# ARM HPC Ecosystem and the Reemergence of Vectors

[Invited Paper]

Alejandro Rico
ARM
Austin, TX, USA
alejandro.rico@arm.com

José A. Joao
ARM
Austin, TX, USA
jose.joao@arm.com

Chris Adeniyi-Jones
ARM
Cambridge, UK
chris.adeniyijones@arm.com

Eric Van Hensbergen
ARM
Austin, TX, USA
eric.vanhensbergen@arm.com

## ABSTRACT

ARM's involvement in funded international projects has helped pave the road towards ARM-based supercomputers. ARM and its partners have collaborately grown an HPC ecosystem with software and hardware solutions that provide choice in a unified software ecosystem. Partners have announced important HPC deployments resulting from collaborations around the globe. One of the key enabling technologies for ARM in HPC is the Scalable Vector Extension, an instruction set extension for vector processing. This paper discusses ARM's journey into HPC, the current state of the ARM HPC ecosystem, the approach to HPC node architecture co-design, and details on the Scalable Vector Extension as a future technology representing the reemergence of vectors.

## CCS Concepts

•**Computer systems organization** → **Single instruction, multiple data; Multicore architectures;** •**Computing methodologies** → *Massively parallel and high-performance simulations;*

## Keywords

ARM; High-performance computing; Scalable Vector Extension; vector processing; HPC

## 1. INTRODUCTION

Modeling and simulation of physical phenomena has established itself as one of the three pillars of science together with theory and experimentation. Despite the extraordinary performance improvement in supercomputing over the last three decades, with current top systems performance in the scale of petaFLOPS ($10^{15}$ floating-point operations per second -FLOPS- on 64-bit values), great societal challenges in fields such as environment, energy, health, materials and transportation require higher levels of performance to simulate larger problems with higher resolution and more accurate models. These requirements have been exacerbated by the availability of enormous amounts of data and recent advances in machine learning - enabling reasoning about and providing solutions to complex systems thereby giving birth to innovative fields such as precision medicine, smart cities, smart factories and precision agriculture.

The next milestone in the quest for higher performance for science is exascale: systems able to provide performance in the scale of exaFLOPS ($10^{18}$ 64-bit FLOPS). The ability to cram an increasing amount of transistors in a single chip without increasing power density, commonly referred to as *Moore's Law*, is slowing down. Achieving higher performance levels at reasonable power figures is, therefore, increasingly challenging and requires more innovation and engineering efforts.

ARM's business model is based on licensing intellectual property (IP) to partners. Informally speaking, this can be seen as the *outsourcing* of processor technology R&D. ARM develops the ARM architecture (including the instruction set), microarchitectural implementations of processor components such as core, sub-system and graphics, and optimized physical implementations of those components. At the same time, it has built a software ecosystem that is leveraged by, contributed to and maintained by the partnership. This business model has been greatly successful in mobile computing for consumer electronics thanks to providing scope for partner differentiation, leading to a myriad of solutions for end products, boosting innovation through competition and collaboration in the ecosystem. The result is the availability of technologies and devices with extraordinarily increasing levels of performance and energy efficiency.

The efforts of ARM and its partners in the domain of high-end systems are creating an ecosystem of hardware and software components for high-performance computing (HPC). The key enabling technologies include the ARMv8 architecture [13]; optimized operating systems, compilers, math libraries, development environment, debugging and profiling tools [5]; and reliability, availability and serviceability (RAS) support. Early fruits of these long-term efforts are apparent in multiple announcements of deployments of large-scale systems based on ARM's ecosystem technology [14, 6, 7].



*CF'17, May 15-17, 2017, Siena, Italy*



DOI: http://dx.doi.org/10.1145/3075564.3095086

Great advances in ARM's road to HPC have been accomplished in the context of funded international projects. The EU-funded Mont-Blanc project [1, 17] led the early exploration of using ARM mobile technology for HPC [16]. Mont-Blanc achievements have helped position ARM and its ecosystem as a viable player in HPC and also aided the development of the HPC software stack. Other funded projects are expanding Mont-Blanc's reach focusing on certain enabling technologies. The FastForward-2 project funded by the US Department of Energy is looking at future compute node architectures for HPC.

One of the main enabling technologies for future ARM-based HPC systems is the ARM Scalable Vector Extension (SVE) [19], which evolved in the context of FastForward-2. SVE is the latest confirmation of the reemergence of vectors as an essential technology for HPC, and is the central piece to achieve higher levels of performance and efficiency for future ARM-based HPC systems.

This paper discusses the work ARM and its partners have developed in the context of the Mont-Blanc and FastForward-2 projects and details the key features of SVE for exploiting higher levels of parallelism across a wide range of applications and system configurations from multiple partners in the ARM HPC ecosystem.

## 2. ROAD TO HPC

ARM Cortex-A9 was the first ARM core with the combination of hardware floating-point unit (codenamed VFPv3), floating-point single-instruction multiple data unit (codenamed NEON) and symmetric multi-processor (SMP) support with up to four cores in a shared-memory configuration.

The Mont-Blanc project selected Cortex-A9 as the building block of Tibidabo, the first ARM-based HPC cluster with 128 nodes and a complete HPC software stack [18]. This prototype was a vehicle to drive the porting of fundamental parts of the HPC software stack including math libraries, compiler, runtimes, profiling and performance analysis tools. It demonstrated real HPC applications scaling to the maximum available number of cores in the system.

Mont-Blanc has had three stages, the first two finished in 2015 and 2016 respectively, and the third is on-going. In the first stage, the main line of work was to evaluate the suitability of mobile processors for HPC [16]. As part of this effort, the project deployed a 1080-node prototype, named the Mont-Blanc prototype, which is fully-commissioned with a full HPC software stack and a rich set of applications showing good scalability on the heterogeneous elements in the system, namely two Cortex-A15 cores and one Mali-T604 GPU per node [17]. These efforts helped identify and prioritize requirements for future platforms, such as 64-bit addressing, RAS features such as pervasive ECC support, improved off-chip interfaces and optimized libraries.

The ARMv8 architecture implemented by 64-bit cores in the Cortex-A family complied with the requirement of wider addressability. Current high-end core and sub-system IP such as Cortex-A73 and Coherent Mesh Network (CMN) 600 include end-to-end RAS support. ARM contributes to the development of emerging high-performance and flexible interfaces and fabrics to connect to accelerators, memory and networking, such as the Cache Coherent Interface for Accelerators (CCIX) [3] and Gen-Z [4]. These are some examples of ARM's success in filling identified gaps in high-performance, energy-efficient and capable technology for HPC.

Apart from assessing requirements and enabling software and future technologies, ARM and its partners have focused on research for future processor technologies including architecture and micro-architecture solutions for high-end systems with a focus on HPC. Simulation tools are critical for research and development and several new methodologies have been proposed and implemented to enable simulation of large HPC systems [15, 12, 11, 10, 8].

Through these efforts, the ARM HPC ecosystem has grown with an increasing number of partners involved in software support, optimizing applications and libraries for ARM-based platforms, and an increasing offering of hardware based on ARM technology. This has led ARM to participate in multiple Exascale programs across the globe with the goal of deploying ARM-based supercomputers:

- **United States:** ARM participated in two Department of Energy funded Exascale projects: Data Movement Dominates and FastForward-2.
- **European Union:** Through FP7 and Horizon 2020, ARM has been involved in several funded pre-Exascale projects including the Mont-Blanc program that deployed one of the first ARM prototype HPC systems.
- **Japan:** At ISC2016, Fujistu and RIKEN announced that the Post-K system targeted at Exascale will be based on ARMv8 with the Scalable Vector Extension.
- **China:** James Lin, vice director for the Center of HPC at Shanghai Jiao Tong University claims China will build three pre-Exascale prototypes to select the architecture for their Exascale system. The three prototypes are based on AMD, SunWei and ARMv8.

Section 3 describes the state of the nation in the ARM HPC Ecosystem in terms of the availability of software and hardware solutions. Section 4 gives details about the focus and work to develop future technologies for high-performance and energy-efficient ARM-based node architectures for HPC.

## 3. ARM HPC ECOSYSTEM

The ARM HPC ecosystem is diverse and vibrant across the globe, with an increasing number of partners providing silicon, software, training and design support, and becoming part of consortia that include ARM.

### 3.1 Software Ecosystem for HPC

ARM engages with partners to provide all the pieces necessary for a software ecosystem for HPC, including Linux OS, compilers, libraries, debuggers, profilers and job schedulers. The current offerings for ARM platforms are a mix of open source, commercial products and ported applications and packages.

ARM is a silver member of OpenHPC, a community effort to provide a common, verified set of open source packages for HPC deployments. The OpenHPC build infrastructure uses ARM-based systems to test that all packages successfully build on ARMv8 with the CentOS and SUSE operating systems. Table 1 lists the major included components.

Furthermore, there is a plethora of open source projects for packages and applications that have been successfully

**Table 1: Non-exhaustive list of OpenHPC package components**

| Functional areas | Components |
|---|---|
| Base OS | RHEL/CentOS 7.1, SLES 12 |
| Administrative Tools | Conman, Ganglia, Lmod, LosF, ORCM, Nagios, pdsh, prun |
| Provisioning | Warewulf |
| Resource Mgmt. | SLURM, Munge, Altair PBS Pro |
| I/O Services | Lustre client (community version) |
| Numerical/Scientific Libraries | Boost, GSL, FFTW, Metis, PETSc, Trilinos, Hypre, SuperLU, Mumps |
| I/O Libraries | HDF5 (pHDF5), NetCDF (C++ and Fortran), Adios |
| Compiler Families | GNU (gcc, g++, gfortran) |
| MPI Families | OpenMPI, MVAPICH2 |
| Development Tools | Autotools (autoconf, automake, libtool), Valgrind, R, SciPy/NumPy |
| Performance Tools | PAPI, Intel IMB, mpiP, pdtoolkit, TAU |

ported to ARM HPC. Examples include GROMACS, OpenFOAM, NAMD, QuantumESPRESSO, Geant 4, LAMMPS, WRF, Unified Communication X (UCX), and OpenSHMEM.

ARM also develops and supports a set of commercial HPC products that are jointly tested and tuned for performance:

- ARM C/C++ Compiler: for Linux user-space HPC applications, based on LLVM.
- ARM Performance Libraries: BLAS, LAPACK, FFT, tuned for Cortex-A72, Cortex-A57 and Cortex-A53, maintained and supported for a wide-range of ARM-based systems-on-chip (SoCs).
- ARM SVE C/C++ Compiler: generates SVE code via auto-vectorization, intrinsics and assembly.
- ARM Code Advisor: provides prioritized advice for programmers in-line with original source code.
- ARM Instruction Emulator: runs future ARM architectures on today's hardware, profiles and generates report for Code Advisor.
- Allinea Forge (DDT+MAP): development suite for C, C++ and Fortran including debugger and profiler.
- Allinea Performance Reports: performance analysis and profiling tool for HPC parallel applications.

## 3.2 ARM-based HPC Systems

ARM HPC hardware partners such as HP, Gigabyte and SoftIron provide rackable solutions built with high-end ARM-based SoCs such as Applied Micro X-Gene, Cavium ThunderX and AMD Opteron A1100. Although these parts target server platforms, they also support the features and interfaces necessary for HPC.

There have been several announcements for ARM-based HPC deployments starting in 2017.

- Isambard supercomputer in University of Bristol: Cray CS-400 system with 10 000+ ARMv8 cores and optimized software stack. Installation Mar-Dec 2017 [6].
- Mont-Blanc 3 prototype: Bull (Atos) Sequana system with Cavium ThunderX2 processors [7].
- Post-K supercomputer: Fujitsu HPC system with custom cores implementing ARMv8 and the Scalable Vector Extension [14].

# 4. EXASCALE NODE ARCHITECTURE

ARM is working on several fronts to design the right hardware components, interfaces and software for a node architecture for Exascale. This architecture will integrate future and emerging key technologies to provide high levels of compute performance, scalability, data bandwidth and energy efficiency. Finding the right dimensioning for the best interplay of these three factors is fundamental to achieve a balanced architecture:

- High compute performance: an effective way to achieve high performance and efficiency is the exploitation of data-level parallelism (DLP). Vector processing allows the execution of many operations over many operands in parallel specified by a single instruction. It reduces the need for the core to extract operations to be executed in parallel through instruction-level parallelism discovery, which requires expensive hardware structures for the bookkeeping of a wide window of instructions and speculation mechanisms that, on mispredictions, are detrimental to efficiency. The ARM Scalable Vector Extension (SVE) provides a rich vector instruction set to exploit data-level parallelism and take ARM-based processors to the next performance and efficiency level. More details on SVE in Section 5.
- Application scalability: compute nodes include multiple processing cores targeting thread-level parallelism (TLP). Parallel applications can benefit by partitioning the problem and mapping compute onto cores. This partitioning and scheduling must use the right granularity to provide sufficient parallelism for high core utilization and minimize parallelization overheads.
- 3D die-stacked memory: with high levels of DLP and TLP, the many high-throughput cores accessing data concurrently will put pressure on the memory subsystem, especially in the case of data-intensive applications. 3D die-stacked memories provide high levels of bandwidth in a limited physical space by stacking multiple DRAM dies. Thanks to the use of through-silicon vias (TSVs), a stack of memory can provide hundreds of gigabytes per second (GB/s). The high bandwidth available through the integration of several stacks unlocks higher-levels of performance in the presence of many cores concurrently accessing memory.
- Energy efficiency: achieving high-performance within reasonable power burden is paramount. ARM's experi-

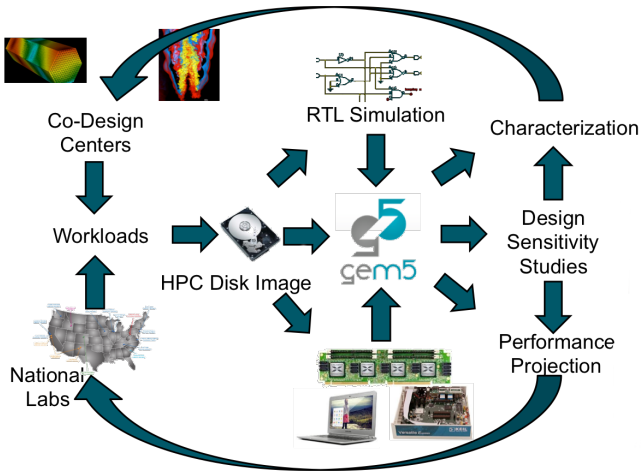


**Figure 1: FastForward-2 co-design cycle.**

ence in designing energy-proportional IP targeting better battery life is an important asset towards energy-efficient HPC systems. Examples of smart power management mechanisms include clock- and power-gating of idle components, dynamic voltage and frequency scaling and efficient waiting capabilities (e.g., wait for interrupt/event).

ARM advocates a co-design approach as a means to achieve balance among these techonologies. Figure 1 shows the co-design cycle strategy used in the FastForward-2 project in collaboration with the US national labs and co-design centers. The co-design centers provide benchmarking workloads. Those are prepared for execution on real hardware, and simulation on detailed RTL models and gem5 [9]. Executions on real hardware and RTL serve as references to calibrate gem5 models for particular core, network and memory implementations. Gem5 is an excellent simulation framework to characterize applications in the target system configuration; explore the design space of the node architecture; and project performance of the target applications on future SoCs integrating emerging technologies. Results are fed back to the national labs and co-design centers to modify their applications to better utilize future technologies.

# 5. SCALABLE VECTOR EXTENSION

The Scalable Vector Extension (SVE) [19, 2] is being added to the ARM ISA to extend its vector processing capability and address the needs of HPC, data analytics, computer vision and machine learning. In particular, SVE is a qualitative jump in ARM's ability to compete in the HPC market. A key goal for SVE has been scalability across multiple implementations, which motivated leaving the vector length as an implementation choice ($\{1$ to $16\} \times 128$ bits). The programming model adapts to the current vector length with no need to modify any executable code.

Longer vectors only achieve significant speedup with high vector utilization. Some of the key SVE features that enable better auto-vectorization support are:

- **Scalable vector length** allowing each implementation to choose the amount of parallelism.
- **Rich addressing modes** including non-linear data accesses.
- **Per-lane predication** allowing vectorization of loops with complex control flow.
- **Predicate-driven loop control and management** to reduce vectorization overhead relative to scalar code.
- **A rich set of horizontal operations** for reducible loop-carried dependencies.
- **Vector partitioning and software-managed speculation** to vectorize loops with data-dependent exits.
- **Scalarized intra-vector sub-loops** to allow vectorizing loops with complex loop-carried dependencies.

## 5.1 Main features

SVE is designed for scalability, reasonable implementation complexity and high impact across multiple application domains. The scalable vector length avoids the need to create a different instruction set for each new vector width to be implemented, which would not be viable within the 32-bit encoding space in the ARM ISA. The new architectural state includes 32 new vector registers (Z0-Z31) that extend and overlap with the 32 128-bit-wide Advanced SIMD registers (V0-V31), and can contain 64-, 32-, 16-, and 8-bit data elements. SVE also adds 16 scalable predicate registers (P0-P15), a special purpose first-faulting register (FFR), and a set of control registers (ZCR_EL1-ZCR_EL3) that allow each exception level to virtualize the effective vector width.

SVE enables the software to be vector-length agnostic through the use of vector partitioning, i.e., operating on a partition of safe elements depending on dynamic conditions, including loop termination conditions and exceptions. Vector partitioning allows vectorization of loops without an explicit iteration count by relying on predication. Predication is central to SVE, since it is used to control loops. SVE introduces a set of `while` instructions that use the iteration counter and limit to generate predicates and update the condition flags efficiently.

Transferring data between memory and vector registers with SVE vector load and store instructions is flexible and unlikely to limit vectorization. Vector loads/stores are able to read/write contiguous elements into/from one or multiple vector registers, as well as non-contiguous elements (gather/scatter operations) using rich addressing modes.

Effective speculative vectorization requires efficiently dealing with data-dependent termination conditions and exceptions. SVE introduces a *first-fault* mechanism for vector load instructions that suppresses memory faults if they are not produced by the first active element in the vector, and sets the first-fault register (FFR) to indicate which elements were not successfully loaded. Non-faulting elements are processed first, the fault is serviced with the correct architectural state, and the loop continues with the rest of the elements in the vector, preserving sequential program order without sacrificing efficiency. This mechanism enables safe vectorization of loops with data-dependent terminations that would not normally be vectorizable, e.g., the C *strlen* function.

Complex loop-carried dependencies introduce serialization in an otherwise vectorizable loop. Techniques like loop splitting require unpacking and packing the data to work on it serially, negating the benefit of vectorization. SVE reduces that cost by providing instructions to serially process elements within a vector without having to unpack it.

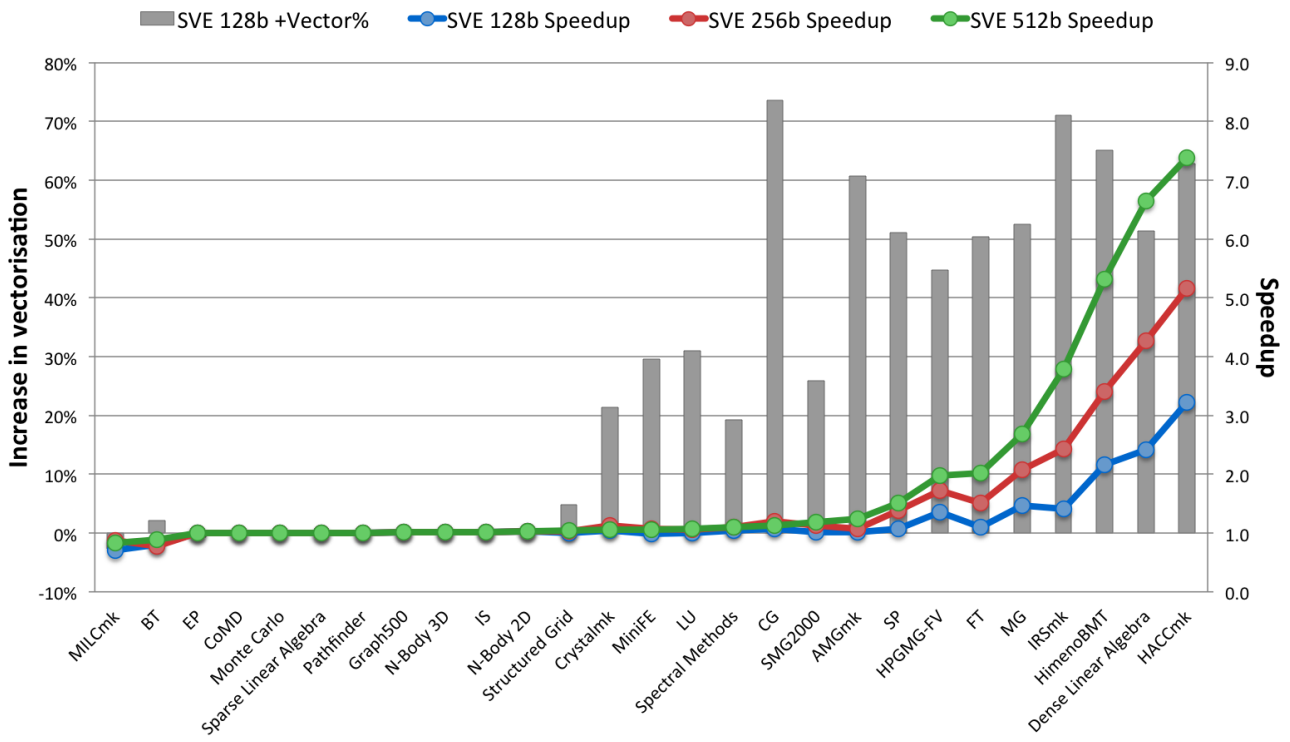


**Figure 2: Performance of SVE with vector lengths of 128, 256 and 512 bits.**

Dependencies across multiple loop iterations that can resolved with simple horizontal reduction operations do not prevent vectorization since SVE has a rich set of logical, integer and floating-point horizontal reductions, including strictly-ordered reductions for floating-point.

## 5.2 Compilation Challenges

Lacking the knowledge of vector length at compile time complicates compilation. A simple approach like first unrolling a loop by the number of elements in a vector and merging scalar operations into vector operations does not work. SVE introduces a family of instructions that use the current vector length as an implicit operand, e.g., the `index` instruction initializes a vector induction variable, and the `inc` instructions increments the induction variable based on the current vector length and a specified element size.

Stack space used by a vector register during spilling and filling is dependent on vector length. Existing stack regions are accessed with the statically known byte offsets, while regions for SVE registers are dynamically allocated and use implicit multiples of vector length as load and store offsets.

Predicates are introduced by conventional *if-conversion*. Conditional branches that exit the loop require the insertion of a `brk` instruction that generates a vector partition where only the lanes prior to the loop exit condition are active.

Vectorizing a reduction loop changes the order of floating-point operations, potentially producing a different result than scalar code. Ordering may also differ for different vector lengths. This can be avoided by disabling vectorization or using a strongly-ordered reduction operation (`fadda`).

## 5.3 Performance

Performance of SVE is heavily dependent on two factors: the coverage of auto-vectorization achieved by the compiler and the microarchitectural implementation. We expect the SVE architecture to be implemented by multiple partners with a variety of target markets and microarchitectures.

Figure 2 shows the speedup and increase in vectorization compared to Advanced SIMD, for a variety of HPC applications from well known benchmark suites. Vectorization is defined as the fraction of dynamically executed instructions at a vector length of 128 bits. We simulate a medium-sized, out-of-order processor that does not correspond to any real design but is representative of the state-of-the-art. The main parameters for this model are shown in the reference SVE paper [19, Table 2]. Our evaluation uses an experimental compiler that is able to auto-vectorize code for SVE, but is limited to only C and C++ applications.

For several applications on the right side of the figure, SVE vectorization is significantly higher than with 128-bit Advanced SIMD due to all the SVE features that allow the compiler to vectorize loops with complex control flow, loop-carried dependencies, non-contiguous memory accesses, etc. This additional vectorization translates in significant speedups of up to 3x even for 128-bit SVE. These applications also scale well with longer SVE vectors, up to 7x with 512-bit SVE. Performance improvement is not as high for some heavily vectorized applications in part because we assume a conservative gather/scatter implementation that cracks those operations into one memory access per vector element, which becomes a limit for scalability due to the higher pressure on the load-store pipeline. This is an ar-

tifact of our early modeling efforts and not necessarily the intended implementation.

On the left of the figure there are benchmarks with very low or zero vector utilization for both Advanced SIMD and SVE. Closer investigations showed that the lack of vectorization is due to the way the code is structured in these unmodified benchmarks, or to limitations on the compiler, but not due to limitations in the architecture. We know that restructuring the code in *CoMD*, for example, can significantly improve vectorization and reduce execution time. Results of a refactored CoMD backed by high-bandwidth memory show over 4× speedup when scaling from 128b to 1024b vectors. The use of some non-vectorized versions of basic math functions like `pow()` and `log()` prevents vectorization, e.g., in *EP*. *Graph500* mostly traverses graph structures chasing pointers, an implementation that is not straightforward to vectorize. Refactoring algorithms with vectorization in mind could provide significant benefit from SVE.

A third class of benchmarks in the middle of the figure shows good additional vectorization with SVE that does not result in significant speedups due to code generation issues with the compiler. Sub-optimimal instruction selection and excessive use of gather instructions (*SMG2000*) or vectorizing the outer loop instead of the inner loop and missing trivially vectorizable loops (*MILCmk*) are some examples. We expect SVE compilers and libraries to significantly improve over time.

# 6. CONCLUSIONS

ARM's efforts to provide technologies for high-end computing platforms is enabling ARM HPC systems. Examples of these technologies are the 64-bit ARMv8 architecture, the Scalable Vector Extension, the inclusion of RAS support, and the development of open source and commercial HPC software solutions for ARM SoCs.

These enabling technologies have evolved within funded international projects focused on research and development of ARM-based HPC. The efforts within the Mont-Blanc project helped provide ARM-based HPC clusters for development and porting of applications and the HPC software stack. The FastForward-2 project provides a co-design framework to explore future HPC technologies and improve reference workloads in the US national labs and co-design centers to efficiently use those technologies.

The Scalable Vector Extension represents the reemergence of vectors as an effective solution providing high-performance and power efficiency. The unique features of SVE, such as vector-length agnosticism, predication and speculative vectorization enable hardware implementations with different vector widths together with compilers to implement aggressive auto-vectorization techniques and improve vectorization coverage across a wider range of codes.

The efforts of the ARM ecosystem towards HPC, early foundational work and recent innovations such as architecture extensions, are poised to show compelling visible results in the first generation of ARM-based supercomputers in the next few years.

# 7. ACKNOWLEDGMENTS

This work has been supported by the FastForward-2 (grant no. US-DoE-B6098229) and Mont-Blanc 3 (grant no. EU-H2020-671697) projects.

# 8. REFERENCES

[1] http://www.montblanc-project.eu.

[2] A-Profile Architecture Specifications. http://developer.arm.com/products/architecture/a-profile/docs.

[3] CCIX: Cache Coherent Interconnect for Accelerators. www.ccixconsortium.com.

[4] GenZ Consortium. www.genzconsortium.com.

[5] HPC on ARM. http://arm.com/hpc.

[6] https://www.top500.org/news/cray-to-deliver-arm-powered-supercomputer-to-uk-consortium, Jan. 18 2017.

[7] https://www.top500.org/news/mont-blanc-project-teams-with-cavium-and-bull-to-build-arm-based-supercomputer, 117 2017.

[8] M. Alian, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim. dist-gem5: Distributed Simulation of Computer Clusters. In *ISPASS'17*, 2017.

[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 Simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[10] A. Ferreron, R. Jagtap, S. Bischoff, and R. Rositoru. Crossing the Architectural Barrier: Evaluting Representative Regions of Parallel HPC Applications. In *ISPASS'17*, 2017.

[11] T. Grass, C. Allande, A. Armejach, A. Rico, E. Ayguadé, J. Labarta, M. Valero, M. Casas, and M. Moreto. MUSA: a Multi-level Simulation Approach for Next-generation HPC Machines. In *SC'16*, page 45, 2016.

[12] T. Grass, A. Rico, M. Casas, M. Moreto, and E. Ayguadé. Taskpoint: Sampled simulation of task-based programs. In *ISPASS'16*, pages 296–306, 2016.

[13] R. Grisenthwaite. ARMv8 Technology Preview. ARM TechCon, 2011.

[14] Y. Ishikawa. The Next Flagship Supercomputer in Japan. In *ICS'16*, 2016.

[15] R. Jagtap, S. Diestelhorst, and A. Hansson. Elastic traces for fast and accurate system performance exploration. In *ISPASS'16*, pages 147–148, 2016.

[16] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero. Supercomputing with commodity CPUs: Are mobile SoCs ready for HPC? In *SC'13*, page 40, 2013.

[17] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, et al. The Mont-Blanc Prototype: An Alternative Approach for HPC Systems. In *SC'16*, pages 444–455, 2016.

[18] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez. Tibidabo: Making the case for an ARM-based HPC System. *Future Generation Computer Systems*, 36:322–334, 2014.

[19] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. The ARM Scalable Vector Extension. *IEEE Micro special issue on Hot Chips*, 37(2), 2017.