GTC 2025 - Accelerated Computing / CUDA Technical Briefing

Software Engineer at Baseten
Core Developer at SGLang

Yineng Zhang

19th March 2025

DeepSeek V3 Kernel Optimization in SGLang Overview

SGLang / LLM Inference Optimization

- SGLang is a fast serving framework for large language models and vision language models
  - SGLang supported MLA in Sep 2024 https://lmsys.org/blog/2024-09-04-sglang-v0-3/
  - SGLang supported DeepSeek V3 in Dec 2024 https://github.com/sgl-project/sglang/releases/tag/v0.4.1
  - SGLang reached 100 tokens/s on H200 in March 2025. https://github.com/sgl-project/sglang/releases/tag/v0.4.4
- Issues
  - Triton backend is not efficient for long context triton backend
  - The initial version of the Block-wise FP8 kernel written with Triton has room for improvement fp8 kernel
  - The initial MTP version built on Triton backend can be improved

DeepSeek V3 Kernel Optimization in SGLang

SGLang / LLM Inference Optimization

- How to use CUDA to improve
  - Optimize DeepSeek V3 MLA Attention with FlashInfer

````
We need two sets of kernels for MLA:

self-attention on ragged tensor, w/o matrix absorption: **head_dim_qk=192, head_dim_vo=128**
cross-attention on paged-kv cache, w/ matrix absorption: **head_dim_qk=576, head_dim_vo=512 (K=V)**
and serving engines are expected to use different kernels according to use cases:

For decoding, use 2
For prefilling (w/o prefix-caching), use 1
For incremental prefilling/chunked-prefill, use the 1+2:

```
o_1, lse_1 = cross_attention(c_q, q_pe, c_kv) (c_q: (n, 128, 512), q_pe: (n, 128, 64), c_kv: (n_kv,
576), o_1: (n, 128, 512), lse_1: (n, 128))

o_2, lse_2 = self_attention(q, k, v_new) (q: (n, 128, 192), k: (n, 128, 192), v: (n, 128, 128), o_2:
(n, 128, 128), lse_2: (n, 128))

o, lse = merge(W_UV(o_1), lse_1, o_2, lse_2)
```
````

  - Optimize MTP on top of FlashInfer [#4218]
  - Optimize Block-wise FP8 with DeepGEMM [#4199]

DeepSeek V3 Kernel Optimization in SGLang

SGLang / LLM Inference Optimization

- How to use CUDA to improve

With the combination of FlashInfer, MTP, DeepGEMM, and Torch Compile optimizations on H200, it can achieve nearly **100 tokens/s**, which is currently the fastest open-source implementation.

```
SGL_ENABLE_JIT_DEEPGEMM=1 python3 -m sglang.launch_server --model deepseek-ai/DeepSeek-R1 --tp 8 --
trust-remote-code --enable-torch-compile --torch-compile-max-bs 1 --speculative-algo EAGLE --
speculative-draft lmsys/DeepSeek-R1-NextN --speculative-num-steps 3 --speculative-eagle-topk 1  --
speculative-num-draft-tokens 4 --enable-flashinfer-mla

python3 -m sglang.bench_one_batch_server --model None --base-url http://localhost:30000 --batch-size 1
2 4 8 --input-len 256 --output-len 256
```

Top 1 Thing CUDA Should Do

- JIT first
  - FlashInfer MLA uses JIT https://github.com/flashinfer-ai/flashinfer

    When integrating FlashInfer, I encountered a bug with CUDA Graph. Since FlashInfer MLA supports JIT, debugging and fixing it became easier

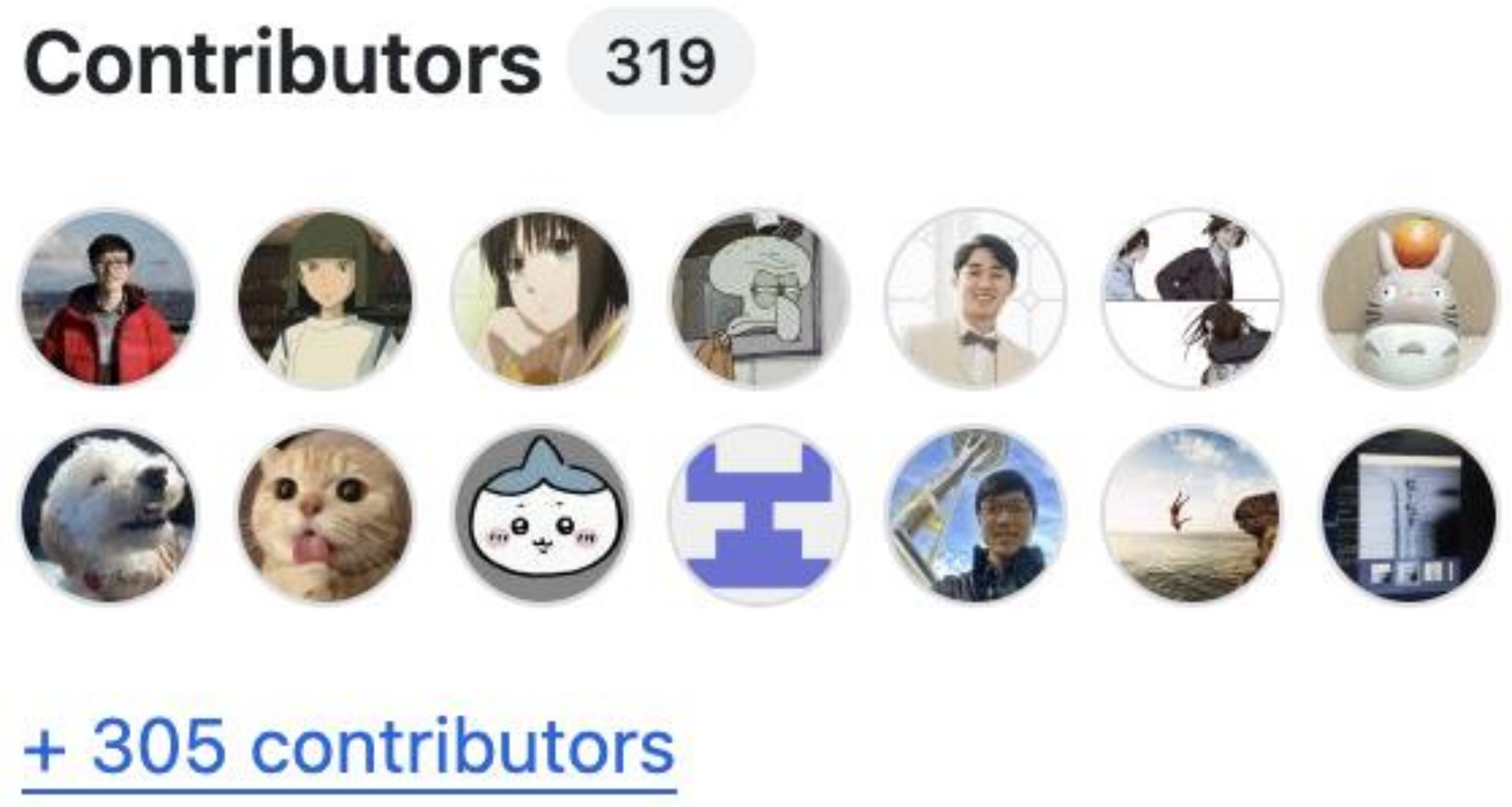
    https://github.com/flashinfer-ai/flashinfer/pull/822

  - DeepGEMM uses JIT https://github.com/deepseek-ai/DeepGEMM

    When integrating DeepGEMM into sgl-kernel at https://github.com/sgl-project/sglang/tree/main/sgl-kernel it is easy to add DeepGEMM as a third-party component due to its use of JIT

ACKNOWLEDGMENT

- Big Thanks
  - Zihao Ye from FlashInfer Team
  - Baizhou Zhang from SGLang Team
  - Pankaj Gupta from Baseten Team
  - Chao Wang, Ying Zhang, Ke Bao from Meituan Team
  - Fan Yin and 305 contributors from SGLang community



NVIDIA®

Thank You!