CS516: Parallelization of Programs

GPU Memories

Vishwesh Jatala

Assistant Professor

Department of CSE

Indian Institute of Technology Bhilai
vishwesh@iitbhilai.ac.in

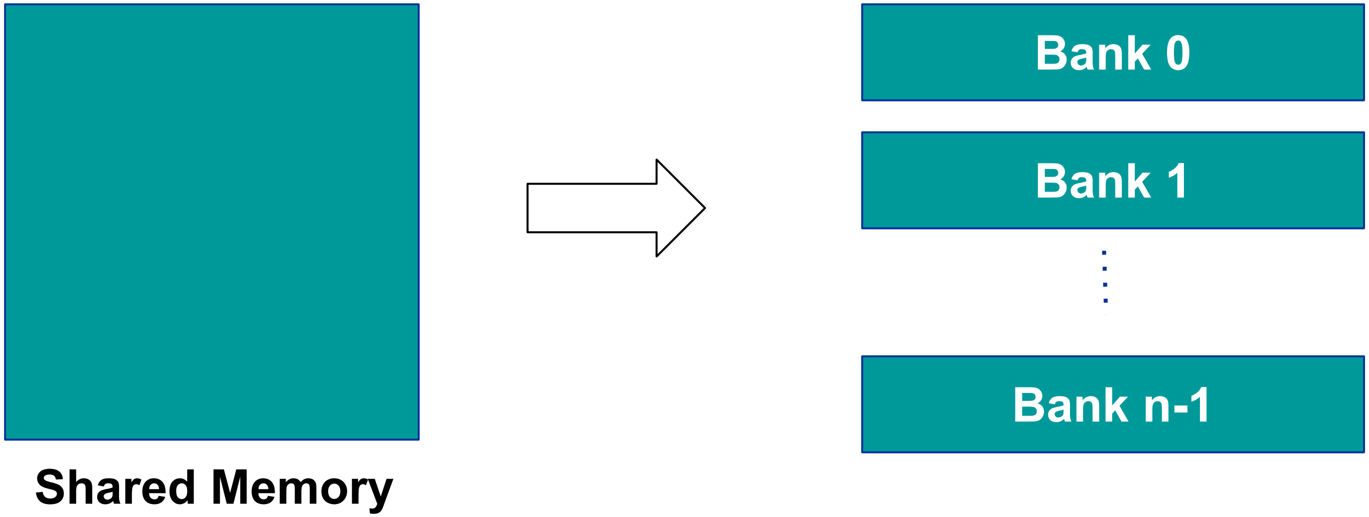
2022-23 W

References

- Miscellaneous resources from internet
- CS6023 GPU Programming
  - https://www.cse.iitm.ac.in/~rupesh/teaching/gpu/jan20/

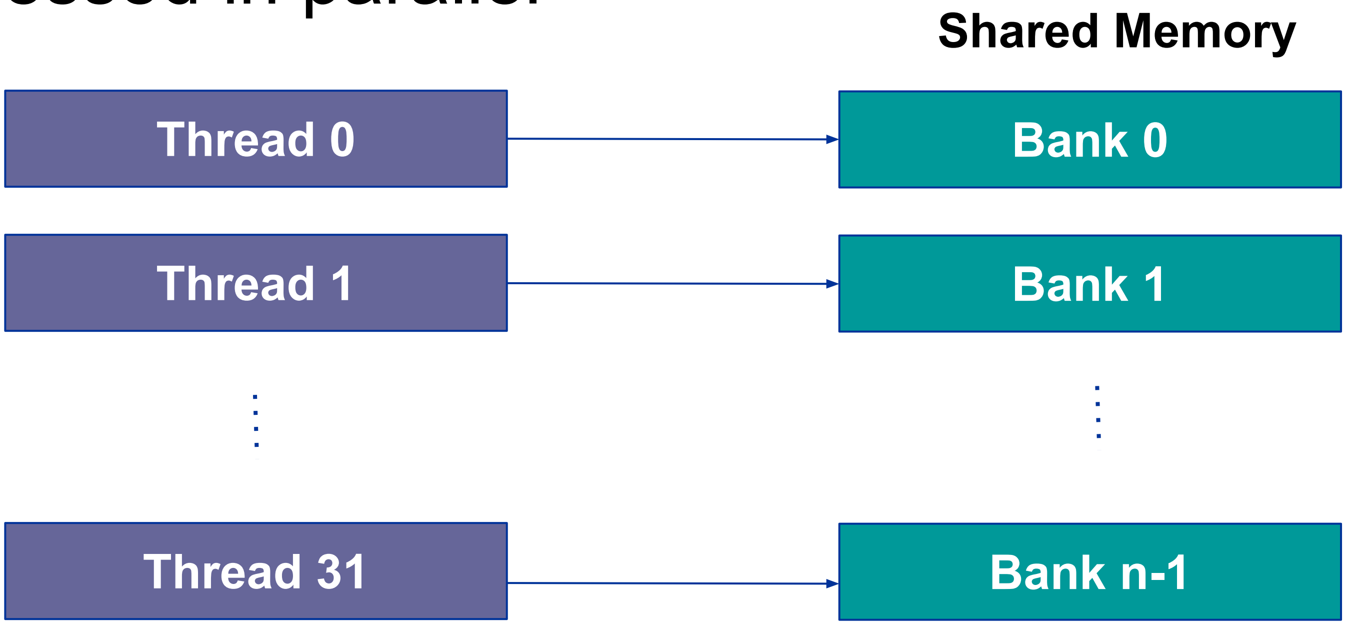
Shared Memory

- Shared memory is faster but limited in size
- To improve concurrency, shared memory is divided into banks



Shared Memory Banks

- A memory address is translated to a bank
- Addresses that fall to different banks can be accessed in parallel

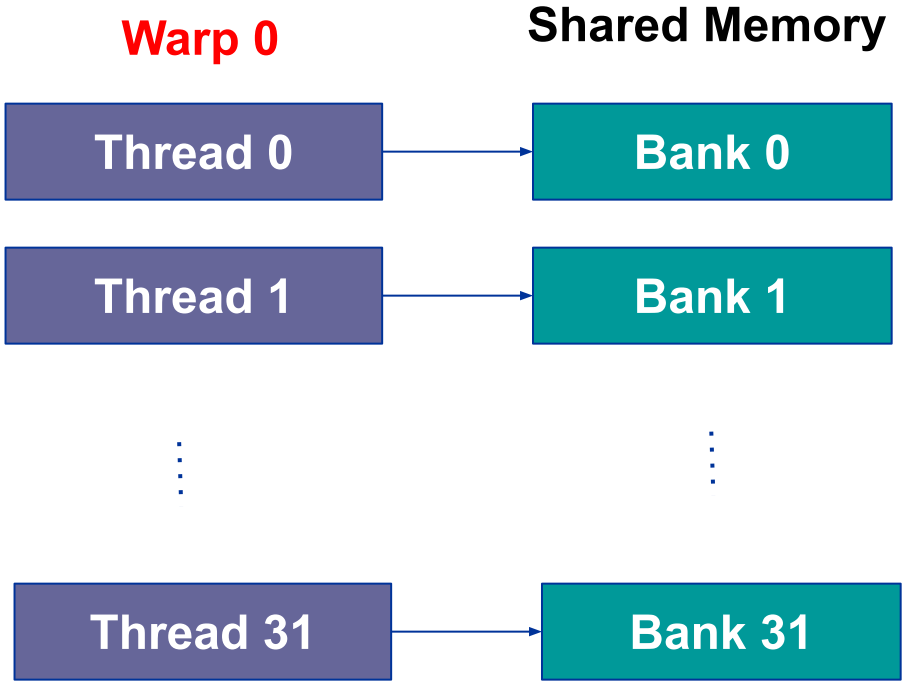


All are parallel!

Shared Memory Banks

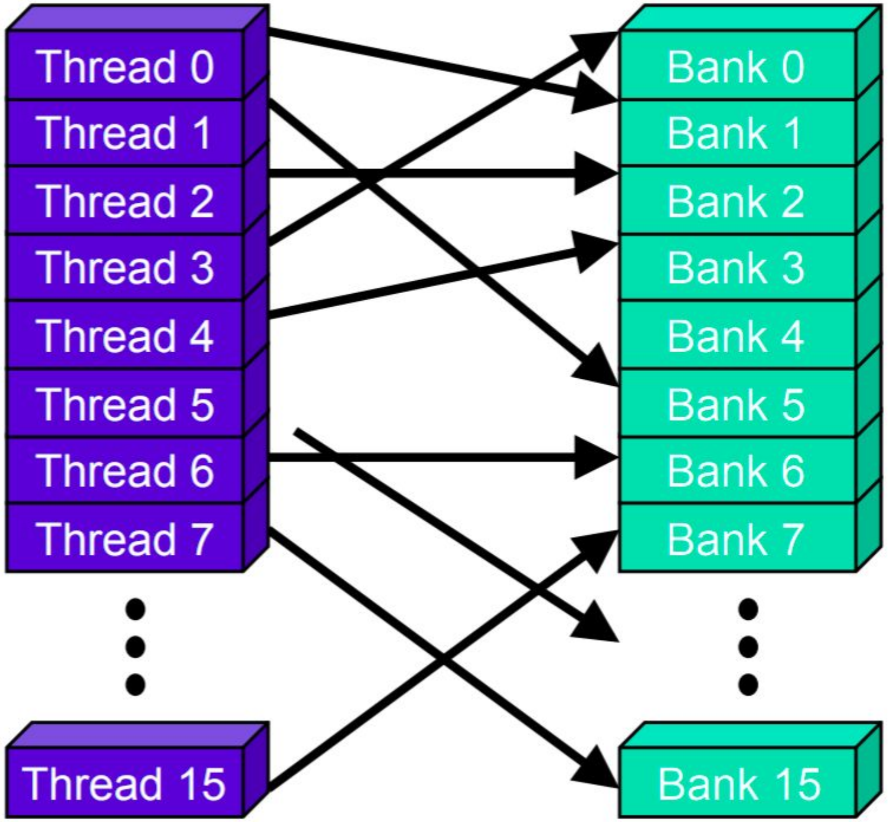
- In the modern GPUs (> 2.x), successive 32-bit words map to successive banks.
  - bank = (address / 4) % 32
- No. of banks = 32

```
__shared__ float shared[32];
int S=1;
float data = shared[S*tid];
```



All are parallel!

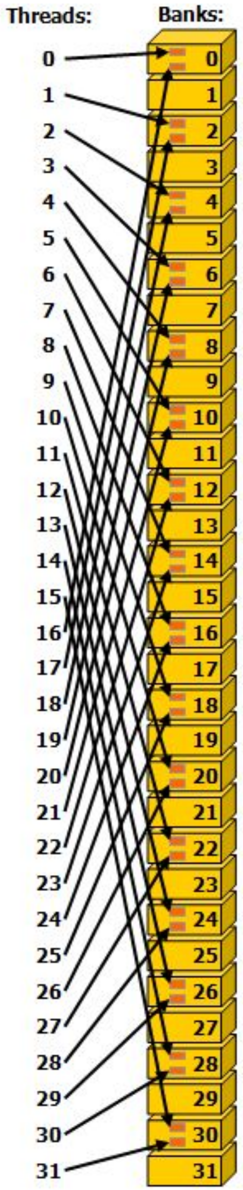
Shared Memory Banks



All are parallel!

Shared Memory Bank Conflicts

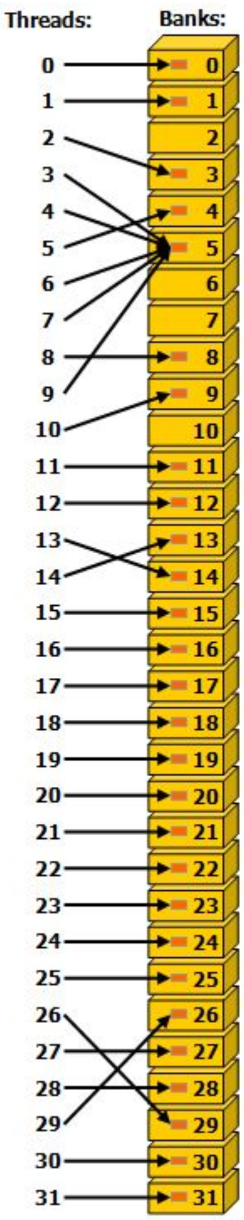
- If shared memory address of any two threads falls in same bank (except when both of them access same address) then a *conflict* occurs
  - Access become ***serial***



2-way bank conflict!

Shared Memory Bank Conflicts

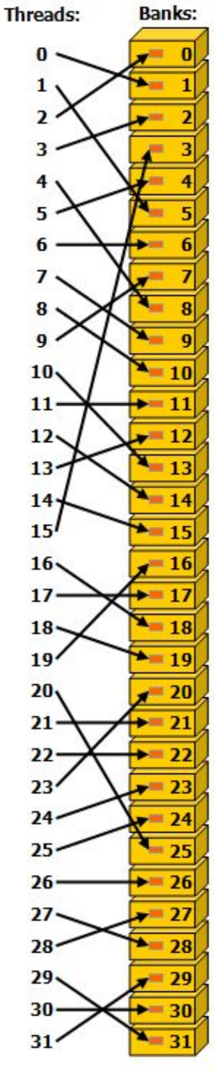
- If shared memory address of any two threads falls in same bank (except when both of them access same address) then a *conflict* occurs
  - Access become ***serial***



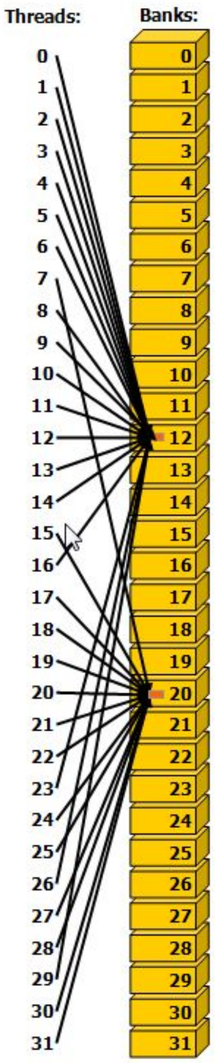
No conflict!

Exercise

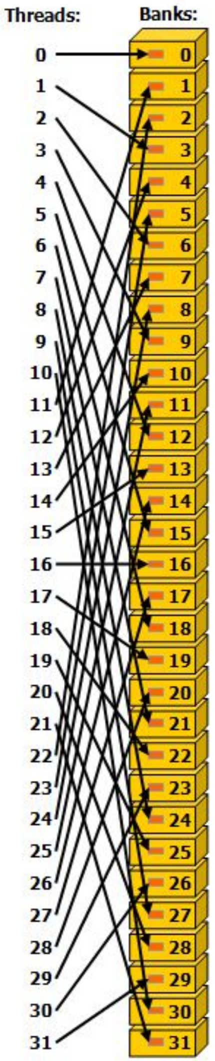
- *Do the patterns have bank conflicts?*



(a)



(b)



(c)

Exercise-1:

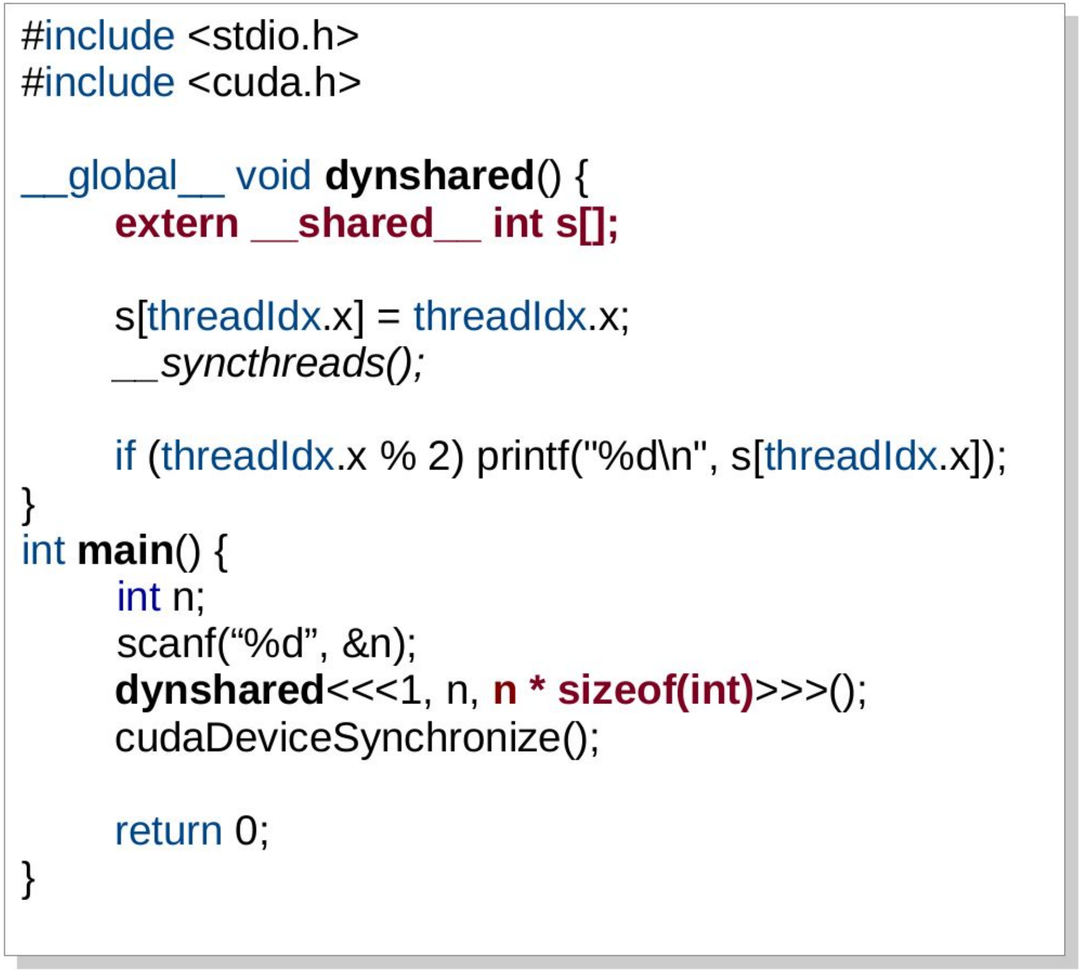
- In the modern GPUs (> 2.x), successive 32-bit words map to successive banks.
  - bank = (address / 4) % 32
- No. of banks = 32.
- *Does the following snippet have bank conflicts?*

```
__shared__ float shared[64];
int S=2;
float data = shared[S*tid];
```

Dynamic Shared Memory

- When the amount of shared memory required is unknown at compile-time, dynamic shared memory can be used.
- This is specified as the third parameter of kernel launch.

Dynamic Shared Memory

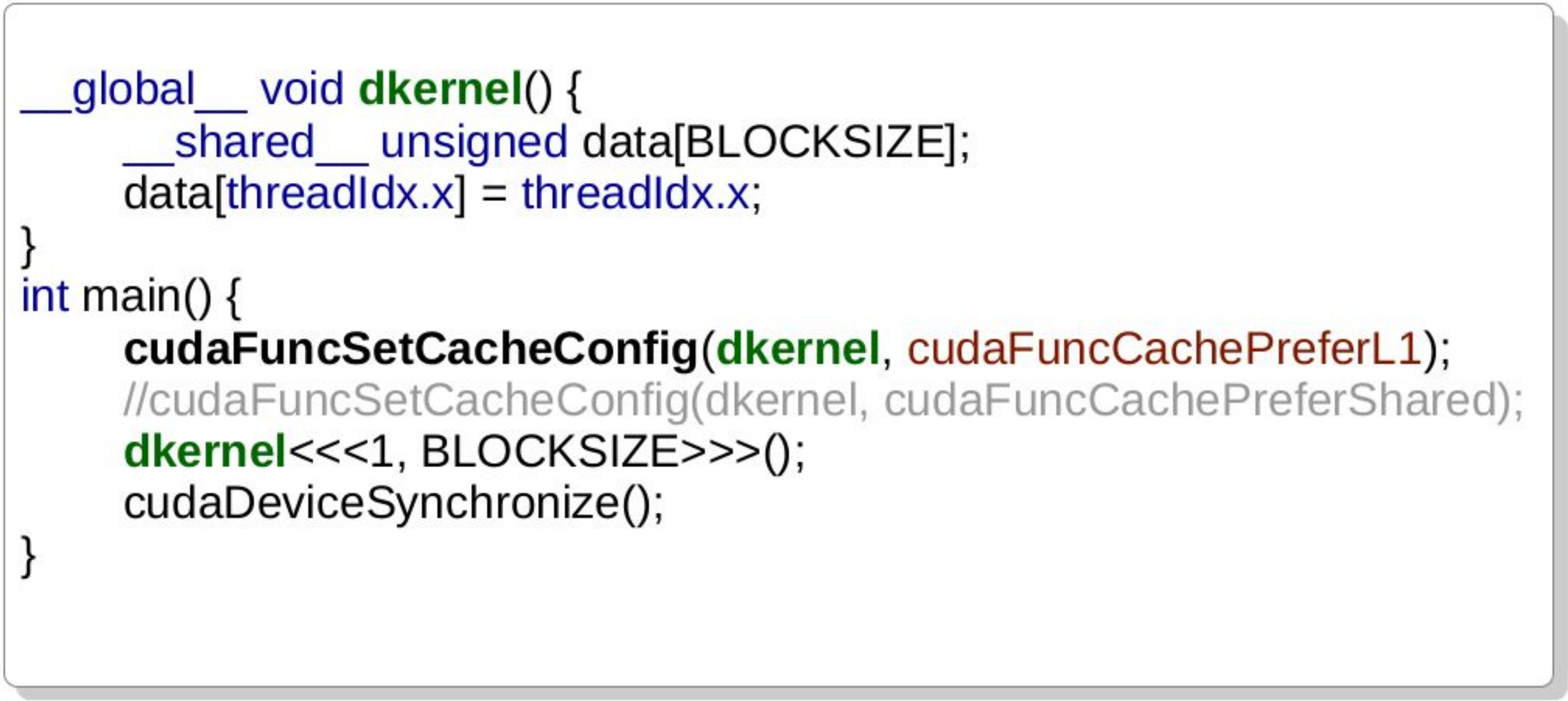


```
#include <stdio.h>
#include <cuda.h>

__global__ void dynshared() {
        extern __shared__ int s[];

        s[threadIdx.x] = threadIdx.x;
        __syncthreads();

        if (threadIdx.x % 2) printf("%d\n", s[threadIdx.x]);
}
int main() {
        int n;
        scanf(“%d”, &n);
        dynshared<<<1, n, n * sizeof(int)>>>();
        cudaDeviceSynchronize();

        return 0;
}
```

Configurable L1 Cache and Shared Memory

- Shared memory and L1 cache can be configured by the programmer.
  - cudaDeviceSetCacheConfig(*kernelname*, *param*);
  - *kernelname* is the name of your kernel
  - param:
    - cudaFuncCachePreferNone: no preference for shared memory or L1 (default)
    - cudaFuncCachePreferShared: prefer larger shared memory and smaller L1 cache
    - cudaFuncCachePreferL1: prefer larger L1 cache and smaller shared memory
    - cudaFuncCachePreferEqual: prefer equal size L1 cache and shared memory

L1 Cache and Shared Memory



```
__global__ void dkernel() {
        __shared__ unsigned data[BLOCKSIZE];
        data[threadIdx.x] = threadIdx.x;
}
int main() {
        cudaFuncSetCacheConfig(dkernel, cudaFuncCachePreferL1);
        //cudaFuncSetCacheConfig(dkernel, cudaFuncCachePreferShared);
        dkernel<<<1, BLOCKSIZE>>>();
        cudaDeviceSynchronize();
}
```

Thank you!