Large-scale Gas Turbine Simulations on GPU clusters

Tobias Brandvik and Graham Pullan

Whittle Laboratory

University of Cambridge
A large-scale simulation
Overview

• PART I: Turbomachinery
• PART II: Stencil-based PDE solvers on multi-core
• PART III: A new solver for turbomachinery flows
• PART IV: Example simulations
PART I: Turbomachinery
Turbomachinery

- Thousands of blades, arranged in rows
- Each row has a bespoke blade profile designed with CFD
Research problem - surge prediction
Research problem - surge prediction
Deveryson low-speed compressor rig
Deverson simulation

- Typical routine simulation
- Structured grid, steady state
- 3 million grid nodes
- 8 hours on four CPU cores
- 25 minutes on Fermi
PART II: Stencil-based PDE solvers
Aim

To produce an order of magnitude reduction in the run-time of CFD solvers for the same hardware cost
Aim

To produce an order of magnitude reduction in the run-time of CFD solvers for the same hardware cost

And maintain performance portability across current and future multi-core processors
Structured grids

- Our work focuses on multi-block structured grids
- Structured grid solvers: a series of stencil operations
- Stencil operations: discrete approximations of the equations
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

<table>
<thead>
<tr>
<th>Stencil operations</th>
<th>Non-stencil operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set flux, Sum flux, Shear Stress ...</td>
<td>Multigrid, Mixing plane, Sliding plane ...</td>
</tr>
</tbody>
</table>

90% of run-time
10% of run-time
Stencil operations on multi-core processors

- Single implementation?
- Multiple implementations?
- Alternative:
  - High level language for stencil operations
  - Source-to-source compilation
Stencil example

- Structured grid indexing
Stencil example

\[ \frac{\partial^2 u}{\partial x^2} \] in Fortran

```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:

```python
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
• 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
Stencil example

- Other stencils are in principle the same

Set fluxes  Sum fluxes  Smoothing
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
Software framework

• Framework: Templates + Run-time library
• Library provides:
  • Uniform API for both CPUs and GPUs
  • MPI
  • Reductions
  • Parallel sparse matrix vector multiplication
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

Nehalem

Phenom II

GT200

Single precision

GFLOP/s

No. of threads

1 2 4 8

GFLOP/s

No. of threads

1 2 4

No. of MPs

30

Double precision

GFLOP/s

No. of threads

1 2 4 8

GFLOP/s

No. of threads

1 2 4

GFLOP/s

No. of MPs

30
PART III: A new solver for turbomachinery flows
Turbostream

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

- Developed by John Denton
- Blocks with arbitrary patch interfaces
- Simple and fast algorithm
- 15,000 lines of Fortran 77
- Main solver routines are only 5000 lines
Denton Codes

• TBLOCK is the latest of the “Denton Codes”
• Dates back to the late 1970s
• Used for some part of the design process at most turbomachinery manufacturers
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
Implementation notes

- Stencil kernels are “easy”: Fortran 77 to Python by hand
- What about the last 10%?
The last 10%

- The last 10% becomes important when the runtime of the other 90% has been reduced by a factor of 10
- Multigrid
- Sliding planes
- Mixing planes
- Inlet flow/outlet flow
Multigrid

• Algorithm for improving convergence
Multigrid

- Algorithm for improving convergence

Coarsening

Interpolation
Multigrid

- Algorithm for improving convergence

Coarsening

Interpolation

- Implemented as a parallel sparse matrix vector multiplication (pSPMV)
Sliding planes

- 1D interpolation between grids in relative motion
Sliding planes

- 1D interpolation between grids in relative motion
1D interpolation between grids in relative motion

Also done with a pSPMV
Mixing planes

• Interface which mixes the flow between a stationary and rotating blade row

• Difficult case:
  • Complicated coding (needs many small reductions)
  • Can take up a significant fraction of runtime for particular cases

• Hand-code implementation for each processor
Inlet and outflow

- Straightforward case
  - Easy coding (local operation)
  - Consumes small fraction of runtime

1. Get data from GPU
2. Perform computation on CPU
3. Write data back to GPU memory
# Turbostream performance

<table>
<thead>
<tr>
<th>Processor</th>
<th>TBLOCK performance</th>
<th>Turbostream performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>1.21</td>
<td>1.48</td>
</tr>
<tr>
<td>Phenom II</td>
<td>1</td>
<td>0.89</td>
</tr>
<tr>
<td>GT200</td>
<td>-</td>
<td>10.2</td>
</tr>
</tbody>
</table>

- TBLOCK runs in parallel on all four CPU cores using MPI
- Initial Fermi results are 2x faster – larger shared memory
Multi-processor performance

• University of Cambridge became a NVIDIA CCoE in December 2008

• NVIDIA donated 32 S1070s which were added to existing “Darwin” supercomputer

• Nehalem-based CPU nodes from Dell

• QDR Infiniband
Multi-processor performance

- Benchmark case is an unsteady simulation of a turbine stage
Multi-processor implementation

1. Stencil kernels
2. Boundary conditions
3. Exchange halo nodes (MPI)
Multi-processor performance
PART IV: Example simulations
Example simulations

- Three-stage turbine (small scale)
- Cooling hole simulation (medium scale)
- Three-stage compressor (large scale)
Three-stage turbine
Single stage geometry - clean
Single stage geometry - real
Entropy function through machine

- 10 hours on a single CPU
- 8 minutes on four GPUs
Total pressure loss, Stator 3 exit
Cooling hole simulation

- Gas at ~1800 K from combustor
- Drill holes in the blade for cooling
- Currently model tens of holes at a time

Image from Texas A&M website:
http://www1.mengr.tamu.edu/tthl/projects.html
Cooling hole simulation

- LES calculation of a single hole
- Used for better understanding and models
- 15 million grid nodes
- Time-accurate
- 2 days on 16 GPUs
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

• 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).