Distributed parallel Qsim

Janek Laudan, Paul Heinrich, Kai Nagel

Link to slides

Structure

1. Problem
2. Idea
3. Implementation
4. Results
5. Discussion

Problem

Free lunch is over

We are in the post Moore era, where hardware miniaturization reaches limits

CPU-Clock Speed is not improving anymore

Performance improvements through multicore CPUs and Accelerators

Performance improvements must come from parallelization

DOI: 10.1126/science.aam9744

Post Moore Era

Of the three main phases, Scoring and Replanning are straightforward to parallelize

Parallelizing the Mobsim is not trivial

Current default implementation QSim scales up to eight processes

The current strategy to parallelize QSim is a shared data approach, using Java's Concurrency primitives

Idea

RustQSim

From shared memory to message passing based implementation

Solve difficult parallelization problem first: The Mobsim

Use domain decomposition to divide simulation into subdomains

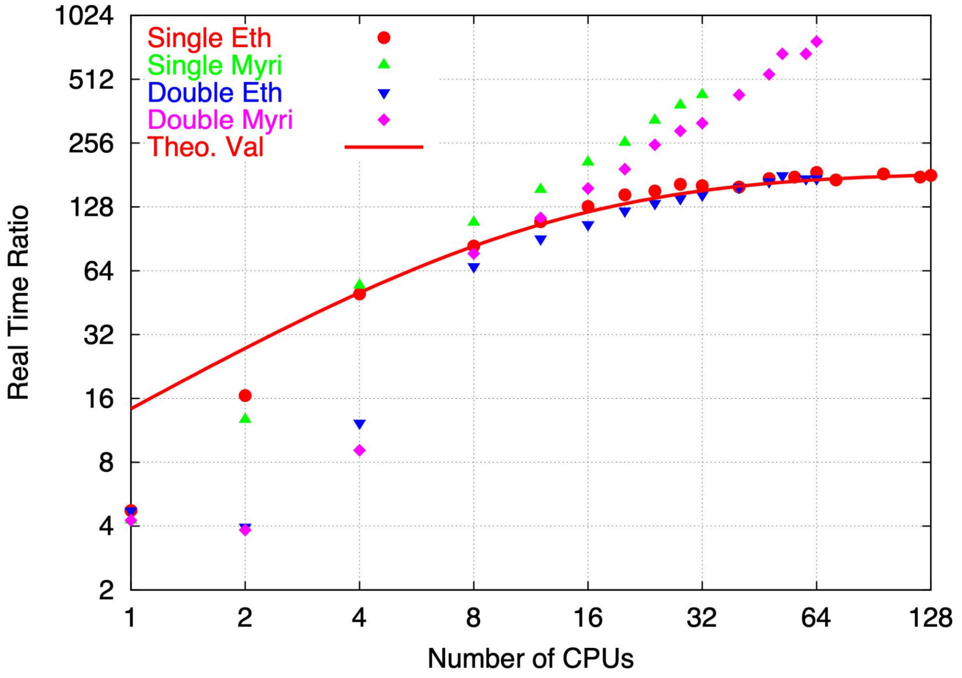
Use message passing to exchange information between subdomains

Previous Work

Cetin, Burri, Nagel implemented a distributed QSim in 2003

Key Findings:

- Domain Decomposition with Message Passing works
- Algorithm is bound by latency of message exchange



https://api.core.ac.uk/oai/oai:CiteSeerX.psu:10.1.1.87.4169

Latency in Computing

Latency, describes the delay between sending and receiving a message

When executing the simulation on multiple computers, messages are exchanged over the network

Most common network is Ethernet with a latency of 100 - 500µs

High-Performance Hardware such as Infiniband/OmniPath have latencies of 1µs

Implementation

Attempt 1 - Java

Implement the DiQSim in Java

- Support High-Performance Hardware
- Use **M**essage **P**assing **I**nterface (MPI) for communication
    - OpenMPI has Java-Bindings
- Enhance the current implementation

Did not work out

Running the JVM in an MPI-context results in random SIGSEV

- JVM uses SIGSEV internally for Garbage Collection
- UCX uses SIGSEV for internal singalling

```
#
# A fatal error has been detected by the Java
Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x000146010801e74,
pid=8652, tid=8654
#
Stack: [0x00001460295eb000,0x00001460296ec000],
sp=0x00001460296e7b60,  free space=1010k
Native frames: (J=compiled Java code, A=aot
compiled Java code, j=interpreted, Vv=VM code,
C=native code)
J 513 c2
java.lang.StringBuilder.append(Ljava/lang/String;)
Ljava/lang/StringBuilder;
```

Attempt 2 - Rust

Compatible to C-Binaries

- Rich Foreign Function Interface
- Direct access to MPI-Implementation
- MPI wrapper library available (https://github.com/rsmpi/rsmpi)

Language Features

- Cargo build tool
- Memory Management without Garbage Collection
- Ownership model lends itself to modelling physical objects
- High-Level Programming in general, but “bare-metal” optimizations possible

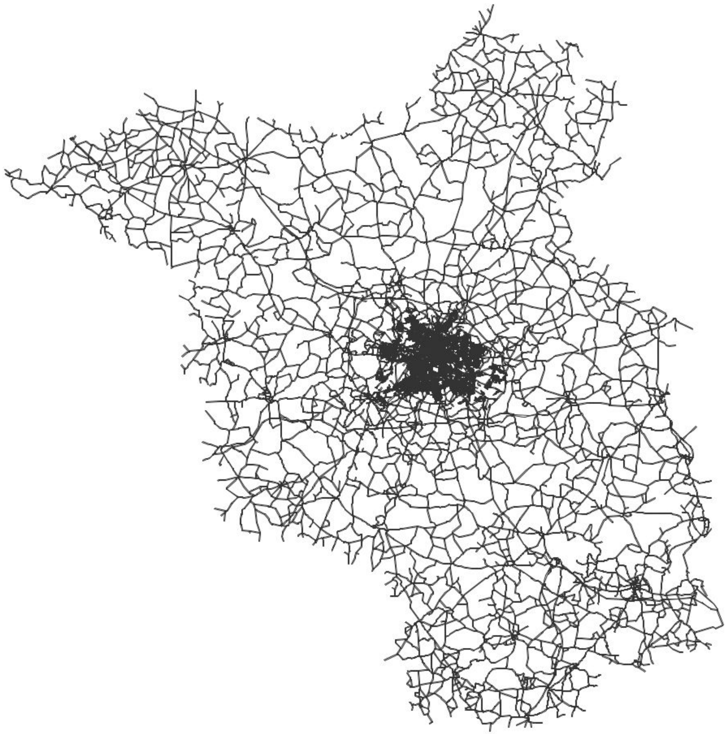
Domain Decomposition

Distribute Workload geographically

Domain Decomposition is a well-understood problem with plenty of algorithms, e.g., METIS

Estimate computational load for each vertice of the graph by estimating #vehicles crossing the node

METIS balances the computational load across partitions



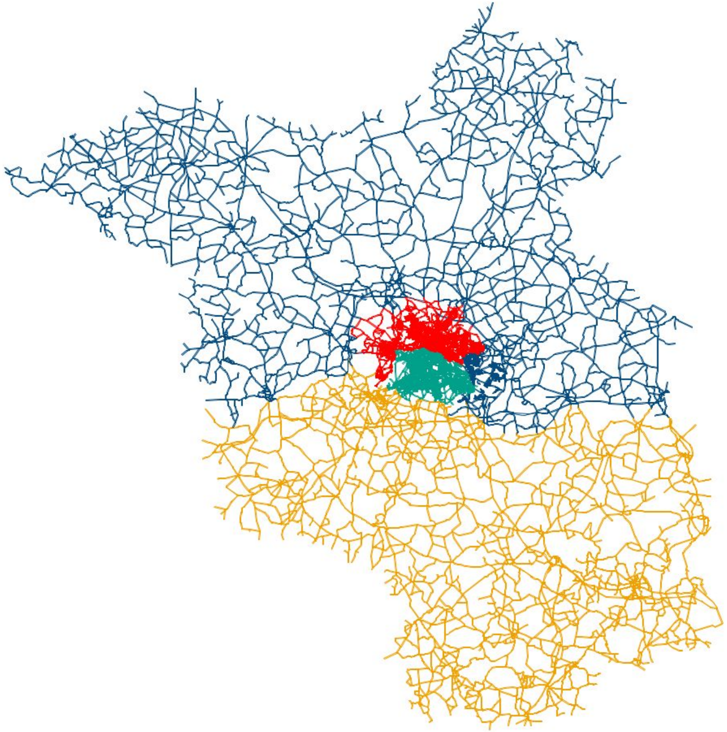
Domain Decomposition

Distribute Workload geographically

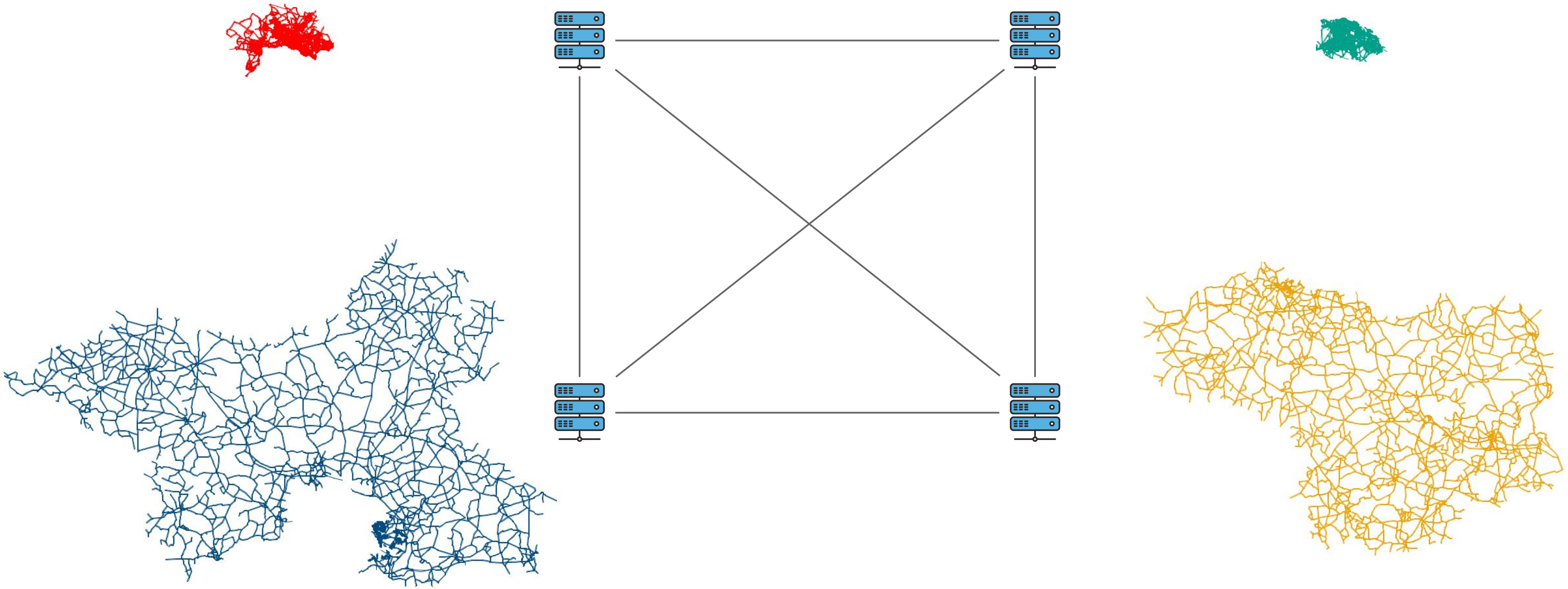
Domain Decomposition is a well-understood problem with plenty of algorithms, e.g., METIS

Estimate computational load for each vertice of the graph by estimating #vehicles crossing the node

METIS balances the computational load across partitions



Domain Decomposition

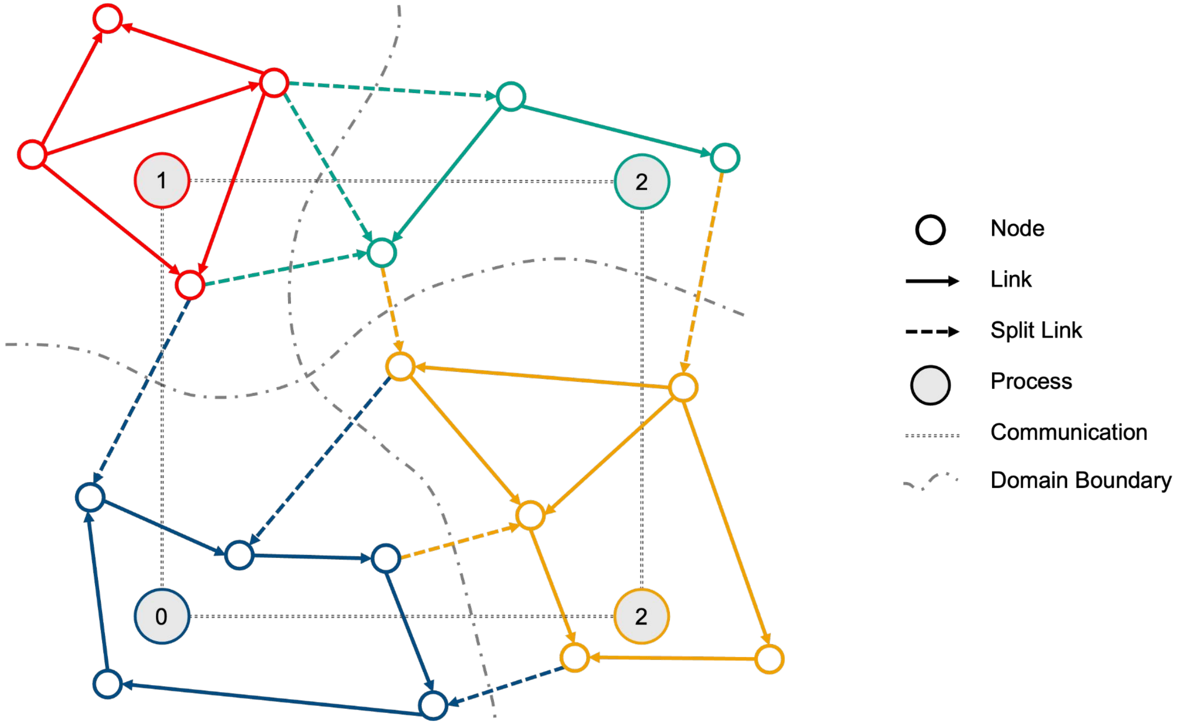


Distributed Simulation

Each Process works on local Mobism

Domain Boundaries are in the middle of links.

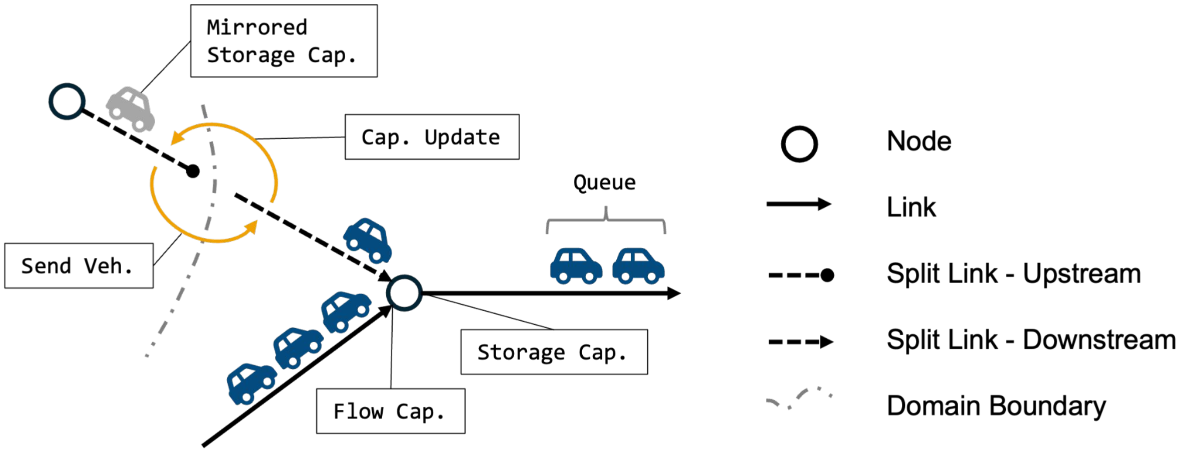
Connections between Domains are represented by split links



Distributed Simulation

Vehicles crossing domain boundaries are passed as messages

Available storage capacities are passed as messages



Results

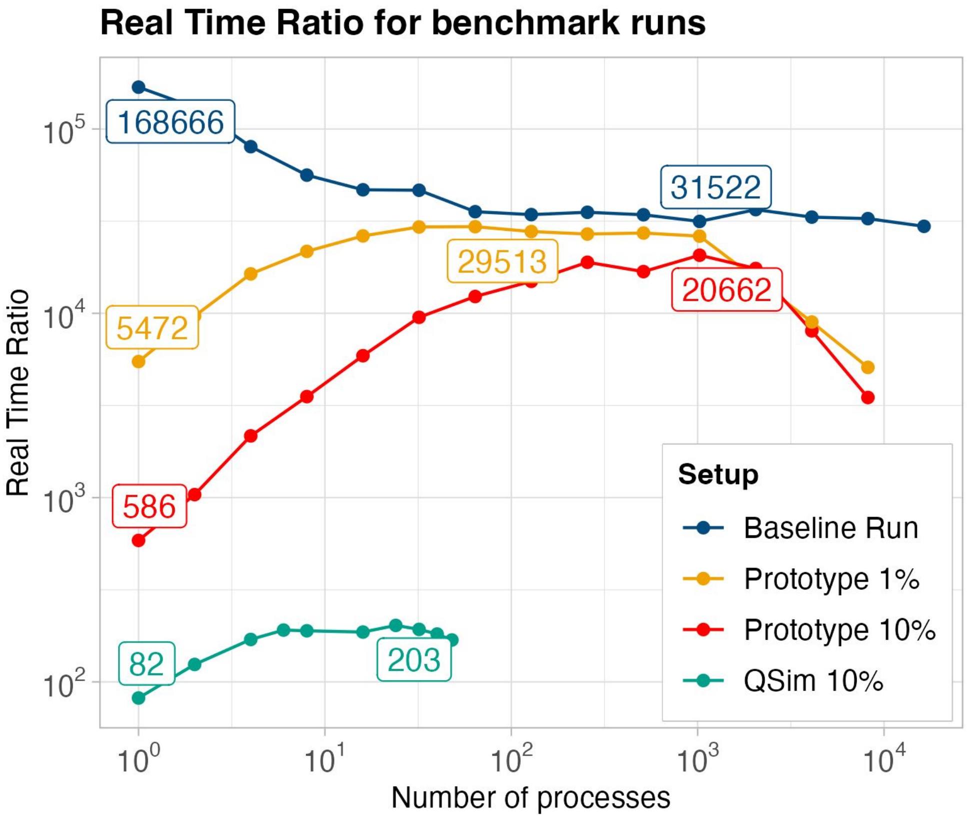
Speedup of 100x

First Results indicate that we are 100x faster, compared to the current QSim

We can scale to a RTR of ± 20,000

This means 24h can be simulated in 4.3s

Different scenario sizes level out at similar RTR

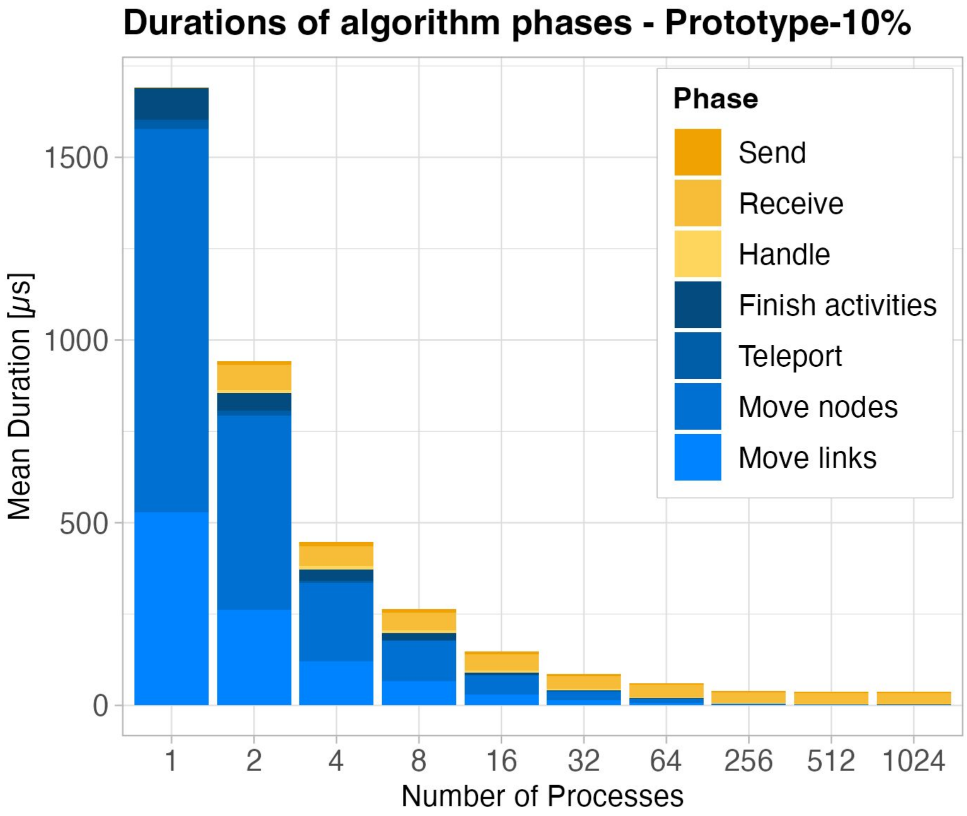


Latency as limiting Factor

Amount of simulation work per process is reduced with growing number of processes

Almost constant messaging overhead

With sufficient processes, the messaging overhead becomes driving factor for execution times



Discussion & Outlook

Improvements

Add formal investigations

- Strong scaling - Speedups
- Weak scaling - scaled Speedups

Take advantage of available hardware

- We only scaled up to 1000 processes
- Increase scenario size - easy
- Reduce messaging overhead - difficult

First write up on our website

DiQSim

Take learnings to a Java implementation

Project by Simunto to bring distributed algorithm to the Java implementation

Hope to benefit single computer setups as well

Provide flexible communication implementation to support commodity hardware

Contact

M. Sc. Janek Laudan
Research Associate @ TU Berlin
laudan@tu-berlin.de

Fachgebiet für Verkehrssystemplanung und Verkehrstelematik

KAI 4-1
Kaiserin-Augusta-Allee 104
10553 Berlin

https://vsp.berlin/