Received: 18 June 2018 | Revised: 27 November 2018 | Accepted: 27 November 2018
DOI: 10.1002/cpe.5110

WILEY

SPECIAL ISSUE PAPER

A performance analysis of the first generation of HPC-optimized Arm processors

Simon McIntosh-Smith | James Price | Tom Deakin | Andrei Poenaru

High Performance Computing Research Group, Department of Computer Science, University of Bristol, Bristol, UK

Correspondence
Simon McIntosh-Smith, High Performance Computing Research Group, Department of Computer Science, University of Bristol, Tyndall Avenue, Bristol BS8 1TH, UK.
Email: S.McIntosh-Smith@bristol.ac.uk

Funding information
Office of Science of the U.S. Department of Energy, Grant/Award Number: DE-AC05-00OR22725; Engineering and Physical Sciences Research Council, Grant/Award Number: EP/P020224/1

Summary

In this paper, we present performance results from Isambard, the first production supercomputer to be based on Arm CPUs that have been optimized specifically for HPC. Isambard is the first Cray XC50 "Scout" system, combining Cavium ThunderX2 Arm-based CPUs with Cray's Aries interconnect. The full Isambard system will be delivered in the summer of 2018, when it will contain over 10 000 Arm cores. In this work, we present node-level performance results from eight early-access nodes that were upgraded to B0 beta silicon in March 2018. We present node-level benchmark results comparing ThunderX2 with mainstream CPUs, including Intel Skylake and Broadwell, as well as Xeon Phi. We focus on a range of applications and mini-apps important to the UK national HPC service, ARCHER, as well as to the Isambard project partners and the wider HPC community. We also compare performance across three major software toolchains available for Arm: Cray's CCE, Arm's version of Clang/Flang/LLVM, and GNU.

KEYWORDS

Arm, benchmarking, HPC, ThunderX2, XC50

1 | INTRODUCTION

The development of Arm processors has been driven by multiple vendors for the fast-growing mobile space, resulting in the rapid innovation of the architecture, greater choice for system companies, and competition between vendors. This market success inspired the European FP7 Mont-Blanc project to explore Arm processors designed for the mobile space, investigating the feasibility of using the Arm architecture for workloads relevant to the HPC community. Mont-Blanc's early results were encouraging, but it was clear that chips designed for the mobile space could not compete with HPC-optimized CPUs without further architecture and implementation developments. Since Mont-Blanc, Cavium announced it will release its first generation of HPC-optimized, Arm-based CPUs in 2018.

In response to these developments, the Isambard* project was set up to provide the world's first production Arm-based supercomputer. Isambard will be a Cray XC50 "Scout" system and will be run as part of Tier 2 within UK's national integrated HPC ecosystem.† Cavium ThunderX2 processors will form the basis of the system; these processors use the Armv8 instruction set but have been optimized specifically for HPC workloads. ThunderX2 CPUs are noteworthy in their focus on delivering class-leading memory bandwidth: each 32-core CPU uses eight DDR4 memory channels, enabling a dual-socket system to deliver in excess of 250 GB/s of memory bandwidth. The Isambard system represents a collaboration between the GW4 Alliance (formed from the universities of Bristol, Bath, Cardiff, and Exeter) along with UK's Met Office, Cray, Arm, and Cavium, and funded by EPSRC. Although Isambard is due to arrive in July 2018, this paper will present results using the project's early-access nodes, delivered in October 2017 and upgraded from A1 to B0 beta silicon in March 2018. These results will be among the first published for the near-production silicon of the Arm-based Cavium ThunderX2 processor and the first using Cray's CCE tools for Arm.

*http://gw4.ac.uk/isambard/
†https://epsrc.ukri.org/research/facilities/hpc/tier2/

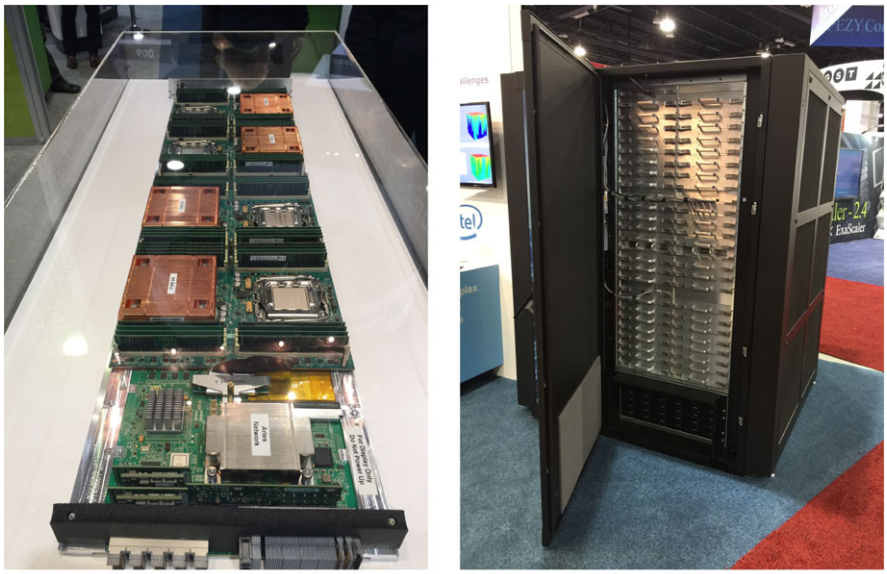

FIGURE 1 Isambard hardware. Pictures © Simon McIntosh-Smith, taken at SC'17. A, An XC50 Scout blade; B, An XC50 cabinet

TABLE 1 Hardware information (peak figures)

Processor	Cores	Clock Speed, GHz	TDP, W	FP64, TFLOP/s	Bandwidth, GB/s
Broadwell	2 × 22	2.2	145	1.55	154
Skylake Gold	2 × 20	2.4	150	3.07	256
Skylake Platinum	2 × 28	2.1	165	3.76	256
ThunderX2	2 × 32	2.2	175	1.13	320

The results we shall present will focus on single-node, dual-socket performance in comparison to other state-of-the-art processors found in the majority of supercomputers today. These results will also lay the foundations for a future study of at-scale, production-style workloads running on Isambard, which will utilize the Cray Aries interconnect.

The top 10 most heavily used codes that are run on UK's national supercomputer, ARCHER[‡] (an Intel x86-based Cray XC30 system), along with a selection of mini-apps, proxy applications, and applications from project partners, provide good representation of the styles of codes used by today's HPC community.[1] As such, they provide an ideal vehicle for which to benchmark the new Cavium ThunderX2 architecture, and the Isambard system presents a unique opportunity for such comparative benchmarking. Its heterogeneous design enables direct comparison between Cavium ThunderX2 CPUs and the best of today's mainstream HPC hardware, including x86 Intel Xeon and Xeon Phi processors, and NVIDIA P100 GPUs.

2 | ISAMBARD: SYSTEM OVERVIEW

The most exciting aspect of the Isambard system will be the full cabinet of XC50 "Scout" with Cavium ThunderX2, delivering 10 752 high-performance Armv8 cores. Each node includes two 32-core ThunderX2 processors running at 2.1 GHz. The processors each have eight 2666-MHz DDR4 channels, yielding a STREAM triad result of over 250 GB/s per node. The XC50 Scout system packs four dual-socket nodes into each blade and then 42 such blades into a single cabinet. Pictures of a Scout blade and an XC50 cabinet are shown in Figure 1.

The results presented in this paper are based on work performed at two hackathons, using the Isambard early-access nodes. These early-access nodes use the same Cavium ThunderX2 CPUs as the full Isambard XC50 system, but in a Foxconn whitebox form factor. These nodes were upgraded from A1 to B0 beta silicon in March 2018 and were run in a dual-socket, 32-core, 2.2-GHz configuration for the purpose of gathering the data presented herein. Each node includes 256 GB of DDR4 DRAM and runs SLES 12 SP3, with a subset of the Cray Programming Environment (CPE). This subset of CPE includes all the elements we needed for benchmarking the ThunderX2 CPUs: the Cray Compiler, CCE, performance libraries, and analysis tools. In addition, we used Arm's LLVM-based compiler and GNU. It should be noted that all of these compilers are still relatively early in their support for HPC-optimized Arm CPUs, given that, at the time of this work, production hardware has yet to ship. Details of which versions of these tools were used are given in Table 2.

[‡]https://www.archer.ac.uk

3 | BENCHMARKS

3.1 | Mini-apps

In this section, we give a brief introduction to the mini-apps used in this study. The mini-apps themselves are all performance proxies for larger production codes, encapsulating the important performance characteristics, such as floating-point intensity, memory access, and communication patterns of their parent application, but without the complexities that are often associated with "real" codes. As such, they are useful for performance modeling and algorithm characterization and can demonstrate the potential performance of the latest computer architectures.

STREAM: McCalpin's STREAM has long been the gold-standard benchmark for measuring the achievable sustained memory bandwidth of CPU architectures.[2] The benchmark is formed of simple element-wise arithmetic operations on long arrays (vectors), and for this study, we consider the Triad kernel of $a(i) = b(i) + \alpha c(i)$. The achieved memory bandwidth is easily modeled as three times the length of the arrays divided by the fastest runtime for this kernel. Arrays of 2^{25} double-precision elements were used in this study, with the kernels run 200 times.

CloverLeaf: This hydrodynamics mini-app solves Euler's equations of compressible fluid dynamics, under a Lagrangian-Eulerian scheme, on a two-dimensional spatial regular structured grid.[3] These equations model the conservation of mass, energy, and momentum. The mini-app is an example of a stencil code and is classed as memory bandwidth bound. CloverLeaf is regularly used to study performance portability on many different architectures.[4] The `bm_16` test case consists of a grid of 3840 $\times$ 3840 cells.

TeaLeaf: This heat diffusion mini-app solves the linear heat conduction equation on a spatially decomposed regular grid, utilizing a five-point finite difference stencil.[5] A range of linear solvers are included in the mini-app, but the baseline method we use in this paper is the matrix-free conjugate gradient (CG) solver. TeaLeaf is memory bandwidth bound at the node level, but, at scale, the solver can become bound by communication. For this study, the focus is on single-node performance, and so, we used the `bm_5` input deck, which utilizes a 4000 $\times$ 4000 spatial grid.

SNAP: This is a proxy application for a modern deterministic discrete ordinates transport code.[6] As well as having a large memory footprint, this application has a simple finite difference kernel that must be processed according to a wavefront dependency, which introduces associated communication costs. The kernel itself is limited by the performance of the memory hierarchy.[7] Benchmarking SNAP is somewhat challenging, as the spatial problem size must divide evenly by the number of MPI ranks (often core count). Therefore, a fixed problem size per core was used for this study with a spatial grid of 1024 $\times$ 2 $\times$ 2 per MPI rank, 136 angles per octant, and 32 groups, with one MPI rank per physical core. The footprint of the angular flux solution in this case is just over 1 GB, which is roughly in line with typical ARCHER usage.[1] The figure of merit for this application is the *grind time*, which is a normalized value taking into account differing problem sizes.

Neutral: The Monte Carlo neutral particle transport mini-app, Netural, was written to explore the challenges of such algorithms on advanced architectures, disproving the old adage that Monte Carlo codes are "embarrassingly parallel."[8] The mini-app takes a mesh-based approach, and so, there is a tension with regard to the memory access of the mesh and particle data, resulting in memory latency becoming the performance-limiting factor. The `csp` input included with the mini-app was used for this study, and the benchmark was run using OpenMP.

3.2 | Applications

The Isambard system has been designed to explore the feasibility of an Arm-based system for real HPC workloads. As such, it is important to ensure that the most heavily used codes are tested and evaluated. To that end, eight real applications have been selected for this study, taken from the top 10 most used codes on ARCHER, UK's national supercomputer.[1] These applications represent over 50% of the usage of the whole supercomputer. Therefore, the performance of these codes on any architecture captures the interests of a significant fraction of UK HPC users, and any change in the performance of these codes directly from the use of different architectures is important to quantify. The test cases were chosen by the group of core application developers and key application users who came to two Isambard hackathon runs in October 2017 and February 2018; details of the attendees are found in our acknowledgments at the end of this paper. Given that we had to focus on comparing the performance of single nodes, we had to choose test cases that were of scientific merit, yet could run in a reasonable time on a single node.

CP2K[§]: This code simulates the ab initio electronic structure and molecular dynamics of different systems such as molecular, solids, liquids, and so on. Fast Fourier transforms (FFTs) form part of the solution step, but it is not straightforward to attribute these as the performance-limiting factor of this code. The memory bandwidth of the processor and the core count both have an impact. The code already shows sublinear scaling up to tens of cores. Additionally, the performance of the code on a single node does not necessarily have the same performance-limiting factors as when running the code at scale. We will need the full Isambard XC50 system to test these effects; in the meantime, we have used the `H2O-64` benchmark, which simulates 64 water molecules (consisting of 192 atoms and 512 electrons) in a 12.4 Å^3 cell for 10 time steps. This is an often studied benchmark for CP2K and, therefore, provides sufficient information to explore the performance across the different architectures in this paper.

[§]https://www.cp2k.org

GROMACS[¶]: This molecular dynamics package is used to solve Newton's equations of motion. Systems of interest such as proteins contain up to millions of particles. It is thought that GROMACS is bound by the floating-point performance of the processor architecture. This has motivated the developers to handwrite vectorized code in order to ensure an optimal sequence of such arithmetic.[9] The hand-optimized code is written using vector intrinsics, which results in GROMACS not supporting some compilers—such as the Cray Compiler—because they do not implement all the required intrinsics. For each supported platform, computation is packed so that it saturates the native vector length of the platform, eg, 256 bits for AVX2, 512 bits for AVX-512, and so on. For this study, we used the `ion_channel_vsites` benchmark.[#] This consists of the membrane protein GluCl, containing around 150 000 atoms (which, for GROMACS, is typically small), and uses "vsites" and a five-femtosecond time step. On the ThunderX2 processor, we used the `ARM_NEON_ASIMD` vector implementation, which is the closest match for the Armv8.1 architecture. However, this implementation is not as mature as some of those targeting x86.

NAMD: This molecular dynamics simulation program is designed to scale up to millions of atoms.[10] It is able to achieve scaling to more than half a million processor cores using the Charm++ parallel programming framework, a high-level library that abstracts the mapping of processors to work items—or *chares*—away from the programmer.[11] The test case used is the STMV benchmark, which simulates one of the smallest viruses in existence and which is a common set of inputs for measuring scaling capabilities. This benchmark includes PME calculations, which use FFTs, and so, its performance is heavily influenced by that of the FFT library used. Due to the complex structure of atomic simulation computation and the reliance of distributed FFTs, it is hard to define a single bounding factor for NAMD's performance—compute performance, memory bandwidth, and communication capabilities all affect the overall performance of the application.

NEMO: The Nucleus for European Modelling of the Ocean[‖] (NEMO) code is one ocean modeling framework used by UK's Met Office and is often used in conjunction with the Unified Model atmosphere simulation code. The code consists of simulations of the ocean, sea ice, and marine biogeochemistry under an automatic mesh refinement scheme. As a structured grid code, the performance-limiting factor is highly likely to be memory bandwidth. The benchmark we used was derived from the `GYRE_PISCES` reference configuration, with a $^1/_{12}\circ$ resolution and 31 model levels, resulting in 2.72M points, running for 720 time steps.

OpenFOAM: Originally developed as an alternative to early simulation engines written in Fortran, OpenFOAM is a modular C++ framework aiming to simplify writing custom computational fluid dynamics (CFD) solvers. The code makes heavy use of object-oriented features in C++, such as class derivation, templating and generic programming, and operator overloading, which enables a powerful, extensible design methodology.[12] OpenFOAM's flexibility comes at the cost of additional code complexity, of which a good example is how every component is compiled into a separate dynamic library, which are then combined at runtime based on the user's input to form the required executables. The two features mentioned above grant OpenFOAM a unique position in our benchmark suite. In this paper, we use the `simpleFoam` solver for incompressible, turbulent flow from version 1712 of OpenFOAM,[**] the most recent release at the time of writing. The input case is based on the RANS DrivAer generic car model, which is a representative case of real aerodynamics simulation and, thus, should provide meaningful insight of the benchmarked platforms' performance.[13] OpenFOAM is almost entirely memory bandwidth bound.

OpenSBLI: This is a grid-based finite difference solver[††] used to solve compressible Navier-Stokes equations for shock-boundary layer interactions. The code uses Python to automatically generate code to solve the equations expressed in mathematical Einstein notation and uses the Oxford Parallel Structured (OPS) software for parallelism. As a structured grid code, it should be memory bandwidth bound under the Roofline model, with low computational intensity from the finite difference approximation. We used the ARCHER benchmark for this paper,[‡‡] which solves a Taylor-Green vortex on a grid of $1024 \times 1024 \times 1024$ cells (around a billion cells).

Unified Model: The UK's Met Office code, the Unified Model[§§] (UM), is an atmosphere simulation code used for weather and climate applications. It is often coupled with the NEMO code. The UM is used for weather prediction, seasonal forecasting, and climate modeling, with timescales ranging from days to hundreds of years. At its core, the code solves the compressible nonhydrostatic motion equations on the domain of the Earth discretized into a latitude-longitude grid. As a structured grid code, the performance-limiting factor is highly likely to be memory bandwidth. We used an `AMIP` benchmark[14] provided by the UK Met Office and version 10.8 of the UM.

VASP: The Vienna Ab initio Simulation Package[¶¶] (VASP) is used to model materials at the atomic scale, particularly performing electronic structure calculations and quantum-mechanical molecular dynamics. It solves the N-body Schrödinger equation using a variety of solution techniques. VASP includes a significant number of settings that affect performance, from domain decomposition options to math library parameters. Previous investigations have found that VASP is bound by floating-point compute performance at scales of up to a few hundred cores. For bigger sizes, its heavy use of MPI collectives begins to dominate, and the application becomes bound by communication latency.[15] The benchmark utilized is known as `PdO`, because it simulates a slab of palladium oxide. It consists of 174 atoms, and it was originally designed by one of VASP's developers, who also found that (on a single node) the benchmark is mostly compute bound; however, there exist a few methods that benefit from increased memory bandwidth.[16]

[¶] http://www.gromacs.org
[#] https://bitbucket.org/pszilard/isambard-bench-pack.git
[‖] https://www.nemo-ocean.eu
[**] https://www.openfoam.com/download/install-source.php
[††] https://opensbli.github.io
[‡‡] http://www.archer.ac.uk/community/benchmarks/archer/
[§§] https://www.metoffice.gov.uk/research/modelling-systems/unified-model
[¶¶] http://www.vasp.at

TABLE 2 Compilers used for benchmarking

Benchmark	ThunderX2	Broadwell	Skylake
STREAM	Arm 18.3	Intel 18	CCE 8.7
CloverLeaf	CCE 8.7	Intel 18	Intel 18
TeaLeaf	CCE 8.7	GCC 7	Intel 18
SNAP	CCE 8.6	Intel 18	Intel 18
Neutral	GCC 8	Intel 18	GCC 7
CP2 K	GCC 8	GCC 7	GCC 7
GROMACS	GCC 8	GCC 7	GCC 7
NAMD	Arm 18.2	GCC 7	GCC 7
NEMO	CCE 8.7	CCE 8.7	CCE 8.7
OpenFOAM	GCC 7	GCC 7	GCC 7
OpenSBLI	CCE 8.7	Intel 18	CCE 8.7
UM	CCE 8.6	CCE 8.5	CCE 8.7
VASP	GCC 7.2	Intel 18	Intel 18

4 | RESULTS

4.1 | Platforms

The early-access part of the Isambard system was used to produce the Arm results presented in this paper. Each of these Arm nodes houses two 32-core Cavium ThunderX2 processors running at 2.2 GHz alongside 256 GB of DDR4 DRAM clocked at 2500 MHz. Note that this is slightly below the 2666-MHz memory speed of Isambard's XC50 nodes. On May 7, 2018, Cavium announced the general availability of ThunderX2, with an RRP for the 32c 2.2 GHz part of $1795 each. The ThunderX2 processors support 128-bit vector Arm Advanced SIMD instructions (sometimes referred to as ''NEON''), and each core is capable of four-way simultaneous multithreading (SMT), for a total of up to 256 hardware threads per node. The processor's on-chip data cache is organized into three levels: a private L1 and L2 cache for each core and a 32-MB L3 cache shared between all the cores. Finally, each ThunderX2 socket utilizes eight separate DDR4 memory channels running at up to 2666 MHz.

The Cray XC40 supercomputer ''Swan'' was used for access to Intel Broadwell and Skylake processors, with an additional internal Cray system, ''Horizon'', providing access to a more mainstream SKU of Skylake.

- Intel Xeon Platinum 8176 (Skylake) 28-core @ 2.1 GHz, dual-socket, with 192 GB of DDR4-2666 DRAM. RRP $8719 each.
- Intel Xeon Gold 6148 (Skylake) 20-core @ 2.4 GHz, dual-socket, with 192 GB of DDR4-2666 DRAM. RRP $3078 each.
- Intel Xeon E5-2699 v4 (Broadwell) 22-core @ 2.2 GHz, dual-socket, with 128 GB of DDR4-2400 DRAM. RRP $4115 each.

The recommended retail prices (RRPs) are correct at the time of writing (May 2018) and taken from Intel's website.[##] The Skylake processors provide an AVX-512 vector instruction set, which means that each FP64 vector operation operates on eight elements; by comparison, Broadwell utilizes AVX2, which is 256 bits wide, operating on four FP64 elements. The Xeon processors all have three levels of on-chip (data) cache, with an L1 and L2 cache per core, along with a shared L3. This selection of CPUs provides coverage of both the state of the art and the status quo of current commonplace HPC system design. We include high-end models of both Skylake and Broadwell in order to make the comparison as challenging as possible for ThunderX2. It is worth noting that in reality, most Skylake and Broadwell systems will use SKUs from much further down the range, of which the Xeon Gold part described above is included as a good example. This is certainly true for the current Top 500 systems.

A summary of the hardware used, along with peak floating-point and memory bandwidth performance, is shown in Table 1, whereas a chart comparing key hardware characteristics of the main CPUs in our test (the three near-top-of-bin parts: Broadwell 22c, Skylake 28c, and ThunderX2 32c) is shown in Figure 2. There are several important characteristics that are worthy of note. First, the wider vectors in the x86 CPUs give them a significant peak floating-point advantage over ThunderX2. Second, wider vectors also require wider datapaths into the lower levels of the cache hierarchy. This results in the x86 CPUs having an L1 cache bandwidth advantage, but we see the advantage reducing as we go up the cache levels, until once at external memory, it is ThunderX2 that has the advantage, due to its greater number of memory channels. Third, as seen in most benchmark studies in recent years, dynamic voltage and frequency scaling (DVFS) makes it harder to reason about the percentage of peak performance that is being achieved. For example, while measuring the cache bandwidth results shown in Figure 2, we observed that our Broadwell 22c parts consistently increased their clock speed from a base of 2.2 GHz up to 2.6 GHz. In contrast, our Skylake 28c parts consistently *decreased* their clock speed from a base of 2.1 GHz down to 1.9 GHz. Our ThunderX2 parts ran at a consistent 2.2 GHz. At the actual, measured clock speeds, the values of the fraction of theoretical peak bandwidth achieved at L1 for Broadwell 22c, Skylake 28c, and ThunderX2 32c were 57%, 55%, and 51%, respectively.

In order to measure the sustained cache bandwidths as presented in Figure 2, we used the methodology described in our previous work.[17] The Triad kernel from the STREAM benchmark was run in a tight loop on each core simultaneously, with problem sizes selected to ensure residency

[##] https://ark.intel.com/

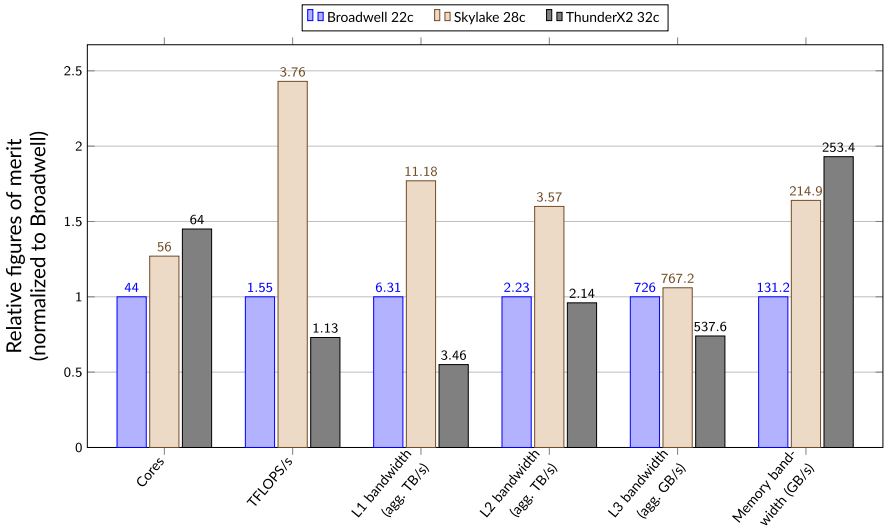

FIGURE 2 Comparison of the properties of Broadwell 22c, Skylake 28c, and ThunderX2 32c. Results are normalized to Broadwell

in each level of the cache. The bandwidth is then modeled using the array size, number of iterations, and the time for the benchmark to run. This portable methodology was previously shown to attain the same performance as handwritten benchmarks, which only work on their target architectures.[18]

We evaluated three different compiler families for ThunderX2 in this study: GCC 7 and 8, the LLVM-based Arm HPC Compiler 18.2 and 18.3, and Cray's CCE 8.6 and 8.7. We believe this is the first study to date that has compared all three of these compilers targeting Arm. The compiler that achieved the highest performance in each case was used in the result graphs displayed below. Likewise for the Intel processors, we used GCC 7, Intel 2018, and Cray CCE 8.5–8.7. Table 2 lists the compiler that achieved the highest performance for each benchmark in this study.

4.2 | Mini-apps

Figure 3 compares the performance of our target platforms over a range of representative mini-apps.

STREAM: The STREAM benchmark measures the sustained memory bandwidth from the main memory. For the processors tested, the available memory bandwidth is essentially determined by the number of memory controllers. Intel Xeon Broadwell and Skylake processors have four and six memory controllers per socket, respectively. The Cavium ThunderX2 processor has eight memory controllers per socket. The results in Figure 3 show a clear trend that Skylake achieves a 1.64× improvement over Broadwell, which is to be expected, given Skylake's faster memory speed (2666 MHz DDR4 vs Broadwell's 2400 MHz). The eight memory controllers per socket on the Cavium ThunderX2 achieve a 1.93× speedup over Broadwell.

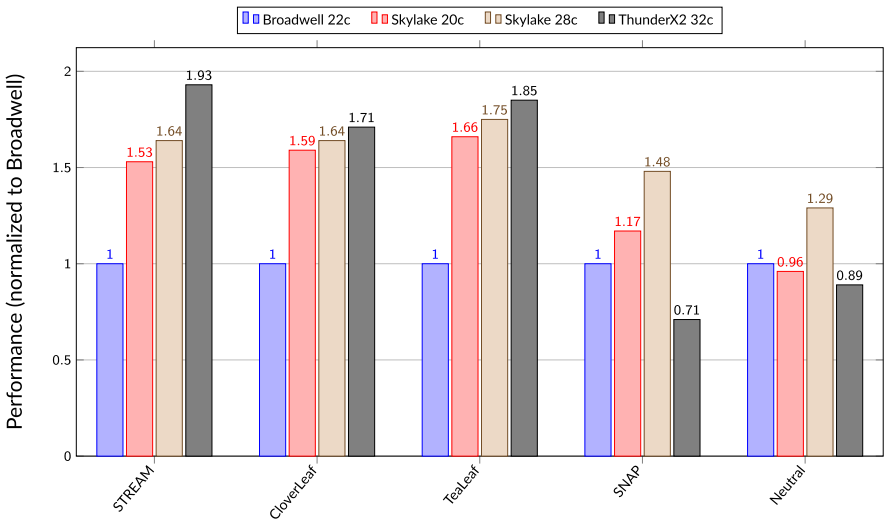

FIGURE 3 Comparison of Broadwell, Skylake, and ThunderX2 for a range of mini-apps. Results are normalized to Broadwell

Broadwell and Skylake are achieving 84.4% and 83.9% of theoretical peak memory bandwidth, respectively. At 253 GB/s, ThunderX2 is achieving 79.2% of theoretical peak memory bandwidth. On ThunderX2, we found that the best STREAM Triad performance was achieved with running 16 OpenMP threads per socket (32 in total), rather than one thread per core (64 total). The CrayPat tool reports cache hit rates of 69.4% for L1 and 66.9% for L2, both for 32 threads, whereas for 64 threads, cache hit rates of 66.6% for L1 and 38.5% for L2 are reported; notice that the L2 hit rate has dropped sharply when using more threads. On the 18-core Broadwell CPUs in the Isambard Phase 1 system, using the Cray Compiler, CrayPat reports cache hit rates of 66.7% for L1 and 11.3% for L2, both with 36 threads; the numbers are similar with only half the number of threads. The low L2 hit rate is showing that there is strong reliance on the main memory bandwidth for data traffic, as expected.

The use of nontemporal store instructions is an important optimization for the STREAM benchmark, as Raman et al showed, where, for the Triad kernel, the use of nontemporal store instructions resulted in a 37% performance improvement.[19] On the Intel architecture, using these instructions for the write operation in the STREAM kernels ensures that the cache is not polluted with the output values, which are not reused; note that it is assumed that the arrays are larger than the last-level cache. As such, if these data occupied space in the cache, it would reduce the capacity available for the other arrays that are being prefetched into the cache. The construction of the STREAM benchmark with arrays allocated on the stack with the problem size known at compile time allows the Intel compiler to generate nontemporal store instructions for all the Intel architectures in this study. Although the GCC compiler does not generate nontemporal stores for the Cavium ThunderX2 architecture (in fact, it cannot generate nontemporal store instructions for any architecture), the implementation of these instructions within the ThunderX2 architecture does not result in a bypass of cache. Instead, the stores still write to the L1 cache, but in a way that exploits the write-back policy and the *least recently used* eviction policy to limit the disruption on cache. As such, this may be limiting the achieved STREAM performance on ThunderX2, as memory bandwidth is being wasted, evicting the output arrays of the kernels.

Despite this lack of true streaming stores, it is clear that the additional memory controllers on the ThunderX2 processors provide a clear external memory bandwidth advantage over Broadwell and Skylake processors.

CloverLeaf: The normalized results for the CloverLeaf mini-app in Figure 3 are very consistent with those for STREAM on the Intel Xeon and Cavium ThunderX2 processors. CloverLeaf is a structured grid code, and a majority of its kernels are bound by the available memory bandwidth. It has been shown previously that the memory bandwidth increases from GPUs result in proportional improvements for CloverLeaf.[4] The same is true on the processors in this study, with the improvements on ThunderX2 coming from its greater memory bandwidth. Therefore, for structured grid codes, we indeed see that the runtime is proportional to the external memory bandwidth of the system, and ThunderX2 provides the highest bandwidth out of the processors tested.

It has been noted that the time per iteration increases as the simulation progresses on the ThunderX2 processor. This phenomenon is not noticeable on the x86 processors. We believe this is due to the data-dependent need for floating-point intrinsic functions (such as `abs`, `min`, and `sqrt`); this can be seen in the `viscosity` kernel, for instance. As the time iteration progresses, the need for such functions is higher, and therefore, the kernel increases in runtime. Although these kernels are memory bandwidth bound, the increased number of floating-point operations increases the computational intensity and is therefore slightly less bandwidth bound (under the Roofline model).

TeaLeaf: The TeaLeaf mini-app again tracks the memory bandwidth performance of the processors, as previously shown on x86 and GPU architectures.[5] The additional memory bandwidth of the ThunderX2 processor clearly improves the performance over those processors with fewer memory controllers.

SNAP: Understanding the performance of the SNAP proxy application is difficult.[7] If truly memory bandwidth bound, the extra bandwidth available on the ThunderX2 processor would increase the performance as with the other memory bandwidth–bound codes discussed in this study; however, the ThunderX2 processor is almost 30% slower than Broadwell for the tested problem for this code. The Skylake processor does give an improvement, however, and while this does have memory bandwidth improvements over Broadwell, this is not the only significant architectural change. Skylake has 512-bit vectors, which creates a wide data path through the cache hierarchy—a whole cache line is loaded into registers for each load operation. In comparison, Broadwell has 256-bit vectors, and ThunderX2 has 128-bit vectors, moving half and a quarter of a 64-byte cache line per load operation, respectively.

The main sweep kernel in the SNAP code requires that a cache hierarchy support both accessing a very large data set and simultaneously keeping small working set data in low levels of cache. The CrayPat profiler reports the cache hit rates for L1 and L2 caches when using the Cray Compiler. On ThunderX2, the hit rates for the caches are both at around 84%, which is much higher than the hit rate for the STREAM benchmark (where the latter is truly main memory bandwidth bound). As such, this shows that the SNAP proxy application is heavily reusing data in cache, and so, the performance of the memory traffic to and from cache is a key performance factor. On the Broadwell processors in Isambard Phase 1, CrayPat reports cache hit rates of 89.4% for L1 and 24.8% for L2. Again, the L1 cache hit rate is much higher than in the STREAM benchmark, indicating high reuse of data in the L1 cache, but that the L2 is not used as efficiently. It is the subject of future work to understand how the data access patterns of SNAP interact with the cache replacement policies in these processors. However, for this study, it is clear that main memory bandwidth is not necessarily the performance limiting factor, but rather, it is the access to the cache where the x86 processors have an advantage.

Neutral: In previous work, it was shown that the Neutral mini-app has algorithmically little data reuse, due to the random access to memory required for accessing the mesh data structures.[8] Additionally, the access pattern is data driven, and this not predictable, and so, any hardware prefetching of data into cache according to common access patterns is likely to be ineffective, resulting in a relatively high cache miss rate. Indeed, CrayPat shows a low percentage (27.3%) of L2 cache hits on the ThunderX2 processor and a similar percentage on Broadwell. The L1 cache hit rate is high on both architectures, with over 95% hits. As such, the extra memory bandwidth available on the ThunderX2 processor

does not provide an advantage over the Intel Xeon processors. Note from Figure 3 that the ThunderX2 processor still achieves performance close to that of Broadwell, with Skylake 28c only offering a small performance improvement of about 29%.

Mini-app performance summary

Many of these results highlight the superior memory bandwidth offered by ThunderX2's eight memory channels, which deliver 253 GB/s for the STREAM Triad benchmark[2]—twice that of Broadwell and 18% more than Skylake. This performance increase can be seen in the memory bandwidth bound mini-apps such as CloverLeaf and TeaLeaf, with the ThunderX2 processor showing similar improvements to STREAM over the x86 processors.

The SNAP and Neutral mini-apps, however, rely more on the on-chip memory architecture (the caches), and so, they are unable to leverage the external memory bandwidth on all processors. As such, the additional memory controllers on the ThunderX2 processors do not seem to improve the performance of these mini-apps relative to processors with less memory bandwidth. The exact nature of the interaction of these algorithms with the cache hierarchy is the subject of future study.

4.3 | Applications

Figure 4 compares the performance of dual-socket Broadwell, Skylake, and ThunderX2 systems for the real application workloads described in Section 3.2.

CP2K: CP2 K comprises many different kernels that have varying performance characteristics, including floating-point–intensive routines and those that are affected by external memory bandwidth. While the floating-point routines run up to 3× faster on Broadwell compared to ThunderX2, the improved memory bandwidth and higher core counts provided by ThunderX2 allow it to reach a ∼15% speedup over Broadwell for this benchmark. The 28c Skylake processor provides even higher floating-point throughout and closes the gap in terms of memory bandwidth, yielding a further 19% improvement over ThunderX2. The H2O-64 benchmark has been shown to scale sublinearly when running on tens of cores,[20] which impacts the improvements that ThunderX2 can offer for its 64 cores.

GROMACS: The GROMACS performance results are influenced by two main factors. First, the application is heavily compute bound, and the x86 platforms are able to exploit their wider vector units and wider datapaths to cache. Performance does not scale perfectly with vector width due to the influence of other parts of the simulation, particularly the distributed FFTs. Second, because GROMACS uses hand-optimized vector code for each platform, x86 benefits from having the more mature implementation, one that has evolved over many years. Since Arm HPC platforms are new, it is likely that the NEON implementation is not yet at peak efficiency for ThunderX2.

NAMD: As discussed in Section 3.2, NAMD is not clearly bound by a single factor, and thus, it is hard to underline a specific reason why one platform is slower (or faster) than another. It is likely that results are influenced by a combination of memory bandwidth, compute performance, and other latency-bound operations. The results observed do correlate with memory bandwidth, making Broadwell the slowest platform of the three for this application. Running more than one thread per core in SMT increased NAMD performance, and this is the most pronounced on ThunderX2, which can run four hardware threads on each physical core. Furthermore, due to NAMD's internal load balancing mechanism, the application is able to efficiently exploit a large number of threads, which confers yet another advantage to ThunderX2 for being able to run more threads (256) than the x86 platforms (112 on Skylake 28c). As a result, while ThunderX2 32c does not quite match the top-bin Skylake 28c, it does outperform the mainstream Skylake 20c by about 18%.

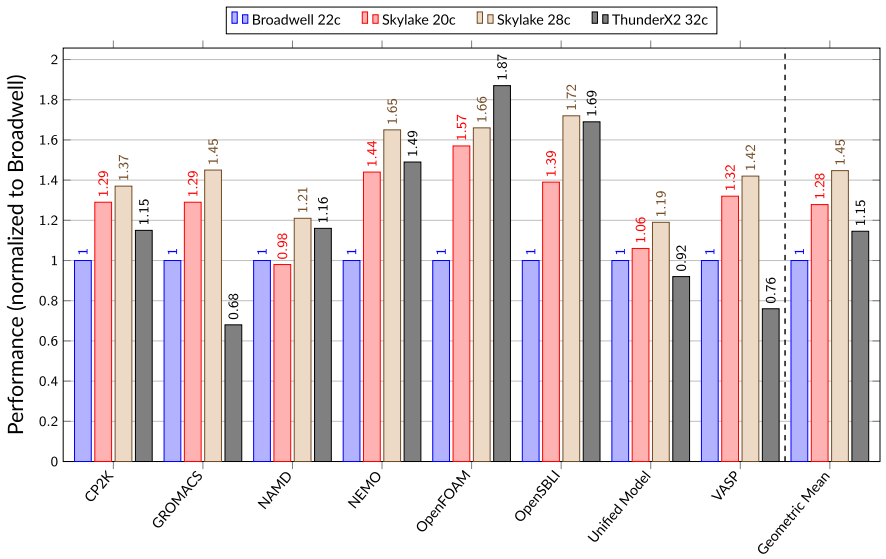

FIGURE 4 Comparison of Broadwell, Skylake, and ThunderX2 for a set of real application codes. Results are normalized to Broadwell

NEMO: For the NEMO benchmark, ThunderX2 is 1.49× faster than Broadwell 22c and is slightly faster than Skylake 20c, while not quite matching Skylake 28c. While the benchmark should be mostly memory bandwidth bound, leading to significant improvements over Broadwell, the greater on-chip cache bandwidth of Skylake gives the top-bin part a slight performance advantage over ThunderX2. Running with multiple threads per core to ensure that the memory controllers are saturated provides a small improvement for ThunderX2.

OpenFOAM: The OpenFOAM results follow the STREAM behavior of the three platforms closely, confirming that memory bandwidth is the main factor that influences performance here. With its eight memory channels, ThunderX2 yields the fastest result, at 1.87× the Broadwell performance. Skylake is able to run by 1.57×-1.66× faster than Broadwell, ie, a bigger difference than in plain STREAM, because it is likely able to get additional benefit from its improved caching, which is not a factor in STREAM. This benchmark strongly highlights ThunderX2's strength in how performance can be improved significantly by higher memory bandwidth.

OpenSBLI: The OpenSBLI benchmark exhibits a similar performance profile to OpenFOAM, providing another workload that directly benefits from increases in external memory bandwidth. The ThunderX2 system produces speedups of 1.69× and 1.21× over Broadwell 22c and Skylake 20c, respectively, and almost matches Skylake 28c.

Unified Model: Comprising 2 million lines of Fortran, the Unified Model is arguably the most challenging of the benchmarks used in this study, stressing the maturity of the compilers as well as the processors themselves. Skylake 28c only yields a 19% improvement over Broadwell 22c, indicating that the performance of this test case is not entirely correlated to memory and cache bandwidth or floating-point compute and that the relatively low-resolution benchmark may struggle to scale efficiently to higher core counts. The ThunderX2 result is around 8% slower than our top-bin Broadwell, but demonstrates the robustness of the Cray software stack on Arm systems by successfully building and running without requiring any modifications. Interestingly, when running on just a single socket, ThunderX2 provides a ∼ 15% improvement over Broadwell. We also observed performance regressions in the more recent versions of CCE on all three platforms; the Broadwell result was fastest using CCE 8.5, which could not be used for either Skylake or ThunderX2.

VASP: The calculations performed by the VASP benchmark are dominated by floating-point–intensive routines, which naturally favor the x86 processors with their wider vector units. While the higher core counts provided by ThunderX2 make up for some of the difference, the VASP benchmark exhibits a similar profile to GROMACS, with ThunderX2 around ∼ 24% slower than Broadwell 22c, and sitting around half the speed of Skylake 28c.

Compiler comparison

Figure 5 compares the latest versions of the three available compilers on the ThunderX2 platform, normalized to the best performance observed for each benchmark. The benefit of having multiple compilers for Arm processors is clear, as none of the compilers dominate performance, and no single compiler is able to build all of the benchmarks. The performance for the mini-apps is broadly similar across all of the compilers, with 15%-20% variations for SNAP and Neutral whose more complex kernels draw out differences in the optimizations applied by the compilers. The Arm HPC compiler uses the LLVM-based Flang, a relatively new Fortran frontend, which, at the time of writing, produces an internal compiler error while building CP2K. Both CP2K and GROMACS crash at runtime when built with CCE; this issue also occurs on the Broadwell system and, hence, is not specific to Arm processors. While the NAMD benchmark builds and runs correctly with GCC 7, it currently hangs after initialization with GCC 8. At the time of writing, it is unclear whether these issues are a result of bugs in the applications themselves or miscompilations by the compilers. NAMD failed to build with CCE because Charm++ uses some inline assembly syntax that is not yet supported by the Cray Compiler. OpenFOAM exhibits multiple syntax errors in its source code, which are only flagged as issues by GCC 8 and CCE. The largest performance difference we observed between the compilers was with the OpenSBLI benchmark, where the code generated by CCE is 2.5× faster than any

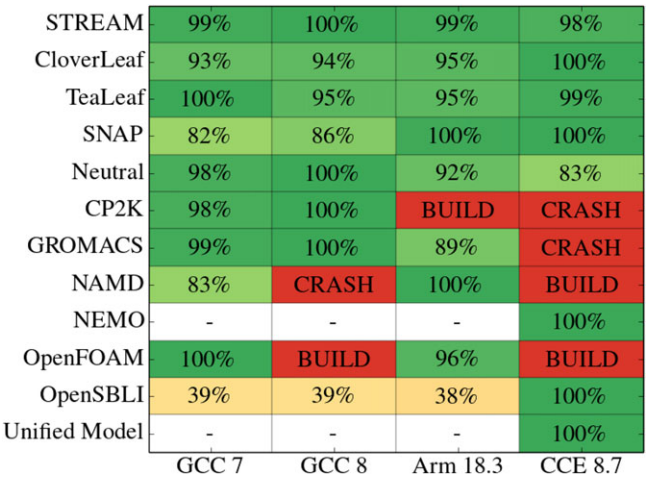

FIGURE 5 Efficiency of different compilers running on Cavium ThunderX2. The `BUILD` and `CRASH` labels denote configurations that either failed to build or crashed at runtime, respectively. The "-" indicates that the build configuration was not supported by the benchmark at the time of writing

of the other compilers. On x86, performance is much closer between all of the compilers, and some other testing we have performed leads us to believe that this performance discrepancy is most likely due to MPI library differences between the CCE and non-CCE builds of OpenSBLI. Investigations are continuing, and we believe the outcome will be that GCC and Arm will come up to be much closer to CCE once we have resolved the issue.

Performance per Dollar

So far, we have focused purely on performance, but one of the main advantages of the Arm ecosystem for HPC should be the increased competition it brings. While Performance per Dollar is hard to quantify rigorously and price is always subject to negotiation, we can still reveal some illuminating information by using the published list of prices of the CPUs to compare the respective Performance per Dollar of the processors. In the results to follow, we only consider the CPU list prices. This deliberately leaves every other factor out of the comparison; factoring our results into a subjective whole system cost is left as an exercise for the expert reader. Figures 6 and 7 show the Performance per Dollar of our platforms of interest, normalized to Broadwell 22c. The numbers are calculated by taking the application performance numbers shown in Figures 3 and 4 and simply dividing them by the published list prices described in Section 4.1, before renormalizing against Broadwell 22c.

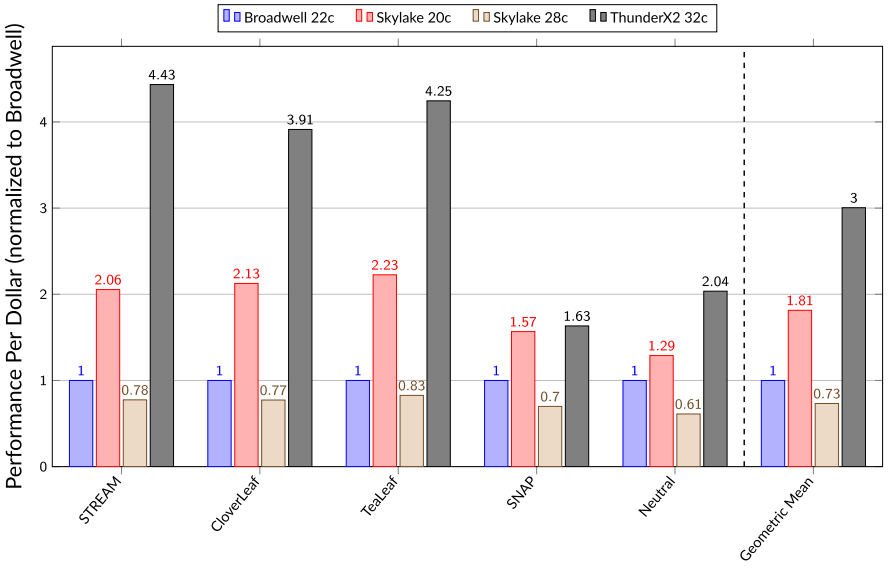

FIGURE 6 Performance per Dollar comparison of Broadwell, Skylake, and ThunderX2 for a range of mini-apps. Results are normalized to Broadwell

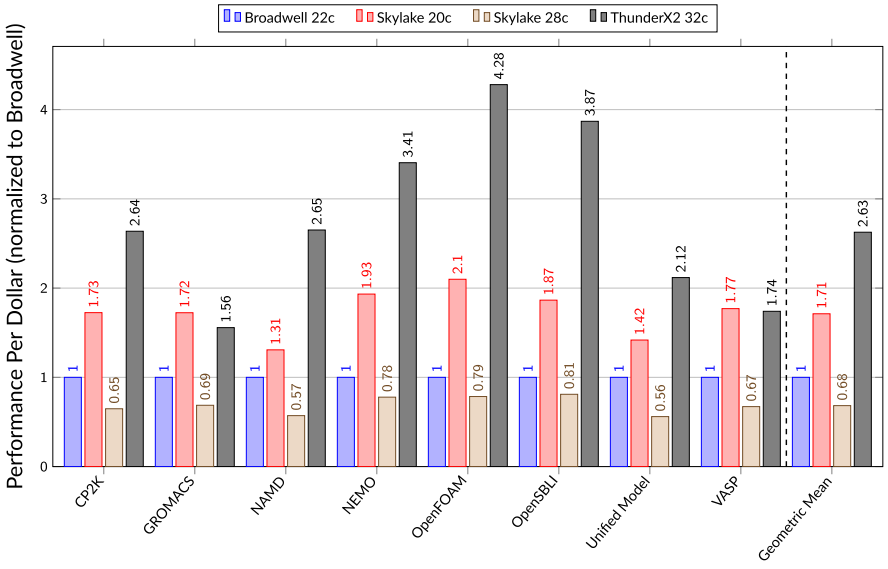

FIGURE 7 Performance per Dollar price comparison of Broadwell, Skylake, and ThunderX2 for our application codes. Results are normalized to Broadwell

There are a number of interesting points to highlight. First, considering the mini-apps in Figure 6, we see that ThunderX2's RRP of just $1795 gives it a compelling advantage compared to all the other platforms. ThunderX2 consistently comes ahead of not just the top-bin—and therefore expensive—Skylake 28c but also of the more cost-conscious, mainstream Skylake 20c SKU. The picture for real applications in Figure 7 is similar, where, even in cases when ThunderX2 had a performance disadvantage, such as in the compute-bound codes GROMACS and VASP, ThunderX2 becomes competitive once cost is considered. Where ThunderX2 was already competitive on performance, adding in cost makes ThunderX2 look even more competitive, often achieving a performance/price advantage over the more cost-oriented Skylake 20c SKUs of 2× or more.

Performance summary

Overall, the results presented in this section demonstrate that the Arm-based Cavium ThunderX2 processors are able to execute a wide range of important scientific computing workloads with performance that is competitive with state-of-the-art x86 offerings. The ThunderX2 processors can provide significant performance improvements when an application's performance is limited by external memory bandwidth, but are slower in cases where codes are compute bound. When processor cost is taken into account, ThunderX2's proposition is even more compelling. With multiple production-quality compilers now available for 64-bit Arm processors, the software ecosystem has reached a point where developers can have confidence that real applications will build and run correctly, in the vast majority of cases with no modifications.

Some of the applications tested highlight the lower floating-point throughput and the L1/L2 cache bandwidth of ThunderX2. Both of these characteristics stem from the narrower vector units relative to AVX-capable x86 processors. In 2016, Arm unveiled the Scalable Vector Extension ISA,[21] which will enable hardware vendors to design processors with much wider vectors of up to 2048 bits, compared to the 128 bits of today's NEON. We therefore anticipate that the arrival of SVE-enabled Arm-based processors in the next couple of years will likely address most of the issues observed in this study, enabling Arm processors to deliver even greater performance for a wider range of workloads.

5 | REPRODUCIBILITY

With an architecture such as Arm, which is new to mainstream HPC, it is important to make any benchmark comparison as easy to reproduce as possible. To this end, the Isambard project is making all of the detailed information about how each code was compiled and run, along with the input parameters to the test cases, available as an open-source repository on GitHub.[‖‖] The build scripts will show which compilers were used in each case, what flags were set, and which math libraries were employed. The run scripts will show which test cases were used and how the runs were parameterized. These two sets of scripts should enable any third party to reproduce our results, provided that they have access to similar hardware. The scripts do assume a Cray-style system but should be easily portable to other versions of Linux on non-Cray systems.

6 | CONCLUSIONS

The results presented in this paper demonstrate that Arm-based processors are now capable of providing levels of performance competitive with state-of-the-art offerings from the incumbent vendors, while significantly improving Performance per Dollar. The majority of our benchmarks compiled and ran successfully *out of the box*, and no architecture-specific code tuning was necessary to achieve high performance. This represents an important milestone in the maturity of the Arm ecosystem for HPC, where these processors can now be considered as viable contenders for future procurements. Future work will use the full Isambard system to evaluate production applications running at scale on ThunderX2 processors.

We did not address energy efficiency in this paper. Our early observations suggest that the energy efficiency of ThunderX2 is in the same ballpark as the x86 CPUs we tested. This is not a surprise—for a given manufacturing technology, a FLOP will take a certain number of Joules, and moving a byte a certain distance across a chip will also take a certain amount of energy. There is no magic, and the instruction set architecture has very little impact on the energy efficiency when most of the energy is being spent moving data and performing numerical operations.

Overall, these results suggest that Arm-based server CPUs that have been optimized for HPC are now genuine options for production systems, offering performance competitive with best-in-class CPUs, while potentially offering attractive price/performance benefits.

ACKNOWLEDGMENTS

As the world's first production Arm supercomputer, the GW4 Isambard project could not have happened without support from a lot of people. First are the coinvestigators at the four GW4 universities, the Met Office, and Cray who helped write the proposal: Prof James Davenport (Bath), Prof Adrian Mulholland (Bristol), Prof Martyn Guest (Cardiff), Prof Beth Wingate (Exeter), Dr Paul Selwood (Met Office), and Adrian Tate (Cray). Second are members of the operations group, who designed and now run the Isambard system: Steven Chapman and Roshan Mathew (Bath); Christine Kitchen and James Green (Cardiff); Dave Acreman, Rosie Rowlands, Martyn Brake, and John Botwright (Exeter); Simon Burbidge and Chris Edsall (Bristol); David Moore, Guy Latham, and Joseph Heaton (Met Office); and Steven Jordan and Jess Jones (Cray). Finally, we thank the

‖‖ https://github.com/UoB-HPC/benchmarks/releases/tag/CCPE-CUG-2018

attendees of the first two Isambard hackathons, who did most of the code porting behind the results in this paper, and Federica Pisani from Cray who organized the events.

Attendees of the hackathons in November 2017 and March 2018 included (listed alphabetically): David Acreman, Lucian Anton, Andy Bennett, Charles Bentham, Steven Chapman, Steven Daniels, Luiz DeRose, Christopher Edsall, Pawan Ghildiyal, Andreas Glöss, Matthew Glover, Indraneil Gokhale, James Grant, Thomas Green, Alistair Hart, Ian Harvey, Phil Hasnip, Mattijs Janssens, Hrvoje Jasak, Jess Jones, Merzuk Kaltak, Ilias Katsardis, JaeHyuk Kwack, Alfio Lazzaro, Unai Lopez-Novoa, David Lusher, Simon McIntosh-Smith, Sachin Nanavati, Szilárd Páll, Pablo Ouro, Andrei Poenaru, Iakov Polyak, Heidi Poxon, James Price, Harvey Richardson, Kevin Roy, Jonathan Skelton, Craig Steffen, Adrian Tate, Adam Voysey, and Christopher Woods.

Access to the Cray XC40 supercomputers "Swan" and "Horizon" was kindly provided though Cray Inc's Marketing Partner Network. The Isambard project is funded by EPSRC, the GW4 alliance, the Met Office, Cray, and Arm. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725. The Isambard project is funded by EPSRC research grant EP/P020224/1.

ORCID

Simon McIntosh-Smith https://orcid.org/0000-0002-5312-0378
Tom Deakin https://orcid.org/0000-0002-6439-4171

REFERENCES

1. Turner A, McIntosh-Smith S. A survey of application memory usage on a national supercomputer: an analysis of memory requirements on ARCHER. In: Jarvis S, Wright S, Hammond S, eds. *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017, Proceedings*. Cham, Switzerland: Springer International Publishing AG; 2018:250-260.
2. McCalpin JD. Memory bandwidth and machine balance in current high performance computers. *IEEE Comput Soc Tech Comm Comput Archit*. 1995:19-25.
3. Mallinson A, Beckingsale DA, Gaudin W, Herdman A, Levesque J, Jarvis SA. CloverLeaf: preparing hydrodynamics codes for exascale. In: Proceedings of the Cray User Group; 2013; Napa Valley, CA.
4. McIntosh-Smith S, Boulton M, Curran D, Price J. On the performance portability of structured grid codes on many-core computer architectures. In: *Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings*. Cham, Switzerland: Springer International Publishing Switzerland; 2014:53-75. *Lecture Notes in Computer Science*; vol. 8488.
5. McIntosh-Smith S, Martineau M, Deakin T, et al. TeaLeaf: a mini-application to enable design-space explorations for iterative sparse linear solvers. Paper presented at: 2017 IEEE International Conference on Cluster Computing (CLUSTER); 2017; Honolulu, HI.
6. Zerr RJ, Baker RS. *SNAP: SN (Discrete Ordinates) Application Proxy - Proxy Description*. Technical report. Los Alamos, NM: Los Alamos National Laboratory; 2013. LA-UR-13-21070.
7. Deakin T, Gaudin W, McIntosh-Smith S. On the mitigation of cache hostile memory access patterns on many-core CPU architectures. In: *High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*. Cham, Switzerland: Springer International Publishing AG; 2017;348-362.
8. Martineau M, McIntosh-Smith S. Exploring on-node parallelism with neutral, a Monte Carlo neutral particle transport mini-app. Paper presented at: 2017 IEEE International Conference on Cluster Computing (CLUSTER); 2017; Honolulu, HI.
9. Páll S, Hess B. A flexible algorithm for calculating pair interactions on SIMD architectures. *Comput Phys Commun*. 2013;184(12):2641-2650.
10. Nelson MT, Humphrey W, Gursoy A, et al. NAMD: a parallel, object-oriented molecular dynamics program. *Int J Supercomput Appl High Perform Comput*. 1996;10(4):251-268.
11. Phillips JC, Sun Y, Jain N, Bohm EJ, Kalé LV. Mapping to irregular torus topologies and other techniques for petascale biomolecular simulation. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2014; New Orleans, LA.
12. Jasak H, Jemcov A, Tuković Z. OpenFOAM: a C++ library for complex physics simulations. In: Proceedings of the International Workshop on Coupled Methods in Numerical Dynamics (IUC); 2007; Dubrovnik, Croatia.
13. Heft A, Indinger T, Adams NA. *Introduction of a New Realistic Generic Car Model for Aerodynamic Investigations*. SAE technical paper. Munich, Germany: Technische Universität München; 2012.
14. Hall C. The UK Meteorological Office climate model: The AMIP run and recent changes to reduce the systematic errors. World Meteorological Organization-Publications-WMO TD. 1995.
15. Catlow R, Woodley S, Leeuw ND, Turner A. Optimising the performance of the VASP 5.2.2 code on HECToR. HECToR; 2010.
16. Zhao Z, Marsman M. Estimating the performance impact of the MCDRAM on KNL using dual-socket Ivy bridge nodes on Cray XC30. In: Proceedings of the Cray User Group; 2016; London, UK.
17. Deakin T, Price J, McIntosh-Smith S. Portable methods for measuring cache hierarchy performance. The International Conference for High Performance Computing, Networking, Storage and Analysis; 2017; Denver, CO.
18. Hofmann J, Hager G, Wellein G, Fey D. An analysis of core- and chip-level architectural features in four generations of Intel server processors. In: *High Performance Computing 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings*. Cham, Switzerland: Springer International Publishing AG; 2017:294-314. *Lecture Notes in Computer Science*; vol. 10266.
19. Raman K, Deakin T, Price J, McIntosh-Smith S. Improving achieved memory bandwidth from C++ codes on Intel Xeon Phi Processor (Knights Landing). Paper presented at: Presentation at International Xeon Phi User Group Spring Meeting; 2017; Cambridge, UK.

20. Bethune I, Reid F, Lazzaro A. CP2K performance from Cray XT3 to XC30. Paper presented at: Cray User Group (CUG); 2014; Lugano, Switzerland.
21. Stephens N, Biles S, Boettcher M, et al. The ARM scalable vector extension. *IEEE Micro*. 2017;37(2):26-39.

How to cite this article: McIntosh-Smith S, Price J, Deakin T, Poenaru A. A performance analysis of the first generation of HPC-optimized Arm processors. *Concurrency Computat Pract Exper*. 2019;e5110. https://doi.org/10.1002/cpe.5110