Evaluating the Arm Ecosystem for High Performance Computing

Evaluating the Arm Ecosystem for High Performance Computing

Adrian Jackson
a.jackson@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, United Kingdom

Andrew Turner
a.turner@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, United Kingdom

Michèle Weiland
m.weiland@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, United Kingdom

Nick Johnson
n.johnson@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, United Kingdom

Olly Perks
olly.perks@arm.com
Arm
Warwick, United Kingdom

Mark Parsons
m.parsons@epcc.ed.ac.uk
EPCC, The University of Edinburgh
Edinburgh, United Kingdom

ABSTRACT

In recent years, Arm-based processors have arrived on the HPC scene, offering an alternative the existing status quo, which was largely dominated by x86 processors. In this paper, we evaluate the Arm ecosystem, both the hardware offering and the software stack that is available to users, by benchmarking a production HPC platform that uses Marvell's ThunderX2 processors. We investigate the performance of complex scientific applications across multiple nodes, and we also assess the maturity of the software stack and the ease of use from a users' perspective. This papers finds that the performance across our benchmarking applications is generally as good as, or better, than that of well-established platforms, and we can conclude from our experience that there are no major hurdles that might hinder wider adoption of this ecosystem within the HPC community.

CCS CONCEPTS

• **General and reference** → *Performance*; • **Computing methodologies** → *Massively parallel algorithms*; *Distributed programming languages.*

KEYWORDS

Benchmarking, Arm, ThunderX2, Marvell, Performance, Distributed Processing

ACM Reference Format:
Adrian Jackson, Andrew Turner, Michèle Weiland, Nick Johnson, Olly Perks, and Mark Parsons. 2019. Evaluating the Arm Ecosystem for High Performance Computing. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '19), June 12–14, 2019, Zurich, Switzerland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3324989.3325722

PASC '19, June 12–14, 2019, Zurich, Switzerland

ACM ISBN 978-1-4503-6770-7/19/06…$15.00
https://doi.org/10.1145/3324989.3325722

1 INTRODUCTION

There is an established ecosystem of server-level processors suitable for computational simulation and machine learning applications built around traditional x86 architectures from processor manufacturers such as Intel and AMD. However, recently, alternative processor technologies have been developed; foremost amongst these are Arm based processors from manufacturers such as Marvell (ThunderX2), Ampere (eMAG), Huawei (Kunpeng 920), Fujitsu (A64FX) and Amazon (Graviton).

The first server class processor to be commercially available in large volume is the ThunderX2 processor from Marvell. The ThunderX2 processor uses the Armv8 instruction set and it has been designed specifically for server workloads. The design includes eight DDR4 memory channels to deliver measured STREAM triad memory bandwidth in excess of 220 GB/s per dual-socket node.

However, hardware only represents one part of the ecosystem that is required to deliver a usable High Performance Computing (HPC) platform for the varied workloads of computational simulation and machine learning applications. Operating system, compiler, and library support is required to provide a functional environment that supports large scale HPC applications and to ensure applications can both be easily ported to such new hardware as well as efficiently exploit it.

In this paper we will evaluate a range of computational simulation applications on a HPC system comprised of nodes with ThunderX2 processors connected together with an Infiniband network. Our paper makes the following contributions to deepening the understanding of the performance of a production HPC system that is based on the Arm ecosystem:

(1) We outline performance measurements of the interconnect network, using established MPI benchmarks, allowing us to assess the potential scaling performance of distributed memory applications.
(2) We present and evaluate the multi-node performance of scientific applications with varying performance characteristics and compare it to the established x86 ecosystem.
(3) We evaluate the portability of applications onto this new system, compared with equivalent systems based on other processor technologies.
(4) We discuss the causes for the performance and scalability results that have been observed, and based on this we draw

conclusions with regards to the maturity of the current generation of Arm-based systems for HPC.

2 RELATED WORK

Some initial porting and performance evaluation work on ThunderX2 processors has been documented [16], outlining performance and cost benefits for some applications using ThunderX2 compared to other available Intel processors. However, these results only consider single node performance, using at most two processors, and do not consider the performance, behaviour, or functionality associated with multi-node applications. The majority of computational simulation applications [2] on large scale HPC systems require many nodes for simulations. The Mont-Blanc project has published a detailed energy usage study for a ThunderX2-based system, using specialist on-node hardware to measure the energy to solution of a number of benchmarks and mini-apps [7]. This paper investigates the performance of distributed memory communications (MPI), as well as scientific applications that use MPI, on a ThunderX2 based system, and the associated libraries required for functionality and performance.

HPC platforms are evaluated using a wide range of benchmarks, each targeting a different performance aspect; popular benchmark suites include [17] [9] [6]. These include application specific benchmarks [1], and have included benchmarking application across multiple systems [18].

3 BENCHMARKING METHODOLOGY

In order to fully evaluate the performance of the Arm-based HPE Apollo 70, we execute a range of benchmarks and applications that stress different aspects of the architecture and the software stack, and compare our results with well established HPC systems. Our benchmarking methodology adheres to the following principles:

MPI performance. The scalability of multi-node workloads relies on a high-performance interconnect and efficient inter-node communication. As the majority of HPC workloads rely on MPI to distribute work across compute nodes, we use MPI benchmarks to assess the communication performance on the HPE Apollo 70. There are a range of MPI libraries and implementations available on the benchmarking systems we are using.

Reproducibility. We use process and thread pinning to cores to ensure our results are not impacted or skewed by the operating system's process/thread management policies, and are reproducible. We also list the compiler versions and flags, as well as the libraries used, in Table 8.

Applications. The applications chosen for this benchmarking study cover different scientific domains, programming languages, libraries and performance characteristics (i.e. compute intensive, memory-bandwidth limited, etc). They also represent widely used real-life applications. GROMACS and OpenSBLI are part of the UK HPC benchmarks[1]. As we are chiefly interested in the compute performance of the applications, we disabled or reduced I/O (in particular output) as much as possible.

Multi-node benchmarks. The scaling behaviour of the applications is benchmarked and analysed, starting from a single node to up to 32 nodes in total. This allows for the identification of performance bottlenecks caused by the network or the communication libraries.

Performance comparison. The results that were achieved on the HPE Apollo 70 are compared with those from three well-established and widely deployed platforms in order to assess the relative performance of the system. The results are generally compared on a per-node basis (rather than per-core) and the test cases configurations are the same across systems.

4 BENCHMARKING SYSTEMS

The system under evaluation, an Arm-based HPE Apollo 70, is compared against well established HPC system architectures with very mature software ecosystems. This not only enables us to compare performance due to difference in the hardware, but also allows us to assess the ease of porting applications and to evaluate the relative maturity of the software stack that is available on the Arm system. Details on each of the systems used for this benchmarking activity are given below, and Table 1 summarises their compute node specifications. We have also include a calculated node memory bandwidth result for each system, collected using the STREAM benchmark, to allow comparison of the achievable memory bandwidth between the different hardware solutions.

HPE Apollo 70. The Arm-based system under evaluation is an HPE Apollo 70 cluster with dual-socket compute nodes connected with Mellanox EDR Infiniband (IB) using a non-blocking fat tree topology. Each compute node consists of two 32-core Marvell ThunderX2 processors running at 2.2GHz, and 256GB DDR4 memory. The processors on this system are set to use 1 hardware thread (SMT) per core.

SGI ICE XA. The SGI ICE XA system has compute nodes each with two 2.1 GHz, 18-core, Intel Xeon E5-2695 (Broadwell) series processors. They have 256 GB of memory shared between the two processors. The system has a single IB FDR fabric with a bandwidth of 54.5 Gb/s.

Cray XC30. This system has 24 cores (two Intel Xeon 2.7 GHz, 12-core E5-2697v2 processors) and 64 GB of DDR3 memory per node (128 GB on a small number of large memory nodes). Nodes are connected by the Cray Aries network.

Dell EMC. The Dell EMC PowerEdge system has two 16-core Intel Xeon Gold Skylake processors per compute node, running at 2.6GHz, and with up to 384 GB of DDR4 memory shared between to processors. The system uses Intel's OmniPath interconnect.

5 MPI PERFORMANCE

We have used the Intel MPI Benchmark suite to compare the MPI performance on the HPE Apollo 70 with that of the other systems evaluated, including the Cray XC30, which is known to demonstrate excellent interconnect performance. We present both point-to-point and collective performance, focussing on the *MPI_Sendrecv* benchmark for point-to-point message and the *MPI_Alltoallv* and *MPI_Allgatherv* collectives. We focus on these collectives specifically as they are generally expensive operations that are important to a range of HPC applications. We report results for the Alltoallv

Table 1: Compute node specifications.

	HPE Apollo 70	SGI ICE XA	Cray XC30	Dell EMC
Processor	Marvell ThunderX2 (ARMv8)	Intel Xeon E5-2695 (Broadwell)	Intel Xeon E5-2697 v2 (IvyBridge)	Intel Xeon Gold 6142 (Skylake)
Processor clock speed	2.2GHz	2.1GHz	2.7GHz	2.6GHz
Cores per processor	32	18	12	16
Cores per node	64	36	24	32
Threads per core	**1**, 2 or 4	**1** or 2	**1** or 2	1
Vector width	128bit	256bit	256bit	512bit
Double Precision (DP) FLOPS per cycle	8	16	8	32
Maximum node DP GFLOP/s	1126.4	1209.6	518.4	2662.4
L1 cache (core)	32KB	32KB	32KB	32KB
L2 cache (core)	256KB	256KB	256KB	1024KB
L3 cache (shared)	32MB	45MB	30MB	22MB
Memory per node	256GB	256GB	**64GB**/128GB	**192GB**/384GB
Memory per core	4GB	7.11GB	**2.66GB**/5.33GB	**6GB**/12GB
Memory channels (processor)	8	4	4	6
Specified memory bandwidth (processor)	160GB/s	76.8GB/s	59.7GB/s	119.2GB/s
Specified memory bandwidth (core)	5.0GB/s	3.3GB/s	6.4GB/s	7.5GB/s
Measured memory bandwidth (node)	221.48GB/s	97.85GB/s	72.86GB/s	144.26GB/s

and Allgatherv benchmarks for two message sizes: 128 KiB and 1 MiB. The Sendrecv benchmark was run in the on-cache mode, meaning MPI processes on the same processor can utilise the last level cache for communicating data.

For each of the systems we used the following MPI libraries:

- HPE Apollo 70: HPE MPT 2.20
- HPE Apollo 70: OpenMPI 4.0.0
- SGI ICE XA: SGI MPT 2.16
- Cray XC30: Cray MPICH2 7.5.5
- Dell EMC: Intel MPI 2017.4

Tables 2 and 3 show the Alltoallv test results for the 128KiB and 1MiB message sizes respectively. Tables 4 and 5 show the Allgatherv test results for the 128KiB and 1MiB message sizes.

The results from the MPI benchmarks highlight that the performance of some collective operations on the HPE Apollo 70 system are poor when compared to the Cray XC30. Initial investigations point towards possible configuration issues with the Infiniband interconnect and MPI library, and work is ongoing to resolve this situation, as poor performance on MPI collectives will have a negative impact on the scalability of applications that heavily rely on these operations.

We can see from the point-to-point performance, outlined in Table 6, that the Apollo 70 system has similar performance to the SGI ICE XA system, both using Infiniband networks. It is interesting that the MPT library on the Apollo 70 system achieves significantly higher bandwidth and lower latency for the on-node communication benchmarks when compared to OpenMPI. It is likely that the MPT library is taking advantage of communicating through the cache, an optimisation that does not appear to be enabled for OpenMPI, but which should be configurable to enable achieving similar performance for OpenMPI. It is also worth noting that this benchmark was using only 2 MPI processes on the node, providing an uncontested environment for the benchmark. It is likely different performance would be observed if larger numbers of MPI processes were occupying the node. We also see that the Cray XC30 exhibits better performance that the other, Infiniband or Omnipath-based, systems.

6 OVERVIEW OF BENCHMARKING APPLICATIONS

To evaluate the performance of the Apollo 70 system we have chosen a set of application and benchmarks that have a range of different performance characteristics, functionalities, and implementations.

We have chosen codes that span the main programming languages typically used within parallel computing (i.e. C, C++, and Fortran) and that have MPI, OpenMP, and hybrid (for our chosen applications MPI and OpenMP) parallel implementations.

We have also ensured that we have a range of different performance characteristics in the applications, from memory bandwidth bound codes, to those dependent on the network for good performance when using multiple nodes. We describe each application in turn in the following paragraphs, together with the test cases and the configuration used for benchmarking.

6.1 HPCG

High Performance Conjugate Gradients (HPCG [10]) is a benchmark for large-scale parallel systems that aims to represent the computational and data access patterns of a broad set of important computational simulation applications. The conjugate gradients algorithm used in the benchmark is not just floating point performance limited, it is also heavily reliant on the performance of the memory system, and to a lesser extend on the network used

Table 2: Mean Alltoallv Intel MPI Benchmark test time (ms) for a message size of 128 KiB as a function of node count.

Nodes	HPE Apollo 70 (MPT)	HPE Apollo 70 (OpenMPI)	SGI ICE XA	Cray XC30	Dell EMC
2	99.531	105.189	74.264	11.731	10.852
4	385.304	322.071	142.151	37.512	36.380
8	887.104	784.300	283.167	117.046	86.948
16	1,863.269	1,679.783	692.317	293.256	278.481

Table 3: Mean Alltoallv Intel MPI Benchmark test time (ms) for a message size of 1 MiB as a function of node count.

Nodes	HPE Apollo 70 (MPT)	HPE Apollo 70 (OpenMPI)	SGI ICE XA	Cray XC30	Dell EMC
2	754.835	833.083	231.816	95.763	89.888
4	2,966.812	2,583.005	2,135.086	307.588	307.026
8	6,971.923	6,086.923	14,811.546	977.187	783.750
16	15,544.344	13,065.089	Out of Memory	2,263.569	2,237.830

Table 4: Mean Allgatherv Intel MPI Benchmark test time (ms) for a message size of 128 KiB as a function of node count.

Nodes	HPE Apollo 70 (MPT)	HPE Apollo 70 (OpenMPI)	SGI ICE XA	Cray XC30	Dell EMC
2	18.840	10.061	29.106	4.016	7.452
4	39.174	18.182	16.394	7.893	14.658
8	86.769	36.039	30.127	19.706	29.214
16	190.696	70.600	70.542	44.222	60.268

Table 5: Mean Allgatherv Intel MPI Benchmark test time (ms) for a message size of 1 MiB as a function of node count.

Nodes	HPE Apollo 70 (MPT)	HPE Apollo 70 (OpenMPI)	SGI ICE XA	Cray XC30	Dell EMC
2	163.610	101.678	59.093	32.400	50.102
4	339.137	190.665	119.589	65.496	93.327
8	740.027	352.193	237.253	147.087	188.210
16	1,611.643	689.009	552.725	297.995	375.877

to connect the processors together. The following functionality is implemented in benchmark:

- Sparse matrix-vector multiplication.
- Vector updates.
- Global dot products.
- Local symmetric Gauss-Seidel smoother.
- Sparse triangular solve (as part of the Gauss-Seidel smoother).
- Multigrid pre-conditioned solvers.

This functionality helps to ensure it must provide a performance assessment that is more comparable to the performance that real applications could achieve for a given system, compared to other benchmarking kernels used for assessing HPC systems, such as High Performance LINPACK (HPL[11]). For this reason, HPCG is rapidly becoming the second most important benchmark for evaluating HPC system performance, and is now included in the Top500 list[1].

There is a reference implementation of HPCG available, written in C++ with MPI and OpenMP support. However, there are also optimised versions of HPCG available for a range of hardware, including Intel processors [2] and Arm systems [3]. These generally provide improved performance over the reference implementation. For this study we used the optimised versions of HPCG.

6.1.1 Test Case. The HPCG benchmark can be configured to run different size problems by providing the size of the local data block each process works on. To ensure consistency across the systems the same local block size was used for all the benchmarks: `--nx=64 --ny=64 --nz=64`. This size of local domain block was used because the optimised Arm version of HPCG is restricted to using sizes that are divisible by 2 at all grid scales. This block size may not give the best performance across all systems, but it does provide a consistent benchmark.

6.1.2 Configuration. We are using version 3.0 of HPCG for these benchmarking. For all systems a range of pure MPI and hybrid (MPI + OpenMP) configurations were tested to evaluate which gave the best performance. We used the optimal configuration for each system to run the benchmarks, outlined in Table 7.

[1] https://www.top500.org/hpcg/lists/2018/11/

[2] https://software.intel.com/en-us/mkl-linux-developer-guide-overview-of-the-intel-optimized-hpcg

[3] https://developer.arm.com/products/software-development-tools/hpc/resources/porting-and-tuning/building-hpcg-with-arm-compiler

Table 6: Mean Sendrecv Intel MPI Benchmark test time (ms) and bandwidth (MB/s)

	Mean test time (ms) *(Bandwidth GB/s)*				
Configuration	**HPE Apollo 70 (MPT)**	**HPE Apollo 70 (OpenMPI)**	**SGI ICE XA**	**Cray XC30**	**Dell EMC**
within a node same socket	0.46 *(21,114.09)*	0.33 *(13,655.82)*	0.58 *(21,342.88)*	0.38 *(26,196.39)*	0.42 *(18,796.07)*
within a node different sockets	0.75 *(18,751.15)*	0.70 *(12,402.8)*	0.78 *(21,346.13)*	0.69 *(26,329.70)*	0.87 *(18,902.10)*
between nodes	2.17 *(10,646.78)*	2.04 *(11,082.26)*	1.67 *(12,517.62)*	1.42 *(14,506.03)*	1.52 *(17,886.42)*

Table 7: HPCG process and thread configuration

System	MPI processes per node	OpenMP threads per MPI process
HPE Apollo 70	32	2
SGI ICE XA	18	2
Cray XC30	12	2
Dell EMC	32	1

6.2 COSA

COSA supports steady, time-domain (TD), and frequency-domain (harmonic balance or HB) solvers, implementing the numerical solution of the Navier-Stokes (NS) equations using a finite volume space-discretisation and multigrid (MG) integration. It is implemented in Fortran [13] and has been parallelised using MPI [12], with each MPI process working on a set of grid blocks (geometric partitions) of the simulation.

In the HB solver there exists an additional dimension compared to the steady and TD solvers, which can be viewed as a harmonic varying from 1 to Nh, a user specified number of elemental flow harmonics. However, the code does not solve directly such elemental harmonics, but rather Nh equally time-spaced snapshots of the required periodic flow field, linked to the Nh elemental harmonics using a Fourier transform.

The code is structured so that the core computational kernels can, for the most part, be reused for the steady solvers and HB simulations, with HB simply requiring an outer loop over the Nh snapshots using the steady solver kernels. COSA has been shown to exhibit good parallel scaling to large numbers of MPI processes with a sufficiently large test case [14].

6.2.1 Test Case. For the benchmarking of COSA we used a HB test case with 4 harmonics and a grid composed of 16, 384 blocks, encompassing a total of 47, 071, 232 grid cells. This limits the MPI domain decomposition to, at most, 16, 384 MPI processes. This test case requires a minimum of 1 Terabyte (TB) of memory across the MPI processes running the simulation, meaning it is not possible to run using a single process or node from the test systems we had access to. To enable fast benchmarking the simulation was run for 20 iterations, which is a significantly smaller number of iteration than a production run would typically use but is enough to provide sufficient simulation functionality to evaluate performance.

6.2.2 Configuration. Writing output data to storage can be a significant overhead in COSA, especially for simulations using small numbers of iterations. To ensure variations in the I/O hardware of the platforms being benchmarked does not affect the results collected we disabled output functionality for benchmark runs of COSA.

The benchmark was run with a single MPI process per core, and all the cores in the node utilised for the benchmarks (i.e. we were not under-populating nodes except for the case of the Cray XC30 using 512 MPI processes, as this left one node with 16 free cores).

6.3 GROMACS

GROMACS [5] 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 that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the non-bonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.

6.3.1 Test Case. The GROMACS 1400k atom benchmark is taken from the HEC BioSim website [4] and has also been used within the ARCHER benchmarks with results freely-available online [1]. This strong-scaling benchmark represents a Pair of hEGFR Dimers of 1IVO and 1NQL and we would expect this benchmark to scale well up to 16-32 compute nodes (depending on the performance of the MPI library and interconnect). This benchmark case performs minimal I/O.

6.3.2 Configuration. The single-precision version of GROMACS was used for all benchmark runs. The GROMACS `mdrun` command line option `-noconfout` was specified for all benchmark runs. Performance of the GROMACS benchmark is measured in ns/day read directly from the GROMACS output log file.

For all systems, 1 MPI process was used per physical core with 1 OpenMP thread per MPI process. A single hardware thread was used in all cases. All compute node cores were used in all benchmark runs (i.e. all compute nodes were fully populated).

6.4 OpenSBLI

OpenSBLI [15] is a Python-based modelling framework that is capable of expanding a set of differential equations written in Einstein notation, and automatically generating C code that performs the finite difference approximation to obtain a solution. This C

Table 8: Compilers, Compiler Flags and Libraries.

	Compiler	Compiler flags	Libraries
COSA			
HPE Apollo 70	GNU Fortran v8.2	-g -fdefault-double-8 -fdefault-real-8 -fcray-pointer -ftree-vectorize -O3 -ffixed-line-length-132	HPE MPT MPI library (v2.20) ARM Performance Libraries (v19.0.0)
SGI ICE XA	GNU Fortran v8.2	-g -fdefault-double-8 -fdefault-real-8 -fcray-pointer -ftree-vectorize -O3 -ffixed-line-length-132	SGI MPT 2.16 Intel MKL 17.0.2.174
Cray XC30	GNU Fortran v7.2	-g -fdefault-double-8 -fdefault-real-8 -fcray-pointer -ftree-vectorize -O3 -ffixed-line-length-132	Cray MPI library (v7.5.5) Cray LibSci (v16.11.1)
GROMACS			
HPE Apollo 70	Arm Clang 19.0.0	-std=c++11 -mcpu=native -O3 -DNDEBUG -funroll-all-loops -fexcess-precision=fast -fPIC -O3	OpenMPI 4.0.0 FFTW 3.3.8
SGI ICE XA	GCC 6.2.0	-march=core-avx2 -std=c++0x -O3 -funroll-all-loops -fexcess-precision=fast	SGI MPT 2.16 FFTW 3.3.5
Cray XC30	GCC 6.3.0	-mavx -static -O3 -ftree-vectorize -funroll-loops -std=c++11 -O3 -funroll-all-loops -fexcess-precision=fast	Cray MPICH2 7.5.5 FFTW 3.3.6
Dell EMC	Intel 17.4	-xCORE-AVX512 -mkl=sequential -std=gnu99 -std=c++11 -O3 -ip -funroll-all-loops -alias-const -ansi-alias -no-prec-div -fimf-domain-exclusion=14 -qoverride-limits	Intel MPI library 17.4 Intel MKL 17.4
HPCG			
HPE Apollo 70	Arm Clang 19.0.0	-O3 -ffast-math -funroll-loops -fopenmp -std=c++11 -ffp-contract=fast -mcpu=native	OpenMPI 4.0.0 ARM Performance Libraries (v19.0.0)
SGI ICE XA	Intel 17.0.2.174	-xCORE-AVX2 -qopenmp -std=c++11 -O3 -DNDEBUG	SGI MPT 2.16 Intel MKL 17.0.2.174
Cray XC30	Intel 17.0.0.098	-qopenmp -std=c++0x -O3 -DNDEBUG	Cray MPICH2 7.5.5 Intel MKL 17.0.0.098
OpenSBLI			
HPE Apollo 70	Arm Clang 19.0.0	-O3 -std=c99 -fPIC -Wall	OpenMPI 4.0.0 HDF5 1.10.4
SGI ICE XA	Intel 17.0.2.174	-O3 -ipo -restrict -fno-alias	SGI MPT 2.16 HDF5 1.10.1
Cray XC30	Cray Compiler v8.5.8	-O3 -hgnu	Cray MPICH2 (v7.5.2) HDF5 (v1.10.0.1)
Dell EMC	Intel 17.4	-O3 -ipo -restrict -fno-alias	Intel MPI 17.4 HDF5 1.10.1
Nektar++			
HPE Apollo 70	GNU 8.2.0	-O3 -fPIC -DNDEBUG	OpenMPI 4.0.0 ARM Performance Libraries (v19.0.0) FFTW 3.2.2 Boost 1.6.7
HPE Apollo 70	ARM Clang 19.0.0	-O3 -fPIC -DNDEBUG	OpenMPI 4.0.0 ARM Performance Libraries (v19.0.0) FFTW 3.2.2 Boost 1.6.7
SGI ICE XA	GCC 6.2.0	-O3 -fPIC -DNDEBUG	SGI MPT 2.18 FFTW 3.2.2 Boost 1.6.7

code is then targeted with the OPS library towards specific hardware backends, such as MPI/OpenMP for execution on CPUs, and CUDA/OpenCL for execution on GPUs.

The main focus of OpenSBLI is on the solution of the compressible Navier-Stokes equations with application to shock-boundary layer interactions (SBLI). However, in principle, any set of equations that can be written in Einstein notation may be solved.

6.4.1 Test Case. The benchmark test case setup using OpenSBLI is the Taylor-Green vortex problem in a cubic domain of length 2π. For this study, we have investigated the strong scaling properties

for the benchmark on grids of sizes 512×512×512 and 1024×1024×1024. The larger benchmark should be more demanding in terms of both computation and memory access. The smaller benchmark is included to allow comparisons between single nodes of different architectures as the larger benchmark requires a minimum of 2 compute nodes. This benchmark performs minimal I/O.

6.4.2 Configuration, Compilation and Libraries. Performance is measured in 'iterations/s'. The total runtime and number of iterations are read directly from the OpenSBLI output and these are used to compute the number of iterations per second.

For all systems, 1 MPI process was used per physical core with 1 OpenMP thread per MPI process. A single hardware thread was used in all cases.

6.5 Nektar++

Nektar++ [8] is a tensor-product based finite-element framework designed to allow construction of efficient, classical, low polynomial order h-type solvers (where h is the size of the finite-element) as well as higher p-order piece-wise polynomial order solvers. Written in C++, Nektar++ is run here as pure-MPI code but can run in a hybrid MPI-OpenMP mode, can make use of hyperthreading and supports use of PGAS communications models via extensions to it's communications library. It is under active development by Imperial College London and the University of Utah.

6.5.1 Test Case. The geometry used for this test case is a 3D segment of a rabbit's descending aorta with two pairs of intercostal arteries branching off. The inlet has a diameter $D = 3.32mm$. The solver used is the incompressible Navier Stokes solver and run for 4000 timesteps of 1 μs giving 4 ms simulated runtime with a backward Euler time integration method. The output is limited to a single write of the flow fields at the end of the simulation.

6.5.2 Configuration, Compilation and Libraries. Performance is measured from the code's "time integration" output which maps to the time spent running the solver routines for the test case. For both systems tested, a single MPI process per physical core was used without shared memory processing; this was run as a purely MPI code to limit the differences between systems.

7 RESULTS AND EVALUATION

This section presents and evaluates the benchmarking applications on a range of node counts to investigate both performance and scalability across the different systems. For HPCG, COSA, OpenSBLI and GROMACS, the HPE Apollo 70 results are contrasted with all three comparison platforms. Nektar++ is compared to the SGI ICE XA only. At the end of this section, we present our evaluation of portability and the Arm software stack maturity.

7.1 HPCG

HPCG is considered to be a more representative HPC benchmark than HPL as it has a more complex resource usage pattern, more akin to real HPC applications. As such, HPCG performance will take into account memory bandwidth, floating point performance and to some extent network performance.

Table 9 shows the HPCG performance for a single node across the test systems. The Dell EMC system was tested with different compilation options targeting the wide vectorisation functionality. The table demonstrates that a significant performance benefit ($\tilde{2}0\%$) is achieved on the Skylake processors in the Dell EMC system when specifically targeting the AVX-512 instruction set. The table also demonstrates that the Apollo 70 nodes are performing very well, providing around 2.7% of theoretical total GFlop/s of a node compared to 1.3% for the Dell EMC system (with Intel Skylake processors) and 1.7% for the ICE XA system (with Intel Broadwell processors). The XC30 system has slightly better single node performance (3.0% of theoretical GFlop/s), albeit with a much lower total floating point capability than the other systems compared.

Table 9: Single node HPCG GFlop/s.

System	Performance
HPE Apollo 70	30.529
SGI ICE XA	21.115
Cray XC30	15.650
Dell EMC (AVX512)	34.581
Dell EMC (AVX2)	28.120

Table 10: Multi-node HPCG GFlop/s.

Nodes	HPE Apollo 70	SGI ICE XA	Cray XC30	Dell EMC (AVX2)
2	61.185	40.491	32.048	56.505
4	120.972	80.317	61.617	111.598
8	241.363	158.774	120.749	214.358
16	482.202	305.534	245.870	400.320
32	939.109	602.092	491.101	747.066

Table 10 presents HPCG results when using multiple nodes. Problems with running the AVX-512 version of the benchmark on the Dell EMC system have restricted us to providing only the AVX2 numbers. We see that the Apollo 70 maintains nearly twice the total GFlop/s when compared to the Cray XC30 across a range of node counts, and outperforms both the SGI ICE XA and Dell EMC system. If we were to assume linear scaling for the AVX-512 results from a single node up to 32 nodes on the Dell EMC system, this translates to around 10% more GFlop/s than the Apollo 70.

We undertook a Pearson correlation analysis of HPCG performance on the various systems against the theoretical peak floating point performance per node, the measured memory bandwidth per node (both detailed in Table 1), the inter-node Sendrecv latency and bandwidth (both detailed in Table 6). The Pearson correlation analysis assesses the level of correlation between the values from two datasets, reporting correlation that varies between -1 (absolute negative correlation) and +1 (absolute positive correlation). Whilst we only have a limited number of data points in our datasets, the correlation analysis does provide some indications of the impact of various performance characteristics of the various systems. The results of this analysis, detailed in Table 11, demonstrate the impact of aggregate memory bandwidth on multi-node HPCG performance, and the importance of floating point capability for the single node results. It is also noted that MPI message latency seems to have a reasonable correlation with the multi-node HPCG performance.

Table 11: HPCG Correlation values with theoretical peak GFlop/s per node, measured memory bandwidth per node, off-node MPI latency and bandwidth.

Nodes	Node DP GFlop/s	Memory bandwidth	MPI latency	MPI bandwidth
1	0.83	0.76	0.42	0.25
8	0.56	0.95	0.74	-0.14

7.2 COSA

Table 14 presents the performance of COSA for each of the four systems under consideration using the benchmark outlined in section 6.2.1. The number of MPI processes used in each of the tests is listed in Table 12. Each benchmark was run three times and the fastest run is presented. For all the benchmarks the variation between the best and worst is less than 2% of the best runtime.

Table 12: COSA benchmark MPI process counts.

Nodes	HPE Apollo 70	SGI ICE XA	Cray XC30	Dell EMC
4	256	144		
8	512	288		256
16	1024	576		512
21			512	
22			528	
32	2048	1152		1024
40			960	

It is important to note that Table 14 outlines the performance as the number of nodes increases. With reference to Table 12 it can be seen that this equates to different numbers of MPI processes on the different systems. For instance, using 512 MPI processes on the Cray XC30 system gives better performance than on the HPE Apollo 70 system, but at the cost of using significantly more nodes (i.e. 21 nodes vs 8 nodes). If we compare using MPI process counts across the systems it is evident that the performance is generally similar with the SGI ICE XA exhibiting the best performance at lower process counts, although it does not maintain this advantage at higher process counts.

However, if we compare on a node to node basis, it is evident that the performance is generally similar between the HPE Apollo 70 and the Dell EMC systems, with the SGI ICE XA and Cray XC30 not providing as quick a time to solution for similar numbers of nodes. Indeed, the Dell EMC and HPE Apollo 70 systems exhibit around 50% increased performance compared to the SGI ICE XA system. The ability to achieve comparable performance using significantly lower numbers of nodes provides potentially lower costs in terms of hardware and energy.

As with the HPCG benchmarks, we undertook a correlation of COSA performance at 8 and 32 nodes on the Apollo 70, ICE XA, and Dell EMC systems (we did not have results from the Cray XC30 for these node counts) against theoretical floating point performance and measured memory bandwidth. Table 13 shows that COSA has a strong correlation with available memory bandwidth (the negative correlation observed is because we are comparing time to solution numbers to performance metrics, which have an inverse relationship, as time to solution decreases as performance metrics like memory bandwidth increase).

Table 13: COSA Correlation values with theoretical peak GFlop/s per node and measured memory bandwidth per node.

Nodes	Node DP GFlop/s	Node memory bandwidth
8	-0.34	-0.86
32	-0.37	-0.86

Table 14: COSA performance. On the Cray XC30, the test case cannot be run on the same number of nodes as on the other systems, but the results are nevertheless presented as a relative indicator of overall performance. The Dell EMC system does not have enough memory per node to run at fewer than 8 nodes.

	Walltime in seconds *(Speedup factor)*			
Nodes	**HPE Apollo 70 (MPT)**	**SGI ICE XA**	**Cray XC30**	**Dell EMC**
4	2036 *(1.00)*	3159 *(1.00)*		
8	1005 *(2.03)*	1591 *(1.99)*		1083 *(1.00)*
16	484 *(4.21)*	771 *(4.10)*		540 *(2.01)*
21			896 *(1.00)*	
22			873 *(1.03)*	
32	239 *(8.52)*	496 *(6.37)*		265 *(4.09)*
40			482 *(1.86)*	

7.3 GROMACS

Table 15 shows the performance of GROMACS for each of the systems as a function of number of nodes for the 1400k atom benchmark.

The ordering of the GROMACS performance on the different systems up to 32 compute nodes matches the ordering of compute node performance. It is well known [5, 18] that GROMACS performance at low process counts for the simulation size is well-correlated with floating point performance so this behaviour is as expected. The majority of MPI time for this GROMACS benchmark is spent in point-to-point communication rather than collective communication. This is underlined by the fact that the scaling behaviour from GROMACS on the Apollo 70 is comparable with that on the other systems, and indeed shows the best scaling performance for the higher node counts. Further investigations is needed to establish what hardware feature(s) are enabling this improved scaling performance, but it could be attributed to the ratio of on-node and off-node MPI communications that the large core counts on the ThunderX2 based systems facilitate.

7.4 OpenSBLI

Table 16 shows the performance of OpenSBLI for each of the systems as a function of number of nodes for the $512\times512\times512$ benchmark and Table 17 shows the performance for the $1024\times1024\times1024$ benchmark.

Table 15: GROMACS performance results, shown both as the total wallclock time in seconds, together with parallel efficiency, and the application's own performance metric (ns/day), together with the speedup factor for increasing node counts.

	Walltime in seconds *(Parallel efficiency %)*				Performance in ns/day *(Speedup factor)*			
Nodes	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC
1	1340.54 *(100%)*	1685.88 *(100%)*	1017.40 *(100%)*	696.54 *(100%)*	1.29 *(1.00)*	1.03 *(1.00)*	1.70 *(1.00)*	2.48 *(1.00)*
2	681.69 *(98%)*	826.78 *(102%)*	523.95 *(97%)*	353.66 *(98%)*	2.54 *(1.97)*	2.09 *(2.04)*	3.30 *(1.94)*	4.89 *(1.97)*
4	345.09 *(97%)*	452.64 *(93%)*	275.21 *(92%)*	183.59 *(95%)*	5.01 *(3.89)*	3.82 *(3.72)*	6.28 *(3.70)*	9.41 *(3.79)*
8	178.49 *(94%)*	246.71 *(85%)*	140.86 *(90%)*	101.29 *(86%)*	9.68 *(7.51)*	7.01 *(6.83)*	12.27 *(7.22)*	17.06 *(6.88)*
16	93.50 *(90%)*	169.33 *(62%)*	72.55 *(88%)*	64.76 *(67%)*	18.48 *(14.34)*	10.21 *(9.96)*	23.82 *(14.02)*	26.69 *(10.76)*
32	62.85 *(67%)*	96.56 *(55%)*	69.12 *(46%)*	35.68 *(61%)*	27.50 *(21.33)*	17.90 *(17.46)*	25.00 *(14.72)*	48.44 *(19.53)*

For the smaller of the two benchmarks, the Cray XC30 is the lowest performing in terms of absolute runtime; the SGI ICE XA and HPE Apollo systems are very evenly matched. The Dell system shows the best outright runtime performance. The test case scales well on all systems, achieving 27-28x speedup on 32 nodes (compared to a single node), and the parallel efficiency is at or above 85%.

The larger benchmark requires a minimum of 2 nodes to satisfy its memory requirements (4 nodes on the XC30). It shows very similar performance trends, although the Apollo 70 system now marginally outperforms the SGI ICE XA (on the order of a few %). As scalability for this test case is even better than for the smaller test case (in fact it is mostly superlinear), the network and communication patterns are evidently not a performance bottleneck here. Taking into account the memory requirements of this test case, as well as the fact that the ThunderX2 CPU does not outperform an Intel Xeon Broadwell CPU on a per-core basis in terms of pure compute performance, the gain for the Apollo70 almost certainly stems from the increased memory bandwidth the Arm-based processor offers.

7.5 Nektar++

Table 18 shows the performance of Nektar++ running on both the SGI ICE XA and HPE Apollo 70 systems, represented as runtime to solution, and showing both the parallel efficiency and the speedup relative to one node. Comparing the performance on a per-node basis, the Apollo 70 system displays better performance than the SGI ICE XA at low node counts, however the gap closes at 8 nodes and the Arm-based system falls behind. One potential explanation is that the test-case does not provide sufficient work which limits the performance with higher core-count on the ARM based system. Unfortunately, a larger test-case was not available at the time of writing. It is also evident that the ARM compiler is providing performance benefits at two to eight nodes when compared to the GNU compiler. This, coupled with the super-linear speed up in those regions, suggests that the ARM compiler is better utilising the cache compared to the GNU compiler.

7.6 Application Portability and Maturity of the Software Stack

The HPE Apollo 70 system offers a range of compilers, including the GNU and Arm compiler suites, supporting C, C++, and Fortran applications. These are complemented by the Arm performance libraries [3], which provide optimised implementations of BLAS, LAPACK, FFT and standard maths routines. There are also a number of different MPI implementations available for use on the system, including OpenMPI and MPICH, with MVAPICH2 shortly to be available. The availability of this combination of libraries and compilers has made porting the applications we have benchmarked for this paper straightforward, requiring no code modification, and only simple adaptations of the build processes for the applications.

The software stack that is available on the Apollo 70 is mature and no major problems were encountered when building the applications or evaluating the benchmarks. The only performance issue that we discovered is the disappointing performance of the MPI collective operations, and this is under active investigation. All in all, we found the software stack to be complete and sufficiently mature to be able to build and run complex real-life applications without difficulties. It is worth noting however that the Apollo 70 system used for the experiments in this paper does not have access to a high performance parallel file system yet (this part of the system is still under development), and the benchmarks therefore do not include any IO; as such we cannot assess that aspect of the system.

Finally, although we did not encounter any significant issues when porting applications to the Arm-based system, the same is not true for performance profiling, in particular with regards to memory access and usage. The tools that are available on the Apollo 70 focus largely on CPU and network performance. However, as discussed above, the additional memory channels available on the Apollo 70 are key to delivering improved performance for applications that are limited by memory bandwidth, but there are no profiling tools available on the system that would allow us to quantify the effect directly. This is a limitation that should be addressed in order to allow application developers to fully understand the performance of their codes in the Arm ecosystem.

8 CONCLUSIONS

This paper gives a first overview of the performance of scientific applications running at scale on a production HPC system that is based on Arm processors. Our experience gained from porting a wide range of applications to the ThunderX2 processors and using the Arm supported software ecosystem demonstrates that this is possible with minimal effort. We have also demonstrated that applications can achieve similar, or better, performance on such a system when compared with a range of existing HPC system architectures, as highlighted by Figure 1. Due to the additional

Table 16: OpenSBLI performance for TGV512ss (100 iterations).

	Walltime in seconds *(Parallel efficiency %)*				Performance in iterations/second *(Speedup factor)*			
Nodes	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC
1	793.65 *(100%)*	999.91 *(100%)*	739.89 *(100%)*	509.92 *(100%)*	0.13 *(1.00)*	0.10 *(1.00)*	0.14 *(1.00)*	0.20 *(1.00)*
2	375.94 *(106%)*	488.33 *(102%)*	386.07 *(96%)*	257.84 *(99%)*	0.27 *(2.11)*	0.20 *(2.05)*	0.26 *(1.68)*	0.39 *(1.99)*
4	207.04 *(96%)*	288.80 *(87%)*	218.82 *(85%)*	131.90 *(97%)*	0.48 *(3.83)*	0.35 *(3.46)*	0.46 *(3.29)*	0.76 *(3.88)*
8	105.60 *(94%)*	142.35 *(88%)*	101.65 *(91%)*	68.05 *(94%)*	0.95 *(7.51)*	0.70 *(7.02)*	0.98 *(7.00)*	1.47 *(7.50)*
16	53.05 *(94%)*	65.85 *(95%)*	51.93 *(89%)*	34.26 *(93%)*	1.89 *(14.96)*	1.52 *(15.18)*	1.93 *(13.79)*	2.92 *(14.89)*
32	28.59 *(87%)*	35.09 *(89%)*	26.75 *(86%)*	18.64 *(85%)*	3.50 *(27.76)*	2.85 *(28.50)*	3.74 *(26.71)*	5.36 *(27.33)*

Table 17: OpenSBLI performance for TGV1024ss (10 iterations). The memory requirements of this test case are such that a minimum of 2 nodes are needed for all systems (apart from the Cray XC30, which needs 4 nodes). The efficiency and speedup factors are provided relative to the lowest possible node count for each system.

	Walltime in seconds *(Parallel efficiency %)*				Performance in iterations/second *(Speedup factor)*			
Nodes	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC	HPE Apollo 70	Cray XC30	SGI ICE XA	Dell EMC
2	312.50 *(100%)*		309.33 *(100%)*	213.47 *(100%)*	0.03 *(1.00)*		0.03 *(1.00)*	0.05 *(1.00)*
4	158.73 *(98%)*	301.07 *(100%)*	174.44 *(89%)*	105.91 *(101%)*	0.06 *(1.97)*	0.03 *(1.00)*	0.06 *(2.00)*	0.09 *(2.02)*
8	80.00 *(98%)*	104.95 *(143%)*	75.88 *(102%)*	53.14 *(100%)*	0.12 *(3.91)*	0.09 *(2.87)*	0.13 *(4.33)*	0.19 *(4.00)*
16	30.02 *(130%)*	49.10 *(153%)*	39.05 *(99%)*	25.05 *(107%)*	0.33 *(10.41)*	0.20 *(6.13)*	0.26 *(8.67)*	0.40 *(8.52)*
32	20.80 *(94%)*	28.27 *(133%)*	22.19 *(87%)*	13.14 *(102%)*	0.48 *(15.02)*	0.35 *(10.65)*	0.45 *(15.00)*	0.76 *(16.25)*

Table 18: Nektar++ performance.

	Walltime in seconds *(Parallel Efficiency & Speedup factor)*		
Nodes	HPE Apollo 70 (GNU+OpenMPI)	HPE Apollo 70 (ARM+OpenMPI)	SGI ICE XA
1	1751 *(100% & 1.00)*	1800 *(100% & 1.00)*	2621 *(100% & 1.00)*
2	1362 *(64% & 1.29)*	842 *(107% & 2.14)*	1227 *(107% & 2.13)*
4	466 *(94% & 3.76)*	364 *(124%& 4.94)*	502 *(131% & 5.22)*
8	185 *(118% & 9.46)*	172 *(122% & 10.46)*	159 *(206% & 16.48)*
16	109 *(100% & 16.06)*	109 *(103% & 16.51)*	88 *(186% & 20.45)*

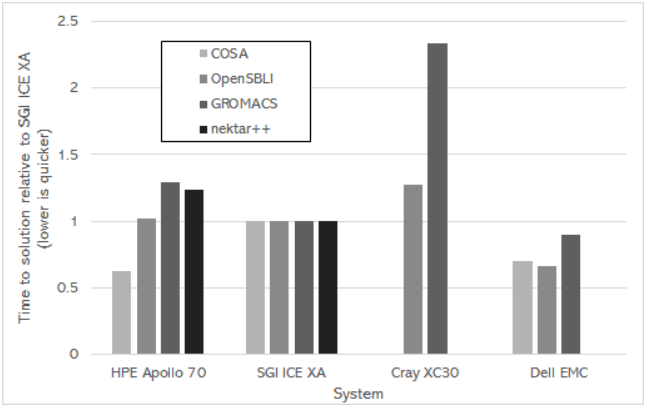

Figure 1: Comparison of application time to solution using 16 nodes normalised to the SGI ICE XA system. Values lower than one are faster than the SGI ICE XA system.

memory channels available to them, the ThunderX2 processors provide clear benefits for applications that are memory bandwidth bound. However, when comparing on a per-node basis, the Apollo 70 also exhibits good performance on applications that are compute rather than memory intensive. Our conclusion is that the Arm ecosystem, both the hardware and the softwar stack, is already proving itself to be a viable alternative to the status quo.

9 ACKNOWLEDGEMENTS

The Fulhame HPE Apollo 70 system is supplied to EPCC as part of the Catalyst UK programme, a collaboration with Hewlett Packard Enterprise, Arm and SUSE to accelerate the adoption of Arm based supercomputer applications in the UK.

This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1).

This work used the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

We thank the University of Cambridge for access to resources provided by the Cambridge Service for Data Driven Discovery (CSD3). CSD3 is operated by the University of Cambridge Research Computing Service (http://www.csd3.cam.ac.uk/), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk).

REFERENCES

[1] [n. d.]. ARCHER Benchmarks. Retrieved 10 January 2019 from http://www.github.com/hpc-uk/archer-benchmarks

[2] [n. d.]. ARCHER usage data. Retrieved 10 January 2019 from https://www.archer.ac.uk/status/codes

[3] [n. d.]. Arm Performance Libraries. Retrieved 15 January 2019 from https://developer.arm.com/products/software-development-tools/hpc/arm-performance-librariess

[4] [n. d.]. HEC BioSim Benchmarks. Retrieved 11 January 2019 from http://www.hecbiosim.ac.uk/benchmarks

[5] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (2015), 19 – 25. https://doi.org/10.1016/j.softx.2015.06.001

[6] David Bailey, John Barton, Thomas Lasinski, and Horst Simon. 1993. The NAS parallel benchmarks. (08 1993). https://doi.org/10.2172/983318

[7] Mont Blanc. [n. d.]. D9.6 Performance analysis of applications and mini-applications and benchmarking on the project test platforms. Retrieved March 2019 from https://www.montblanc-project.eu/project/deliverables

[8] CD Cantwell, D Moxey, A Comerford, A Bolis, G Rocco, G Mengaldo, D De Grazia, S Yakovlev, J-E Lombard, and D Ekelschot et al. 2015. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications* 192 (2015), 205–219.

[9] Jack Dongarra and Piotr Luszczek. 2004. Introduction to the HPCChallenge Benchmark Suite. (12 2004).

[10] Jack J. Dongarra, Michael A. Heroux, and Piotr Luszczek. 2015. HPCG Benchmark : a New Metric for Ranking High Performance Computing Systems âĹŰ.

[11] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. 2003. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience* 15 (2003), 803–820.

[12] MPI Forum. [n. d.]. MPI: A Message-Passing Interface Standard, Version 2.2. Retrieved September 2009 from http://www.mpi-forum.org

[13] Adrian Jackson and M. Sergio Campobasso. 2011. Shared-memory, distributed-memory, and mixed-mode parallelisation of a CFD simulation code. *Computer Science - Research and Development* 26, 3-4 (6 2011), 187–195. https://doi.org/10.1007/s00450-011-0162-4

[14] William Jackson, M. Sergio Campobasso, and Jernej Drofelnik. 2018. Load balance and Parallel I/O: Optimising COSA for large simulations. *Computers and Fluids* (9 3 2018). https://doi.org/10.1016/j.compfluid.2018.03.007

[15] C. T. Jacobs, S. P. Jammy, and N. D. Sandham. 2017. OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures. *Journal of Computational Science* 18 (2017), 12–23. https://doi.org/10.1016/j.jocs.2016.11.001

[16] Simon McIntosh-Smith, James Price, Tom Deakin, and Andrei Poenaru. 2018. Comparative Benchmarking of the First Generation of HPC-Optimised Arm Processors on Isambard. In *Cray User Group*.

[17] Matthias MÃijller, Matthijs van Waveren, Ron Lieberman, Brian Whitney, Hideki Saito, Kalyan Kumaran, John Baron, William Brantley, Chris Parrott, Tom Elken, Huiyu Feng, and Carl Ponder. 2010. SPEC MPI2007-an application benchmark suite for parallel systems using MPI. *Concurrency and Computation: Practice and Experience* 22 (02 2010), 191–205. https://doi.org/10.1002/cpe.1535

[18] Andrew Turner and Jeffrey Salmond. 2018. hpc-uk/archer-benchmarks: Initial performance comparison report. https://doi.org/10.5281/zenodo.1288378