

Migration FasterTransformer from CUDA to SYCL

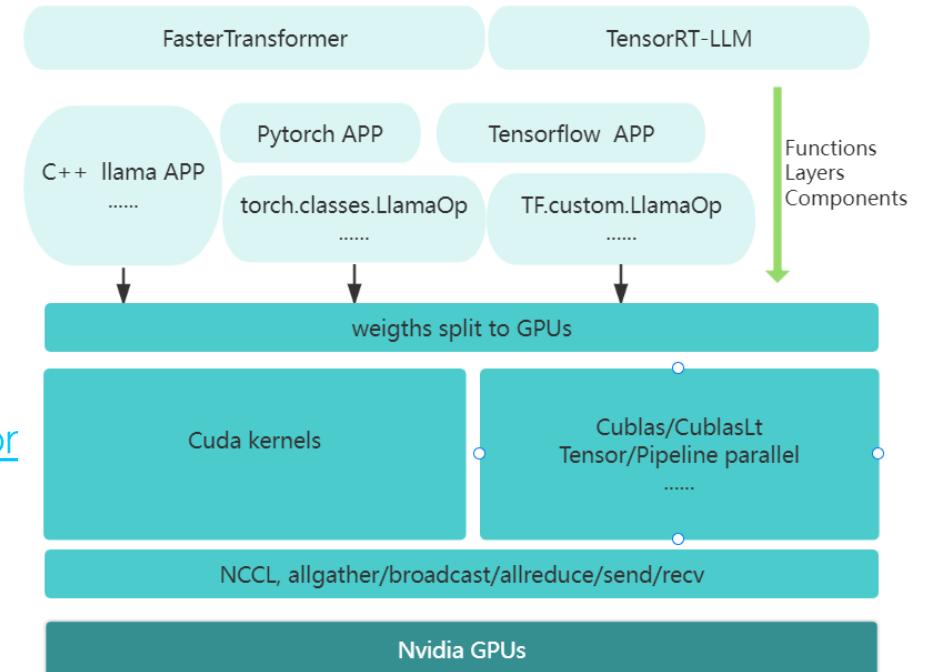
Xiaoping Duan, Chen Wang - DTCE/DSE/SATG

Feng Ding - AISE/DAIS/DCAI



FasterTransformer

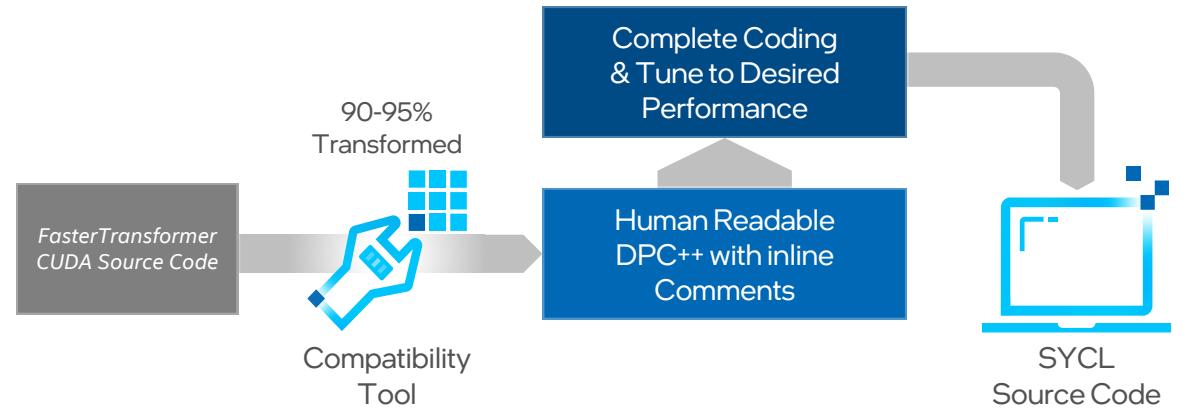
- A highly optimized transformer layer for both the encoder and decoder for NLP inference built on top of CUDA, cuBLAS, cuBLASLt and C++
 - Over 90 CUDA source files and 45k code lines
 - Over 20 models: BERT, XLNet, GPT, WeNet, Llama ...
- Multiple CSPs AI engine are based on FasterTransformer
 - Alibaba cloud - <https://github.com/alibaba/rtp-llm>
 - Tencent cloud - <https://github.com/InternLM/Imdeploy>
 - ByteDance <https://github.com/bytedance/ByteTransformer>
 - Amazon - <https://aws.amazon.com/blogs/machine-learning/deploy-large-models-at-high-performance-using-fastertransformer-on-amazon-sagemaker/>
- Migrate Llama model to SYCL
 - A popular and representative AI workload in CUDA
 - The migration BKM can be scaled out to other models in FasterTransformer



Rapid CUDA to SYCL Migration with DPCT/SYCLomatic

- Over 96% CUDA code automatically migrated by DPCT

Intel DPC++ Compatibility Tool Usage Flow



- 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

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

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

An Example of Code Editing - inline PTX code migration

CUDA : fma.rn.fl6x2

```
inline __device__ uint32_t fma(uint32_t a, uint32_t b, uint32_t c)
{
    uint32_t d;
    asm volatile("fma.rn.fl6x2 %0, %1, %2, %3; \n" : "=r"(d) : "r"(a), "r"(b), "r"(c));
    return d;
}
```



```
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);

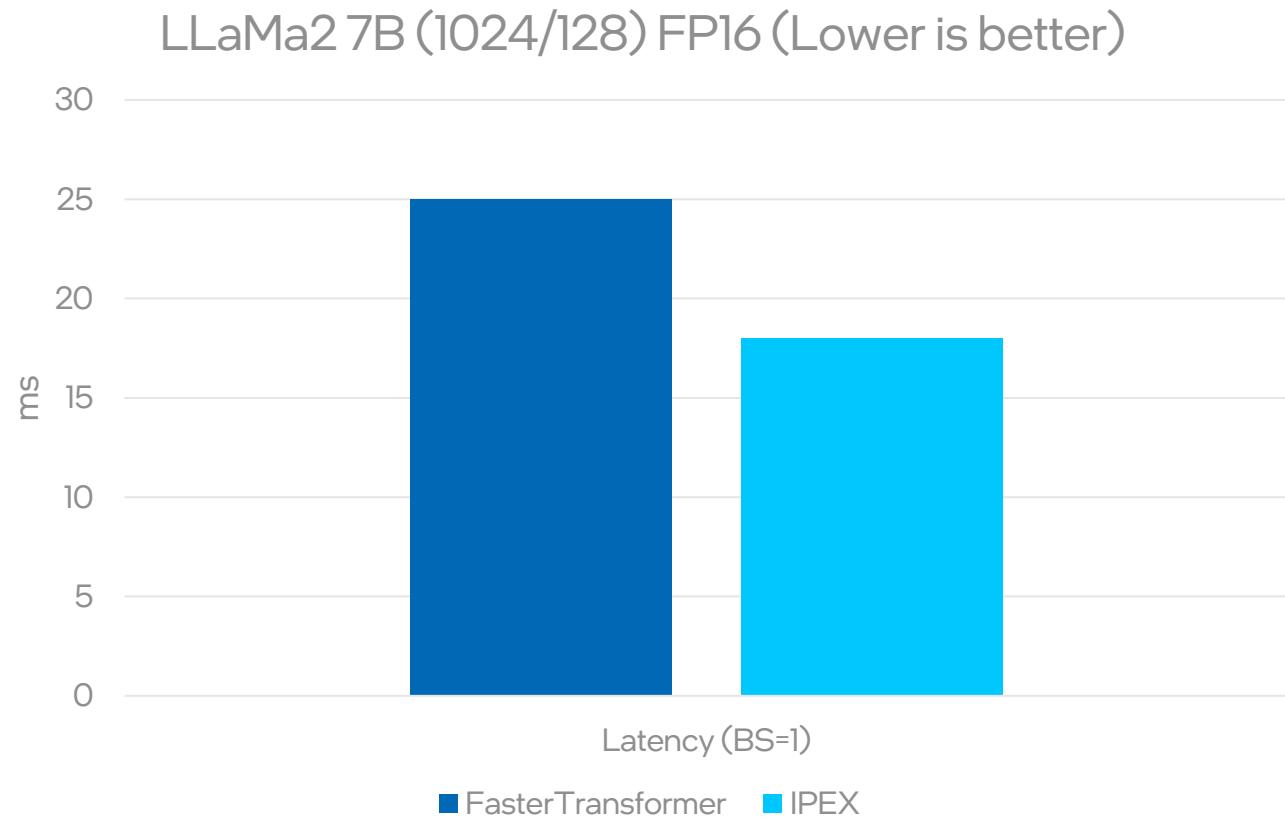
    sycl::half lowdf = lowaf * lowbf + lowcf;
    sycl::half highdf = highaf * highbf + highcf;

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

    d=(highd <<16) | lowd;

    return d;
}
```

Out of Box Performance vs. IPEx on Max 1550



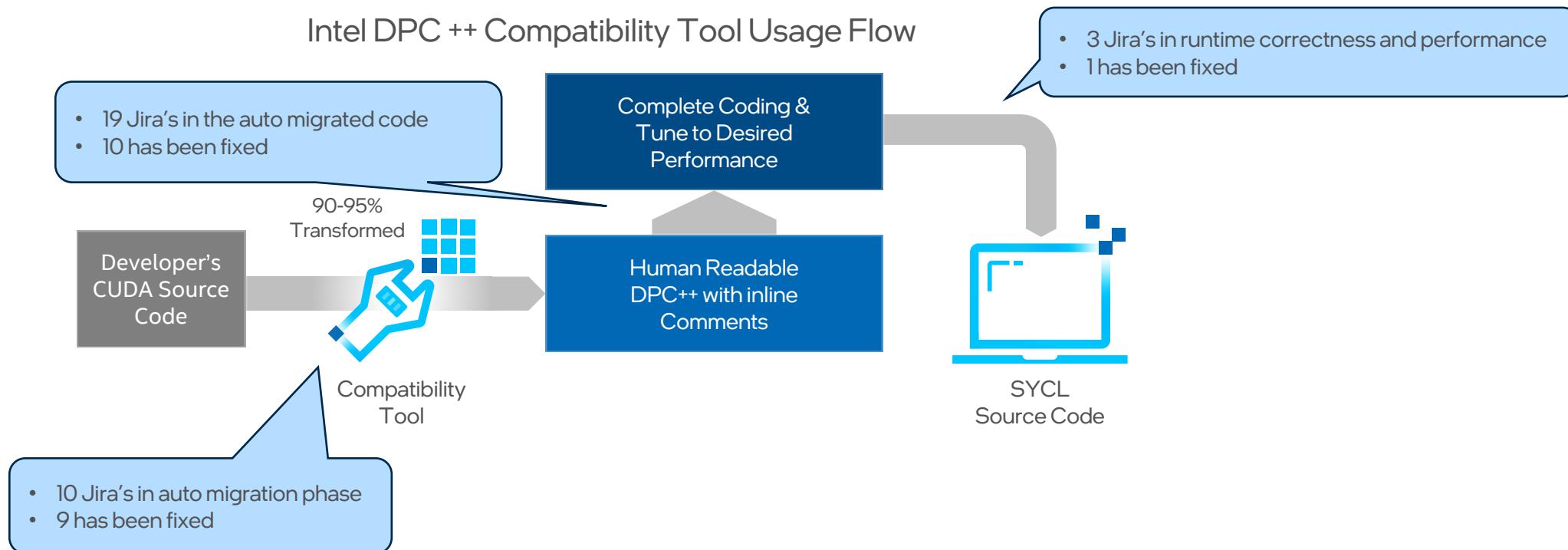
Out of box performance is 28% slower than the optimized IPEx in LLaMa2 7B inference

MORE DETAILS

How the migration work improved Intel tools in AI application domain

The Improvement of Intel oneAPI Tool by Migration

- Tools improvements were made throughout the whole migration workflow
- Total 32 Jira's opened for Intel oneAPI tool
- 20 Jira's have already been fixed by engineering team



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!
```

- In the example code the temporary memory pointed by “p_m” has to be freed before the function return. In CUDA code “cudaFree” API will do an implicit synchronization with the GPU kernel running in “invokeD2DCpyAdd” call so the memory will not be released until the kernel running is finished. DPCT will directly migrate it to only SYCL memory free API “sycl::free”. However, in current SYCL spec this memory free API will not do any synchronization even the queue “q_ctl” is a in order queue. It will make SYCL runtime possibly release the memory when the kernel running in “invokeD2DCpyAdd” is still alive. The error will happen if the kernel duration is long enough with a larger data size.
- Memory free API is a very commonly used API so it may impact lots of migration workload. The current fix provided by DPCT will introduce an unnecessary synchronization which will impact performance. We still need a fix from the underlying SYCL runtime. Underneath LO has already provided such support but SYCL runtime need to add it into the spec first and then expose this feature through a new API.

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

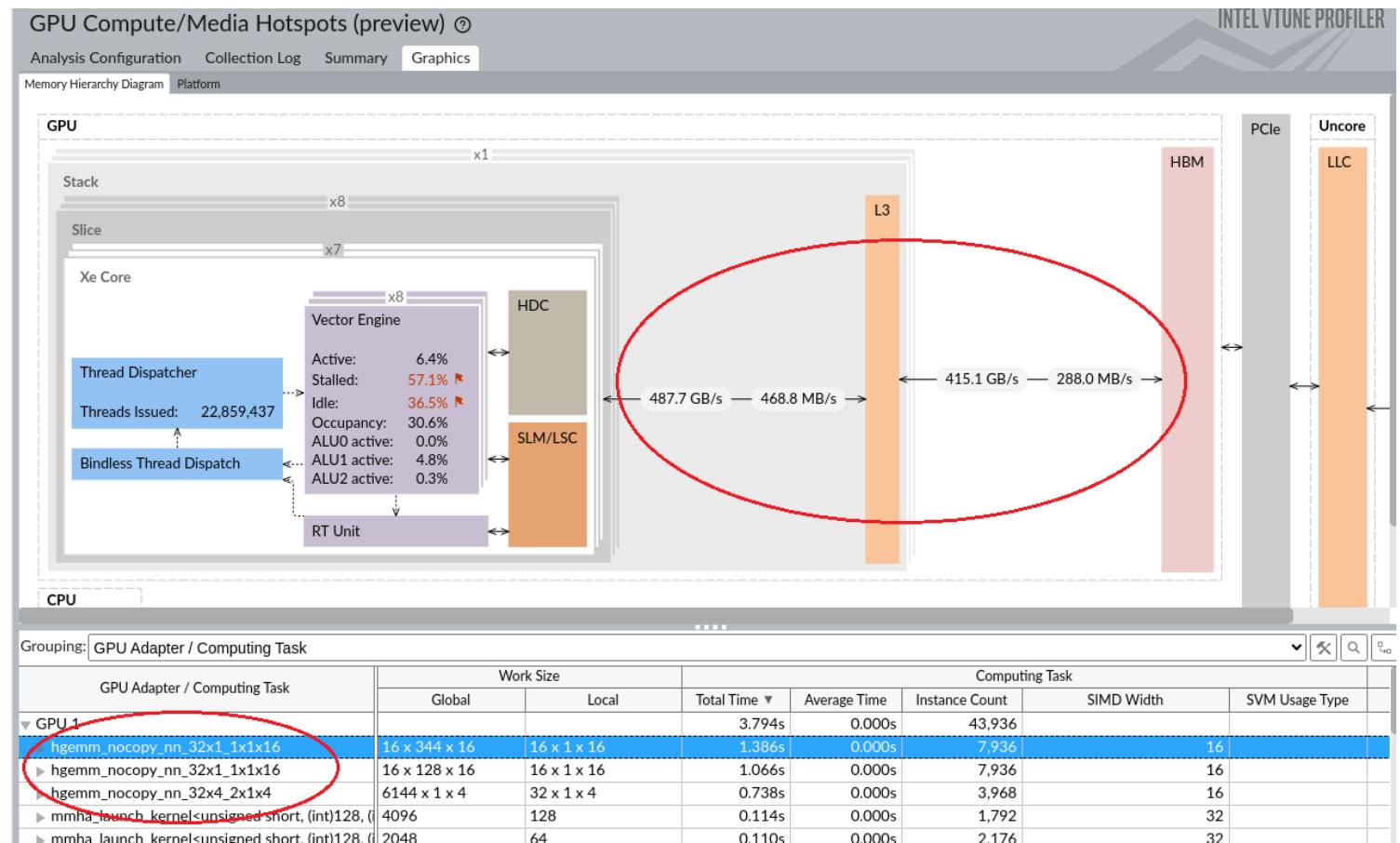
- Multiple NVCC dialects can't be correctly parsed by Clang FE of DPCT: [CTST-5517](#), [CTST-5518](#), [CTST-5519](#) and [CTST-5520](#)
- DPCT will issue parsing error on unsupported dialect without additional info
- Such error messages are misleading and easily misunderstood as migration tool's failure
- 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



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

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

The Intel logo is displayed in white against a solid blue background. The word "intel" is written in a lowercase, sans-serif font. A small, solid blue square is positioned above the letter "i". The letter "i" has a vertical stroke extending upwards from its top loop. The letter "t" has a vertical stroke extending downwards from its top loop. The letters "n", "e", and "l" are standard lowercase forms.

Back up

FasterTransformer Jira Full List

<https://jira.devtools.intel.com/issues/?jql=labels%20%3D%20FasterTransformer>