Back to the Core-Memory Age: Running Operating Systems in NVRAM only

Abstract. Classical core memory was entirely non-volatile and could keep at least part of the operating system (OS) in main memory even across power cycles. These days we can have terabytes of NVRAM to repeat this approach, albeit on an entirely different scale and with large parts of the OS state still kept in the volatile CPU caches. In this paper, we discuss our experiences of running large modern operating systems including their applications entirely in NVRAM. We adapted stock Linux and FreeBSD kernels to work exclusively with NVRAM by hiding all DRAM from the kernels at boot time to establish a realistic performance baseline without changing anything else. Following this entirely NVRAM-agnostic approach, we could observe an effective performance penalty of a factor of about four, but only negligible increases in whole-system power draw. For our system with two CPU sockets and 56 cores total, we also observed a reduction in power draw in several scenarios. Due to prolonged execution times, the energy consumption increased as well for these measured workloads. While this might be discouraging at first sight, this result was achieved *without any* performance tuning as to the specific characteristics of today's NVRAM technology. Therefore, we are also discussing means to mitigate the observed shortcomings by integrating NVRAM appropriately into the memory hierarchy of future robust persistent systems.

Keywords: NVRAM · Operating Systems · Energy

1 Introduction

The current trend towards fast byte-addressable *non-volatile memory* (NVM)—NVRAM for short as synonymous—with latencies and a write resistance much closer to ordinary RAM[1] than to Flash, positions this *storage class memory* (SCM [3]) as a possible replacement for the established volatile technologies. This opens up a fundamentally new approach to the resilient operation of computing systems, in which all programs, including the operating system, are regularly executed directly in NVRAM.

Completely dispensing with DRAM promises a positive effect on energy consumption, which is particularly important for the service life of mobile devices, but stationary setups would also benefit in terms of electricity costs or cooling. Such an "NVM-only" solution, however, still has to cover the volatile state stored

[1] In the following, RAM stands for any form of volatile main memory, such as DRAM or SRAM, whenever the precise specification is immaterial.

in CPU registers and caches and keep it consistent with the counterparts held in the NVRAM.

Given such a top of the "memory pyramid" with both volatile and non-volatile features, the primary advantages of NVRAM—direct byte-addressability and persistence—are not, in fact, transparently made available to the vast majority of applications and imply major challenges, especially for the programming of such systems [10] as "using NVM for persistence requires fail-safe guarantees from application or device failures" [7]. For example, power failures in combination with NVRAM cause control flows that can unexpectedly transform a sequential process into a non-sequential process: A program has to deal with its own state from previous interrupted runs [9].

With quite simple precautions, however, these problems can be solved efficiently and functionally transparently even without special hardware support, both for the machine programs (i.e., applications) and for a large part of the operating system programs at the operating system level itself:

1. by a sporadically triggered checkpointing mechanism integrated into the exception-handling subsystem [2] and
2. by integration of this new storage class into the memory hierarchy via the virtual memory subsystem, whereby the persistence of NVRAM pages buffered in RAM is ensured by the former (point 1., analogous to [5]).

Both concepts combined pave the way for direct execution from NVRAM for the operating system and the machine programs running on it, but especially for legacy software.

But the advantage of such a purely *software-based whole-system persistence* with direct execution from NVRAM is of little value if the resulting performance is too poor for the common use case, making this mode of operation unattractive. It is therefore essential to first compare the cost of an NVRAM-based (general purpose) operating system with that of its traditional RAM-based variant before moving on to implementing NVRAM-specific concepts for an energy-efficient transparent execution of legacy software, in order to be able to better classify the hoped-for added value—in a real environment, not on a simulation or emulation basis.

This is exactly what this paper does: it presents a case study where Linux and FreeBSD, including machine programs, are executed from both NVRAM and DRAM, and compares the obtained performance characteristics. DRAM-based operation is nothing special, but it determines the baseline for comparison. In contrast, the NVRAM-based operation of "vanilla" Linux or FreeBSD is—to the best of our knowledge—an unprecedented act, which, as we will describe, requires some bold engineering work. The migration of Linux and FreeBSD from DRAM to NVRAM will not only show whether the higher access latencies of the latter are drowned out in the background noise of the overall system but also help to gain insights and guidelines for operating system development in general. Please note again, that none of the aforementioned challenges of a persistent operating system are addressed at this early stage and that our modified OS kernels are still *functionally* volatile, only running in non-volatile memory. We go further

than [13] and supplement their work with the evaluation of systems that run entirely in NVRAM. In this sense, the paper makes the following contributions:

- A field report on the adventure of freeing Linux and FreeBSD from DRAM dependencies and preparing both operating systems for the exclusive use of NVRAM.
- A before-and-after comparison of the performance and energy efficiency of the systems in different configurations using either NVRAM or DRAM, with various benchmarks and practical, real-world workloads.
- *Recommendations for action* for the transition towards NVRAM-based operating systems that paves the functionally transparent use of NVRAM for legacy software.

The rest of the paper is organised as follows: Section 2 introduces the hardware used for the experiments, explains the operating system configurations and briefly describes the evaluation scenarios and benchmarks. Section 3 characterises the performance and energy efficiency of the hardware and operating system platforms we use and presents a before-and-after comparison. The empirical results are then discussed in Section 4, followed by a presentation of relevant related work in Section 5. Finally, Section 6 gives a brief summary of our work.

2 Fundamentals and Methodology

In this Section, we present the computer hardware used for the experiments, explain the operating system configurations required for them and describe the evaluation scenarios as well as the benchmarks.

2.1 Hardware Platform

In addition to the processor system components of our test device, especially the NVRAM equipment, the raw data of the main memory regarding timing and bandwidth are interesting as a reference for the theoretical (manufacturer's specification) performance. The latter is especially significant as a baseline for the subsequent evaluation.

Processing Systems The evaluation platform used to run and benchmark both Linux and FreeBSD is a Dell PowerEdge R650 equipped with two Intel® Xeon® Gold 6330 processors. Each processor has 28 cores and 56 threads, respectively. Their base frequency is 2.0 GHz, even though they can boost to up to 3.10 GHz. As conventional memory, there are eight 32 GB DDR4 RDIMMs.

For NVRAM, there are eight DIMMs of Optane™ Persistent Memory 200 with a capacity of 128 GB each. Both kinds of memory are distributed evenly between the CPUs, populating each memory channel with one DIMM. In total, the system has 256 GB RAM and 1024 GB NVRAM. Even though the available RAM and NVRAM support up to 3200 MT/s, both are limited by the CPUs to 2933 MT/s. The NVRAM is configured to App Direct Mode.

NVRAM Baseline The manufacturer specifications of the memory bandwidth for the 128 GB NVRAM DIMM are shown in Table 1. The idle average latency

	Access granularity	
Access mode	64 B	256 B
100% read	1.86 GB/s	7.45 GB/s
100% write	0.56 GB/s	2.25 GB/s
67% read, 33% write	1.06 GB/s	4.25 GB/s

Table 1: Optane^{TM} Persistent Memory 200 performance [6].

for 64 B accesses in App Direct Mode is given as 340 ns. Note that each load or store instruction fetches a 64-byte cache line. At the persistent memory module, this results in a read or write of 256 bytes of data accessed [4].

For the evaluation carried out here, namely running "vanilla" Linux and FreeBSD, respectively, directly from NVRAM, the access mix of 67% read and 33% write is baseline relevant. In contrast, the (to-be) NVRAM-based backup of the volatile system state will have 100% write in focus.

2.2 Operating Systems

While our modified versions themselves would allow running on an “NVM-only” platform, we can neither modify the hardware nor the firmware. Common problems tackled by both implementations revolve around x86-64’s startup process, which requires physical memory below 4 GiB for switching from real mode to long mode. In addition, some legacy drivers depend on 32-bit addressable memory, while NVRAM populates physical addresses well above 4 GiB.

However, we would like to point out that once the system finished booting, all allocations are served from NVRAM only. As such, the RAM is solely used as a workaround for hardware and firmware limitations and not used for essential functionality of the operating system in execution.

Linux NVRAM is already extensively supported by current versions of Linux and can be used as a block device (DAX), directly mapped into the address space of an application or extend the system memory. Together with the ability to dynamically on- and offline memory regions during runtime, usage of RAM could be almost eliminated for userspace applications. However, as the memory of the kernel itself can not be moved, it would remain in RAM.

To overcome this limitation, the most recent version available during the development process (*v6.1*) was modified to allow to only use NVRAM already from the early boot stages on. During bootstrapping, a stub decompresses the kernel image and allows to randomise the used kernel start addresses. To detect suitable regions, as well as later on, when initialising the memory allocation

pools, a firmware-provided memory map is used. As this map also provides information about non-volatile memory regions, we introduced a kernel parameter (`memtype`) to allow the selection of memory types to be used on each boot.

FreeBSD As the basis for our changes to the FreeBSD system, we used the 13.1 release, the most recent one at the time of writing. The current state of NVRAM support in FreeBSD is limited to a device driver exposing the detected NVRAM DIMMs. From there on, it can be used as a fast backing store for conventional filesystems—which, however, does not make the whole system NVRAM-capable by itself.

FreeBSD uses a custom, single-stage UEFI bootloader maintained in lockstep with the rest of the operating system for non-legacy bootstraps by default. Based on the EFI memory map, we adjusted the loader to place the kernel and modules into NVRAM. In addition, the initial virtual address mapping used during the early boot stage had to be adapted to reflect this change.

The FreeBSD kernel for x86-64 has a documented dependency on the kernel being loaded into the lower 4 GiB of memory, and many internal invariants rely on that property. We identified each of these reliant components in the kernel and modified them to work with the altered kernel placement. With the kernel placed in NVRAM, the initialisation of the memory subsystem had to be changed as well. Based on the EFI memory map, most of the RAM pages are ignored. A small, single-digit number of RAM pages in the low physical address range are set aside and stored in a separate data structure to serve the architecture-specific requirements. In addition, all NVRAM regions are assigned to the NUMA domain of their corresponding CPU in order to avoid memory accesses across domains. To satisfy the 32-bit address DMA requirements of old drivers, IOMMU-assisted DMA remapping had to be enabled as well. This required no further changes aside from setting the appropriate kernel parameter, however, the DMA remapping driver requires a single page of RAM below 4 GiB for itself. These changes by themselves suffice to fully bootstrap the kernel in NVRAM with only a single CPU core. For multi-core support the set aside memory was used to initialize a transient allocator, which can be used to satisfy any memory allocations to low-address memory. The architecture of x86-64 requires 20-bit addresses for startup and 32-bit addresses for the transition to long mode. With all cores online, all references to the transient allocator and DRAM memory are dropped, resulting in the execution of the operating system and all future allocations being served from NVRAM.

2.3 Evaluation Approach

During the evaluation, the systems were mostly isolated from any outside-induced noise, by shutting down any non-evaluation–related service and disabling SSH access on the standard port.

As our modified Linux allows to select the memory region type that should be used as main memory dynamically by a command line parameter, the same

binary had been used in the comparison. Our configuration is based on Debian, with some unused modules removed.

The evaluation of the FreeBSD systems is done on the same hardware with the same kernel configuration (*GENERIC*). Both kernels and loaders (original and modified) are loaded via iPXE netboot, while the whole FreeBSD userland is already installed on the local harddrive.

All our power measurements are conducted using an external measurement device, the Microchip MCP39F511N Power Monitor Demonstration Board [12]. The server's (cf. Section 2.1) two power-supply units (PSUs) are each connected to one of the MCP39F511N measurement channels, allowing for a realistic whole-system measurement, including the power losses in the PSUs and other hardware components. The MCP39F511N is connected to another server to ensure that the power measurement does not influence the system under test. Thus, our measurement approach provides realistic, real-world power values of the otherwise unaltered, off-the-shelf hardware platform.

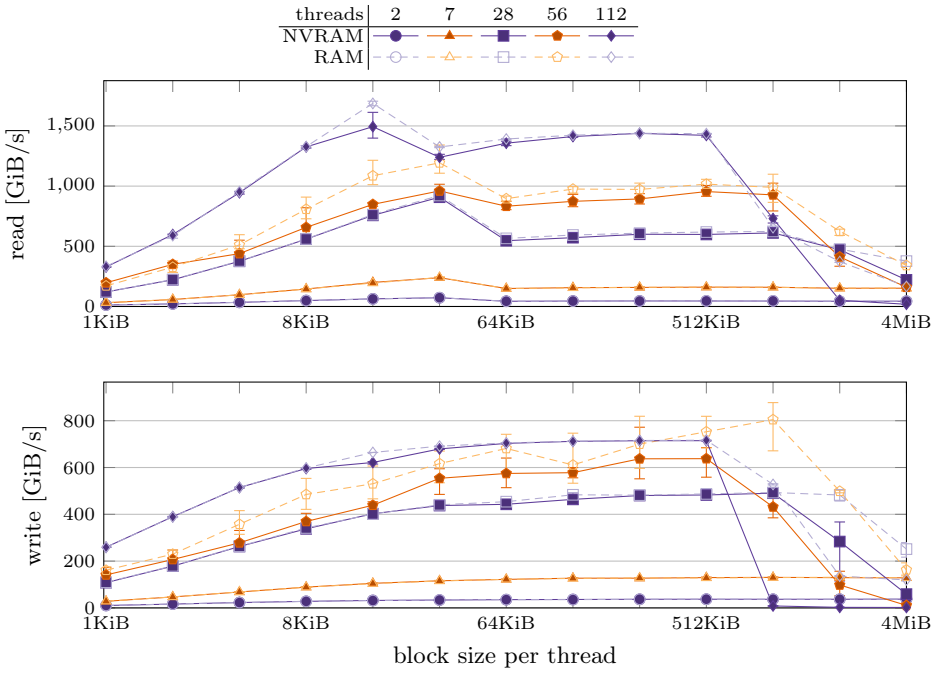
Microbenchmark To gain detailed measurements regarding the performance characteristics of NVRAM vs. RAM and placement strategies of the operating systems under test, the *sysbench* [14] test suite was used. The baseline was established by using sysbench's *memory* benchmark. For estimating the impact on the whole system performance, *fileio* was used, which interacts with more of the OSs subsystems, such as the virtual file system, buffer caches and system call handling. The sysbench version used is supplied by the operating system vendor, Debian respectively FreeBSD—in both cases version 1.0.20.

During testing, different combinations of sysbench configurations were run multiple times to test a variety of workloads. These options for adjustment are the number of used threads, the block size of memory chunks and the kind of access. By modifying the number of threads, the OS is forced to make NUMA-placement decisions, while different block sizes influence cache usage.

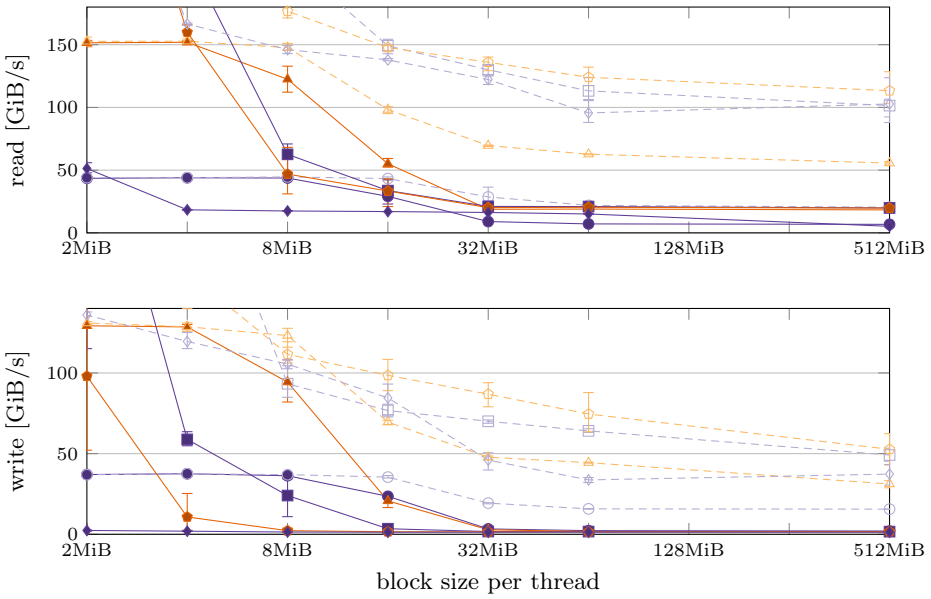
Application In order to evaluate the systems' performance with a real-world workload, the build-system of the respective OS is deemed as suitable. Most of the workload is comprised of source to binary translation, whereas source files are read into memory, optimised and written back as an object file. Since the option `-pipe` is used during translation, no intermediate files are generated, leading to greater memory consumption. During linking, all object files are read again, and merged into a single binary. In between are some transformations made by tools such as *awk* for generating header files and *gzip* for compression of build artefacts.

3 Performance Characterisation

Following [13], we measure the performance of the NVRAM hardware (Intel Optane$^{\text{TM}}$) underlying the experiments on the one hand and the operating systems on the other. The focus is on the timing and power requirements of Linux



(a) within the caches



(b) out of the caches

Fig. 1: sysbench memory throughput for different block sizes per thread executed on top of our modified Linux.

and FreeBSD for the conventional DRAM-based deployment and the NVRAM-based ("NVM-only") approach to be evaluated. Every measurement was repeated five times, and the presented numbers are the median thereof with a 95% confidence level.

Memory Benchmarks The sysbench benchmark was conducted in the local, non-shared configuration, where each thread allocates its own memory buffer to perform the read and write operations. The memory read benchmark (Fig. 1a) reveals in the case of Linux the impact of the cache architecture of the CPU. NVRAM and DRAM curves are close together with sharp drops near the 32 KiB and 512 KiB tick marks, which strongly correlates with the sizes of the private level 1 (48 KiB) and level 2 (1.25 MiB) data caches per core. In the case of 112 threads, two hyperthreads per core share the private caches, and the drop happens therefore earlier. As expected, the NVRAM throughput also drops sharply from the DRAM curve as well after the L2 boundary due to the inclusive caching strategy of the CPUs, which fills up the shared 42 MiB L3 cache as well. Once we fall out of the cache hierarchy (Fig. 1b) the NVRAM achieves around 21 GiB/s with 28 threads. DRAM scales further to 56 threads and achieves close to 121 GiB/s (Fig. 1b), where NVRAM still remains at 21 GiB/s. NVRAM does not scale very well for a high number of threads, the difference to DRAM increases to a factor between 5 to 6. However, at 112 threads where hyperthreads come into play, the NVRAM rate sharply drops to less than 6 GiB/s while DRAM still holds up at approx. 110 GiB/s which gives us a high penalty factor of about 18.

The memory-write benchmark for Linux (Fig. 1a) shows no sharp drops at the boundary of the L1 cache but at the L2 boundary, similar to the read benchmark, Generally, when we still operate in the caches the write rate is about half the read rate which is most probably caused by the fact, that each cache line that is newly written to must first be fetched from memory as well. Once we are out of the caches (Fig. 1b), NVRAM maxes out at approx. 2.1 GiB/s already with two threads. In the case of DRAM, the combined memory controllers are then saturated with 56 threads offering a combined write bandwidth of about 56 GiB/s while NVRAM stays at approx. 1.7 GiB/s, which gives us a severe penalty factor of nearly 33 for the experiments with a large number of threads. The curves for FreeBSD are largely similar, therefore we do not show them here. There is, however, a significant difference in the cache case. FreeBSD always revealed a lower throughput than Linux, which might be caused by less aggressive settings for the frequency scaling of the CPUs.

The theoretical performance maximum of our system should be 8×0.56 GiB/s for 64 B write and 8×1.86 GiB/s for 64 B read if all our eight DIMMs in the system can really be operated in parallel. The measurements revealed a significantly higher rate for the read operations, which can probably by attributed to the wide internal 256 B accesses of the DIMMs. However, the data rate of the write operations was less than half of the theoretical maximum. We attribute this to the fact that there is always a simultaneous read stream back to the caches when one continuously writes to cacheable memory. Additionally, read and write

rates strongly depend on NUMA placement strategies as well as thread placement/migration strategies of the underlying operating systems, which we did not change at all.

When the combined working sets of all applications fit mostly into the cache hierarchy, there is only a slight performance penalty for the NVRAM-based systems caused by working set changes. Memory-bound applications, however, might suffer severely, especially the low write rates are the biggest problem and can inhibit parallel execution significantly.

File Benchmarks In Figure 2, the results of the sequential memory read and write benchmarks using the traditional file system interface via system calls are shown for Linux and FreeBSD. In the case of NVRAM, Linux reaches its read peak of about 5.4 GiB/s already with four threads, while FreeBSD reaches its peak of nearly 4.9 GiB/s with 28 threads. From 14 threads on, however, the curves of the two systems are relatively close together. Both systems do not scale well beyond seven threads, irrespective of the type of memory used. In the case of DRAM, Linux outperforms FreeBSD by a large margin, 15.5 GiB/s achieved already with seven threads vs about 10 GiB/s with 28 threads. All in all, for the read benchmark, the performance penalty of NVRAM is about 3x in the case of Linux and approx. 2x for FreeBSD.

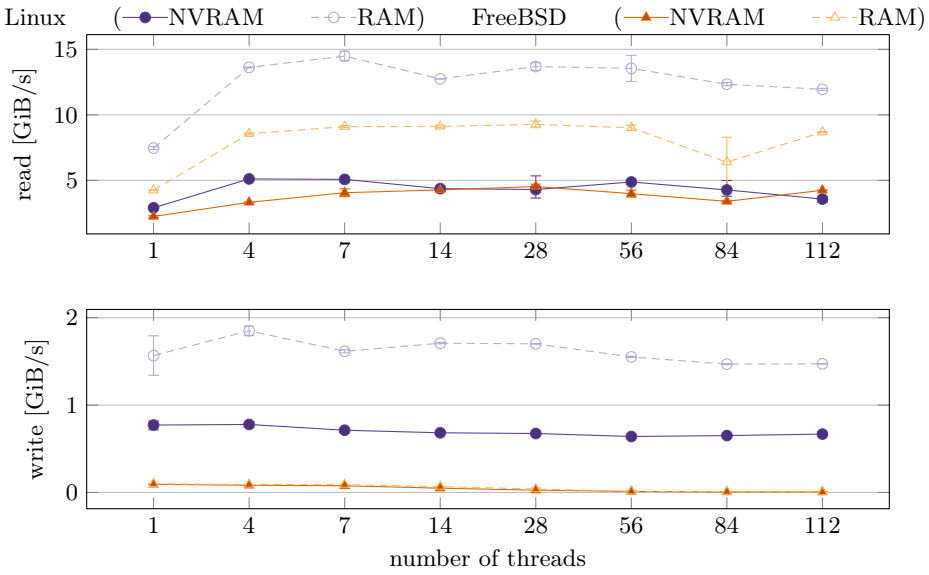


Fig. 2: sysbench fileio sequential read and write throughput

In the case of the write benchmark (Fig. 2), the throughput is way lower than in the case of read, probably because an `fsync()` operation is automatically inserted for every 100 write requests by the benchmark, that caused real write

operations to storage. The scalability is very low for this benchmark, both systems are saturated with a few numbers of threads already. We strongly assume that even the occasional `fsync()` operations quickly saturated the SSDs of our system. Linux achieved slightly less than 2 GiB/s with DRAM and four threads. With NVRAM it achieved slightly less than 0.85 GiB/s with four threads as well. The penalty for NVRAM was about 2.4x. In the case of FreeBSD, one thread already achieved the peak of 0.1 GiB/s for NVRAM and about 0.18 GiB/s for DRAM. In all cases, the penalty for NVRAM was less than 1.5x.

Parallel Make Benchmarks For the before and after comparison, we use a parallel make on each of the system's build infrastructure to show how painful an "NVM-only" approach really is in terms of system performance and whether or not the differences in performance are lost in the system-related background noise and are no longer perceived.

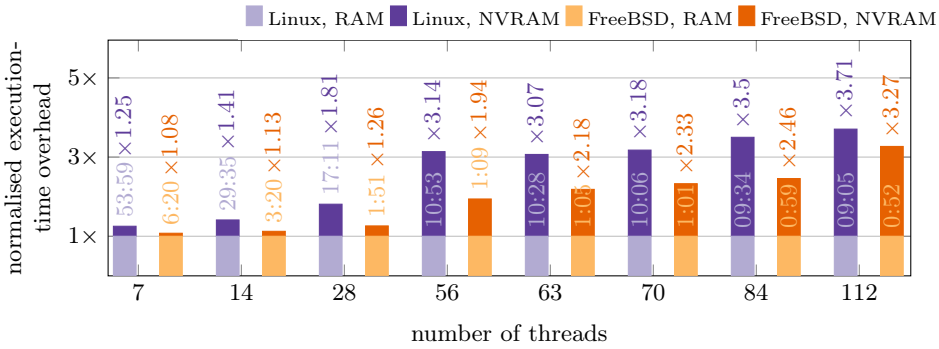


Fig. 3: Linux and FreeBSD parallel make

In Figure 3, parallel makes of Linux and FreeBSD kernels are shown were 7–112 threads were applied for system generation. This gives us a first insight what performance penalties can be expected in a real system used for software development. Interestingly enough, the NVRAM-based systems can hold up well to their DRAM-based counterparts, as long as no more than 28 threads are used. A performance loss of around 25% is way better than the memory performance numbers suggest. The overall "system jitter" hides the disadvantages of NVRAM to a large extent here. From 56 threads on, the NVRAM-based system continuously loses ground but never more than a factor of four. In the case of FreeBSD most benchmarks even revealed a factor of less than three. The DRAM-based systems are able to continuously achieve a slight performance benefit from more threads but at a very low margin. There is only a negligible achievement from 56 threads onwards, probably due to increasing serial portions of the build process.

In absolute times, the generation of the FreeBSD system was nearly a magnitude faster than the generation of Linux, probably due to the size of the sources.

The penalty factors for NVRAM are slightly better (from 1.08x to 3.27x) for any degree of parallelism than in the case of the Linux kernel. But the general observations are otherwise similar. For up to 28 threads the NVRAM-based system holds up very well with its DRAM counterpart, with a penalty of less than 1.3x.

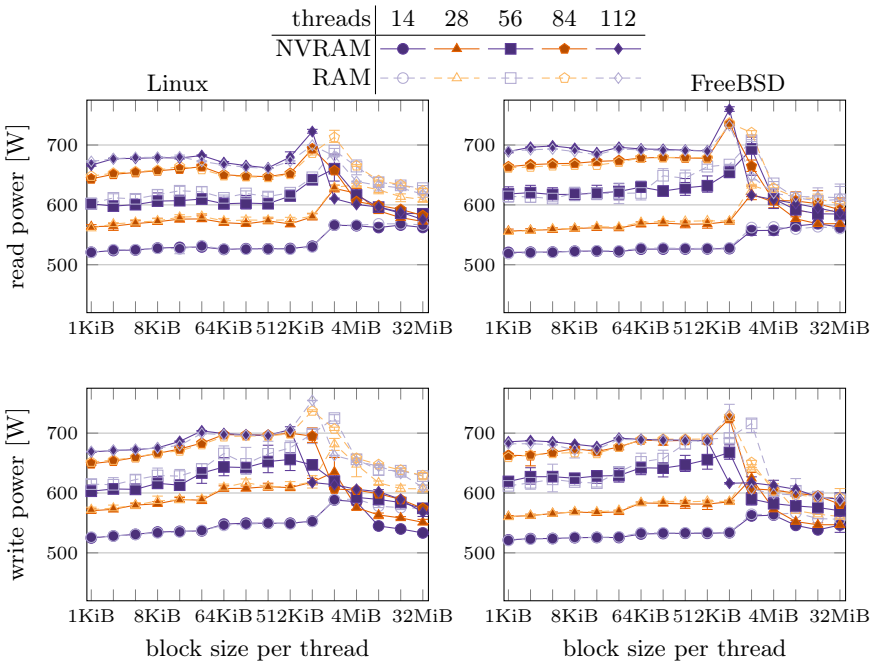


Fig. 4: Whole-system power draw for the benchmarks presented in Figure 1.

Power Consumption Figure 4 illustrates the average power draw for the sysbench memory evaluation scenarios presented in Figure 1: On Linux, the power demand for reading and writing blocks of up to 512 KiB (or 1 MiB, depending on the level of parallelism) from/to NVRAM (solid shapes and lines) is almost identical compared to running the respective benchmark on DRAM (blank shapes and dashed lines). When surpassing the boundary of 512 KiB/1 MiB—the same block size the overall throughput decreases (see Figure 1a) due to caching effects—the power draw starts to decrease alongside the throughput. As for the throughput, the power decrease is more pronounced when operating on NVRAM due to its lower performance, causing the same benchmarks running on DRAM to cause a higher power draw while exhibiting a higher throughput. This decrease in power draw is most likely caused by the CPUs becoming idle by waiting for the memory. Also, in terms of power draw, the difference between reading from and writing to memory is small: While writing to both DRAM and NVRAM draws slightly more

power for small block sizes, these values begin to fall below the corresponding values for reading when surpassing the block size of 512 KiB/1 MiB.

When running the same benchmarks on FreeBSD, the observed values are very similar to the values observed on Linux.

4 Discussion

In this Section, we discuss insights gained from the experiments and deduct the following *Recommendations for Action* (RFAs) to adapt current operating systems to NVRAM:

NVRAM Awareness at OS-Level (RFA #1) Today's hardware systems still need some RAM because of legacy considerations. Typical Intel-compatible CPUs reveal a kind of embryonal development in the boot phase going through 16-bit real and various protected modes with segments, 32-bit mode with segments, paging and PAE, and finally, paged 64-bit mode. During these phases, the lower 32-bit physical address space must be used and must be changed later. Finally, all volatile parts of the physical memory must be kept away from the kernel to boot, which requires some system-dependent adaptations. All in all, adapting the boot trampoline and kernel initialisation is rather tedious and fiddly.

The benchmarks have clearly shown that a naive usage of NVRAM by simply replacing DRAM with NVRAM causes a significant performance penalty, depending on the workload. A simple system like that might still be useful because of its pure capacity. Huge out-of-core workloads can, in principle, be put onto rather small and relatively cheap machines. Similarly, thousands of low-intensity processes might run as well. For those applications, the Optane Memory Mode that manages some of the available DRAM as hardware-controlled cache, would probably be sufficient, for example. However, in that mode, a part of the DRAM is lost and NVRAM is treated as volatile.

Transition from DRAM to NVRAM (RFA #2) Current Linux systems can already use NVRAM following the SNIA recommendations. Furthermore, (parts of) the NVRAM can also be used by the virtual memory subsystem as a kind of overflow memory in times of high memory pressure. The NVRAM is handled as a "far away", still volatile, NUMA domain in this case without taking advantage of its persistence features at all. In contrast, we clearly want to go beyond pure capacity scaling. In further work, we want to use NVRAM truly as storage-class memory, targeting robust systems, that provide efficient whole system persistence with stock hardware and fast recovery times after power losses and the like. Therefore, in the next step, we will develop a suspend-to-NVRAM mechanism, that allows us to freeze and wakeup entire systems with minimal latency on demand. Following [5, 2], we will integrate this mechanism into the handling of power failures, such that systems will be able to survive power losses, ideally with low amounts of residual energy from the power supply or a low-capacity UPS.

Persistency Guarantees and Power/Energy (RFA #3) Contemporary operating systems, such as Linux and FreeBSD, are designed for running in and working with volatile main memory. Due to this volatility, the operating system conducts a variety of persistency measures—such as periodic file-system cache writebacks—just to protect from the loss of volatile data in the rare event of a power outage. Such persistency measures, however, come at the cost of computational and energetic overhead, and their implementation also increases the trusted code base.

Once the persistency of the main memory is guaranteed by the hardware, that is, the system is using non-volatile main memory instead of volatile DRAM, the software-implemented persistency measures become superfluous and can be removed. Despite the lower throughput of NVRAM, without such measures, the operating system can finally leverage the advantages of NVRAM over conventional, volatile DRAM and, thus, compensate for the lower performance at least partially—and achieve even higher performance compared to DRAM-only by implementing the subsequent RFA #4.

Architectural Changes due to NVRAM (RFA #4) Finally and most importantly, we will make virtual memory subsystems NVRAM-aware and develop a *virtual* NVRAM that will manage the DRAM only as a cache for NVRAM to mitigate the inherent performance penalties while retaining its persistence. Such a multi-level virtual memory subsystem is most likely needed anyway in the future since other memory technologies like *high-bandwidth memory* (HBM) also have to be integrated appropriately.

Apart from the memory hierarchy, NVRAM awareness offers plenty of opportunities for improvements ranging from simplistic ones to highly sophisticated schemes based on machine learning and similar techniques. For example, when a scheduler decides to move low-intensity processes to "slow" economy cores, its memory contents might be moved lazily to NVRAM as well. Program code often has a high locality of reference and fits into the internal CPU caches, thus code could always be placed into NVRAM without much performance loss, as our benchmarks show. Accessing code and data in large persistent file system caches would be possible without hard page faults but fast lazy mappings. All in all, the observed performance penalties can most certainly be overcome but require substantial changes to rather complex parts of current operating systems.

5 Related Work

There is as yet no Linux or FreeBSD that, together with all machine programs (i.e., applications), runs directly and exclusively from NVRAM. Thus, at present, our approach cannot be compared with other solutions on a level playing field.

This includes the concept of *whole system persistence* [8], which achieves the persistence of complete systems on the basis of special DIMMs that contain both DRAM and equally sized Flash memory. The content of the DRAM part is saved with the help of backup capacitors on the local Flash as soon as a power failure is

detected. The same applies to NV-Hypervisor [11], where DRAM content is also hardware-based persisted. In contrast, with our "NVM-only" vision, we do not expect any special hardware support other than NVRAM to keep main memory contents persistent.

Twizzler [1] seems to be an exception to these developments: It is presented as a system from which a complete NVRAM-based operating system can be built with a data-centric design. Whether Twizzler itself can be considered an "NVM-only" system, however, remains an open question. In addition, Twizzler must be understood as a replacement for an exokernel-based operating system (implemented in Rust) and would establish Linux or FreeBSD as a guest operating system, if at all. The programming model of Twizzler unfolds its positive effect, especially when used directly by the machine programs that are then to run in NVRAM; it does not make the latter transparent for legacy software as our approach does.

6 Conclusion

We strongly argue that NVRAM needs to be integrated appropriately into the memory hierarchy to realize its true potential. Since this is a highly complex undertaking, we have first shown with our work here that state-of-the-art operating systems can be run directly from NVRAM, which is a first to the best of our knowledge. What's more: entire software stacks, including legacy software, can actually be run from NVRAM and don't have to do without the persistence properties of this memory technology.

We have now established a nearly worst-case baseline for the possible performance of such systems under heavy load. While the synthetic memory benchmarks showed a wide variety of performance penalties, *all* of the kernel build benchmarks only revealed a penalty of significantly less than 4x. Thus, the expected degradation was drowned in "system jitter" to a large extent. The power draw increased only insignificantly and even decreased for several workloads, while the overall energy consumption was proportional to the prolonged execution times of the benchmarks.

Although these measurement results may seem discouraging, one should not forget that these numbers have been achieved with systems where we performed a rigorous technology swap from DRAM to NVRAM without taking *any* special precautions for efficient NVRAM operation. When we additionally use certain amounts of DRAM as a software-controlled cache for NVRAM integrated into the virtual memory subsystem, the way is paved for robust, moderately priced servers with huge memory capacities—and which provide an efficient abstraction layer in particular for legacy software that can also be directly run from NVRAM without any change. We want to show this in further work.

Unfortunately, Intel has ceased to produce the Optane DIMMs used for our work. However, first approaches like "memory semantics SSDs" were already announced that might fill the gap and lead in a similar direction. The idea is in

the world, and we strongly expect CXL-based solutions for storage class memory in the near future.

References

1. Bittman, D., Alvaro, P., Mehra, P., Long, D.D.E., Miller, E.L.: Twizzler: a data-centric OS for non-volatile memory. In: Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC'20). pp. 65–80. USENIX Association (2020)
2. Eichler, C., Hofmeier, H., Reif, S., Hönig, T., Nolte, J., Schröder-Preikschat, W.: Neverlast: An NVM-centric operating system for persistent edge systems. In: Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems. pp. 146—153. APSys '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3476886.3477513, https://doi.org/10.1145/3476886.3477513
3. Freitas, R.F., Wilcke, W.W.: Storage-class memory: The next storage system technology. IBM Journal of Research and Development **52**(4/5), 439–447 (Jul/Sep 2008)
4. Hady, F.T.: Faster access to more data. Technology brief, Intel Non-Volatile Memory Solutions Group, Intel Corporation, USA (2022)
5. Heiser, G., Le Sueur, E., Danis, A., Budzynowski, A., Salomie, T.l., Alonso, G.: RapiLog: Reducing system complexity through verification. In: Proceedings of the 8th ACM European Conference on Computer Systems. pp. 323–336. EuroSys '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2465351.2465383, https://doi.org/10.1145/2465351.2465383
6. Intel Corporation: Achieve greater insight from your data. Product brief (2022)
7. Kannan, S., Qureshi, M., Gavrilovska, A., Schwan, K.: Energy aware persistence: Reducing energy overheads of memory-based persistence in NVMs. In: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT). pp. 165–177 (2016). https://doi.org/10.1145/2967938.2967953
8. Narayanan, D., Hodson, O.: Whole-system persistence. In: Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12). pp. 401–410 (2012)
9. Ransford, B., Lucia, B.: Nonvolatile memory is a broken time machine. In: Proceedings of the 2014 Workshop on Memory Systems Performance and Correctness (MSPC '14). p. 5 (2014)
10. Ren, J., Hu, Q., Khan, S., Moscibroda, T.: Programming for non-volatile main memory is hard. In: Proceedings of the 8th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '17). pp. 1–8. No. 13, ACM Digital Library (2017)
11. Sartakov, V.A., Kapitza, R.: NV-Hypervisor: Hypervisor-based persistence for virtual machines. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 654–659 (2014). https://doi.org/10.1109/DSN.2014.64
12. Technology, M.: MCP39F511N power monitor demonstration board. https://www.microchip.com/en-us/development-tool/ADM00706, https://www.microchip.com/en-us/development-tool/ADM00706
13. Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J., Swanson, S.: An empirical guide to the behavior and use of scalable persistent memory. In: Proceedings of the 18th USENIX Conference on File and Storage Technologies. FAST '20 (2020)
14. Zaitsev, P.: sysbench. https://github.com/akopytov/sysbench (2004)