GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Evaluation of Container Solutions for HPC Systems

Abdallah Abdelnaby

MatrNr: 19334664

Supervisor: Azat Khuziyakhmetov
Requested Credits: 9 credits

Georg-August-Universität Göttingen
Institute of Computer Science

March 22, 2025

Abstract

This paper presents a performance evaluation of containerized and native high-performance computing (High-Performance Computing (HPC)) environments, focusing on configurations like Virtual Native HPC, Podman and Apptainer; and Real Native HPC, and Apptainer. Metrics such as core time, wall time, throughput, load balancing, Input Output Operation (I/O) performance, and memory usage were analyzed across 1 Thread/Core and 4 Threads/Core configurations.

Results showed that Podman leads in core and wall time efficiency and excels in I/O operations, making it suitable for data-intensive workloads. Apptainer, with slightly less performance, offers robust security benefits through its rootless architecture. Native execution, particularly with A (AVX-512) optimization, provides high computational throughput but at the cost of longer execution times. These findings highlight the trade-offs between performance, resource efficiency, and security in HPC environments. Future work will broaden the scope to include more container engines and varied benchmarking scenarios.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☐ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☐ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☐ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

HPC High-Performance Computing

I/O Input Output Operation

CPU Central Processing Unit

GB Gigabyte(s)

NFS Network File System

OCI On-demand Cloud Computing Interface

SLURM Slurm Workload Manager

Spack APackage Manager for Supercomputers

RAM Random Access Memory

x86 x86 is a family of complex instruction set computer (CISC) instruction set architectures initially developed by Intel.

AVX-512 AVX-512 are 512-bit extensions to the 256-bit Advanced Vector Extensions SIMD instructions for x86 instruction set architecture proposed by Intel in July 2013.

AVX2-256 An expansion of the AVX instruction set introduced in Intel's Haswell microarchitecture. AVX2 makes SSE and AVX instructions to 256 bits

1 Introduction

High-Performance Computing (HPC) has become an indispensable tool for solving complex scientific and engineering problems. The increasing demand for computational power and efficiency has led to the adoption of container technologies in HPC environments. Containers offer a lightweight, portable, and scalable solution for deploying and managing applications across heterogeneous systems, facilitating reproducibility and ease of use [Mül+22; App23].

The evolution of container technologies has been driven by the need to address specific challenges in HPC, such as resource allocation, security, and compatibility with existing cluster infrastructures. Traditional virtualization methods, while effective, often introduce significant overhead, making them less suitable for performance-critical applications. Containers, with their minimal runtime overhead and ability to encapsulate application dependencies, have emerged as a compelling alternative [Fel+15]. Moreover, their compatibility with modern orchestration tools has further enhanced their adoption in large-scale HPC systems [Mül+22].

Among the various container solutions, Podman and Apptainer (formerly Singularity) have gained prominence in HPC environments. Podman is lauded for its daemonless architecture, enabling rootless container execution and enhanced security. It provides a seamless transition for users accustomed to Docker, offering similar command-line interfaces but with added features tailored for secure multi-user environments [App23; Pet23]. On the other hand, Apptainer is specifically designed for scientific computing, prioritizing user isolation and seamless integration with HPC job schedulers, making it a favored choice for researchers [Pet23].

Despite their advantages, the impact of containerization on HPC workloads remains a topic of debate. Factors such as computational throughput, memory usage, and I/O performance can vary significantly depending on the container engine, configuration, and workload type. While early studies have demonstrated the feasibility of containerization in HPC [Fel+15], the nuanced performance trade-offs between different container solutions and native environments continue to warrant in-depth exploration [Mül+22].

The interplay between container design and HPC workload characteristics highlights the need for a comprehensive understanding of performance implications. By examining the nuances of containerized execution, researchers and practitioners can make informed decisions about adopting and optimizing container technologies for HPC applications. The growing body of research underscores the potential of containers to transform HPC workflows, paving the way for enhanced scalability, portability, and resource efficiency.

1.1 Main Objectives

The main objectives of this study are as follows:

- To compare the performance of Podman, Apptainer, and native execution in an HPC environment using various benchmarks and metrics, including core time, wall time, and I/O performance.
- To evaluate the resource utilization of Podman and Apptainer in rootless container configurations, assessing their effectiveness in shared HPC environments.

- To analyze the impact of different workload configurations, such as single-threaded and multi-threaded executions, on the performance of the container systems.
- To run GROMACS on an HPC cluster with a different microarchitecture, comparing the performance of native HPC and Apptainer HPC environments in molecular dynamics simulations.
- To provide a comprehensive analysis of the trade-offs between containerized and native execution, highlighting the strengths and weaknesses of each approach in various HPC scenarios.

The findings of this study aim to contribute to the body of knowledge on containerization in HPC and assist practitioners in making informed decisions when deploying containerized applications in HPC environments.

2 Tools and Softwares Overview

To evaluate the performance of container solutions and native execution, several software tools were employed. These tools were selected to cover a wide range of metrics, including computational performance, I/O operations, memory utilization, and system latency. The following sections provide an overview of these tools and their roles in the evaluation process.

2.1 Gromacs

Gromacs is a widely used molecular dynamics simulation software, optimized for high-performance computing environments. It is primarily employed in the fields of biochemistry and computational biology for simulating the interactions of molecules over time. In this project, Gromacs served as the primary tool for benchmarking computational performance. Several predefined benchmarks, including `water-cut1.0_GMX50_bare`, `adh_dodec`, `adh_cubic`, `rnase_cubic`, and `rnase_dodec`, were collected from here [GRO24] and executed to collect data on various performance metrics. These benchmarks involved simulating systems with varying levels of complexity, providing insights into the computational efficiency and scalability of containerized and native execution.

2.2 Sysbench

Sysbench is a multi-threaded benchmarking tool designed to evaluate system performance under various workloads. It is versatile and can measure CPU performance, I/O operations, memory utilization, and database performance, making it an ideal choice for this project.

2.3 Container Tools: Podman and Apptainer

The project involved evaluating two container solutions: Podman and Apptainer. Podman is a root-dependent container engine designed for developing, managing, and running OCI-compliant containers. It offers flexibility and is particularly suited for environments where root access is available. In this project, Podman was tested exclusively in the virtualized

HPC environment due to its root access requirements, which were not available in the real HPC system.

Apptainer, on the other hand, is a rootless container engine designed for scientific computing. It allows users to create and run containers without requiring elevated privileges, making it well-suited for multi-tenant HPC environments. Apptainer was tested in both the virtualized and real HPC environments, enabling a direct comparison of containerized and native execution across different scenarios.

The choice of these tools allowed for an exploration of the trade-offs between root-dependent and rootless containerization in HPC systems. By using Podman and Apptainer alongside native execution, the project aimed to identify the strengths and limitations of each approach.

2.4 Native Execution

Native execution served as the baseline for all benchmarks. By running Gromacs and Sysbench directly on the host system without containerization, the project established a reference point for evaluating the performance impact of container solutions. Metrics collected during native execution were compared against those obtained from containerized runs, highlighting any overhead or efficiency gains introduced by the containers.

2.5 Virtualized HPC Environment Configuration and Setup

This setup was specifically chosen to simulate a typical HPC environment, albeit with limited computational resources, to focus on the comparative performance of containerized and native execution. The virtualized cluster consisted of three nodes: a head node and two worker nodes, each configured to perform specific roles in the cluster.

2.5.1 Head Node Configuration

The head node was configured with 1 CPU core and 4 GB of RAM, functioning as the central management unit of the cluster. Its primary responsibilities included task scheduling, resource allocation, and overall cluster orchestration. To facilitate these tasks, SLURM (Simple Linux Utility for Resource Management) was deployed on the head node as the job scheduling system. SLURM ensured efficient workload distribution across the worker nodes, providing a foundation for running computational benchmarks.

In addition to SLURM, the head node was equipped with Spack, a package management tool optimized for HPC environments. Spack enabled the installation and management of software dependencies required for the benchmarks, ensuring consistency across all nodes. For shared storage, the head node utilized Network File System (NFS), which provided a unified file system accessible by all nodes in the cluster. This setup ensured seamless data sharing and consistent execution environments.

2.5.2 Worker Node Configuration

Two worker nodes, each configured with 1 CPU core and 4 GB of RAM, were deployed to execute the computational workloads. These nodes were designed to simulate the typical resource constraints of HPC environments, providing a realistic testing ground for the benchmark workloads. All computational tasks submitted via SLURM were distributed

to these worker nodes, which executed them in isolation or concurrently, depending on the resource availability and workload requirements.

Despite their limited hardware resources, the worker nodes were capable of running the benchmarks effectively, allowing for meaningful comparisons of containerized and native execution across various scenarios.

2.5.3 Software and Operating System

Rocky Linux 8 was chosen as the operating system for all nodes in the cluster. This enterprise-grade Linux distribution is optimized for reliability, scalability, and security, making it an ideal choice for HPC applications. The uniform installation of Rocky Linux ensured compatibility across all nodes and minimized configuration discrepancies.

Two container solutions, Podman and Apptainer, were installed on the virtualized cluster. Podman was configured with root privileges, as its container operations often require elevated access. This setup allowed for the evaluation of containerized workloads in an environment where administrative privileges are available. Conversely, Apptainer was configured in rootless mode, aligning with its design philosophy of operating with minimal privileges. This distinction enabled a direct comparison between root-dependent and rootless container solutions, providing insights into their relative ease of use, security implications, and performance characteristics.

3 Methodology

This section outlines the approach used to evaluate container solutions against native execution for running benchmarks in a virtualized HPC environment and a real HPC environment. The methodology encompasses the setup of the environments, configuration of the container solutions, execution of benchmarks, and the metrics used for comparison.

3.1 Virtualized HPC Environment Setup

The virtualized HPC environment was provisioned using virtual machines (VMs) running on AVX2-256 CPU micro architecture and configured to simulate a typical HPC cluster. The setup included:

- **Head Node:** One node with 1 CPU core and 4 GB RAM, which also served as the controller for the cluster using SLURM.
- **Worker Nodes:** Two nodes, each with 1 CPU core and 4 GB RAM, configured for computational tasks.
- **Software Stack:** The environment ran Rocky Linux 8 as the operating system, with SLURM for workload scheduling, Spack for software management, and NFS for shared storage.

Podman and Apptainer were installed on the virtualized HPC environment. Podman was configured with root access, while Apptainer utilized its rootless capabilities.

3.2 Real HPC Environment Setup

The real HPC environment consisted of a shared cluster with access to higher computational resources. Here, benchmarks were executed to compare the performance of native execution and Apptainer, as Podman required root access, which was not available in this setup. The focus was on assessing the performance of general-purpose x86 architecture against specific CPU micro architecture(AVX-512) using Gromacs compiled with Spack. In this case a batch script is optimized and submitted to one of the HPC nodes and with 1 CPU core and 4 Gigabytes of RAM.

3.3 Benchmark Execution

Benchmarks were conducted using Gromacs for scientific workloads and Sysbench for system-level performance analysis.

3.3.1 Gromacs Benchmarks

Gromacs benchmarks were executed under multiple scenarios to evaluate computational efficiency:

- **Thread Configurations:** Two configurations were tested—4 threads on a single CPU core and 1 thread per core.
- **Execution Scenarios:** Benchmarks were run natively, in Podman, and in Apptainer in the virtualized environment. In the real HPC environment, only native and Apptainer executions were possible. The performance results for the GROMACS benchmarks were derived from the average of 11 benchmark tests collected from the GROMACS benchmark repository. This approach provides a representative evaluation of typical workloads encountered in molecular dynamics simulations.

A range of metrics was used to evaluate the performance of the container systems, focusing on different aspects of system efficiency and resource utilization.

- **Core Time (s):**

  Core time refers to the total time the CPU spends actively processing tasks. This metric provides insight into how efficiently the CPU is utilized in each environment. Lower core times indicate more efficient processing and resource use.

- **Wall Time (s):**

  Wall time measures the total elapsed time from the start to the completion of a task, including all processing and any delays caused by system overhead. This metric is crucial for understanding the overall speed of task execution in each system, with lower wall times signifying faster task completion.

- **Performance:**

  Performance was evaluated using two specific metrics: nanoseconds per day (ns/day) and hours per nanosecond (hour/ns). The ns/day metric indicates the simulation progress rate, where higher values denote better performance. Conversely, hour/ns measures the time required to simulate a single nanosecond, with lower values indicating superior performance. These metrics are particularly relevant for molecular dynamics simulations such as those performed by GROMACS.

- **Load Balancing and Waiting Time (%):**

  In the 4 Threads/Core configuration, the study also measured load imbalance and waiting time percentages. Load imbalance reflects how evenly the computational load is distributed across the available threads, while waiting time indicates the duration threads spend idle, waiting for resources. Lower values in both metrics are desirable, indicating efficient parallel execution.

3.3.2 Sysbench Benchmarks

Sysbench was used to evaluate I/O and memory performance under varying workloads:

- **I/O and Memory Performance:** Benchmarks were conducted with varying numbers of files (1, 10, 100, 1000) and total file sizes (5 GB, 10 GB, 20 GB, 40 GB). Metrics collected included:
  - **Writes/s and Fsyncs/s:** Measure the frequency of write and synchronization operations.
  - **Written Data (MiB/s):** Indicates the throughput in megabytes per second.
  - **RAM Usage (%):** Tracks memory consumption during the benchmark.

3.4 Comparison Metrics

The evaluation was based on the performance and resource utilization metrics collected from Gromacs and Sysbench benchmarks. The comparison focused on:

- **Performance Efficiency:** Comparing execution times, throughput, and simulation speeds for different execution scenarios.
- **Resource Utilization:** Assessing CPU, memory, and I/O efficiency across native, Podman, and Apptainer environments.
- **Scalability:** Analyzing performance trends with varying workload sizes, thread configurations, and file operations.

This methodology provided a structured approach to evaluate the capabilities and limitations of Podman, Apptainer, and native execution in both virtualized and real HPC environments.

4 Results

This section presents the performance evaluation of various configurations: Native (AVX2-256), Podman, Apptainer, Native HPC (AVX-512), and Apptainer HPC (AVX-512). The metrics evaluated include Core Time, Wall Time, and Performance, with data for both 1 Thread/Core and 4 Threads/Core configurations. Corresponding figures aid in the visualization of these results.

4.1 Core Time (s)

Core time measures the CPU's active processing time. Figure 1 shows the core time for each configuration.

1. **1 Thread/Core**:
   (a) Native (AVX2-256) and Podman have similar core times, with Podman being slightly faster.
   (b) Apptainer takes longer.
   (c) Native HPC (AVX-512) and Apptainer HPC (AVX-512) show significantly higher core times.
2. **4 Threads/Core**:
   (a) Podman and Apptainer have comparable core times.
   (b) Native (AVX2-256) is slightly better than both.
   (c) Native HPC (AVX-512) and Apptainer HPC (AVX-512) have the longest core times.

Lower core time indicates better CPU efficiency.

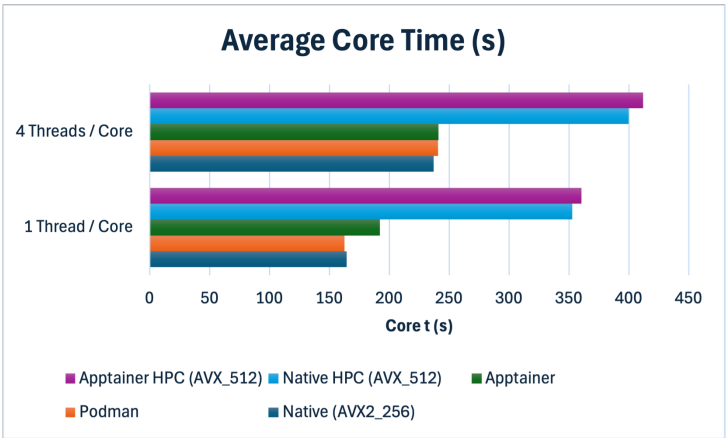

Figure 1: Core time comparison for 1 Thread/Core and 4 Threads/Core configurations.

4.2 Wall Time (s)

Wall time, representing the total time from task initiation to completion, is presented in Figure 2.

1. **1 Thread/Core**:
   (a) Podman slightly outperforms Native (AVX2-256).
   (b) Apptainer takes longer than both Podman and Native (AVX2-256).
   (c) Native HPC (AVX-512) and Apptainer HPC (AVX-512) show much higher wall times.

2. **4 Threads/Core**:
   (a) Podman leads among standard configurations.
   (b) Native HPC (AVX-512) and Apptainer HPC (AVX-512) have the highest wall times.

Lower wall time implies faster task completion.

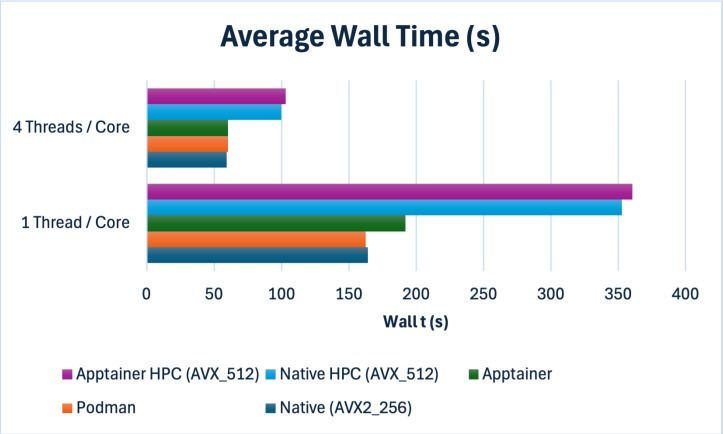

Figure 2: Wall time comparison for 1 Thread/Core and 4 Threads/Core configurations.

4.3 Performance (ns/day and hour/ns)

Performance is assessed using ns/day and hour/ns metrics, shown in Figures 3 and 4.

1. **1 Thread/Core**:
   (a) Podman slightly surpasses Native (AVX2-256) in ns/day.
   (b) Apptainer lags behind Podman and Native (AVX2-256).
   (c) Native HPC (AVX-512) and Apptainer HPC (AVX-512) perform significantly worse.
2. **4 Threads/Core**:
   (a) Native (AVX2-256) leads in ns/day among standard systems.
   (b) Podman and Apptainer trail behind Native (AVX2-256).
   (c) The HPC-optimized configurations perform similarly.

Higher ns/day and lower hour/ns are desirable for better performance.

4.4 4 Thread Load Balancing and Waiting Time (%)

Load imbalance and waiting time are vital metrics for efficient resource utilization. The load imbalance and waiting times across three systems, Native, Apptainer, and Podman, are compared in Table 4.4

In terms of load imbalance, Native exhibits the highest, followed by Apptainer, and then Podman. Similarly, considering waiting time, Native has the longest duration followed by Apptainer and then Podman showing lower values. Lower values for both metrics are more desirable for optimal performance.

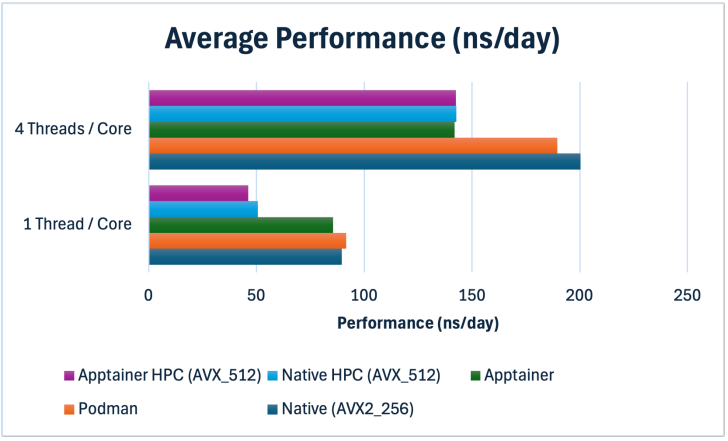

Figure 3: Performance comparison in ns/day for 1 Thread/Core and 4 Threads/Core configurations.

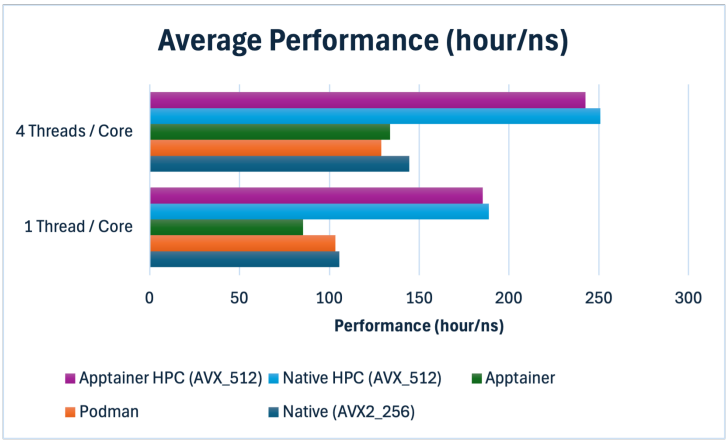

Figure 4: Performance comparison in hour/ns for 1 Thread/Core and 4 Threads/Core configurations.

4.5 I/O Performance

The I/O performance, critical for data-intensive workloads, is evaluated in terms of Writes/s, Fsyncs/s, Written (MiB/s), and memory usage percentage. Figures 5, 6, 7, and 8 display the results for different file sizes and numbers of files.

4.5.1 Writes/s

Podman dominates in writes per second across all file sizes and numbers of files. Native and Apptainer show similar performance with negligible differences.

4.5.2 Fsyncs/s

Podman demonstrates exponential growth in Fsyncs/s with an increasing number of files, outperforming Native and Apptainer.

	Native	Podman	Apptainer
Load Imbalance (%)	11.7	10.29	11.18
Waiting Time (%)	2.92	2.55	2.64

Table 1: Load imbalance and waiting times for different systems

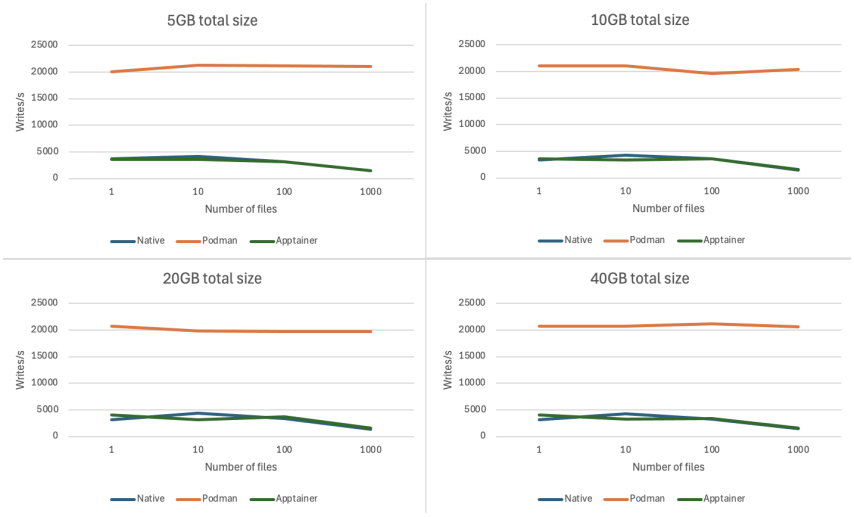

Figure 5: Writes per second comparison across different file sizes and numbers of files.

4.5.3 Written (MiB/s)

Podman consistently achieves the highest write speeds in MiB/s across all file sizes and numbers of files, with Native and Apptainer showing close performance.

4.5.4 Memory Usage (%)

Podman utilizes the most memory, followed by Apptainer, with Native maintaining the lowest memory usage across all file sizes.

5 Discussion

The performance evaluation across different configurations—Native (AVX2-256), Podman, Apptainer, Native HPC (AVX-512), and Apptainer HPC (AVX-512)—provides a comprehensive view of how containerization impacts high-performance computing (HPC) workloads. This section delves deeper into the implications of the observed performance metrics, exploring core time, wall time, throughput, load balancing, I/O performance, and memory usage.

5.1 Core Time and Wall Time

The core time, which measures the CPU's active processing period, indicated that Podman slightly outperforms Native (AVX2-256) for 1 Thread/Core, possibly due to optimized resource management within its containerized environment. However, Apptainer's longer core times suggest a trade-off in its rootless design, which adds layers of management for security and user isolation. The significantly higher core times for Native

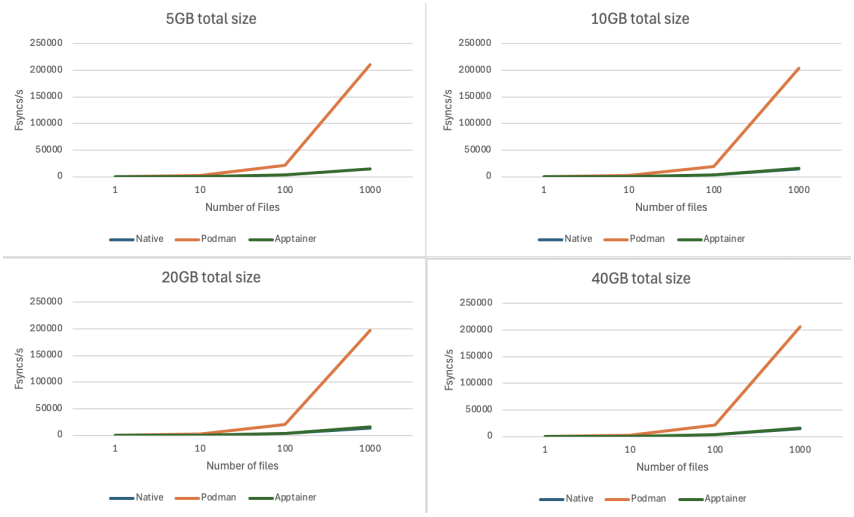

Figure 6: Fsyncs per second comparison across different file sizes and numbers of files.

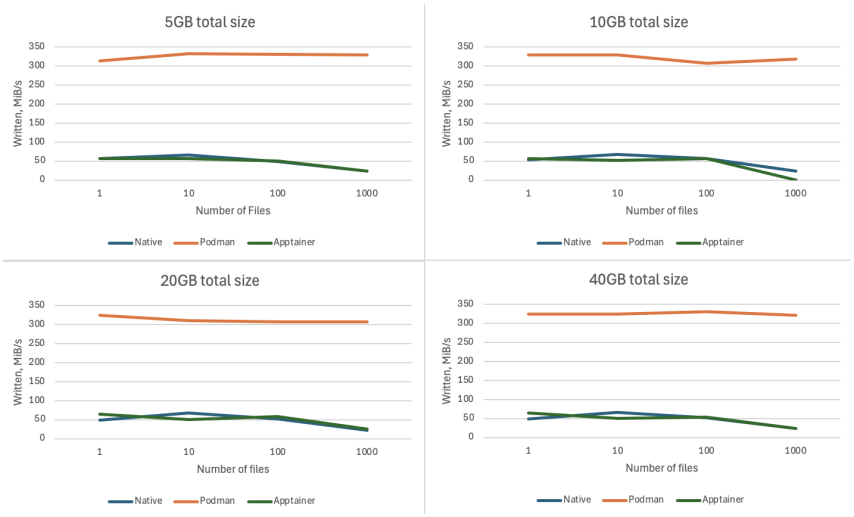

Figure 7: Write speeds in MiB/s comparison across different file sizes and numbers of files.

HPC (AVX-512) and Apptainer HPC (AVX-512) underscore the intensive computational demands of AVX-512 instructions, which, while offering higher throughput for suitable workloads, also require more processing time.

Wall time results showed a similar pattern, with Podman achieving slightly lower times compared to Native (AVX2-256), highlighting its efficiency in handling task execution within a container. Apptainer's increased wall times suggest an overhead likely attributed to its rootless operation and runtime management. The elevated wall times for the HPC configurations reflect the complexity and time-consuming nature of using advanced vector instructions like AVX-512, indicating these configurations are better suited for workloads specifically optimized for such instruction sets.

5.2 Performance Metrics

The performance metrics of ns/day and hour/ns provide insights into the computational throughput of each configuration. Podman's slight edge over Native (AVX2-256) in ns/day

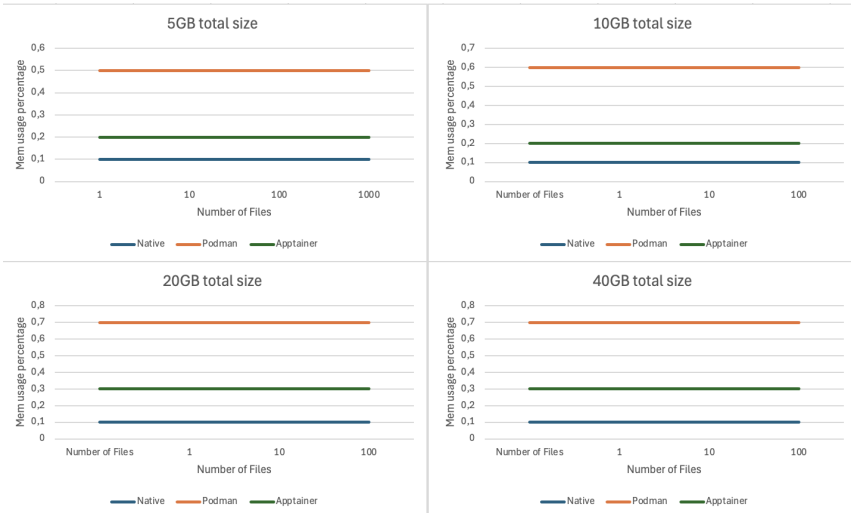

Figure 8: Memory usage percentage comparison across different file sizes.

for 1 Thread/Core can be attributed to its ability to efficiently handle the task distribution and execution within its container framework. For 4 Threads/Core, Native (AVX2-256) reclaimed the lead, demonstrating its raw computational power unencumbered by container overhead. Apptainer's performance lag, particularly in higher thread counts, emphasizes the need for optimization in its container runtime for multi-threaded scenarios.

The HPC-optimized configurations showed a marked decline in ns/day, indicating that while AVX-512 provides a theoretical advantage for certain types of operations, the practical execution within these environments introduces complexities that diminish overall throughput. This suggests that while these configurations can excel in highly specialized tasks, general HPC workloads may not benefit as much without further optimization.

5.3 Load Balancing and Waiting Time

The load balancing and waiting time metrics, critical for resource utilization and job scheduling efficiency, revealed that Podman had the lowest load imbalance and waiting times. This efficiency may stem from its container orchestration capabilities, which manage tasks and resources more dynamically. Apptainer, with slightly higher values than Podman, still performed well, indicating effective handling of resource allocation despite the rootless design's slight overhead. Native's higher load imbalance and waiting time percentages suggest less dynamic resource management, potentially leading to less efficient use of HPC cluster resources in queued job scenarios.

5.4 I/O Performance and Memory Usage

Podman's dominance in I/O performance metrics: Writes/s, Fsyncs/s, and Written MiB/s suggests that its containerized environment is highly optimized for handling file operations, which is crucial for data-intensive HPC applications. The exponential growth in Fsyncs/s with an increasing number of files indicates Podman's superior scaling capabilities in file synchronization tasks.

In contrast, Native and Apptainer showed similar, though slightly lower, performance, likely due to less aggressive I/O optimizations. However, Native's lower memory usage across various file sizes points to its efficient memory management, a vital consideration

in environments with constrained resources. Apptainer's slightly higher memory usage compared to Native, yet lower than Podman, underscores its balanced approach between performance and resource utilization.

5.5 Impact of Containerization on Performance

The comparative analysis of Podman and Apptainer highlights the impact of containerization strategies on performance. Podman's slight edge in core time and wall time for 1 Thread/Core configurations, coupled with its I/O performance supremacy, suggests that its architecture effectively leverages the advantages of containerization without significantly compromising on speed. However, its higher memory usage indicates a trade-off in resource efficiency.

Apptainer's rootless design, while slightly less performant in core and wall times, offers enhanced security benefits, making it a strong candidate for multi-user HPC environments where isolation and security are critical. The slight performance penalties observed can be mitigated with further optimization in its container runtime management.

5.6 Limitations and Future Work

The study's limitations, including the virtualized environment and limited scope of configurations and benchmarking tools, highlight the need for broader research. Future work should encompass more diverse container engines, such as Docker and Singularity, and a wider array of benchmarking tools to capture a more comprehensive performance landscape. Additionally, real-world testing on larger, heterogeneous HPC clusters would provide deeper insights into the scalability and practical applicability of these container solutions in diverse HPC scenarios.

Moreover, exploring additional metrics such as energy efficiency and fault tolerance would offer a more rounded evaluation of containerized environments in HPC. This would not only aid in selecting the most suitable container technology but also in optimizing HPC resource management for enhanced performance and sustainability.

6 Conclusion

This study evaluated the performance of containerized and native HPC environments, focusing on configurations such as Native (AVX2-256), Podman, Apptainer, Native HPC (AVX-512), and Apptainer HPC (AVX-512). The analysis covered core time, wall time, throughput, load balancing, I/O performance, and memory usage across different thread configurations.

The findings indicate that Podman generally achieves lower core and wall times, with superior I/O performance, particularly in data-intensive tasks, making it a robust choice for environments requiring high throughput and frequent file operations. Apptainer, while slightly slower, offers enhanced security through its rootless design and demonstrates strong performance in scenarios demanding user isolation.

Native configurations, especially those optimized for AVX-512, excel in computational tasks that can leverage advanced vector instructions, although they exhibit higher core and wall times. The trade-offs between performance and security are evident, with con-

tainerized solutions providing a balanced approach between speed, resource utilization, and security.

Future research should extend these findings by exploring additional container technologies and more diverse benchmarking scenarios to further understand the trade-offs in HPC environments.

References

[App23] Appsilon. "Docker vs. Podman vs. Singularity: Which Containerization Platform Should You Choose?" In: (2023). URL: https://www.appsilon.com/post/docker-vs-podman-vs-singularity.

[Fel+15] Wes Felter et al. "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments". In: *IEEE International Symposium on High Performance Distributed Computing* (2015). URL: https://ieeexplore.ieee.org/document/6498558/.

[GRO24] GROMACS. *GROMACS Benchmarks*. 2024. URL: https://ftp.gromacs.org/pub/benchmarks/.

[Mül+22] Tiziano Müller et al. "Containerisation for High Performance Computing Systems: Survey and Prospects". In: *IEEE Transactions on Software Engineering* (2022). URL: https://dl.acm.org/doi/10.1109/TSE.2022.3229221.

[Pet23] Dirk Petersen. "dirkpetersen/hpc-containers: You should offer both Podman and Apptainer (Singularity) and allow users to create containers in rootless mode with user namespaces on your HPC systems". In: (2023). URL: https://github.com/dirkpetersen/hpc-containers.

A Code samples

To create benchmark.tpr file from the downloaded directory:

```
gmx_mpi grompp -f pme_verlet.mdp -c conf.gro -p topol.top -o benchmark.tpr
```

To create .sif for Apptainer:

```
apptainer build gromacs.sif docker://gromacs/gromacs
```

Gromacs Benchmark BATCH script:

```

#!/bin/bash

#SBATCH --job-name=gromacs_benchmark
#SBATCH --output=gromacs_benchmark_%j.log
#SBATCH --error=gromacs_benchmark_%j.err
#SBATCH --time=02:00:00          # Max wall time
#SBATCH --mem=4GB                # Memory per node
#SBATCH --cpus-per-task=1        # Number of CPUs per task
#SBATCH --ntasks=1               # Number of tasks
#SBATCH --nodes=1                # Number of nodes

# Load required modules
spack load gromacs              #  Load module or spack
spack load apptainer
spack load podman

# Path to the Apptainer image
APPTAINER_IMAGE="/NFS/benchmark/gromacs.sif"

# Array of benchmark directories
BENCHMARK_DIRS=(
    "/NFS/benchmark/water-cut1.0_GMX50_bare/3072"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/1536"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/0768"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/0003"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/0001.5"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/0000.96"
    "/NFS/benchmark/water-cut1.0_GMX50_bare/0000.65"
    "/NFS/benchmark/ADH/adh_dodec"
    "/NFS/benchmark/ADH/adh_cubic"
    "/NFS/benchmark/rnase_cubic"
    "/NFS/benchmark/rnase_dodec"
    "/NFS/benchmark/rnase_dodec_vsites"
)

# Check if the Apptainer image exists
if [[ ! -f "$APPTAINER_IMAGE" ]]; then
    echo "Error: Apptainer image '$APPTAINER_IMAGE' not found!"
    exit 1
fi

# Loop through each benchmark directory
for BENCHMARK_DIR in "${BENCHMARK_DIRS[@]}"; do
    if [[ ! -d "$BENCHMARK_DIR" ]]; then
        echo "Warning: Directory '$BENCHMARK_DIR' not found. Skipping..."
        continue
    fi

    echo "Processing benchmark in directory: $BENCHMARK_DIR"
    cd "$BENCHMARK_DIR"

    # Native execution
    echo "Running native GROMACS..."
    mpirun -np $SLURM_CPUS_PER_TASK gmx_mpi mdrun -s benchmark.tpr

    # Podman execution
    echo "Running GROMACS with Podman..."
    podman run --rm -v "$PWD:/data" \
```

```
        -e OMPI_ALLOW_RUN_AS_ROOT=1 \
        -e OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1 \
        -e OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK \
        docker.io/gromacs/gromacs \
        mpirun --allow-run-as-root -np $SLURM_CPUS_PER_TASK gmx mdrun -s /data/benchmark.tpr

    # Apptainer execution
    echo "Running GROMACS with Apptainer..."
    apptainer exec --bind "$PWD:/data" \
        --env OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK \
        "$APPTAINER_IMAGE" mpirun -np $SLURM_CPUS_PER_TASK gmx mdrun -s /data/benchmark.tpr

    echo "Completed processing for: $BENCHMARK_DIR"
    echo "----------------------------------------"
done

echo "All benchmarks processed."

}
```

Sysbench I/O Benchmark BATCH script:

```
#!/bin/bash

#SBATCH --job-name=gromacs_benchmark
#SBATCH --output=gromacs_benchmark_%j.log
#SBATCH --error=gromacs_benchmark_%j.err
#SBATCH --time=02:00:00          # Max wall time
#SBATCH --mem=4GB                # Memory per node
#SBATCH --cpus-per-task=1        # Number of CPUs per task
#SBATCH --ntasks=1               # Number of tasks
#SBATCH --nodes=1                # Number of nodes


# Parameters
file_nums=(1 10 100 1000)
total_sizes=(5G 10G 20G 40G)
results_dir="/NFS/benchmark/benchmark_results"
sif_file="/NFS/benchmark/alpine_sysbench.sif"

# Ensure sysbench, podman, and apptainer are installed
sudo dnf install -y sysbench podman apptainer

# Create results directory
mkdir -p $results_dir

# Function to run benchmark natively
run_native() {
    echo "Running native benchmarks..."
    for file_num in "${file_nums[@]}"; do
        for total_size in "${total_sizes[@]}"; do
            echo "== Native: File Num: $file_num, Total Size: $total_size =="
            sysbench fileio --file-test-mode=seqwr --file-total-size=$total_size --file-num=$file_num \
            --threads=1 run >> ${results_dir}/results_${file_num}_files_${total_size}_size.txt
        done
    done
}

# Function to run benchmark in Podman container
run_podman() {
    echo "Running Podman benchmarks..."
    podman pull docker.io/library/alpine:latest
    for file_num in "${file_nums[@]}"; do
        for total_size in "${total_sizes[@]}"; do
            echo "== Podman: File Num: $file_num, Total Size: $total_size =="
            podman run --rm -v ${results_dir}:/results docker.io/library/alpine:latest /bin/sh -c "
                apk add --no-cache sysbench &&
                sysbench fileio --file-test-mode=seqwr --file-total-size=$total_size --file-num=$file_num \
                --threads=1 run" >> ${results_dir}/results_${file_num}_files_${total_size}_size.txt
        done
    done
}

```

```
# Function to run benchmark in Apptainer container
run_apptainer() {
    echo "Running Apptainer benchmarks..."
    for file_num in "${file_nums[@]}"; do
        for total_size in "${total_sizes[@]}"; do
            echo "== Apptainer: File Num: $file_num, Total Size: $total_size =="
            apptainer exec -B {results_dir}:/results $sif_file /bin/sh -c "
                sysbench fileio --file-test-mode=seqwr --file-total-size=$total_size --file-num=$file_num \
                --threads=1 run" >> ${results_dir}/results_${file_num}_files_${total_size}_size.txt
        done
    done
}

# Run benchmarks
run_native
run_podman
run_apptainer

echo "Benchmarking completed! Consolidated results are saved in the ${results_dir} directory."

```