

MPIWasm: Executing WebAssembly on HPC Systems

APART Workshop February 2024

Mohak Chadha: mohak.chadha@tum.de

Chair of Computer Architecture and Parallel Systems (CAPS)

Technical University of Munich

Germany

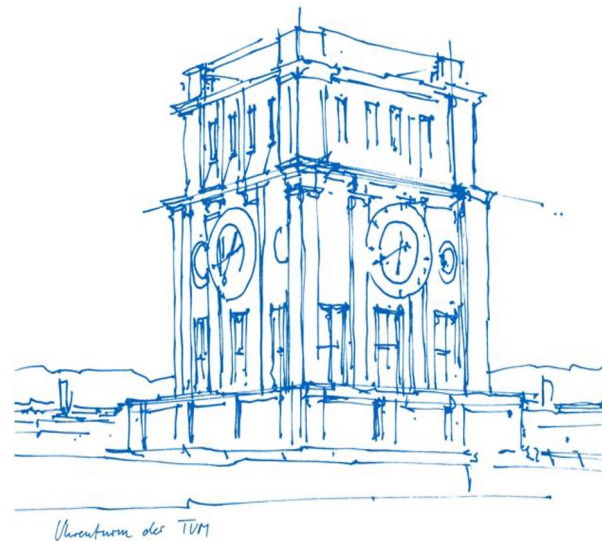



Table of Contents

 Motivation

 What is WebAssembly?

 MPIWasm

 Evaluation

 Future Directions

Rise of Containers in HPC



[SC'17]



[PloS one'17]



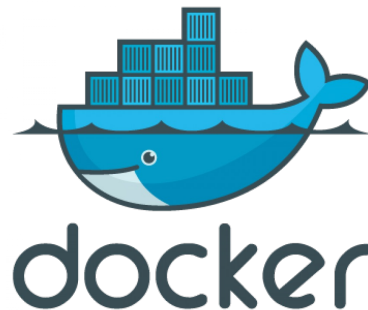
[ISC'19]



[J. Phys.'17]

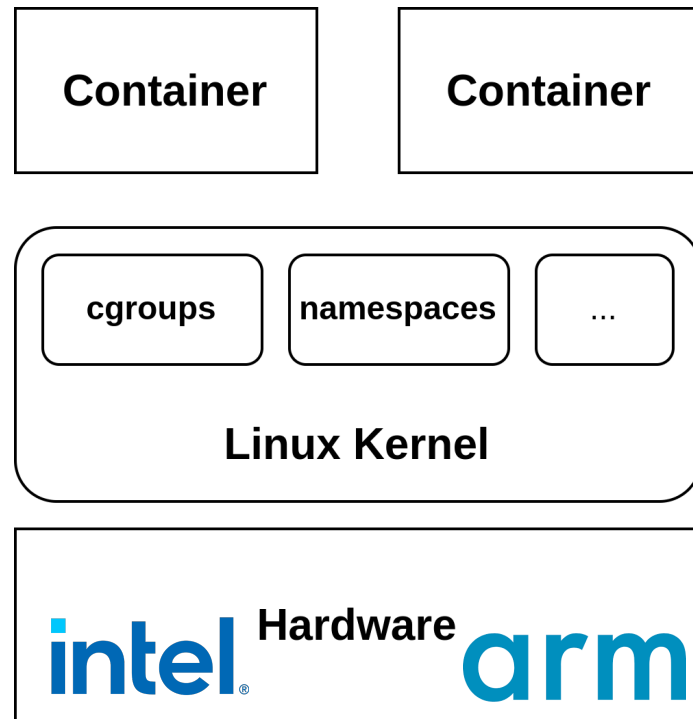


[Redhat, ISC'19]



What are containers? (Recap)

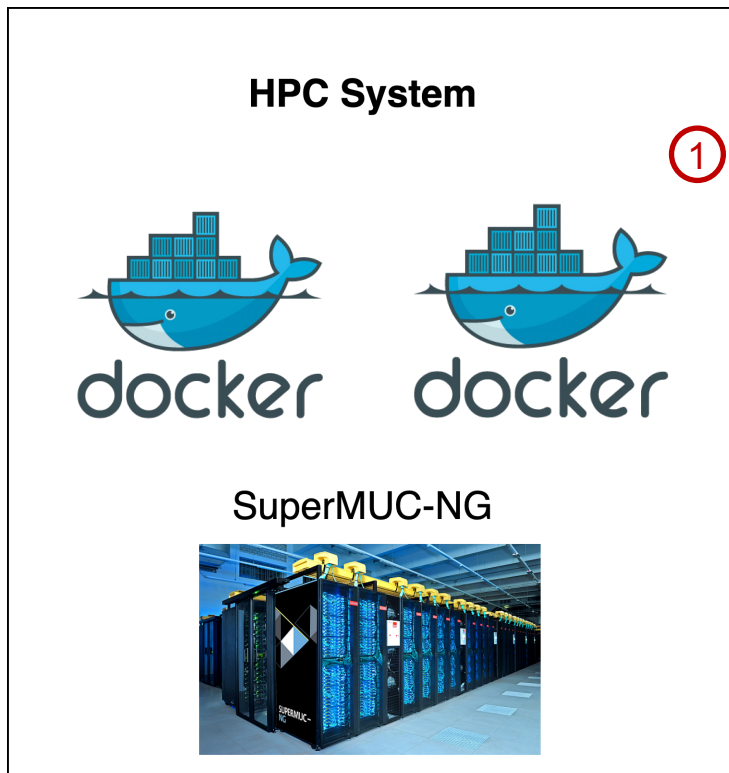
- Using Linux primitives.
- Share Linux Kernel.
- Fast Starts, minimal overheads.
- Flexible Isolation.



Why containers in HPC?

Enabling custom user-defined software stacks

Challenges in Container-based HPC application Development



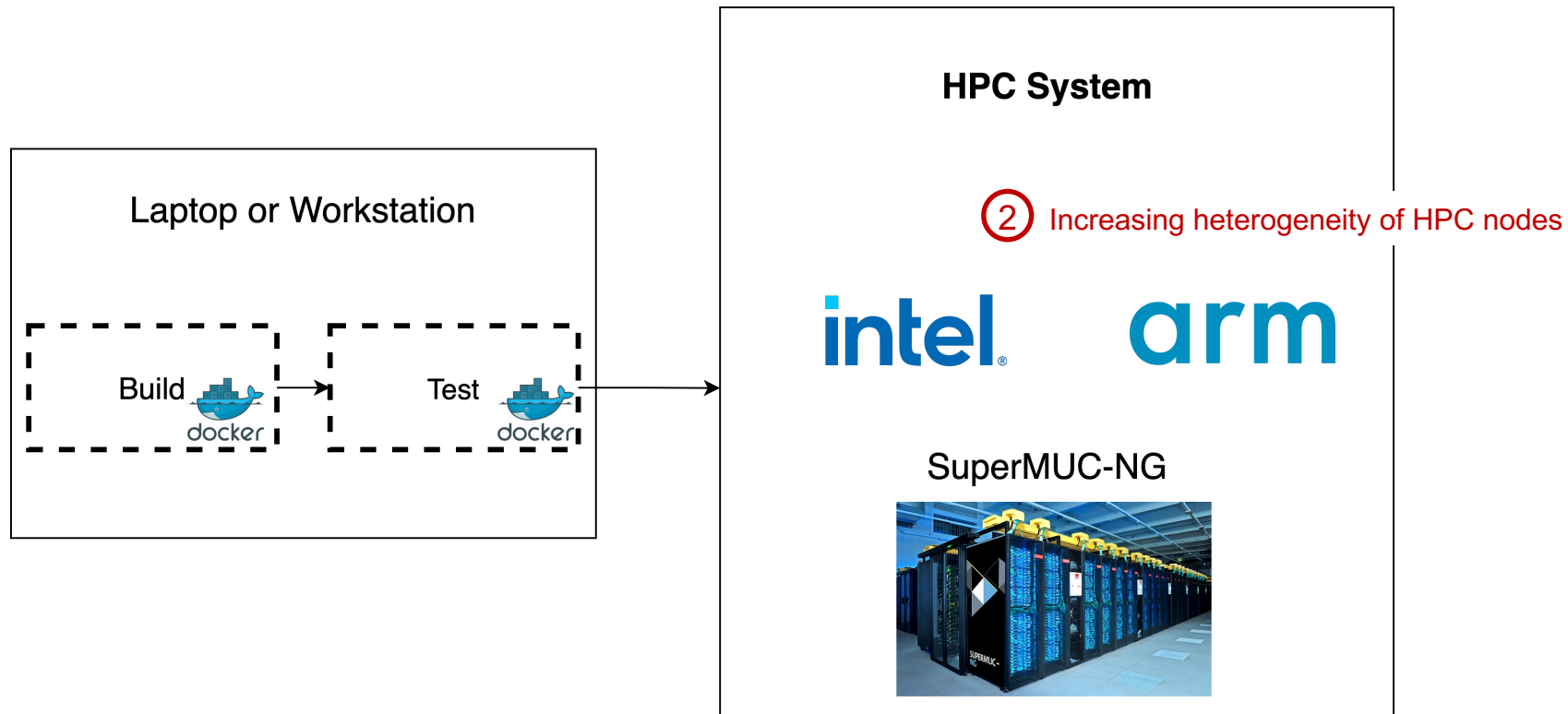
Only, 8% of the total jobs at NERSC use containers [2018]

NOTE: Unsupported file systems in rootless mode

The Overlay file system (OverlayFS) is not supported with kernels prior to 5.12.9 in rootless mode. The fuse-overlays package is a tool that provides the functionality of OverlayFS in user namespace that allows mounting file systems in rootless environments. It is recommended to install the fuse-overlays package. In rootless mode, Podman will automatically use the fuse-overlays program as the mount_program if installed, as long as the \$HOME/.config/containers/storage.conf file was not previously created. If storage.conf exists in the homedir, add `mount_program = "/usr/bin/fuse-overlays"` under `[storage.options.overlay]` to enable this feature.

The Network File System (NFS) and other distributed file systems (for example: Lustre, Spectrum Scale, the General Parallel File System (GPFS)) are not supported when running in rootless mode as these file systems do not understand user namespace. However, rootless Podman can make use of an NFS Homedir by modifying the \$HOME/.config/containers/storage.conf to have the `graphroot` option point to a directory stored on local (Non NFS) storage.

Challenges in Container-based HPC application Development



Challenges in Container-based HPC application Development

Mostly x86_64

Laptop or Workstation

Build



docker

Test



docker

Build for aarch64

Uninformative

HPC System

② Increasing heterogeneity of HPC nodes

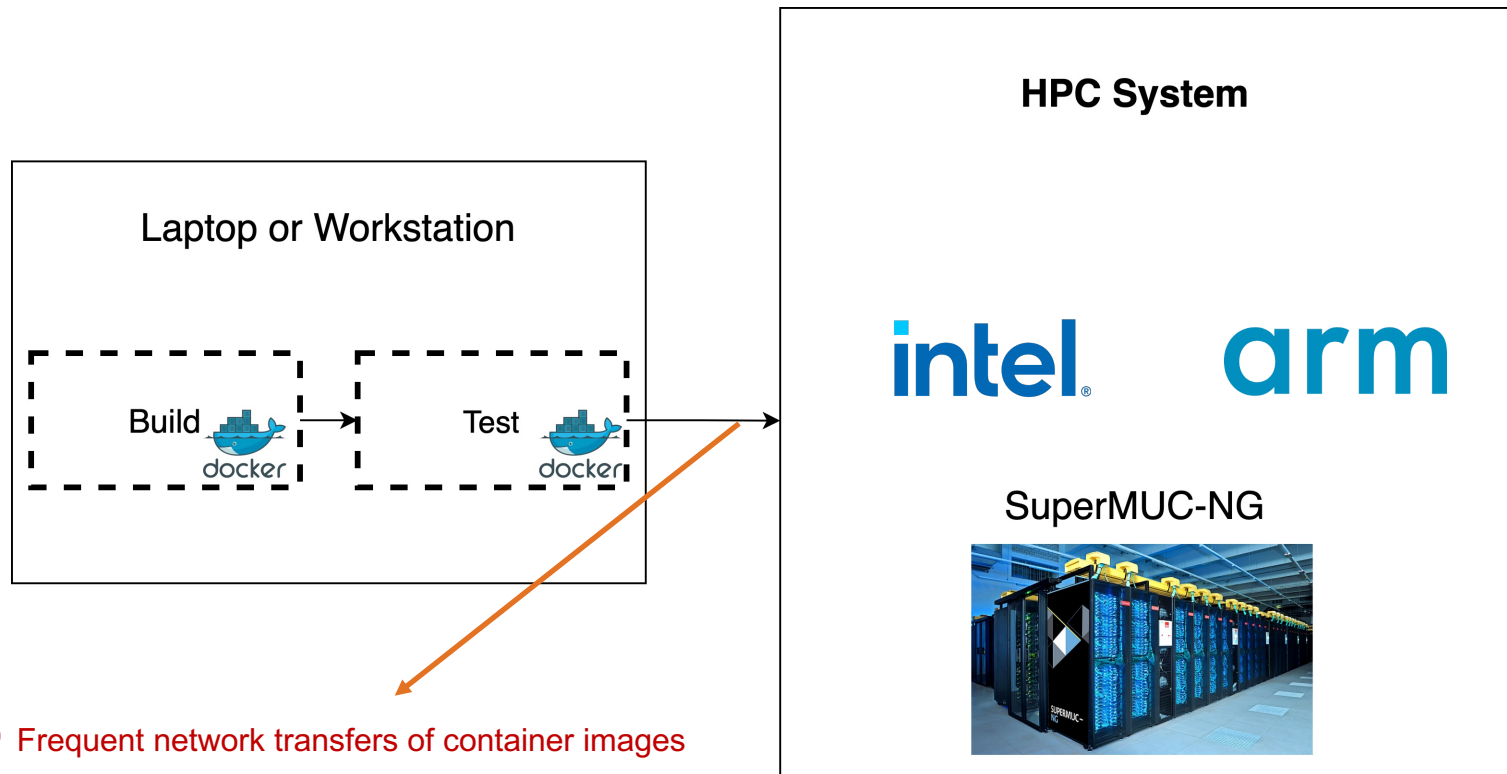
intel®

arm

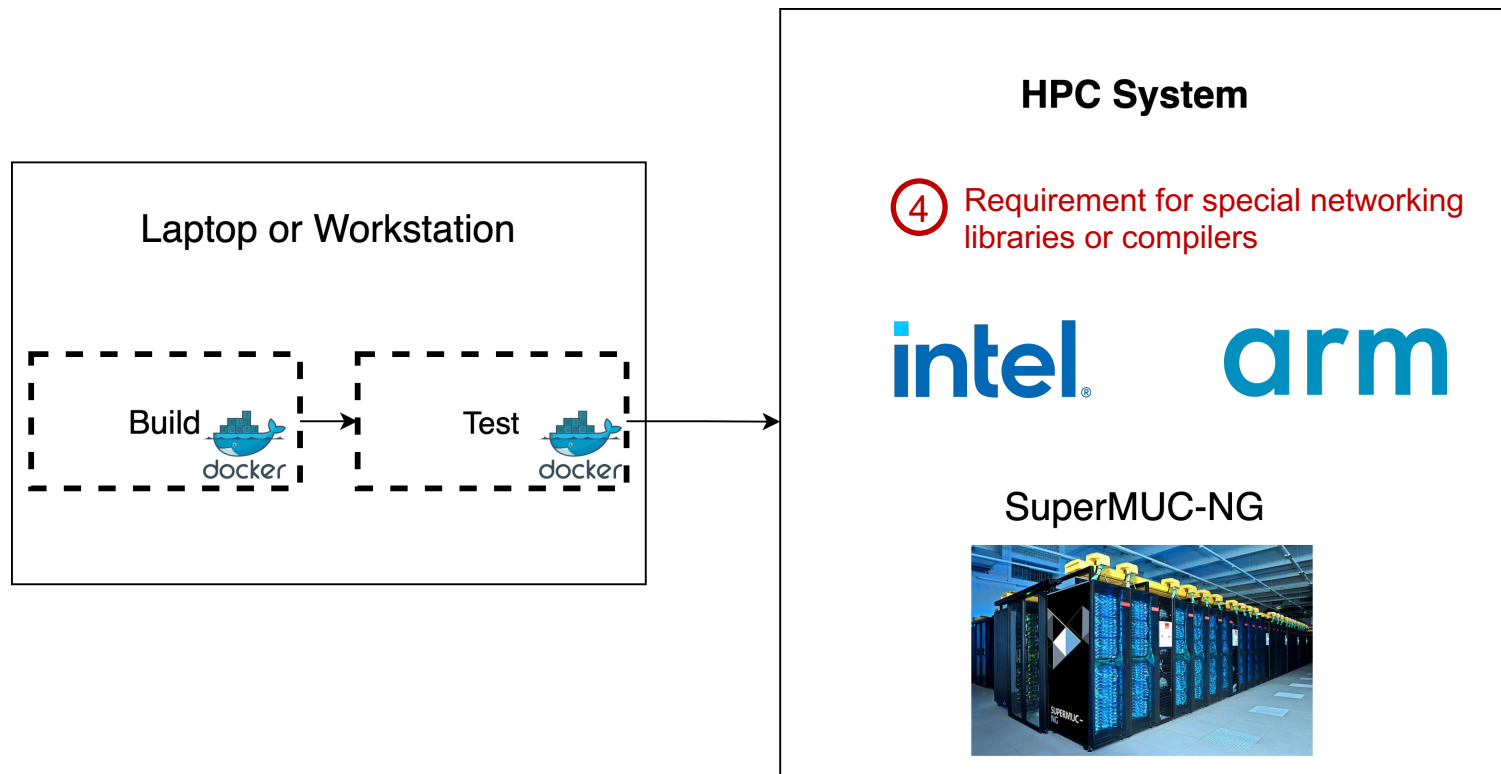
SuperMUC-NG



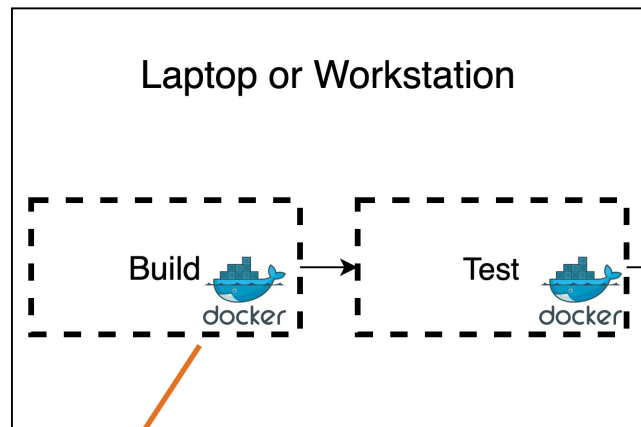
Challenges in Container-based HPC application Development



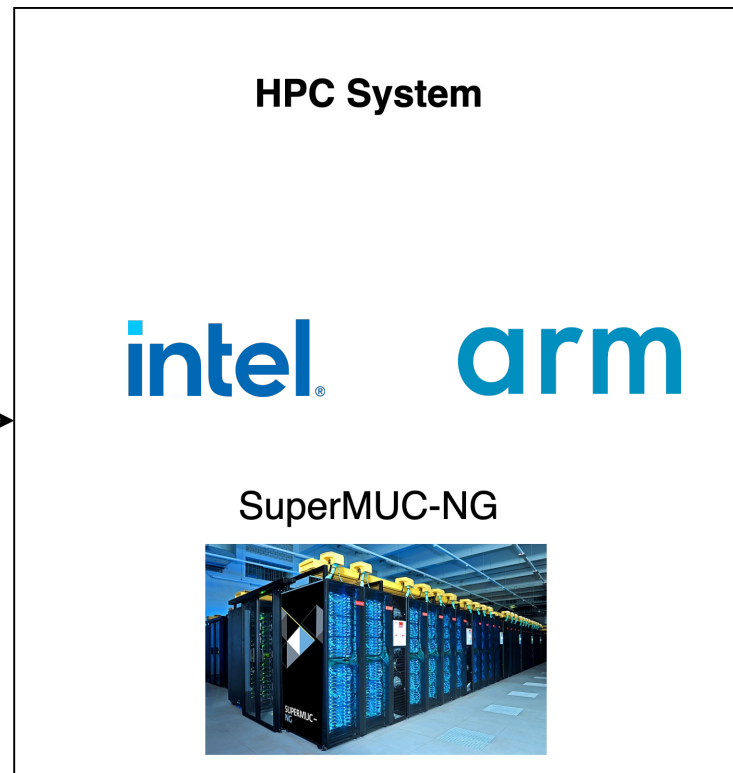
Challenges in Container-based HPC application Development



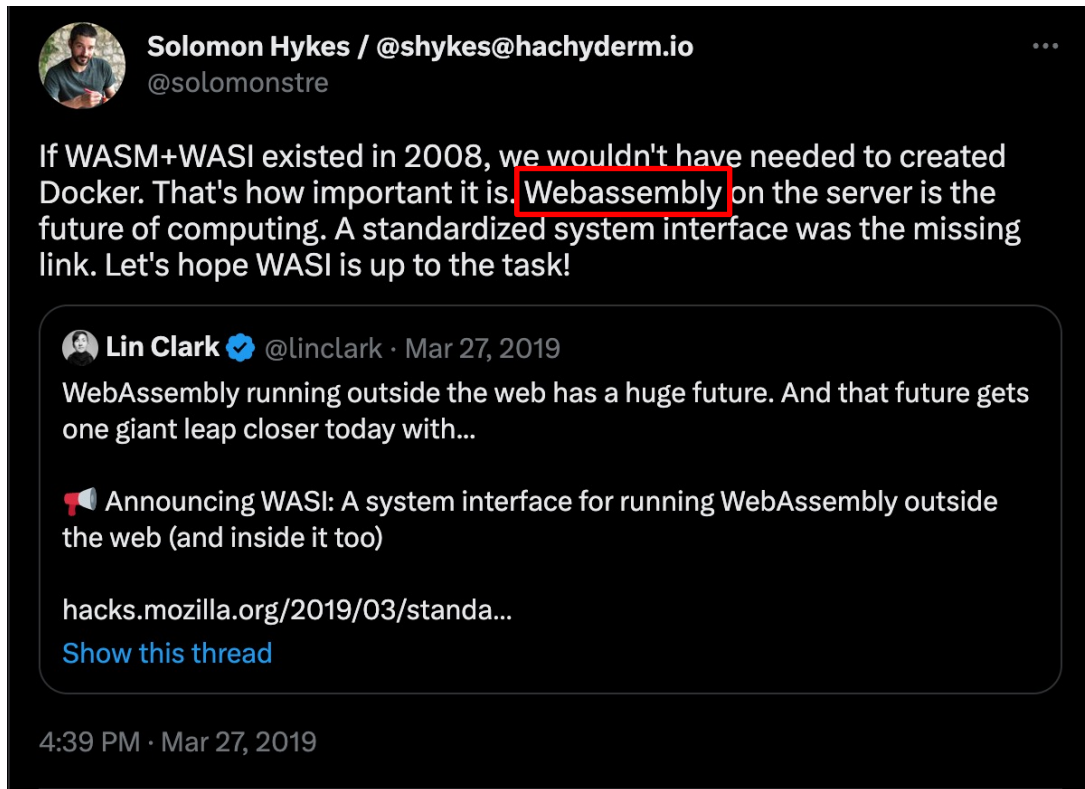
Challenges in Container-based HPC application Development




⑤ Building high-performant application container images.



Alternative to containers?



Introduction: WebAssembly (Wasm)

 Binary format, with alternative human-readable text representation

 Virtual ISA

 Linear 32-bit memory space

 Lightweight userspace isolation mechanism

 Import/export system for granting capabilities



Introduction: WASI: Wasm System Interface



Standardized non-Web system-oriented API for Wasm



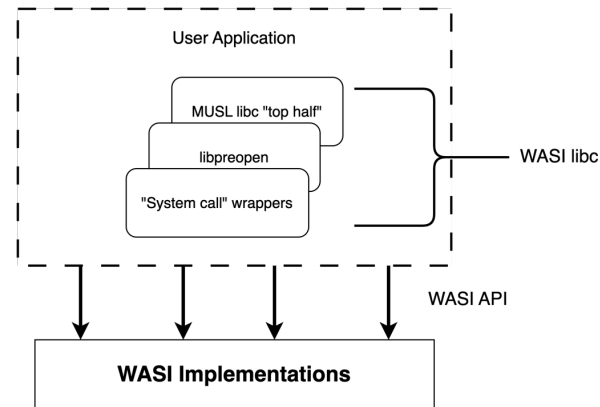
Capability-oriented



Portable



Custom libc implementation integrated into WASI-SDK



WebAssembly Embedders

Wasmtime



Wasmer

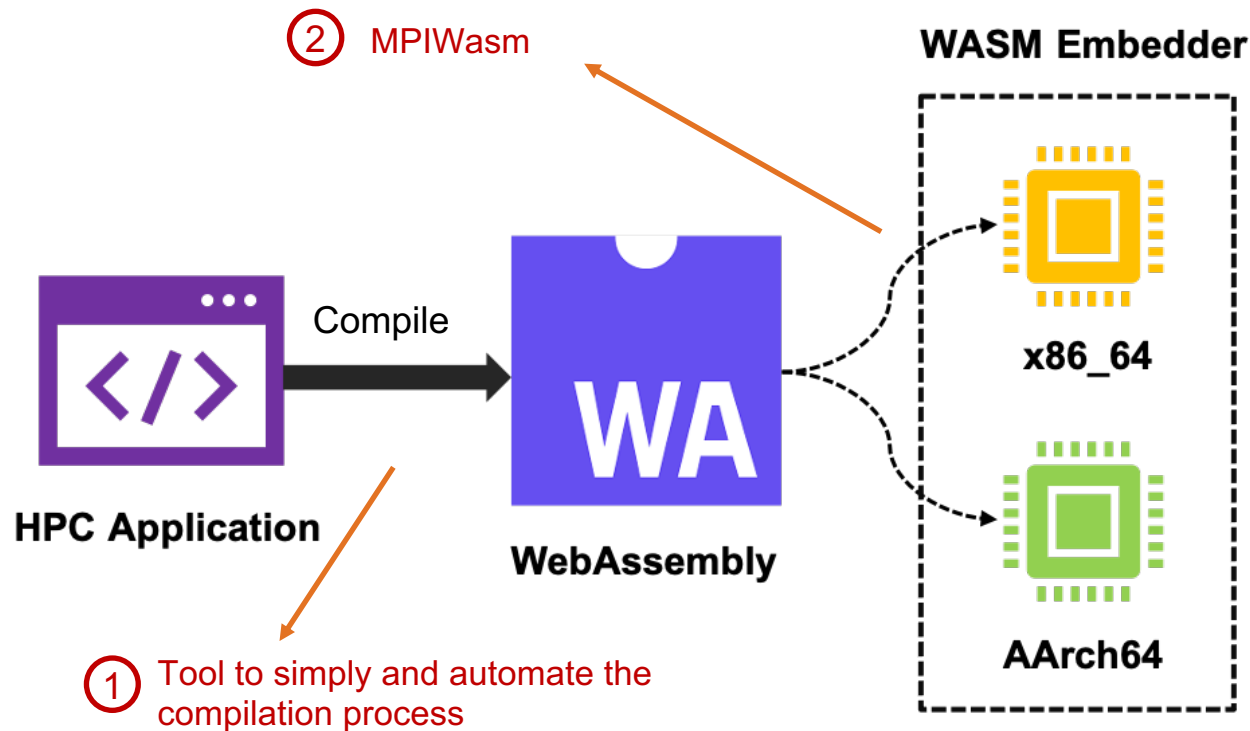


WasmEdgeRuntime

Wasm-micro-runtime (WAMR)



What we did?



Compiling MPI applications to Wasm

① Extend the WASI-SDK

```
typedef int MPI_Comm;
typedef int MPI_Datatype;

...
int MPI_Init(int* argc, char*** argv);
int MPI_Finalize(void);
int MPI_Send(
    const void* buf, int count, MPI_Datatype datatype,
    int dest, int tag, MPI_Comm comm
);
int MPI_Recv(
    void* buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Status* status
);
```

Clang

```
(import "env" "MPI_Init" (func $MPI_Init (param i32 i32) (result i32)))
(import "env" "MPI_Finalize" (func $MPI_Finalize (result i32)))
(import "env" "MPI_Send" (
    func $MPI_Send (param i32 i32 i32 i32 i32 i32) (result i32)
))
(import "env" "MPI_Recv" (
    func $MPI_Recv (param i32 i32 i32 i32 i32 i32) (result i32)
))
```

② Custom python-based tool.



Wasm Module



OPEN MPI



MVAPICH

MPIWasm



Extends Wasmer.



Support for C/C++ applications conforming to MPI-2.2 standard.



Support for both x86_64 and aarch64 processors.



High performance execution of MPI-based Wasm modules.



Low-overhead for MPI calls through zero-copy memory operations.



Support for high-performance network interconnects.



Wasmer

Executing Wasm Code with High-Performance



AoT Compilation



Caching mechanism for generated machine code.

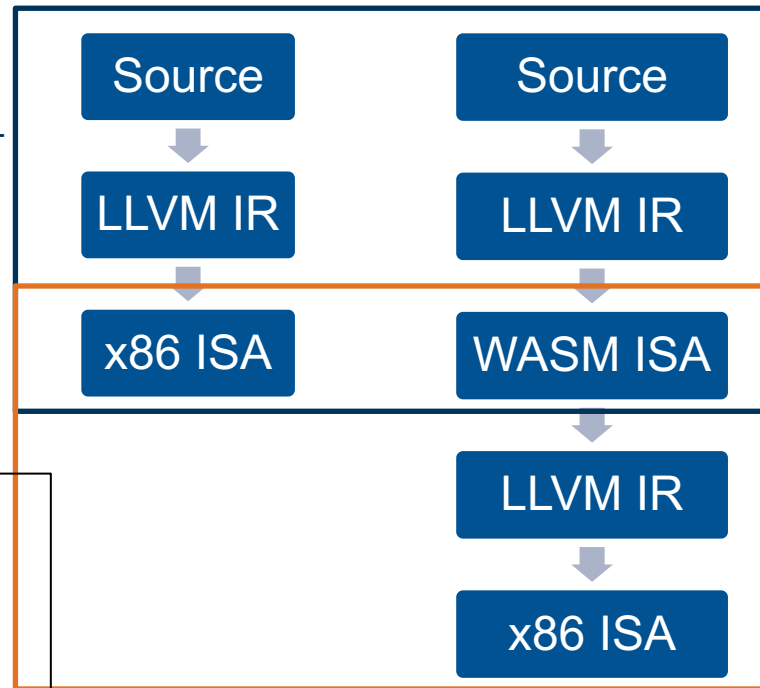


```
$ tree ./cache
./cache
  de5afe5d4f24cf986e1fe1c3e614304a4d860f14d6794e0e99a97ff38887cafe
  e025e7872b4c6b4d852cfc475472f7ab2bc7f67654a31597a0e1076978f939d1
0 directories, 2 files
```

```
$ file cache/de5afe5d4f24cf986e1fe1c3e614304a4d860f14d6794e0e99a97...
ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, not stripped
```

Compiler

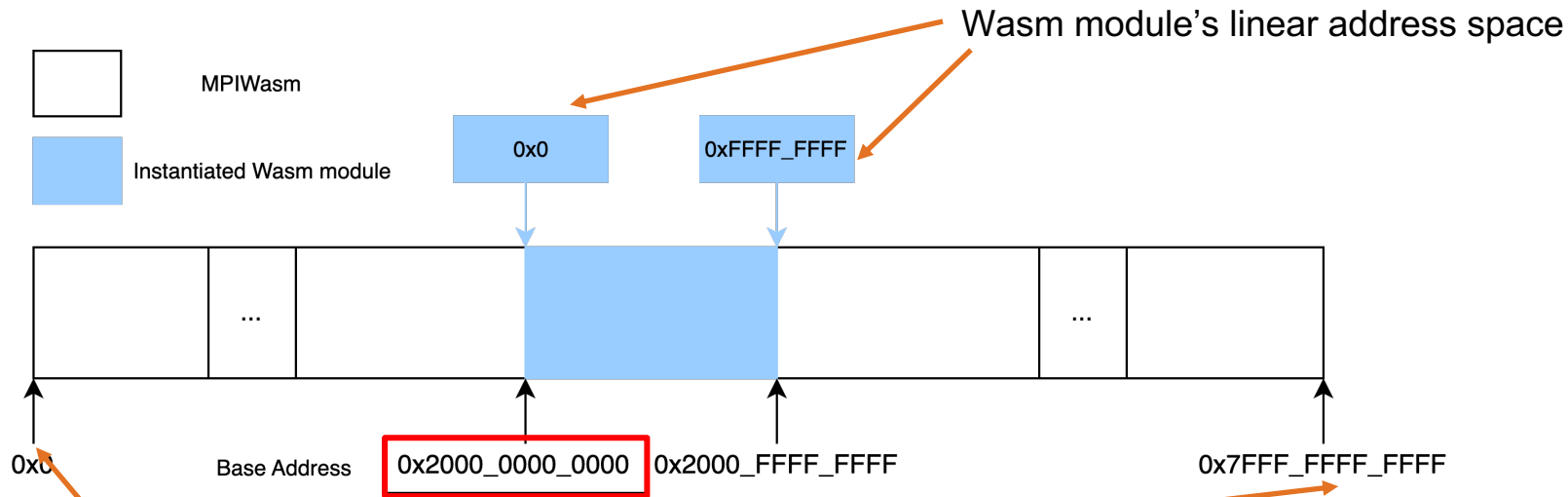
Embedder



Native (left) vs. Wasm (right) code generation flow for LLVM-based compiler and MPIWasm.

Memory Address Translation

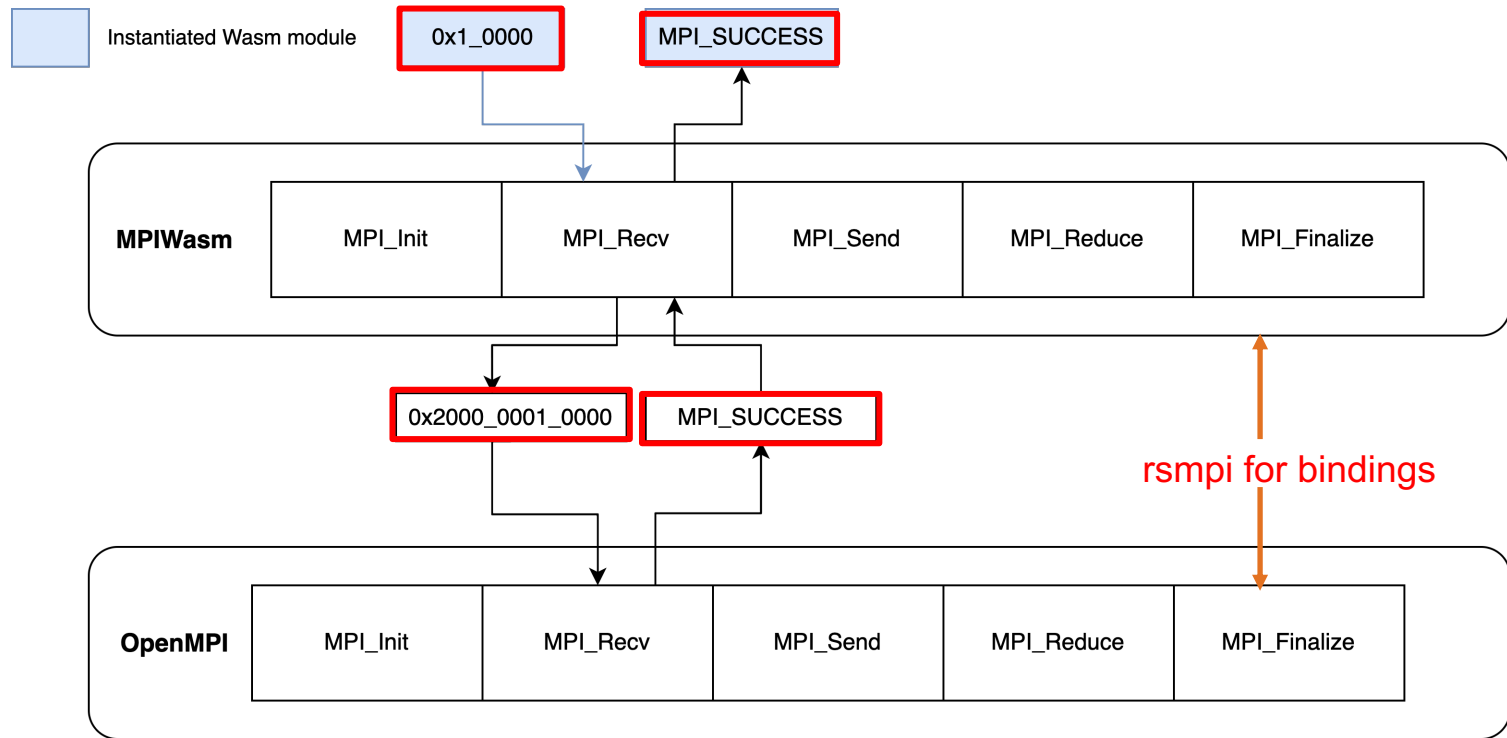
WASM to host memory address conversion with **base offset** of linear memory address space



MPIWasm's address space

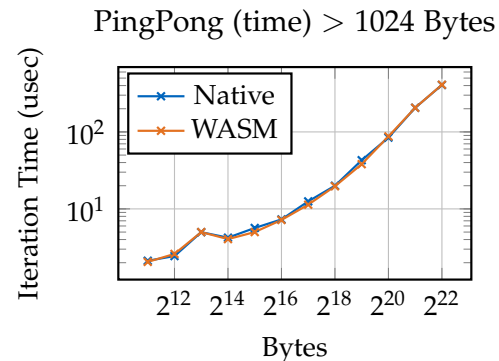
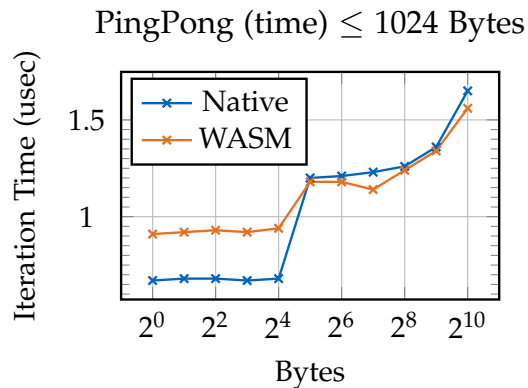
```
pub struct WasmPtr<T, M: MemorySize = Memory32>
```

Implementing MPI Functions

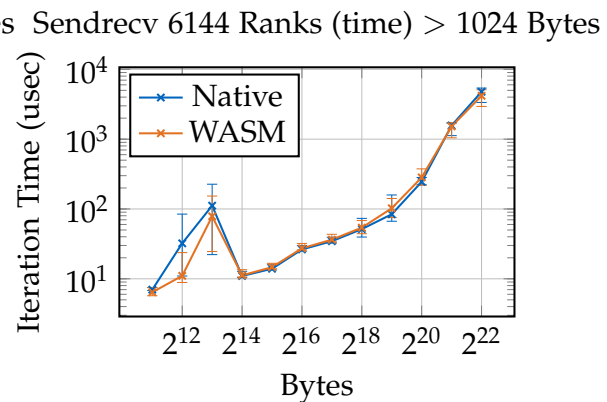
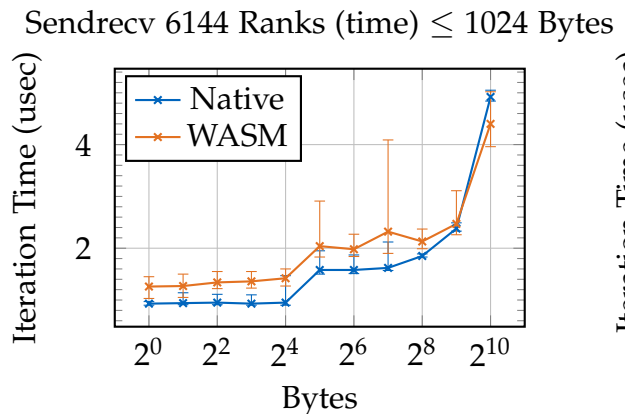


PingPong and SendRecv (x86_64)

 PingPong: 0.05x GM average slowdown

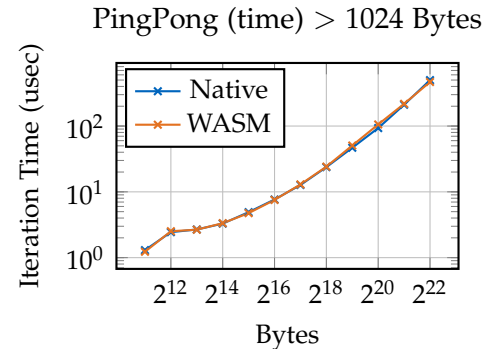
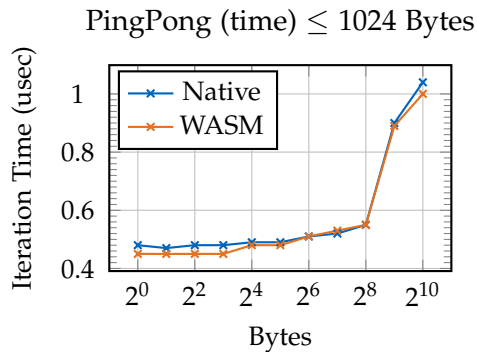


 SendRecv: 0.06x GM average slowdown

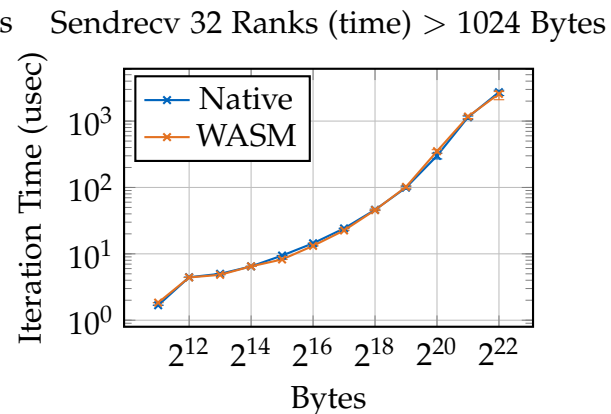
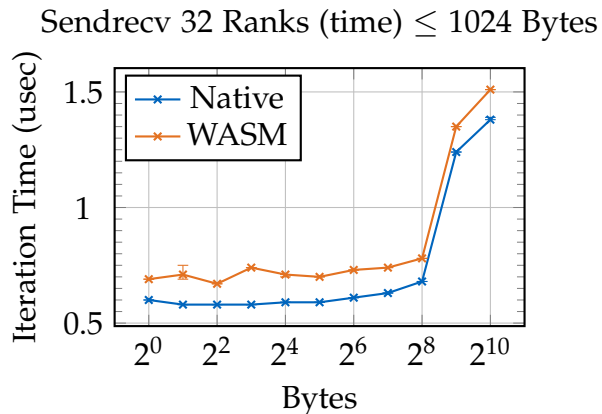


PingPong and SendRecv (aarch64)

 PingPong: 1.01x GM average speedup



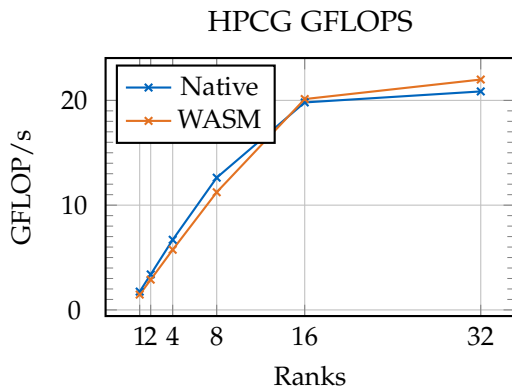
 SendRecv: 0.07x GM average slowdown



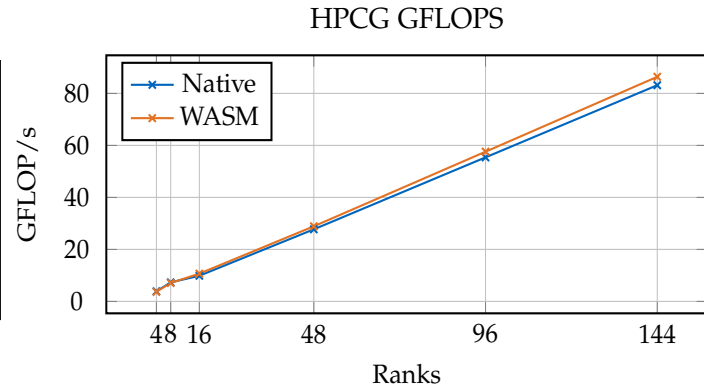
HPCG



Similar performance upto 192 MPI processes



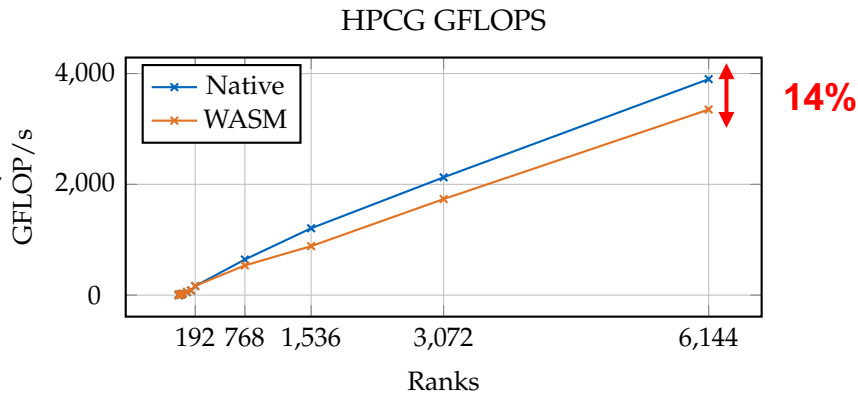
aarch64



x86_64



Translation overhead adds up for higher number of processes



x86_64

Into the Future: Wasm and HPC

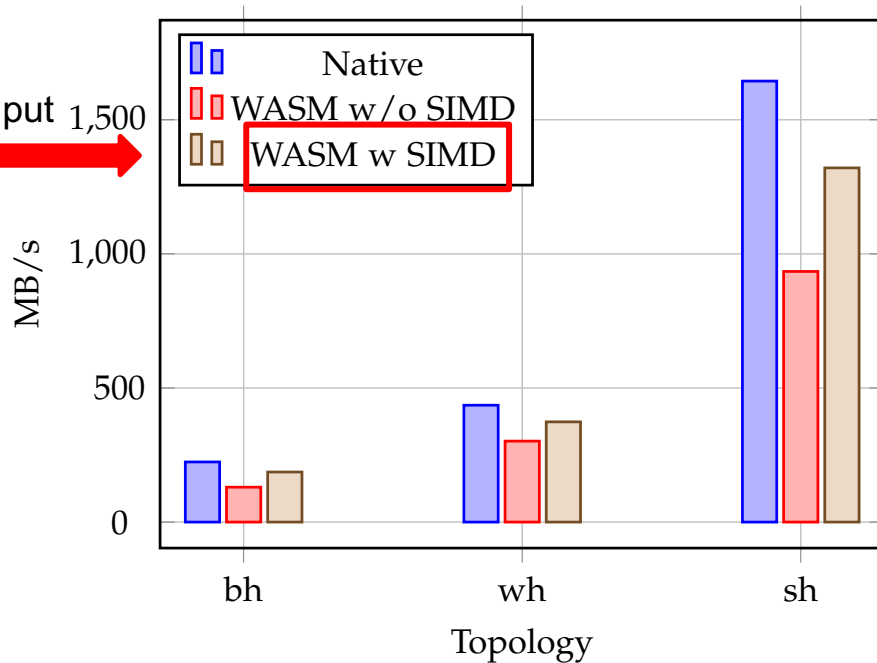


Wasm Extended SIMD

1.36x better throughput



DT Total Throughput



More Details

Exploring the Use of WebAssembly in HPC

Mohak Chadha, Nils Krueger, Jophin John,
Anshul Jindal, Michael Gerndt
Chair of Computer Architecture and Parallel Systems,
Technische Universität München, Germany

Shajulin Benedict
Department of Computer Science and Engg., Indian
Institute of Information Technology Kottayam, Kerala

Abstract

Containerization approaches based on *namespaces* offered by the Linux kernel have seen an increasing popularity in the HPC community both as a means to isolate applications and as a format to package and distribute them. However, their adoption and usage in HPC systems faces several challenges. These include difficulties in unprivileged running and build-

ACM Reference Format:

Mohak Chadha, Nils Krueger, Jophin John, Anshul Jindal, Michael Gerndt and Shajulin Benedict. 2023. Exploring the Use of WebAssembly in HPC. In *The 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '23)*, February 25-March 1, 2023, Montreal, QC, Canada. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3572848.3577436>



Scan Me

Conclusion



WASM ecosystem has the potential to provide competitive performance in the HPC domain.



WASM enables the scientific community to benefit from:



CPU Architecture and OS portability



Capabilities-based security model

Questions?



Key Takeaways:

- ❑ Wasm and HPC is an exciting research direction.
- ❑ MPIWasm delivers competitive native application performance.
- ❑ Support for x86_64 and aarch64 architectures.
- ❑ Support for applications written with the MPI-2.2 standard.
- ❑ Support for OpenMPI and MVAPICH.

Thank you for your attention!

Find Us:



MPIWasm:

