

```

__global__ void multiply(float *A, float *B, float *C){

    // thread position in block
    int row = threadIdx.y;
    int col = threadIdx.x;

    // absolute position
    int absRow = blockIdx.y*blockDim.y + threadIdx.y;
    int absCol = blockIdx.x*blockDim.x + threadIdx.x;
    int index = absRow*NC + absCol; // location in contiguous 1-d

    int j;
    int sum = 0;
    for(j=0;j<NC/BLOCKSIZE;j++){
        __shared__ float Apatch[BLOCKSIZE][BLOCKSIZE];
        __shared__ float Bpatch[BLOCKSIZE][BLOCKSIZE];

        // fetch the corresponding rows and cols of A,B
        // each thread gets one element
        Apatch[row][col] = A[absRow*NC+j*BLOCKSIZE+col];
        Bpatch[row][col] = B[absCol+j*BLOCKSIZE*NC+row*NC];
        __syncthreads();

        int i;
        for(i=0; i<BLOCKSIZE; i++) sum += Apatch[row][i]*Bpatch[i][col];
        __syncthreads();
    }

    C[index] = sum;

}

```

Results

Table 1: Time taken. N - matrix size, n - threads/processes

N	serial	OMP n= 24	MPI n=24	CUDA(16)	CUDA(32)
100	0.008	0.007	0.005	0.160	0.140
200	0.065	0.011	0.023	0.170	0.140
400	0.610	0.081	0.092	0.170	0.150
800	4.080	0.486	0.589	0.170	0.170
1000	5.574	0.594	0.718	0.200	0.200
2000	45.544	4.065	4.581	0.390	0.350
5000	703.575	62.475	57.713	1.970	1.770
8000	2878.791	234.618	233.627	13.420	12.060
10000	6591.409	499.030	557.603	26.030	23.440

Hardware used The machine on which the above results were produced has 24 cores, 15MB L3 cache and it is running Debian OS.

Code

```
#define N 10016
#define NR N
#define NC N
#define BLOCKSIZE 32

#include<stdio.h>
#include<stdlib.h>
#include<time.h>

void printMat(float A[NR][NC]);
void initMat(float A[NR][NC], float B[NR][NC]);
__global__ void multiply(float *A, float *B, float *C);

int main(){
    static float A[NR][NC];

    static float B[NR][NC];

    static float C[NR][NC] = {{0}}; /* initialize to 0 */

    clock_t start_time, end_time;
    double elapsed;

    float *dev_A, *dev_B, *dev_C;
    int size = NR*NC*sizeof(float);

    start_time = clock();

    cudaMalloc((void **)&dev_A, size);
    cudaMalloc((void **)&dev_B, size);
    cudaMalloc((void **)&dev_C, size);

    initMat(A,B);

    cudaMemcpy(dev_A,&A,size,cudaMemcpyHostToDevice);
    cudaMemcpy(dev_B,&B,size,cudaMemcpyHostToDevice);

    dim3 dimGrid(N/BLOCKSIZE,N/BLOCKSIZE);
    dim3 dimBlock(BLOCKSIZE,BLOCKSIZE);

    multiply<<<dimGrid,dimBlock>>>(dev_A,dev_B,dev_C);

    cudaError_t err = cudaGetLastError();
    if (err != cudaSuccess)
        printf("Error:_%s\n", cudaGetErrorString(err));

    cudaMemcpy(&C,dev_C,size,cudaMemcpyDeviceToHost);

    cudaFree(dev_A);
    cudaFree(dev_B);
    cudaFree(dev_C);
```

```

    end_time = clock();
    elapsed = ( (double) (end_time-start_time))/ CLOCKS_PER_SEC;

    //printMat(C);

    printf("_\n_Time_taken_is_%f_\n",elapsed);

    return 0;
}

void printMat(float A[NR][NC]){

    int i,j;

    for( i=0; i<NR; i++){
        printf("ROW_%d:",i+1);
        for( j=0; j<NC; j++){
            printf("%.3f\t",A[i][j]);
        }
        printf("\n");
    }

}

void initMat(float A[NR][NC],float B[NR][NC]){

    int i,j;

    for( i=0; i < NR; i++){
        for( j=0; j<NC; j++){
            A[i][j] = i+j;
            B[i][j] = i*j;
        }
    }

}

__global__ void multiply(float *A, float *B, float *C){

    // thread position in block
    int row = threadIdx.y;
    int col = threadIdx.x;

    // absolute position
    int absRow = blockIdx.y*blockDim.y + threadIdx.y;
    int absCol = blockIdx.x*blockDim.x + threadIdx.x;
    int index = absRow*NC + absCol; // location in contiguous 1-d

    int j;
    int sum = 0;
    for(j=0;j<NC/BLOCKSIZE;j++){
        __shared__ float Apatch[BLOCKSIZE][BLOCKSIZE];
        __shared__ float Bpatch[BLOCKSIZE][BLOCKSIZE];

```

```
        // fetch the corresponding rows and cols of A,B
        // each thread gets one element
        Apatch[row][col] = A[absRow*NC+j*BLOCKSIZE+col];
        Bpatch[row][col] = B[absCol+j*BLOCKSIZE*NC+row*NC];
        __syncthreads();

        int i;
        for(i=0; i<BLOCKSIZE; i++) sum += Apatch[row][i]*Bpatch[i][col];
        __syncthreads();
    }

    C[index] = sum;
}
```