

SC22

Dallas, TX | hpc accelerates.

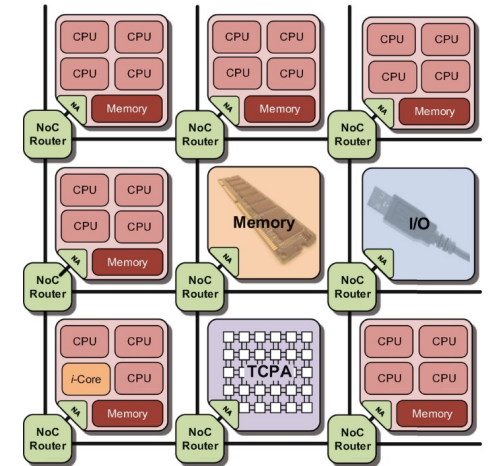
Asynchronous Workload Balancing through Persistent Work-Stealing and Offloading for a Distributed Actor Model Library

Yakup Budanaz, **Mario Wille**, and Michael Bader

Technical University of Munich

School of Computation, Information and Technology – Department of Computer Science

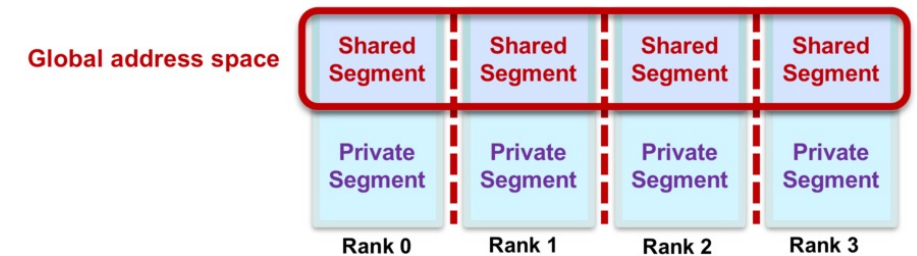
- Motivation stems from **Invasive Computing**¹
 - Dynamic resource allocation and deallocation
 - Provides explicit handles to specify resource requirements desired or required in different phases of execution
 - Usage of actors to facilitate the specification of requirements
- Use **UPC++** to shift from embedded to HPC applications
 - Development of an UPC++ based actor framework^{3 4}
 - Extension of the framework to enable migration of actors for asynchronous workload balancing



Typical tiled invasive architecture – image from (2)

- (1) <https://www.invasic.de/>
- (2) https://link.springer.com/chapter/10.1007/978-3-030-47487-4_9
- (3) <https://github.com/TUM-I5/Actor-UPCXX>
- (4) [Pöppel, A.; Baden S.; Bader, M.: A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application, PAW-ATM 2019](#)

- Asynchronous Partitioned Global Address Space (**APGAS**) Model
- Designed for writing efficient, scalable parallel programs on distributed-memory parallel computers²
- Key communication facilities in UPC++ are one-sided **Remote Memory Access** (RMA) and **Remote Procedure Call** (RPC)
- Focused on maximizing scalability
- Communication operations are asynchronous
- Uses GASNet³ for communication across a wide variety of platforms



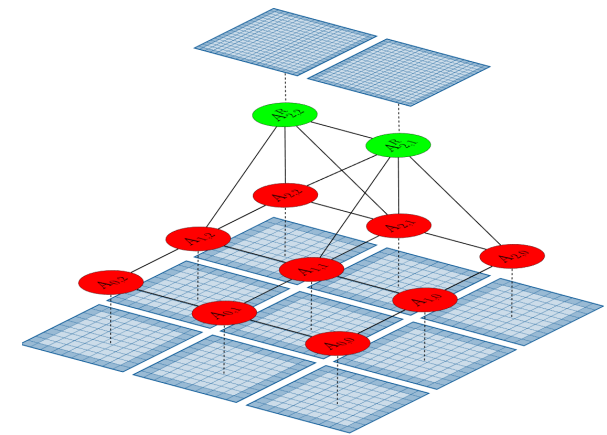
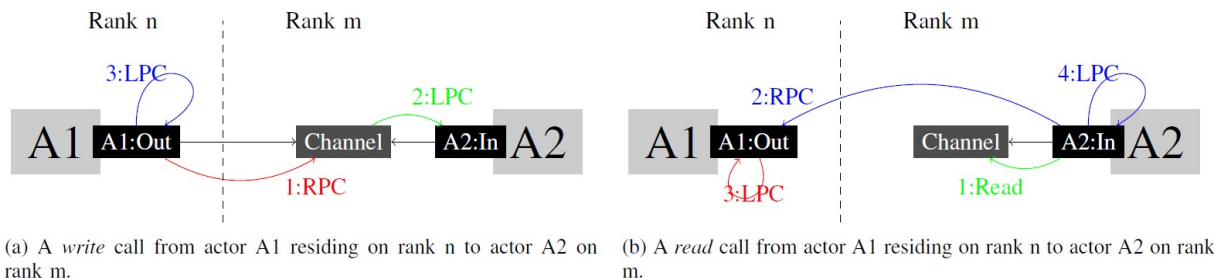
PGAS Memory Model – image from (2)

(1) <https://bitbucket.org/berkeleylab/upcxx/wiki/Home>

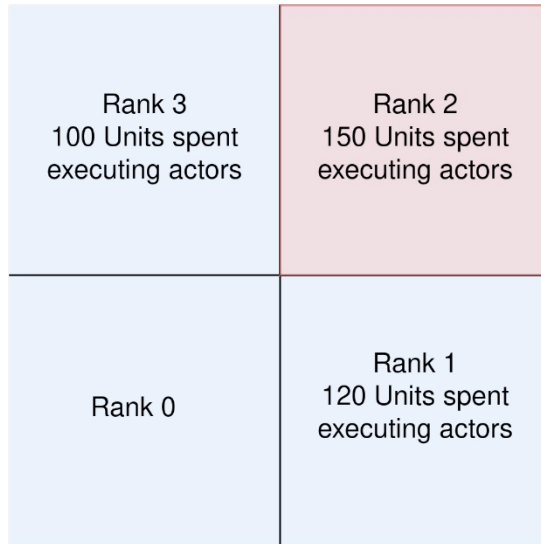
(2) <https://upcxx.lbl.gov/docs/html/guide.html>

(3) <https://gasnet.lbl.gov/>

- Actor encapsulates specific functionality, data and behavior
- Is an object that is identified by its unique name and assigned to a part of the parallel simulation
- Actor and its data can be serialized for sending it to another rank using UPC++
- Connections of actors are saved in a global graph structure that is replicated on every rank in Actor-UPCXX
- Communication through one-sided asynchronous messages
- Facilitate actors for dynamic load balancing

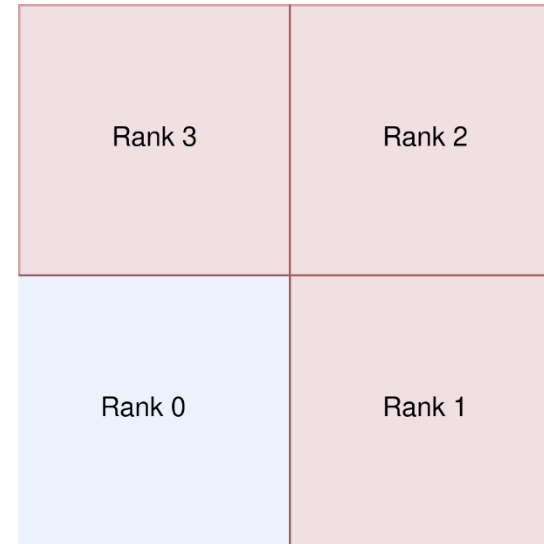


- Two strategies: actor stealing and actor offloading
- Actor stealing: underloaded rank chooses rank to steal an actor
- Actor offloading: rank that detects an imbalance may decide to offload an actor to a rank that is underloaded
- Both migration strategies perform every action through asynchronous calls (RPCs)
- Actor stealing strategy is further divided into
 - *Global vs. Local*: specifies the set of remote ranks which can be stolen
 - *Random vs. Busy*: specifies the type of polling which is applied to the set of remote ranks



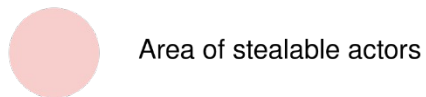
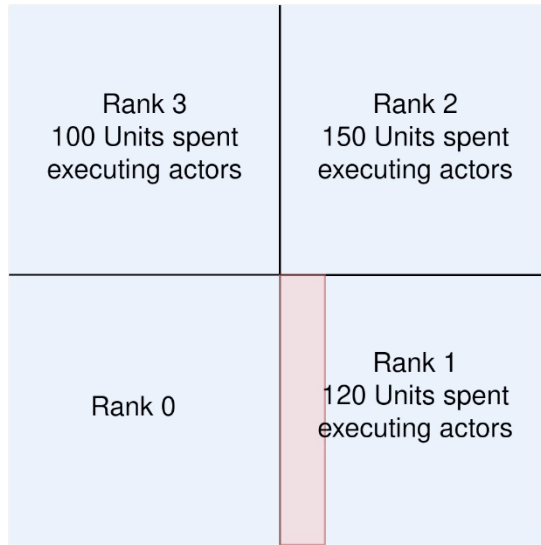
Area of stealable actors

Global-busy: steals an actor from the rank that has spent the most time executing actors

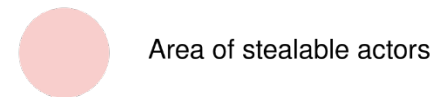
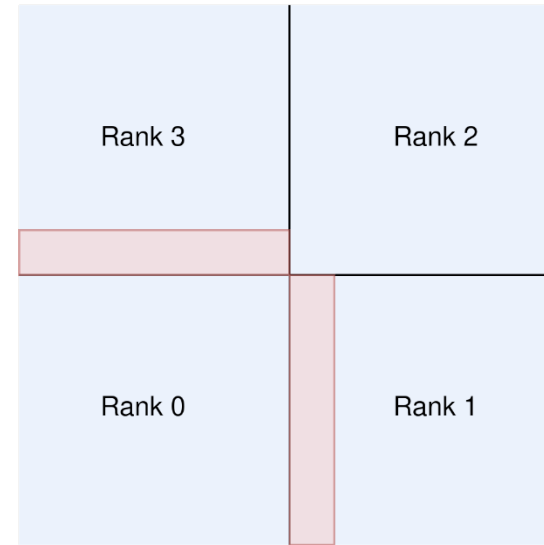


Area of stealable actors

Global-random: steals an actor from any rank without limitations



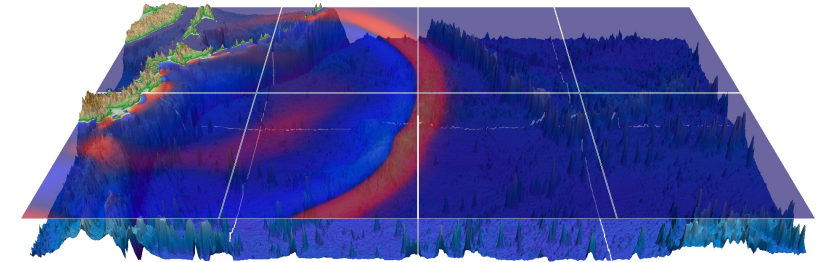
Local-busy: steals an actor from a neighboring rank that has spent the most time executing actors



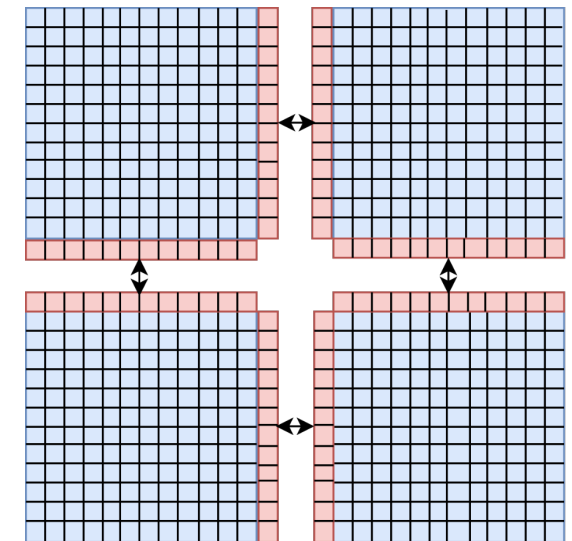
Local-random: steals any neighboring actor from one of the neighboring ranks

Pond – A Shallow Water Proxy Application

- Uses Actor-UPCXX as actor library
- Implements finite volume solvers which solves the shallow water equations



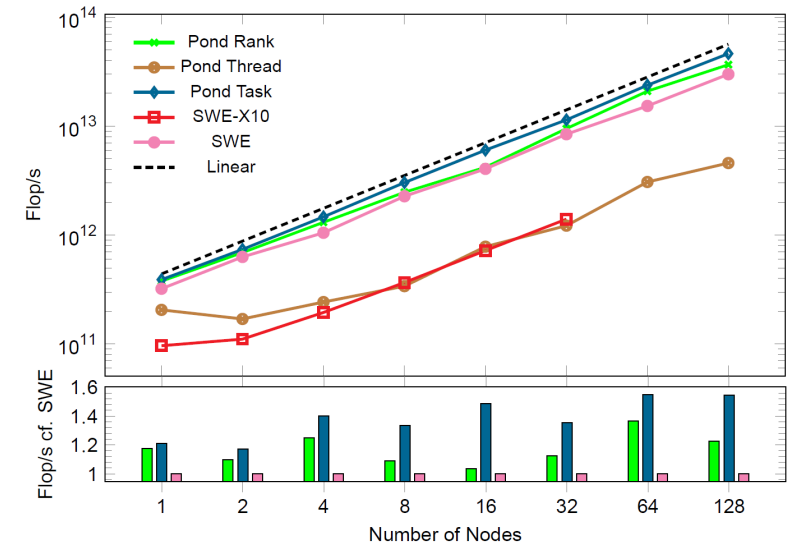
- Pond organizes the 2D Cartesian discretization grid as patches
- Every patch is assigned to an actor permanently
- Example grid decomposed into four patches
- Cells of the patches are marked in blue
- Ghost layers (in red) are used to synchronize data between patches



Pond – A Shallow Water Proxy Application

- Update the cells in the ghost layer according to boundary conditions or with values communicated by neighbor patches
- For each edge compute approximate fluxes between the adjacent cells
- Accumulate the fluxes as net updates
- Update the cell quantities using the net updates
- Ghost cell values are updated by sending one-sided messages between Pond's actors
- **Lazy activation:** patches are only activated when a propagating wave enters the patch

- (1) [Pöpl, A.; Baden S.; Bader, M.: A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application](#)
- (2) https://invasic.informatik.uni-erlangen.de/en/tp_a4_PhIII.php



Weak scaling of Pond – image from (1)

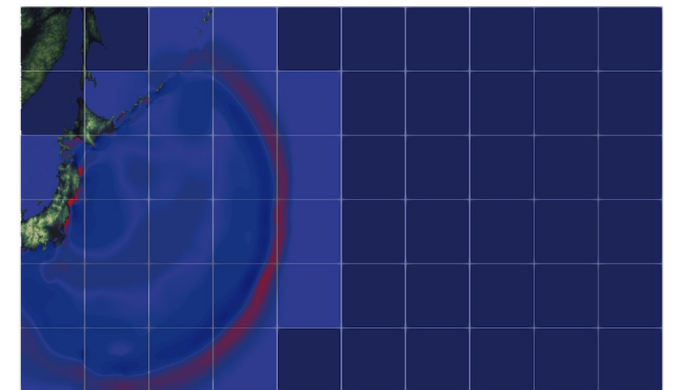


Illustration of lazy activation – image from (2)

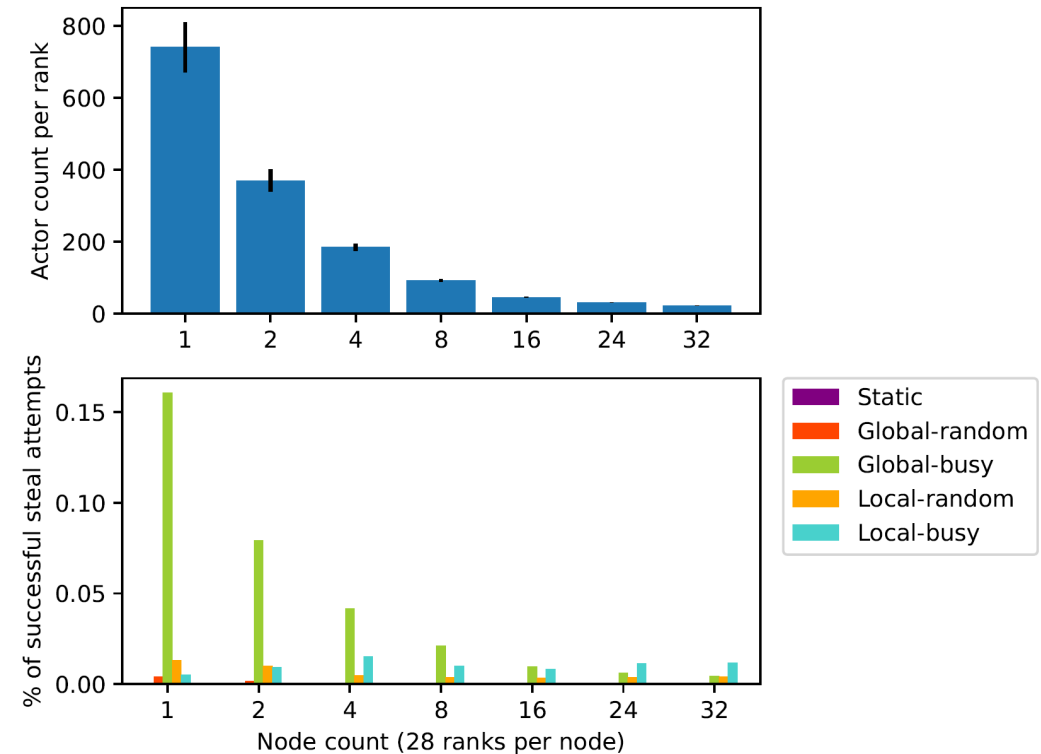
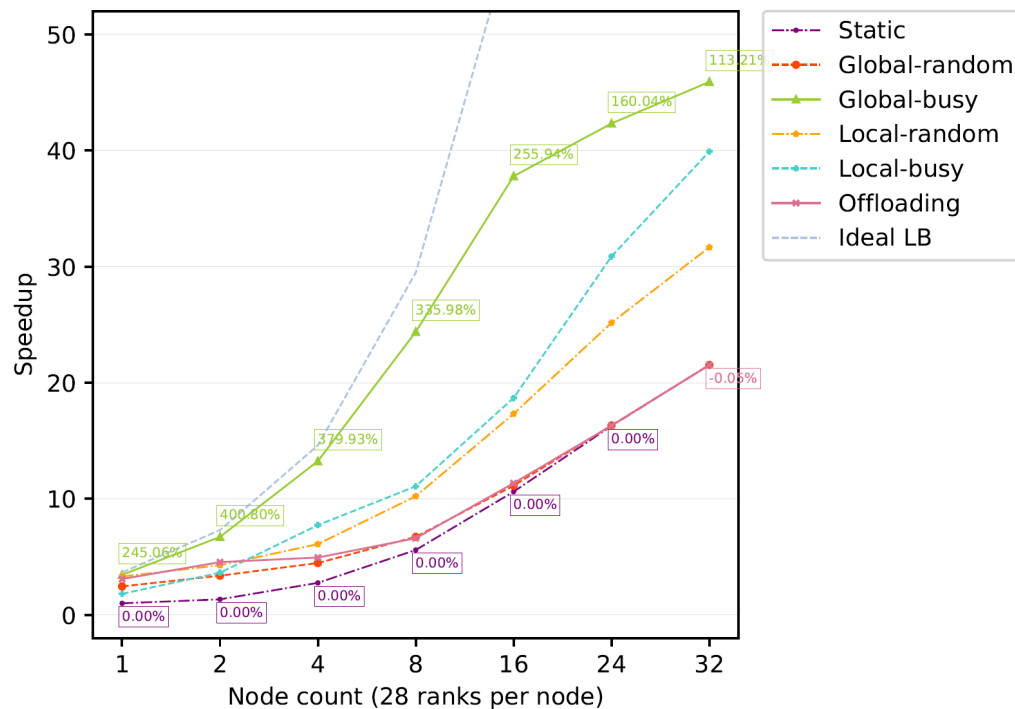
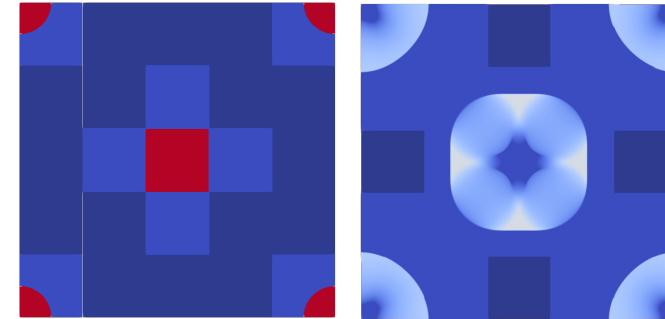
- Load balancing strategies have been compared using the scenarios
 1. Static workload: 18000×18000 grid divided into 250×250 patches with 1 patch per actor
 2. Node slowdown: 18000×18000 grid divided into 250×250 patches with 1 patch per actor
 3. Lazy activation: 36000×36000 grid divided into 250×250 patches with 1 patch per actor
- All scenarios are strong scaling tests
- All scenarios use the same solver
 - Performed on CoolMUC-2 Cluster hosted by the Leibniz Supercomputing Center (LRZ)¹
 - Equipped with 28-way Intel Xeon E5-2690 v3 compute nodes and FDR14 Infiniband interconnect
 - UPC++ version used is 2022.03.0.
 - UPC++, and Actor-UPCXX were compiled with the Intel oneAPI compilers²
 - OpenMPI v4.1.2 and HWLoc 2.6.0 are used by the communication backend of UPC++
 - GASNet-EX for the job launch

(1) <https://doku.lrz.de/display/PUBLIC/Linux+Cluster>

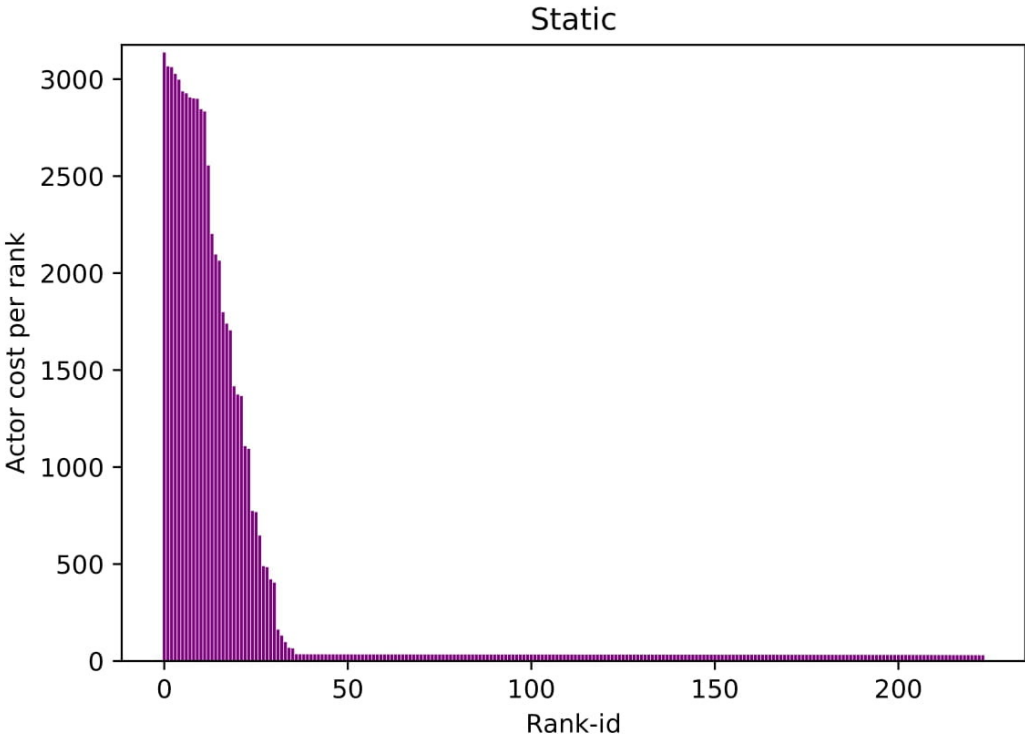
(2) <https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html>

Evaluation: Lazy Activation Scenario – Speedup

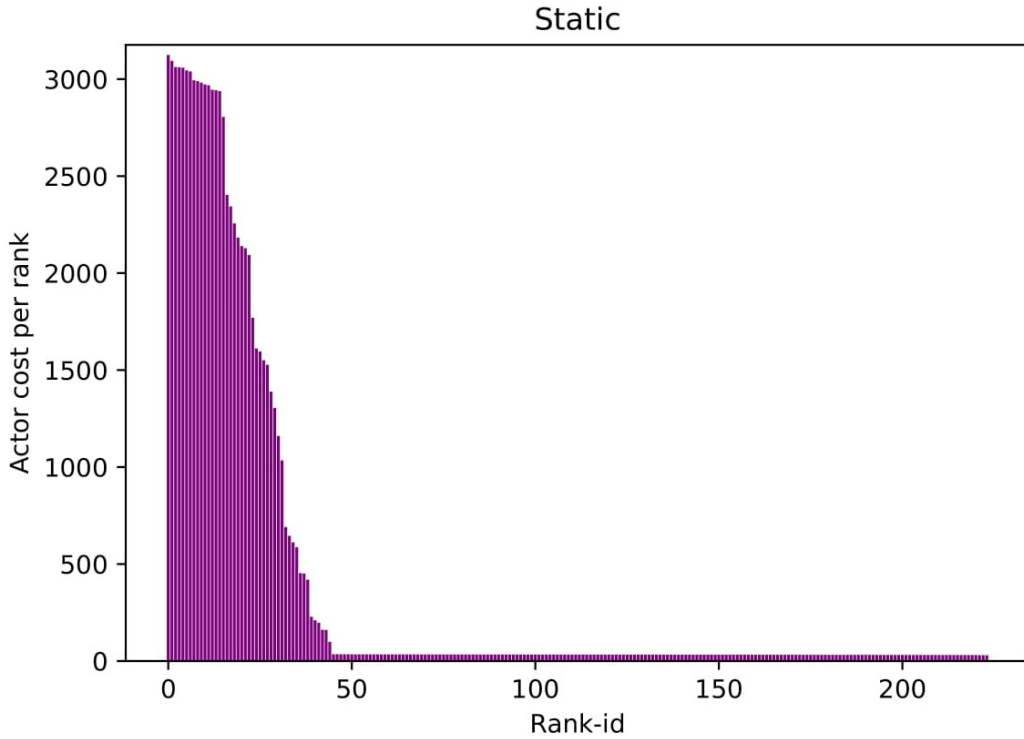
- Workload is determined by the evolving solution
- Actors are activated as waves enter their patch
- Static: no actor migrations



Evaluation: Lazy Activation Scenario – Load Distribution

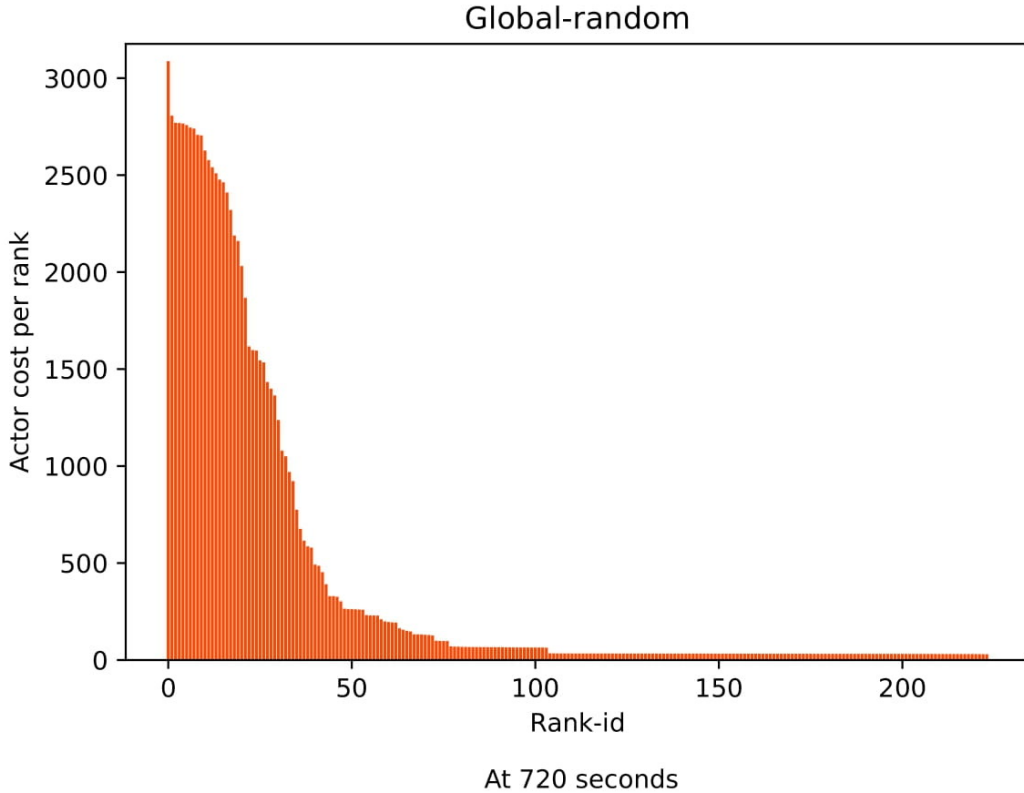
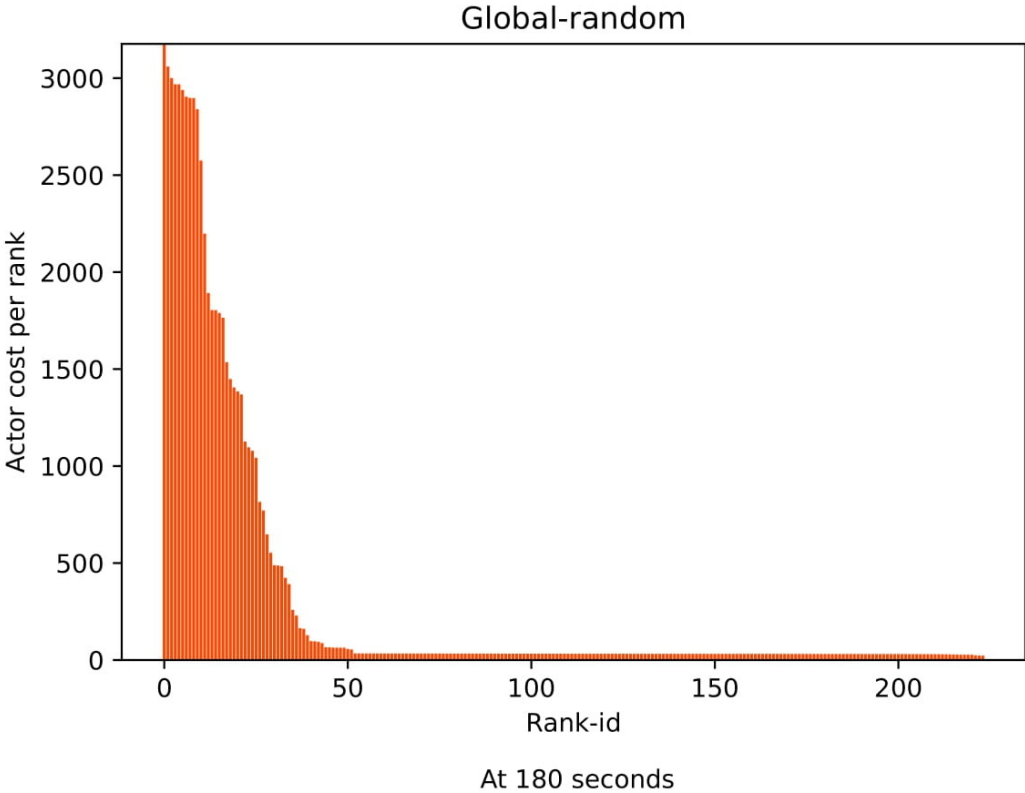


At 180 seconds

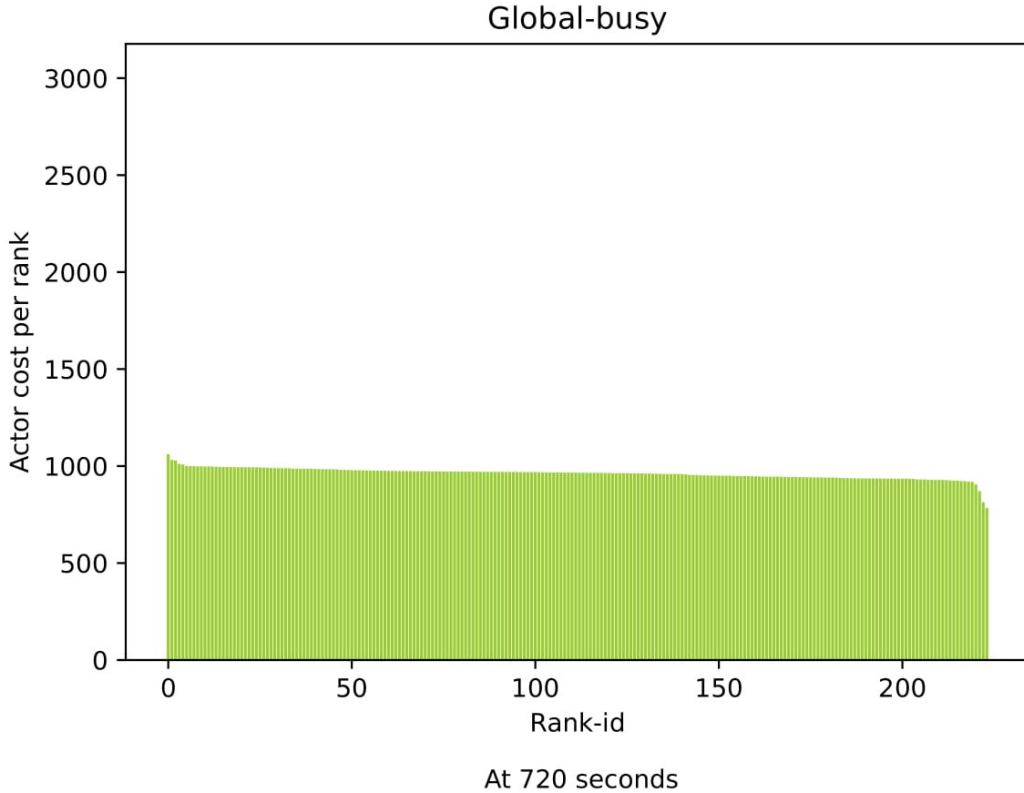
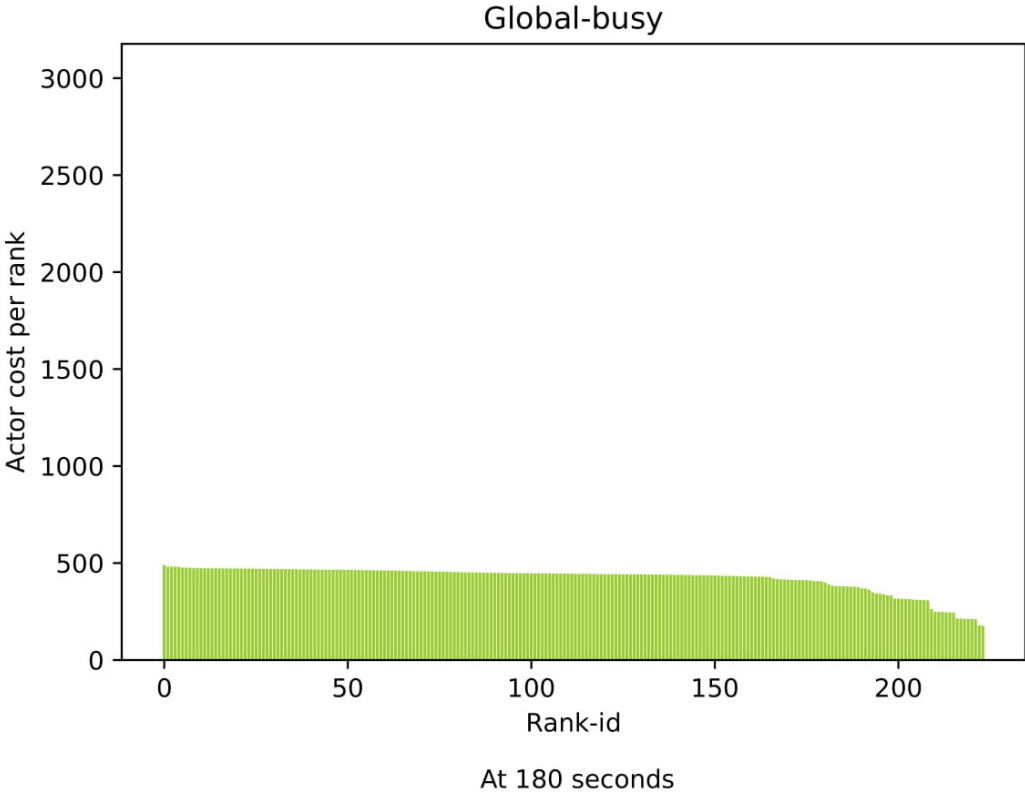


At 720 seconds

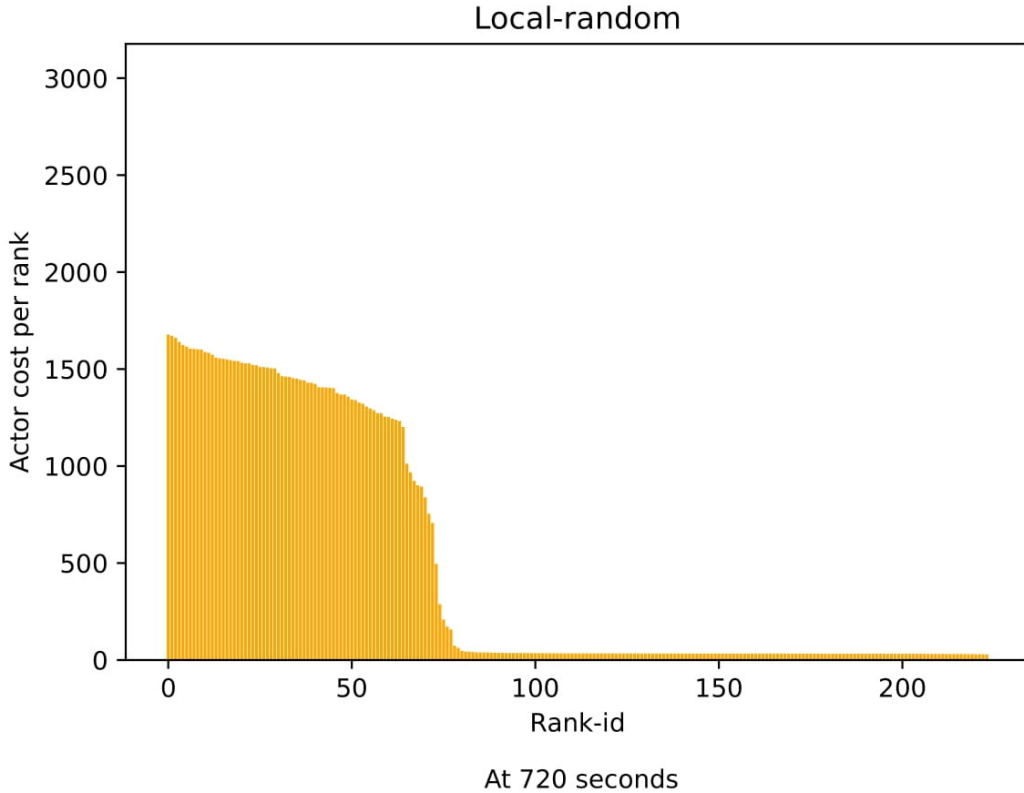
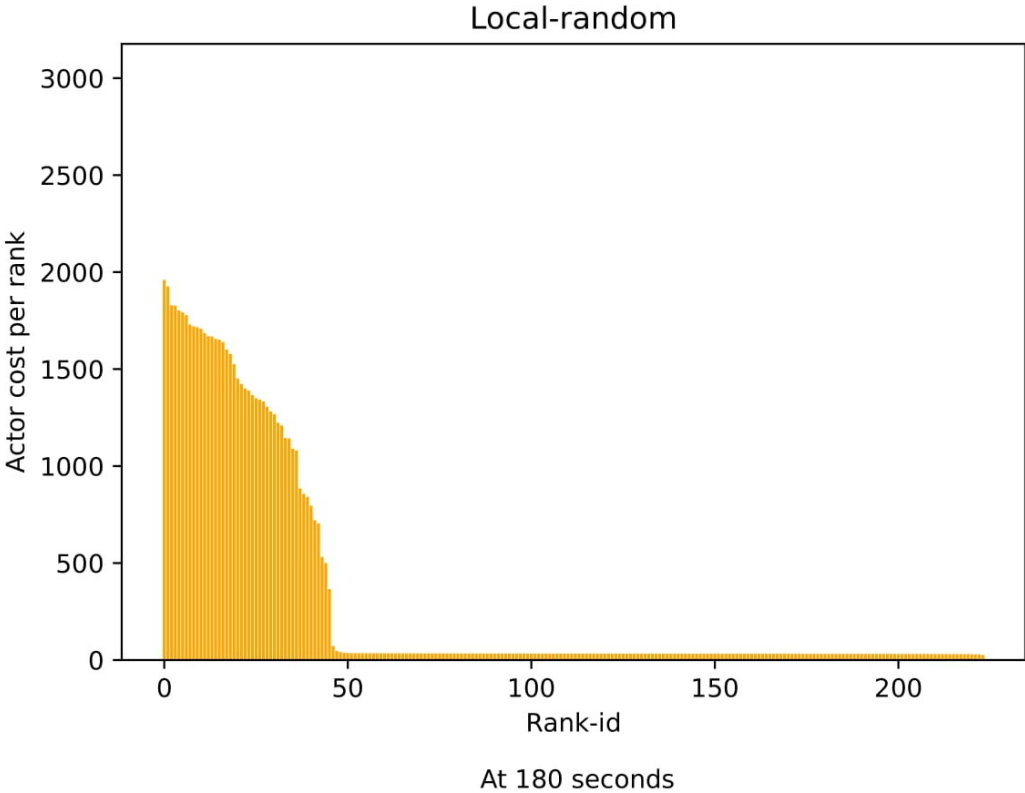
Evaluation: Lazy Activation Scenario – Load Distribution



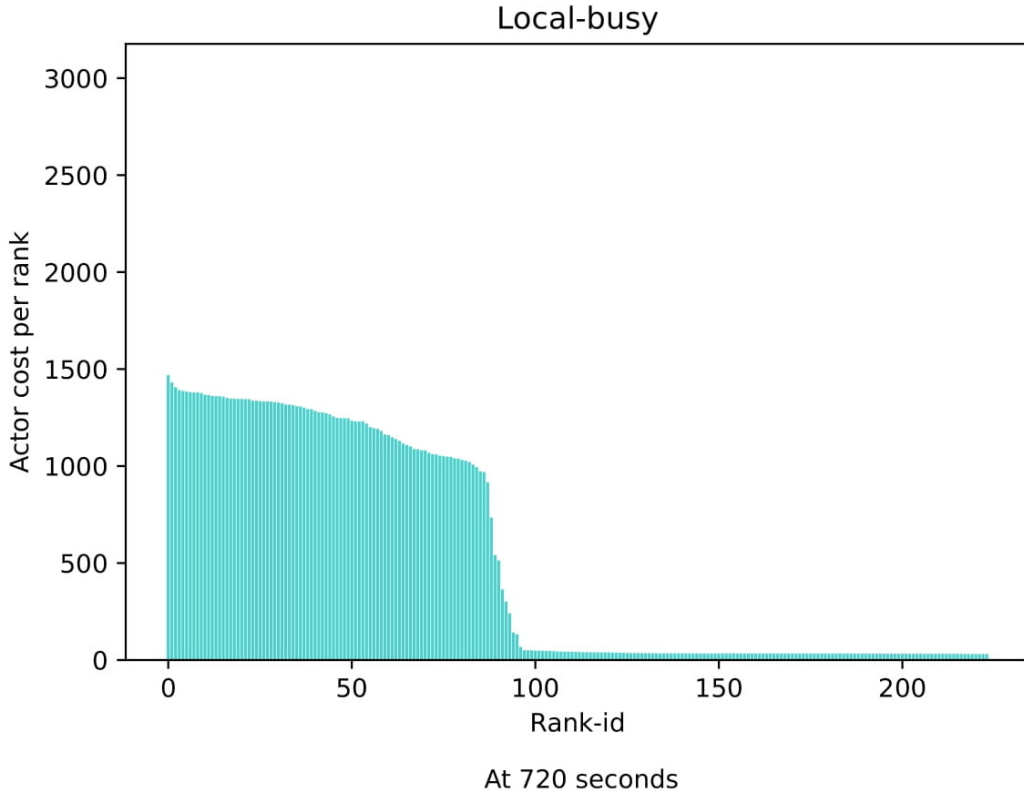
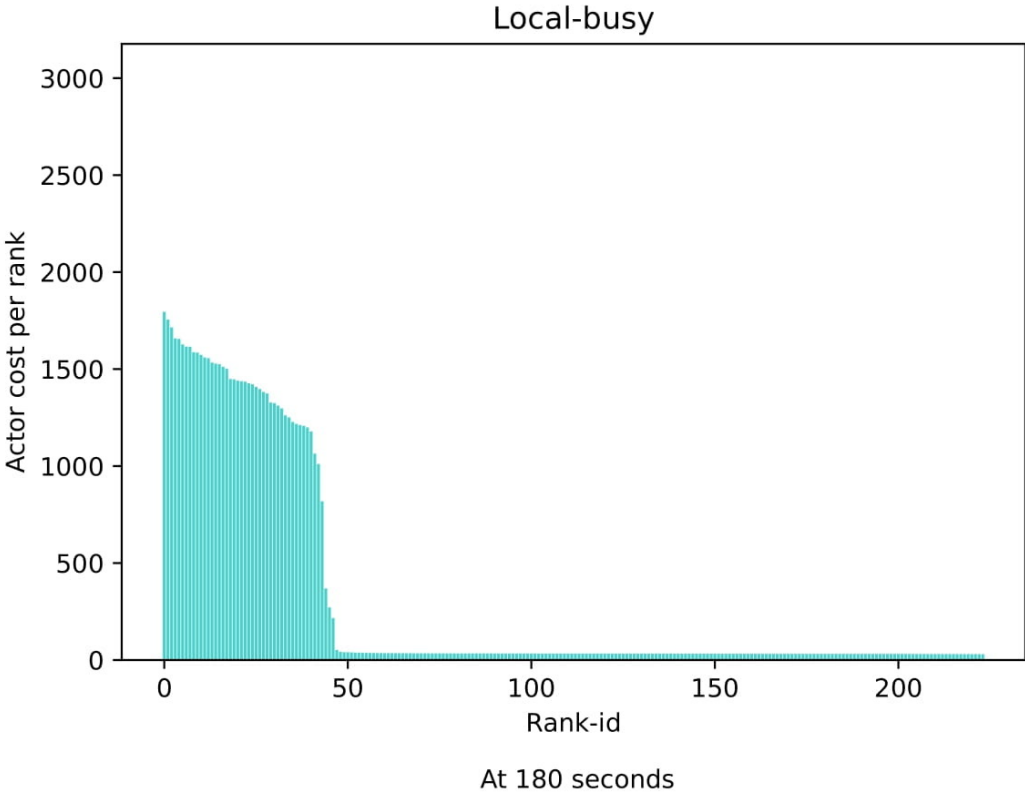
Evaluation: Lazy Activation Scenario – Load Distribution



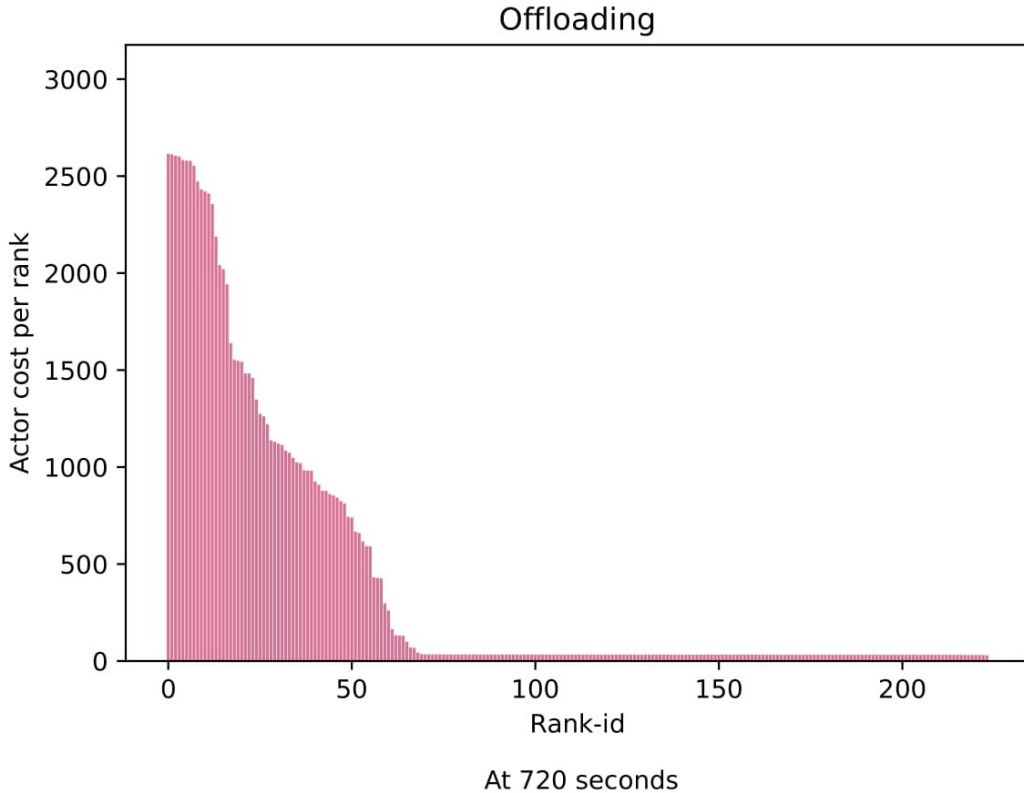
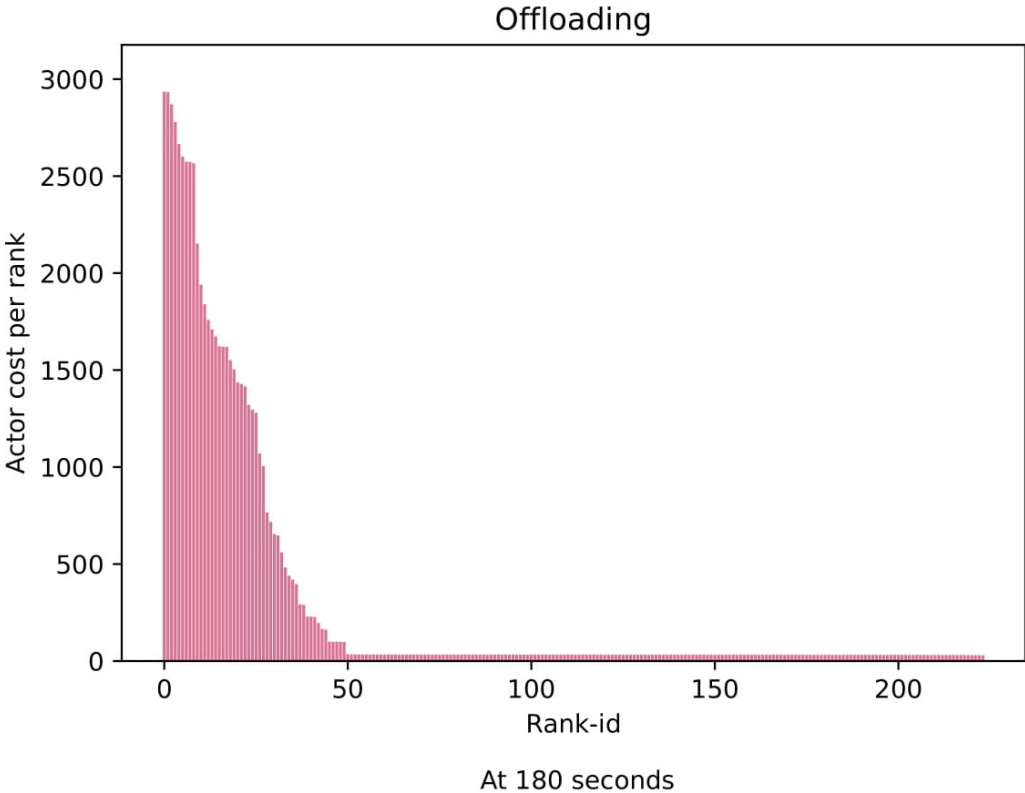
Evaluation: Lazy Activation Scenario – Load Distribution



Evaluation: Lazy Activation Scenario – Load Distribution



Evaluation: Lazy Activation Scenario – Load Distribution



- This research was funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft), Project number 14671743, TRR 89 Invasive Computing
- We would like to thank
 - Philipp Samfass for his advice, which has been of great help during this research and
 - Alexander Pöppl for laying the foundations with Actor-UPCXX and Pond

A UPC++ Actor Library and Its Evaluation On a Shallow Water Proxy Application

Alexander Pöppl
Department of Informatics
Technical University of Munich
Munich, Germany
poeppl@in.tum.de

Scott Baden
Computational Research Division
Lawrence Berkeley National Laboratory
Department of Computer Science and Engineering
University of California, San Diego
baden@ucsd.edu

Michael Bader
Department of Informatics
Technical University of Munich
Munich, Germany
bader@in.tum.de

Abstract—Programmability is one of the key challenges of Exascale Computing. Using the actor model for distributed computations may be one solution. The actor model separates computation from communication while still enabling their overlap. Each actor possesses specified communication endpoints to publish and receive information. Computations are undertaken based on the data available on these channels. We present a library that implements this programming model using UPC++, a PGAS library, and evaluate three different parallelization strategies: one based on rank-sequential execution, one based on multiple threads in a rank, and one based on OpenMP tasks. In an evaluation of our library using shallow water proxy applications, our solution compares favorably against an earlier implementation based on X10, and a BSP-based approach.

Index Terms—Actor-based computation, tsunami simulation, programming models, PGAS

1. INTRODUCTION

With this work, we demonstrate the performance and usability benefits of using the actor model for classical HPC. We will introduce an actor model based on the FunState [1] approach, and its implementation as a library in UPC++. There, we will explore and evaluate three different parallelization strategies for the actor library. We apply the actor model to a tsunami simulation proxy application, and compare its performance against our prior application SWE-X10 based on actorX10, an X10 implementation of our actor library, and SWE, the original tsunami application using MPI and OpenMP with the BSP approach for parallelization. We show that our solution demonstrates significantly higher performance in a weak scaling test, and also a significantly better performance with a lower per-core computational load compared to SWE-X10. We also demonstrate a clear performance benefit compared to SWE.

II. MOTIVATION AND RELATED WORK

The imminent arrival of exascale computing introduced the debate on how to program these machines so that they can

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

978-1-5386-5541-2/18/\$31.00 ©2019 IEEE

offer their expected performance. Currently, many applications still follow the *Bulk Synchronous Parallel (BSP)* model, with clearly defined phases for computation, communication and synchronization. The most widely used approach here is to use MPI for inter-node communication and parallelization, and OpenMP for the on-node parallelization. The BSP approach enables a clear separation of concerns, but the structure, especially with the synchronization step at the end may be too rigid to obtain the best performance. As the number of nodes increases, so will the difficulty of maintaining the pure BSP model, and therefore the burden to the application programmer.

A promising model is the *Partitioned Global Address Space (PGAS)* programming model [2]. This model assumes a global address space, but exposes the separate physical address domains. This may ease the burden on the application programmers, as they no longer need to think about in terms of message-passing, but can access data on remote ranks directly. Another promising model is the *task-based* programming model [3]. Here, the programmer specifies pieces of computation and communication as tasks, and also their dependencies. Afterwards, the resulting task graph is handed to a scheduling system that schedules them onto available computing resources. This model has been implemented in OpenMP [4] and also in runtime systems, for example in StarPU, which enables distributed task scheduling onto heterogeneous machines [5], or the AIIScale project [6], which aims to separate the specification of parallelism from its low-level management on the target hardware. Task-based parallelism has been employed successfully in complex applications, for example in the Uintah application framework [7].

In the Invasive Computing project¹, we investigate novel approaches to use future, parallel and heterogeneous computers [8]. Most of the research is focused around the project's own hardware architecture, a cache-incoherent heterogeneous Multiprocessor System-on-Chip (MPSoC). This architecture features multiple smaller groups of CPU cores (called tiles) that share a cache hierarchy and a memory. The different tiles are connected using a Network-on-Chip. There are different types of tiles, such as tiles containing normal CPU

¹<http://www.invasive.de>

Asynchronous Workload Balancing through Persistent Work-Stealing and Offloading for a Distributed Actor Model Library

Yakup Budanaz
Department of Informatics
Technical University of Munich
Garching, Germany
yakup.budanaz@tum.de

Mario Wille
Department of Informatics
Technical University of Munich
Garching, Germany
mario.wille@tum.de

Michael Bader
Department of Informatics
Technical University of Munich
Garching, Germany
bader@in.tum.de

Abstract—With dynamic imbalances caused by both software and ever more complex hardware, applications and runtime systems must adapt to dynamic load imbalances. We present a diffusion-based, reactive, fully asynchronous, and decentralized dynamic load balancer for a distributed actor library. With the asynchronous execution model, features such as remote procedure calls, and support for serialization of arbitrary types, UPC++ is especially feasible for the implementation of the actor model. While providing a substantial speedup for small- to medium-sized jobs with both predictable and unpredictable workload imbalances, the scalability of the diffusion-based approaches remains below expectations in most presented test cases.

Index Terms—Asynchronous, Actors, Work-stealing, Distributed, Persistent, Offloading, UPC++, Library

I. INTRODUCTION

Numerics of modern scientific applications introduce dynamic workload imbalances. Static mapping of the workload to compute nodes will fall short due to runtime deviations, and dynamic balancing of the workload is fundamental to minimize the time-to-solution and to not waste available resources [1]. For example, in adaptive mesh refinement (AMR, e.g., [2]–[4]), the accuracy of the solution will be adapted for certain regions, dynamically changing the workload in each refinement. In particle simulations, spatial domains decomposition will lead to imbalances when the domain is not homogeneous [5]. Vacuum regions will result in imbalances in workload, and the decomposition of the particles has to be dynamically changed to adapt for best performance. State-space search problems including unbalanced tree search, SAT, and N-Queens are often irregular and show unpredictable workloads [6], and therefore dynamic and predictive workload balancing is mandatory to maintain high performance.

In this work, we consider a solver for the shallow water equations (SWE) that avoids unnecessary computation by lazily activating patches of the computational grid only when a propagating wave enters the patch, thus dynamically changing the workload with each increment of the simulation time [7].

Workload imbalance can also be caused by the hardware, for example with features like dynamic voltage and frequency scaling (DVFS), where the frequency of the CPU is adapted

dynamically to the processing capabilities of each compute node may dynamically differ. Performance variability due to hardware, as reported in [8], can severely impede scalability. Even without faulty hardware run-to-run variability caused by the hardware [9] provides another reason why applications and runtimes need to dynamically migrate workload between compute nodes.

We implement a fully decentralized asynchronous reactive dynamic workload balancing feature for the distributed actor model library Actor-UPCXX¹, implemented with Unified Parallel C++ (UPC++) [10]. UPC++ is a C++ library that implements the asynchronous partitioned global address space model (APGAS). It provides one-sided remote *put* and remote *get* operations, and functions that can be executed on remote UPC++ ranks² called remote procedure calls (RPCs). The actor model [11] is an asynchronous message-driven model of concurrent computation, where the actor is the universal primitive model. Actors do not share their state (i.e., any simulation data), but communicate only through asynchronous one-sided messages. The messages sent are limited in size and the received messages are stored in buffers until their recipient consumes them. Discrete states of actors prevent data races and side effects, enabling the actor model for distributed computing. Various industry-oriented implementations of the actor model are already in use, such as Erlang [12] and the C++ Actor Framework [13]. The actor model is also a popular choice in network frameworks such as the Akka framework for Scala and Orleans [14]; the framework for .Net, Charm++ [15] implements a computational model similar to the actor model and is being used in high-performance systems.

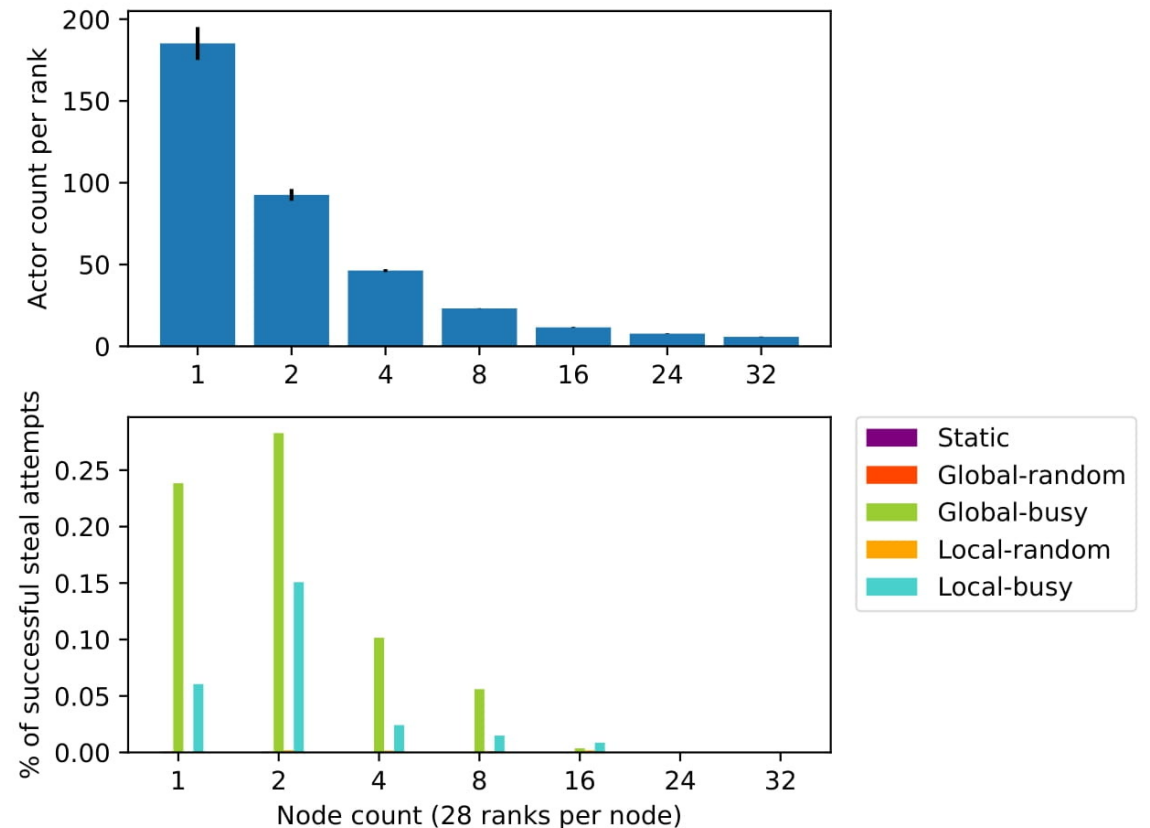
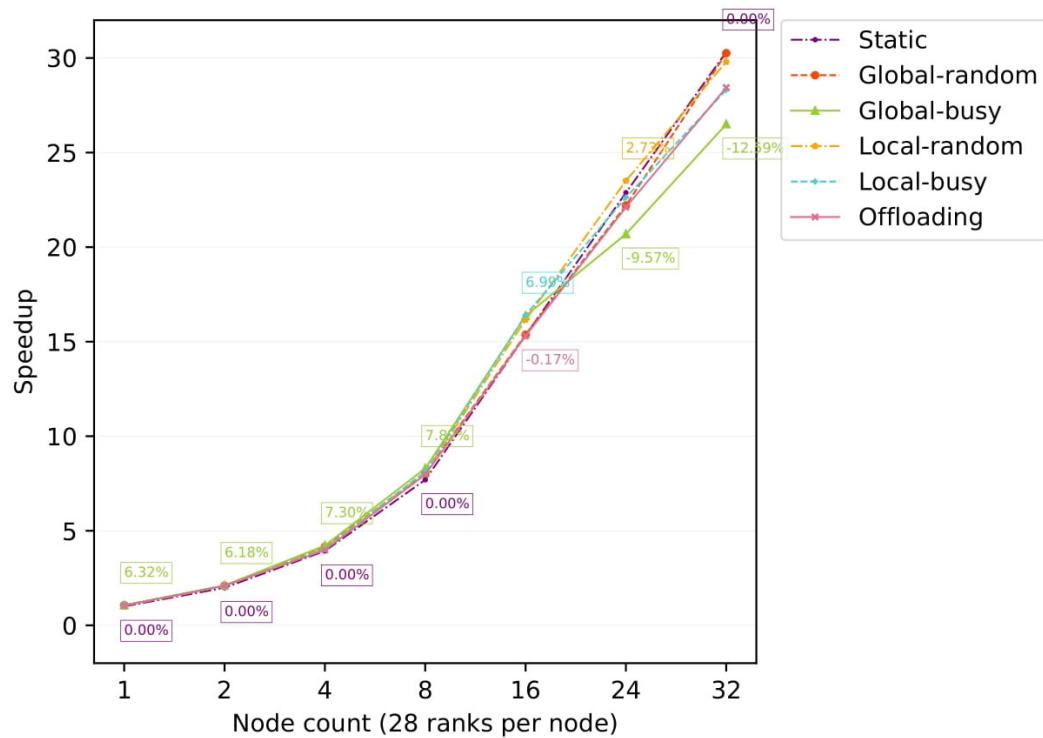
We present a simple diffusion-based approach [16, e.g.] for dynamic workload balancing in Actor-UPCXX and support both stealing and offloading of the workload, by persistently transferring actors between compute nodes. Actor stealing is based on work stealing [17, e.g.], where underloaded ranks steal actors from their overloaded neighbors; actor offload-

¹Available under GPL at <https://github.com/TUM-15/Actor-UPCXX>
²From hereon, we just refer to UPC++ ranks as ranks

- Fully asynchronous and decentralized dynamic workload balancing scheme
- UPC++ accelerates the implementation of the migration strategies
- UPC++ allows easy implementation of serialization to migrate actors
- Asynchronous nature of UPC++ enables the implementation of actor migration as a chain of RPCs
- Implemented strategies for dynamic load balancing improve runtime in predictable and unpredictable load imbalances
- Achieve speedup of up to 400% compared to the static base case with no actor migration

Backup Slides

- Initial workload distribution of the actors modeled as a graph partitioning problem
- Static mapping of actors to compute nodes is calculated with METIS¹



(1) <http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>

Evaluation: Node Slowdown Scenario – Speedup

- Artificial scenario to test the performance under unpredictable workload imbalance
- A subset of ranks is slowed down artificially

