Intel® XeTLA: Templates Based Linear Algebra Library for Intel Xe GPU

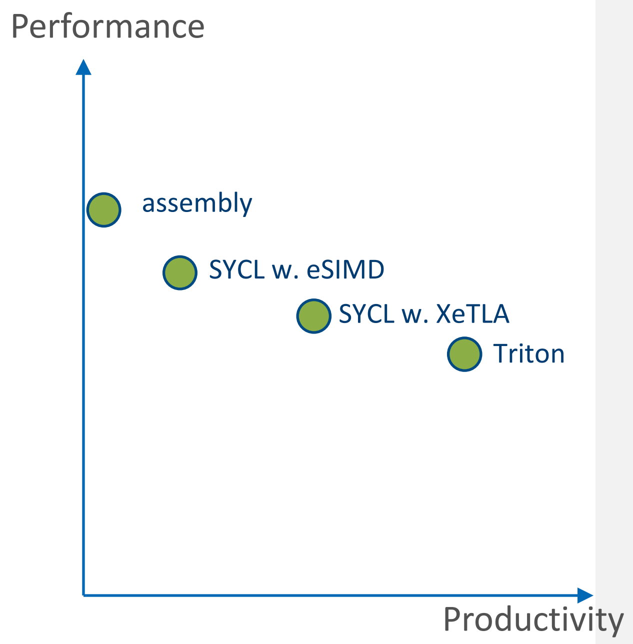
Fangwen Fu, Patric Zhao, Xiaodong Qiu, Eric Lin, Hong Jiang

Problem statement

- TensorRT-LLM delivers 2x performance improvement with software optimization: https://www.maginative.com/article/nvidias-groundbreaking-tensorrt-llm-doubles-inference-performance-of-language-models/
  - it provides a straightforward, open-source Python API that encapsulates the TensorRT Deep Learning Compiler, **optimized kernels from FasterTransformer**, pre-and post-processing, and multi-GPU/multi-node communication.
- Industry requires ease-of-use and highly optimized GPU kernels to extract hardware peak performance
  - Cutlass, Faster Transformer/Deepspeed kernel injection, AITemplate
  - Flash Attention v2: https://github.com/Dao-AILab/flash-attention
- What is the solution for Intel XeGPU (e.g. Intel® Data Center GPU Max Series)?
  - Intel® XeTLA: https://github.com/intel/xetla

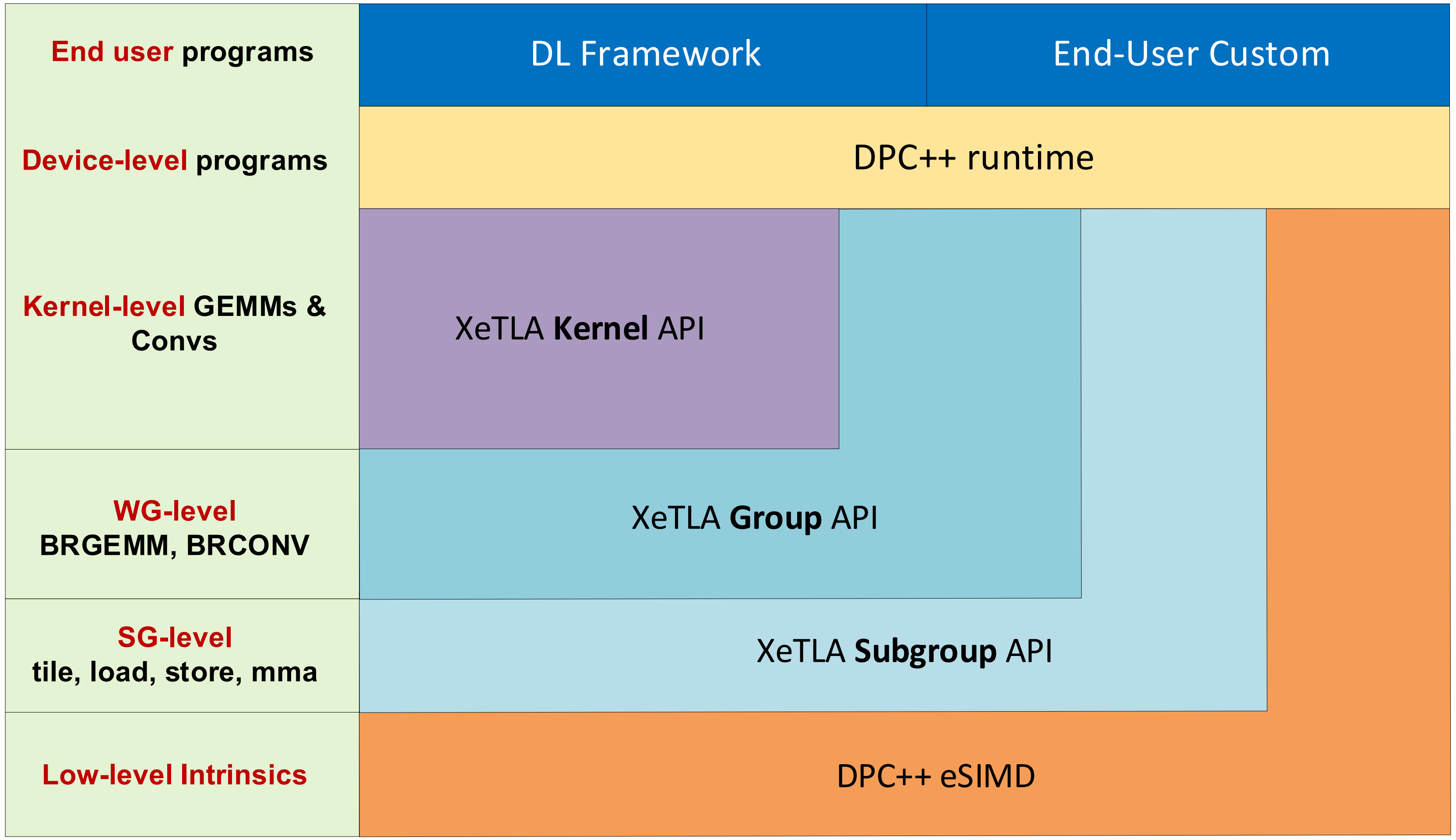
Overview

- XeTLA (Xe Template Linear Algebra) goal: high kernel performance with development productivity by leveraging flexibility of meta programming
  - oneAPI provides an Explicit SIMD SYCL extension (ESIMD) for lower-level Intel GPU programming: https://intel.github.io/llvm-docs/doxygen/group__sycl__esimd.html
  - XeTLA is a collection of SYCL/eSIMD template, target for the **building blocks** to implementing high-performance GEMM, CONV and adjacent computations with reasonable efforts
  - XeTLA provides reusable, flexible C++ templates for subgroup level, workgroup level, kernel level primitives. The developers can implement & optimize kernels by specializing and tuning with different data types, tiling policies, algorithms, fusion policies, etc.
  - XeTLA primitives hide/abstract different Xe HW implementation details from SYCL/DPC++ developers and yet provide enough performance control knobs (e.g. prefetch distance, tiling)



XeTLA is built upon SYCL/eSIMD

XeTLA Architecture



Intention is to promote high-level abstraction, e.g. workgroup level programming.

Group level GEMM API

XeTLA hides lots of HW details and developer can focus on algorithm level, such as how to map workgroup into whole workload

- BRGEMM template provides the workgroup level computation
- Accumulator (e.g. matAcc) is exposed to developer for easy control and pre/post-operations by epilogue
- Low level API is encapsulated to template API for easy to use
- Similar Constructions as Nvidia CUTLASS and easy to migration

Code Link

```
// Configure brgemm
using brgemm_t = xetla::group::brgemm_selector_t<
                        datatype_bf16,         // input datatype for A
                        datatype_bf16,         // input datatype for B
                        mem_layout::row_major, // memory layout for A
                        mem_layout::row_major, // memory layout for B
                        mem_space::global,     // memory reading from global mem for
A
                        mem_space::global,     // memory reading from global mem for
B
                        8,                     // Buffer A alignment
                        8,                     // Buffer B alignment
                        data_type_acc,         // accumulator data type for
intermediate resutls
                        tile_shape,            // computation tile shape
                        k_stride,              // k dim stride in each iteration
                        mma_engine::xmx,       // compute engine
                        gpu_arch::Xe>          // GPU arch
              ::brgemm;
// Configure epilogue
using epilogue_t = xetla::group::epilogue_t< … >;

auto e_esimd = Queue.submit([&](handler& cgh) {
cgh.parallel_for<Test>(nd_range, [=](nd_item<3> item) SYCL_ESIMD_KERNEL{
// Configure memory descriptor, and group level start offset
    mem_desc_input_a md_a({A}, {matrix_k, matrix_m, lda}, {start_k, start_m});
    mem_desc_input_b md_b({B}, {matrix_n, matrix_k, ldb}, {start_n, start_k});
    mem_desc_output_c md_c({C}, {matrix_n, matrix_m, ldc}, {start_n, start_m});

// Allocate accumulation buffer && clear
    brgemm_t::matAcc_t matAcc(0);
// Get local id
    brgemm_t::work_group_t g(item.get_local_linear_id());
// Launch brgemm
    brgemm(g, matAcc, md_a, md_b, inner_loop_count);
// Launch epilogue
    epilogue(g, matAcc, md_c);
}
```

Group level brgemm API

Device kernel using group level APIs

Inside XeTLA

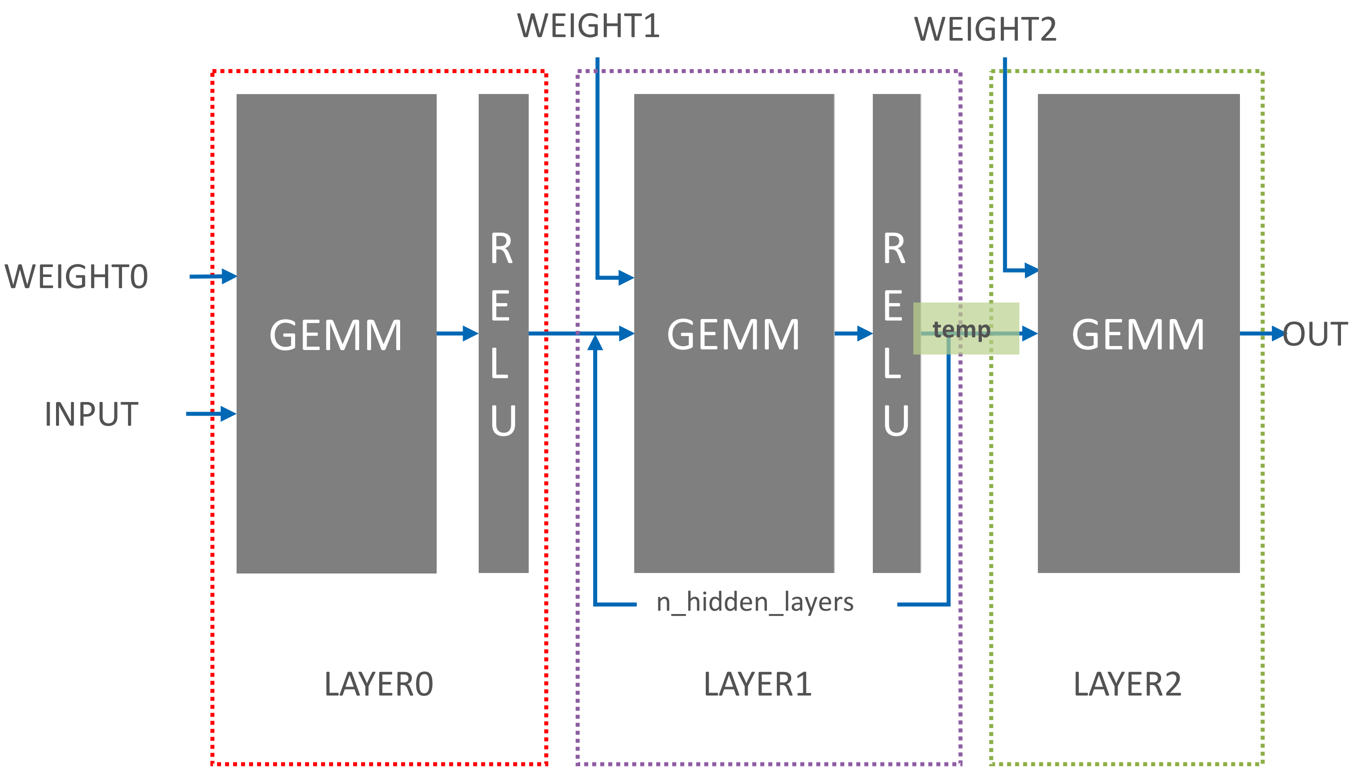
- Close to metal programming through Sycl ESIMD
- Bare metal programming requires deep understanding on Xe GPU architecture
  - MMA programming: tiling, VNN layout, matA/B caching with correct sequence, matC reuse
  - block2d message programming (transpose vs non-transpose, 64/256B cacheline, address alignment)
  - Cooperative prefetch across threads, L1/L2 prefetch
  - Periodic sync mechanism for workgroup locality control
  - Many more…
- XeTLA abstracts them in the way easy to programming
  - Subgroup level tile based load/store/prefetch, MMA
  - Workgroup level brgemm/brconv, flexible sync with named barrier, etc.

Simple subgroup level programming with XeTLA

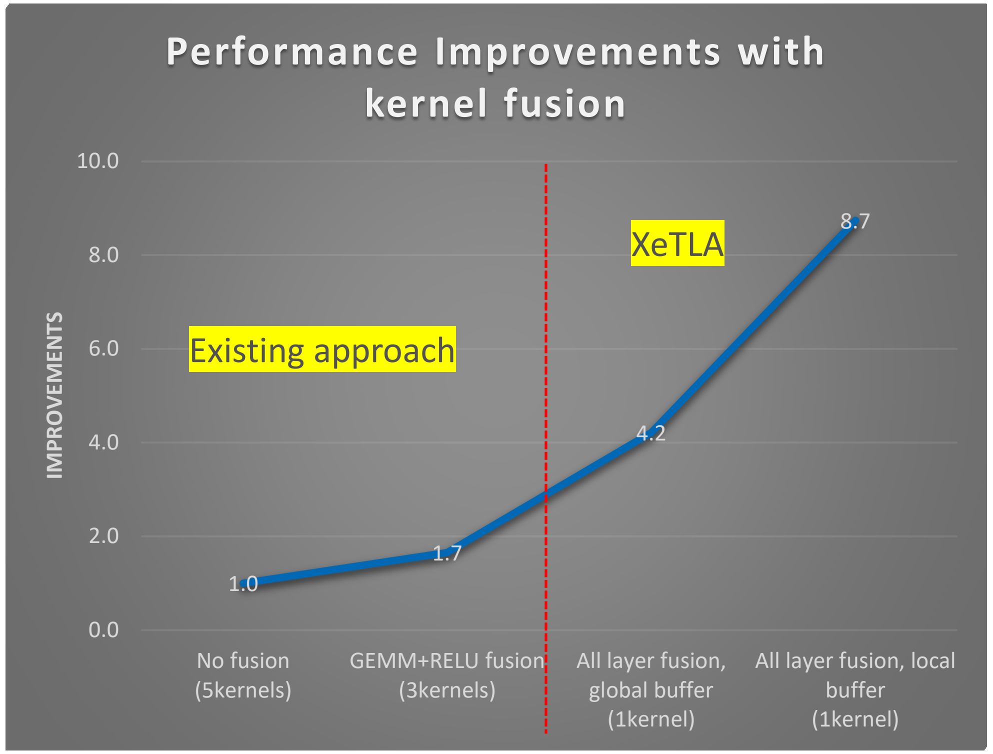
```
// GEMM function
void gemm(work_group_t& g, matAcc_t& matAcc,
mem_desc_a_t& md_a, mem_desc_b_t& md_b, int
loop_count){
// Allocate matA and matB in register
    subgroup::tile_t<dtype_a, matA_tile_shape_t>
matA;
    subgroup::tile_t<dtype_b, matB_tile_shape_t>
matB;

// Compose payload for memory access
    subgroup::mem_payload_t<dtype_a,…>
matA_payload(md_a);
    subgroup::mem_payload_t<dtype_b,…>
matB_payload(md_b);

// GEMM inner loop, reduction along k dim
    for(int i = 0; i < loop_count; i++){

// Load matA and matB data from memory to
register
        subgroup::tile_load(matA, matA_payload);
        subgroup::tile_load(matB, matB_payload);

// Update the memory address
        matA_payload.update_tdesc(k_stride);
        matB_payload.update_tdesc(k_stride);

// Do matrix-multiple-accumulation
        subgroup::tile_mma(matAcc, matAcc, matB,
matA);
```

NeRF: MLP Layer Fusion with XeTLA

- ❑ Network Structure



- ❑ Performance improvement with fusion

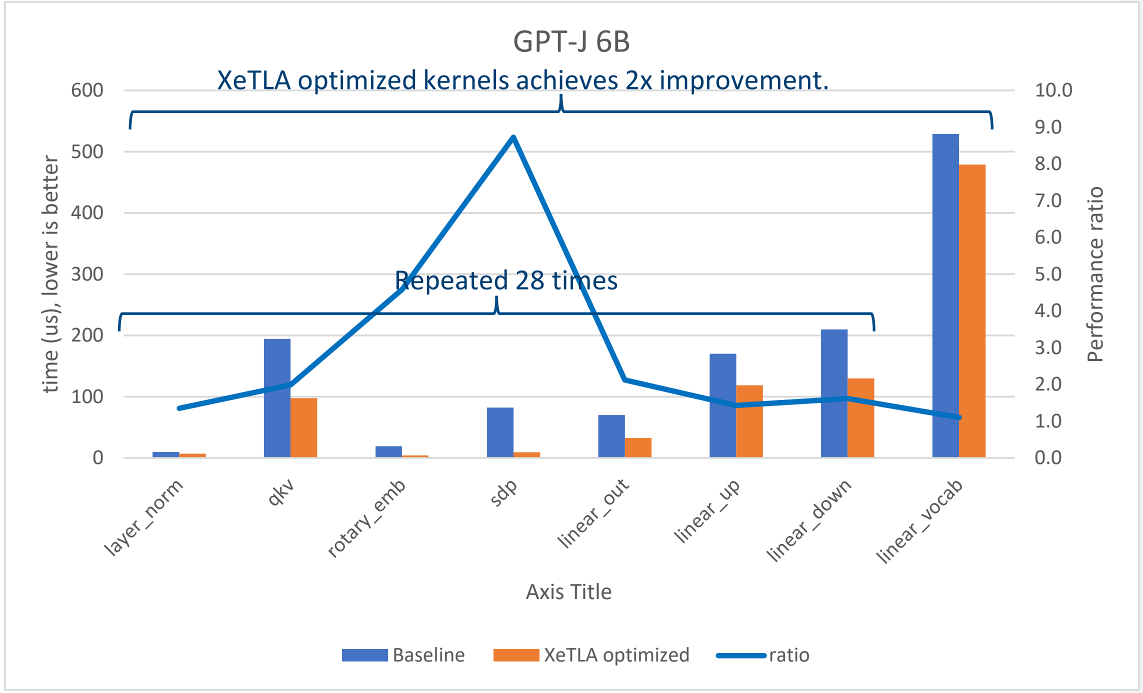


Note:
- Tested with INPUT_M=1M, n_hidden_layers=2;

intel.

Performance Optimization for Small LLM Models

- XeTLA are open sourced and integrated in Intel Pytorch EXtension (IPEX).
  - Achieved peak performance for GPT inference in Intel Max GPU.
- GPT inference speed up by 2x within 1 week
  - Replacing oneDNN kernels with XeTLA kernels and applying aggressive fusion.
- XeTLA optimized LLM models can achieve 70-80% performance in Intel GPU Max 1550 (1 Tile) compared to 1 card A100, given that Max 1550 only has 60% HBM BW.



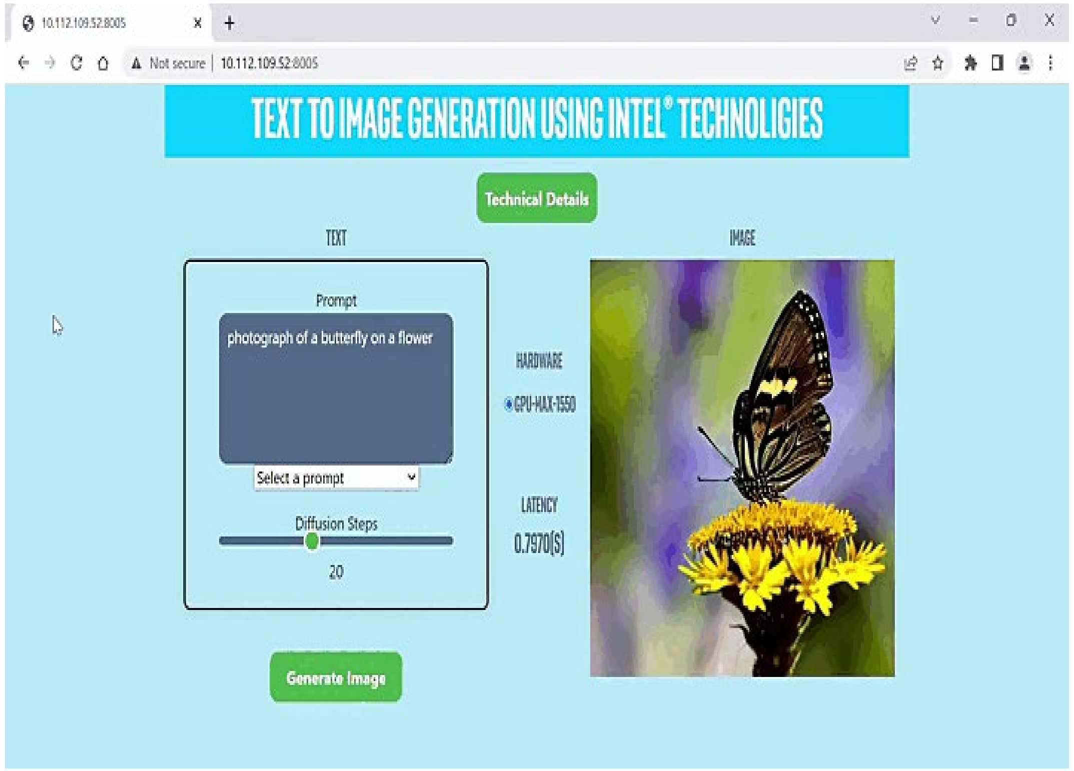
Note: performance data is collected by June, 2023. Latest oneDNN has performance improvement.

Model	Parameter Size	Input/Output #token	Precision	Max GPU IPEX Optimized	A100-PCIe-80GB (DeepSpeed, kernel injection, Greedy search)	A100-PCIe-40GB (DeepSpeed, kernel injection, Greedy search)	Total Latency Ratio	
				Total latency (s)	Total latency (s)	Total latency (s)	A100-80GB/PVC	A100-40GB/PVC
GPT-J	6B	32/32	FP16	0.427	0.311	0.342	72.83%	80.09%
LLaMa	7B	32/32	FP16	0.48	0.355	0.388	73.96%	80.83%
LLaMa	13B	32/32	FP16	0.92	0.629	0.699	68.37%	75.98%
GPT-J	6B	1024/128	FP16	1.874	1.343	1.403	71.66%	74.87%
LLaMa	7B	1024/128	FP16	2.074	1.58	1.745	76.18%	84.14%
LLaMa	13B	1024/128	FP16	3.984	2.786	3.121	69.93%	78.34%

Stable Diffusion Performance

Stable Diffusion BF16 inference perf

- Fused scaled dot product, 6x speedup
- ~1.6x perf boost in E2E


10.112.109.52:8005

Not secure | 10.112.109.52:8005

TEXT TO IMAGE GENERATION USING INTEL® TECHNOLIGIES

Technical Details

TEXT

Prompt

photograph of a butterfly on a flower

Select a prompt

Diffusion Steps

20

HARDWARE

● GPU-MAX-1550

LATENCY

0.7970(S)

IMAGE

Generate Image

Latency(sec)	Step20	Step50
JAX	1.30	3.08
JAX+XeTLA	0.79	1.84

Next step

- Features
  - More developer friendly
    - decouple and abstract tile (tensor with shape constraint), memory layout, thread layout, index … (CuTe like)
    - Error handling & debuggability
    - Acceptable performance with default recipe/schedule
    - Documentation
  - More shape coverage
  - StreamK/CONV support
- Programming Language
  - Current: XeTLA APIs are implemented with eSIMD; XeTLA users code must be eSIMD => Basically the kernel is eSIMD
  - Target: XeTLA APIs are implemented with SYCL and invoke_simd, which plays a “inline assembly” role => The kernel is mostly SYCL with small portion of eSIMD

Thank you!