

NVIDIA Grace Superchip Early Evaluation for HPC Applications

Fabio Banchelli*
fabio.banchelli[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Joan Vinyals-Ylla-Catala
joan.vinyals[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Josep Pocurull
josep.pocurull[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Marc Clascà
marc.clasc[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Kilian Peiro
kilian.peiro[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Filippo Spiga
fspiga[at]nvidia.com
NVIDIA Inc.
Cambridge, UK

Marta Garcia-Gasulla
marta.garcia[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

Filippo Mantovani
filippo.mantovani[at]bsc.es
Barcelona Supercomputing Center
Barcelona, Spain

ABSTRACT

Arm-based system in HPC are a reality since more than a decade. However, when a new chip enters the market always implies challenges, not only at ISA level, but also with regards to the SoC integration, the memory subsystem, the board integration, the node interconnection, and finally the OS and all layers of the system software (compiler and libraries). Guided by the procurement of an NVIDIA Grace HPC cluster within the deployment of MareNostrum 5, and emulating the approach of a scientist who needs to migrate its scientific research to a new HPC system, we evaluated five complex scientific applications on engineering sample nodes of NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip (CPU-only). We report intra-node and inter-node scalability and early performance results showing a speed-up between 1.3× and 4.28× for all codes when compared to the current generation of MareNostrum 4 powered by Intel Skylake CPUs.

CCS CONCEPTS

• **Hardware** → **Emerging architectures**; • **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → *Multicore architectures*.

KEYWORDS

NVIDIA Grace, NVIDIA Superchip, Arm, HPC, Performance, Scalability, Evaluation, Cluster

ACM Reference Format:

Fabio Banchelli, Joan Vinyals-Ylla-Catala, Josep Pocurull, Marc Clascà, Kilian Peiro, Filippo Spiga, Marta Garcia-Gasulla, and Filippo Mantovani. 2024.

*Corresponding author

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in: Fabio Banchelli, Joan Vinyals-Ylla-Catala, Josep Pocurull, Marc Clascà, Kilian Peiro, Filippo Spiga, Marta Garcia-Gasulla, and Filippo Mantovani. 2024. NVIDIA Grace Superchip Early Evaluation for HPC Applications. In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsia '24 Workshops). Association for Computing Machinery, New York, NY, USA, 45–54. <https://doi.org/10.1145/3636480.3637284>

NVIDIA Grace Superchip Early Evaluation for HPC Applications. In *International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsiaWS 2024)*, January 25–27, 2024, Nagoya, Japan. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3636480.3637284>

1 INTRODUCTION AND RELATED WORK

For more than a decade, Arm, the leading architecture in the mobile and embedded market, has entered the HPC market, powering servers that serve as the basic building blocks of both HPC systems [7, 14], as well as cloud commercial deployments with the AWS Graviton series based on the Arm Neoverse architecture¹.

While the ISA plays an important role at the level of chip design, the actual “alignment of stars” necessary for an ISA to be adopted in a production system enabling real scientific applications is much more complex.

Conducting real scientific research on an HPC cluster requires functional integration at the chip level (SoC design), compute node level (interconnection to the main memory and board design), and full system level (multi-node integration into a cluster).

In parallel to hardware integration, significant effort at the system software level needs to be undertaken. Complex scientific applications require support for multiple languages (e.g., FORTRAN, C/C++, Python) as well as the availability of libraries enabling basic functions external to the scientific application (e.g., linear algebra functions, Fast-Fourier transform, parallel I/O, etc.).

Each functional HPC system requires this long chain of components to work together, meeting two basic requirements: it needs to be faster than previous systems, and it needs to allow some level of abstraction between the application implementation and the underlying hardware. In other words, it is crucial for scientists that the adoption of a new computing system brings performance improvement and requires little to no adaptation of the codes implementing the models driving the scientific simulations.

NVIDIA has recently launched a new Arm-based CPU called Grace. Its incarnation for HPC foresees a single socket with 144

¹<https://www.nextplatform.com/2022/01/04/inside-amazons-graviton3-arm-server-processor/>

cores (NVIDIA Grace CPU Superchip) and an additional configuration with 72 Arm cores tightly coupled with an NVIDIA Hopper GPU (NVIDIA Grace Hopper Superchip). The cores are Arm Neoverse V2 cores.

As part of the procurement of MareNostrum 5, the Barcelona Supercomputing Center will host a cluster powered by several NVIDIA Grace CPU Superchip CPUs. Thus, this work is a preliminary evaluation performed on engineering samples made available by NVIDIA to address the following research questions: *i)* How difficult is it for a scientist to transition to a Grace-based system? *ii)* How does the NVIDIA Grace CPU behave with complex scientific codes? *iii)* How does the NVIDIA Grace CPU compare to other HPC systems?

We addressed these questions by compiling and running five HPC applications (Alya, OpenFOAM, NEMO, LAMMPS, and PhysCell), studying their scalability within a single node and multiple nodes of Grace CPU Superchip and NVIDIA Grace Hopper Superchip SoCs. To limit variables and emulate the approach of a scientist wanting to quickly adapt to the new system, we focused on a CPU-only study.

Since this is an evaluation of a new HPC system, we are not aware of any related work at a comparable level of complexity. Works on different supercomputers that inspired our work include [11, 18]. Still, there were evaluation efforts targeting previous Arm-based systems that also paved the way of our work. Brank et al. in [2] presented a study of three different Arm-based high-end CPUs (by Huawei, Fujitsu and Marvell) using WaLBerla, NEST and miniFE. Simkov et al. in [15] provided a study of fairly complex workflows including HPCC, NWChem, OpenFOAM, GROMACS, AI Benchmark Alpha, and Enzo on Amazon Graviton 3/2, Fujitsu A64FX, Ampere Altra, and Thunder-X2. However, this study was more oriented to the monitoring tool XDMoD and included GPUs accelerated platforms. A monographic study of Nek5000/RS on A64FX is provided by Tsuji et al. in [17]. Brank et al. in [3] focused more on the analysis of the auto-vectorization capabilities of the compilers for leveraging the Scalable Vector Extension of Arm using A64FX as benchmark platform.

The remaining part of the document is structured as follows: Section 2 introduces the Grace architecture and the clusters used for this study; Section 3 reports the evaluation methodology as well as the scientific applications used for this preliminary evaluation of the Grace CPU in an HPC environment; In Section 4 we describe the system software used in our study, and report its readiness and maturity level for HPC workloads. Section 5 summarizes the measurements and the scalability study performed on each scientific application; Section 6 closes the paper with final remarks and comments about future work.

2 HARDWARE ENVIRONMENT

This paper aims to evaluate the readiness and performance of NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip. For that, we will use two different clusters: *NVIDIA Cluster* and *MareNostrum*. Within these clusters, we found three different platforms: Intel Skylake, NVIDIA Grace CPU Superchip, and NVIDIA Grace Hopper Superchip.

Table 1: Hardware for MareNostrum and NVIDIA Cluster

	MareNostrum	NVIDIA Cluster	
		GraceHopper	GraceGrace
Cluster architecture			
Architecture	x86_64	Armv9	Armv9
Micro-architecture	Sykylake-X	Neoverse V2	Neoverse V2
Cores per socket	24	72	144
Sockets per node	2	1	1
Frequency [MHz]	2100	>=3200	>=3200
Full node floating-point performance			
Vector ISA	AVX512	SVE	SVE
Peak performance [Flop/cycle]	1536	1152	2304
Peak performance [GFlop/s]	3225.6	3801.6	7603.2
Full node main memory			
Number of memory channels	6	8	16
Size [GB]	96	480	480
Technology	DDR4-2666	LPDDR5	LPDDR5
Peak bandwidth [GB/s]	256	600	1200
Operative System			
OS distribution	SUSE Ent. Server 12 SP2	Ubuntu 22.04.2 LTS	
Kernel version	4.4.120-92.70-default	6.2.0-1009-nvidia-64k	

For reference, Figure 1 shows a schematic view of the node topologies in each cluster, and Table 1 summarizes the hardware and system configurations of the clusters under study. In the following sections, we describe in detail the setup of each platform.

2.1 NVIDIA Early Access System

To prepare for the general availability of the NVIDIA Superchip family of products in mid-summer 2023, NVIDIA has deployed a private mini-cluster reserved for selected customers and developers in Early Access mode. The purpose of this mini-cluster was to facilitate code enablement and the early evaluation of the NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip.

This resource, called *NVIDIA Cluster* in the rest of the paper, comprised a few interconnected NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip units, linked by NVIDIA Infiniband NDR400, all connected to a single Infiniband NDR switch. Storage for home directories was provisioned via NFS, while a scratch area of several terabytes was provided via DDN Lustre.

The login node was an NVIDIA Grace CPU Superchip, exporting additional mount points to all compute nodes to provide a basic software environment (compilers, libraries, and system software) deployed on demand based on users’ needs. Slurm was used to orchestrate job execution.

The system’s NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip nodes were Early Production units. These units are functional parts but only partially reflect the exact final product that will be available to the mass market. For example, the CPU frequency, as well as STREAM peak bandwidth, was below what has been measured internally by NVIDIA on Production Units.

2.2 MareNostrum 4

The MareNostrum is a Tier-0 supercomputer with 3456 x86 compute nodes. Each node is equipped with two sockets of Intel Xeon Platinum 8160 CPU with 24 cores, each running at 2.10 GHz, adding

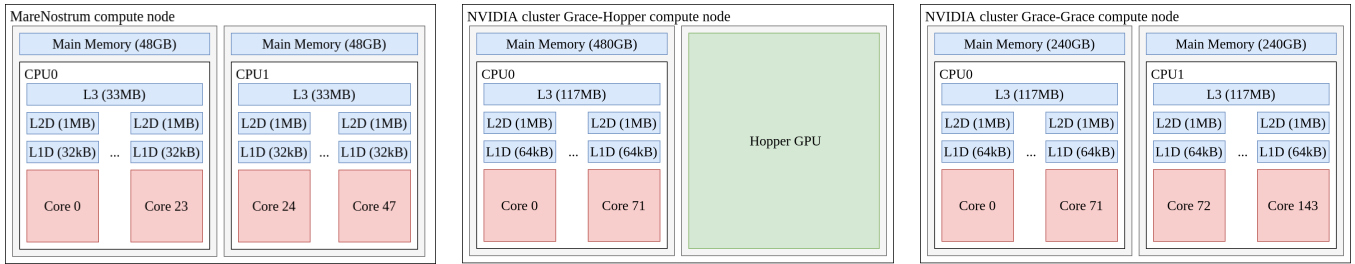


Figure 1: Node topology of the clusters under study.

up to a total of 48 cores per node. The nodes of the system are interconnected using an Intel Omni-Path high-performance network and run the SuSE Linux Enterprise Server as an operating system.

3 METHODOLOGY

The methodology we follow tries to mimic the steps a scientist should perform to run a scientific application on a new cluster. First, we analyzed which compilers and libraries are available: in the case of the NVIDIA Grace, we can experiment with the GNU Compiler Suite, the NVIDIA HPC Compiler, the Arm HPC Compiler, and the LLVM Compiler. Concerning libraries, we used either Arm Performance Libraries released by Arm Ltd or those provided natively by NVIDIA HPC SDK. At the time of the evaluation, Arm Performance Libraries version 23.04.1 did not have explicit support for Arm Neoverse V2 microarchitecture but was still capable of using SVE-optimized code. That is expected to improve toward the end of 2023.

To understand the readiness of the system software, we try to compile all the applications with all the combinations of compilers available in each cluster. A detailed list can be found in Table 2. If the compilation is successful, we run a first set of experiments with the different compilers to evaluate their performance differences.

Concerning the executions to evaluate the performance achieved, we use a common approach for almost all applications²: first, we study the scalability within one compute node to provide a core-to-core performance figure; then, we analyze the scalability over several nodes, providing a node-to-node comparison.

Since the NVIDIA Grace will be part of the installation of MareNostrum 5, we considered the previous installation of MareNostrum 4 (introduced in Section 2) as a baseline for our performance comparison both core-to-core and node-to-node.

For all applications, we considered inputs that are relevant for production scientific runs following the philosophy already introduced of mimicking the steps that a domain scientist should perform when migrating his research on a new HPC system.

All runs have been performed only on the CPU, disregarding the GPU functionalities of the NVIDIA Grace Hopper Superchip chip. Since engineering samples of both NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip were available, we evaluated both platforms, focusing on the CPU only.

Our study involved five scientific applications that are widely used in scientific and industrial environments and are frequently used as reference points within the procurement requirements of HPC systems. The selection has been made to cover a broad spectrum of scientific domains, including fluid dynamics, molecular dynamics, life science, and weather forecasting.

In the remaining part of this section, we briefly introduce the characteristics of each application.

3.1 Alya

Alya is a high-performance computational mechanics code developed at the Barcelona Supercomputing Center since 2005. It can solve multi-physics and coupled problems. Some of the physics solved by Alya are incompressible/compressible flows, non-linear solid mechanics, chemistry, particle transport, heat transfer, turbulence modeling, and electrical propagation. The application is written in Fortran and parallelized with the MPI programming model, focusing on enabling state-of-the-art simulations while achieving high performance and efficiency [19].

We use the benchmark Sphere16M that simulates the turbulent flow over a sphere, a finite element problem with 16 million elements. It employs the Vreman turbulence model, convective term using the EMACS scheme and fractional step with Runge-Kutta of order three and conjugate gradient as the algebraic solver.

We run the benchmark for 1000 timesteps and use the time reported by the application. This time corresponds to the time spent in the time steps, excluding initialization and finalization.

3.2 OpenFOAM

OpenFOAM (Open-source Field Operation And Manipulation) [8] is a free, open-source CFD code primarily developed by OpenCFD Ltd. OpenFOAM constitutes a C++ CFD toolbox for customized numerical solvers (over sixty of them) that can perform simulations of basic CFD, combustion, turbulence modeling, electromagnetic, heat transfer, multiphase flow, stress analysis, and even financial mathematics. It has a large user base across most areas of engineering and science, from both commercial and academic organizations. The parallel computing paradigm used by OpenFOAM is based on domain decomposition on top of an MPI parallelization. We use OpenFOAM v2206 for our experiments. In this paper, we run the well-known motorBike⁴ tutorial case with 5.2 million cells distributed with OpenFOAM. This benchmark calculates the steady

²In the case of PhysiCell the study of the inter-node scalability was not possible because the application uses a parallelization paradigm with shared memory with OpenMP

³<https://develop.openfoam.com/Development/openfoam/-/tree/OpenFOAM-v2206/tutorials/incompressible/pisoFoam/LES/motorBike>

flow around a motorcycle and rider. The case uses the `pi soFoil` solver based on the Large Eddy Simulation (LES) model. We configured the run to perform 10 timesteps and measured the time spent to perform the core calculation, excluding initialization, mesh partition, and finalization.

3.3 NEMO

The Nucleus for European Modelling of the Ocean (NEMO) [9, 10] is a modeling framework for research activities and forecasting services in ocean and climate sciences. It comprises several modules with numerical techniques, including ocean, sea ice, biogeochemistry, grid refinement tools, and data assimilation. It can be coupled with other codes simulating other components of the climatic system like the atmosphere and land surface. It was developed by a European consortium to ensure long-term reliability and sustainability. NEMO is written in Fortran and parallelized with MPI.

The ORCA family is a series of global ocean configurations. The NEMO system has five built-in ORCA configurations that differ in the horizontal resolution. For the NEMO runs, we have used the Orca 1-like configurations as input, distributed with the application sources and found under the `tests` subdirectory. We report the execution time of 700 iterations, averaged across three runs.

3.4 LAMMPS

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [16] is a classical molecular dynamics code focusing on materials modeling. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale. It is written mainly in C++ and parallelized with MPI, following a modular structure that allows it to be easily extended or modified. However, simultaneously, each model needs to be individually ported to parallel programming paradigms or accelerators.

In our study, we run the 3d Lennard-Jones melt benchmark input set distributed with the application. We increase the number of atoms in the grid to 256 million and use the `loop time` as reported by the application.

3.5 PhysiCell

PhysiCell is an open-source multiscale multicellular simulation framework written in C++ with minimal dependencies. It was developed as an agent-based modeler where the agents are cells with properties like secretion/uptake values for different substrates, mechanical properties, and growth rates, interacting with the tissue environment and other cells. For this evaluation, we are using PhysiBoss [5] version 2.0.0, which is a fork of PhysiCell v1.9.0 co-developed by BSC and Institut Curie, that includes MaBoSS as a library that expands PhysiCell’s agent-based functionalities with intracellular cell signaling. We added the code changes described in [4] to minimize the scalability problems inherent in the application in order to unveil the architectural effects on the performance.

PhysiCell is parallelized using OpenMP as a shared memory parallelization model. The input case consists of an initial setup of 1 million cells evenly distributed in a rectangular box of size $6 \times 2 \times 2$ mm. This test is suitable for performance evaluations. The present work is based on the intracellular sample project called `physiboss-tnf-model`. We report the total walltime, excluding

initialization, as reported by the application, averaged across five runs.

4 SOFTWARE ENVIRONMENT

4.1 Compilers and libraries

Table 2 summarizes the compilers and libraries available in the different clusters and used for our study. We include the compiler optimization flags that were used to compile all the benchmarks and applications. We omit flags that are application or compiler-specific (e.g., `-fopenmp` and `-qopenmp`).

Table 2: Software for MareNostrum and NVIDIA Cluster

Name	Version	Comments/Flags
MareNostrum		
GNU Compiler	gcc/12.1.0	-Ofast -march=skylake-avx512
Intel Compiler	intel/2021.4	-Ofast -xCORE-AVX512 -mtune=skylake
Math library	mk1/2021.4	Provided by environment modules
MPI library	impi/2018.4	Provided by environment modules
NVIDIA Cluster		
GNU Compiler	gcc/12.3.0	-Ofast -mcpu=native
NVIDIA Compiler	nvhpc/23.9	-O3 -tp=host
Arm Compiler	acfl/23.04.1	-Ofast -mcpu=neoverse-v2
Math library	armp1/23.04.1	Provided by environment modules
MPI library	openmpi/4.1.5-rc2	Provided by environment modules

Flags for MareNostrum were chosen based on the guidelines provided to the users by the support team. Flags for NVIDIA Cluster were chosen based on previous experience in other Arm-based clusters.

In the NVIDIA Cluster, the MPI library is provided via an environment module paired with the NVIDIA HPC compiler and the HPCX software package. We use the environment variables `$OMPI_CC`, `$OMPI_CXX`, and `$OMPI_FC` to enable other compilers.

4.2 Software and application compatibility

Table 3 lists the applications under study and with which compilers we were able to compile them. Each column represents an application, while each row represents a compiler. Rows are grouped per cluster.

Table 3: Software summary for MareNostrum and NVIDIA Cluster

	Alya	OpenFOAM	NEMO	LAMMPS	PhysiCell
MareNostrum					
GNU Compiler	✓	✓	✓	✓	✓
Intel Compiler	✓	✓	✓	✓	✓
NVIDIA Cluster					
GNU Compiler	✓	✓	×	✓	✓
NVIDIA Compiler	✓	✓	✓	✓	✓
Arm Compiler	✓	✓	×	✓	✓

Some of the applications required changes to the source code to compile in NVIDIA Cluster. The first example is Alya, which failed to compile with the NVIDIA compiler due to a compiler bug when accessing members in data structures with the same name as

the instance of the structure itself. In this case, changes to the code were minimal and required a simple renaming of local variables.

In the case of NEMO, the source code used a non-standard intrinsic function, ISNAN, which is only available in certain compiler implementations. After the code modifications, the compilation of NEMO with the Arm compiler succeeds, but the execution yields a segmentation fault. In the case of the NVIDIA compiler, there is a similar issue that we tracked down to the allocation of data structures during the initialization of the application. This issue is not present when compiling with the GNU compiler. In the case of the Arm compiler in the NVIDIA Cluster, the compilation produces a binary but the execution fails with an address-not-mapped error. This error is likely related to incompatibilities between system libraries (e.g., MPI and mathematical libraries).

Lastly, the source code of PhysiCell includes the initialization of static member variables inside C++ classes, which is considered bad practice. This practice is supported by the GNU compiler and Intel compiler in MareNostrum but not so with the NVIDIA compiler in NVIDIA Cluster. The initialization of the variables was moved outside of the class definition.

4.3 Bindings

We always pin threads/processes to a single core regardless of their programming model (OpenMP or MPI). With OpenMPI, we achieve this using the `--bind-to core` flag. With the Intel MPI library, we use the `$I_MPI_PIN_DOMAIN=core` environment variable. For OpenMP pinning, we use the `$OMP_PLACES=cores` environment variable.

For MPI applications, we use two mapping policies depending on the topology of the node. We use a close binding policy for the NVIDIA Grace Hopper Superchip nodes in the NVIDIA Cluster, which are single-socket. This policy maps contiguous processes sequentially to the cores in the CPU. In OpenMPI, this policy is enabled with `--map-by core`, while in Intel MPI, it is enabled with `$I_MPI_ORDER=compact`. We use a spread binding policy for clusters where nodes are dual-socket, such as the NVIDIA Grace CPU Superchip nodes in the NVIDIA Cluster and the nodes in MareNostrum. This policy maps contiguous processes to cores in different sockets of the node. In OpenMPI, this policy is enabled with `--map-by socket`, while in Intel MPI is enabled with `$I_MPI_ORDER=scatter`.

For Physicell, which is parallelized with OpenMP, we always use a close binding policy since the input set we use fits within the memory capacity of one socket, and threads benefit from close NUMA accesses.

5 EVALUATION AND RESULTS

In this section we present the performance results of each application under study. For each application, we have evaluated it in three different scenarios: in NVIDIA Cluster with NVIDIA Grace CPU Superchip nodes and with NVIDIA Grace Hopper Superchip nodes and in MareNostrum.

In Table 3, we have enumerated all the compilers utilized in our study. In the majority of cases, there are negligible performance differences between compilers, with performance variations consistently below 10%. For clarity, we present results in the plots using a

single compiler per cluster, specifically the one that attains the best performance for each application. The compiler used is explicitly indicated in the label of each series in every plot.

For each application, we conduct three types of comparisons. The first is a core-to-core comparison, where we plot the execution time while scaling from 1 core to 144 cores for the three platforms considered in the study. It is important to note that 144 cores correspond to a full NVIDIA Grace CPU Superchip node, 2 NVIDIA Grace Hopper Superchip nodes, and 3 MareNostrum nodes. The second type of comparison is a node-to-node comparison, where we use 1 and 2 nodes of each platform and plot the execution time. In this study, we demonstrate the scalability of the Grace nodes in comparison to the same run on MareNostrum. Finally, we present the speedup achieved by the Grace nodes in the core-to-core comparison.

In this section, for brevity, we refer to NVIDIA Grace CPU Superchip as GraceGrace and NVIDIA Grace Hopper Superchip as GraceHopper.

5.1 Alya

Figure 2a illustrates the elapsed time for 1000 time steps of Alya across the three platforms under consideration in this study. In MareNostrum, the input case utilized for the evaluation did not fit in memory when using fewer than 32 cores. Consequently, we present data only for configurations with 32 cores and above in MareNostrum.

It is evident from the core-to-core comparison that Grace nodes outperform MareNostrum nodes in all cases, and the scalability exhibits a similar trend. When comparing GraceGrace and GraceHopper nodes, no significant performance differences are observed between them, except for configurations with more than 32 cores, where the performance of GraceHopper is slightly inferior to GraceGrace. This suggests that the input benefits from higher memory bandwidth when running with more than 32 cores. This can be attributed to the fact that the dominant phase in a time step is the matrix assembly, which is a compute-bound computation. Figure 2b compares the execution time for 1000 time steps of Alya using one and two full nodes in each cluster. The labels adjacent to each point indicate the speedup in comparison to MareNostrum. On a node-by-node basis, we observe a 2.0× speedup on GraceHopper and a speedup ranging between 4.3× and 4.1× on GraceGrace, relative to MareNostrum, for both one and two nodes.

In Figure 3, we show the speedup achieved by Grace cores when running Alya in comparison to the same number of cores in MareNostrum. On the GraceGrace platform, we observe a speedup ranging between 1.81× and 1.67× with respect to a run in MareNostrum with the same number of cores. Meanwhile, on the GraceHopper platform, the obtained speedup ranges between 1.57× and 1.31×.

5.2 OpenFOAM

Figure 4a displays the elapsed time when running OpenFOAM on the three different platforms under consideration, when increasing the number of cores within a single node. Notably, both GraceGrace and GraceHopper exhibit faster performance than MareNostrum in all cases. The same behavior is observed in each cluster: the curve

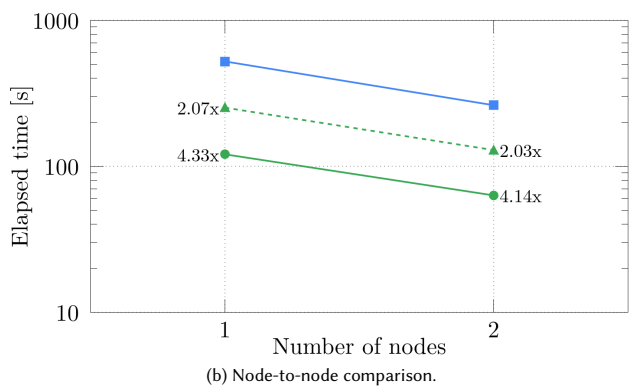
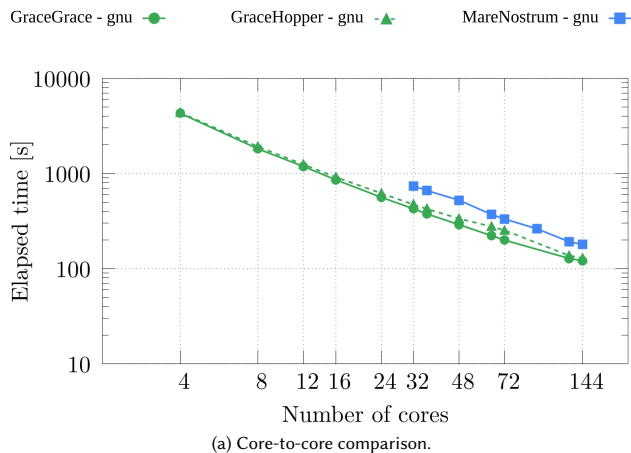


Figure 2: Alya scalability.

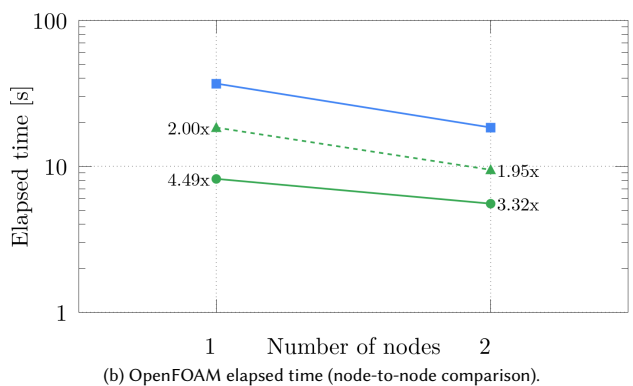
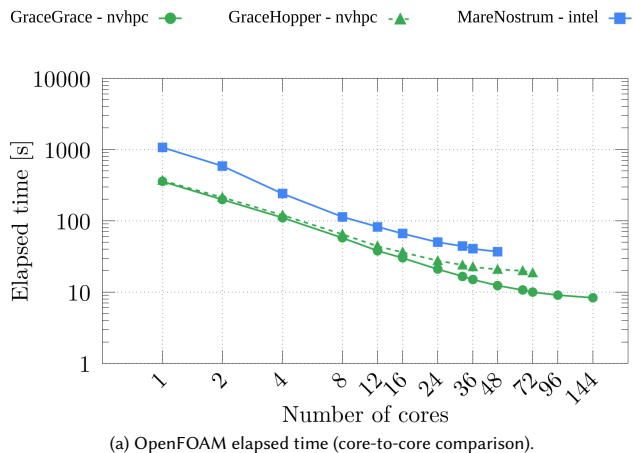


Figure 4: OpenFOAM scalability.

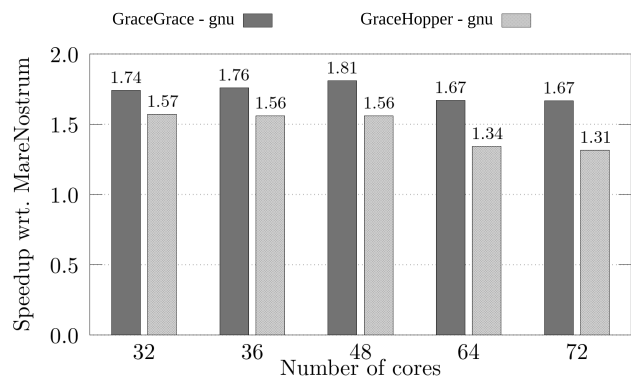


Figure 3: Alya speedup with respect to MareNostrum.

flattens as the number of cores increases, indicating saturation as the cores within the node are filled. This behavior is expected, as an increasing number of processes compete for memory bandwidth. This observation aligns with findings from prior works [6], which established that the region under study in OpenFOAM is memory bandwidth bound.

Additionally, it is noteworthy that in GraceHopper, the curve flattens after MareNostrum. This behavior can be attributed to the fact that the NVIDIA Superchip nodes have more memory channels than MareNostrum, resulting in increased bandwidth to the cores and an overall performance improvement in OpenFOAM. Furthermore, the curve flattens even later in GraceGrace, indicating even better performance and the advantageous utilization of having twice the number of memory channels compared to GraceHopper nodes.

In Figure 5, we plot the speedup achieved in GraceHopper and GraceGrace with respect to MareNostrum in a core-to-core comparison. GraceGrace achieves a speedup ranging between 1.97 \times and 3.01 \times compared to a run in MareNostrum with the same number of cores. Meanwhile, GraceHopper achieves a speedup ranging between 1.75 \times and 2.92 \times compared to MareNostrum with the same number of cores.

The speedup obtained in a node-to-node comparison with respect to MareNostrum, when running on GraceGrace and GraceHopper cores, is depicted in Figure 4b. We observe a 2.00 \times speedup on GraceHopper and a 4.49 \times speedup on GraceGrace when running in one full node compared to MareNostrum. The speedup obtained in two nodes is lower but remains above 3 for GraceGrace nodes and almost 2 for GraceHopper nodes. The observed difference in

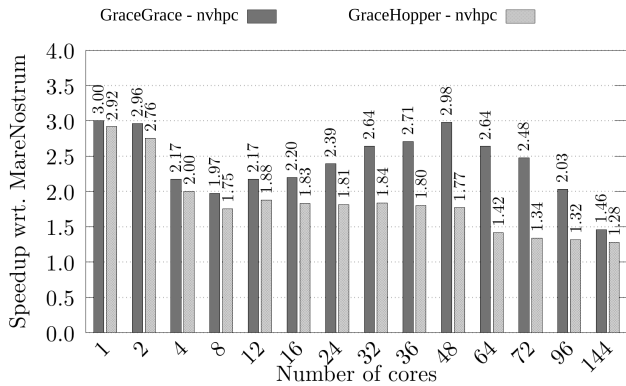


Figure 5: OpenFOAM speedup with respect to MareNostrum.

scalability when transitioning from one node to two nodes could indeed be attributed to the limitations of the application parallelization. Further investigation with a more detailed analysis should be performed to better understand and characterize the behavior and performance patterns in different node configurations.

5.3 NEMO

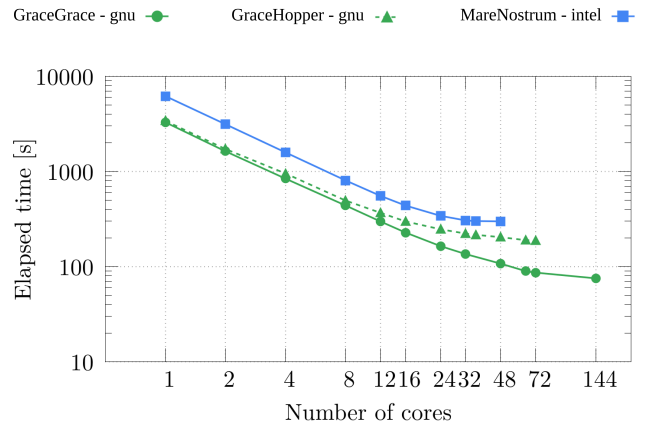
In Figure 6a, we show the execution time of Nemo when scaling inside a single node of the different platforms. On this core-to-core test, we can see that GraceHopper and GraceGrace always perform better than MareNostrum. However, we observe that the scalability of Nemo is far from ideal in all the cases, and we know that Nemo is a memory bandwidth-bound code. We can see that all the lines of the plot flatten at some point, showing when the memory bandwidth starts to saturate and becomes the bottleneck. It is interesting to observe that the GraceGrace line flattens much later than the other two, showing that it is able to take advantage of the higher number of memory channels.

In Figure 6b, we can see the execution time of Nemo in one and two nodes of each platform. On a node-by-node basis, we measure 1.59x speedup on GraceHopper with respect to MareNostrum and 3.97x speedup on GraceGrace in one node and very similar speedup in two nodes. The scaling from one node to two nodes in all the cases is quite good, confirming that the scaling issue observed when scaling inside the node was due to the memory bandwidth limitation.

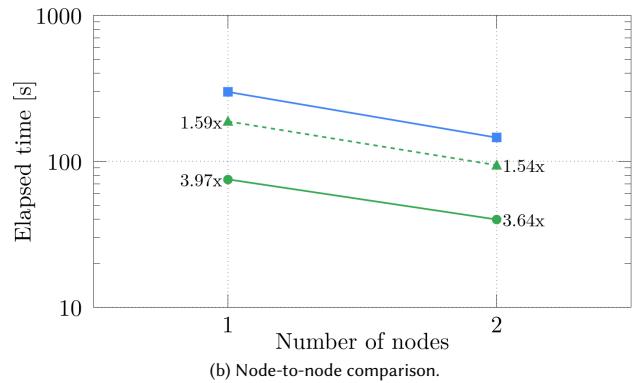
To finish the Nemo evaluation, we plot the speedup achieved by GraceHopper and GraceGrace nodes when comparing it with the same number of nodes on MareNostrum in Figure 7. Here, we observe that GraceGrace takes advantage of the higher memory bandwidth and achieves a 2.78x speedup with respect to MareNostrum with 48 cores.

5.4 LAMMPS

In Figure 8a, we can see the elapsed time when running LAMMPS on the three platforms. In this plot, we can compare the performance achieved when increasing the number of cores for the simulation. We observe very good scalability of LAMMPS across all clusters and



(a) Core-to-core comparison.



(b) Node-to-node comparison.

Figure 6: NEMO scalability

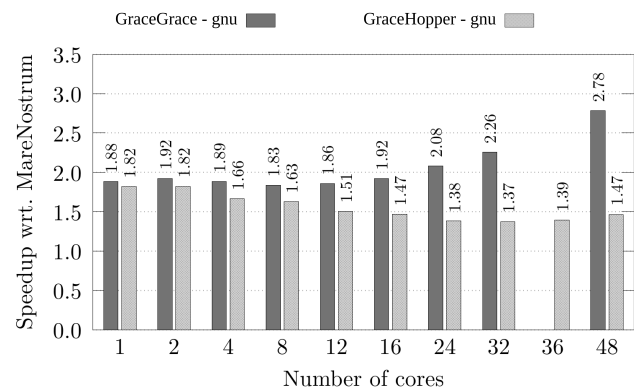


Figure 7: NEMO speedup with respect to MareNostrum.

almost no difference between running in GraceGrace or GraceHopper. Both GraceHopper and GraceGrace outperforms MareNostrum if we compare on a core to core basis. This means that LAMMPS does not benefit from a higher memory bandwidth, since it is a core-bound code.

In Figure 8b, we show the execution time of LAMMPS when using 1 and 2 full nodes of each platform. The labels shown in the plot

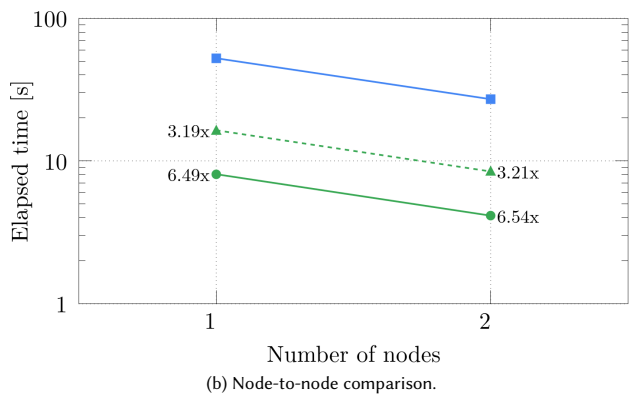
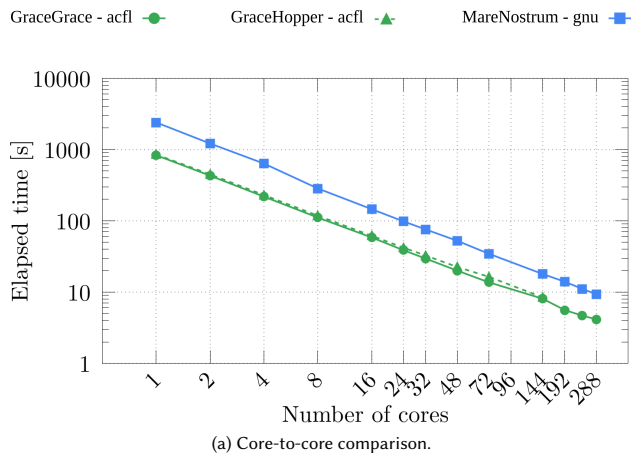


Figure 8: LAMMPS scalability.

display the speedup obtained by GraceGrace and GraceHopper with respect to the same run in MareNostrum. On a node-by-node basis, we measure 3.19 \times speedup on GraceHopper and 6.49 \times speedup on GraceGrace, with respect to MareNostrum. This plot also shows that LAMMPS, with the input case we are using, is able to exploit the amount of parallelism offered without suffering from parallelization inefficiencies, as it achieves an almost perfect strong scaling.

For LAMMPS, we also plot the speedup achieved with GraceHopper and GraceGrace cores compared with MareNostrum cores in Figure 9. We can see that both in GraceHopper and GraceGrace, the performance with respect to MareNostrum is quite sustained between 2.9 \times and 2.1 \times .

5.5 PhysiCell

As explained in the previous section, PhysiCell is parallelized with OpenMP, so it can only benefit from shared memory parallelism. Therefore, it is only executed with up to 72 threads in the GraceHopper node, 144 threads in the GraceGrace node, and 48 threads in MareNostrum.

In Figure 10, we plot the execution time used to run in the different platforms with different number of cores. We observe that GraceGrace and GraceHopper are much faster than MareNostrum and that the difference between the two platforms is not relevant

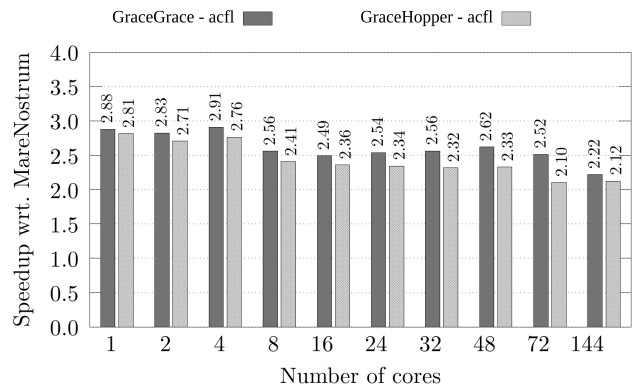


Figure 9: LAMMPS speedup with respect to MareNostrum.

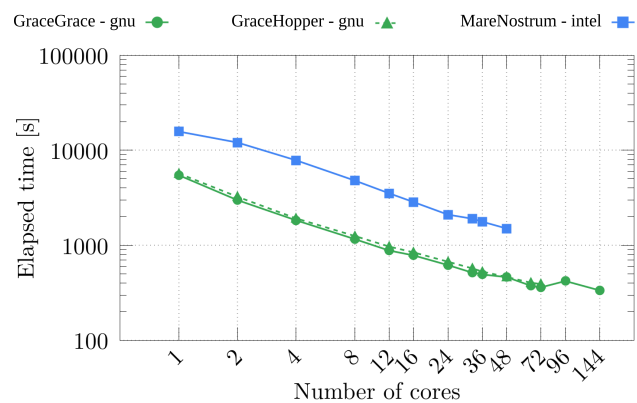


Figure 10: PhysiCell core-to-core comparison.

in this case. All the platforms show a very bad scalability that we know from previous works that is inherent to the application due to some code regions not being parallelized.

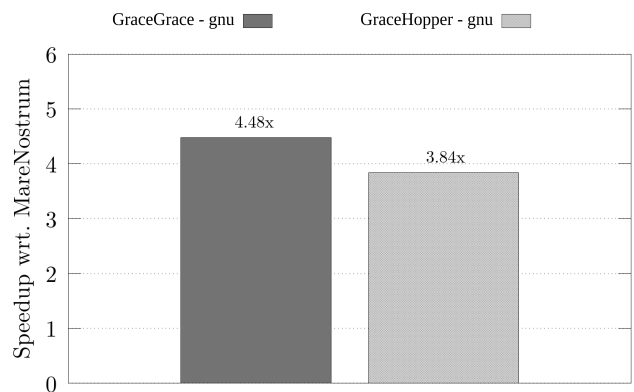


Figure 11: PhysiCell node-to-node comparison.

On a node-by-node basis, we measure 3.84 \times speedup on GraceHopper and 4.48 \times speedup on GraceGrace, with respect to MareNostrum, as seen in Figure 11.

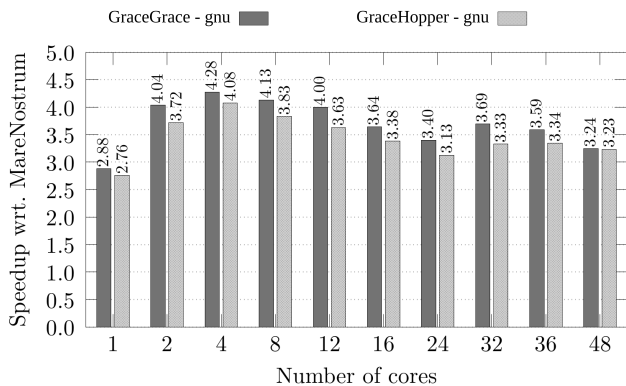


Figure 12: PhysiCell speedup with respect to MareNostrum.

Figure 12 shows the speedup obtained on a core-to-core comparison between the NVIDIA platforms and MareNostrum. We can see that GraceHopper achieves between 2.76 \times and 4.08 \times speedup with respect to a run in MareNostrum with the same number of cores, and GraceGrace obtains a speedup between 4.28 \times and 2.88 \times .

Lastly, we present in Figure 13 a comparison of the elapsed time of PhysiCell in the NVIDIA Cluster depending on the thread bindings. We expected that the spread binding would improve performance since it allows a balanced distribution of threads and, in principle, an even use of the memory channels. However, our measurements show that the close binding exhibits better performance (i.e., lower execution time). This is due to the fact that memory allocations in PhysiCell are not NUMA aware by default, which means that all threads access the same physical memory regardless of their placement. Since half of the threads must access data from a different CPU with the spread binding, their perceived memory bandwidth decreases, and thus, they take longer to execute. This issue is not inherent to the hardware, but it could be present in other scientific applications and become a performance bottleneck if not studied on a case-by-case basis.

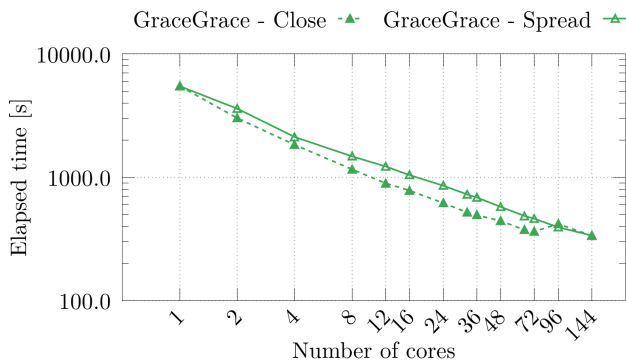


Figure 13: PhysiCell time depending on thread bindings.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we conducted an early evaluation of a small cluster equipped with the first NVIDIA Superchips family. Access was

granted between July and September 2023, and the data were collected on engineering sample silicon within a short timeframe. The evaluated system served as a test platform scheduled for retirement by the end of 2023.

The purpose of the evaluation was to simulate the transition for a scientist moving from a current HPC cluster to an NVIDIA Grace-based one. We analyzed both the performance and system software maturity of the Arm-based CPU within the NVIDIA Superchips family, comparing them with the Intel-based cluster MareNostrum. For the evaluation, we considered five HPC applications relevant to different scientific areas: Alya, OpenFOAM, NEMO, LAMMPS, and PhysiCell.

Regarding performance, all applications exhibited improvements in both core-to-core and node-to-node performance. A portion of the improvement is due to a higher clock frequency in the NVIDIA Cluster compared to MareNostrum (3.2 GHz and 2.1 GHz, respectively). The difference in frequency is most impactful in codes that are compute-bound, like Alya in Section 5.1. The performance enhancement was particularly pronounced in applications limited by memory bandwidth, such as OpenFOAM, NEMO, and PhysiCell. The first reason for this is that the memory technology in the NVIDIA Cluster is newer than in MareNostrum and has a higher theoretical bandwidth peak. Secondly, the CPUs in the NVIDIA Cluster have more memory channels, allowing a higher concurrency of memory accesses. We observed that the high number of cores proved beneficial for shared memory applications. Specifically, we successfully executed a functional run of an OpenMP application (PhysiCell), utilizing all 144 cores of the NVIDIA Grace CPU Superchip. In the case of PhysiCell, scalability limitations were noted, which were inherent to the application rather than the CPU design. Future steps may involve improving the shared memory parallelization of the application or testing others without such scalability limitations.

Regarding system software, we found that compilers and scientific libraries are well-prepared to support complex HPC workloads. The only minor issue identified was related to the binaries of NEMO compiled with Arm and NVIDIA compilers, rendering them inexecutable. NVIDIA is aware of these issues and plans to address them in a future release of NVIDIA HPC SDK. The performance of binaries generated with all compilers was similar across all applications. We imagine that the next steps of the Arm HPC community and the Arm providers will be to incorporate the optimizations for the Grace CPU microarchitecture into dedicated numerical libraries. As a fun fact, we observed that, after decades of x86 architecture supremacy, some HPC application developers adopted proprietary coding styles that require minor adjustments when transitioning to a new architecture.

In a final general note, we draw on our extensive experience evaluating Arm-based systems since the early days of Arm entering HPC [12, 13]. The NVIDIA Grace CPU Superchip marks the first instance where both hardware and software feel mature and performance reaches HPC levels from day one. Comparatively, in the case of Fujitsu A64FX, the system software proved lacking, and the experience of a "standard" user would be severely compromised [1]. On the Marvell ThunderX2, performance was limited [11].

For future work, we aim to expand our study to a production system, including more scientific applications, and conduct deeper analyses of the machine's efficiency.

ACKNOWLEDGMENTS

This research has received funding from the European Commission via the Horizon Europe research and innovation funding programme, under grant agreement 101092993 (RISER), the EuroHPC H2020 Industrial Leadership programme grant agreement 956416 (exaFOAM), and the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 (European Processor Initiative) and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland. The EPI-SGA2 project, PCI2022-132935 is also co-funded by MCIN/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR.

REFERENCES

- [1] Fabio Banchelli, Kilian Peiro, Guillem Ramirez-Gargallo, Joan Vinyals, David Vicente, Marta Garcia-Gasulla, and Filippo Mantovani. 2021. Cluster of emerging technology: evaluation of a production HPC system based on A64FX. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 741–750.
- [2] Bine Brank, Stepan Nassyr, Fatemeh Pouyan, and Dirk Pleiter. 2020. Porting Applications to Arm-based Processors. *IEEE Computer Society*, 559–566. <https://doi.org/10.1109/CLUSTER49012.2020.00079>
- [3] Bine Brank and Dirk Pleiter. 2022. Assessing the State of Autovectorization Support based on SVE. *IEEE Computer Society*, 556–562. <https://doi.org/10.1109/CLUSTER51413.2022.00073>
- [4] Marc Clascà, Marta Garcia-Gasulla, Arnau Montagud, José Carbonell Caballero, and Alfonso Valencia. 2023. Lessons Learned from a Performance Analysis and Optimization of a Multiscale Cellular Simulation. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. ACM. <https://doi.org/10.1145/3592979.3593403>
- [5] Miguel Ponce de Leon, Arnau Montagud, Vincent Noel, Gerard Pradas, Anika Meert, Emmanuel Barillot, Laurence Calzone, and Alfonso Valencia. 2022. PhysiBoSS 2.0: a sustainable integration of stochastic Boolean and agent-based modelling frameworks. (Jan. 2022). <https://doi.org/10.1101/2022.01.06.468363>
- [6] Marta Garcia-Gasulla, Fabio Banchelli, Kilian Peiro, Guillem Ramirez-Gargallo, Guillaume Houzeaux, Ismail Ben Hassan Saïdi, Christian Tenaud, Ivan Spisso, and Filippo Mantovani. 2020. A Generic Performance Analysis Technique Applied to Different CFD Methods for HPC. *International Journal of Computational Fluid Dynamics* 34, 7-8 (2020), 508–528. <https://doi.org/10.1080/10618562.2020.1778168> arXiv:<https://doi.org/10.1080/10618562.2020.1778168>
- [7] Simon David Hammond et al. 2018. *The Astra Supercomputer*. Technical Report. Sandia National Lab. <https://www.osti.gov/servlets/purl/1574565>
- [8] Hrvoje Jasak. 2009. OpenFOAM: Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering* 1, 2 (2009), 89–94. <https://doi.org/10.2478/IJNAOE-2013-0011>
- [9] Gurvan Madec, Mike Bell, Adam Blaker, Clément Bricaud, Diego Bruciaferri, Miguel Castrillo, Daley Calvert, Jérôme Chanut, Emanuela Clementi, Andrew Coward, Italo Epicoco, Christian Éthé, Jonas Ganderton, James Harle, Katherine Hutchinson, Doroteaciro Iovino, Dan Lea, Tomas Lovato, Matt Martin, Nicolas Martin, Francesca Mele, Diana Martins, Sébastien Masson, Pierre Mathiot, Francesca Mele, Silvia Mocavero, Simon Müller, A.J. George Nurser, Stella Paronuzzi, Mathieu Peltier, Renaud Person, Clément Rousset, Stefanie Rynders, Guillaume Samson, Sibylle Téchené, Martin Vancoppenolle, and Chris Wilson. 2023. NEMO Ocean Engine Reference Manual. <https://doi.org/10.5281/zenodo.8167700>
- [10] Gurvan Madec, Romain Bourdallé-Badie, Pierre-Antoine Bouttier, Clément Bricaud, Diego Bruciaferri, Daley Calvert, Jérôme Chanut, Emanuela Clementi, Andrew Coward, Damiano Delrosso, et al. 2017. NEMO ocean engine. (2017).
- [11] Filippo Mantovani, Marta Garcia-Gasulla, José Gracia, Esteban Stafford, Fabio Banchelli, Marc Josep-Fabrego, Joel Criado-Ledesma, and Mathias Nachtmann. 2020. Performance and energy consumption of HPC workloads on a cluster based on Arm ThunderX2 CPU. *Future generation computer systems* 112 (2020), 800–818.
- [12] Nikola Rajovic, Alejandro Rico, Filippo Mantovani, Daniel Ruiz, Josep Oriol Vilarrubi, Constantino Gomez, Luna Backes, Diego Nieto, Harald Servat, Xavier Martorell, et al. 2016. The Mont-Blanc prototype: an alternative approach for HPC systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 444–455.
- [13] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. 2014. Tibidabo: Making the case for an ARM-based HPC system. *Future Generation Computer Systems* 36 (2014), 322–334.
- [14] Mitsuhsisa Sato, Yutaka Ishikawa, Hirofumi Tomita, Yuetsu Kodama, Tetsuya Odajima, Miwako Tsuji, Hisashi Yashiro, Masaki Aoki, Naoyuki Shida, Ikuo Miyoshi, et al. 2020. Co-Design for A64FX Manycore Processor and "Fugaku". In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.
- [15] Nikolay A. Simakov, Robert L. Deleon, Joseph P. White, Matthew D. Jones, Thomas R. Furlani, Eva Siegmann, and Robert J. Harrison. 2023. Are we ready for broader adoption of ARM in the HPC community: Performance and Energy Efficiency Analysis of Benchmarks and Applications Executed on High-End ARM Systems. In *Proceedings of the HPC Asia 2023 Workshops (HPC Asia '23 Workshops)*. Association for Computing Machinery, New York, NY, USA, 78–86. <https://doi.org/10.1145/3581576.3581618>
- [16] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comput. Phys. Comm.* 271 (2022), 108171. <https://doi.org/10.1016/j.cpc.2021.108171>
- [17] Miwako Tsuji, Misun Min, Stefan Kerkemeier, Paul Fischer, Elia Merzari, and Mitsuhsisa Sato. 2022. Performance tuning of the Helmholtz matrix-vector product kernel in the computational fluid dynamics solver Nek5000/RS for the A64FX processor. In *International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsia 2022 Workshop)*. Association for Computing Machinery, New York, NY, USA, 49–59. <https://doi.org/10.1145/3503470.3503476>
- [18] Sudharshan S Vazhkudai, Bronis R De Supinski, Arthur S Bland, Al Geist, James Sexton, Jim Kahle, Christopher J Zimmer, Scott Atchley, Sarp Oral, Don E Maxwell, et al. 2018. The design, deployment, and evaluation of the CORAL pre-exascale systems. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 661–672.
- [19] Mariano Vázquez, Guillaume Houzeaux, Seid Koric, Antoni Artigues, Jazmin Aguado-Sierra, Ruth Aris, Daniel Mira, Hadrien Calmet, Fernando Cucchietti, Herbert Owen, et al. 2016. Alya: Multiphysics engineering simulation toward exascale. *Journal of Computational Science* 14 (2016), 15–27.