Register Tiling Matrix Multiplication

by: TalkativeTPUs
Members: Abhinav Garg, Yuanfeng Niu,
Minsoo Kim, Tanmay Patel

Recap on Shared memory tiling matrix multiplication

Essence: Load operands first into shared memory and then reuse multiple times to reduce the need for global memory accesses

- Each thread loops through TILE_WIDTH elements from matrix A and TILE_WIDTH elements from matrix B to calculate the partial sum for **one input value**
- Shared memory has a significantly lower read latency
- Reduces latency between each multiply-add operation

Shortcomings:

- Require 2 accesses to shared memory per multiply-add operation
- Two-step loading into global, shared and then register memory is wasteful
- Need to call __syncthreads twice for each **TILE_WIDTH** multiply-adds
- There’s no pattern to reduce memory re-loading

Making better use of GPU resources

Nvidia A40 GPU has a 256KB register file

All data must be loaded into registers before being operated upon, so no “additional cost”

Registers are private to threads, so it requires a change in approach to allow for reuse

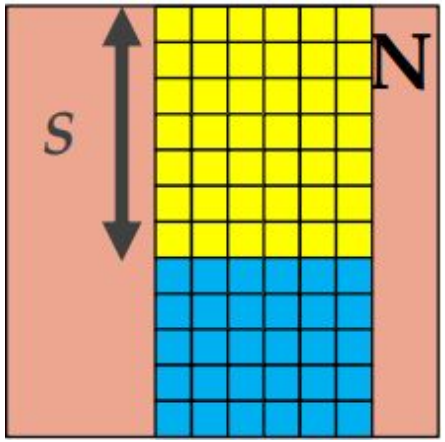
Register Tiling Matrix Multiplication AxB = C

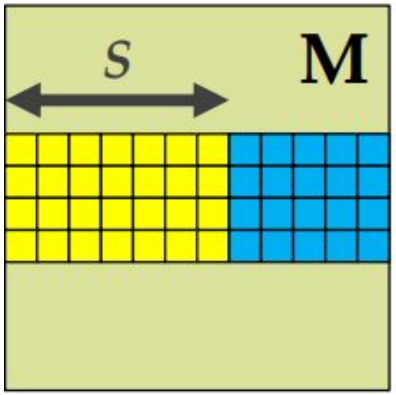
Instead:

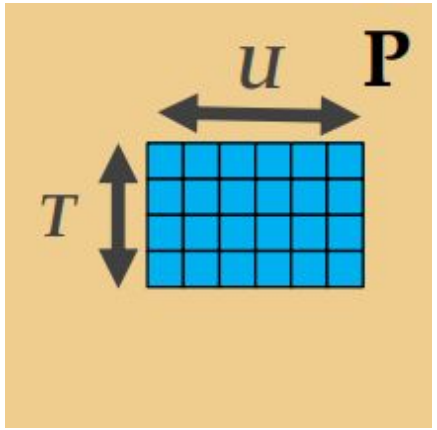
Each thread loops to calculate partial sums for **U consecutive output values**

- A **row of width S** within the tile of matrix A is loaded into **register memory**
- A tile of data from matrix B is loaded into shared memory
- The thread accumulates the partial sum for **U** output values in **register memory**

From: https://lumetta.web.engr.illinois.edu/508/slides/lecture4.pdf







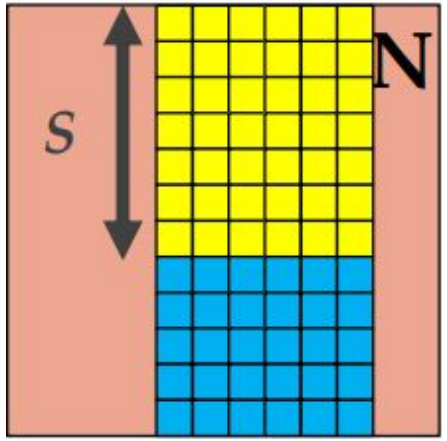
Reuse

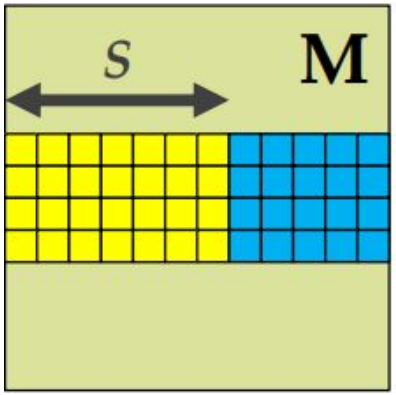
Once **S** values of matrix A are loaded into the registers, the partial dot product is calculated with **S** elements from matrix B

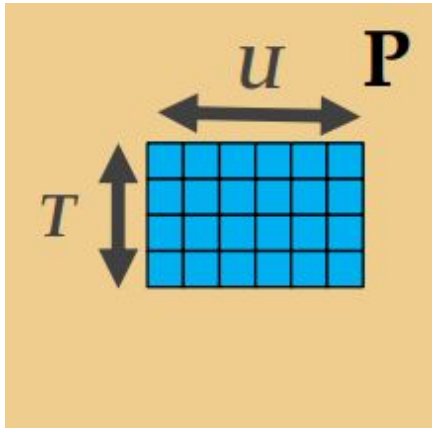
U consecutive columns within the tile of matrix N are operated upon

The partial sum for each of the **U** output values is stored in a register

From: https://lumetta.web.engr.illinois.edu/508/slides/lecture4.pdf







Memory Access

Input matrix: Each thread in a warp accesses consecutive memory (coalesced)

Weight matrix: Shared memory eliminates redundant global memory access

Output matrix: Coalesced write pattern

Implementation Overview: Pseudocode

```
REGISTER_TILING_MATMUL(out, inp, weight, bias, B, T, C, OC):
    b = blockIdx.z
    t = blockIdx.y
 output_start = blockIdx.x * TILE_SIZE

    out_reg[TILE_SIZE] = {0.0f}

    IF bias != NULL:
        Load bias values into out_reg

    FOR each tile of C:
        Load weight[output_idx * C + tile_offset] into shared memory
        SYNC_THREADS()

        FOR each element in tile:
            inp_val = Load input value
            FOR each output element:
                out_reg[e] += inp_val * tile_weight[i][e]
        SYNC_THREADS()

    Write out_reg values to out[b][t][output_idx]
```

Thread Organization

- Grid dimensions: (ceil(OC/TILESIZE), T, B)
- Block dimensions: (32, 1, 1)
- Each thread processes one output element

Shared Memory Tiling

- **shared** float tile_weight[S][TILE_SIZE]
- Collaborative loading of weight matrix tiles per phase
- Reused by all threads in a block

Register Usage

- float out_reg[TILE_SIZE] accumulates in registers
- Each thread responsible for TILE_SIZE consecutive outputs
- Registers cache partial results throughout computation

Improvement

Shared Memory Matrix Multiplication

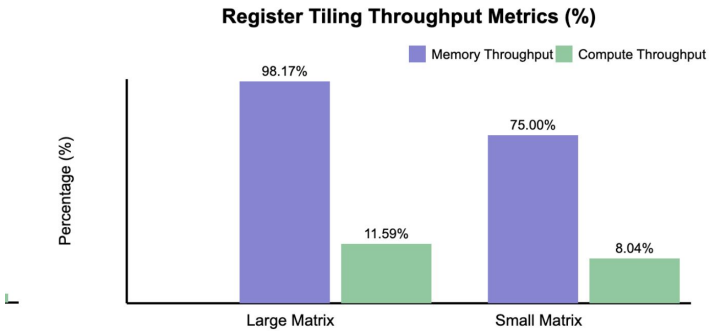
- Load both operands into shared memory and then into registers
- Does not reuse memory loaded into registers

Register Tiling Matrix Multiplication

- Load one operand directly into registers
- Reuses memory loaded into registers **U** times

Profiling Results

Metric\Type	Baseline	Reg Tiling
MatMul Total Time (%)	99.6	99.0
Memory Throughput (%)	17.68	96.28
Compute Throughput(%)	20.33	11.37



Total matmul time

- ~70% reduction in the largest instance
  - Also a ~50% improvement in compute throughput in this instance
- ~50% reduction in overall time

Memory throughput

- HUGE improvement over baseline
- Due to maximal coalescing

Compute throughput

- Worse than baseline

Overall performance better than baseline