H L R S

High Performance Computing in Science and Engineering

The 6th Results & Review Workshop
of the HPC Center Stuttgart (HLRS)

parpp3d++ – a Parallel HPC Research Code for CFD

Block smoothers in parallel multigrid methods; Hitachi SR8000 vs. Linuxcluster

Sven H.M. Buijssen <sven.buijssen@math.uni-dortmund.de>,
Stefan Turek <ture@featflow.de>

Institute of Applied Mathematics
University of Dortmund
Germany

Outline of this talk

- Presentation of a portable research code for the simulation of 3-D incompressible nonstationary flows.

Outline of this talk

- Presentation of a portable research code for the simulation of 3-D incompressible nonstationary flows.
- Comparison of run times and scaling on Hitachi SR8000, Cray T3E (Jülich), Linuxcluster HELICS

Outline of this talk

- Presentation of a portable research code for the simulation of 3-D incompressible nonstationary flows.
- Comparison of run times and scaling on Hitachi SR8000, Cray T3E (Jülich), Linuxcluster HELICS
- Use of block smoothers in parallel multigrid methods
  $\hookrightarrow$ (numerical) consequences for parallel efficiencies

Outline of this talk

- Presentation of a portable research code for the simulation of 3-D incompressible nonstationary flows.
- Comparison of run times and scaling on Hitachi SR8000, Cray T3E (Jülich), Linuxcluster HELICS
- Use of block smoothers in parallel multigrid methods
  $\hookrightarrow$ (numerical) consequences for parallel efficiencies
- Some current and recent applications of the code

The underlying problem

- Incompressible nonstationary Navier–Stokes equations

$$\mathbf{u}_t - \nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f}, \qquad \nabla\cdot\mathbf{u} = 0$$

  have to be solved

- Finite element discretisation of this system of PDEs leads to huge systems of (non-)linear equations ($\geqslant 10^7$ unknowns per timestep)

- Solving with parallel multigrid methods (chosen for their optimal numerical complexity for ill-conditioned PDE problems)

Mathematical background (1)

Numerics applied to solve the Navier–Stokes equations:

- (implicit) 2nd order discretisation in time
  (both Fractional-Step-Θ- and Crank-Nicolson-scheme
  $\hookrightarrow$ adaptive time stepping possible)

Mathematical background (1)

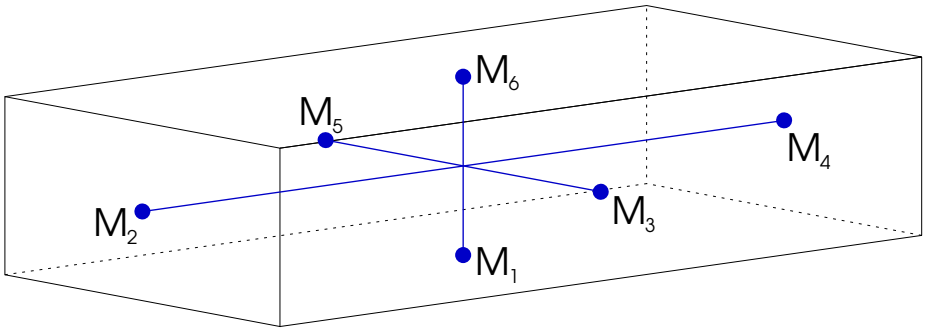
Numerics applied to solve the Navier–Stokes equations:

- (implicit) 2nd order discretisation in time
  (both Fractional-Step-Θ- and Crank-Nicolson-scheme
  $\hookrightarrow$ adaptive time stepping possible)
- finite element approach for space discretisation
  (non-parametric non-conforming $\tilde{Q}_1/Q_0$ ansatz)

Mathematical background (1)

Numerics applied to solve the Navier–Stokes equations:

- (implicit) 2nd order discretisation in time
  (both Fractional-Step-Θ- and Crank-Nicolson-scheme
  $\hookrightarrow$ adaptive time stepping possible)
- finite element approach for space discretisation
  (non-parametric non-conforming $\tilde{Q}_1/Q_0$ ansatz)

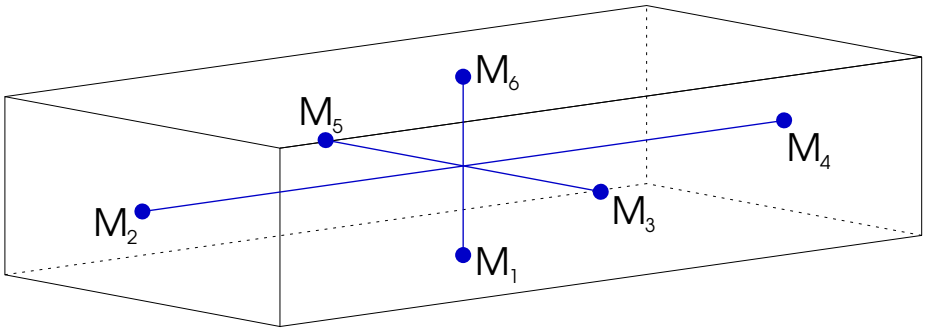


$\hookrightarrow$ hexahedral grids

Mathematical background (1)

Numerics applied to solve the Navier–Stokes equations:

- (implicit) 2nd order discretisation in time
  (both Fractional-Step-Θ- and Crank-Nicolson-scheme
  $\hookrightarrow$ adaptive time stepping possible)
- finite element approach for space discretisation
  (non-parametric non-conforming $\tilde{Q}_1/Q_0$ ansatz)



  $\hookrightarrow$ hexahedral grids
- Stabilisation of convective term with (Samarskij)
  Upwind scheme

Mathematical background (2)

Numerics applied to solve the Navier–Stokes equations (cont.):

- Within each time step:
  - Discrete Projection method to decouple velocity-pressure problem

Mathematical background (2)

Numerics applied to solve the Navier–Stokes equations (cont.):

- Within each time step:
  - Discrete Projection method to decouple velocity-pressure problem
  - The resulting nonlinear Burgers equation in $\mathbf{u}$ is solved by fixed point defect correction method (outer loop) and multigrid (inner loop)

Mathematical background (2)

Numerics applied to solve the Navier–Stokes equations (cont.):

- Within each time step:
  - Discrete Projection method to decouple velocity-pressure problem
  - The resulting nonlinear Burgers equation in $\mathbf{u}$ is solved by fixed point defect correction method (outer loop) and multigrid (inner loop)
  - Remaining linear problem in p (Pressure Poisson problem, ill-conditioned!) is solved with multigrid preconditioned conjugate gradient method

Mathematical background (4)

Parallelisation strategy

- Domain decomposition using graph-oriented partitioning tool (Metis or PARTY library)

Mathematical background (4)

Parallelisation strategy

- Domain decomposition using graph-oriented partitioning tool (Metis or PARTY library)
- Uniform refinement of each parallel block, typically 4-6 times

Mathematical background (4)

Parallelisation strategy

- Domain decomposition using graph-oriented partitioning tool (Metis or PARTY library)
- Uniform refinement of each parallel block, typically 4-6 times
- local communication between at most two adjacent parallel blocks (due to FEM ansatz: $\tilde{Q}_1/Q_0$!)

Mathematical background (4)

Parallelisation strategy

- Domain decomposition using graph-oriented partitioning tool (Metis or PARTY library)
- Uniform refinement of each parallel block, typically 4-6 times
- local communication between at most two adjacent parallel blocks (due to FEM ansatz: $\tilde{Q}_1/Q_0$!)
- block smoothing

Mathematical background (5)

Smoothing:

- Numerical and computational complexity of multigrid stands and falls with the smoothing algorithms used.
- smoothers, however, have in general a highly recursive character

Mathematical background (5)

Smoothing:

- Numerical and computational complexity of multigrid stands and falls with the smoothing algorithms used.
- smoothers, however, have in general a highly recursive character

Idea of block smoothing:

- Avoid direct parallelisation of global smoother (significant amount of communication!)
- Instead: Apply the same smoothing algorithm within each parallel block only
  (parallel block = one patch of elements from the partitioning algorithm)

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

In limit case:

block smoother turns into simple Jacobi iteration!

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

In limit case:
block smoother turns into simple Jacobi iteration!

- Thus, the number of multigrid sweps will increase.

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

In limit case:
block smoother turns into simple Jacobi iteration!

- Thus, the number of multigrid sweps will increase.

Significantly?

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

In limit case:
block smoother turns into simple Jacobi iteration!

- Thus, the number of multigrid sweps will increase.

  Significantly?

  – for Burgers equation: probably not, good conditioned (scaling with time step k)

Mathematical background (6)

Consequences of block smoothing:

With increasing number of parallel processes:

- It takes more than 1 iteration to spread information across the grid (weakened smoothing property)

In limit case:
block smoother turns into simple Jacobi iteration!

- Thus, the number of multigrid sweps will increase.

  Significantly?

  – for Burgers equation: probably not, good conditioned (scaling with time step k)

  – **for discrete Pressure Poisson equation: probably, problem with condition of** $\mathrm{O}(h^{-2})$

Implementation

- Code written in C/C++
  (thrifty usage of comfortable, but performance reducing language elements)

Implementation

- Code written in C/C++
  (thrifty usage of comfortable, but performance reducing language elements)
- designed to run on most MPP units running some kind of Unix flavour and providing an MPI environment.
  (tested on clusters of Sun, SGI & Alpha Workstations, Linux-PCs, Cray T3E, SR8000, ...)

Implementation

- Code written in C/C++
  (thrifty usage of comfortable, but performance reducing language elements)
- designed to run on most MPP units running some kind of Unix flavour and providing an MPI environment.
  (tested on clusters of Sun, SGI & Alpha Workstations, Linux-PCs, Cray T3E, SR8000, ...)

  $\hookrightarrow$ does not incorporate explicit vector processing routines

Implementation

- Code written in C/C++
  (thrifty usage of comfortable, but performance reducing language elements)
- designed to run on most MPP units running some kind of Unix flavour and providing an MPI environment.
  (tested on clusters of Sun, SGI & Alpha Workstations, Linux-PCs, Cray T3E, SR8000, ...)

  ↪ does not incorporate explicit vector processing routines
- has a well-tested sequential (F77) counterpart from the FEATFLOW package (author: Turek et al., since 1985)

Numerical section

Numerical section

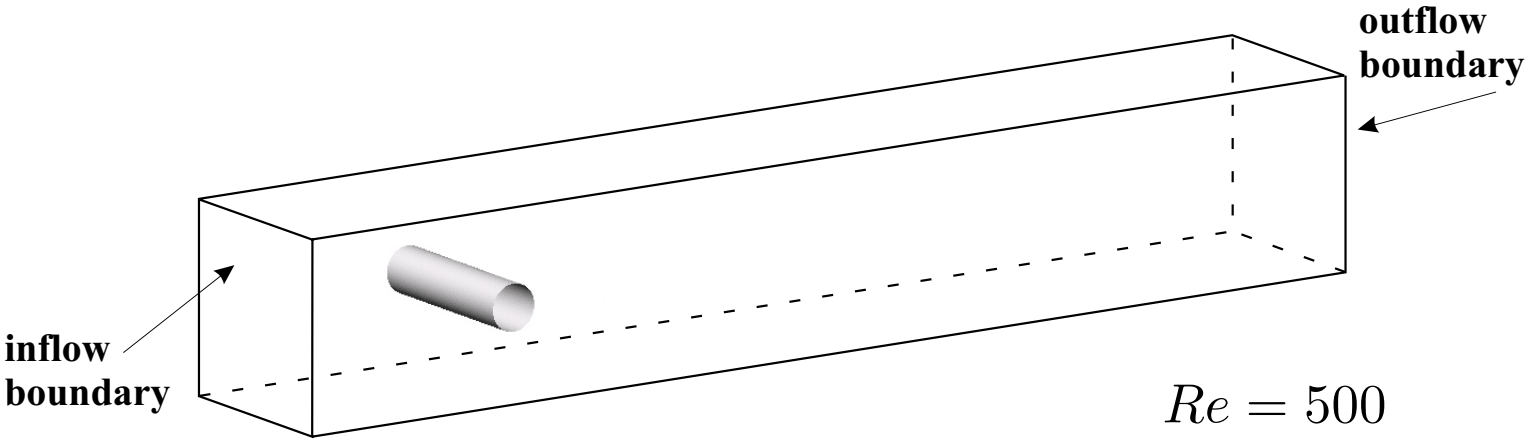
- Roughly a dozen different flow problems have been simulated yet

Numerical section

- Roughly a dozen different flow problems have been simulated yet
- Pars pro toto, the typical effects that can be observed will be illustrated on the basis of the DFG benchmark 3D-2Z from 1995 ("channel flow around a cylinder")

Numerical section

DFG benchmark 3D-2Z from 1995:

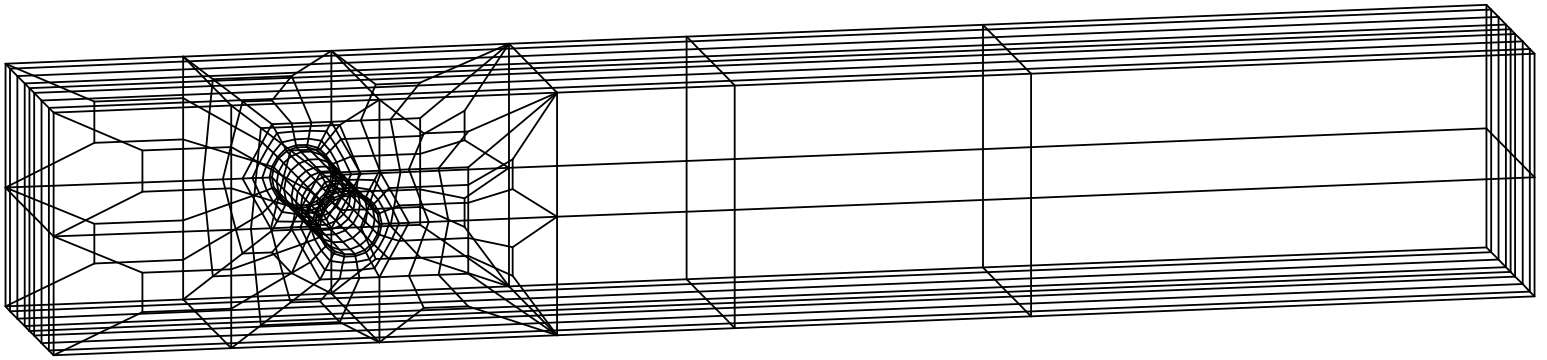


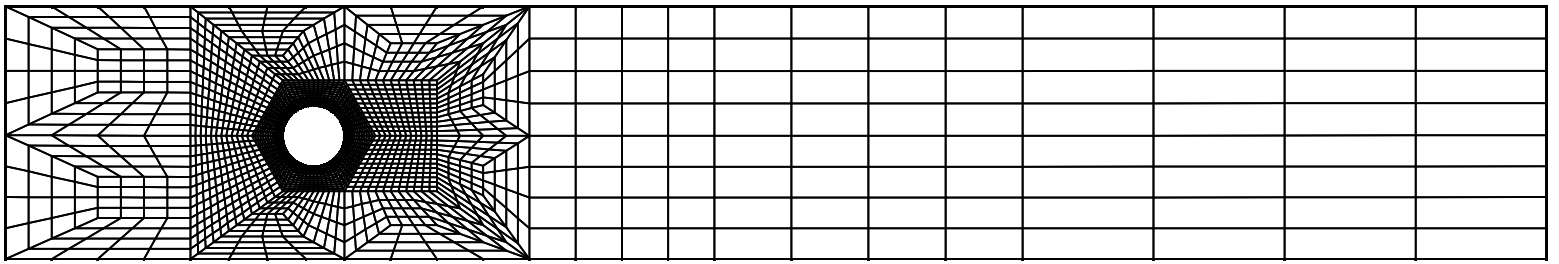
$Re = 500$

aspect ratio ≈ 20

Numerical section

Grids:





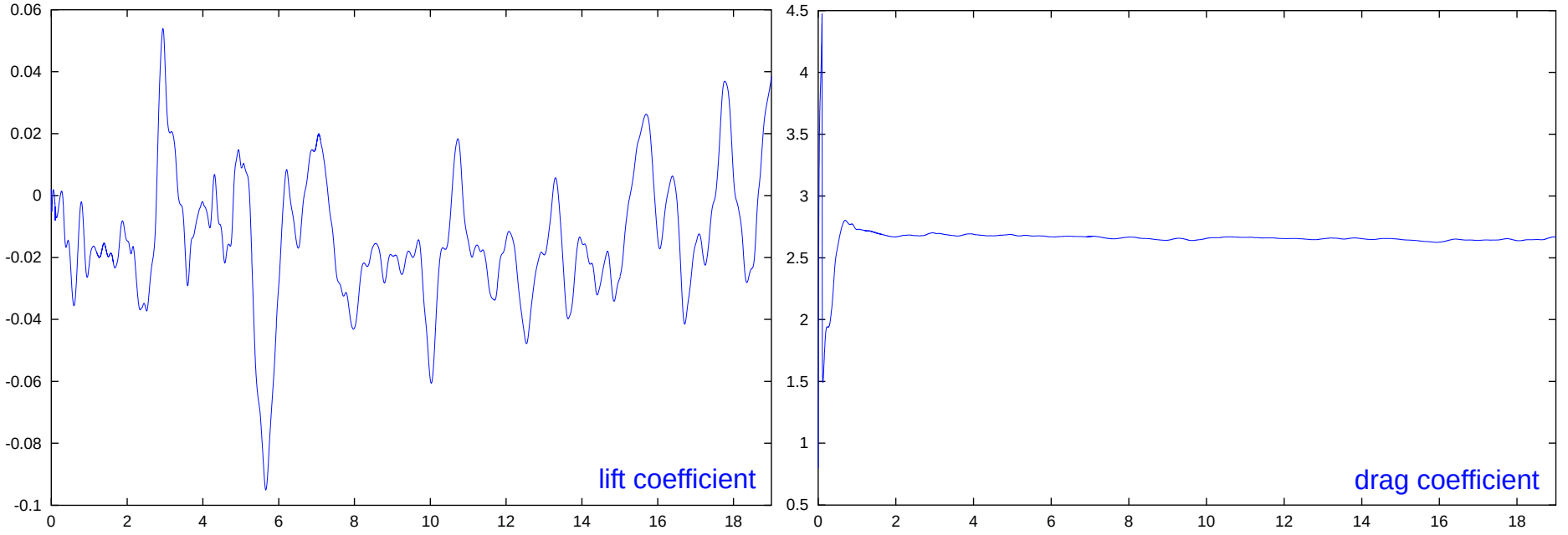
2x refined grid, side view

Numerical section

We did some long term simulation ($T_{End} = 20s$) ...

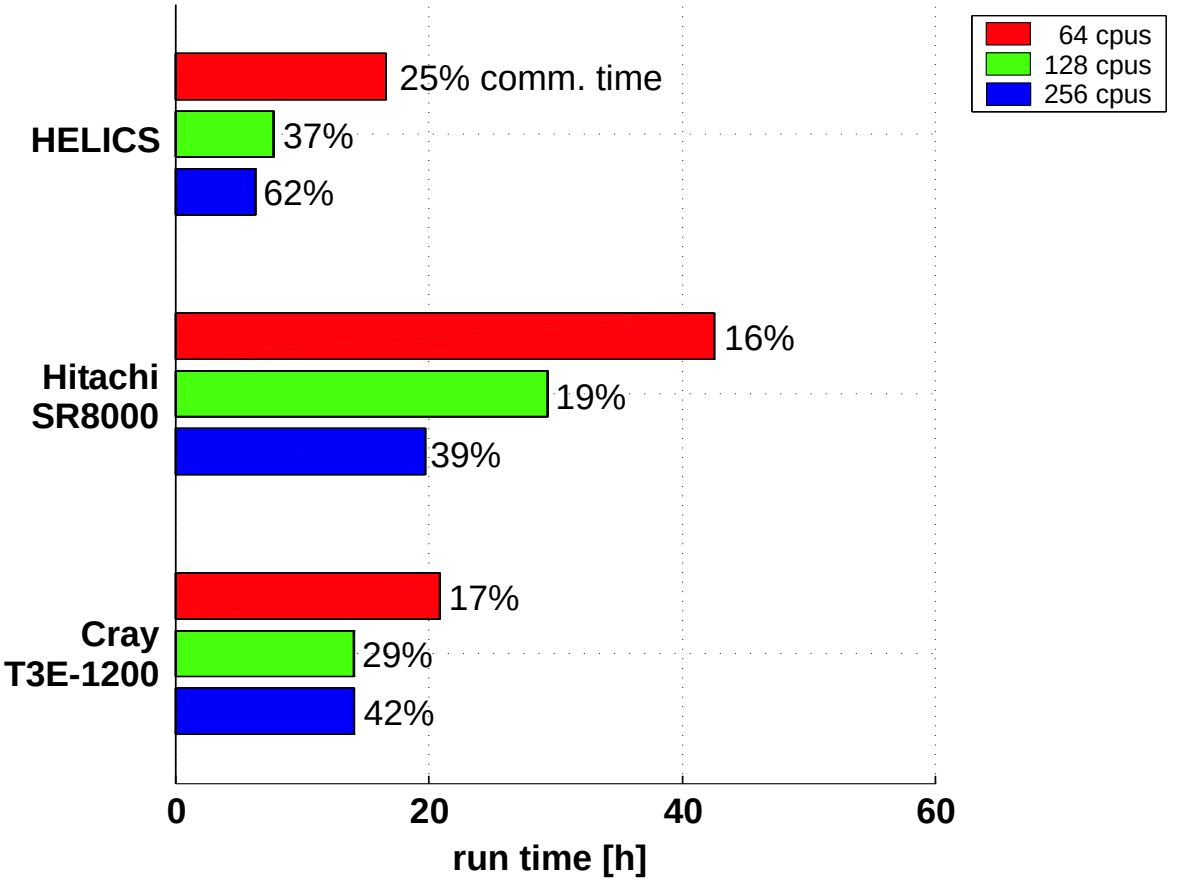
- degrees of freedom: 32 million (9 GByte RAM)
- #time steps: 6.500

... computed lift and drag coefficients ...



Numerical section

... and studied the scaling of the program on different platforms for this problem (64, 128, 256 cpus) ($T = [0, 1]$)



(HELICS = Linuxcluster at IWR Heidelberg, 512 Athlons 1.4 GHz, Myrinet, www.helics.de)

Observations (1)

- Hitachi SR8000 is conspicuously in last position
  - sCC compiler (latest release) used
  - run times with g++ worse
  - code does not compile with KCC
    (although it does on Cray T3E-1200)
- The fact that Hitachi is outperformed by a (much cheaper) Linux cluster (factor 2–3 in average) has been perceived for different problem sizes, degrees of parallelism and geometries.
  That's the price for just using MPI and not incorporating vector processing techniques directly into the code.

Observations (2)

- Performance of a single cpu on SR8000 is just not appropriately enough exploited by the code

Observations (2)

- Performance of a single cpu on SR8000 is just not appropriately enough exploited by the code
- But:
  How many of the codes being granted access to SR8000 are especially designed for this architecture?
  Percentage?

Observations (2)

- Performance of a single cpu on SR8000 is just not appropriately enough exploited by the code
- But:
  How many of the codes being granted access to SR8000 are especially designed for this architecture?
  Percentage?
- I bet: a significant percentage uses SR8000 as one MPP unit among others, too.
  (relying on MPI and compiler optimisations of the code)

Observations (3)

Things are not merely bad at SR8000!

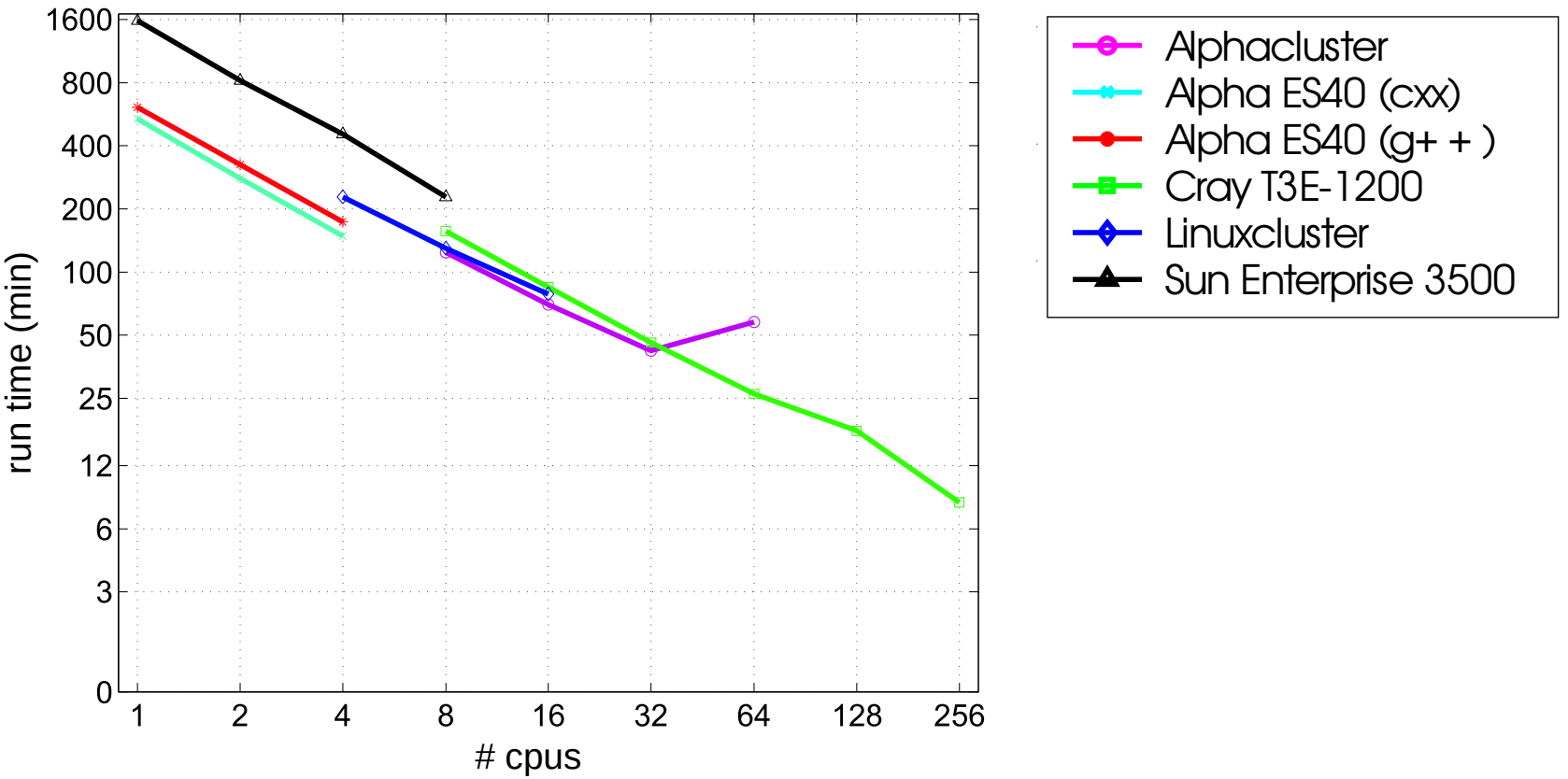
- best communication network, least time spent in communication routines

Observations (3)

Things are not merely bad at SR8000!

- best communication network, least time spent in communication routines
- hence, scaling is best of all tested platforms yet

Reduction in run time for different problem



(Lid-Driven Cavity, 11 million d.o.f., 100 time steps)

Parallel efficiencies

- Parallel efficiencies are rather good at few cpus: 0.9 - 0.95
- But drop then to 0.6 - 0.7 at higher degrees of parallelism (as long as problem size is reasonable big)

Parallel efficiencies

- Parallel efficiencies are rather good at few cpus: 0.9 - 0.95
- But drop then to 0.6 - 0.7 at higher degrees of parallelism (as long as problem size is reasonable big)

Reason?

Parallel efficiencies

- Parallel efficiencies are rather good at few cpus:
  0.9 - 0.95
- But drop then to 0.6 - 0.7 at higher degrees of parallelism (as long as problem size is reasonable big)

Reason?

- communication time increases at half the speed the parallel efficiencies drop
  $\hookrightarrow$ there must be a different effect!

The Pressure Poisson Problem

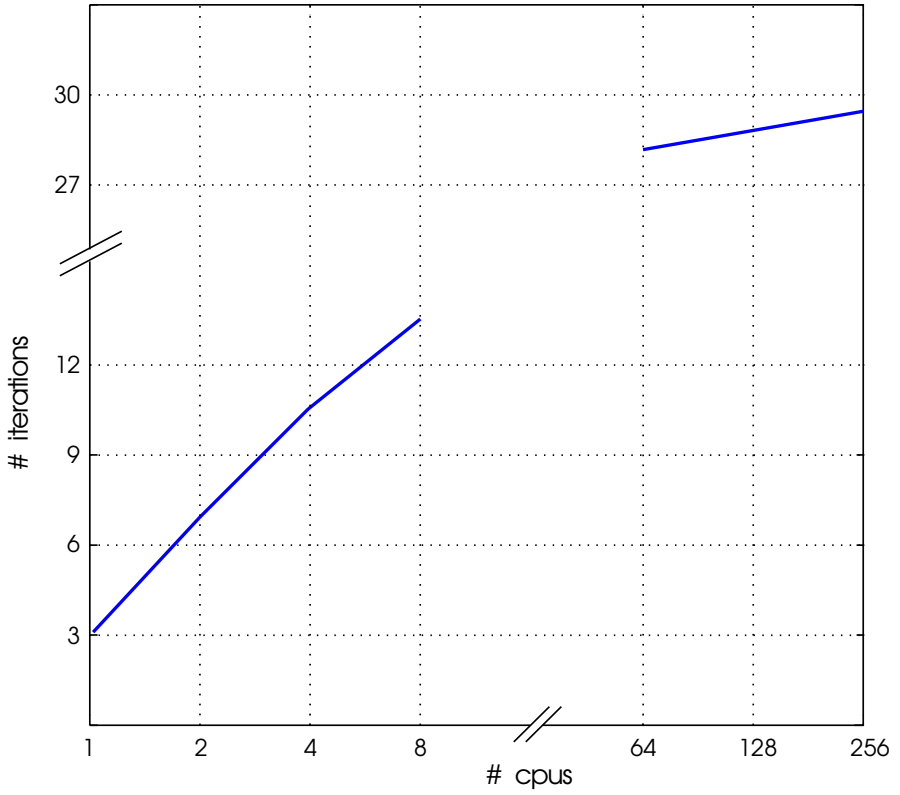
- Solving the Pressure Poisson Problem takes 10-15% of overall run time for 1-2 cpus

The Pressure Poisson Problem

- Solving the Pressure Poisson Problem takes 10-15% of overall run time for 1-2 cpus
- Same problem, 64 cpus or more: 50-65% of run time! What is going on?

The Pressure Poisson Problem

- Solving the Pressure Poisson Problem takes 10-15% of overall run time for 1-2 cpus
- Same problem, 64 cpus or more: 50-65% of run time! What is going on?



Remarks

- Deterioration is depending on the aspect ratios.

Remarks

- Deterioration is depending on the aspect ratios.
- Speedup is not that bad if comparing e.g. 16 vs. 128 cpus

Remarks

- Deterioration is depending on the aspect ratios.
- Speedup is not that bad if comparing e.g. 16 vs. 128 cpus
- But:
  The performance of the code is not yet satisfying enough to kick off incorporating more features like
  - heat transfer (Boussinesq)
  - $k - \epsilon$–model
  - free surface
  - multiphase flow (bubble colon reactors)

  which already exist in sequential.

Remarks

- Hence, we're looking for a solver for the Pressure Poisson Problem which does not deteriorate with the number of cpus

Remarks

- Hence, we're looking for a solver for the Pressure Poisson Problem which does not deteriorate with the number of cpus
- And we have found a candidate. The implementation is done in the context of the projects ScaRC and FEAST.

A current application

A current application:

BMBF project 03C0348A: design of ceramic wall reactors

- Intension: development of ceramic wall reactors and ceramic plate heat exchangers as micro reactors for heterogeneously catalysed gas phase reactions.

A current application:

BMBF project 03C0348A: design of ceramic wall reactors

- Intension: development of ceramic wall reactors and ceramic plate heat exchangers as micro reactors for heterogeneously catalysed gas phase reactions.
- Main aim: increasing performance of reactor by optimising its geometry to gain a equally distributed velocity field.

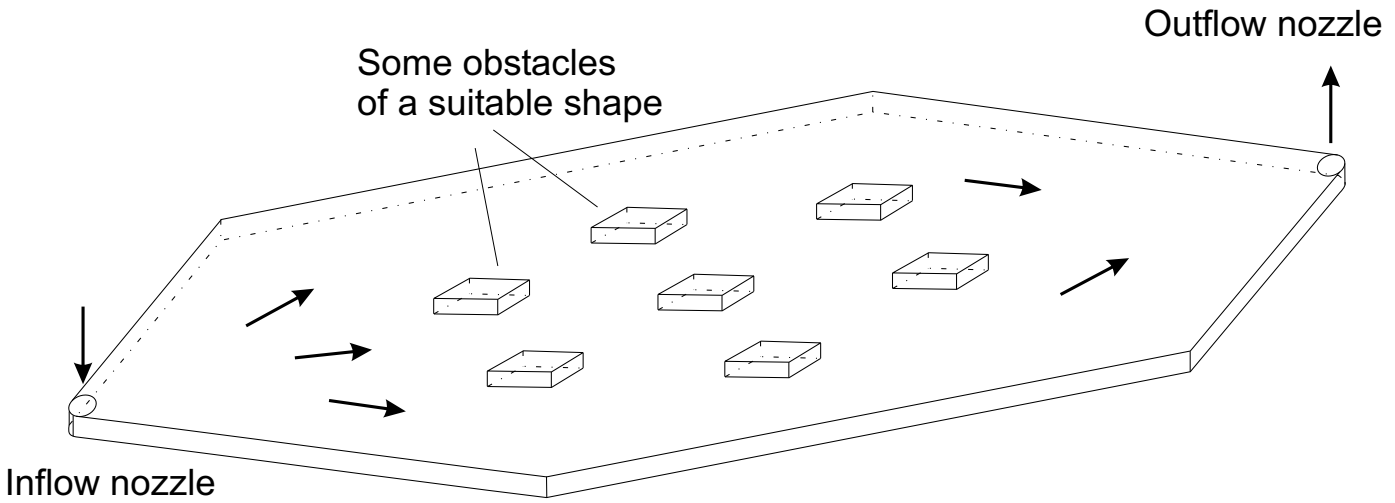
A current application:

BMBF project 03C0348A: design of ceramic wall reactors

- Intension: development of ceramic wall reactors and ceramic plate heat exchangers as micro reactors for heterogeneously catalysed gas phase reactions.
- Main aim: increasing performance of reactor by optimising its geometry to gain a equally distributed velocity field.
- Given this, the partners (Institute of Chemical Engineering, University of Dortmund) and Hermsdorfer Institute for Technical Ceramics) will try to calibrate catalytic activity, diffusive mass transport and heat removal to attain an optimal temperature distribution.

A current application:

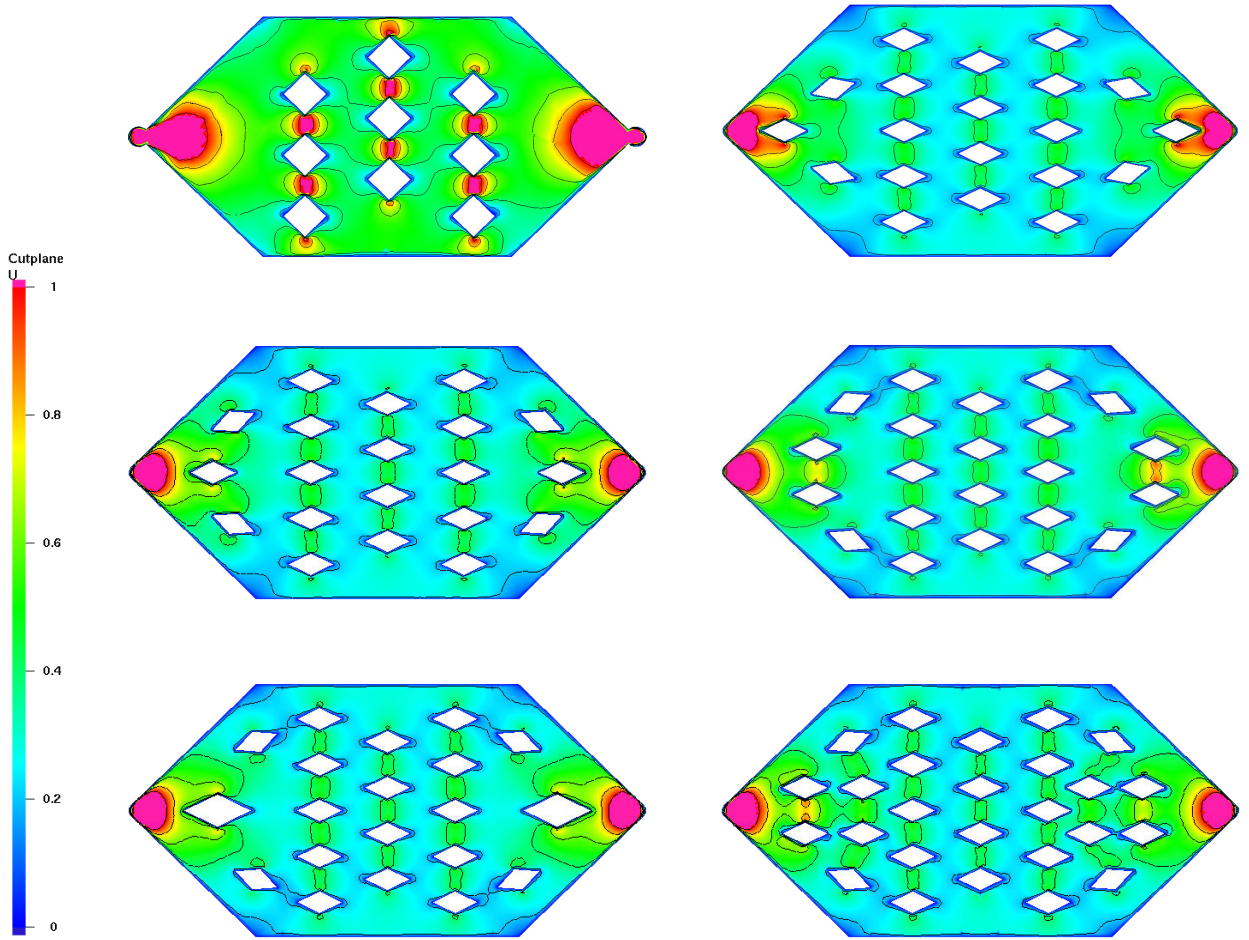
Sketch of overall geometry of ceramic wall reactor and flow directions



- 2 dozen different geometries so far
- average problem size: 60 million d.o.f., 100 time steps to stationary limit case
- 12h with 16 cpus on SR8000 per simulation

A current application:

Velocity field for some geometries



Conclusion and Outlook

- We have a parallel solver for 3-D incompressible nonstationary Navier–Stokes equations which is fast, robust and portable.

Conclusion and Outlook

- We have a parallel solver for 3-D incompressible nonstationary Navier–Stokes equations which is fast, robust and portable.
- Exploitation of performance of computers like Cray T3E1200 and SR8000 is still too poor (due to implementation and numerics)

Conclusion and Outlook

- We have a parallel solver for 3-D incompressible nonstationary Navier–Stokes equations which is fast, robust and portable.
- Exploitation of performance of computers like Cray T3E1200 and SR8000 is still too poor (due to implementation and numerics)
- But hopefully, my access to SR8000 will not be discarded within the next days!

Conclusion and Outlook

- We have a parallel solver for 3-D incompressible nonstationary Navier–Stokes equations which is fast, robust and portable.
- Exploitation of performance of computers like Cray T3E1200 and SR8000 is still too poor (due to implementation and numerics)
- But hopefully, my access to SR8000 will not be discarded within the next days!
- A new package is currently written which incorporates both better numerics (hardly any deterioration) and hardware-oriented implementation techniques (vectorisation, better cache exploitation).
  First tests show that we can get nearly 30-50% of machine's peak performance.