计算机体系结构实验 4 - 5 实验报告

实验 4

实验流程

完善代码

得到以下代码：

```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P, int width)
{
  // Calculate the row index of the P element and M
  int row = blockIdx.y * blockDim.y + threadIdx.y;

  // Calculate the column index of the P element and N
  int col = blockIdx.x * blockDim.x + threadIdx.x;

  // Ensure the thread is within bounds
  if (row < width && col < width) {
    float pValue = 0.0;

    // Each thread computes one element of the matrix
    for(int k = 0; k < width; ++k) {
      pValue += d_M[row * width + k] * d_N[k * width + col];
    }

    // Store the computed value into the output matrix
    d_P[row * width + col] = pValue;
  }
}
```

修改 `main()` **使得使用** `MatrixMulKernel<<<grid, block>>>(d_M, d_N, d_P, m);`

```
for (int j = 0; j < nIter; j++) {
    // matrixMulCPU(reference, h_M, h_N, m, k, n);
    MatrixMulKernel<<<grid, block>>>(d_M, d_N, d_P, m);
    // MatrixMulSharedMemKernel<<<grid, block>>>(d_M, d_N, d_P, m, n);
    // cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, n, m, k, &alpha, d_N, n,
d_M, k, &beta, d_P, n);
}
```

编译执行

```bash
bash compile.sh
./MatrixMulKernel 1 1000
```

观察发现计算结果正确

```
 ./MatrixMulKernel 1 1000
Kernel Elpased Time: 0.557 ms
Performance= 3588.04 GFlop/s, Time= 0.557 msec, Size= 2000000000 Ops
Computing result using host CPU...done.
Listing first 100 Differences > 0.000010...
```

更改矩阵尺寸，对比不同参数下的计算结果

```
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-mair
 ./MatrixMulKernel 0 1000
Kernel Elpased Time: 0.555 ms
Performance= 3603.97 GFlop/s, Time= 0.555 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-mair
 ./MatrixMulKernel 0 3000
Kernel Elpased Time: 21.366 ms
Performance= 2527.34 GFlop/s, Time= 21.366 msec, Size= 54000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-mair
 ./MatrixMulKernel 0 5000
Kernel Elpased Time: 101.788 ms
Performance= 2456.09 GFlop/s, Time= 101.788 msec, Size= 250000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-mair
 ./MatrixMulKernel 0 10000
Kernel Elpased Time: 856.724 ms
Performance= 2334.47 GFlop/s, Time= 856.724 msec, Size= 2000000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-mair
 ./MatrixMulKernel 0 20000
Kernel Elpased Time: 8493.021 ms
Performance= 1883.90 GFlop/s, Time= 8493.021 msec, Size= 16000000000000 Ops
```

可以看见，随着矩阵尺寸的增大，`Performance`也下降

更改`TILE_SIZE`，对比不同参数下的计算结果

```
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./2 0 1000
Kernel Elpased Time: 4.222 ms
Performance= 473.71 GFlop/s, Time= 4.222 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./4 0 1000
Kernel Elpased Time: 1.334 ms
Performance= 1498.93 GFlop/s, Time= 1.334 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./8 0 1000
Kernel Elpased Time: 0.681 ms
Performance= 2935.57 GFlop/s, Time= 0.681 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./16 0 1000
Kernel Elpased Time: 0.555 ms
Performance= 3602.80 GFlop/s, Time= 0.555 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./32 0 1000
Kernel Elpased Time: 0.591 ms
Performance= 3384.09 GFlop/s, Time= 0.591 msec, Size= 2000000000 Ops
```

可以看见，在保证计算正确的前提下，随着`TILE_SIZE`增大，`Performance`升高，但大于一定值后会下降，但在`TILE_SIZE`大于64的时候，计算开始出错，如下图：

```
 ./64 1 1000
Kernel Elpased Time: 0.001 ms
Performance= 2840909.00 GFlop/s, Time= 0.001 msec, Size= 2000000000 Ops
Computing result using host CPU...done.
Listing first 100 Differences > 0.000010...
    Loc(0,0)    CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(1,0)    CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(2,0)    CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(3,0)    CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(4,0)    CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(5,0)    CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(6,0)    CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(7,0)    CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(8,0)    CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(9,0)    CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(10,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(11,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(12,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(13,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(14,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(15,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(16,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(17,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(18,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(19,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(20,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(21,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(22,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(23,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(24,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(25,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(26,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(27,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(28,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
    Loc(29,0)   CPU=750.00000   GPU=0.00000     Diff=750.000000
    Loc(30,0)   CPU=375.00000   GPU=0.00000     Diff=375.000000
```

实验结果及原理分析

根据实验测出的结果，我们可以得到下表：

矩阵尺寸	TILE_WIDTH	Performance(GFlopps)
1000	2	473
1000	4	1499
1000	8	2936
1000	16	3602
1000	32	3384
3000	32	2527

矩阵尺寸	TILE_WIDTH	Performance(GFlopps)
5000	32	2456
10000	32	2334
20000	32	1883

我们可以得到结论：随着矩阵尺寸的增大，`Performance`降低，随着`TILE_SIZE`增大，`Performance`升高但大于一定值后会下降。主要有以下原因：

- `TILE_WIDTH`增大时，每个线程块处理的矩阵区域变大，因此所需的线程块总数减少。减少线程块总数可以降低 GPU 的调度开销，使得 GPU 能更加专注于计算任务，而非频繁调度线程块。更大的线程块可以更好地利用 GPU 的并行计算能力，因为每个 SM 可以处理更多的活跃线程，从而提高吞吐量。
- 当矩阵尺寸增大时，带宽的限制会使得性能下降。当矩阵尺寸变大时，数据可能无法完全放入共享内存或缓存，从而需要频繁访问慢速的全球存储器，进一步拖慢性能。
- 当TILE_WIDTH过大时，线程块总数减少，GPU 的并行计算能力不能完全发挥。

实验 5

实验流程

完善代码

得到以下代码：

```
const int BLOCK_SIZE = TILE_WIDTH;
__global__ void MatrixMulSharedMemKernel(float *A,
    float *B, float *C, int wA,
    int wB) {
  // Block index
  int bx = blockIdx.x;
  int by = blockIdx.y;

  // Thread index
  int tx = threadIdx.x;
  int ty = threadIdx.y;

  // Index of the first sub-matrix of A processed by the block
  int aBegin = wA * BLOCK_SIZE * by;

  // Index of the last sub-matrix of A processed by the block
  int aEnd   = aBegin + wA - 1;

  // Step size used to iterate through the sub-matrices of A
  int aStep  = BLOCK_SIZE;

  // Index of the first sub-matrix of B processed by the block
  int bBegin = BLOCK_SIZE * bx;
```

```
  // Step size used to iterate through the sub-matrices of B
  int bStep  = BLOCK_SIZE * wB;

  // Csub is used to store the element of the block sub-matrix
  // that is computed by the thread
  float Csub = 0;

  // Loop over all the sub-matrices of A and B
  // required to compute the block sub-matrix
  for (int a = aBegin, b = bBegin;
       a <= aEnd; // Ensure all tiles are covered
       a += aStep, b += bStep) {
    // Declaration of the shared memory array As used to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each **thread** loads
    // one element of each matrix
    // --- TO DO :Load the elements of the sub-matrix of A into As ---
    int aRow = a / wA + ty; // Calculate row in A
    int aCol = a % wA + tx; // Calculate column in A
    if (aRow < wA && aCol < wA)
      As[ty][tx] = A[aRow * wA + aCol];
    else
      As[ty][tx] = 0.0f;

    // ---        Load the elements of the sub-matrix of B into Bs ---
    int bRow = b / wB + ty; // Calculate row in B
    int bCol = b % wB + tx; // Calculate column in B
    if (bRow < wA && bCol < wB)
      Bs[ty][tx] = B[bRow * wB + bCol];
    else
      Bs[ty][tx] = 0.0f;

    // Synchronize to make sure the matrices are loaded
    __syncthreads();

    // Multiply the two matrices together;
    // each thread computes one element
    // of the block sub-matrix
#pragma unroll
    // --- TO DO :Implement the matrix multiplication using the sub-matrices As
and Bs ---
    for (int k = 0; k < BLOCK_SIZE; ++k) {
        Csub += As[ty][k] * Bs[k][tx];
    }

    // Synchronize to make sure that the preceding
    // computation is done before loading two new
    // sub-matrices of A and B in the next iteration
```

```
        __syncthreads();
    }

    // Write the block sub-matrix to device memory;
    // each thread writes one element
    int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
    // --- TO DO :Store the computed Csub result into matrix C ---
    int row_C = by * BLOCK_SIZE + ty;
    int col_C = bx * BLOCK_SIZE + tx;
    if (row_C < wA && col_C < wB)
        C[c + ty * wB + tx] = Csub;
}
```

修改`main()`使得使用`MatrixMulKernel<<<grid, block>>>(d_M, d_N, d_P, m);`

```
for (int j = 0; j < nIter; j++) {
    // matrixMulCPU(reference, h_M, h_N, m, k, n);
    // MatrixMulKernel<<<grid, block>>>(d_M, d_N, d_P, m);
       MatrixMulSharedMemKernel<<<grid, block>>>(d_M, d_N, d_P, m, n);
    // cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, n, m, k, &alpha, d_N, n,
d_M, k, &beta, d_P, n);
}
```

编译执行

```bash
bash compile.sh
./MatrixMulSharedMemKernel 1 1000
```

观察发现计算结果正确

```
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-
 ./MatrixMulSharedMemKernel 1 1000
Kernel Elpased Time: 0.483 ms
Performance= 4142.03 GFlop/s, Time= 0.483 msec, Size= 2000000000 Ops
Computing result using host CPU...done.
Listing first 100 Differences > 0.000010...

  Total Errors = 0
```

更改矩阵尺寸，对比不同参数下的计算结果

```
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main
 ./MatrixMulSharedMemKernel 0 1000
Kernel Elpased Time: 0.483 ms
Performance= 4144.29 GFlop/s, Time= 0.483 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main
 ./MatrixMulSharedMemKernel 0 3000
Kernel Elpased Time: 12.652 ms
Performance= 4268.07 GFlop/s, Time= 12.652 msec, Size= 54000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main
 ./MatrixMulSharedMemKernel 0 5000
Kernel Elpased Time: 62.401 ms
Performance= 4006.37 GFlop/s, Time= 62.401 msec, Size= 250000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main
 ./MatrixMulSharedMemKernel 0 10000
Kernel Elpased Time: 497.037 ms
Performance= 4023.84 GFlop/s, Time= 497.037 msec, Size= 2000000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main
 ./MatrixMulSharedMemKernel 0 20000
Kernel Elpased Time: 4010.275 ms
Performance= 3989.75 GFlop/s, Time= 4010.275 msec, Size= 16000000000000 Ops
```

可以看见，随着矩阵尺寸的增大，`Performance`也下降

更改`TILE_SIZE`，对比不同参数下的计算结果

```
 ./2 0 1000
Kernel Elpased Time: 19.070 ms
Performance= 104.88 GFlop/s, Time= 19.070 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./4 0 1000
Kernel Elpased Time: 2.557 ms
Performance= 782.25 GFlop/s, Time= 2.557 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./8 0 1000
Kernel Elpased Time: 0.712 ms
Performance= 2809.92 GFlop/s, Time= 0.712 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./16 0 1000
Kernel Elpased Time: 0.480 ms
Performance= 4163.45 GFlop/s, Time= 0.480 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-202
 ./32 0 1000
Kernel Elpased Time: 0.480 ms
Performance= 4164.72 GFlop/s, Time= 0.480 msec, Size= 2000000000 Ops
```

调用`CUDA`自带的矩阵乘法算子，编译运行

```
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-20
 ./cublasSgemm 1 1000
Kernel Elpased Time: 216.314 ms
Performance= 9.25 GFlop/s, Time= 216.314 msec, Size= 2000000000 Ops
Computing result using host CPU...done.
Listing first 100 Differences > 0.000010...

  Total Errors = 0
```

更改矩阵尺寸，对比不同参数下的计算结果

```
 ./cublasSgemm 0 1000
Kernel Elpased Time: 204.143 ms
Performance= 9.80 GFlop/s, Time= 204.143 msec, Size= 2000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main/l
 ./cublasSgemm 0 3000
Kernel Elpased Time: 208.129 ms
Performance= 259.46 GFlop/s, Time= 208.129 msec, Size= 54000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main/l
 ./cublasSgemm 0 5000
Kernel Elpased Time: 219.477 ms
Performance= 1139.07 GFlop/s, Time= 219.477 msec, Size= 250000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main/l
 ./cublasSgemm 0 10000
Kernel Elpased Time: 298.059 ms
Performance= 6710.07 GFlop/s, Time= 298.059 msec, Size= 2000000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main/l
 ./cublasSgemm 0 20000
Kernel Elpased Time: 1187.059 ms
Performance= 13478.69 GFlop/s, Time= 1187.059 msec, Size= 16000000000000 Ops
(base) inspur@inspur:/data/inspur/workspace-j0hnny/HITSZ-Comp-Arch-2024-main/l
 ./cublasSgemm 0 40000
Kernel Elpased Time: 8640.741 ms
Performance= 14813.54 GFlop/s, Time= 8640.741 msec, Size= 128000000000000 Ops
```

`TILE_WIDTH`不影响`cublasSgemm`，故不进行测试。

实验结果及原理分析

我们重点分析三种方法之间性能的差异，经过整理，得到下表：

方法	矩阵尺寸	TILE_WIDTH	Performance(GFlopps)
MatrixMulKernel	1000	2	473
MatrixMulKernel	1000	4	1499
MatrixMulKernel	1000	8	2936
MatrixMulKernel	1000	16	3602
MatrixMulKernel	1000	32	3384
MatrixMulKernel	3000	32	2527
MatrixMulKernel	5000	32	2456

方法	矩阵尺寸	TILE_WIDTH	Performance(GFlopps)
MatrixMulKernel	10000	32	2334
MatrixMulKernel	20000	32	1883
MatrixMulSharedMemKernel	1000	2	108
MatrixMulSharedMemKernel	1000	4	782
MatrixMulSharedMemKernel	1000	8	2809
MatrixMulSharedMemKernel	1000	16	4163
MatrixMulSharedMemKernel	1000	32	4144
MatrixMulSharedMemKernel	3000	32	4268
MatrixMulSharedMemKernel	5000	32	4006
MatrixMulSharedMemKernel	10000	32	4023
MatrixMulSharedMemKernel	20000	32	4010
cublasSgemm	500	/	1.3
cublasSgemm	1000	/	9.8
cublasSgemm	3000	/	259
cublasSgemm	5000	/	1139
cublasSgemm	10000	/	6710
cublasSgemm	20000	/	13478
cublasSgemm	40000	/	14813

分析数据我们可以知道

- `MatrixMulSharedMemKernel`比`MatrixMulKernel`表现要好，而且随着举证变大性能下降并不明显。
- `cublasSgemm`在矩阵较小的时候性能并不好，比其他两种方法相差很多，但是随着矩阵增大，其性能也升高明显，最后比其他两种方法优秀很多。

分析原理

- `MatrixMulSharedMemKernel`有效地减少了全局内存访问次数，而共享内存的性能比全局内存强很多，并且更好地利用了GPU的缓存层次结构，因此性能更好。
- `cublasSgemm`在小矩阵时性能不佳是因为其优化针对大规模并行计算，存在启动开销；随着矩阵增大，其高效利用GPU资源和高度优化的算法特性得以充分发挥，从而展现出显著的性能优势。