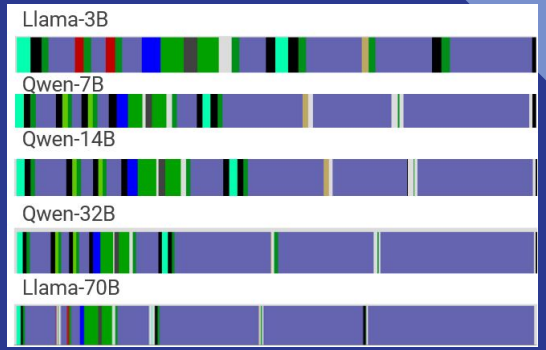
AI (LLM's) are everywhere!

- But how much Production Know-How do we have…?
- What are their TCO…?
- How to design a system given expected requirements?
  - Model type
  - Model size
  - # User requests
  - Response times
  - Etc

LLM Model Size Impact on Application Performance



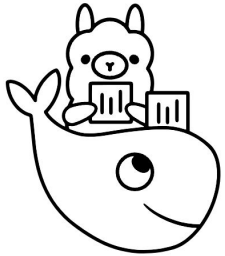
Juan Jose Olivera Loyola

PPTM Final Project - Professor Jesus Jose Labarta
TMIRI-HPC

Goals

1. Understand intrinsic nature of hardware utilization of LLM applications.
    a. Computation Structure, Compute vs Memory Boundness, Characteristic Values
2. Be able to derive characteristic numbers of LLM computation in llama.cpp
    a. Formulas of token gen/sec, ratios, times, etc.
3. Have enough information to be able to compare other models or other applications/implementations

What is llama.cpp (the profiled application)?



Ollama: A LLM Containerization and Inference Platform

…. Docker for LLM’s

llama.cpp: Inference engine behind Ollama

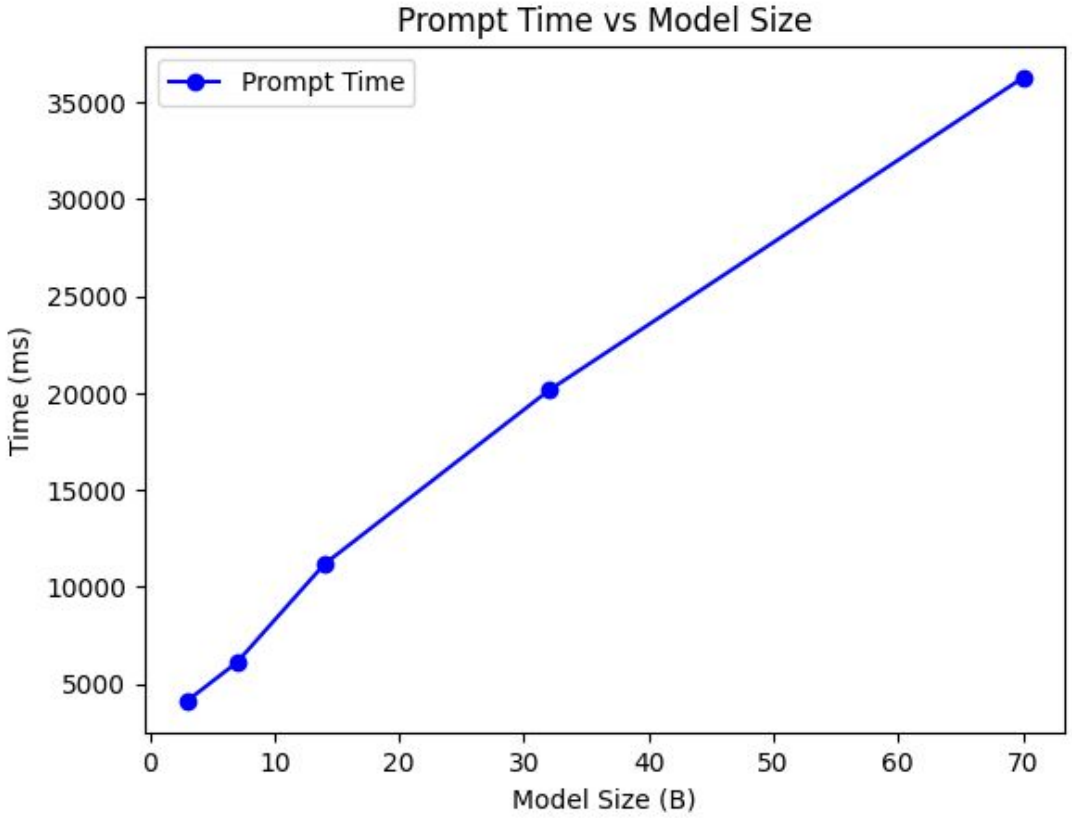
- Created in February 2023
- Spinoff of GGML a matrix multiplication library
- Creator of the common **GGUF model format**.
- Referenced even by Nvidia Technical Blog:
  Accelerating LLMs with llama.cpp on NVIDIA RTX Systems | NVIDIA Technical Blog

LLM Models

Model Name (GGUFs)	# Layers	Quantization (avg. bits per weight)	(GB)
Llama 3B-Q4_K_M	28	4.8 bits	2GB
Qwen 7B-Q4_K_L	28	5.1 bits	5GB
Qwen 14B-Q4_K_L	48	5.1 bits	9.5GB
Qwen 32B-Q4_K_L	64	5.1 bits	20.4GB
Llama 70B-Q4_K_M	80	4.8 bits	42GB

Family of GPT (Generative Pre-Trained **Transformer**) Models

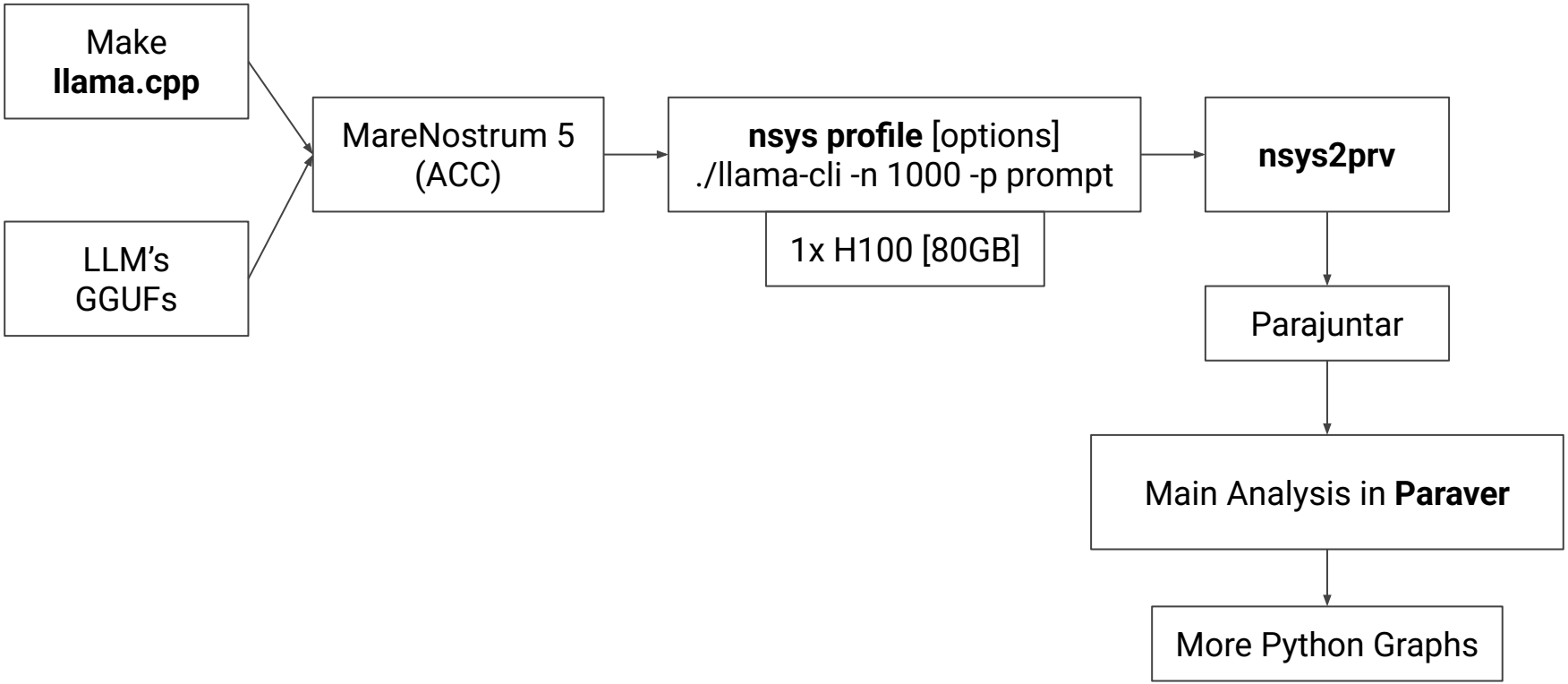
Total Prompt Execution Time (1000 tokens)



AI and GPT's use heavily Matrix Multiplications ~O(N^2.37) to O(N^3)

...Seems to scale linearly with 1 GPU

Project Tools & Methodology



Slurm Compilation Script

```
export SRUN_CPUS_PER_TASK=${SLURM_CPUS_PER_TASK}

module load cmake
module load nvidia-hpc-sdk/25.1
module load binutils

cd "$PWD/llama.cpp"
rm -rf build
cmake -DCMAKE_C_COMPILER=gcc -DLLAMA_AVX_VNNI=ON -DCMAKE_CXX_FLAGS="-march=native" -DCMAKE_CXX_COMPILER=g++ -B build
-DGGML_CUDA=ON -DCMAKE_CUDA_ARCHITECTURES="90;90;90;90"

cmake --build build --config Release -j 80
```

Slurm Profile Script

```
#!/bin/sh
#SBATCH --job-name=profile_llamacpp                                    #SBATCH -D .
#SBATCH --output=sjobs/profile_llamacpp_%j.out                         #SBATCH
--error=sjobs/profile_llamacpp_%j.error                                #SBATCH --time=00:20:00
#SBATCH --nodes=1                                                      #SBATCH --ntasks=80
#SBATCH --ntasks-per-node=80                                           #SBATCH --cpus-per-task=1
#SBATCH --constraint=perfparanoid                                      #SBATCH --gres=gpu:4
#SBATCH --qos=acc_debug                                                #SBATCH -A nct_323
# Model loaded from llms/
modelFile="DeepSeek-R1-Distill-Llama-3B-Q4_K_M.gguf"                   # For skipping program setup
and avoid large trace useless traces                                   traceOffset=0
inputPrompt="Explain, in at least 800 words, the usage of HPC on addressing modern world problems and its impact on economy"
nTokens=1000                                                           # number of model layers to be offset (run) in the
GPU                                                                    modelGpuLayers=10000
# CPU metrics/hw counters sampling frequency (Hz)                      cpuSamplingFreq=231250
# GPU metrics/hw counters sampling frequency (Hz)                      gpuSamplingFreq=10000
nsysFileName="run-$modelFile-$nTokens.nsys-rep"
baseFileName="run-$modelFile-$nTokens"
export SRUN_CPUS_PER_TASK=${SLURM_CPUS_PER_TASK}

module load cuda/12.6 nvidia-hpc-sdk/25.1
```

Slurm Profile Script

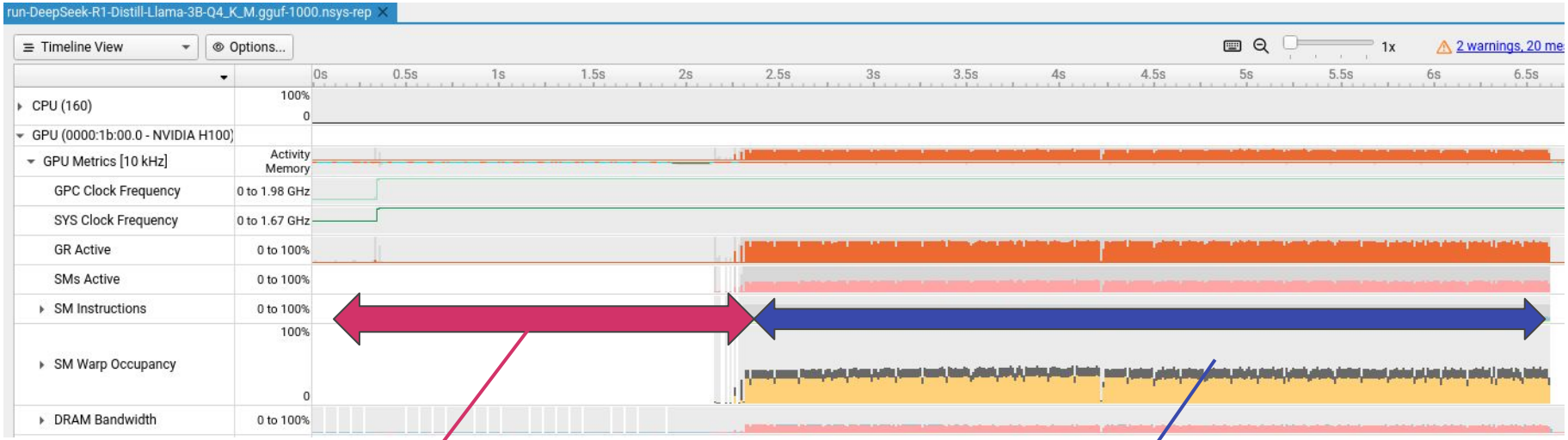
```
CUDA_VISIBLE_DEVICES=0 nsys profile \
    --output "experiments/$baseFileName" \
    --sample=process-tree --sampling-period=$cpuSamplingFreq \
    --trace=cuda,nvtx,osrt,cublas,cudnn --cuda-graph-trace=node \
    --gpu-metrics-devices=cuda-visible --gpu-metrics-frequency=$gpuSamplingFreq \
    --cpuctxsw=process-tree \
    --delay $traceOffset \
    ./llama.cpp/build/bin/llama-cli -m "llms/$modelFile" -no-cnv \
    -n $nTokens -ngl $modelGpuLayers -p "$inputPrompt"


cd experiments
module load intel mkl python/3.12.1 sqlite3
nsys2prv -t cuda_api_trace,nvtx_pushpop_trace,graph,gpu_metrics "$nsysFileName" "$baseFileName"
tar -cvjf "$baseFileName.prv.tar.bz2" "$baseFileName.pcf" "$baseFileName.prv" "$baseFileName.row"
rm "$baseFileName.pcf" "$baseFileName.prv" "$baseFileName.row" "$baseFileName.sqlite"
```

Project Tools & Methodology

1. Generate Traces	2. Analysis of 32B Model (Identifying Structure)	3. Analysis of rest of Models (Focus of Analysis)	4. Characterizing Performance (Perf Model & KPIs)

Application Work excludes Initialization



Initialization/Model Loading

~ 35% total time for 1 1000 token prompt Llama-3B

Calls: CUDA profiling init, fopen, mmap, ioctl, cudaMalloc/cpy.

Real Work: Token Prediction

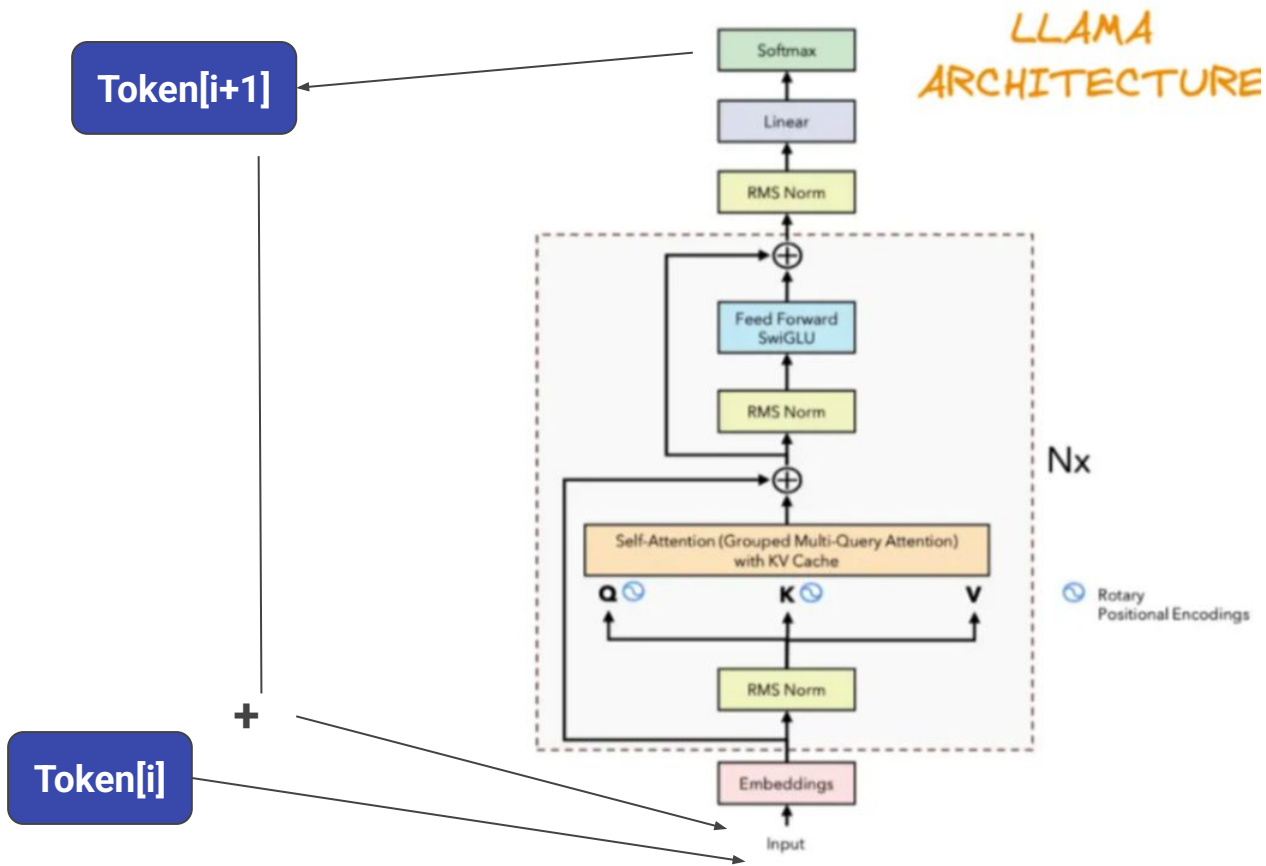
Calls: Kernels, CUDASync, memCpy, etc.

Identifying Structure

Setup Work
(Repetitive) Body Work
Termination Work

Model: **Qwen-32B**-Q4_K_L
-n **Tokens = 1000**
-p “Explain the usage of HPC on addressing modern world problems and its impact on economy” (20 - 25 tokens)

Llama Inference Process



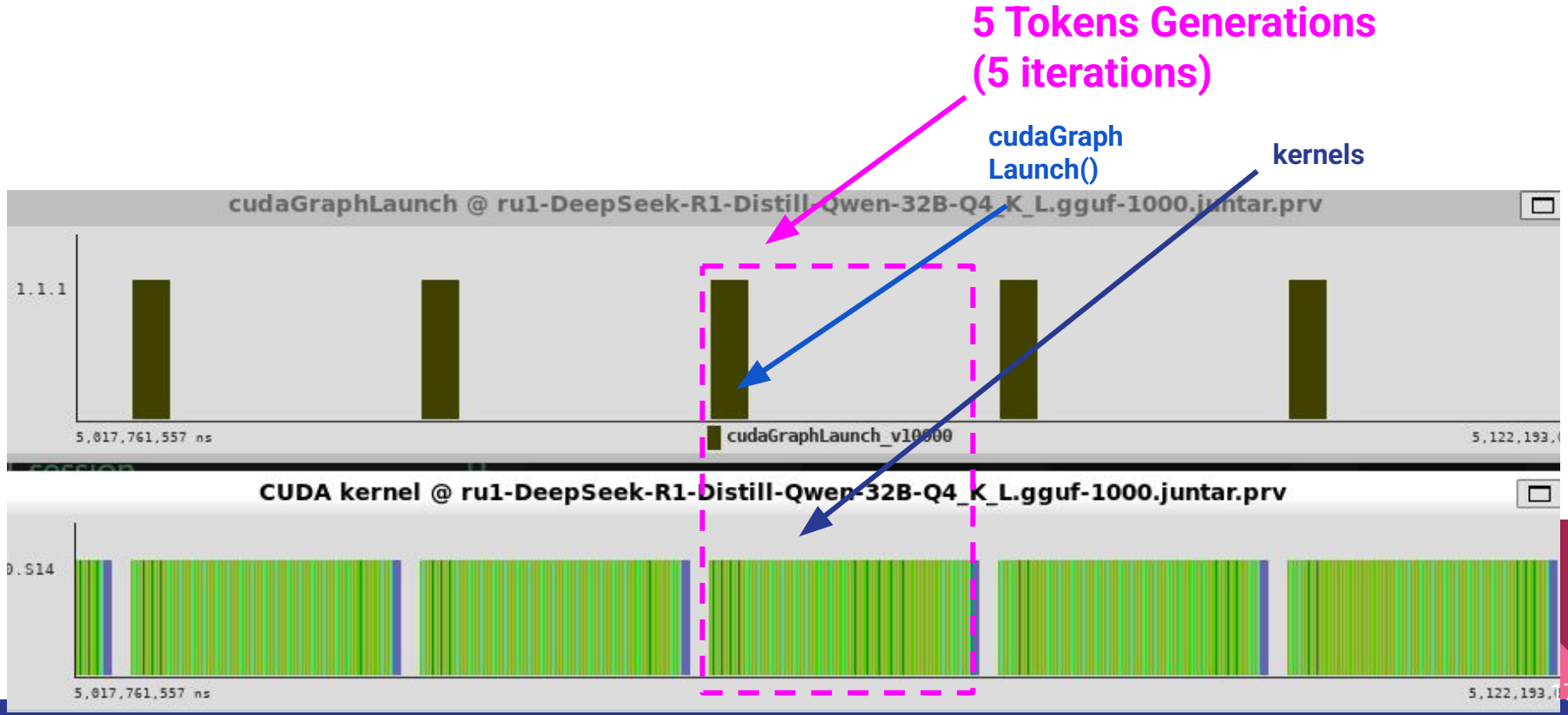
How? By finding Token Generation Structure

How? By finding Token Generation Structure

cuGraphLaunch Calls = 999 + 1 Graph Instantiation = 1000 Tokens

How? By finding Token Generation Structure

cuGraphLaunch Calls = 999 + 1 Graph Instantiation = 1000 Tokens

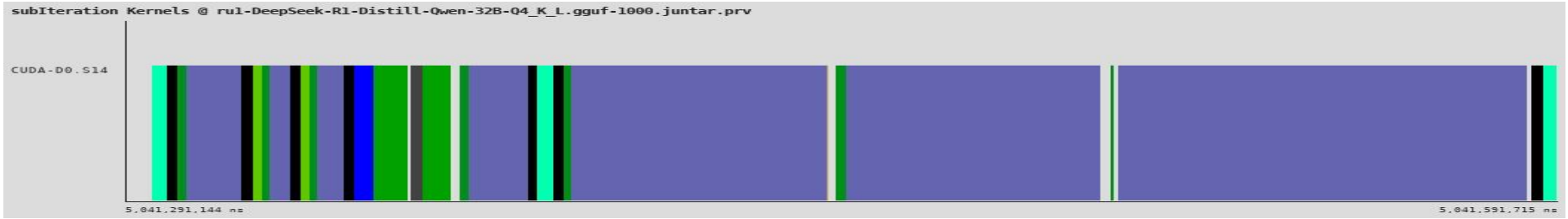


Zoom into 1 Iteration

63 normal sub iterations + 1 PostWork Subiteration

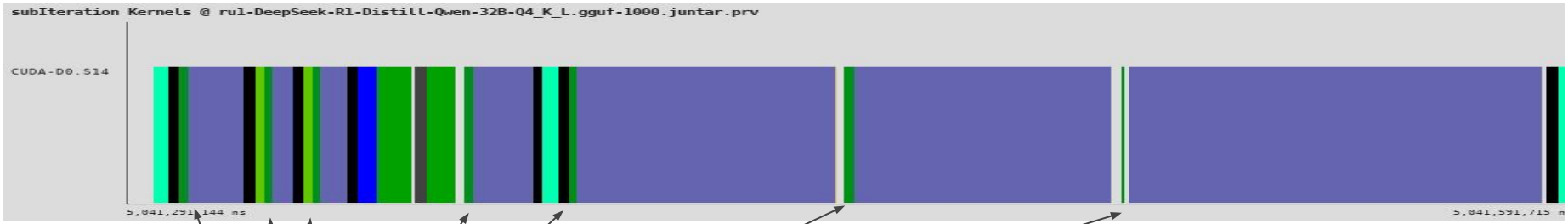
x64 soft_max_f32 = **# layers in Qwen2.5 32B**

Zoom into 1 Sub Iteration



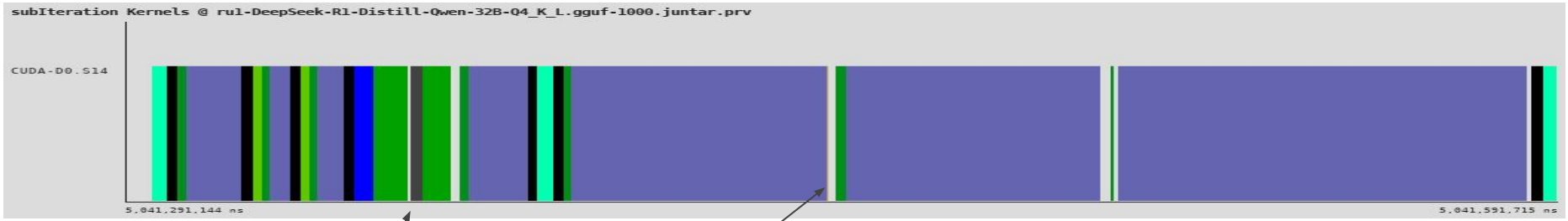
- End
- cpy_f32_f16
- k_bin_bcast
- mul_mat_vec
- mul_mat_vec_q
- quantize_q8_1
- rms_norm_f32
- soft_max_f32
- unary_op_kernel
- rope_neox

Zoom into 1 Sub Iteration



- End
- cpy_f32_f16
- k_bin_bcast
- mul_mat_vec
- mul_mat_vec_q
- quantize_q8_1
- rms_norm_f32
- soft_max_f32
- unary_op_kernel
- rope_neox

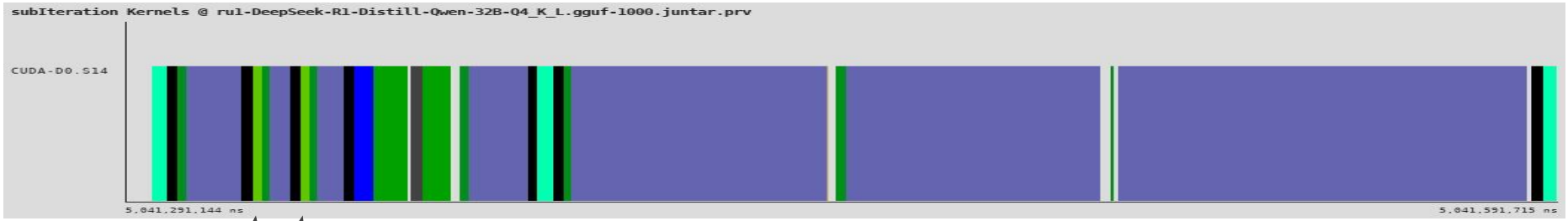
Zoom into 1 Sub Iteration



- End
- cpy_f32_f16
- k_bin_bcast
- mul_mat_vec
- mul_mat_vec_q
- quantize_q8_1
- rms_norm_f32
- soft_max_f32
- unary_op_kernel
- rope_neox

unary_op_kernel = SILU

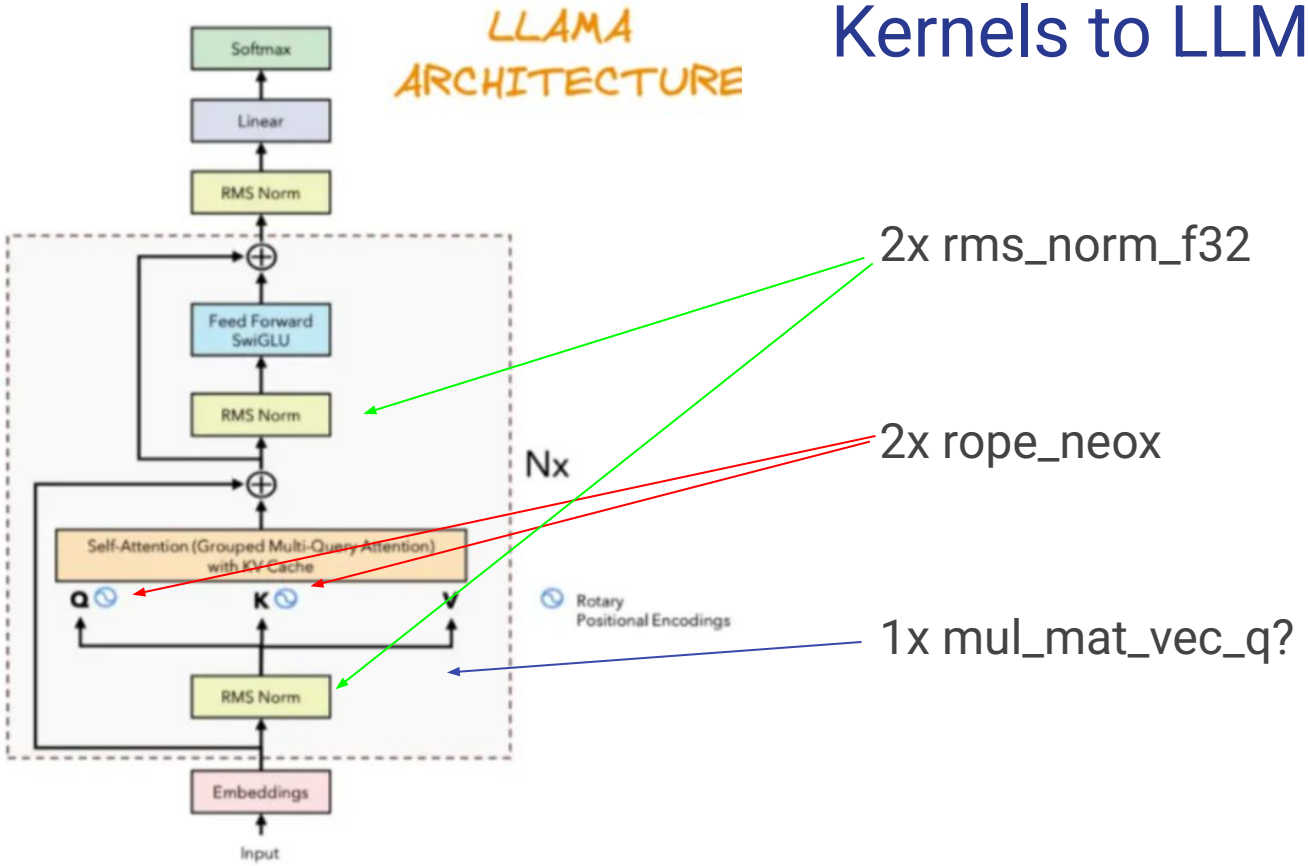
Zoom into 1 Sub Iteration



- End
- cpy_f32_f16
- k_bin_bcast
- mul_mat_vec
- mul_mat_vec_q
- quantize_q8_1
- rms_norm_f32
- soft_max_f32
- unary_op_kernel
- rope_neox

rope = Rotational Positional Embedding

Kernels to LLM Architecture



$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

1x soft_max_f32

2x (mul_mat_vec,

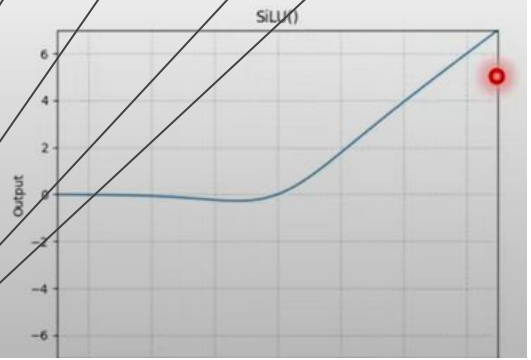
mul_mat_vec_q)

LLaMA

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

We use the swish function with $\beta = 1$. In this case it's called the **Sigmoid Linear Unit** (**SiLU**) function.

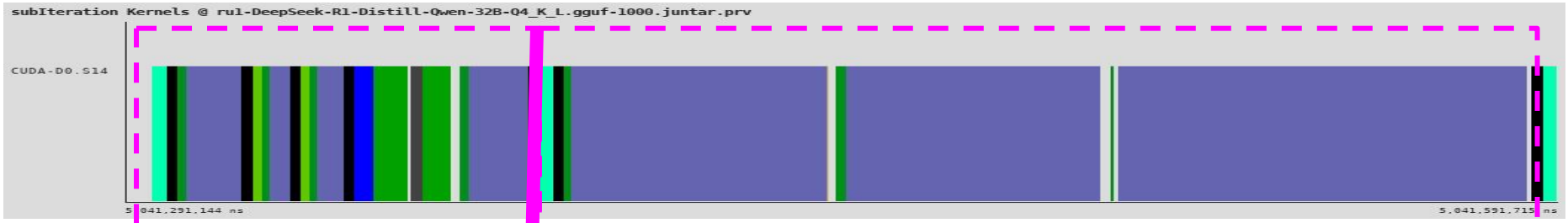
$$\mathrm{swish}(x) = x\,\mathrm{sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}}$$



1x unnary_op_kernel

3x mat_mul_vec_q

Zoom into 1 Sub Iteration



- End
- cpy_f32_f16
- k_bin_bcast
- mul_mat_vec
- mul_mat_vec_q
- quantize_q8_1
- rms_norm_f32
- soft_max_f32
- unary_op_kernel
- rope_neox

Total Time: 300 us

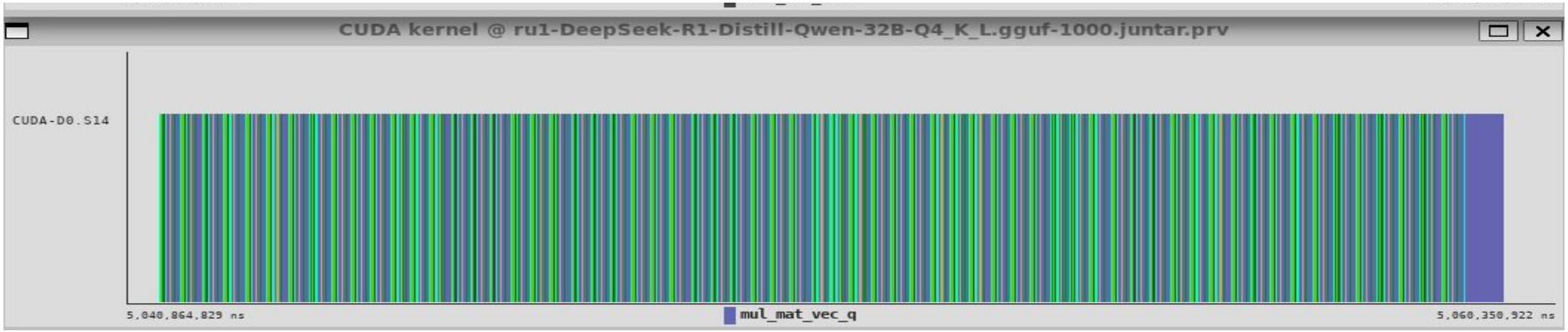
Multi-Head Attention

Time: 90 us
30%

FeedForward NN

Time: 210 us
70%

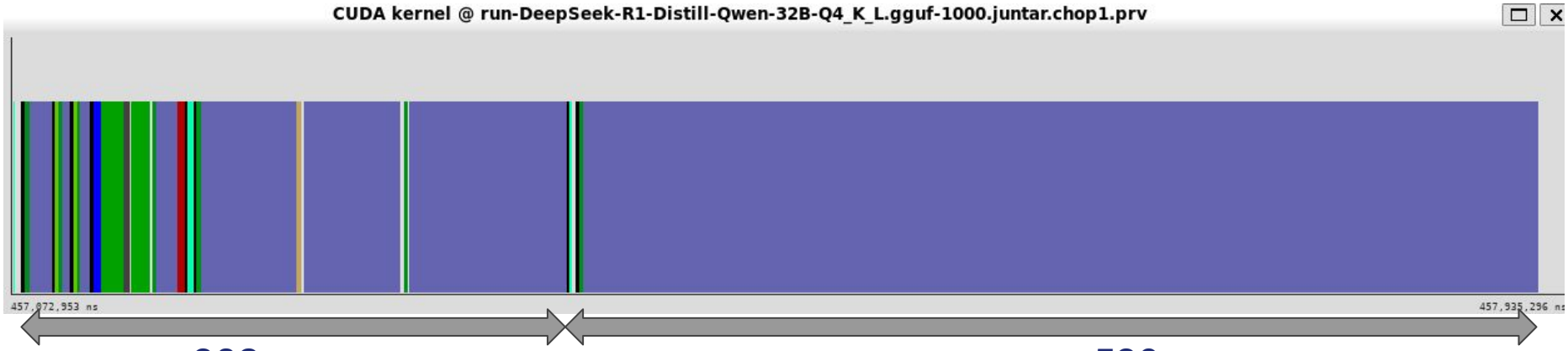
Start and Ending of Iteration



Start: cudaMemCpyHostToDevice

End: Bigger mul_mat_vec_q, cudaMemCpyDeviceToHost

Final Subiteration is Last FF



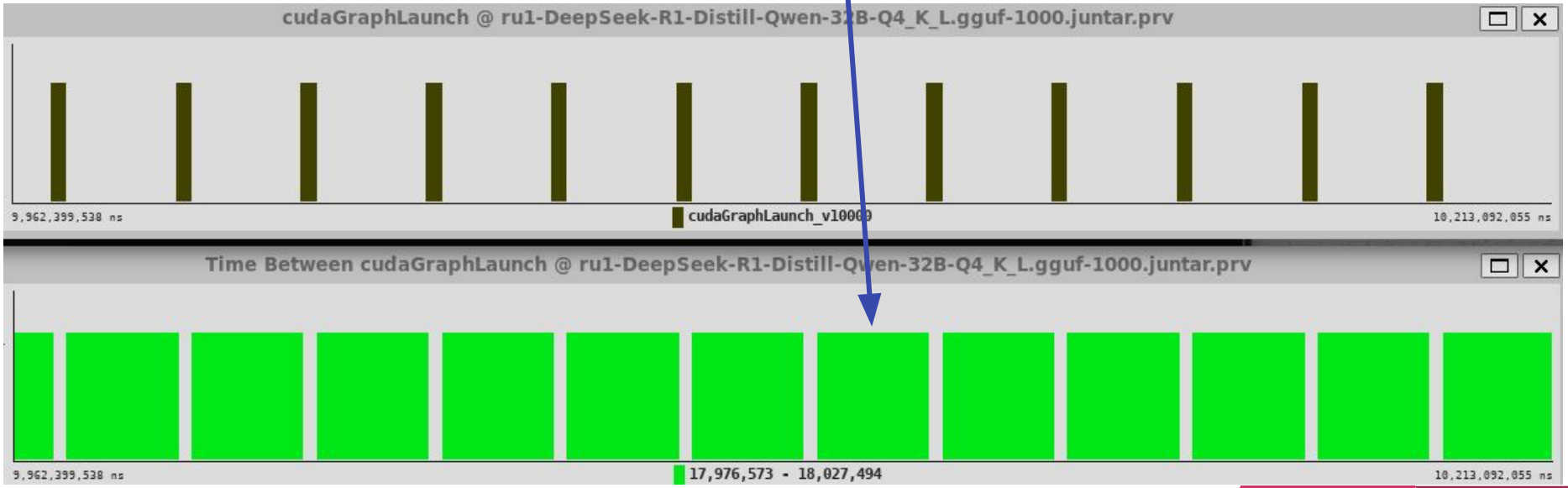
302us

539us

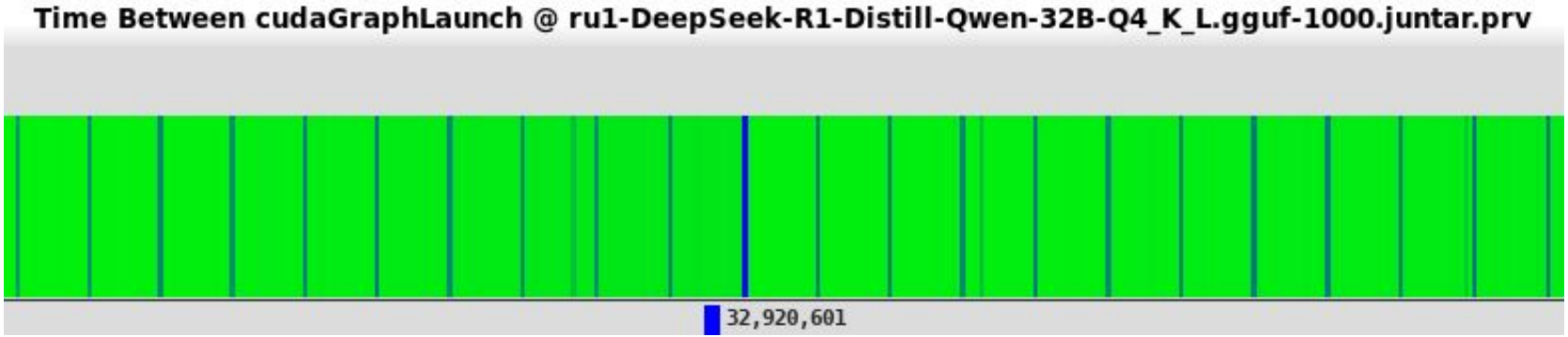
rms_norm,
k_bin_bcast,
quantize_q8_1,
mul_mat_vec_q

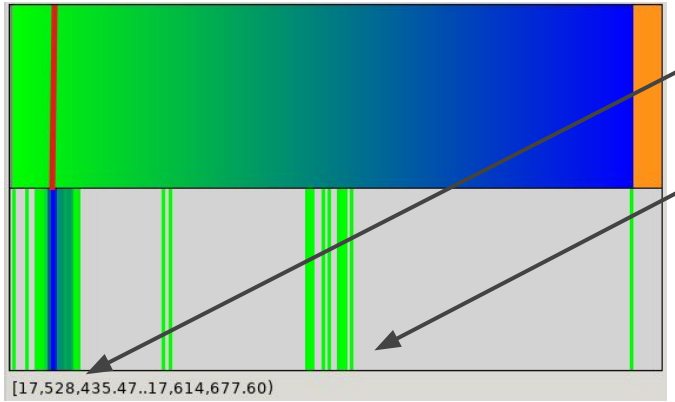
No softmax…
maybe in CPU?

Iteration Times ~ Time Between cudaGraphLaunch()



Time Between cudaGraphLaunch() in all trace

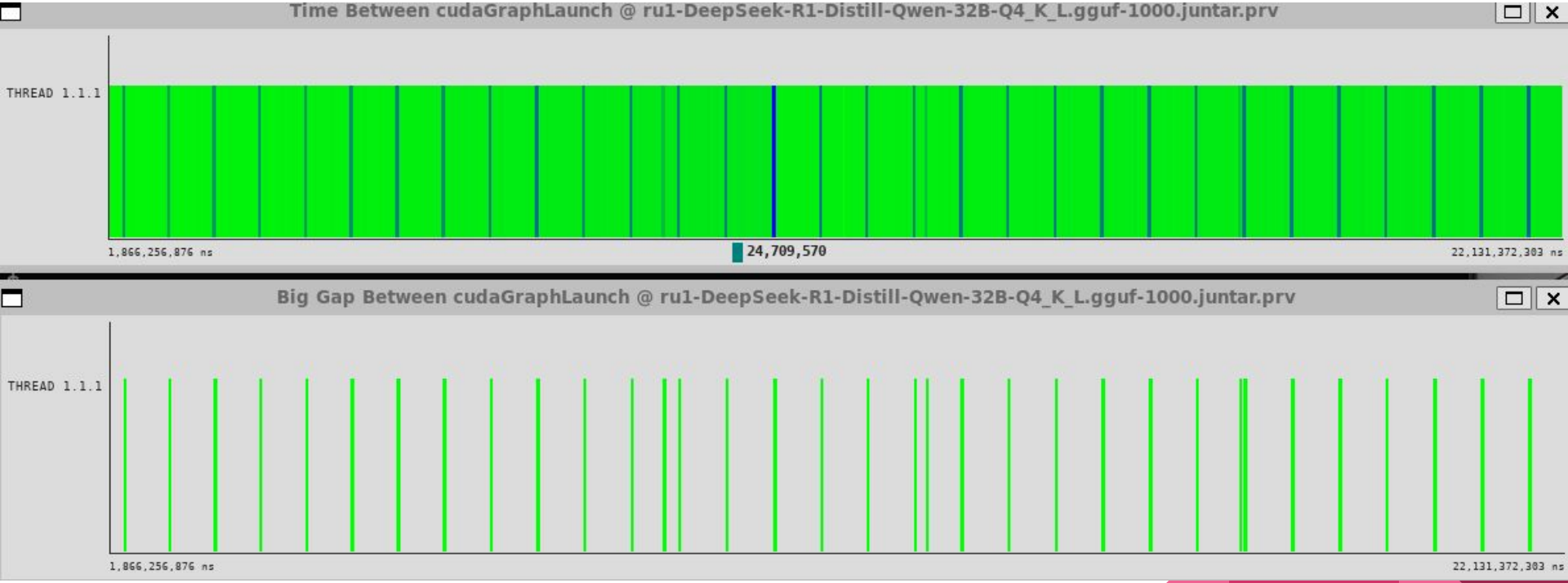




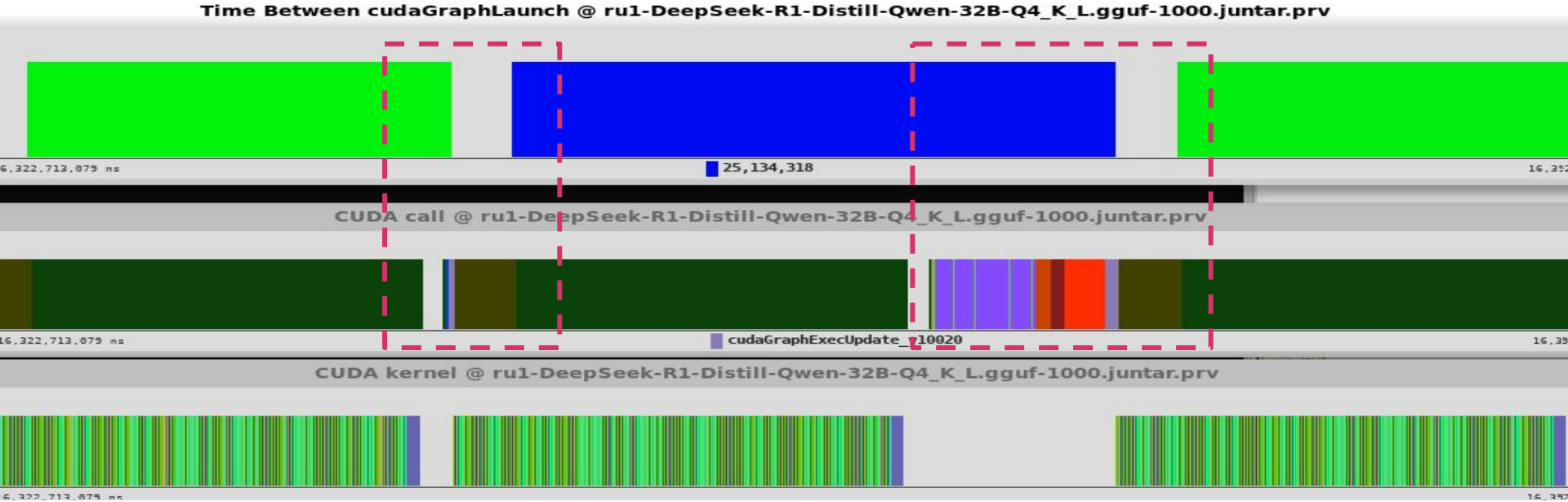
Usually the same time: ~17us

But periodically some bigger times (blue lines)

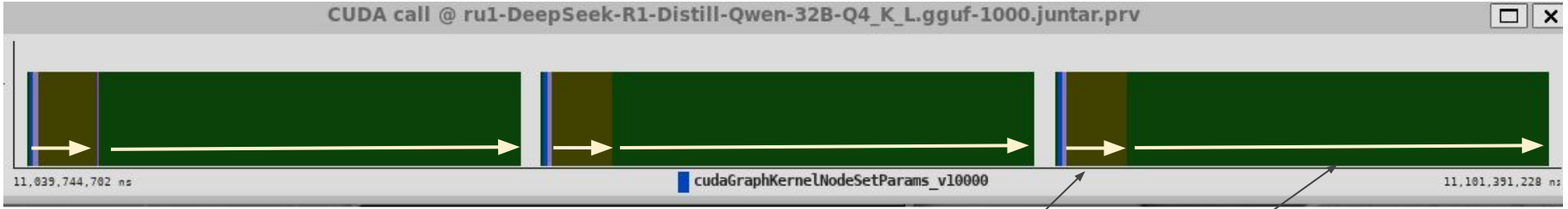
How many big times between cudaGraphLaunch?

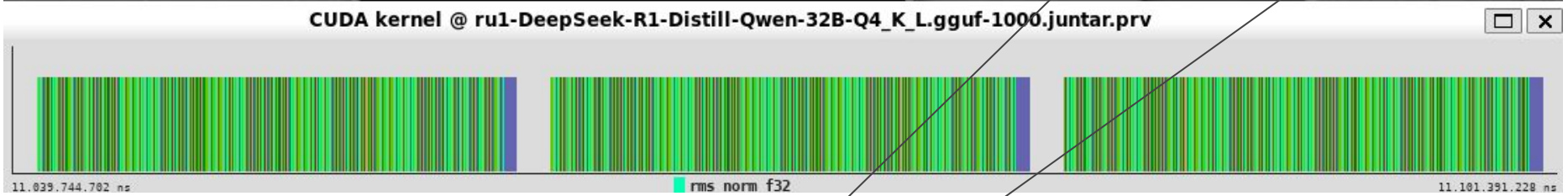


34 long calls / 1000 tokens = 3.4% or 1 every 30 tokens


Time Between cudaGraphLaunch @ ru1-DeepSeek-R1-Distill-Qwen-32B-Q4_K_L.gguf-1000.juntar.prv
25,134,318
CUDA call @ ru1-DeepSeek-R1-Distill-Qwen-32B-Q4_K_L.gguf-1000.juntar.prv
cudaGraphExecUpdate_v10020
CUDA kernel @ ru1-DeepSeek-R1-Distill-Qwen-32B-Q4_K_L.gguf-1000.juntar.prv

Regular CPU - GPU Synchronization

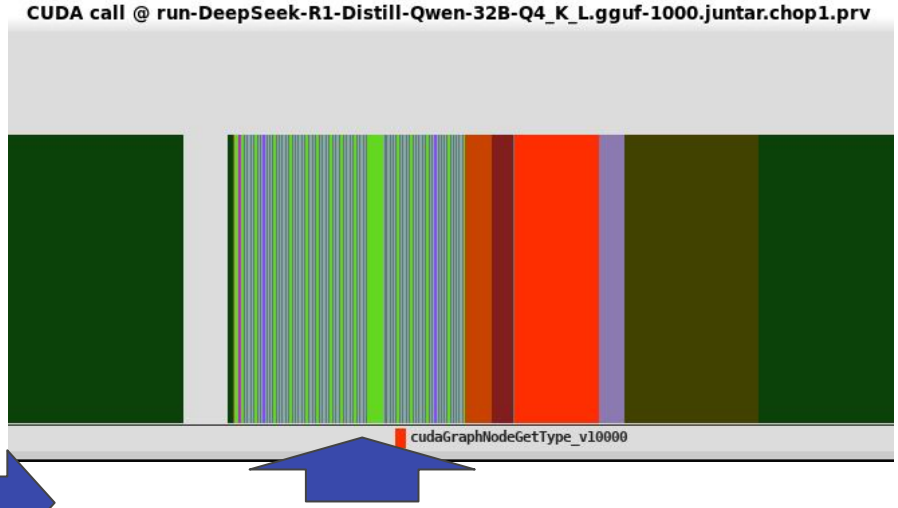




	THREAD 1.1.1
cudaGraphExecUpdate_v10020	1.32 %
cudaGraphKernelNodeSetParams_v10000	0.29 %
cudaGraphLaunch_v10000	12.15 %
cudaMemcpyAsync	0.09 %
cudaStreamSynchronize	86.14 %

Long CPU - GPU Synchronization

	THREAD 1.1.1
cuKernelGetFunction	0.07 %
cuLaunchKernel	0.49 %
cudaGraphDestroy_v10000	2.22 %
cudaGraphExecUpdate_v10020	2.21 %
cudaGraphGetNodes_v10000	0.11 %
cudaGraphKernelNodeGetParams_v10000	2.89 %
cudaGraphLaunch_v10000	10.56 %
cudaGraphNodeGetType_v10000	1.89 %
cudaLaunchKernel	10.94 %
cudaMemcpyAsync	0.46 %
cudaStreamBeginCapture_v10000	0.02 %
cudaStreamEndCapture_v10000	1.74 %
cudaStreamSynchronize	66.40 %



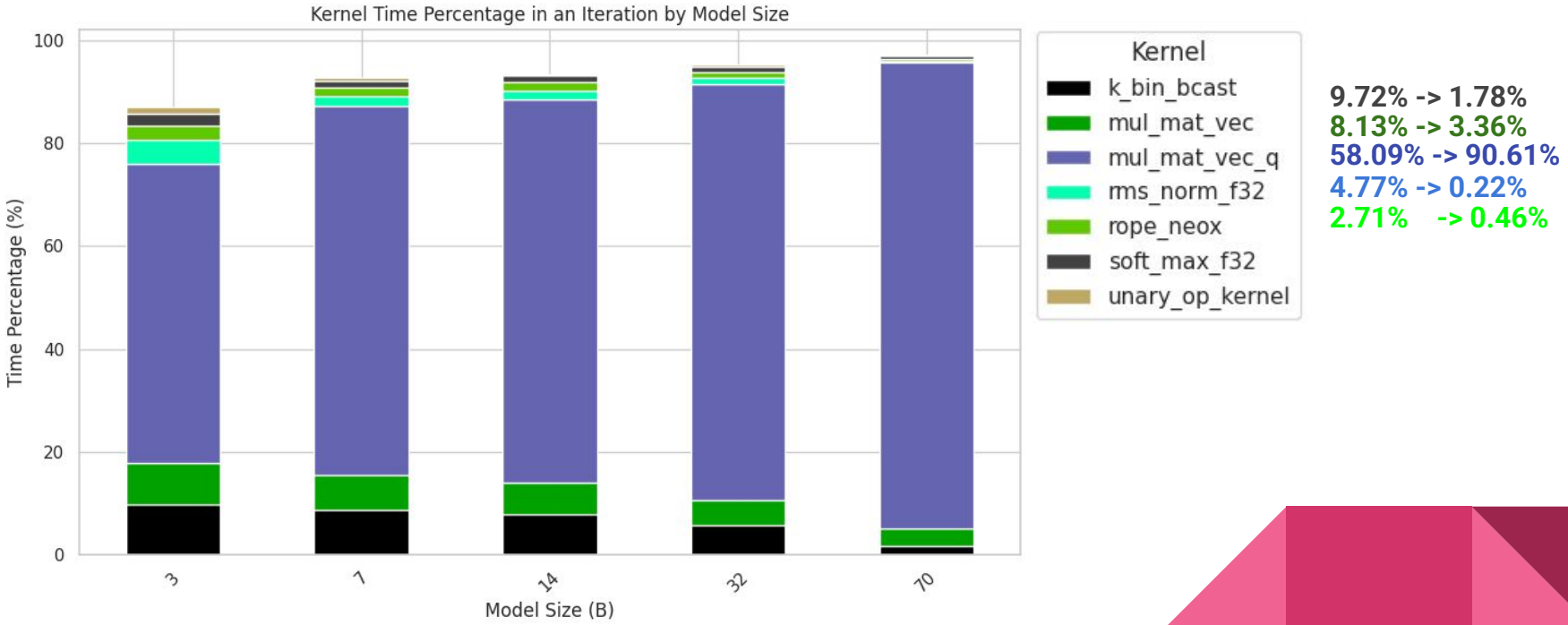
Identifying Structure Recap

- Token generation = 1 iteration identified by cudaGraphLaunch()
- # iterations = **# tokens**
- Each iteration has **#layers** subiterations
- 30% of subiteration time is Multi-Head Attention
- 70% of subiteration time is FeedForward NN
- All iterations begin and end with hostToDeviceCpy and deviceToHostCpy
- The final subiteration has a larger matrix multiplication
- Some iterations (3.4%) require bigger setup time for updating cuda graph parameters

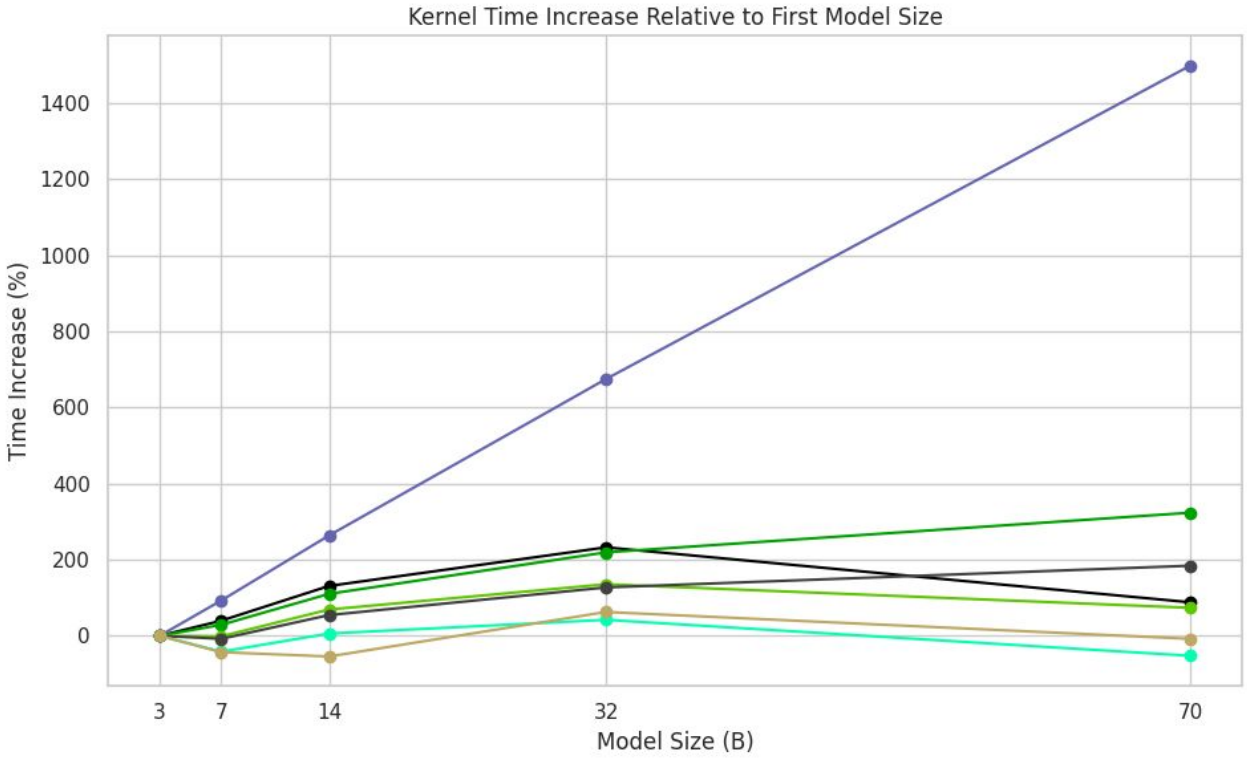
Focus of Analysis (FoA)

What impacts the most?
What changes most when varying model size?

Kernel Percentage



Kernel Relative Time Increase



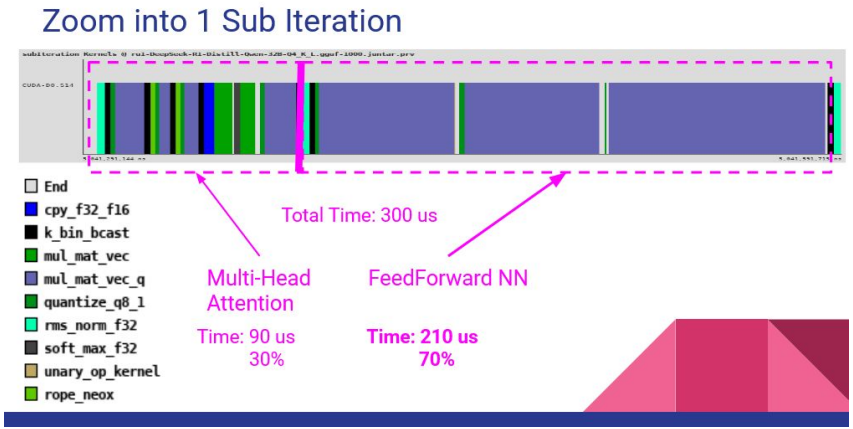
It does increment

It's doubles along to model size

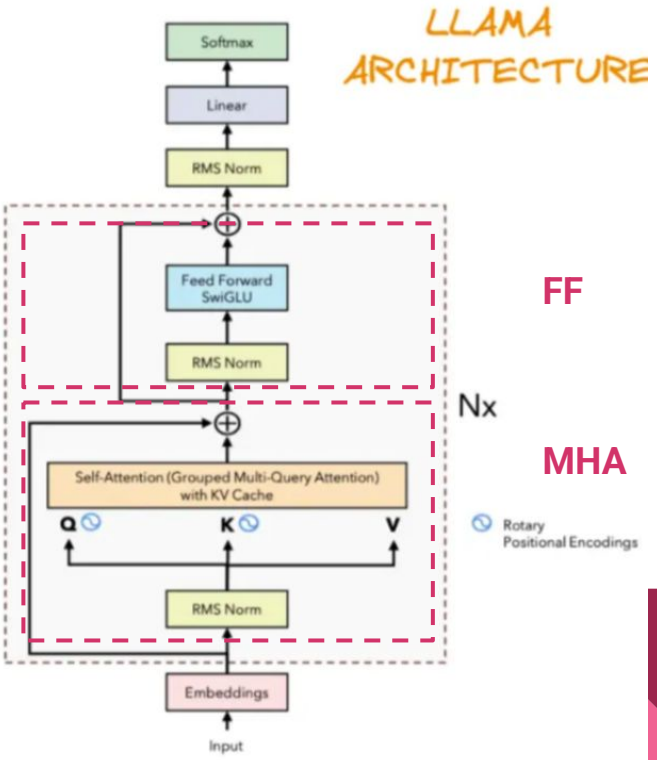
mul_mat_vec_q ∝ model size

mul_mat_vec ∝ sqrt(model size)

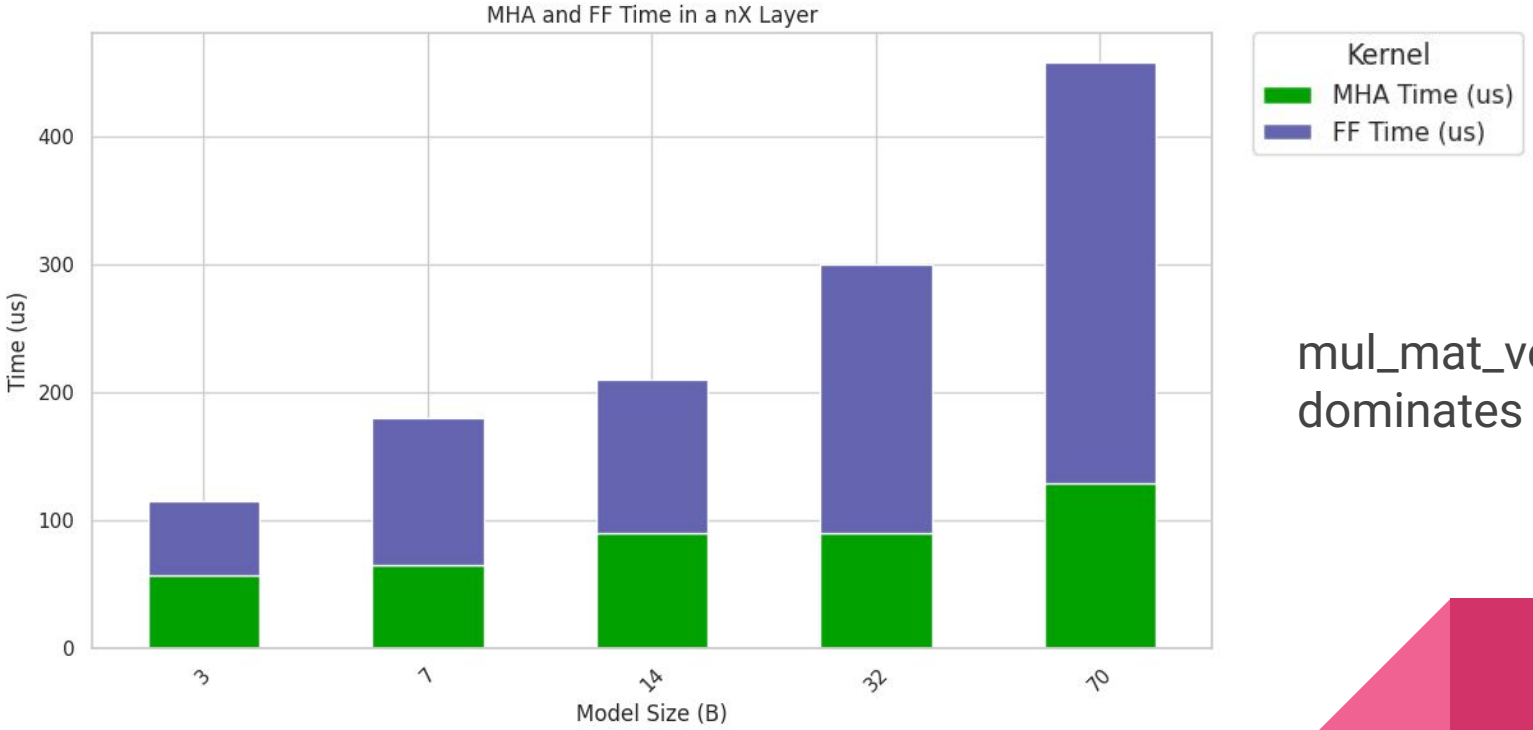
MHA vs FF Work Proportion



Slide 24 Identifying Structure for Qwen 32B-Q4_K_M



MHA vs FF Work Proportion



mul_mat_vec_q dominates the time

Subiteration Kernels Proportion

Llama-3B

Qwen-7B

Qwen-14B

Qwen-32B

Llama-70B

% Stream Multiprocessors (SM) Active

H100 SMs: 114

Render Config	
Shading Units:	14592
TMUs:	456
ROPs:	24
SM Count:	114
Tensor Cores:	456
L1 Cache:	256 KB (per SM)
L2 Cache:	50 MB

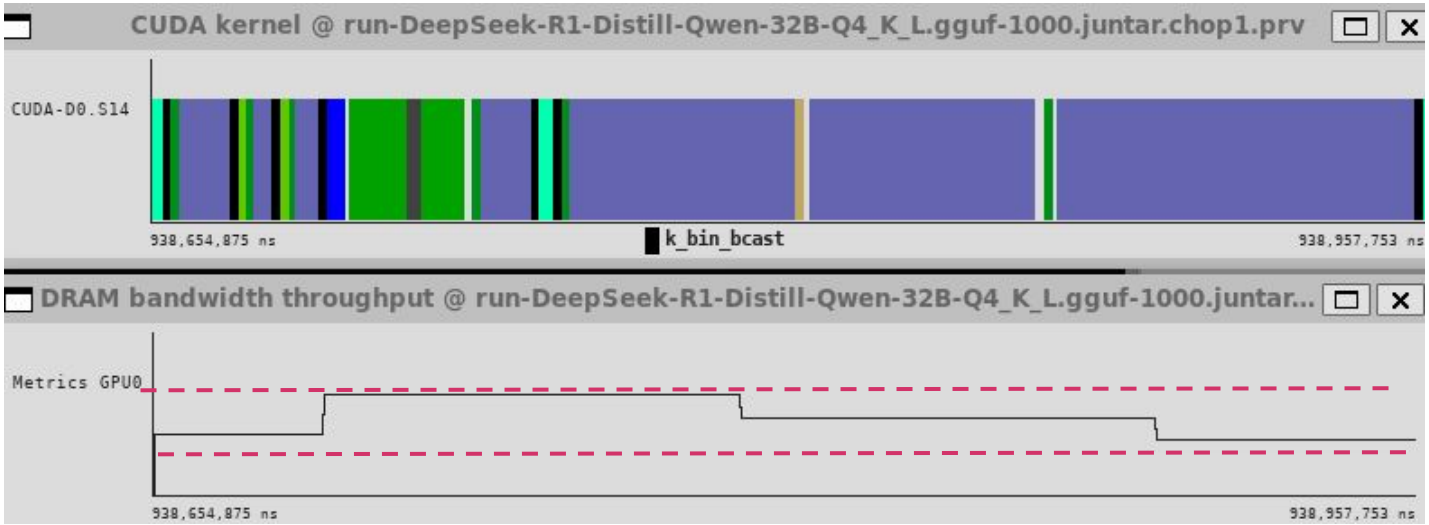
90%

72%

Bandwidth Throughput (%)

H100 BW: ???

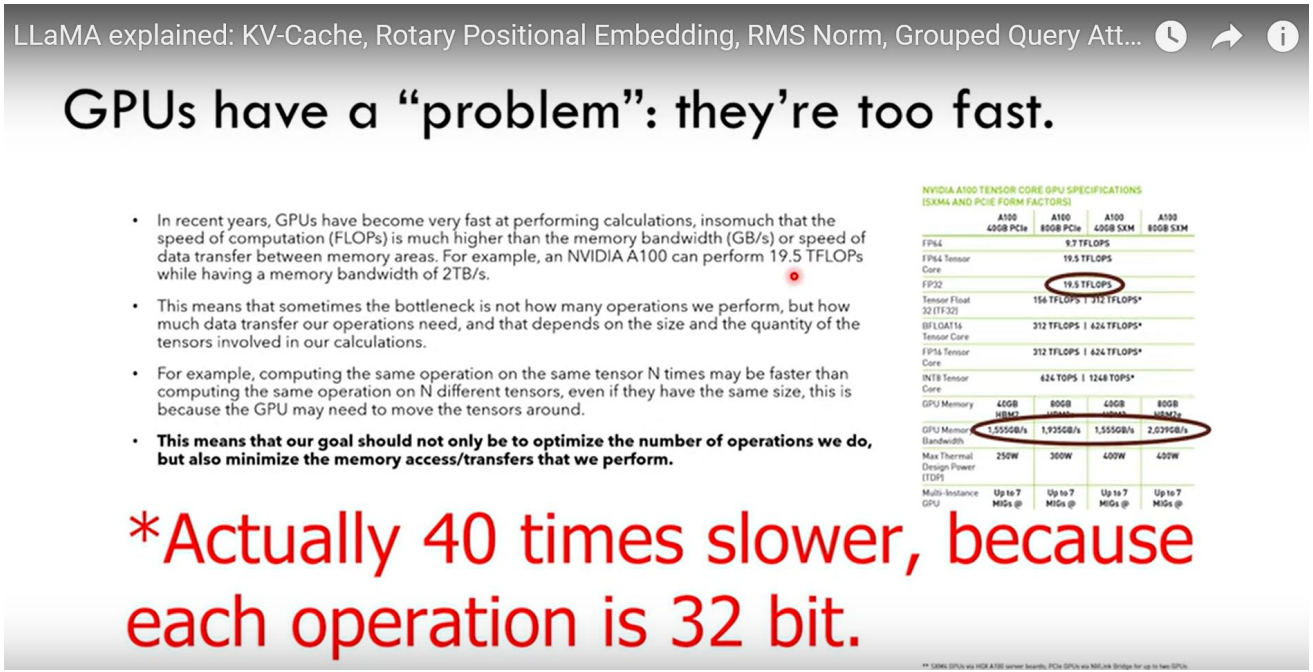
Memory	
Memory Size:	80 GB
Memory Type:	HBM2e
Memory Bus:	5120 bit
Bandwidth:	2.04 TB/s



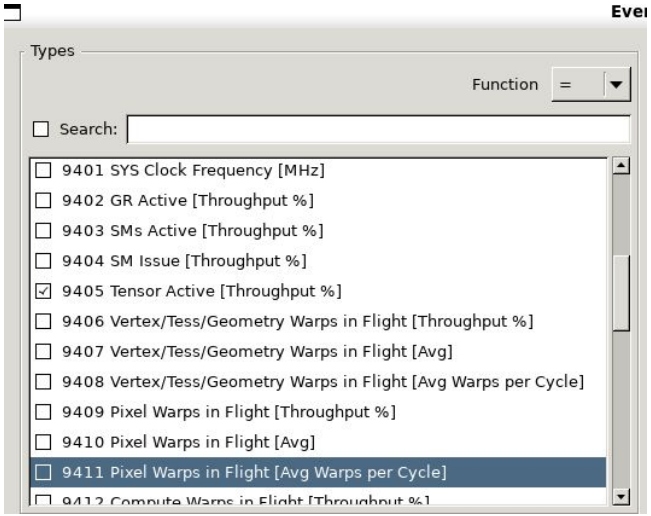
82%

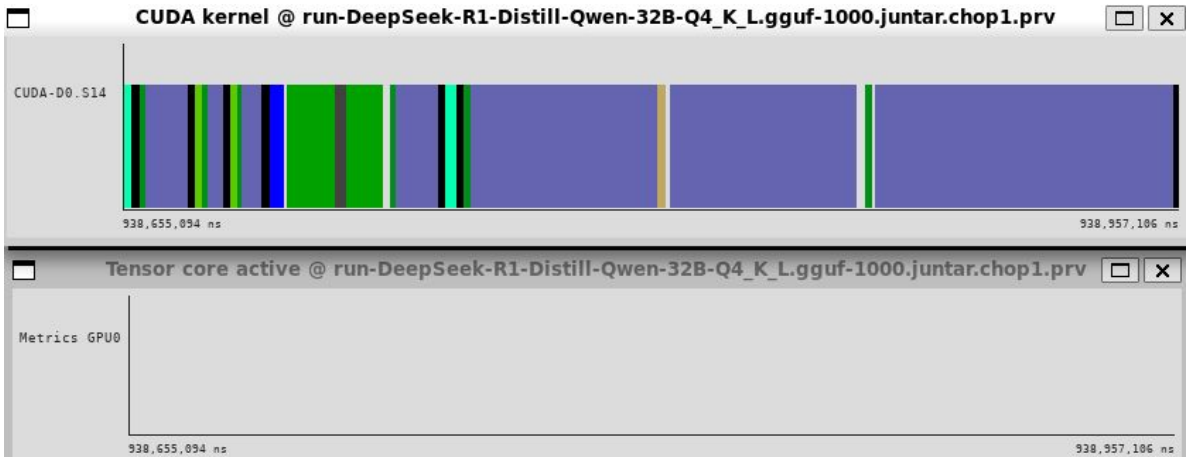
46%

Common Opinion that Memory is Bottleneck



NO Tensor Cores Active per Iteration





0% Tensor Cores Utilization

NO Tensor Cores Active per Iteration

nsys-ui also shows no Tensor Activity

0% Tensor Cores Utilization

=> Everything is in Traditional CUDA Threads

Focus of Analysis Recap

- **mul_mat_vec_q ∝ Model Size.** FF mostly has mul_mat_vec_q
- **mul_mat_vec, k_bin_bcast ∝ sqrt(Model Size).**
- FF grows bigger than MHA
- The Memory Bandwidth is higher for MHA portion and Lower in End of FF
- No Tensor Cores Utilization so all computations are in regular SM Warps/Threads
- **FF is the most Compute Heavy, so mul_mat_vec_q is more impacted by compute bottlenecks than memory in llama.cpp with H100**

Thank You

Appendix

References

bartowski. (2025) Models GGUFs bartowski/DeepSeek-R1-Distill-Llama-70B-GGUF · Hugging Face,bartowski/DeepSeek-R1-Distill-Qwen-32B-GGUF · Hugging Face, bartowski/DeepSeek-R1-Distill-Qwen-14B-GGUF · Hugging Face,bartowski/DeepSeek-R1-Distill-Qwen-7B-GGUF · Hugging Face, hassenhamdi/DeepSeek-R1-Distill-Llama-3B-GGUF · Hugging Face

Chockalingam, A (2024). Accelerating LLMs with llama.cpp on NVIDIA RTX Systems | NVIDIA Technical Blog

QwenTeam, Alibaba Group. QWEN2 TECHNICAL REPORT. 2407.10671 Pag 5

Meta. The Llama3 Herd of Models 2407.21783 Pag 7.

Umar Jamil (2024) LLaMA explained: KV-Cache, Rotary Positional Embedding, RMS Norm, Grouped Query Attention, SwiGLU.(132) LLaMA explained: KV-Cache, Rotary Positional Embedding, RMS Norm, Grouped Query Attention, SwiGLU - YouTube

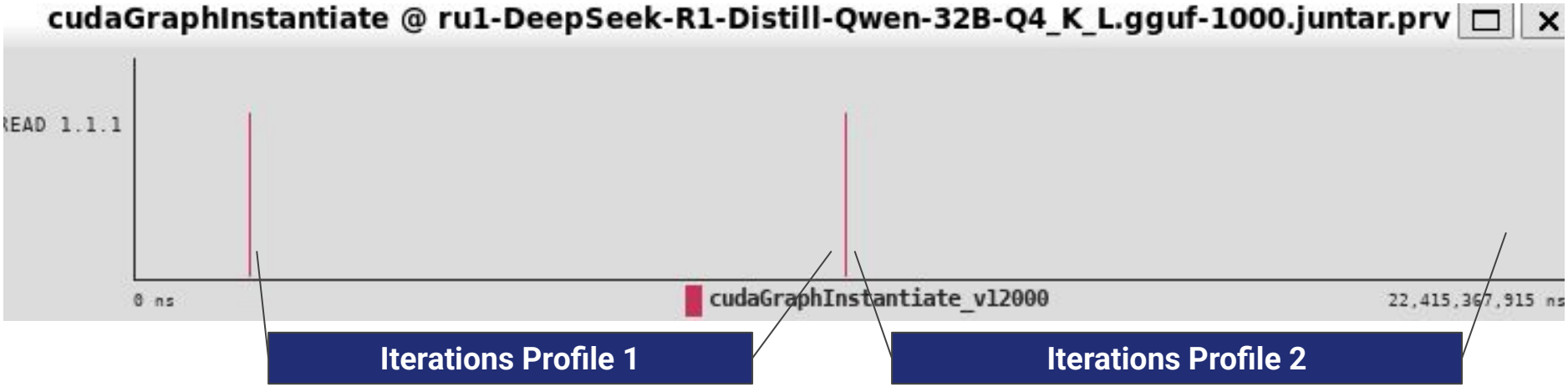
Kansal, V. (2023) Understanding the GGUF Format: A Comprehensive Guide. Understanding the GGUF Format: A Comprehensive Guide | by Vimal Kansal | Medium

Quantization Bits Table. GGUF quantizations overview

Setup Execution + First Token Iteration

2 Irregular Execution iterations happen before GraphLaunched (blue calls) iterations. Irregular execution uses more kernel types

Something odd… There are 2 Graph Instantiations…



Kernels by Iteration Type

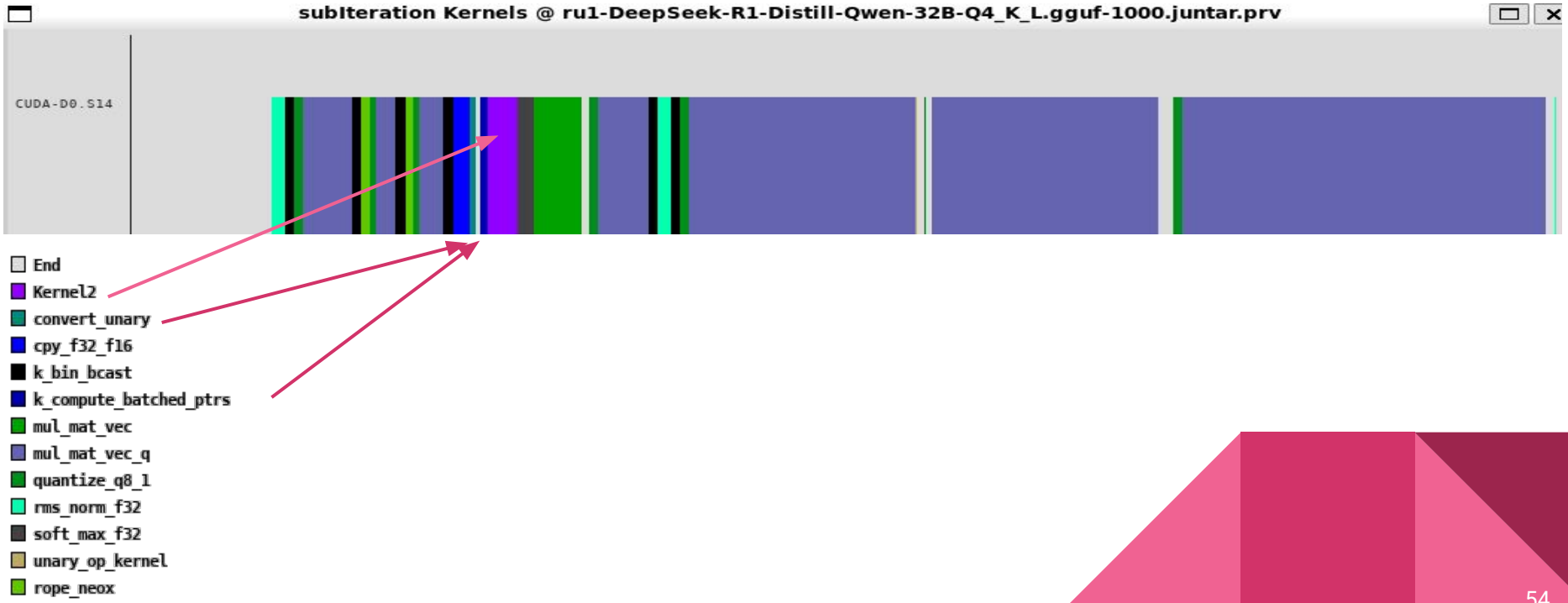
Profile 1 (Early) Iteration

cpy_f32_f16	128
k_bin_bcast	513
k_get_rows_float	2
mul_mat_vec	128
mul_mat_vec_q	449
quantize_q8_1	449
rms_norm_f32	129
soft_max_f32	64
unary_op_kernel	64
rope_neox	128

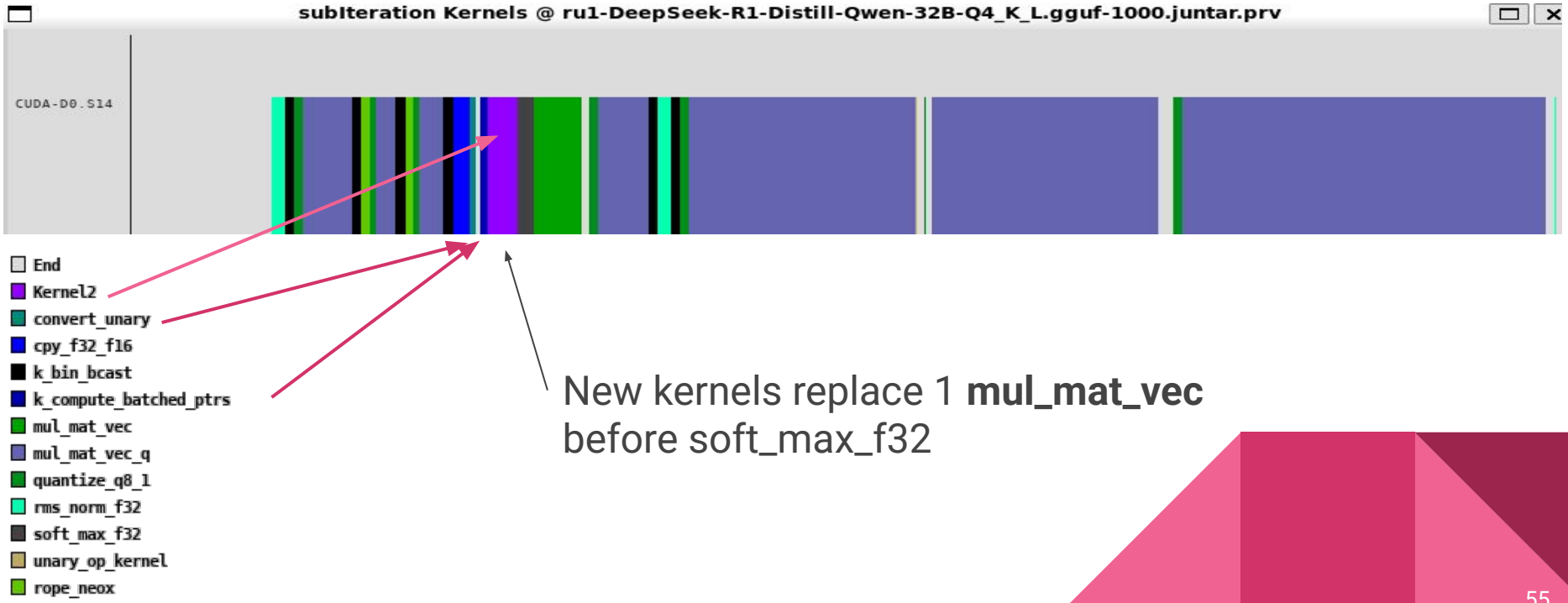
Profile 2 (Late) Iteration

Kernel2	64	New
convert_unary	64	New
cpy_f32_f16	128	
k_bin_bcast	513	
k_compute_batched_ptrs	64	New
k_get_rows_float	2	
mul_mat_vec	64	Reduced
mul_mat_vec_q	449	
quantize_q8_1	449	
rms_norm_f32	129	
soft_max_f32	64	
unary_op_kernel	64	
rope_neox	128	

SubIteration Profile 2



SubIteration Profile 2



New kernels replace 1 **mul_mat_vec** before soft_max_f32

Marenostrum 5 profile llama.cpp model

CUDA_VISIBLE_DEVICES=0,1,2,3 nsys profile --output experiments/longrun_job_12_delay_all_duration_graph_mode.nsys-rep --sample=process-tree --sampling-period=1850000 --trace=cuda,nvtx,osrt,cublas,cudnn --cuda-graph-trace=graph --gpu-metrics-devices=cuda-visible --gpu-metrics-frequency=10000 --cpuctxsw=process-tree --delay 12 ./llama.cpp/build/bin/llama-cli -m LLM_Models/DeepSeek-R1-Distill-Qwen-32B-Q4_K_L.gguf -no-cnv -n 1000 -ngl 1000 -p "Explain the usage of HPC on addressing modern world problems and its impact on economy"

Marenostrum 5 convert nsys to paraver

module load intel mkl python/3.12.1 sqlite3

nsys2prv -t cuda_api_trace,nvtx_pushpop_trace,graph,gpu_metrics
longrun_job_12_delay_all_duration_graph_mode.nsys-rep
longrun_job_12_delay_all_duration_graph_mode

tar -cvjf longrun_job_12_delay_all_duration_graph_mode.prv.tar.bz2
longrun_job_12_delay_all_duration_graph_mode.pcf
longrun_job_12_delay_all_duration_graph_mode.prv
longrun_job_12_delay_all_duration_graph_mode.row