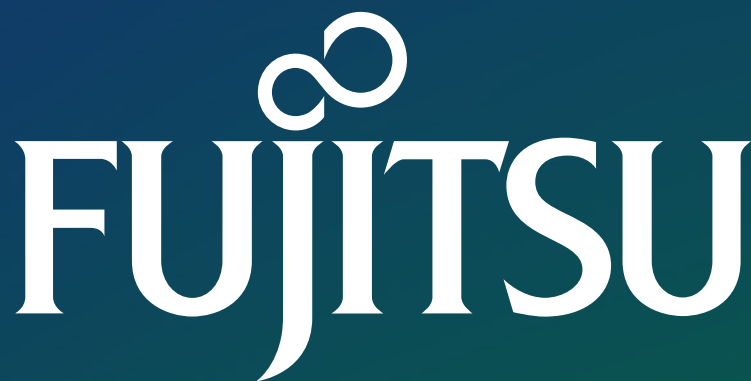


Improving PyTorch Efficiency on ARM with SVE-Accelerated Memory Primitives

N Maajid Khan, Ashish Chopra
Fujitsu Research of India Pvt. Ltd.



Abstract

The increasing deployment of PyTorch workloads on Arm-based cloud infrastructure has amplified the importance of efficient memory operations. **Memory initialization** and **data movement** are fundamental to tensor allocation and lifecycle management, directly impacting runtime efficiency. We present SVE-accelerated implementations of **memcpy** and **memzero** in **mimalloc**, the allocator adopted by PyTorch for AArch64 builds, enabling more efficient memory operations on SVE-enabled Arm processors.

This work introduces **Vector Length Agnostic (VLA)** memory primitives using Arm **Scalable Vector Extensions (SVE)**, runtime hardware-aware dispatch based on available vector length and compiler-safe integration that preserves baseline **AArch64** compatibility while enabling accelerated execution on supported systems. The implementation uses **SVE ACLE primitives** for **vectorized loads, stores, and predicated tail handling**, with size and hardware-aware fallback to the generic implementation.

Benchmarking on an **AWS Graviton3** system demonstrates up to **31.3%** higher **memory throughput** and **18.3%** fewer **retired instructions for memcpy**, with the largest benefits observed at selected buffer sizes and thread counts. These optimizations improve a foundational component of PyTorch execution and contribute to more efficient AI workloads on Arm infrastructure.

Keywords: memcpy, memzero, mimalloc, Vector Length Agnostic (VLA) , AArch64

Methodology

Our investigation was motivated by **PyTorch PR #164741 [1]**, which enabled **mimalloc** as the **default CPU allocator for AArch64**. Building on this existing integration, we investigated whether Arm SVE could further optimize the memory operations used within mimalloc. In current PyTorch AArch64 builds, mimalloc is statically linked into **libc10.so**, establishing the execution path **PyTorch CPU inference → C10::DefaultCPUAllocator → c10::alloc_cpu() → mimalloc → memory operations**. Our work, **PR [2]**, therefore focuses on optimizing mimalloc's memory-operation primitives without modifying PyTorch's allocator, tensor, or model code.

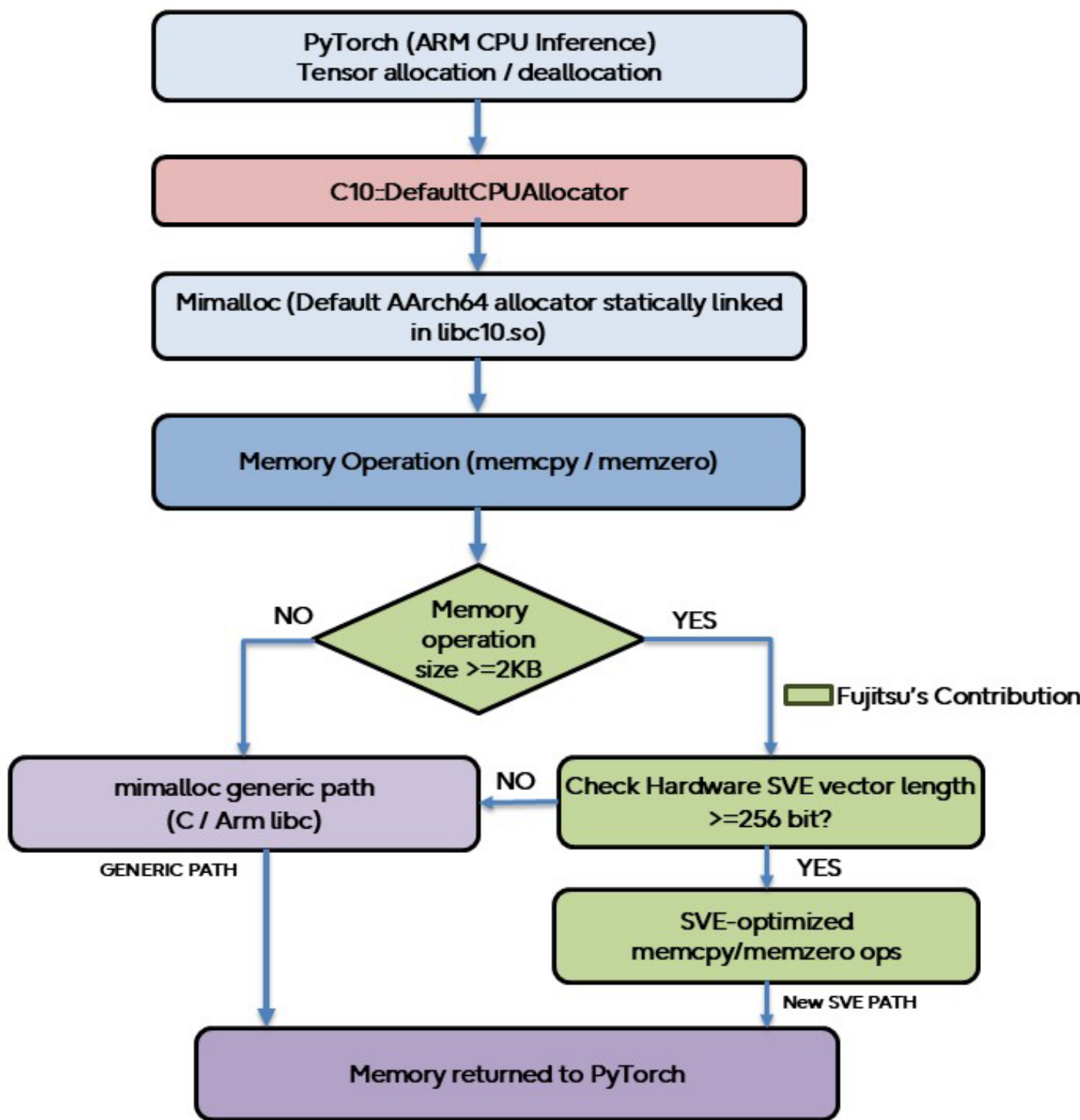


Figure 1: SVE-optimized memory-operation flow in mimalloc for PyTorch AArch64 CPU inference.

As shown in **Figure 1**, we extend **mimalloc** with **SVE-optimized memcpy** and **memzero** while retaining the generic fallback. Using **Arm SVE ACLE (<arm_sve.h>)**, the implementation provides vector-length-agnostic loads, stores and predicated operations. Runtime dispatch checks SVE support and **VL ≥ 256 bits**, with a **2 KB threshold** to avoid overhead for small transfers. The kernels use 4x unrolled vector loops with predicated tail handling.

The optimization is transparent to PyTorch, requiring no application or model changes. As these primitives are used in PyTorch's CPU **memory-management** path, **tensor allocation** and **data-movement** operations can directly benefit from the optimized kernels. We evaluate **OSS** and **SVE-enabled mimalloc** across memory transfer sizes and 1/32 threads, measuring **memcpy throughput (GB/s)** and to quantify performance and instruction-efficiency gains.

Results

We evaluate the **OSS** and **SVE-enabled mimalloc** implementations on an **AWS Graviton3 (Arm Neoverse-V1, 32 vCPU, SVE256)** system using **GCC 13** and **PyTorch v2.14.0** for **AArch64 CPU**. Experiments cover **1 and 32 threads** and **buffer sizes** from **2 KB to 64 KB**, under identical conditions for both implementations. We measure **memory throughput** and **retired instruction count** to quantify both **memory-performance** and **instruction-efficiency** improvements.

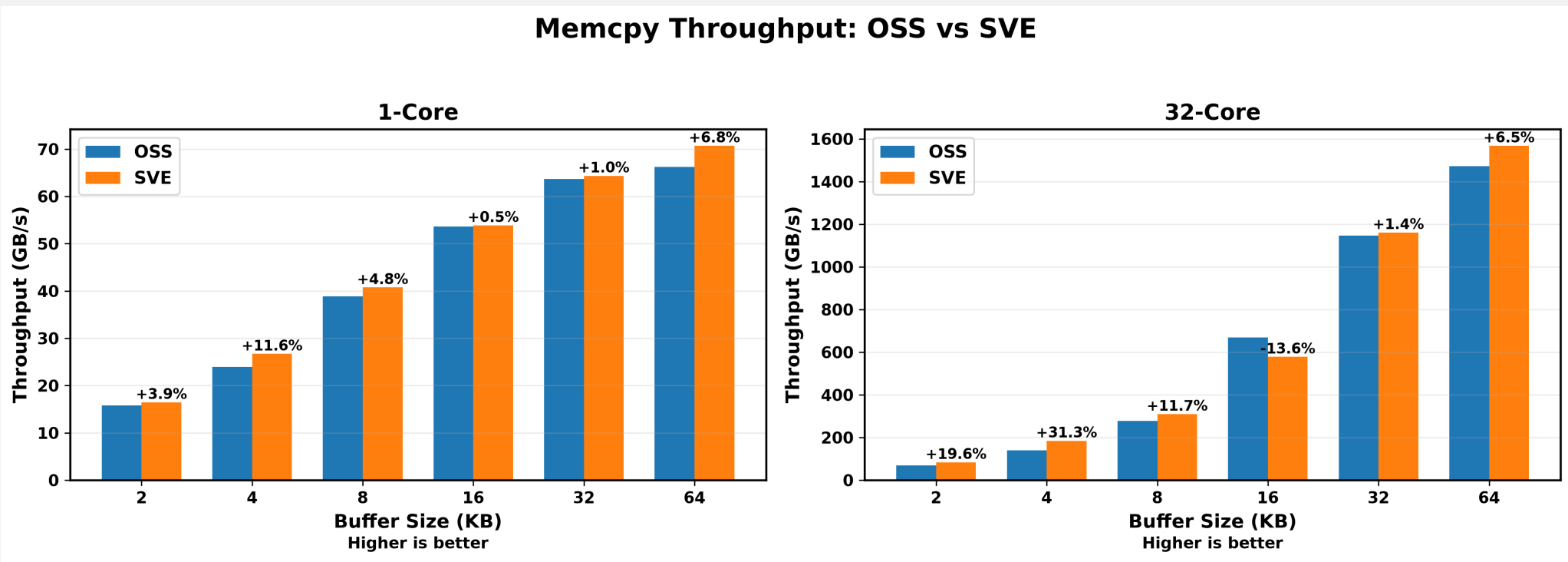


Figure 2: memcpy throughput comparison between OSS and SVE-enabled mimalloc.

Figure 2 demonstrates that SVE improves **memcpy throughput** across several evaluated buffer sizes. At **1 thread**, throughput improves by up to **11.6%**, while at **32 threads** the gains reach up to **31.3%**. The improvement varies with buffer size, reflecting the interaction between transfer size and parallelism.

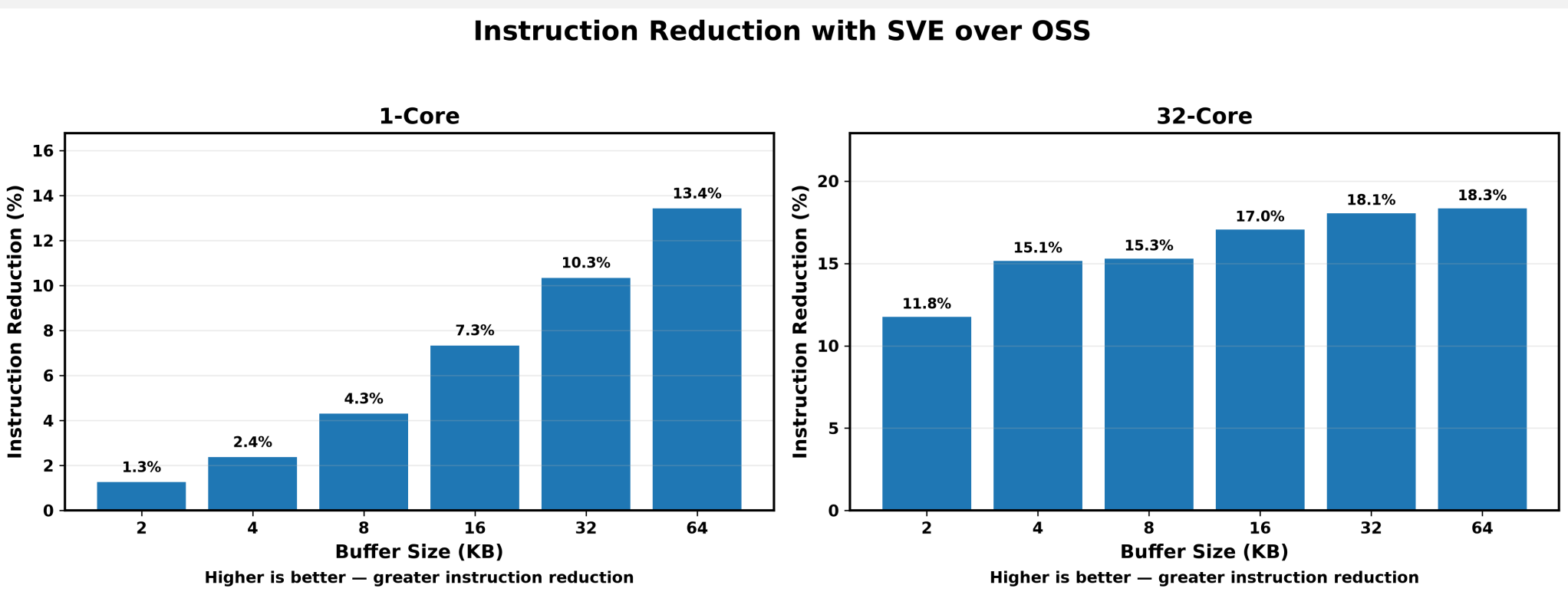


Figure 3: Instruction reduction with SVE over OSS mimalloc.

Figure 3 demonstrates a consistent reduction in retired instructions across all evaluated buffer sizes. Instruction count decreases by up to **13.4%** at **1 thread** and **18.3%** at **32 threads**. The reduction is more pronounced for larger transfers and higher thread counts, demonstrating the instruction-efficiency advantage of the SVE implementation.

Conclusion and Future Work

Our work demonstrates that SVE-optimized memory primitives can improve the efficiency of **mimalloc** on Arm CPUs while preserving compatibility through size and hardware-aware dispatch. As mimalloc is part of PyTorch's AArch64 CPU memory path, these optimizations provide a transparent opportunity to improve memory-intensive PyTorch operations.

Future work will focus on end-to-end PyTorch workload evaluation, further **kernel and dispatch optimizations**, and **power/energy** measurements to assess the potential for improved energy efficiency in Arm-based AI workloads.

References

- [1] Enable mimalloc on non-Windows platforms and make default for AArch64 builds. <https://github.com/pytorch/pytorch/pull/164741>
- [2] aarch64: add SVE-accelerated memcpy and memzero. [Fujitsu's OSS PR] <https://github.com/microsoft/mimalloc/pull/1300/>
- [3] Extending the Pytorch vec backend for SVE (ARM). [Fujitsu's OSS PR] <https://github.com/pytorch/pytorch/pull/119571>



Enable mimalloc for
AArch64 PR



mimalloc SVE PR



SVE256 Backend
for ARM CPU's