Beginner’s guide to Metal kernels

malfet
__malfet__

Preamble

- Laptop those slides are stream from is a pretty powerful machine
  - 110 Gflop of CPU perf vs 7 Tflop of GPU perf
  - Alas GPU is not CUDA capable, so it could not be used by existing ML frameworks
- MPS backend was first released in PyTorch-1.12 to expose more HW features to PyTorch users
  - It doesn’t cover all the operations that one might need
  - On the MPS(which stands for Metal Performance Shaders) interoperates nicely with Metal, so it must be fun learning some
- GPU programming languages are all a bit alike
  - So if you can write CUDA, you can do Metal as well

History of GPGPU

- OpenGL 1.0 was released in 1992, but it only had fixed pipelines

History of GPGPU

- OpenGL 1.4 GLSlang ARB/ OpenGL 2.0 (2004) introduced programmable shaders
- Around the same time people noticed that fragment shaders could also be used for scientific computations (see Ian Buck's PHD thesis)
- And so in 2007 CUDA was born
- And OpenCL (Authored by Apple) two years later
- In 2014/2015 Metal replaced OpenGL and OpenCL and became the standard for Apple devices

Prerequisites

- To write Metal shaders (either for Mac, iPhone or VisionPro) a MacOS device with developer tools installed is necessary
- One should also be familiar a little bit with ObjectiveC/Swift
    - Though Metal C++ interface technically exists
- And of course with [Metal Shading Language Spec]()

```
id<MTLCommandBuffer> cmdBuffer = [queue commandBufferWithDescriptor:desc];
id<MTLComputeCommandEncoder> encoder = [cmdBuffer computeCommandEncoder];
[encoder setComputePipelineState:cpl];
[encoder setBuffer:rc offset:0 atIndex:0];
[encoder setBytes:&triu_size length:sizeof(uint64_t) atIndex:5];
[encoder dispatchThreads:MTLSizeMake(triu_size, 1, 1)
   threadsPerThreadgroup:MTLSizeMake(32, 1, 1)];
[encoder endEncoding];
[cmdBuffer commit];
[cmdBuffer waitUntilCompleted];
```

```
template <typename scalar_t>
kernel void triu_indices(device scalar_t * tensor,
                         constant int64_t& rectangle_size,
                         constant int64_t& triu_size,
                         uint linear_index [[thread_position_in_grid]]) {
  int64_t r, c;
  if (linear_index < rectangle_size) {
    // the coordinate is within the top rectangle
    r = linear_index / col;
    c = linear_index % col;
```

Terminology

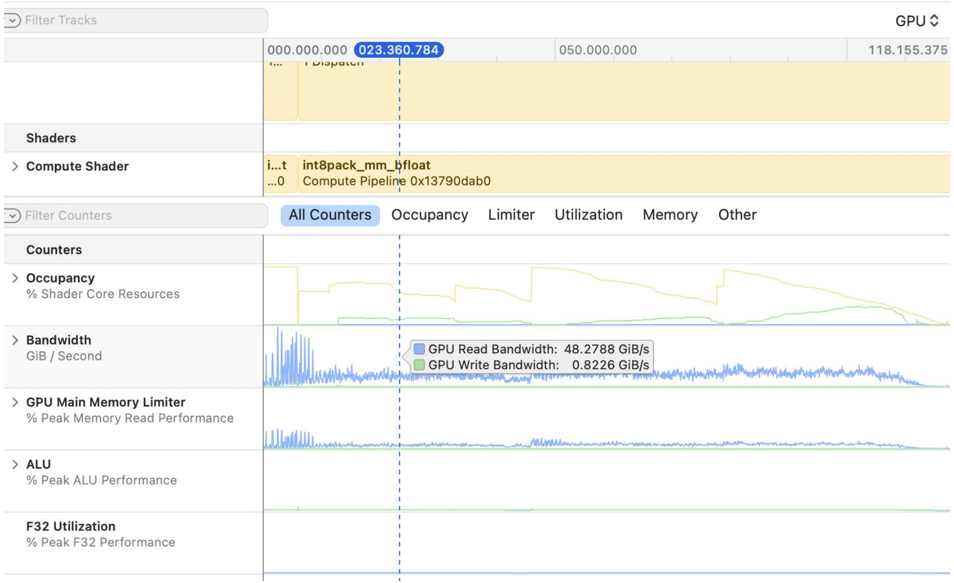
- Metal Shader - code to run on GPU
- MTLBuffer - GPU memory, can be shared with CPU
- Command Buffer - more or less command stream
- Metal Performance Shaders (hence name MPS) - collection of optimized shaders for popular operations (aka CUBLAS/cuDNN), for example see [LU factorization on MPS]
- Metal Performance Shaders Graph (MPSGraph) is a graph-like compiler that fuses multiple shaders together, default interface to access GPU

How to call Metal kernels from PyTorch

- Use `MetalShaderLibrary` class to JIT-compile your kernel
- Kernel invocation from ATEN looks as follows:

```
dispatch_sync_with_rethrow(mpsStream->queue(), ^() {
 @autoreleasepool {
  id<MTLComputeCommandEncoder> computeEncoder = mpsStream->commandEncoder();
  auto crossPSO = lib.getPipelineStateForFunc("cross_" + scalarToMetalTypeString(out));
  [computeEncoder setComputePipelineState:crossPSO];
  mtl_setBuffer(computeEncoder, input, 0);
  mtl_setBuffer(computeEncoder, other, 1);
  mtl_setBuffer(computeEncoder, out, 2);
  [computeEncoder setBuffer:kernelDataOffsets offset:0 atIndex:3];
  [computeEncoder setBytes:&out_dim_stride length:sizeof(int64_t) atIndex:4];
  [computeEncoder setBytes:&input_dim_stride length:sizeof(int64_t) atIndex:5];
  [computeEncoder setBytes:&other_dim_stride length:sizeof(int64_t) atIndex:6];
  mtl_dispatch1DJob(computeEncoder, crossPSO, numThreads);
 }
});
```

How to profile/debug Metal Kernels



- On Mac there are no better tool that Xcode :)
- Use MTL_CAPTURE_ENABLED environment variable to enable capture
- Then open .gputrace

How to profile Metal from command line

- Short answer: you can not
- But you can measure CPU wall clock time to estimate your kernel perf (but make sure to sync your pipeline at the end)
- See [benchmark_unary]() for example

```
def bench_unary(m, n, unary_func, dtype=torch.float32,device="cpu"):
    if device == "mps":
        sync_cmd = "torch.mps.synchronize()"
    elif device == "cuda":
        sync_cmd = "torch.cuda.synchronize()"
    else:
        sync_cmd = ""
    t = Timer(
        stmt=f"f(x);{sync_cmd}",
        setup=f"x=torch.rand(({m}, {n}), dtype={dtype}, device='{device}')",
        globals = {'f': unary_func},
        language="python", timer=default_timer
    )
    return t.blocked_autorange()
```

How to debug Metal from command line

From Apple's Validating your app Metal API usage / man MetalValidation:

```
[(base) malfet@Nikitas-MacBook-Pro benchmarks %
[(base) malfet@Nikitas-MacBook-Pro benchmarks % git diff mps_sum_sincos.mm
diff --git a/benchmarks/mps_sum_sincos.mm b/benchmarks/mps_sum_sincos.mm
index 252f107..e646dad 100644
--- a/benchmarks/mps_sum_sincos.mm
+++ b/benchmarks/mps_sum_sincos.mm
@@ -9,7 +9,7 @@ kernel void sum_sincos(constant T* x,
                        device   T* out,
                        uint index [[thread_position_in_grid]])
 {
-    out[index] = static_cast<T>(sin(x[index]) + cos(x[index]));
+    out[index + 1] = static_cast<T>(sin(x[index]) + cos(x[index]));
 }

 template [[host_name("sum_sincos_float")]] kernel void sum_sincos(constant float*, device float*, uint);
[(base) malfet@Nikitas-MacBook-Pro benchmarks % MTL_SHADER_VALIDATION_REPORT_TO_STDERR=1 MTL_SHADER_VALIDATION=1 \
[>   python benchmark_torch_trig.py
2024-10-12 10:11:39.065 python[39102:1314087] Metal GPU Validation Enabled
Invalid device store at offset 92432171008, executing kernel function: "sum_sincos_float"
buffer: <unnamed>, length:536870912, resident:Read Write
pipeline: "(null)", UID: "B40D46B1D6AF128BEE8071FBAB2E3F50822CFAD15AB54497A963B2A393A452D7" encoder: "0", dispatch: 0
        * frame #0: sum_sincos<float>() - /program_source:9:20
```

Case study 1: Accelerating GemV

(Intermission) Is AI ready to replace us? (by seeking Llama2 help)

> Write fast matrix vector multiplication metal kernel

GPU acceleration of matrix vector multiplication using OpenCL

This is an example of a GPU-accelerated matrix vector multiplication kernel written in OpenCL. It demonstrates how to implement a simple matrix vector multiplication operation using OpenCL.

```
// Matrix vector multiplication kernel
kernel void MatrixVectorMul(
 global float *A, // Input matrix A
 global float *x, // Input vector x
 global float *y, // Output vector y
 int M, int N, int K // Matrix dimensions
) {
 int idx = get_global_id(0); // Get global ID
 // Calculate the result
 float sum = 0.0f;
 for (int i = 0; i < K; i++) {
  sum += A[idx * K + i] * x[i];
 }
 y[idx] = sum;
}
```

First step towards better perf: classic SIMD

```
constant float *A_ptr = A + m * lda;
constant float *B_ptr = B + n * lda;

float rc = 0.0;
for (uint k = 0; k < sizes.y; k++) {
  const auto a_val = A_ptr[k];
  const auto b_val = B_ptr[k];
  rc += a_val * b_val;
}
outputData[m * ldc + n] = rc;
```

```
constant float4 *A_ptr = reinterpret_cast<constant float4 *>(A + m * lda);
constant float4 *B_ptr = reinterpret_cast<constant float4 *>(B + n * lda);

float rc = 0.0;
for (uint k = 0; k < sizes.y / 4; k++) {
  const auto a_val = A_ptr[k];
  const auto b_val = B_ptr[k];
  rc += dot(a_val, b_val);
}
outputData[m * ldc + n] = rc;
```

2nd step: using mat4 SIMD

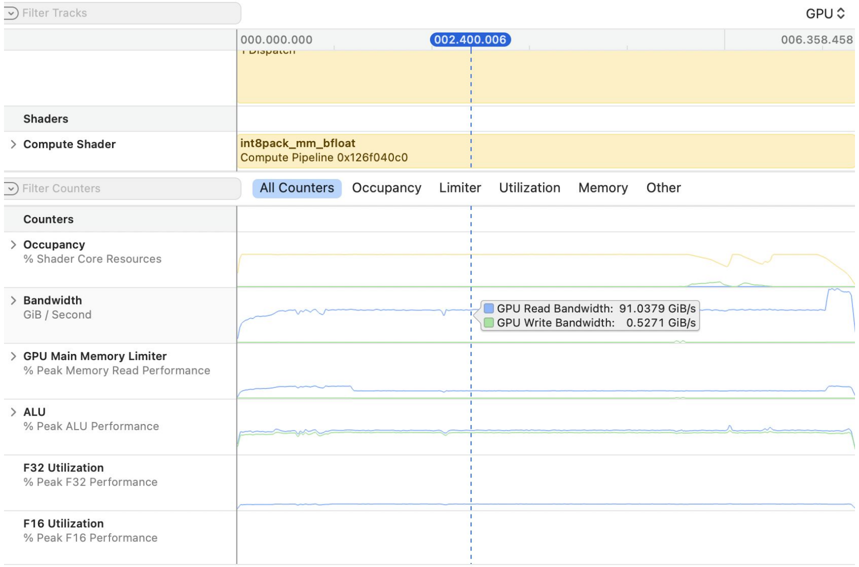
```
float rc = 0.0;
for (uint k = 0; k < sizes.y / 4; k++) {
  const auto a_val = A_ptr[k];
  const auto b_val = B_ptr[k];
  rc += dot(a_val, b_val);
}
outputData[m * ldc + n] = rc;
```

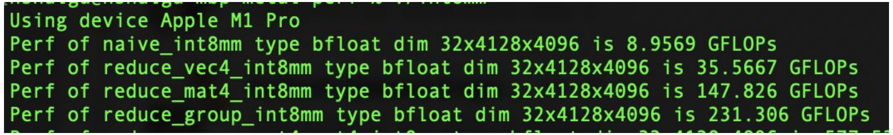
```
float4 rc = 0.0;
for (uint k = 0; k < sizes.y / 4; k++) {
 float4x4 b_mat;
 for(int j = 0; j < 4; ++j) {
   b_mat[j] = B_ptr[k + j * lda /4];
 }
 const auto a_vec = A_ptr[k];
 rc += transpose(b_mat) * a_vec;
}
```

3rd step: optimize memory access pattern

- GPU threads are organized in threadgroups
- Threads further organized into simd groups (something like CUDA warps)
- Scheduling threads from single threadgroups to access memory concurrently can significantly improve perf
- See MLX GEMVT kernel’s nice comment explaining the idea in a bit more details

Progress so far...





Links

- https://pytorch.org/get-started/locally/
- https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/mps/operations/CrossKernel.mm
- https://github.com/ml-explore/mlx/blob/main/mlx/backend/metal/kernels/gemv.metal
- https://github.com/malfet/llm_experiments/blob/main/metal-perf/gemm_perf_studies.mm
- Metal Shading Language Specification - Apple Developer
- MPS missing ops issue
- Metal Puzzles

Interactive part: let's try to implement i0

Defined torch.special — PyTorch 2.4 documentation as

$$\text{out}_i = I_0(\text{input}_i) = \sum_{k=0}^{\infty} \frac{(\text{input}_i^2/4)^k}{(k!)^2}$$

Implemented in https://github.com/pytorch/pytorch/pull/137849