Llama.cpp GPU Acceleration

Johannes Gäßler

About Me

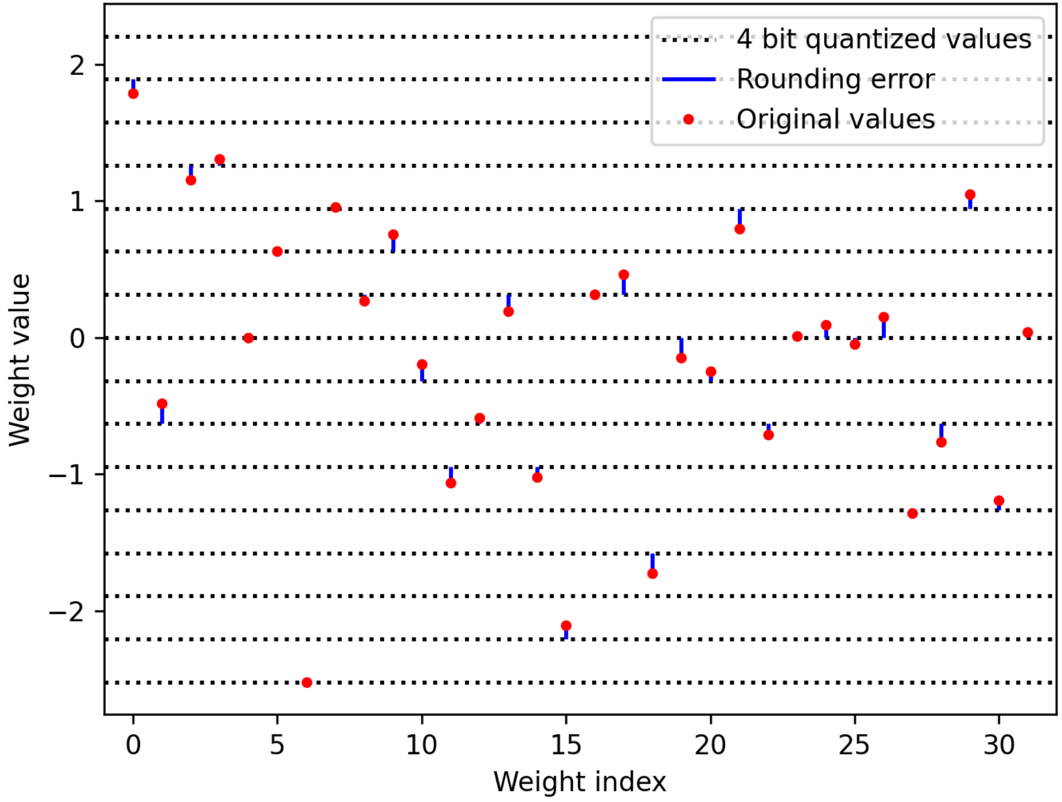
- Currently writing my master’s thesis on experimental particle physics at KIT
- Additional bachelor’s degree in informatics
- kafe2 (Karlsruhe Fit Environment 2) developer
- No prior experience with CUDA or language models

llama.cpp

- Open-source C/C++ program for LlaMA inference
- Wide support across hardware and OSs
- Very good CPU performance
- I’m working on CUDA support

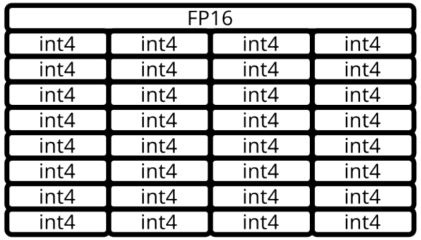
Weight Quantization

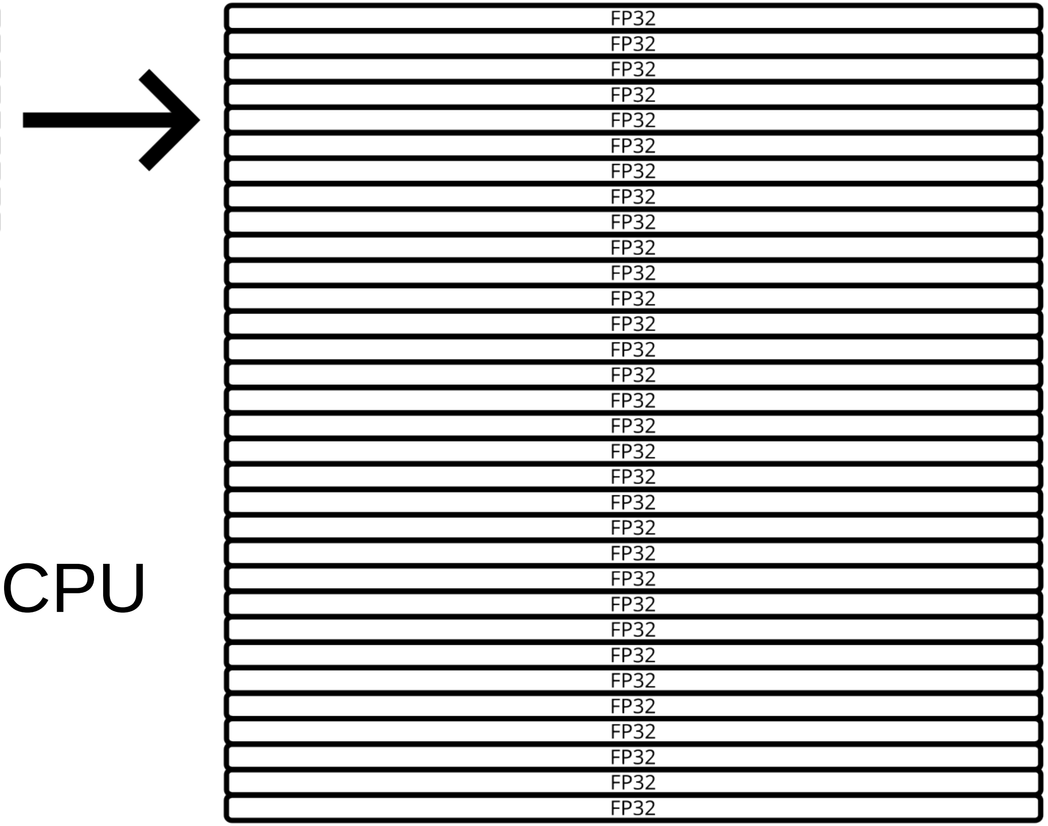
- Original LlaMa weights are FP16
- Can be quantized to 4 bit ints with moderate quality loss
- int4 big model > FP16 small model



Initial CUDA Implementation

- Dequantize weights to FP16/FP32, then use cuBLAS GEMM





- Performance only good for large batches
- Small batches: slower than CPU

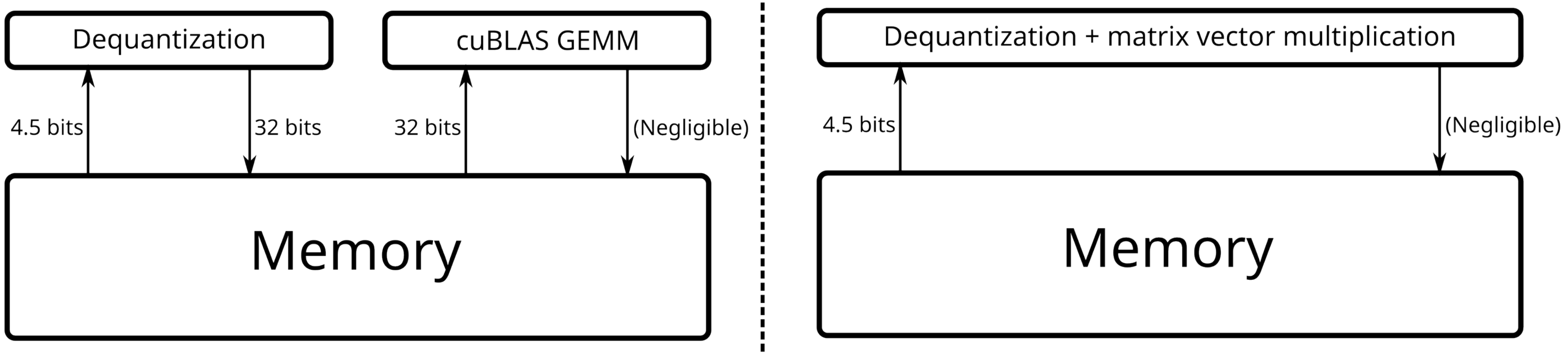
Initial CUDA Implementation

- Matrix mult. I/O vs. compute depends on shape
- 2x square matrix: $O(N^2)$ data, $O(N^3)$ compute
- Square matrix + vector: $O(N^2)$ data, $O(N^2)$ compute
- Prompt processing: compute bound
- Token generation: I/O bound

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

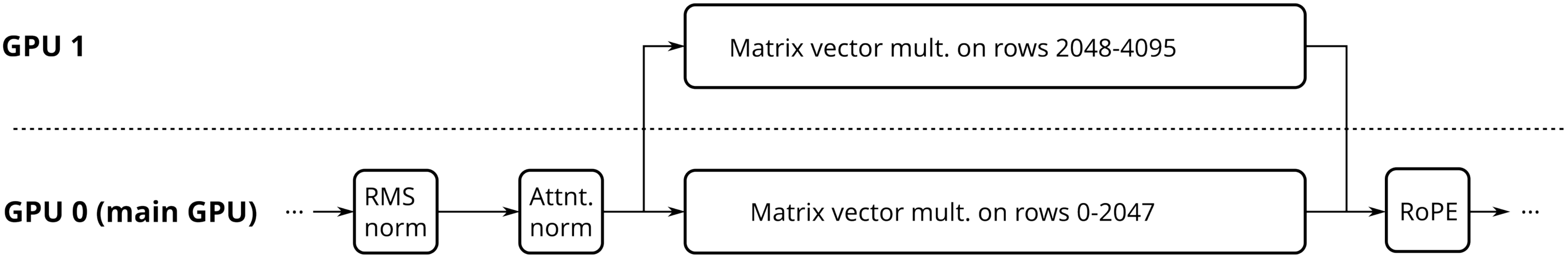
Better CUDA Implementation

- Matrix vector multiplication + on-the-fly dequantization
- Per weight:
  36.5 bits read + 32 bits write => 4.5 bits read



Better CUDA Implementation

- Multi GPU: split weight matrices across GPUs by rows
- Small tensors on main GPU only
- KV cache parallelization not implemented



Current CUDA Implementation

- Don’t dequantize matrix
- Instead quantize hidden state to 8 bit
- Use per-byte integer intrinsics (similar to CPU)

Current CUDA Implementation

- N blocks with 1 scale d and M values q_m each
- Dequantization: $a_{inm} = d^a_{in} q^a_{inm}, \quad b_{nm} = d^b_n q^b_{nm}$

$$c_i = \sum_{n=1}^{N} \sum_{m=1}^{M} a_{inm} b_{nm} = \sum_{n=1}^{N} \sum_{m=1}^{M} d^a_{in} q^a_{inm} d^b_n q^b_{nm}$$

$$= \sum_{n=1}^{N} d^a_{in} d^b_n \sum_{m=1}^{M} q^a_{inm} q^b_{nm}.$$

Accessibility

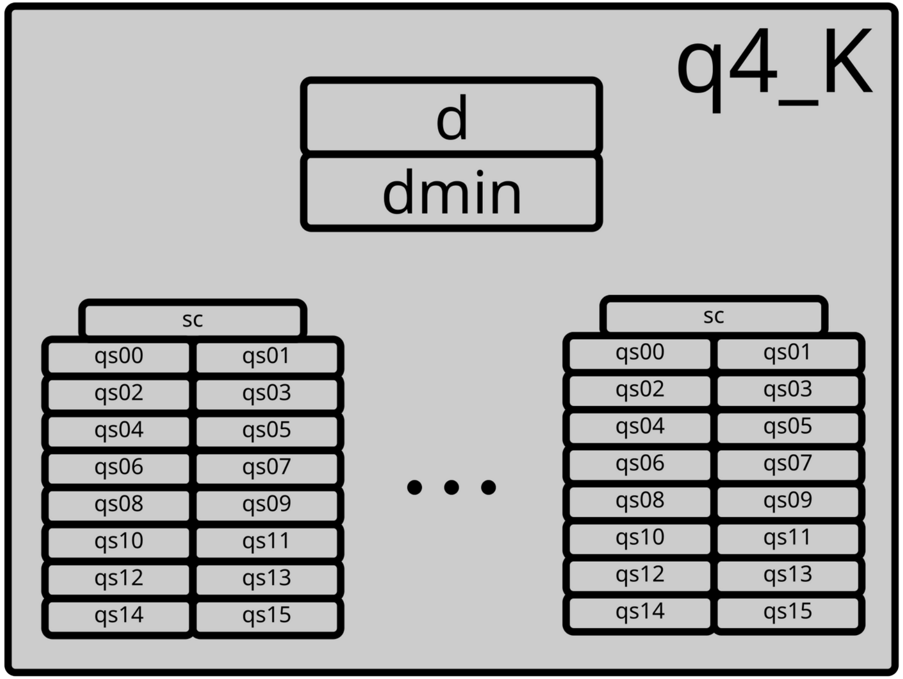
- Open-source is a positive sum game
- Jevons Paradox: more efficiency => more use
- Main goal: reduce hardware requirements, get more users

Accessibility: Memory

- Move weights from RAM to VRAM
- Can use total RAM + VRAM to fit model
- Goal: 33b q4 with standard settings on 16 GB RAM + 8 GB VRAM

Accessibility: Quantization

- Hierachical k-quants by I. Kawrakow
- 1 6 bit scale per 16 values
- 2 16 bit scales per 256 values
- Lower size/quantization error at the cost of more memory accesses



Comparison to GPTQ

LLaMA vs. LLaMA 2

WIP: Quantized GEMM