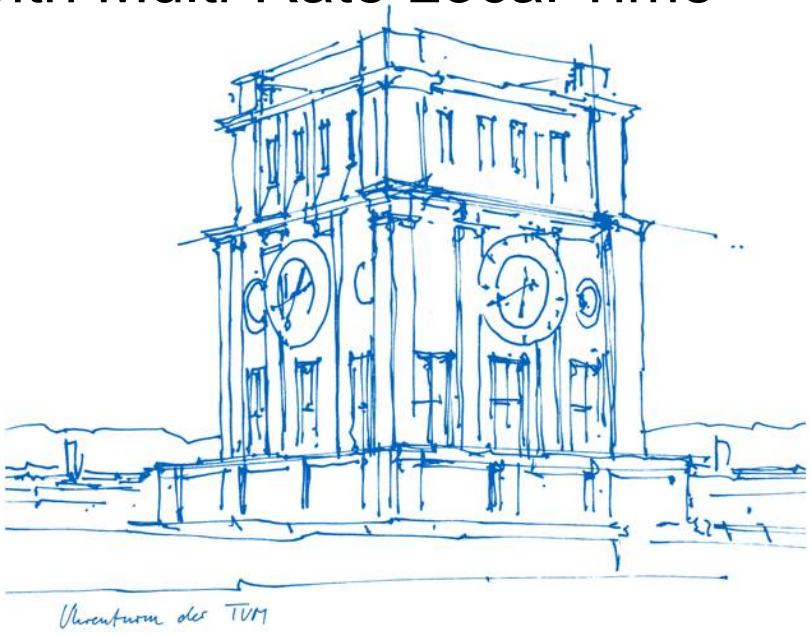
Evaluation of Multiple HPC Parallelization Frameworks in a Shallow Water Proxy Application with Multi-Rate Local Time Stepping

Martin Bogusz[1], Philipp Samfass[1], Alexander Pöppl[1], Jannis Klinkenberg[2], Michael Bader[1]

[1]Department of Informatics
Technical University of Munich

[2]Chair for High Performance Computing
RWTH Aachen University

Parallel Applications Workshop, Alternatives To MPI+X
November 12th 2020
Atlanta, Georgia



Motivation

New Challenges in HPC:

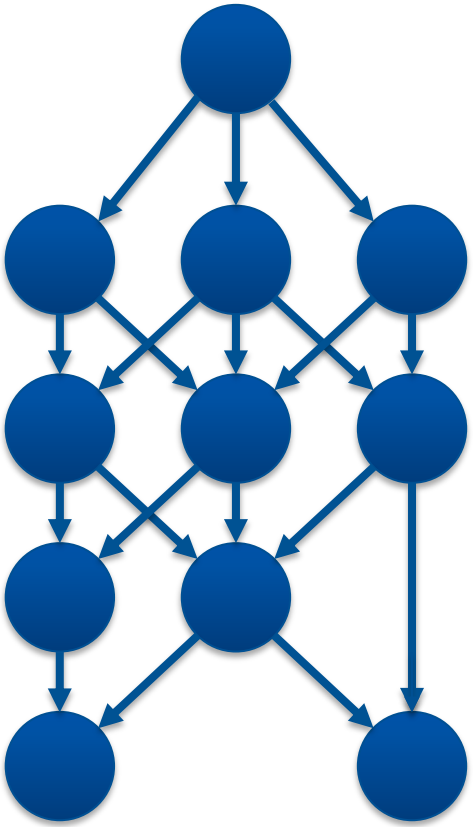
- Widening gap between processor speed and communication latencies
- Variable CPU clock frequencies (DVFS)
- Novel Numerics: local time stepping

Novel runtimes and parallelization frameworks

- Task-based (distributed) load balancing
- Overlap of communication and computation

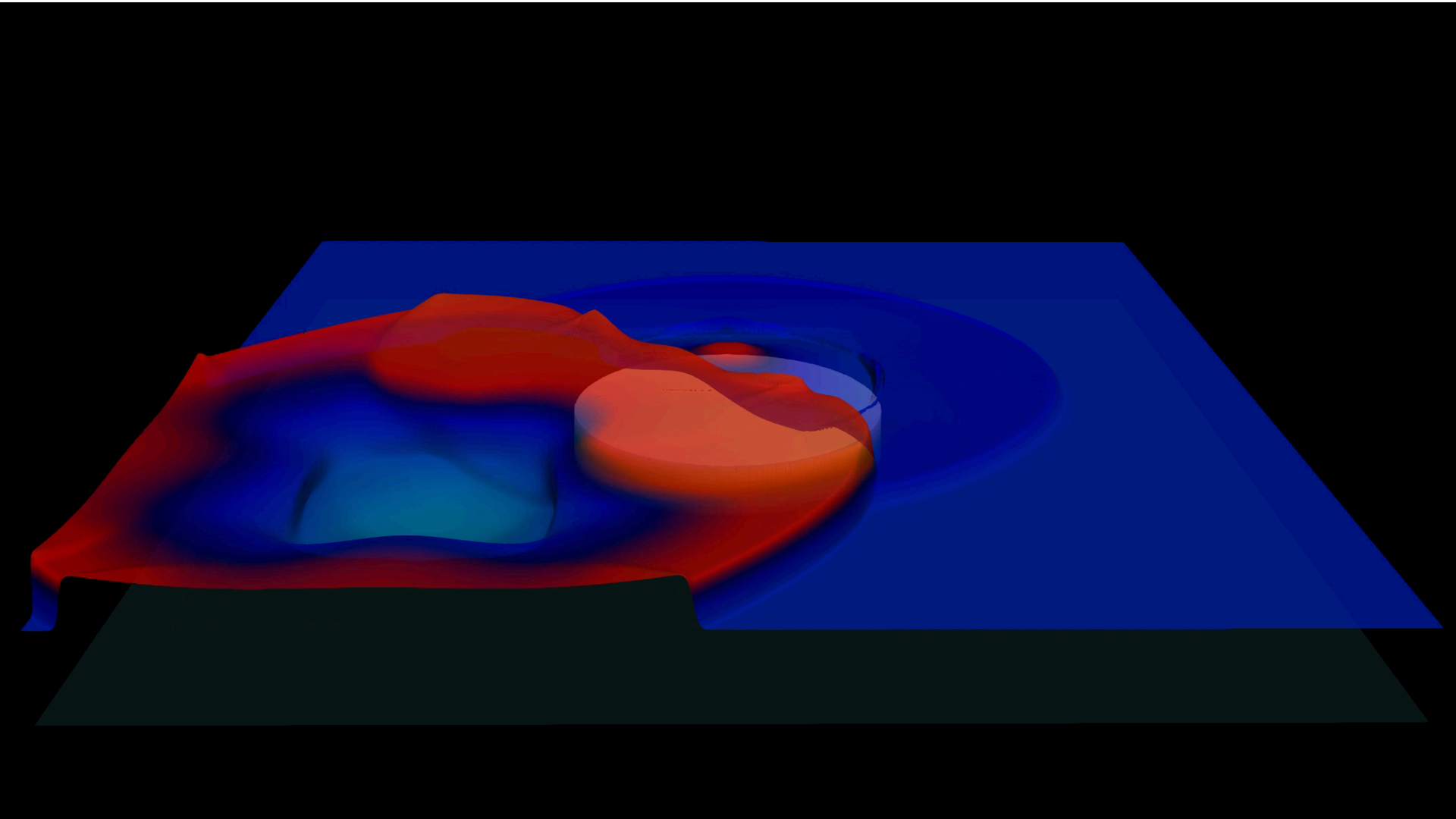
Study of four parallelization frameworks in a proxy application (SWE) for solving the shallow water equations compared to MPI baseline

- Frameworks: **Charm++, UPC++, Chameleon, HPX**



SWE-PPM – A Shallow Water Proxy Application

- Based on prior application **SWE**, a BSP-based code written using MPI and OpenMP
- Parallelized using different frameworks
- Efficiently vectorized code



SWE-PPM – A Shallow Water Proxy Application

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = S(t, x, y)$$

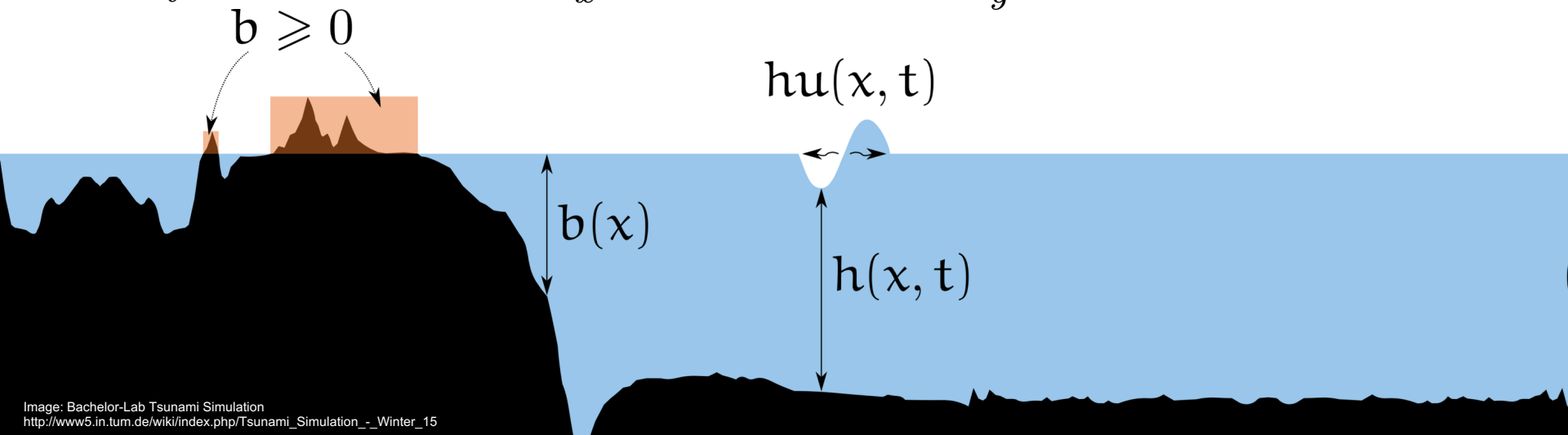


Image: Bachelor-Lab Tsunami Simulation
http://www5.in.tum.de/wiki/index.php/Tsunami_Simulation_-_Winter_15

SWE-PPM – A Shallow Water Proxy Application

Finite volume scheme on a Cartesian grid with piecewise constant unknown quantities and Euler time step

$$Q_{i,j}^{n+1} = Q_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathcal{A}^{+}\Delta Q_{i-\frac{1}{2},j} + \mathcal{A}^{-}\Delta Q_{i+\frac{1}{2},j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathcal{B}^{+}\Delta Q_{i,j-\frac{1}{2}} + \mathcal{B}^{-}\Delta Q_{i,j+\frac{1}{2}}^{n}\right)$$

Numerical approach based on LeVeque (R. J. LeVeque, D. L. George, and M. J. Berger. Tsunami modelling with adaptively refined finite volume methods. Acta Numerica, 20:211–289, 2011)

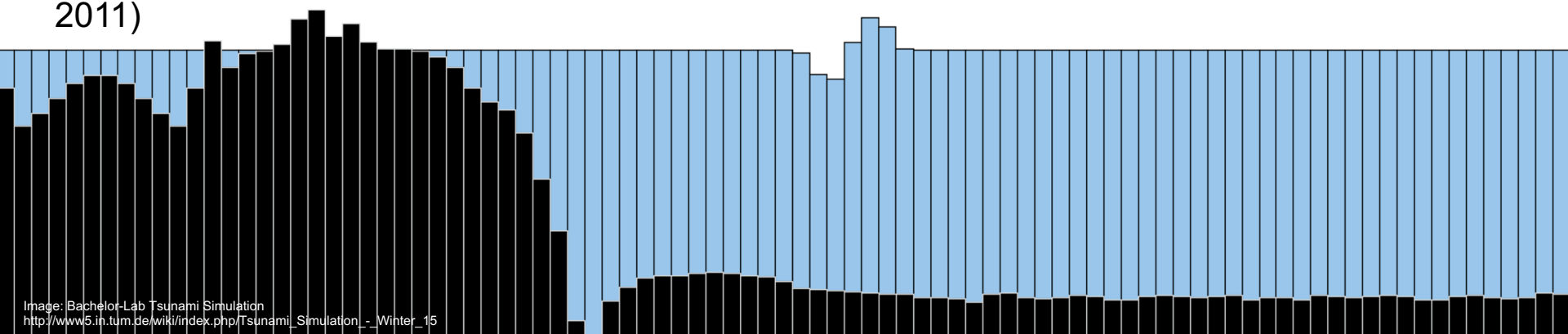
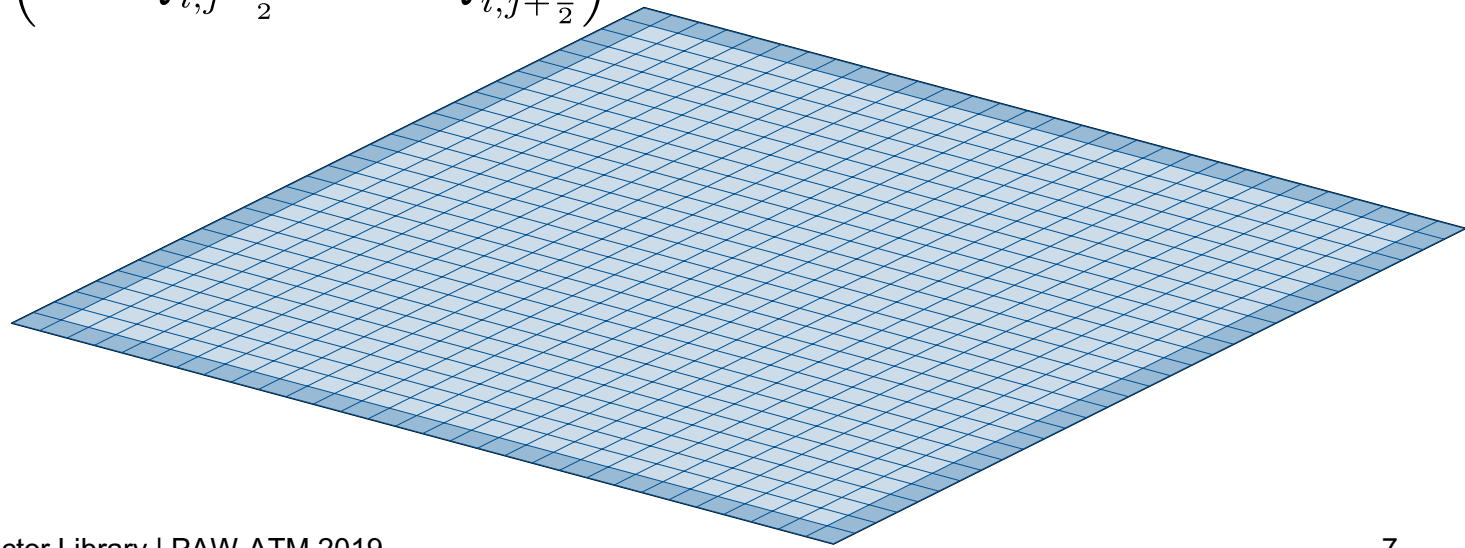


Image: Bachelor-Lab Tsunami Simulation
http://www5.in.tum.de/wiki/index.php/Tsunami_Simulation_-_Winter_15

SWE-PPM – A Shallow Water Proxy Application

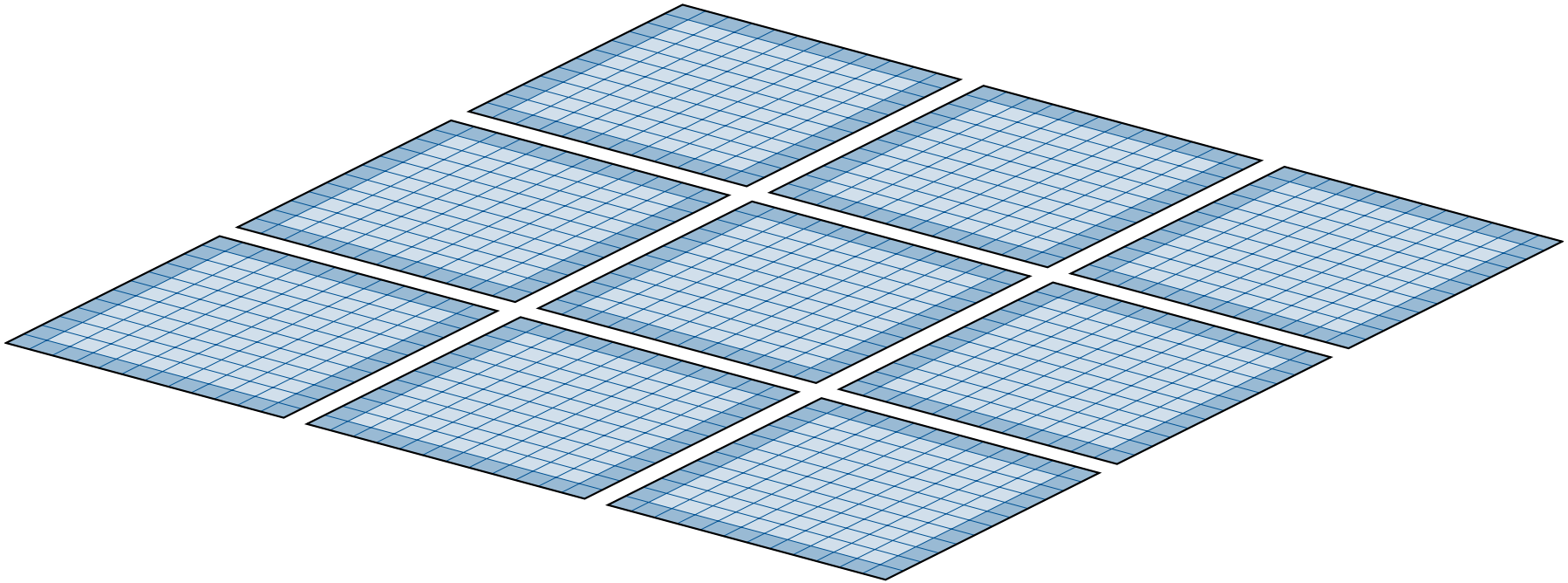
Finite volume scheme on a Cartesian grid with piecewise constant unknown quantities and Euler time step

$$Q_{i,j}^{n+1} = Q_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(\mathcal{A}^{+}\Delta Q_{i-\frac{1}{2},j} + \mathcal{A}^{-}\Delta Q_{i+\frac{1}{2},j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(\mathcal{B}^{+}\Delta Q_{i,j-\frac{1}{2}} + \mathcal{B}^{-}\Delta Q_{i,j+\frac{1}{2}}^{n}\right)$$

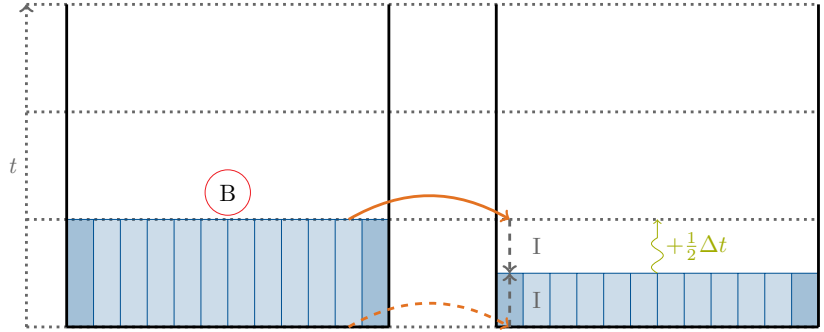


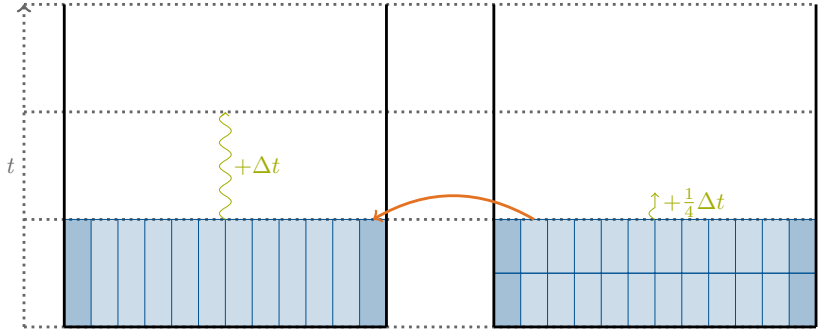
SWE-PPM – A Shallow Water Proxy Application

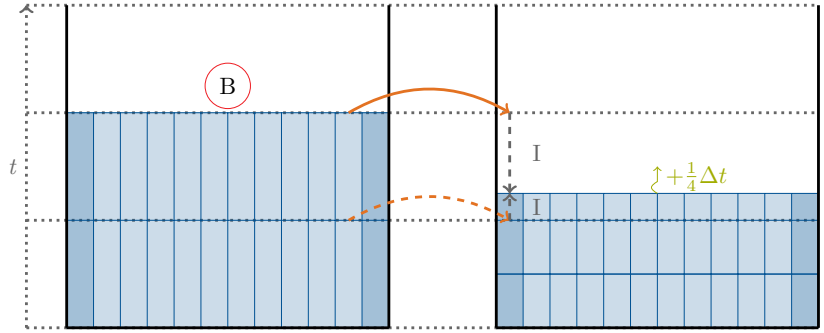
Subdivision into rectangular, equally-sized patches with halo regions

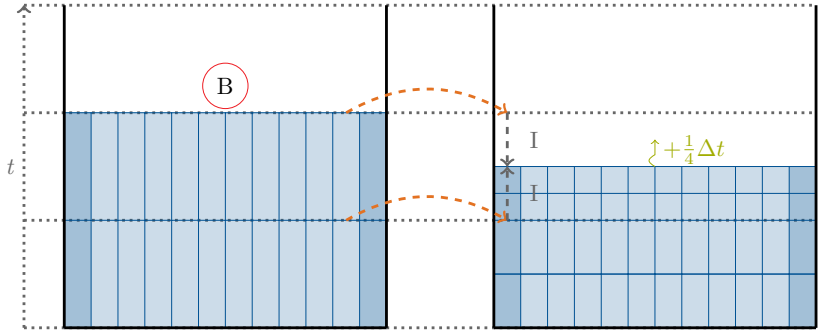


SWE-PPM – Local Time Stepping





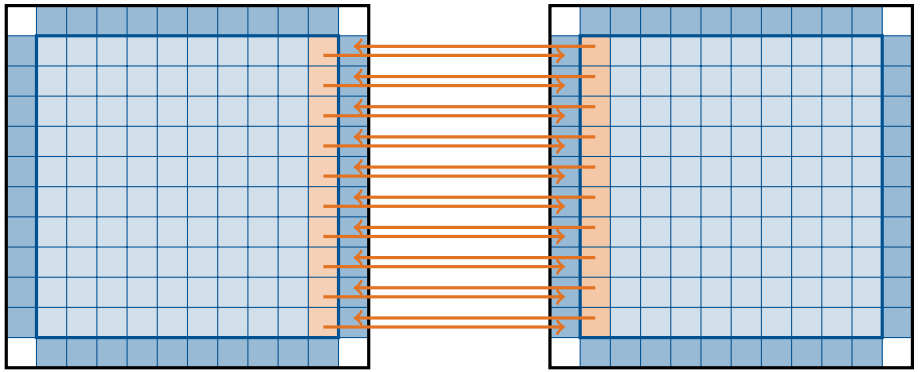




The MPI Baseline

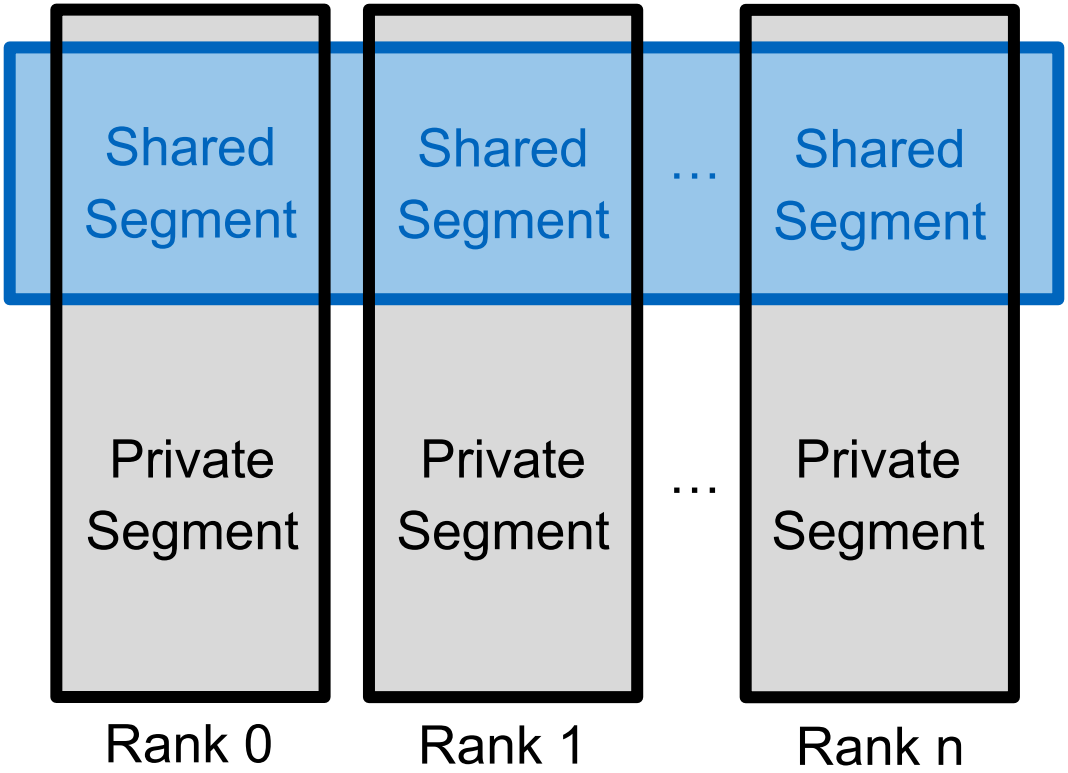
- Classical domain decomposition of grid
- Boundary exchange:
  - Copy and halo layers
  - Two-sided, non-blocking communication
  - Bulk-synchronous style

- Two variants:
  - Single block per rank
  - Multiple blocks per rank with
    ➡ OpenMP fork-join parallelism
    overdecomposition



UPC++

- **A**synchronous **P**artitioned **G**lobal **A**ddress **S**pace (APGAS) Model
- Reliance on one-sided communication
- Asynchronous, continuation-based API
- Based on GASNet-EX, makes direct use of InfiniBand and (some) Cray interconnects

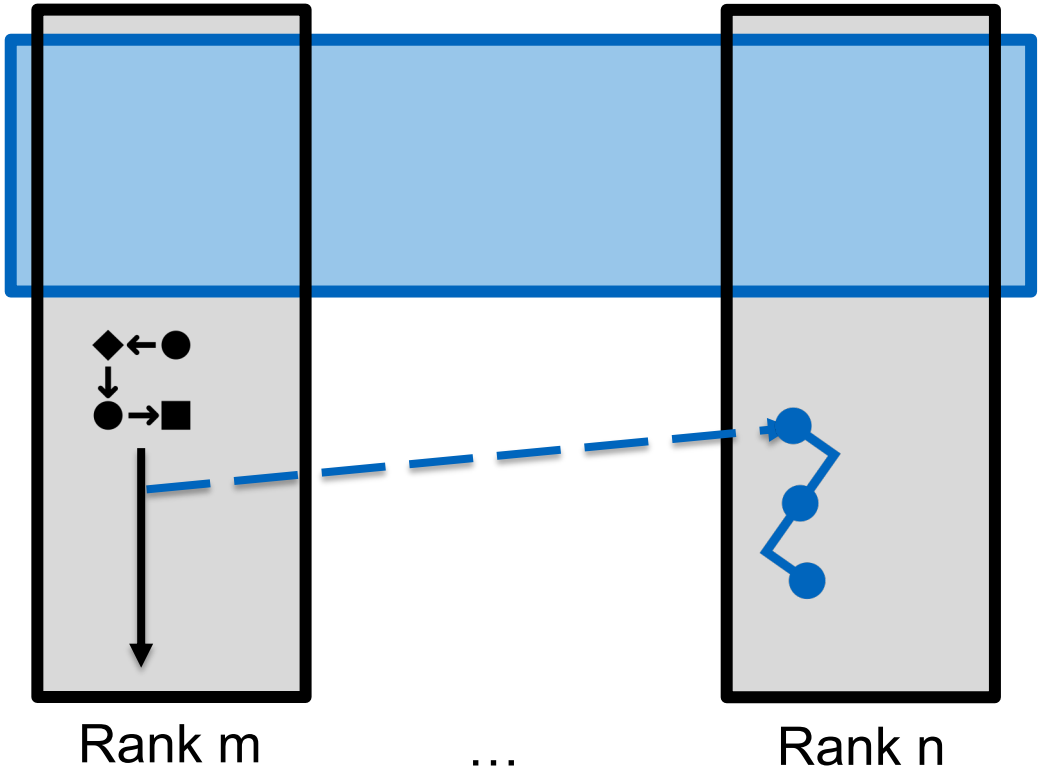


Adapted from: *UPC++ Specification v1.0 Draft 10, available at https://upcxx.lbl.gov*

UPC++

RPCs

- Executed asynchronously
- Serialization and transfer of parameters, return value
- Completion events available after the local part (or overall RPC execution) is finished



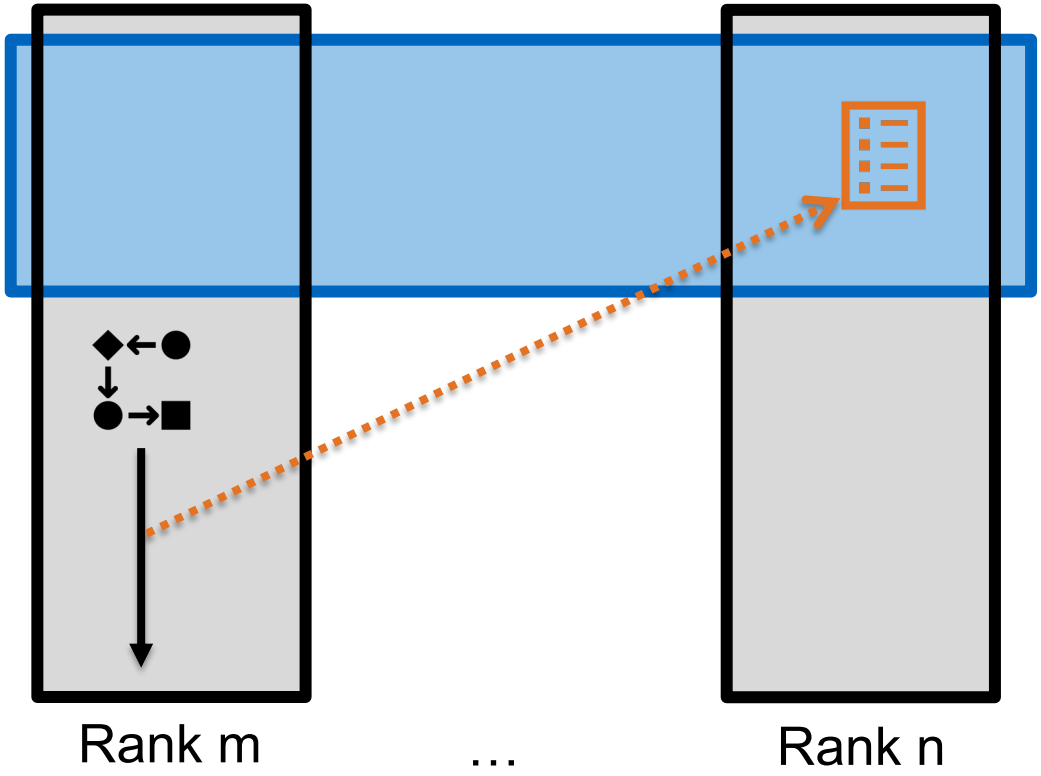
UPC++

RPCs

- Executed asynchronously
- Serialization and transfer of parameters, return value
- Completion events available after the local part (or overall RPC execution) is finished

Global Pointers

- Point to data in shared segment
- May be used as target for RMA operations



UPC++

RPCs

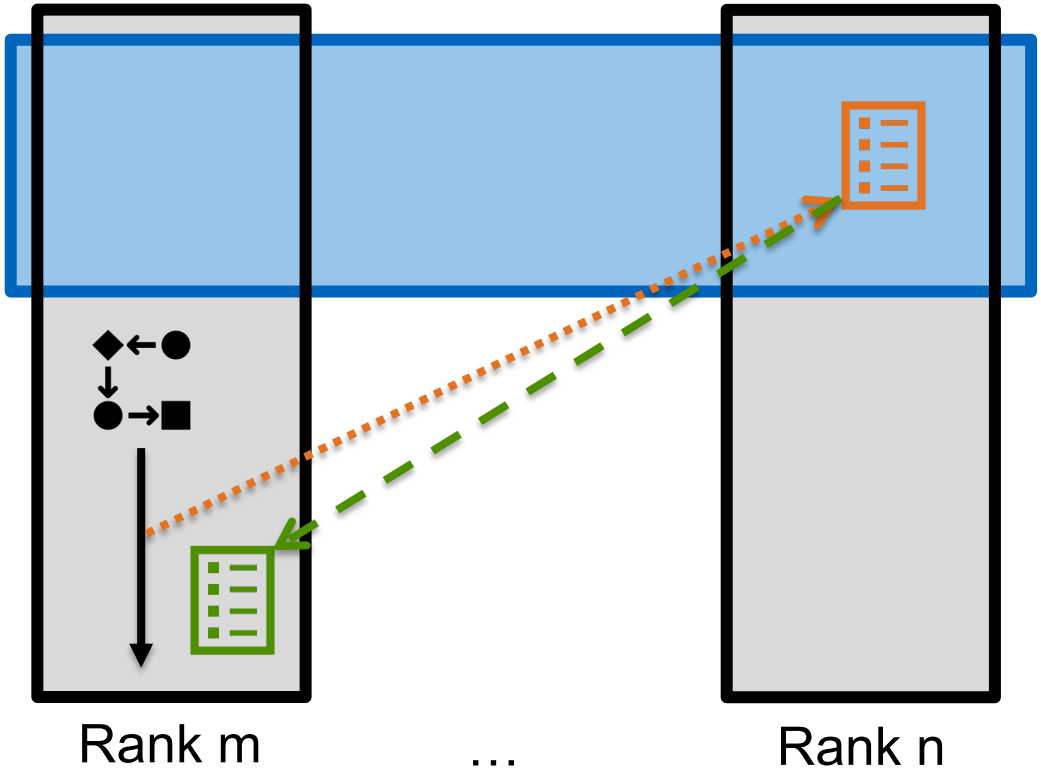
- Executed asynchronously
- Serialization and transfer of parameters, return value
- Completion events available after the local part (or overall RPC execution) is finished

Global Pointers

- Point to data in shared segment
- May be used as target for RMA operations

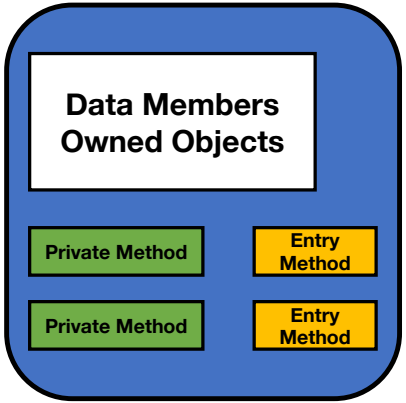
RMA Operations

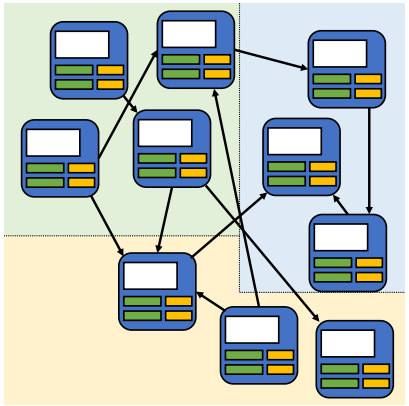
- Read from & and write to remote memory
- Executed asynchronously
- Completion events available after local, remote or operation finished



Charm++

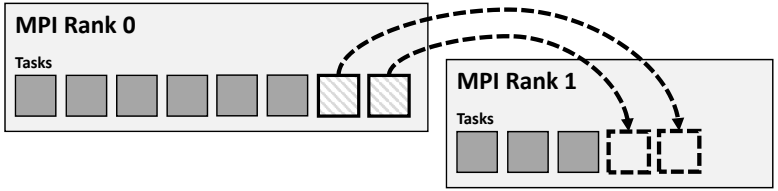
- Message-driven parallel programming model
- Emphasizes dynamic execution by employing a **runtime system**
- Unit of parallelization: **Chare objects**
- **Chare –** a capsuled C++ object with asynchronous methods which are invoked by active messaging
- May be overdecomposed and migrated
  ➡ **load balancing**
- Predefined set of measurement-based load balancing strategies

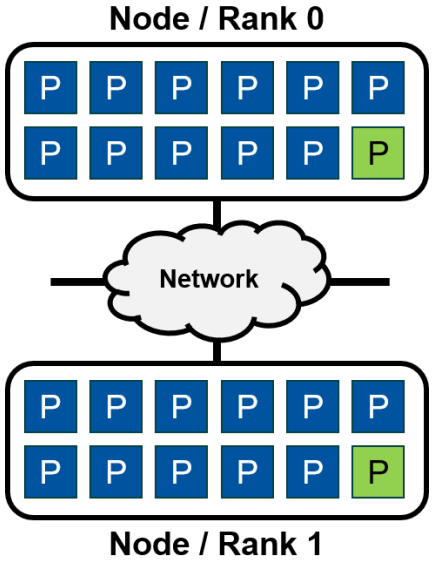




Chameleon

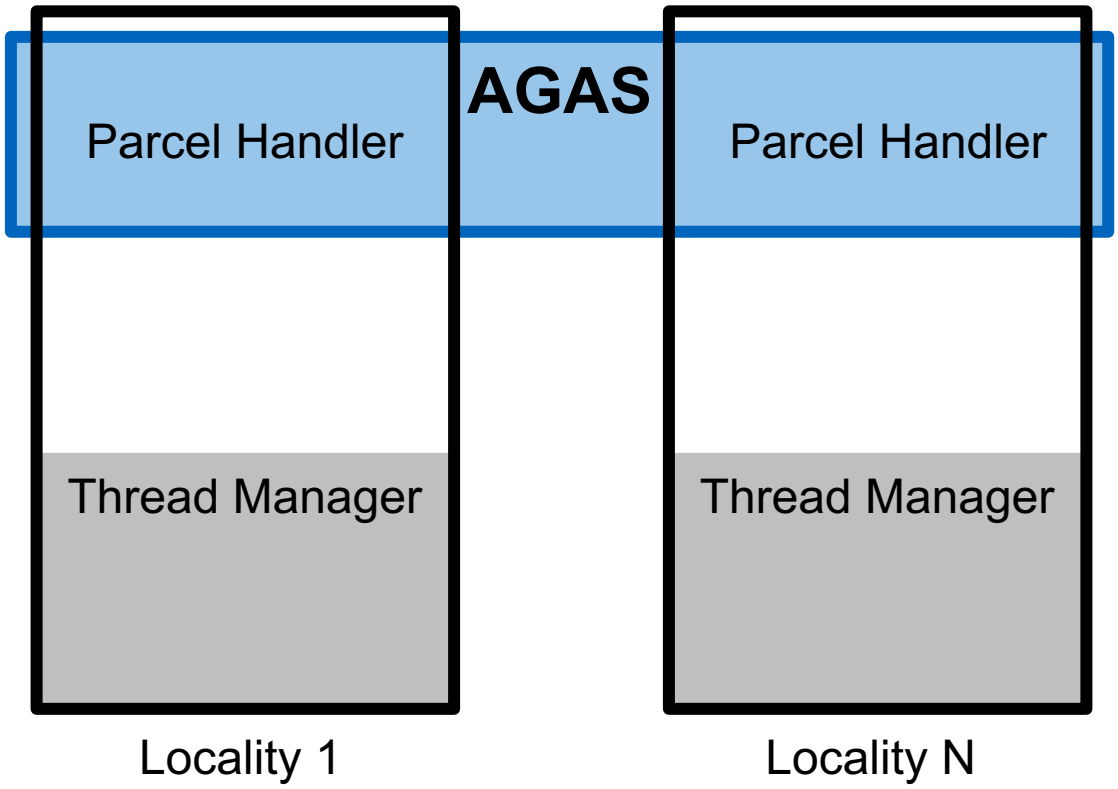
- Reactive task migration in distributed memory
- Incremental integration into existing task-based **MPI+OpenMP** codes
- Targets (un-)predictable load imbalances caused by
  - Hardware variability
  - Dynamic numerical algorithms (e.g., SWE-PPM)
- Unit of parallelization: **Migratable task**
- Dedicated service thread
  - Continuous performance introspection (e.g. work load)
  - Decision making & migration
- Features:
  - Overlapping load balancing communication & computation
  - Customizability: Predefined migration strategy + tools interface





HPX

- **H**igh **P**erformance Parall**eX** (HP**X**)
- Based on the Parallex execution model
- Asynchronous task based fine-grained parallelization
- Flow-Control, message-driven RPC
- Implementation of C++17 parallel algorithms



HPX

Local Control Objects (LCOs)

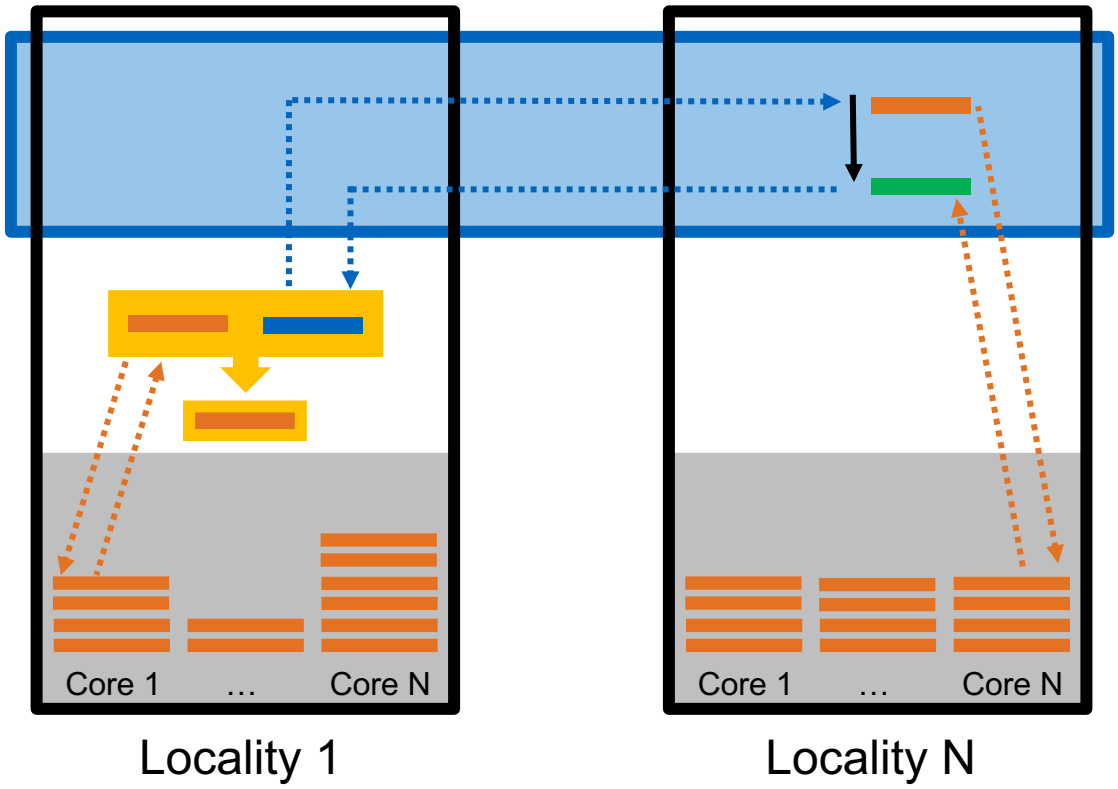
- Handles to asynchronous tasks
- May be used to create dependencies among LCOs which are resolved by the runtime system

Asynchronous Tasks

- Unit of parallelization
- Low-overhead user-space threads
- Scheduled by the **Thread Manager**

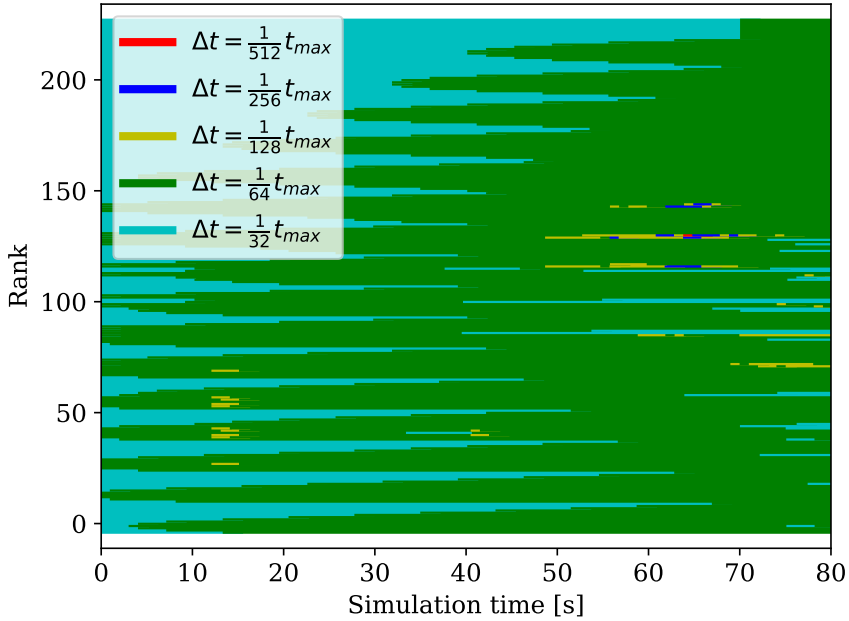
RPCs

- Executed asynchronously
- Serialization and transfer of parameters and return values by the **Parcel Handler**
- Completion is signalized by remote locality

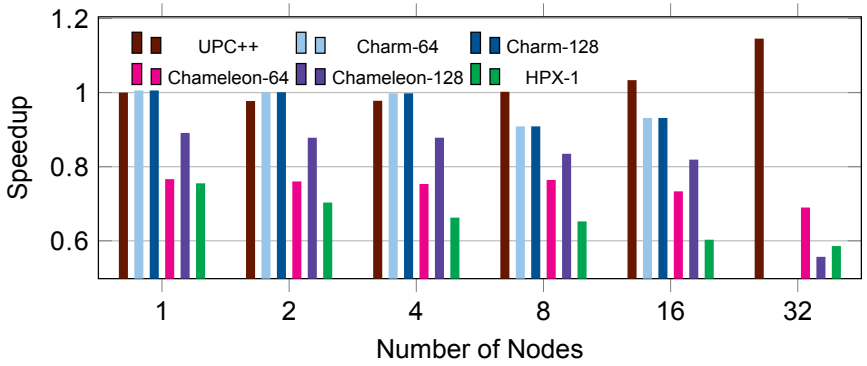


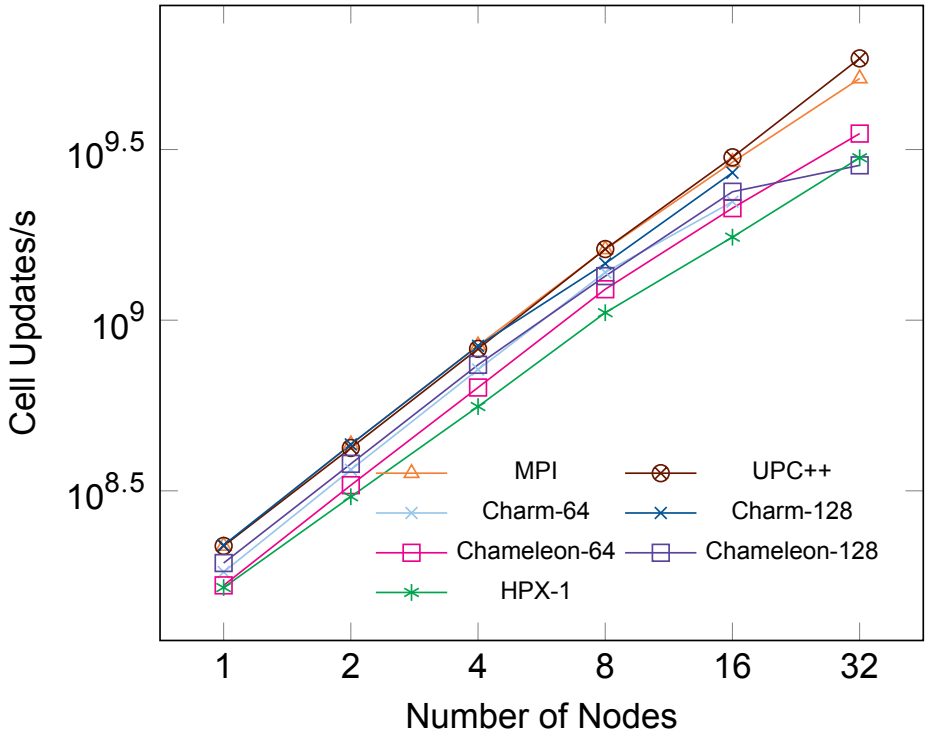
Evaluation

- Performed on LRZ CoolMUC 2
  - Dual socket Intel Xeon (Haswell) nodes
  - 28 cores (56 hyperthreads) per node
  - 64GB RAM
  - 6 TFlop/s (SP) peak performance
  - Intel Compiler 19, vectorization using AVX2
- Setup
  - All variants use the same numerical approach and same Riemann solver
  - Performance measurements based on **time to solution**

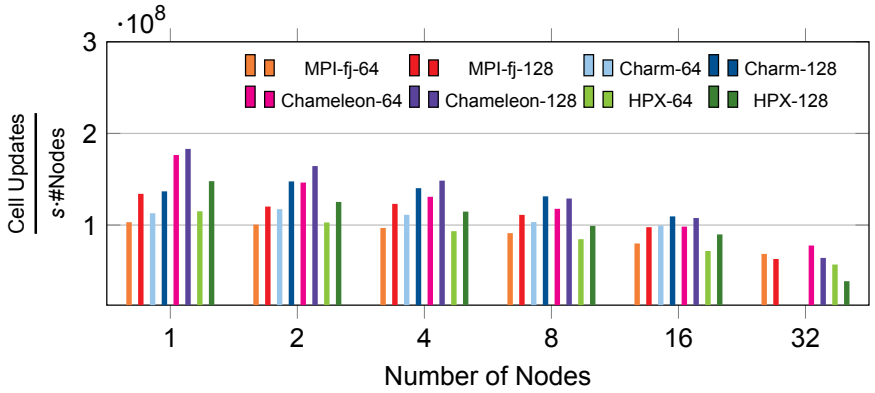


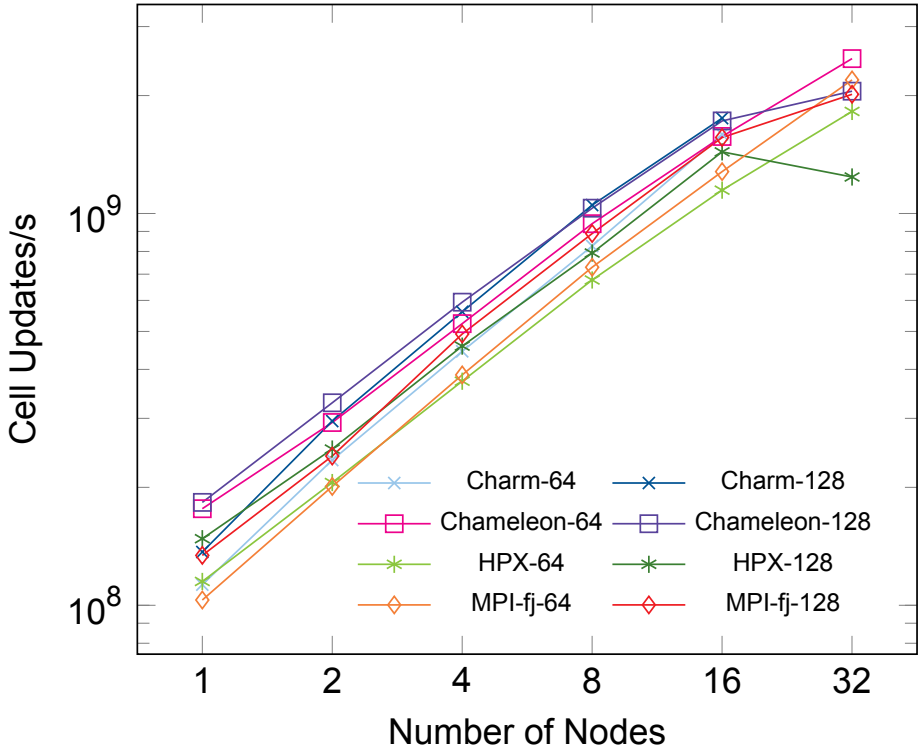
Evaluation – Global Time Stepping





Evaluation – Local Time Stepping





Conclusion and Outlook

SWE-PPM

- Support for **5** different frameworks
- Evaluated **17** different configurations

Modern Frameworks:

- Focus on asynchronous communication
- Enable improved load-balancing for local time stepping
- Competitive with MPI + OpenMP

Outlook:

- Add support for more frameworks: Legion, Co-Array Fortran, OpenSHMEM, Chapel, …
- Explore runtime systems for heterogeneous hardware architectures
- Optimize HPX performance, test latest features such as *libfabric* backend

Questions + Acknowledgements

- This research was funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) - Project number 14671743 - TRR 89 Invasive Computing (www.invasic.de).
- This work has received funding from the German Federal Ministry of Education and Research (BMBF) under Grant Nos. 01IH16004B and 01IH16004C (project Chameleon, www.chameleon-hpc.org).
- We thank the Leibniz Supercomputing Centre for providing compute resources on the CoolMUC2 cluster.
- Finally, we express our gratitude to Jurek Olden and Simon Convent for their initial implementations on Charm++ and Chameleon.

- SWE-PPM is available on GitHub: https://github.com/TUM-I5/SWE-PPM

Fork us on GitHub. Pull Requests are encouraged!