Table 1: normal vs. cache-friendly

N	normal	cache-friendly
100	0.008075	0.008205
200	0.063293	0.064621
400	0.594145	0.610361
800	5.386019	4.080133
1000	9.487167	5.573937
2000	87.209641	45.544228
5000	1908.666260	703.574890
8000	8362.995117	2878.790771
10000	17790.554688	6591.408691

Results

Table 2: Time taken. N - matrix size, n - threads/procesess

N	serial	OMP n=2	OMP n=4	OMP n=8	OMP n= 24	MPI n=2	MPI n=4	MPI n=8	MPI n=24
100	0.008	0.009	0.003	0.002	0.007	0.005	0.005	0.003	0.005
200	0.065	0.007	0.042	0.022	0.011	0.059	0.033	0.018	0.023
400	0.610	0.330	0.186	0.106	0.081	0.346	0.251	0.183	0.092
800	4.080	2.195	1.173	0.574	0.486	2.328	1.368	0.696	0.589
1000	5.574	2.842	2.338	0.804	0.594	2.899	2.536	1.166	0.718
2000	45.544	22.061	11.367	5.903	4.065	22.534	11.836	10.047	4.581
5000	703.575	342.611	175.707	90.065	62.475	346.949	179.308	91.602	57.713
8000	2878.791	1400.648	697.816	363.322	234.618	1434.891	715.345	636.908	233.627
10000	6591.409	2737.575	1798.366	740.367	499.030	2812.980	1680.102	877.236	557.603

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;

}
```