TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing – TU Dresden

Efficient Clustering and Scheduling for Task-Graph based Parallelization

Marc Hartung

02. February 2015

E-Mail: marc.hartung@tu-dresden.de

Center for Information Services & High Performance Computing

HPC-OM

HPCOM

www.hpc-om.de

Rexroth
Bosch Group

TECHNISCHE UNIVERSITÄT DRESDEN

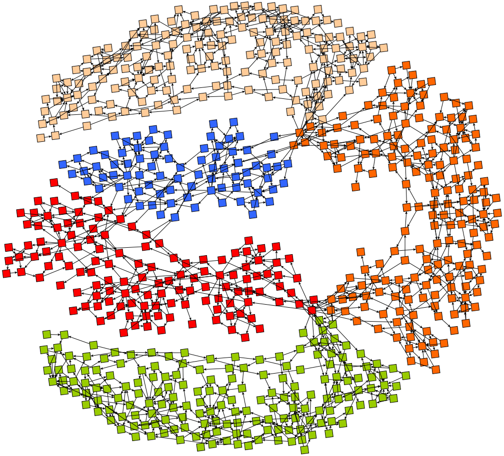
Supporting your visions

SPONSORED BY THE

Federal Ministry of Education and Research

Content

1. Motivation
2. Scheduling
3. TGSim - Framework
4. Results

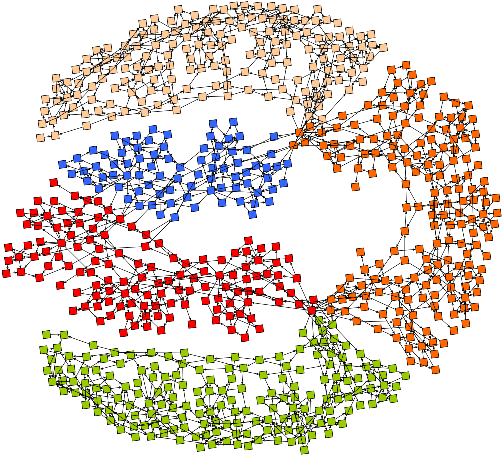


TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

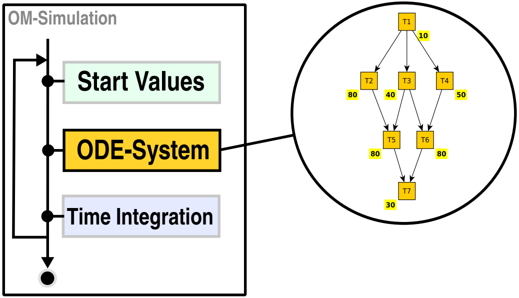
Outline

1. Motivation
2. Scheduling
3. TGSim - Framework
4. Results



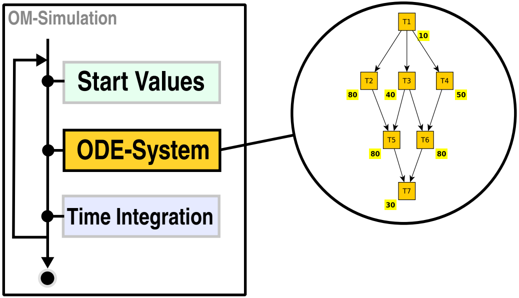
Motivation

- Achieve speed-up through parallel execution of the ODE-system's tasks



Motivation

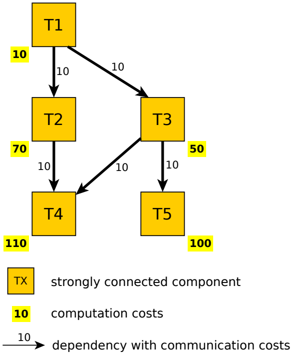
- Achieve speed-up through parallel execution of the ODE-system's tasks



- Assigning tasks to more than one CPU reduces simulation time
- Improvement depends on model

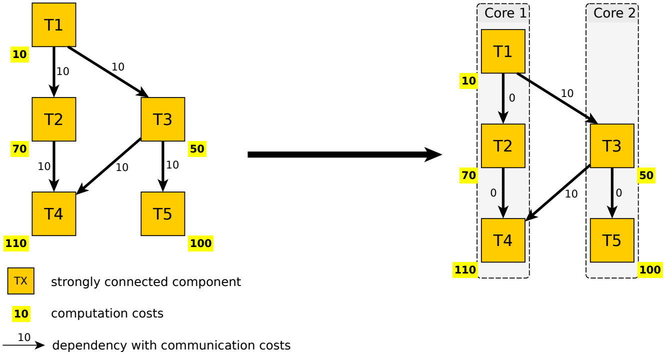
Motivation

- Task Graph visualizes right-hand side evaluation
- Contains computation costs, dependencies and communication costs



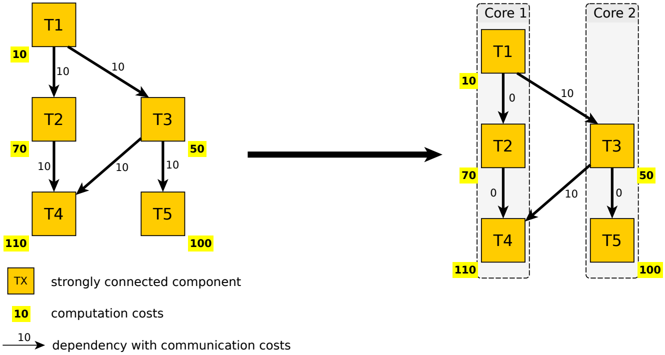
Motivation

- Task Graph visualizes right-hand side evaluation
- Contains computation costs, dependencies and communication costs



Motivation

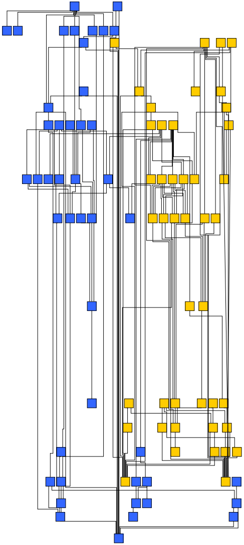
- Task Graph visualizes right-hand side evaluation
- Contains computation costs, dependencies and communication costs



Algorithms for task-to-core mapping and ordering are needed!

Motivation

- Task Graph based Parallelization
  - + Heterogeneous data dependencies
  - + Allows nested parallelism
  - + Numerical stable
  - + Universal parallel solution (in theory)

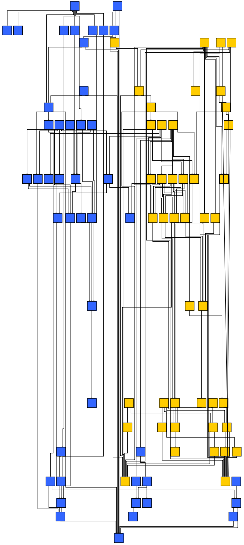


TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Motivation

- Task Graph based Parallelization
  - + Heterogeneous data dependencies
  - + Allows nested parallelism
  - + Numerical stable
  - + Universal parallel solution (in theory)
- Obstacles
  - – Compile time
  - – Parallel efficiency
  - – Model dependent

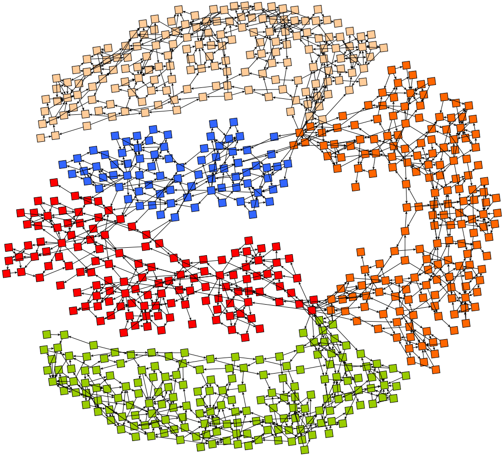


TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Outline

1. Motivation
2. Scheduling
3. TGSim - Framework
4. Results



Scheduling

- Scheduling is a NP-complete decision problem
- Many greedy algorithms available
- Complexity between $\mathcal{O}(n)$ and $\mathcal{O}(n^4)$ (n ... number of tasks)

Scheduling

- Scheduling is a NP-complete decision problem
- Many greedy algorithms available
- Complexity between $\mathcal{O}(n)$ and $\mathcal{O}(n^4)$ (n ... number of tasks)

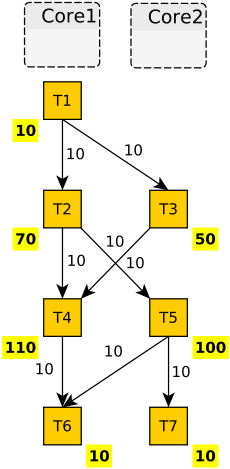
- Low cost algorithms achieve usable solutions
- But: No speed up guaranty

Scheduling - Taxonomy

Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

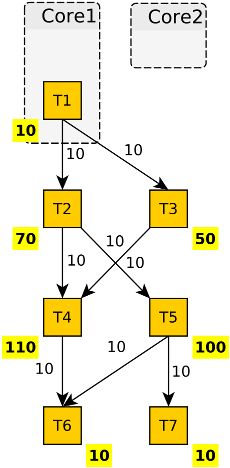
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$ (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

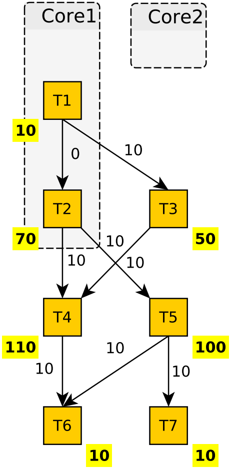
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$ (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$
  (p ... number of cores)



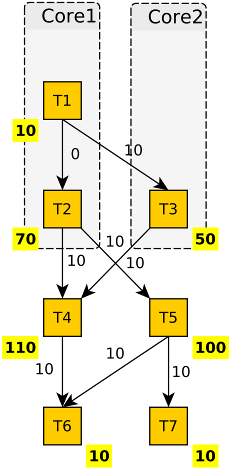
TECHNISCHE UNIVERSITÄT DRESDEN

ZIH
Center for Information Services & High Performance Computing

Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

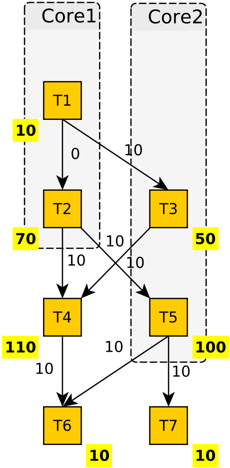
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$ (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

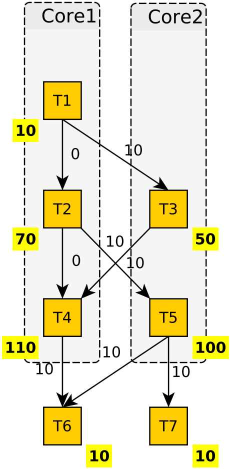
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$
  (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

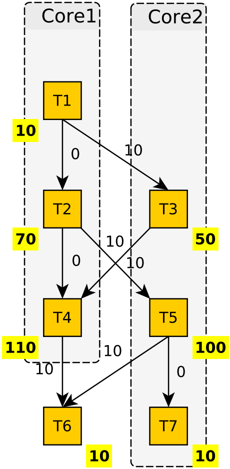
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$
  (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

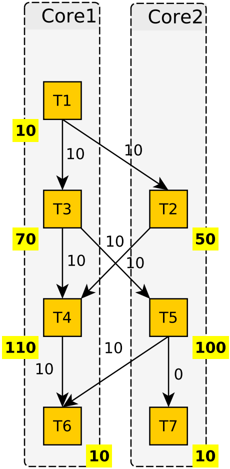
- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$
  (p ... number of cores)



Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$ (p ... number of cores)



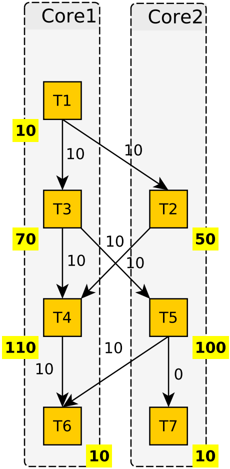
Scheduling - ETF

Earliest **T**ime **F**irst - Algorithm

- List scheduler (for bounded number of processors)
- Checks every ready task for earliest start time
- Draws solved by highest bottom level
- Complexity: $\mathcal{O}(p \cdot n^2)$
  (p ... number of cores)

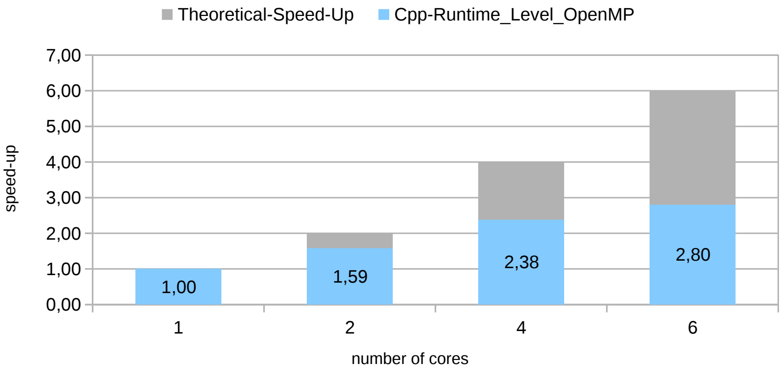
Other list scheduler:

- LVL ... Level Scheduler
- DLS ... Dynamical Level Scheduling
- MCP ... Modified Critical Path



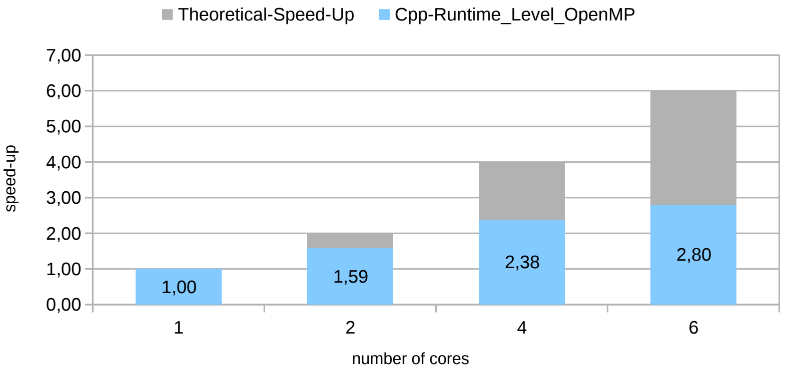
Scheduling - Status

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel i7-3930K 6x 3.20 GHz, Linux



Scheduling - Status

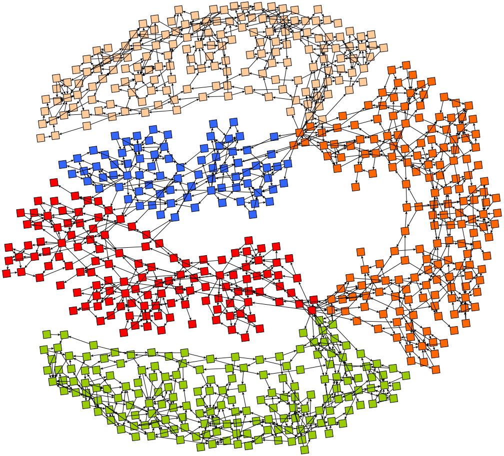
Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel i7-3930K 6x 3.20 GHz, Linux



- Approach: Analyse scheduler and parallelization methods to close gaps

Outline

1. Motivation
2. Scheduling
3. TGSim - Framework
4. Results



TGSim-Framework

Task **G**raph **Sim**ulation Framework

- Analyse and evaluate scheduling and clustering algorithms
- Benchmark different parallelization methods
- Parallel runtime prediction for OM simulations and other traceable programs

TGSim-Framework

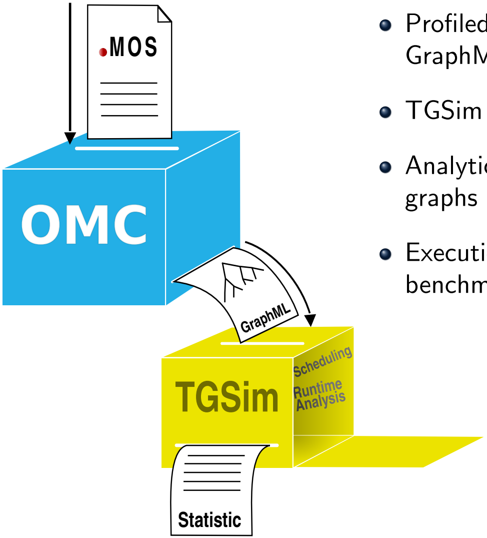
Task **G**raph **Sim**ulation Framework

- Analyse and evaluate scheduling and clustering algorithms
- Benchmark different parallelization methods
- Parallel runtime prediction for OM simulations and other traceable programs

Implementation:

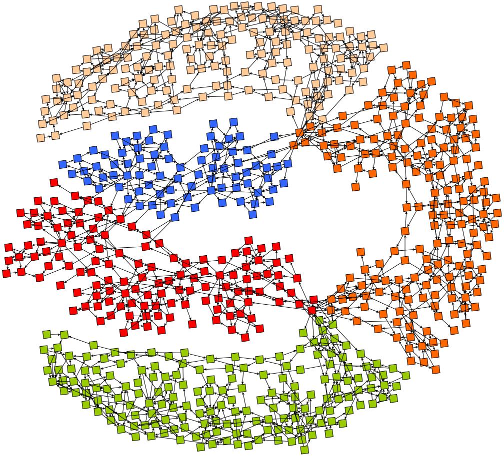
- Written in C++ using OOP
- Easy to expand and user-friendly
- Creates OM-simulation alike programs with low overhead tracing mechanisms
- ODE-tasks replaced by wait tasks to reduce unintended influences

TGSim workflow



- Profiled CppRuntime-simulation creates GraphML-file
- TGSim uses GraphML-File as input
- Analytical evaluation of scheduled task graphs
- Execution of scheduled simulations to benchmark parallel methods

Outline

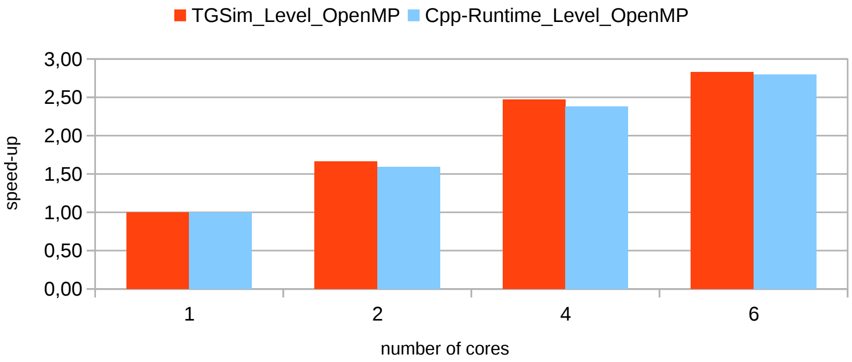


Results - Outline

1. TGSim runtime simulation vs. OM-Cpp-Runtime simulation
   - Comparable?
2. Compare parallelization methods
   - Dynamic scheduling and static scheduling
3. Compare TGSim scheduler
   - Scheduling algorithms: MCP, DLS, ETF, LVL

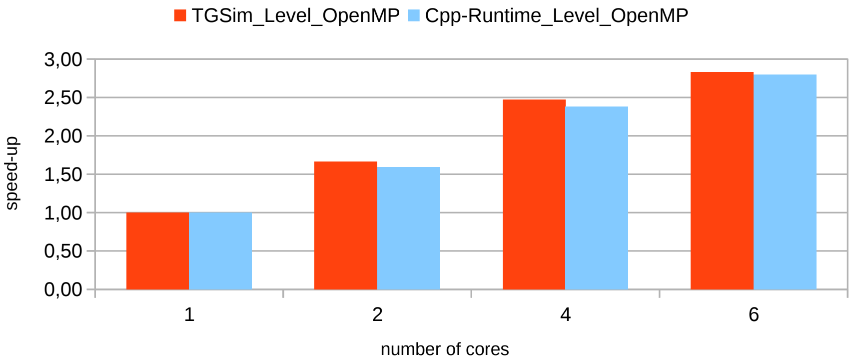
Results - TGSim vs. Cpp-Runtime

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



Results - TGSim vs. Cpp-Runtime

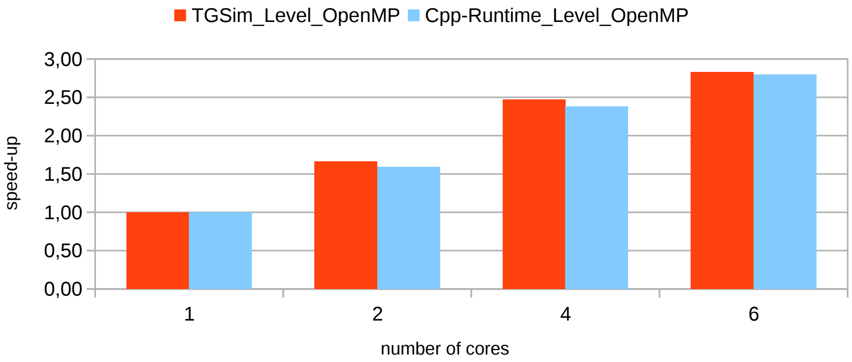
Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



- TGSim simulates OM-simulation work flow very well

Results - TGSim vs. Cpp-Runtime

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



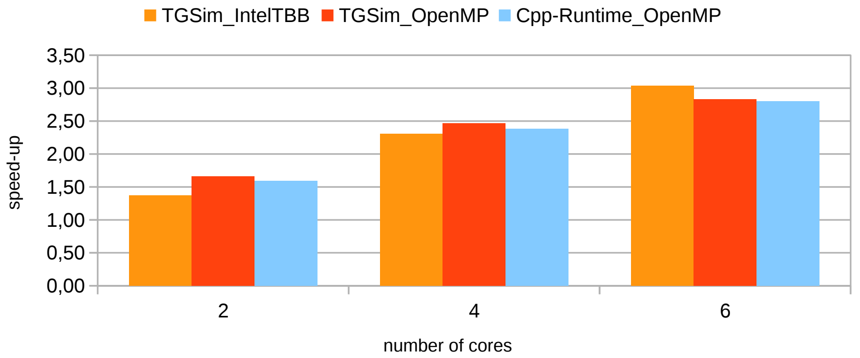
- TGSim simulates OM-simulation work flow very well
- To simplify comparing, the OpenMP-OM-Cpp-Runtime results will be in every diagram

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Results - Benchmark Dynamic Methods

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



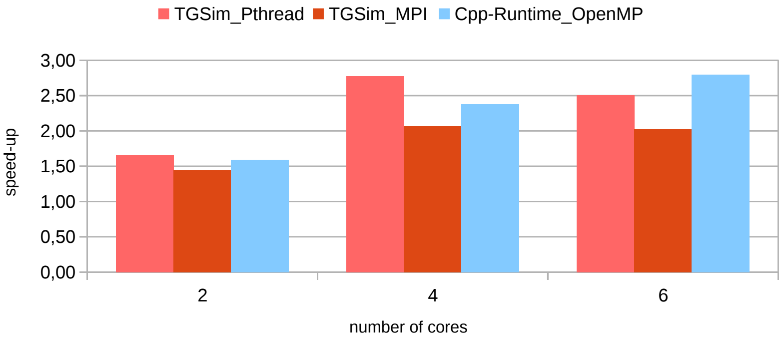
- IntelTBB is comparable to OpenMP
- Small disadvantage: Initialization time of IntelTBB ist 3700μs and of OpenMP 4μs

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH Center for Information Services & High Performance Computing

Results - Benchmark Static Methods

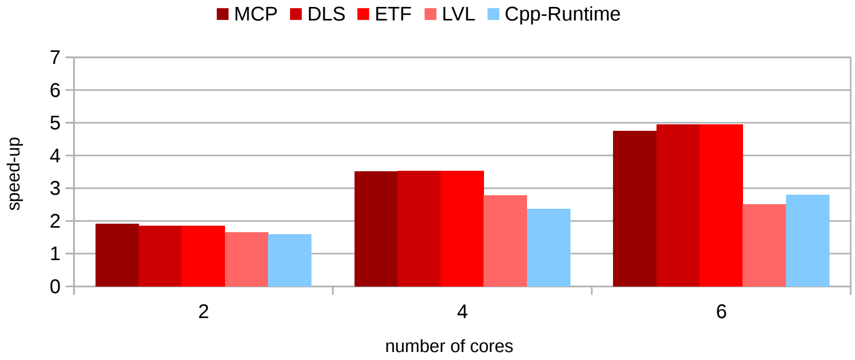
Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



- PThread Performance in the first test cases better
- Static parallelization depends on scheduling

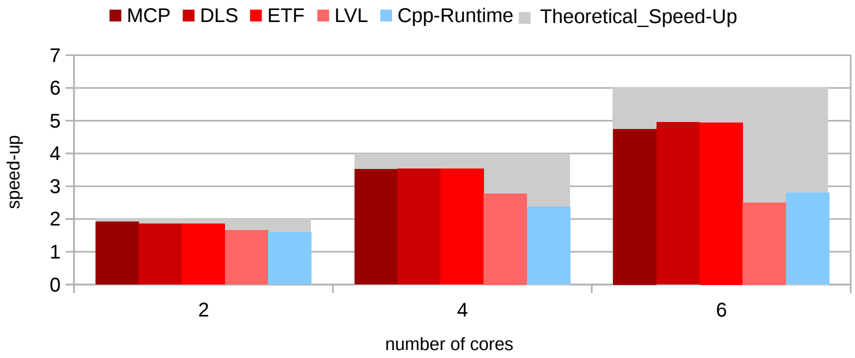
Results - Benchmark Scheduling

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



Results - Benchmark Scheduling

Model: Modelica.Fluid.Examples.BranchingDynamicPipes
System: Intel Xeon E5-2690 8x 2.90GHz, Linux



- Proper scheduling leads to high improvements

Summary & Future Work

Summary

- With increasing number of cores static scheduling performs better than dynamic
- Scheduler which consider communication costs comparable in performance and much better than other
- PThreads fastest parallelization method, OpenMP and IntelTBB comparable

Summary & Future Work

Summary

- With increasing number of cores static scheduling performs better than dynamic
- Scheduler which consider communication costs comparable in performance and much better than other
- PThreads fastest parallelization method, OpenMP and IntelTBB comparable

Future Work

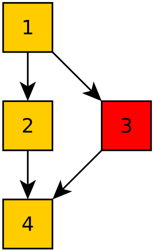
- Extend HPCOM OpenModelica library including TGSim optimizations

Thank you for your attention.

E-Mail: marc.hartung@tu-dresden.de

Test Framework TGSim II

MPI example




```
// core 1

//task 1
wait(costs1);
MPI\_Isend(data1,...,core2,...);

//task 2
wait(costs2);

//task4
MPI\_Irecv(data3,...,core2,...);
wait(costs4);
```

```
// core 2

//task 3

MPIIrecv(data1,...,core1,...);


wait(costs3);


MPIsend(data3,...,core1,...);
```