CUDA Programming

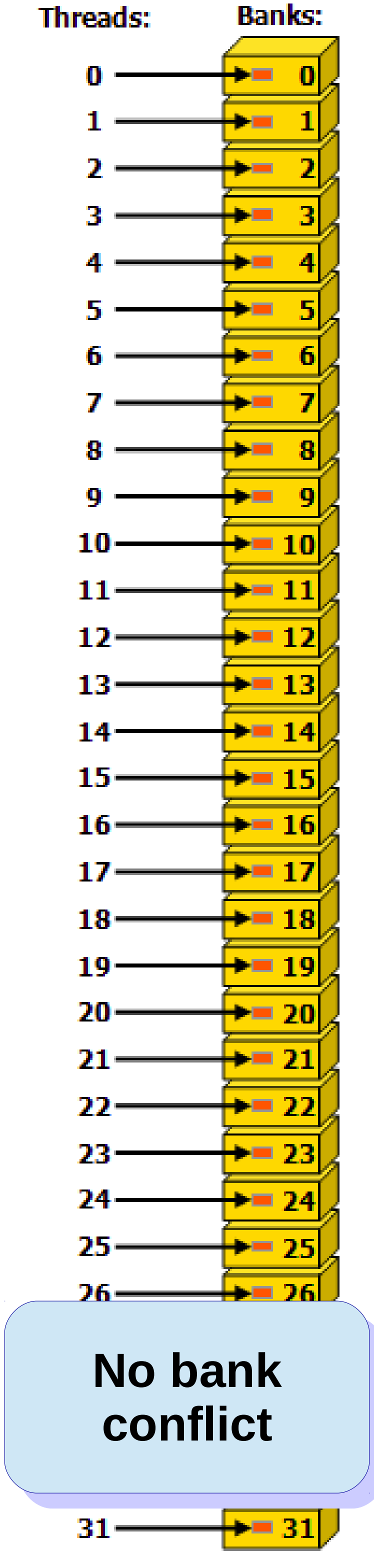
Bank Conflicts

- Shared memory is organized into 32 banks.
- Accesses to the same bank are sequential.
- Consecutive words are stored in adjacent banks.
  - Useful for coalesced access.
- **Exception:** Warp accesses to the same word are not sequentialized.

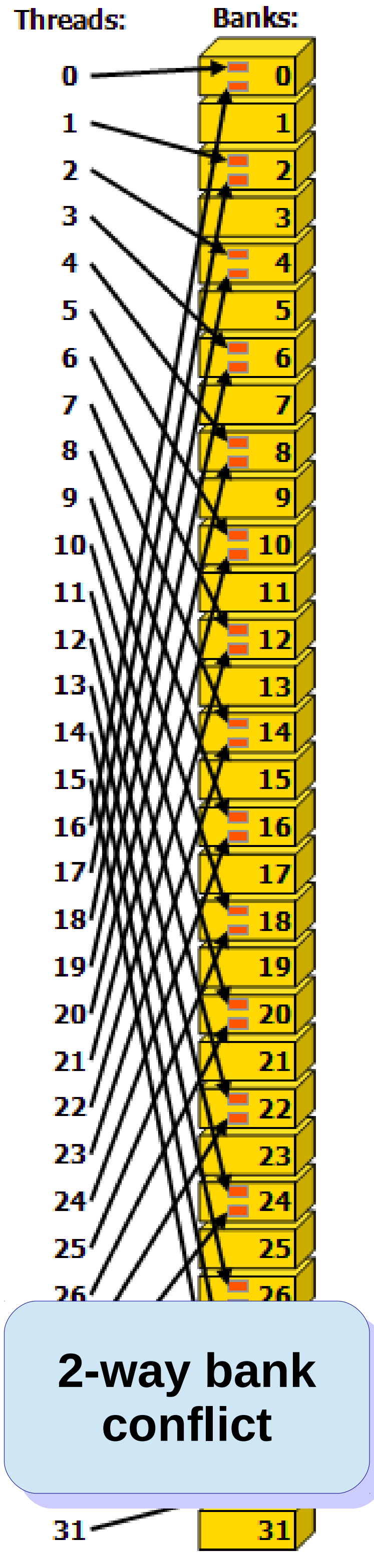
```
__global__ void bankNOconflict() {
        __shared__ unsigned s[1024];
        s[1 * threadIdx.x] = threadIdx.x;
}
```

```
__global__ void bankconflict() {
        __shared__ unsigned s[1024];
        s[32 * threadIdx.x] = threadIdx.x;
}
```

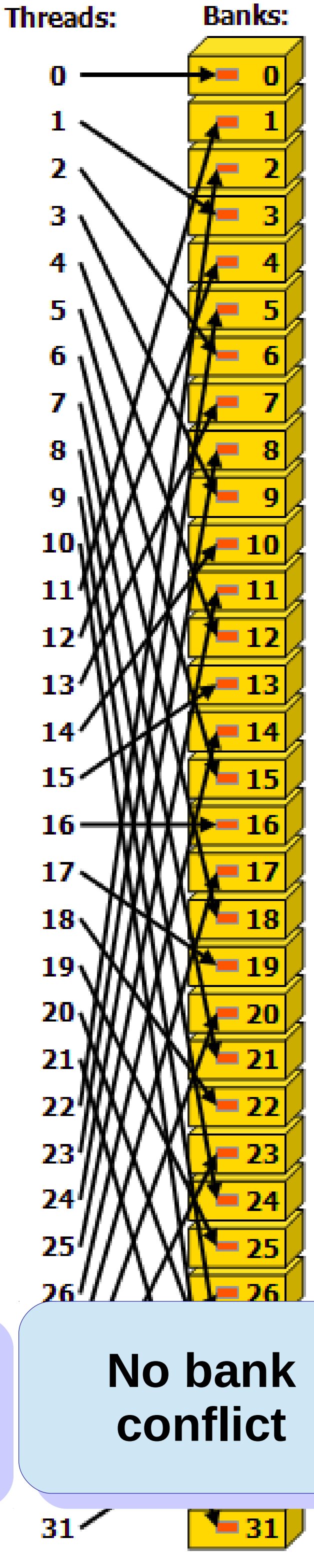
Source: CUDA Programming Guide



No bank conflict

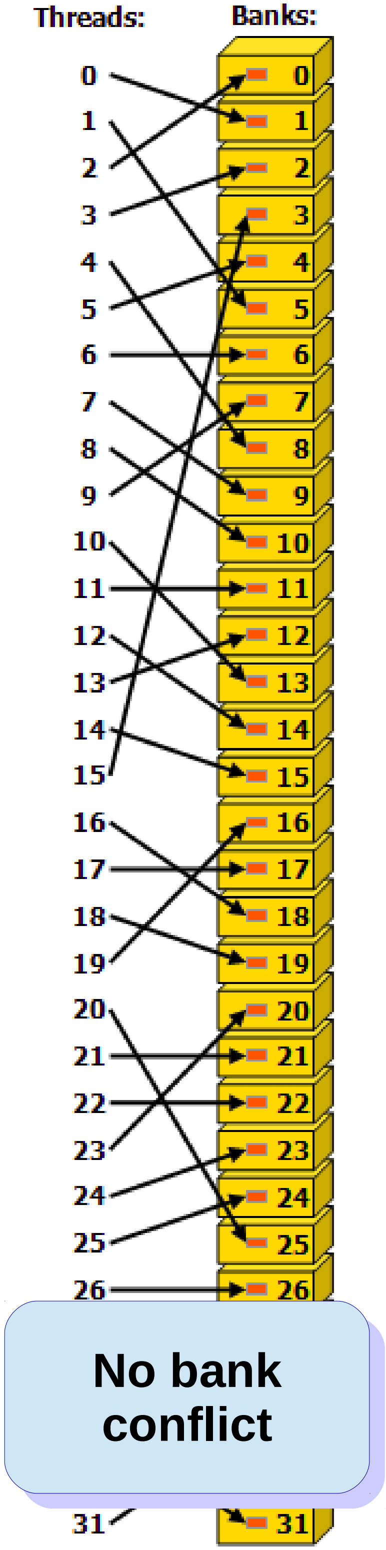


2-way bank conflict

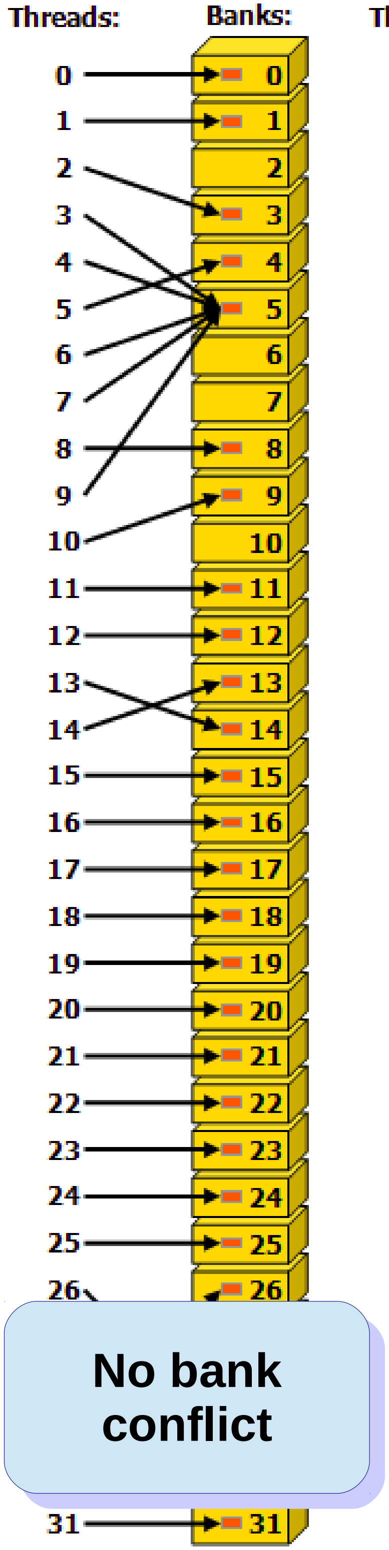


No bank conflict

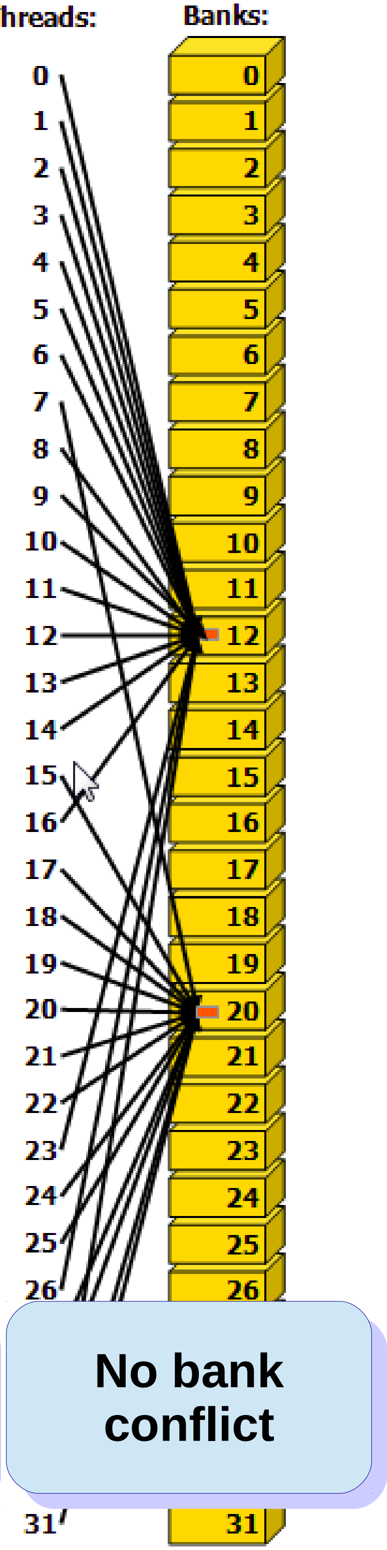
Source: CUDA Programming Guide



No bank conflict



No bank conflict



No bank conflict

Texture Memory

- Fast read-only memory
- Optimized for 2D spatial access
- Definition: texture<float, 2, cudaReadModeElementType> *tex*;
- In main: cudaBindTextureToArray(*tex*, cuArray, …);
- In kernel: ... = tex2D(*tex*, …);

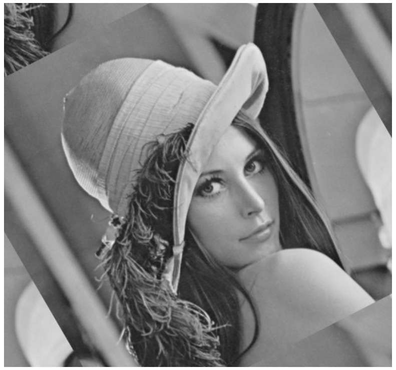
Texture Memory

- Example from CUDA SDK

```
__global__ void transformKernel(float *output, int width, int height, float theta)
{
        unsigned x = blockIdx.x * blockDim.x + threadIdx.x;
        unsigned y = blockIdx.y * blockDim.y + threadIdx.y;

        float u = (float)x - (float)width / 2;
        float v = (float)y - (float)height / 2;
        float tu = (u * cosf(theta) - v * sinf(theta)) / width;
        float tv = (v * cosf(theta) + u * sinf(theta)) / height;

        output[y * width + x] = tex2D(tex, tu + 0.5, tv + 0.5);
}
```



Constant Memory

- Read-only Memory
- 64KB per SM
- Definition: **__constant__** unsigned meta[1];
- Main: **cudaMemcpyToSymbol**(meta, &hmeta, 1*sizeof(unsigned));
- Kernel: data[threadIdx.x] = meta[0];

Constant Memory

```
#include <cuda.h>
#include <stdio.h>

__constant__ unsigned meta[1];

__global__ void dkernel(unsigned *data) {
        data[threadIdx.x] = meta[0];
}
__global__ void print(unsigned *data) {
        printf("%d %d\n", threadIdx.x, data[threadIdx.x]);
}
int main() {

        unsigned hmeta = 10;
        cudaMemcpyToSymbol(meta, &hmeta, sizeof(unsigned));
        unsigned *data;
        cudaMalloc(&data, 32 * sizeof(unsigned));
        dkernel<<<1, 32>>>(data);
        cudaDeviceSynchronize();
        print<<<1, 32>>>(data);
        cudaDeviceSynchronize();
        return 0;
}
```

Compute Capability

- Version number: **Major.minor** (e.g., 6.2)
  - Features supported by the GPU hardware.
  - Used by the application at runtime (*-arch=sm_62*).
- CUDA version is the software version (e.g., CUDA 10.1).

Compute Capability

Major number	Architecture
1	Tesla
2	Fermi
3	Kepler
5	Maxwell
6	Pascal
7	Volta
8	Turing
9	Ampere
10	Lovelace
11	Hopper

Compute Capability

Feature	2.x	3.0	3.5, 3.7, 5.0, 5.2	6.x	7.x	8.0
Atomics int, float	Yes	Yes	Yes	Yes	Yes	Yes
warp-vote	Yes	Yes	Yes	Yes	Yes	Yes
__syncthreads	Yes	Yes	Yes	Yes	Yes	Yes
Unified memory		Yes	Yes	Yes	Yes	Yes
Dynamic parallelism			Yes	Yes	Yes	Yes
Atomics double				Yes	Yes	Yes
Tensor core					Yes	Yes
Hardware async copy						Yes

CUDA, in a nutshell

- Compute Unified Device Architecture. It is a hardware and software architecture.
- Enables NVIDIA GPUs to execute programs written with C, C++, Fortran, OpenCL, and other languages.
- A CUDA program calls parallel kernels. A kernel executes in parallel across a set of parallel threads.
- The programmer or compiler organizes these threads in thread blocks and grids of thread blocks.
- The GPU instantiates a kernel program on a grid of parallel thread blocks.
- Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results.
- A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory.
- A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, and write results to global memory.
- Each thread has a per-thread private memory space used for register spills, function calls, and C array variables.
- Each thread block has a per-block shared memory space used for inter-thread communication, data sharing, and result sharing in parallel algorithms.