


# A dynamic load-balancing strategy for large scale CFD-applications

Philipp Offenhäuser H L R I S 

10.10.2017



# Outline

Motivation and Background

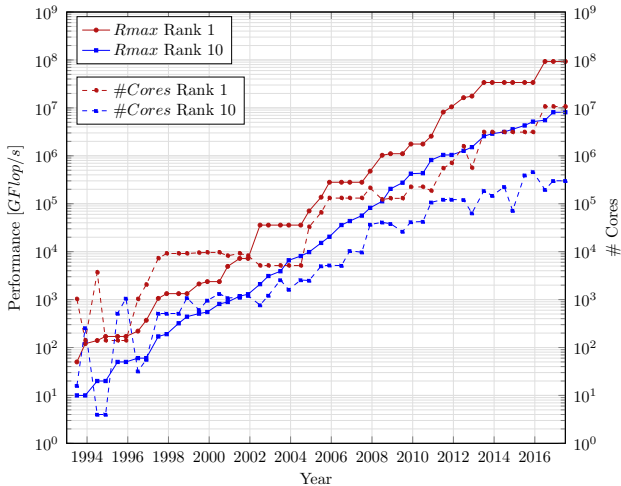
Load-Balancing strategy

Results

Further Challenges

Conclusion

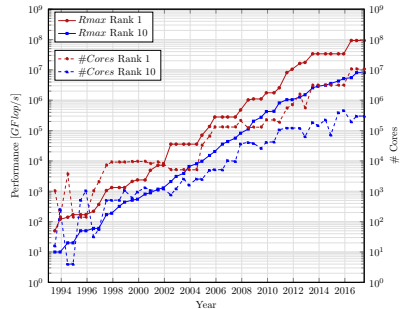
## Motivation - Top500 from 1993 to 2017



(Source: top500.org)

## Motivation

- ▶ HPC systems generate their power by facilitating hundreds of thousands of cores
- ▶ Massive parallel algorithms are implemented
- ▶ Computational effort must be distributed evenly across all cores
- ▶ Initial domain decomposition techniques are well known
- ▶ Well balanced applications for scientific use-cases are developed



(Source: top500.org)

## Definitions

### **Element-weight:**

Specific computational effort of an element.

## Definitions

### **Element-weight:**

Specific computational effort of an element.

### **Load of a MPI-domain:**

Sum of the element-weights of a MPI-domain.

## Definitions

### Element-weight:

Specific computational effort of an element.

### Load of a MPI-domain:

Sum of the element-weights of a MPI-domain.

### Load-imbalance:

$$Load-imbalance_L = \frac{\text{maximum load}}{\text{average load}}$$

$$Load-imbalance_T = \frac{\text{maximum computation time}}{\text{average computation time}}$$

## Example 1 - Finite-Volume-Code FS3D

### References:

Institute of Aerospace Thermodynamics - University of Stuttgart

<http://www.uni-stuttgart.de/itlr/forschung/tropfen/fs3d/index.php?lang=en&lang=en:w>



## Examples for load-imbalance - Volume of Fluid Method

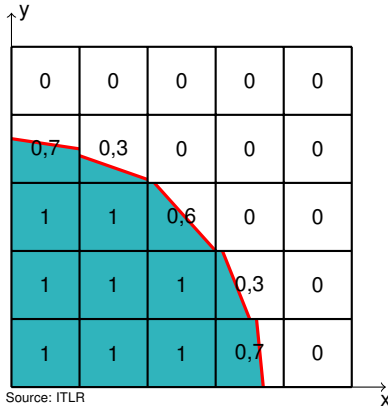


### Treatment of multiple phases

$$f(x, t) = \begin{cases} 0 & \text{liquid} \\ (0; 1) & \text{interface} \\ 1 & \text{solid} \end{cases}$$

- ▶ Reconstruction of the interface
- ▶ Additional computational effort for the interface elements

## Examples for load-imbalance - Volume of Fluid Method



### Treatment of multiple phases

$$f(x, t) = \begin{cases} 0 & \text{liquid} \\ (0; 1) & \text{interface} \\ 1 & \text{solid} \end{cases}$$

- ▶ Reconstruction of the interface
- ▶ Additional computational effort for the interface elements

## Example 2 - Discontinuous Galerkin Spectral Element Method (DG SEM) based code FLEXI

### References:

Institute of Aerodynamics and Gas Dynamics - University of Stuttgart  
Numerical research group <https://nrg.iag.uni-stuttgart.de/>

FLEXI on github: <https://github.com/flexi-framework/flexi>

## FLEXI - a high-order numerical framework

- ▶ High-order numerical approach
- ▶ Massively parallel CFD-code designed for HPC-systems
- ▶ DG SEM enables efficient communication-pattern (communication hiding)
- ▶ FLEXI tested on different HPC-systems: Hornet, HazelHen, JUQueen, Marconi
- ▶ FLEXI is used for very large academic use-cases

## Example - gas injection

- ▶ Numerical approach was extended to simulate complex fluid-flow
- ▶ The FV-Sub-Cell-Method is used for shock capturing
- ▶ Code is used for industrial use cases
- ▶ Depending on the local solution the FV-Sub-Cell-Method is turned on/off
- ▶ FV-Sub-Cell-Method adds local computational effort

....



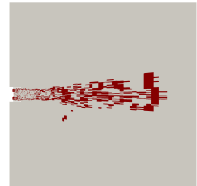
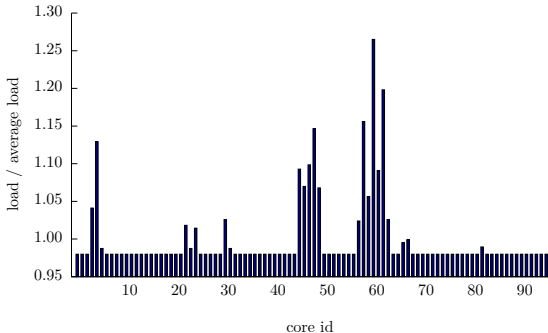
## Local computational effort

$T = 0,00 \text{ ms}$

$T = 0,25 \text{ ms}$

$T = 0,50 \text{ ms}$

## Motivation for dynamic load-balancing



Load distribution of a test-case on 96 cores.

- ▶ Small number of cores are over-loaded
- ▶ Huge number of cores are under-loaded
- ▶ Performance of the application suffers from the overload on a small number of cores

## A dynamic load-balancing strategy

**Step 1: Calculate a new optimized load distribution**

**Step 2: Distribute the load between MPI-processes**



## A dynamic load-balancing strategy

**Step 1: Calculate a new optimized load distribution**

**Step 2: Distribute the load between MPI-processes**

Challenges:

- ▶ Global optimization problem
- ▶ Simple and fast calculation of the optimized load distribution
- ▶ Keep the communication overhead small

## A dynamic load-balancing strategy

### Step 1: Calculate a new optimized load distribution

#### Challenges:

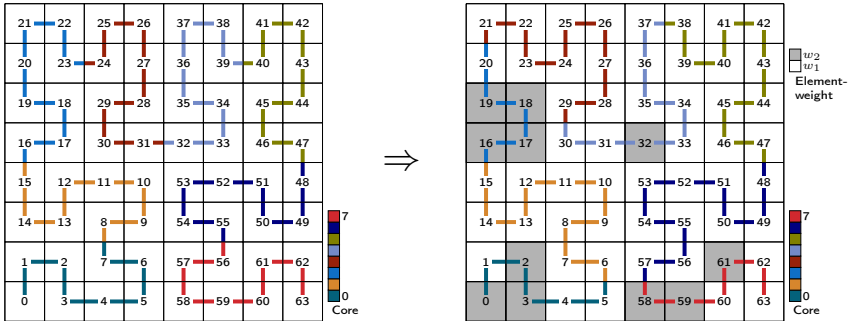
- ▶ Global optimization problem
- ▶ Simple and fast calculation of the optimized load distribution
- ▶ Keep the communication overhead small

### Step 2: Distribute the load between MPI-processes

#### Challenges:

- ▶ Communication-structure originates during run-time
- ▶ Communication-structure changes during run-time
- ▶ Keep the communication overhead small

## A dynamic load-balancing strategy - approach

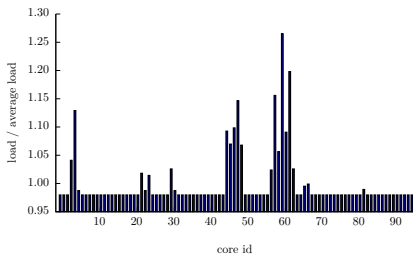


- ▶ Using a space-filling-curve for the domain-decomposition
- ▶ Mapping the 3D-problem to a 1D-problem
- ▶ Minimal communication (only one time)
- ▶ Simple communication pattern

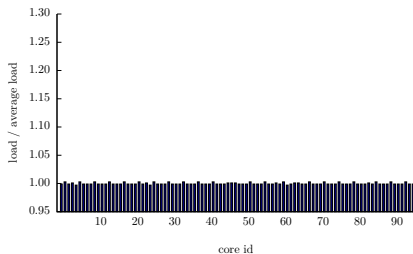
# 1. Calculate a new optimized load distribution

An optimization algorithm based on the element specific computational effort is used for the load-distribution.

$$\Delta c_{avg}^{P_n} = \min(|\Delta c_{avg}^{P_{n-1}} + \Delta_1^{P_n}|, |\Delta c_{avg}^{P_{n-1}} + \Delta_2^{P_n}|)$$



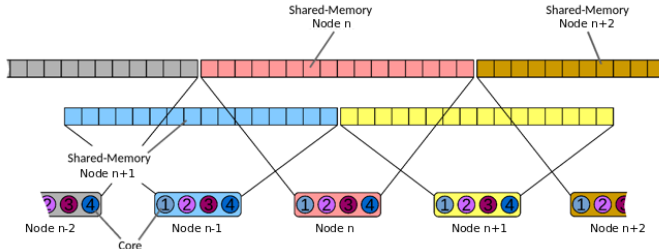
Load distribution without load-balancing



Load distribution with load-balancing

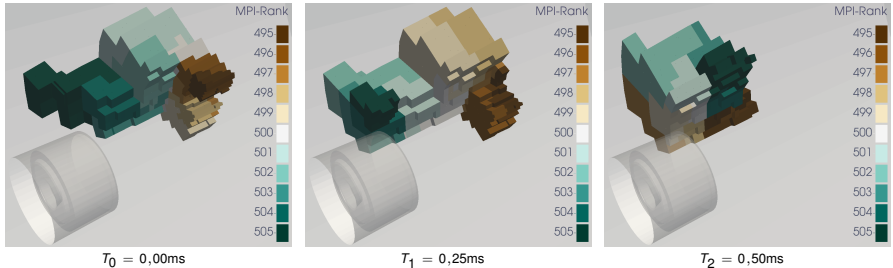
Load distribution of a test-case on 96 cores.

## Step 2: Distributing the load between MPI-processes



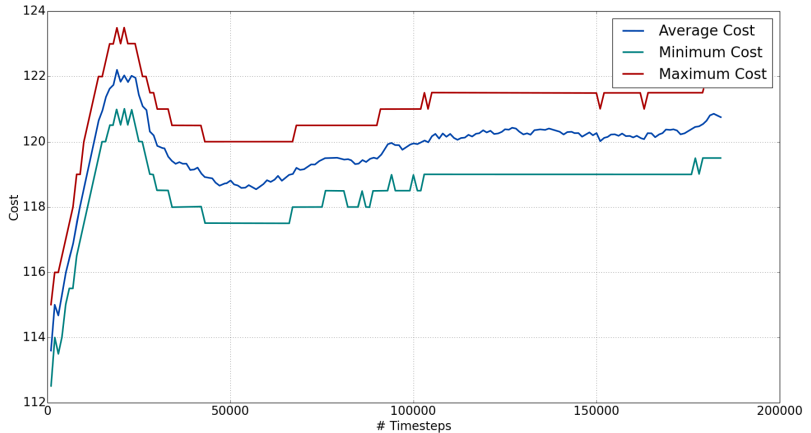
- ▶ Using again the 1D-structure of the space-filling-curve
- ▶ Intra-node-communication via MPI-Shared-Memory-Window
- ▶ Shared-Memory used for communication and to store the results during the load-balancing
- ▶ MPI-communication only between nodes

## Results - Element distribution



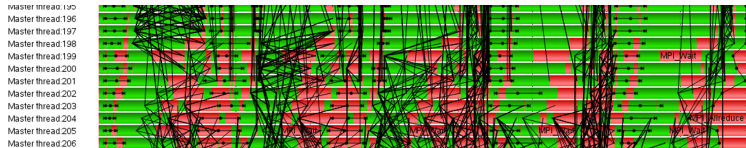
- ▶ Element distribution depends on the computational effort
- ▶ Redistribution of the elements every 1,000 timesteps
- ▶ 10% reduction of the wall-time

## Dynamic load distribution





## Further Challenges - detection of load-imbalance at run-time



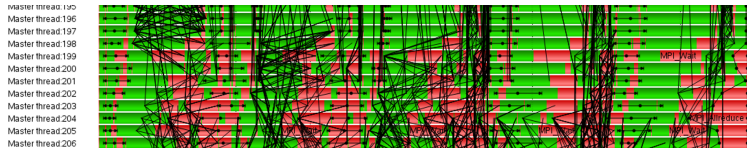
### State of the art:

- ▶ Different powerful performance measuring tools are available
- ▶ All tools need an instrumentation and produce overhead
- ▶ Results of the measurement only available after simulation
- ▶ No feedback loop to the application





## Further Challenges - detection of load-imbalance at run-time



### State of the art:

- ▶ Different powerful performance measuring tools are available
- ▶ All tools need an instrumentation and produce overhead
- ▶ Results of the measurement only available after simulation
- ▶ No feedback loop to the application

### Challenges:

- ▶ Measurement has to be integrated in the application
- ▶ Feedback loop to the application
- ▶ Measurement of the load-imbalance with a minimal overhead

## Conclusion

- ▶ HPC systems generate their power by facilitating hundreds of thousands of cores
  - ▶ Massively parallel CFD-codes are implemented
  - ▶ Initial domain decomposition techniques are well know
  - ▶ Well balanced CFD-applications
- ▶ Simulation of complex fluid flow
  - ▶ varying load distribution during run-time
  - ▶ occurrence of the load distribution hard to predict spatially and chronologically
  - ▶ simulation specific load-imbalance occurs
- ▶ For efficient application execution dynamic load-balance strategies are indispensable
  - ▶ Efficient redistribution of the computational load
  - ▶ Detection of load-imbalance with a very low overhead

Thank you for your attention!

