This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

arXiv:2505.09462v1 [cs.DC] 14 May 2025

ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace

Ruimin Shi[1[0009−0003−4387−367X]], Gabin Schieffer[1[0009−0003−6504−7109]], Maya Gokhale[2[0000−0003−4229−5735]], Pei-Hung Lin[2[0000−0003−4977−814X]], Hiren Patel[3[0000−0003−2750−4471]], and Ivy Peng[1[0000−0003−4158−3583]]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
[2] Lawrence Livermore National Laboratory, Livermore, USA
[3] University of Waterloo, Waterloo, Canada

Abstract. Vector architectures are essential for boosting computing throughput. ARM provides SVE as the next-generation length-agnostic vector extension beyond traditional fixed-length SIMD. This work provides a first study of the maturity and readiness of exploiting ARM and SVE in HPC. Using selected performance hardware events on the ARM Grace processor and analytical models, we derive new metrics to quantify the effectiveness of exploiting SVE vectorization to reduce executed instructions and improve performance speedup. We further propose an adapted roofline model that combines vector length and data elements to identify potential performance bottlenecks. Finally, we propose a decision tree for classifying the SVE-boosted performance in applications.

1 Introduction

The landscape of processors on high-performance computing (HPC) systems has changed. For a long time, ARM processors have been dominating the embedded system market for their power efficiency, licensing flexibility, and wide toolchain support. In contrast, most HPC platforms have been powered by x86 processors, as represented by Intel and AMD processors. Although x86 processors are still used in many supercomputers, in the latest Top 500 list, 2 out of the Top 10 supercomputers in the world are powered by ARM processors, including the Supercomputers Fugaku [14] and Alps. Also, Jupiter, the upcoming exascale supercomputer in Europe, will be powered by ARM processors. This trend indicates that server-class ARM processors have emerged as a strong contender in high-end computing systems, especially due to increasing energy concerns, endorsement from vendors like Amazon and Nvidia, and diminishing gains from x86 processors [23, 16].

Recent server-grade ARM processors have started to use the ARM scalable vector extension (SVE) [17] to increase computing throughput through vectorized instruction execution. Unlike conventional SIMD engines that only support

pre-print submitted for publication

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

a fixed vector length, SVE instructions can support a variable vector length by masking the predicate registers. This vector length agonistic (VLA) design is also used in the RISC-V vector extension (RVV) [3, 8]. Unlike fixed-length SIMD designs, which require scalar loops to handle leftover tailing elements in irregular loops, and require instruction set expansion for every new vector length, SVE and RVV can support different vector lengths in one instruction set. Previous works [22, 21, 18] have optimized selected applications on specific ARM architectures. However, understanding the readiness and maturity to leverage ARM SVE in HPC applications remains an open question.

This work aims to provide a first answer to the question. We propose a benchmark suite to reflect the evolving workload mixtures on HPC systems, including 13 applications from different domains, such as machine learning, drug discovery, scientific simulations, and quantum computing. This benchmark suite represents various code complexity, compute intensity and memory access patterns. We focus on compiler autovectorization in these applications as it is likely the most used approach for exploiting ARM SVE in existing applications. On a real hardware implementation of ARM SVE– the Nvidia Grace processor–we leverage profiling and analytical models to evaluate how vectorized code can reduce the overall executed instructions and improve performance quantitatively.

We validate relevant performance hardware events on the ARM Grace processor and select a small set of events for deriving performance metrics. Though ARM processors have improved their support for PMU, there is still limited study of performance events on ARM processors. Based on the profiling results, we further propose an adapted roofline model that combines vector length in SVE architecture and data elements in applications to identify potential performance bottlenecks in applications. Guided by the roofline mode, we identified that both increased vector length and reduced data element sizes (via reduced data precision) could transform some compute-bound workloads into memory-bound workloads, highlighting the importance of matching memory subsystems on the vector architectures. Due to double-precision data formats, HPC applications cannot exploit short SVE like Grace CPU as much as single-precision machine learning workloads.

We propose a decision tree for classifying the performance speedup in applications on ARM processors with SVE into four classes. On 26 tested cases, 15 cases achieved speedup by ARM SVE without any porting efforts and 6 cases can be vectorized in compilation, have reduced retired instructions but cannot achieve performance speedup due to memory bound. In summary, we made the following contributions in this work.

- We provide a benchmark suite of 13 applications to assess the maturity level of exploiting ARM SVE in HPC applications.
- We propose new metrics and validate hardware counters on Nvidia Grace processor for quantifying the vectorization effectiveness and identifying performance bottlenecks.
- We identify performance bottlenecks using an adapted roofline model combing vector lengths in SVE architecture and data element sizes in applications.

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, “ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace,” in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

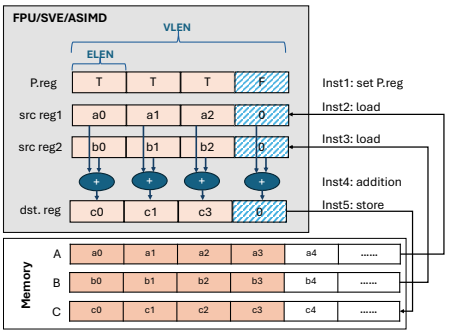


Fig. 1: SVE supports variable vector length by masking predicate registers.

Scalar version

```
.L3:
  ldr     d0, [x1, x4, lsl 3] // d0 = a[i]
  ldr     d1, [x2, x4, lsl 3] // d0 = b[i]
  fadd    d0, d0, d1          // d0 = a[i] + b[i]
  str     d0, [x0, x4, lsl 3] // store d0
  add     x4, x4, 1           // i = i + 1
  cmp     x3, x4              // continued until (i < size)
  bne     .L3
```

SVE version

```
  whilelo p0.d, xzr, x9                      //x9: array size
.LBB0_1:
  ld1w    { z0.d }, p0/z, [x1, x8, lsl #3] //z0 = a[i:i+VL-1]
  ld1w    { z1.d }, p0/z, [x2, x8, lsl #3] //z1 = b[i:i+VL-1]
  add     z0.d, z0.d, z1.d                 //z0 = z0 + z1
  st1w    { z0.d }, p0, [x0, x8, lsl #3]   //store z0
  incd    x8                               //x8 = x8 + VL
  whilelo p0.d, x8, x9                     //set p0
  b.mi    .LBB0_1
```

Fig. 2: Assembly code of a simple vector-vector addition kernel.

– We propose a decision tree for classifying the performance impact on ARM SVE and validate it in 26 cases on Nividia Grace processor.

2 Background

ARM Scalable Vector Extension. Vector architectures explore data-level parallelism by simultaneously processing multiple elements in one instruction. Effective vectorization can reduce the number of retired instructions, improve the alignment of memory accesses, and increase computing throughput via parallel processing in arithmetic logic units (ALUs). ARM Scalable Vector Extension (SVE) is a long vector architecture introduced by the ARM A64 instruction set as part of the ARMv8-A and ARMv9-A architecture [13]. It has 32 vector registers and 16 predicate registers. Recent hardware implementations of ARM SVE include the Fujitsu A64FX processor (vector length 512-bit), Neoverse V1-based AWS Graviton processor (vector length 256-bit), and Neoverse V2-based Nvidia's Grace processor (vector length 128-bit). ARM SVE exploit Vector Length Agnostic (VLA) programming models with vector registers ranging from 128-bit to 2048-bit at 128-bit increments on various architecture implementations. `VL` represents the number of elements operated by a specific instruction. Figure 1 illustrates a simple example where the fourth element is masked by setting predicate register (P.register) to false, and thus only three results are written back to the memory. In this way, SVE achieves fine-grained control of each vector element through the setting of a predicate, enabling seamless handling of edge cases and exploring irregular data patterns. Figure 2 compares the scalar and SVE vectorized implementations of a simple kernel performing element-wise addition of two vectors.

Software Support. To develop applications that exploit SVE engines, there are multiple approaches with different trade-offs between programming complexity and vectorization efficiency. Modern compilers support automatic vectorization that transforms a scalar code into a vectorized code using vector instruc-

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

tions. Auto-vectorization reduces programming complexity compared to writing SVE assembly, using SVE intrinsics, or relying on highly optimized ARM libraries. By setting proper optimization flags, compilers can generate efficient vectorized codes without any rewriting. Auto-vectorization typically happens in loops, consecutive memory accesses, and tree data structures. For SVE, the GNU tools version 8.0+ and Arm Compiler (based on LLVM Clang) have vectorizers that detect suitable scalar operations to be optimized with SVE instructions. Auto-vectorization greatly reduces the programming overhead and improves portability across platforms. However, compiler-based auto-vectorization may suffer from insufficient optimization in complex codes. In this work, we focus on compiler-based auto-vectorization as the main approach for getting SVE adopted in realistic parallel codes, running on HPC systems.

2.1 Related Works

Compiler Support. Compiler support for vectorization have been widely studied. The vector effectiveness across the SVE-support compiler on mini-apps and SVE uasge are analyzed from instruction level [12]. Source-to-source compiler-performed transformation has also been proposed as a solution [15, 4]. Specific auto-vectorization improvements have been proposed to compilers and runtimes [10]. Instead of targeting application- or compiler-specific optimizations, our work assesses the opportunities and impact of compiler auto-vectorization for a variety of applications.

Algorithm Co-design. Several algorithms have been ported to use ARM SVE, focusing on providing insight on key design choices for vector architectures, either on simulators or actual hardware. These algorithms include the GEMM dense matrix-matrix multiplication routine [21, 19] and Convolutional Neural Network (CNN) [6]. These works demonstrated the speedup of SVE vectorization for applications and presented different co-design methods and their effects. For guiding porting efforts, our work provides theoretical explanation, backed by a roofline model, to identify potential benefits of vectorization.

Application Porting. Porting efforts have been deployed to leverage ARM SVE in application codes, by using intrinsics. Examples include quantum simulator [18], LLM model training [13], DNA alignment tool [7], and the widely-used NumPy Python library [22]. We provide a method for application developers to identify potential benefits – or non-benefits – of auto-vectorization, based on their application's characteristics.

3 Methodology

Auto-vectorization We investigate both GCC compiler and ARM compiler for autovectorizing HPC applications. Our results indicate that binaries vectorized by GCC compiler obtain better performance than the ARM compiler for most applications in test. Thus, if not specified otherwise, GCC compiler is used. We use three sets of compilation flags to create three versions for each application.

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

First, the baseline version that only uses scalar instructions (denoted as Baseline) is obtained by disabling all vectorization options. In particular, we used `-fno-tree-vectorize` to disable vectorization on trees, `-fno-tree-loop-vectorize` to disable loop vectorization, and `-fno-tree-slp-vectorize` to disable the basic block vectorization. The second version uses the default Advanced SIMD (denoted as ASIMD) for vectorization, which is compiled with optimization flags `-march=armv8-a+simd` and `-mcpu=neoverse-v2`. Finally, a third version (denoted as SVE) is created by specifically enabling the SVE vectorization using compilation flags `-march=armv8.5-a+sve` and `-mcpu=neoverse-v2`. We also verify the vectorization of generated codes by using `$(CC) -S` to dump assembly code into `.s` files, then searching the identifiers of predicate and vector registers, such as `z0-z31`, `v0-v31` and `p0-p15`, to locate the vectorized code regions, and then confirm correct vector instructions are in use.

Experimental Platform We conduct our experiments on real hardware by using a testbed of the Nvidia Grace CPU [11, 16]. The processor features 72 Armv9-A Neoverse V2 cores [1], equipped with 480 GB LPDDR main memory. It has L1 64KB I-cache and 64KB D-cache per core, 1MB L2 cache per core, and 117 MB LLC. The core also implements four 128-bit SIMD functional units, which are able to execute both SVE/SVE2 instructions and Advanced SIMD (also known as NEON) instructions. In this architecture, the maximum CPU frequency is 3447 MHz with four FPU pipelines per core, and the memory bandwidth tested by STREAM Triad benchmark is 30 GB/s and 250 GB/s at 1- and 72-threads, respectively. The system runs RHEL 9.4 with Linux kernel 5.14. GCC 11.4 compiler and ARM clang 23.10 are used on the platform.

3.1 Profiling Approaches

Table 1: ARM PMU events used for profiling on the ARM testbed.

Hexcode	Event Name	Description
0x8	INST_RETIRED	Instruction executed
0x37	LL_CACHE_MISS_RD	LLC read, miss
0x66	MEM_ACCESS_RD	Memory access, load
0x24	STALL_BACKEND	Cycles due to backend stall
0x11	CPU_CYCLES	Cycles
0x75	VFP_SPEC	Floating-point instruction

Table 2: Benchmark suite

Application	**Kernels**	**Problems**
LLM training	train	124M
LLM inference	test	124M
QC simulator	RX_gate	21 qubits
FFT1D	fft1D	16384
FFT2D	fft2D	262144
STREAM	copy	1-10G
DGEMM	dgemm (FP64)	12kx12k
SGEMM	sgemm (FP32)	12kx12k
SPMV	spmv_csr	2048^2
Jacobi2D	sweep	4-32k
YOLOv3	detector	$608^2 \times 3$
AlexNet	classifier	1k
AutoDock	scoring	`1iep` complex

We extend a lightweight profiler library based on `perf` to profile the instruction and memory details of selected kernels [5, 9]. With this profiler, we can collect the hardware counters provided by the ARM PMU in regions of interest (ROI). This profiler provides simple API in C/C++: `configure_measure()` will configure and initialize the counters; `start_measure()` and `stop_measure()`

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

will enable/resume and disable/pause counting for the hardware events; `print_results()` will print the value of results into the terminal.

The profiler uses the Linux system call `perf_event_open` to create a special file descriptor, each recording the measurement from an event. We group multiple descriptors together to set up different events at one time. In Neoverse V2 cores, at most six events can be collected simultaneously. The configuration structure of events uses `PERF_TYPE_RAW` where an event hexcode can be looked up for the specific hardware implementation. Table 1 lists the events used in this work. For instance, we use retired instructions to quantify the vectorization effectiveness in the two vectorized versions, i.e., the ASIMD and SVE versions. Compared to static instructions that remain constant for the same executables, dynamic retired instructions reflect executed instructions on hardware.

We validated a set of hardware counters provided by ARM PMU and found that some of them are not stable or accurate enough to be used in calculating evaluation metrics, like `STALL_BACKEND_MEM`, `L3D_CACHE_LMISS_RD` and `SVE_INST_SPEC`. Perf also provides `simd_percentage` metric that is defined as the ratio between `ASE_INST_SPEC` and `INST_SPEC` to represent the impact of vectorization. However, since both the numerator and denominator are capturing only speculatively executed instructions, `simd_percentage` cannot reflect the overall reduction of total executed instructions. Instead, `INST_RETIRED` is the architecturally executed instructions, which are more reliable without the interference of speculative execution in the superscalar processor.

3.2 The Benchmark Suite

We compose a diverse set of benchmarks that covers different application domains on HPC systems, code complexity levels, and various compute intensity and memory access patterns. We selected 13 applications from scientific simulation, machine learning, and quantum computing. We leverage our profiler to focus on key computational kernels and exclude initialization, preprocessing, and finalization stages, such as reading and preprocessing the input image in YOLOv3 and AlexNet, and loading real data matrices in SpMV. Table 2 summarizes these applications with their respective largest input problems used, along with key computational kernels. These workloads support multi-threaded execution and we set the environment variables `OMP_NUM_THREADS` to control the thread count. Each experiment is repeated at least five times. We always guarantee the execution time of tests above 0.1s and the standard deviation within 5%.

We also propose a synthetic benchmark based on SpMV ($y = Ax$) to configure different compute intensities and data formats. Assuming the sparse matrix A is stored in CSR format and `x` is the vector, each row is accessed by iterating through its nonzero elements, calculating `temp = val[j] * x[colind[j]] + temp` to accumulate the results in spmv. Within one loop, three memory accesses are issued, and the load of `x[colind[j]]` is usually from the main memory due to its pointer-chasing nature with only two operations: `*` and `+`. To increase compute intensity, we repeat the computation for a configurable number of times,

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

e.g., 20 in this example. To ensure compilers do not optimize or eliminate dead code automatically, this region uses `#pragma unroll loop(1)` and disables the dead-code elimination (DCE) optimization flag. In the modified version, the corresponding values request of `colind`, `val` and `x` stays in L1D cache after the first computation. When the number of repeated computations increases, the benchmark can transform from memory-latency bound to compute-bound.

3.3 Analytical Models

We leverage analytical modeling to derive a set of theoretical metrics to guide the evaluation. First, we derive the theoretical upper bound of the vectorization ratio, based on the maximum vector length (VLEN) and data element formats (ELEN), as shown in the following equation for VB. To approximate ELEN in a kernel, we choose the dominant data formats, i.e., if the main computation is in double-precision, FP64 will be used.

We then use the profiler and captured events to obtain the achieved overall instruction reduction. For this, we define a metric called instruction reduction ratio (denoted as $R_{ins_reduction}$) to quantify the end-to-end effectiveness of exploiting vectorization for a given scientific problem. As shown in the equation below, it is defined as the ratio between the number of retired instructions using the non-vectorized version with the two vectorized versions using SVE and SIMD, respectively. $R_{ins_reduction}$ quantifies the reduction of total retired instructions for solving the same computation, i.e., instructions to a solution. This metric could reflect the impact of vectorized instructions in overall instructions. If vectorized instructions only compose a small fraction of all executed instructions, this application cannot effectively exploit vectorization for acceleration.

$$Vectorization\ Bound(VB) = \frac{VLEN}{ELEN} \qquad R_{ins_reduction} = \frac{Ins_{nonvec}}{Ins_{simd|sve}} \tag{1}$$

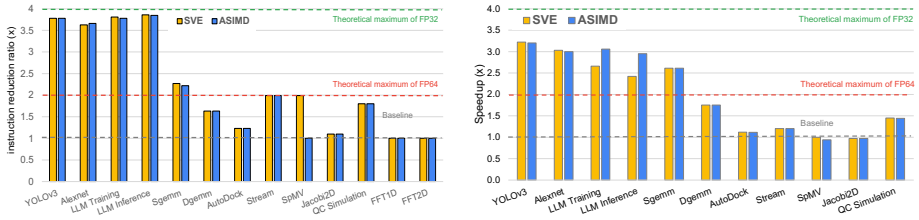
Finally, we adapt the roofline model [20] to capture the vectorization bound in a computation kernel. We leverage the adapted roofline model to identify performance bottlenecks. The inflection points on the roofline model for scalar and SVE, shorted respectively as IRR and IRV, are defined as

$$AI_{IRR} = \frac{Peak\ Compute\ Throughput}{PeakBW} \qquad AI_{IRV} = AI_{IRR} * \frac{VLEN}{ELEN} \tag{2}$$

PeakBW represents the peak achievable memory bandwidth on the platform. If the arithmetic intensity (AI) of a kernel is smaller than the inflection point, this application is memory-bound, and increasing the memory bandwidth is the key optimization direction while vectorization cannot bring performance benefits. If its AI is greater than the inflection point, the application is compute-bound, and increasing the peak performance via vectorization can bring performance improvement.

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.



(a) Instruction reduction ratio (b) Achieved performance speedup

Fig. 3: Measured metrics in the 11 workloads that can be autovectorized using SVE and SIMD, respectively.

4 Vectorization Effectiveness

We present an overview of vectorization effectiveness in 13 applications by quantifying the reduced ratio of total executed instructions, i.e., $R_{ins_reduction}$, in single thread execution in Figure 3a. On the Grace CPU, since the maximum vector length is 128 bits, the upper vectorization bounds (VB) for FP64 and FP32 data elements are 2× and 4×, respectively, as indicated by the two dashed lines on Figure 3a. For each vectorized code, we also check its assembly code to confirm vectorization instructions are used. 11 out of 13 applications can be vectorized by compilers (via checking assembly code), and also have $R_{ins_reduction} > 1$. Many of the workloads achieved reduction ratios close to the vectorization bound. For instance, YOLOv3, AlexNet, LLM training, and LLM inference are single-precision workloads with $VB = 4$ and they achieved 3.6-3.8× reduction in retired instructions. DGEMM, STREAM, and quantum circuit simulation are double-precision workloads with $VB = 2$ and they achieved 1.6-1.8× reduction.

Only one application, the FFT benchmark in 1D and 2D, has limited auto-vectorization. This benchmark is implemented atop the FFTW subroutines library. By analyzing the source code and assembly code, we attribute this lack of auto-vectorization to complex intrinsic and pre-optimization based on the Radix-N algorithm in library design, requiring manual porting efforts for effective utilization of vector instructions. Note that the latest FFTW supports the ARM Neon extension, but still lacks support for SVE.

SVE and Advanced SIMD have similar vectorization ratios for 12 benchmarks, except SpMV. The SpMV benchmark shows the advantage of SVE in processing the dynamic irregular loop length. In this benchmark, the loop lengths vary because the number of non-zero elements in each row of the sparse matrix is different, and this variability is challenging for the compiler to determine at compilation time before execution. Unlike the advanced SIMD, which relies on padding with scalar instructions on tailing elements or programmers' efforts to match the fixed vector length, SVE can use the predicate registers to manage the variable vector length at runtime. As a result, SVE achieved a 1.99× instruction reduction ratio, whereas advanced SIMD only reaches 1.0×.

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

We further compare the achieved performance speedup (Figure 3b) with the reduced instruction ratios (Figure 3a) for 11 applications in the benchmark suite, except the FFT benchmarks that cannot be vectorized. The four applications that have the highest reduction of retired instructions, i.e., YOLOv3, AlexNet, LLM training, and LLM inference, also achieved the highest performance speedup of $2.4 - 3.2\times$. However, the performance speedup in double-precision workloads is more diverse, DGEMM and QC simulation achieved $1.5 - 1.8\times$ speedup, which is consistent with their high $R_{ins_reduction}$. However, STREAM and SpMV show little performance improvement from vectorization, even though their retired instructions are effectively reduced by almost $2\times$.

Though most applications have the same speedup using SVE and ASIMD, LLM training and inference have a higher speedup with ASIMD than SVE. In LLM applications, the loop length in every layer is variable and parts of them have the dependency of previous results. The overhead of frequently setting the vector length using the dynamic vector length of SVE in runtime is not negligible. This may result in a lower speedup on SVE than ASIMD, which uses the fixed vector length.

The overall evaluation in auto-vectorization, reduction of instructions, and achieved performance in 13 applications indicates that on recent platforms like Grace, the maturity of compiler support for SVE is already comparable to the long-term maturity in support for the advanced SIMD. The support for single-precision workloads is better than double-precision workloads. Since HPC applications mostly use double-precision floating-point data formats, they may have a lower chance of speedup on ARM processors that implement short SVE, compared to other workloads.

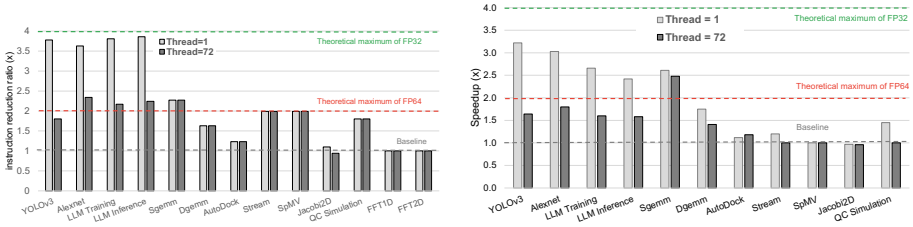
4.1 Impact of Thread Counts

We compare the reduction ratios of total executed instructions at two thread counts in Figure 4a. Applications other than YOLOv3, AlexNet, and LLM training and inference maintain similar ratios at different threads, indicating that their total retired instructions are still reduced in the vectorized version when running with multiple threads. YOLOv3, AlexNet, and LLM applications are more complex applications with multiple tasks synchronized and control paths in parallelized loops. Their single-threaded runs achieve a good reduction ratio of retired instructions using vectorized versions. However, the reduction ratios at 72-threaded runs are much lower than those at a single-threaded run. One possible explanation could be the overhead in scheduling and synchronization on a large number of threads, when the codes call the dynamic OpenMP library to manage the multiple threads. The achieved performance speedup at different numbers of threads is presented in Figure 4b. As expected from their instruction reduction ratios, YOLOv3, AlexNet, LLM training and inference, show significant differences in speedup at the two thread counts.

Some applications, such as STREAM and quantum circuit simulations, show performance speedup from vectorization on single-threaded runs. However, with 72 threads, they no longer exhibit any speedup from the vectorized version even

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.



(a) Instruction reduction ratio (b) Achieved performance speedup

Fig. 4: Measured metrics in 13 workloads using SVE on Nvidia Grace processor at single and 72 threads.

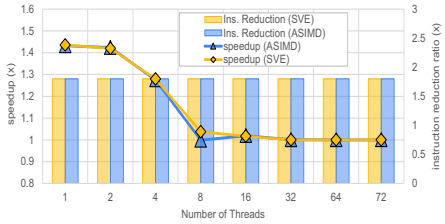


Fig. 5: The speedup in the quantum circulation simulation.

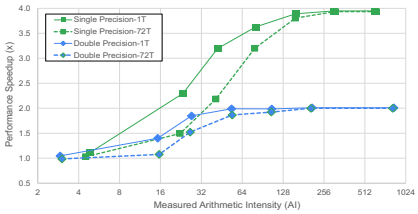


Fig. 6: The speedup of the synthetic benchmark.

though the number of retired instructions is still effectively reduced in these runs. This behavior can likely be attributed to the increased memory contention when a large number of threads are running, potentially limiting the memory bandwidth and thus offsetting the potential performance speedup from using vectorization. Figure 5 presents the sensitivity test of a state-vector quantum circuit simulation of RX Gates at increased number of threads. Speedup of QC simulator decreases rapidly until the thread count equals 8, indicating the shift to memory-bound, and the saturation of memory bandwidth at 8 threads.

4.2 Impact of Data Element Size

We use the modified SpMV benchmark presented in Section 3.2 to assess the impact of element length on ARM SVE. We focus on SpMV as the overall performance results show that SVE is more flexible than SIMD for vectorizing irregular codes. Floating-point formats are ubiquitous in HPC applications, and on ARM architectures, three IEEE 754 compliant floating-point formats, i.e., double-precision (FP64), single-precision (FP32), and half-precision (FP16), are supported [2]. Thus, we use these three data formats in the synthetic benchmark to evaluate 16-, 32-, to 64-bit element lengths. Given the 128-bit SVE vector length, each vector register should be able to hold 8, 4, and 2 data elements, respectively. However, half-precision (FP16) is excluded due to the lack

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, “ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace,” in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

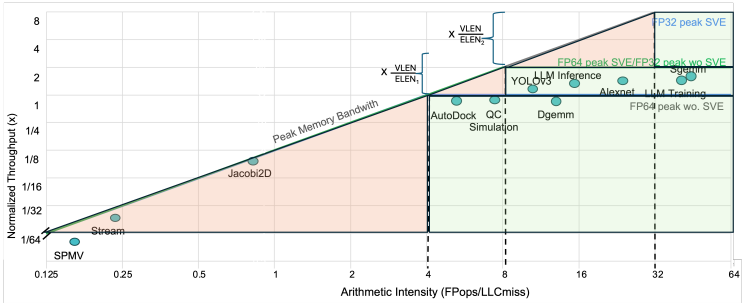


Fig. 7: A Roofline Model that captures peak vectorization on a system with 128-bit SVE for single and double precision applications.

of compiler support, i.e., GCC compiler throws compilation errors while ARM clang cannot generate vectorized code with correct `.h` instructions.

Figure 6 presents the achieved speedup at different data element types and the measured arithmetic intensities. We changed the arithmetic intensity by increasing the computation repeat times, as described in Section 3.2. For retired instructions, the reduction ratios are 2× in FP64 and 3.5× in FP16. It is clear from Figure 6 that the performance speedup steadily increases as the compute intensity increases and saturates at around the calculated vectorization bound (VB), which equals 2 and 4 for FP64 and FP32, respectively. When the measured arithmetic intensity is low, their performance is bounded by memory, their achieved performance speedup is not proportional to the reduction ratio of retired instructions.

To further investigate the compiler support for FP16, we also performed a test by changing the data types in the STREAM benchmark. We find that both GCC and ARM clang can correctly vectorize the code with `.h` instructions. Since the benchmark is bound by the memory bandwidth, no performance speedup is observed. However, the ratio of reduced instructions closely approximates the vectorization bound. When using GCC compiler, the instruction reduction ratio achieves ×2, ×4 and ×7.1 for FP64, FP32, and FP16, respectively; while ARM clang reaches ×2, ×4.4, ×3.5.

5 Performance Modeling

We leverage a roofline model that combines SVE VLEN and applications ELEN to identify performance bottlenecks and further propose a decision tree that takes in profiling metrics from the non-vectorized version for classifying its performance impact on SVE-supported platforms.

Roofline model. We propose a roofline model with extension for SVE architectures, as illustrated in Figure 9, to guide optimization directions in applications. We normalize the peak compute throughput with respect to the peak FP64 throughput without vectorization. In an ideal case, if an FP64 kernel

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

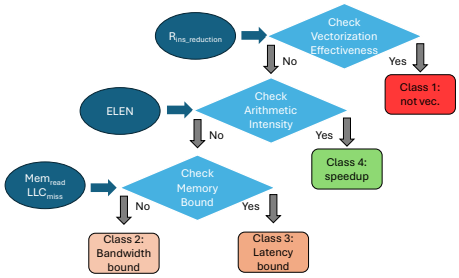
R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

is perfectly vectorized on 128-bit SVE, the peak compute throughput will be boosted by 2×. With an ideal vectorization in FP32, the peak compute throughput will be boosted by 4× on 128-bit SVE. We validate the model by using the modified SpMV benchmark with increased computation repetition so that the benchmark moves from the memory-bound region to the compute-bound region. When using 20 repeated computations, the transformed kernel not only achieves 2× reduction in retired instructions as its original version (in Figure 3a) but also achieves 1.8× performance speedup unlike its original version (in Figure 3b), which has no performance speedup from vectorization. In fact, this achieved speedup is close to the 2× boost in peak throughput in the roofline model. Furthermore, we change the data element size of the benchmark from FP64 to FP32. As predicted by the roofline model, when the ELEN is 32-bit, with 128-bit SVE, the modified benchmark achieves 4× reduction in retired instructions and 3.8× boost in peak throughput. In Figure 7, we annotate each application based on their estimated arithmetic intensity measured from single-threaded non-vectorized execution. The SpMV and STREAM benchmarks are in the memory-bound region (the left of the first dashed line). As their performance bottleneck is the peak memory throughput on the platform, vectorization cannot improve their performance. This is consistent with the observation that little performance speedup is reported in Figure 3b even though their retired instructions are reduced by vectorization in Figure 3a. Theoretically, the vectorization can improve the peak performance corresponding to the architectural vector length. However, it may also move some workloads from compute-bound into memory-bound as illustrated in the two red triangles atop the green regions in Figure 7, like both quantum simulation and AutoDock. YOLOv3 and AlexNet are FP32 workloads and compute-bound. Thus, they both achieved performance improvement from SVE vectorization as reported in Figure 3b, and the vectorized version may also enter the red region. These observations indicate that pairing long vector engines with high-bandwidth memory may be important to avoid vectorized codes being bottlenecked by memory bandwidth. Such design is already used in Supercomputer Fugaku that uses HBM2 with 512-bit SVE [14].

Decision Tree. Figure 8 presents the main required input and stages for determining an application as one of the four classes, i.e., not vectorized, memory latency bound, memory bandwidth bound, and speedup. In this decision tree, we first determine whether a target kernel can be vectorized effectively using the introduced $R_{ins_reduction}$ metric. This metric filters out those applications that either cannot be auto-vectorized by the compiler or their vectorized instructions only compose a small fraction of total executed instructions. The former scenario may be caused by the complex control logic inside the loops and the recursive calling of functions or libraries within the algorithm. The latter, as seen in Section 4.1, could be caused by an increased number of non-vectorized instructions when other parts of the code such as threading runtime increase. The next step checks whether the kernel belongs to the memory- or compute-bound domains as shown in the roofline. For this, an estimated arithmetic intensity is used by approximation of $\frac{FP_op}{LLC_read_miss}$. The LLC_read_miss is selected instead of the

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, “ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace,” in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.



SN	Application	1-thread Case	72-thread Case
1	YOLOv3	Class 4	Class 4
2	LLM training	Class 4	Class 4
3	LLM inference	Class 4	Class 4
4	QC simulator	Class 4	Class 2
5	FFT1D	Class 1	Class 1
6	FFT2D	Class 1	Class 1
7	STREAM	Class 2	Class 2
8	DGEMM	Class 4	Class 4
9	SGEMM	Class 4	Class 4
10	SPMV	Class 3	Class 3
11	Jacobi2D	Class 2	Class 1
12	AlexNet	Class 4	Class 4
13	AutoDock	Class 4	Class 4

Fig. 8 & Table 3: Classification table(right) of the benchmark suite using the proposed decision tree(left).

total memory accesses because memory accesses also included prefetching traffic while LLC_read_miss is more realistic to reflect the main part limiting the response speed from memory. The estimated arithmetic intensity is then compared against the inflection point $AI_{inflection}$ on the roofline model in Figure 7. The memory-bound class is further divided into bandwidth or latency bound, by comparing the obtained last level cache miss ratio (R_{llc}) with an ideal miss ratio ($R_{llc} = \frac{ELEN}{cache_line}$), assuming an application with large input problem filling the LLC. On Grace processor, the LLC line size is 64-byte, with 8-byte double float-point, 13% is used as the threshold value.

Using the decision tree, we managed to classify 26 cases from 13 applications in the benchmark suite. As summarized in Table 3, 15 out of 26 tested workloads can be sped up by vectorization without any porting efforts. Five cases, including FFT1D and 2D, and Jacobi2D in 72 threads, cannot be vectorized effectively and thus no performance gain is obtained from SVE. Finally, four cases, including the quantum circuit simulation at 72 threads, STREAM, and Jacobi2d at single thread, can be vectorized by compilation, but they cannot achieve performance speedup on the platform due to memory bound.

6 Conclusion

This work assesses the maturity of exploiting ARM SVE in HPC applications. We provided a suite of 13 applications from machine learning, scientific, and quantum computing. We defined quantitative metrics, validated hardware counters, and proposed a roofline model for identifying performance bottlenecks on ARM SVE on the Nvidia Grace processor. With a decision tree, we managed to classify 26 cases and found that 15 achieved speedup via SVE vectorization. Our results indicate that double-precision HPC applications face more challenges in exploiting SVE than single-precision machine learning workloads because they need longer vector lengths and higher memory bandwidth.

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

Acknowledgment

This research is supported by the European Commission under the Horizon project OpenCUBE (101092984). This work was supported by the Lawrence Livermore National Laboratory LDRD Program under 25-ERD-016. LLNL-CONF-2005986.

References

1. ARM: Arm Neoverse V2 Core Technical Reference Manual, revision: r0p2 edn. (2021)
2. ARM: Arm® Architecture Reference Manual, arm ddi 0487 edn. (2024)
3. Asanović, K., Patterson, D.A.: Instruction sets should be free: The case for risc-v. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146 (2014)
4. Flynn, P., Yi, X., Yan, Y.: Exploring source-to-source compiler transformation of openmp simd constructs for intel avx and arm sve vector architectures. In: Proceedings of the Thirteenth International Workshop on Programming Models and Applications for Multicores and Manycores. pp. 11–20 (2022)
5. Foundation, L.: perf: Linux profiling with performance counters. https://perfwiki.github.io/main/ (2024)
6. Gupta, S.R., Papadopoulou, N., Pericas, M.: Accelerating cnn inference on long vector architectures via co-design. In: 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 145–155. IEEE (2023)
7. Langarita, R., Armejach, A., Ibáñez, P., Alastruey-Benedé, J., Moretó, M.: Porting and optimizing bwa-mem2 using the fujitsu a64fx processor. IEEE/ACM Transactions on Computational Biology and Bioinformatics **20**(5), 3139–3153 (2023)
8. Lee, J.K., Jamieson, M., Brown, N., Jesus, R.: Test-driving risc-v vector hardware for hpc. In: International Conference on High Performance Computing. pp. 419–432. Springer (2023)
9. Miksits, S., Shi, R., Gokhale, M., Wahlgren, J., Schieffer, G., Peng, I.: Multi-level memory-centric profiling on arm processors with arm spe. In: SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 996–1005. IEEE (2024)
10. Noor, M.A., Kent, K., Konno, K., Maier, D.: SIMD support to improve eclipse openj9 performance on the aarch64 platform. In: Proceedings of the 19th ACM International Conference on Computing Frontiers. pp. 49–57 (2022)
11. Nvidia: Nvidia grace cpu superchip. https://www.nvidia.com/en-us/data-center/grace-cpu-superchip/?ncid=no-ncid (2024)
12. Poenaru, A., McIntosh-Smith, S.: Evaluating the effectiveness of a vector-length-agnostic instruction set. In: Euro-Par 2020: Parallel Processing: 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24–28, 2020, Proceedings 26. pp. 98–114. Springer (2020)
13. Rossi, F., Cococcioni, M., Saponara, S.: Llama-2 acceleration using the arm scalable vector extension. International Conference on Applications in Electronics Pervading Industry, Environment and Society (2024)
14. Sato, M., Kodama, Y., Tsuji, M., Odajima, T.: Co-design and system for the supercomputer Fugaku. IEEE Micro **42**(2), 26–34 (2021)

This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in Euro-Par 2025: Parallel Processing, and is available online at https://doi.org/10.1007/978-3-031-99857-7_3.

R. Shi, G. Schieffer, M. B. Gokhale, P.-H. Lin, H. Patel, and I. B. Peng, "ARM SVE Unleashed: Performance and Insights Across HPC Applications on Nvidia Grace," in Euro-Par 2025: Parallel Processing, Aug. 2025, pp. 33–47. doi: 10.1007/978-3-031-99857-7_3.

15. Sato, M., Tsuji, M.: OpenACC execution models for manycore processor with ARM SVE. In: Proceedings of the HPC Asia 2023 Workshops. pp. 73–77 (2023)
16. Schieffer, G., Wahlgren, J., Ren, J., Faj, J., Peng, I.: Harnessing integrated cpu-gpu system memory for hpc: a first look into grace hopper. In: Proceedings of the 53rd International Conference on Parallel Processing. pp. 199–209 (2024)
17. Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., et al.: The arm scalable vector extension. IEEE micro **37**(2), 26–39 (2017)
18. Takahashi, K., Mori, T., Takizawa, H.: Prototype of a batched quantum circuit simulator for the vector engine. In: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. pp. 1499–1505 (2023)
19. Wei, Y., Deng, L., Sun, S., Li, S., Shen, L.: Dgemm optimization oriented to arm sve instruction set architecture. In: 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). pp. 514–521. IEEE (2023)
20. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM **52**(4), 65–76 (2009)
21. Wu, D., Meng, J., Zhu, W., Deng, M., Wang, X., Luo, T., Wahib, M., Wei, Y.: autogemm: Pushing the limits of irregular matrix multiplication on arm architectures. In: SC24: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–15. IEEE (2024)
22. Yamada, F., Kawakami, K., Kurihara, K., Matsuda, K., Tabaru, T.: Optimization of numpy transcendental functions for arm sve. In: Proceedings of the HPC Asia 2023 Workshops. pp. 50–54 (2023)
23. Yokoyama, D., Schulze, B., Borges, F., Mc Evoy, G.: The survey on ARM processors for HPC. The Journal of Supercomputing **75**, 7003–7036 (2019)