InfoLab

IC621: Distributed and Parallel Computing

Lab 08: Tiled Matrix Multiplication using GPUs

Min-Soo Kim

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

GPU-based Matrix Multiplication

- **Implement two versions of matrix multiplication**
  - **naïve version**
  - **tiled version**
- **Input matrices (prepared by TAs)**
  - size: 20000 x 20000
  - element: float value in the range of 0.0~1.0
- **Result of matrix multiplication MUST be correct**
- **Try to find the best parameters (e.g., grid and block sizes) for improving the performance**

Submission

- **Source code**
- **Output matrix**
- **Short report**
  - elapsed times of naïve and tiled versions
  - GPU parameters (e.g., grid and block sizes)
- **Submit your {code, output, report} by 1:00pm, 12/5(Tue) to TAs (bm010515@dgist.ac.kr, chan150@dgist.ac.kr)**

A simple GPU version with CUDA (1)

```
void MatrixMulOnDevice(float* M, float* N, float* P, int Width)
{
    int size = Width * Width * sizeof(float);
    float* Md, *Nd, *Pd;

    // 1. Allocate and Load M, N to device memory
    cudaMalloc(&Md, size);
    cudaMemcpy(Md, M, size, cudaMemcpyHostToDevice);
    cudaMalloc(&Nd, size);
    cudaMemcpy(Nd, N, size, cudaMemcpyHostToDevice);
    // Allocate P on the device
    cudaMalloc(&Pd, size);

    // Kernel invocation code – to be shown later
    MatrixMulKernel<<< ... >>>();

    cudaMemcpy(P, Pd, size, cudaMemcpyDeviceToHost);
    // Free device matrices
    cudaFree(Md); cudaFree(Nd); cudaFree (Pd);
}
```

A simple GPU version with CUDA (3)

```
__global__ void MatrixMulKernel(float *M, float *N, float *P, int width)
{
    // calculate the row & col index of the element
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    float result = 0;

    // each thread computes one element of the block sub-matrix
    for(int k = 0; k < width; ++k)
        result += M[row * width + k] * N[k * width + col];

    P[row * width + col] = result;
}
```

Kernel code of tiled matrix multiplication

```
__global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width) {
    __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
    __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x;  int by = blockIdx.y;
    int tx = threadIdx.x; int ty = threadIdx.y;

    // Identify the row and column of the Pd element to work on
    int Row = by * TILE_WIDTH + ty;
    int Col = bx * TILE_WIDTH + tx;

    float Pvalue = 0;
    // Loop over the Md and Nd tiles required to compute the Pd element
    for (int m = 0; m < Width / TILE_WIDTH; ++m) {

        // Collaborative loading of Md and Nd tiles into shared memory
        Mds[ty][tx] = Md[Row*Width + (m*TILE_WIDTH + tx)];
        Nds[ty][tx] = Nd[(m*TILE_WIDTH + ty)*Width + Col];
        __syncthreads();

        for (int k = 0; k < TILE_WIDTH; ++k)
            Pvalue += Mds[ty][k] * Nds[k][tx];
        __syncthreads();
    }
    Pd[Row*Width + Col] = Pvalue;
}
```

InfoLab

Thank you!

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology