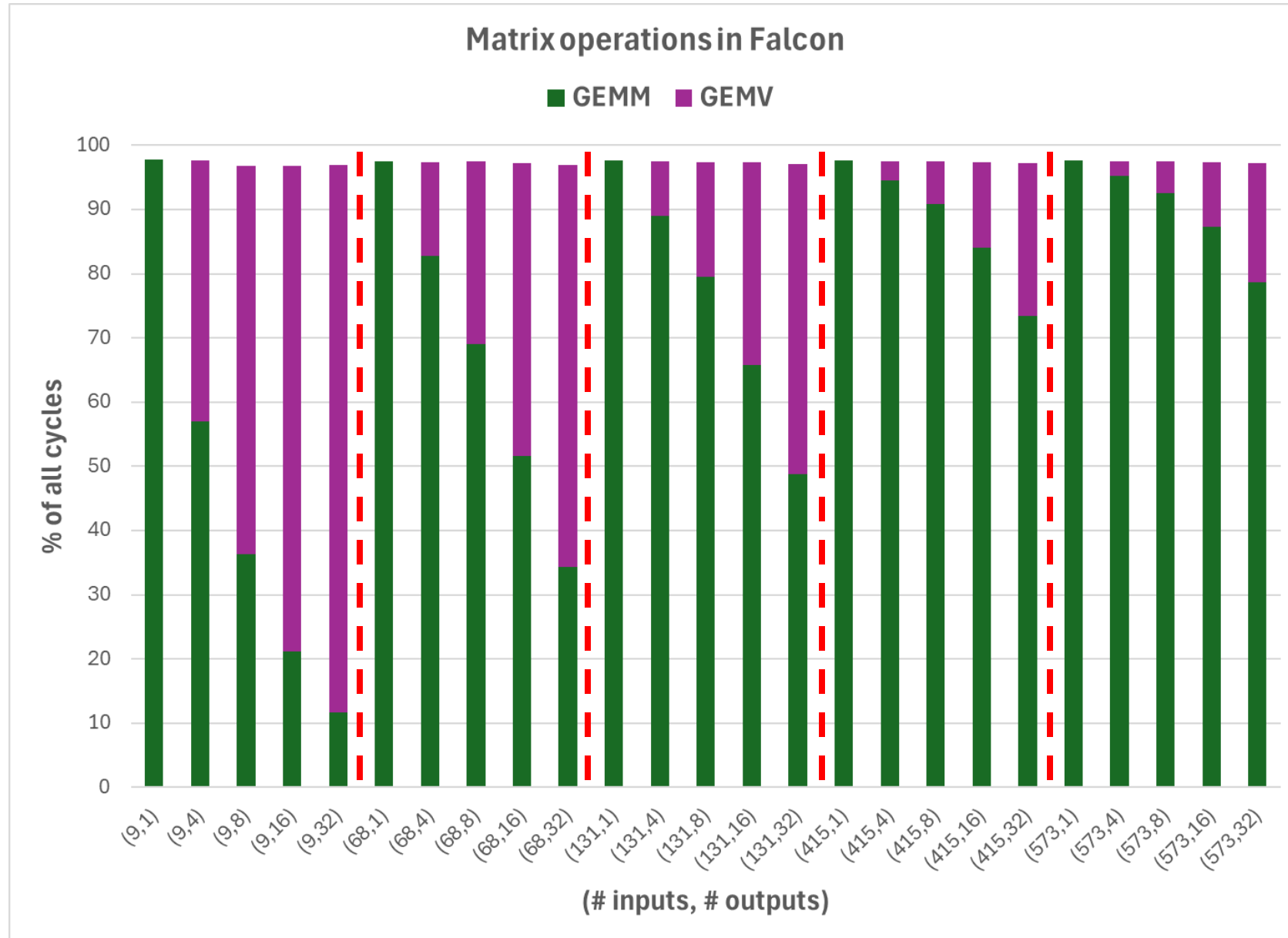


Why (and how) matrix processing?

José Moreira
IBM Research

Presented at NHR PerfLab Seminar, December 09, 2025

Execution profile of FALCON-7B (LLM on PyTorch)



Two regimes of LLM execution

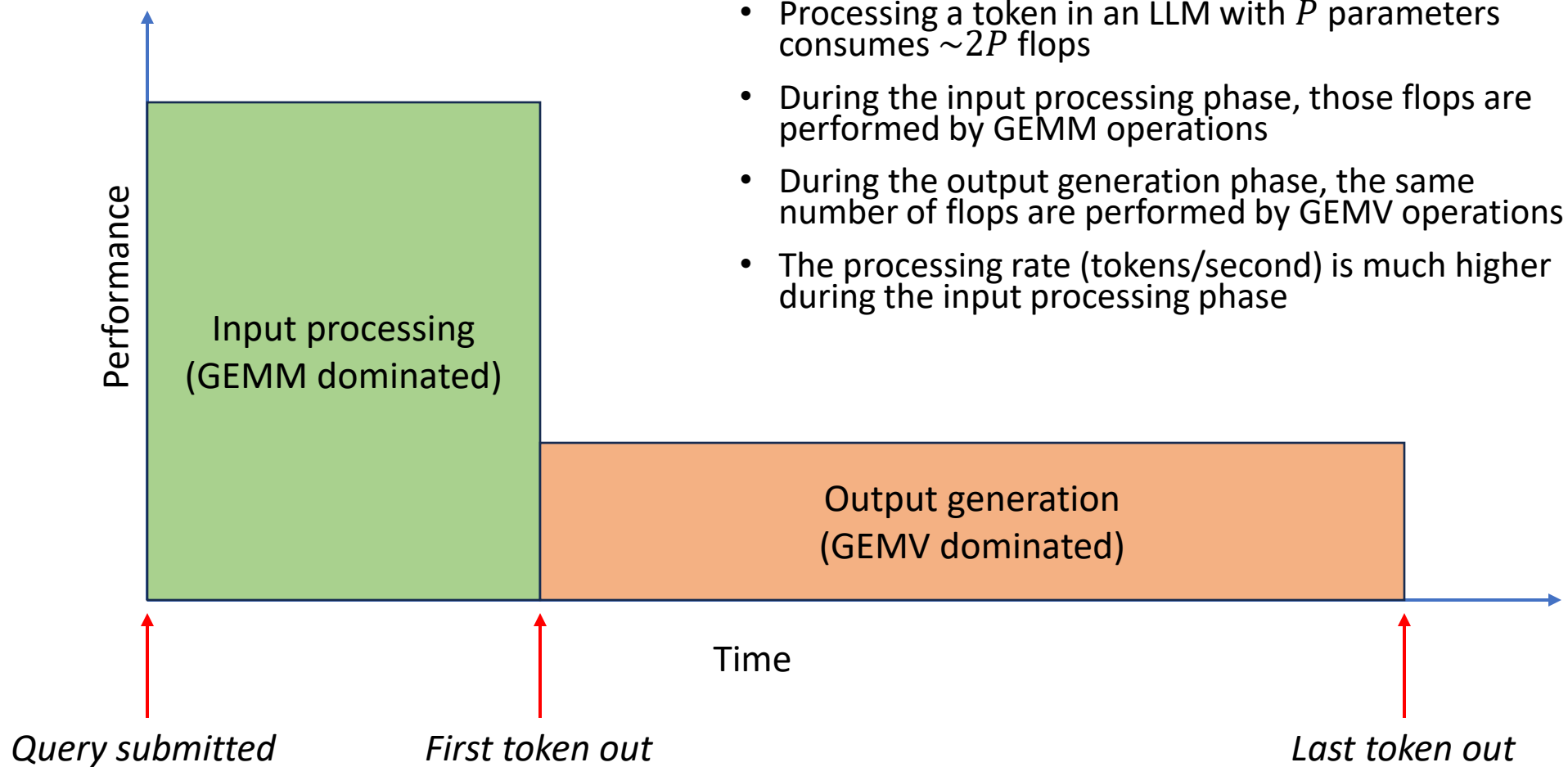
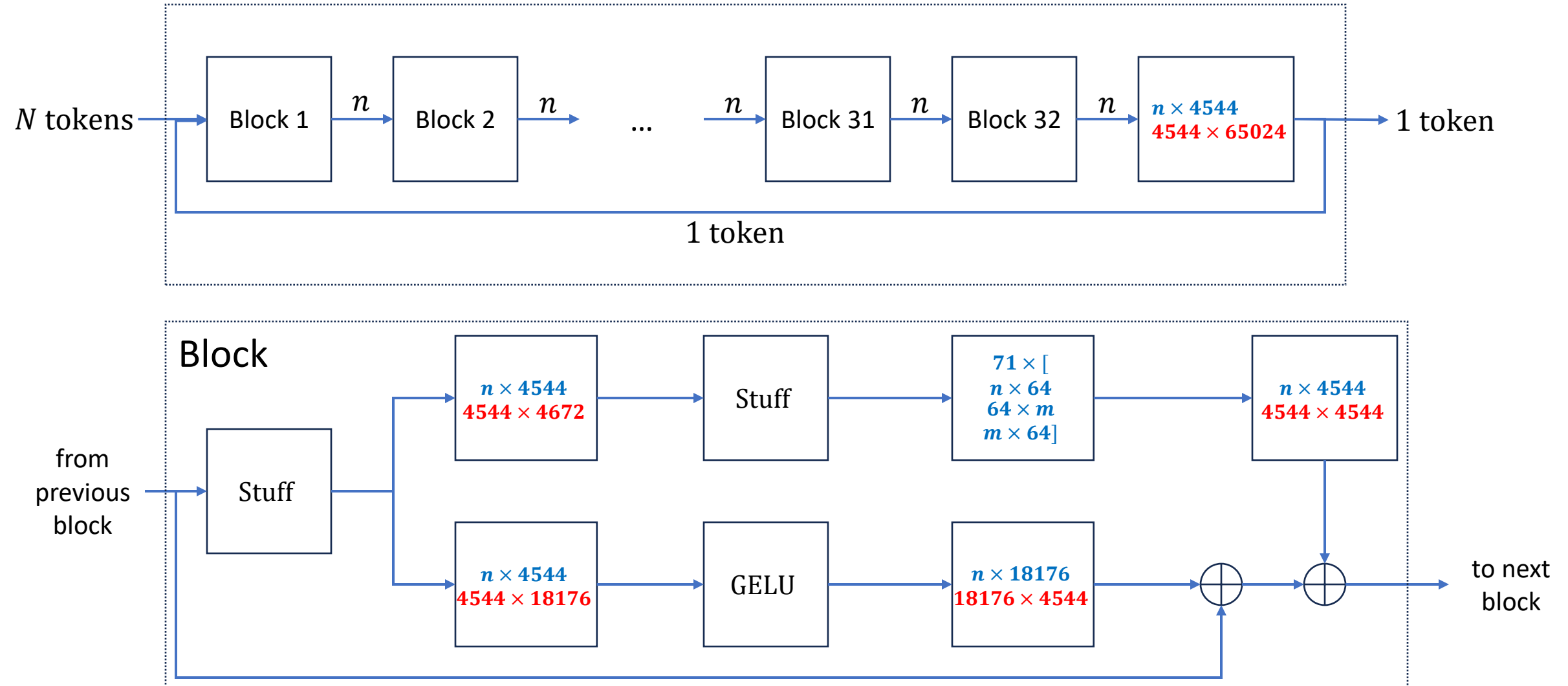
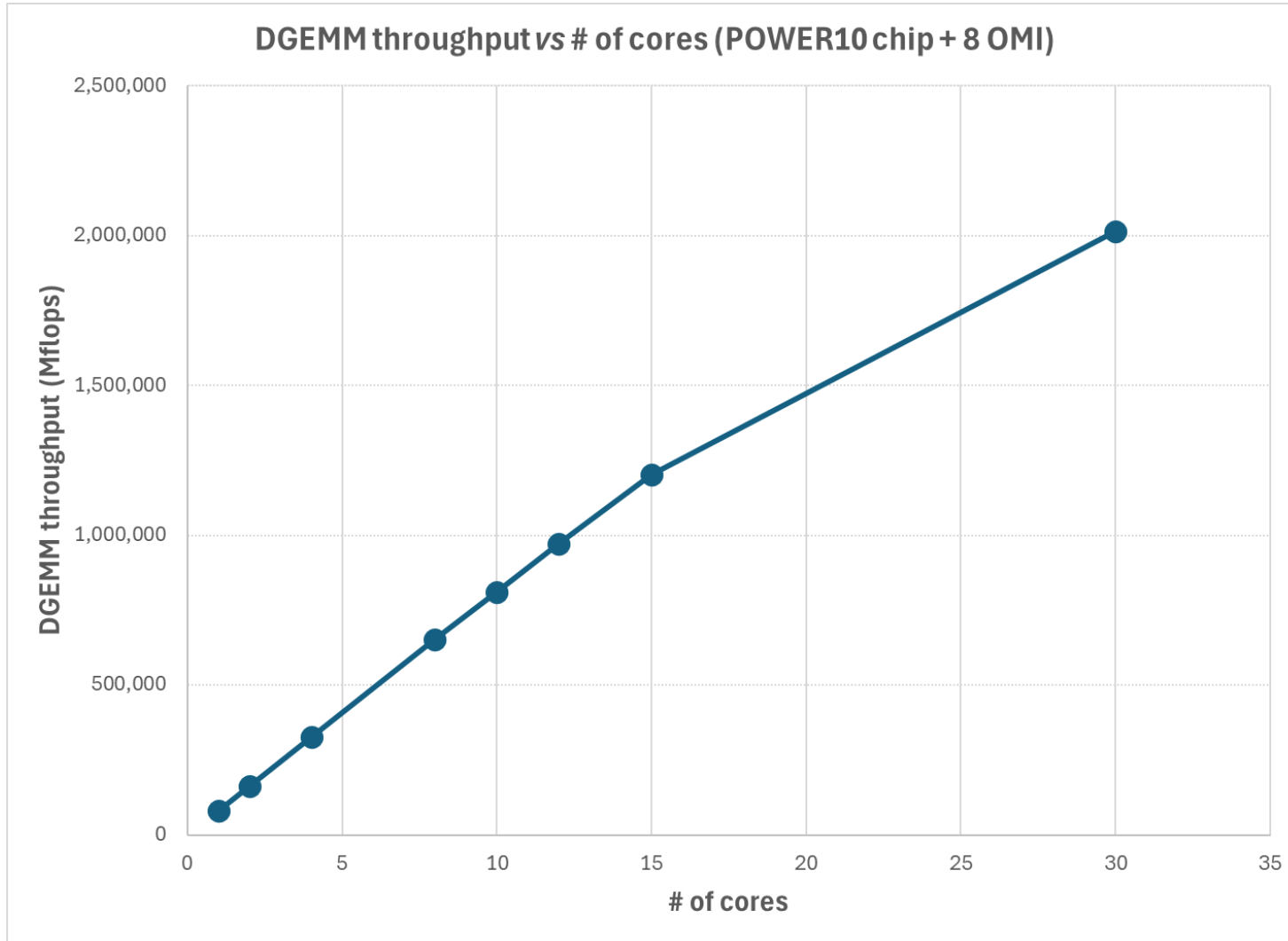


Diagram of FALCON-7B LLM (Hugging Face)

- A token is a 4544-element vector
- $m = N$ for first pass, increments by 1 each pass
- $n = N$ for the first pass, 1 for additional passes

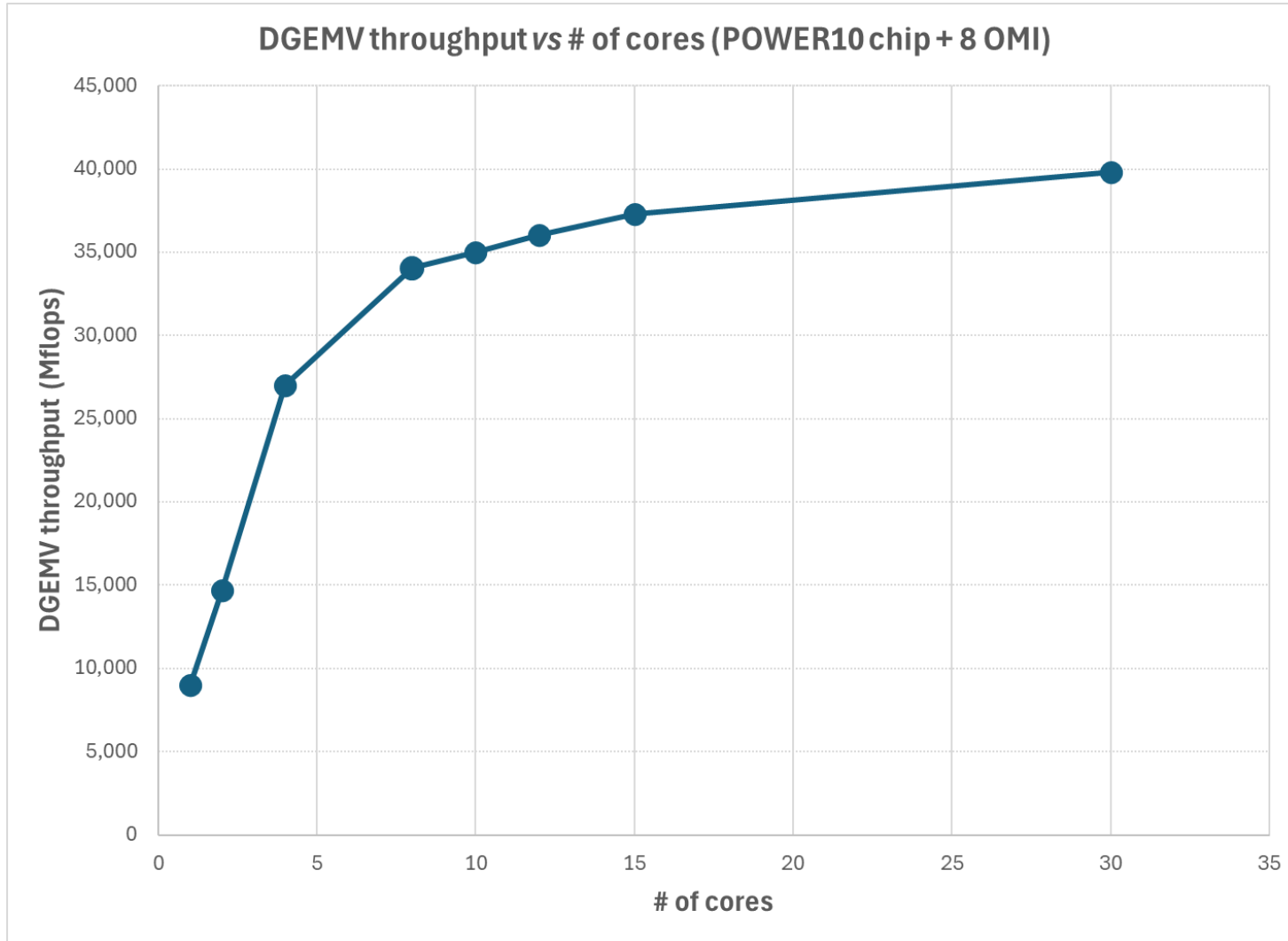


The architect's dream (GEMM)



- GEMM computes
$$C \leftarrow \alpha A^{[T]} B^{[T]} + \beta C$$
- If all matrixes are $N \times N$, the computational intensity is
$$\eta = \frac{\text{\#flops}}{\text{\#load/store}} = \frac{2N^3 + 2N^2}{4N^2} \sim \frac{N}{2}$$
- Almost “unlimited” scalability for large problems
- Proliferation of matrix units:
 - IBM POWER10/POWER11
 - Intel Sapphire Rapids, ...
 - NVIDIA GPUs
 - AMD GPUs
 - Dedicated AI chips

The architect's nightmare (GEMV)



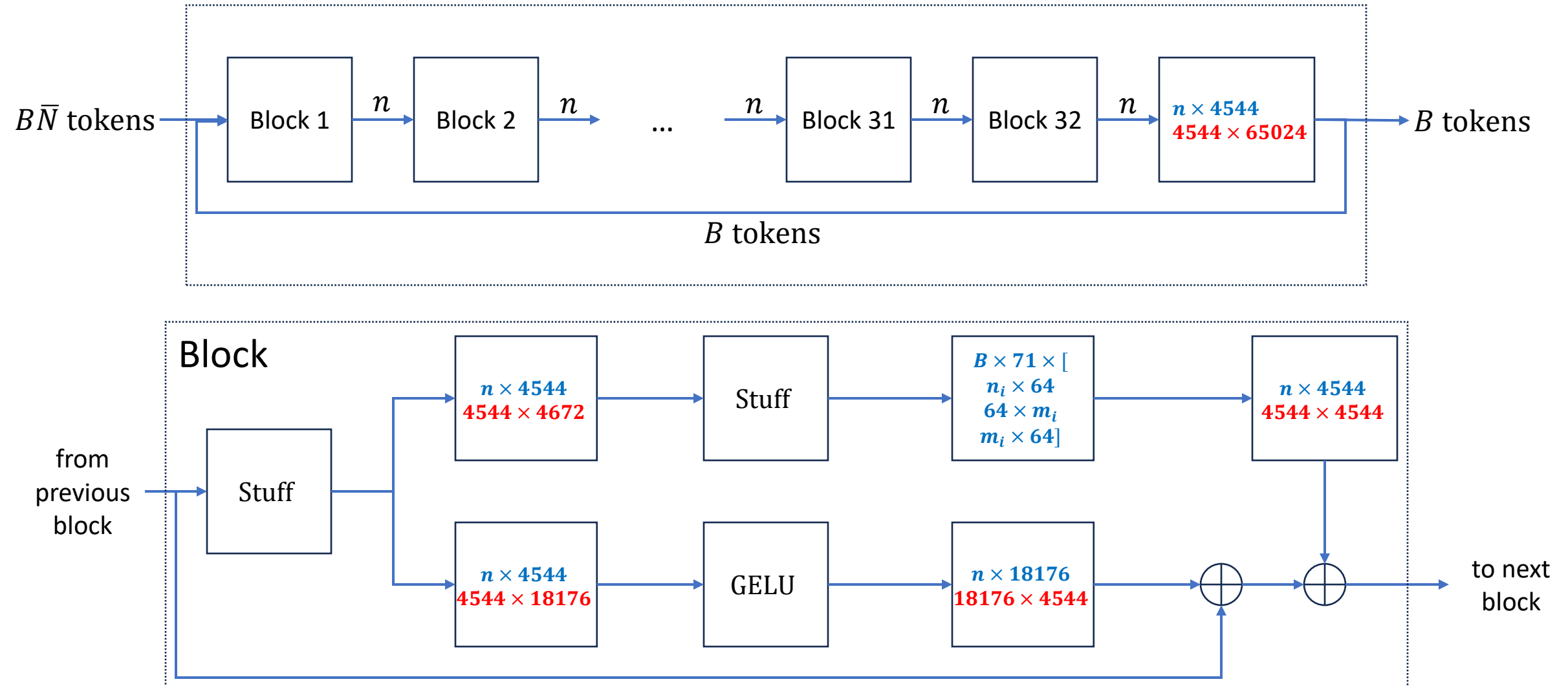
- GEMV computes
$$y \leftarrow \alpha A^{[T]}x + \beta y$$
- If the matrix is $N \times N$, the computational intensity is
$$\eta = \frac{\text{\#flops}}{\text{\#load/store}} = \frac{2N^2 + 2N}{N^2 + 2N} \sim 2$$
- Scalability quickly constrained by memory bandwidth
- No significant progress since the Cray-1, because there is none to be had!

Increasing the size of the input

- *Prompt engineering* is the general name for techniques that transform a user-generated LLM prompt (typically short) to a different prompt (typically much longer) which is then submitted to the original LLM
- A popular example of prompt engineering technique is *Retrieval-Augmented Generation* (RAG) – the original prompted is augmented with supporting information retrieved from a database
- Example:
 1. **Initial Query** (before augmentation): “What are the main features of transformers in NLP?”
 2. **Document Retrieval:**
 - **Snippet 1:** “Transformers in NLP use a self-attention mechanism that allows the model to weigh the importance of different words in a sentence, regardless of their distance from each other.”
 - **Snippet 2:** “A key feature of transformers is their ability to process input in parallel rather than sequentially, which leads to faster training times.”
 - **Snippet 3:** “The transformer architecture has revolutionized NLP tasks like machine translation, text generation, and question answering, surpassing older models like RNNs and LSTMs.”
 3. **Final Query Submitted to the LLM:** “Using the self-attention mechanism, transformers in NLP models can weigh the importance of words in a sentence, regardless of distance, allowing for parallel processing of input. This architecture is faster to train and has revolutionized tasks like machine translation and text generation. Can you summarize the main features of transformers in NLP?”

Processing batches of B queries (N_i tokens/query)

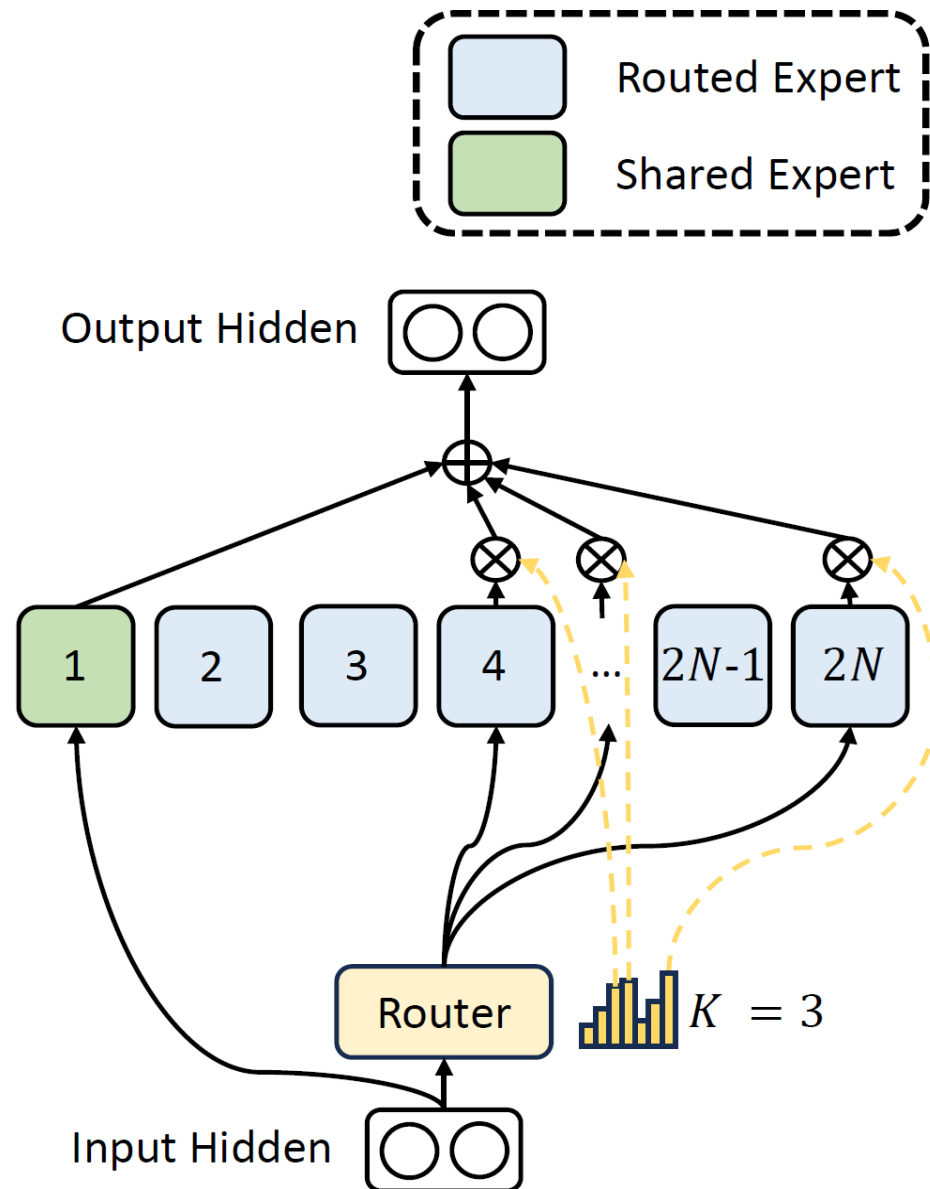
- A token is a 4544-element vector
- $n = B\bar{N}$ for the first pass, B for additional passes
- $m_i = N_i$ for first pass, increments by 1 each pass
- $n_i = N_i$ for first pass, 1 for additional passes



When to batch?

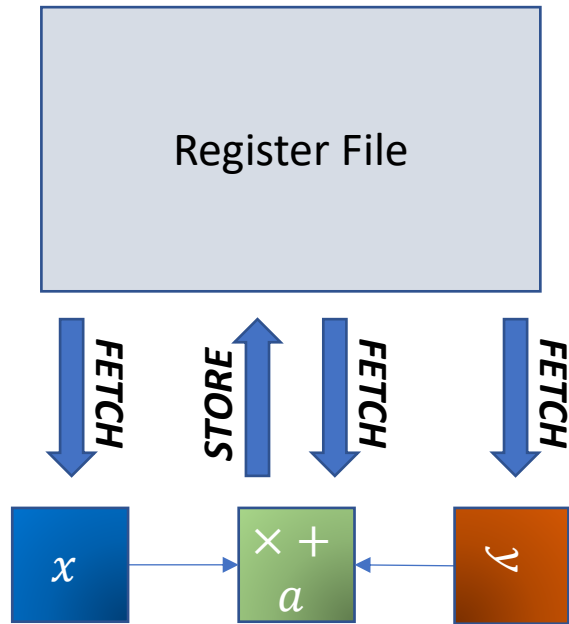
- Yes, batching transforms matrix-vector multiplication into matrix-matrix multiplication
- And it transforms matrix-matrix multiplication into *larger* matrix-matrix multiplication
- It is commonly used in benchmarking of LLMs
- Very relevant (even essential) to training – DeepSeek V3 uses batches of up to 15360
- The Intel Gaudi 3 presentation at HotChips 2024 mentions inference batch sizes of several hundreds (<https://www.servethehome.com/intel-gaudi-3-for-ai-training-and-inference/>)
- Personally, I don't quite understand how practical it is for inferencing in most cases ...
 - It does not make any one query go faster
 - It makes them more efficient and increases the throughput of the LLM, which reduces its cost – very important for large providers to offer a cost-effective solution
 - To batch, we must first collect various queries, which means most of them will wait ...
 - Furthermore, different queries will have different input/output lengths, so the batching will degrade
- *Continuous batching* is becoming popular to manage batching
- Is there some concept of smart batching, which groups similar *content* queries together and ideally uses the same retrieval augmentation for all of them?

Mixture of experts pulls it away from bigger GEMM

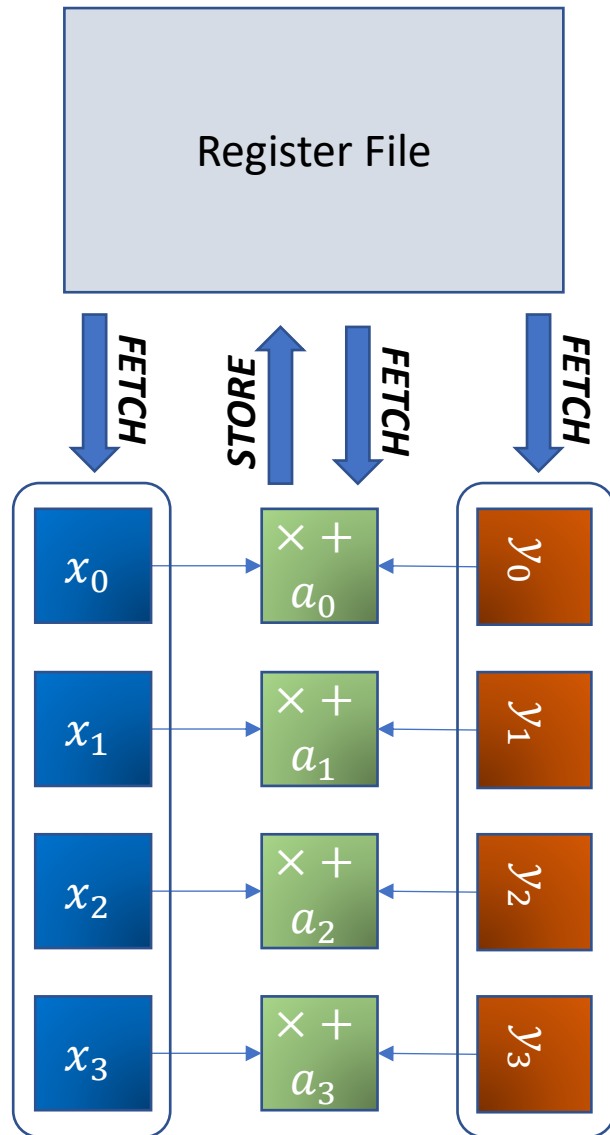


- From the paper “DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models”, by Dai et al. (<https://arxiv.org/abs/2401.06066>)
- Different tokens go through different subsets of the weight matrices, controlled by the “Router” in each layer – makes more difficult for each weight matrix to “receive” a group of tokens
- Not as bad as it looks because of affinity between tokens in a query, but it still move towards smaller GEMMs and requires dynamic packing

Scalar computing (1 operation/element)



Vector computing (n operations/ n elements)



Matrix computing (n^2 operations/ n elements)



- **BLAS-2 rank-1 update (GER):** $A \leftarrow x \otimes y + A$
- **One-dimensional input vectors** from main register file
- **Two-dimensional accumulator** resides local to unit
- **Reduced-precision data types**, perform multiple updates: $A \leftarrow \sum X^i \otimes Y_i + A$

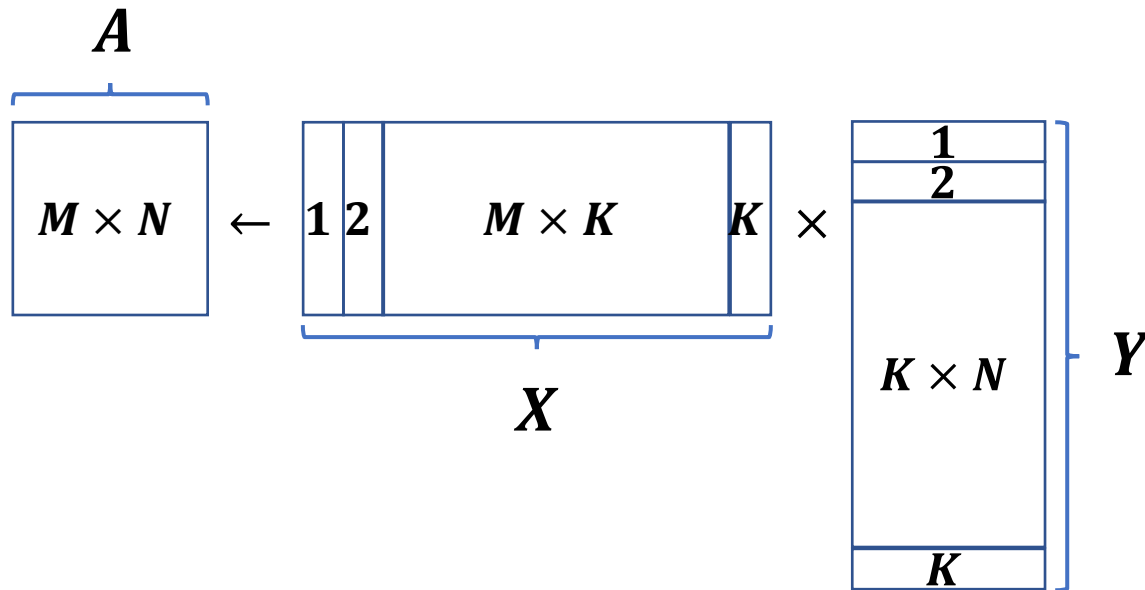
Rank-k update can be exploited by a variety of computation kernels

- **BLAS-3 and BLAS-2**
- **Sparse matrix-operations** (SpMM, Cholesky factorization)
- **Direct convolution**
- **Discrete-Fourier Transform**
- **Stencil computation**

Matrix processing is inherently more power efficient



Matrix-multiply with outer products



- We want to compute

$$A = X \times Y$$

- Textbook definition

$$A_i^j = \sum_{k \in K} X_i^k \cdot Y_k^j$$

- This can be rewritten as

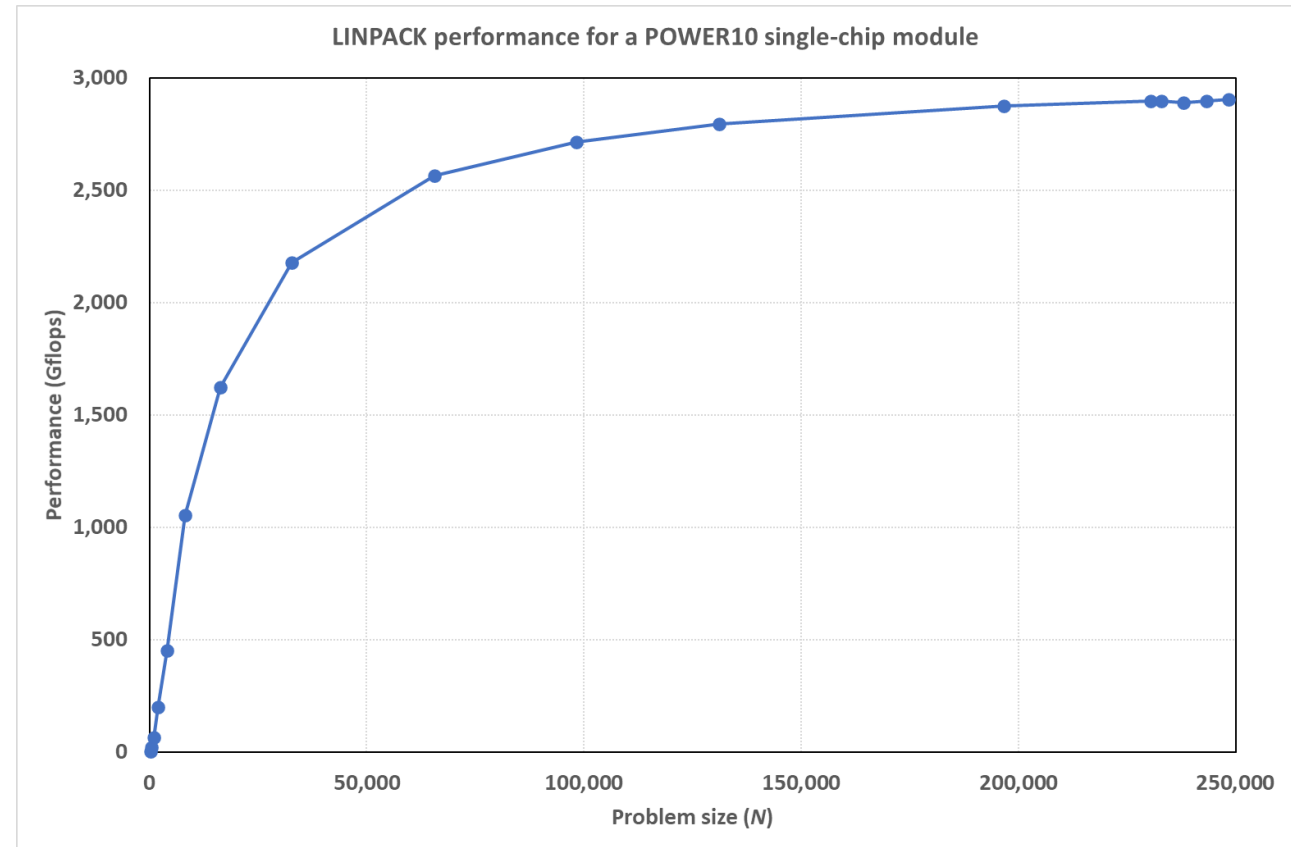
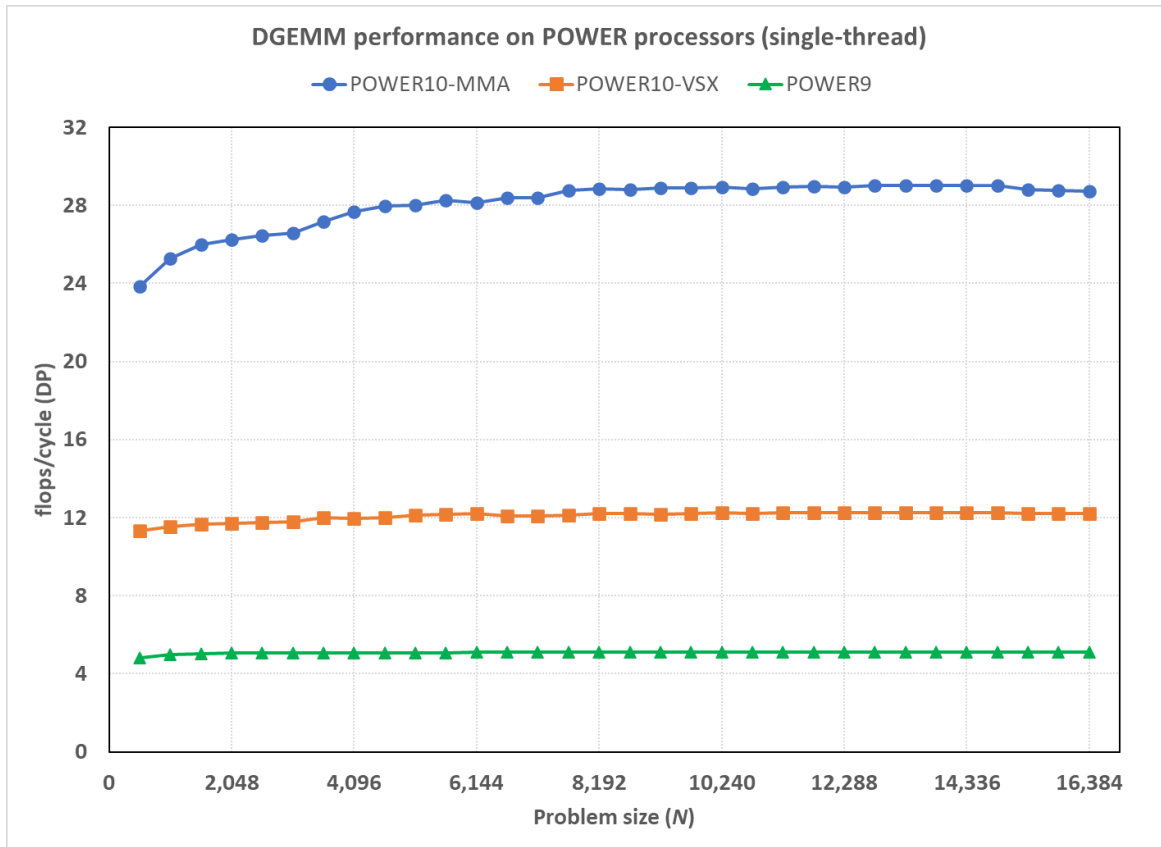
$$A = \sum_{k \in K} X^k \otimes Y_k$$

where

X^k is the k -th column of X ,

Y_k is the k -th row of Y

DGEMM and LINPACK performance in POWER10

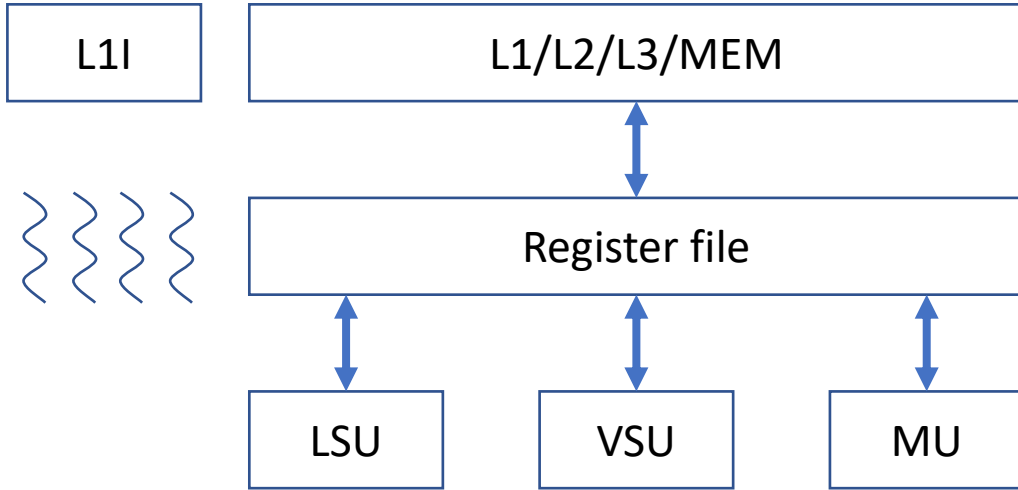


- Linpack is the classical benchmark for dense matrix computations
- Give matrix A and vector b , solve $Ax = b$ for vector x
- Linpack on a POWER10 single-chip approaches 3 Teraflops, or 85% of peak

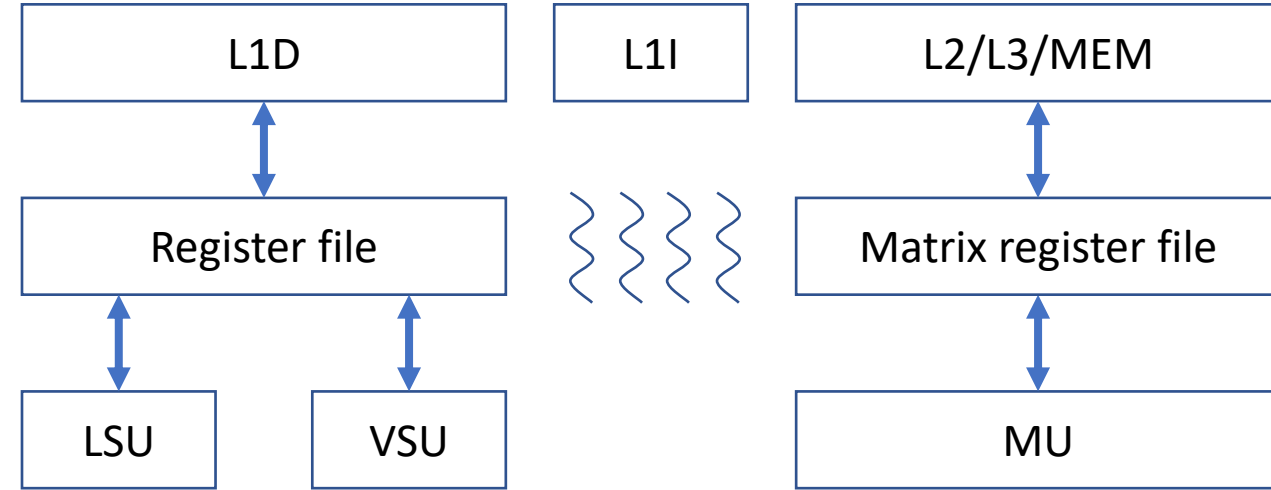
```
LAPACKE_dgetrf(LAPACK_COL_MAJOR, n, n, A, ld, piv);  
LAPACKE_dgetrs(LAPACK_COL_MAJOR, 'N', n, 1, A, ld, piv, x, n);
```


Four types of matrix units (not exhaustive)

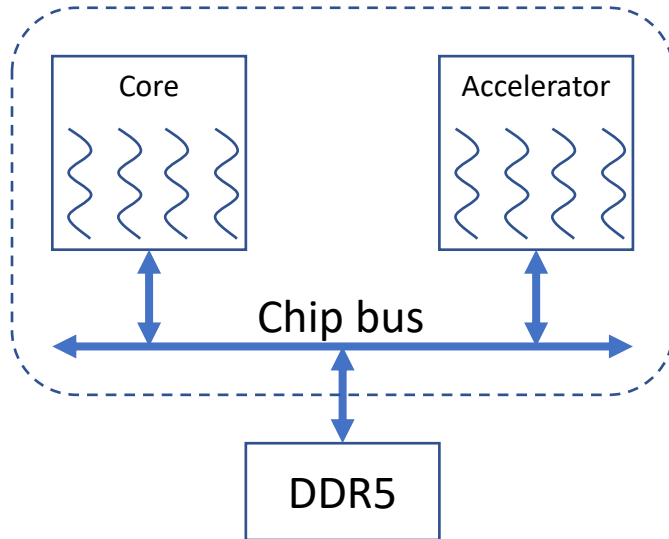
Integral to the core



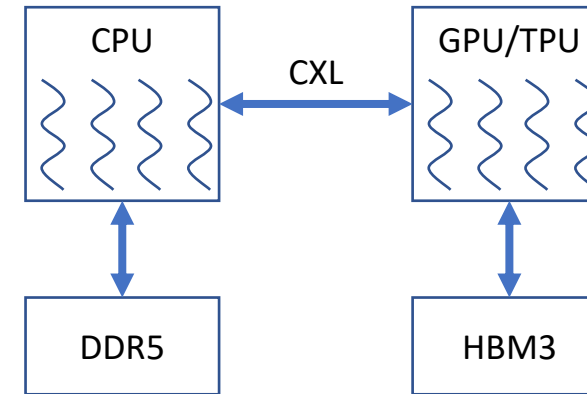
Attached functional unit



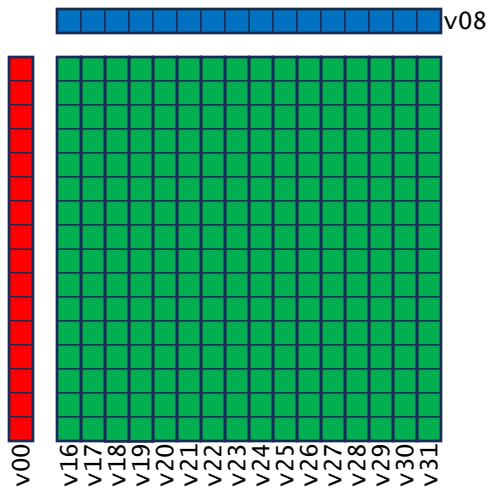
Coprocessor



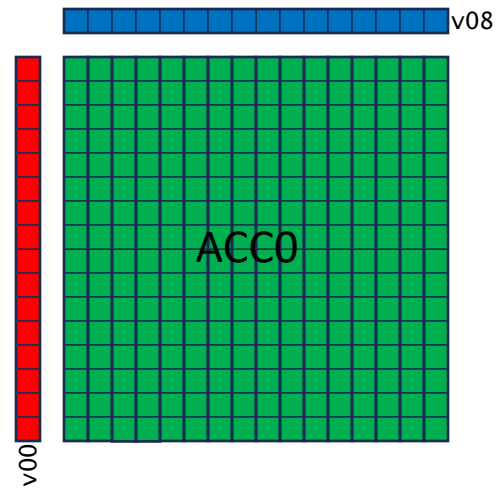
External



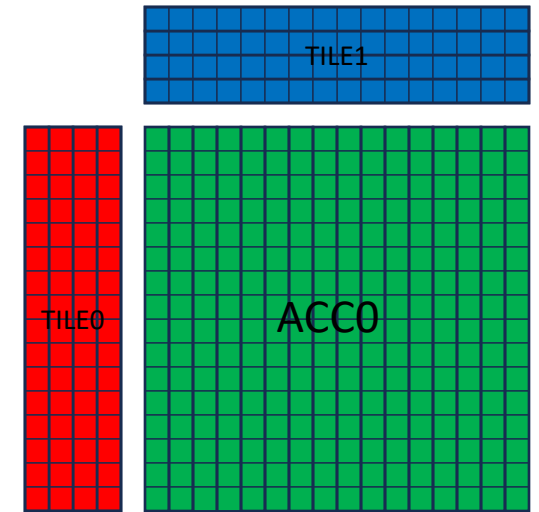
Matrix processing in RISC-V ISA (in development)



IME – sources and accumulators are conventional vector registers



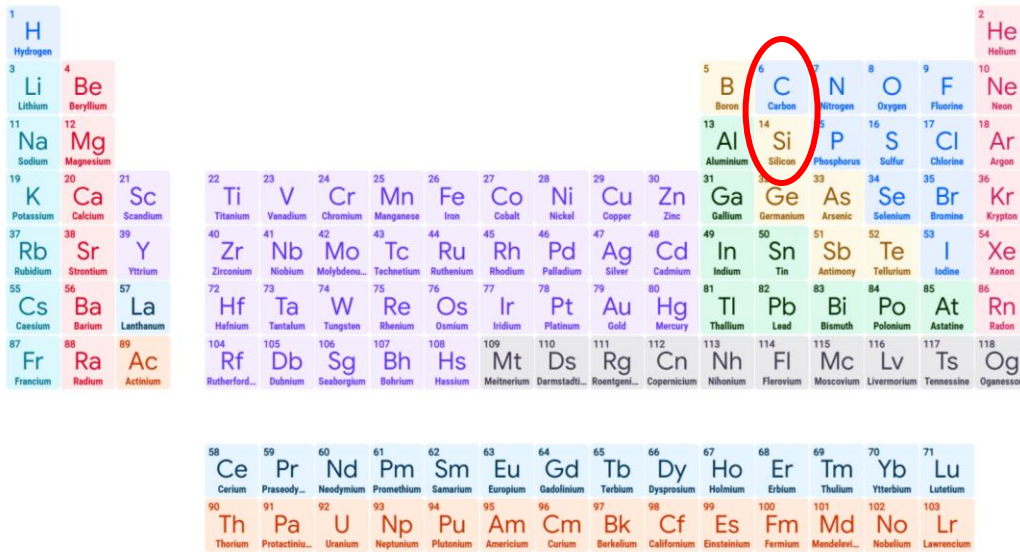
VME – sources are conventional vector registers, accumulators are new registers



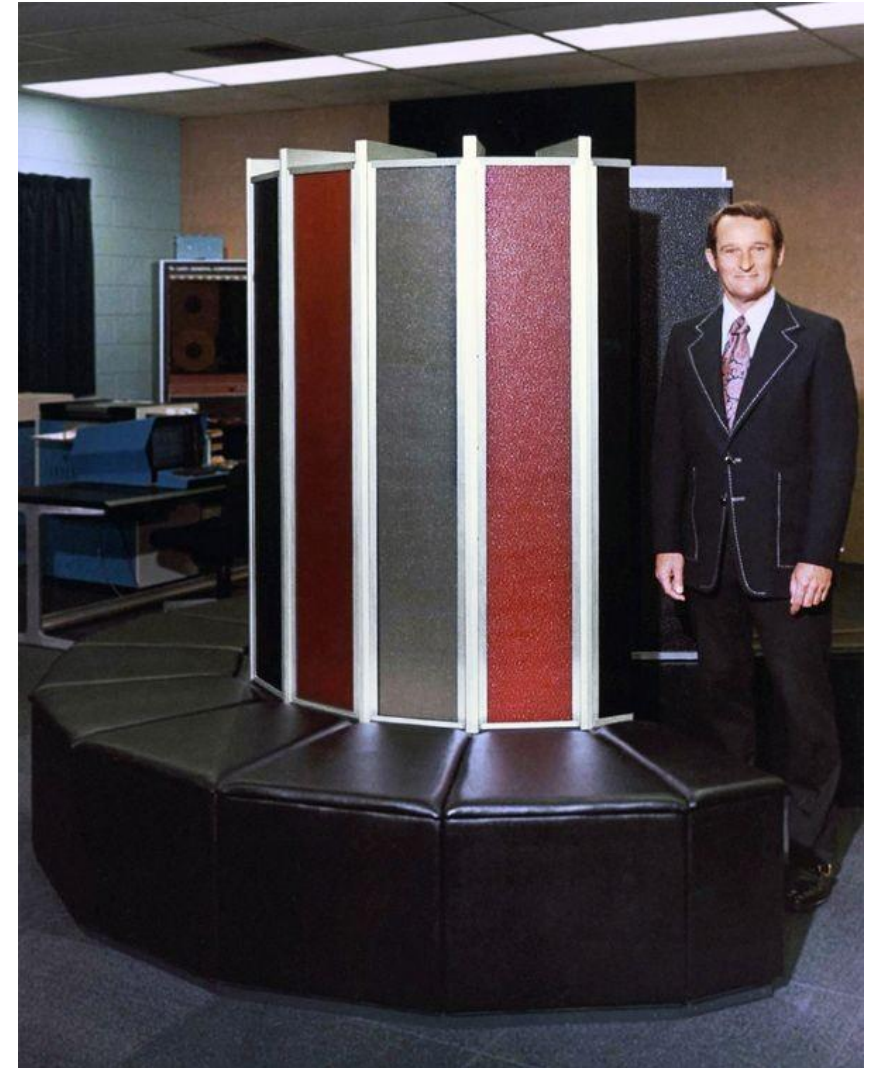
AME – source tiles and accumulator tiles are all new registers

Scalar, vector, matrix: a happy family

- Who recognizes these two?

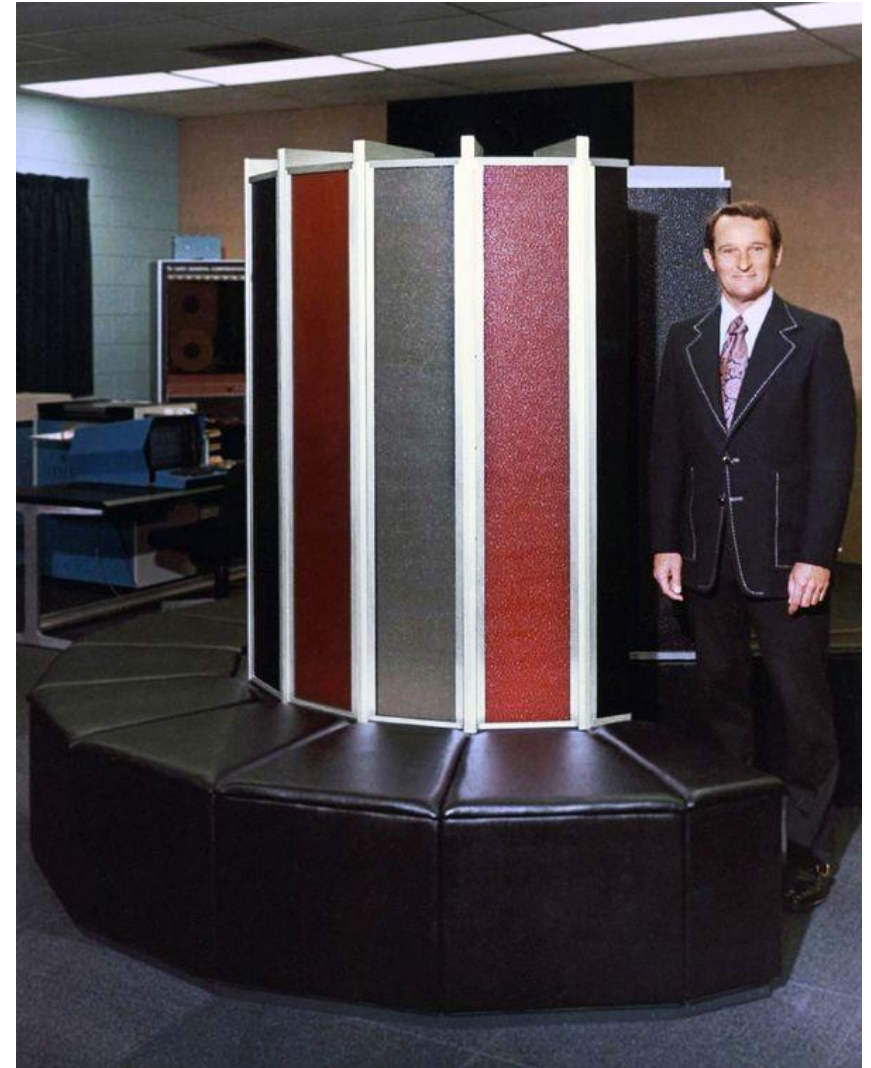


The image shows a standard periodic table of elements. The elements Carbon (C) and Silicon (Si) are circled in red. Carbon is located in the second row, sixth column (atomic number 6). Silicon is located in the third row, tenth column (atomic number 14). The table includes elements from Hydrogen (1) to Oganesson (118), with the Lanthanide and Actinide series shown separately at the bottom.

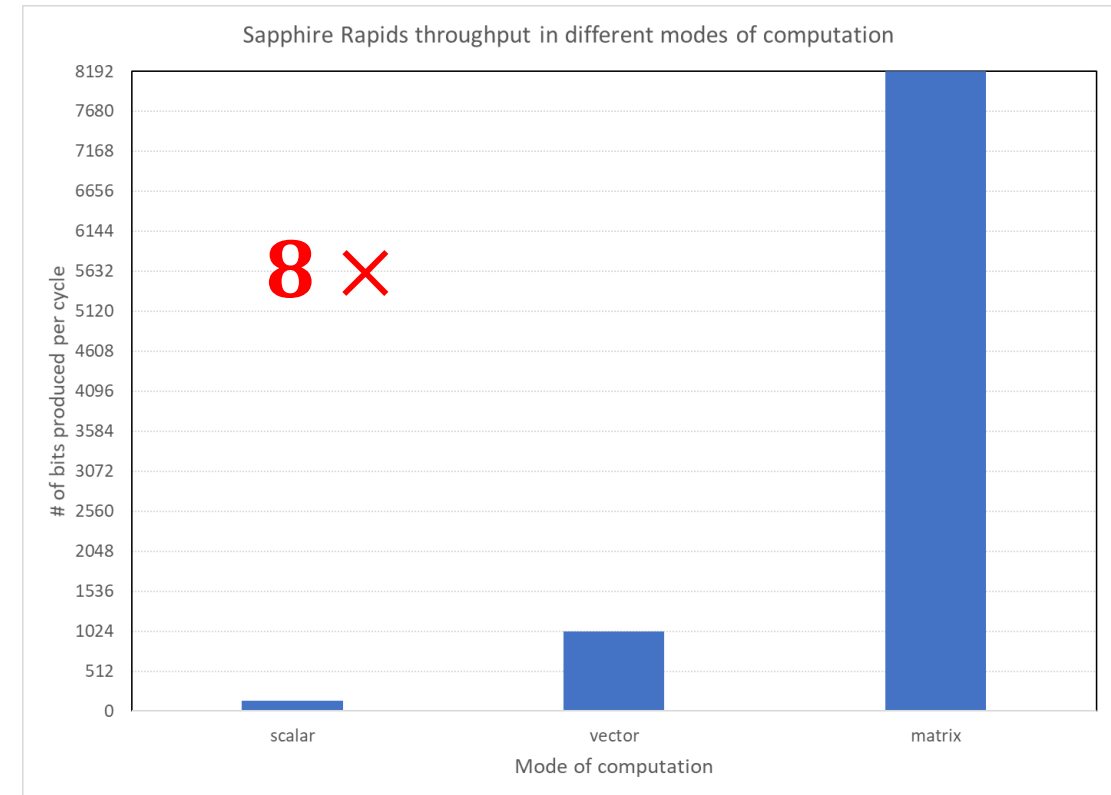
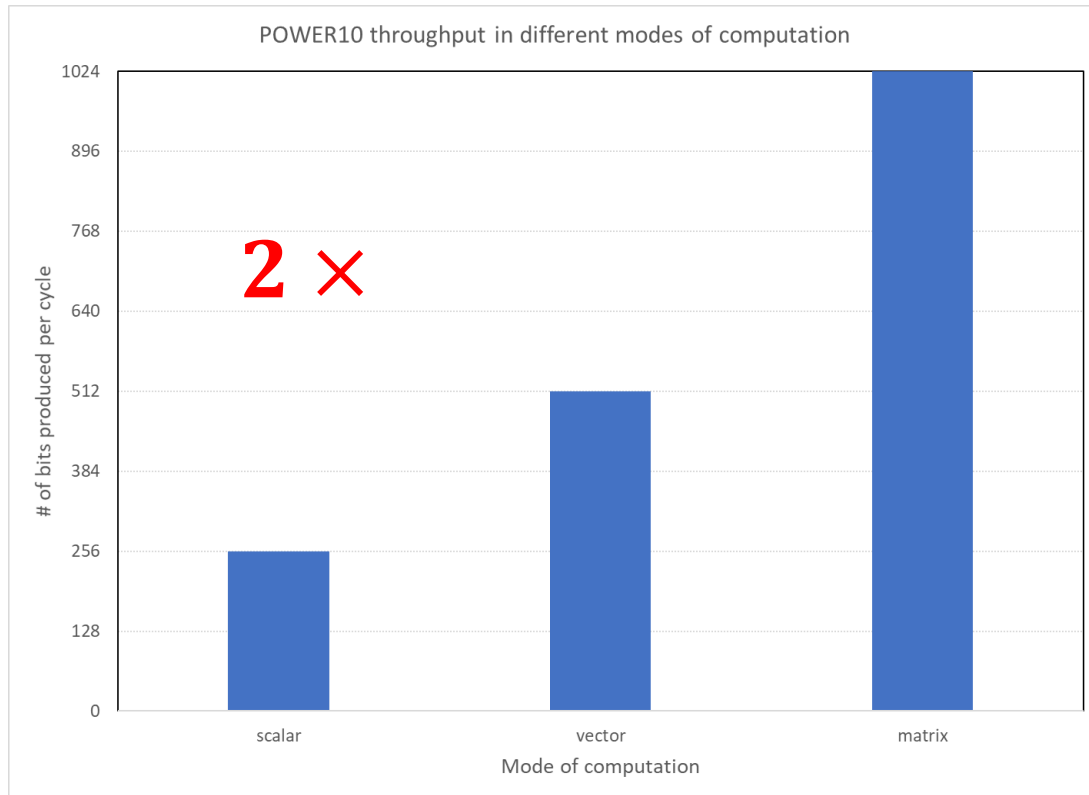


Scalar, vector, matrix: a happy family

- Who recognizes these two?
- The Cray-1 (Silicon-based) and Seymour Cray (Carbon-based)
- The very definition of “supercomputer” in 1975
 - 80 MHz
 - 160 Mflops (DP)
 - 8 MiB main memory
 - 64-element vectors
- The Cray-1 was the most successful supercomputer of its time, and not because it was the fastest vector processor (it wasn't), but because
 - It was the fastest scalar processor
 - It was much more efficient on short vectors
 - It was competitive for very long vectors

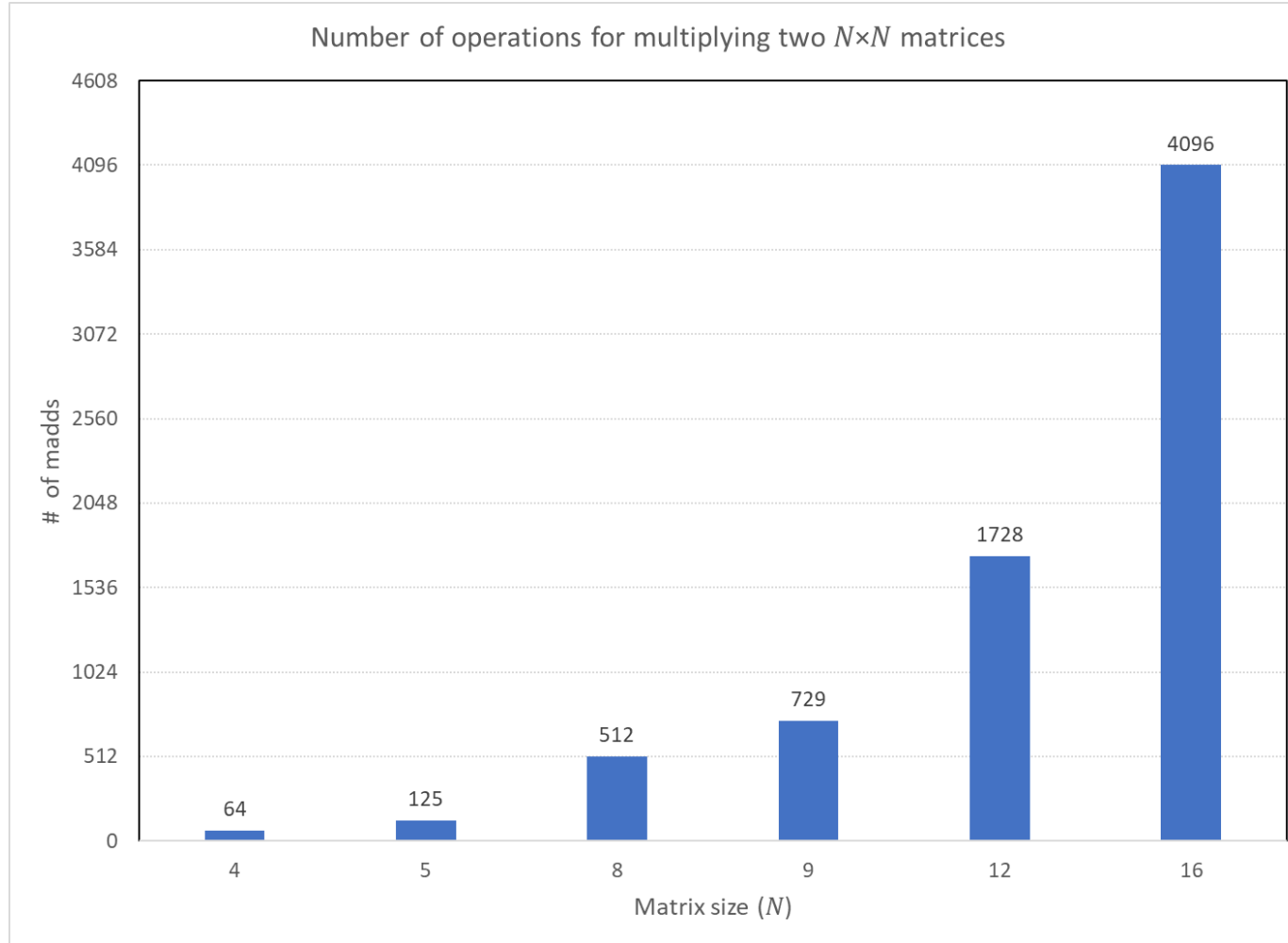


Matrix units will complement scalar and vector units



- You need some differentiation to make the loss of generality worth it
- You also need to continue to support high-performance for scalar and vector processing
- I suspect a **4x** increase from scalar to vector and from vector to matrix may be the right balance

Keep small matrices in mind ...



- If you need to pad with zeros to match the size of your matrix unit, you can quickly lose any performance gains
- For example:
 - 5×5 vs 8×8 GEMM : 24% of operations
 - 9×9 vs 16×16 GEMM: 18% of operations
- Finer grain matrix units can really help with small matrix computations – particularly important for sparse matrices, where finding small blocks is much easier than finding big blocks
- Beware of load/store or packing, which can quickly become a bottleneck

Conclusions

- The vast majority (95+%) of computations in LLM processing consist of matrix math – either GEMM (matrix-matrix multiplication) or GEMV (matrix-vector multiplication)
- The breakdown between GEMM and GEMV depends on the scenario
 - Long inputs and short outputs (e.g., summarization) favor GEMM
 - Short inputs and long outputs (e.g., chat bots) favor GEMV
- GEMM is the computer architect's best friend – New chips with matrix units can achieve Teraflops/Petaflops of performance in GEMM
- After 50 years of pursuing performance through vector processing, we are ready to move to the next level – matrix processing through dedicated units
- Matrix processing will not replace vector processing, just like vector processing did not replace scalar processing – CPUs will have to deliver performance on all three
- I am looking forward to more sparsity support in matrix units (some already there)
- Maybe we will see operations beyond the arithmetic semiring (see GraphBLAS)

Acknowledgements

- I want to thank my many current and former colleagues at IBM that made this work possible – it took ~1000 people to realize POWER10, the first mainstream CPU with a matrix unit
- Many special thanks to my colleague Eknath Ekanadham and manager Pratap Pattnaik, who both helped me tremendously during my 30 years at IBM
- Thanks to Georg Hager for inviting me to give this talk and having the patience to deal with me
- And last but not least, many thanks to the audience for enduring this talk!