Cluster of emerging technology: evaluation of a production HPC system based on A64FX

Fabio Banchelli*, Kilian Peiro*, Guillem Ramirez-Gargallo*, Joan Vinyals*,
David Vicente*, Marta Garcia-Gasulla* and Filippo Mantovani*

* *Barcelona Supercomputing Center*
Plaça Eusebi Güell, 1-3 08034 Barcelona (Spain)

***Abstract*—Clusters of emerging technologies are appearing with more and more frequency in HPC. After years of skepticism, data-centers are adopting them as production systems thanks to several geopolitical and technological factors. The most honorable example is the Fugaku supercomputer, powered by the latest Fujitsu A64FX CPU. Which is the behavior of mature HPC codes on such emerging technology clusters? Which performance will obtain scientists when running their HPC applications "as is" on these clusters? This paper presents the evaluation of CTE-Arm, a Fugaku-like system, including both fine-tuned micro-benchmarks and five scientific applications run without prior fine-tuning: Alya, NEMO, Gromacs, OpenIFS, and WRF. Results show that while micro-architectural benchmarks show performance as expected, the performance obtained running HPC applications not tuned for a specific architecture are between $2\times$ and $4\times$ slower compared with a standard Intel-based HPC system. Therefore further effort is needed to improve tools (e.g., compilers) and system software (e.g., MPI libraries) to ease applications deployment and improve their performance.**

***Index Terms*—A64FX, Fujitsu, Arm, Benchmarks, Cluster evaluation, HPC applications**

I. Introduction and related work

After years of bring-up, the Arm-based systems reached the ranking of the most powerful supercomputer in the world. The Fugaku supercomputer powered by the Fujitsu A64FX CPU is currently leading the lists of most powerful supercomputers both in the LINPACK and the HPCG rankings.

Beyond the pure "muscle" performance reported with synthetic benchmarks, scientists and data-centers are interested in understanding the maturity of Arm-based systems as production machines. This means they are interested not only in their peak performance but also in the maturity of the system software, the ease of deployment and maintenance, and the performance of complex scientific codes.

This paper evaluates CTE-Arm, an emerging technology cluster deployed as a production system at the Barcelona Supercomputing Center. CTE-Arm is a smaller version of the Fugaku supercomputer, composed of 192 compute nodes powered by the Fujitsu A64FX CPU. Our basic idea is to follow an evaluation process from simple codes evaluating basic architectural features (floating-point, memory, and network performance) up to synthetic benchmarks (LINPACK and HPCG) to finish with real scientific applications. Concerning the HPC applications, our evaluation aims to report the experience and the results that can obtain an average HPC user who accesses for the first time a Fugaku-like system and tries to run complex codes without any prior tuning. We consider this as the most relevant contribution of our paper which nicely complements the effort of other evaluations of Arm-based systems such as [1]–[4]. As a reference, we compare all our results with MareNostrum 4, an Intel Skylake-based system.

The paper is organized as follows: in Section II, we introduce the basic concept of our Fugaku-like system and its configuration; Section III and Section IV shortly reports about the performance of micro-architectural benchmarks and synthetic HPC benchmarks; in Section V we show performance and scalability results obtained running five HPC complex applications on a CTE-Arm and MareNostrum 4; our paper ends with conclusions remarks in Section VI.

II. System configuration

CTE-Arm has the same architecture as the Fugaku supercomputer [5] and houses 192 nodes. Each node contains a single A64FX CPU with 48 cores. This CPU is the first commercial Armv8 CPU to implement the SVE vector extension. The cores are distributed across Core Memory Groups (CMG) in groups of 12 cores. Each CMG connects to an HBM module, and communication across CMGs is accomplished through a ring bus. Compute nodes in CTE-Arm are connected via the TofuD interconnect in a six-dimensional torus topology. The job scheduler of the cluster is aware of the network topology and can allocate nodes for user jobs to exploit proximity and reduce the latency of messages.

The reference runs of this paper have been performed on the MareNostrum 4 supercomputer. MareNostrum 4 is the flagship supercomputer at Barcelona Supercomputing Center. It has 3456 compute nodes based on Intel's x86 architecture. Each node includes two Skylake CPUs, each one with 24 cores and six DDR4 memory channels. Compute nodes in MareNostrum 4 are connected via Intel's OmniPath interconnect.

Table I summarizes the characteristics of the system. We refer to the official Microarchitecture Manual written by Fujitsu [1] to fill in the architectural specifications of CTE-Arm.

[1] https://raw.githubusercontent.com/fujitsu/A64FX/master/doc/A64FX_Microarchitecture_Manual_en_1.0.pdf

© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI 10.1109/Cluster48925.2021.00110

For better clarity, measurements of CTE-Arm and MareNostrum 4 are depicted in red and blue, respectively, throughout this document.

TABLE I
HARDWARE CONFIGURATION OF CTE-ARM AND MARENOSTRUM 4

	CTE-Arm	MareNostrum 4
System integrator	Fujitsu	Lenovo
Core architecture	Armv8	Intel x86
SIMD extensions	NEON, SVE	AVX512
CPU name	A64FX	Intel Xeon Platinum 8160
Frequency [GHz]	2.20	2.10
Turbo Boost	Disabled	Disabled
Simultaneous Multi-Threading	Disabled	Disabled
Sockets / node	1	2
Core / node	48	48
DP Peak / core [GFlop/s]	70.40	67.20
DP Peak / node [GFlop/s]	3379.20	3225.60
L1 cache size / core	64 kB	32 kB
L2 cache size / core	32 MB	1 MB
L3 cache size / core	-	33 MB
Memory / node [GB]	32	96
Memory tech.	HBM	DDR4-2666
Memory channels	4	6 per socket
Peak memory bandwidth [GB/s]	1024 GB/s	256 GB/s
Num. of nodes	192	3456
Interconnection	TofuD	Intel OmniPath
Peak network bandwidth [GB/s]	6.80	12.00

III. MICRO-BENCHMARKS

A. Floating point throughput

We designed a micro-kernel, called FPU_μKernel, to measure the peak floating-point throughput of the machine. It contains exclusively fused-multiply-accumulate assembly instructions with no data dependencies between them. The kernel has six versions given by combining two types of instructions (scalar/vector) and three datatype precisions (half/single/double).

The theoretical peak performance of the vector unit P_v can be computed as $P_v = s \cdot i \cdot f \cdot o$, where s is the number of elements processed in parallel by the vector unit (e.g., four single-precision elements in NEON), i is the number of instructions issued per cycle; f is the frequency of the core, and o is the number of floating-point operations made by the instruction (e.g., fused-multiply-accumulate does two floating-point operations).

Figure 1 shows the results of the FPU_μKernel on one core of each machine. The x-axis represents different floating-point datatypes, and the y-axis represents performance. Each bar has a number representing the percentage of the theoretical peak performance achieved by the FPU_μKernel. We observe that the measurements shown in Figure 1 match almost perfectly with the theoretical values of both machines.

We verified there is no variability of the performance within a node running a multi-threaded version of the FPU_μKernel. Finally, we verified that there is no variability across the nodes.

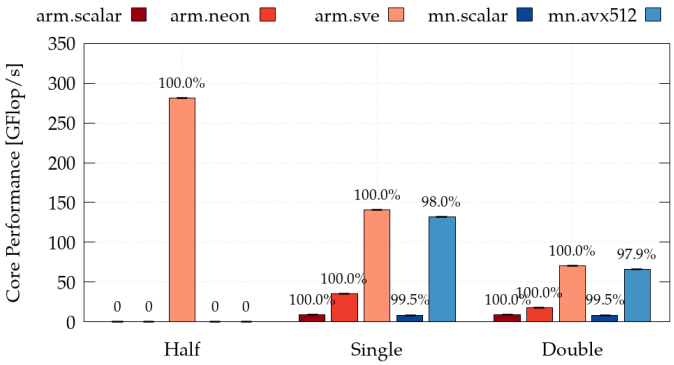

Fig. 1. Sustained performance in one core of the six versions of the FPU_μKernel in CTE-Arm and MareNostrum 4 nodes

B. Memory performance

Here we evaluate the memory bandwidth using STREAM [6], a simple synthetic benchmark to measure sustainable memory bandwidth. Our study analyses an A64FX node from the CTE-Arm cluster, compared side by side with a Skylake node from MareNostrum 4 using the C-version[2] and the Fortran-version[3] of STREAM. STREAM kernels iterate through data arrays of double-precision floating-point elements (8 bytes) with a size fixed at compile time. The number of elements of each array E must be greater than the maximum between ten million elements and four times the size of the sum of all the last-level caches. $E \geq max\ \{10^7\ ;\ 4 \cdot S/8\}$, where E is the number of elements of each array, and S is the size of the last level cache in bytes.

We run the benchmark by fixing the problem size and increasing the number of OpenMP threads. Table II shows the compiler version and flags for each machine. For the Fortran version of STREAM, we added `-CcI4I8` to index large arrays. We executed using $E = 610 \cdot 10^6$ and $E = 400 \cdot 10^6$ elements for CTE-Arm and MareNostrum 4, respectively.

Figure 2 shows the achieved bandwidth on CTE-Arm (red lines) and MareNostrum 4 (blue lines) executing the C-version (darker colors) and the Fortran version (lighter colors) of all STREAM kernels. The $x - axis$ represents the number of OpenMP threads, growing up to the number of cores in each node, and the $y - axis$ indicates the maximum bandwidth. The figure also includes two horizontal lines representing the peak bandwidth achieved on each processor. We repeated each test several times, and we verified that the variability across different executions is negligible.

Figure 2 shows that A64FX reaches its highest bandwidth (292.0 GB/s corresponding to 29% of the peak) when running with 24 OpenMP threads while MareNostrum 4 obtains its best result (201.2 GB/s 66% of the peak) with 48 OpenMP threads. On CTE-Arm, it is essential to note that the OpenMP-only code reaches a very low percentage of the peak (29%) and

[2] http://www.cs.virginia.edu/stream/FTP/Code/stream.c
[3] http://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.f

TABLE II
BUILD CONFIGURATIONS FOR STREAM

Build	Compiler	Compiler Flags
CTE-Arm OpenMP	Fujitsu/1.2.26b	`-Kfast,parallel -KA64FX -KSVE -KARMV8_3_A -Kopenmp -Kzfill=100 -Kprefetch_sequential=soft -Kprefetch_iteration=8 -Kprefetch_iteration_L2=16 -Knounroll -mcmodel=large`
CTE-Arm MPI+OpenMP	Fujitsu/1.2.26b	`-Kfast,parallel -KA64FX -KSVE -KARMV8_3_A -Kopenmp -Kzfill=100 -Kprefetch_sequential=soft -Kprefetch_iteration=8 -Kprefetch_iteration_L2=16 -Knounroll`
MareNostrum 4 OpenMP	Intel/19.1.1.217	`-O3 -xHost -qopenmp-link=static -qopenmp`
MareNostrum 4 MPI+OpenMP	Intel/19.1.1.217	`-O3 -xHost -qopenmp-link=static -qopenmp`

that different languages deliver slightly different performance, with C running $\sim 10\%$ faster than Fortran.

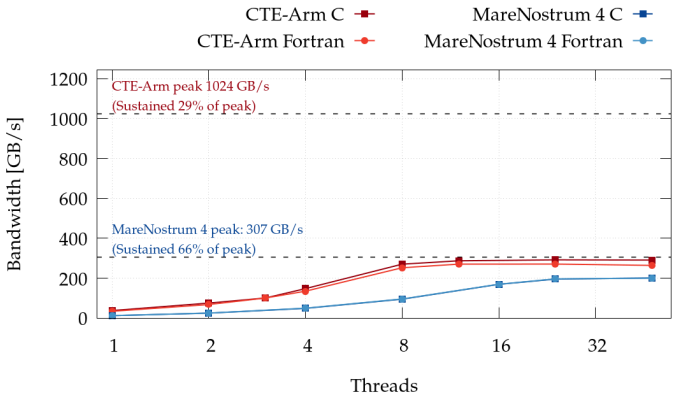

Fig. 2. STREAM Triad bandwidth with OpenMP in CTE-Arm and MareNostrum 4 nodes (thread binding: spread)

Since the OpenMP version of STREAM delivers such a poor performance, we decided to test the hybrid version of the same TRIAD kernel coded using MPI+OpenMP. Figure 3 presents the bandwidth measured when running the STREAM Triad version leveraging the shared memory parallelization (OpenMP) combined with MPI. We pinned at most one process per NUMA node in each cluster (CMG in CTE-Arm and Socket in MareNostrum 4). The combination of MPI ranks and OpenMP threads are shown in the plot on top of each point in the form of *MPI-rank* $\times$ *OMP-thread*. The Fortran version of the STREAM Triad benchmark reaches the highest bandwidth on CTE-Arm, 862.6 GB/s, corresponding to 84 % of the theoretical peak. Interestingly, the C version of the same benchmark reaches only 421.1 GB/s, but we do not have an explanation for this.

C. Network performance

In this section, we present our evaluation of the communication network using a custom micro-benchmark based on the OSU Benchmarks [4] (version 5.6.3), a collection of small synthetic codes to measure different network performance aspects. We refer to [7], where it is stated that the peak bandwidth of the TofuD interconnect is 6.8 GB/s. In CTE-Arm we used the MPI implementation provided by Fujitsu version 1.1.18.

[4] http://mvapich.cse.ohio-state.edu/benchmarks/

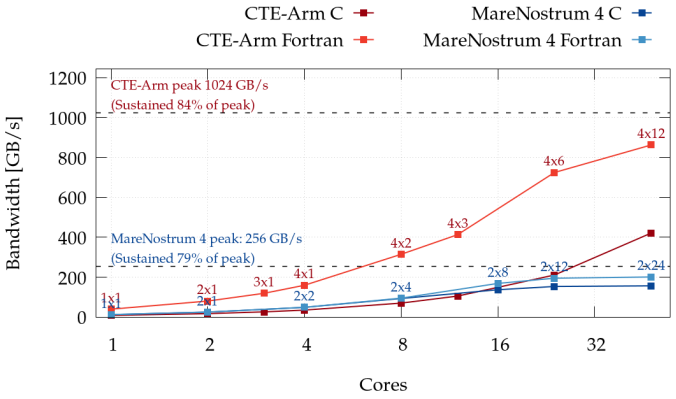

Fig. 3. STREAM Triad bandwidth with MPI+OpenMP in CTE-Arm and MareNostrum 4 nodes

The program measures the network bandwidth on a point-to-point communication between two processes. The program iterates a loop N times given a fixed message size of s Bytes. For each iteration, there is one `MPI_Sendrecv` call. We measure the time it took to complete N iterations by placing two timestamps (t_s and t_e) at the beginning and the end of the loop. Thus, the bandwidth reported by the test is computed as $B = \frac{s \cdot N}{t_e - t_s}$.

We repeated the tests for multiple pairs of nodes to determine if there were systematic weak links on CTE-Arm. Figure 4 shows a map where the axes represent the pair of nodes, and each cell is color-coded to indicate the bandwidth. We present the measurements for message sizes of 256 B as representative of medium message sizes of a hypothetical HPC application. We observe recurring patterns along the diagonals where pairs of nodes have higher bandwidth. This fact is due to the network topology, as a point-to-point message exchange on a torus network such as TofuD requires a different number of hops depending on the node's position on the torus topology. The color pattern also highlights a node `arms0b1-11c` that achieves very low bandwidth when operates as a Receiver. Interestingly enough, the same node does not seem to have bandwidth limitations when it operates as Sender.

While the color code of Figure 4 is helpful to identify patterns visually, it has limitations when we want to quantify the network bandwidth across different pairs of nodes and with different sizes of exchanged data. For this reason, in

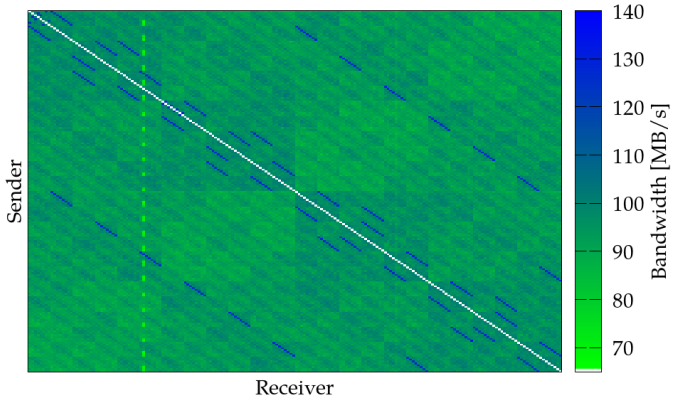

Fig. 4. Bandwidth of all node-pairs of CTE-Arm (msg. size: 256 B)

Figure 5, we analyze the results of all tests with all pairs of nodes plotting in a color scale the density of nodes achieving a given bandwidth. On the x-axis, of Figure 5 we represent a range of bandwidths. On the y-axis, we represent the message size exchanged by each pair of nodes in the CTE-Arm cluster. The color scale represents a histogram of how the bandwidth results are distributed, with green light meaning a small number of occurrences, while dark blue represents a high number of occurrences.

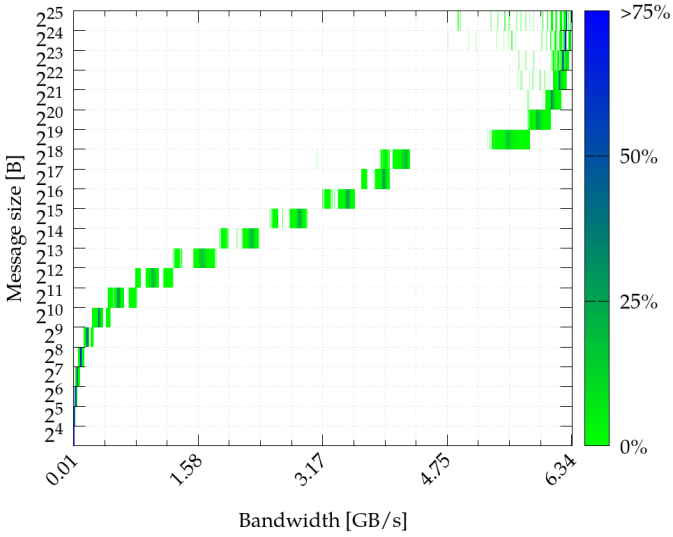

Fig. 5. Distribution of the bandwidth on all pair of nodes in the CTE-Arm network with different message sizes

It is interesting to note the high variability with message sizes above 2^{20} (1 MB) and the bimodal characterization of the bandwidth distribution with message sizes between 2^{10} and 2^{18} (1 kB to 256 kB). We currently have no explanation for this behavior.

IV. HPC BENCHMARKS

A. Linpack

We run all our tests using a vendor-provided binary which is specifically optimized for the target architecture. We present our scalability study for both clusters, CTE-Arm and MareNostrum 4, where we fill whole nodes up to 192 nodes. The Linpack binary supports hybrid parallelization with MPI+OpenMP. In CTE-Arm, we mapped 4 MPI ranks per node, allocated to different CMGs. In MareNostrum 4, we mapped 1 MPI rank per node, Intel's recommended configuration, and maximizes performance. For each run, the N parameter has been chosen so that the problem size is $\geq 80\%$ of the total available memory across all nodes. The P and Q parameters have been chosen so that, for n MPI ranks, $P \times Q = n$.

Figure 6 shows the scalability of Linpack in CTE-Arm and MareNostrum 4. The x-axis represents the number of nodes, and the y-axis represents the performance in GFlop/s. Each point is the performance reported by the benchmark in a single run. The colored horizontal dashed lines indicate the maximum performance achieved in each cluster. The black horizontal dashed lines indicate the theoretical peak performance of each cluster.

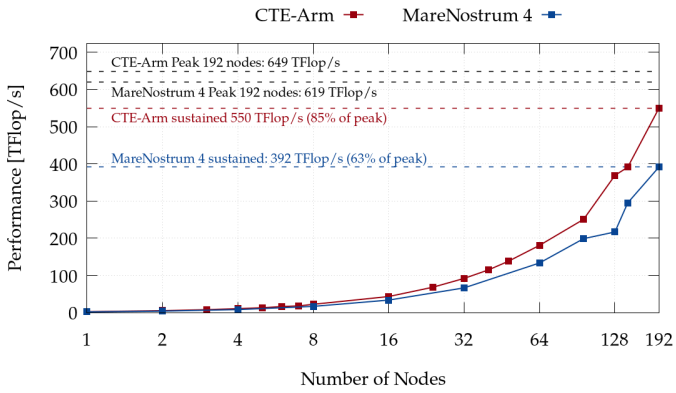

Fig. 6. Linpack scalability in CTE-Arm and MareNostrum 4

We observe that 192 A64FX nodes perform closer to the theoretical peak compared to 192 nodes of MareNostrum 4. CTE-Arm achieves 85% of the peak compared to 63% of MareNostrum 4. In the HPL Top500 list of November 2020[5], Fugaku recorded 82% of the theoretical peak, which is 3% below our results in CTE-Arm.

B. HPCG

We run two versions of HPCG: *i)* the Vanilla version compiled *as-is* from the official repository [6], and *ii)* the Optimized version provided as a binary specifically tuned for each machine by the vendor.

To compile de Vanilla version in CTE-Arm, we modified the default Makefile adding the compilation flags `-HPCG_NO_OPENMP -DHPCG_CONTIGUOUS_ARRAYS -Kfast -Krestp=all`. We used the Fujitsu compiler 1.1.18. In MareNostrum 4, we used the default `ICPC_MPI` Makefile, which includes the optimization

[5] https://www.top500.org/lists/top500/2020/11/

[6] https://github.com/hpcg-benchmark/hpcg/releases/tag/HPCG-release-3-1-0

flags `$(HPCG_DEFS) -O3 -mavx`. We tried other combinations of flags such as `-O3 -mtune=skylake -xCORE-AVX512` but performance did not improve. We used the Intel compiler 2018.3 paired with Intel MPI and the MKL mathematical libraries.

Based on the study by Ruiz et al. [8], we know that the Vanilla version of HPCG does not take advantage of OpenMP. Thus, we focused on the MPI version of the benchmark. All executions have been performed with the parameters `-nx=48 -ny=88 -nz=88 -rt=300` running with the MPI-only version of the benchmark and one rank per core (i.e., 48 ranks per node). In CTE-Arm, we set the environment variables `FLIB_FASTOMP=TRUE`, `FLIB_HPCFUNC=TRUE`, and `XOS_MMM_L_PAGING_POLICY=demand:demand:demand`.

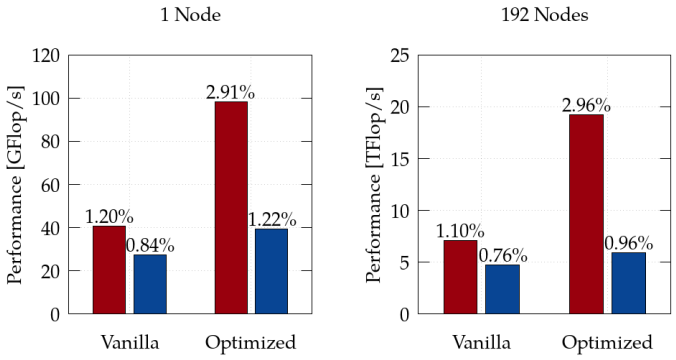

Fig. 7. HPCG performance in CTE-Arm and MareNostrum 4 for one and 192 nodes

Figure 7 shows the performance reported by HPCG in CTE-Arm and MareNostrum 4 for one and 192 nodes. We show both the Vanilla and Optimized versions of the benchmark. The y-axis represents the performance (GFlop/s for one node and TFlop/s for 192 nodes). The numbers on top of each bar represent the percentage of the theoretical peak performance.

In CTE-Arm, the performance of HPCG with one and 192 nodes is 2.91% and 2.96% of the theoretical peak performance, respectively. This is slightly below to the 3.62% of Fugaku in the HPCG Top500 list of November 2020 [7].

V. Scientific Applications

In this section, we present the first evaluation on CTE-Arm of five HPC applications: Alya, NEMO, Gromacs, OpenIFS, and WRF. Alya, NEMO, and Gromacs are part of the Unified European Applications Benchmark Suite (UEABS) of PRACE, a set of twelve relevant codes together with their data sets, which can realistically be run on large systems. We voluntarily tested the applications "as is", prioritizing MPI-only parallelization, to report the performance that a scientist could obtain when compiling without deep knowledge of the architecture and without tuning the code for it. For all applications, we measure their strong scaling behavior on CTE-Arm and MareNostrum 4. In Table III we show the configurations we used for building all applications. Compilers or libraries with the suffix `-sve` offers SVE support.

[7] https://www.top500.org/lists/hpcg/hpcg-november-2020/

A. Alya

Alya [9] is a high-performance computational mechanics code developed at the Barcelona Supercomputing Center. Alya can solve different physics, including incompressible/compressible turbulent flows, solid mechanics, chemistry, particle transport, heat transfer, and electrical propagation. Our study uses the TestCaseB input set, which models a sphere mesh with 132 million elements. The code[8] and input set[9] are publicly accessible in the UEABS repository. We run Alya with an MPI-only parallelization. Our first attempt to compile Alya was using the Fujitsu compiler 1.2.26b. Alya uses a Makefile hierarchy to compile each Fortran module and link them all together. However, we were not able to complete the compilation process because the compiler hanged for the most complex files. In this document, we present performance results when compiling Alya with the GNU compiler.

The execution of Alya is divided into an initialization phase, a computational phase, and a finalization phase. The computational phase is further divided into iterations which we call time steps. The number of time steps to complete the simulation depends on the input set. The TestCaseB input set has 20 time steps.

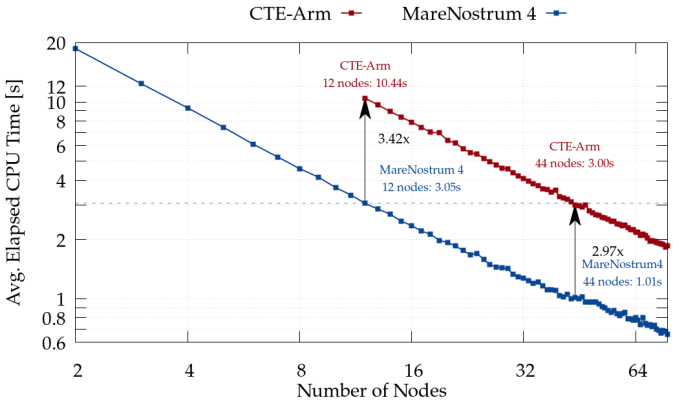

Fig. 8. Scalability of Alya in CTE-Arm and MareNostrum 4

Figure 8 shows the strong scalability study of Alya in CTE-Arm and MareNostrum 4. The x-axis represents the number of nodes, while the y-axis represents the average time step. Each point represents the *Elapsed CPU Time* as reported by the application averaged across 19 time steps (the first iteration is discarded). The input set requires at least 12 A64FX nodes because it utilizes a large amount of memory. For runs between 12 and 16 nodes, CTE-Arm is consistently 3.4× slower than MareNostrum 4.

We extended our study of CTE-Arm up to 78 nodes to find the configuration that can match the performance of 16 nodes of MareNostrum 4. The run with 44 A64FX nodes achieves the same elapsed time than 12 MareNostrum 4 nodes.

Within one time step of Alya, we find multiple phases. The two most time-consuming phases are the Assembly phase and

[8] https://repository.prace-ri.eu/ueabs/ALYA/2.1/Alya.tar.gz
[9] https://repository.prace-ri.eu/ueabs/ALYA/2.1/TestCaseB.tar.gz

TABLE III
BUILD CONFIGURATIONS FOR ALL HPC APPLICATIONS

Application		CTE-Arm	MareNostrum 4
Alya	Compiler	GNU/8.3.1-sve	GNU/8.4.2
	Flags	`-O3 -march=armv8.2-a+sve -msve-vector-bits=512 -ffree-line-length-512 -DNDIMEPAR -DVECTOR_SIZE=16 -DMETIS`	`-O3 -march=skylake-avx512 -ffree-line-length-none -fimplicit-none -DNDIMEPAR -DVECTOR_SIZE=16 -DMETIS`
	MPI Flavor	Fujitsu/1.1.18	OpenMPI/4.0.2
	Metis	metis/4.0	metis/4.0
NEMO	Compiler	GNU/8.3.1-sve	Intel/2017.4
	MPI Flavor	Fujitsu/1.2.26b	Intel/2018.4
	Dependencies	HDF5/1.12.0 NetCDF-C/4.7.4 NetCDF-F/4.5.3	HDF5/1.8.19 NetCDF-C/4.2 NetCDF-F/4.2
	C Flags	`-O3`	`-O3`
	Fortran Flags	`-fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer -ffree-line-length-none`	`-g -i4 -r8 -O3 -xCORE-AVX512 -mtune=skylake -fp-model strict -fno-alias -traceback`
Gromacs	Compiler	GNU/11.0.0	Intel/2018.4
	Flags	`-O3 -fopenmp -march=armv8.2-a+sve -msve-vector-bits=512`	`-O3 -qopenmp -xCORE-AVX512 -qopt-zmm-usage=high`
	MPI Flavor	Fujitsu/1.2.26b	Intel/2018.4
	Dependencies	fftw3/3.3.9-sve Fujitsu SSL2/1.2.26b	fftw/3.3.8 MKL/2018.4
OpenIFS	Compiler	GNU/8.3.1-sve	Intel/2018.4
	C Flags	`-O0`	`-O0`
	Fortran Flags	`-O2 -fconvert=big-endian -fopenmp -ffree-line-length-none -fdefault-real-8 -fdefault-double-8`	`-m64 -O2 -fpe0 -fp-model precise -fp-speculation=safe -convert big_endian -r8`
	MPI Flavor	Fujitsu/1.2.26b	Intel/2018.4
	Dependencies	HDF5/1.12.0 NetCDF-C/4.7.4 NetCDF-F/4.5.3 eccodes/2.18.0 BLAS/Internal LAPACK/Internal	HDF5/1.8.19 NetCDF-C/4.4.1.1 NetCDF-F/4.4.1.1 eccodes/2.18.0 MKL/2018.4
WRF	Compiler	GNU/8.3.1-sve	Intel/2017.4
	MPI Flavor	Fujitsu/1.2.26b	Intel/2017.4
	Dependencies	NETCDF/4.2 HDF5/1.8.19	NETCDF/4.4.1.1 HDF5/1.8.19
	CFLAGS_LOCAL	`-w -O3 -c`	`-w -O3 -ip`
	FCOPTIM	`-O2 -ftree-vectorize -funroll-loops`	`-O3`
	FORMAT_FIXED	`-ffixed-form`	`-FI -cpp`
	FORMAT_FREE	`-ffree-form -ffree-line-length-none`	`-FR -cpp`
	BYTESWAPIO	`-fconvert=big-endian -frecord-marker=4`	`-convert big_endian`
	FCBASEOPTS_NO_G	`-w $(FORMAT_FREE) $(BYTESWAPIO)`	`-ip -fp-model precise -w -ftz -align all -fno-alias $(FORMAT_FREE) $(BYTESWAPIO)`
	FCBASEOPTS	`$(FCBASEOPTS_NO_G) $(FCDEBUG)`	`$(FCBASEOPTS_NO_G) $(FCDEBUG)`

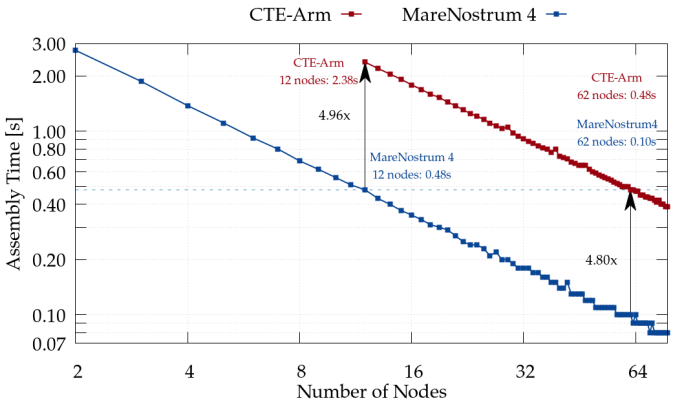

Fig. 9. Alya: Assembly phase in CTE-Arm and MareNostrum 4

the Solver phase. The Assembly phase is a computationally intensive phase that can benefit from SIMD and vectorization techniques. Figure 9 shows the scalability of this phase across multiple nodes of CTE-Arm and MareNostrum 4. Each point represents the elapsed time of the Assembly phase of the slowest process averaged across 19 time steps. We observe that 12 nodes of MareNostrum 4 are $4.96\times$ faster than 12 nodes of CTE-Arm. It takes at least 62 nodes of CTE-Arm to achieve the same performance as 12 nodes of MareNostrum 4.

The Solver phase is further divided into multiple iterations, which are separated by collective MPI communications. Thus, the Solver phase is heavily dominated by process communication and memory transactions. Figure 10 shows the scalability of the Solver phase across multiple nodes of CTE-Arm and MareNostrum 4. Each point represents the elapsed time of the Solver phase of the slowest process averaged across 19 time steps. In contrast to the Assembly phase, we observe a much smaller gap between the performance of 12 MareNostrum 4 nodes and 12 CTE-Arm nodes ($1.79\times$ in the Solver phase compared to $4.96\times$ in the Assembly phase). Being the Solver

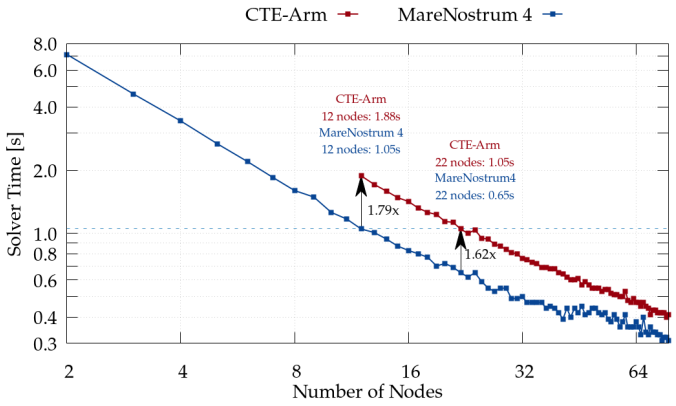

Fig. 10. Alya: Solver phase in CTE-Arm and MareNostrum 4

phase more memory-bound than the Assembly phase, we suspect that it can take advantage of the higher performance of the HBM within a A64FX compared to the DDR4 of MareNostrum 4. Also, we observe that it takes at least 22 nodes of CTE-Arm to achieve the same performance as 12 nodes of MareNostrum 4.

B. NEMO

NEMO[10] (Nucleus for European Modelling of the Ocean) is a mathematical modeling framework for research activities and prediction services in ocean and climate sciences developed by a European consortium. It is intended to be a tool for studying the ocean and its interaction with the other components of the earth's climate system over a large number of space and time scales. The model uses a curvilinear orthogonal grid in the horizontal direction, and in the vertical direction, a full or partial step z-coordinate, or s-coordinate, or a mixture of the two. The distribution of variables is a three-dimensional Arakawa C-type grid for most of the cases. The model is implemented in Fortran 90, with preprocessing (C-pre-processor) and parallelized by domain decomposition with MPI. The ORCA family[11] is a series of global ocean configurations. The NEMO system is provided with five built-in ORCA configurations, which differ in horizontal resolution. The use case executed is BENCH [10], a simplified configuration designed to evaluate performance that mimics the computational behavior and communication patterns of an ORCA-like execution with a horizontal resolution of 1 degree (corresponding to ORCA1).

Our first attempt to compile the application was using the Fujitsu compiler. We encountered several compilation errors, and we were not able to successfully compile NEMO. The compilation errors do not appear when compiling with the GNU compiler toolchain, so we decided to continue our study with the GNU compiler. For our evaluation, we compiled the version of NEMO v4.0.2 with the compiler setting and dependencies reported in Table III.

[10] https://www.nemo-ocean.eu/
[11] https://www.nemo-ocean.eu/doc/node108.html

We measured the execution time averaging three runs of NEMO running from 8 up to 192 compute nodes of CTE-Arm and from 1 up to 24 nodes in MareNostrum 4. Results are depicted in Figure 11. On the x-axis, we report the number of compute nodes (fully populated with MPI processes) on logarithmic scale. The y-axis reports the execution time in seconds, also on logarithmic scale.

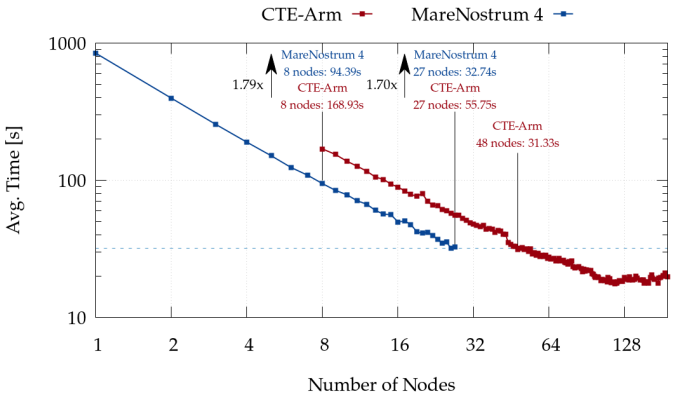

Fig. 11. NEMO: Scalability in CTE-Arm and MareNostrum 4

We note that we need at least 8 nodes of CTE-Arm to fit the input of NEMO selected for this evaluation because of memory constraints. The performance of MareNostrum 4 is between 1.70× and 1.79× higher than CTE-Arm It takes 48 A64FX nodes to achieve the same performance as 27 nodes of MareNostrum 4. We also note that the scalability on CTE-Arm flattens at around 128 nodes because of strong scalability limitations (problem size too small for the number of nodes).

C. Gromacs

Gromacs is a versatile package to perform molecular dynamics, i.e., simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids, and nucleic acids with many complicated bonded interactions. Since Gromacs is extremely fast at calculating the nonbonded interactions (that usually dominate simulations), several research groups are also using it for research on non-biological systems, e.g., polymers.

Our first attempt to compile Gromacs was using the Fujitsu compiler 1.2.26b. Gromacs uses `cmake` as its building framework so we also used a compatible version of `cmake`. During the `cmake` configuration, we enabled the MPI parallelization and indicated to use the external libraries that we supplied: `fftw3`, BLAS, and LAPACK. We also enabled the `SVE` optimization implemented in Gromacs using the `-DGMX_SIMD=ARM_SVE` flag. Lastly, we preceded the `cmake` command with environment variables that indicate the compiler and compiler flags that should be used. However, we were not able to compile the application because of an error in the `cmake` step of the build process. Since we could not compile Gromacs using the Fujitsu compiler, we decided to try the GNU compiler with the basic flags reported in Table III.

The study presented in this section includes results of Gromacs in the CTE-Arm cluster compiled with the GNU compiler. We employed version 11.0.0 because 8.3.1-sve does not meet the requirements of Gromacs.

We used the input set `lignocellulose-rf`[12] [11] from the UEABS repository for our study. The `lignocellulose-rf` use case uses a reaction field for electrostatics, which implies good scalability in a large number of nodes. It leverages the MPI and OpenMP parallelization of Gromacs and runs a simulation with 10000 steps. We run a scalability study of Gromacs at two levels of scale: *i)* Single-node, and *ii)* Multi-node. For each run, we fixed the number of OpenMP threads per MPI process to 6 and increased the number of processes like recommended by Gromacs developers.

Figure 12 and Figure 13 show the scalability study within one node and across multiple nodes of CTE-Arm and MareNostrum 4 respectively. The x-axes represent the number of cores and nodes, respectively, while the y-axes represent the time (in days) to compute one nanosecond of simulation.

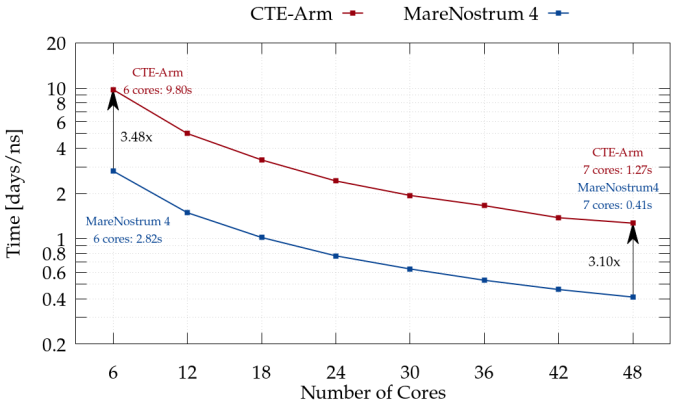

Fig. 12. Gromacs: Scalability in one node of CTE-Arm and MareNostrum 4

Looking at the single-node scalability study, we observe that MareNostrum 4 is consistently outperforming CTE-Arm. With 6 cores, CTE-Arm is $3.48\times$ slower than MareNostrum 4. With a whole node, CTE-Arm is $3.10\times$ slower than MareNostrum 4.

Looking at the multi-node scalability study, we observe that the run with 16 MPI processes performs unexpectedly bad in both machines. We currently do not have an explanation for this behavior. We tested a run with the same number of cores distributed in 12 MPI processes and 8 OpenMP threads. The performance of this alternative configuration follows the scalability trend in both machines as indicated by the dotted lines in Figure 13. With 144 full nodes, CTE-Arm is $1.5\times$ slower than MareNostrum 4.

D. OpenIFS

OpenIFS is a numerical weather prediction system, from medium-range to seasonal timescales. It is developed and

[12] https://repository.prace-ri.eu/ueabs/GROMACS/1.2/GROMACS_TestCaseB.tar.gz

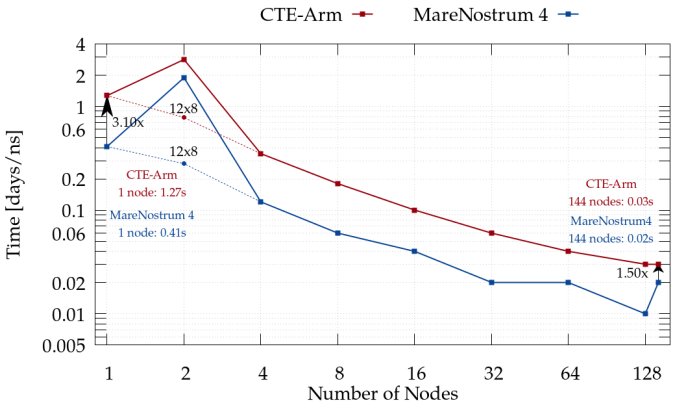

Fig. 13. Gromacs: Scalability across nodes of CTE-Arm and MareNostrum 4

maintained by the European Centre for Medium-Range Weather Forecasts (ECMWF) and provides the same forecast capability as IFS (Integrated Forecasting System) but with an easy-to-use version. In this section, we present our results of OpenIFS in CTE-Arm and MareNostrum 4. We used version oifs43r3v1 in both clusters. Our first attempt to compile the application in CTE-Arm was using the Fujitsu compiler. However, we encountered a compiler error that required modifying the code. The modifications to the code were minimal. After successfully compiling OpenIFS with the Fujitsu compiler, we encountered an error during the execution that we could not solve. Finally, we decided to compile and run OpenIFS using the GNU compiler toolchain.

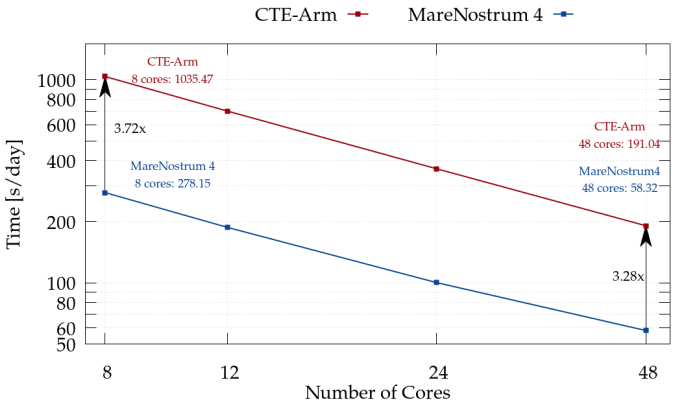

Fig. 14. OpenIFS: Scalability in one node of CTE-Arm and MareNostrum 4

Table III shows the software configuration of OpenIFS in CTE-Arm and MareNostrum 4. The configuration in MareNostrum 4 is based on the Makefile distributed with the application. Software dependencies were already installed in the cluster and available to all users. The configuration in CTE-Arm is based on MareNostrum 4, translating the optimization flags from the Intel compiler to the GNU compiler. It was necessary to compile and install some of the software dependencies from the source codes. BLAS and LAPACK are marked as *Internal* in CTE-Arm because the build uses the implementation of these libraries provided with the code of OpenIFS.

Our study of OpenIFS is at two levels of scale: *i)* Single-node, using the `TL255L91` input set and; *ii)* Multi-node, using the `Tco511L91` input set. Figure 14 and Figure 15 show the scalability study of both input sets in CTE-Arm and MareNostrum 4. The x-axes represent the number of MPI ranks or the number of nodes while the y-axes the time to simulate one day measured in seconds. Each point in the plot represents the average performance reported by the application across five executions. With 8 ranks, the CTE-Arm is 3.72× slower than MareNostrum 4. With one full node, the performance CTE-Arm is 3.28× slower than MareNostrum 4.

The multi-node input set runs on a minimum of 32 A64FX nodes because of memory requirements. With 32 nodes, CTE-Arm is 3.55× slower than MareNostrum 4. With 128 nodes, CTE-Arm is 2.56× slower than MareNostrum 4.

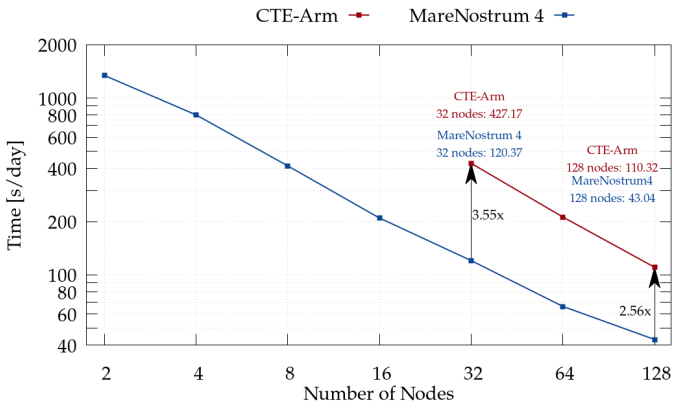

Fig. 15. OpenIFS: Scalability across nodes of CTE-Arm and MareNostrum 4

E. WRF

The Weather Research and Forecasting (WRF) Model is a next-generation mesoscale numerical weather prediction system designed for atmospheric research and operational forecasting applications. The model serves a wide range of meteorological applications across scales from tens of meters to thousands of kilometers. Our study used an input set representing the Iberian peninsula with 4 km resolution and 56 hours of simulation. During execution, WRF generates an output frame for each hour of simulation, summing up a total of 54 frames per run. In order to evaluate the effects of the IO operations during the output of the frames, we run all our tests two times: *i)* with IO enabled and, *ii)* with IO disabled.

Figure 16 shows the scalability study of WRF in CTE-Arm and MareNostrum 4. The x-axis represents the number of nodes, while the y-axis represents time. Each point represents the elapsed time as reported by the application. We observe that there is little difference in time between the runs that enable IO and the runs that do not, giving the runs with IO disabled a slight advantage. With one node, CTE-Arm is 2.16× slower than MareNostrum 4. With 64 nodes, CTE-Arm is 2.23× slower than MareNostrum 4. MareNostrum 4 is consistently outperforming CTE-Arm.

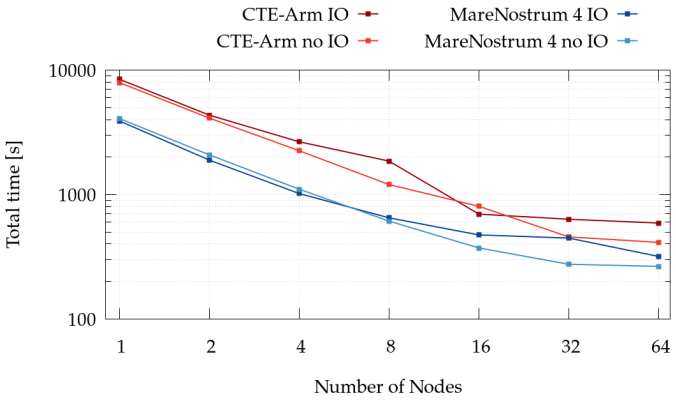

Fig. 16. WRF: Scalability across nodes of CTE-Arm and MareNostrum 4

VI. Conclusions

This paper evaluated the CTE-Arm machine, an HPC cluster housing 192 nodes with the same architecture of the Fugaku supercomputer. Table IV summarizes the speedups of the benchmarks and applications tested.

TABLE IV
SPEEDUP OF CTE-ARM RELATIVE TO MARENOSTRUM 4

	Number of compute nodes					
Applications	**1**	**16**	**32**	**64**	**128**	**192**
LINPACK	1.25	1.28	1.38	1.35	1.70	1.40
HPCG	2.50	N/A	N/A	N/A	N/A	3.24
Alya	NP	0.30	0.31	0.37	N/A	N/A
OpenIFS	0.31	NP	0.28	0.31	0.39	N/A
Gromacs	0.32	0.36	0.38	0.43	0.54	0.33
WRF	0.49	0.46	0.60	0.64	N/A	N/A
NEMO	NP	0.56	N/A	N/A	N/A	N/A

We notice that synthetic benchmarks have a speedup of up to 1.7× for LINPACK and up to 3.4× for HPCG compared to MareNostrum 4. The HPC applications tested suffer a slow-down between 1.6× and 3.4× compared to MareNostrum 4. We verified that the compiler could not leverage the SVE unit in several cases, leaving the performance to be delivered by the scalar core. Thus, the overall poor performance of applications can be explained by the weaker out-of-order capabilities of the scalar core of the A64FX CPU compared to Intel one. For this reason, tools should focus on more aggressive vectorization, so to take advantage of SVE. The weaker scalar core of A64FX is somewhat compensated by its fast memory subsystem, which mitigates performance drops with memory-bound applications (e.g., the Solver phase of Alya).

Besides the slowdown, other factors are harming the adoption of CTE-Arm as a general-purpose production machine: *i)* Single node memory limitations: e.g., Alya, OpenIFS, and NEMO can not be run with a low number of nodes due to this limitation, NP in Table IV. *ii)* Compiler restrictions: only Fujitsu compiler is available, and not all applications can be compiled with it. *iii)* MPI restrictions: the only available MPI implementation that supports the Tofu network is strongly

dependent on the Fujitsu Compiler. Combining this MPI installation with the GNU compiler suite is not accessible for an average user. *iv)* Job scheduler restrictions: the job scheduler does not allow allocating specific nodes or enforcing specific process binding. The effort of Fujitsu and the Arm community should lead towards a more complete and robust set of tools (e.g., compilers) and system software (e.g., MPI libraries, job scheduler) for A64FX and future Arm-based CPU targeting HPC. As a side note, we point out that HPCG is supposed to be a "more representative" benchmark than LINPACK. However, it does not seem to predict/mimic the trend of any of the applications tested.

ACKNOWLEDGMENTS

This work is partially supported by the Spanish Government (SEV-2015-0493), by the Spanish Ministry of Science and Technology (TIN2015-65316-P), by the Generalitat de Catalunya (2017-SGR-1414), by the European and Horizon 2020 POP CoE (GA n. 824080).

REFERENCES

[1] S. McIntosh-Smith *et al.*, "A performance analysis of the first generation of HPC-optimized Arm processors," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 16, 2019.

[2] F. Mantovani, M. Garcia-Gasulla *et al.*, "Performance and energy consumption of HPC workloads on a cluster based on Arm ThunderX2 CPU," *Future Generation Computer Systems*, 2020.

[3] M. Nakao, K. Ueno, K. Fujisawa, Y. Kodama, and M. Sato, "Performance of the supercomputer fugaku for breadth-first search in graph500 benchmark," in *International Conference on High Performance Computing*. Springer, 2021, pp. 372–390.

[4] E. Arima, Y. Kodama, T. Odajima, M. Tsuji, and M. Sato, "Power/performance/area evaluations for next-generation hpc processors using the a64fx chip," in *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2021, pp. 1–6.

[5] M. Sato, Y. Ishikawa, H. Tomita, Y. Kodama, T. Odajima, M. Tsuji, H. Yashiro, M. Aoki, N. Shida, I. Miyoshi *et al.*, "Co-design for a64fx manycore processor and" fugaku"," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[6] J. D. McCalpin *et al.*, "Memory bandwidth and machine balance in current high performance computers," *IEEE computer society technical committee on computer architecture (TCCA) newsletter*, vol. 1995, pp. 19–25, 1995.

[7] Y. Ajima, T. Kawashima, T. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Y. Ikeda, T. Yoshikawa, K. Uchida *et al.*, "The tofu interconnect d," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 646–654.

[8] D. Ruiz, F. Spiga, M. Casas, M. Garcia-Gasulla, and F. Mantovani, "Open-source shared memory implementation of the hpcg benchmark: analysis, improvements and evaluation on cavium thunderx2," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019, pp. 225–232.

[9] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen *et al.*, "Alya: Multiphysics engineering simulation toward exascale," *Journal of Computational Science*, vol. 14, pp. 15–27, 2016.

[10] S. V. P. Ticco, M. C. Acosta, M. Castrillo, O. Tintó, and K. Serradell, "Keeping computational performance analysis simple: an evaluation of the nemo bench test." [Online]. Available: https://bit.ly/nemo-bench

[11] B. Lindner, L. Petridis, R. Schulz, and J. C. Smith, "Solvent-driven preferential association of lignin with regions of crystalline cellulose in molecular dynamics simulation," *Biomacromolecules*, vol. 14, no. 10, pp. 3390–3398, 2013.