Efficient Clustering and Scheduling for Task-Graph based Parallelization

Marc Hartung

02. February 2015

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

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

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

![Task Graph Diagram]

Computation costs:
- T1: 10
- T2: 70
- T3: 50
- T4: 110
- T5: 100

Dependency with communication costs:
- TX strongly connected component

Computation costs:
- 10

Dependency with communication costs:
- 10
**Motivation**

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

![Task Graph](image)

- TX: strongly connected component
- 10: computation costs
- 10: dependency with communication costs

Algorithms for task-to-core mapping and ordering are needed!
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)
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
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 $O(n)$ and $O(n^4)$ ($n$ ... number of tasks)
- Low cost algorithms achieve usable solutions
- But: No speed up guaranty
**Scheduling - ETF**

**Earliest Time First - Algorithm**

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

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

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

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

**Earliest Time First - Algorithm**

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

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

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

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

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

Other list scheduler:
- LVL ... Level Scheduler
- DLS ... Dynamical Level Scheduling
- MCP ... Modified Critical Path
System: Intel i7-3930K 6x 3.20 GHz, Linux
Scheduling - Status

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
Task Graph Simulation Framework

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

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

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

![Bar chart showing speed-up comparison between TGSim and Cpp-Runtime]

- TGSim-Level_OpenMP
- Cpp-Runtime-Level_OpenMP
Results - TGSim vs. Cpp-Runtime

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.
Results - TGSim vs. Cpp-Runtime

System: Intel Xeon E5-2690 8x 2.90GHz, Linux

- TGSim simulates OM-simulation work flow very well
- To simplify comparing, the OpenMP-OM-Cpp-Run time results will be in every diagram
Results - Benchmark Dynamic Methods

System: Intel Xeon E5-2690 8x 2.90GHz, Linux

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

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

System: Intel Xeon E5-2690 8× 2.90GHz, Linux

Proper scheduling leads to high improvements
Results - Benchmark Scheduling

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

Future Work

Extend HPCOM OpenModelica library including TGSim optimizations
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
MPI example

1 // core 1
2
3 // task 1
4 wait(costs1);
5 MPI_Isend(data1,...,core2,...);
6
7 // task 2
8 wait(costs2);
9
10 // task 4
11 MPI_Irecv(data3,...,core2,...);
12 wait(costs4);

1 // core 2
2
3 // task 3
4
5 MPI_Irecv(data1,...,core1,...);
6
7
8 wait(costs3);
9
10
11 MPIsend(data3,...,core1,...);