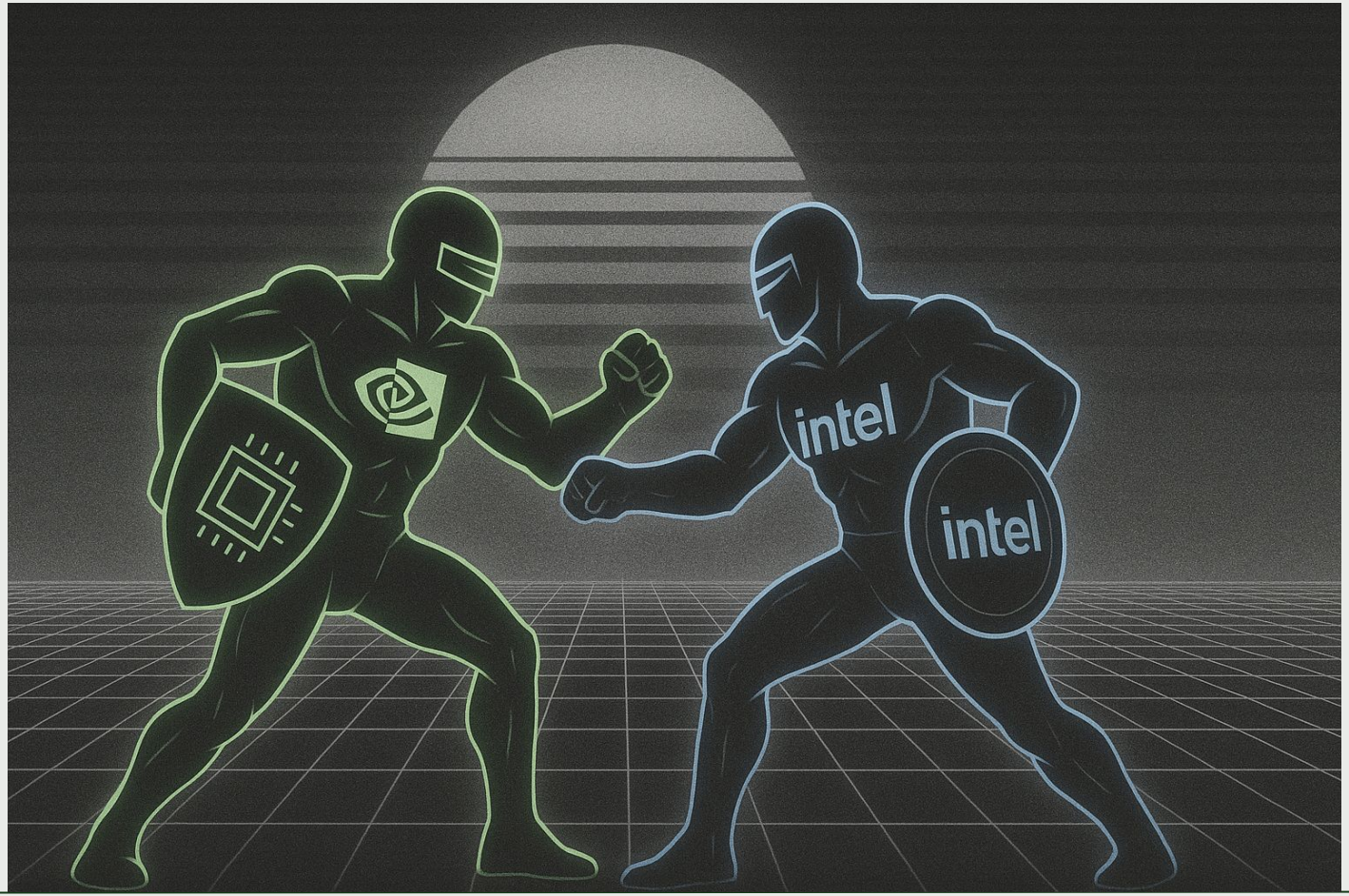
Debunking the CUDA Myth Towards GPU-based AI Systems

Evaluation of the Performance and Programmability of Intel's Gaudi NPU for AI Model Serving

Written by Yunjae Lee & Juntaek Lim et. al. March 2025

Presented by Tyler Crosse November 2025


intel
intel

Core Questions:

1. How does Gaudi-2 hardware performance compare to NVIDIA A100?
2. Is Gaudi-2 programmable enough for real-world optimization?
3. Is CUDA itself the moat, or is it something else?

Key Contributions:

1. First comprehensive characterization of Gaudi NPU (performance + programmability)
2. Novel microbenchmark suite for NPU evaluation
3. Two detailed case studies on optimization strategies
4. Energy efficiency analysis for production workloads

01 Background & Architecture

Table 1: Comparison of NVIDIA A100 and Intel Gaudi-2.

		NVIDIA A100	Intel Gaudi-2	Ratio
Compute	TFLOPS (BF16)	312 (Tensor Cores)	432 (MME)	1.4×
		39 (SIMD Cores)	11 (TPC)	0.3×
Memory	HBM Type	HBM2E		-
	HBM Capacity	80 GB	96 GB	1.2×
	HBM bandwidth	2 TB/sec	2.46 TB/sec	1.2×
	SRAM capacity	40 MB (L2 Cache)	48 MB (Shared)	1.2×
Communication		600 GB/sec bidirectional (NVLink)	600 GB/sec bidirectional (RoCE)	1.0×
Power		400 Watts	600 Watts	1.5×

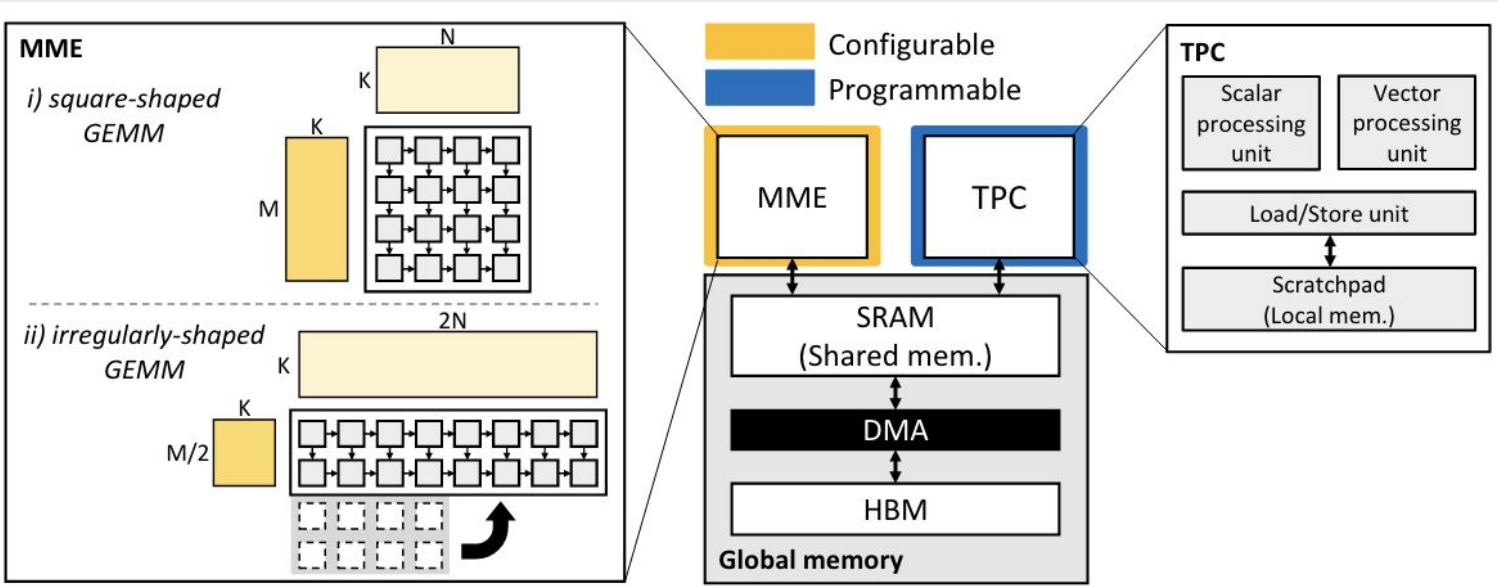


Figure 1: High-level overview of Intel's Gaudi NPU architecture.

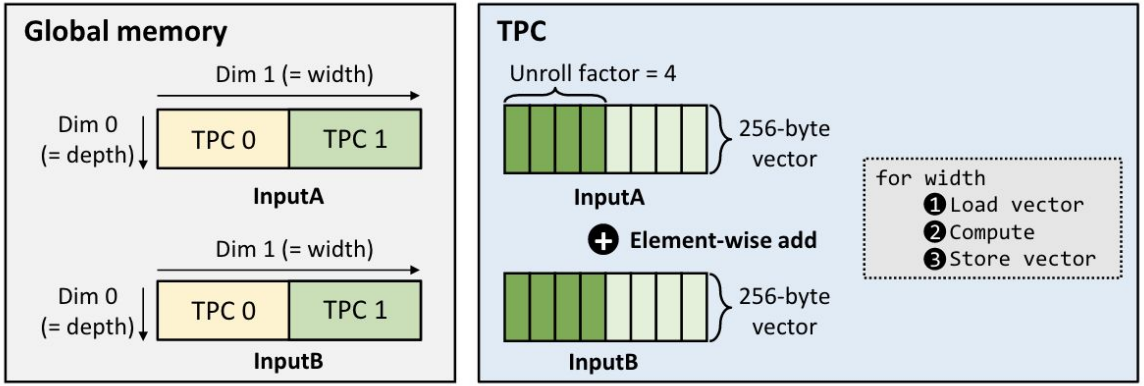


Figure 3: High-level overview of the TPC programming model. Example assumes a program performing an element-wise vector addition operation, partitioned into two dimensions of index space, with each partition being executed by a single TPC. The loop within the TPC program is assumed to be unrolled by a factor of "4" to maximize both instruction-level parallelism and memory-level parallelism.

Example code showing how a matrix multiply-add operation can be programmed for execution on an NVIDIA GPU and Intel Gaudi.

```python
# 64x64 FP32 Matrix multiply-add
# dev = GPU('cuda') or Gaudi('hpu')
A = torch.rand((64,64), device = dev)
B = torch.rand((64,64), device = dev)
C = torch.ones((64,64), device = dev)
D = torch.zeros((64,64), device = dev)

# Compute D = A x B + C
if dev == 'cuda':
    D = torch.wmma_cuda(A, B, C, D)

elif dev == 'hpu':
    # Launch GEMM on MME
    result = torch.matmul(A, B)

    # Launch ADD on TPC
    D = torch.add_tpc(result, C, D)
```

(a) At the PyTorch level, the corresponding low-level kernels are executed based on the device type (i.e., GPU ("cuda", line 9) or Gaudi ("hpu", line 12)).

```cuda
// For simplicity, we ignore matrix tiling
__global__ void wmma_cuda (float *A, float *B,
                           float *C, float *D) {
  // Declare matrices and load them from global memory
  wmma::fragment<wmma::matrix_a, …> frag_a;
  wmma::load_matrix_sync(frag_a, A, …);
  …
  // Compute A x B using Tensor Cores
  wmma::mma_sync(frag_acc, frag_a, frag_b, frag_acc);

  // ADD C matrix to the intermediate result
  for (int i = 0; i < frac_c.num_elements; i++)
    frag_c.x[i] = frag_acc.x[i] + frag_c.x[i];

  // Store the result
  wmma::store_matrix_sync(D, frag_c, …);
}
```

In NVIDIA's CUDA, the matrix multiply-add operation is performed within a single kernel launch using the WMMA APIs, which leverage Tensor Cores alongside SIMD Cores.

Example code showing how a matrix multiply-add operation can be programmed for execution on an NVIDIA GPU and Intel Gaudi.

```
1  # 64x64 FP32 Matrix multiply-add
2  # dev = GPU('cuda') or Gaudi('hpu')
3  A = torch.rand((64,64), device = dev)
4  B = torch.rand((64,64), device = dev)
5  C = torch.ones((64,64), device = dev)
6  D = torch.zeros((64,64), device = dev)
7
8  # Compute D = A x B + C
9  if dev == 'cuda':
10     D = torch.wmma_cuda(A, B, C, D)
11
12 elif dev == 'hpu':
13     # Launch GEMM on MME
14     result = torch.matmul(A, B)
15
16     # Launch ADD on TPC
17     D = torch.add_tpc(result, C, D)
```

(a) At the PyTorch level, the corresponding low-level kernels are executed based on the device type (i.e., GPU ("cuda", line 9) or Gaudi ("hpu", line 12)).

In Gaudi, the GEMM operation can only be handled at the PyTorch level (line 14 in (a)), so a TPC-C kernel (add_tpc) is called at the PyTorch level to execute the subsequent add operation.

```
1  void add_tpc(tensor inputA, tensor inputB, tensor outputC) {
2    // Get index space information
3    int5 InputCoord, OutputCoord;
4    int  depthStart, depthEnd, widthStart, widthEnd = get_index_space_information();
5
6    // A single step in the depth dimension is 256B / 4B (FP32) = 64
7    int depthStep = 64; int depth_dimension = 0; int width _dimension = 1;
8
9    // Declare input/output for 256-byte FP32 vectors (=float64)
10   float64 x, y, result;
11
12   for (int d = depthStart; d < depthEnd; d += depthStep) {
13     InputCoord[depth_dimension] = d; OutputCoord[depth_dimension] = d;
14
15     // Unroll factor is set as 4
16     #pragma unroll(4)
17     for (int w = widthStart; w < widthEnd; w += 1) {
18       InputCoord[width_dimension] = w; OutputCoord[width_dimension] = w;
19
20       // Fetches 256-byte vector from global memory
21       x = v_f32_ld_tnsr(InputCoord, inputA); y = v_f32_ld_tnsr(InputCoord, inputB);
22
23       // Element-wise vector add operation
24       result = v_f32_add_b(x, y);
25
26       // Stores the 256-byte vector to global memory
27       v_f32_st_tnsr(OutputCoord, outputC, result);
28       …
29 }
```

02 Performance

Key takeaway #1:
When performing GEMM, Gaudi-2 achieved both higher absolute performance and greater compute utilization than A100.
This superior GEMM performance and efficiency can be attributed not only to Gaudi-2's higher max compute throughput but, more importantly, to the configurability of the Gaudi-2 MME, which enables its systolic array to flexibly adapt its geometry to be most optimal for the target GEMM's (M,K,N) shape.

Table 2: Evaluated microbenchmarks.

Microbenchmark		System	Implementation
Compute	GEMM	Gaudi-2	PyTorch API
		A100	PyTorch API
	non-GEMM	Gaudi-2	TPC-C
		A100	CUDA
Memory	Vector gather-scatter	Gaudi-2	TPC-C
		A100	CUDA
Communication	Collective communication	Gaudi-2	Intel HCCL [28]
		A100	NVIDIA NCCL [59]

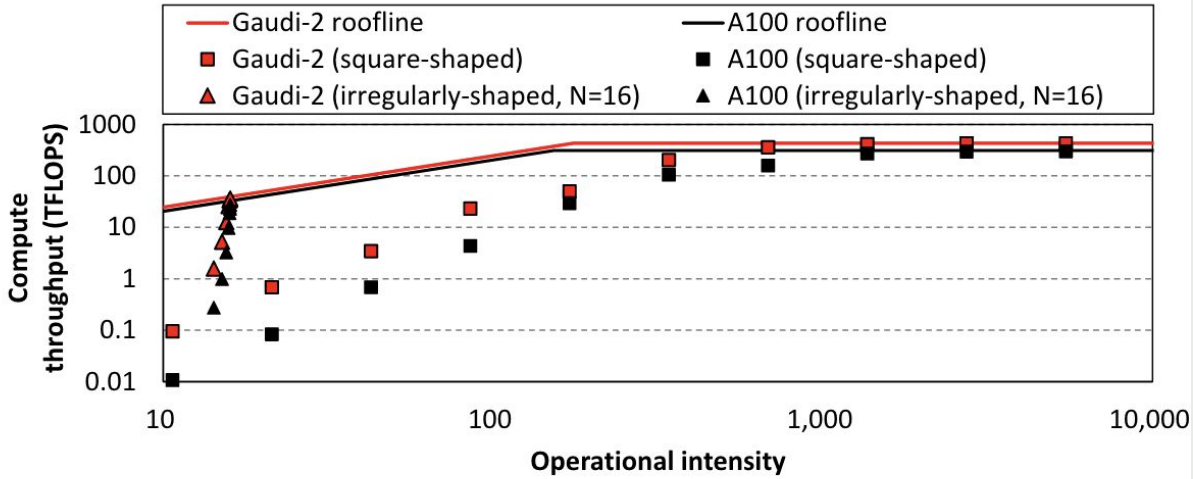


Figure 4: Roofline model showing the achieved TFLOPS of Gaudi-2 and A100 (BF16). The irregularly-shaped GEMMs, represented by triangle markers, have the N dimension size fixed at 16. Performance is measured when both Gaudi-2 and A100 are configured to operate at its maximum possible frequency.

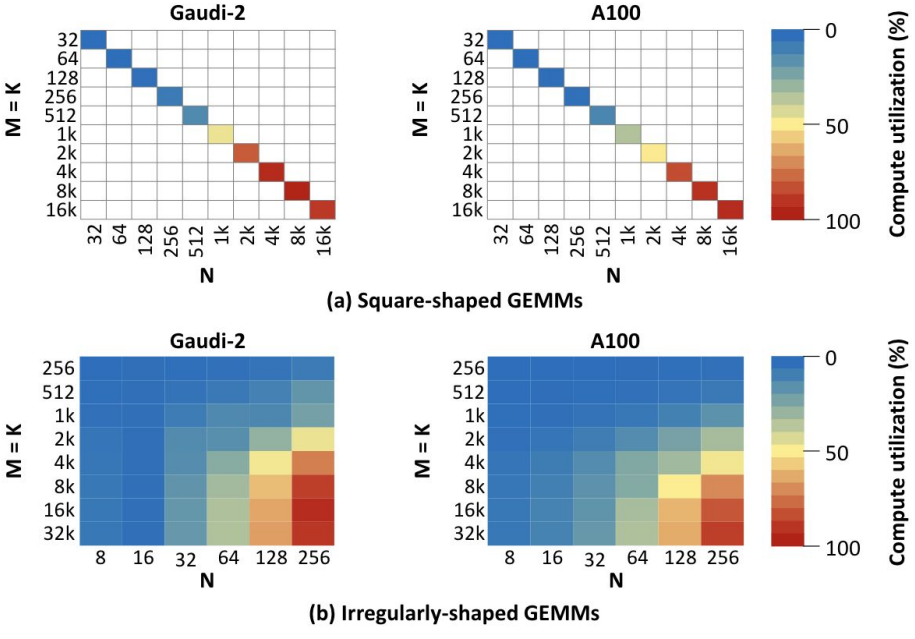


Figure 5: Compute utilization when the GEMM operations are (a) square-shaped (M=K=N) and (b) irregularly-shaped (M and K are relatively larger than the fixed N). In all the thermal plots presented in this paper, warmer colors indicate higher values. In (a), we leave the GEMM shapes that do not satisfy M=K=N vacant.

Key takeaway #2:
Due to the 3.5× performance gap in vector math throughput, Gaudi-2 falls short of A100 in terms of absolute non-GEMM performance. However, in terms of compute “efficiency”, Gaudi-2 is comparable to A100 across all evaluated microbenchmarks, demonstrating the competitiveness of its design.

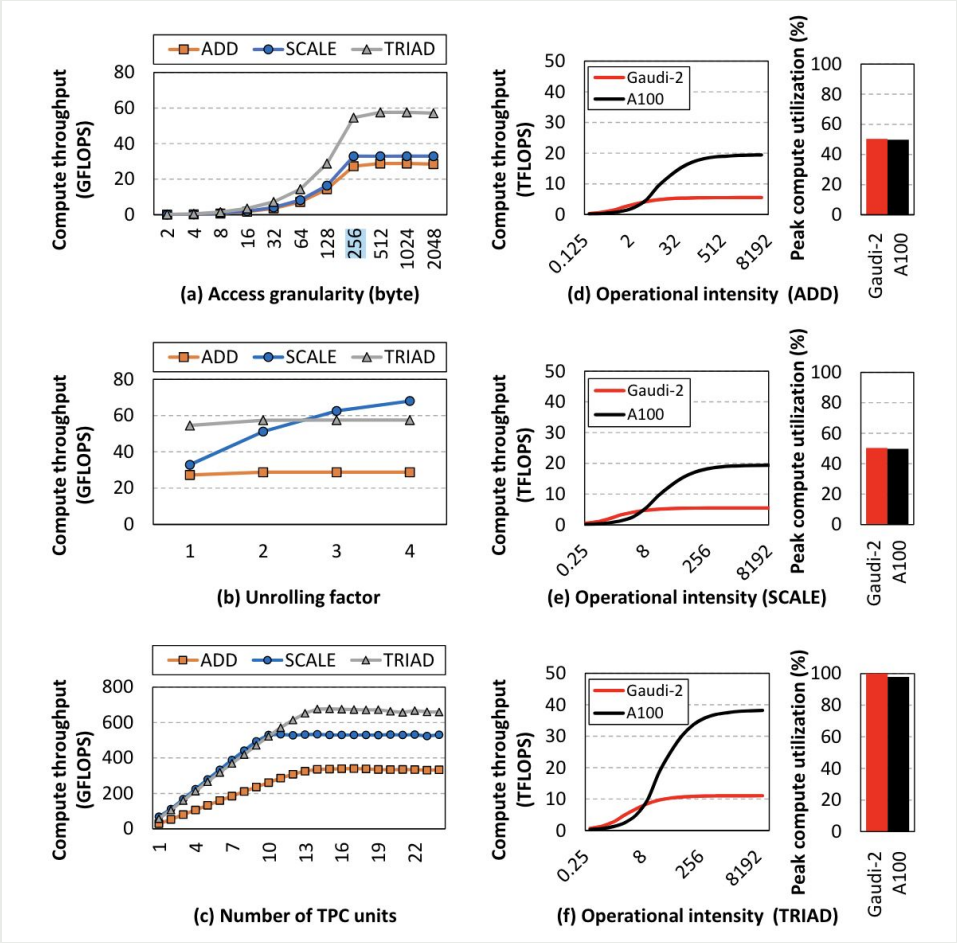


Figure 8: Compute throughput of ADD, TRIAD, and SCALE over a vector with 24 million scalar elements (BF16). The effects of (a) data access granularity and (b) unrolling factor on a single TPC's throughput are shown in figures (a and b). In (c), we scale out the number of TPCs by weak scaling the three microbenchmarks.

The left axis on figures (d, e, and f) illustrate the compute throughput when the operational intensity of (d) ADD, (e) SCALE, and (f) TRIAD is artificially increased. The right axis on these figures (d, e, and f) shows the compute utilization when Gaudi-2 and A100's compute throughput peaks at its saturation point.

Key takeaway #3:
Gaudi-2 provides competitive memory performance for regular data transfers with streaming access patterns. However, for random memory accesses like vector gather-scatter, Gaudi-2's performance falls short of A100's when the data transfer size is smaller than its 256-byte minimum access granularity.

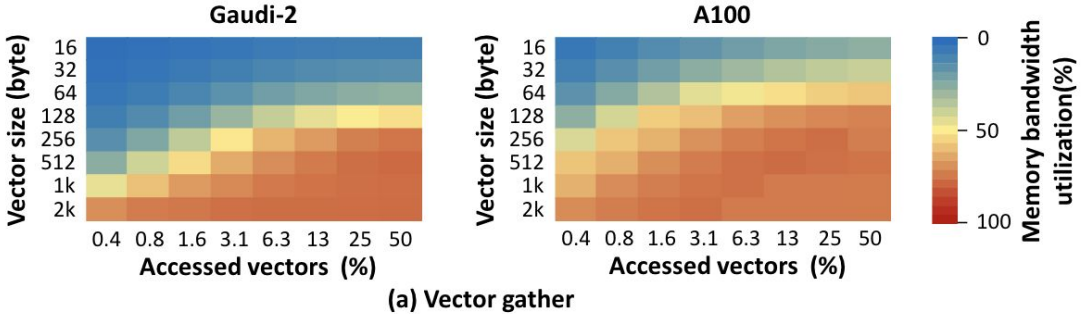


Figure 9: Memory bandwidth utilization of (a) vector gather and (b) scatter operations. The x-axis shows the proportion of vectors accessed among the total 4M vectors, i.e., the fraction of vectors either gathered from or scattered to random memory locations.

Key takeaway #4:
The system-level collective communication performance of Gaudi-2 based system falls short of A100, not because of the limitations of the Gaudi-2 processor architecture itself, but because of its lack of an all-to-all network switch provisioned in NVIDIA's DGX A100 system. This switch enables A100 GPUs to more flexibly exploit intra-node network bandwidth, regardless of the number of devices involved in communication, a feature that is currently missing in Intel Gaudi-2 systems.

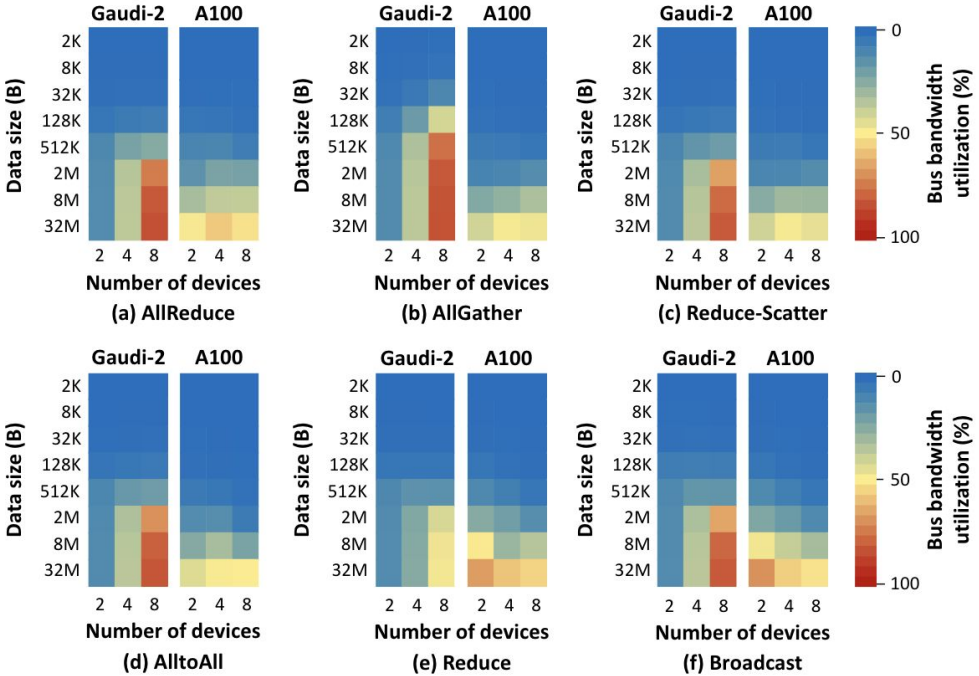


Figure 10: Bus bandwidth utilization of Gaudi-2 and A100 for collective communication operations for data sizes ranging from 2 KB to 32 MB: (a) AllReduce, (b) AllGather, (c) Reduce-Scatter, (d) AlltoAll, (e) Reduce and (f) Broadcast operations.

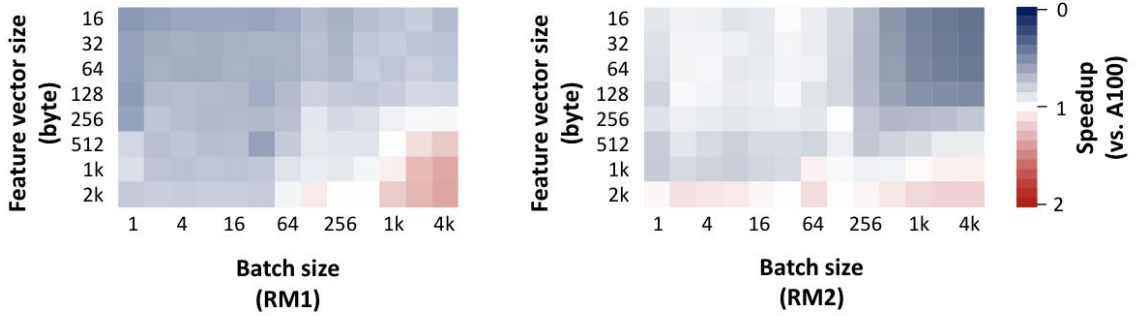
03 End-to-End Application Analysis

Table 3: Evaluated end-to-end AI workloads.

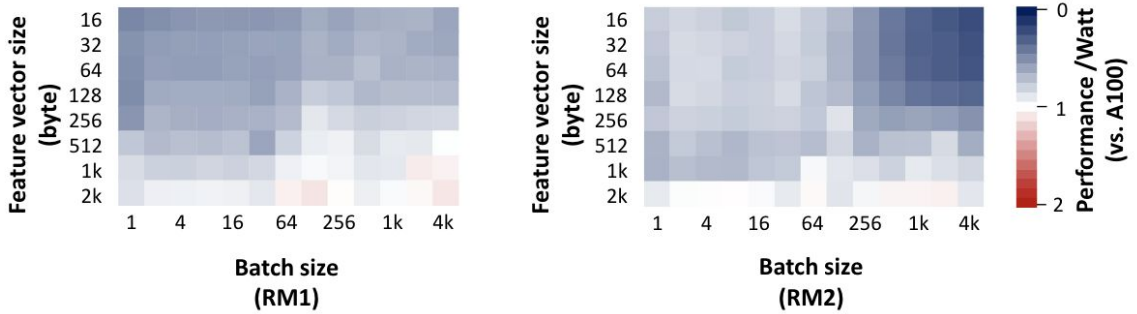
Model		Embedding layer	MLP layer	Interaction layer
DLRM-DCNv2 [52]	RM1	# tables: 10 # embeddings: 1M # gathers: 10	Bottom: 512-256-64 Top: 1024-1024-512-256-1	Low rank dim: 512 # layers: 3
	RM2	# tables: 20 # embeddings: 1M # gathers: 100	Bottom: 256-64-64 Top: 128-64-1	Low rank dim: 64 # layers: 2

Model		Embedding layer	Decoder layer
Llama-3.1 [12]	8B	# vocabularies: 128,256	# layers: 32 # heads for query: 32 # heads for key, value: 8 hidden/intermediate size: 4,096/14,336
	70B	# vocabularies: 128,256	# layers: 80 # heads for query: 64 # heads for key, value: 8 hidden/intermediate size: 8,192/28,672

Key takeaway #5:
LLM serving is primarily dominated by matrix multiplications, so Gaudi-2 demonstrates superior energy-efficiency than A100, achieving an average 48% and 52% improvement for single- and multi-device serving, respectively. However, RecSys relies on vector gathers and small MLP layers, so Gaudi-2 generally lags behind A100 with an average 20% performance slowdown and an average 28% drop in energy-efficiency.



(a) Performance



(b) Energy-efficiency

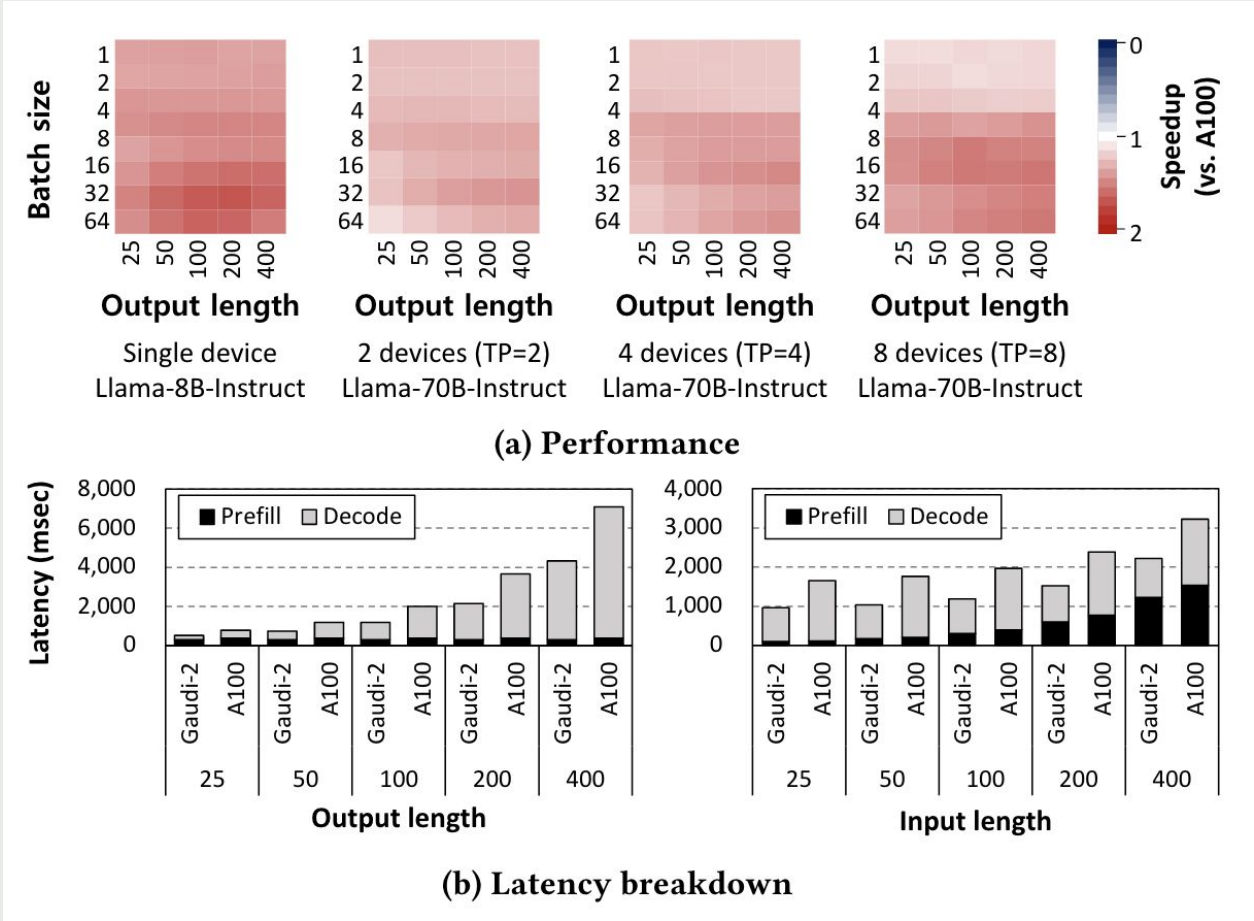
Figure 11: Gaudi-2's improvement in (a) performance and (b) energy-efficiency over A100 when RM1, RM2 are served on a single device.

Figure 12: (a) Gaudi-2's improvement in performance over A100 when the Llama-3.1 models are served on a single device and on multiple devices.

(b) For the Llama-3.1-8B-Instruct model served on a single device, we divide the latency into prefill and decoding stages, keeping the batch size fixed at 64.

The left graph shows the latency breakdown when the input length is fixed at 100 while varying the output length.

The right graph shows the breakdown when the output length is fixed at 100 while the input length is varied.



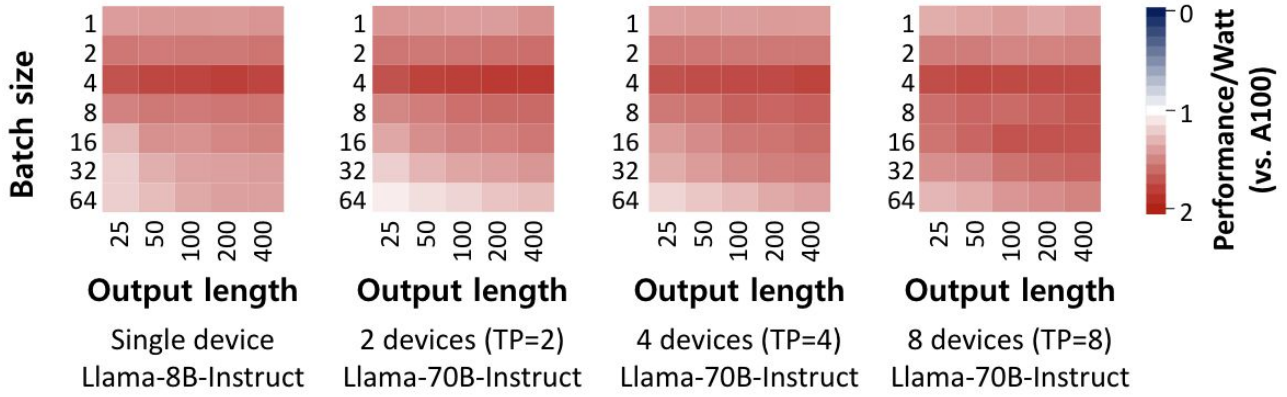


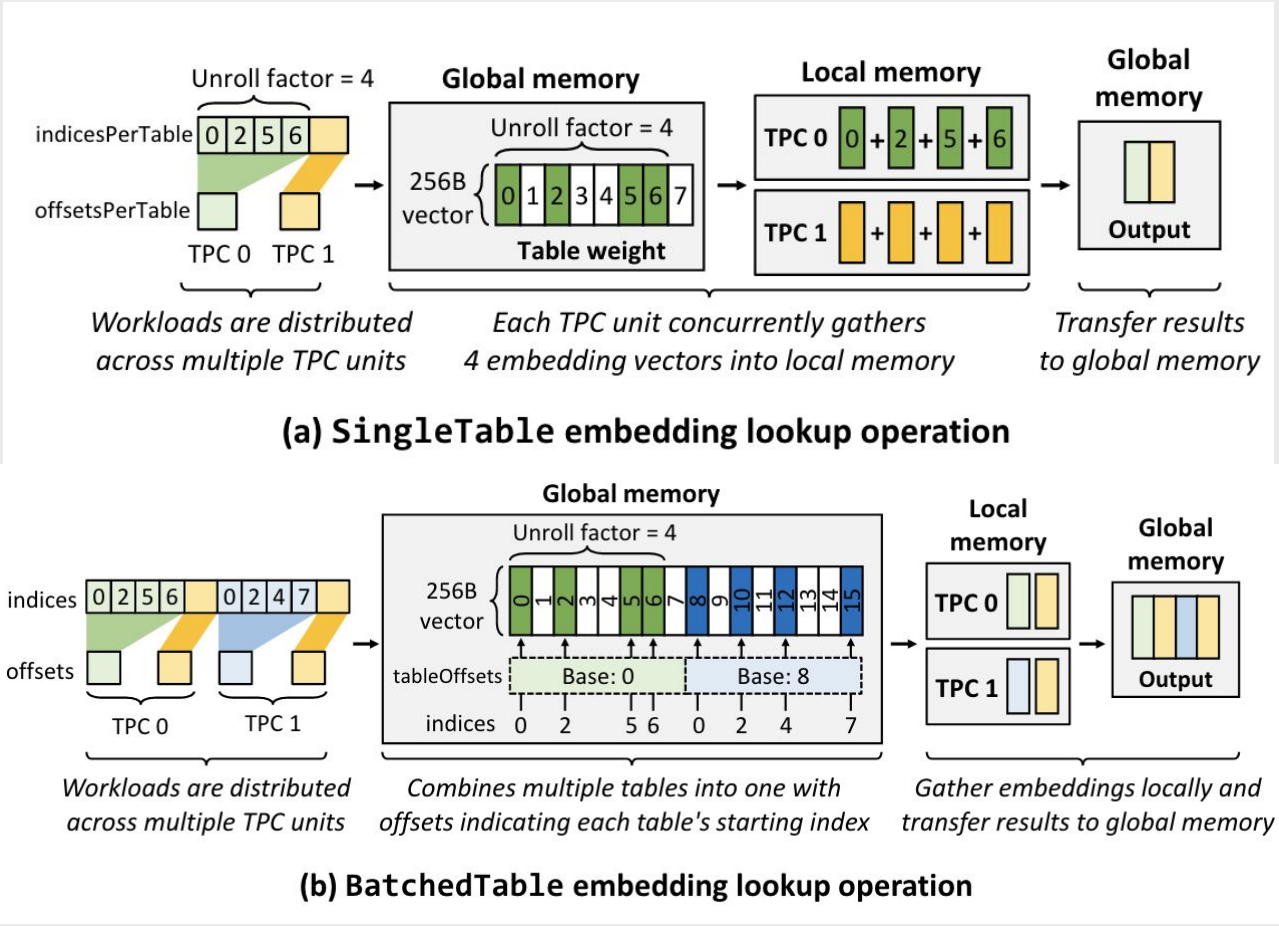
Figure 13: Gaudi-2's improvement in energy-efficiency over A100 when the Llama-3.1 models are served on a single device and on multiple devices.

04
Programmability Case Studies

Key takeaway #6:
This case study confirmed that the TPC-C programming system provides a sufficient level of flexibility for low-level performance optimizations. Compared to state-of-the-art FBGEMM-based A100 executions, our Gaudi-2 optimized kernel for embedding layers achieved, on average, 95% of the throughput of A100 for large embedding vector sizes (≥256 bytes) but only 47% for small vectors (<256 bytes). The noticeable performance degradation for small vectors primarily stems from A100's superior hardware architecture (which better supports fine-grained memory accesses) rather than from the differences in the programming models.

Figure 14: Block diagrams illustrating
(a) SingleTable embedding lookup operation, which processes embedding tables sequentially with loop unrolling to improve throughput, and

(b) BatchedTable embedding lookup operation, which consolidates multiple tables into a single large table, enhancing memory bandwidth utilization at lower batch sizes by treating each table with offset-based indexing.



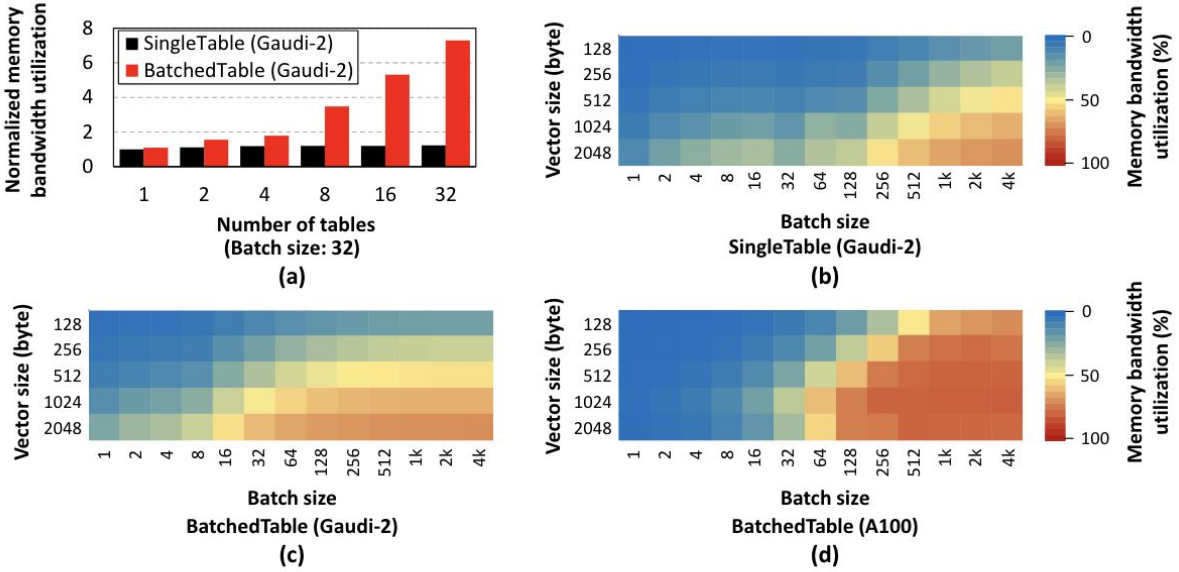


Figure 15: Memory bandwidth utilization of embedding lookup operations, using the embedding layer configuration from RM2 (Table 3). (a) Utilization is normalized to `SingleBatch` with a vector size fixed at 256 bytes while varying the numbers of tables. (b, c, d) Utilization when varying both embedding vector sizes and batch sizes.

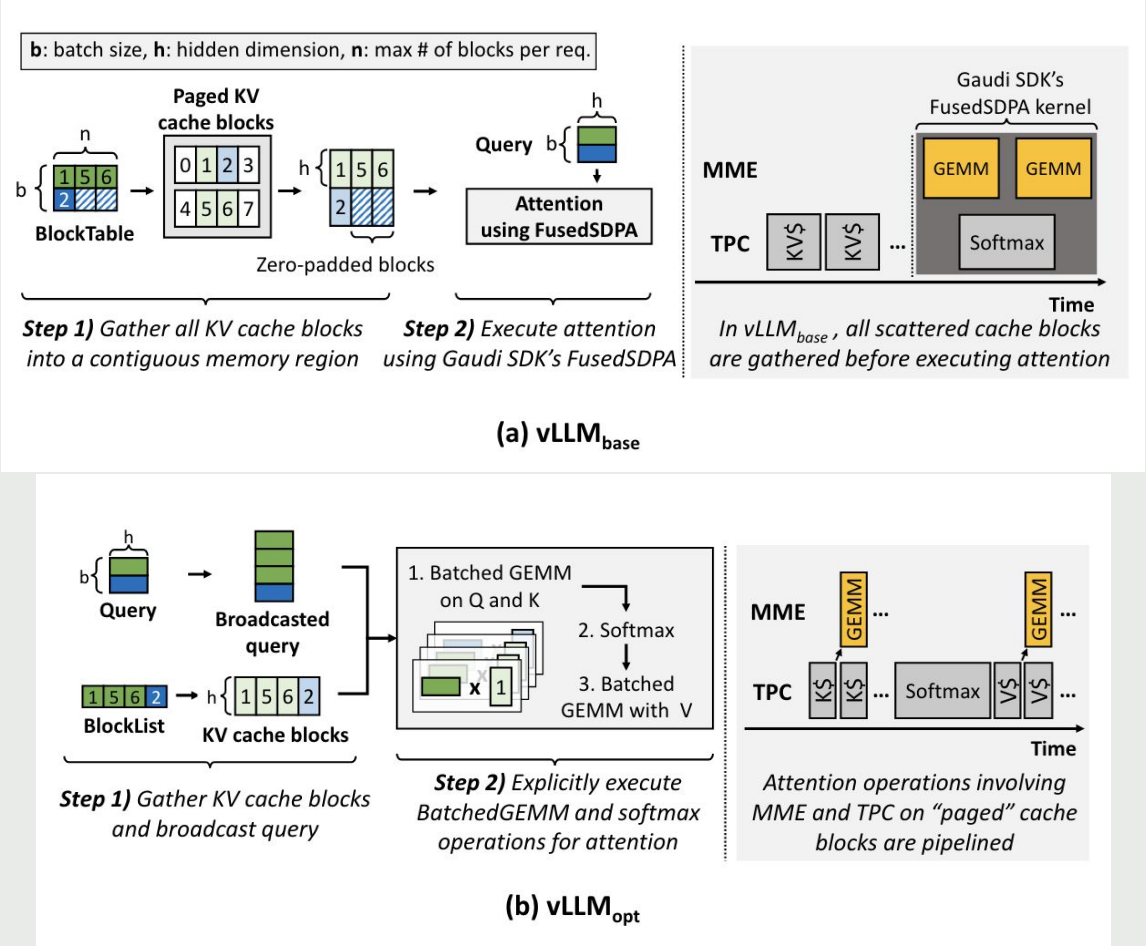
Key takeaway #7:

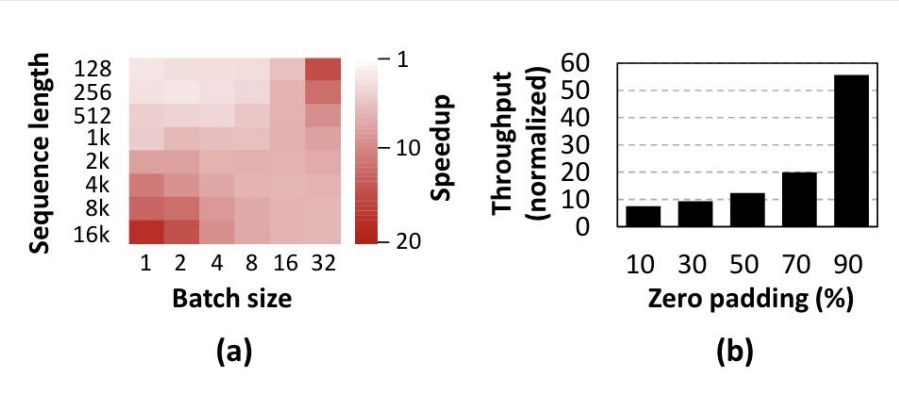
This case study showed that, while Gaudi SDK's current lack of support for directly programming MMEs within the low-level TPC-C kernel imposes restrictions on programmer flexibility, graph compiler can still effectively capture the appropriate level of parallelism to better utilize its compute resources when programmed properly at the PyTorch level. Consequently, while the performance-optimized Gaudi vLLM achieved 45% of the performance of a GPU-optimized vLLM, Gaudi-2's end-to-end LLM performance was shown to be competitive to A100.

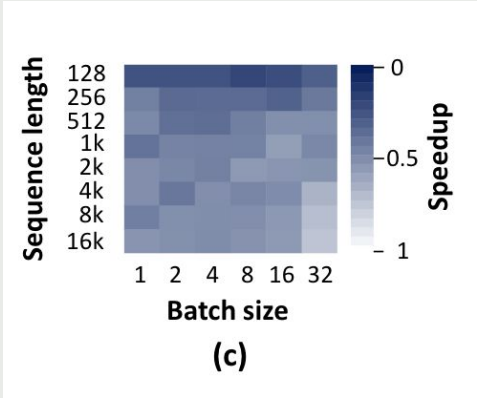
Figure 16: High-level overview of the PagedAttention implementation and its execution timeline in

(a) baseline vLLMbase and

(b) performanceoptimized vLLMopt
.





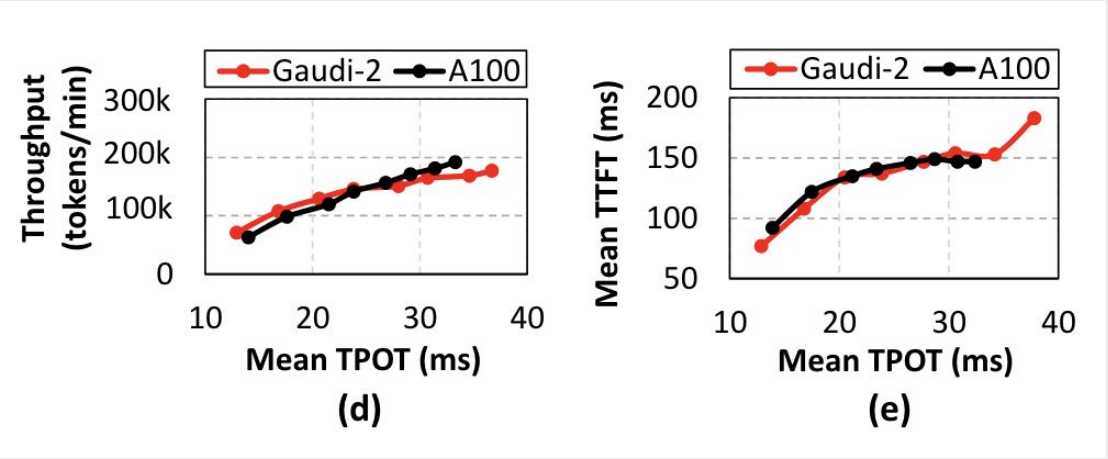


(a and b) Effectiveness of vLLMopt on improving the performance of PagedAttention vs. vLLMbase (results are normalized to vLLMbase):

(a) we vary the input sequence length and batch size and measure output token generation latency (fraction of zero-padded indices are 0% in this experiment), and

(b) under the sequence length=4K and batch size=32 datapoint in (a), we vary the proportion of zero-padded indices in BlockTable from 10 to 90% to evaluate the effect of redundant KV cache block gathers.

In (c), we show how much PagedAttention's throughput improves with vLLMopt by comparing it to the A100, with results normalized to the A100.



In (d) and (e), we sweep the maximum decode stage batch size [80] and present

(d) the changes in end-to-end serving throughput and

(e) the observed mean TTFT (Time-To-First-Token) and mean TPOT (Time-Per-Output-Token) values. The results are collected on a single vLLMopt based Gaudi-2 and A100. To properly reflect LLM serving system's dynamism and variable output length, we used the Dynamic-Sonnet dataset [13].

05 Discussion & Future Work

Future Work mentioned in the paper:

- Comparison against other hardware (AMD GPUs, other NPUs).
- Evaluating performance on large-scale AI training scenarios.

Final Thoughts:

- Intel's Gaudi-2 has the potential to be a strong contender in the AI server market, especially for LLM-centric workloads.
- Its success will depend less on winning over low-level programmers and more on Intel's ability to provide a flawless, high-performance software experience within major AI frameworks.

Accelerator	Announce → GA	HBM (capacity / BW)	Scale-out fabric	Typical 2024–2025 pricing signal*
Intel Gaudi 2	May 10 2022 → 2022	96 GB HBM2e / **2.45 TB/s**	**24×100 GbE** on-chip	**8-GPU kit $65k** (Intel ref)
Intel Gaudi 3	Apr 9 2024 → 2H 2024	128 GB HBM2e / **3.7 TB/s**	**24×200 GbE** on-chip	**8-GPU kit $125k** (Intel ref)
NVIDIA A100	May 14 2020 → 2020	40/80 GB HBM2(e) / **1.6–2.0 TB/s**	NVLink + IB/Ethernet	**DGX A100 $199k** (8×A100 system at launch)
NVIDIA H100	Mar 2022 → Sept 2022 prod	80 GB HBM3 (3.35 TB/s); NVL 96 GB	NVLink + IB/Ethernet	**$27k–$40k** per GPU (range) **$216k–$320k** x8 (range)

Thank You!

Questions?