CFD Solvers on Many-core Processors

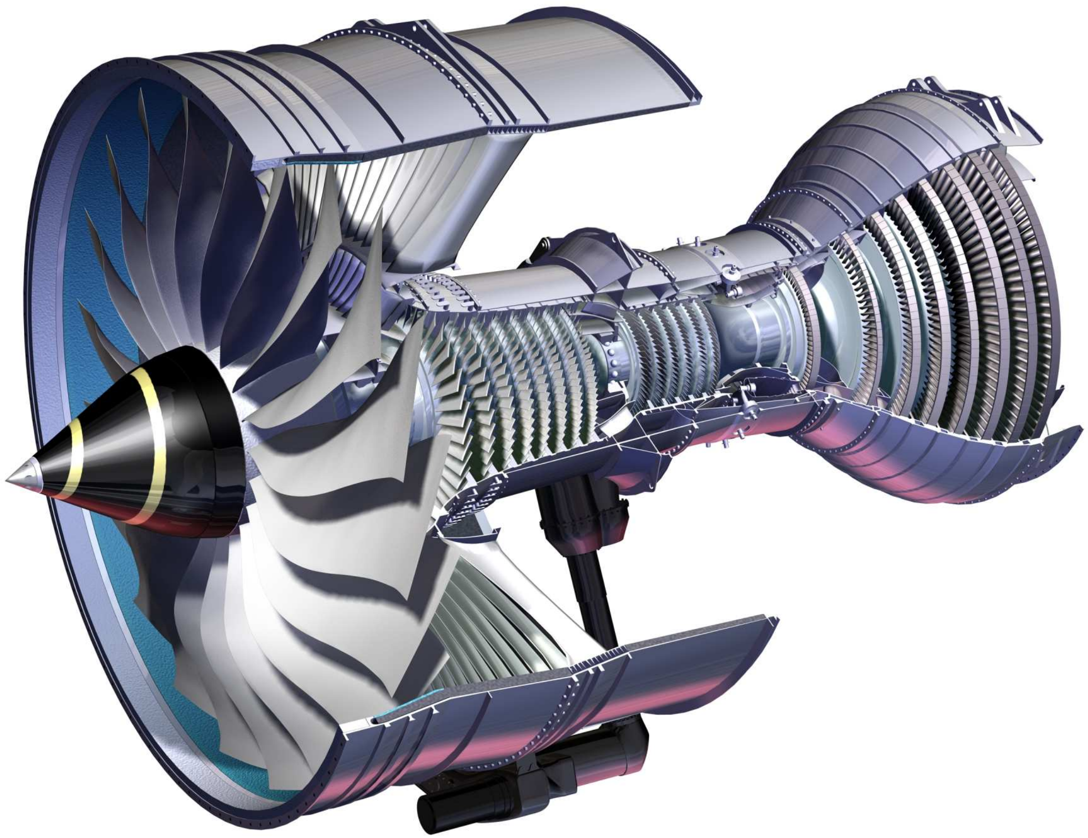
Tobias Brandvik

Whittle Laboratory

CFD Backgroud

- CFD: Computational Fluid Dynamics
- Whittle Laboratory - Turbomachinery

Turbomachinery



Compute requirements

- Steady models (no wake/blade interaction etc.)

1 blade	0.5 Mcells	1 CPU hour
1 stage (2 blades)	1.0 Mcells	3 CPU hours
1 component (5 stages)	5.0 Mcells	20 CPU hours

Compute requirements

- Steady models (no wake/blade interaction etc.)

1 blade	0.5 Mcells	1 CPU hour
1 stage (2 blades)	1.0 Mcells	3 CPU hours
1 component (5 stages)	5.0 Mcells	20 CPU hours

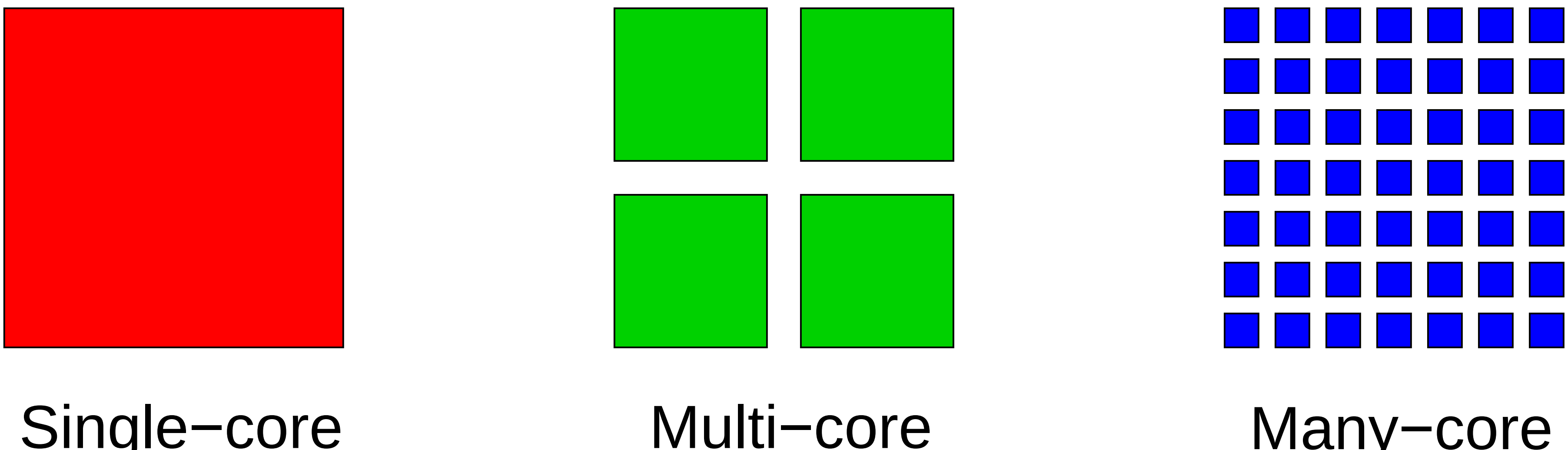
- Unsteady models (with wakes etc.)

1 component (1000 blades)	500 Mcells	0.1M CPU hours
Engine (4000 blades)	2 Gcells	1M CPU hours

Objectives

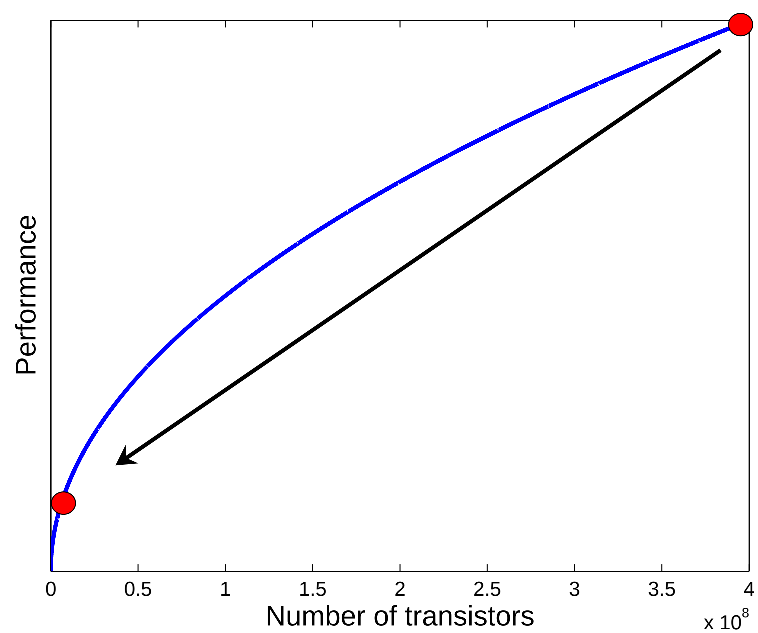
- Can CFD be made to run faster by using other types of processors?
- How to make sure it will continue to get faster with better processors in the future?

Background



Processor design

- $P \approx \sqrt{N_{trans}}$
- For a modern chip, $N_{trans} \approx 4 \cdot 10^8$
- 100 small cores with $4 \cdot 10^6$ transistors each gives 10 times the performance as 1 big core



Everyone is going parallel

- Every major chip vendor is switching to many-core processors
- All future processors will be massively parallel

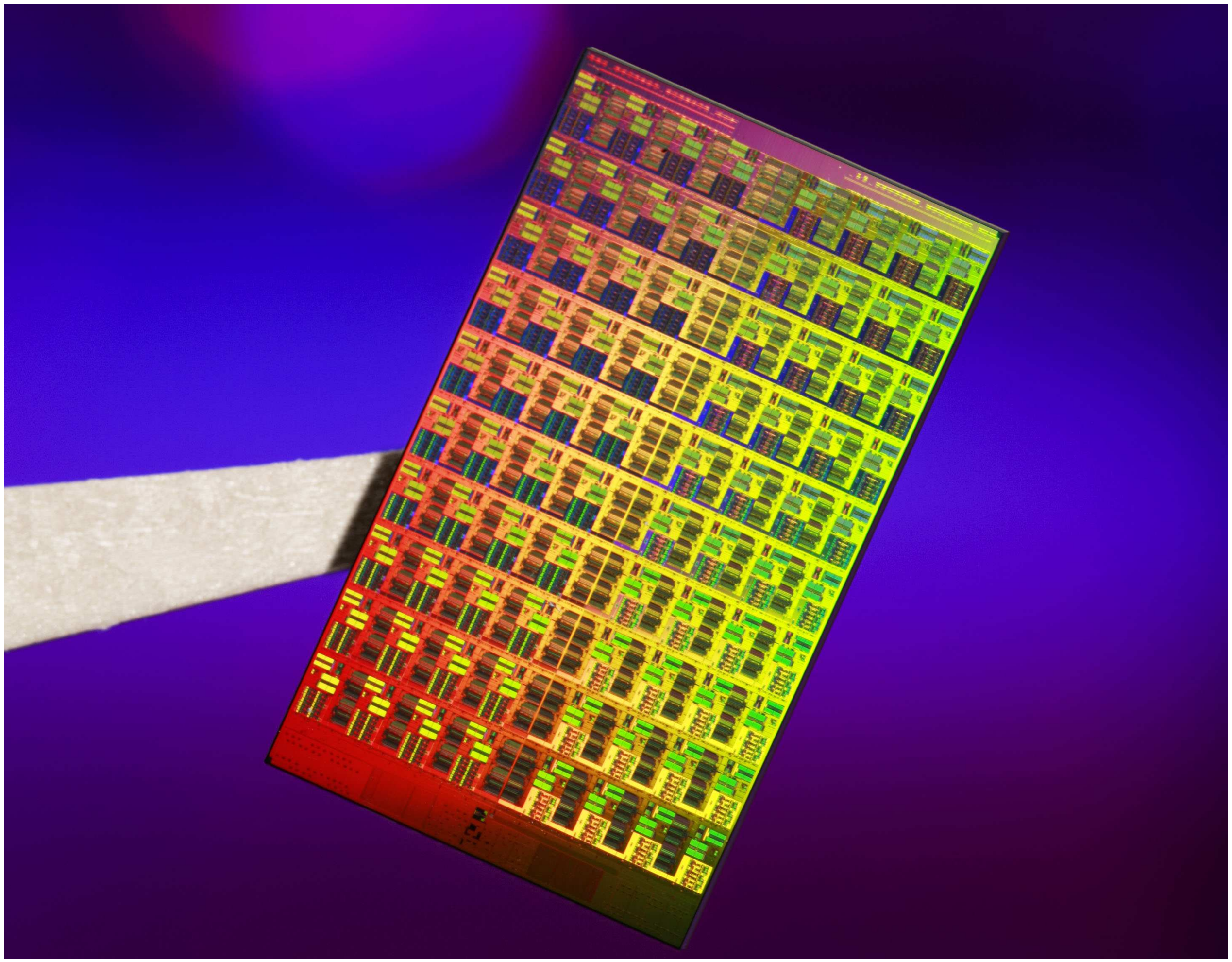
NVIDIA Tesla

IBM Cell

Sun Niagara


Sun
microsystems
solaris
ULTRASPARC
CoolThreads Technology

Intel Larrabee



Challenges

- Every processor has different characteristics (and in some cases languages and libraries)
- Codes have to be rewritten
- More difficult - use to have 1 process/thread per core with 1 MB cache
- Thousands of threads/processes per core
- 10-100KB on-chip memory per core

Benefits

- Possible to achieve step change in performance NOW
- Once the job is done, can expect to scale with Moore's Law like in the 80s and 90s

Scientific computing

There are two possible approaches

- Horizontal: Single language for all problems
- Vertical: Different language for every problem

A View From Berkeley

Scientific computing consists of seven different applications

1. Dense Linear Algebra
2. Sparse Linear Algebra
3. Spectral Methods
4. N-Body Methods
5. Structured Grids
6. Unstructured Grids
7. MapReduce

A View From Berkeley

Scientific computing consists of seven different applications

1. Dense Linear Algebra
2. Sparse Linear Algebra
3. Spectral Methods
4. N-Body Methods
5. Structured Grids
6. Unstructured Grids
7. MapReduce

Red applications relevant to CFD

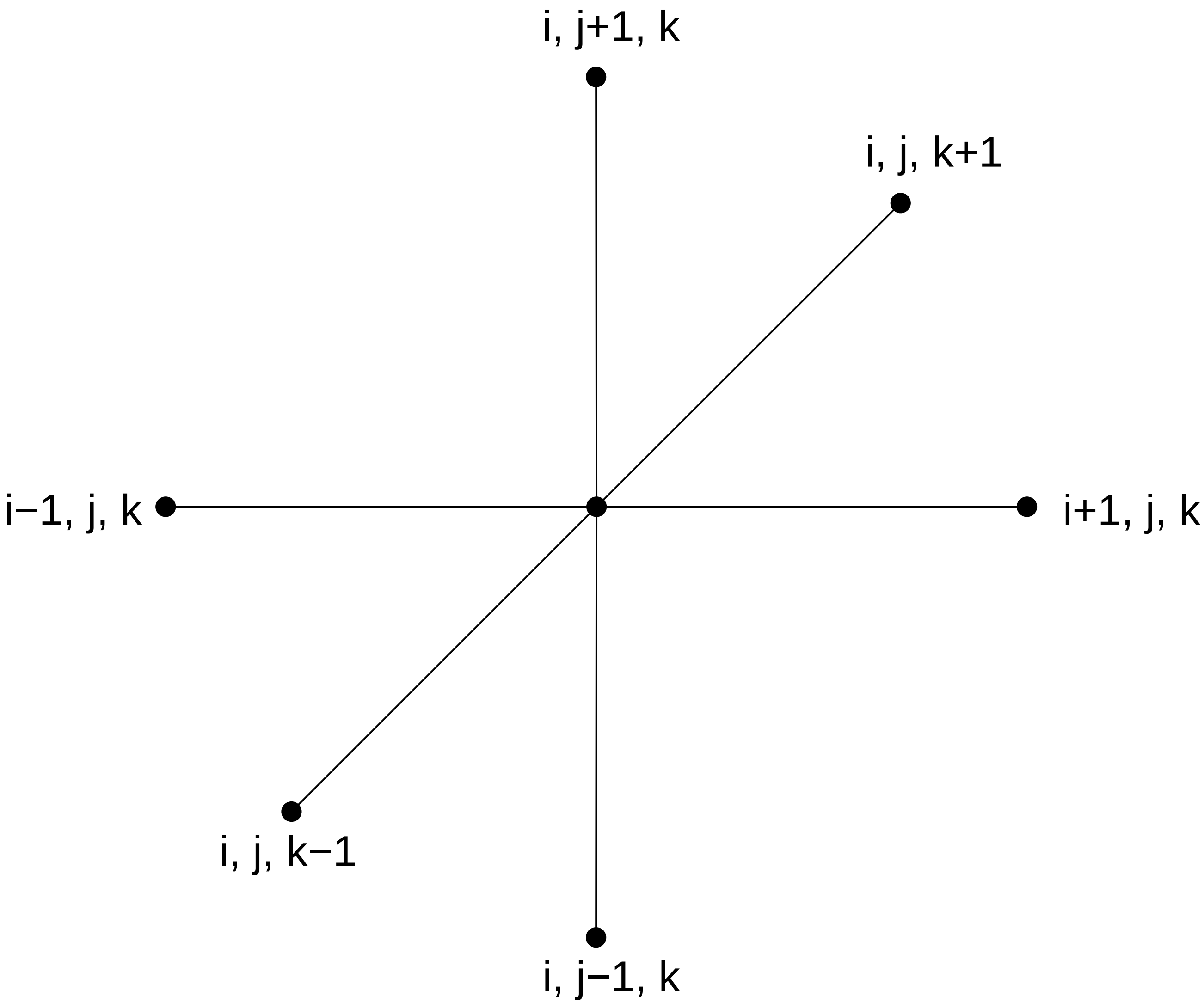
The Seven Dwarfs

The Seven Dwarfs

Structured grids

- Basic spatial discretisation for many CFD solvers
- Solvers consist of a series of *stencil operations* + *boundary conditions*

Stencil operations



Stencil operations

Evaluate $\frac{\partial^2 u}{\partial x^2}$ on a regular grid:

```
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
```

Boundary conditions

Set a variable to a fixed value on the $i = 0$ face:

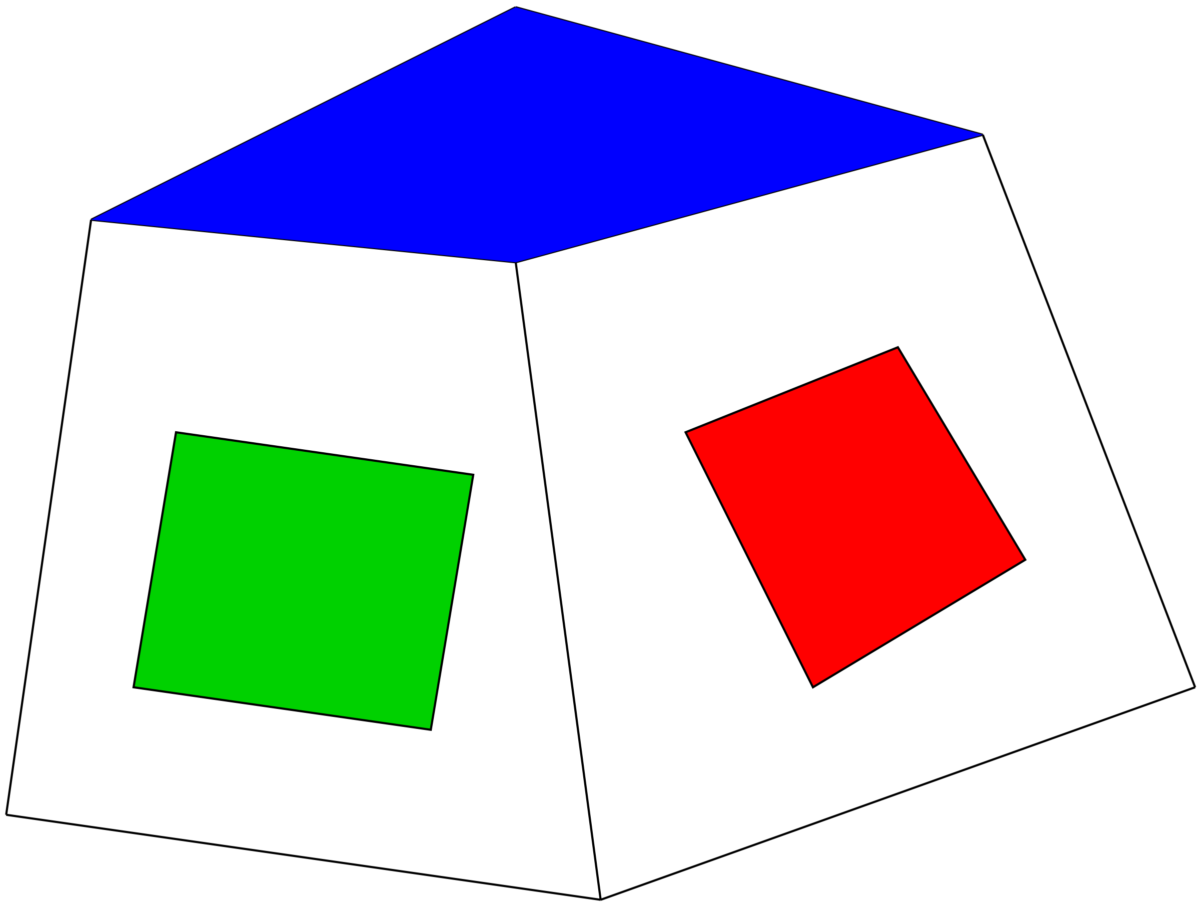
```
DO K=1,NK
  DO J=1,NJ
      U(0,J,K) = 300.0
  END DO
END DO
```

SBLOCK

- Vertical approach to structured grids
- Mini-language and library
- Can target any processor without changing the solver definition
- Currently supports CPUs and NVIDIA GPUs (Cell support is coming)

Fundamental abstraction

Blocks with patches



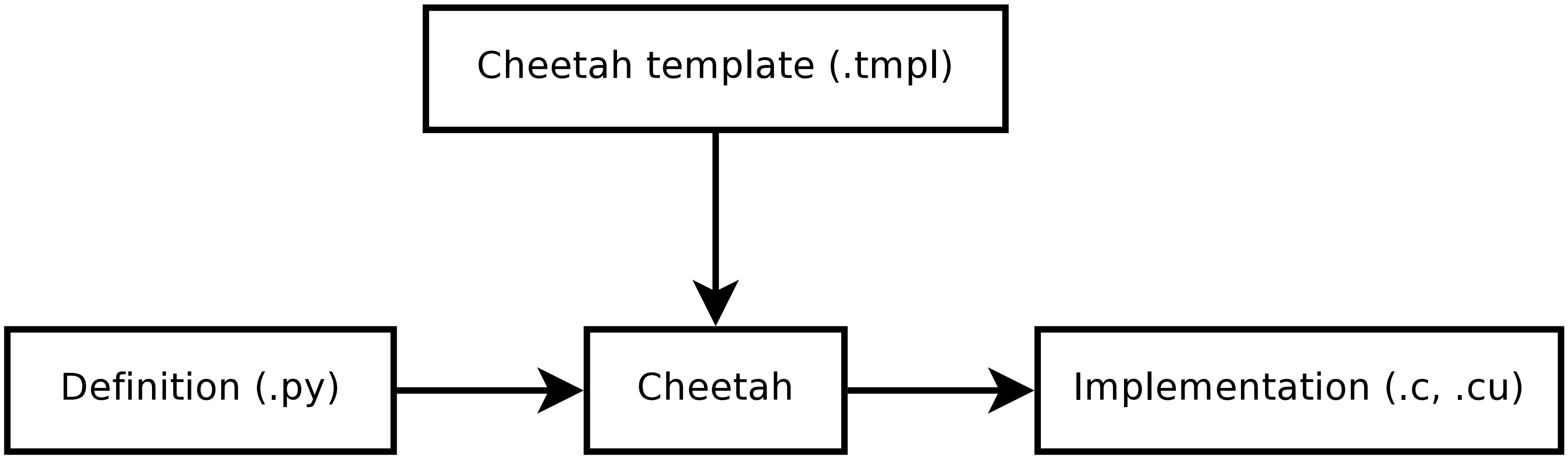
Stencil kernels

```
kind = "stencil"

avin = ["dx"]
bpin = ["u"]
bpout = ["d2udx2"]

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

Kernel compilation



TBLOCK

- Developed in-house at the lab by John Denton
- Blocks with arbitrary patch interfaces
- Simple and fast algorithm
- 15,000 lines of Fortran 77
- Main solver routines are only 5,000 lines
- Widely used in industry and academia

Turbostream

- Turbostream is TBLOCK in SBLOCK
- 2000 lines of C
- 3000 lines of Python kernels
- Code generated from Python kernels is 15,000 lines
- Source code is very similar to TBLOCK – every subroutine has an equivalent SBLOCK kernel

Speed-up results

- Two different scenarios

High-end desktop

- 2 Intel Quad Cores
- 6 NVIDIA GPUs
- £3,000
- 30x speed-up
- Can do routine design calculations in less than 2 minutes

Cluster

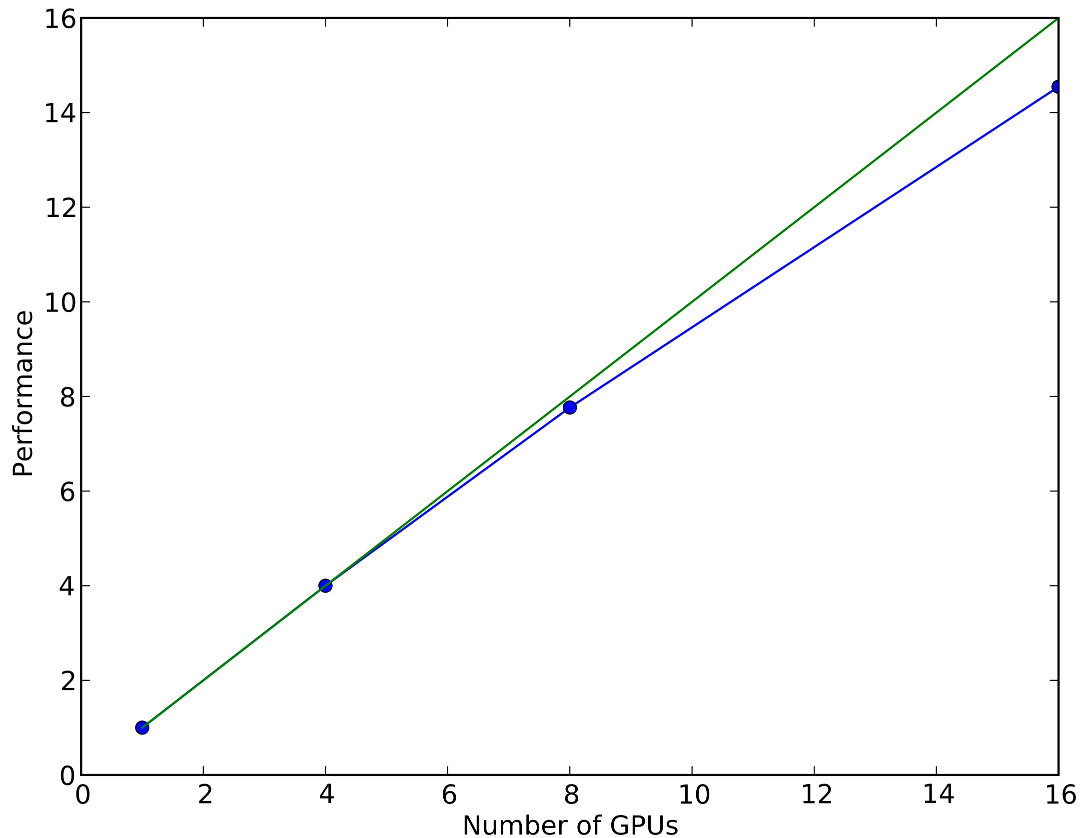
- 4 GPUs in 1 U (NVIDIA Tesla)
- But needs extra control unit
- Not as dense as CPU clusters (yet)
- Speed-ups of 10x on a per-cost, per-watt basis

Cluster

We now have one of these!

Cluster scaling

- Scaling when increasing job size:



Conclusions

- Many-core processors can speed up CFD calculations
- Difficult to support all platforms and mantain portability through hand-coding
- Use a framework instead - write once run anywhere