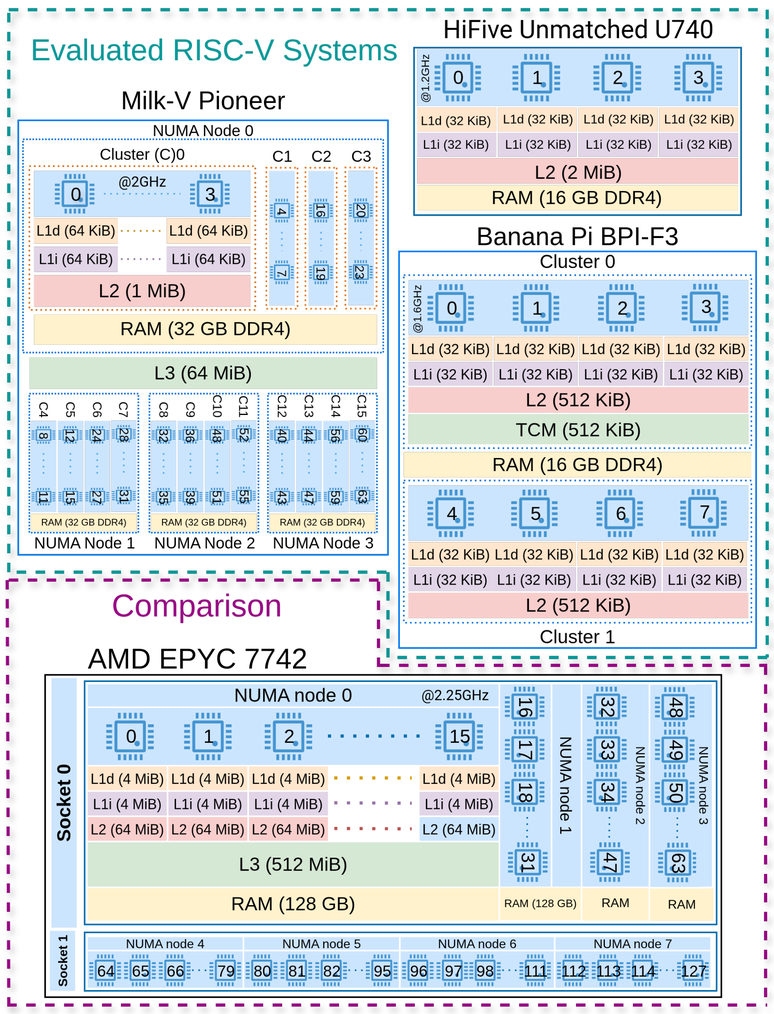
Evaluating Memory and Multi-Core Performance of RISC-V Systems on HPC-Driven Workloads

Thomas Pasquali ✉, Brahmaiah Gandham, Flavio Vella, Salvatore Andaloro

University of Trento, Italy. Correspondence to: thomas.pasquali@unitn.it

Abstract

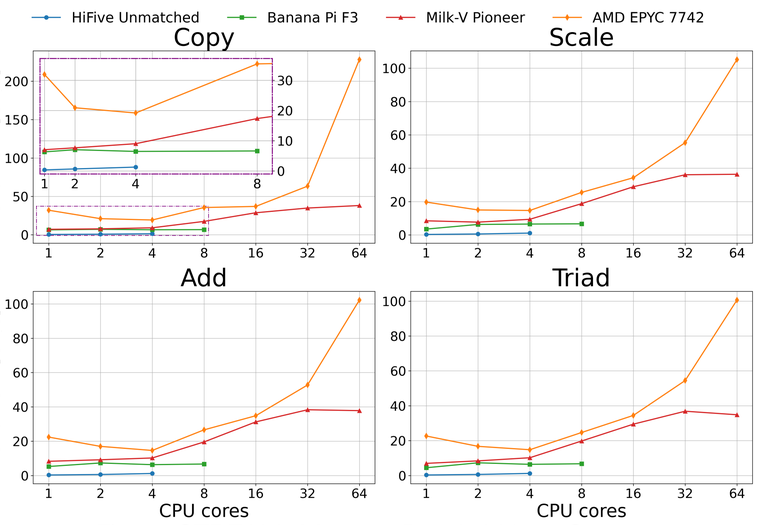
This study presents a comprehensive benchmarking analysis of RISC-V-based systems, focusing on **memory performance**, **thread synchronization efficiency**, and **scalability**. We evaluate three representative **RISC-V-based systems**, covering a spectrum from low-power to more advanced multicore designs: HiFive Unmatched U740, Banana Pi BPI-F3, and Milk-V Pioneer. For **comparison**, we include the **AMD** EPYC 7742, a server-grade x86-64 processor widely used in HPC environments. We evaluate these systems using a suite of synthetic benchmarks (**STREAM**, **Pointer Chasing**, and **Thread Synchronization Primitives**) and a **parallel Breadth-First Search** (BFS) algorithm.



We compile all benchmarks using `CLANG v22` (`OpenMP 5.1`) from BSC EPI-LLVM for RISC-V CPUs. For AMD, we use `GCC 13.3.0` (`OpenMP 4.5`).

STREAM Benchmark

STREAM is a widely used benchmark that measures **sustainable memory bandwidth** through vector operations. It represents the performance of **memory-bound applications** and provides an architecture-neutral way to **compare memory and cache efficiency** across different hardware.



All values are of type `double` (64-bit, 8 bytes). For our experiments, we set $n = 20M$, resulting in a total memory usage of 457MiB (152MiB per array). Each kernel is executed 10 times; the first run serves as a warm-up and is excluded from the final arithmetic mean.

Fig. 1: STREAM. *Memory Bandwidth* vs *Cores*.

Fig. 1 illustrates that the evaluated systems exhibit substantial differences in peak memory bandwidth. On **HiFive** and **Banana** platforms, **bandwidth does not scale** with the number of active cores. The **Pioneer's memory hierarchy fails to sustain adequate data throughput** when more than 32 cores are utilized. Across all systems, introducing **arithmetic** operations **has minimal impact on bandwidth**, except for AMD using 64 cores.

Pointer Chasing Benchmark

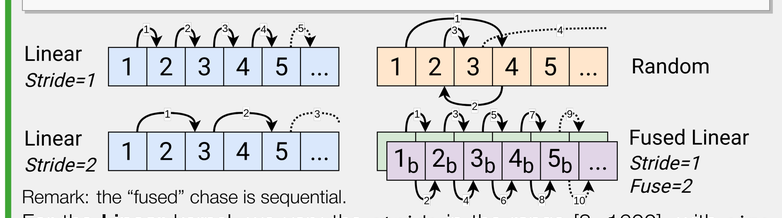
Pointer chasing evaluates **memory latency** by traversing a linked list with **various** non-contiguous **access patterns**, stressing latency rather than bandwidth. It reveals the **efficiency of the memory subsystem** and cache hierarchy behavior under latency-sensitive workloads.
Note: All experiments are conducted using a single CPU core.

This benchmark includes three kernels:

1. **Linear** accesses memory with increasing `stride`.
2. **Random** uses a uniformly shuffled list.
3. **Fused** traverses multiple lists concurrently (not in parallel).

Pointer Chasing Benchmark - General Pointer Chase

```
1 void** p = (void**) list;
2 while (size-- > 0) p = (void**) *p;
```



Remark: the "fused" chase is sequential.

For the **Linear** kernel, we vary the `stride` in the range [8, 1600], with `size` adjusted accordingly: $\min(2^{26}, 2^{10} \cdot \texttt{stride} \cdot \texttt{sizeof(void*)})$. In the **Random** kernel, we vary the list `size` from 1KiB to 128MiB. For the **Fused-Linear** kernel, we explore `stride` values in the range [8, 80] and `fuse` in [1,8].

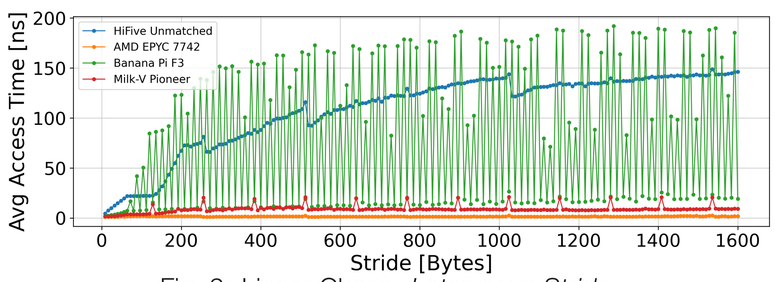


Fig. 2: Linear Chase. *Latency* vs *Stride*.

Fig. 2 shows that as the stride increases, systems with **less effective cache** performance exhibit **increasing memory access latencies**. **AMD** and **Pioneer** maintain **low and consistent latencies** across all stride values, indicating efficient memory and cache subsystems. In contrast, **HiFive** and **Banana** experience **significant latency degradation**.

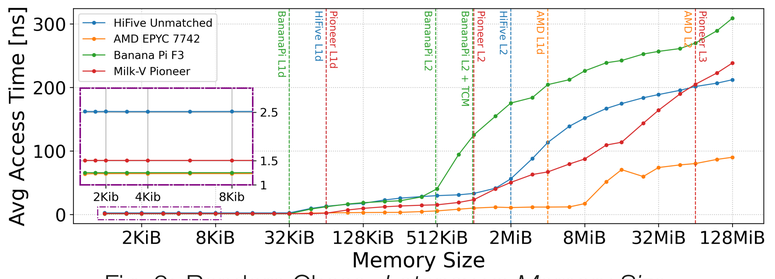


Fig. 3: Random Chase. *Latency* vs *Memory Size*.

Fig. 3 presents memory latency under unpredictable access patterns that **bypass hardware prefetching**. As `size` exceeds the capacity of cache levels (vertical lines), latencies increase. **AMD** demonstrates the **best performance**, attributed to its large and efficient cache hierarchy. **Pioneer** also **performs well**, exhibiting lower latency compared to **HiFive** and **Banana**, both of which show **sharp latency increases** due to limited cache capacity and slower memory subsystems. However, when `size` exceeds Pioneer's L3 cache, its performance degrades and becomes worse than that of HiFive.

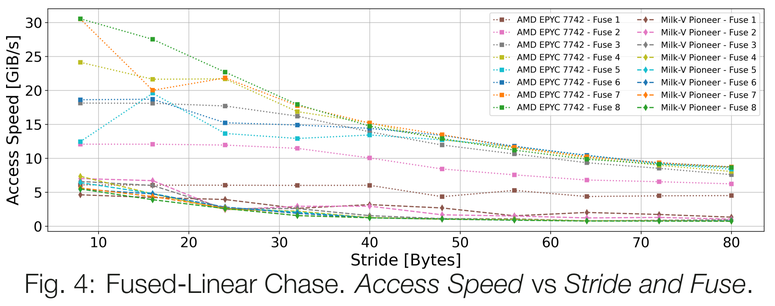
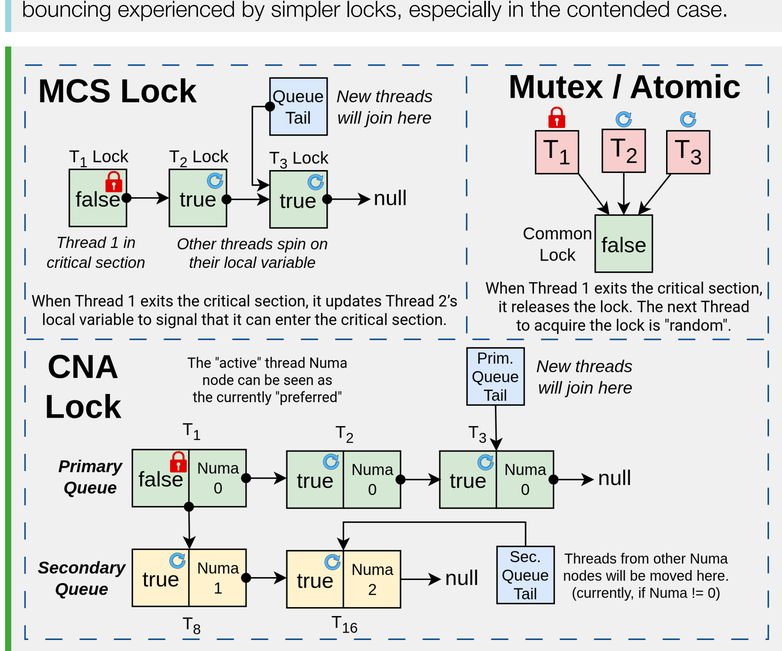


Fig. 4: Fused-Linear Chase. *Access Speed* vs *Stride and Fuse*.

As shown in Fig. 4, **AMD consistently outperforms Pioneer**. AMD **maintains high throughput** at **small strides** - exceeding 30 GiB/s at high fusion levels (7-8) - and **degrades as** `stride` **increases**. In contrast, **Pioneer's throughput is significantly lower** (<10 GiB/s) and drops with increasing stride. At larger strides, performance of both platforms converges across all fusion levels.

Thread Synchronization Benchmark

This benchmark evaluates the performance of **thread synchronization primitives**. We consider *Mutexes*, *Atomics*, *Queue Locks*, and *NUMA-aware Locks* under **varying levels of contention**. *CNA Lock* is designed to optimize performance on NUMA architectures. *MCS Lock* eliminates much of the cache-line bouncing experienced by simpler locks, especially in the contended case.



In these tests, each thread **(1) acquires a lock**, **(2) updates a shared variable**, and **(3) releases the lock** so that the next thread can proceed. We let each experiment run for 20 seconds and average the results of 5 runs.

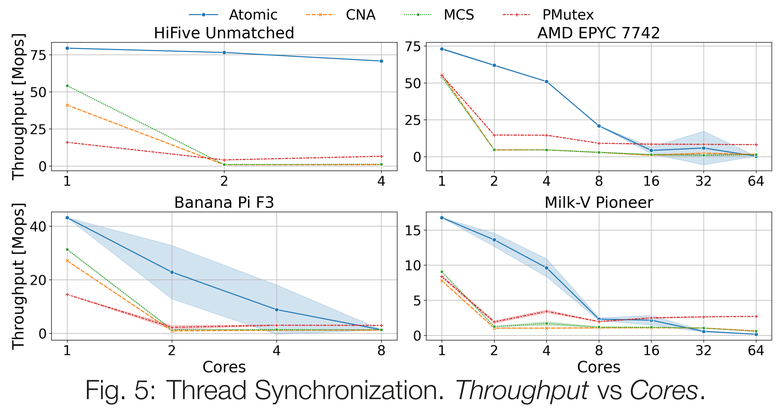


Fig. 5: Thread Synchronization. *Throughput* vs *Cores*.

Fig. 5 shows that **atomic operations** generally achieve the **highest throughput** both on RISC-V and AMD, but **drop as core counts increase**, due to contention on shared memory, although only HiFive achieves a similar throughput to AMD. **Lock-based methods** (*CNA, MCS, PMutex*) show **much lower throughput**, likely due to the software overhead involved with managing locks. **Pioneer's performance is significantly worse than any other system.**

Breadth-First Search (BFS) Benchmark

We use a **custom cache-optimized parallel BFS** [1] as a realistic benchmark to evaluate scalability. BFS is a fundamental graph algorithm widely used in search, analytics, and scientific computing. It stresses **irregular memory access**, **dynamic work distribution**, and **synchronization overheads**. Its inherently irregular and data-dependent behavior makes it particularly suitable for benchmarking parallel architectures.

[1] Andaloro, Pasquali, and Vella "Cache-optimized BFS on multi-core CPUs", 2025

We evaluate three BFS variants: **OpenMP with static scheduling** (which outperforms dynamic and guided in our setup), **Pthreads with Mutexes** (explicit locking), and **Pthreads with Atomics** (fine-grained, lock-free coordination). In the Pthreads versions, the `CHUNK_SIZE` parameter sets the minimum vertices per thread, which affects the amount of contention during work-stealing. The implementation targets **large-diameter graphs** and is tested on **real-world datasets**. Each experiment is run at least 15 times; we report the geometric mean of runtimes, excluding the first 2 runs. Each run starts from a different source vertex in a similar-diameter connected component. Results shown are for the **GAP-road graph** (~24M vertices, ~60M edges); additional results are available for other graphs.

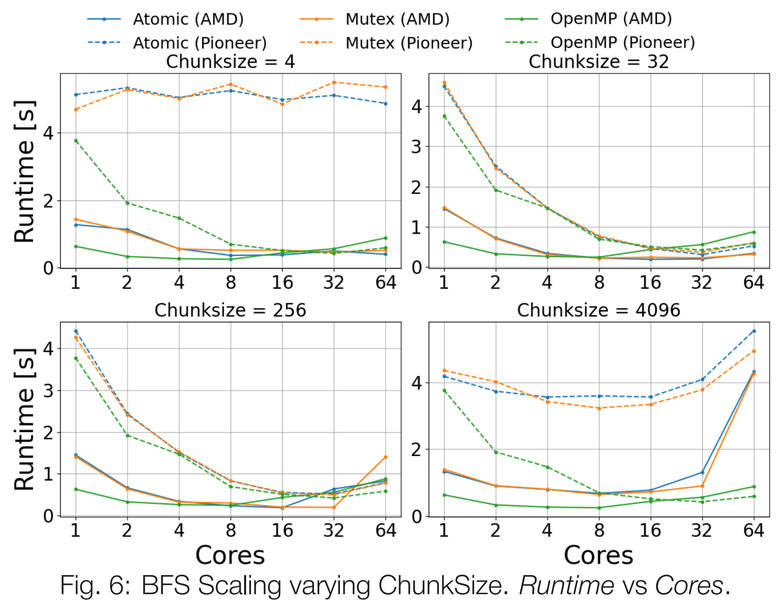


Fig. 6: BFS Scaling varying ChunkSize. *Runtime* vs *Cores*.

Fig. 6 shows that **AMD consistently outperforms Pioneer up to 16 cores, but the gap narrows beyond 32 cores.**
At **small chunk** sizes (4), **only Pioneer** suffers a marked **performance drop** in the Pthreads variants, indicating **higher contention sensitivity**. Conversely, at **large chunk** sizes (4096), **both systems** experience **degraded performance** due to **load imbalance**, with AMD showing greater robustness.
OpenMP scales worse than the Pthreads variants across both systems. However, **despite differences in compilers and OpenMP runtime** implementations, the overall **performance trends remain consistent across the two platforms**.

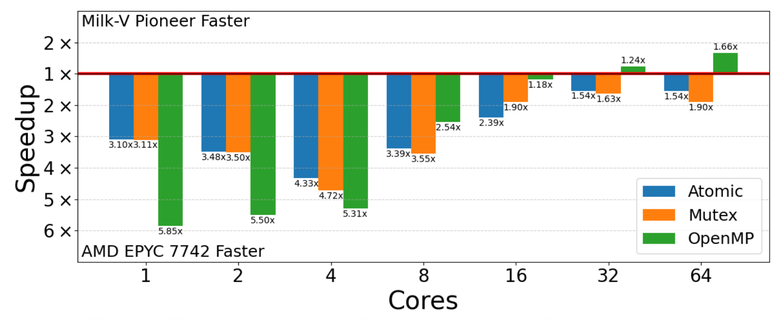


Fig. 7: Pioneer and AMD Comparison. Chunksize=32

Fig. 7 shows that at **lower core counts** (1–16), **AMD** achieves significantly **better performance**, particularly with OpenMP, reaching up to $5.85\times$ faster execution. However, as the number of **cores increases beyond 16**, the **performance gap narrows**, and **Pioneer** begins to **outperform AMD**, demonstrating **superior scalability but only using OpenMP**.
Results on additional graphs confirm the general trend, although **Pioneer does not consistently surpass AMD**.

Future Directions

1. Conduct a **deeper** results **analysis** to **identify underlying** performance **factors**;
2. Introduce **new benchmarks** focused **on machine learning**;
3. **Expand the evaluation** to additional RISC-V platforms;
4. Extend the benchmarking to **RISC-V-based distributed-memory systems**.

Acknowledgments

We gratefully acknowledge the Barcelona Supercomputing Center for providing access to the RISC-V systems. We also thank E4 for their valuable feedback on this work and for organizing this event.