|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <assert.h> |
|  |  |
|  | #define BLOCK\_SIZE 16 |
|  |  |
|  | \_\_global\_\_ void gpu\_matrix\_mult(int \*a,int \*b, int \*c, int m, int n, int k) |
|  | { |
|  | int row = blockIdx.y \* blockDim.y + threadIdx.y; |
|  | int col = blockIdx.x \* blockDim.x + threadIdx.x; |
|  | int sum = 0; |
|  | if( col < k && row < m) |
|  | { |
|  | for(int i = 0; i < n; i++) |
|  | { |
|  | sum += a[row \* n + i] \* b[i \* k + col]; |
|  | } |
|  | c[row \* k + col] = sum; |
|  | } |
|  | } |
|  |  |
|  |  |
|  | \_\_global\_\_ void gpu\_square\_matrix\_mult(int \*d\_a, int \*d\_b, int \*d\_result, int n) |
|  | { |
|  | \_\_shared\_\_ int tile\_a[BLOCK\_SIZE][BLOCK\_SIZE]; |
|  | \_\_shared\_\_ int tile\_b[BLOCK\_SIZE][BLOCK\_SIZE]; |
|  |  |
|  | int row = blockIdx.y \* BLOCK\_SIZE + threadIdx.y; |
|  | int col = blockIdx.x \* BLOCK\_SIZE + threadIdx.x; |
|  | int tmp = 0; |
|  | int idx; |
|  |  |
|  | for (int sub = 0; sub < gridDim.x; ++sub) |
|  | { |
|  | idx = row \* n + sub \* BLOCK\_SIZE + threadIdx.x; |
|  | if(idx >= n\*n) |
|  | { |
|  | // n may not divisible by BLOCK\_SIZE |
|  | tile\_a[threadIdx.y][threadIdx.x] = 0; |
|  | } |
|  | else |
|  | { |
|  | tile\_a[threadIdx.y][threadIdx.x] = d\_a[idx]; |
|  | } |
|  |  |
|  | idx = (sub \* BLOCK\_SIZE + threadIdx.y) \* n + col; |
|  | if(idx >= n\*n) |
|  | { |
|  | tile\_b[threadIdx.y][threadIdx.x] = 0; |
|  | } |
|  | else |
|  | { |
|  | tile\_b[threadIdx.y][threadIdx.x] = d\_b[idx]; |
|  | } |
|  | \_\_syncthreads(); |
|  |  |
|  | for (int k = 0; k < BLOCK\_SIZE; ++k) |
|  | { |
|  | tmp += tile\_a[threadIdx.y][k] \* tile\_b[k][threadIdx.x]; |
|  | } |
|  | \_\_syncthreads(); |
|  | } |
|  | if(row < n && col < n) |
|  | { |
|  | d\_result[row \* n + col] = tmp; |
|  | } |
|  | } |
|  |  |
|  | \_\_global\_\_ void gpu\_matrix\_transpose(int\* mat\_in, int\* mat\_out, unsigned int rows, unsigned int cols) |
|  | { |
|  | unsigned int idx = blockIdx.x \* blockDim.x + threadIdx.x; |
|  | unsigned int idy = blockIdx.y \* blockDim.y + threadIdx.y; |
|  |  |
|  | if (idx < cols && idy < rows) |
|  | { |
|  | unsigned int pos = idy \* cols + idx; |
|  | unsigned int trans\_pos = idx \* rows + idy; |
|  | mat\_out[trans\_pos] = mat\_in[pos]; |
|  | } |
|  | } |
|  | void cpu\_matrix\_mult(int \*h\_a, int \*h\_b, int \*h\_result, int m, int n, int k) { |
|  | for (int i = 0; i < m; ++i) |
|  | { |
|  | for (int j = 0; j < k; ++j) |
|  | { |
|  | int tmp = 0.0; |
|  | for (int h = 0; h < n; ++h) |
|  | { |
|  | tmp += h\_a[i \* n + h] \* h\_b[h \* k + j]; |
|  | } |
|  | h\_result[i \* k + j] = tmp; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char const \*argv[]) |
|  | { |
|  | int m, n, k; |
|  | /\* Fixed seed for illustration \*/ |
|  | srand(3333); |
|  | printf("please type in m n and k\n"); |
|  | scanf("%d %d %d", &m, &n, &k); |
|  |  |
|  | // allocate memory in host RAM, h\_cc is used to store CPU result |
|  | int \*h\_a, \*h\_b, \*h\_c, \*h\_cc; |
|  | cudaMallocHost((void \*\*) &h\_a, sizeof(int)\*m\*n); |
|  | cudaMallocHost((void \*\*) &h\_b, sizeof(int)\*n\*k); |
|  | cudaMallocHost((void \*\*) &h\_c, sizeof(int)\*m\*k); |
|  | cudaMallocHost((void \*\*) &h\_cc, sizeof(int)\*m\*k); |
|  |  |
|  | // random initialize matrix A |
|  | for (int i = 0; i < m; ++i) { |
|  | for (int j = 0; j < n; ++j) { |
|  | h\_a[i \* n + j] = rand() % 1024; |
|  | } |
|  | } |
|  |  |
|  | // random initialize matrix B |
|  | for (int i = 0; i < n; ++i) { |
|  | for (int j = 0; j < k; ++j) { |
|  | h\_b[i \* k + j] = rand() % 1024; |
|  | } |
|  | } |
|  |  |
|  | float gpu\_elapsed\_time\_ms, cpu\_elapsed\_time\_ms; |
|  |  |
|  | // some events to count the execution time |
|  | cudaEvent\_t start, stop; |
|  | cudaEventCreate(&start); |
|  | cudaEventCreate(&stop); |
|  |  |
|  | // start to count execution time of GPU version |
|  | cudaEventRecord(start, 0); |
|  | // Allocate memory space on the device |
|  | int \*d\_a, \*d\_b, \*d\_c; |
|  | cudaMalloc((void \*\*) &d\_a, sizeof(int)\*m\*n); |
|  | cudaMalloc((void \*\*) &d\_b, sizeof(int)\*n\*k); |
|  | cudaMalloc((void \*\*) &d\_c, sizeof(int)\*m\*k); |
|  |  |
|  | // copy matrix A and B from host to device memory |
|  | cudaMemcpy(d\_a, h\_a, sizeof(int)\*m\*n, cudaMemcpyHostToDevice); |
|  | cudaMemcpy(d\_b, h\_b, sizeof(int)\*n\*k, cudaMemcpyHostToDevice); |
|  |  |
|  | unsigned int grid\_rows = (m + BLOCK\_SIZE - 1) / BLOCK\_SIZE; |
|  | unsigned int grid\_cols = (k + BLOCK\_SIZE - 1) / BLOCK\_SIZE; |
|  | dim3 dimGrid(grid\_cols, grid\_rows); |
|  | dim3 dimBlock(BLOCK\_SIZE, BLOCK\_SIZE); |
|  |  |
|  | // Launch kernel |
|  | if(m == n && n == k) |
|  | { |
|  | gpu\_square\_matrix\_mult<<<dimGrid, dimBlock>>>(d\_a, d\_b, d\_c, n); |
|  | } |
|  | else |
|  | { |
|  | gpu\_matrix\_mult<<<dimGrid, dimBlock>>>(d\_a, d\_b, d\_c, m, n, k); |
|  | } |
|  | // Transefr results from device to host |
|  | cudaMemcpy(h\_c, d\_c, sizeof(int)\*m\*k, cudaMemcpyDeviceToHost); |
|  | cudaThreadSynchronize(); |
|  | // time counting terminate |
|  | cudaEventRecord(stop, 0); |
|  | cudaEventSynchronize(stop); |
|  |  |
|  | // compute time elapse on GPU computing |
|  | cudaEventElapsedTime(&gpu\_elapsed\_time\_ms, start, stop); |
|  | printf("Time elapsed on matrix multiplication of %dx%d . %dx%d on GPU: %f ms.\n\n", m, n, n, k, gpu\_elapsed\_time\_ms); |
|  |  |
|  | // start the CPU version |
|  | cudaEventRecord(start, 0); |
|  |  |
|  | cpu\_matrix\_mult(h\_a, h\_b, h\_cc, m, n, k); |
|  |  |
|  | cudaEventRecord(stop, 0); |
|  | cudaEventSynchronize(stop); |
|  | cudaEventElapsedTime(&cpu\_elapsed\_time\_ms, start, stop); |
|  | printf("Time elapsed on matrix multiplication of %dx%d . %dx%d on CPU: %f ms.\n\n", m, n, n, k, cpu\_elapsed\_time\_ms); |
|  |  |
|  | // validate results computed by GPU |
|  | int all\_ok = 1; |
|  | for (int i = 0; i < m; ++i) |
|  | { |
|  | for (int j = 0; j < k; ++j) |
|  | { |
|  | //printf("[%d][%d]:%d == [%d][%d]:%d, ", i, j, h\_cc[i\*k + j], i, j, h\_c[i\*k + j]); |
|  | if(h\_cc[i\*k + j] != h\_c[i\*k + j]) |
|  | { |
|  | all\_ok = 0; |
|  | } |
|  | } |
|  | //printf("\n"); |
|  | } |
|  |  |
|  | // roughly compute speedup |
|  | if(all\_ok) |
|  | { |
|  | printf("all results are correct!!!, speedup = %f\n", cpu\_elapsed\_time\_ms / gpu\_elapsed\_time\_ms); |
|  | } |
|  | else |
|  | { |
|  | printf("incorrect results\n"); |
|  | } |
|  |  |
|  | // free memory |
|  | cudaFree(d\_a); |
|  | cudaFree(d\_b); |
|  | cudaFree(d\_c); |
|  | cudaFreeHost(h\_a); |
|  | cudaFreeHost(h\_b); |
|  | cudaFreeHost(h\_c); |
|  | cudaFreeHost(h\_cc); |
|  | return 0; |
|  | } |