Rust GPU Compute

Mike Seddon

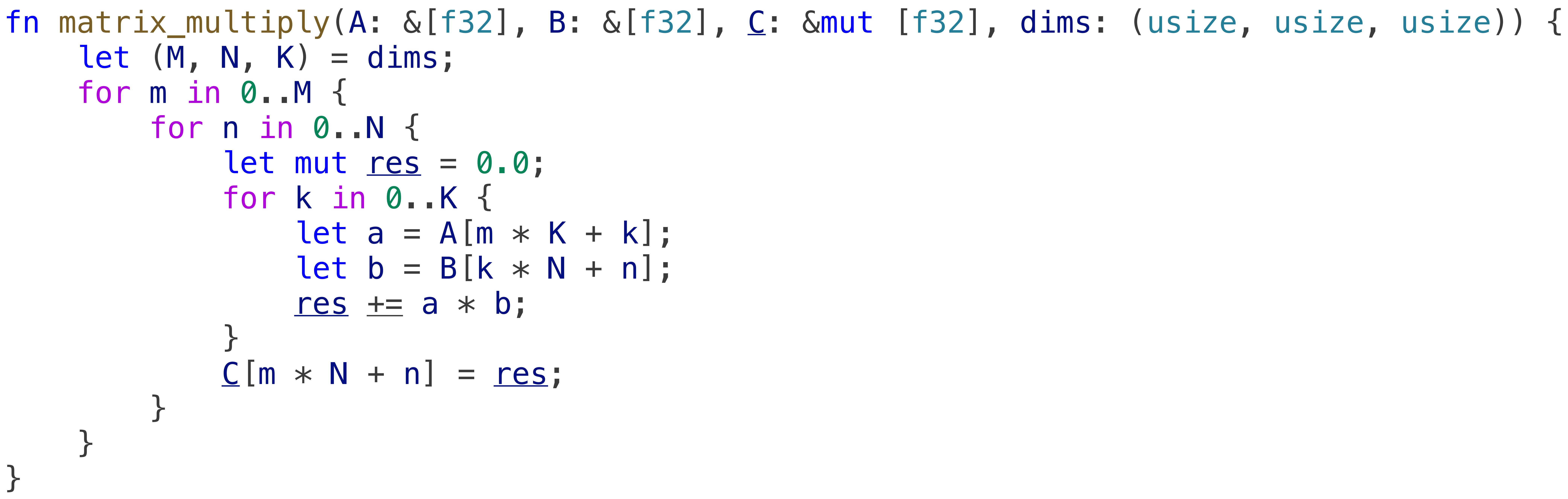
About Me

- **Rust** exclusively for the last 3 years
    - Machine learning workloads in production.
    - Started with Apache Datafusion.

- **Scala** mainly due to Apache Spark.

- **Go** as an antidote to type-theorists.

Single Precision General Matrix Multiply



```
fn matrix_multiply(A: &[f32], B: &[f32], C: &mut [f32], dims: (usize, usize, usize)) {
    let (M, N, K) = dims;
    for m in 0..M {
        for n in 0..N {
            let mut res = 0.0;
            for k in 0..K {
                let a = A[m * K + k];
                let b = B[k * N + n];
                res += a * b;
            }
            C[m * N + n] = res;
        }
    }
}
```

- At 1024 x 1024 there are 1,048,576 elements.
- 2 x 1024 x 1024 x 1024 = 2,147,483,648 FLOP
- Execution 3.17s - Performance = **0.7GFLOPS**

Naive vs Faer[1]

```
faer_core::mul::matmul(
    C.as_mut(),
    A.as_ref(),
    B.as_ref(),
    None,
    0.0,
    faer_core::Parallelism::None,
);
```

- Single thread + Intrinsics = **27.0GFLOPS** (38x)
- 12 thread + Intrinsics = **154.8GFLOPS** (5.7x)

[1] https://faer-rs.github.io

What can I do with a GFLOP?

- LLAMA2 - Facebook's GPT Competitor is released with 7B, 13B and 70B parameters.

- Rule of thumb is ~2 x parameters = GFLOPS[1].

- 70B parameters ~= 140GFLOPS per token.

- At 20 tokens/sec = 2800GFLOPS = **2.8TFLOPS**.

1. https://cursor.sh/blog/llama-inference

The contenders

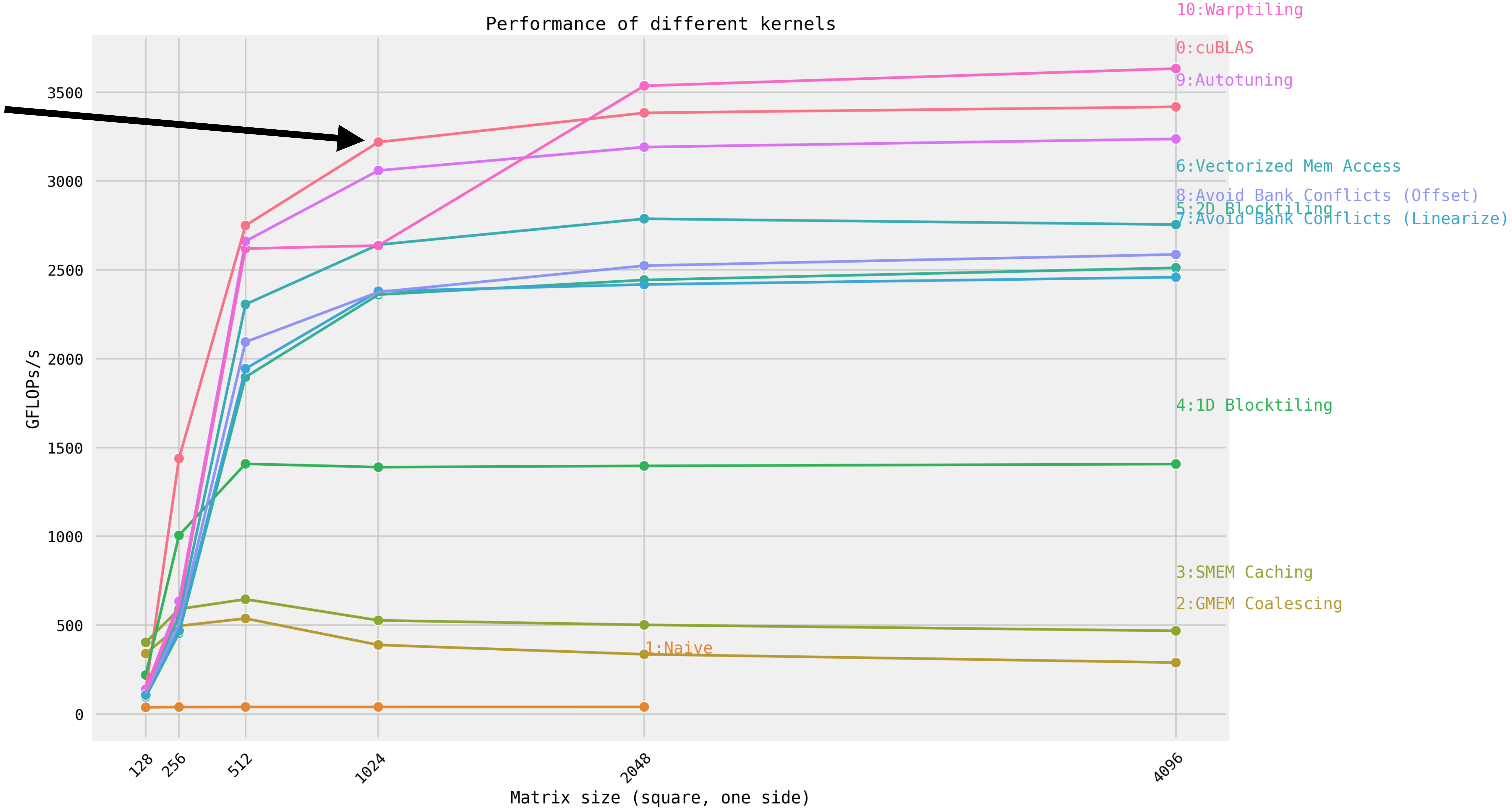
NVIDIA
CUDA

cuBLAS

3.2TFLOPS

vs GPU cap. (4TFLOPS) = 80%

vs 0.15 CPU = 21x



1. https://siboehm.com/articles/22/CUDA-MMM

cudarc[1] - minimal and safe api over the cuda toolkit

```
// Get the device
let gpu = CudaDevice::new(0)?;

// Load the kernel
gpu.load_ptx(ptx, "sgemm", &["sgemm_naive"])?;
let sgemm = gpu.get_func("sgemm", “sgemm_naive")?;

// Run the kernel
unsafe {
    sgemm
        .launch(cfg, (m, n, k, alpha, &a_dev, &b_dev, beta, &c_dev))
}?;

// Sync data to make sure it has finished
gpu.synchronize()?;
```

[1] https://github.com/coreylowman/cudarc

cudarc - example kernel

```
extern "C" __global__ void sgemm_naive(int M, int N, int K, const float *A, const float *B, float *C) {
  const unsigned int x = blockIdx.x * blockDim.x + threadIdx.x;
  const unsigned int y = blockIdx.y * blockDim.y + threadIdx.y;

  float tmp = 0.0;
  for (int i = 0; i < K; ++i) {
    tmp += A[x * K + i] * B[i * N + y];
  }
  C[x * N + y] = tmp;
}
```

1. https://github.com/coreylowman/cudarc

wgpu[1] - cross-platform, safe, pure-rust graphics api

- Implementation of the WebGPU standard and used as the WebGPU implementation for Firefox.

- Abstracts away Vulcan, Metal, DX12, DX11, OpenGL.

- Uses the WGSL shader language.

[1] https://wgpu.rs

wgpu[1] - WGSL example

```
//Naive matrix multiplication
//https://github.com/siboehm/SGEMM_CUDA/blob/master/src/kernels/1_naive.cuh
@group(0) @binding(0)
var<storage, read> A: array<f32>;

@group(0) @binding(1)
var<storage, read> B: array<f32>;

@group(0) @binding(2)
var<storage, read_write> C: array<f32>;

@compute @workgroup_size({{ workgroup_size_x }}, {{ workgroup_size_y }}, {{ workgroup_size_z }})
fn main(
  @builtin(global_invocation_id) global_id: vec3<u32>
) {
    let M = {{ M }}u;
    let N = {{ N }}u;
    let K = {{ K }}u;
    let x = global_id.x;
    let y = global_id.y;
    if (x < M && y < N) {
        var tmp = 0f;
        for (var i = 0u; i < K; i = i + 1u) {
          tmp += A[x * K + i] * B[i * N + y];
        }
        C[x * N + y] = tmp;
    }
}
```

[1] https://github.com/FL33TW00D/wgpu-mm

wgpu[1] - cross-platform, safe, pure-rust graphics api

```
// Get the device
let backends = wgpu::util::backend_bits_from_env().unwrap_or(wgpu::Backends::PRIMARY);
let instance = wgpu::Instance::new(InstanceDescriptor {
    backends,
    ..Default::default()
});

let adapter = wgpu::util::initialize_adapter_from_env_or_default(&instance, None)
    .await
    .expect("No GPU found given preference");
let (device, queue) = adapter
    .request_device(
        &wgpu::DeviceDescriptor {
            label: Some("test"),
            features: wgpu::Features::default() | wgpu::Features::TIMESTAMP_QUERY,
            limits: Limits::default(),
        },
        None,
    )
    .await
    .expect("Could not create adapter for GPU device”);
```

[1] https://github.com/FL33TW00D/wgpu-mm

wgpu[1] - cross-platform, safe, pure-rust graphics api

```
// Prepare the kernel code
let shader_module = unsafe {
    handle
        .device()
        .create_shader_module_unchecked(wgpu::ShaderModuleDescriptor {
            label: None,
            source: wgpu::ShaderSource::Wgsl(Cow::Borrowed(&shader)),
        })
};

// Create a pipeline for the shader
let pipeline = handle
    .device()
    .create_compute_pipeline(&wgpu::ComputePipelineDescriptor {
        label: None,
        layout: None,
        module: &shader_module,
        entry_point: "main",
    });
```

[1] https://github.com/FL33TW00D/wgpu-mm

wgpu[1] - cross-platform, safe, pure-rust graphics api

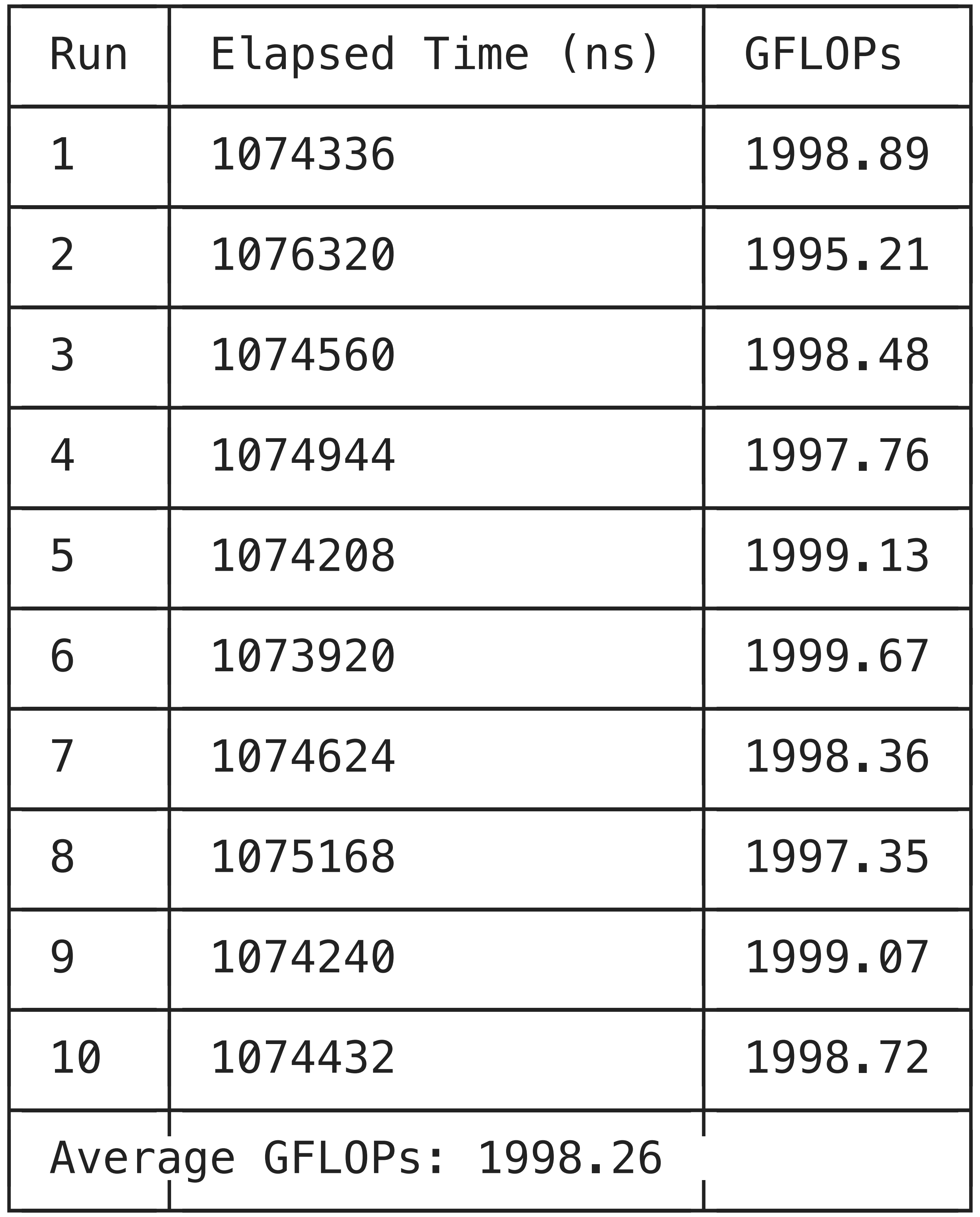
```
// Create a command encoder
let mut encoder = handle
    .device()
    .create_command_encoder(&wgpu::CommandEncoderDescriptor { label: None });

// Define the computation
let mut cpass = encoder.begin_compute_pass(&wgpu::ComputePassDescriptor {
    label: None,
    timestamp_writes: None,
});
cpass.set_bind_group(0, bind_group, &[]);
cpass.set_pipeline(pipeline);
cpass.dispatch_workgroups(workgroup_count.0, workgroup_count.1, workgroup_count.2);

// Submit the workload
handle.queue().submit(Some(encoder.finish()));
```

[1] https://github.com/FL33TW00D/wgpu-mm

wgpu[1] - cross-platform, safe, pure-rust graphics api



Run	Elapsed Time (ns)	GFLOPs
1	1074336	1998.89
2	1076320	1995.21
3	1074560	1998.48
4	1074944	1997.76
5	1074208	1999.13
6	1073920	1999.67
7	1074624	1998.36
8	1075168	1997.35
9	1074240	1999.07
10	1074432	1998.72
Average GFLOPs: 1998.26		

Vulkan = 2.0 TFLOPS
cuBLAS = 3.2TFLOPS

~ 0.62x 😞

[1] https://github.com/FL33TW00D/wgpu-mm

Is there hope?

```
simdgroup_float8x8 A[4];
simdgroup_float8x8 B[4];
for (uint k = 0; k < {N}; k+=8) {{
  threadgroup_barrier(mem_flags::mem_threadgroup);
  simdgroup_load(A[0], data1+k+{0*N}, {N}, ulong2(0, 0));
  simdgroup_load(A[1], data1+k+{8*N}, {N}, ulong2(0, 0));
  ..
  simdgroup_load(B[2], data2+16+k*{N}, {N}, ulong2(0, 0));
  simdgroup_load(B[3], data2+24+k*{N}, {N}, ulong2(0, 0));

  simdgroup_multiply_accumulate(acc[0][0], A[0], B[0], acc[0][0]);
  simdgroup_multiply_accumulate(acc[0][1], A[1], B[0], acc[0][1]);
  simdgroup_multiply_accumulate(acc[0][2], A[2], B[0], acc[0][2]);
  ...
  simdgroup_multiply_accumulate(acc[3][1], A[1], B[3], acc[3][1]);
  simdgroup_multiply_accumulate(acc[3][2], A[2], B[3], acc[3][2]);
  simdgroup_multiply_accumulate(acc[3][3], A[3], B[3], acc[3][3]);
}}
simdgroup_store(acc[0][0], a+{0+0*N}, {N}, ulong2(0, 0));
simdgroup_store(acc[1][0], a+{8+0*N}, {N}, ulong2(0, 0));
simdgroup_store(acc[2][0], a+{16+0*N}, {N}, ulong2(0, 0));
...
simdgroup_store(acc[1][3], a+{8+24*N}, {N}, ulong2(0, 0));
simdgroup_store(acc[2][3], a+{16+24*N}, {N}, ulong2(0, 0));
simdgroup_store(acc[3][3], a+{24+24*N}, {N}, ulong2(0, 0));
```

Custom metal shader which uses **simdgroup** instructions hits ~75% theoretical performance on Apple M1.

simdgroup is not available in Vulcan yet.

1. https://github.com/geohot/tinygrad

Links

Blog:

https://reorchestrate.com

Others:

https://faer-rs.github.io

https://siboehm.com/articles/22/CUDA-MMM

https://github.com/coreylowman/cudarc

https://github.com/FL33TW00D/wgpu-mm