Experiences of an HPC analyst in AI land

POP Webinar #43. September 4th, 2026

Jesus Labarta(jesus.labarta@bsc.es) , BSC

HORIZON-EUROHPC-JU-2023-COE

EuroHPC Joint Undertaking

1 January 2024– 31 December 2026

Grant Agreement No 101143931

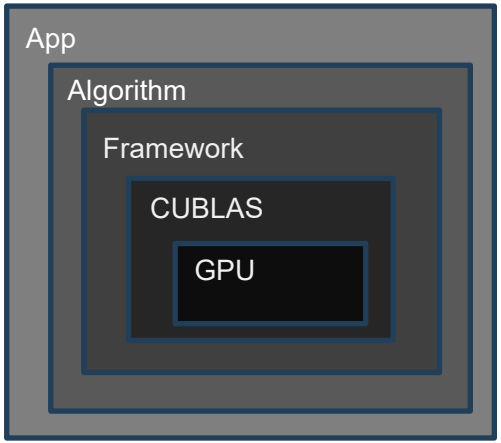
POP: From HPC to AI ?

- AI has become an even higher consumer of compute cycles than HPC
- HPC and AI ... speak a different language
  - Task, architecture, performance, … same words, DIFFERENT meaning
- But still, we should be able to …
  - … build on HPC performance analysis tools and methodologies …
  - … towards seamless analysis of AI apps from very small to very large scale
- … can we ?

How efficiently are we using our resources?

AI

- Black matryoshkas



- "no" visibility
  - What are we really doing?
  - How are we performing ?

vendors, …

- Just leave it to us !!
  - Our libraries, frameworks will do the BEST for you.
  - We "do not" offer mechanisms for you to take/steer our decision
  - Why would you want explicit control of …
    - Kernel variant selected , #blocks, threads
    - In GEMMs, NCCLs, …

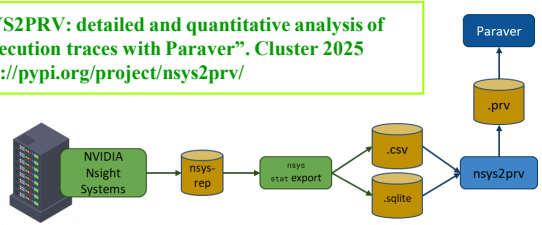
Assessments

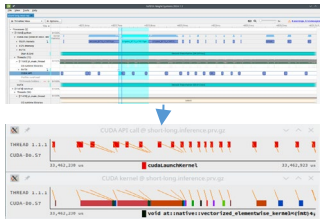
- Important to understand **my** case
- May be optimize it ?
  - A bit more informed decisions than exhaustive exploration, somebody dixit, …
- Evidences to discus with vendors, **AIs**, …

nsys2prv

- Can we leverage Nsight Systems´s acquisition capability …
- … and use Paraver to better squeeze the information in those traces ?
  - Scalability: dynamic range from very coarse to extremely fine grain
  - Analytics: timelines, histograms, correlations beyond profiles

M. Clascá et al. “NSYS2PRV: detailed and quantitative analysis of large-scale GPU execution traces with Paraver”. Cluster 2025
https://pypi.org/project/nsys2prv/





Towards microscopic insight …

… from single GPU to “large” clusters

A few tales …

A few tales and worries/doubts

- Long, long time ago, …
- Common wisdom, real wisdom?
- On Sparsity: concurrency, (a)synchronism, APIs
- Load imbalance in AI?
- How important is communication?
  - To overlap or not to overlap
  - Computation interfering communication?
  - How to measure bandwidth?
- On pears and apples
  - Gemm performance variability for large problem size?
  - How many gemm variants exist?
  - “Same” code, different frameworks?
- Role of CPUs in a GPU centric world?

+ Efficiency model

Training and inference
Understanding/Improving

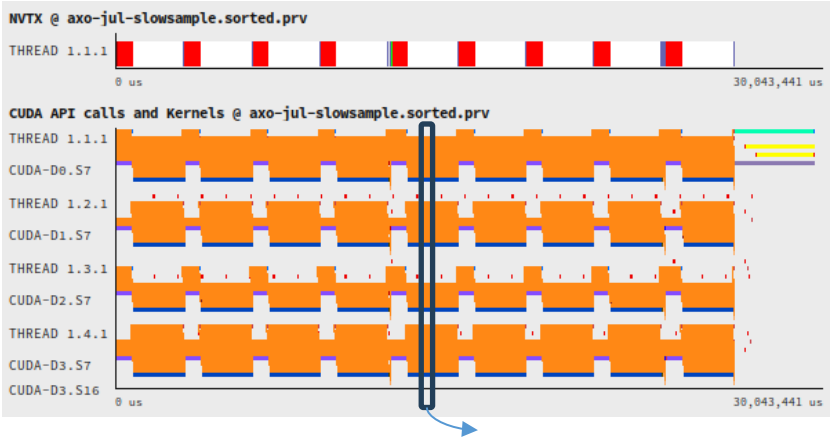
Long, long time ago

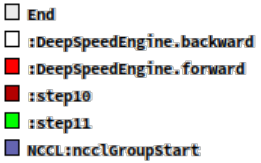
(24 months)

in a far, far …

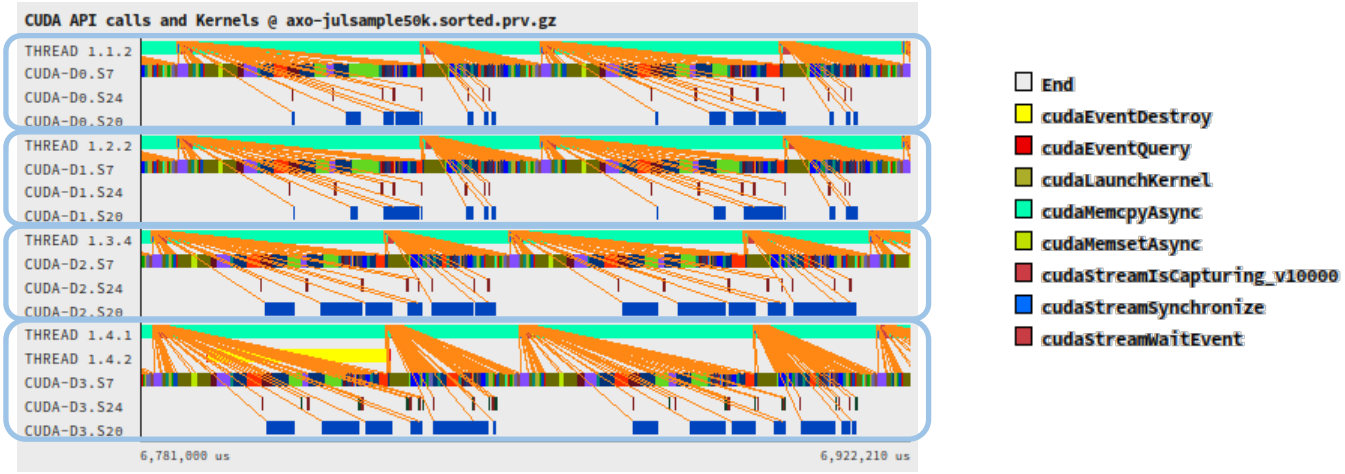
My first Paraver view …

- … of an LLM training (fine tuning)
- @ 4 GPUs

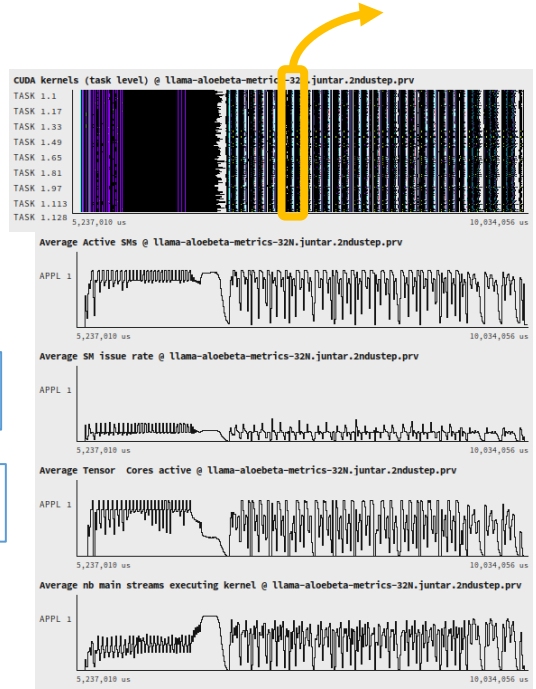


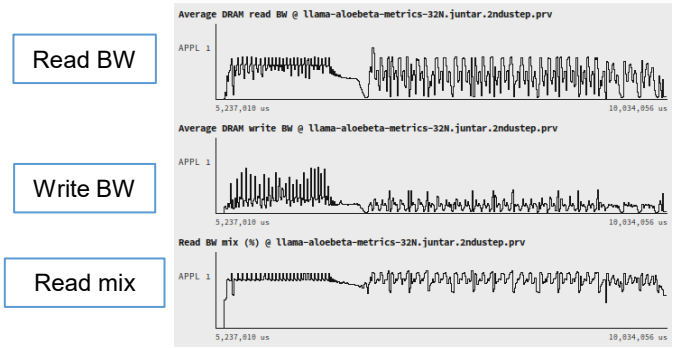


API calls, launch lines and kernels



@ 128 GPUs ?


Kernels
SM active
SM instr. Issue rate
Tensor core active
% active kernels
CUDA kernels (task level) @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Average Active SMs @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Average SM issue rate @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Average Tensor Cores active @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Average nb main streams executing kernel @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv


Read BW
Write BW
Read mix
Average DRAM read BW @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Average DRAM write BW @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv
Read BW mix (%) @ llama-aloebeta-metrics-32N.juntar.2ndustep.prv

@ 128 GPUs ?

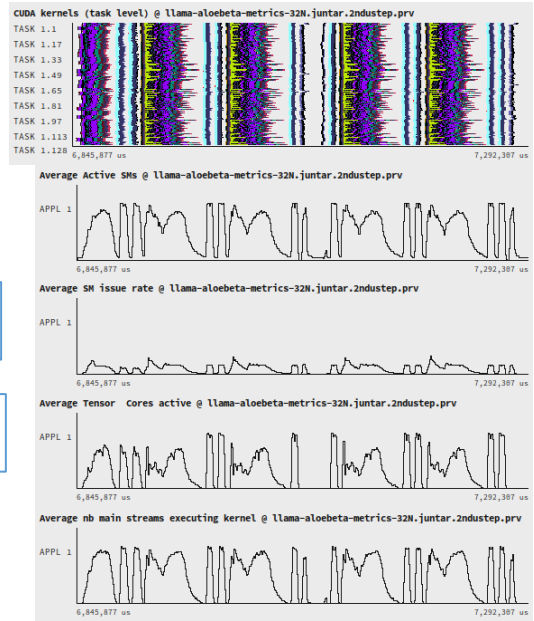
Kernels

SM active

SM instr. Issue rate

Tensor core active

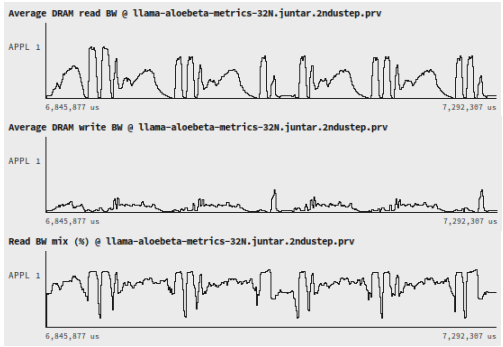
% active kernels



Read BW

Write BW

Read mix

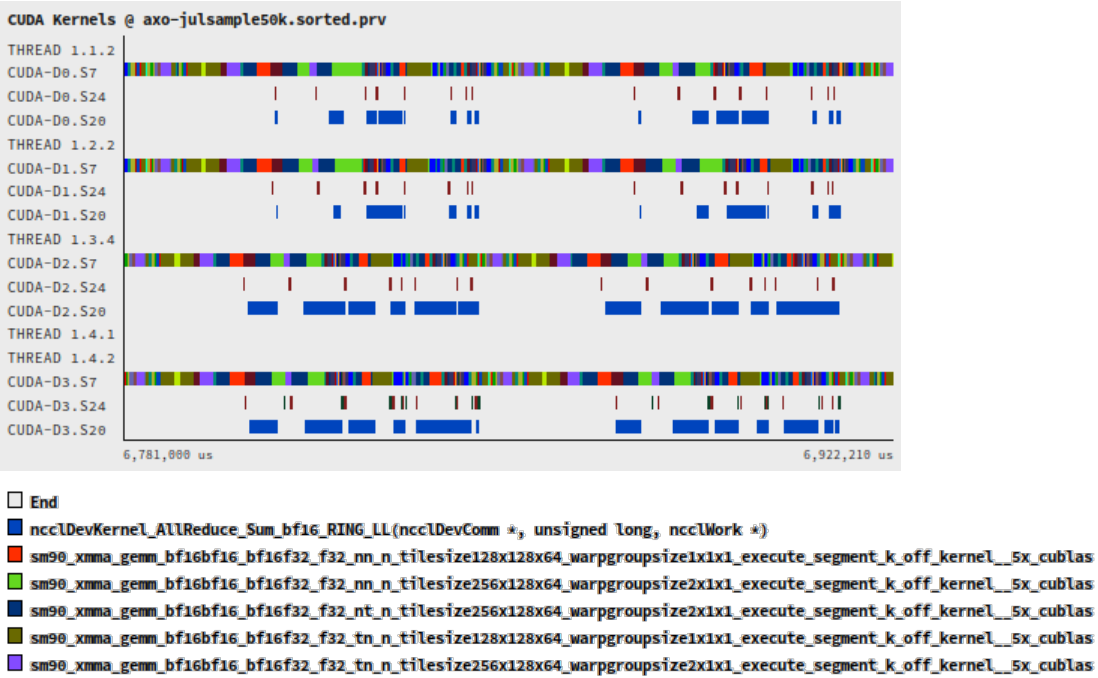


Lots of opportunities … who to “blame”?

Common wisdom … real wisdom?

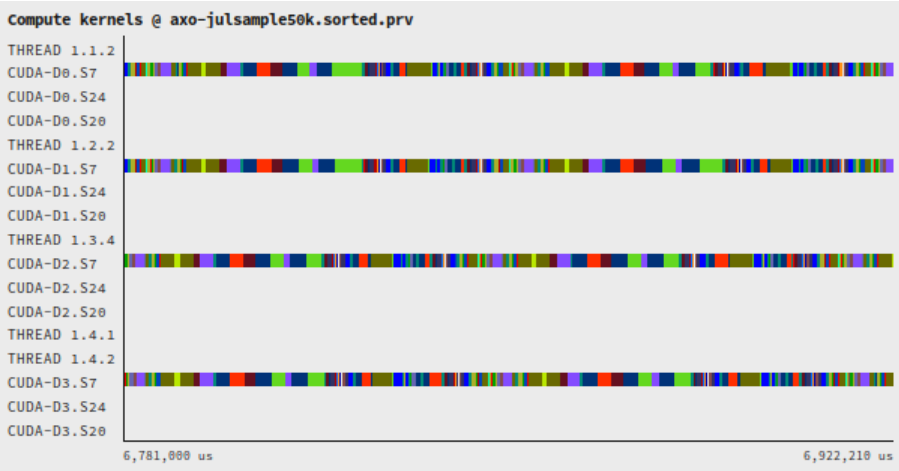
Common wisdom (I)

- “AI == communication bound” ?



Common wisdom (I)

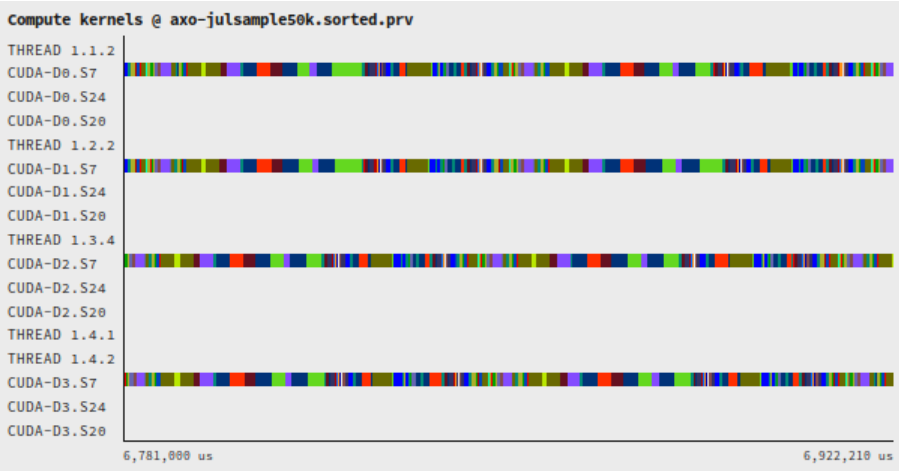
- "AI == communication bound"
  - … but this training case is certainly NOT communication bound



Idle %:	3.56

Common wisdom (II)

- “AI == MxM”



Idle %:	3.56

Common wisdom (II)

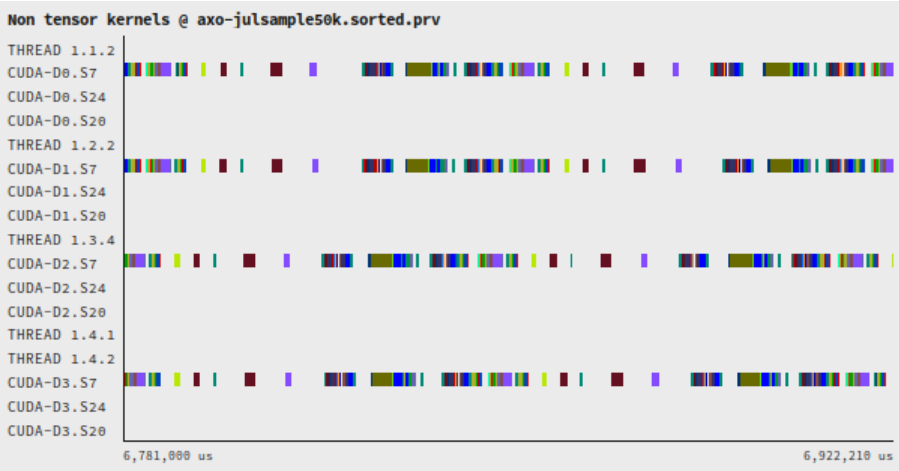
- "AI == MxM"
  - … at least in current platforms NOT the only/bottleneck

Idle %:	3.56
Gemm %:	49.95

sm90_xmma_gemm_bf16bf16_bf16f32_f32_nn_n_tilesize128x128x64_warpgroupsize1x1x1_execute_segment_k_off_kernel__5x_cublas
sm90_xmma_gemm_bf16bf16_bf16f32_f32_nn_n_tilesize256x128x64_warpgroupsize2x1x1_execute_segment_k_off_kernel__5x_cublas
sm90_xmma_gemm_bf16bf16_bf16f32_f32_nt_n_tilesize256x128x64_warpgroupsize2x1x1_execute_segment_k_off_kernel__5x_cublas
sm90_xmma_gemm_bf16bf16_bf16f32_f32_tn_n_tilesize128x128x64_warpgroupsize1x1x1_execute_segment_k_off_kernel__5x_cublas
sm90_xmma_gemm_bf16bf16_bf16f32_f32_tn_n_tilesize256x128x64_warpgroupsize2x1x1_execute_segment_k_off_kernel__5x_cublas

Common wisdom (II)

- “AI == MxM”
  - … at least in current platforms NOT the only/bottleneck

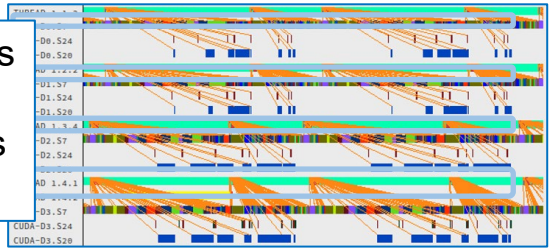


Idle %:	3.56
Gemm %:	49.95
Non Gemm %:	46.49

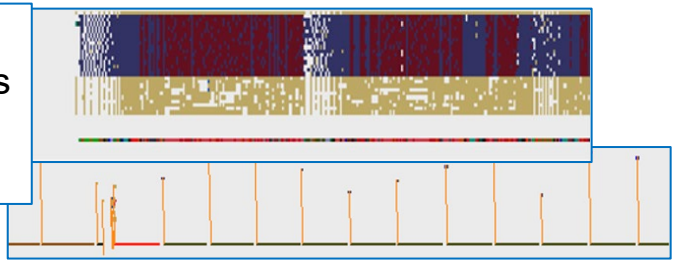
On Sparsity: concurrency, (a)synchronism, APIs

On Sparsity

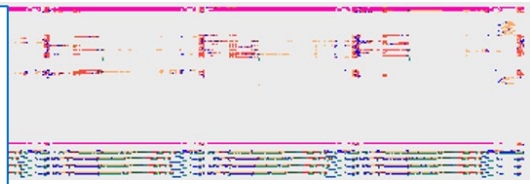
4 processes
5 threads
12 streams
4 GPUs



1 process
A few (~25) threads
1 stream
1GPU



1 process
Many (211) threads
40 streams
4 GPUs



"Lots" of distinct "agents"
- CPU threads, GPU streams

"Lots" of distinct resources

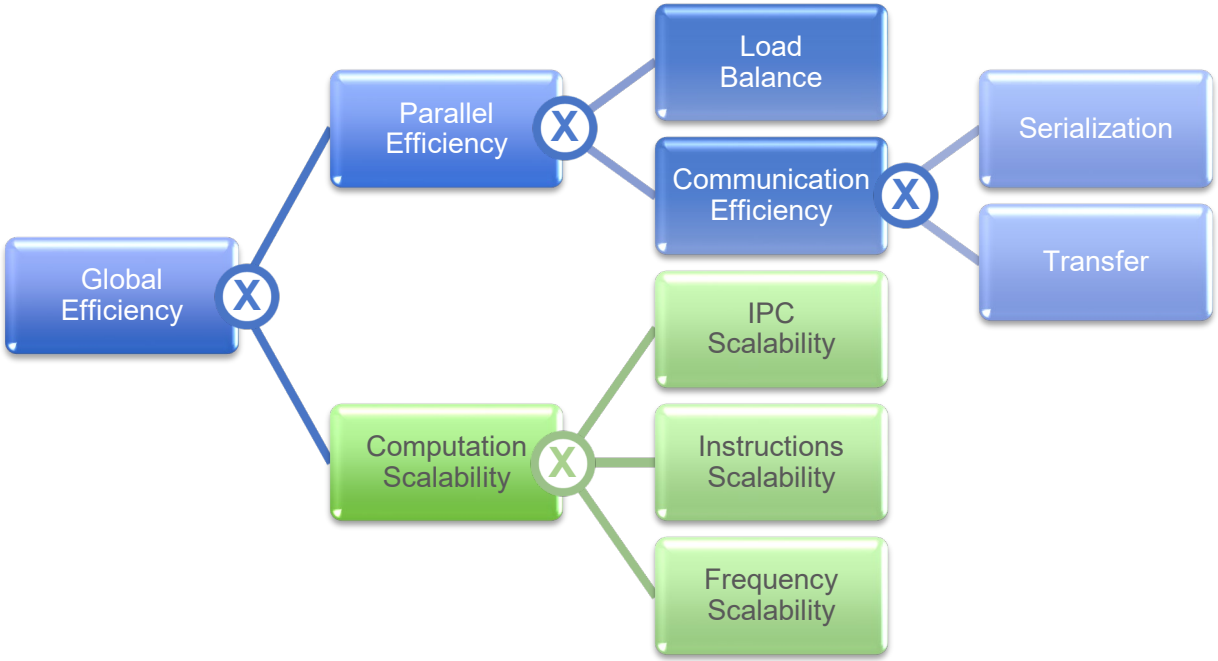
Lots of mechanisms to establish order:
- Processes, Pthreads, MPI, OpenMP, Julia, …
- "In order" within stream
- Host device synchronization
- Interstream syncs
- HostFunctions
- Shared memory (polling, barriers, …)

"poor" visibility of "dependences"

→ The importance of tools aggregation and browsing capabilities !!!
→ The importance of abstraction

Efficiency model

Efficiency Model



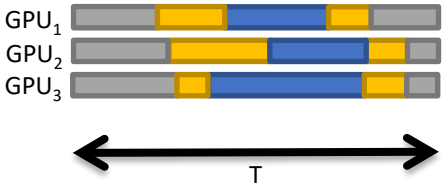
Semantics: Fundamentals of (parallel) computing !!

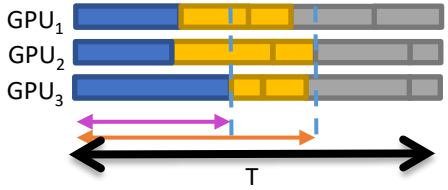
M. Casas et al, "Automatic analysis of speedup of MPI applications". ICS 2008.

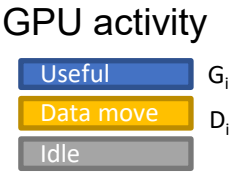
J. Giménez, et al, "Analyzing the Efficiency of Hybrid Codes" 2020 19th International Symposium on Parallel and Distributed Computing (ISPDC), 2020

TALP Device Metrics

N = num GPUs
T = Total elapsed time
G_i = Useful time of GPU_i
D_i = Time waiting for data in GPU_i







efficiency

$$Par.Eff. = \frac{\sum_{i=1}^{N} G_i}{T*N} = \frac{\sum_{i=1}^{N} G_i}{T*N}$$

$$LB = \frac{\sum_{i=1}^{N} G_i}{\max(G_i)*N} = \frac{\sum_{i=1}^{N} G_i}{\max(G_i)*N}$$

$$Comm.Eff. = \frac{\max(G_i)}{\max(G_j+D_j))} =$$

$$Orch.Eff. = \frac{\max(G_i+D_i)}{T} = \frac{\max(G_i+D_i)}{T}$$

computational scaling

$$CompEff = \frac{\sum_{i=1}^{N} G_i}{\sum_{i=1}^{N_{ref}} G_i^{ref}}$$

Load imbalance in AI?

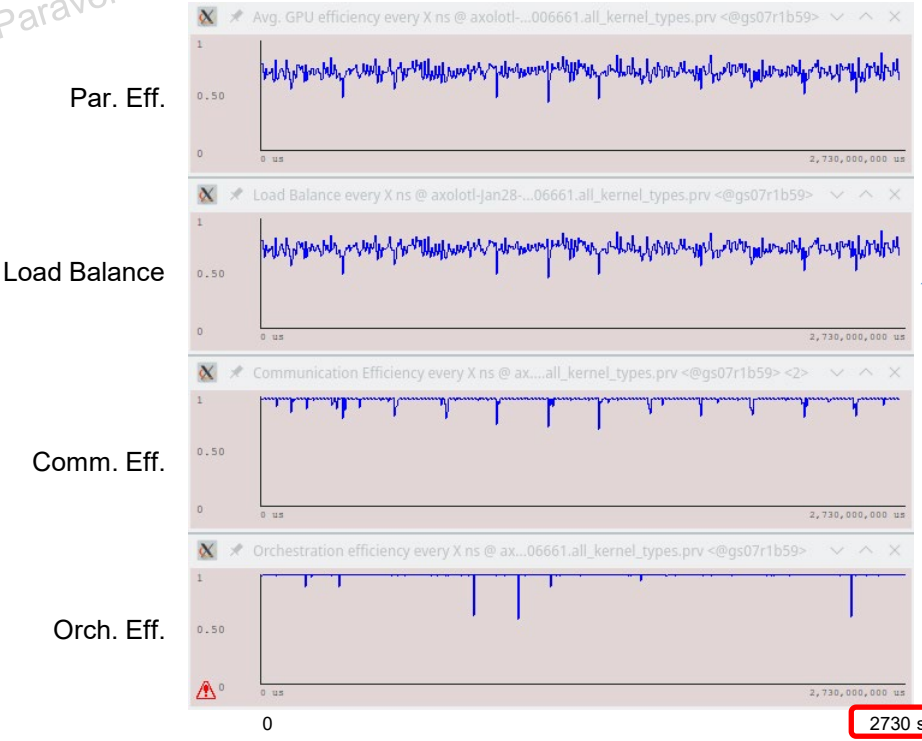
Is there load imbalance?

- Efficiency model for **structure** identification and global quantification

Paraver == DSL

efficiency **every 20ms** (avg)

SFT, Llama 8B, seq. len. 32K, 360 steps, 32 GPUs
17 GB trace



"Bad" ↔ Good

Good ↔ "Bad"

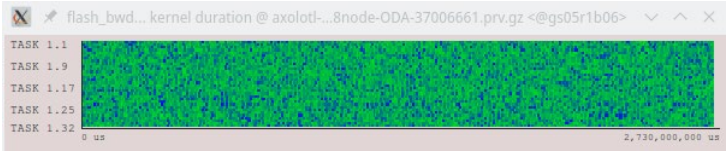
efficiency for **whole run**

	Original
Total	0.723
Par. Eff	0.723
LB	0.939
Comm. Eff.	0.773
Orch. Eff.	0.996
Comp. Eff.	1.000
Normalized total	1.00
Elapsed (us)	2702269532
Time based eff	1.00

Packing impact along time

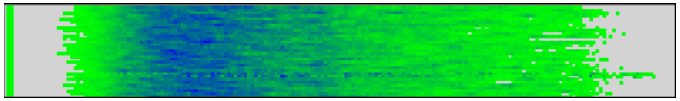
- Root cause of the microscopic imbalance? flash_bwd_dq_...

flash_bwd_dq_… kernel duration (avg not null)



0 2730 s

flash_bwd_dq_… kernel duration histogram

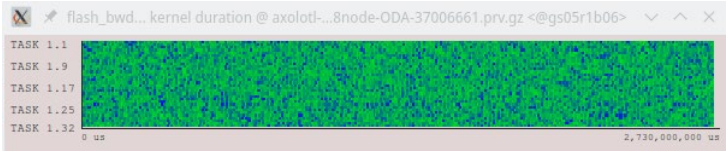


0 85 ms

Packing impact along time

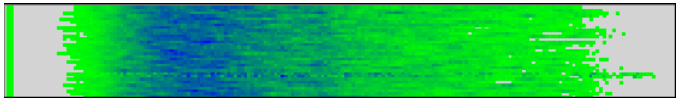
- Where to improve? How? (Flash_bwd_dq_... ? Elsewhere?)

flash_bwd_dq_… kernel duration (avg not null)



0 2730 s

flash_bwd_dq_… kernel duration histogram



0 85 ms

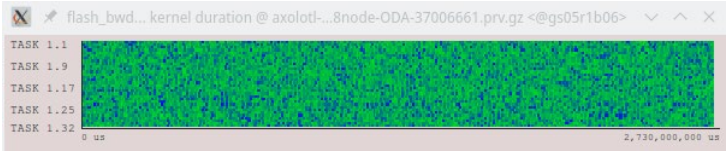
Packing impact along time

- Where to improve? How? (Flash_bwd_dq_... ? Elsewhere?)

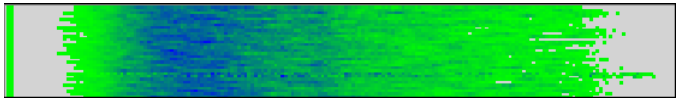
flash_bwd_dq_… kernel duration (avg not null)

flash_bwd_dq_… kernel duration histogram

Original

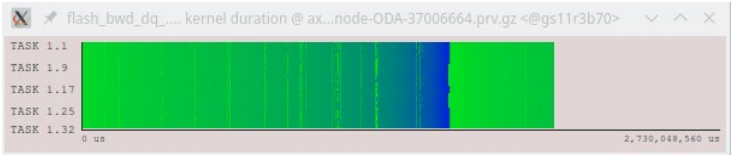


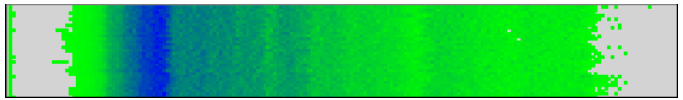
0 2730 s



0 85 ms

Reordered





	Original	Reordered
Total	0.723	0.970
Par. Eff	0.723	0.925
LB	0.939	0.987
Comm. Eff.	0.773	0.944
Orch. Eff.	0.996	0.992
Comp. Eff.	1.000	1.049
Normalized total	1.00	1.34
Elapsed (us)	2702269532	2017510818
Time based eff	1.00	1.34

Fixing microscopic load imbalance highly improves communication metric
(avoids circular waits @ global syncs)

Comm. Eff. (not necessarily only data transfer) still remains as main **small** inefficiency

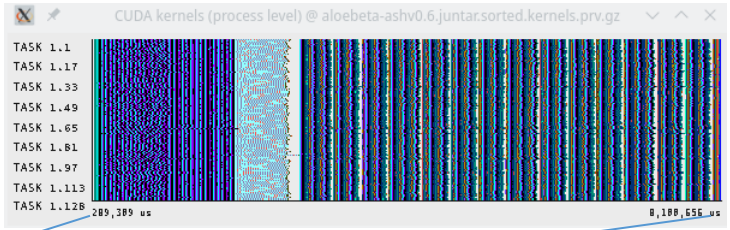
A comment on precision … really representative / informative digits?

How important is “communication”?

To overlap or not to overlap

To overlap or not to overlap

Non Overlapping



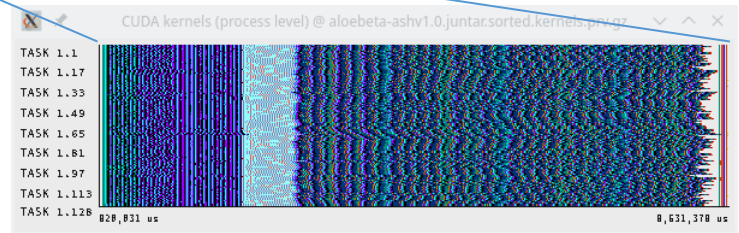
GPU ǁ efficiency: 85.3 %
Load balance: 94 %
Comm. Eff. 90.3 %
Comp. Eff. 100 %

To overlap or not to overlap

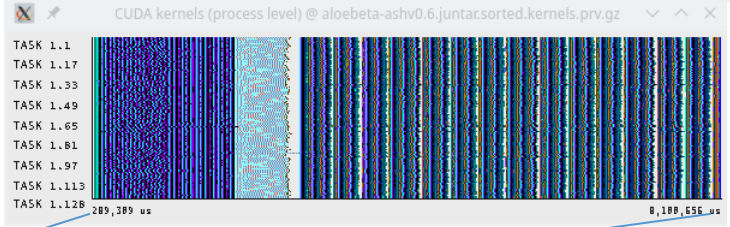
- Different microscopic behaviors … similar macroscopic duration !!!!

GPU ǁ efficiency: 91.2 %
Load balance: 96 %
Comm. Eff.: 94.7 %
Comp. Eff.: 93.5 %

Overlapping



Non Overlapping



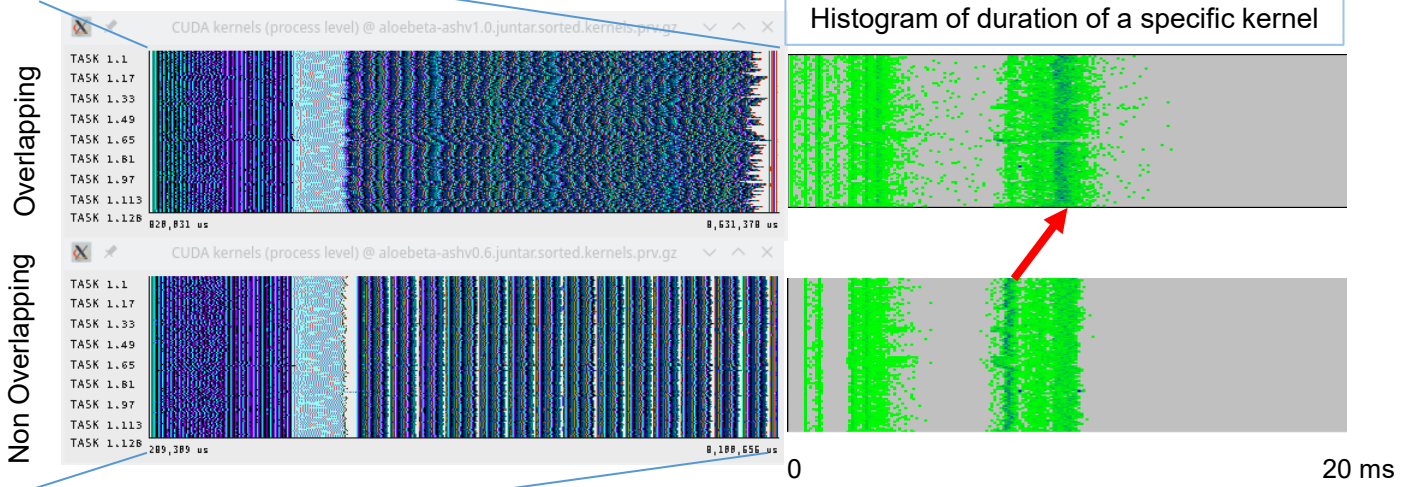
GPU ǁ efficiency: 85.3 %
Load balance: 94 %
Comm. Eff. 90.3 %
Comp. Eff. 100 %

To overlap or not to overlap

- Different microscopic behaviors … similar macroscopic duration !!!!

GPU ǁ efficiency: 91.2 %
Load balance: 96 %
Comm. Eff.: 94.7 %
Comp. Eff.: 93.5 %

Histogram of duration of a specific kernel



GPU ǁ efficiency: 85.3 %
Load balance: 94 %
Comm. Eff. 90.3 %
Comp. Eff. 100 %

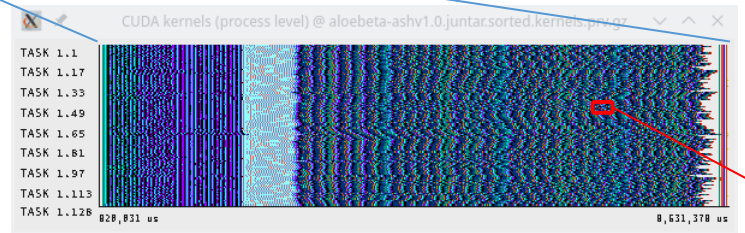
Communication overlap interferes in computation cost !!!

To overlap or not to overlap

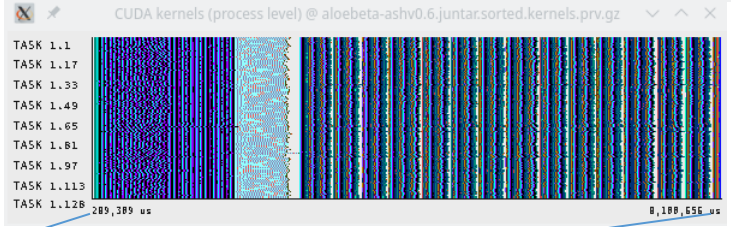
- Different microscopic behaviors … similar macroscopic duration !!!!

GPU ‖ efficiency: 91.2 %
Load balance: 96 %
Comm. Eff.: 94.7 %
Comp. Eff.: 93.5 %

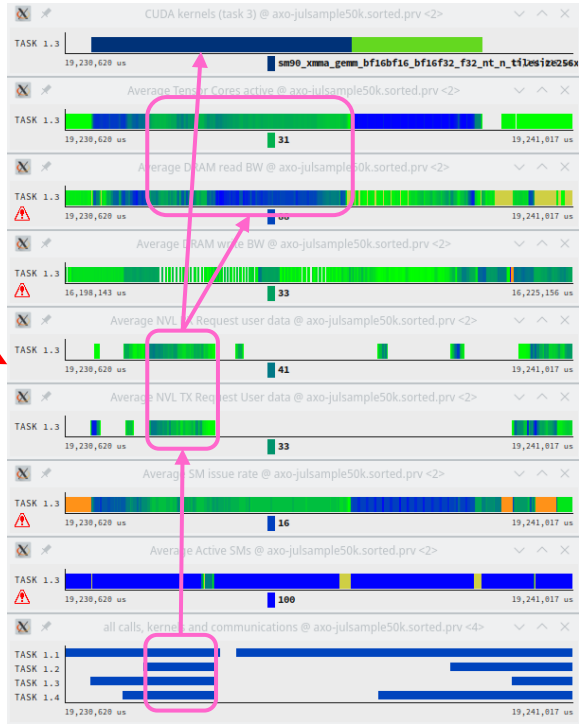
Overlapping



Non Overlapping



GPU ‖ efficiency: 85.3 %
Load balance: 94 %
Comm. Eff. 90.3 %
Comp. Eff. 100 %

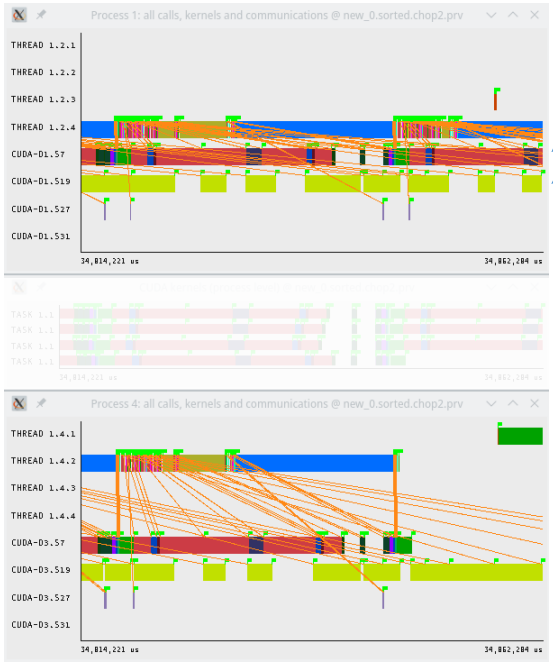


M. Clascá et al. "NSYS2PRV: detailed and quantitative analysis of large-scale GPU execution traces with Paraver". Cluster 2025

Computation interfering Communication?

Comp – Comm interferences ?

Tight zoom in fwd



Compute kernels

NCCL kernels

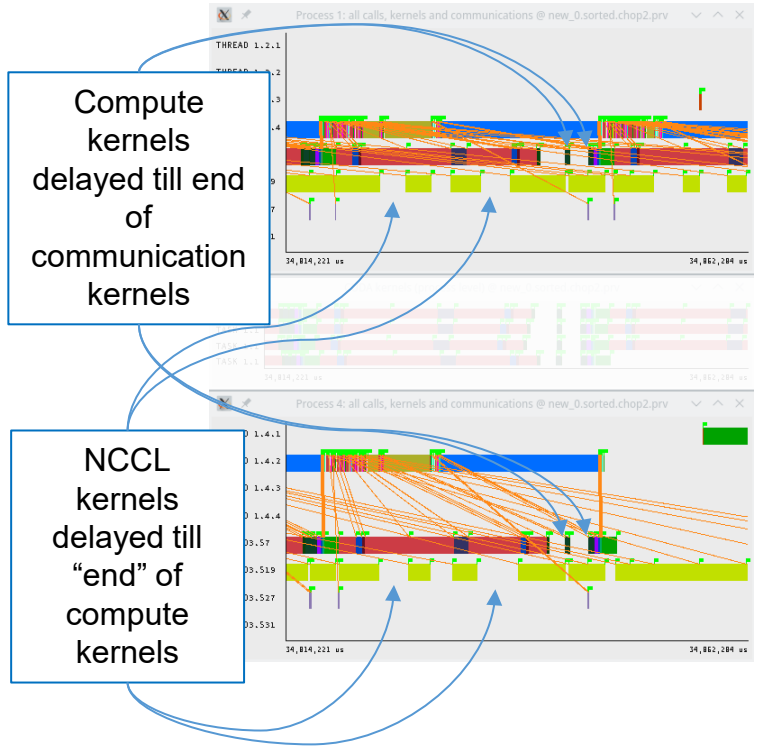
First process activity

Compute kernels 4 processes

Fourth process activiy

Comp – Comm interferences ?

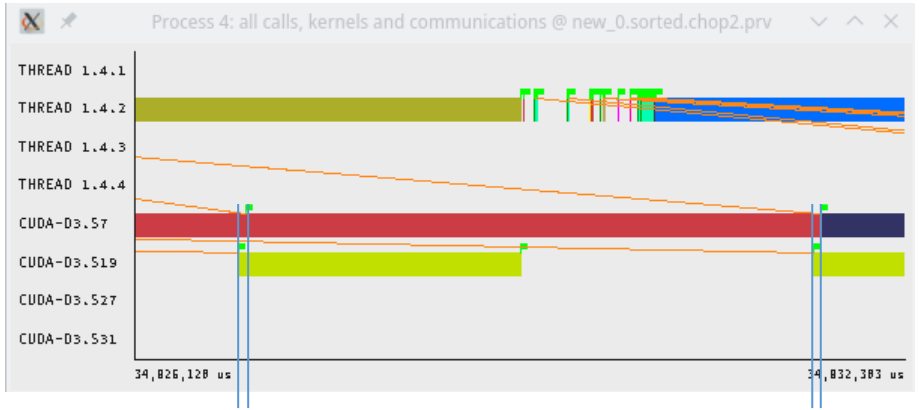
Tight zoom in fwd



Not quite

- What “end” means ??

Further zoom



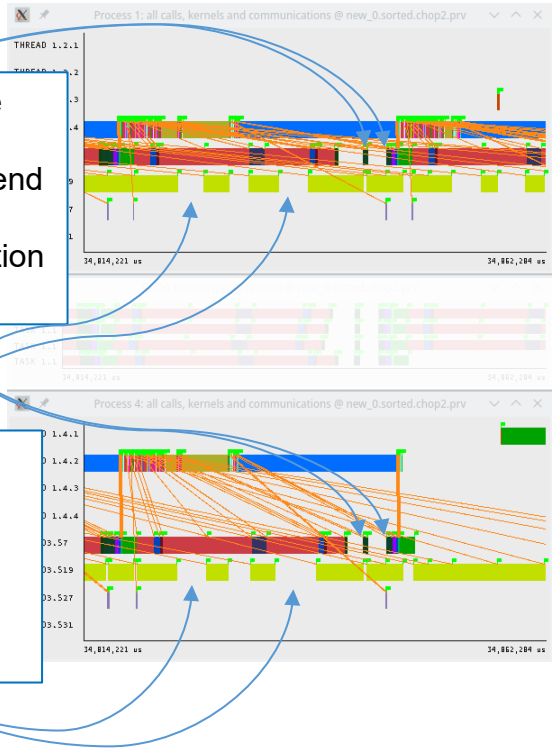
… almost end !!!!
Why?
Instrumentation timings bug ? → ask Nsight Systems. Not probable !
Synchronization ? With what ?
Missing data in trace? → Ask Nsight Systems, nsys2prv
Other ??

SM scheduling “features” + application “features” + …

Comp – Comm interferences ?

- Call it communication …
- … mean SM scheduling

Tight zoom in fwd



First process activity

Compute kernels 4 processes

Fourth process activiy

Actual Dependences

Compute kernels delayed till end of communication kernels

Resource contention !!!!

NCCL kernels delayed till “end” of compute kernels

That may or may not move actual data transfer into the critical path …

… that may change the criteria of actual communication bandwidth importance …

… scares

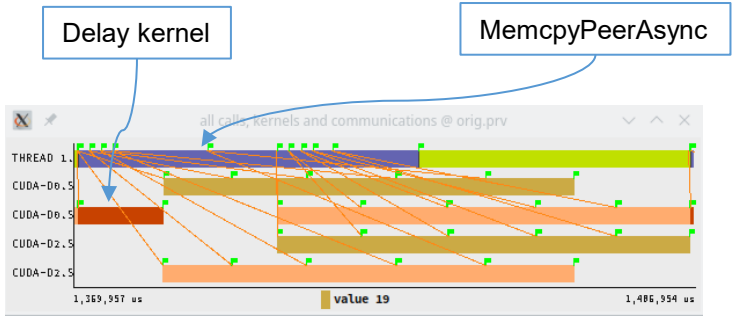
How to measure bandwidth?

Measuring BW? (p2pBandwidthLatencyTest)

- Measured PCIe bandwidth

Original

1 process
2 GPUs
5 reps
delay
blocked issues

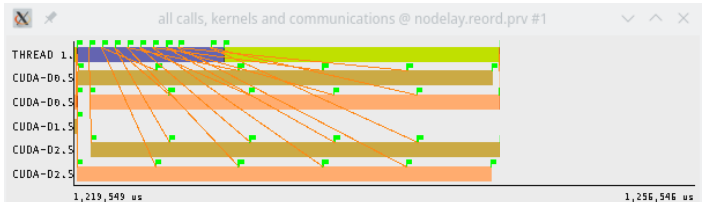



```
Bidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 1461.30  42.69  50.92  52.36
     1  45.01 1460.43  50.90  52.65
     2  50.58  50.99 1462.12  51.10
     3  51.72  51.65  51.05 1461.43
```

Reported BW improvement
19%

Insight based refactoring

5 reps
no delay
nterleave issues




```
Bidirectional P2P=Disabled Bandwidth Matrix (GB/s)
   D\D     0      1      2      3
     0 1446.78  60.82  63.38  63.17
     1  59.87 1454.65  60.01  59.72
     2  63.16  60.13 1456.96  58.71
     3  62.31  60.14  58.84 1457.85
```

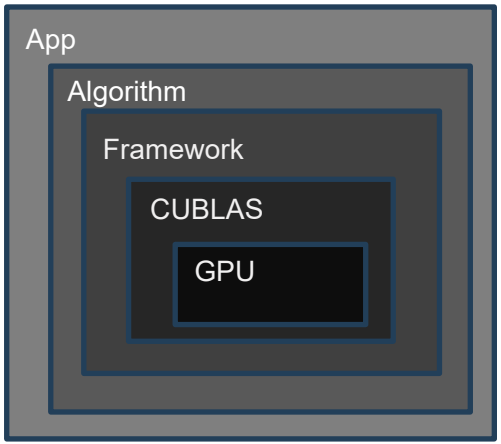
On  s and  s

Comparability

- Probabilistic & non linear recurrent algorithm

+

- Black matryoshkas

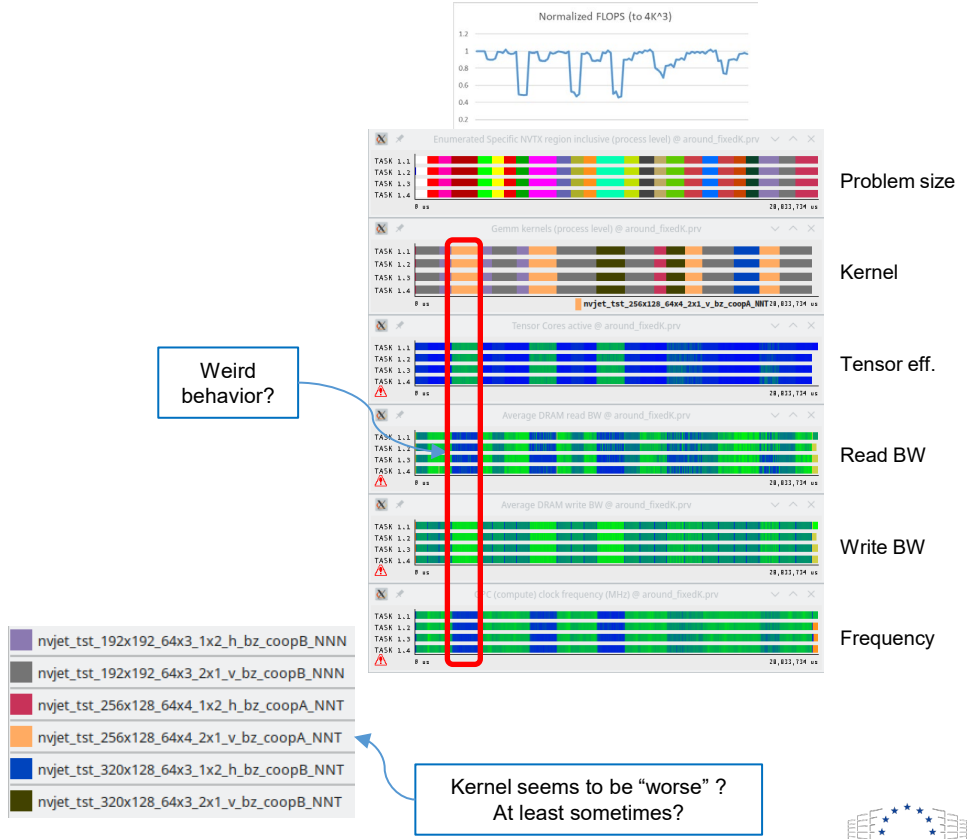


Difficult to ensure “fair” comparison

Gemm performance variability for large problem size?

Gemm kernels

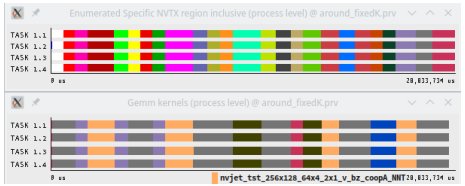
- Case :
  - M = [13184, 13696, 14208, 14720, 15232]
  - N = [7168, 7680, 8192, 8704, 9216]
  - K = [16384]
- I would expect:
  - High percentage of the peak of the device.
  - Very little difference across problem sizes
  - Not much variability along multiple runs of a given problem size
- Observed significant variability !!
  - Different kernels !!
  - Difference in Tensor core activity
  - Difference in frequency !!!!
  - Difference in Memory BW !!
  - Differences across instances of the same kernel



How many gemm variants exist?

Number of gemm kernels ?

Sweep M,N,Ks

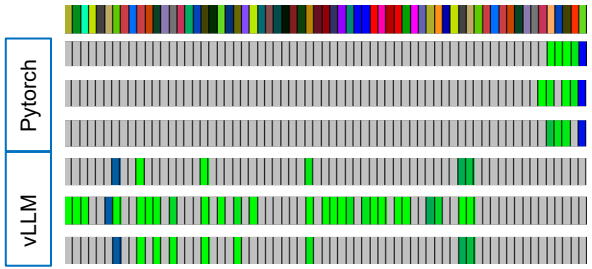


Problem size

Kernel 6

Kind of weird
- architecture ?
- code explosion ?
- “optimality” ?

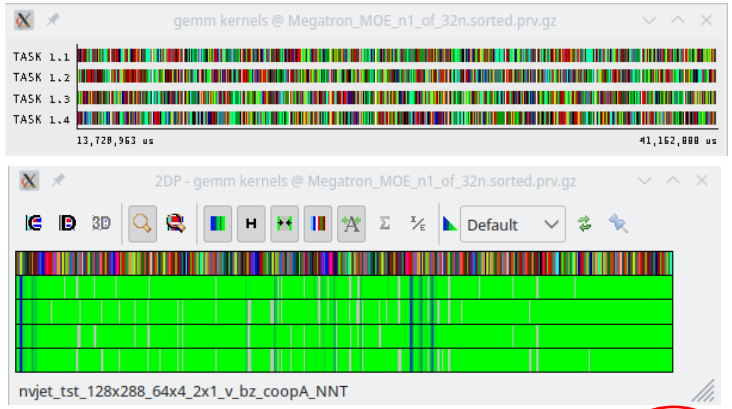
Pytorch & vLLM inferences



6

27

Megatron MoE



1

400

Same model, task …
… different frameworks

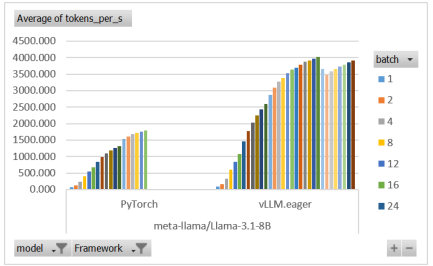
Pytorch vs vLLM inference ?

- Tried
  - offline execution of same prompts
  - Llama-3.1-8B
  - i:1024 , o:256
  - Impact of varying batch size?

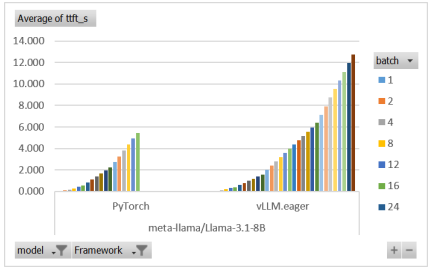
- vLLM “better”
  - Different OOM batch size
  - Tok/s: ~ 2.02
  - Ttft: ~ 0.73
  - Tpot: ~ 0.33
  - Difference slightly improves with batch size

How to compare? Same bs? Max bs? (bs160 both)

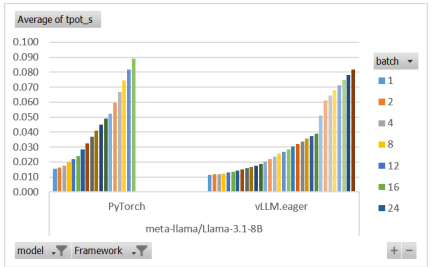
tok/s



ttft



tpot



Pytorch vs. vLLM: step

- Batch sizes 1 and 160

Application-level metrics

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
tok/s	64.6	1797.8	86.2	3634.6
ttft	0.041	5.46	0.029	3.980
tpot	0.015	0.089	0.012	0.029

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff	1.000	32.019	1.430	64.796
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

- Model:
  - Describe/explain
  - Question generator?

Pytorch vs. vLLM: step

- Batch sizes 1 and 160

Application-level metrics

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
tok/s	64.6	1797.8	86.2	3634.6
ttft	0.041	5.46	0.029	3.980
tpot	0.015	0.089	0.012	0.029

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff				64.796
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

Partial improvement on two metrics

Pytorch vs. vLLM: step

- Batch sizes 1 and 160

Application-level metrics

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
tok/s	64.6	1797.8	86.2	3634.6
ttft	0.041	5.46	0.029	3.980
tpot	0.015	0.089	0.012	0.029

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff				64.796
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

2x on computational scalability !!

Pytorch vs. vLLM: step

- Batch sizes 1 and 160

Application-level metrics

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
tok/s	64.6	1797.8	86.2	3634.6
ttft	0.041	5.46	0.029	3.980
tpot	0.015	0.089	0.012	0.029

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498

Huge improvement in computational scaling !!
But should have been even higher ?

A bit deeper on Comp. Eff.

- But still high level
- Complementary metrics
  - Hardware counters

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff	1.000	32.019	1.430	64.796
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

A bit deeper on Comp. Eff.

- But still high level
- Complementary metrics
  - Reasonable ?
  - Hint on reasons for reported efficiency

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time [illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.46	0.25	0.32

Non proportional increment of TC ops and instructions
"small" increment in reads
"Large" increment in writes !!!

scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

A bit deeper on Comp. Eff.

- But still high level
- Complementary metrics
  - Reasonable ?
  - Hint on reasons for reported efficiency

Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff				64.796
Kernel time (us)	3599703	22239113	2877157	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

REDUCTION in tensor core ops ??
Reduction in writes and instructions !!!

A bit deeper on Comp. Eff.

- But still high level
- Complementary metrics
  - Reasonable ?
  - Hint on reasons for reported efficiency

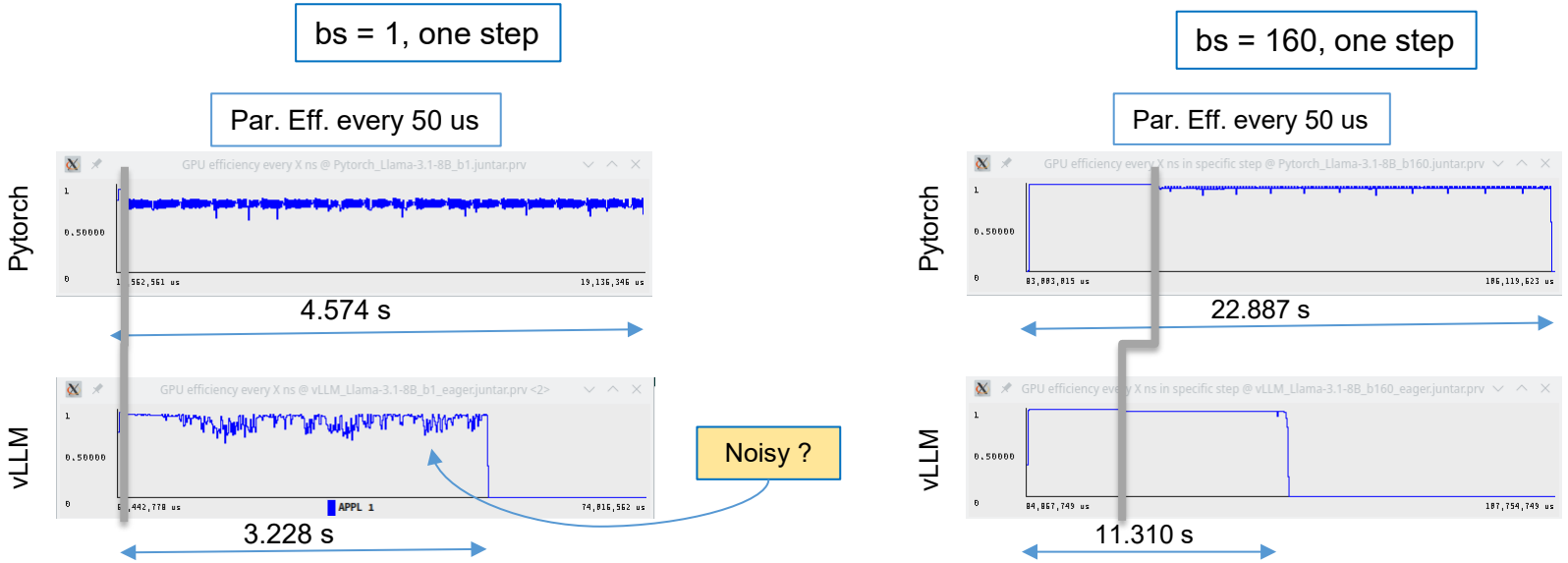
Model efficiencies

	PyTorch	PyTorch	vLLM	vLLM
bs	1	160	1	160
Total Eff.	0.79	25.04	1.12	50.70
Par. Eff.	0.79	0.97	0.89	0.98
LB	1.00	1.00	1.00	1.00
Comm. Eff.	1.00	1.00	1.00	1.00
Orch. Eff.	0.79	0.97	0.89	0.98
Comp. Eff. (scaled by bs)	1.00	25.90	1.25	51.57
Normalized total	1.000	31.862	1.420	64.498
Elapsed (us)	4580116	22886781	3203761	11309590
Time based eff	1.000	32.019	1.430	64.796
Kernel time (us)	[illegible]	[illegible]	[illegible]	11167915
Comp. Eff. (Not scaled)	1.00	0.16	1.25	0.32
scaled counts				
TC ops	1.00	16.47	0.20	17.11
Bytes read	1.00	5.35	1.00	3.39
Bytes written	1.00	81.03	0.52	11.25
SM issues	1.00	8.67	0.36	4.16

Similar tensor core ops
Less reads
LESS writes and instructions !!!

And more detail

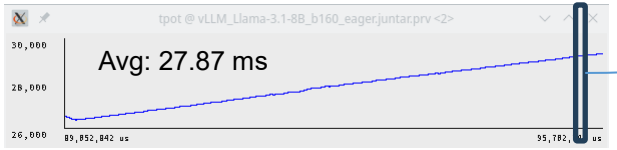
- Deeper level
- Batch size impact along time
  - Efficiency, prefill/decode balance, noise, …



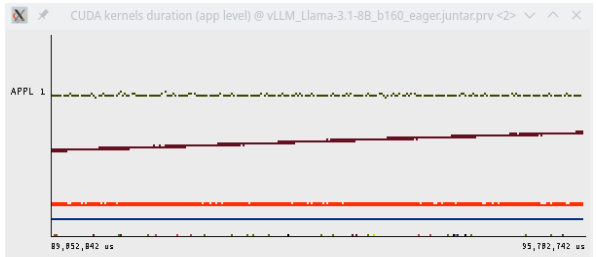
Sometimes not really great match with some of the AI estimations. Which ones, when?

vLLM decode. bs=160

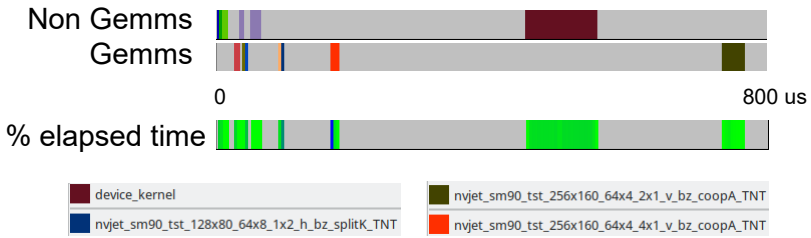
tpot



Timeline of individual kernel duration

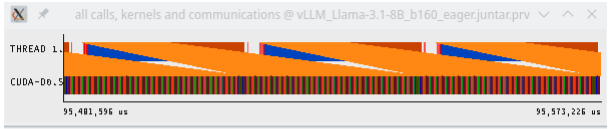


Histogram of kernel duration colored by id

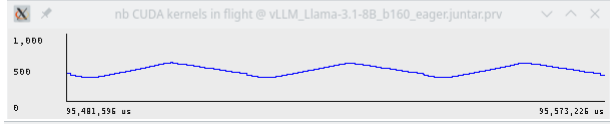


Generation of 3 tokens

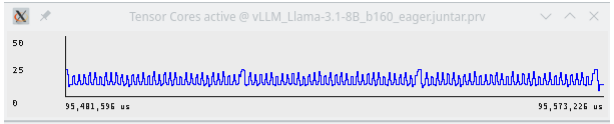
API calls and kernels



Kernels in flight

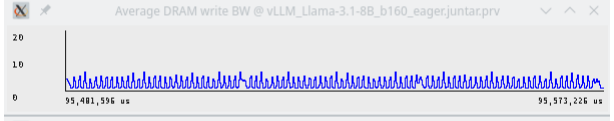


Tensor core active (%)



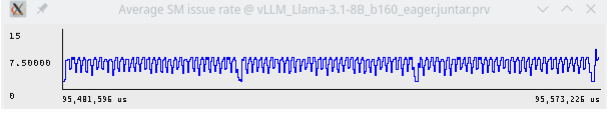
DRAM read BW (%)

DRAM write BW (%)



Active SMs (%)

SM issue rate



Role of CPUs in a GPU centric world

Life at “fine” granularities

- When “kernel launch overhead” approaches kernel execution time

vLLM, bs=1, generation of one token

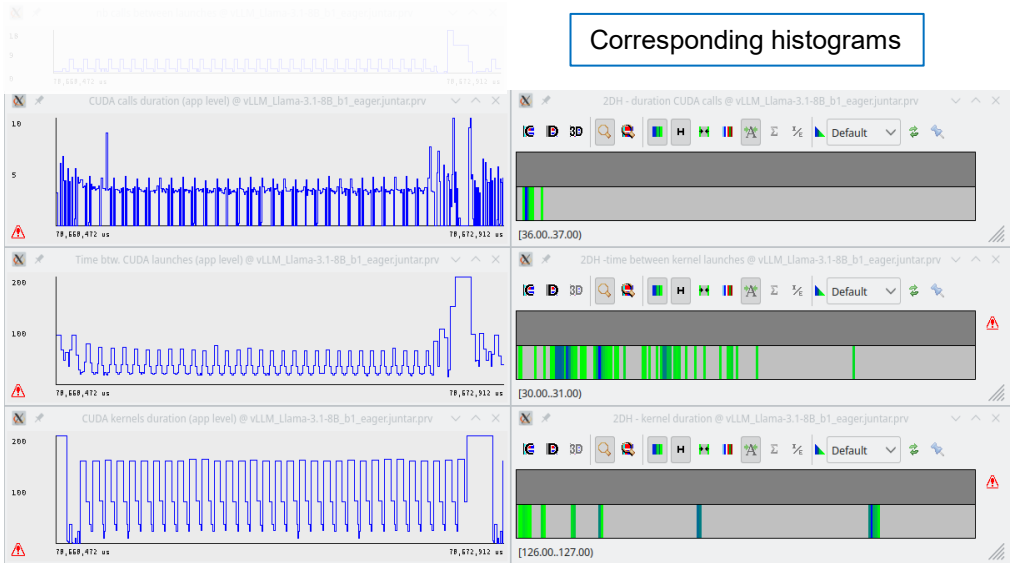
nb CUDA calls btw. launches

CUDA call duration

Time btw. Kernel launches

Kernel duration

Corresponding histograms



~3.5 us

Stats : Time

0

200 us

Life at “fine” granularities

- When “kernel launch overhead” approaches kernel execution time

vLLM, bs=1, generation of one token

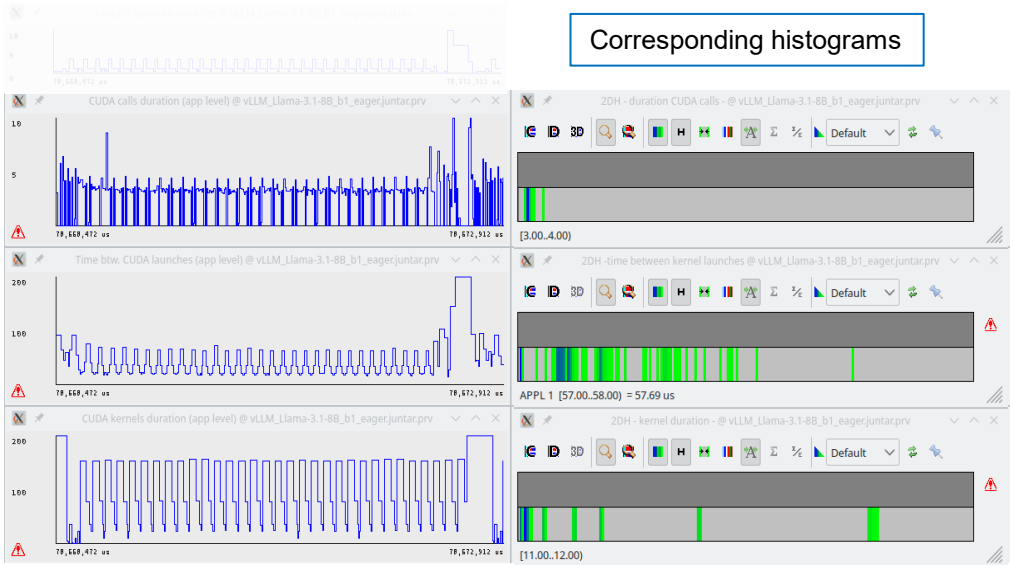
nb CUDA calls btw. launches

CUDA call duration

Time btw. Kernel launches

Kernel duration

Corresponding histograms



~3.5 us

Stats : # instances

0 200 us

Life at “fine” granularities

- Struggle to ramp up and fill the pipe
- Can get something out of the critical path?

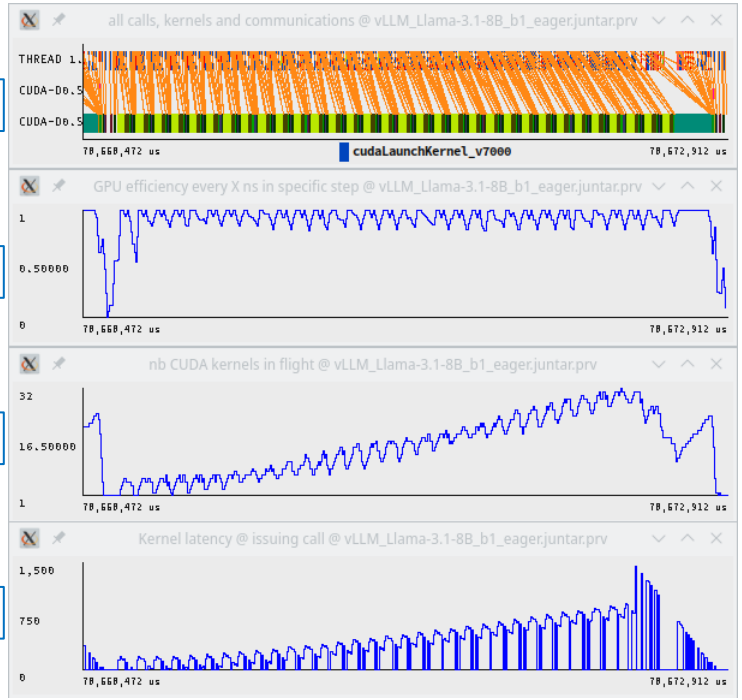
vLLM, bs=1, generation of one token

Call and kernels

Par. Eff. every 50 us

Kernels in flight

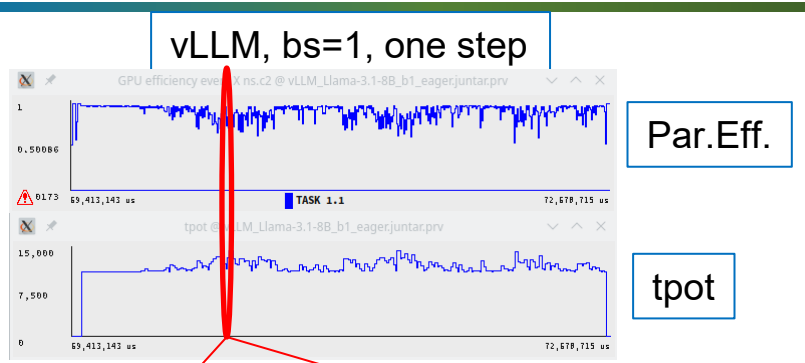
Kernel latency



Min: ~ 5.3 us

Noise ?

- Real “noise” ?
- Correlated to CPU activity ?
  - If so, can be taken out of the critical path?



Patterns?

- Short range, long range, …

vLLM, bs=1, 3 successive tokens

Cuda calls and kernels

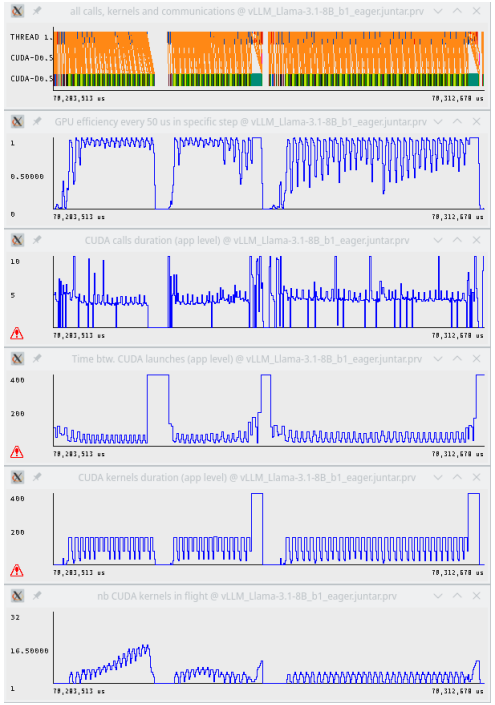
Par. Eff.

CUDA call duration

Time btw. Kernel launches

Kernel duration

Kernels in flight



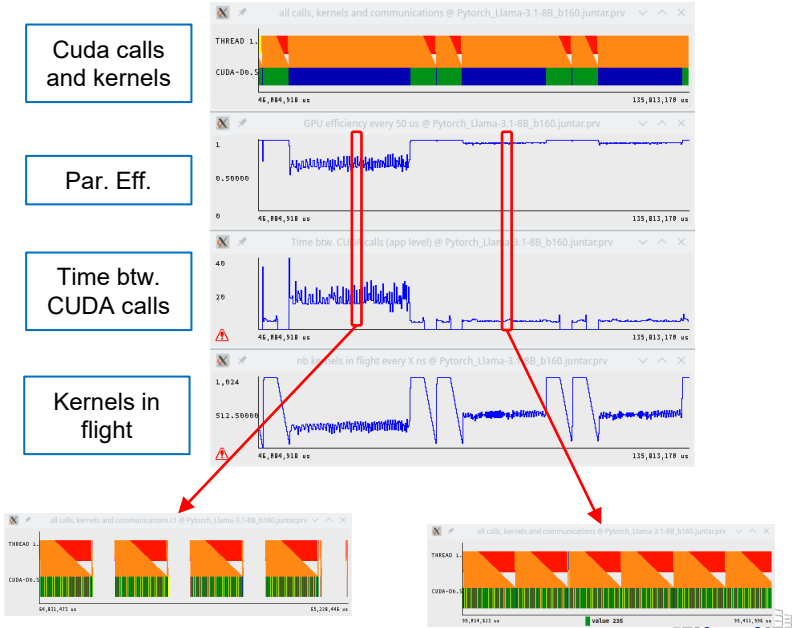
PyTorch, bs=160, 3 successive inferences

Cuda calls and kernels

Par. Eff.

Time btw. CUDA calls

Kernels in flight



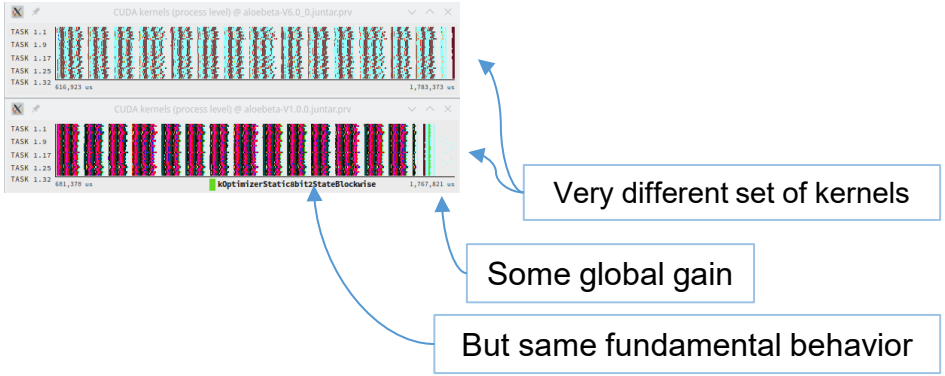
To conclude …

Life in AI times

“My dear, here we must run as fast as we can, just to stay in place.
And if you wish to go anywhere you must run twice as fast as that.”

Lewis Carroll. Through the Looking-Glass

From when you say is your framework version?
Three months ago? … ☹ … update !!

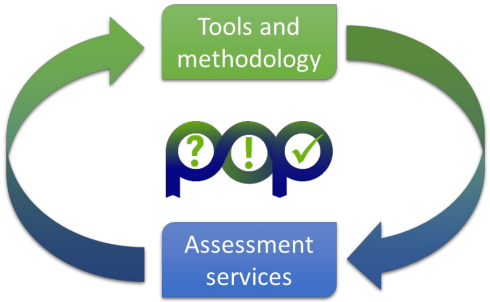


Vindicate an HPC-ish view @ AI land

We can leverage HPC Performance analysis tools, methodology and background to get **a lot insight** on the behavior of AI apps !

POP

- Performance optimization and productivity COE



FREE Services provided by the CoE

- **Parallel Application Performance Assessment**
  - Primary service
  - Initial analysis measuring a range of performance metrics to assess quality of performance and identify the issues affecting performance (at customer site)
  - If needed, undertakes further performance evaluations to identify the root causes of the issues found and qualify and quantify approaches to address them (recommendations)
- **Second Level Services**
  - Second level services may follow after conclusion of an initial performance assessment:
    - **Proof-of-concept**: explore the potential benefit of proposed optimisations by applying them to selected regions of the applications
    - **Correctne**: evaluate the correctness of hybrid MPI + OpenMP applications
    - **Ener udy**: investigate improvements of energy consumption or efficiency
    - ng consultancy for customers that choose to implement proposed
    - own

our experts and customer!

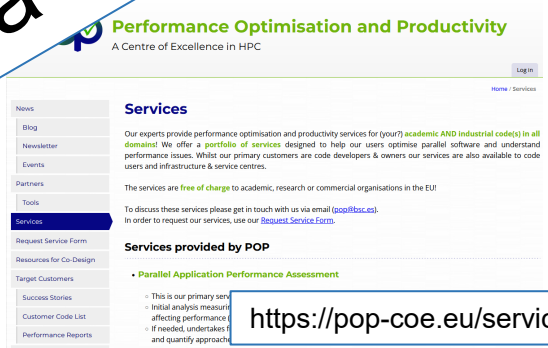
10

HORIZON-EUROHPC-JU-2023-C

Grant Agreem 43931

1 January 2024– 3 December 2026

Give POP a try



https://pop-coe.eu/services

EuroHPC Joint Undertaking

Performance Optimisation and Productivity 3

A Centre of Excellence in HPC

Contact:

- https://www.pop-coe.eu
- pop@bsc.es
- @POP_HPC
- youtube.com/POPHPC

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931. The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal and the Czech Republic.