



UNIVERSITY OF  
CAMBRIDGE

# Turbomachinery CFD on many-core platforms – experiences and strategies

Graham Pullan

Whittle Laboratory, Department of Engineering, University of Cambridge

MUSAF Colloquium, CERFACS, Toulouse

September 27-29 2010



**UNIVERSITY OF  
CAMBRIDGE**

# **Turbomachinery CFD on many-core platforms – experiences and strategies**

**Graham Pullan and Tobias Brandvik**

**Whittle Laboratory, Department of Engineering, University of Cambridge**

**MUSAF Colloquium, CERFACS, Toulouse**

**September 27-29 2010**

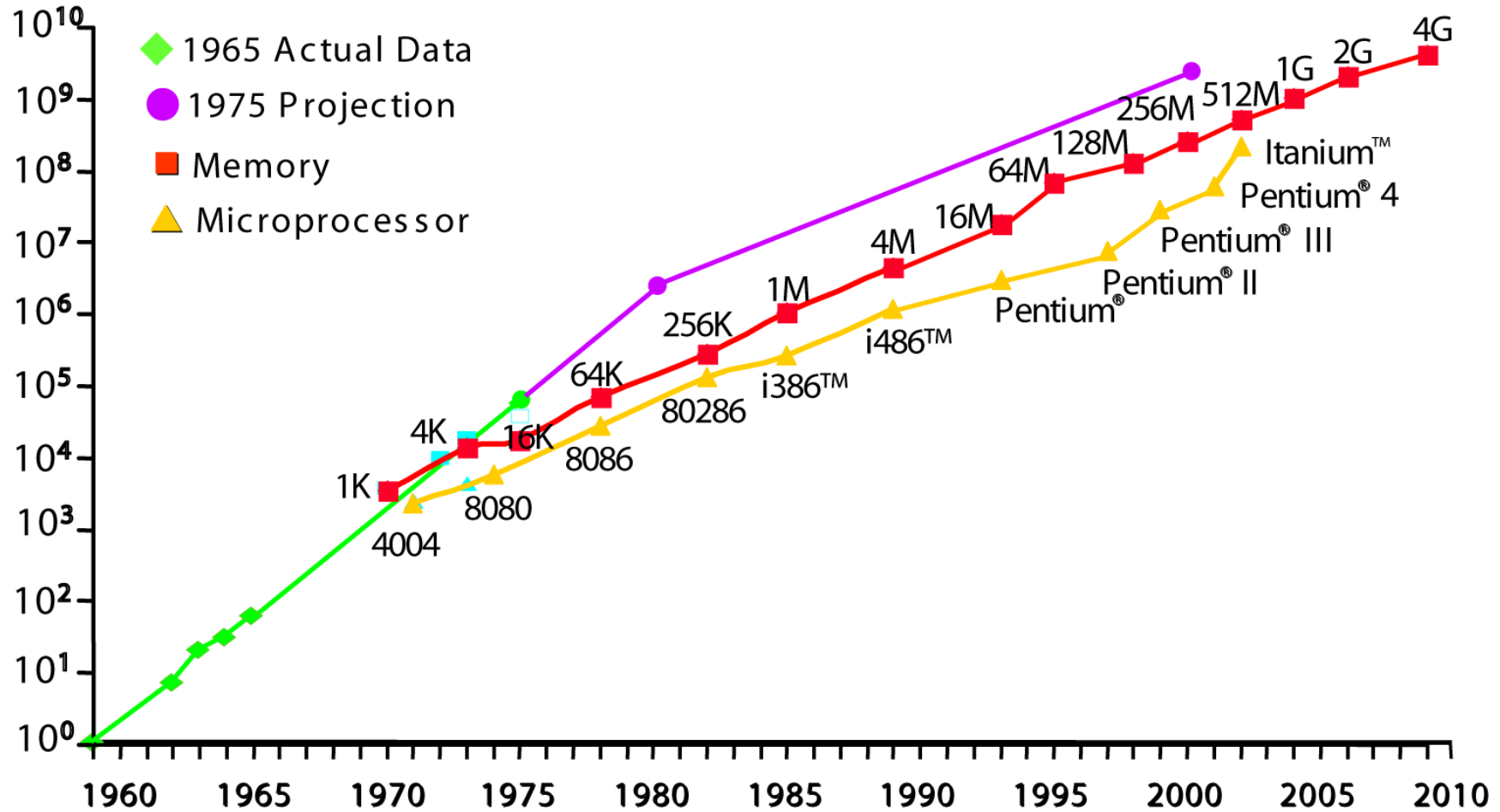
# Overview

- The future (the present) is “many-core”
- GPU hardware
- A strategy for programming GPUs for CFD
- Turbostream – a many-core CFD solver for Turbomachinery
- Example simulations
- Conclusions

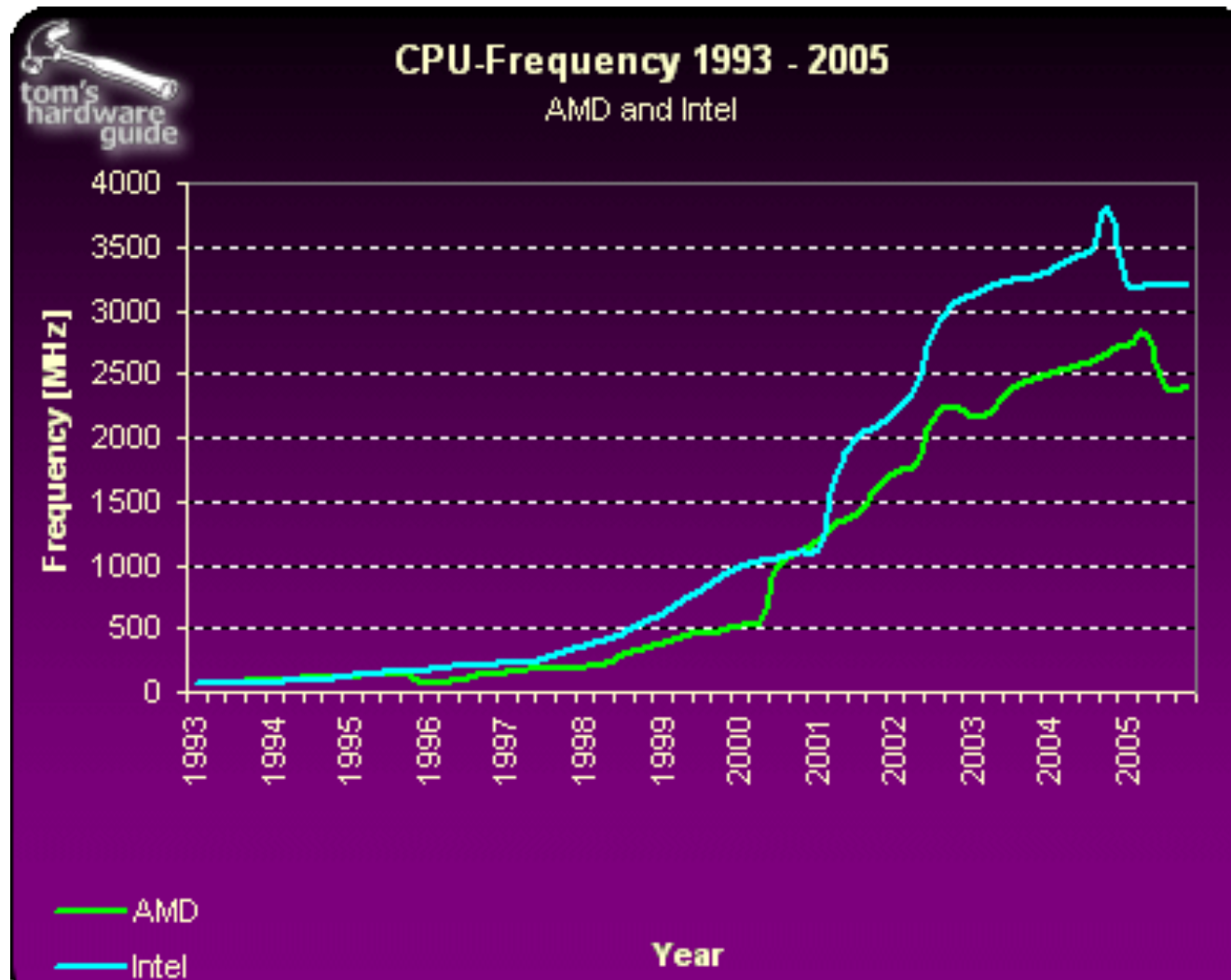
# Many-core

# Moore's "Law"

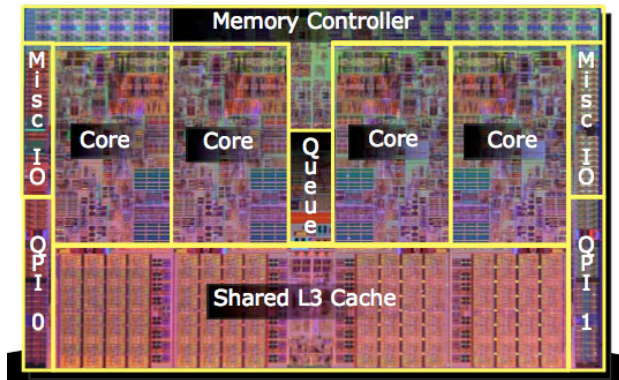
## Transistors



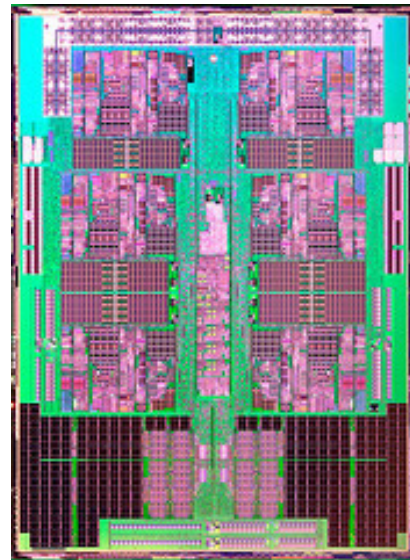
# Clock speed – the “power wall”



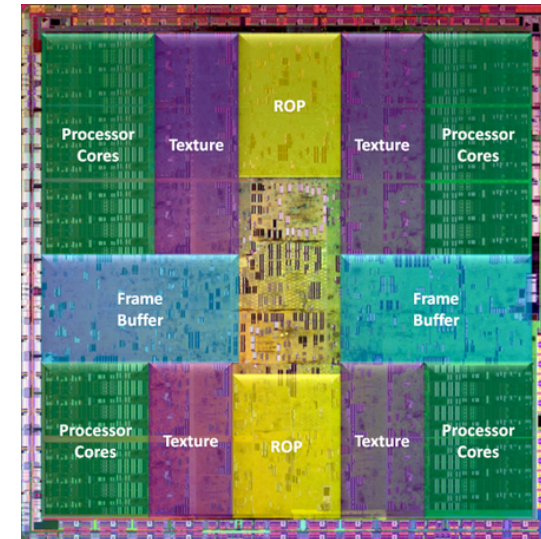
# Today's many-core chips



Intel 4 core CPU



AMD 6 core CPU



NVIDIA 240 cores GPU

# NVIDIA Tesla GPUs

- C2050 card for desktop PC;  
448 cores, 3GB;  
peak 515 GFLOP/s (DP)
- S2050 1U rack mount;  
4 x C2050 cards;  
requires connection to CPU  
host – typical average density  
is 2 GPUs per 1U





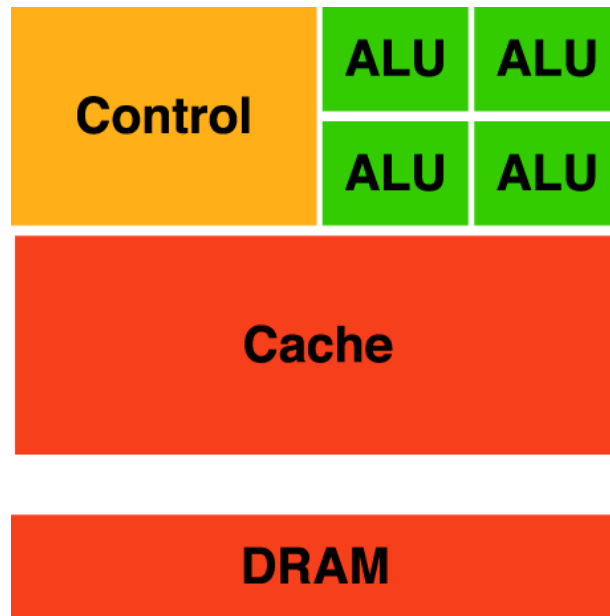
# Many-core summary

- Many-core computing is here today – and the trend is for greater and greater core count in future
- The run-time of a scalar (non-parallel) code is no longer guaranteed to reduce with each new computer (it may even increase)
- Codes need to be parallel to take advantage of new hardware
- The current distributed parallel computing model (MPI) will not work well here (for many-core CPUs), and may not work at all (for GPUs)

**GPUs – what do we get from all those cores?**

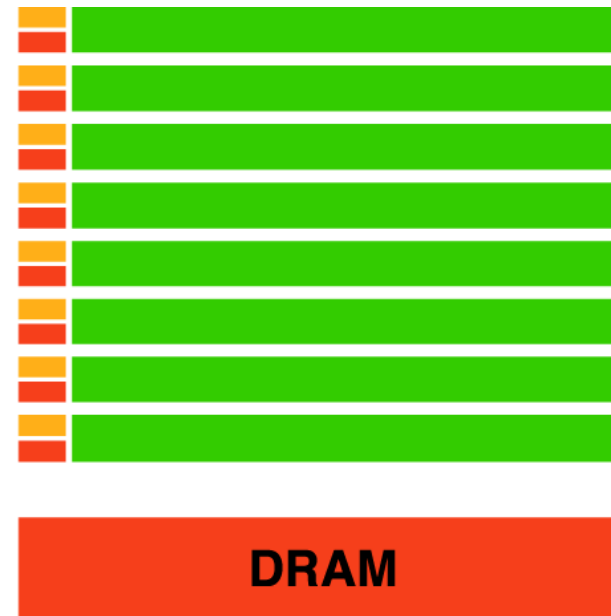
# How so many cores on a GPU?

Transistor usage:



**CPU**

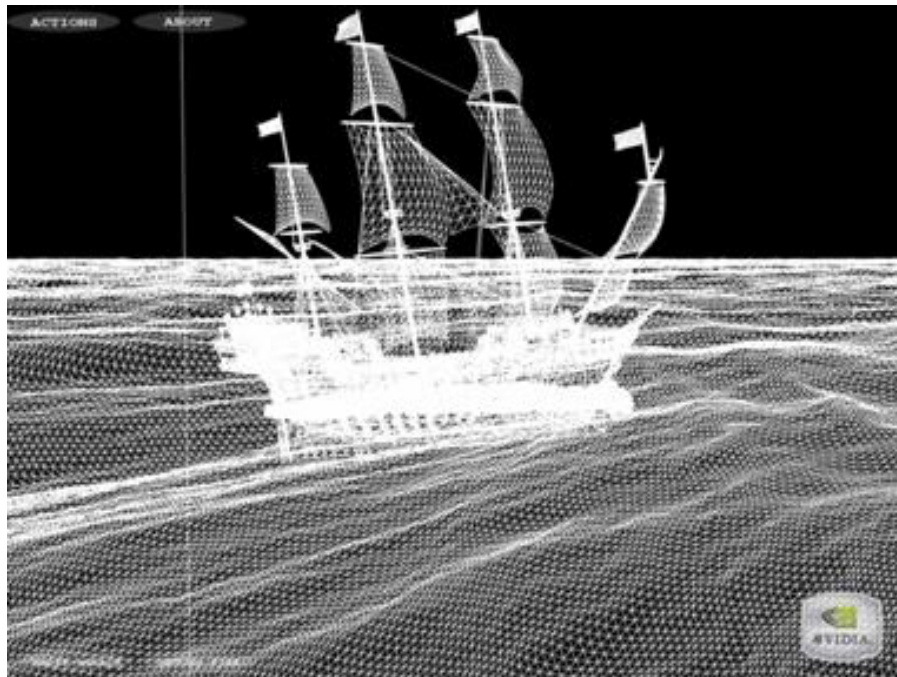
4 cores



**GPU**

>200 cores

# Why so many cores on a GPU?



# GPUs and scientific computing

GPUs are designed to apply the  
same *shading function*  
to many *pixels* simultaneously

# GPUs and scientific computing

GPUs are designed to apply the

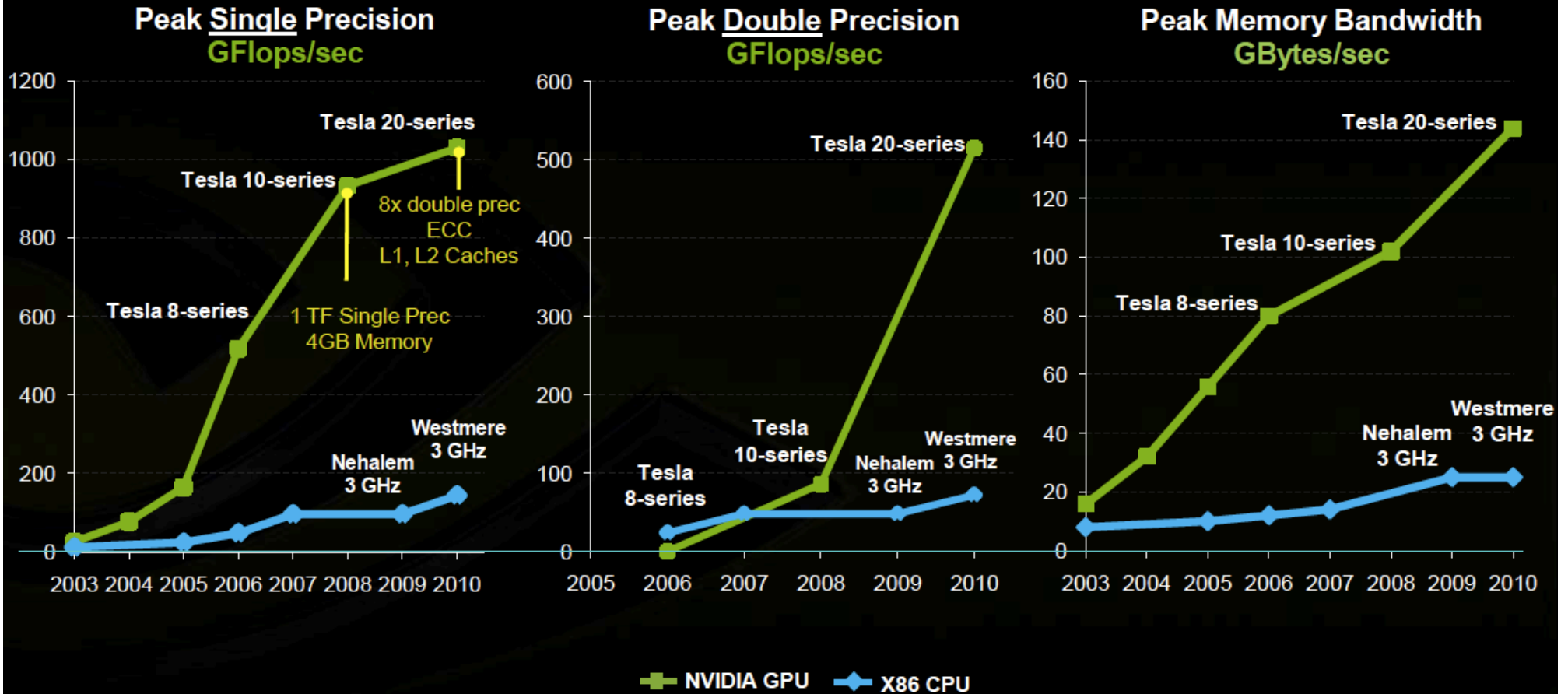
same *function*

to many *data* simultaneously

This is what CFD needs!

# GPU performance

## The Performance Gap



© NVIDIA Corporation 2010

# GPUs and CFD

Good speedups ( $> 10x$ ) can be expected for:

- Codes that apply the same function to lots of data
- Codes where amount communication to/from card is small (compared to the time to compute the data on the card – up to 4GB)



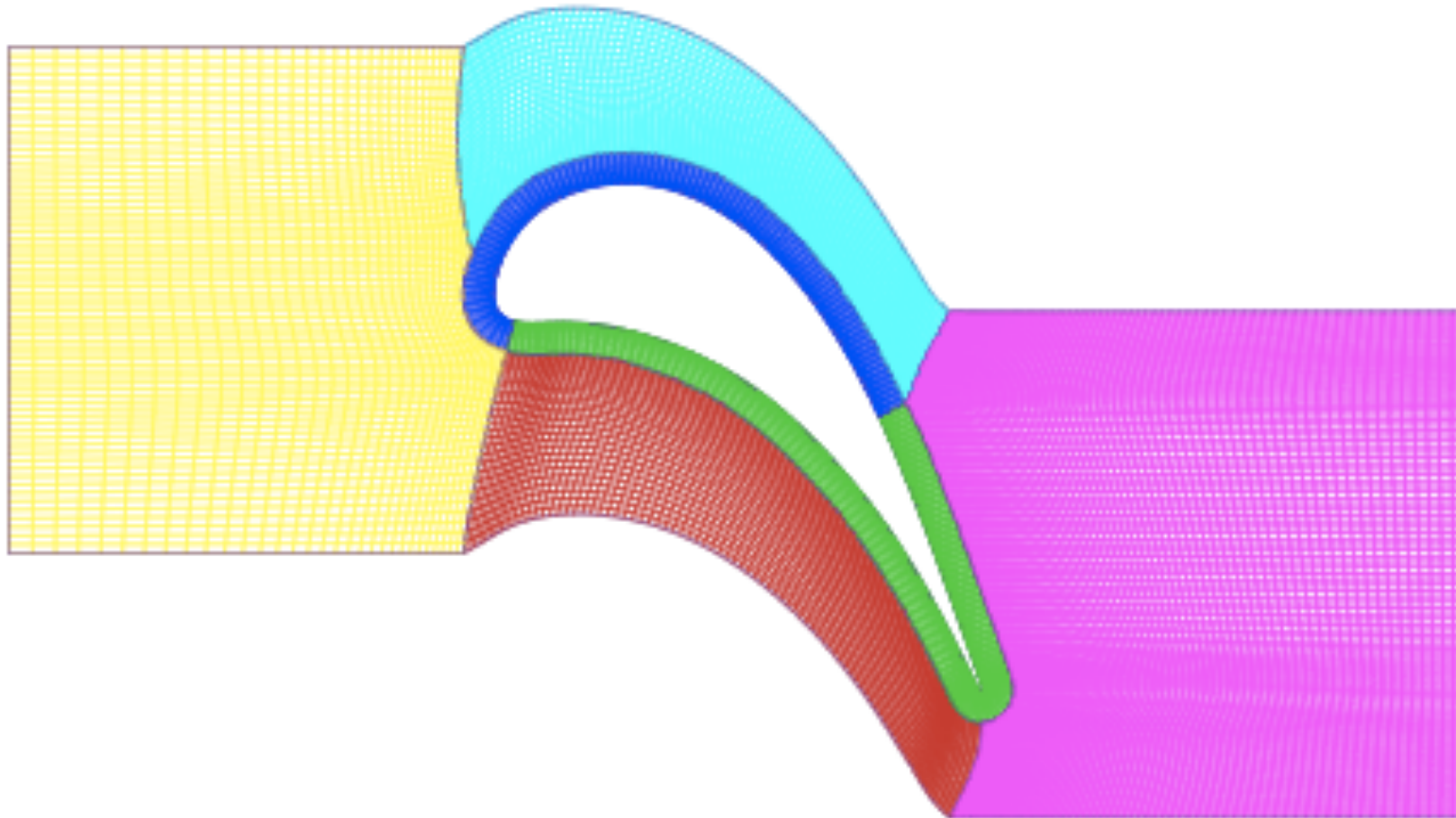
# GPUs and CFD

Good speedups ( $> 10x$ ) can be expected for:

- Codes that apply the same function to lots of data
- Codes where amount communication to/from card is small (compared to the time to compute the data on the card – up to 4GB)
- Turbomachinery CFD is an excellent fit!

# Programming GPUs for CFD

# Solving PDEs on multi-core processors



# Solving PDEs on multi-core processors

Navier-Stokes

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

Set flux, Sum flux, Shear Stress ...

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

Set flux, Sum flux, Shear Stress ...

Multigrid, Mixing plane, Sliding plane ...



# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

**Stencil  
operations**

Set flux, Sum flux, Shear Stress ...

Multigrid, Mixing plane, Sliding plane ...

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

**Stencil  
operations**

Set flux, Sum flux, Shear Stress ...

**Non-stencil  
operations**

Multigrid, Mixing plane, Sliding plane ...

# Solving PDEs on multi-core processors

Navier-Stokes

Finite volume, explicit time stepping

Second order central differences, Scree scheme

**Stencil  
operations**

Set flux, Sum flux, Shear Stress ...

**90% of  
run-time**

**Non-stencil  
operations**

Multigrid, Mixing plane, Sliding plane ...

**10% of  
run-time**

# Stencil operations on multi-core processors

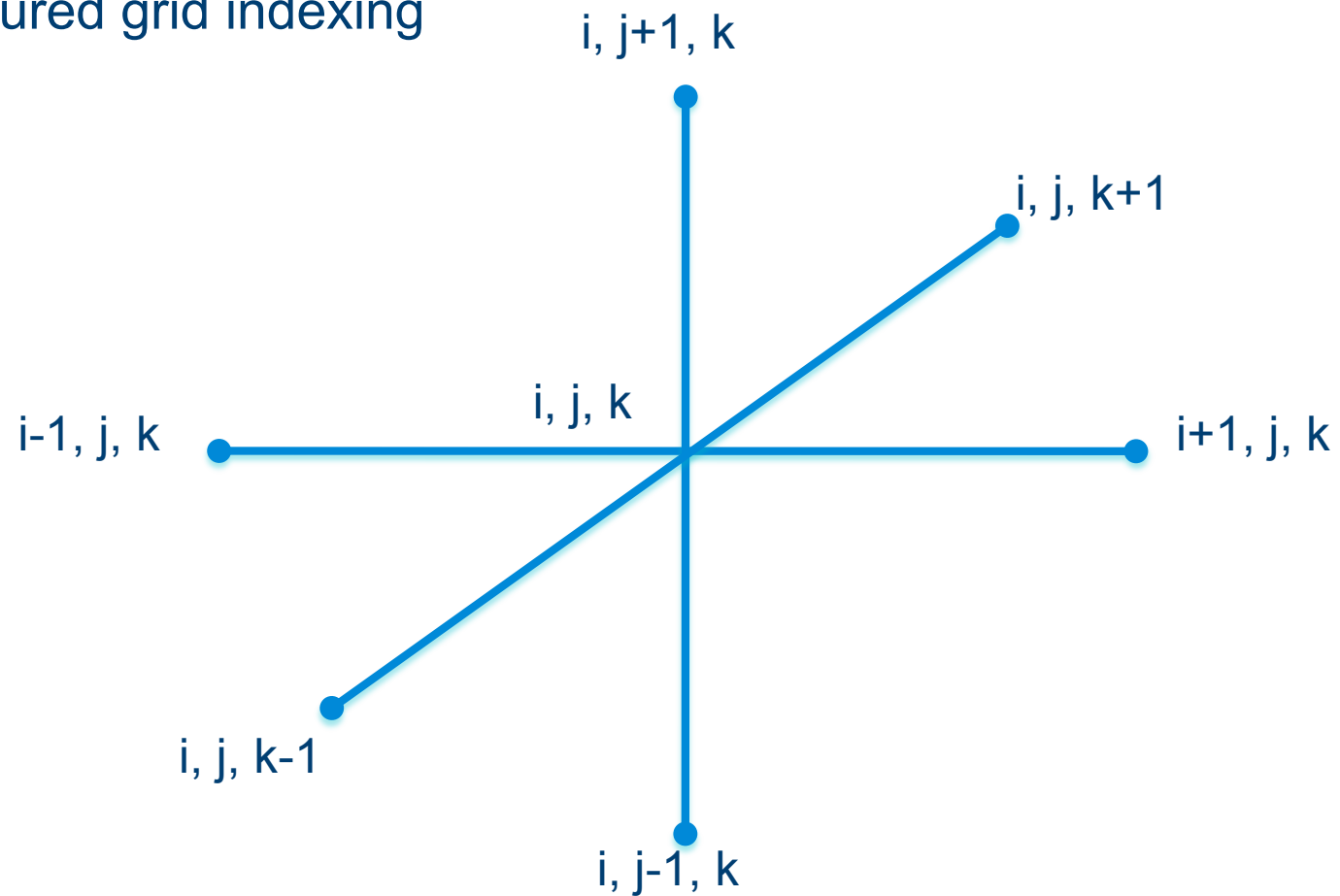
- Single implementation? e.g. OpenCL – but need to optimize for hardware
- Multiple implementations? e.g. C with SSE for CPU, CUDA for NVIDIA GPUs, ...
- Alternative:
  - High level language for stencil operations – a DSL
  - Source-to-source compilation

Brandvik, T. and Pullan, G. (2010)

“SBLOCK: A Framework for Efficient Stencil-Based PDE Solvers on Multi-Core Platforms”  
1<sup>st</sup> International Workshop on the Frontier of GPU Computing, Bradford, July 2010

# Stencil example

- Structured grid indexing



# Stencil example

- $\frac{\partial^2 u}{\partial x^2}$  in Fortran

```
DO K=2,NK-1
  DO J=2,NJ-1
    DO I=2,NI-1
      D2UDX2(I,J,K) = (U(I+1,J,K) - 2.0*U(I,J,K) +
        &          U(I-1,J,K))/(DX*DX)
    END DO
  END DO
END DO
```

# Stencil example

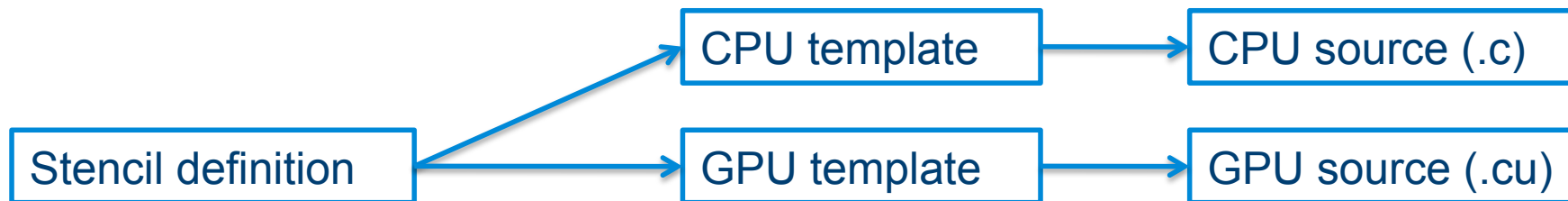
- Stencil definition:

```
input_scalars = ["dx"]
input_arrays = ["u"]
output_arrays = ["d2udx2"]
```

```
inner_calc = [
  {"lvalue": "d2udx2"
   "rvalue": ""u[1][0][0] - 2.0f*u[0][0][0] +
              u[-1][0][0])/(dx*dx)""
}
```

# Source-to-source compilation

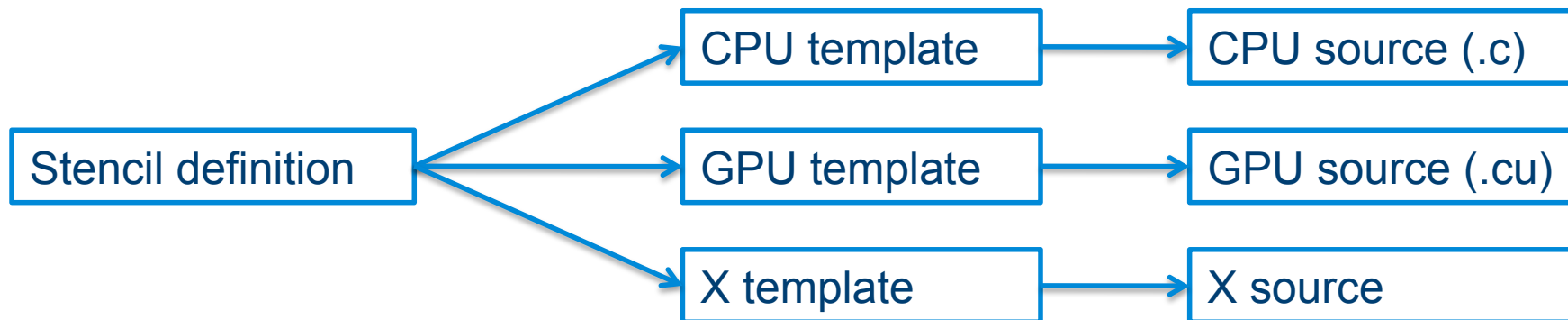
- The stencil definition is transformed at compile-time into code that can run on the chosen processor
- The transformation is performed by filling in a pre-defined template using the stencil definition





# Source-to-source compilation

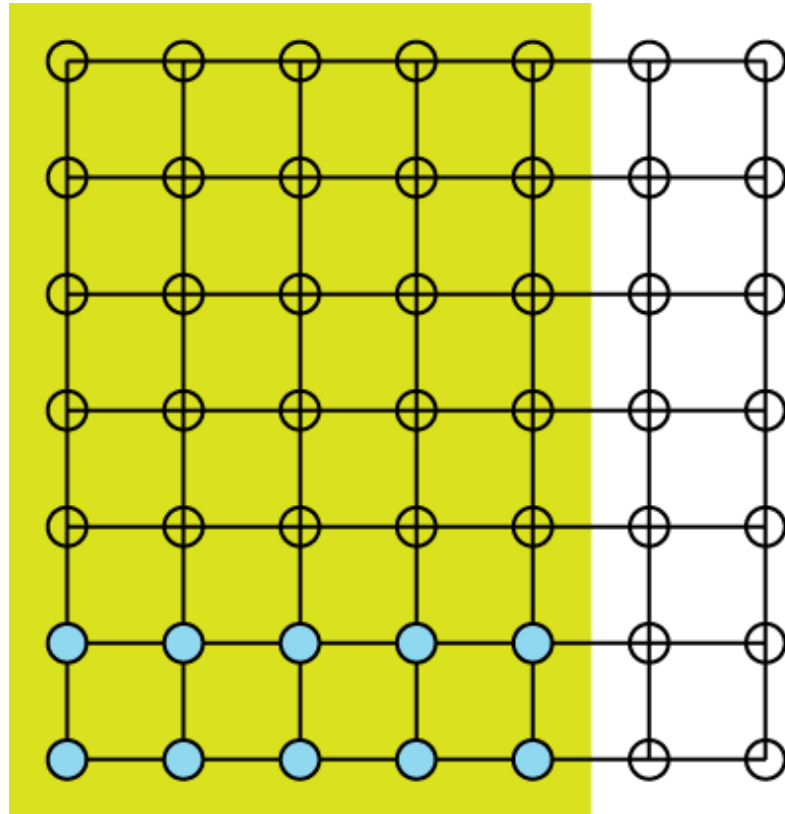
- The stencil definition is transformed at compile-time into code that can run on the chosen processor
- The transformation is performed by filling in a pre-defined template using the stencil definition



# Implementation details

- There are many optimisation strategies for stencil operations (see paper from Supercomputing 2008 by Datta et al.)
- CPUs:
  - Parallelise with pthreads
  - SSE vectorisation
- GPUs:
  - Cyclic queues

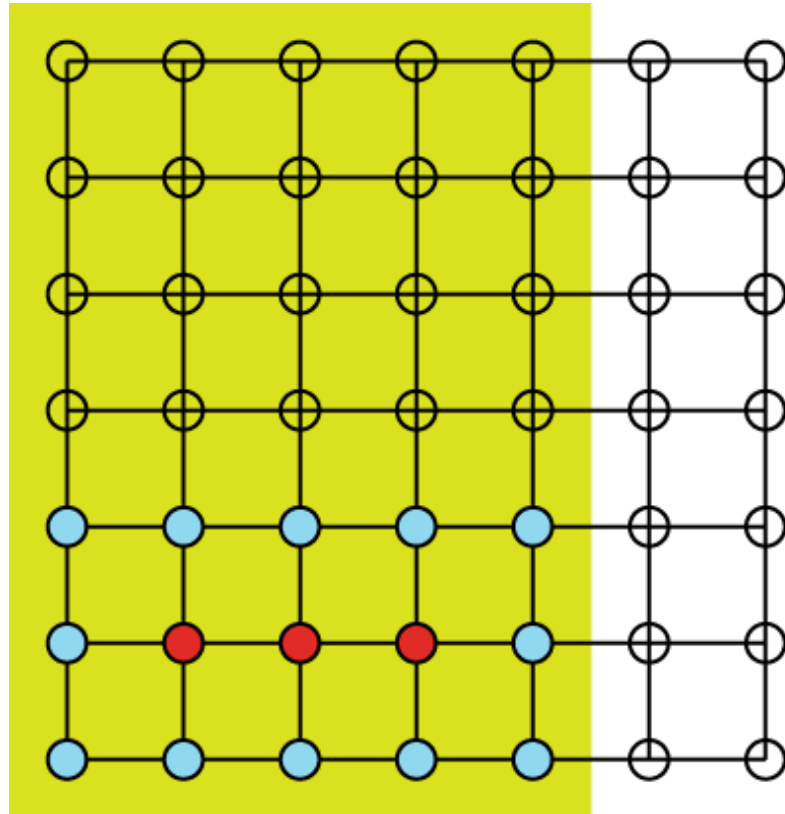
# CUDA strategy





Zone solved by Block 1

1. Load 2 rows into shared mem 

# CUDA strategy

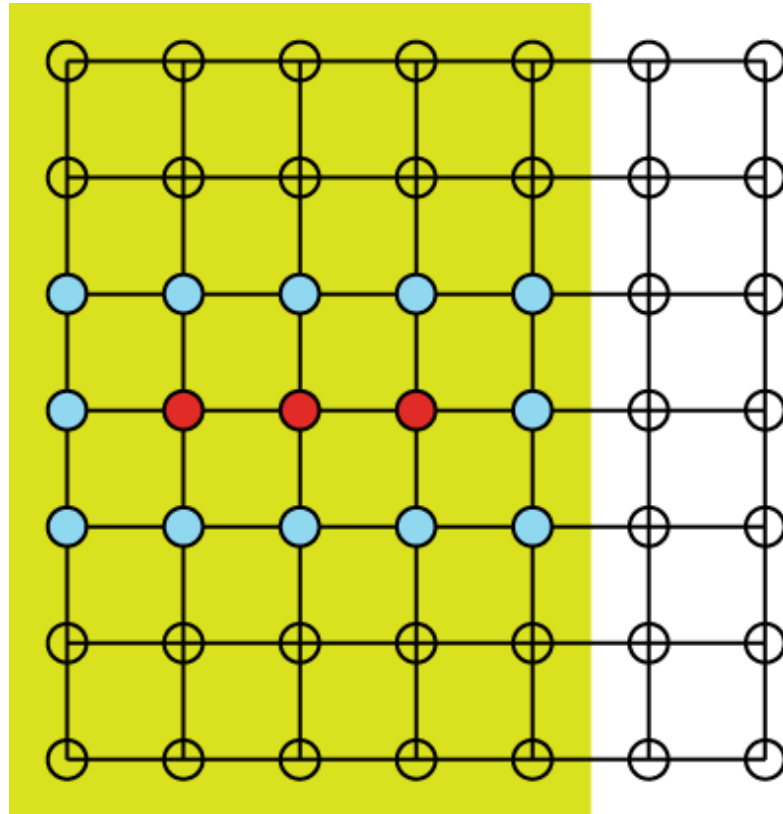


Zone solved by Block 1



1. Load 2 rows into shared mem 
2. To compute  points, load next row into shared mem



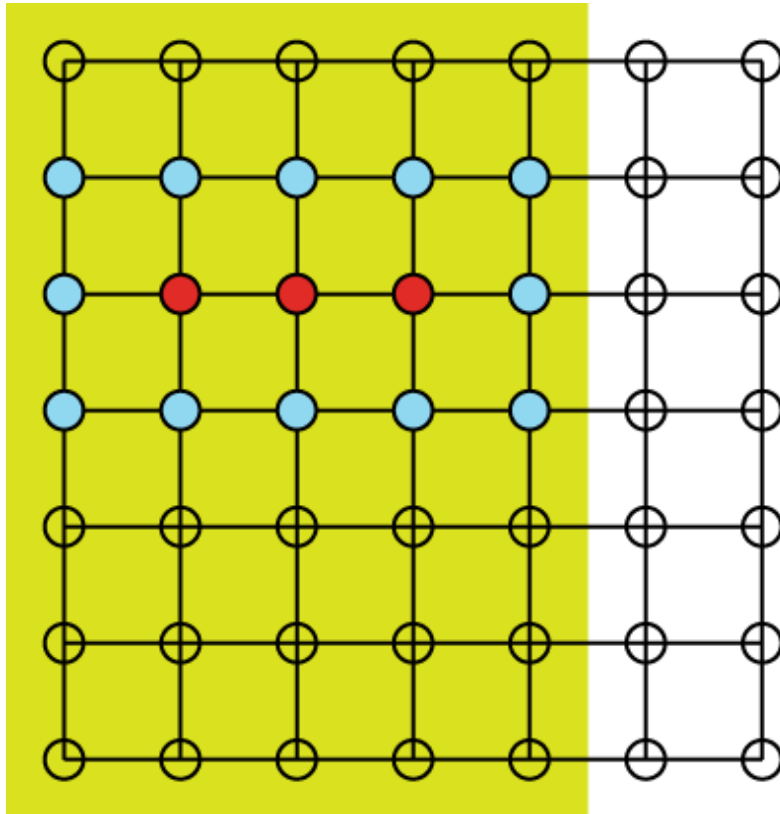
# CUDA strategy





Zone solved by Block 1

1. Load 2 rows into shared mem 
2. To compute  points, load next row into shared mem
3. Move up domain, row by row (load new row into shared mem, drop lowest row out of shared mem)

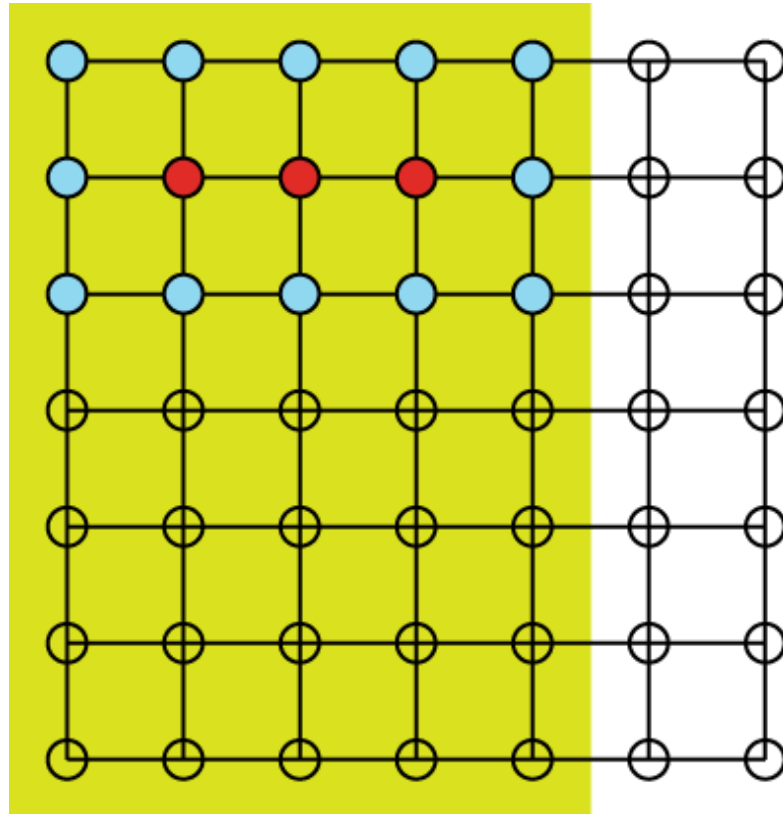
# CUDA strategy





Zone solved by Block 1

1. Load 2 rows into shared mem 
2. To compute  points, load next row into shared mem
3. Move up domain, row by row (load new row into shared mem, drop lowest row out of shared mem)

# CUDA strategy



Zone solved by Block 1

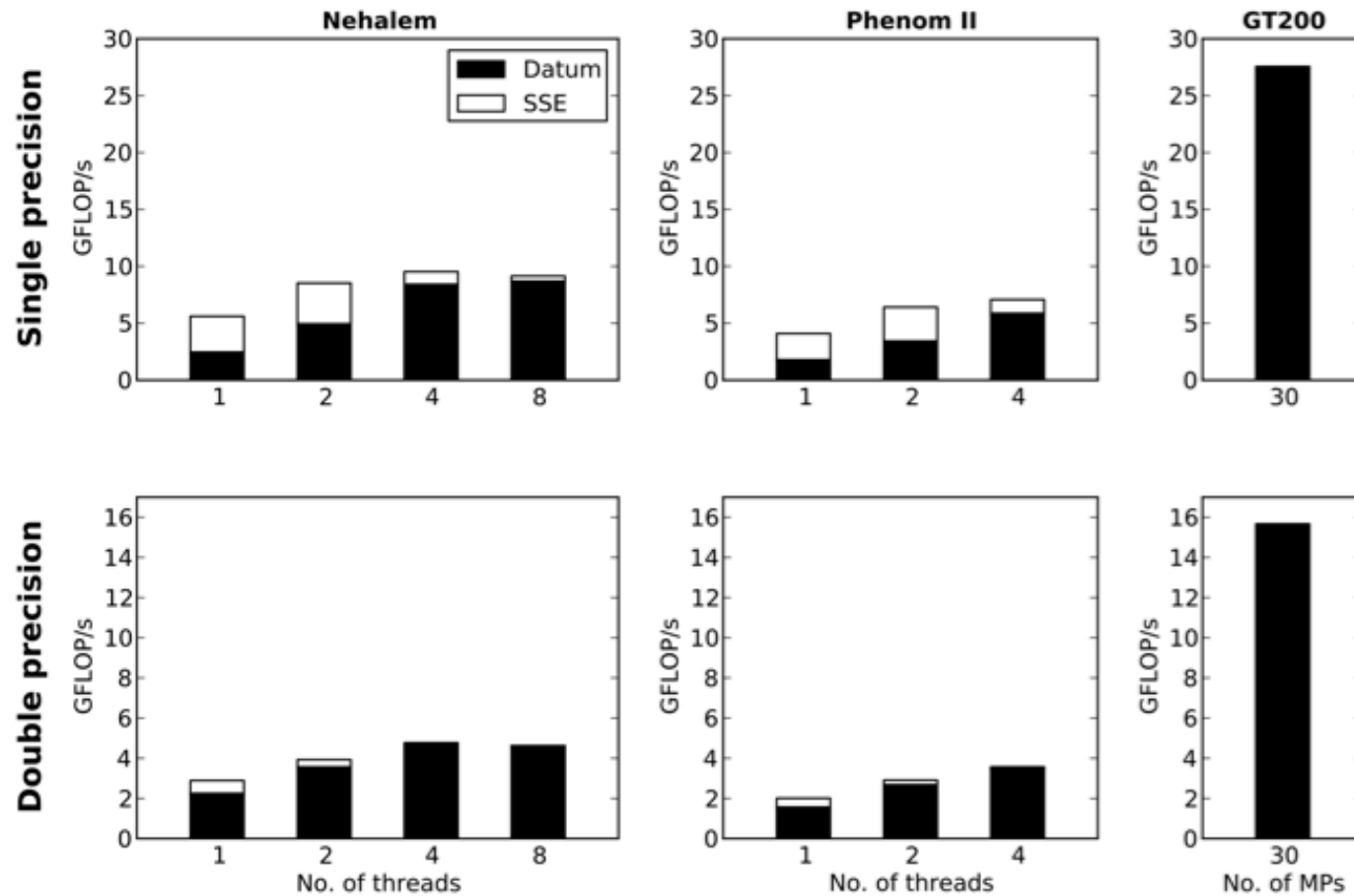
1. Load 2 rows into shared mem 
2. To compute  points, load next row into shared mem
3. Move up domain, row by row (load new row into shared mem, drop lowest row out of shared mem)

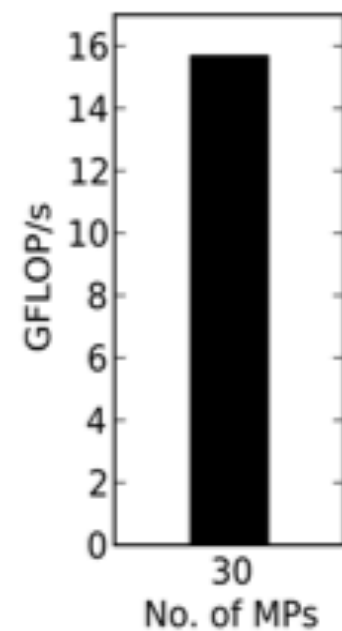
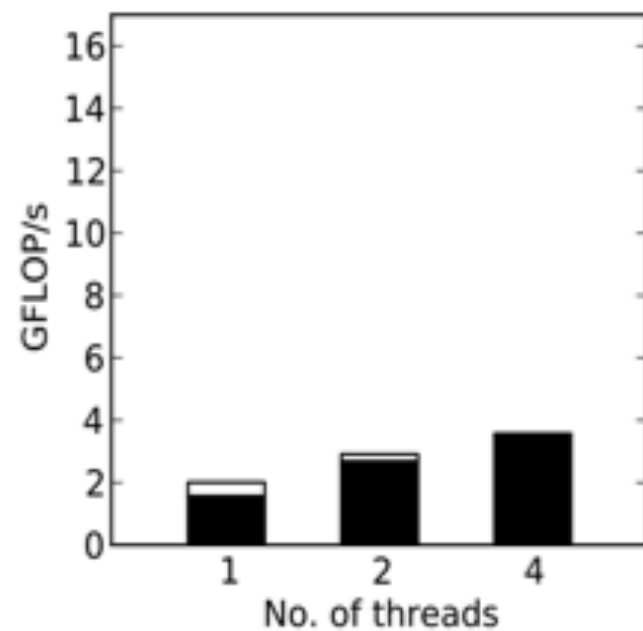
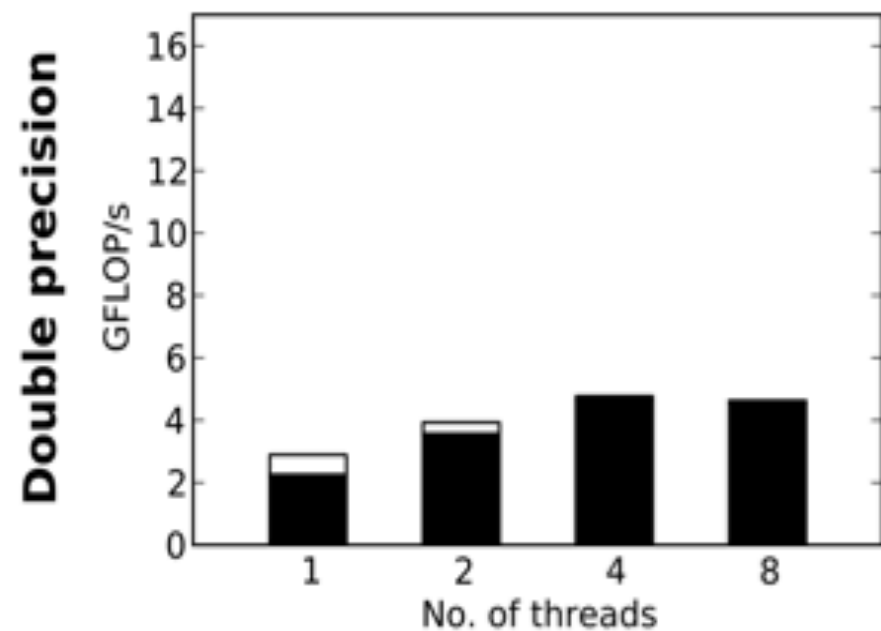
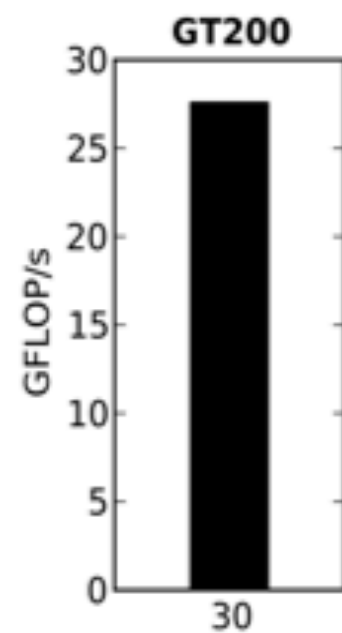
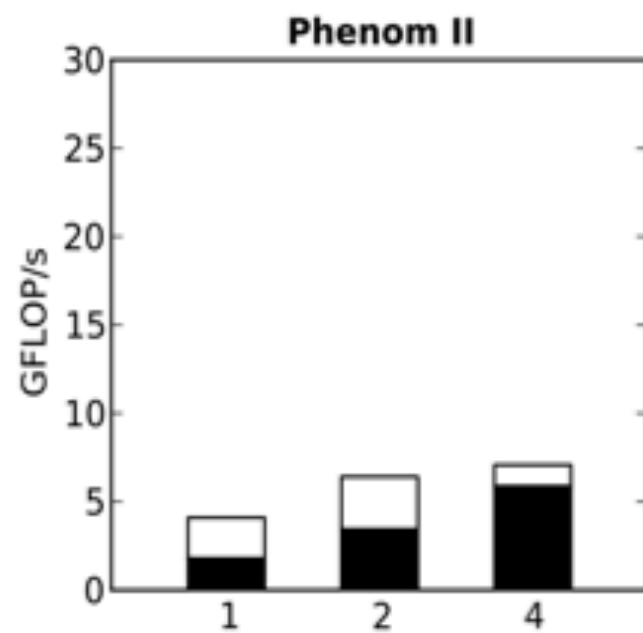
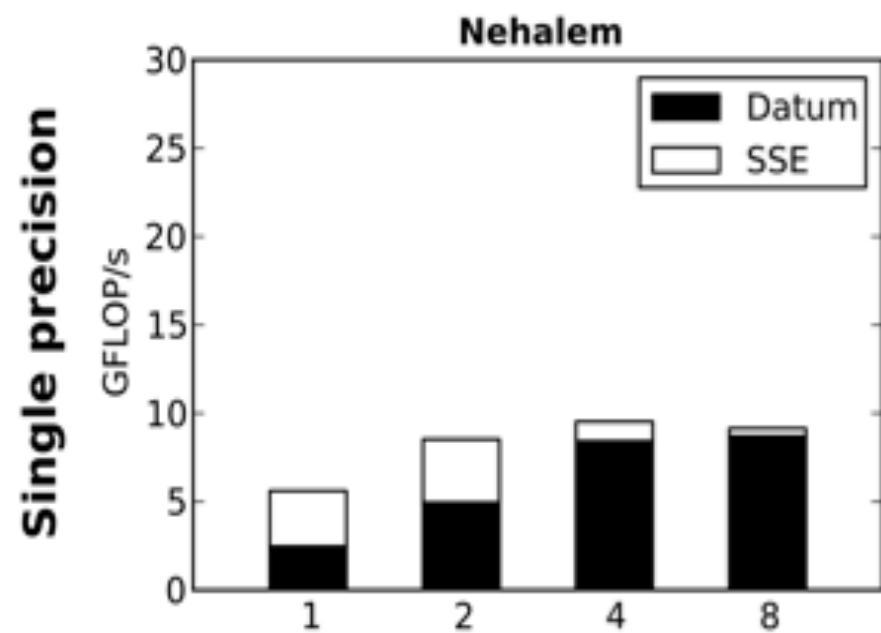


# Testbed

- CPU 1: Intel Core i7 920 (2.66 GHZ)
- CPU 2: AMD Phenom II X4 940 (3.0 GHz)
- GPU: NVIDIA GTX 280

# Stencil benchmark





# Turbostream

# Turbostream

- We have implemented a new solver that can run on both CPUs and GPUs
- The starting point was an existing solver (Fortran, MPI) called TBLOCK
- The new solver is called Turbostream

Brandvik, T. and Pullan, G. (2009)

*“An Accelerated 3D Navier-Stokes Solver for Flows in Turbomachines”*

ASME IGTI Turbo Expo, Orlando, June 2009

# TBLOCK

- Developed by John Denton
- Blocks with arbitrary patch interfaces
- Simple and fast algorithm
- 20,000 lines of Fortran 77
- Main solver routines are only 10,000 lines

# Turbostream

- 3000 lines of stencil definitions (~15 different stencil kernels)
- Code generated from stencil definitions is 15,000 lines
- Additional 5000 lines of C for boundary conditions, file I/O etc.
- Source code is very similar to TBLOCK – every subroutine has an equivalent stencil definition

# Turbostream performance

Processor	TBLOCK	Turbostream
Intel Nehalem	1.21	1.48
AMD Phenom II	1	0.89
NVIDIA GT200	-	10.2

- TBLOCK runs in parallel on all four CPU cores using MPI
- Initial results from latest NVIDIA Fermi GPU show an further 2x speedup



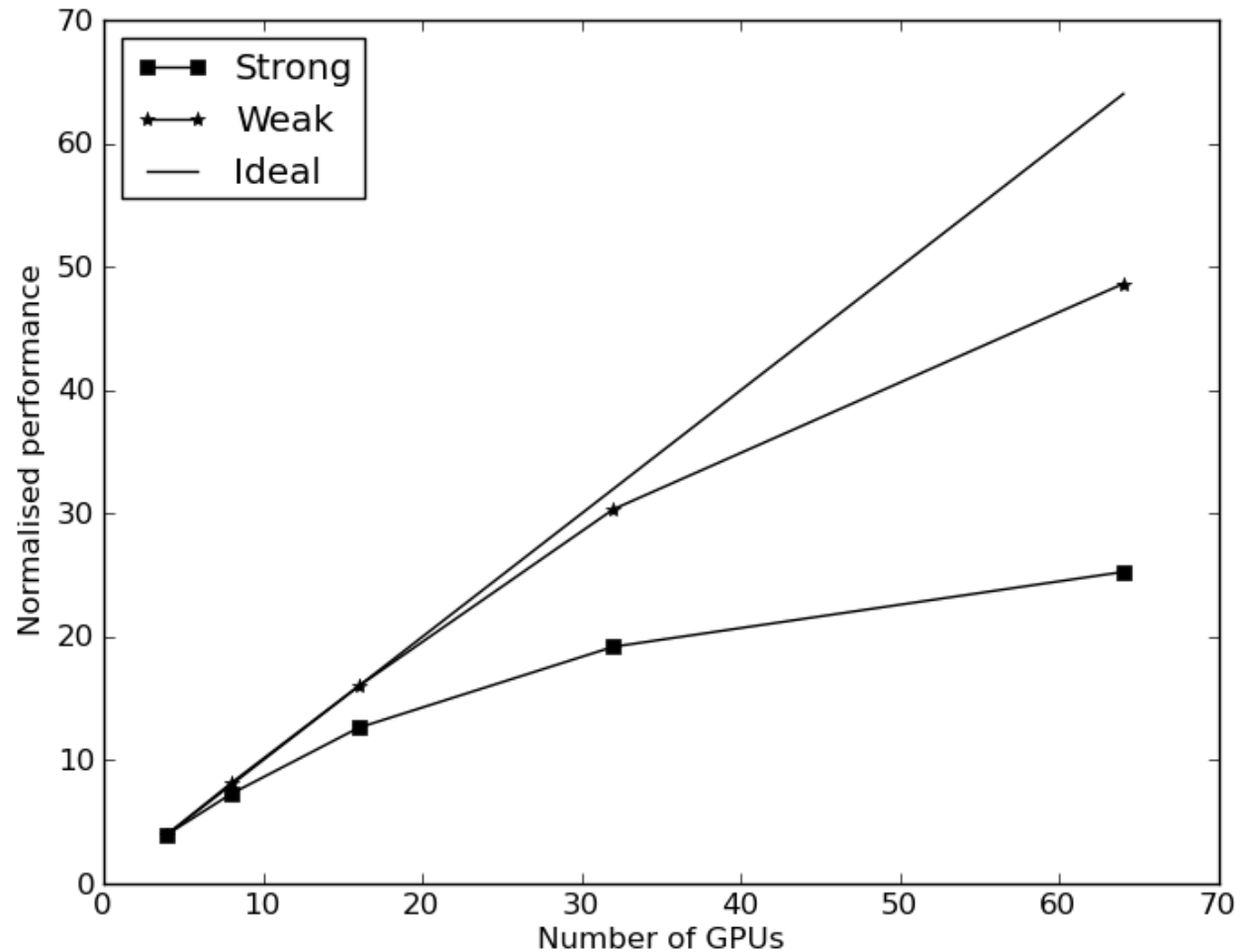
# “Darwin” GPU cluster

- Darwin is the central Cambridge cluster
- In 2008, NVIDIA made Cambridge a CUDA Centre of Excellence, donating 128 GPUs to Darwin

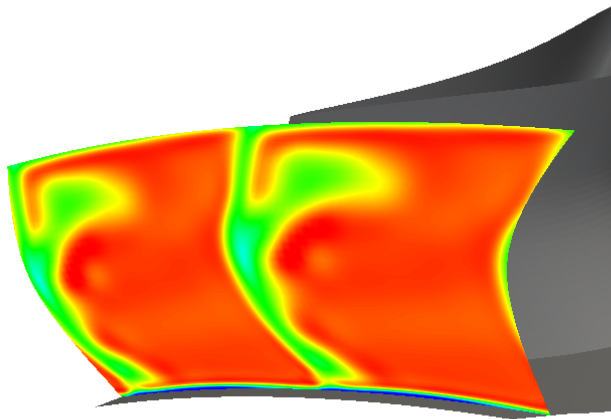


# Scaling results

- Unsteady multi-stage turbine
- Baseline: 27 E6 nodes
- Largest: 432 E6 nodes
- Weak scaling: Increase grid size with # GPUs
- Strong scaling: Constant grid size (27 E6 nodes)



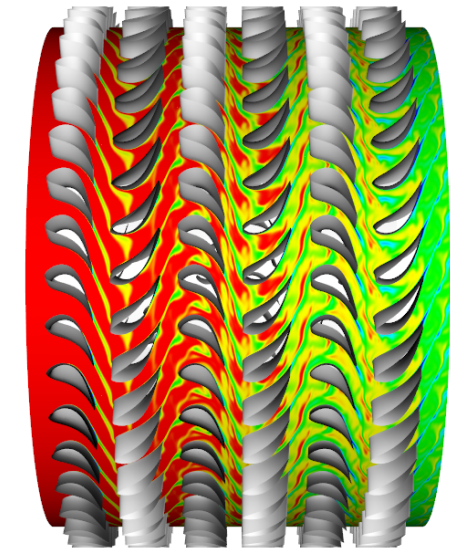
# What can this be used for?



Steady, single passage:  
0.5 million nodes  
40s (1 GPU)



Steady, 3 stage:  
3 million nodes  
7.5 mins (1 GPU)



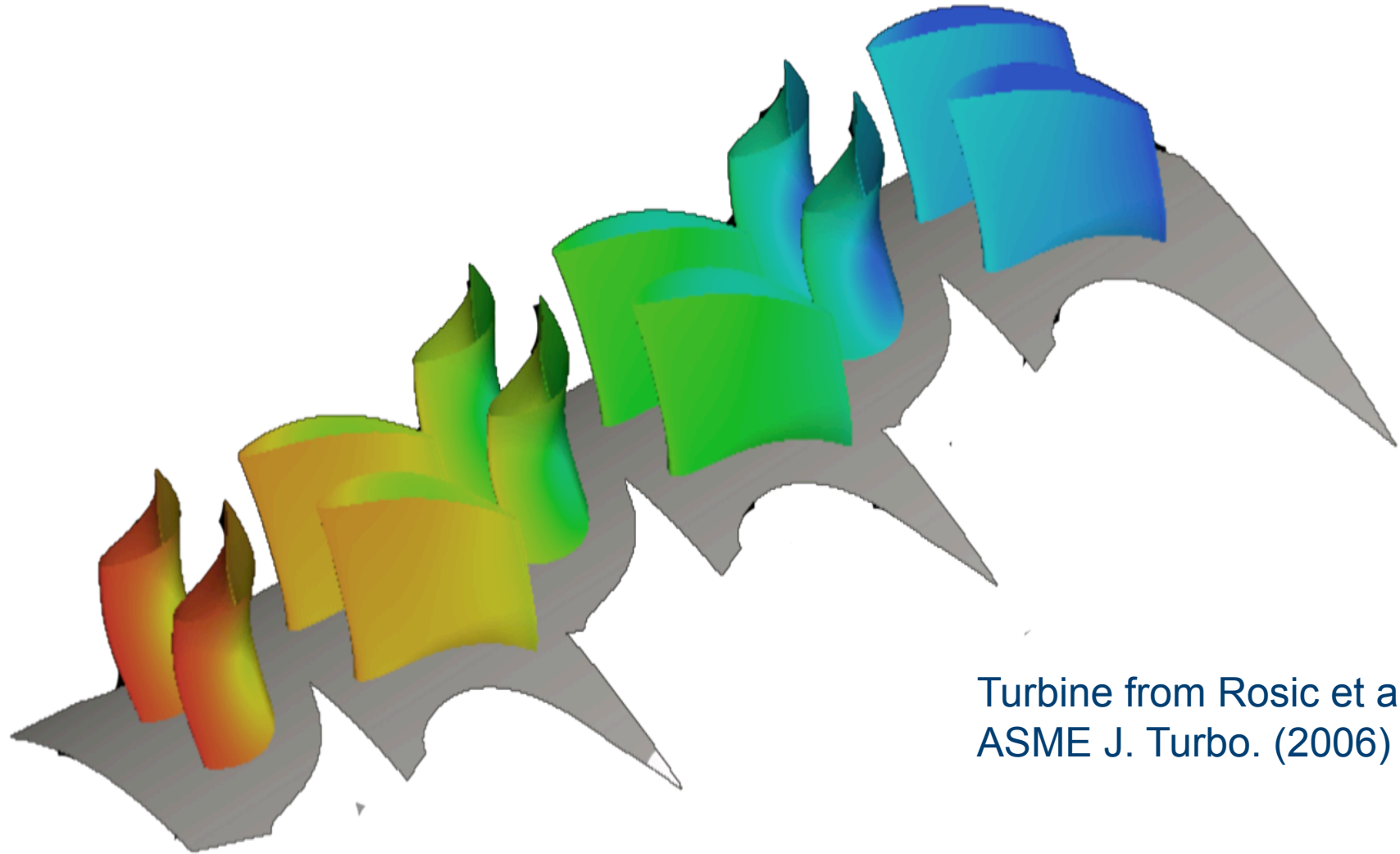
Unsteady, 3 stage:  
120 million nodes  
4 hours (16 GPUs) per rev

# Example simulations

# Example simulations

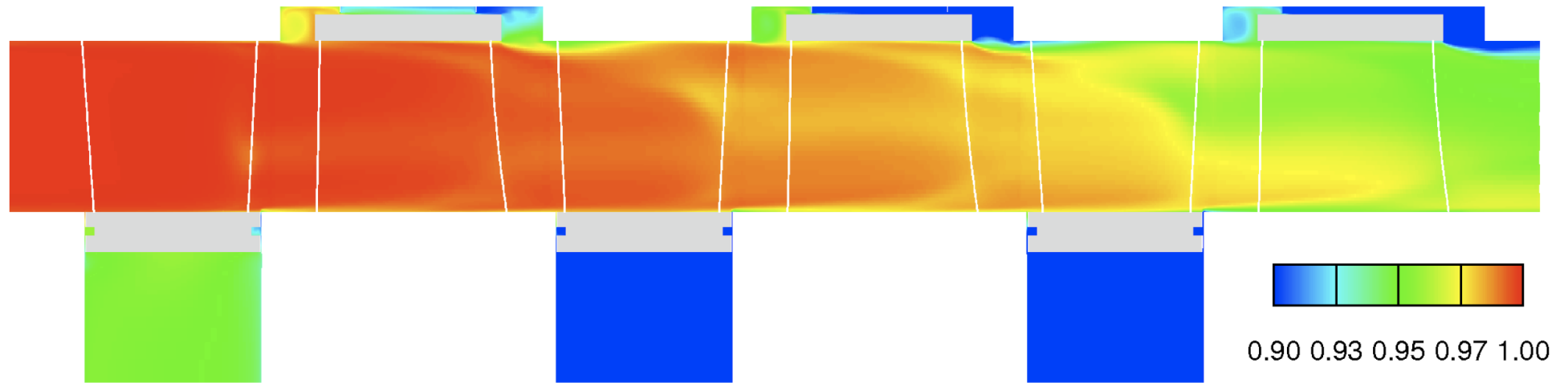
- Three-stage steady turbine simulation (mixing planes)
- Three-stage unsteady compressor simulation (sliding planes)

# Three-stage turbine



Turbine from Rosic et al,  
ASME J. Turbo. (2006)

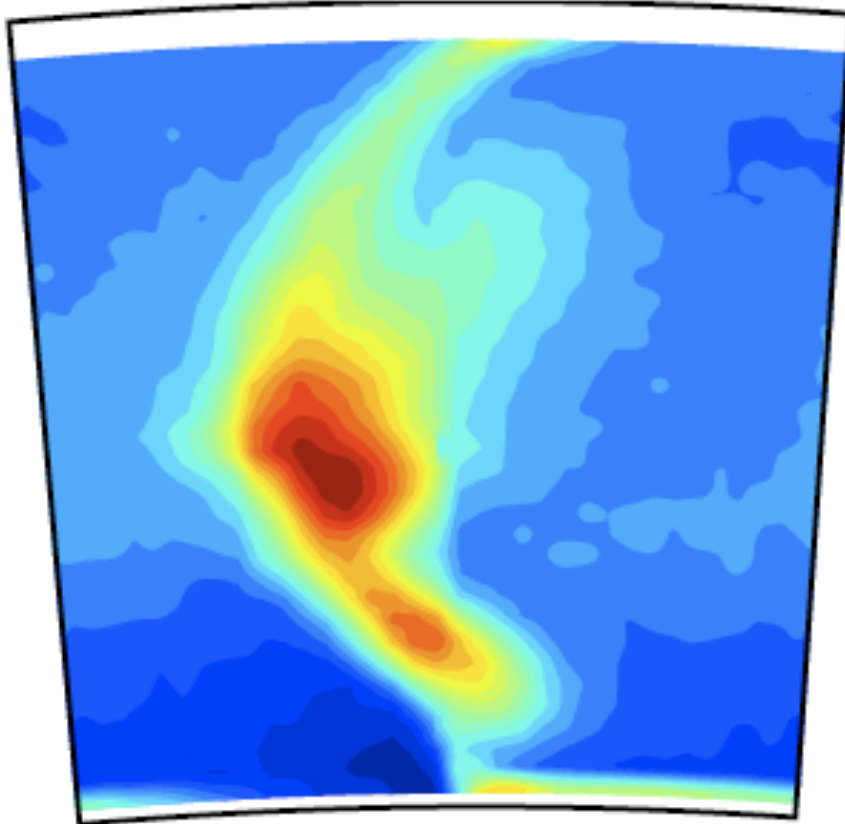
# Entropy function through machine



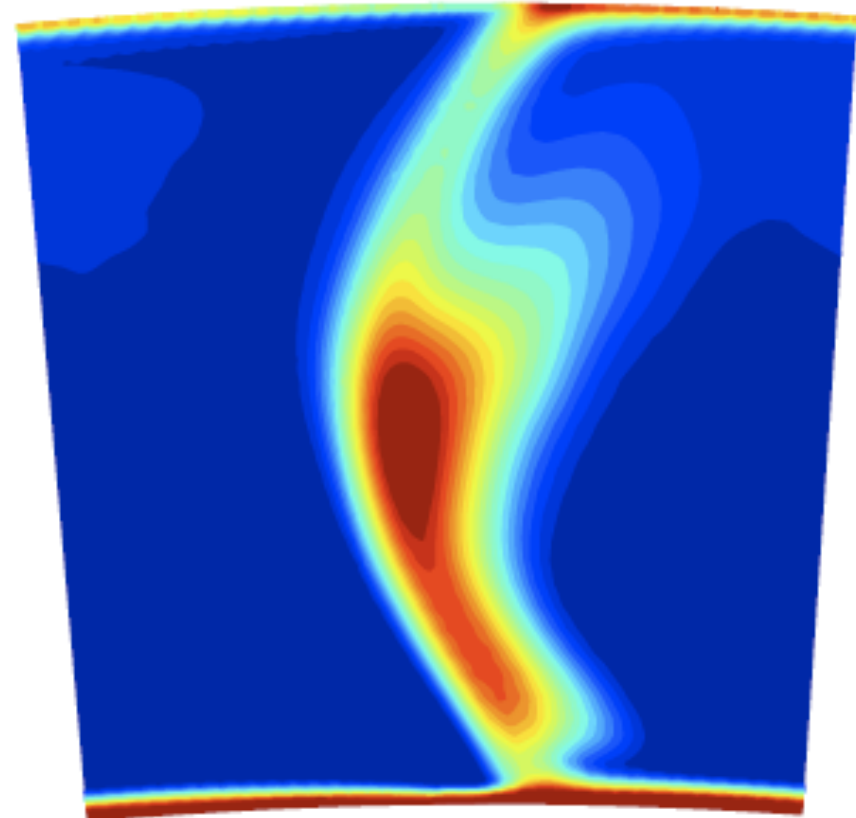
- 10 hours on a single CPU
- 8 minutes on four GPUs

# Total pressure loss, Stator 3 exit

Experiment



Turbostream

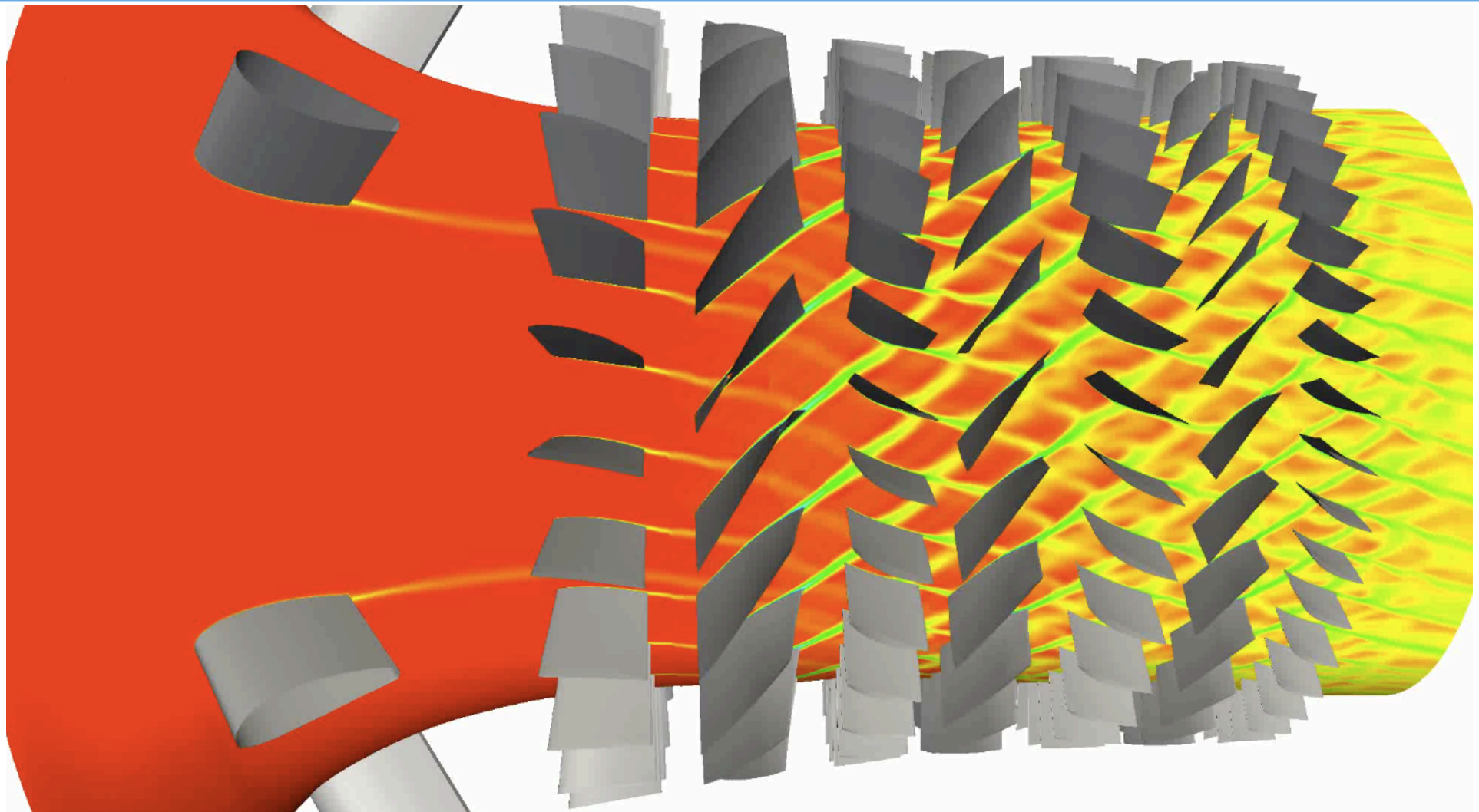




# Compressor simulation

- Three-stage compressor test rig at Siemens, UK
- 160 million grid nodes
- 5 revolutions needed to obtain a periodic solution (22500 time steps)
- On 32 NVIDIA GT200 GPUs, each revolution takes 24 hours

# Entropy function contours at mid-span



# Conclusions

# Conclusions

- The switch to multi-core processors enables a step change in performance, but existing codes have to be rewritten
- The differences between processors make it difficult to hand-code a solver that will run on all of them
- We suggest a high level abstraction coupled with source-to-source compilation

# Conclusions

- A new solver called Turbostream, which is based on Denton's TBLOCK, has been implemented
- Turbostream is ~10 times faster than TBLOCK when running on an NVIDIA GPU as compared to a quad-core Intel or AMD CPUs
- Single blade-row calculations almost interactive on a desktop (10 – 30 seconds)
- Multi-stage calculations in a few minutes on a small cluster (\$10,000)
- Full annulus unsteady calculations complete overnight on a modest cluster (\$100,000)