LAB5 实验报告

何跃强 PB22111649

2025-05-27

一、实验要求

实现要求

分块矩阵实现了global memory和shared memory，我们需要进一步实现GPU的寄存器矩阵乘法。

问题回答

分析三种GPU矩阵乘法中对GPU global memory、shared memory、register的访问次数（忽略非矩阵元素的寄存器访问，如坐标索引变量row对寄存器的访问）寄存器矩阵乘法减少了线程数目及shared memory使用量，但增加了寄存器使用量。

请结合课程所学分析这样做有何好处，有何坏处。（提示：从warp个数及block 所需资源上分析）

请结合课程所学从线程访存合并方面分析基础矩阵乘法哪里可以继续优化。这一优化手段对分块矩阵乘法、寄存器矩阵乘法有效吗？请说明理由并给出优化位置。

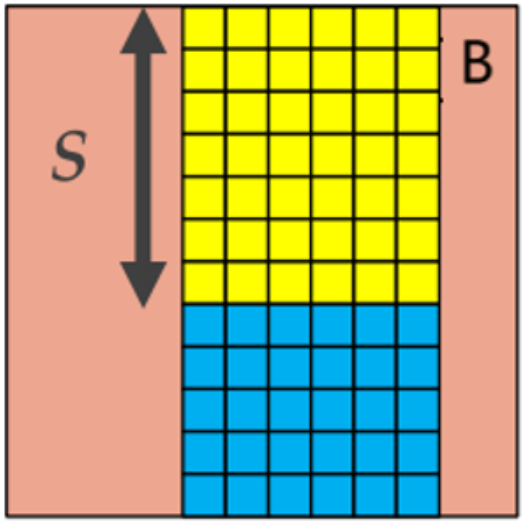
二、实验代码讲解

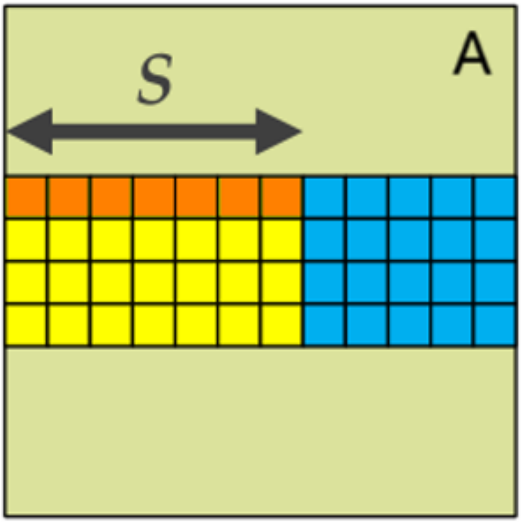
```
__global__ void Matmul3(float *A, float *B, float *C, unsigned N) {
  unsigned tid = threadIdx.x;
  unsigned c_row = blockIdx.y * T + tid;
  unsigned c_col = blockIdx.x * U;

  float Areg[S];
  float Creg[U] = {0};

  __shared__ float Bsub[S][U];

  for (unsigned kk = 0; kk < N; kk += S) {
// 把A的S个元素加载到寄存器中
#pragma unroll
    for (unsigned i = 0; i < S; i++) {
      Areg[i] = A[c_row * N + (kk + i)];
    }

    // 把B的S行U列加载到shared memory中
    unsigned b_row = tid / U; // B的行索引
    unsigned b_col = tid % U; // B的列索引
    Bsub[b_row][b_col] = B[(kk + b_row) * N + (c_col + b_col)];

    __syncthreads();

// 计算C的U个元素
#pragma unroll
    for (unsigned i = 0; i < S; i++) {
      float a_val = Areg[i];
#pragma unroll
      for (unsigned j = 0; j < U; j++) {
        Creg[j] += a_val * Bsub[i][j];
```

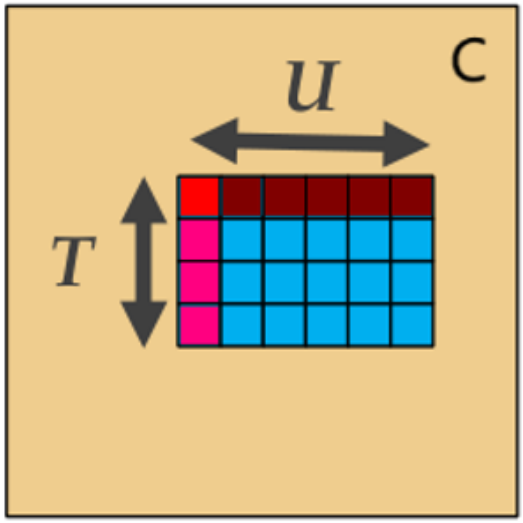
```
      }
    }

    __syncthreads();
  }

#pragma unroll
  for (unsigned j = 0; j < U; j++) {
    C[c_row * N + (c_col + j)] = Creg[j];
  }
}
```

代码思路：

按照这张图的坐标进行计算







先把A的对应元素取入寄存器，再把B的对应矩阵存入shared memory，最后再计算C的对应元素。

三、实验结果

```
(base) hyq@hyq:~/Arch$ nvcc -O3 lab5.cu -o lab5
(base) hyq@hyq:~/Arch$ ./lab5
equal!
equal!
equal!
pass!
N:  4096  time1: 215.049896240  time2: 127.721946716 time3: 96.212265015
```

```
(base) hyq@hyq:~/Arch$ nsys profile --stats=true ./lab5
Collecting data...
equal!
equal!
equal!
pass!
N:  4096  time1: 204.851760864  time2: 129.028762817 time3: 97.004432678
Generating '/tmp/nsys-report-9a1c.qdstrm'
[1/8] [========================100%] report2.nsys-rep
[2/8] [========================100%] report2.sqlite
[3/8] Executing 'nvtx_sum' stats report
SKIPPED: /home/hyq/Arch/report2.sqlite does not contain NV Tools Extension (NVTX)
data.
[4/8] Executing 'osrt_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)    Min (ns)  Max
(ns)   StdDev (ns)       Name
 --------  ---------------  ---------  ----------  ----------  --------  -------
--  -----------  --------------
     98.7      10505438330        109  96380168.2  101398167.0    496686
 101561480   16586257.5  poll
      1.1        119494530        732    163243.9       7195.0      1313
10866554     649278.6  ioctl
      0.1         14562224          7   2080317.7     293882.0      4521
12929956    4786982.9  fread
      0.0          2944074          3    981358.0    1113058.0    613587
 1217429     322746.0  pthread_create
      0.0          1371416         18     76189.8       6434.5      1210
672666     193831.2  fopen
      0.0          1111773          3    370591.0     152423.0    113343
846007     412185.7  sem_timedwait
      0.0           401090          1    401090.0     401090.0    401090
401090          0.0  pthread_join
      0.0           335064         16     20941.5       2007.0      1257
230535      56968.2  fclose
      0.0           225595         22     10254.3       9394.0      2092
 42350       8074.6  mmap
      0.0           118150          1    118150.0     118150.0    118150
118150          0.0  fgets
      0.0            65819          4     16454.8      17090.5      9745
 21893       5203.7  write
      0.0            44283          4     11070.8       8034.0      4005
 24210       8982.2  open
      0.0            13413          3      4471.0       5131.0      1979
6303       2236.3  pipe2
```

```
      0.0            11679          6      1946.5       1557.0      1041
3471       1023.3  fcntl
      0.0            10843          2      5421.5       5421.5      5300
5543        171.8  fwrite
      0.0             7430          4      1857.5       1759.0      1351
2561        522.5  read
      0.0             5065          2      2532.5       2532.5      2438
2627        133.6  close
      0.0             1407          1      1407.0       1407.0      1407
1407          0.0  fflush

[5/8] Executing 'cuda_api_sum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)  Max
(ns)   StdDev (ns)           Name
 --------  ---------------  ---------  -----------  -----------  --------  ------
---  -----------  ---------------------
     94.4       4035015996         30  134500533.2  120814522.0  90393392
 199971441   43075048.0  cudaEventSynchronize
      4.4        187449437          9   20827715.2     911108.0    277097
 181043996   60082518.8  cudaMalloc
      0.9         37855391          7    5407913.0    1869651.0    716044
13842181    5367977.3  cudaMemcpy
      0.2          7517169          8     939646.1     528023.0    150613
 4194547    1357520.7  cudaFree
      0.1          3564898         33     108027.2      40268.0      5956
 2084325     355516.0  cudaLaunchKernel
      0.0          2126051         60      35434.2      37182.0      4192
 80057      29841.0  cudaEventRecord
      0.0           219430         31       7078.4       6688.0      6163
 15496       1611.0  cudaDeviceSynchronize
      0.0             6951          2       3475.5       3475.5       619
6332       4039.7  cudaEventCreate

[6/8] Executing 'cuda_gpu_kern_sum' stats report
SKIPPED: /home/hyq/Arch/report2.sqlite does not contain CUDA kernel data.
[7/8] Executing 'cuda_gpu_mem_time_sum' stats report
SKIPPED: /home/hyq/Arch/report2.sqlite does not contain GPU memory data.
[8/8] Executing 'cuda_gpu_mem_size_sum' stats report
SKIPPED: /home/hyq/Arch/report2.sqlite does not contain GPU memory data.
Generated:
        /home/hyq/Arch/report2.nsys-rep
        /home/hyq/Arch/report2.sqlite
```

四、问题解答

1.分析三种GPU矩阵乘法中对GPU global memory、shared memory、register的访问次数：

Matmul1

- **Global Memory**：
  - 每个线程读取A的元素N次，读取B的元素N次，写入C一次。
  - 总访问次数：`2*N + 1`。

- **Shared Memory**：未使用，访问次数为`0`。
- **Register**：每个线程使用1个寄存器（`sum`变量）。

Matmul2

- **Global Memory**：
  - 每个分块加载A和B各一次，总循环次数为`N/Blocksize`。
  - 总访问次数：`2*(N/Blocksize) + 1`（读两次分块，写一次结果）。
- **Shared Memory**：
  - 每个分块内每个线程读取`Asub`和`Bsub`各`Blocksize`次，总循环`N/Blocksize`次。
  - 读次数：`2*Blocksize*(N/Blocksize) = 2N`。
  - 写次数：`2*(N/Blocksize)`（每个分块写两次）。
- **Register**：每个线程使用1个寄存器（`sum`变量）。

Matmul3

- **Global Memory**：
  - 每个线程读取A的N次，读取B的`N/S`次（`S = T/U`），写入C的U次。
  - 总访问次数：`N + N/S + U`。
- **Shared Memory**：
  - 每个线程加载B的`N/S`次，每次循环读`S*U`次。
  - 读次数：`U*N`，写次数：`N/S`。
- **Register**：每个线程使用`S + U`个寄存器（`Areg`和`Creg`数组）

2.结合课程所学分析这样做有何好处，有何坏处。

优点：

1. **减少线程数量**：每个线程处理更多数据，降低线程调度开销，提高SM内warp的并行度。
2. **减少Shared Memory使用**：降低对Shared Memory的依赖。

缺点：

1. **增加寄存器使用**：可能导致寄存器资源不足。

3.请结合课程所学从线程访存合并方面分析基础矩阵乘法哪里可以继续优化。这一优化手段对分块矩阵乘法、寄存器矩阵乘法有效吗？请说明理由并给出优化位置。

将B矩阵转置存储，使相邻线程访问连续的内存地址。

分块矩阵乘法：在加载到Shared Memory时，若B的访问仍按列，优化手段仍然有效。

寄存器矩阵乘法：需调整线程映射，确保同一Warp内的线程访问连续的B元素。