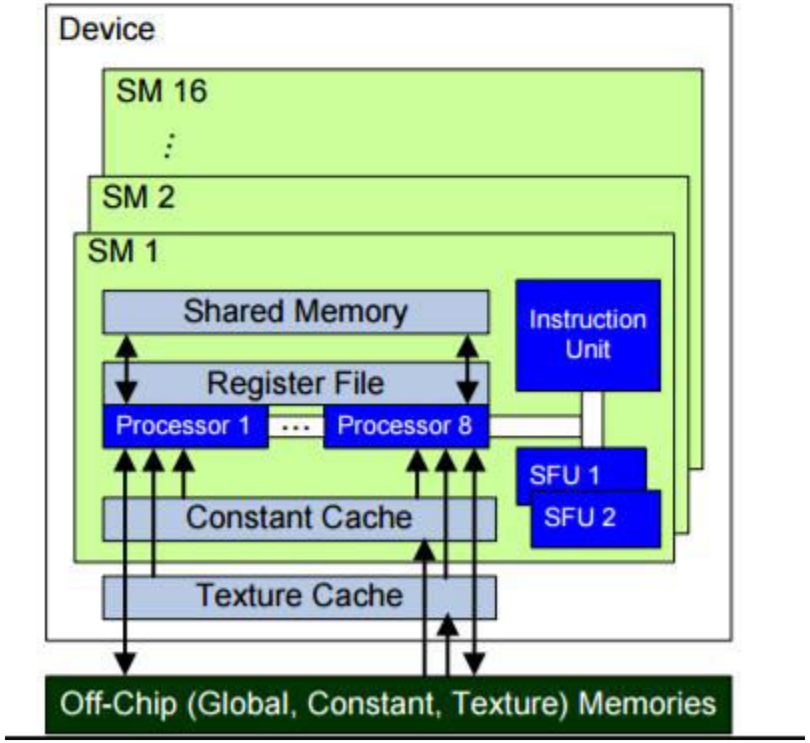
Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA

GPU architecture



Memory	Location	Size	Hit Latency	Read-Only	Program Scope
Global	off-chip	768MB total	200-300 cycles	no	global
Local	off-chip	up to global	same as global	no	function
Shared	on-chip	16KB per SM	$\simeq$register latency	no	function
Constant	on-chip cache	64KB total	$\simeq$register latency	yes	global
Texture	on-chip cache	up to global	$>$100 cycles	yes	global

GeForce 8800 GPU

- 16 Streaming multiprocessors
- 8 Streaming processors pr SM
- 8192 registers pr SM
- 768 threads pr SM
- 8 blocks can be run at a time pr SM
- 32 thread warp

Example

- 4K by 4K matrix multiplication
- 768 threads.
- 10 registers pr thread
- Potential throughput of 43.2 FLOPS
- Performance is 10.58 FLOPS
- Global memory access

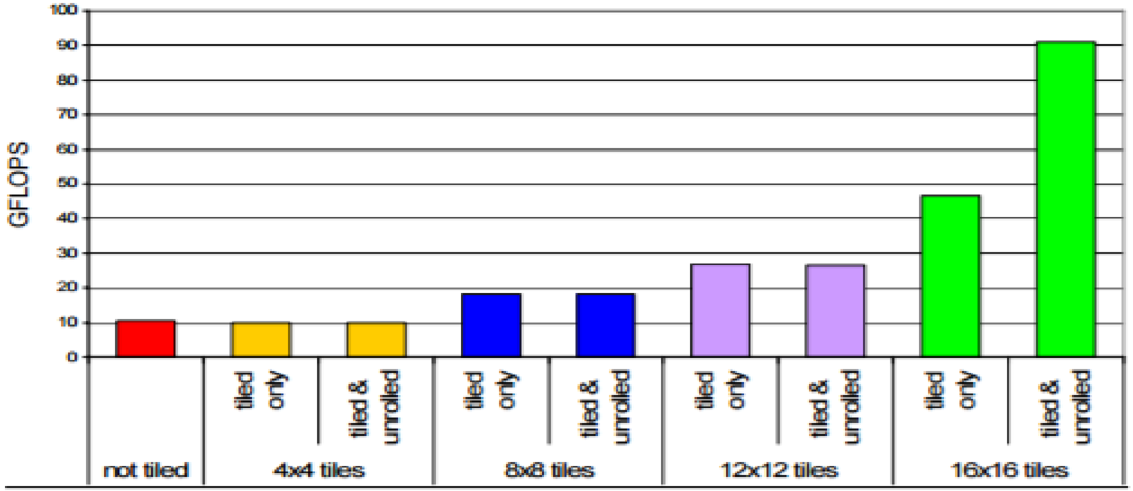
```
Ctemp = 0;

for (i = 0; i < widthA;
     i++)
  {
    Ctemp += A[indexA]
      * B[indexB];
    indexA++;
    indexB += widthB;
  }

C[indexC] = Ctemp;
```

Tiling

```
Ctemp = 0;
for (...) {
  __shared__ float As[16][16];
  __shared__ float Bs[16][16];

  // load input tile elements
  As[ty][tx] = A[indexA];
  Bs[ty][tx] = B[indexB];
  indexA += 16;
  indexB += 16 * widthB;
  __syncthreads();

  // compute results for tile
  for (i = 0; i < 16; i++)
    {
      Ctemp += As[ty][i]
        * Bs[i][tx];
    }

  __syncthreads();
}
C[indexC] = Ctemp;
```

- Doing computations on smaller «tiles»
- Put tiles in shared memory
  - 4x4 – Only 16 threads, half warps
  - 8x8 – Occupies 2 warps, but need 12 blocks to use all threads. Can only use 8.
  - 12x12 – 144 threads which is not divisible by 32 (warp size).
  - 16x16 – 256/32 = 8 warps. Use three blocks: 256*3 = 768
- Reduced global load by a factor of 16
- 46.49 GFLOPS
- 3 blocks/SM * 256 threads/block

* 11 registers = 8488 registers > 8192



Unrolling

- Unroll the loop
- Removing
  - Loop address calculation instructions
  - Iterator variable increments (register)
- Potential throughput: 93.72 GFLOPS
- Performance: 91.14 GFLOPS

```
Ctemp = 0;
for (...) {
  __shared__ float As[16][16];
  __shared__ float Bs[16][16];

  // load input tile elements
  As[ty][tx] = A[indexA];
  Bs[ty][tx] = B[indexB];
  indexA += 16;
  indexB += 16 * widthB;
  __syncthreads();

  // compute results for tile
  Ctemp +=
     As[ty][0] * Bs[0][tx];
  ...
  Ctemp +=
     As[ty][15] * Bs[15][tx];

  __syncthreads();
}
C[indexC] = Ctemp;
```