HPC Deployment of OpenFOAM in an Industrial Setting

Hrvoje Jasak

`h.jasak@wikki.co.uk`

Wikki Ltd, United Kingdom

PRACE Seminar: Industrial Usage of HPC

Stockholm, Sweden, 28-29 March 2011

Overview

Objective

- Review the state and prospects for massive parallelisation in CFD codes, with review of implementation in OpenFOAM

Topics

1. Introduction: Parallelisation of CFD simulation work-flow
2. Components
3. Parallel algorithms
4. Parallelisation and efficiency of the linear algebra toolkit
5. “Auxiliary algorithms” and parallel efficiency
6. Points of interest and summary

Parallelisation in CFD

Implications of Deployment of Open Source CFD

- Academic rationale for open source code is clear: open collaboration and sharing
- Industrial users rely on commercial software with strict quality control and dedicated support teams
- . . . but its flexibitily is not sufficient, development is too slow, support quality varies and **parallel CFD licences are massively over-priced**

Open Source CFD Solution

- Reminder: Open Source and GPL does not imply zero price
  - Computer time is still expensive – but cost is unavoidable
  - Software support, help with running and customisation is still required
  - Engineers running the code are the most costly part: **better!**
- Mode of operation
  - When a CFD works well in a design process, it will be used in large volume
  - Development and validation may need to be funded by user **but further cost drops significantly**: no annual license fee to pay
  - Parts of acceptance and validation effort become responsibility of the user
- In some cases, scaling up the computational power is essential to success, especially for complex physics. Example: SOFC fuel cell simulation

Parallelisation in CFD

Massively Parallel Computing for Industrial CFD

- Today, most large-scale CFD solvers rely on distributed-memory parallel computer architectures for all medium-sized and large simulations
- Parallelisation of CFD solvers is complete: if the algorithm does not parallelise, it is not used. Example: wall distance calculation
- Complete simulation work-flow is still not parallelised! Bottle-necks:
  - **Parallel mesh generation** is missing or under development
  - Scaling issues related to the linear algebra toolkit: solver technology
  - Parallel efficiency of “auxiliary algorithms” is sometimes very poor
- **User expectation**: linear scaling to thousands of CPUs

Parallel Computer Architecture

- Parallel CCM software operates almost exclusively in **domain decomposition mode**: a large loop (*e.g.* cell-face loop in the FVM solver) is split into bits and given to a separate CPU. Data dependency is handled explicitly by the software
- Using **distributed memory machines** with communications overhead; architecture dictates speed of communications and limits scalability
- Handling of multi-core processors sometimes questionable: memory access bandwidth is the limiting factor in serial execution speed

OpenFOAM: Executive Overview

What is OpenFOAM?

- **OpenFOAM** is a free-to-use Open Source numerical simulation software with extensive CFD and multi-physics capabilities
- Free-to-use means using the software without paying for license and support, including **massively parallel computers**: free 1000-CPU CFD license!
- Software under active development, capabilities mirror those of commercial CFD
- Substantial installed user base in industry, academia and research labs
- Possibility of extension to non-traditional, complex or coupled physics: Fluid-Structure Interaction, complex heat/mass transfer, internal combustion engines, nuclear

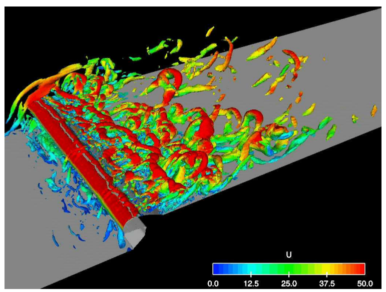
Main Components

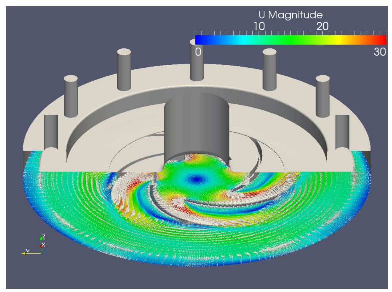
- Discretisation: Polyhedral Finite Volume Method, second order in space and time
- Lagrangian particle tracking, Finite Area Method (2-D FVM on curved surface)
- Massive parallelism in domain decomposition mode
- Automatic mesh motion (FEM), support for topological changes
- All components implemented in library form for easy re-use
- Physics model implementation through **equation mimicking**

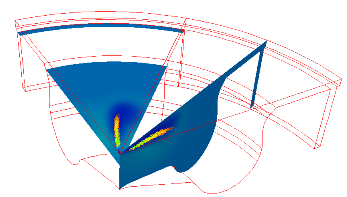
OpenFOAM: Capabilities Highlights

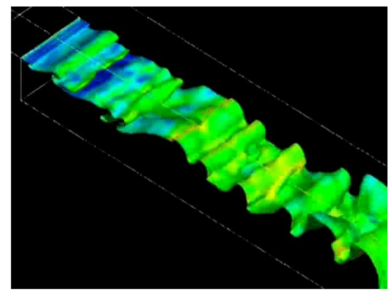
Physical Modelling Capability Highlights

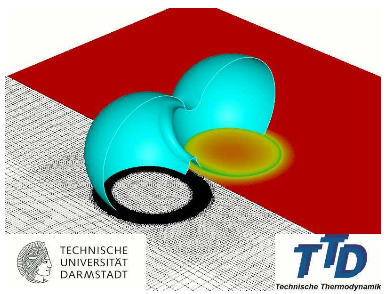
- Basic: Laplace, potential flow, passive scalar/vector/tensor transport
- Incompressible and compressible flow: segregated pressure-based algorithms
- Heat transfer: buoyancy-driven flows, conjugate heat transfer
- Multiphase: Euler-Euler, VOF free surface capturing and surface tracking
- RANS for turbulent flows: 2-equation, RSTM; full LES capability
- Pre-mixed and Diesel combustion, spray and in-cylinder flows
- Stress analysis, fluid-structure interaction, electromagnetics, MHD, etc.

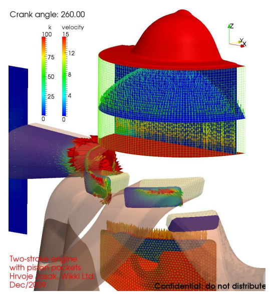


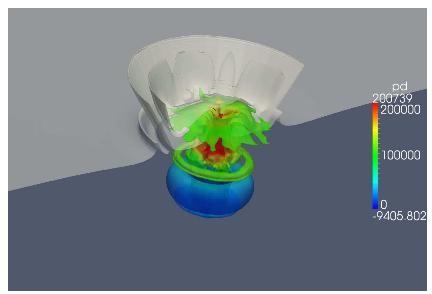


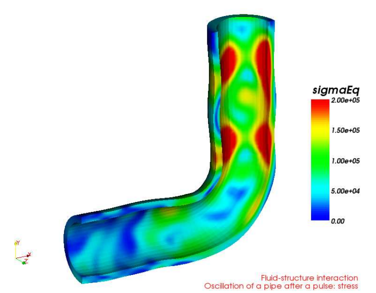












Components

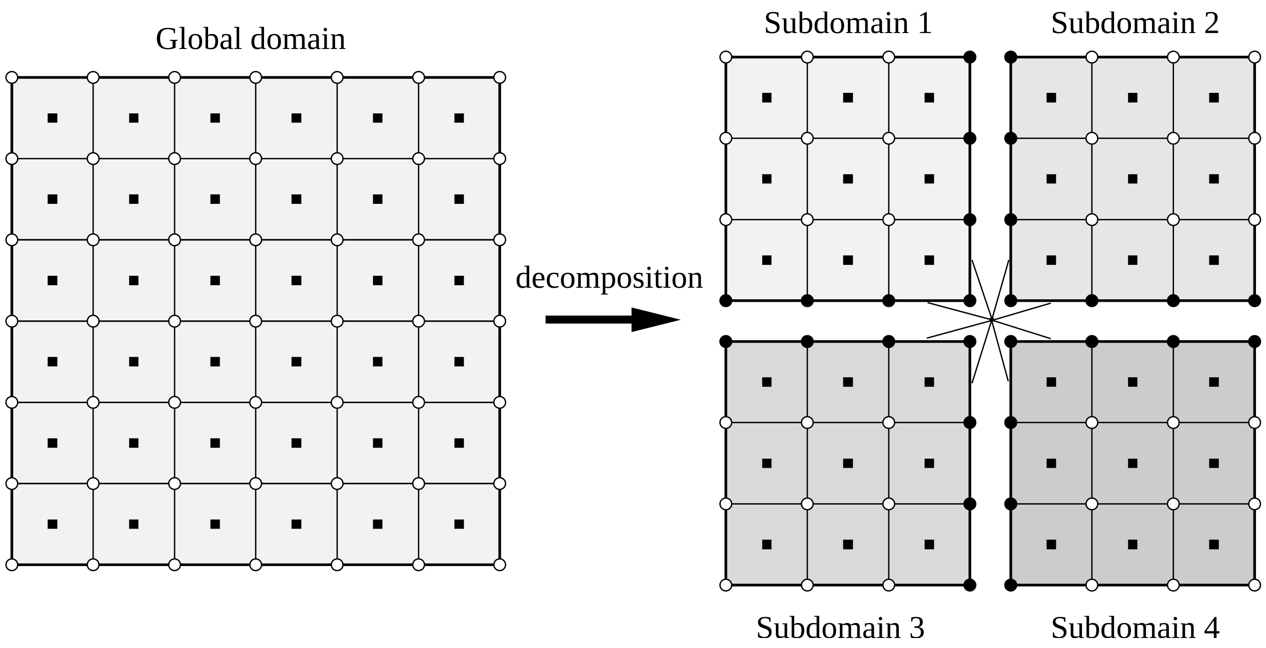
Parallel Components and Functionality

1. Parallel communication wrapper
   - Basic information about the run-time environment: serial or parallel execution, number of processors, process IDs etc.
   - Passing information in transparent and protocol-independent manner
   - Optimised global gather-scatter communication operations
2. Mesh-related operations
   - Mesh and data decomposition and reconstruction
   - Global mesh information, e.g. global mesh size, bounding box etc.
   - Handling patched pairwise communications
   - Processor topology communication scheduling data
3. Discretisation support
   - Operator and matrix assembly: executing discretisation operations in parallel
   - Data updates across processor boundaries: data consistency
4. Linear equation solver support (highest impact on solver performance!)
5. Auxiliary operations, e.g. messaging or algorithmic communications, non-field algorithms (e.g. particle tracking), data input-output, solution sampling and acquisition of (point, line, surface) post-processing data

Parallel Algorithms

Zero Halo Layer Approach in Discretisation

- Traditionally, FVM parallelisation uses the **halo layer** approach: data for cells next to a processor boundary is duplicated. Halo layer covers all processor boundaries and is explicitly updated through parallel communications calls: prescribed communications pattern, at pre-defined points
- OpenFOAM operates in **zero halo layer approach**: flexibility in communication pattern, separate setup for FVM and FEM solvers
- FVM and FEM operations "look parallel" without data dependency: perfect scaling



Linear Algebra Toolkit

Matrix Assembly and Solution

- Performance of linear solvers practically dictates parallel scaling in CFD
- In terms of code organisation, each sub-domain creates its own **numbering space**: locally, equation numbering always starts with zero and one cannot rely on global numbering: it breaks parallel efficiency
- Processor interfaces are updated separately, involving communications
- Explicit codes/operations scale well: no data dependency
- Implicit algorithms are $\approx$ 100 times faster but involve parallelised linear algebra

Choice of Linear Equation Solvers

- As a rule, **Krylov space solvers** parallelise naturally: global updates on scaling and residual combined with local vector-matrix operations: global sum
- **Algebraic Multigrid** (AMG) performs much worse due to coarse level hierarchy: balance of work and communications at coarse levels
- . . . but AMG is intrinsically 3 times faster in the number of operations
- Currently, all algorithms assume uniform communications performance across the machine: improvement is needed but may require complete algorithmic rewrite
- **Outlook**: an (unknown) new approach is needed to achieve performance scaling

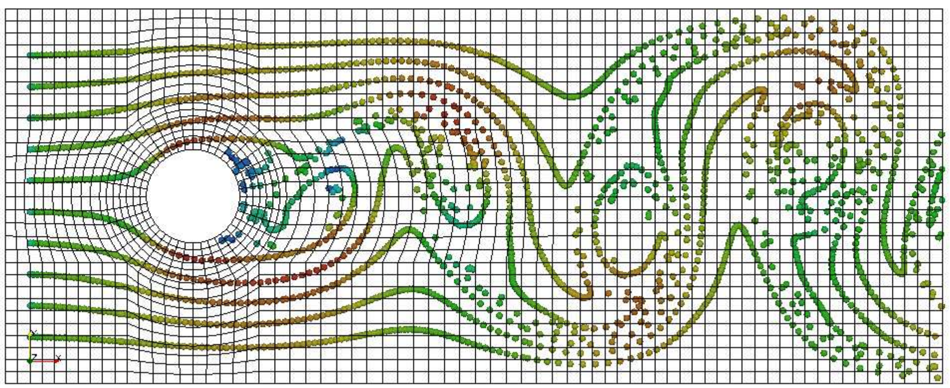
Auxiliary Algorithms

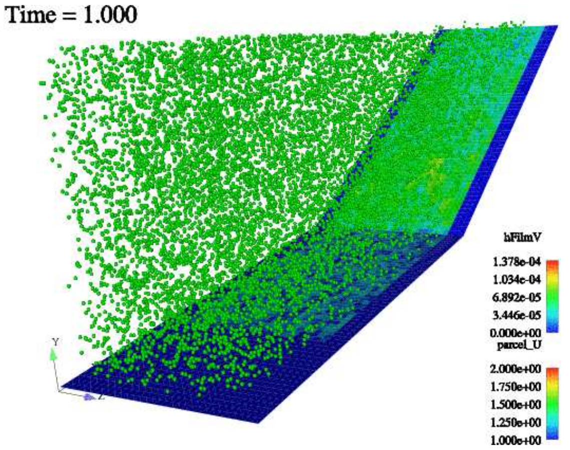
Efficiency in Auxiliary Algorithms

- Overall, parallelisation efficiency is satisfactory on low 000s of CPUs for field-based operations: explicit field algebra, matrix assembly and linear solvers
- Parallelisation of some components is bound to domain decomposition but operations are not field-based: surface physics, particle tracking, patch integrals
- Massive load imbalance or inappropriate parallelisation limits performance

Example: Lagrangian Particle Tracking in Parallel

- Particle tracking algorithm operates on each segment of decomposed mesh by tracking local particles. Load balancing issues: **particles are not fields!**
- Processor boundary interaction: a particle is migrated to connecting processor





Points of Interest

Current Status: Performance Scaling

- Implicit general purpose CFD solvers work reasonably well up to low 000s of CPUs
- Current approach is inappropriate for inhomogeneous communication speed and an order of magnitude increase in computer size
- Communication speed and memory access are critical: clusters must be co-located with fast inter-connect; moving jobs around is not practical
- Limiting factors in performance scale-up:
  - Parallelisation of complete CFD process, from geometry to post-processing
  - Iterative sparse matrix solver algorithms
  - Non-field operations in CFD algorithms: spray, radiation, surface physics

Summary

- Current CFD technology involving implicit solvers is close to its limit
- Some necessary algorithms are badly parallelised due to requirement of method-to-method coupling
- New approach is needed: how do we do massively parallel linear algebra?
- We may remain at the level of 000s of CPUs for a long time, looking for other avenues of improvement: multi-core CPU, handling inhomogeneous communications speed, fast memory access, complete migration to GPU