CUDA Programming

Recap

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

#define BLOCKSIZE      1024

__global__ void dkernel() {
      __shared__ unsigned s;

      if (threadIdx.x == 0) s = 0;

      if (threadIdx.x == 1) s += 1;

      if (threadIdx.x == 100) s += 2;

      if (threadIdx.x == 0) printf("s=%d\n", s);
}
int main() {
      int i;
      for (i = 0; i < 10; ++i) {
            dkernel<<<2, BLOCKSIZE>>>();
            cudaDeviceSynchronize();
      }
}
```

```
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=1
s=3
s=3
s=3
```

Classwork

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

#define BLOCKSIZE      1024

__global__ void dkernel() {
      __shared__ unsigned s;

      if (threadIdx.x == 0) s = 0;

      if (threadIdx.x == 1) s += 1;

      if (threadIdx.x == 31) s += 2;

      if (threadIdx.x == 0) printf("s=%d\n", s);
}
int main() {
      int i;
      for (i = 0; i < 10; ++i) {
            dkernel<<<2, BLOCKSIZE>>>();
            cudaDeviceSynchronize();
      }
}
```

Shared Memory

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

#define BLOCKSIZE      1024

__global__ void dkernel() {
      __shared__ unsigned s;

      if (threadIdx.x == 0) s = 0;
       __syncthreads(); // barrier across threads in a block
      if (threadIdx.x == 1) s += 1;
       __syncthreads();
      if (threadIdx.x == 100) s += 2;
       __syncthreads();
      if (threadIdx.x == 0) printf("s=%d\n", s);
}
int main() {
      int i;
      for (i = 0; i < 10; ++i) {
            dkernel<<<2, BLOCKSIZE>>>();
            cudaDeviceSynchronize();
      }
}
```

This one is redundant.

```
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
s=3
```

What is the output of this program?

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

#define BLOCKSIZE      ...

__global__ void dkernel() {
      __shared__ char str[BLOCKSIZE+1];
      str[threadIdx.x] = 'A' + (threadIdx.x + blockIdx.x) % BLOCKSIZE;
      if (threadIdx.x == 0) {
            str[BLOCKSIZE] = '\0';
      }

      if (threadIdx.x == 0) {
            printf("%d: %s\n", blockIdx.x, str);
      }
}
int main() {
      dkernel<<<10, BLOCKSIZE>>>();
      cudaDeviceSynchronize();
}
```

What is the bug in this code?

What is the output of this program?

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

#define BLOCKSIZE      ...

__global__ void dkernel() {
      __shared__ char str[BLOCKSIZE+1];
      str[threadIdx.x] = 'A' + (threadIdx.x + bl...Idx.x) % BLOCKSIZE;
      if (threadIdx.x == 0) {
            str[BLOCKSIZE] = '\0';
      }
      __syncthreads();       // barrier across threads in a block
      if (threadIdx.x == 0) {
            printf("%d: %s\n", blockIdx.x, str);
      }
}
int main() {
      dkernel<<<10, BLOCKSIZE>>>();
      cudaDeviceSynchronize();
}
```

This is redundant if BLOCKSIZE <= 32.

L1 versus Shared

- On CPU:
  - cudaDeviceSetCacheConfig($kernelname$, $param$);
  - $kernelname$ is the name of your kernel.
  - $param$ is {cudaFuncCachePreferNone, L1, Shared}.
  - 3.x onward, one may also configure it as 32KB L1 + 32KB Shared. This is achieved using cudaFuncCachePreferEqual.

L1 versus Shared

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

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;
}
```

Shared Memory with Multiple Arrays

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

__global__ void dynshared(int sz, int n1) {
        extern __shared__ int s[];
        int *s1 = s;
        int *s2 = s + n1;
        if (threadIdx.x < n1) s1[threadIdx.x] = threadIdx.x;
        if (threadIdx.x < (sz - n1)) s2[threadIdx.x] = threadIdx.x * 100;
        __syncthreads();
        if (threadIdx.x < sz && threadIdx.x % 2) printf("%d\n", s1[threadIdx.x]);
}
int main() {
        int sz;
        scanf("%d", &sz);
        dynshared<<<1, 1024, sz * sizeof(int)>>>(sz, sz / 2);
        cudaDeviceSynchronize();

        return 0;
}
```

Bank Conflicts

- Shared memory is organized into 32 banks.
- Accesses to the same bank are sequential.