FORMULUS BLACK™

ForsaOS 2.0 host level performance analysis

Rachel Wooten Stevenson and Andrei Khurshudov

Abstract

In this document, we summarize the input-output performance of a server using the new ForsaOS software operating system for IMC (In-Memory Computing). The results shown highlight that the software enables exceptionally high throughput for both read and write operations while simultaneously maintaining very low latencies. Performance is quoted for the host level only, rather than for virtual machines.

Introduction to In-Memory Computing

In-memory computing (IMC) represents a new paradigm for maximizing modern computer system performance in which system RAM is used for both system memory and data storage. While there have previously been some specialized applications which have boosted their operational speed by being run in memory while still utilizing traditional solid-state drives or hard disks as additional storage, ForsaOS software instead bypasses the need for slower storage entirely by keeping all of the system's data in memory at all times. The complete switch to In-Memory Computing grants an appreciable boost to speed for any data-intensive processes.

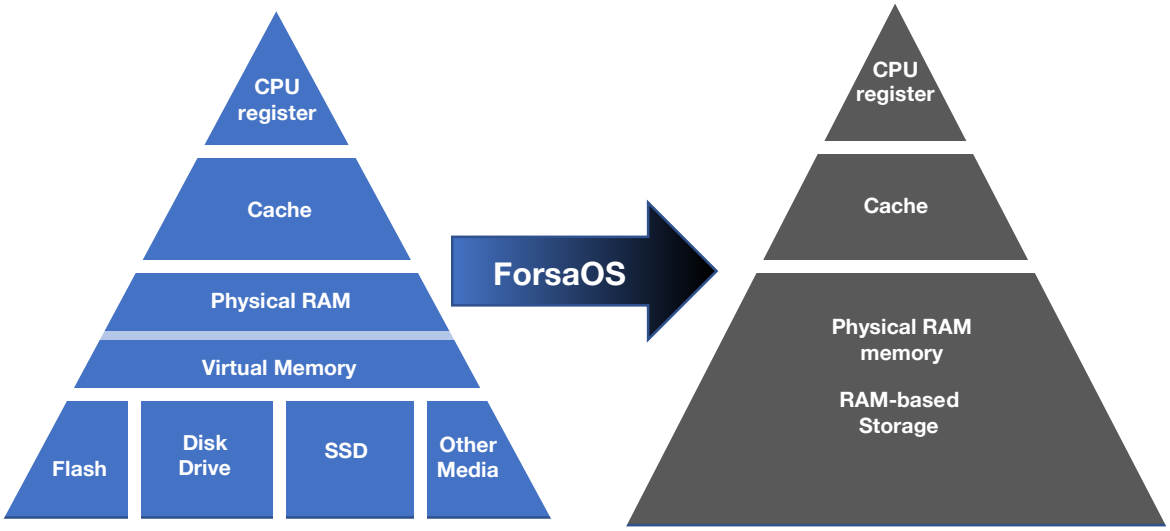

Figure 1. Diagram showing the memory hierarchy of a conventional computer (left) and a computer using ForsaOS (right). Formulus Black software enables the user to utilize RAM as a persistent storage medium, significantly improving performance over systems relying on solid state drives or hard disks.

FORMULUS BLACK

Test environment and tools

Tests listed here for ForsaOS were performed on the AIC-made server with 2 Intel Xeon Platinum 8160M CPUs (2.10 GHz) with 24 cores per socket at host level.

System configuration	
Processor	*2 Intel Xeon Platinum 8160M CPUs (2.10 GHz)*
Cores	24 cores per CPU socket
Graphics Card	*ASPEED Graphics Family, version 4*
Memory	*187GB RAM as system memory, declared from 12x2x64GB DDR4 Samsung 2400 MT/s DIMMs (1.5 TB total installed memory);*
OS	ForsaOS 2.0
Storage Drive (for host-level installation)	Samsung SSD 960 EVO 250GB
LEM size	500 GB

We used Flexible I/O Tester, or FIO, to measure system performance. FIO is a straightforward synthetic workload generator that characterizes fundamental disk read and write operations (sequential and random read and write, or mixed jobs) under a variety of settings (e.g. queue depth, block size, multiple disks, etc.) in order to mimic various types of workloads typically requested of the disk. In these tests, a ForsaOS 'LEM' (logical extension of memory) is used as the block device (disk) by the host.

Each individual read and write test point was taken over 30 minutes of sustained I/O operations, and for each trial, the LEM was completely overwritten multiple times[1]. This methodology was used to ensure accuracy of the latency measurements as well as to ensure that the LEMs were completely overwritten multiple times in each test. Prior to each new run with a different block size or a new LEM, the 500GB LEM was preconditioned with an FIO write operation of the same block size to fill the LEM.

Single-job results

[1] Example script:

fio --filename=drive_location --direct=1 –numa_cpu_nodes=0 --cpus_allowed_policy=split --nice=-19 --readwrite=read --refill_buffers --norandommap --randrepeat=0 --ioengine=libaio bs=4K --iodepth=1 --numjobs=1 --runtime=1800 --time_based --name=jobname --output=filename

In this set of tests, we examine performance with only single-job tasks (FIO parameter numjobs=1) as a function of I/O type for various block sizes. We show only the results for queue depth 1 for brevity, but also for a few more practical reasons.

For conventional solid-state drives (SSDs), increasing the queue depth will typically increase the throughput until the system reaches a maximum value, but at the unfortunate cost of significantly increasing the latency. However, we discovered that our system typically performed best for tasks with a queue-depth of one, while simultaneously maintaining the ultra-low latencies typical of short queues.

Rather than report only our peak performance numbers, we are publishing the bandwidth, total IOPS and latencies of the same trials in order to show that achieving ForsaOS peak output does not require sacrificing ultra-low latency. Throughputs are recorded in megabytes per second (MB/s); the latencies shown are the total average latencies (where total latency is the sum of submission latency and completion latency[2]) and are given in microseconds (µs). The results are shown below for two typical block sizes: small (4 KB) and medium-large (128 KB).

4 KB Block size results, Queue depth = 1, Numjobs = 1

I/O task	Bandwidth, MB/s	IOPS	Avg. Total Latency, µs
Sequential Read	2,073	506,000	1.8
Random Read	2,002	489,000	1.9
Sequential Write	1,177	294,126	2.7
Random Write	1,104	276,038	2.9

128 KB Block size results, Queue depth = 1, Numjobs = 1

I/O task	Bandwidth, MB/s	IOPS	Avg. Total Latency, µs
Sequential Read	7,184	54,800	18.1
Random Read	7,298	55,700	17.8
Sequential Write	2,059	16,089	45.1
Random Write	2,071	16,179	45.1

[2] In the table we report the average total latencies, which is the average of the total time from FIO initiating an I/O operation to the completion of the I/O operation. The average completion latency (defined as the average time that passes between submission to the kernel and when the IO completes) does not vary as strongly as the submission latency in this system as block size increases. For example, for single threaded sequential reads, the total average latencies are 1.8µs for 4K blocks, and 10 times slower for 128K blocks; in contrast, the average completion latencies for single threaded sequential reads are 314ns (nanoseconds) for 4K blocks and only 635ns for 128K, or only roughly 2 times slower. For ForsaOS, submission times dominate, especially for large blocks; this is not true for normal SSDs, where completion times that typically dominate for larger block sizes.

Unlike in many systems, our peak single-job 4 KB block IOPS and minimum latency were both achieved in the same trial (sequential reads, queue depth = 1).

It is worth noting that in ForsaOS, sequential and random read operations are substantially faster than write tasks. This is because ForsaOS's write performance is impacted by its **built-in data reduction algorithm, which allows the user to effectively expand (aka "amplify") physical RAM and thereby reduce system TCO (Total Cost of Ownership).** Considering the high cost of RAM, this makes it easier for the user to enter the realm of IMC. This data-reduction algorithm also provides a layer of intrinsic data-protection for the user that requires additional overhead to incorporate for many conventional SSDs.

Multi-threading results

For this set of tests, we increased the number of read/write operations being performed on a single LEM from 1 in the previous data to 32 (FIO parameter numjobs=32). As with the single-job tasks above, we report the full results for queue depth =1 for both 4 KB and 128 KB block operations[3].

4 KB Block size results, Queue depth = 1, Numjobs = 32

I/O task	Bandwidth, MB/s	IOPS	Avg. Latency, µs
Sequential Read	41,446	10,121,000	2.9
Random Read	36,507	8,922,000	3.3
Sequential Write	24,696	6,174,015	4.2
Random Write	22,870	5,717,675	4.7

128 KB Block size results, Queue depth = 1, Numjobs = 32

I/O task	Bandwidth, MB/s	IOPS	Avg. Latency, µs
Sequential Read	168,557	1,282,400	24.9
Random Read	78,276	593,500	56.5
Sequential Write	49,177	384,198	59.2
Random Write	44,345	346,450	69.3

[3] As for single jobs, in ForsaOS systems, the submission time dominates the total average latency, especially for larger blocks. For example, for 32 job sequential reads, the average completion latency is 574ns for 4K blocks and still only 634 ns for 128K blocks. In both cases, the submission latency completely dominates the total latency: on average, less than 20% of the total latency time is spent in the completion step for 4K blocks with 32 jobs, but less than 3% of the total latency is spent in the completion step for 128K blocks with 32 jobs.

ForsaOS's exceptional performance in multi-threaded jobs is particularly clear when comparing plots of throughput, IOPS, and latency versus the total number of parallel I/O jobs to the same LEM. As can be seen in the plots below, FIO read/write performance on this system is not saturated yet with 32 parallel jobs. Especially noteworthy is that increasing from 1 to 8 jobs increases the 4K IOPS by nearly 7-fold while leaving the average latency nearly completely unaffected. But even increasing the total number of jobs up to 32 simultaneous reads/writes does not double the latency.

4 KB Block size performance versus number of jobs:

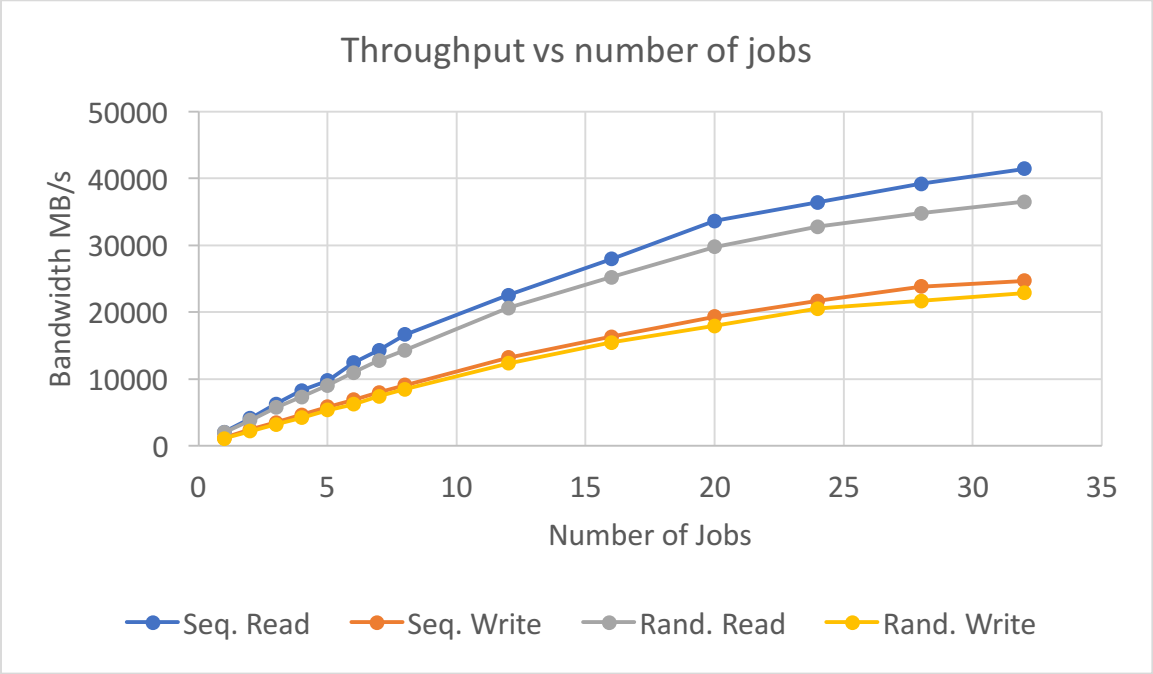

Figure 1. 4 KB Block size throughput in MB/s versus the number of parallel jobs for sequential and random read and write tasks performed on a single LEM. The highest measured value shown on this plot is for sequential reads at greater than 40 GB/s. Queue depth = 1 for all results shown. Higher bandwidth is better.

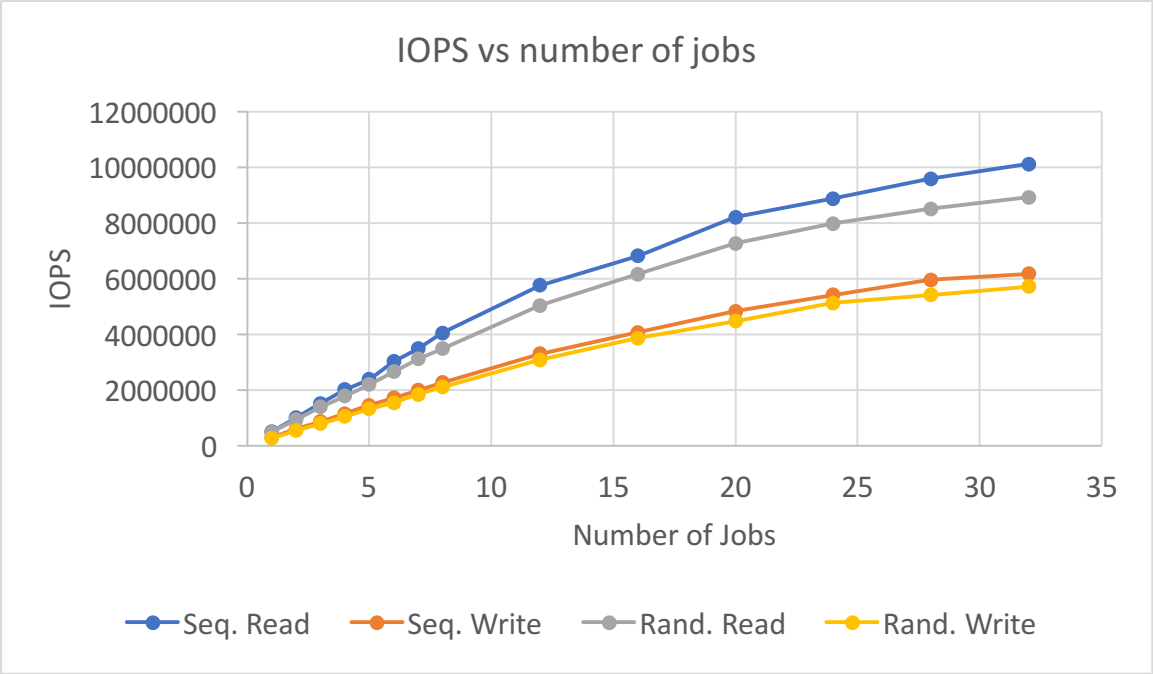

Figure 2. 4 KB Block size IOPS versus the number of parallel jobs for sequential and random read and write tasks performed on a single LEM. For sequential reads, the highest IOPS measured for 4K reads is greater than 10 million. Queue depth = 1 for all results shown. Higher IOPS is better.

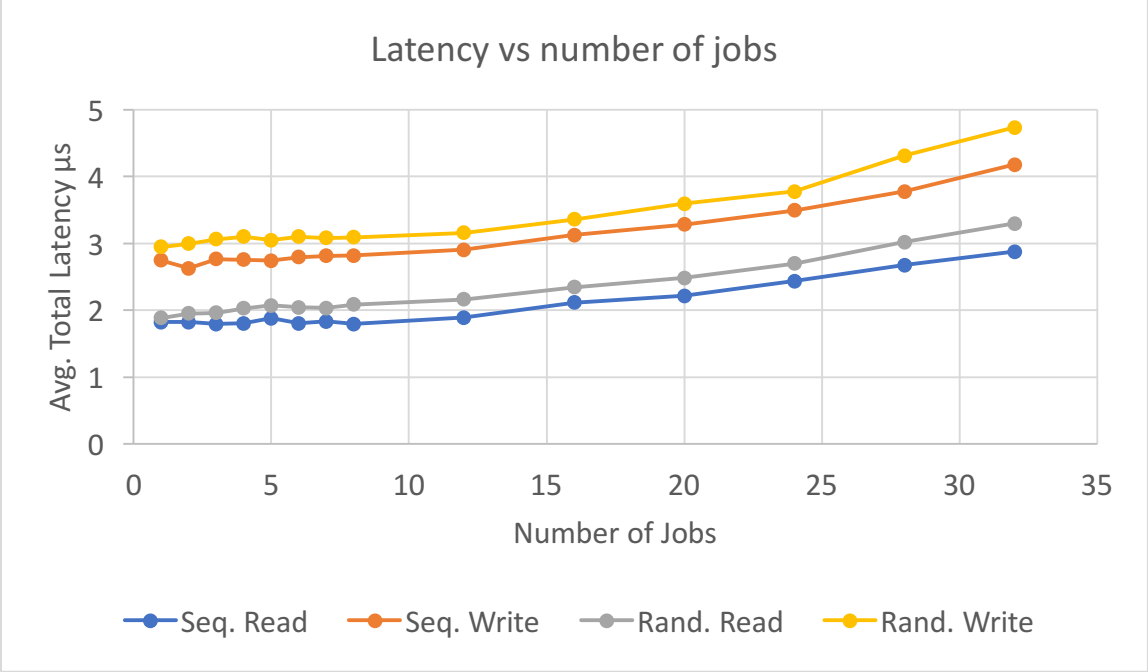

Figure 2. 4 KB Block size latency versus the number of parallel jobs for sequential and random read and write tasks performed on a single LEM. Note that the latency is essentially the same for 1-8 parallel jobs. Queue depth = 1 for all results shown. Lower latency is better.

For parallel jobs, ForsaOS is able to achieve millions of 4K-IOPS for all read and write tasks, and many tens of GBs per sec of throughput performance without sacrificing very low latencies. This is particularly evident in the case of sequential read operations for 128 KB blocks, where ForsaOS can achieve more than 160 GB/s of throughput in parallel read operations while still maintaining an average latency of under 25 microseconds.

In addition, like for the single-job trials, write operation performance in ForsaOS is not quite as exceptional as the read performance, but this is, again, due to the native, in-line data reduction/memory amplification discussed above.

Summary

ForsaOS showed exceptional peak performance in IOPS (in millions), Bandwidth (in tens of GB/sec), while simultaneously maintaining very low end-to-end latency (few-µs for 4KB/single or multi-thread jobs) during FIO tests for read operations, and only somewhat decreased (by design) performance during write operations caused by the data amplification step during LEM writes. **The most remarkable result,** however, is that these high throughputs and IOPS are achieved in this system without sacrificing latency for tasks requiring a shallow queue depth. In common enterprise practice, many applications issue I/O in a serialized manner – so the result is even more significant. In other words, many enterprise applications will significantly benefit from the ForsaOS approach.