Cost-Effective Memory for HPC, AI and Enterprise Storage Planning

Last update: **9 September 2026**

Measured Data from real HPC, AI and Storage Workloads on ThinkSystem Servers

Price to Performance Comparison of RDIMMs vs MRDIMMs for current generation of Memory

Discussions on situations where workloads are unaffected or minimally affected by underpopulating server memory

Which memory options make the most sense for mixed workload datacentres

Conor Elrick

LENOVO PRESS

Table of Contents

Abstract

With DRAM costs rising dramatically due to increasing AI-driven demand and constrained supply, fully populating memory in systems is no longer always economically justified. As memory becomes an increasingly significant component of system cost, organisations must carefully balance performance requirements against infrastructure budgets. The terms *RAMageddon* and *RAMpocalypse* have emerged to describe the unprecedented increase in DRAM pricing and the resulting pressure across the technology industry, from consumer devices to hyperscale datacentres. In this paper, we examine practical strategies available to maximise workload performance while minimising memory-related expenditure in this constrained market.

Using Lenovo ThinkSystem platforms, we present a series of price/performance studies spanning HPC, enterprise AI, and storage workloads, investigating where memory capacity can be reduced without materially impacting application performance or overall datacentre productivity. The workloads evaluated represent a broad range of academic and industrial use cases, including computer-aided engineering (CAE), computational fluid dynamics (CFD), molecular dynamics, computational chemistry, weather and climate modelling, media processing, AI training, AI inference, and enterprise storage. By analysing these workloads collectively, we identify memory population and sizing strategies that deliver the optimal balance of cost and performance for mixed-workload environments. The results provide practical guidance for organisations seeking to maximise infrastructure value, increase resource efficiency, and maintain user productivity during a period of sustained memory cost inflation.

1 Introduction

Designing a HPC cluster or AI datacentre involves deciding the perfect balance of hardware so as not to bottleneck the maximum performance of one component by underinvesting in another. In modern datacentres the performance of applications (and therefore overall throughput of the datacentre) depends on the choice of CPU, GPU/Accelerator, Memory, Storage and Networking. In 2025-2026, the pricing of DDR memory increased significantly due to the increased demand from the global adoption of AI, disrupting the decades long trend of memory pricing per GB decreasing over time as seen in Figure 1.

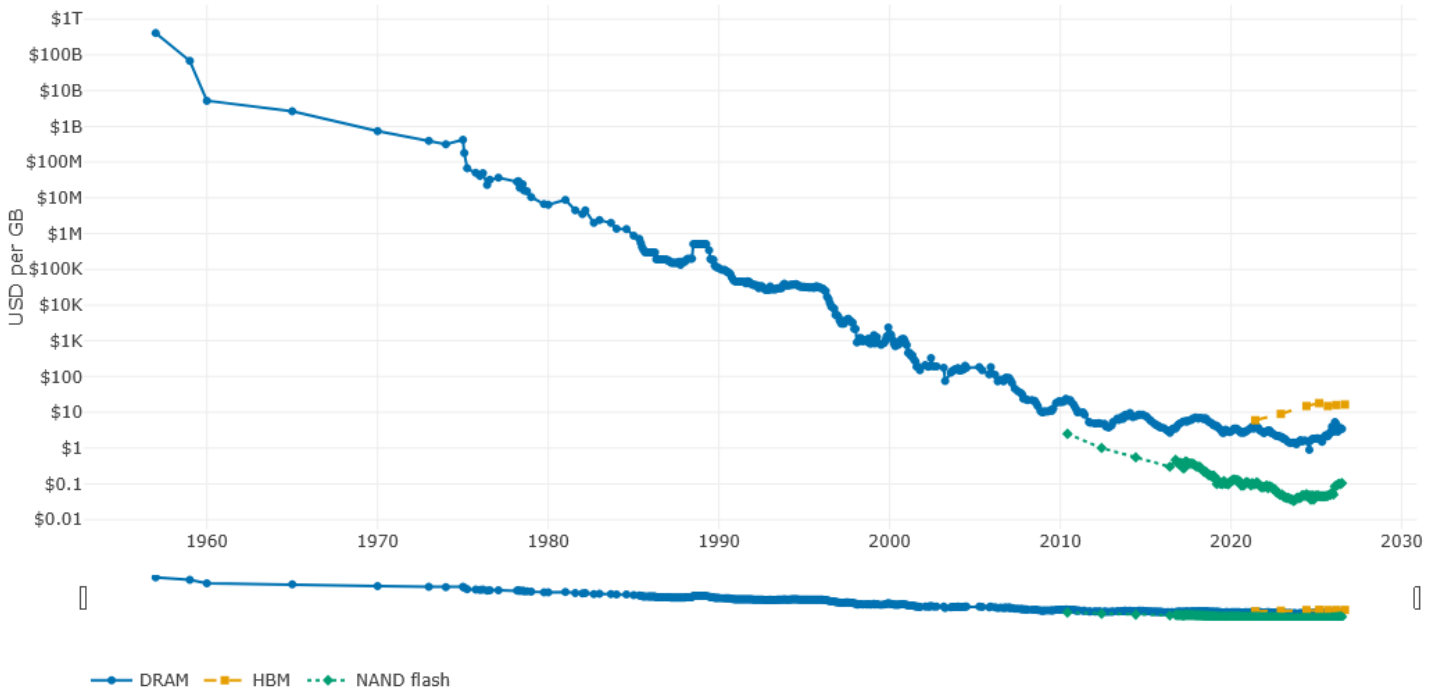


Figure 1 Historical Memory Pricing from 1950's until today showing cost of a GB of memory trending steadily down over time. Price per GB is the cheapest listed retail price in nominal USD - Taken from Stanford University website [1].

With the supply constraints affecting only the price of memory (and components that make use of said memory), the price of the memory as a fraction of the price of the full solution has been increasing significantly since the end of last year. Figure 2 shows the price of a standard 48U compute rack populated with three Lenovo ThinkSystem N1380 Neptune water cooled chassis [2], each with 8 trays / 16 nodes of Lenovo ThinkSystem SC750 V4 [3] powered by Intel 6980p processors[1] as an example of a deployment for a HPC site. In this plot we isolate the cost of the twenty-four 64GB DDR memory units per server (twelve memory channels per CPU and dual socket system) from the remaining cost of the system. Between December 2025 and June of 2026, this fractional cost more than doubled for this setup due to the difference in the increase of memory price and the increase in the price of the other components in the system.

Beyond just the price, the increase in demand for memory has led to an increase in procurement risks as suppliers create

[1] Other processors are supported for this server https://lenovopress.lenovo.com/lp2009-thinksystem-sc750-v4-neptune-server#processors

longer lead times to deal with prioritization of HBM/LPDDR5X memory (used in GPUs). This then requires customers to make early commitments to ensure hardware can be sourced and delivered according to their deployment timeline.

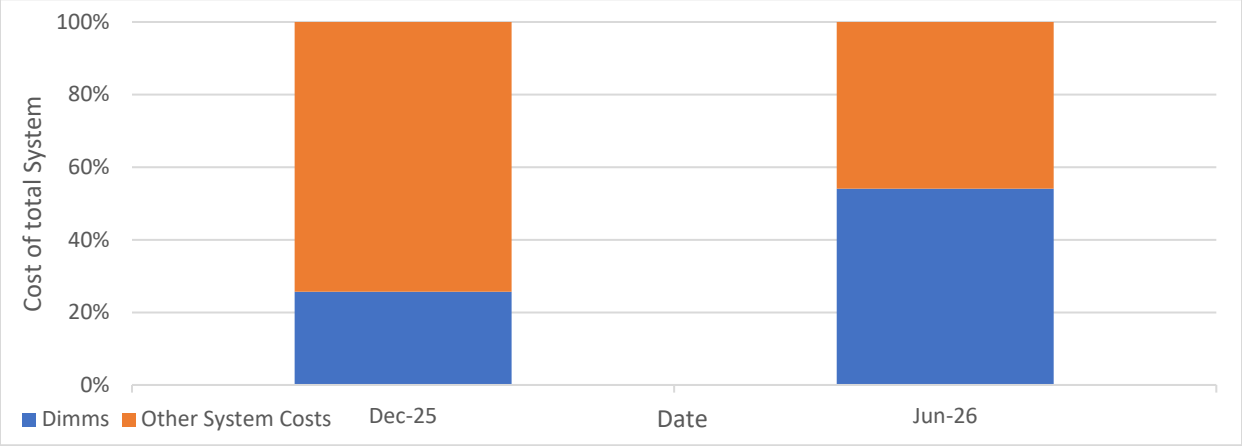


Figure 2 Percentage Cost of a ThinkSystem SC750 V4 Server with 24x RDIMMs and Intel 6980P servers. Total price is based off of a rack containing three N1380 Neptune cooled enclosures each populated with 16 compute nodes.

At time of writing the situation is projected to continue for some time. Sanjay Mehrotra, CEO of Micron, wrote in a public investors report in late June of 2026 [4]:

DRAM and NAND industry demand continues to significantly exceed industry supply. We expect tight conditions to persist beyond calendar 2027 as a result of AI-driven demand across all segments coupled with structural supply constraints.

With this in mind, it is important that we develop an understanding of the cases on where it currently makes sense from a price to performance point of view to use MRDIMMs instead of RDIMMs and to use all the available memory channels or only a fraction of the available channels. For HPC and AI workloads, we present data in this whitepaper to help datacentre architects, administrators, and infrastructure planners decide on the best approach to avoid overspending on memory for their solutions.

The structure of the paper is as follows:

- In Section 2 we look at the effect of underpopulating memory on a selection of artificial HPC benchmark kernels. This is both to show the extreme conditions that will bound the behaviour of real-world workloads, and to show how underpopulating memory can impact the sizing of a cluster if it is done based on the performance of these kernels.
- Section 3 contains price/performance results for HPC workloads looking at underpopulating memory with RDIMMs and MRDIMMs to conclude where it makes sense to use each of these options.
- Section 4 contains a similar study but for AI training and inference workloads.
- In Section 5 we look at the impact on IO performance on the Lenovo DSS-G storage for HPC [5] as an example of the effects that underpopulating memory can have on enterprise storage.
- Finally in Section 6 we take a high-level view of all the results presented in this paper to summarise what the key recommendations are for designing HPC and AI systems in the constrained market.

2 Understanding the performance of HPC Microbenchmarks with underpopulated memory channels.

In this section we look at the impact on some common artificial HPC microbenchmarks that are often used to either understand the bounds around the performance of real-world applications from the hardware, or to size the full cluster. We start with a summary of the methodology used for this paper and then go on to review the differences we have observed for each kernel, explaining the reasons for the differences in performance for each.

2.1 Hardware, software and Methodology

In the Lenovo Benchmarking Centre, we deployed a Lenovo ThinkSystem N1380 Neptune Enclosure containing 8 Trays / 16 nodes of ThinkSystem SC750 V4 Servers. Each node contains:

- Dual Socket Intel Xeon 6980P 128c, 500W Processors

And then either

- 24 (12 per socket) Units of 32GB DDR5 RDIMM Memory operating at 6400MT/s
- 16 (8 per socket) Units of 32GB DDR5 RDIMM Memory operating at 6400MT/s
- 24 (12 per socket) Units of 32GB DDR5 MRDIMM Memory operating at 8800MT/s
- 16 (8 per socket) Units of 32GB DDR5 MRDIMM Memory operating at 8800MT/s

For the SC750 V4 server, currently the only supported memory configuration has all 12 channels per CPU fully populated. There are though 3 failover configurations which still allow the server to boot – each of which have 8 memory channels disabled (4 per CPU), leaving 16 total DIMMs active per server. In this study where 8 DIMMs per CPU were needed, we manually failed DIMM 7 and 18, forcing DIMMs 5,6,8,17,19,20 to also be disabled by the BIOS to put us into one of the valid failover configurations. This could also have been accomplished by physically removing the DIMMs in the slots mentioned above.

Option	Setting used
OS	Rocky Linux 9.5
Linux Kernel	5.14.0-503.35.1.el9_5.x86_64
UEFI – Workload Profile	“HPC Mode” with the additional changes below
UEFI – Additional Options	UEFI.Processors_UncoreFrequencyScaling "Enabled" UEFI.Processors_VirtualNuma "Disabled" UEFI.Processors_SNC "Enabled" UEFI.Processors_L0p "Enabled"

Table 1 Server Options and OS info for the ThinkSystem SC750 V4 servers used in this study

In Table 1 we present the operating system and UEFI options used in this study. With the ThinkSystem SC750 V4 servers,

there is a Workload Profile option[2] designed to set the UEFI options on the system to match the typically used options for that cluster use-case. Beyond the standard HPC profile, we adjusted the following settings:

- UncoreFrequencyScaling – To dynamically change CPU speed based on the workload
- VirtualNuma – To disable creating additional NUMA domains beyond what is done with SNC
- SNC – To enable dividing up the CPU package into multiple NUMA nodes with cores and last level cache
- L0p – To disable the power saving from CPU UPI interface and decrease latency

It was found that this combination gave the best overall performance for a mixed workload HPC cluster.

2.2 Performance Study of HPC Kernels

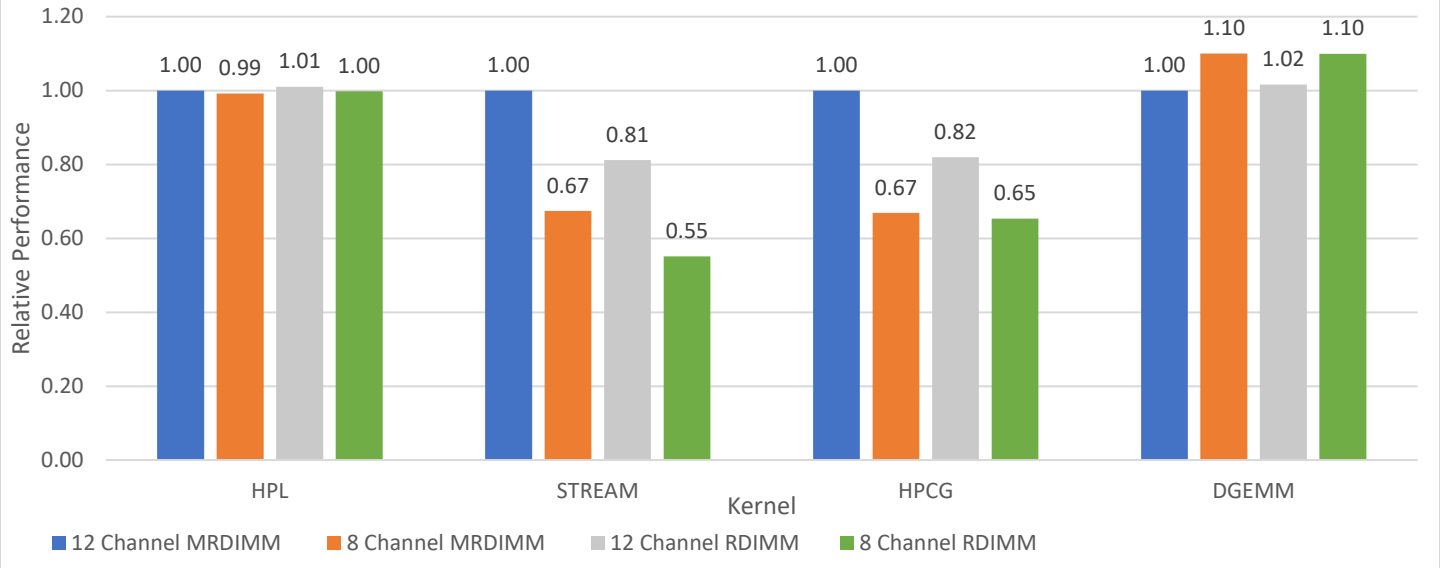


Figure 3 Performance of HPL, Stream, HPCG and DGEMM on ThinkSystem SC750 V4 with Intel 6980p Processors normalised to the fully populated configuration with MRDIMMs

In Figure 3 we present performance numbers for four standard artificial HPC benchmarks to compare having fully populated RDIMMs and MRDIMMs with having only 8 out of 12 CPU memory channels populated. For each kernel, results have been normalised to result from fully populated MRDIMMs.

2.2.1 HPL

HPL (High Performance Linpack [6]) is a tool designed to stress the CPU(s) in a node to measure their total computational throughput. Since it is the purest form of a CPU bound code, it is commonly used to provide a limit to the performance of real-world applications which will be bound by CPU, Memory, IO or the network for communication. For this reason, it is also commonly used for sizing a HPC cluster, measuring the total compute capability of a cluster. In Figure 3 we see that the HPL result is effectively unchanged for the 4 memory configurations studied, which was to be expected for this kernel. This highlights though that underpopulation of memory can be an option without affecting the sizing of the cluster if it is done with HPL.

[2] https://lenovopress.lenovo.com/lp2009-thinksystem-sc750-v4-neptune-server#uefi-operating-modes

2.2.2 STREAM

The Stream benchmark [7] is designed as a memory bandwidth test for a single compute node on a HPC cluster. Similarly to how HPL will bound the compute performance of real-world applications, Stream will bound the memory bandwidth performance of applications. When going from 12 to 8 memory channels active, the total memory capacity and bandwidth drops by 1/3 which is why we see a drop of around ~0.33 in Figure 3 between 12 and 8 channels for both RDIMMs and MRDIMMs.

2.2.3 HPCG

The HPCG (High Performance Conjugate Gradients) benchmark [8] is used to complement HPL benchmark by focusing on a different set of computational and data access patterns than what would be well approximated with HPL. Like Stream, HPCG is known to be very memory sensitive so it makes sense that the behaviour of HPCG in Figure 3 closely resembles the behaviour seen with the Stream benchmark. Both Stream and HPCG should in principle represent the worst possible performance drop that could be seen in a real-world application when underpopulating memory. Since real-world applications will lie somewhere between HPL and Stream in terms of CPU-Memory dependency, we need to compare the performance difference with the cost saving from underpopulating memory channels to figure out if underpopulating memory is a good idea.

2.2.4 DGEMM

Finally, to wrap up our discussion on kernels we look at a DGEMM (Double Precision General Matrix Multiply) implementation. Similar to HPL, DGEMM will test the compute capabilities of the HPC server except that it uses different access patterns to calculate the matrix multiplications. This is important since Figure 3 shows an increase in DGEMM performance when underpopulating memory, which we wouldn't expect with a purely compute bound kernel.

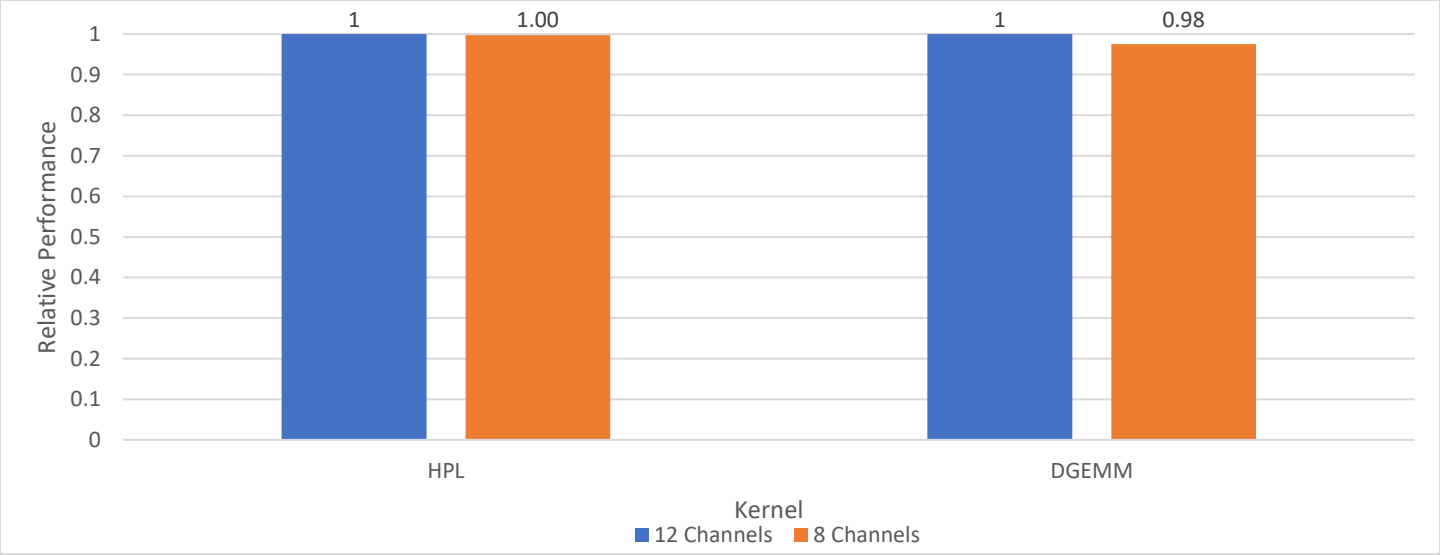


Figure 4 Performance of HPL and DGEMM on ThinkSystem SC750 V4 with Intel 6980p Processors and MRDIMMs normalised to the fully populated configuration with MRDIMMs with SNC disabled in UEFI

In Figure 4 we show that this unexpected behaviour seen in Figure 3 with DGEMM performance increasing as we lower memory bandwidth is no longer present when we disable Sub NUMA Clustering (SNC) in the UEFI options. This suggests that with SNC enabled there is a significant performance penalty due to the communications between NUMA domains

with DGEMM that is not present in the way that HPL performs the data access. After further study it was found that the 12 channel DGEMM results with SNC enabled could be brought closer to the 8 channel results by adjusting THP (Transparent Huge Pages) and NUMA affinity.

In the following section, we extend this discussion to real-world applications and start to look at cost data to discern for what workloads either swapping from RDIMMs to MRDIMMs or the underpopulation of memory channels makes sense for a real deployment.

3 Impact of MRDIMMs and Underpopulated Memory Channels on Real-World HPC Workloads

In this section we examine the price/performance ratio for a broad range of HPC workloads to determine when RDIMMs or MRDIMMs provide the best value and whether underpopulating memory channels can be a cost-effecitve deployment strategy.

3.1 Summary of codes and Methodology

As part of this study we looked at 28 real-world workloads across 14 HPC applications covering the Computer-Aided Engineering (CAE), Molecular Dynamics (MD), Chemistry and Weather/Climate domains. Each code was compiled and ran on the target hardware using our best internal recipes from the application performance team. For codes which used MPI, the number of ranks was varied in order to achieve the best performance. For multithreaded and hybrid (MPI and OMP threads) workloads, the number of MPI ranks and OMP threads was then also varied to ensure that our setup was optimal for the hardware configuration.

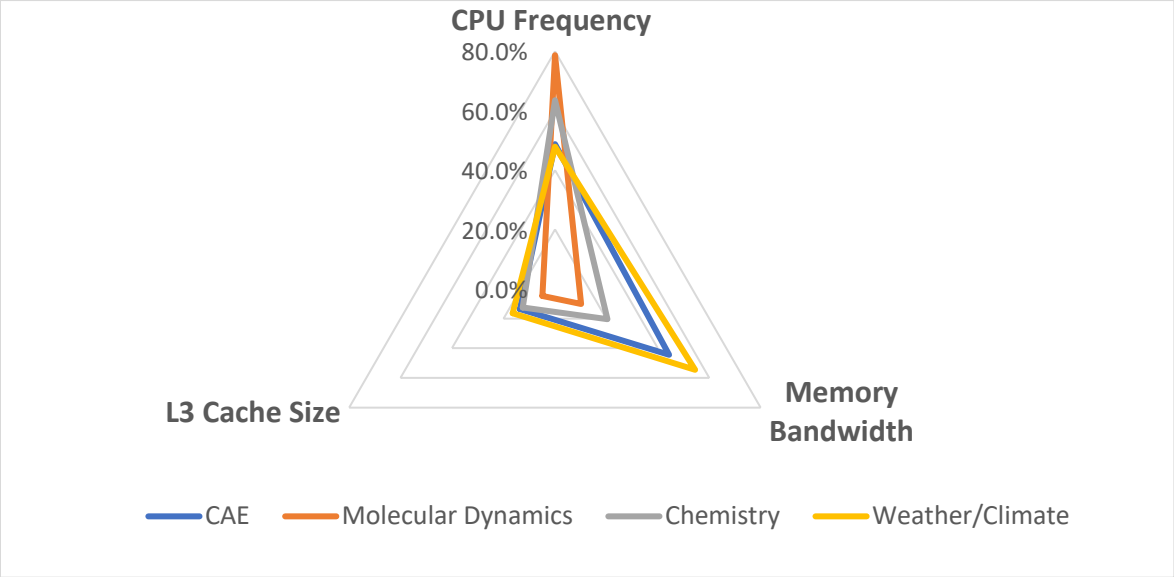


Figure 5 Radar plot for the dependency on Compute, Memory and Last Level Cache for some of the domains presented in this study

The complete list of applications and workloads included in this study are presented in Appendix A1 broken down by HPC segment. We ran each code to get a performance result that was then normalised to the performance measured with RDIMMs and fully populated memory channels. This gave us the relative performance for each of the workloads. These numbers were then averaged across the domain to give us average performance per domain, $P(Domain, Memory)$. Each domain has a particular extent that it is Compute bound or memory bound, as shown in Figure 5, allowing us to label domains as Memory Bound, Compute Bound or Balanced.

For pricing, we looked at the total cost for a rack containing three ThinkSystem N1380 Neptune cooled enclosures, each fully populated with 16 ThinkSystem SC750 V4 servers. We took the price of this setup when fully populated with MRDIMMs as our baseline and then normalised the price of the setup with MRDIMMs and with underpopulated memory channels to this baseline. We define $C(\mathrm{Memory}, \mathrm{Month})$ to be this price for a given month.

In the subsections which follow, we define price-performance for a given memory configuration on a given month to be

$$\frac{P(Domain, Memory)}{C(Memory, Month)}$$

Therefore values greater than 1 are for results with higher than baseline (12CH RDIMMs) performance with a cost that doesn't grow as much as the performance uplift, or for where the performance results are lower than the baseline, but the cost of the solution dropped significantly enough as to make this option desirable as a solution.

3.2 Performance Study by Sector

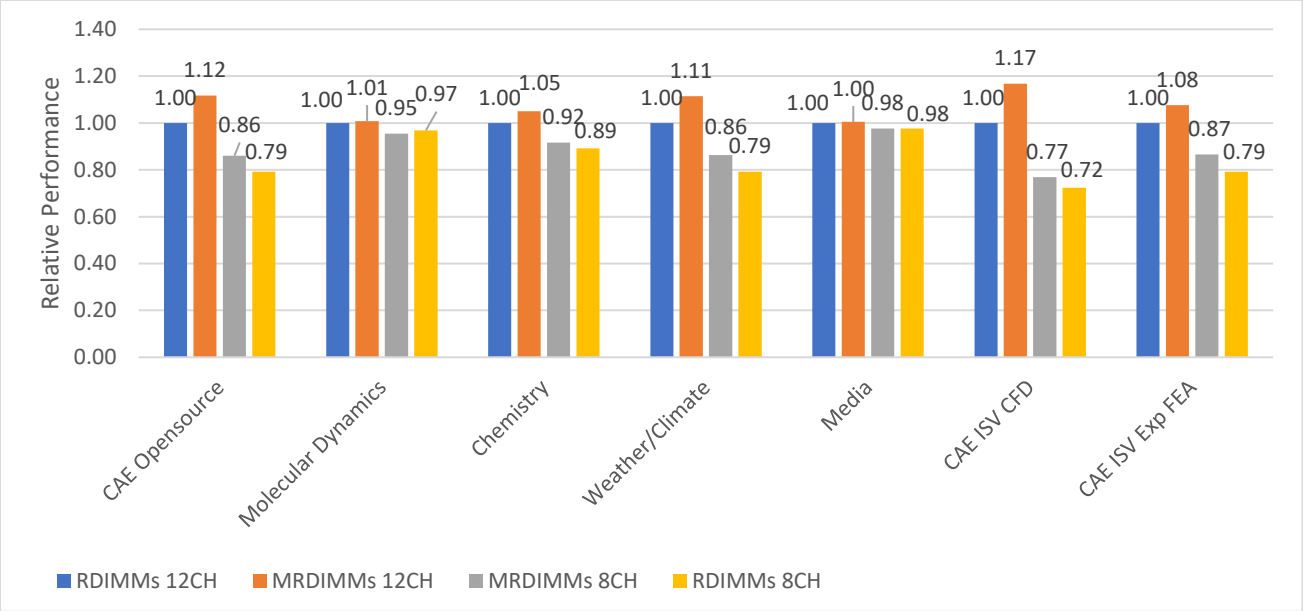


Figure 6 Relative performance of averaged workloads from each sector, normalised to the performance of the 12ch RDIMM memory configuration

In Figure 6 we showcase the relative performance of the average workload from each HPC sector on SC750 V4 normalised to the average workload performance of using a fully populated RDIMM memory configuration. From this we can make broad statements about the performance characteristics of codes in each sector. For the price to performance study we grouped together domains into three broad categories: Memory Bound workloads, Compute Bound workloads and Balanced workloads. This was done so that developers and users of HPC applications not covered in this study can gain some insights into the price-performance comparison for their workload by knowing how that workload is bound.

From Figure 6 we see that the molecular dynamics workloads and the media workloads show little difference between the four configurations, which we interpret as those workloads being mostly compute bound. In contrast, we see significant variation in the CAE workloads as well as the weather/climate workloads which we interpret as being mostly memory bound. Since the chemistry workloads are between these two extremes, we classify them as being "balanced" for the purposes of this study.

3.3 Price to Performance Study by Hardware Bounds

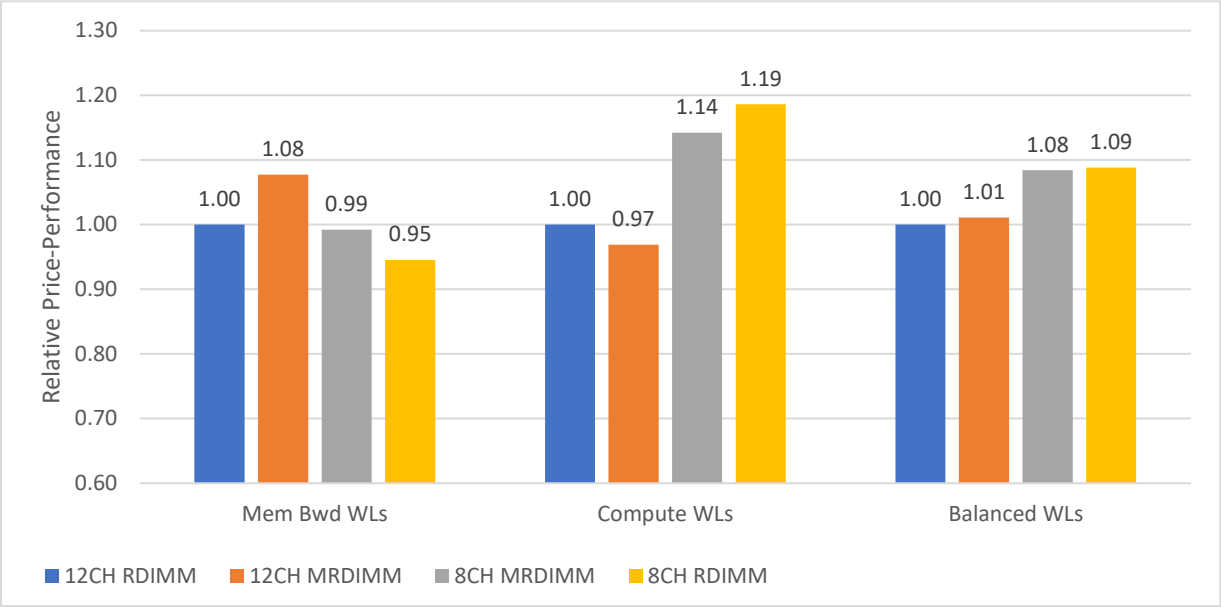


Figure 7 Relative Price-Performance for different memory options on ThinkSystem SC750 V4 compared to fully populated RDIMMs in June 2026

In Figure 7 we present the price to performance ratio (price-performance) for each of the three categories normalised to the RDIMM configuration with fully populated memory channels. Comparing the first two bars in each cluster allows us to compare MRDIMMs vs RDIMMs for a fully populated configuration. Here we see that for memory bound workloads, the uplift from the increased memory bandwidth more than compensates for the difference in cost between RDIMMs and MRDIMMs, making the MRDIMMs a good option for such workloads. For compute bound workloads where we don't expect a significant performance uplift when using MRDIMMs over RDIMMs, we see that the increased cost makes the MRDIMM solution a less cost-effective option than just using RDIMMs. For balanced workloads the cost effectiveness of both solutions is very similar with the slight edge going to MRDIMMs.

Comparing first two bars in each cluster versus the second two bars allows us to compare fully populated vs underpopulated memory configurations. For memory bound workloads we see that the 8ch RDIMM option has the lowest cost effectiveness of the four options since the performance of these workloads drop significantly when going from 12 to 8 channels and the benefit from the price decrease is not enough to compensate. For 8ch MRDIMM though the cost effectiveness of the solution is almost identical to using 12ch of RDIMMs, making it a good potential option instead of buying 12 channels of RDIMMs per server which would mean ordering more components in a hardware constrained market. For the compute bound workloads, since the performance does not change considerably when going from 12 to 8 memory channels used, the uplift shown here is purely due to the 1/3 reduced memory cost in underpopulating of DIMMs. For compute bound workloads we see that the 8ch RDIMM option gives the most cost-effective solution, being 19% better than the fully populated solution. For the balanced workloads, we see that the two most cost-effective solutions are the ones where only 8 memory channels have been populated, since the high price of memory means that the modest performance difference between 12 and 8 memory channels is more than compensated for from the difference in the price of the solution.

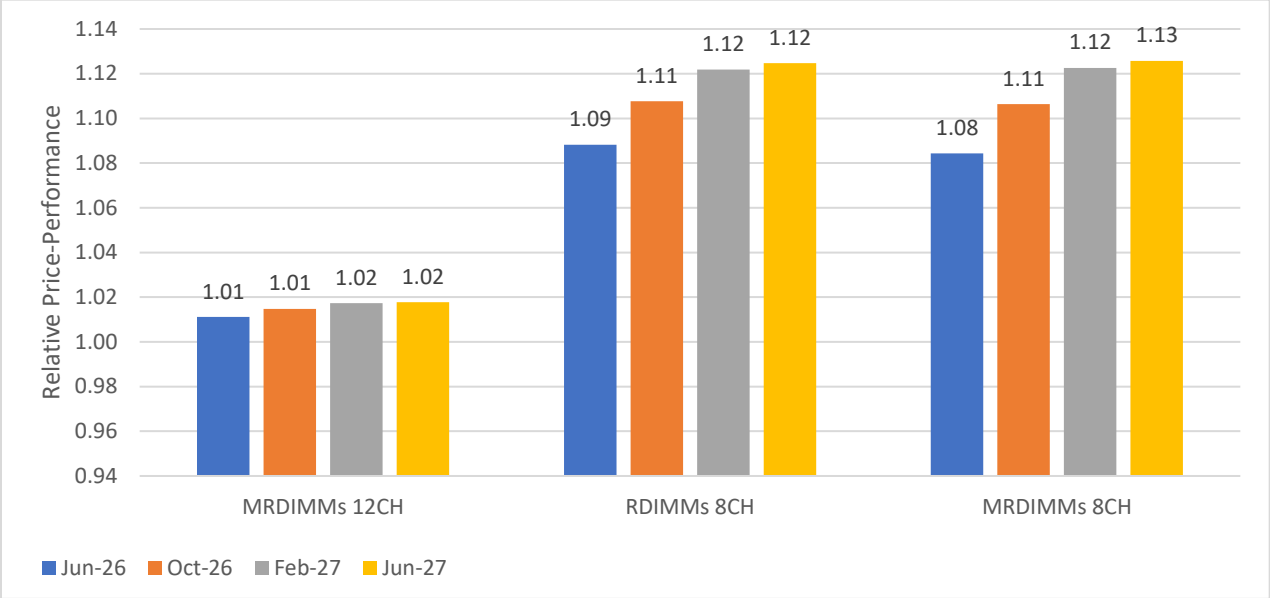


Figure 8 Projections on the price-performance effectiveness of Balanced Workloads in the next year

The cost effectiveness of the underpopulated options is also projected to increase over the next 12 months (not accounting for changes to supply of memory from manufacturers). In Figure 8 we look at the projected price-performance cost effectiveness of each of the memory configurations when compared to the 12 CH RDIMM configuration on the same date. We see from the first cluster that the fully populated MRDIMM solution is projected to remain very similar effectiveness to the fully populated RDIMM solution, increasing effectiveness only about a percent in 12 months. For the underpopulated configurations though, the cost effectiveness of the solution is projected to rise several percent in the next year making them the most cost-effective options to consider for future HPC deployments.

As the production of the current generation of MRDIMM is planned to wind down soon in preparation for the next generation of memory. For a mixed workload cluster we recommend the 8ch RDIMM configuration be reviewed as a potential option for new deployments as the most cost-effective price to performance solution.

4 Impact of Underpopulated Memory Channels on AI Training and Inference Workloads

In this section we focus on the ways in which underpopulating CPU memory channels could be used to save money on a GPU rich system for AI workloads. For GPU rich systems, the cost of the GPU units and network equipment required to keep them supplied with data and enable large-scale inter-GPU communication typically dominates overall system cost. As a result, reducing the memory population by a small number of DIMMs is unlikely to have a significant impact on the total cost of an AI deployment. However, there are three good reasons to look into this: underpopulation due to limited availability / reduced lead times for when memory is the bottleneck, saving money on the smallest AI deployments with only a few GPUs designed as a test bed for planning larger deployments, and to identify where memory resources could be repurposed to increase overall memory capacity. For the results presented in this section we made use of a Lenovo ThinkSystem SR675 V3 [9] server containing eight NVIDIA RTX 6000 Pro Blackwell Server Edition GPUs to test workloads that may be of interest to current and future AI deployments. We split the setup and results section according to training and inferencing workloads since the former is mostly straightforward whilst the latter is a much wider topic depending on the size of your deployment and your tiered storage for caching.

4.1 Summary of codes and Methodology

The server used for this study was a Lenovo ThinkSystem SR675 V3 configured with the following:

- Two AMD 9634 84C processors in a dual socket configuration
- 24 units of 128GB Lenovo ThinkSystem DDR5 Memory DIMMs operating at 4800 MT/s
- Eight NVIDIA RTX Pro 6000 Blackwell Server Edition GPUs
- Five Lenovo ThinkSystem 7.68TB NVMe SSDs in a RAID5 setup to provide a local XFS filesystem
- Four NVIDIA ConnectX-7 NDR200 2-port PCIe Gen5 adapters
- Ubuntu 24.04.4 LTS (Noble Numbat)
- Cuda 13.2 / NVIDIA open driver 595.84

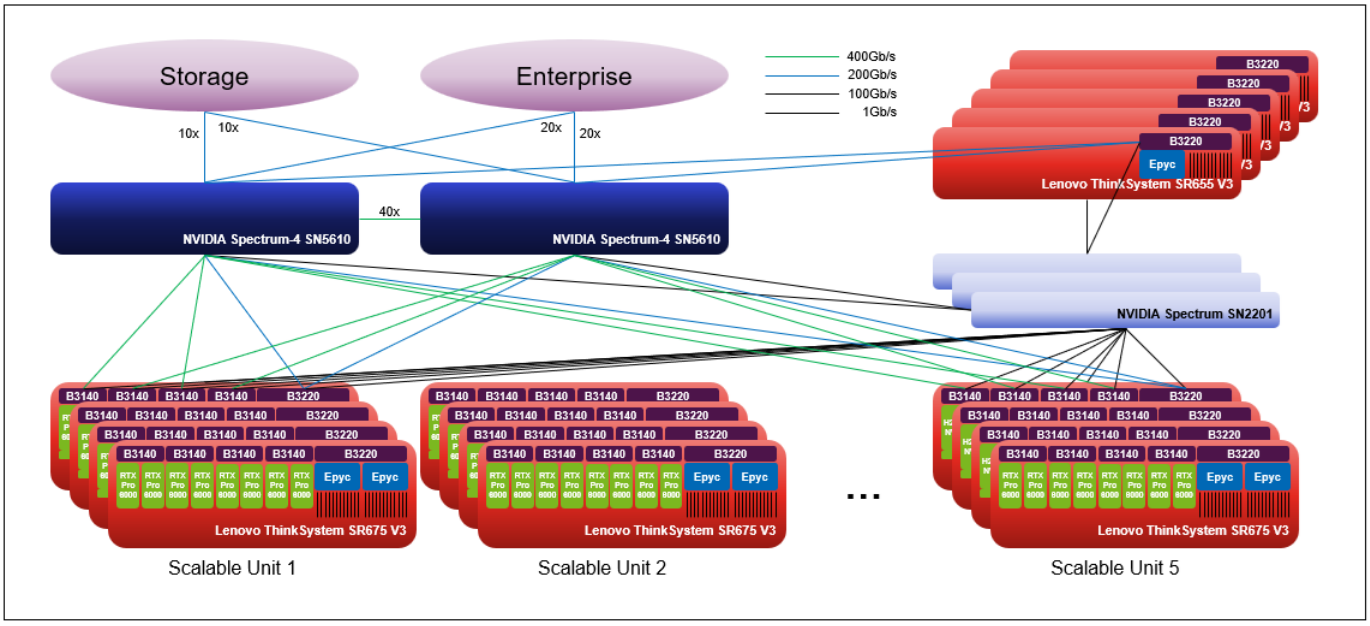


Figure 9 Hardware components making up the Lenovo Hybrid AI 285 platform with up to five scalable units.

Reproduced from LenovoPress [10].

Whilst this study is restricted to a single node, we include the network adapters in the above configuration since the node is set up as part of a scalable unit in the Lenovo Hybrid AI 285 platform [10] designed as a complete solution for rapid deployment of a Hybrid AI factory. The RTX6000 Pro Blackwell GPU variant of the 285 platform is chosen for "cost-effective, high-density inferencing, 'Agentic AI' workloads, and graphics-heavy generative tasks". The full 285 scalable unit contains four GPU rich servers each with eight GPUs installed for a total of 32 GPUs, but the 285 can be extended to 5 scalable units for up to 160 GPUs in one complete solution as shown in Figure 9.

Total DIMMs	Processor 1											
	12	11	10	9	8	7	6	5	4	3	2	1
2							6					
4						7	6					
8				9		7	6		4			
12				9	8	7	6	5	4			
16		11		9	8	7	6	5	4		2	
20		11	10	9	8	7	6	5	4	3	2	
24	12	11	10	9	8	7	6	5	4	3	2	1
Total DIMMs	**Processor 2**											
	24	23	22	21	20	19	18	17	16	15	14	13
2							18					
4						19	18					
8				21		19	18		16			
12				21	20	19	18	17	16			
16		23		21	20	19	18	17	16		14	
20		23	22	21	20	19	18	17	16	15	14	
24	24	23	22	21	20	19	18	17	16	15	14	13

Figure 10 Valid memory configuration options for the Lenovo ThinkSystem SR675 v3 server. Reproduced from table 26 of the SR675 V3 User Guide [11].

For the testing procedure, we started with baseline measurements from a server with all memory channels populated and then proceeded to remove 4 and then 8 DIMMs, which lets us go to the next most populated valid memory configurations for the SR675 v3 server as shown in Figure 10. Once the memory was removed, we confirmed from the XCC that the memory configuration was accepted, booted the OS and then ran the same tests again to compare with the baseline measurements.

4.2 MLPerf AI Training

For the training benchmarks in the testing bundle, we ran a modified version of the scripts which are available publicly for the MLPerf training v5.1 submissions. MLPerf is a collection of several benchmarks published by MLCommons [12] aiming to provide useful and relevant measures of performance for AI and ML solutions. For AI training, the MLPerf Training group periodically releases reference implementations for a few training models and then collects together

submissions from vendors from using their architecture tuned implementations to create a list comparing each solution. The training results and vendor optimised implementations can be found on GitHub and used to reproduce the result on other systems. For this study we looked at two different training models which were part of the MLPerf Training 5.1 submissions[3]:

- Llama2-70B: Based on NVIDIA NeMo LLama2-70B LoRA PyTorch MLPerf Benchmark
- Llama3.1-8B: Based on NVIDIA Large Language Model Llama 3.1 8B PyTorch MLPerf Benchmark

These benchmarks were prepared according to the README included in the model subdirectory of the Lenovo's submissions[3]. Since these benchmarks were designed to run on older hardware, some modifications were needed in order to run on the RTX6000 GPUs. These changes are outlined in the subsection which follows.

4.2.1 Required Changes to MLPerf Training setup

For Llama3.1-8B:

- The Dockerfile for the container image was edited to
  - Increase the Transformer Engine version from v2.8 to v2.12
  - Use the default commits for cudnn-frontend, googletest and cutlass submodules
  - Change the NVTE Cuda architectures to also include 120a
  - Rebuild the custom_embedding library included in the repo to account for the above changes
- The config scripts were edited to
  - Reduce the Micro Batch Size from 2 to 1 to account for less GPU memory on the RTX6000 vs the B300 GPU that the scripts were provided for
  - Increase the walltime from 140 to 1200 to account for the increased runtime versus the B300 GPU setup
  - Disable the FP8_DPA option to disable FP8 Dot Product Attention as there were compatibility issues with the RTX6000 GPUs.

For Llama2-70B:

- The Dockerfile for the container image was edited to
  - Change the NVTE Cuda architectures to also include 120a
- The config scripts were edited to
  - Increase the Tensor Parallelism from 1 to 8 to use all 8 GPUs
  - Increase the maximum steps from 800 to 12000 and walltime from 25 to 36000 to account for the increased runtime versus the B300 GPU setup
  - Disable the FP8_DPA option to disable FP8 Dot Product Attention as there were compatibility issues with the RTX6000 GPUs.

[3] https://github.com/mlcommons/training_results_v5.1/tree/main/Lenovo/benchmarks

The benchmarks were then submitted[4] with the commands:

```
source config_SR780aV3_SR680aV3_SR680aV4.sh
sbatch -N 1 -t 3-23:00:00 run.sub
```

4.2.2 Training Results

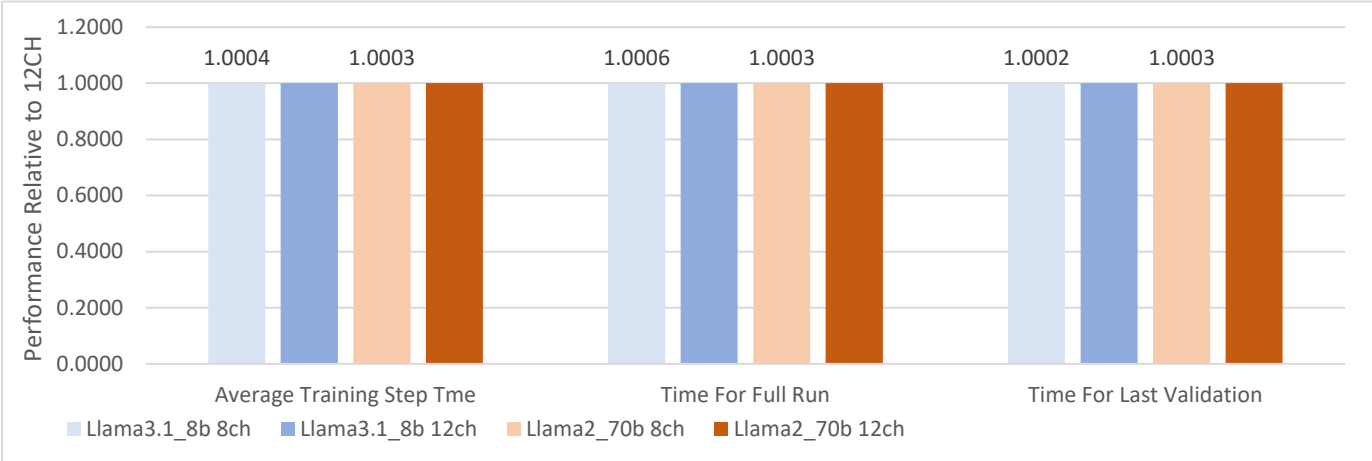
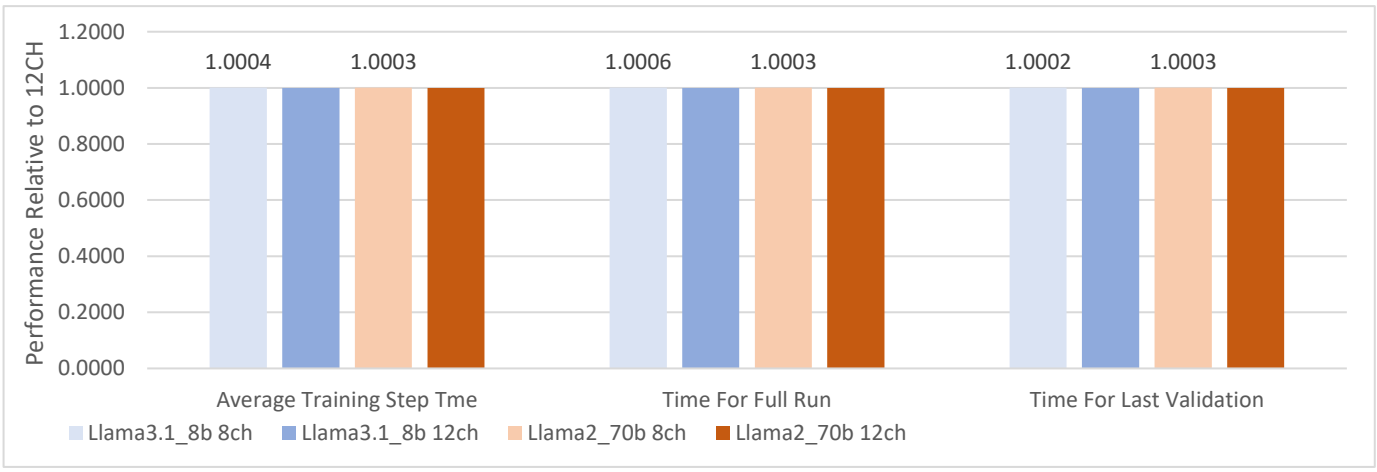


Figure 11 Performance difference between using 8 and 12 memory channels on the host CPUs for MLPerf Training with Llama models. Results for each model are normalised to the 12 channel result.



In Figure 11 we present three metrics for the training of the two Llama models with 12 and 8 memory channels on the host populated: Average training step time, time for the full run using the “run_start” and “run_stop” timestamps from the MLPerf logs, and the time it took for the last validation step when checking if the error was below the required threshold. For all three of these metrics we noticed a sub 0.1% variation between the setups with 12 and 8 memory channels indicating that these workloads are not sensitive to the host memory bandwidth. This is consistent with our understanding of the training workflow as after the training data is loaded onto the GPU(s) the GPU performs the training steps and validation tests without moving data back and forth to the host. For MLPerf training specifically though, we do little to no

[4] Slurm + enroot + pyxis was installed on the host since that was required to reuse these scripts. The slurm configuration was minimal and only included localhost in the list of servers.

checkpointing of the trained model which would have involved copying data back to the host and then to a filesystem on the host. If we were doing training which involved heavy amounts of time spent checkpointing then we would expect more of a dependence on the host memory bandwidth. For training with low to no checkpointing though (such as what is done for an MLPerf submission) we conclude that underpopulating host memory could be a viable option with minimal performance penalty.

4.3 vLLM Inferencing

For testing inferencing throughput on the test server with underpopulated memory, we deployed a setup with the popular free open-source tool vLLM [13]. We chose to look at a couple of models with a different number of parameters to see if there were any differences between the two. For a model which

- fit on one GPU, Llama3.1-8B was used
- fit only on multiple GPUs, Llama3.3-70B was used

Both models were downloaded from Hugging Face[5,6] using the hf command line tool [14]. vLLM was deployed using the vllm/vllm-openai:v0.28.0 container, which includes fixes required to run with the RTX6000 GPUs. A common deployment for vLLM on a system is to use the container image and the `vllm serve` command which runs as an HTTP server on the node allowing end users to submit requests using one of the many compatible APIs and their frontend of choice. For this study we used the `vllm bench throughput` command designed as a drop-in replacement for vllm serve, allowing us to use standard vllm serve options whilst running an offline throughput test with many concurrent requests being served at the same time.

The most significant task offloaded to host memory on the system was anticipated to be storage of the Key-Value (KV) cache. In inference workloads, KV caching speeds up response generation by storing previously computed data so that it can be reused when generating subsequent tokens, reducing the amount of computation required. This is often done with either long conversations where the same inputs are analysed repeatedly or when multiple requests share a large amount of the input tokens to avoid repeated work. KV caching is often implemented in a tiered approach, making use of all the available memory in the solution including GPU memory, adjacent GPU memory, host memory, adjacent host memory or with an external storage server. The optimal setup will depend on the use case of the end-users, if they foresee using repeated tokens across the user group either through long conversations or repeated inputs across users. The full KV cache tiered solution will often be provided by various software and hardware components to try and optimize the movement of KV pairs for when they are needed.

4.3.1 GPU Based KV Caching

To start with, we study the case where the KV cache pairs are all contained in GPU memory so that host RAM is not used as a KV store but instead only for the limited other tasks that need to be run on the host for this workflow, including things like logging, scheduling, control flow from the Python libraries and reading tokenised input data.

[5] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[6] https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct

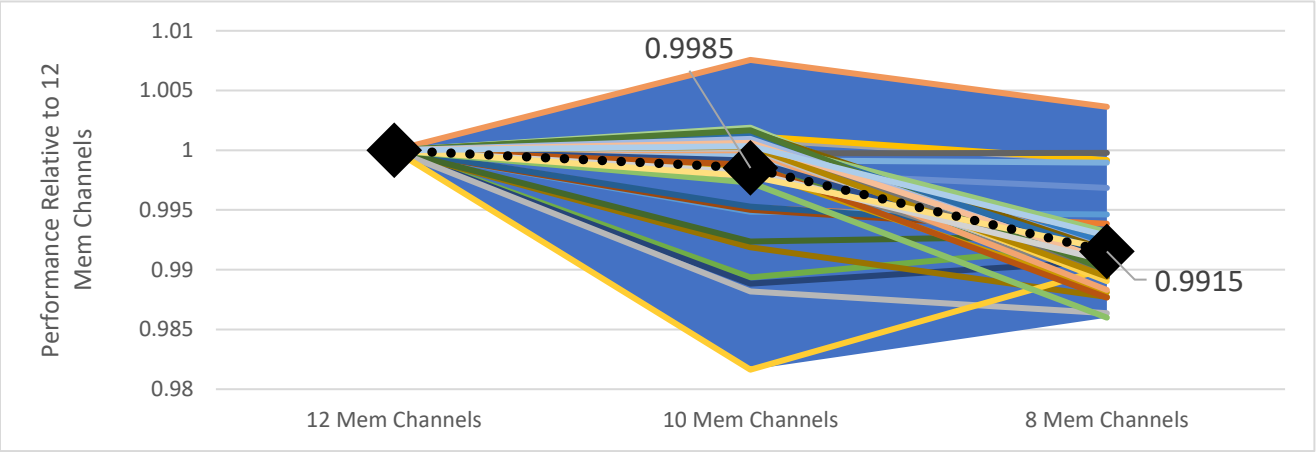


Figure 12 Total token throughput relative to the 12 Memory Channel throughput for vLLM with GPU only KV Cache

For Figure 12 we ran vLLM with a number of different options to account for users with different access profiles interacting with the vLLM server. We varied (excluding combinations which didn't fit on the GPU memory):

- GPU memory utilization (--gpu-memory-utilization) in {0.50,0.70,0.85,0.92}
- Model (--model) between Meta-Llama-3.1-8B-Instruct and Meta-Llama-3.3-70B-Instruct.
- Tensor Parallelism (--tensor parallel-size) in {1,8}
- Context Lengths (Input Length --input-len / Output Length --output-len / Number of Prompts --num-prompts) in:
    - 1024 / 128 /1000
    - 4096 / 512 / 1000
    - 16384 / 512 / 512
    - 32768 / 512 / 256
    - 65536 / 512 / 64

To compare the differences between 12, 10 and 8 memory channels we measured the total throughput reported in tokens/s and divided by the throughput result for all 12 memory channels populated to get the relative throughput in each of the combinations of the options outlined above. In Figure 12 we show the differences between 12, 10 and 8 memory channels using a different line for each test case. We also add in a shaded band showing the range of results for each memory config as well as a black dotted line showing the average relative performance for each memory config. From this we see that overall the difference in the results between the memory configs is at most 2%, which is very small. For each individual setup, the performance order between the 3 memory configurations varies. In some cases, the 8channel throughput is lower than the 12 channel throughput, while in others, the opposite is observed. On average though, we lose about 1% throughput going from 12 memory channels to 8 memory channels.

4.3.2 Host Based KV Caching

For memory bandwidth dependency, it is necessary to look at the case where the KV Cache is extended to include the host memory. There are various options for doing this depending on the expected reach of the KV cache. For the fastest memory options, a tiered approach would likely be optimal with the KV cache using GPU memory first, then overflowing to host memory / external storage systems. For a production solution this can be done with vLLM's native offloading feature, but in this case though, since we want to stress the memory on the host, we instead move the entire KV cache to the host memory. This also allows us to make sure that the memory on the host designated for the KV cache is actually used in the inferencing requests and not just allocated during the initialisation phase and then blocked off but not touched for the

actual workload.

For an external KV cache solution we deployed LMCache [15] on the server, which is a tool to manage cache using system memory and local storage that can be interacted with via an API and a HTTP endpoint. The vLLM container comes packaged with a compatible version of LMCache or it can be installed manually via pip. To setup a LMCache server we ran the following:

```
lmcache server --host localhost --port 5555 --l1-size-gb 512 --max-workers 8 --eviction-policy LRU
```

With LMCache we allocated 512 GB of memory on the host for the KV cache and then set the max workers to 8 to match the 8 GPUs that would be putting requests into the server. The “Least Recently Used” policy was chosen as the default eviction policy for freeing up memory when limits are reached. With vLLM we then use the following options to use the LMCache server on the default port number:

```
--kv-offloading-backend lmcache
--disable-hybrid-kv-cache-manager
--max-num-seqs 16
```

One interesting issue we saw when running throughput studies with vLLM and LMCache is a form of “Cache Thrashing” that causes severe slowdown / locking in our workflow. As LMCache and the vLLM throughput test was running, and writing data to the KV Cache, we would suddenly encounter a warning from vLLM about looping pre-empt requests and the vLLM benchmark progress wouldn’t update. The reason we believe this happened is that we would reach a deadlock state where KV tokens on the GPU(s) were being constantly populated and then evicted to make room in memory for other KV tokens. This meant that KV tokens were being constantly moved around without giving the benchmark time to make reasonable progress on the requests. In a real deployed system with vLLM acting as an inferencing server to many clients at once, we could just hold back one of the client requests until the deadlock cleared, but with doing a throughput benchmark (that isn’t just using GPU memory for caching) this option is not available to us. Other inferencing tools such as NVIDIA Dynamo include a KV Block Manager [16] for dealing with Blocks across the KV cache hierarchy but for our simple setup we mitigated the chance of this happening by setting the maximum number of concurrent inference requests to 16 (2 per GPU).

		Total Tokens Per Second			Output Tokens Per Second		
Input Length (B)	**Num Prompts**	**12 Mem Ch**	**10 Mem Ch**	**8 Mem Ch**	**12 Mem Ch**	**10 Mem Ch**	**8 Mem Ch**
16384	512	2618.48	2600.06	2640.42	79.35	78.79	80.01
32768	128	2675.15	2684.09	2621.02	41.16	41.29	41.13
65536	32	2621.02	2611.59	2640.42	20.32	20.24	20.34

Table 2 vLLM + LMCache throughput results for different quantities of memory channels populated for different input lengths.

Table 2 vLLM + LMCache throughput results for different quantities of memory channels populated for different input lengths. We showcase the total tokens per second processed and output tokens per second when running vLLM with 512GB of KV cache on the host system. We see from these results no noticeable performance drop going from 12 memory channels populated per CPU to 8 channels populated per CPU, which could be for a number of reasons.

- Even though a lot of data is written to the KV cache, there may be minimal read back moving the KV pairs back into GPU memory. Having significant read operations for our single node setup may require using much longer input lengths to simulate longer conversations or more concurrency reducing the space available per inference thread. With our setup we are restricted by Cache Thrashing mentioned earlier, but for a real deployment with lots of concurrent users / long conversation history this is likely something that needs to be reviewed.
- vLLM and LMCache are able to handle the KV transfer requests effectively for our setup with even 8 memory channels on the host populated, moving KV pairs between GPU and host memory before the start of any execution of any inferencing thread in a time that was negligible compared to the time spent generating the response.

For this setup we conclude that underpopulating memory channels has minimal effect on the throughput performance of the inferencing server, either using GPU memory only or also using host memory for KV caching. We suspect however that memory bandwidth will become more important when scaling out to more concurrent users and more GPUs using the shared cache as this will entail more significant read back of the KV cache blocking the start of the inferencing thread unlike with writing the KV cache which can be done asynchronously. Using other tools on the market that alleviate blocking when moving around KV pairs may also allow for higher concurrency on a single node than we were able to test here. For a single node, however, we show that the 285 solution has minimal host memory usage at low concurrency and for conversation lengths shorter than 64 kB.

5 Lenovo DSS-G Storage Solution with reduced memory

In this section we use Lenovo DSS-G as a real example to showcase what can happen with enterprise-level storage solutions as a result of underpopulating memory channels on the storage controllers.

5.1 Summary of hardware

Lenovo ThinkSystem DSS-G [5] is a complete storage solution offered by Lenovo for datacentres combining ThinkSystem Servers, high capacity storage JBODs and IBM Storage Scale software (GPFS) [17]. It is designed to be completely scalable in order to offer both high capacity and high throughput to sustain IO traffic for every level of HPC cluster.

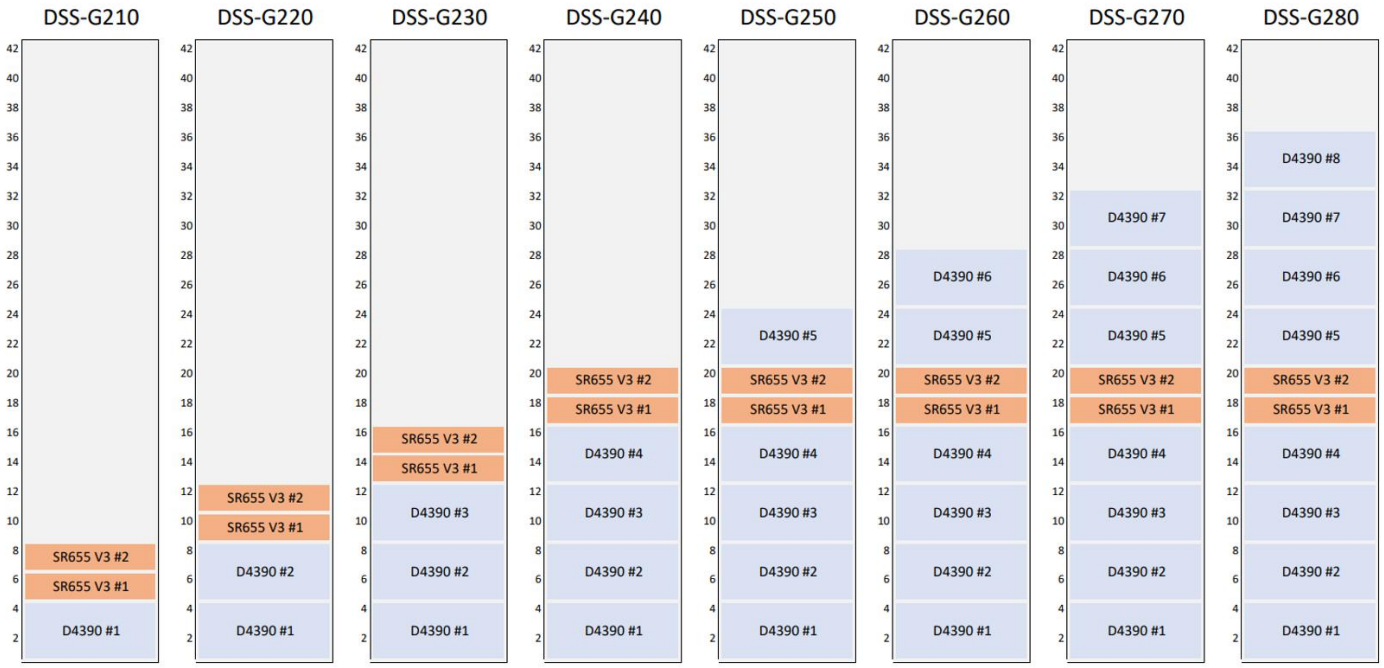


Figure 13 Overview of the supported configurations with Lenovo DSS-G Storage Solution

Figure 13 shows the supported configuration for a single rack of the current generation of the DSS-G solution. For larger HPC clusters, this single rack can be used as a scalable unit to scale up to the IO needs of the cluster. For each rack/building block, DSS-G comprises of

- Two Lenovo ThinkSystem SR655 V3 single socket servers [18] each powered by AMD EPYC Zen 4 or Zen 5 processors with 12 total memory channels per server.
- Between one and eight Lenovo ThinkSystem D4390 high density HDD storage arrays [19] each connected to the storage controllers with four 24Gbps Mini-SAS connections for data movement.

Network adapters on the storage control nodes provide access to the GPFS cluster and filesystems for the rest of the HPC cluster via NDR/NDR200/HDR InfiniBand networking[7].

In the Lenovo HPC Benchmarking Cluster, a DSS-G280 was setup with NDR200 connections to the rest of the cluster

[7] For a list of available network options see https://lenovopress.lenovo.com/lp1842-lenovo-dss-g-thinksystem-v3#table-network-adapter

with a large number of compute nodes to serve as clients to test the IO performance. The ThinkSystem SR655 V3 storage control servers support 1,2,4,6,8,10 or 12 DIMMs populated per server as valid options, so for this study we started with fully populated controllers and then physically removed memory DIMMs in order to study the effects of underpopulating memory on the performance of the created filesystem.

Note that for GPFS to function, there is a minimum memory requirement for the storage controllers based on the filesystem properties. GPFS needs to hold in memory the RAID tracks detailing how the data is distributed across the HDD enclosures and underlying drives in a setup with RAID arrays configured. The smaller the blocksize used in the filesystem, the more RAID tracks that need to be recorded, leading to more memory needed. The exact minimum memory needed can be calculated from the filesystem blocksize, making sure to include a factor of two to allow for each storage controller failing and all of the filesystem RAID data being able to be hosted on a single controller.

Drive	File System Block Size				
Size	1M	2M	4M	8M	16M
12T	384	384	384	384	384
14T	768	384	384	384	384
16T	768	384	384	384	384
18T	768	384	384	384	384
20T	768	768	384	384	384
22T	768	768	384	384	384
24T	768	768	384	384	384
26T	768	768	768	384	384
28T	1536	768	768	384	384
30T	1536	768	768	768	384
32T	1536	768	768	768	384
48T	1536	1536	768	768	768

Table 3 Example minimum memory requirements (in GB) when fully populating memory channels in the DSS-G240 storage controllers due to RAID track data and failover resiliency.

Table 3 shows an example of the minimum memory requirements needed per storage controller for GPFS in order to support all the RAID information as well as have failover of the recovery groups supported between the two controllers. For this data we restricted to only fully populating memory channels using 32GB, 64GB and 128GB memory DIMMs. The results here will change with drive type, drive size, number of drive enclosures used and filesystem blocksize. Regardless of whether underpopulation of memory channels is used in the deployment of the storage system, this minimum requirement for memory usage should always be taken into account when sizing the solution. Whilst this study was undertaken internally at Lenovo to investigate the effects of running a storage solution with reduced memory, Lenovo does not currently support underpopulation of memory channels in the storage controllers.

5.1.1 Testing Methodology

For each test on the GPFS filesystem, we used the standard IO benchmark tool IOR [20] to measure the performance of the sequential read and write operations and MDTest to measure random small file read and write operations. For clients we used 18 nodes/24ppn of ThinkSystem SC750 V4 with Intel GNR 6980p 128c CPUs and 1xNDR400 IB link per node. Here we build a filesystem with a 16m blocksize and then chose the IOR transfer size to match the filesystem blocksize for best performance. For IOR we used the following command

```
mpiexec ior -a POSIX -X 1655286037 -Z -Z -i5 -r -w -b100G -t16m -vvv -e -g -d 10 -F -o
${tgtdir}/ior
```

and for MDTest we used the following command:

```
mpiexec mdtest -d ${tgtdir}/mdt/ -R -i 5 -n 1000000 -F -w 3072 -e 5 -C -E -r -T -p 15 -u -P -G -
1572010289 -N 1 -Y -W 300 -x ${tgtdir}/mdt-stonewall
```

using the -w flag with 3072 for a file size that would fit inside GPFS inode and 15360 for a file size that was deliberately too big to fit inside an inode.

For each memory configuration tested, we built up recovery groups and built a filesystem using between one and eight of the drive enclosures with a selection of filesystem blocksizes. We then ran the IOR and MDTest benchmarks before removing the filesystem and the recovery groups before the next test.

5.2 Performance characteristics

For readability, we have split the results from the IOR benchmarks into three different sets based on the performance characteristics.

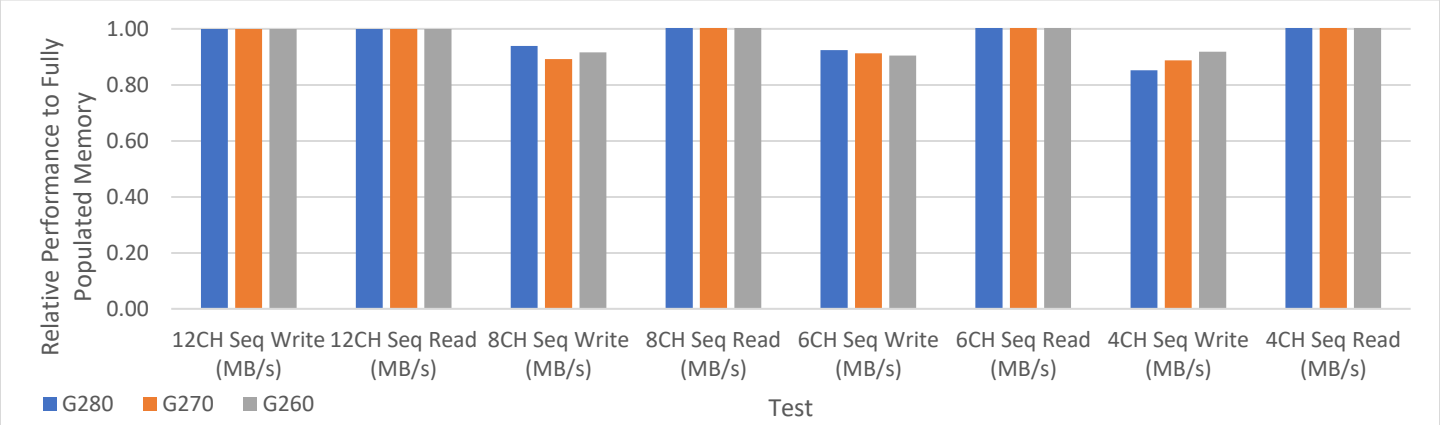


Figure 14 Sequential Read and Write Throughput measured on DSS-G270 and G280 with memory channels on the controllers depopulated.

Figure 14 contains the first set of results including the larger building blocks of the G260, G270 and G280. For these three options, the effect on performance from depopulating memory on the read performance was very different from the write performance. For the sequential read performance, the measured result was unchanged when going from 12 memory channels populated down to 4 channels populated. We suspect that this is due to the larger number of drives in the DSS-G blocks with the most number of drive enclosures, GPFS read ahead caching was able to completely mask the performance drop from the reduced memory bandwidth. For sequential write performance we see a roughly ~10% drop in write performance when you depopulate memory which continues to degrade slightly until you get to 4 memory channels used.

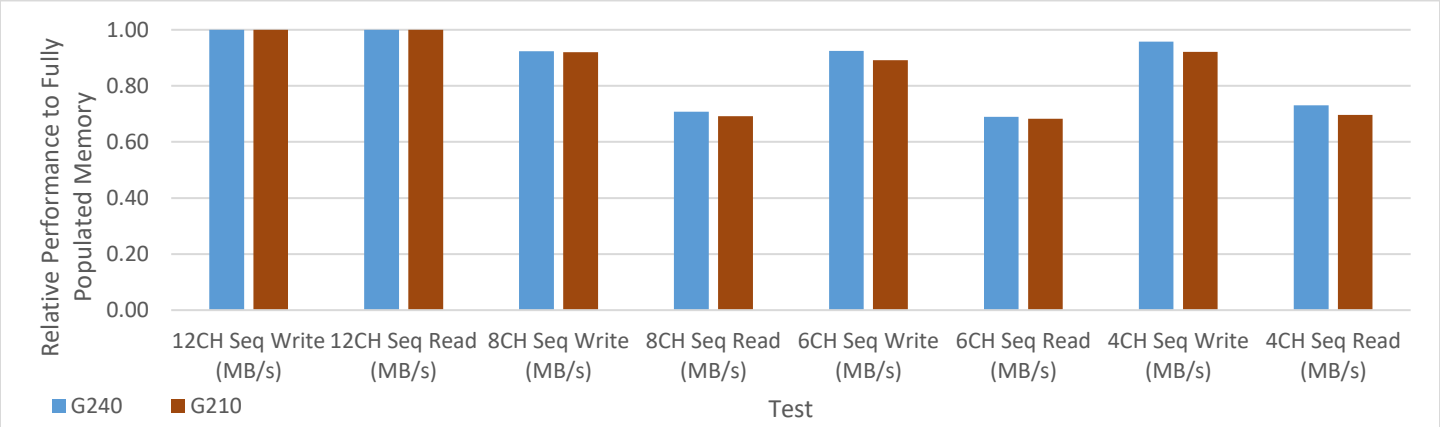


Figure 15 Sequential Read and Write Throughput measured on DSS-G210 and G240 with memory channels on the controllers depopulated.

In Figure 15 we see the results from the same tests for the DSSG configurations with fewer enclosures. For these we see a roughly ~30% drop in the sequential read performance when we start to depopulate memory channels in the storage controllers. We account for this change in behaviour by suggesting that for these configurations there are fewer drives than in the G260-G280, resulting in GPFS read-ahead not being able to hide the effect of reduced memory bandwidth. For sequential write performance we again see a ~10% drop in performance from underpopulating memory channels.

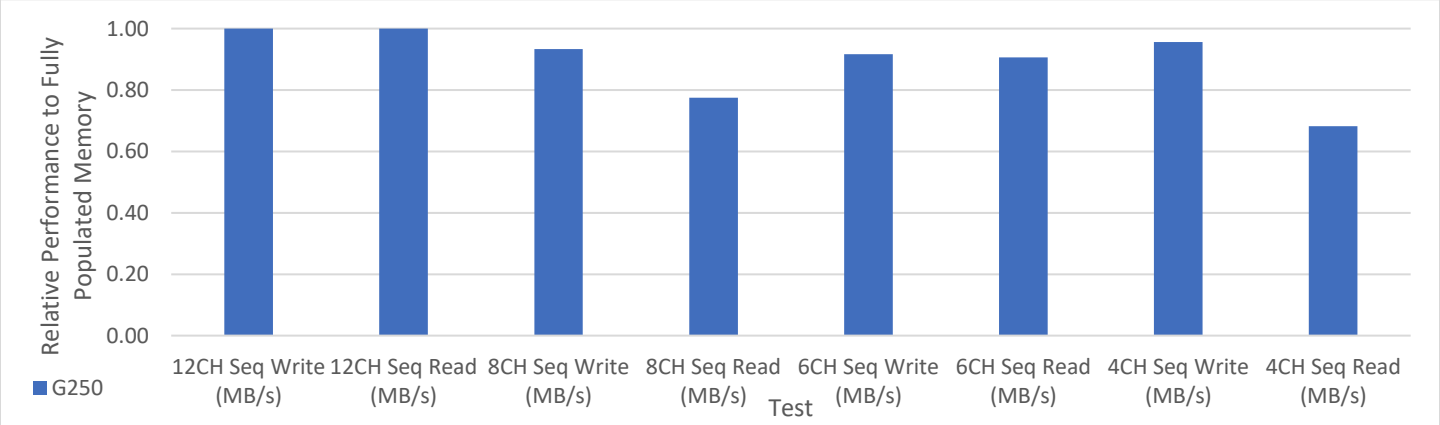


Figure 16 Sequential Read and Write Throughput measured on DSS-G250 with memory channels on the controllers depopulated.

Finally in Figure 16 we show the results from the G250 which ends up being the middle ground between the previous two groups. With five drive expansions used, we suspect that we are at the threshold number of drives where the GPFS read-ahead is able to start to offset the reduction in performance from the reduced memory bandwidth, but not enough to completely mask it as we saw for the G260-G280. The sequential write performance decreases again by ~10% consistent with the other two groups of results.

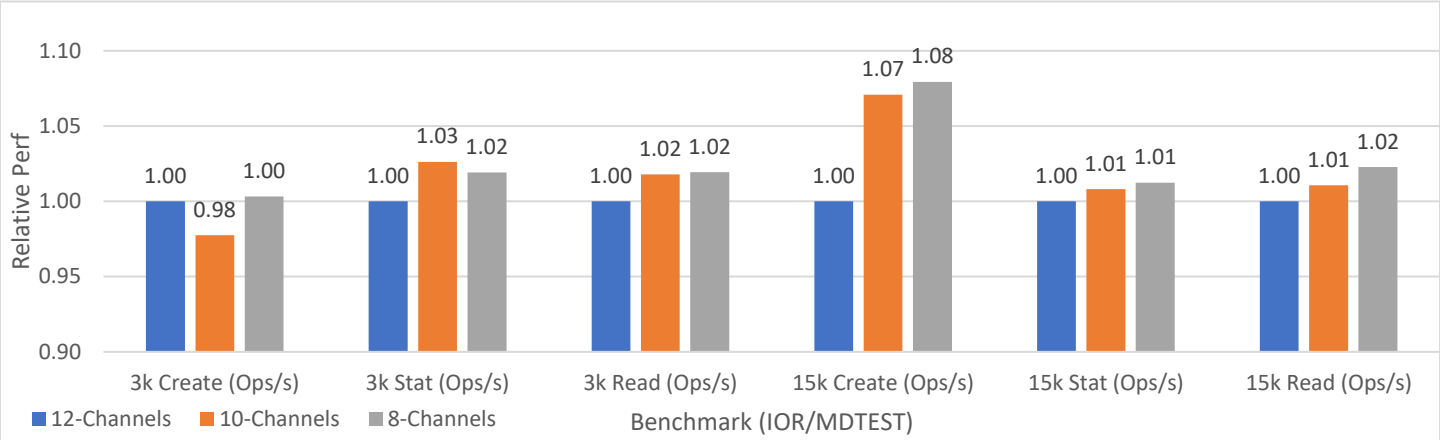


Figure 17 Random Read and Write Throughput measured on DSS-G280 with memory channels on the controllers depopulated.

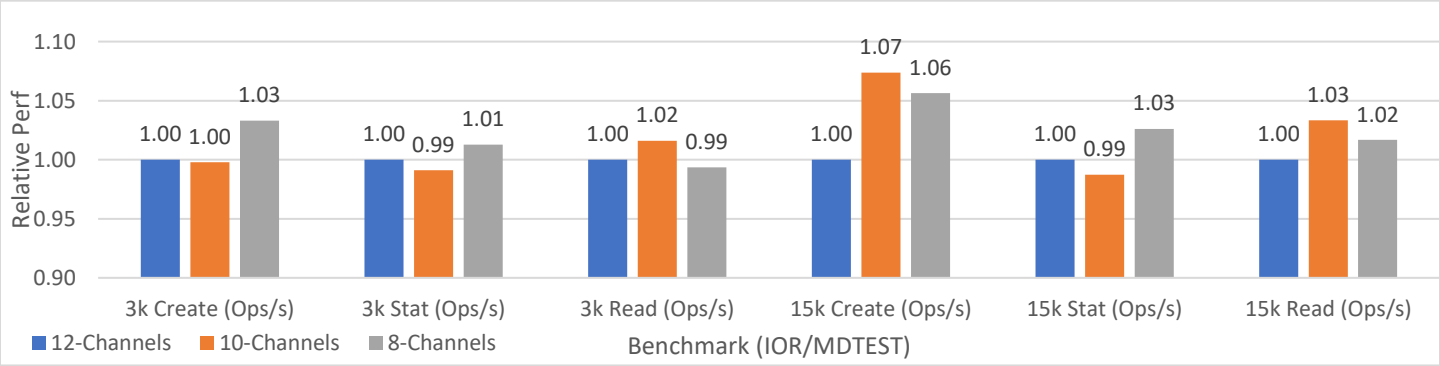


Figure 18 Random Read and Write Throughput measured on DSS-G210 with memory channels on the controllers depopulated.

In Figure 17 and Figure 18 we present the MDTest benchmark results for a G280 and a G210 respectively. Compared to the IOR benchmark results for sequential read and write, the MDTest results with a random access pattern have a lot less variability in the results, with the largest difference from the baseline measurements with fully memory populated controllers excluding the 15k create test being only 3%. For the 15k create test although we measure up to an eight percent increase in the maximum result across iterations, the average result only changes by at most 2% whilst the minimum result drops by 8% and the standard deviation increases by more than 10 fold from the 12 channel result. We suspect that this measurement is likely affected by GPFS deallocated inodes being tracked in memory being able to artificially boost performance of 15k create tasks across MDtest iterations. With most of the tests, the variation between the memory configurations was only a few percent, showing that the memory limitations on the controller are not a bottleneck on the performance of small file operations with this setup.

5.3 Price-Performance Study

In the previous subsections, we deliberately went down to fewer memory channels populated than may make sense for most production systems that aren't using 32GB memory DIMMs. For example, if you had a memory capacity requirement that required 768 GB of RAM on the controllers, using 64GB memory and depopulating to use only 6 DIMMs per server may not be as appealing as using all 12 channels and 32GB, unless there are supply constraints on the lower capacity memory. This being the case, we will assume for the discussion that follows that above 6 channels populated of 64GB memory DIMMs or 128 GB memory DIMMs, we would use all 12 channels of 32GB DIMMs or 64GB DIMMs respectively.

If we look at the current cost of a DSS-G rack (including GPFS costs, drive costs for 16TB ThinkSystem drives, enclosure costs and controller costs) and compare the savings you would get from depopulating memory, we arrive at the following conclusions:

- With a G210-G240, you can save up to 11% on the cost of the entire rack by planning for as few as 4 memory channels populated on the storage controllers. The performance cost of this is around a ~10% drop in the sequential write performance of a 16m block filesystem and around a ~30% drop in the filesystem sequential read performance.
- With a G280, you can save up to 3% on the cost of the entire rack by planning for as few as 4 memory channels populated on the storage controllers. The performance cost of this is around a ~10% drop in the sequential write performance of a 16m block filesystem but from our testing there may still be no noticeable drop in read performance.

Exact cost benefit and performance differences will depend strongly on the exact details of the setup used including the drive manufacturer/capacity and the properties of the underlying GPFS file system blocksize.

We showcase this as a real example of how depopulating memory sometimes can affect the IO performance significantly and sometimes not at all. In the case of the DSS-G specifically we make the following recommendations for Lenovo sellers and partners selling Lenovo solutions:

- For a G210-G240: whilst the cost saving can be significant from underpopulating memory channels in the controller servers, there is a significant performance penalty to the solution that is greater than what would be equivalent from the cost savings. If there are no performance requirements for a storage solution and the sizing is based purely on capacity, then underpopulating memory may be a viable option, but care must be taken to ensure the customer is happy with the degraded performance versus what is advertised and that they still meet the minimum memory needed for recovery groups.
- For a G250-G280, whilst the performance penalty is significantly less of an issue versus the G210-G240, the increased total cost of the system means that underpopulating memory will likely only need to provide a negligible discount on the price of the total solution. The performance penalty here being less than for the G210-G240 is somewhat counter intuitive, which highlights the need for testing of your actual workload when considering underpopulating memory channels.

The recommendations above may change over time depending on changes to market pricing. Underpopulating memory channels is currently neither supported or recommended by Lenovo on the ThinkSystem DSS-G solutions.

6 Conclusions and Recommendations

In this paper we have demonstrated several opportunities to reduce infrastructure costs across HPC, AI or storage environments by only provisioning memory capacity based on actual workload requirements. This is not a complete study of every HPC/AI/storage solution offered by Lenovo or every possible customer workload that could utilise these solutions, but we have provided some examples in key areas of interest of where provisioning for fewer than full memory DIMMs in a server can make sense from a cost to performance perspective and where fully populating memory is still likely the best solution.

In the HPC space where we look at CPU only servers without any GPU acceleration, we presented price-performance data for different workloads to show that underpopulating memory **does** make financial sense for some workloads depending on where performance bottlenecks exist. Our recommendations and takeaways are as follows:

1. For **Memory Bound** applications, using MRDIMMs can provide better price to performance ratio than using RDIMMs. For **Compute Bound** applications, the extra cost in MRDIMMs is not worth it over RDIMMs.
2. For **Compute Bound** applications, providing a cluster with underpopulated memory channels can lead to significant price savings with minimal performance degradation.
3. For provisioning HPC clusters with **Mixed Workloads**, the best possible price to performance compromise can sometimes be found by underpopulating memory channels. For the SC750 V4 configuration we studied, we found an 8-9% price/performance improvement using only 8 out of 12 memory channels compared to a solution with fully populated memory channels, and we project this improvement to increase over the next year.

In the AI space with GPU accelerated computing the cost saving from underpopulating memory is a lot less impactful due to the high cost of the GPUs being factored into the system price. However, in a supply constrained market it is still important to see where we can afford to reduce memory bandwidth without compromising on training and inferencing performance since we could end up with shorter hardware lead times or can provision the memory on the host system for other uses. In particular, if the latency of KV cache data between host and GPU is not a significant bottleneck in the performance of the setup, then we can use memory compression to significantly increase the capacity of our KV cache on the host allowing for more KV pairs to be reused in our conversations. For AI-based solutions, our recommendations and takeaways are as follows:

4. For small scale **AI Training** with infrequent or no checkpointing, since the workload is entirely GPU based other than scheduling and communication organisation, underpopulating memory channels can end up having little to no effect on the training time.
5. With larger scale **AI Training** with frequent checkpointing, we expect that memory bandwidth will play a much more significant role in total training time.
6. For small scale **AI Inferencing** with low concurrency and conversation lengths that aren’t long so that KV cache can fit effectively on GPU memory, underpopulating memory channels on the host has minimal effect on inferencing throughput.
7. For larger scale **AI Inferencing** with high concurrency, long conversation length and a tiered KV cache across GPU / Host and potentially additional storage blocks, we expect full memory bandwidth will be required for best total aggregate inferencing throughput.

8. Overall for **small single node deployments**, either for a smaller deployment or for a test bench to get familiar with the Lenovo solutions in preparation for a larger scale deployment, saving money by reducing memory channels populated on the host server could be an appealing option for a customer to consider.

Finally, we presented data for the effects of underpopulating memory in the Lenovo DSS-G storage solution. Our recommendations and takeaways are as follows:

9. For a **G210-G240** the cost saving from underpopulating memory channels does not make up for the performance penalty we measured. If there are no performance requirements for a storage solution and the sizing is based purely on capacity, then underpopulating memory may be a viable option, but care must be taken to ensure the customer is happy with the degraded performance versus what is advertised and that they still meet the minimum memory needed for recovery groups.
10. For a **G250-G280**, whilst the performance penalty is significantly less of an issue versus the G210-G240, the increased total cost of the system means that underpopulating memory will likely only need to provide a negligible discount on the price of the total solution.

Overall, we looked at only some of the ways we can continue to get the best performance out of the Lenovo ThinkSystem based solutions whilst operating in a memory supply constrained environment. The numbers presented here are what we measured on the hardware details in the previous sections and exact savings will vary based on the hardware configured and the cost at the time of sale. We recommend that Lenovo sellers, partners and customers can use the work presented here as a starting point to discuss ways in which they can cost optimise memory in their deployments going forward.

Appendix A1 – List of Codes in HPC Benchmark Suite

Domain	Application	Version #	Input
Chemistry	ABINIT	8.10.3	GS AU
	CP2K	9.1	H2O DFT LS
			H20 512 MD
	GPAW	24.6.0	CU Filament
	Quantum ESPRESSO	7.3.1	AuSurf
			GRIR

Table 4 Chemistry segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
Computer-Aided Engineering (CAE) Open Source	Code Saturne	7.0.0	Cavity 13M
	Nek5000	17.0.0	LPBM 10M
	OpenFOAM	2012	Motorbike 21M
	OpenRadioss	20240729	Camry Impact
			Taurus 10M
			Flycab
			AVL 15M

Table 5 Open Source CAE segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
CAE ISV CFD	Ansys Fluent	2023.1	Pump 2M
			Sedan 4M
			Aircraft Wing 14M
			Truck 14M
			F1 Car 140M
	Altair AcuSolve	2022.3	Nozzle 8M
			Wind Turbine 100M
	Siemens STAR-CCM+	18.04.009-R8	Civil Trim 20M
			LeMans Poly 17M

			Vtm Uhood Fan 64M
			LeMans Coupled 100M
			Vtm Bench 178M

Table 6 CAE ISV Computational Fluid Dynamics segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
CAE ISV Explicit FEA	Ansys LS-DYNA	12.1.0	Car2Car
			ODB 10M
	Altair Radioss	2022.3	Neon 5M
			Taurus 10M
	Dassault Systemes Abaqus Explicit	2023 HF4	E9 Taurus Impact 1M Elem 5M DOF
			E10 1.6M Elem 1M DOF
			E12 2.4M Elem 7M DOF
			E13 Venza Car 43M Elem 144M DOF
			E14 Phone Droptest 2M Elem 7M DOF

Table 7 CAE ISV Finite Segment Analysis segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
Media	OpenMoonRay	Mon Jul 15 16:21:12 2024	bedroom
			coffee_maker
			contemporary_bathroom
			country_kitchen
			glass-of-water
			modern_hall
			salle_de_bain
			the_wooden_staircase
			veach-bidir

	veach-mis

Table 8 Media segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
Molecular Dynamics (MD)	Gromacs	2023.4	ArcherLarge
			Lignocellulose
	NAMD	3.0	20STMV
			STMV
	LAMMPs	22-Dec-22	LJ 535K Atoms

Table 9 Molecular Dynamics segment workloads used in the HPC Application discussion in Section 3

Domain	Application	Version #	Input
Weather/Climate	NEMO	4.2.0	ORCA01
			ORCA025
			GYRE
	MOM6	6.1	Baltic OM4 25
			OM4 05
			SIS2
	WRF	4.6.0	NCAR Bench
			CONUS 12km
			CONUS 2.5km
			Maria 1km

Table 10 Weather segment workloads used in the HPC Application discussion in Section 3

References

[1] Stanford University, “Memory Prices,” 01 07 2026. [Online]. Available: https://dam.stanford.edu/memory-prices.html. [Accessed 10 08 2026].

[2] Lenovo, “Lenovo N1380 Neptune Chassis,” [Online]. Available: https://www.lenovo.com/gb/en/p/servers-storage/servers/supercomputing/lenovo-thinksystem-n1380-neptune-chassis/len21ts0038?.

[3] Lenovo, “Lenovo ThinkSystem SC750 V4 Neptune Server,” [Online]. Available: https://lenovopress.lenovo.com/lp2009-thinksystem-sc750-v4-neptune-server.

[4] Micron Technology Inc, “Fiscal Q3 2026 Earnings Call Prepared Remarks,” 24 June 2026. [Online]. Available: https://investors.micron.com/static-files/631b1a32-5537-46ae-8f40-82e42fc79dfe. [Accessed 10 08 2026].

[5] Lenovo , “Lenovo Distributed Storage Solution for IBM Storage Scale (DSS-G) on ThinkSystem V3,” [Online]. Available: https://lenovopress.lenovo.com/lp1842-lenovo-dss-g-thinksystem-v3.

[6] Netlib, University of Tennessee, “HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers,” [Online]. Available: https://www.netlib.org/benchmark/hpl/.

[7] University of Virginia, “STREAM: Sustainable Memory Bandwidth in High Performance Computers,” [Online]. Available: https://www.cs.virginia.edu/stream/.

[8] HPCG, “HPCG Benchmark,” [Online]. Available: https://www.hpcg-benchmark.org/.

[9] Lenovo, “Lenovo ThinkSystem SR675 V3 and SR675i V3 Servers,” [Online]. Available: https://lenovopress.lenovo.com/lp1611-thinksystem-sr675-v3-server. [Accessed 31 08 2026].

[10] Lenovo, “Lenovo Hybrid AI 285 Platform Guide,” 11 Jun 2026. [Online]. Available: https://lenovopress.lenovo.com/lp2181-lenovo-hybrid-ai-285. [Accessed 31 08 2026].

[11] Lenovo, “ThinkSystem SR675 V3,” June 2026. [Online]. Available: https://pubs.lenovo.com/sr675-v3/sr675_v3_user_guide.pdf. [Accessed 31 08 2026].

[12] MLCommons, “MLCommons Benchmarks,” [Online]. Available: https://mlcommons.org/benchmarks/. [Accessed 31 08 2026].

[13] vLLM, “vLLM,” [Online]. Available: https://vllm.ai/.

[14] Hugging Face Inc., “HuggingFace CLI,” [Online]. Available: https://huggingface.co/docs/huggingface_hub/en/guides/cli.

[15] LMCache Contributors, “LMCache,” 2026. [Online]. Available: https://lmcache.ai/en/. [Accessed 01 09 2026].

[16] NVIDIA, “NVIDIA Dynamo,” 2026. [Online]. Available: https://developer.NVIDIA.com/dynamo. [Accessed 01 09 2026].

[17] IBM, “IBM Storage Scale,” [Online]. Available: https://www.ibm.com/products/storage-scale.

[18] Lenovo, “Lenovo ThinkSystem SR655 V3 Server,” [Online]. Available: https://lenovopress.lenovo.com/lp1610-thinksystem-sr655-v3-server.

[19] Lenovo, “Lenovo ThinkSystem D4390 Direct Attached Storage Enclosure,” [Online]. Available: https://lenovopress.lenovo.com/lp1681-lenovo-storage-thinksystem-d4390-high-density-expansion-enclosure.

[20] HPC Github Group, “IOR and MDTest Github Project,” [Online]. Available: https://github.com/hpc/ior.

Authors

Conor Elrick is an HPC/AI Solutions Architect at Lenovo, specialising in the design, benchmarking, and optimisation of HPC and AI infrastructure. His work focuses on delivering high-performance solutions spanning compute, storage, networking, and accelerated AI systems. He holds a PhD in Theoretical Particle Physics from the University of Edinburgh, with a background in computational physics and large-scale simulation.

We would also like to thank the following people for their insightful discussions into the various topics presented in this work:

- Connor Blumsack
- Kevin Dean
- Aurelien Ortiz
- Raymond Paden
- Simon Thompson
- Mircea Troaca

Trademarks and special notices

References in this document to Lenovo products or services do not imply that Lenovo intends to make them available in every country.

The following terms are trademarks of Lenovo in the United States, other countries, or both:

Lenovo®
Neptune®
ThinkSystem®
XClarity®

The following terms are trademarks of other companies:

Intel® is a trademark of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the U.S. and other countries.

IBM® is a trademark of IBM in the United States, other countries, or both.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used Lenovo products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-Lenovo products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by Lenovo. Sources for non-Lenovo list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. Lenovo has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-Lenovo products. Questions on the capability of non-Lenovo products should be addressed to the supplier of those products.

All statements regarding Lenovo future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local Lenovo office or Lenovo authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in Lenovo product announcements. The information is presented here to communicate Lenovo’s current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard Lenovo benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.

Any references in this information to non-Lenovo websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Lenovo product and use of those websites is at your own risk.