标题？
TensorRT
FasterTransformer
FasterTransformer ---DPCT
xFasterTransformer

Feng Ding

TensorRT

- Parser， Network
- Tensor, Nodes, Graph
- Layer fusion
- Tactic （autotuning）
- Memory management
- Execute
- iPlugin

TensorRT

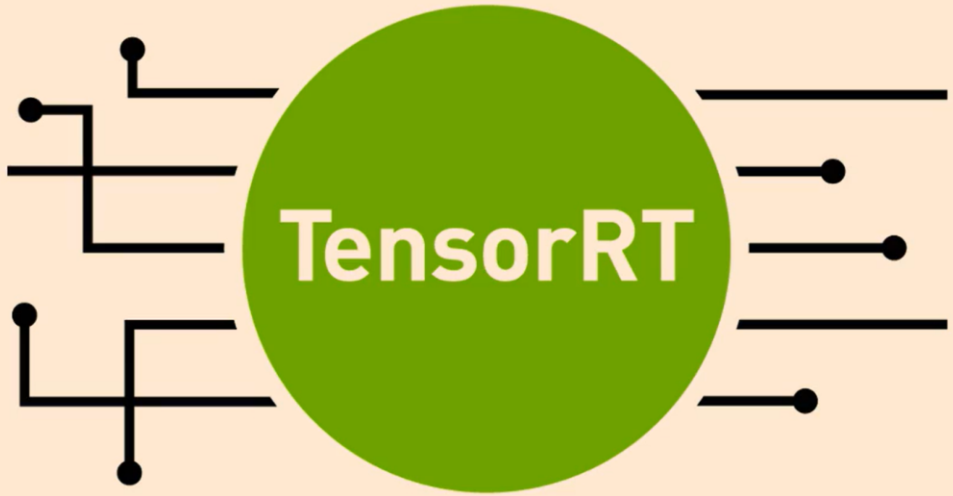
Overview
Arch
Roadmap

Ter

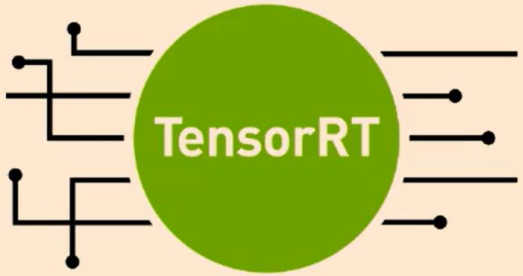
TensorRT

DNN inference compiler

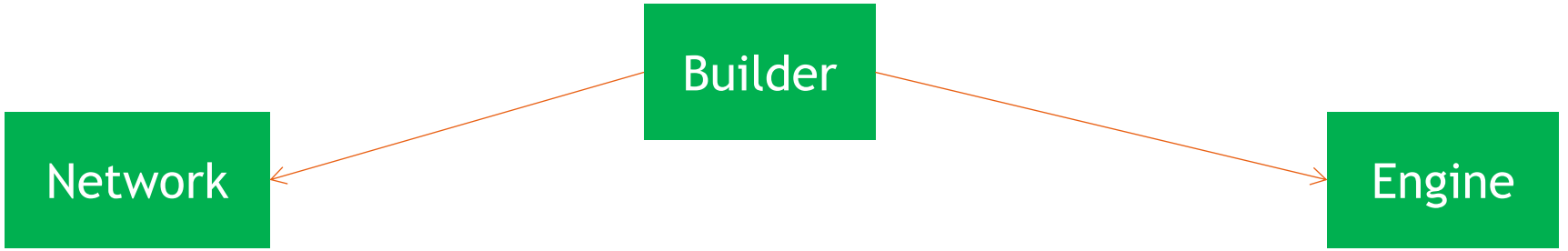
- TensorRT is versatile DNN graph compiler
- Performs kernel fusion
- Uses autotuning to select best kernels
- Emits optimized engines to execute on GPUs
- Supports custom plug-ins
- Low precisions (int8, fp8, etc.)



PyTorch → ONNX → TensorRT



TensorRT



App Network Description
Owns API Layers & Tensors
Responsible for
- pre-build validation
- per-layer output size computation

Maps Network to Engine
- Graph Optimization
- Tactic/Format Optimization
- Memory Optimization
- Int8 Calibration
- Engine Building

Execution Engine
- Serialization
- Execution
- Profiling
- *Debugging*
- *Memory Tracking*

Network layer -> graph node

Parser -> Network-> Graph

- Caffe parser, onnx parser, ….
- Network layers -> nodes
  softmax layer -> softmax nodes
  plugin layer -> plugin node
- Graph {
  Tensors;
  Nodes; (kernels)
  InputTensors;
  OutputTensors;
  Regions; (memory graph) // Allocating memory for each tensor only for the duration it's usage
  }

Graph optimization -> New Graph

- Layer Fusion https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#enable-fusion

  patterns （30？） conv + relu

- Tactic

  data type, layout (format)

  cuda core, tensor core

  reformat -> node -> reformat .... -> kernel auto tuning

- Memory management

  Region graph

  cudaMalloc is blocking

- Execute

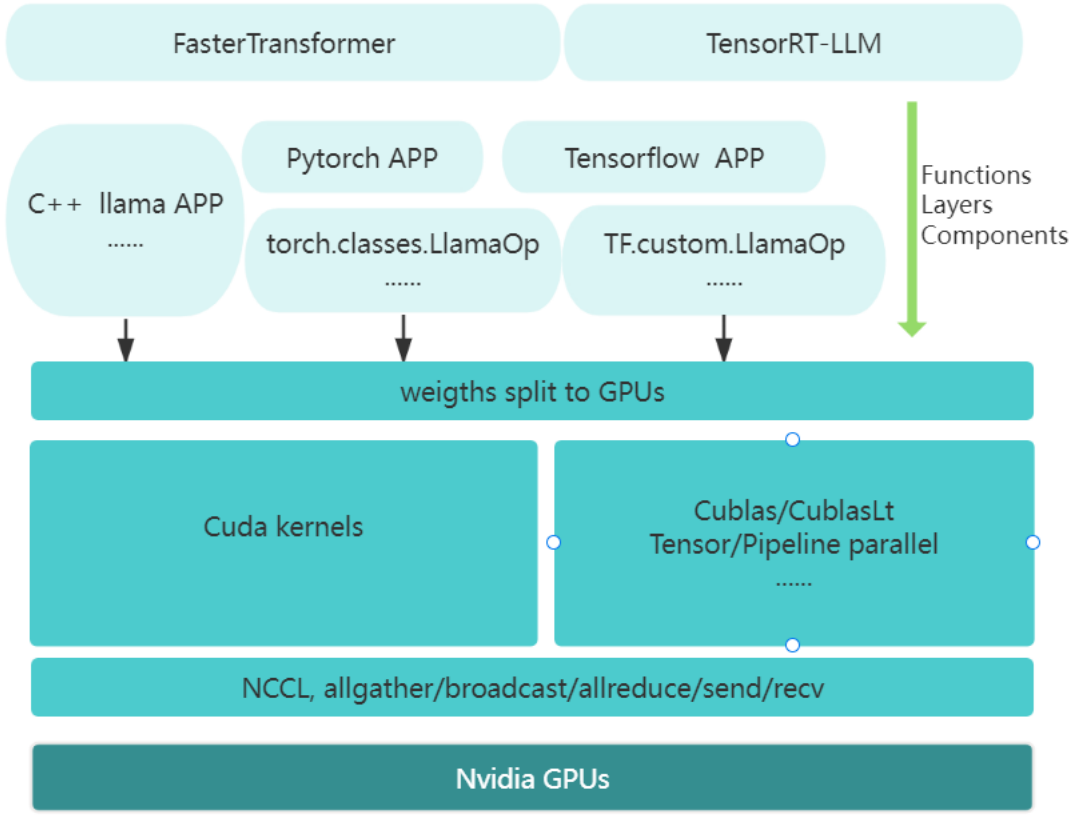
  Engine： weights, tactic choice, kernel arguments, memory graph, tensor binding/index/strides

  serialization/deserialization

- iPlugin

  addPlugin -> Initalize, input/output tensor, execute func, register list.

FasterTransformer



- Weights

torch.load() safetensors import safe_open

```
#### filename  ../meta-llama-Llama-2-7b-chat-hf-w4-g128-awq/pytorch_model.bin
model.embed_tokens.weight torch.Size([32000, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.self_attn.q_proj.qweight torch.Size([4096, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.q_proj.qzeros torch.Size([32, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.q_proj.scales torch.Size([32, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.self_attn.k_proj.qweight torch.Size([4096, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.k_proj.qzeros torch.Size([32, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.k_proj.scales torch.Size([32, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.self_attn.v_proj.qweight torch.Size([4096, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.v_proj.qzeros torch.Size([32, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.v_proj.scales torch.Size([32, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.self_attn.o_proj.qweight torch.Size([4096, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.o_proj.qzeros torch.Size([32, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.self_attn.o_proj.scales torch.Size([32, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.self_attn.rotary_emb.inv_freq torch.Size([64]) torch.float32 <class 'torch.Tensor'>
model.layers.0.mlp.gate_proj.qweight torch.Size([4096, 1376]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.gate_proj.qzeros torch.Size([32, 1376]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.gate_proj.scales torch.Size([32, 11008]) torch.float16 <class 'torch.Tensor'>
model.layers.0.mlp.up_proj.qweight torch.Size([4096, 1376]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.up_proj.qzeros torch.Size([32, 1376]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.up_proj.scales torch.Size([32, 11008]) torch.float16 <class 'torch.Tensor'>
model.layers.0.mlp.down_proj.qweight torch.Size([11008, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.down_proj.qzeros torch.Size([86, 512]) torch.int32 <class 'torch.Tensor'>
model.layers.0.mlp.down_proj.scales torch.Size([86, 4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.input_layernorm.weight torch.Size([4096]) torch.float16 <class 'torch.Tensor'>
model.layers.0.post attention layernorm.weight torch.Size([4096]) torch.float16 <class 'torch.Tensor'>
```

- Kernels -> Layers -> models
- Pytorch wrapper

  torch.classes.FasterTransformer.LlamaOp
- Distribution, NCCL

  Tensor parallel, Pipeline parallel

```
Model.forward() {
  // Handle first step   max_input_length?
  invokeInputIdsEmbeddingLookupPosEncoding
  invokeBuildDecoderAttentionMask
  context_decoder.forward() {
      invokeGetPaddingOffsetAndCuSeqLens()
   invokeGeneralT5LayerNorm()
   self_attention_layer_->forward()
   invokeGeneralLayerNorm or invokeGeneralAddResidualT5PreLayerNorm
      ffn_layer_->forward()
   invokeAddBiasAttentionFfnResidual or invokeAddBiasResidual
  }
  // Handle next tokens
 for (step=0; steps < output_len; step++) {
    gpt_deocder.forward() {
        for (layer=0; layer < num_layer; layer++) {
            invokeGeneralT5LayerNorm()
            self_attention_layer_->forward()
            invokeGeneralLayerNorm()  or  invokeGeneralAddResidualT5PreLayerNorm()
            ffn_layer_->forward()
            invokeAddBiasAttentionFfnResidual() or invokeAddBiasResidual()
        }
    }
      invokeGeneralT5LayerNorm()
    dynamic_decoder()
  }
}
```

TensorRT-LLM

- FasterTransformer
  - + TensorRT addPlugin

    TensorRT-LLM/cpp/tensorrt_llm/plugins/api/tllmPlugin.cpp -> initTrtLlmPlugins() ->
  - + TensorRT build Engine,
  - + TensorRT graph optimizer
  - + python binding

== TensorRT-LLM

```
static std::array pluginCreators
    = { creatorPtr(identityPluginCreator),
        creatorPtr(bertAttentionPluginCreator),
        creatorPtr(gptAttentionPluginCreator),
        creatorPtr(gemmPluginCreator),
        creatorPtr(moePluginCreator),
#if ENABLE_MULTI_DEVICE
        creatorPtr(sendPluginCreator),
        creatorPtr(recvPluginCreator),
        creatorPtr(allreducePluginCreator),
        creatorPtr(allgatherPluginCreator),
        creatorPtr(reduceScatterPluginCreator),
#endif // ENABLE_MULTI_DEVICE
        creatorPtr(layernormPluginCreator),
        creatorPtr(rmsnormPluginCreator),
        creatorPtr(smoothQuantGemmPluginCreator),
        creatorPtr(layernormQuantizationPluginCreator),
        creatorPtr(quantizePerTokenPluginCreator),
        creatorPtr(quantizeTensorPluginCreator),
        creatorPtr(rmsnormQuantizationPluginCreator),
        creatorPtr(weightOnlyGroupwiseQuantMatmulPluginCreator),
        creatorPtr(weightOnlyQuantMatmulPluginCreator),
        creatorPtr(lookupPluginCreator),
        creatorPtr(loraPluginCreator),
      };
```

- Paged attention and flash decoding code is cubin

- Hands on ?

$ python3 demo_txt2img.py "a beautiful photograph of Mt. Fuji during cherry blossom"

- Running StableDiffusion pipeline
- |----------------|-------------|
- | Module | Latency |
- |----------------|-------------|
- | CLIP | 5.69 ms |
- | UNet x 30 | 3144.48 ms |
- | VAE-Dec | 146.74 ms |
- |----------------|-------------|
- | Pipeline | 3297.14 ms |
- |----------------|-------------|
- Throughput: 0.30 image/s

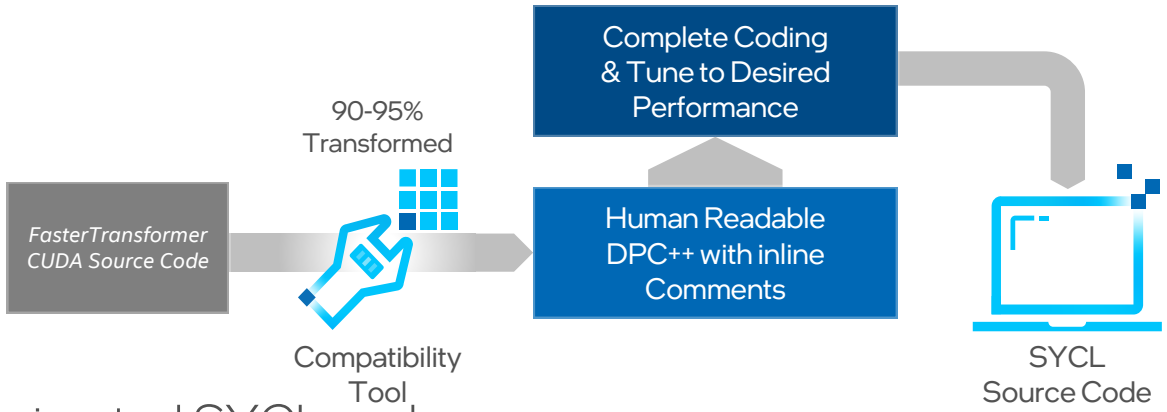
FasterTransformer DPCT DPC++ Compatibility Tool

---- Rapid CUDA to SYCL Migration with DPCT/SYCLomatic

- Over **96%** CUDA code automatically migrated by DPCT

- Manually code editing on DPCT output:
  - Rewrite inline PTX code with C++
  - Replace cublasLT with oneMKL call
  - Remove unnecessary CUDA API's
  - Some misc. changes

Intel DPC ++ Compatibility Tool Usage Flow



- Debug the compilation and runtime errors in the migrated SYCL code

- Tune the SYCL code performance with VTune and Unified Tracing and Profiling Tool

- https://github.com/intel-sandbox/FasterTransformer-dpct

FasterTransformer DPCT

- 1. Clone the required Github repository to your local environment.
- 2. Run the cmake command and generate the Makefile

```
$ mkdir build && cd build
$ cmake -DSM=80 -DCMAKE_BUILD_TYPE=Release ..
```

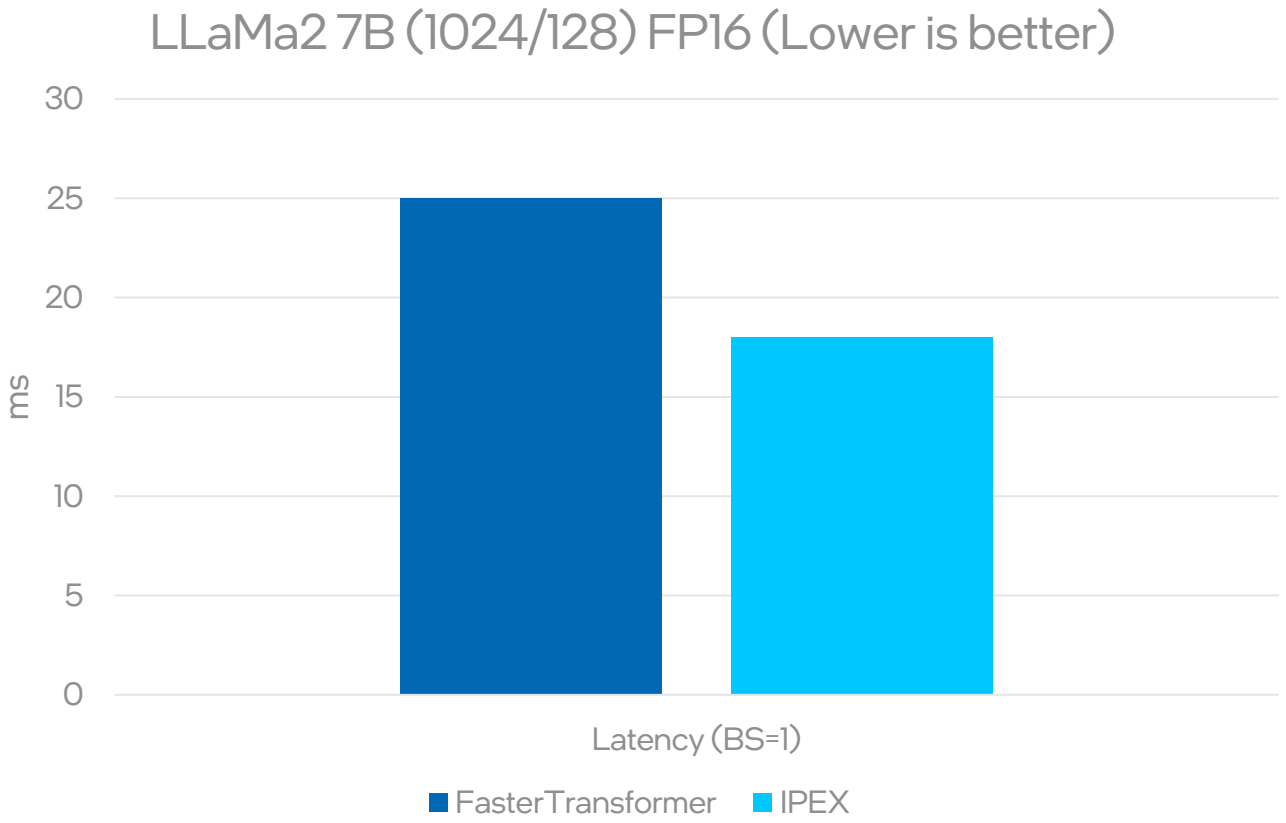
- 3. Generate the compilation database with intercept-build command

```
$ intercept-build make
```

- 4. Do the migration with dpct/c2s command

```
$ dpct --cuda-include-path=/usr/local/cuda -p ./compile_commands.json --in-root=../ --out-root=dpct_output --gen-build-script
```

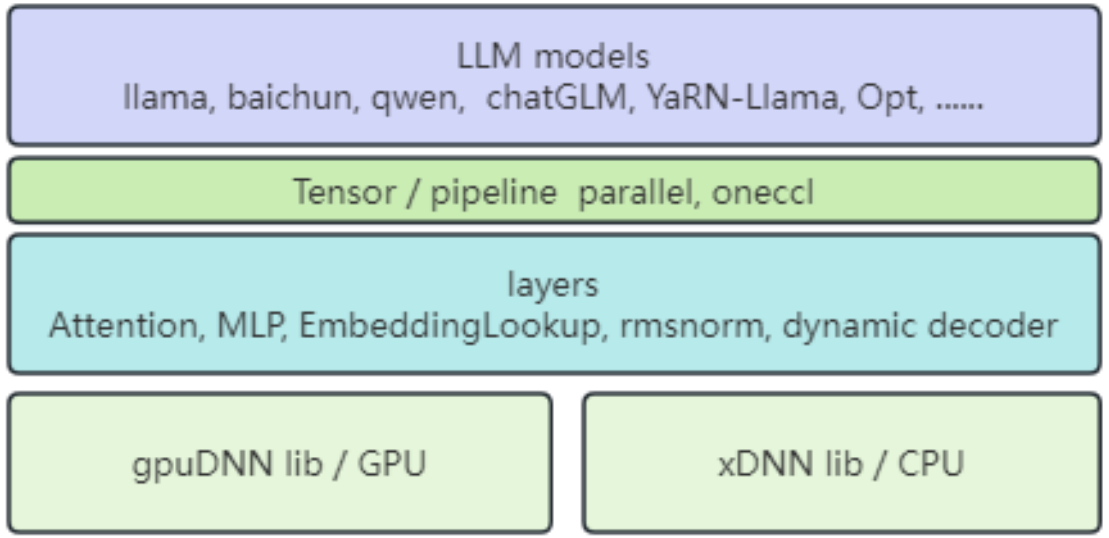
Out of Box Performance vs. IPEX on Max 1550



Out of box performance is 28% slower than the optimized IPEX in LLama2 7B inference

xFastertransformer

- https://github.com/intel-sandbox/gpuDNN
- https://github.com/intel/xFasterTransformer



- Gpu fs, pvc , arc ....


intel

Back up

An Example of Code Editing - inline PTX code migration

CUDA : fma.rn.f16x2

```cuda
inline __device__ uint32_t fma(uint32_t a, uint32_t b, uint32_t c)
{
    uint32_t d;
    asm volatile("fma.rn.f16x2 %0, %1, %2, %3;\n" : "=r"(d) : "r"(a), "r"(b), "r"(c));
    return d;
}
```

```sycl
inline uint32_t fma(uint32_t a, uint32_t b, uint32_t c)
{
    uint32_t d;

    uint16_t lowa=a&0xFFFF;
    uint16_t lowb=b&0xFFFF;
    uint16_t lowc=c&0xFFFF;

    uint16_t higha=(a>>16)&0xFFFF;
    uint16_t highb=(b>>16)&0xFFFF;
    uint16_t highc=(c>>16)&0xFFFF;

    sycl::half lowaf = sycl::bit_cast<sycl::half, short>(lowa);
    sycl::half lowbf = sycl::bit_cast<sycl::half, short>(lowb);
    sycl::half lowcf = sycl::bit_cast<sycl::half, short>(lowc);

    sycl::half highaf = sycl::bit_cast<sycl::half, short>(higha);
    sycl::half highbf = sycl::bit_cast<sycl::half, short>(highb);
    sycl::half highcf = sycl::bit_cast<sycl::half, short>(highc);

    sycl::half lowdf = lowaf * lowbf + lowcf;
    sycl::half highdf = highaf * highbf + highcf;

    uint16_t lowd = sycl::bit_cast<short>(lowdf);
    uint32_t highd = sycl::bit_cast<short>(highdf);

    d=(highd <<16) | lowd;

    return d;
}
```

Next Step

- Further performance optimization
- More data types support - fp8, int4 ...
- Migration of other LLM in FasterTransformer
- Migration of FasterTransformer-based AI frameworks from CSPs

Summary

- FasterTransformer is a popular light-weight NLP inference engine adopted by multiple CSP's
- Rapid migration of FasterTransformer Llama model demonstrated the effectiveness of DPCT in AI application domain
- The issues found in the migration can help improve DPCT in migrating other AI workloads

SYCL Runtime Issue

SYCL Spec - A New Device Memory Free API

CTST-5766 / CMPLRLLVM-54788

- SYCL spec has no synchronized memory free API
- “sycl::free” auto-migrated from “cudaFree” leads to GPU memory being deallocated before its release from a kernel running
- DPCT fixed the error by having to introduce an additional explicit sync for each “sycl::free” call with performance penalty
- SYCL spec need provide synchronized memory free API to improve the quality of migrated SYCL code

```
15
16 void LoadData(int * p_dst, int * p_src, int val, unsigned int len)
17 {
18       int * p_m;
19
20       cudaMalloc(&p_m,(len*sizeof(int)));  // Allocate temporary memory region
21       cudaMemcpy(p_m, p_src, sizeof(int) * len, cudaMemcpyHostToDevice);
22       invokeD2DCpyAdd(p_dst,p_m,val,len);
23       cudaFree(p_m);        //Released temporary memory pointed by "p_m"
24 }
```

```
25 void LoadData(int * p_dst, int * p_src, int val, unsigned int len)
26 {
27  dpct::device_ext &dev_ct1 = dpct::get_current_device();
28  sycl::queue &q_ct1 = dev_ct1.in_order_queue();
29       int * p_m;
30
31       p_m = (int *)sycl::malloc_device(
32          (len * sizeof(int)), q_ct1); // Allocate temporary memory region
33       q_ct1.memcpy(p_m, p_src, sizeof(int) * len).wait();
34       invokeD2DCpyAdd(p_dst,p_m,val,len);
35       sycl::free(p_m, q_ct1); // Released temporary memory pointed by "p_m"
37 }
```

```
$dpct main.cu
$icpx -fsycl main.dp.cpp
$./a.out 1000
length = 1024000
10100
$./a.out 10240 //Error happens with larger data size
length = 10485760
100 //Incorrect!
```

Code Conversion Issue

Incorrect Compilation Database and Build Script

- CTST-5423 - Incorrectly duplicated nvcc command lines in compilation database
- CTST-5524 - DPCT internal error. Migration rule causing the error skipped. Migration continues
- CTST-5546 - DPCT incomplete header search paths options in generated Makefile.dpct
- CTST-5646 - String values provided to -isystem option in compilation database cause error in migration

Better CUDA NVCC Dialect Support

- Multiple NVCC dialects can't be correctly parsed by Clang FE of DPCT: CTST-5517, CTST-5518, CTST-5519 and CTST-5520
- DPCT will issue parsing error on unsupported dialect without additional info
- Such error messages are misleading and easily misunderstood as migration tool's failure
- FR CTST-5577 has been submitted to improve the support and diagnostic on unsupported NVCC dialect

```
621 xFastertransformer-XPU-master/src/fastertransformer/kernels/matrix_vector_multiplication.cu:144:35:
    error: use of overloaded operator '=' is ambiguous (with operand types 'half2' (aka '__half2') and
    'void')
622   144 |             input_val_0[m_i]      = {input_val.x, input_val.y};
623       |
```

Performance Issue

Main Bottleneck - Memory Bandwidth Bounded oneMKL GEMM

- Vtune profiling shows the main bottleneck is oneMKL GEMM function bounded by PVC memory bandwidth
- MKLD-16877 has been opened to MKL team with a reduced test case
- Further optimization can be done by replacing oneMKL with other optimized GEMM implementation on PVC