CUDA Programming

Product of Matrices

Matrix multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 0 & 7 \end{bmatrix} = \begin{bmatrix} 1*5 + 2*0 & 1*6 + 2*7 \\ 3*5 + 4*0 & 3*6 + 4*7 \end{bmatrix} = \begin{bmatrix} 5 & 20 \\ 15 & 46 \end{bmatrix}$$

$$C[\![i]\!][j] = \sum_{k=0}^{n} A[i][k] * B[k][j]$$

Matrix Product on the Device: `product()`

- Parallelized `product()` kernel

```
__global__ void product(int *a, int *b, int *c, int n) {
        int rowIndex, colIndex, width;

        rowIndex = blockIdx.y;
        colIndex = blockIdx.x;

        for (int k=0; k < n; k++)
           c[rowIndex][colIndex] += a[rowIndex][k] * b[k][colIndex];
}
```

Running the kernel with a grid of N * N Blocks of 1 thread each.

Matrix Product on the Device: `product()`

- Parallelized `product()` kernel

```
__global__ void product(int *a, int *b, int *c, int n) {
        int rowIndex, colIndex, width;

        rowIndex = threadIdx.y;
        colIndex = threadIdx.x;

        for (int k=0; k < n; k++)
           c[rowIndex][colIndex] += a[rowIndex][k] * b[k][colIndex];
}
```

`Running the kernel with with a grid of 1 Block of N * N threads.`

Matrix Product on the Device: `product()`

- Parallelized `product()` kernel

```
__global__ void product(int *a, int *b, int *c, int n) {
        int rowIndex, colIndex, width;

        rowIndex = blockIdx.y * blockDim.y + threadIdx.y;
        colIndex = blockIdx.x * blockDim.x + threadIdx.x;

        for (int k=0; k < n; k++)
           c[rowIndex][colIndex] += a[rowIndex][k] * b[k][colIndex];
}
```

Running the kernel with a grid of N * N Blocks each of which is contains N * N threads.

Matrix Product on the Device: `main()`

```
#define N 16
int main(void) {
    int *a  *b  *c                // host copies of a, b, c
    int *d_a, *d_b, *d_c, *w;     // device copies of a, b, c
    int width = 16;
    int nb = N * N * N * N;
    int size = nb * sizeof(int);

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    // Alloc space for host copies of a, b, c and setup input values
    a = (int *)malloc(size); random_ints(a, nb);
    b = (int *)malloc(size); random_ints(b, nb);
    c = (int *)malloc(size);
```

Matrix Addition on the Device: `main()`

```
    // Copy inputs to device
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
   dim3 grid(N, N);
   dim3 block(N, N);

    // Launch add() kernel on GPU with N blocks
    product<<<grid,block>>>(d_a, d_b, d_c, N * N);

    // Copy result back to host
    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

    // Cleanup
    free(a); free(b); free(c);
    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    return 0;
}
```

Matrix Product on the Device: `product()`

- Parallelized `product()` kernel

```
__global__ void product(int *a, int *b, int *c, int width, int n) {
        int rowIndex, colIndex, width;

        rowIndex = (blockIdx.y * blockDim.y + threadIdx.y) * width;
        colIndex = (blockIdx.x * blockDim.x + threadIdx.x) * width;

        for (int i=rowIndex; i < rowIndex + width; i++)
                for (int j=colIndex; j < colIndex + width; j++)
                        for (int k=0; k < n; k++)
                                c[i][j] += a[i][k] * b[k][j];

}
```

Matrix Product on the Device: `main()`

```
#define N 16
int main(void) {
    int *a  *b  *c                      // host copies of a, b, c
    int *d_a, *d_b, *d_c;  // device copies of a, b, c
    int width = 16;
    int nb = N * N * width * N * N * width;
    int size = nb * sizeof(int);

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    // Alloc space for host copies of a, b, c and setup input values
    a = (int *)malloc(size); random_ints(a, nb);
    b = (int *)malloc(size); random_ints(b, nb);
    c = (int *)malloc(size);
```

Matrix Product on the Device: `main()`

```
    // Copy inputs to device
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
   dim3 grid(N, N);
   dim3 block(N, N);

    // Launch add() kernel on GPU with N blocks
    product<<<grid,block>>>(d_a, d_b, d_c, width, N * N * width );

    // Copy result back to host
    cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

    // Cleanup
    free(a); free(b); free(c);
    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    return 0;
}
```