上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

CS4302-01
Parallel and Distributed Computing

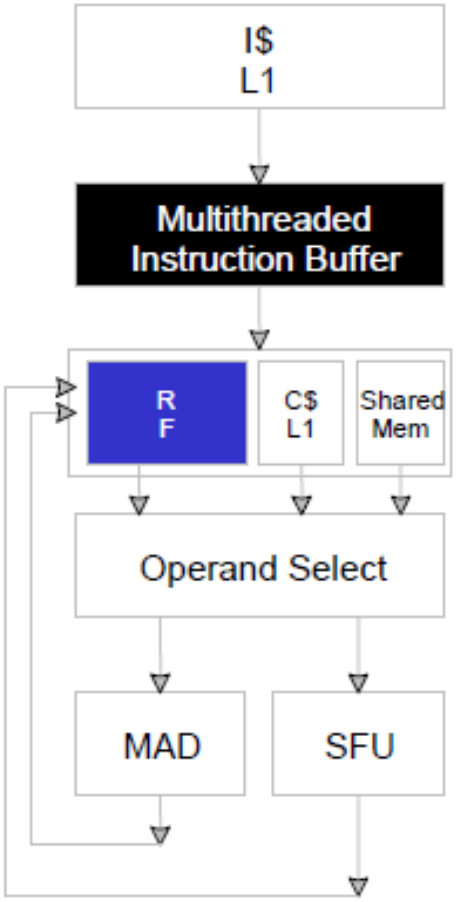
Lecture 8 CUDA Optimization

Zhuoran Song

2023/10/24

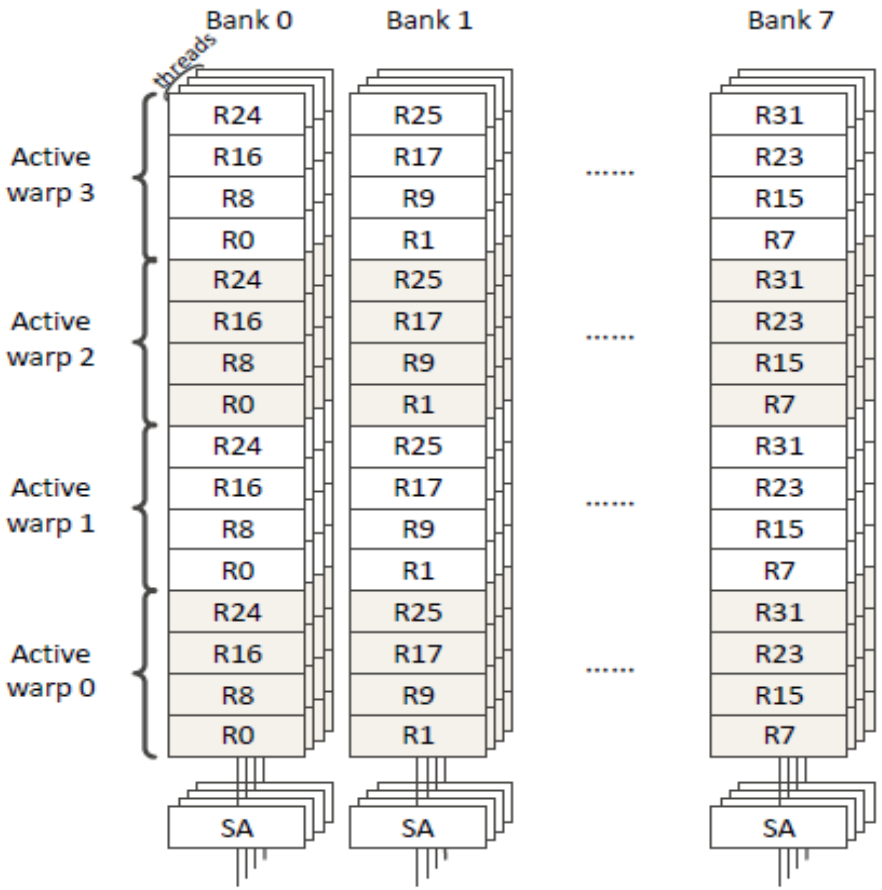
Shared Memory

- Faster than global memory
- Reduce access time by reusing data
- Threads can cooperate through shared memory
- Avoid accesses that cannot be coalesce
  - Rearrange data



Shared Memory

- 很多线程访问shared memory
  - 被分为bank
  - 连续32-bit访存被分到连续bank
- 每个bank每个周期可以相应一个地址
  - 如果有多个bank则可以相应多个地址
- 对同一bank进行多个并发访问，会产生冲突
  - 冲突必须串行执行



Bank Confilct

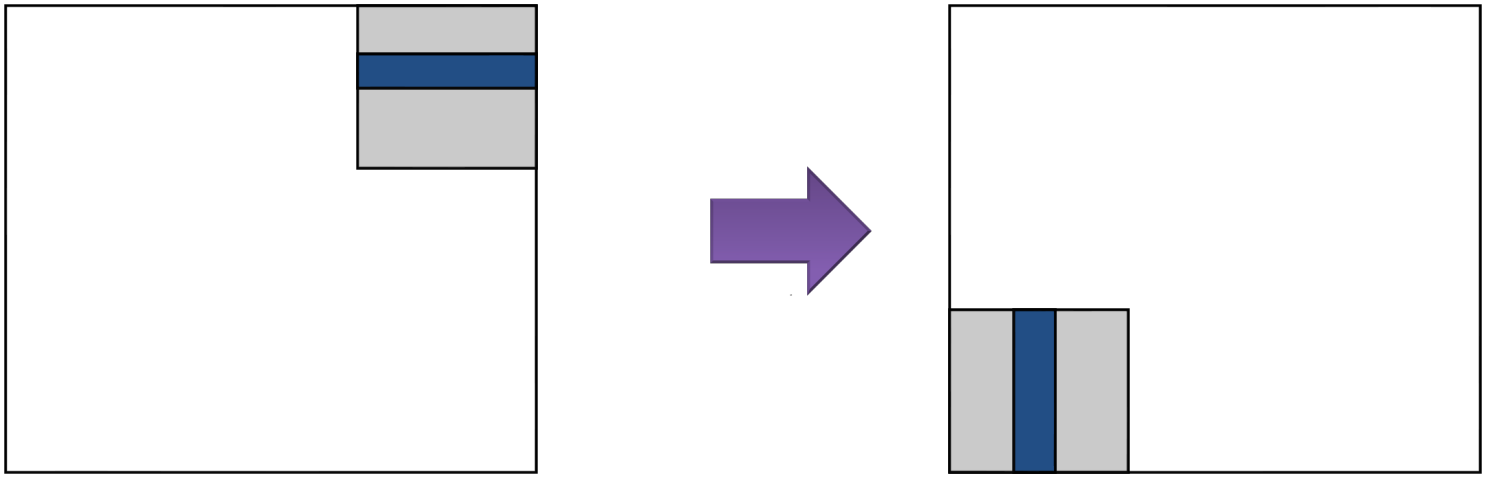
- **Shared memory is as fast as registers if there are no bank conflicts**

- **The fast case:**
  - *If all threads access different banks, there is no bank conflict*
  - *If all threads access the identical address, there is no bank conflict (broadcast)*

- **The slow case:**
  - *Bank Conflict: multiple threads access the same bank*
  - *Must serialize the accesses*
  - *Cost = max # of simultaneous accesses to a single bank*

Case study：Matrix Transpose

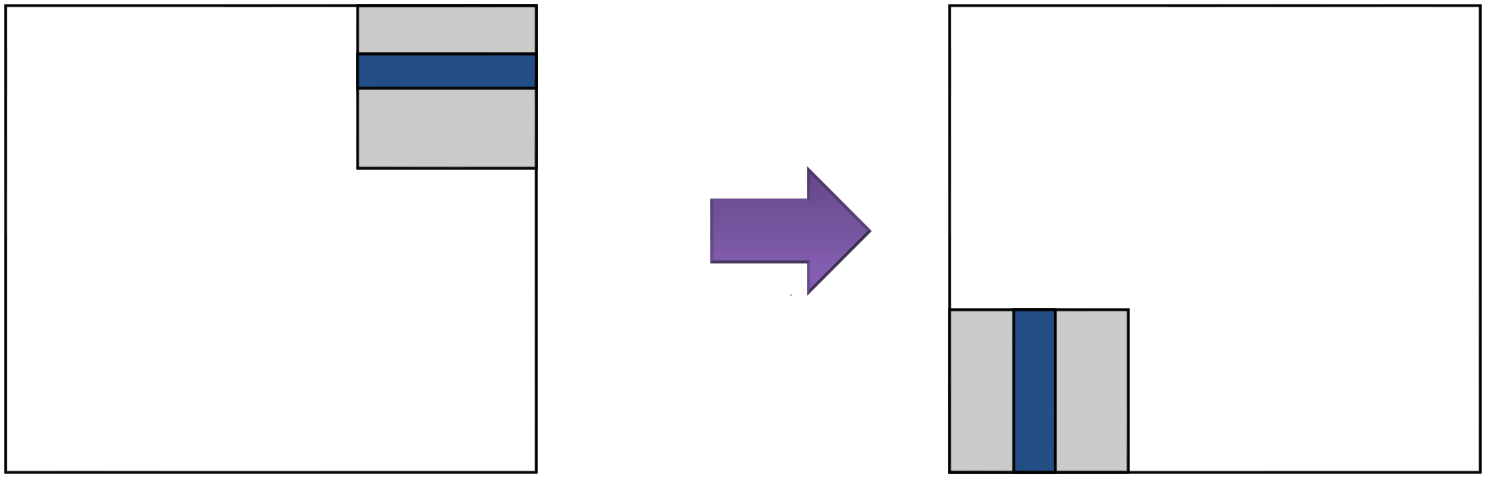
- 每个线程块在矩阵的一个瓦片（**tile**）操作
- 原始版本存在对**global memory**按步长访问的情况



Case study：Matrix Transpose

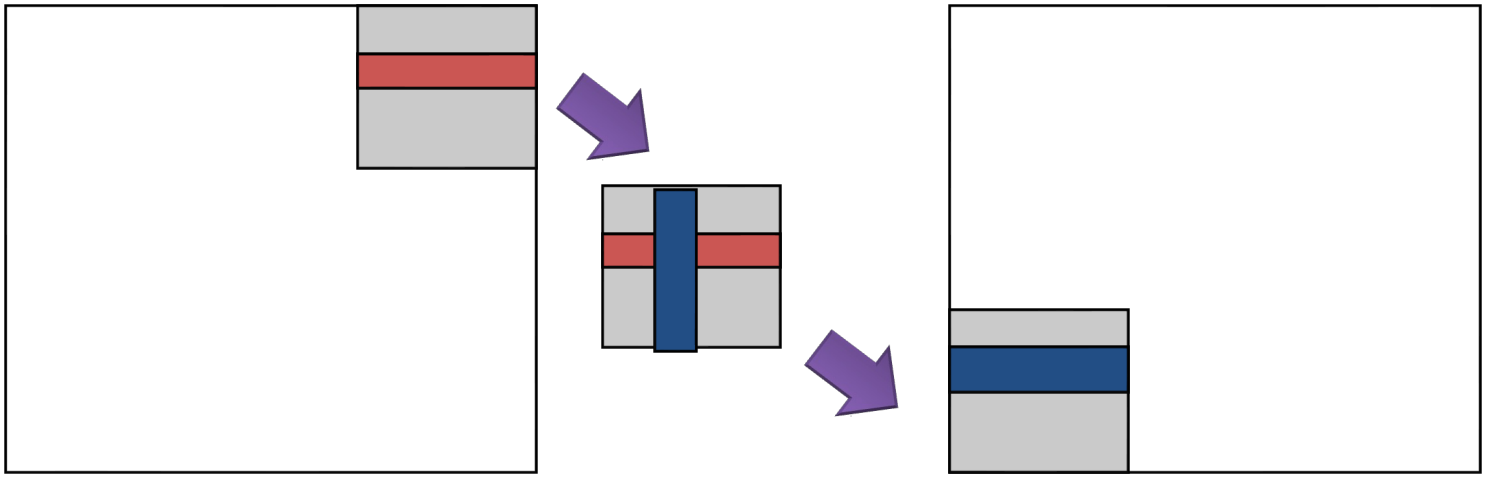
- 读操作支持合并，写操作不支持

```
__global__ void transposeNaive(float *odata, const float *idata) {
        int x = blockIdx.x * TILE_DIM + threadIdx.x;
        int y = blockIdx.y * TILE_DIM + threadIdx.y;
        int width = gridDim.x * TILE_DIM;
        for (int j = 0; j < TILE_DIM; j+= BLOCK_ROWS){
                odata[x*width + (y+j)] = idata[(y+j)*width + x];
        }
}
```



Case study：Matrix Transpose

- 通过shared memory实现合并
- 先将tile的多列元素存入shared memory，再以连续化的数据写入global memory
- 需要同步 __syncthreads()

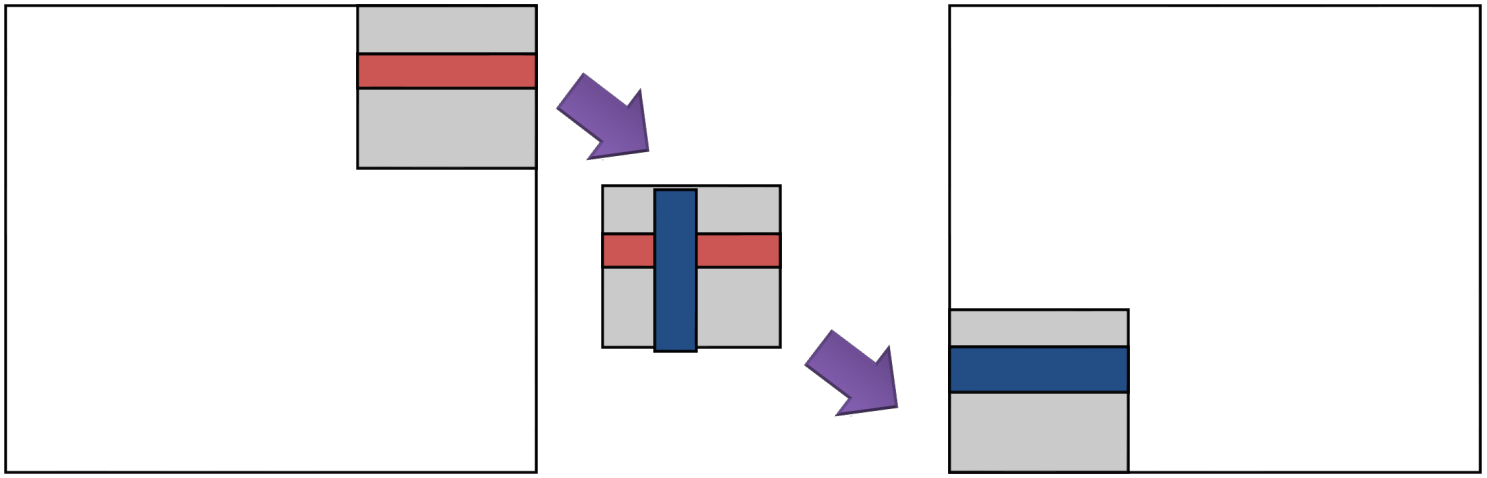


Case study：Matrix Transpose

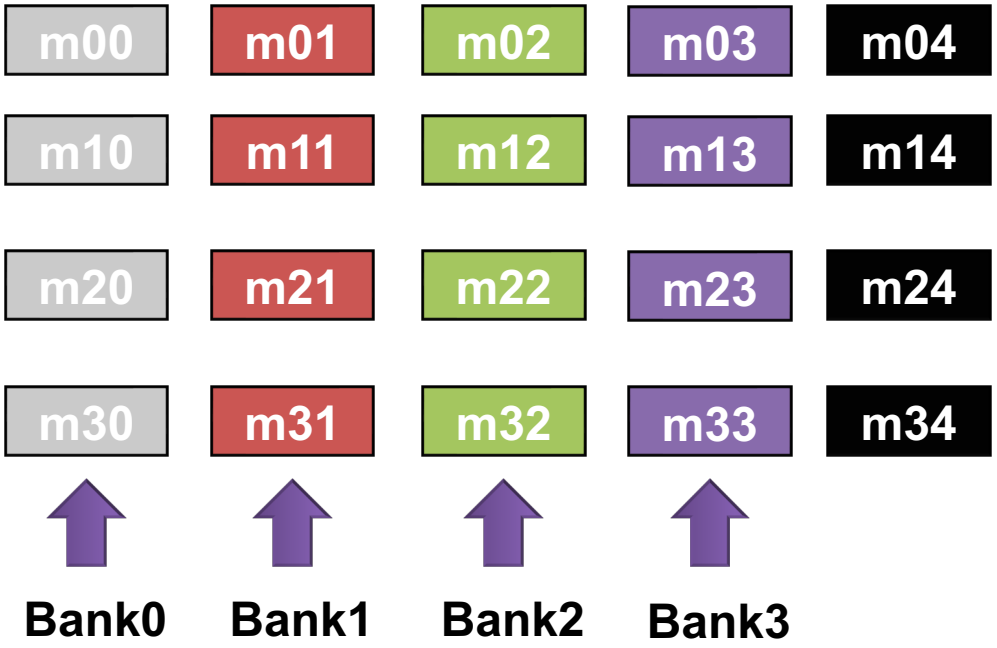
```
__global__ void transposeCoalesced(float *odata, const float *idata, int
width, int height) {
        __shared__ float tile[TILE_DIM][TILE_DIM];
        int x = blockIdx.x * TILE_DIM + threadIdx.x;
        int y = blockIdx.y * TILE_DIM + threadIdx.y;
        int index_in = x + y* width;

        x = blockIdx.y * TILE_DIM + threadIdx.x;
        y = blockIdx.x * TILE_DIM + threadIdx.y;
        int index_out = x + y* height;
        tile[threadIdx.y][threadIdx.x] = idata[index_in];
        __syncthreads();
        odata[index_out] = tile[threadIdx.x][threadIdx.y];
}
```

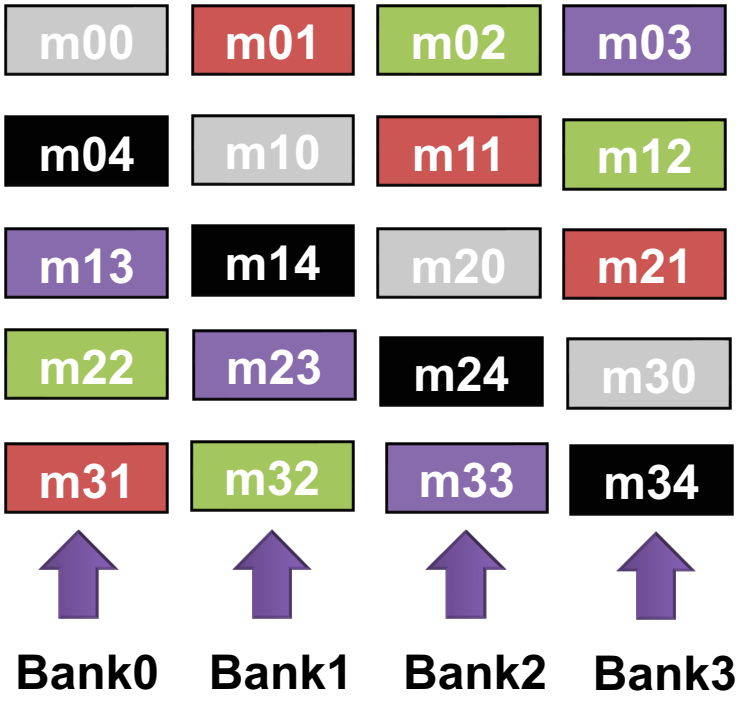
Case study：Matrix Transpose

- Tile内16*16个floats存于shared memory
  - 列中的数据存于相同的bank
  - 读入tile一列数据存在16-way bank conflict
- 解决方法：填充shared memory数组
  - __shared__ float tile[TILE_DIM][TILE_DIM+1];



Case study : Matrix Transpose





Data Prefetching

- 在一次**global memory**读操作与实际用到该数据的语句中间插入独立于以上数据的指令
  - 隐藏访存

```
float m = Md[i];
float f = a * b + c;
float f2 = m * f;
```

- 从**global memory**预取数据可以有效提升独立指令的数量，减少访存开销

Data Prefetching

- 回顾tile based 矩阵乘法

```
__global__ void matrix_multiplication(float *odata, const float
*idata) {
        for (…){
        // Load current tile into shared memory
        __syncthreads();
        // Accumulate dot product
        __syncthreads();
        }
}
```

Data Prefetching

- 回顾tile based 矩阵乘法

```
__global__ void matrix_multiplication(float *odata, const float
*idata) {
        // Load current tile into registers
        for (…){
        // Deposit registers into shared memory
        __syncthreads();
        // Load next tile into registers
        // Accumulate dot product
        __syncthreads();
        }
}
```

Optimization for IPC

- 指令优化
  - 如果不够仔细，计算密集型算法很容易受限于带宽
  - 典型情况，在存储器和执行配置优化完成后，担心指令优化
- **Int multiplication: 2 cycles**
- **Int divide and module are expensive**
  - **2^n：采用>>n**
  - **以2^n求模，采用&(2^n-1)**
- **避免double到float类型的自动转换**
  - **添加”f”到float常量，缺省为double**

Optimization for IPC

- 循环展开

```
for (int k = 0; k < Block_size; ++k){
        Pvalue += Ms[ty][k] * Ns[k][tx];
}
```

每轮循环包括到指令：

一条浮点数乘法

一条浮点数加法

？

Optimization for IPC

- 循环展开

```
for (int k = 0; k < Block_size; ++k){
        Pvalue += Ms[ty][k] * Ns[k][tx];
}
```

每轮循环包括到指令：

一条浮点数乘法

一条浮点数加法

更新循环计数器

分支

地址运算

Optimization for IPC

- 循环展开

```
for (int k = 0; k < Block_size; ++k){
        Pvalue += Ms[ty][k] * Ns[k][tx];
}
```

循环展开的缺点？

可扩展性

错误