Portability and scalability evaluation of large-scale statistical modeling and prediction software through HPC-ready containers

Sameh Abdulah [a],[*], Jorge Ejarque [b], Omar Marzouk [c], Hatem Ltaief [a], Ying Sun [a], Marc G. Genton [a], Rosa M. Badia [b], David E. Keyes [a]

[a] *Extreme Computing Research Center (ECRC), King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia*
[b] *Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[c] *Brightskies Inc., Alexandria Governorate, Egypt*

A B S T R A C T

HPC-based applications often have complex workflows with many software dependencies that hinder their portability on contemporary HPC architectures. In addition, these applications often require extraordinary efforts to deploy and execute at performance potential on new HPC systems, while the users expert in these applications generally have less expertise in HPC and related technologies. This paper provides a dynamic solution that facilitates containerization for transferring HPC software onto diverse parallel systems. The study relies on the HPC Workflow as a Service (HPCWaaS) paradigm proposed by the EuroHPC eFlows4HPC project. It offers to deploy workflows through containers tailored for any of a number of specific HPC systems. Traditional container image creation tools rely on OS system packages compiled for generic architecture families (x86_64, amd64, ppc64, ...) and specific MPI or GPU runtime library versions. The containerization solution proposed in this paper leverages HPC Builders such as Spack or Easybuild and multi-platform builders such as buildx to create a service for automating the creation of container images for the software specific to each hardware architecture, aiming to sustain the overall performance of the software. We assess the efficiency of our proposed solution for porting the geostatistics ExaGeoStat software on various parallel systems while preserving the computational performance. The results show that the performance of the generated images is comparable with the native execution of the software on the same architectures. On the distributed-memory system, the containerized version can scale up to 256 nodes without impacting performance.

1. Introduction

Scientific computing has experienced a drastic shift from relying on single-core machines to massively parallel distributed-shared memory systems. High-Performance Computing (HPC) tools have added tremendous value to many applications, allowing them to handle larger workloads by providing more computing power and memory capacity. However, HPC systems have become increasingly more complicated and heterogeneous due to the large variety of hardware architectures and accelerators they incorporate. The software components of these systems, including operating systems, programming languages, compilers, libraries, and tools, also exhibit wide variation; this heterogeneity also applies to the security constraints and usage restrictions implemented by the system administrators [1]. This variety creates challenges for deploying existing HPC software, including poor portability and low reusability. An extra virtualization layer is a natural way to tackle these challenges and allow easy and efficient portability

of HPC-oriented software. The primary objective of this layer is to isolate the software stack from the underlying hardware platform and help domain scientists run HPC-oriented scientific applications without fully understanding how to optimize these applications on their HPC hardware. When the isolation happens at the operating system level, and the application can run in an isolated user space regardless of the underlying hardware, this is called "containerization".

Container systems operate by utilizing a shared operating system kernel across multiple containers. Each has a separate and isolated user-space environment, allowing many instances to run on the same host without interfering with each other. This technology appeared first to manage user access privileges inside the Unix operating system and gained popularity with the release of Docker [2] and its interaction with the cloud computing community in 2014. Containers have become popular in the cloud computing community due to their impact on improving the portability of the code and the small overhead they add

to the execution compared to other virtual machine environments such as KVM [3], VMware Fusion [4], Hyper-V [5], and Xen [6]. Recently, containers have gained more attention in the HPC community, where several studies provide performance assessments on the benefits of containers in improving the scalability and portability of the HPC software. The studies argue that containers can be a "golden solution" for avoiding the complexity of the HPC software setup and can guarantee portability on contemporary and future HPC systems. However, the performance achieved by containers relies on how the container images have been created. To achieve the best performance on an HPC system, applications must be installed taking into account the features of the system, such as the computer architecture, the available networking fabric, and/or the available accelerators.

Traditionally, container images are created using OS packages that are binary installations pre-compiled with generic optimization flags, and communication libraries are compiled for generic networking fabric available in most computers. These installations are usable in all the processors of a family. Still, they cannot use the features of the more advanced processors, such as vector instructions and other compile-time optimizations. So, container images created with this methodology are more portable but cannot achieve the best performance available in the system.

Trying to close the gap between deployment simplicity and performance, the eFlows4HPC project[1] proposes a methodology to leverage containers to simplify the installation of complex workflows in HPC infrastructures while maintaining the performance at the same level as native installations. This paper proposes a service to automate the creation of container images tailored to specific HPC systems. From a generic recipe and a target system description, the service automates the creation of an image for this system, leveraging HPC package builders such as Spack [7] and multi-platform container builders such as buildx, which can customize the building process for the HPC system features. The suggested approach enables the one-time preparation of installation scripts for all software dependencies using Spack, followed by the creation of container images for all necessary architectures. This approach simplifies the deployment of HPC software to the user by overcoming all the variability of HPC systems mentioned above into just two steps: initiating a container image creation service call and transferring the resulting container image file to the HPC system.

We assess the performance of the proposed solution by porting the geospatial statistics application *ExaGeoStat* [8]. *ExaGeoStat* is a parallel software library for modeling spatial and spatio-temporal data through the Maximum Likelihood Estimation (MLE) method. The software can also predict missing geospatial data in space or space–time with the help of observed data and a set of obtained statistical parameters from the modeling process. In [9], the authors propose *ExaGeoStatR*, an *R* package that wraps the *ExaGeoStat* software under the *R* environment. Statisticians can use *ExaGeoStat* and *ExaGeoStatR* to tackle many operations on large geospatial data volumes if the complexity of installation does not limit its adoption.

In this work, we present the experience of using containers with the *ExaGeoStat* software by building HPC-ready containers for many architectures from a generic builder recipe. The assessment relies on pre-prepared Spack scripts to generate HPC-ready Singularity containers. The performance obtained in the experimentation demonstrates that containers can preserve the accuracy of shared-memory systems from various vendors and distributed memory systems under different conditions. The assessment includes two computation variants from *ExaGeoStat*, including dense and Tile Low-Rank (TLR) approximation. The experiments also implicitly assess the performance of two commonly used parallel linear algebra libraries, i.e., Chameleon [10], HiCMA [11], and the popular runtime system StarPU [12].

---

[1] https://eflows4hpc.eu.

Our contributions can be summarized as follows:

- We present an automated approach for generating HPC-ready containers for HPC software, ensuring portability across various hardware architectures while maintaining performance.
- We rely on the combined usage of Spack, Singularity, and Docker buildx in our container creation process. We employ a straightforward bash script to streamline the process of retrieving the generated containers on a specific hardware system.
- We comprehensively explain the pipeline involved in creating HPC-ready containers that can be easily applied to existing HPC software with minimal modifications.
- We employ the *ExaGeoStat* software and its dependencies as an example to evaluate and analyze the effectiveness of the created containers. While our focus is primarily on the *ExaGeoStat* software, it is important to note that our HPC-ready containers can be applied to a wide range of existing HPC software, provided they are compatible with the targeted hardware.
- We use multiple shared-memory systems and a distributed-shared-memory system to demonstrate the performance of our generated containers compared to the bare-metal installation of *ExaGeoStat*. This enables us to assess the efficiency and effectiveness of the containers in various computing environments.

The paper is organized as follows: Sections 2 and 3 present related work and background technologies, Section 4 describes the eFlows4HPC project, and Section 5 gives an overview of the *ExaGeoStat* software and its components. Section 6 presents the methodology for generating HPC-Ready containers. Section 7 describes extensive experimentation performed to evaluate the multiple HPC-ready containers, and Section 9 summarizes the work.

2. Related work

Many studies aim to evaluate the efficiency of VMs and containers with HPC applications and systems. For instance, in [13], an evaluation study on the Xen virtualization system for the HPC systems has been shown. The Xen virtualization system is a technique that allows running separate operating systems on the same hardware [6]. Xen is popular for its high performance, scalability, and reliability. The authors prove that Xen virtualization can be a practical solution for the HPC software portability with 2%–3% overhead compared to executing the original kernels. Another study example comparing three VM solutions has been proposed in [14]. Among VMWare Server, Xen, and OpenVZ, OpenVZ provides the most satisfactory performance for HPC applications—other studies in the virtualization solution are [15–18].

Containers were proposed through the release of Docker [2] in 2013. Containerization has gained more attention in software portability than VMs since it depends on virtualizing the software layer, not the entire machine, where different instances share the same host OS kernel. It allows the ability to isolate the software layer with less overhead than VMs, which makes this technology interesting for the HPC community to port complex applications. Many studies have evaluated the efficiency of containers in porting HPC software. For instance, the authors of [19] provide a scalability and portability study for a production biological simulation application that relies on containers. The study embraces three container systems, Docker, Singularity [20], and Shifter [21], and claims that Singularity can provide the scalability and portability performance required by the target application. Another study [22] includes performance evaluation of containers under heavy HPC workloads. The study also assesses the efficiency of container-based HPC software with different Linux kernels on HPC systems. Other studies evaluating containers in HPC contexts are [23–26].

Although containers have been used to port HPC applications for several years, all the efforts were made individually and for a specific application or software. The HPC community can benefit from a standard workflow manager that can help simplify the porting process

for different HPC applications. In [27], the HPC Workflow as a Services (HPCWaaS) paradigm has been proposed under the eFlows4HPC project to enable encapsulating the HPC software workflow more dynamically and intelligently. The new paradigm adopted the concept of "HPC-ready containers" to build efficient and more reliable containers for contemporary HPC applications. Despite containers having a low execution overhead in executing isolated processes, the HPC applications are also required to benefit from the particular hardware (CPU features, accelerators, and network fabric). Therefore, the container image build processes have to ensure that the software is correctly configured, such as compilation for specific CPU architecture and with compatible versions of the MPI and GPU runtimes (CUDA, ROCm). Previous work on this topic has been tackled in [28,29]. In the first study, they propose to create a base image per HPC site, which already includes a proper installation of MPI and GPU runtime, and create images on top of them. This approach has two drawbacks: it requires the maintenance of the base images by a system administrator, and the compilation of container images must be performed in the same platform as the target HPC site, taking care of the compilation with the target CPU architecture. In the second case, they propose a tool to create container images with a specific MPI and CUDA configuration using their Python-based software description. This approach does not require maintaining different base images, but as in the previous work, it does not deal with multiple platforms and CPUs. In our approach, we support different architectures due to the usage of multi-platform builders and the management optimal CPU architecture compilations and MPI and GPU runtime version configuration is provided by HPC package managers such as Spack.

In this paper, we assess the HPC-ready containers provided by the HPCWaaS paradigm in the eFlows4HPC project using the *ExaGeoStat* software. The selected software mainly relies on dense linear algebra operations through exact and low-rank approximation computations to a covariance matrix that can reach millions by millions in size [30]. *ExaGeoStat* can be executed in all leading-edge parallel hardware architectures, including accelerators, which motivates us to use it as a test case in our study. More details about *ExaGeoStat* can be found on [8,31].

3. Background

The following subsections provide an overview of the Containerization concept, Spack package manager, and Singularity. Spack and Singularity are the two main components of the HPC-ready containers in the eFlows4HPC project.

3.1. Containerization

Performance portability of HPC software on different HPC systems is difficult because of the complicated structure of its dependencies. The search for a robust solution allowing fast and secure porting on HPC systems led to containerization, which aims to package the software code and the required operating system libraries and dependencies into a single executable file that can run on different HPC systems while preserving the expected performance on these systems. Furthermore, containers are more robust than Virtual Machines (VMs) solutions since containers isolate the software layer from the hardware layer through the underlying operating system namespaces, which places the software much closer to the physical system, thus improving performance [19]. Docker [2] is the most used engine in personal computers and cloud environments. However, it is not highly extended in HPC. Some features and requirements of Docker do not fit traditional HPC system security constraints and resource managers. To fill this gap, other container engines were created that can easily be executed from the non-privileged user space and easily integrated with HPC schedulers and resource managers. Singularity [20], Shifter [21], and Saurus [32] are some examples of these types of container engines.

3.2. Spack package manager

The Spack package manager [7] provides a novel, recursive specification syntax to invoke parametric builds of packages and dependencies. It allows any number of builds to coexist on the same system, and it ensures that installed packages can find their dependencies and libraries, regardless of the environment. Spack provides a simple spec syntax, allowing users to configure versions and options easily and precisely. It simplifies the packaging job for authors as the package files are written in Python. Also, the specs allow them to maintain a single file for different package builds.

Spack provides Spack Environments for the creation of container images. An environment is used to group together a set of specs for the purpose of building, rebuilding, and deploying coherently. Environments separate the steps of (a) choosing what to install, (b) concretizing, and (c) installing. This allows Environments to remain stable and repeatable. Also, environments allow several specs to be built at once. In addition, an Environment that is built as a whole can be loaded as a whole into the user environment. The environment to be created is defined in a manifest file in YAML format (spack.yaml).

3.3. Singularity container system

The appearance of container solutions such as Docker[2] offered improvements over standard virtual machines. However, for the scientific world, and specifically HPC communities, the Docker technology does not fit cleanly. Installing Docker on HPC environments would mean a level of security risk deemed unreasonable, preventing it from being embraced by a large community. This security risk comes from Docker's container processes spawned as children of a root-owned Docker daemon.

Singularity [20] is a container solution created by the necessity for scientific application-driven workloads. Singularity containers can be built on the user's laptop and run on many large HPC clusters and in single servers or the cloud. Singularity offers mobility by enabling environments to be ported via a single image file. It is designed with the necessary features to allow seamless integration with scientific computational resources. Singularity is a pioneer in providing an accessible environment for users and administrators since it was developed in collaboration with HPC administrators, developers, and research scientists.

The goal of Singularity is to support existing and traditional HPC resources as easily as installing a single package onto the host operating system. Singularity can run on a large number of Linux distributions, including vintage ones. Singularity natively supports technologies such as InfiniBand and Lustre while integrating seamlessly with any resource manager (e.g., SLURM, Torque, SGE, etc.) as a result of the fact that Singularity is run like any other command on the system. Singularity also includes a SLURM plugin that allows SLURM jobs to be natively run within a Singularity container. Singularity also enables access to unique host resources such as GPUs. Regarding security issues, Singularity does not provide the ability to escalate permission inside a container. With Singularity, if a user does not have root access on the target system, the user cannot escalate privileges within the container to root either.

4. The eFlows4HPC project

The EuroHPC Joint Undertaking (EuroHPC JU) aims to develop a world-class European supercomputing ecosystem. It is procuring and deploying five petascale and three pre-exascale systems in Europe and has announced the installation of the first European Exascale system in 2023. These systems will be capable of running large and complex applications, which in many cases will not be exclusively traditional

---
[2] Docker website, https://www.docker.com.

simulation workloads but include aspects of artificial intelligence and data analytics. The EuroHPC JU, in addition, is funding projects that co-design new systems and the necessary software stack and applications that will leverage future exascale systems.

The eFlows4HPC [27] is a EuroHPC project focusing on developing workflows. eFlows4HPC aims to provide a software stack that simplifies the development of workflows that combine HPC simulation and modeling with artificial intelligence and data analytics. The project also seeks to enable the development of workflows that can react to external or internal events (i.e., changes in input data or code exceptions) and dynamically change the behavior of the workflow. In addition, the eFlows4HPC software stack provides runtimes that execute the workflows efficiently in terms of time and energy. The project also develops the concept of the HPC Workflows as a Service (HPCWaaS) concept, which aims to provide a mechanism to make the use and reuse of the workflows in HPC infrastructures easier. With this goal, the project has designed the HPCWaaS interface, which supports all the phases of the workflow lifetime: workflow development, deployment, credential management, and execution.

This paper leverages the deployment methodologies developed in the eFlows4HPC project, which are based on a service that can generate HPC-ready containers. The container greatly simplifies the software deployment phase in any computing infrastructure. However, if a generic container image is used in an HPC system, this image will probably not benefit from the specific hardware features and specialized libraries of the system. Taking this into account, the project has designed a methodology that enables the automatic generation of containers that leverage the hardware and software features of the specific system.

The methodology used in this article is based on combining multi-platform container builders with HPC package managers to provide the desired functionality. To build the specific HPC-ready container image, the application developer generically provides the software requirements using the HPC package manager format, i.e., without including specific hardware requirements. The container generation service uses these software requirements and the information about the target system we would like to generate the container image for. This information is used to configure the specific container image build process for the target HPC system. Once the image for a particular system has been created, it is stored in an image repository for reuse. In this implementation, we have used to enable the creation of Docker buildx for building the images for multiple platforms, Spack as HPC Package manager.

5. ExaGeoStat: The use case software

Spatial statistics methods aim to model spatial and spatio-temporal data to obtain a set of statistical parameters that can be later used to predict missing values in space or space–time. Geospatial data can be found in many disciplines, including climatology, topography, and geology. Indeed, traditional geostatistical tools are readily overwhelmed with the huge amount of geospatial data coming from sources such as satellites, and novel solutions are needed to handle these large data volumes. *ExaGeoStat* [8] is high-performance software for computational geostatistics on many-core systems for tackling large data volumes through state-of-the-art HPC-based solutions that has run on numerous Top100 systems of different architectures and was selected as a 2022 Gordon Bell finalist [30].

5.1. ExaGeoStat software operations

ExaGeoStat offers three primary geostatistical operations: synthetic geospatial data generation, geospatial data modeling using Maximum Likelihood Estimation (MLE), and geospatial data prediction through kriging. Fig. 1 explains the operations involved in each component.

5.1.1. Synthetic geospatial data generation

ExaGeoStat provides an internal data generator of geospatial data for experiments under prescribed conditions such as distribution shape and correlation lengths in space and time. For illustrations herein, which focus on performance portability and not statistical features, we consider only the 2D spatial Gaussian distribution. Assuming n spatial locations uniformly distributed in the unit square, the covariance matrix $\Sigma(\theta)$ can be built using a wide range of covariance functions, including the Matérn covariance function [8]. The first task is to compute the inverse of $\Sigma(\theta)$ using Cholesky factorization: $\Sigma(\theta) = \mathbf{L} \cdot \mathbf{L}^\top$. The second task is to compute the measurement vector $\mathbf{Z}$ as $\mathbf{Z} = \mathbf{L} \cdot \mathbf{e}$, where $\mathbf{e}$ is another random standard Gaussian distributed vector. The geospatial data are represented by the set of locations and corresponding measurements in each location, $\mathbf{Z}$. The entire operation is shown by Fig. 1(a).

5.1.2. Geospatial data modeling

The Maximum Likelihood Estimation (MLE) method is used for geospatial data modeling in *ExaGeoStat*. The geospatial data are usually modeled in spatial statistics as a realization from a Gaussian spatial random field. Assuming $\mathbf{Z}$ vector be a realization of a Gaussian random field $\mathbf{Z}(\mathbf{s})$ where $\mathbf{s}$ is the set of spatial locations, $\Sigma(\theta)$ is a covariance matrix with dimension n, and the random field $\mathbf{Z}$ has a mean zero, the statistical parameter vector θ can be obtained based on the Gaussian log-likelihood function:

$$\ell(\theta) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma(\theta)| - \frac{1}{2}\mathbf{Z}^\top \Sigma(\theta)^{-1}\mathbf{Z}. \tag{1}$$

Fig. 1(b) explains in detail how to compute the likelihood function in five steps: (1) generate the covariance function $\Sigma(\theta)$ from an initial parameter vector θ_{init}; (2) Compute the inverse of $\Sigma(\theta)$ using the Cholesky factorization operation $\Sigma(\theta) = \mathbf{L} \cdot \mathbf{L}^\top$; (3) Compute the log determinant of $\Sigma(\theta)$; (4) Solve the system of linear equations $\mathbf{L}^{-1}\mathbf{Z}$; (5) Compute the dot product $\mathbf{Z}^\top \cdot \mathbf{Z}$. The constant expression $c = -\frac{n}{2}\log(2\pi)$ can be ignored since it does not affect the optimization problem.

5.1.3. Geospatial data prediction

Data modeling using MLE produces a set of statistical parameters ($\hat{\theta}$) that can be used in predicting missing values. Assuming unknown values $\mathbf{Z}_1$ with size m and known values $\mathbf{Z}_2$ with size n, and $\mathbf{Z}_2$ has a zero-mean function, so $\mathbf{Z}_1$ can be obtained as

$$\mathbf{Z}_1 = \Sigma_{12}\Sigma_{22}^{-1}\mathbf{Z}_2, \tag{2}$$

where Σ_{12} is the covariance matrix generated from the distance matrix between the locations of the missing values and the observed values, Σ_{22} is the covariance matrix generated from the distance matrix between the locations of the observed values. The proof and the explanation of Eq. (2) can be found on [33]. Fig. 1(c) explains the prediction operation in detail.

5.2. Parallel linear algebra mathematics in ExaGeoStat

ExaGeoStat enables parallel computation of the MLE operation through the state-of-the-art parallel linear algebra libraries, Chameleon for dense computation, and HiCMA for Tile Low-Rank (TLR) approximation [11]. Both libraries rely on the StarPU runtime system to enable porting the code into different parallel hardware systems, including GPUs. Through tile-based algorithms, Chameleon and HiCMA performed better on shared-memory and distributed-memory systems compared to block-based algorithms in LAPACK [34] and ScaLAPACK [35]. In tile-based algorithms, matrix tiles are distributed to different processing units to allow faster execution of the underlying linear algebra operations that can accelerate applications that mainly rely on linear algebra such as *ExaGeoStat* [8].

In HiCMA, low-rank approximation has been used in each matrix tile with a certain accuracy by compressing each tile using Singular
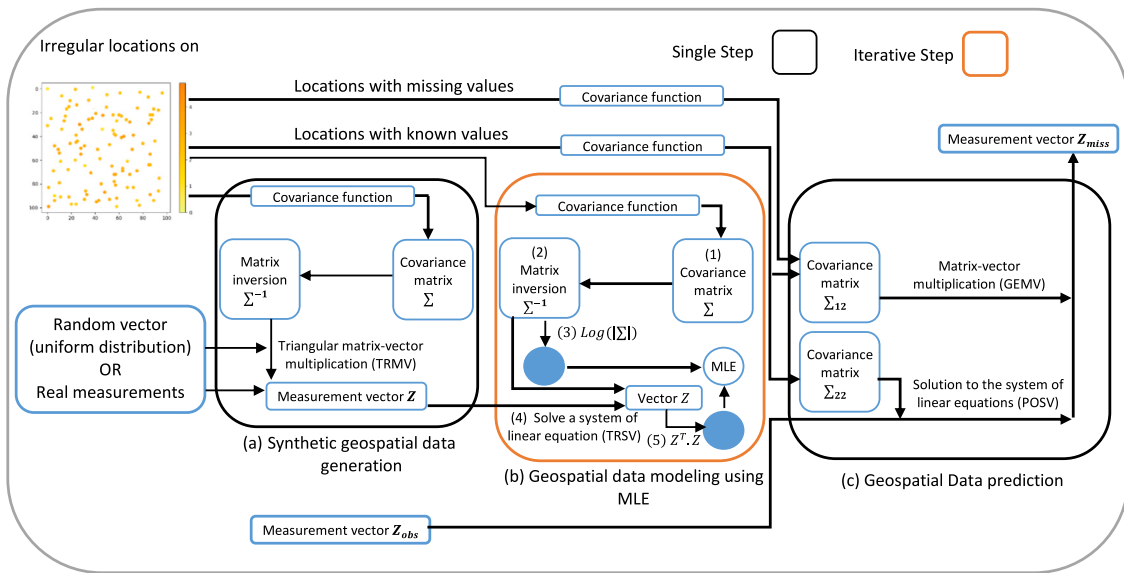
Fig. 1. *ExaGeoStat* main operations: (a) Synthetic geospatial data generation (b) Geospatial data modeling using Maximum Likelihood Estimation (MLE) (c) Geospatial data prediction.

Value Decomposition (SVD). SVD can be replaced by Randomized SVD (RSVD), or Adaptive Cross Approximation (ACA), to approximate each off-diagonal tile up to a certain user-defined accuracy threshold. Using SVD, each tile (i, j) in the covariance matrix can be represented by the product of two matrices $\mathbf{U}_{ij}$ and $\mathbf{V}_{ij}$, with a size of $nb \times k$, where nb represents the tile size which is a tunable parameter that has an impact on the overall performance. TLR approximation can compress the whole covariance matrix and improve the performance of executing linear algebra operations on large-scale systems. More information about TLR *ExaGeoStat* can be found in [31].

6. HPC-ready containers creation mechanism

Like many existing HPC software, *ExaGeoStat* relies on several software dependencies to improve the code portability across various hardware architectures. *ExaGeoStat* depends on several libraries, including BLAS, CUDA, MPI, NLOPT, HWLOC, GSL, StarPU, Chameleon, STARS-H, and HiCMA [11]. Therefore, using containers is essential to facilitating seamless code portability.

In this work, we propose generating containers for HPC software that can be easily created and adapted for different hardware architectures. In this section, we describe the proposed HPC-Ready containers in the context of *ExaGeoStat* software. The eFlows4HPC project has proposed the Container Image Creation service,[3] which aims at simplifying the creation of container images for HPC systems. This service leverages Docker buildx and Spack environments to automatize the generation of the containers for specific HPC platforms. On one hand, Docker Buildx allows the Container Image Creation service to build container images for various Linux platforms (x86_64, amd64, arm64, and ppc64) regardless of the server architecture. On the other hand, the Spack package manager enables the service to compile and configure the necessary software dependencies according to the features of the target HPC system. It allows developers to use containers to facilitate deployment on HPC platforms. To run the software on a new hardware system, the developer simply needs to copy or pull the container image. This allows for easy execution without additional steps and achieves the same performance as a native installation.

Fig. 2 shows the complete pipeline the Container Image Creation service uses to create the HPC-ready containers. The Repository contains a generic and platform-agnostic Spack environment `spack.yaml` manifest file per application that includes the software required to build an image for an HPC application without including any platform-related information (CPU architecture, accelerators, ...). The repository also stores a generic `package.py` that includes the installation description according to the Spack schemas (the steps and dependencies needed to install the HPC application). Since these files are generic and platform-agnostic, the developer only needs to write these two files once, as they can be reused to create containers for different HPC platforms. The Container Image Creation service will concretize the Spack environment with the specific features of the HPC machine to build the HPC-ready containers. As an example, for the *ExaGeoStat* case, we store the Spack environment manifest file at https://github.com/eflows4hpc/workflow-registry/tree/main/kaust/exageostat/spack.yaml and Spack installation descriptions (package.py) for ExaGeoStat and its dependencies at https://github.com/eflows4hpc/software-catalog/tree/main/packages/exageostat.

To request the creation of a container image, the developer should create a JSON document to indicate the application to build and the characteristics of the specific HPC system. As shown by the given example in Fig. 3, the JSON document has several fields. The field `machine` specifies the target platform and architecture of the underlying system. In the example, two optional fields are specified to indicate that we are creating an image to be managed by Singularity and a specific MPI version needed in that system. The fields `workflow` and `step_id` are used to obtain the `spack.yml` and the `package.py` files in the repository.

The service uses all this data to create a container build context. The service extends the generic Spack environment with the machine features provided in the machine description as depicted in Fig. 4. The extended environment will specify the original required software packages and the specific MPI and GPU runtimes supported by the target HPC machine. It will also configure the environment to ensure that all the software packages are considering the optimizations for the CPU architecture available in the HPC system regardless of the host where the image is built, and we are also configuring the shared linking to enable the loading of the host MPI and GPU drivers and libraries at execution time. Apart from the extended environment, the build context will include the required Spack descriptions downloaded from

---

[3] https://github.com/eflows4hpc/image_creation.
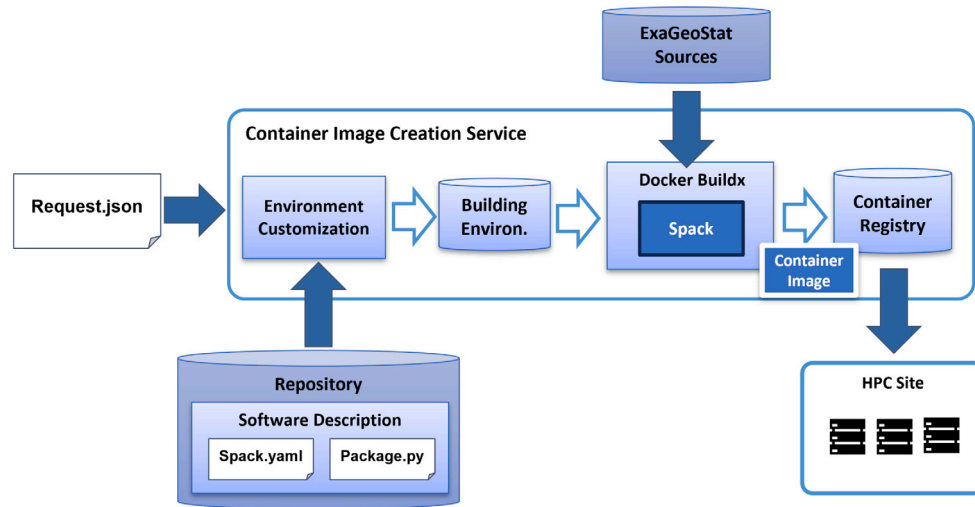
Fig. 2. Pipeline for the creation of HPC-ready containers.

```
{
    "machine": {
        "platform": "linux/amd64",
        ↪  "architecture": "skylake",
        ↪  "container_engine": "singularity",
        ↪  "mpi":"openmpi@4.0.1"
        },
    "workflow":"kaust",
    "step_id" :"exageostat"
}
```

Fig. 3. Container configuration (JSON file). The machine section describes the target HPC system, and the workflows and step_id sections provide the reference to the location of the generic Spack file in the Repository.

```
# Generic enviroment
spack:
    specs:
        - exageostat@master

---
# Customized environment
spack:
    specs:
        - exageostat@master
        - openmpi@4
    concretizer:
        reuse: true
        targets:
            granularity: microarchitectures
            host_compatible: false
        unify: true
    config:
        shared_linking:
            type: runpath
    packages:
        all:
            target:
                - skylake
    view: /opt/software/
```

Fig. 4. Spack environment customization.

the repository and a Dockerfile, which executes the Spack command to install the environment in the container image.

After generating the build context, the service invokes the Docker buildx tool, passing the generated build context and the platform specified in the machine description. The Docker buildx tool enables the creation of container images regardless of the underlying host platform. If the requested platform is different from the host running the container image creation service, the buildx tool will create an emulated environment with the desired platform to build the container image. Within the build process, Spack is called with the corresponding options to build the specific image. In the case of *ExaGeoStat*, the Spack build will download the source code of *ExaGeoStat*, its dependencies, and the corresponding MPI runtime version. Finally, it installs all of them according to the target HPC platform features.

After the build process, the container image is stored in the repository with accompanying metadata indicating the characteristics of the target machine. This is useful to detect if an image can be reused for another request, which requires the build the same application for an HPC platform with the same characteristics. Once it is stored, the image is converted to the format which is supported by the target machine. In the case of the example, the generated image is converted to a Singularity Image Format (SIF).

To manage the creation of the container images, the service provides an API with three methods to build, check the status, and download the image. To simplify the access to the service, we have developed a simple bash client to invoke the service. Fig. 5 shows the usage of this client.

7. Experimental evaluation

In this section, we assess the efficiency of the HPC-ready containers using the *ExaGeoStat* software by conducting experiments in different parallel architectures and comparing the performance with the native build. We rely on shared-memory systems with/without accelerators and a distributed-memory system to evaluate the performance of the Singularity images under various workloads.

7.1. Experimental testbed

To assess the performance and the portability of the HPC-ready containers, we use a wide range of manycore systems: a dual-socket 18-core Intel Haswell Intel Xeon E5-2699 running at 2.30 GHz, a dual-socket 28-core Intel Skylake Intel Xeon Platinum 8176 running at 2.10 GHz, a dual-socket 32-core AMD Naples AMD EPYC 7601 running

```
 1  $> cic_cli https://eflows4hpc.bsc.es build exageostat_request.json
 2  Response:
 3  {"id":"f1f4699b-9048-4ecc-aff3-1c689b855adc"}
 4
 5  $> cic_cli https://eflows4hpc.bsc.es status f1f4699b-9048-4ecc-aff3-1c689b855adc
 6  Response:
 7  {"filename":"reduce_order_model_sandybridge.sif", "image_id":"ghcr.io/eflows4hpc/reduce_order_model_sandybridge",
    ↪  "message":null,"status":"FINISHED"}
 8
 9  $> cic_cli https://eflows4hpc.bsc.es download exageostat_skylake.sif
10  Response:
11  Resolving eflows4hpc.bsc.es (eflows4hpc.bsc.es)... 84.88.52.251
12  Connecting to eflows4hpc.bsc.es (eflows4hpc.bsc.es)|84.88.52.251|:443... connected.
13  HTTP request sent, awaiting response... 200 OK
14  Length: 2339000320 (2.2G) [application/octet-stream]
15  Saving to: 'exageostat_skylake.sif'
16
17  exageostat_skylake.sif        0%[                          ]   4.35M   550KB/s    eta 79m 0s
```

Fig. 5. Client usage. The first command shows how to request the creation of a container image. The second command shows how to test the status of the creation. The last command shows how to download the created image.

at 2.2 GHz, a dual-socket 64-core AMD Milan AMD EPYC 7713, a dual-socket 28-core Intel Icelake Intel Xeon Gold 6330 CPU running at 2.00 GHz equipped with 4 NVIDIA A100 GPUs, a dual-socket 20-core Intel Skylake Intel Xeon Gold 6148 running at 2.40 GHz equipped with 2 NVIDIA V100 GPUs. We also rely on MareNostrum 4 system, a Barcelona Supercomputing Center (BSC) supercomputer with 3,456 Lenovo ThinkSystem compute nodes where each node has two 24-core Intel Xeon Platinum chips (a total of 165,888) processors and 390 terabytes memory. In our experiments, we use up to 256 nodes on this system.

We rely on GCC v10.2.0 to compile and link against HWLOC v2.8.0, GSL v2.7.1, StarPU v1.3.9, Intel MKL v11.3.1, NLOPT v2.7.0, HiCMA v0.1.1, Chameleon v0.9.2, STARS-H v0.1.0, and ExaGeoStat v1.1.0. For all the CPU-based architectures, the Singularity image size is about 2.2 GB, the CPU/GPU-based image size is about 8.4 GB, and the MPI-based (only CPU) image size is about 2.3 GB.

All the performance results are an average of 10 different runs for the three primary operations in *ExaGeoStat*: data generation, modeling, and prediction. In addition, the iterative modeling operation has been performed for 10 iterations. This mechanism allows better performance assessment to avoid any errors in the runs.

7.2. Containers portability on shared-memory systems

In this subsection, we aim to evaluate the performance of the Singularity images compared to the native build of the software and its dependencies using shared-memory systems. Fig. 6 shows the assessment results on five different CPU-based architectures. The runs include different workloads representing the number of spatial locations on the x-axis. Considering many spatial locations produces a larger covariance matrix, which requires more memory and computing resources. Each subfigure represents a specific hardware architecture and includes three subfigures. On the left, the subfigure represents the execution time to perform each operation in the software in the dense mode, where the y-axis represents the execution time in seconds. In the middle, the subfigure represents the average performance in GFlops/s (y-axis) under different workloads and in dense mode. Finally, on the right, the subfigure shows the execution time of the three operations when applying TLR approximation under various workloads.

Fig. 6(a) shows the performance comparison using an Intel Haswell chip. The variation in performance in the dense computation between the Singularity image and the native build is between −1% and +0.4%, while in TLR execution, it is between −1% and +2%. The variation when running on Intel Skylake is between −0.8% and +1% in the dense case, as shown in Fig. 6(b). In the TLR case, the variation is between −2% and +0.8%. Fig. 6(c) shows the performance variation when the Intel Icelake chip is used. The variation between the Singularity image and the native build-in performance is between −7% and +2% in the

dense case and −4% and +2% in the TLR case. On the AMD Milan chip, the variation in performance is −1% and −0.4% between the image and native build in the dense case and between in the TLR case as shown in Fig. 6(d). Finally, on the AMD Naples chip, the variation in performance between −0.2% and 1% between the image and native build in the dense case and between −2% and −1% in the TLR case as shown in Fig. 6(e).

We also use two shared-memory systems with GPU accelerators to assess the performance, i.e., 20-core Intel Skylake with NVIDIA V100 GPU and 20-core Intel Icelake with A100 GPU. We report the variation in performance using just the dense case since the software does not support TLR when relying on GPUs. Fig. 7 shows the variation in performance on the two systems. The variation in performance between the Singularity image and the native build using the NVIDIA V100 + Intel Skylake CPU is between −3% and −1% while using the NVIDIA A100 + Intel Icelake CPU is between −2% and +1% for the dense case. The degradation in performance with larger problem sizes is because of the GPU memory wall, so the software starts to use the CPU memory and move the data between the two devices when needed.

7.3. Containers portability and scalability on distributed-memory systems

We also conducted a set of experiments on the Marenostrum 4 supercomputer at BSC using up to 256 nodes to validate the performance of the generated images compared to the native build. Herein, we decided to include only the dense experiments because of the page limits and because we found that the variation in performance between the generated image and the native build is the same in the TLR case. Fig. 8 shows the average performance in TFlops/s for the Singularity image and the native build. Fig. 8(a) shows the performance in TFlops/s using 4 nodes, which reaches close to 7 TFlops/s. We tuned the tile size (ts), and the best performance is obtained when $ts = 760$. The performance of the Singularity image varies by ±1% compared to the native build. We can consider that there is no difference between both cases since the variation in different runs of the native build is about ±1%. Fig. 8(b) shows the performance difference on 16 nodes. The best performance has been obtained when $ts = 760$ – up to 22 TFlops/s. The singularity image shows 0.7 to 7% lower performance than the native build. This can also be acceptable since the variation in different runs of the native build can reach this percentage. The singularity image shows −2 to +0.5% compared to the native build using $ts = 760$ and Using 64 nodes as shown by Fig. 8(c). The obtained performance in both cases reaches up to 60 TFlops/s. Fig. 8(d) shows that the performance of the Singularity image is 0.5 to 2% lower than the native build using 256 nodes, and the performance can reach up to 144 TFlops/s in both cases.

In Figs. 6, 7, and 8, we tuned the tile size for each problem size in each architecture. In Fig. 9, we compare the performance of the HPC-ready containers with the native build using 256 nodes on
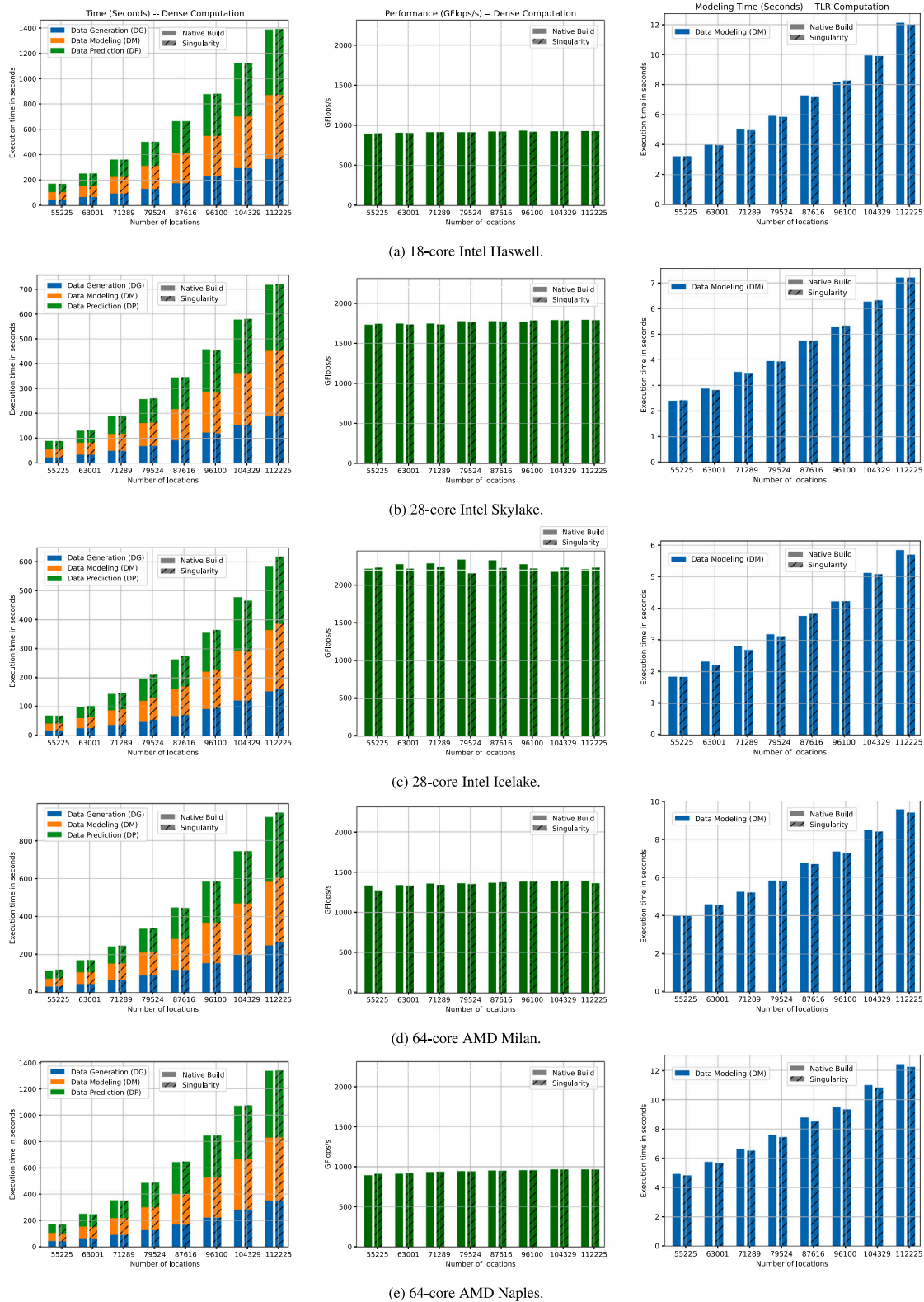
(a) 18-core Intel Haswell.

(b) 28-core Intel Skylake.

(c) 28-core Intel Icelake.

(d) 64-core AMD Milan.

(e) 64-core AMD Naples.

Fig. 6. Performance assessment on CPU-based shared-memory systems.

MareNostrum 4 using different tile sizes regardless of the obtained performance. The obtained results vary from around 28 TFlops/s using $ts = 320$ to about 144 TFlops/s using $ts = 960$. The variation in performance between containers and native build is between −3% (matrix size = 202 500, and $ts = 760$) to +10% (matrix size = 202 500, and $ts = 320$) in the worst case.

8. Discussion

The method proposed in this article combines multi-platform container builders with HPC package managers to enable application portability on different HPC systems. By integrating the capabilities of container builders and HPC package managers, we can create the
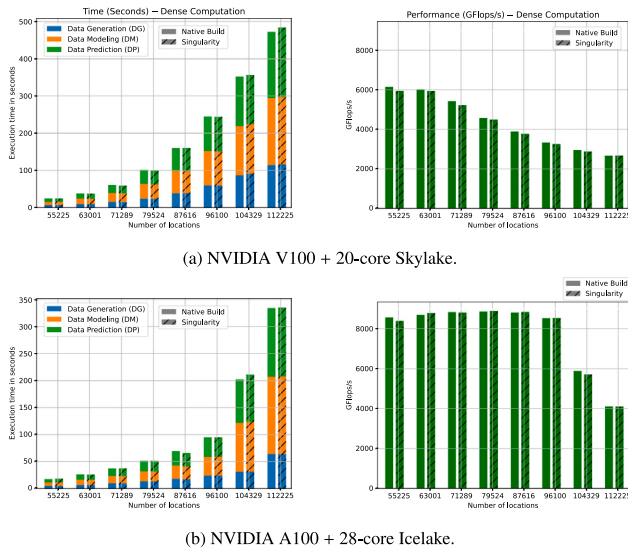
(a) NVIDIA V100 + 20-core Skylake.

(b) NVIDIA A100 + 28-core Icelake.

Fig. 7. Performance on heterogeneous shared-memory systems.

(a) 4 nodes.
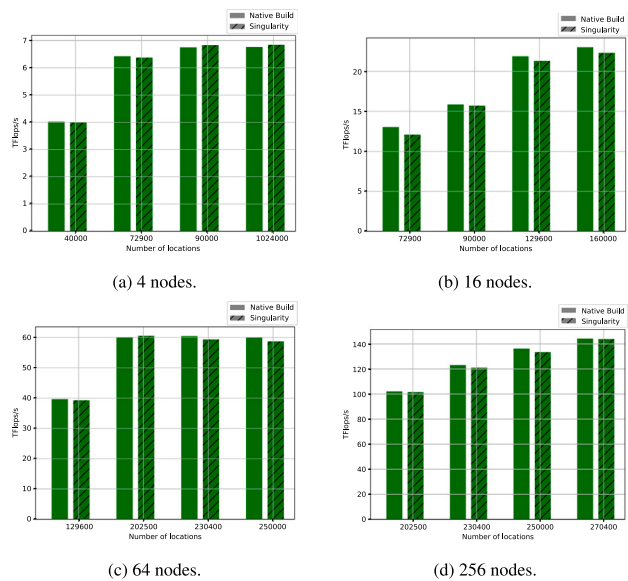
(b) 16 nodes.

(c) 64 nodes.

(d) 256 nodes.

Fig. 8. Performance on MareNostrum 4 supercomputer using a different number of nodes. Tile Size ts has been tuned for each number of nodes. (a) $ts = 760$ (b) $ts = 760$ (c) $ts = 760$ (d) $ts = 960$.

container images offline and move them later to the target HPC system. This process relies on software requirements and information about the target HPC system, eliminating the need to build on systems with restricted access privileges. Once the image for a particular system is created, it is stored in an image repository for reuse. In this implementation, we utilize Docker buildx for building images across multiple platforms and Spack as the HPC package manager.

Existing solutions for container images have two main drawbacks: root access is required to build the image or part of it on the target system on shared-access HPC systems, and the performance may not meet expectations if the image is built on a different system. Our proposed method for building HPC-ready containers overcomes these two issues, offering a more reliable solution for the HPC community to port their software across different HPC systems. One limitation of our building process is that it requires a custom programming paradigm, requiring users to have some experience creating and building their images to guarantee performance. However, the traditional method for

creating container images is also not straightforward, demands effort from the user, and is not fully suitable for HPC systems.

In this study, we also evaluate the performance of our HPC-ready containers using ExaGeoStat software, which requires several dependencies that must be fine-tuned for different hardware architectures. ExaGeoStat is an ideal candidate for our assessment because it shares the same complexity as many existing HPC applications, with its performance being highly influenced by the installation process and underlying hardware architecture. Our results demonstrate that our proposed container images can maintain performance across different architectures, even when built on a local machine far from the target system.

9. Conclusions

Containers can be a "gold solution" to increase the usability of the HPC software in scientific domains through their ability to preserve performance while allowing easy portability on different HPC systems. Unlike Virtual Machines (VM), containers take up less space, consume fewer resources, and are easier to manage and scale as they can be quickly cloned and deployed. This paper assesses the portability and scalability of pre-generated Singularity images through a parallel geospatial statistics software *ExaGeoStat*, which allows large-scale synthetic geospatial data generation, modeling, and prediction on manycore systems. We detail how to create HPC-ready containers based on the Spack package manager. Our results show how the pre-generated containers perform compared to the native build of the *ExaGeoStat* software. Our experiments explore seven distinct shared-memory systems, two of which are equipped with NVIDIA accelerators. Additionally, we utilize the MareNostrum 4 supercomputer at the Barcelona Supercomputing Center to evaluate the performance of containers on a distributed-memory system. Our findings indicate that HPC-ready containers demonstrate performance levels nearly equivalent to native builds while offering enhanced portability across various architectures.

CRediT authorship contribution statement

Sameh Abdulah: Conceptualization, Investigation, Software, Validation, Writing – original draft, Writing – review & editing. **Jorge Ejarque:** Software, Validation, Writing – original draft. **Omar Marzouk:** Software, Validation. **Hatem Ltaief:** Writing – review & editing. **Ying Sun:** Writing – review & editing. **Marc G. Genton:** Conceptualization, Writing – review & editing. **Rosa M. Badia:** Conceptualization, Project administration, Supervision, Writing – original draft, Writing – review & editing. **David E. Keyes:** Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

(a) $ts = 320$.

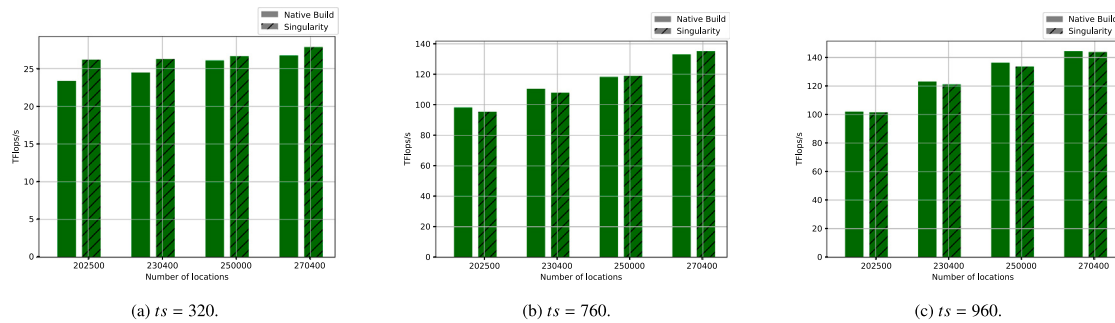(b) $ts = 760$.

(c) $ts = 960$.

Fig. 9. Performance on 256 nodes of MareNostrum 4 with different tile sizes.

References

[1] eFlows4HPC consortium, Deliverables D1.1 and D1.3: Requirements metrics and arquitecture design, 2022, https://eflows4hpc.eu/deliverables/, (Accessed 13 May 2024).

[2] C. Boettiger, An introduction to docker for reproducible research, Oper. Syst. Rev. 49 (1) (2015) 71–79.

[3] I. Habib, Virtualization with KVM, Linux J. 2008 (166) (2008) 8.

[4] P. Li, Selecting and using virtualization solutions: our experiences with VMware and VirtualBox, J. Comput. Sci. Coll. 25 (3) (2010) 11–17.

[5] B. Đorđević, V. Timčenko, O. Pavlović, N. Davidović, Performance comparison of native host and hyper-based virtualization VirtualBox, in: 2021 20th International Symposium INFOTEH-JAHORINA, INFOTEH, IEEE, 2021, pp. 1–4.

[6] T. Abels, P. Dhawan, B. Chandrasekaran, An overview of xen virtualization, Dell Power Solut. 8 (2005) 109–111.

[7] T. Gamblin, M. LeGendre, M.R. Collette, G.L. Lee, A. Moody, B.R. De Supinski, S. Futral, The Spack package manager: bringing order to HPC software chaos, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, pp. 1–12.

[8] S. Abdulah, H. Ltaief, Y. Sun, M.G. Genton, D.E. Keyes, ExaGeoStat: A high performance unified software for geostatistics on manycore systems, IEEE Trans. Parallel Distrib. Syst. 29 (12) (2018) 2771–2784.

[9] S. Abdulah, Y. Li, J. Cao, H. Ltaief, D.E. Keyes, M.G. Genton, Y. Sun, Large-scale environmental data science with ExaGeoStatR, Environmetrics (2022) e2770.

[10] Chameleon, a dense linear algebra software for heterogeneous architectures, 2023, https://project.inria.fr/chameleon, (Accessed 01 March 2023).

[11] S. Abdulah, K. Akbudak, W. Boukaram, A. Charara, D.E. Keyes, H. Ltaief, A. Mikhalev, D. Sukkari, G. Turkiyyah, Hierarchical computations on manycore architectures (HiCMA), 2019, See http://github.com/ecrc/hicma.

[12] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: a unified platform for task scheduling on heterogeneous multicore architectures, in: Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, the Netherlands, August 25-28, 2009. Proceedings 15, Springer, 2009, pp. 863–874.

[13] L. Youseff, R. Wolski, B. Gorda, C. Krintz, Evaluating the performance impact of Xen on MPI and process execution for HPC systems, in: First International Workshop on Virtualization Technology in Distributed Computing, VTDC 2006, IEEE, 2006, p. 1.

[14] J.P. Walters, V. Chaudhary, M. Cha, S. Guercio, S. Gallo, A comparison of virtualization technologies for HPC, in: 22nd International Conference on Advanced Information Networking and Applications, Aina 2008, IEEE, 2008, pp. 861–868.

[15] J.N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, J. Owens, Quantifying the performance isolation properties of virtualization systems, in: Proceedings of the 2007 Workshop on Experimental Computer Science, 2007, pp. 6–es.

[16] M.F. Mergen, V. Uhlig, O. Krieger, J. Xenidis, Virtualization for high-performance computing, ACM SIGOPS Oper. Syst. Rev. 40 (2) (2006) 8–11.

[17] G. Somani, S. Chaudhary, Application performance isolation in virtualization, in: 2009 IEEE International Conference on Cloud Computing, IEEE, 2009, pp. 41–48.

[18] M.G. Xavier, M.V. Neves, F.D. Rossi, T.C. Ferreto, T. Lange, C.A. De Rose, Performance evaluation of container-based virtualization for high performance computing environments, in: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE, 2013, pp. 233–240.

[19] O. Rudyy, M. Garcia-Gasulla, F. Mantovani, A. Santiago, R. Sirvent, M. Vázquez, Containers in hpc: A scalability and portability study in production biological simulations, in: 2019 IEEE International Parallel and Distributed Processing Symposium, IEEE, 2019, pp. 567–577.

[20] G.M. Kurtzer, V. Sochat, M.W. Bauer, Singularity: Scientific containers for mobility of compute, PLoS One 12 (5) (2017) e0177459.

[21] L. Gerhardt, W. Bhimji, S. Canon, M. Fasel, D. Jacobsen, M. Mustafa, J. Porter, V. Tsulaia, Shifter: Containers for HPC, in: Journal of Physics: Conference Series, Vol. 898, IOP Publishing, 2017, 082021.

[22] C. Ruiz, E. Jeanvoine, L. Nussbaum, Performance evaluation of containers for HPC, in: European Conference on Parallel Processing, Springer, 2015, pp. 813–824.

[23] S. Abraham, A.K. Paul, R.I.S. Khan, A.R. Butt, On the use of containers in high performance computing environments, in: 2020 IEEE 13th International Conference on Cloud Computing, CLOUD, IEEE, 2020, pp. 284–293.

[24] H. Gantikow, S. Walter, C. Reich, Rootless containers with Podman for HPC, in: High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers, Springer-Verlag, Berlin, Heidelberg, ISBN: 978-3-030-59850-1, 2020, pp. 343–354.

[25] J. Higgins, V. Holmes, C. Venters, Orchestrating docker containers in the HPC environment, in: International Conference on High Performance Computing, Springer, 2015, pp. 506–513.

[26] A. Torrez, T. Randles, R. Priedhorsky, HPC container runtimes have minimal or no performance impact, in: 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC, CANOPIE-HPC, IEEE, 2019, pp. 37–42.

[27] J. Ejarque, R.M. Badia, L. Albertin, G. Aloisio, E. Baglione, Y. Becerra, S. Boschert, J.R. Berlin, A. D'Anca, D. Elia, et al., Enabling dynamic and intelligent workflows for HPC, data analytics, and AI convergence, Fut. Gen. Comput. Syst. 134 (2022) 414–429.

[28] G. Li, J. Woo, S.B. Lim, HPC cloud architecture to reduce HPC workflow complexity in containerized environments, Appl. Sci. 11 (3) (2021) 923.

[29] S. McMillan, Making containers easier with hpc container maker, in: Proceedings of the SIGHPC Systems Professionals Workshop, HPCSYSPROS 2018, Vol. 10, Dallas, TX, USA, 2018.

[30] Q. Cao, S. Abdulah, R. Alomairy, Y. Pei, P. Nag, G. Bosilca, J. Dongarra, M.G. Genton, D.E. Keyes, H. Ltaief, et al., Reshaping geostatistical modeling and prediction for extreme-scale environmental applications, in: 2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, IEEE Computer Society, 2022, pp. 13–24.

[31] S. Abdulah, H. Ltaief, Y. Sun, M.G. Genton, D.E. Keyes, Parallel approximation of the maximum likelihood estimation for the prediction of large-scale geostatistics simulations, in: 2018 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2018, pp. 98–108.

[32] L. Benedicic, F.A. Cruz, A. Madonna, K. Mariotti, Sarus: Highly scalable docker containers for hpc systems, in: High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers 34, Springer, 2019, pp. 46–60.

[33] M.G. Genton, Separable approximations of space-time covariance matrices, Environ.: Off. J. Int. Environ. Soc. 18 (7) (2007) 681–695.

[34] E. Anderson, Z. Bai, C. Bischof, L.S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al., LAPACK Users' Guide, SIAM, 1999.

[35] J. Choi, J.J. Dongarra, R. Pozo, D.W. Walker, Scalapack: A scalable linear algebra library for distributed memory concurrent computers, in: The Fourth Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society, 1992, pp. 120–121.

Sameh Abdulah is a research scientist at the Extreme Computing Research Center, King Abdullah University of Science and Technology, Saudi Arabia. Sameh received his M.S. and Ph.D. from Ohio State University, Columbus, US, in 2014 and 2016, respectively. His work centers around high-performance computing applications, especially statistical modeling and prediction, bitmap indexing in big data, algorithm based fault tolerance, and machine learning and data mining algorithms. In 2022, he stood as a finalist for the ACM Gordon Bell Award as a part of the KAUST team at the SC conference. He is also a member of the IEEE,

ACM, SIAM, and American Statistical Association (ASA) communities.

Jorge Ejarque holds Ph.D. on Computer Science (2015) from the Technical University of Catalonia (UPC). From 2005 to 2008 he worked as research support engineer at the UPC, and joined Barcelona Supercomputing Center BSC at the end of 2008. He has contributed in the design and development of different tools and programming models for complex distributed computing platforms. He has published over 30 research papers in conferences and journals and he has been involved in several National and European R&D projects (FP6, FP7, H2020 and Horizon Europe). He is member of a program committee of several international conferences, reviewer of journal articles and he was member of the Spanish National Grid Initiative panel. His current research interests are focused on parallel programming models for heterogeneous parallel distributed computing environments and the interoperability between distributed computing platforms. He is the main architect of the eFlows4HPC project.

Omar Marzouk is an HPC Senior Software Engineer, part of the Advanced Computing and AI Team at Brightskies Inc., Egypt.

Hatem Ltaief is the Principal Research Scientist of the Extreme Computing Research Center, King Abdullah University of Science and Technology (KAUST), Saudi Arabia. He received a Masters in applied mathematics in 2004 and a Ph.D. in computer science in 2007 from the University of Houston. His research interests include parallel numerical algorithms, parallel programming models, high-performance computing, and performance optimizations for multicore architectures and hardware accelerators. He received several best papers awards at ACM PASC'18 and EuroPar'16'20 conferences as well as the GAUSS award for best paper at ISC'20. He was among the Gordon Bell finalists team at SC22'23. He is currently the vice chair of SIAM activity group on supercomputing and the IEEE TCPP coordinator for Africa, Europe, and Middle East.
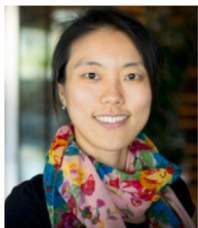
Ying Sun is an Associate Professor of Statistics at King Abdullah University of Science and Technology (KAUST). She received her Ph.D. degree in statistics from Texas A&M University in 2011. At KAUST, she established and leads the Environmental Statistics Research Group, developing statistical models and methods for complex data to address important environmental problems. She has made original contributions to environmental statistics, in particular in the areas of spatiotemporal statistics, functional data analysis, visualization, and computational statistics, with an exceptionally broad array of applications.

Marc G. Genton is Al-Khawarizmi Distinguished Professor of Statistics at the King Abdullah University of Science and Technology (KAUST) in Saudi Arabia. He received the Ph.D. degree in Statistics (1996) from the Swiss Federal Institute of Technology (EPFL), Lausanne. He is a fellow of the

American Statistical Association (ASA), of the Institute of Mathematical Statistics (IMS), and the American Association for the Advancement of Science (AAAS), and is an elected member of the International Statistical Institute (ISI). In 2010, he received the El-Shaarawi award for excellence from the International Environmetrics Society (TIES) and the Distinguished Achievement award from the Section on Statistics and the Environment (ENVR) of the American Statistical Association (ASA). He received an ISI Service award in 2019 and the Georges Matheron Lectureship award in 2020 from the International Association for Mathematical Geosciences (IAMG). He led a Gordon Bell Prize finalist team with the ExaGeoStat software for Super Computing 2022. He received the Royal Statistical Society (RSS) 2023 Barnett Award for his outstanding research in environmental statistics. His research interests include statistical analysis, flexible modeling, prediction, and uncertainty quantification of spatio-temporal data, with applications in environmental and climate science, as well as renewable energies.

Rosa M. Badia holds a Ph.D. on Computer Science (1994) from the Technical University of Catalonia (UPC). She is the manager of the Workflows and Distributed Computing research group at the Barcelona Supercomputing Center (BSC). She is considered one of the key researchers in Parallel programming models for multicore and distributed computing due to her contribution to task-based programming models during the last 15 years. The research group focuses on PyCOMPSs/COMPSs, a parallel task-based programming distributed computing, and its application to the development of large heterogeneous workflows that combine HPC, Big Data, and Machine Learning. The group is also doing research around the dislib, a parallel machine learning library parallelized with PyCOMPSs. Dr Badia has published near 200 papers in international conferences and journals on the topics of her research. She has been very active in projects funded by the European Commission in contracts with industry, and currently she is the PI of the EuroHPC project eFlows4HPC. She is a member of HiPEAC Network of Excellence. She received the Euro-Par Achievement Award 2019 for her contributions to parallel processing, the DonaTIC award, category Academia/Researcher in 2019 and the HPDC Achievement Award 2021 for her innovations in parallel task-based programming models, workflow applications and systems, and leadership in the high performance computing research community. In 2023, she has been invited to be a member of the Institut d'Estudis Catalans (Catalan academy).

David Keyes directs the Extreme Computing Research Center at KAUST, where he was a founding Dean in 2009 and is a professor of Applied Mathematics, Computer Science, and Mechanical Engineering. He works at the interface between parallel computing and PDEs and statistics. Before joining KAUST, Keyes led multi-institutional scalable solver software projects in the SciDAC and ASCI programs of the US Department of Energy (DoE), ran university collaboration programs at US DoE and NASA institutes, and taught at Columbia, Old Dominion, and Yale Universities. He is a Fellow of SIAM, the AMS, and the AAAS. He has been awarded the Gordon Bell Prize from the ACM, the Sidney Fernbach Award from the IEEE Computer Society, and the SIAM Prize for Distinguished Service to the Profession. He earned a B.S.E. in Aerospace and Mechanical Sciences from Princeton in 1978 and a Ph.D. in Applied Mathematics from Harvard in 1984.