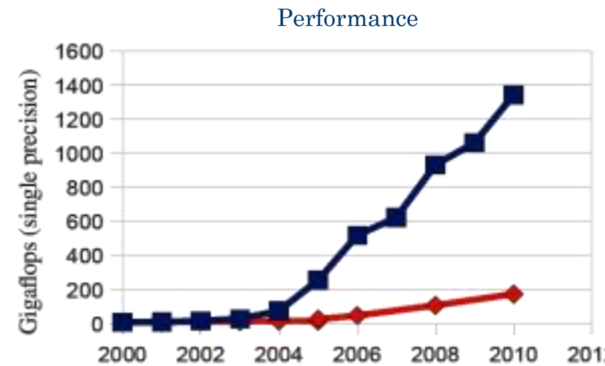


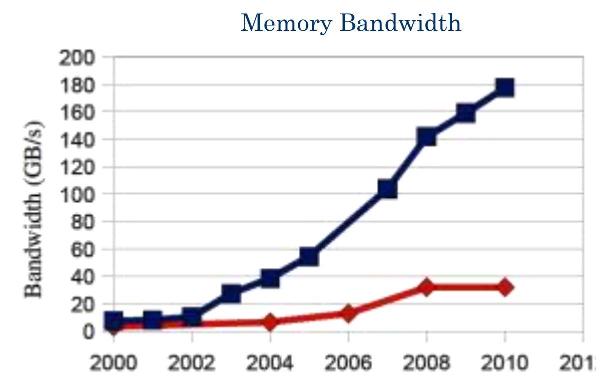

- Cell BE
  - Used in Roadrunner (13k PowerXcell), and Condor (1.7k PS3), and PS3s
  - A nine-core heterogeneous chip
  - Peak performance is achievable
  - Future is uncertain (at best)



# GPUs

## The Graphics Processing Unit (GPU)

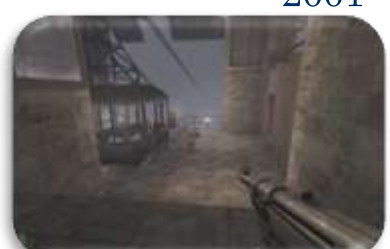




1981

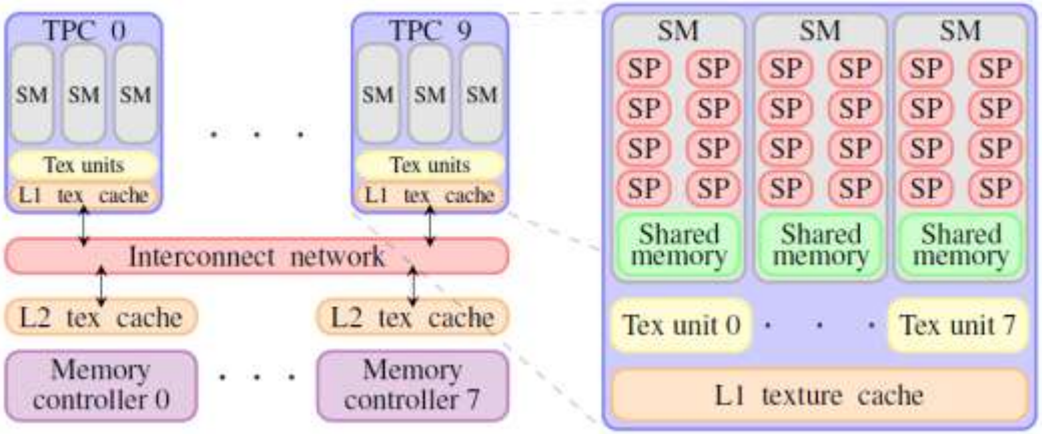
1992

2001



2009

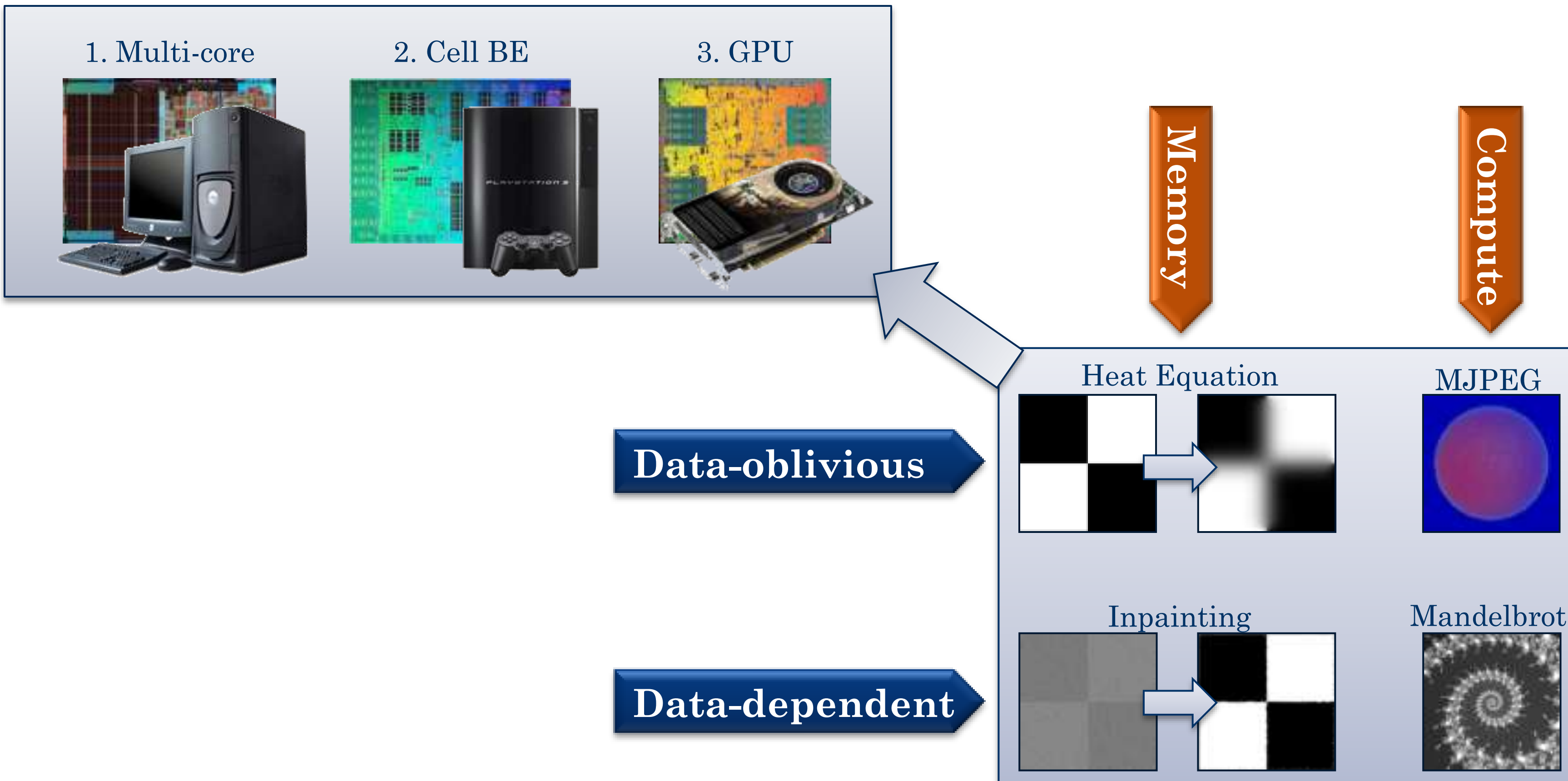
# GPU Architecture




- Tens of "cores" (SM)
  - Each core contains 8-16 arithmetic-logic units
  - Logically 32-way SIMD
  - Shared memory as a programmable cache
  - Graphics functionality (texture cache, interpolation, trigonometry)
  - Massively threaded
- Easy to get started programming
- Can be difficult to map algorithms onto its parallel execution model

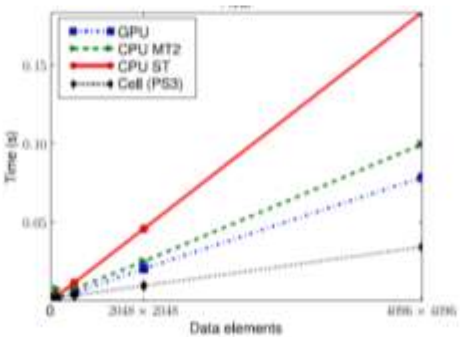
# Mapping Algorithms to Architectures

Embarrassingly parallel algorithms mapped to parallel architectures

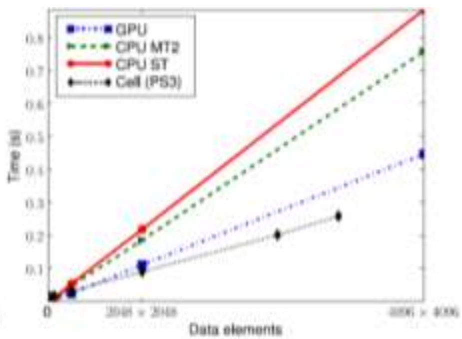


# Mapping Algorithms to Architectures

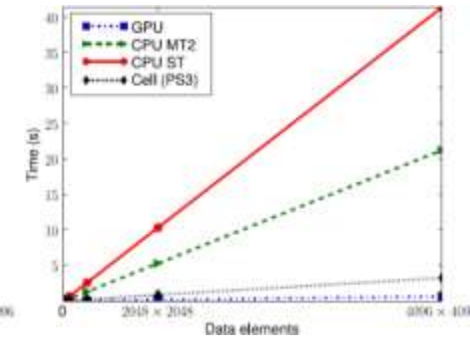
**Heat**



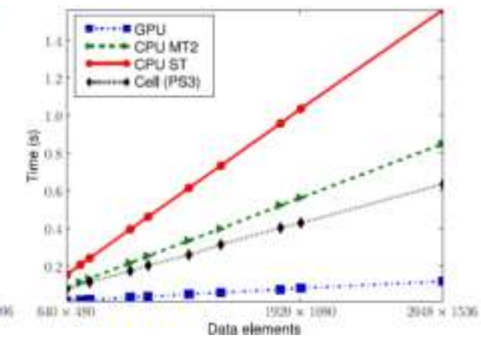

**Inpainting**




**Mandelbrot**




**MJPEG**

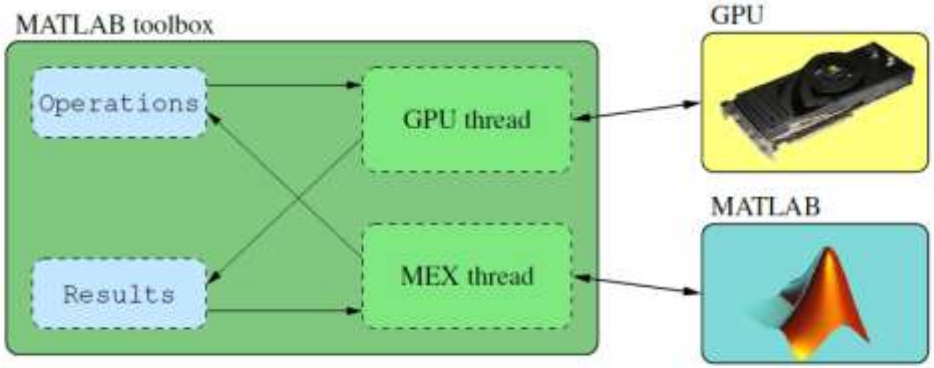



| | |
|---|---|
| GPU | ■ Highest performance for computationally bound algorithms<br>■ Slow data-transfer across PCI-express bus |
| Cell BE | ■ Highest performance for memory bound algorithms<br>■ Slow branching<br>■ Slow CPU core |
| CPU | ■ Scales well for computationally bound algorithms<br>■ Quickly saturates memory bus for memory bound algorithms |

# GPU Toolbox for MATLAB

Accelerate linear algebra in MATLAB

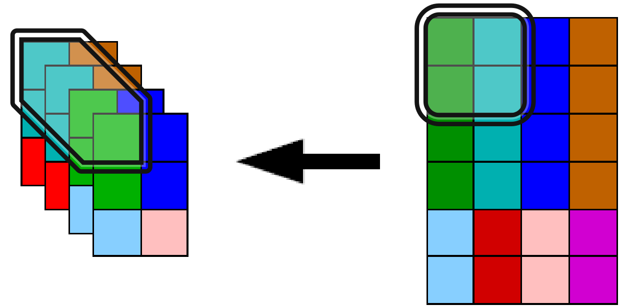
- MATLAB is a high-level tool with more than 1M users
- Linear algebra is a core functionality



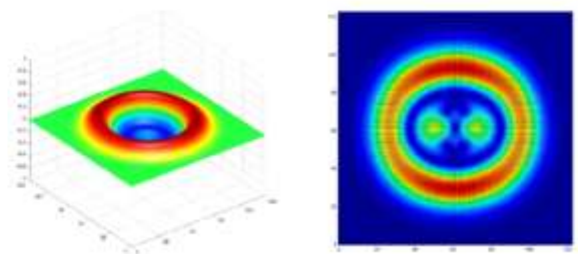

- Coupling MATLAB and the GPU is nontrivial
  - Operations implemented through OpenGL (predates CUDA)
  - MATLAB calls C functions through the MEX API
  - Neither MEX nor OpenGL are thread-safe APIs

# GPU Toolbox for MATLAB

- OpenGL implementation packs data into four-long (color) vectors
  - All operations must use the four-long vectors as the basic unit
  - New 2x2 packing strategy



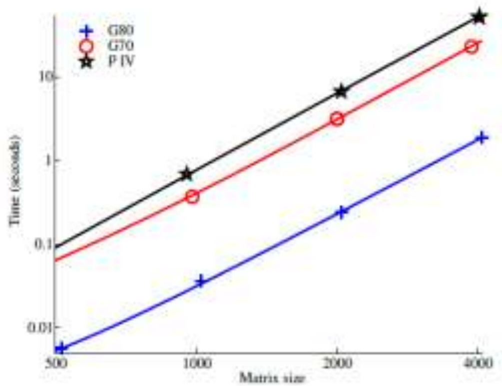
- Implemented Operations:
  - Full matrix-matrix multiplication
  - Gauss-Jordan elimination
  - PLU factorization
  - Tridiagonal Gaussian elimination (banded representation)



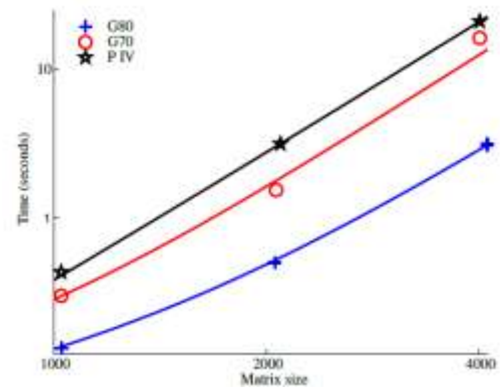
**Tridiagonal GE use: ADI Shallow Water**

# GPU Toolbox for MATLAB

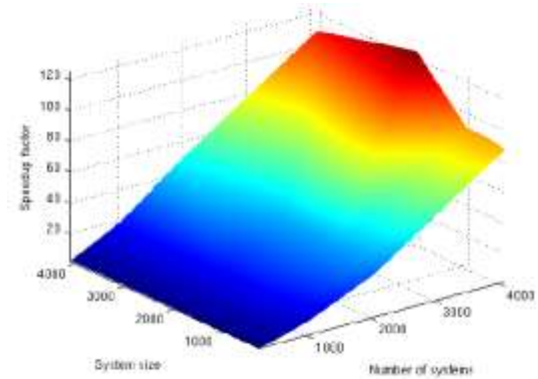
**Matrix multiplication**




**PLU**




**Tridiagonal GE**

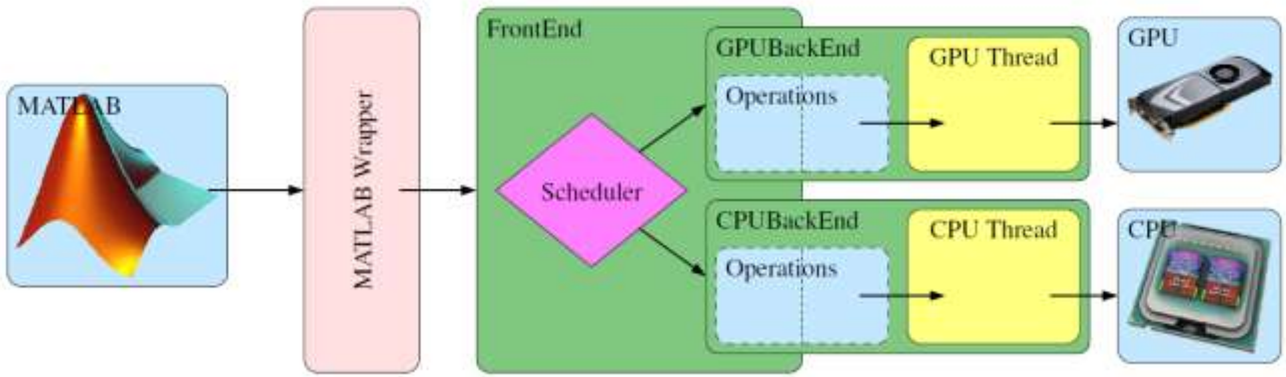



- Large speed-ups for algorithms running on the GPU
- Slow data transfer between GPU and CPU gives a large performance hit
- Solution: Run as many as possible operations back-to-back on the GPU
- Accelereyes has commercialized a similar solution

| | |
|---|---|
| Matrix Multiplication | 31x |
| PLU Factorization | 7x |
| Tridiagonal GE | <125x |

# Asynchronous Linear Algebra

Use multiple compute resources asynchronously




- CUDA was released in 2007
  - Enabled use of GPUs without going through graphics APIs.
  - CUBLAS implements BLAS functionality using GPUs
- Asynchronous execution for higher performance and resource use

# Asynchronous Linear Algebra

- Uses the same basic idea, but use a scheduler to use multiple back-ends
  - Cluster dependent operations and schedule to a specific back-end
  - Includes MATLAB interface.

- Scheduler criteria:
  - Load: Historic, queue, and incoming operation
    (Use compilation auto-tuning stage for initial statistics)
  - Cost of moving dependent operations

- Scheduler imposes small overheads
  - Works well with multiple back-ends
  - But makes sub-optimal choices for heterogeneous back-ends

- GPU Calculations in double precision are equivalent to CPU results

# Shallow Water Simulation on GPUs

Investigate explicit shallow water simulations on GPUs

- The shallow water equations are hyperbolic.
- We can use explicit schemes (stencil computations)
- Most stencil computations are memory bound, but more complex ones can be computationally bound

- When the data is on the GPU, visualize it directly!

# Shallow Water Simulation on GPUs

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghB_x \\ -ghB_y \end{bmatrix}$$

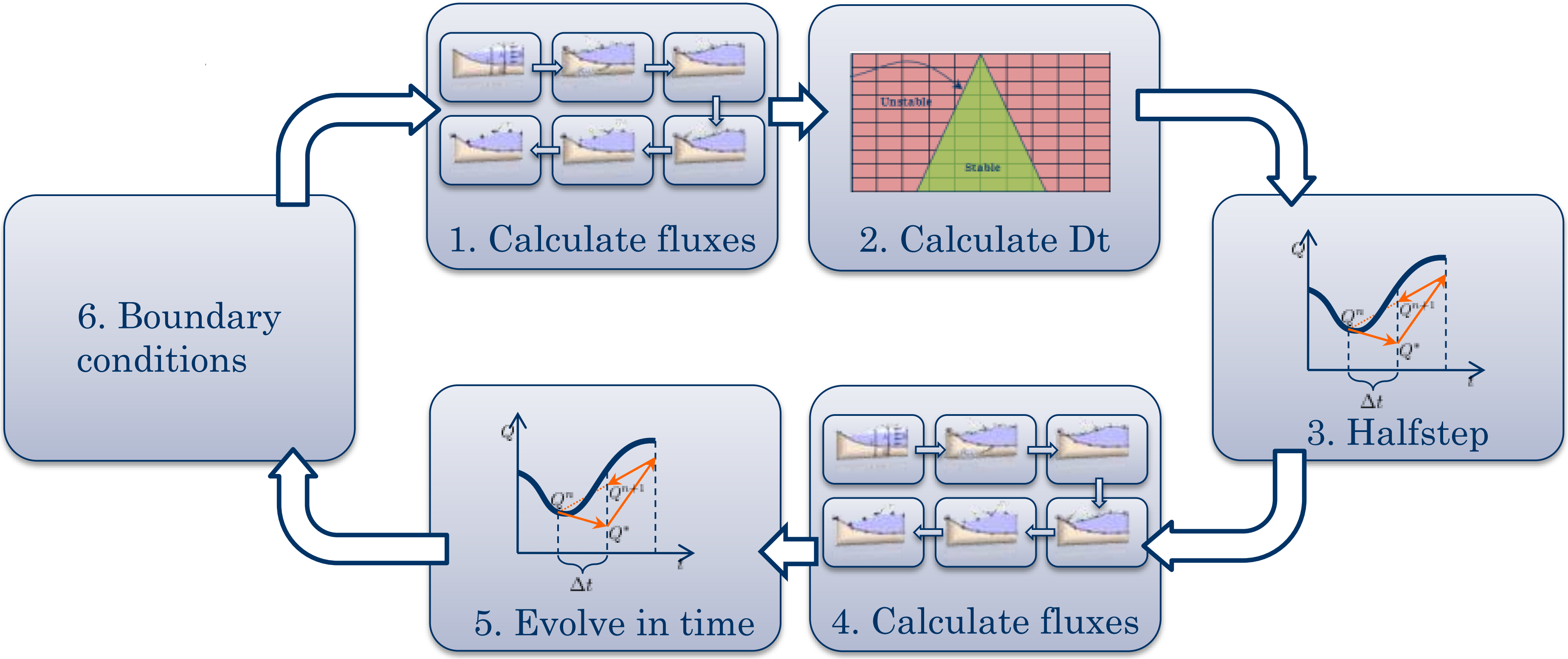
Vector of conserved variables

Flux functions

Bed slope source term

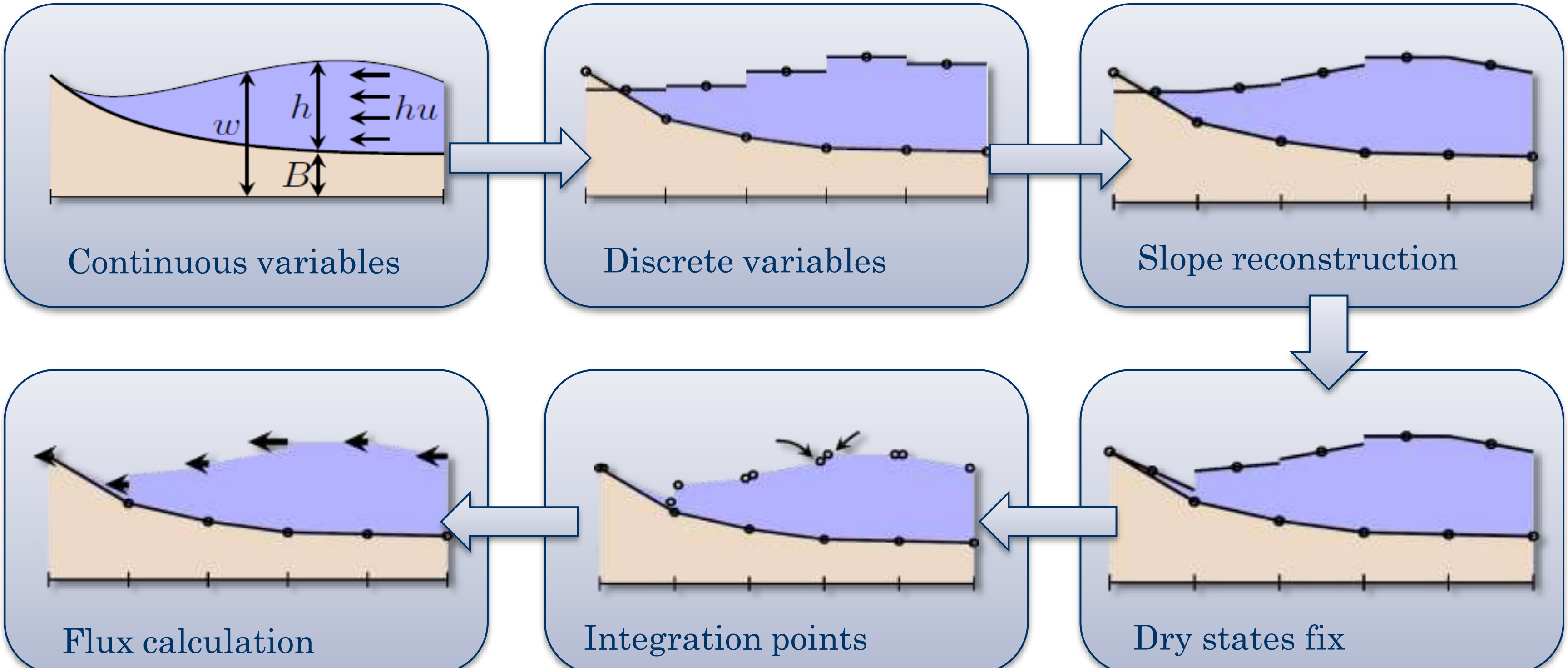
- Three modern high-resolution schemes:
  - Kurganov-Levy 2002
  - Modivied Kurganov-Levy 2005 (Hagen, Hjelmervik, Natvig, Lie)
  - Kurganov-Petrova 2007

# Shallow Water Simulation on GPUs



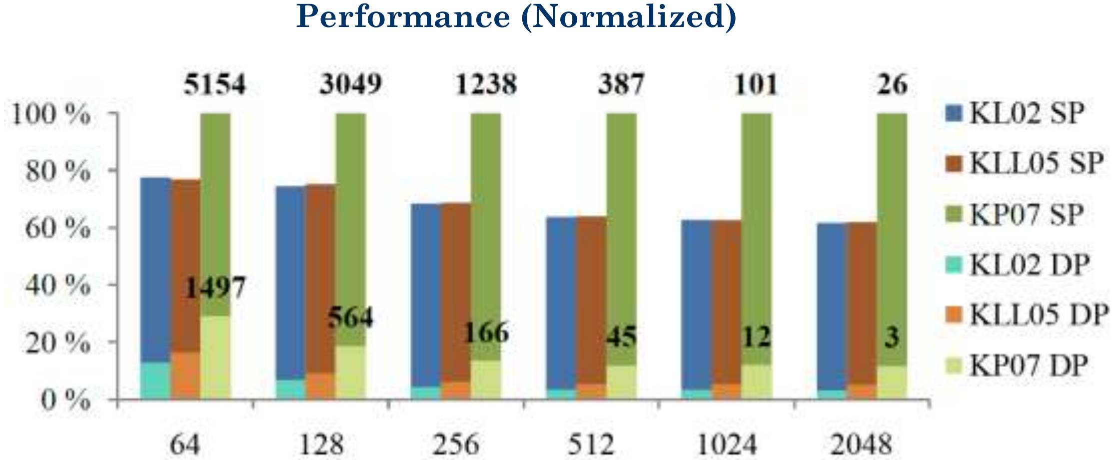

- To propagate the simulation a timestep (dt), execute the above operations
- Each block corresponds to one CUDA kernel

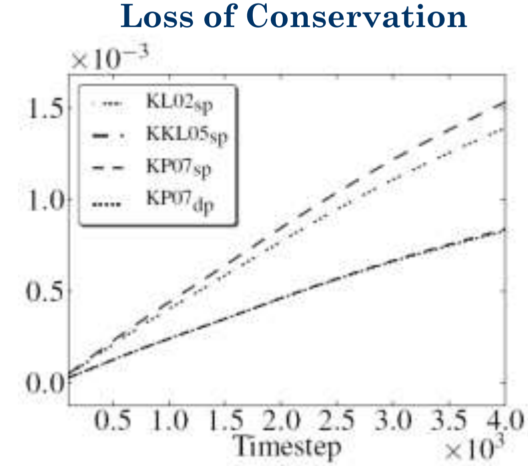
# Shallow Water Simulation on GPUs




- Flux calculation reads 11 of floating point values, and performs over 300 floating point operations for the *least* computationally demanding scheme

# Shallow Water Simulation on GPUs




- Single precision is sufficiently accurate for cases with wetting/drying
- Double precision takes eight times as long as single precision (expected)
- Kurganov-Petrova the best scheme wrt. both performance *and* accuracy
  (But modified Kurganov-Levy can support higher-order reconstruction)
- CUDA Profiler indicates good resource use (80% instruction throughput for flux)

# Shallow Water Simulation on GPUs

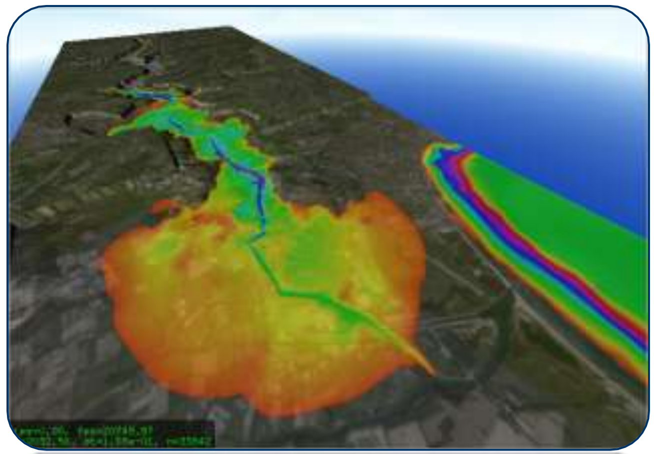
Focus on Kurganov-Petrova and Simulate real-world phenomena

**Tsunamis**

2004 Indian Ocean (230000)

**Storm Surges**

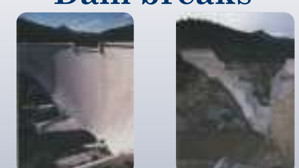
2005 Hurricane Katrina (1836)



**Floods**

2010: Pakistan (2000+)

**Dam breaks**



1959 Malpasset (423)

Hydrologists are not satisfied with speed alone

Simulation of real-world cases require thorough verification and validation

# Shallow Water Simulation on GPUs

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghB_x \\ -ghB_y \end{bmatrix} + \boxed{\begin{bmatrix} 0 \\ -gu\sqrt{u^2+v^2}/C_z^2 \\ -gv\sqrt{u^2+v^2}/C_z^2 \end{bmatrix}}$$

Vector of conserved variables

Flux functions

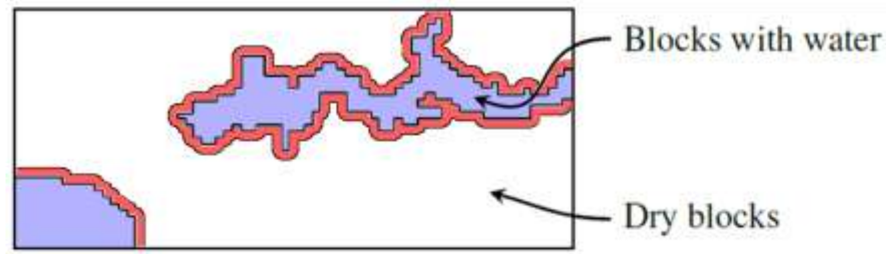
Bed slope source term

Bed friction source term

- Add friction source term
- Semi-implicit discretization and add to time integration kernel

# Shallow Water Simulation on GPUs

- Early exit optimization:
  Do not perform calculation on dry blocks
  - Up-to 6x speedup
  - Extra reads to global memory
  - One wet cell is sufficient




- New kernel layout
  - 31% decrease in memory footprint
  - Faster time integration, but slower flux calculation
  - Fermi optimizations (cache, launch bounds, register use, etc.)

# Shallow Water Simulation on GPUs

- Barrage de Malpasset near Fréjus
  - 66.5 m high
  - 220 m crest length
  - 55 million cubic metres of water
  - Bursts at 21:13 December 2nd 1959
  - 40 meter high wall of water, 70 km/h
  - 423 casualties, $68 million in damages
- Experimental data from 1:400 model
  - 1100 x 440 bathymetry values
  - 482 000 cells
  - 15 meter resolution
- Implementation accurately predicts maximum water elevation and front arrival time
  - Discrepancy at gauges
    14 (arrival time) and 9 (elevation)
  - Compares well with published results

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

# Summary

- Heterogeneous Architectures and their properties
  - One review article
  - One comparative article
- Linear Algebra and coupling with Matlab
  - Initial OpenGL implementation
  - Scheduling version enabling multiple backends
- Stencil Computations and the Shallow Water Equations
  - Intial investigation of three schemes
  - From prototype to verified and validated

# Bibliography


- **State-of-the-Art in Heterogeneous Computing**, A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik and O. O. Storaasli. In Scientific Programming, IOS Press, 18(1) (2010), pp. 1-33

- **A Comparison of three Commodity-Level Parallel Architectures: Multi-core CPU, Cell BE and GPU**, A. R. Brodtkorb and T. R. Hagen. In proceedings of the Seventh International Conference on Mathematical Methods for Curves and Surfaces, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, 5862 (2010), pp. 70–80

- **A MATLAB Interface to the GPU**, A. R. Brodtkorb. In proceedings of The Second International Conference on Complex, Intelligent and Software Intensive Systems, IEEE Computer Society, (2008), pp. 822–827

- **An Asynchronous API for Numerical Linear Algebra**, A. R. Brodtkorb. In Scalable Computing: Practice and Experience, West University of Timisoara, 9(3) (Special Issue on Recent Developments in Multi-Core Computing Systems) (2008), pp. 153–163.

- **Simulation and Visualization of the Saint-Venant System using GPUs**, A. R. Brodtkorb, T. R. Hagen, K.-A. Lie, and J. R. Natvig. In Computing and Visualization in Science, Springer-Verlag Berlin Heidelberg, (special issue on Hot Topics in Computational Engineering), (2010).

- **Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification and Validation**, A. R. Brodtkorb, M. L. Sætra, and M. Altinakar. In review, 2010