CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

2201

Matrix Copy

행렬 복사

내용 contents

- **matrix copy – theoretical limit**
  - CPU version
  - memcpy version
  - CUDA naïve version – global memory
  - CUDA shared memory version
  - CUDA memcpy2D

  - naïve : (전문적) 지식이 없는

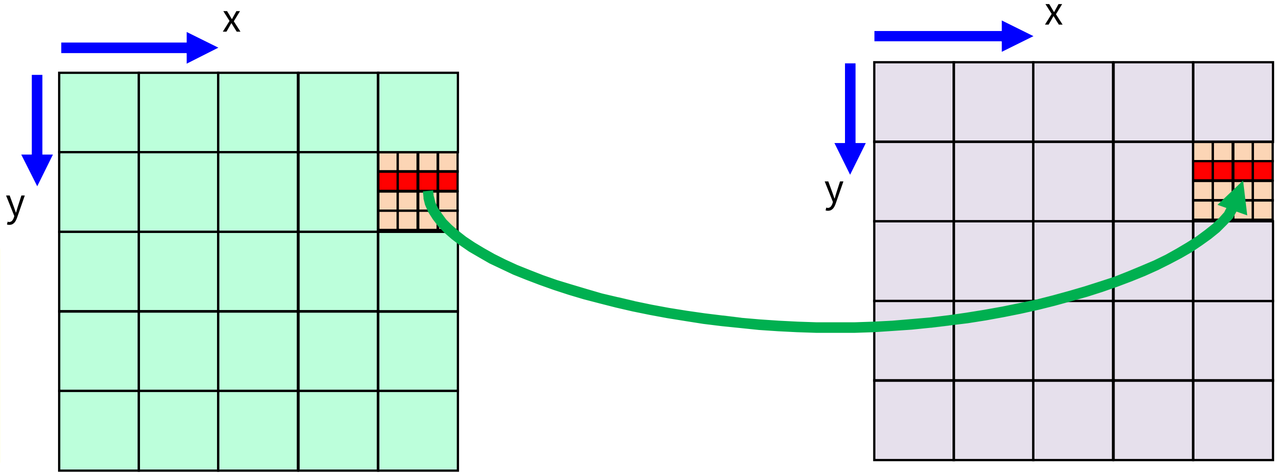
Matrix Copy

- **simply copy a matrix to another**
  - C[i,j] = A[i,j]
  - **pitched matrices** for the best performance
  - for simplicity, we assume **square matrices**

- **다른 matrix 연산의 입장에서는?**
  - theoretical limit 이론적 한계 for matrix operations
  - the best score 최고 기록 for any matrix operations
  - 이 기록에 접근할 수록 최적화가 잘 되었음
  - 이 기록을 추월했다면, 뭔가 잘못 되었음. . .

Matrix Copy – CPU version

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix}$$



2D matrix
in mathematics

2D matrix
in the **main memory**

another 2D matrix
in the **main memory**

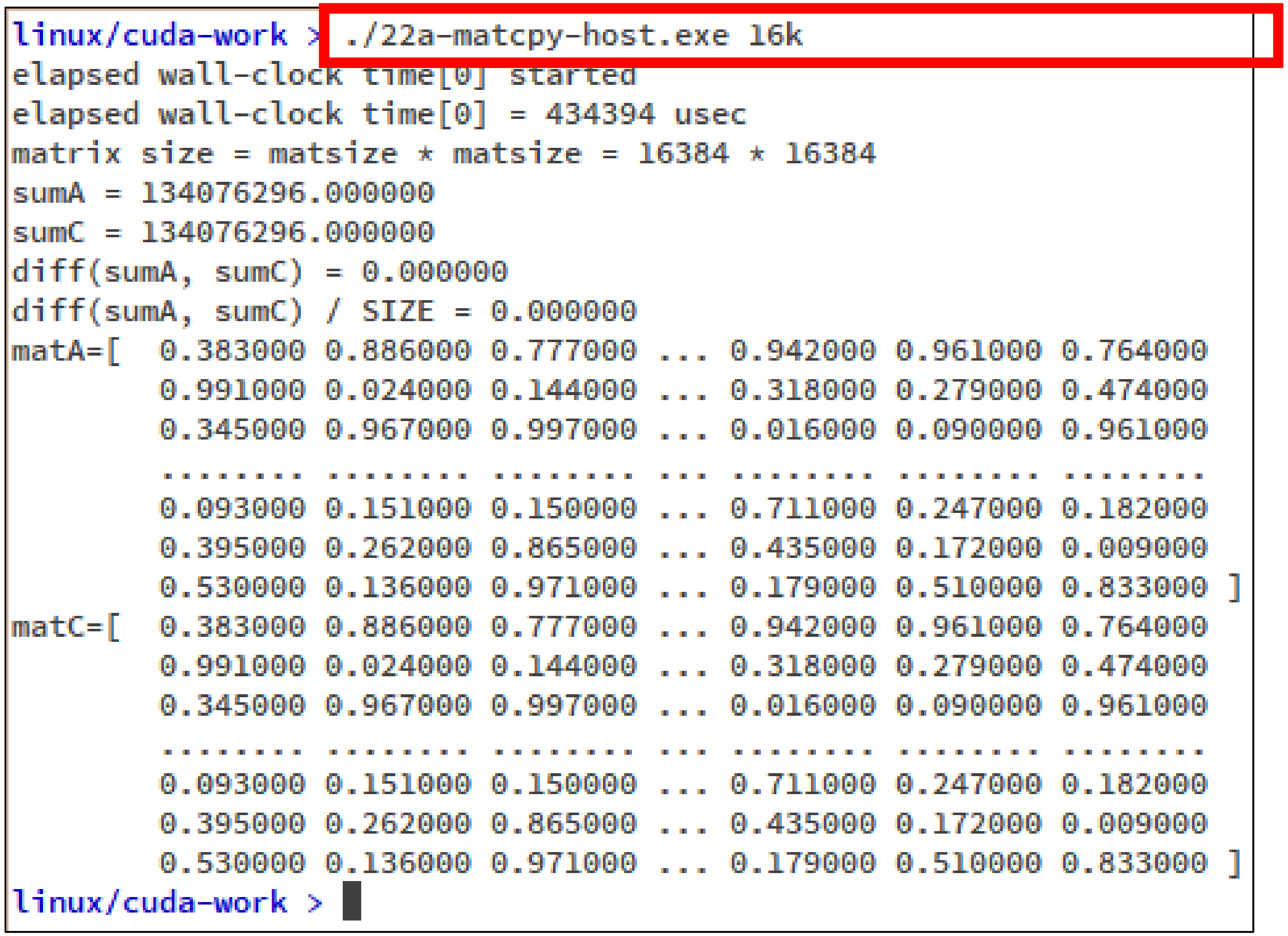
© Illustration by biztripcru@gmail.com

matcpy-host.cpp

```
// input parameters
unsigned matsize = 4000; // num rows and also num cols

int main(const int argc, const char* argv[]) {
   . . .
   float* matA = new float[matsize * matsize];
   float* matC = new float[matsize * matsize];
   . . .
   // kernel processing
   for (register int y = 0; y < matsize; ++y) {
      for (register int x = 0; x < matsize; ++x) {
         register unsigned idx = y * matsize + x;
         matC[idx] = matA[idx];
      }
   }
   . . .
}
```

matcpy-host.cpp 실행 결과

- **434,394 usec for simple 16k x 16k matrix copy** **(Intel Core i5-3570)**



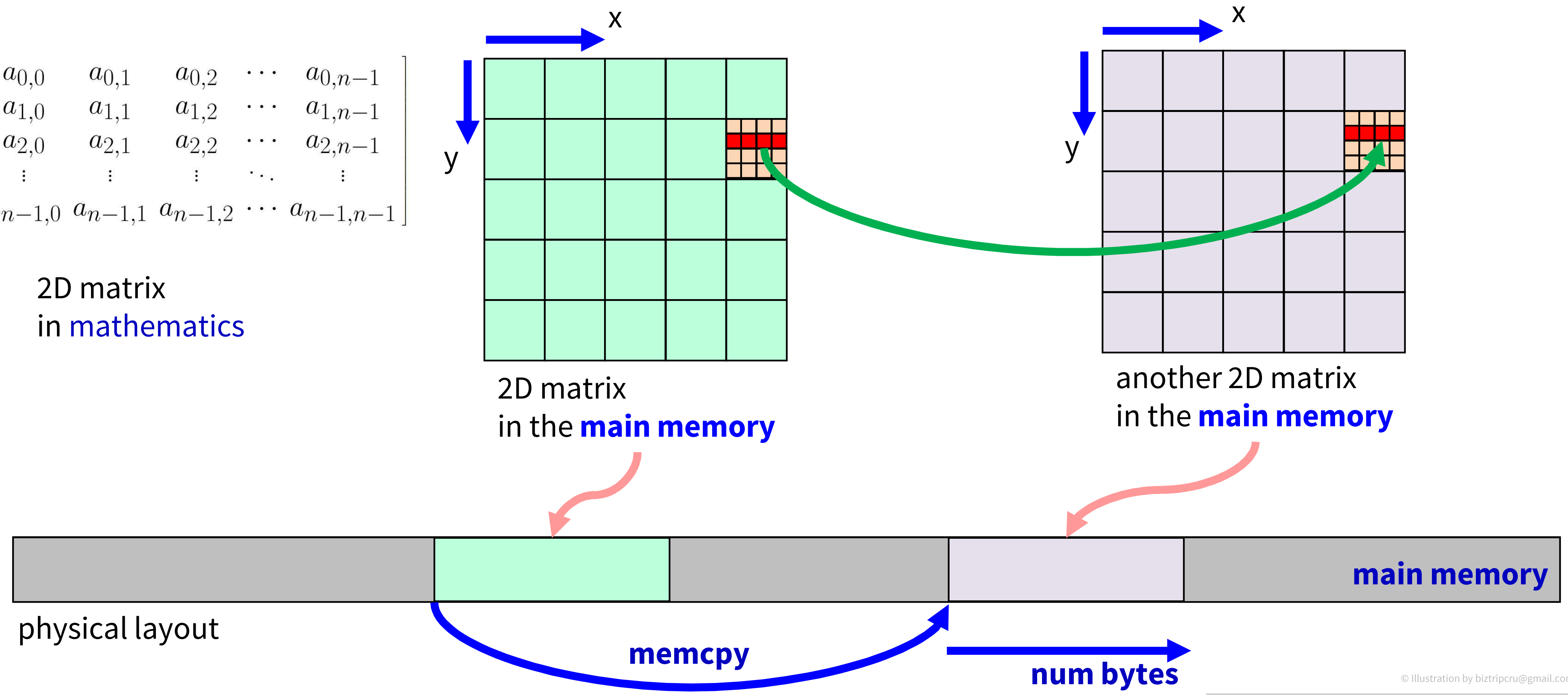
```
linux/cuda-work > ./22a-matcpy-host.exe 16k
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 434394 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076296.000000
diff(sumA, sumC) = 0.000000
diff(sumA, sumC) / SIZE = 0.000000
matA=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
matC=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
linux/cuda-work >
```

CPU version	434,394 usec

Matrix Copy – memcpy version

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

2D matrix
in mathematics



x

y

2D matrix
in the **main memory**

x

y

another 2D matrix
in the **main memory**

main memory

physical layout

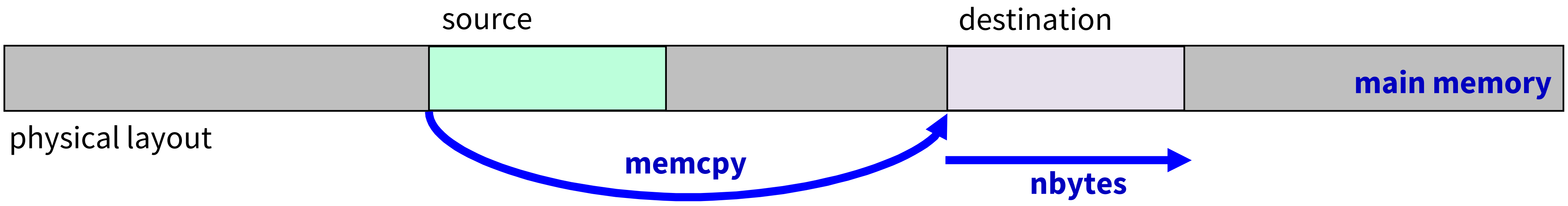
memcpy

num bytes

memcpy( )

void * **memcpy**(void * *destination*, const void * *source*, size_t *num*);

- main memory 상에서, *source* → destination 으로 *num* byte 를 copy 복사
- 겹치는 부분이 있으면, 복사 실패 가능 (memmove( ) 참고)



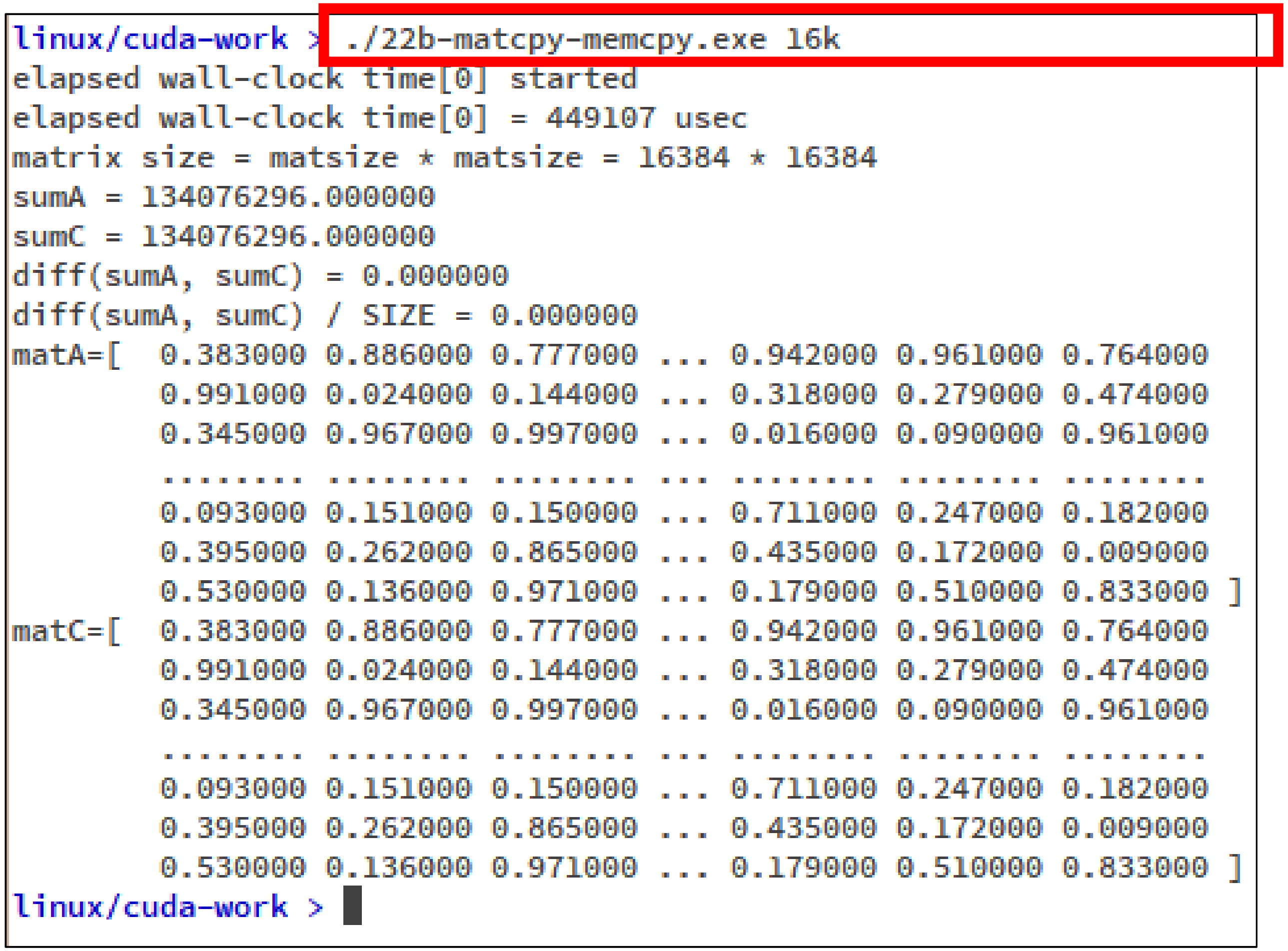
© Illustration by biztripcru@gmail.com

matcpy-memcpy.cpp

```
// input parameters
unsigned matsize = 4000; // num rows and also num cols

int main(const int argc, const char* argv[]) {
    . . .
    // host-side data
    float* matA = new float[matsize * matsize];
    float* matC = new float[matsize * matsize];
    . . .
    setNormalizedRandomData( matA, matsize * matsize );
    . . .
    // kernel processing
    ELAPSED_TIME_BEGIN(0);
    memcpy( matC, matA, matsize * matsize * sizeof(float) );
    ELAPSED_TIME_END(0);
    . . .
}
```

matcpy-memcpy.cpp 실행 결과

- 449,107 usec for simple 16k x 16k matrix copy (Intel Core i5-3570)



```
linux/cuda-work > ./22b-matcpy-memcpy.exe 16k
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 449107 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076296.000000
diff(sumA, sumC) = 0.000000
diff(sumA, sumC) / SIZE = 0.000000
matA=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
matC=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
linux/cuda-work >
```

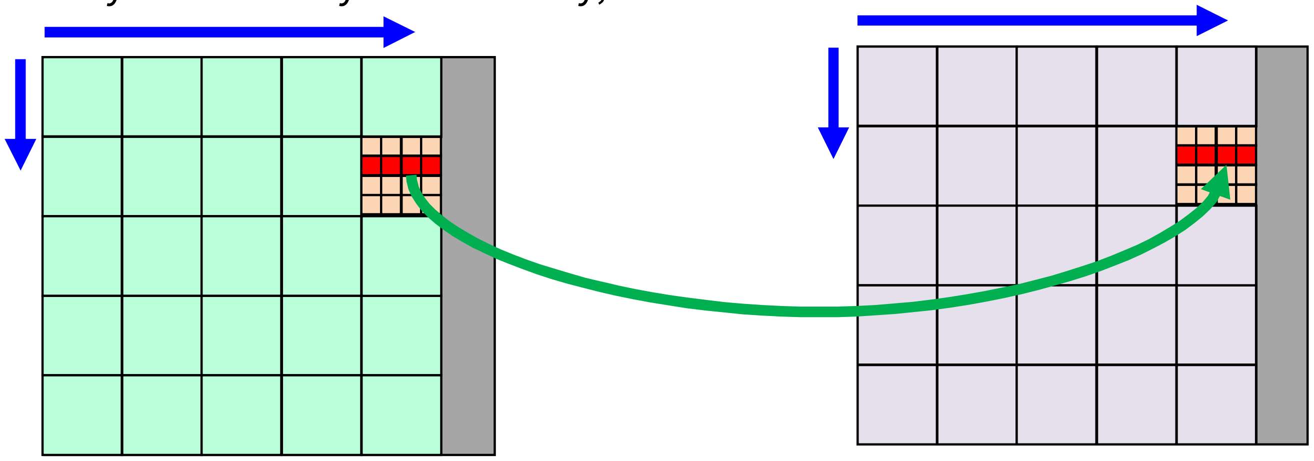
CPU version	434,394 usec
memcpy version	449,107 usec

Matrix Copy – CUDA version

global index
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;

global index
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix}$$



may be pitched !

may be pitched !

2D matrix
in mathematics

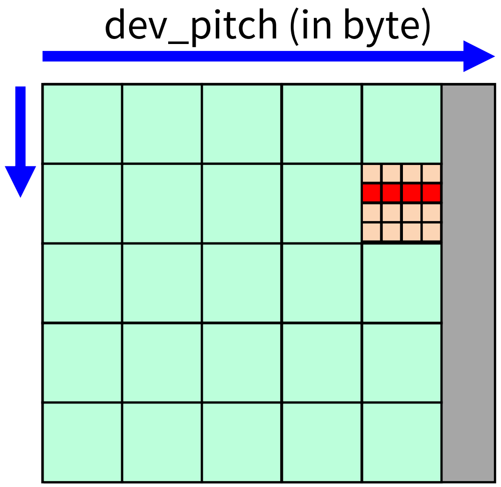
2D matrix
in the **global memory**

another 2D matrix
in the **global memory**

© Illustration by biztripcru@gmail.com

index 계산

- **pitched matrix**
  - dev_pitch : cudaMallocPitch( ), cudaMalloc3D( ) → byte 단위
  - data : float → sizeof(float) = 4 byte
- **A[y][x] 의 위치 계산: byte 기준**
  - offset = y * dev_pitch + x * sizeof(float); // in byte
  - *((float*)((char*)A + offset) = 0.0f;
- **A[y][x] 의 위치 계산: index 기준**
  - assert(dev_pitch % sizeof(float) == 0);
  - pitch_in_elem = dev_pitch / sizeof(float);
  - idx = y * pitch_in_elem + x;
  - A[idx] = 0.0f;



pitched 2D matrix

matcpy-dev.cu

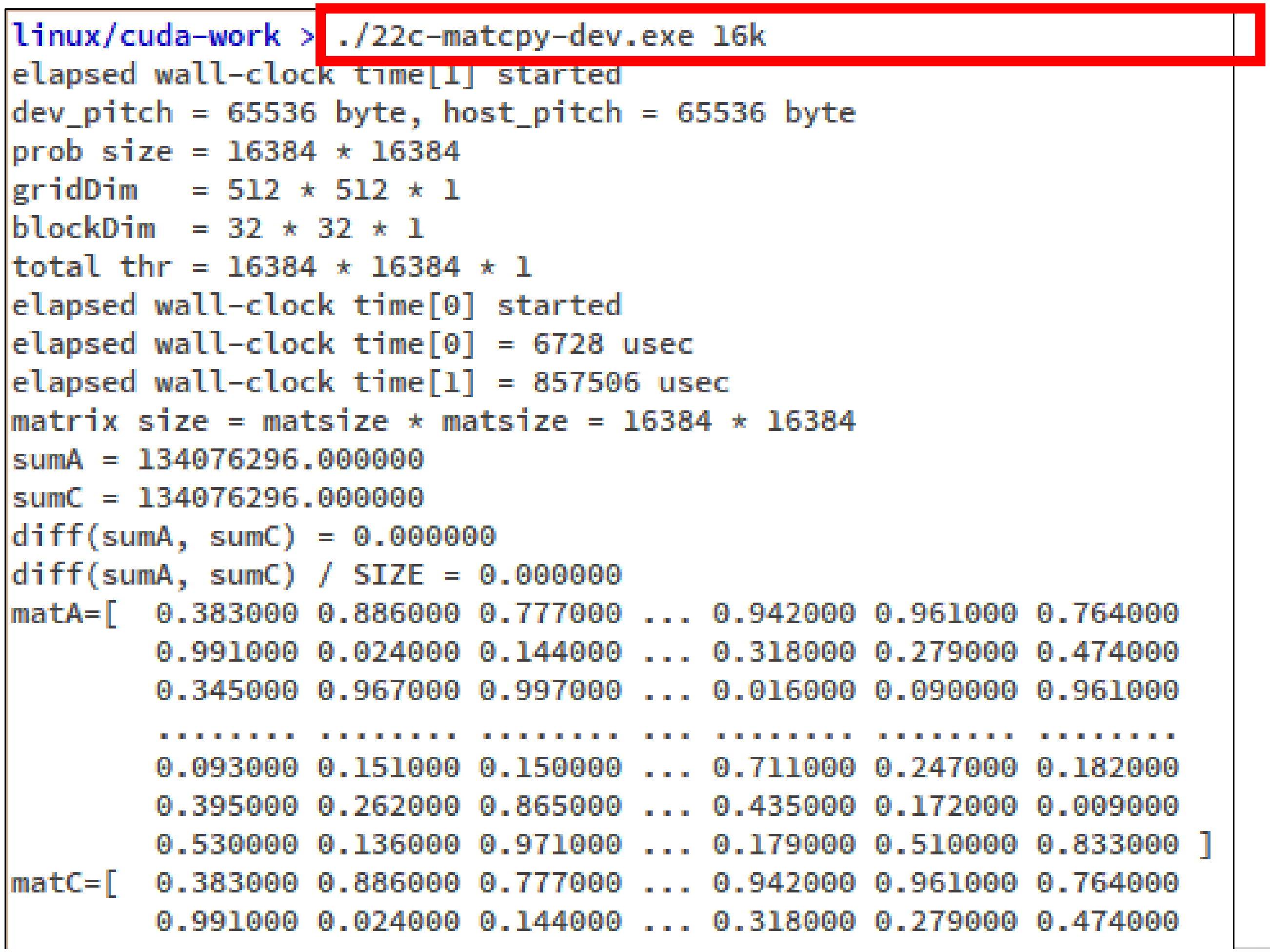
```
// input parameters
unsigned matsize = 4000; // num rows and also num cols

__global__ void kernelMatCpy( float* C, const float* A, int matsize, size_t pitch_in_elem ) {
   register unsigned gy = blockIdx.y * blockDim.y + threadIdx.y; // CUDA-provided index
   if (gy < matsize) {
      register unsigned gx = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
      if (gx < matsize) {
         register unsigned idx = gy * pitch_in_elem + gx; // in element
         C[idx] = A[idx];
      }
   }
}
```

matcpy-dev.cu 계속

```
int main(const int argc, const char* argv[]) {
    . . .
    // allocate device memory
    size_t dev_pitch = 0;
    cudaMallocPitch( (void**)&dev_matA, &dev_pitch, matsize * sizeof(float), matsize );
    cudaMallocPitch( (void**)&dev_matC, &dev_pitch, matsize * sizeof(float), matsize );
    // copy to device from host
    cudaMemcpy2D( . . . );
    . . .
    // CUDA kernel call
    dim3 dimBlock(32, 32, 1);
    dim3 dimGrid(div_up(matsize, dimBlock.x), div_up(matsize, dimBlock.y), 1);
    assert(dev_pitch % sizeof(float) == 0);
    register unsigned pitch_in_elem = dev_pitch / sizeof(float);
    kernelMatCpy <<< dimGrid, dimBlock >>> ( dev_matC, dev_matA, matsize, pitch_in_elem );
    cudaDeviceSynchronize();
    . . .
}
```

matcpy-dev.cu 실행 결과

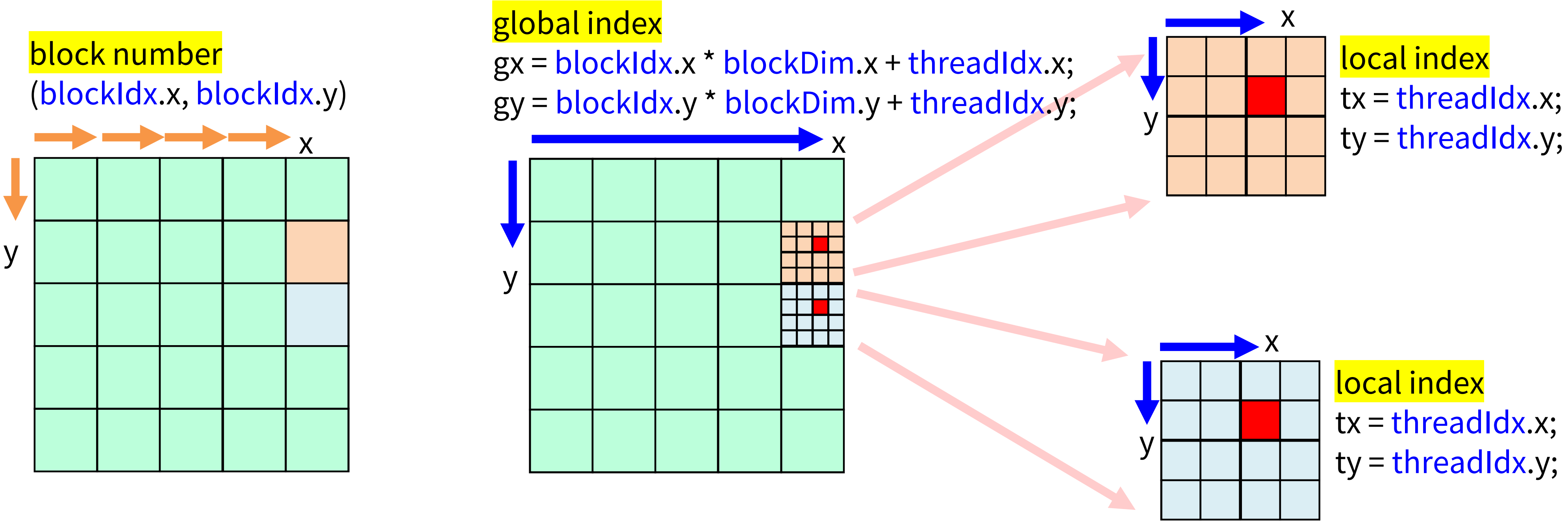
- **6,728 usec for simple 16k x 16k matrix copy** **(GeForce RTX 2070)**



```
linux/cuda-work > ./22c-matcpy-dev.exe 16k
elapsed wall-clock time[1] started
dev_pitch = 65536 byte, host_pitch = 65536 byte
prob size = 16384 * 16384
gridDim   = 512 * 512 * 1
blockDim  = 32 * 32 * 1
total thr = 16384 * 16384 * 1
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 6728 usec
elapsed wall-clock time[1] = 857506 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076296.000000
diff(sumA, sumC) = 0.000000
diff(sumA, sumC) / SIZE = 0.000000
matA=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
matC=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
```

CPU version	434,394 usec
memcpy version	449,107 usec
CUDA naïve copy	6,728 usec

2D Tiled Layout for Matrix Operations



block number
(blockIdx.x, blockIdx.y)

global index
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;

local index
tx = threadIdx.x;
ty = threadIdx.y;

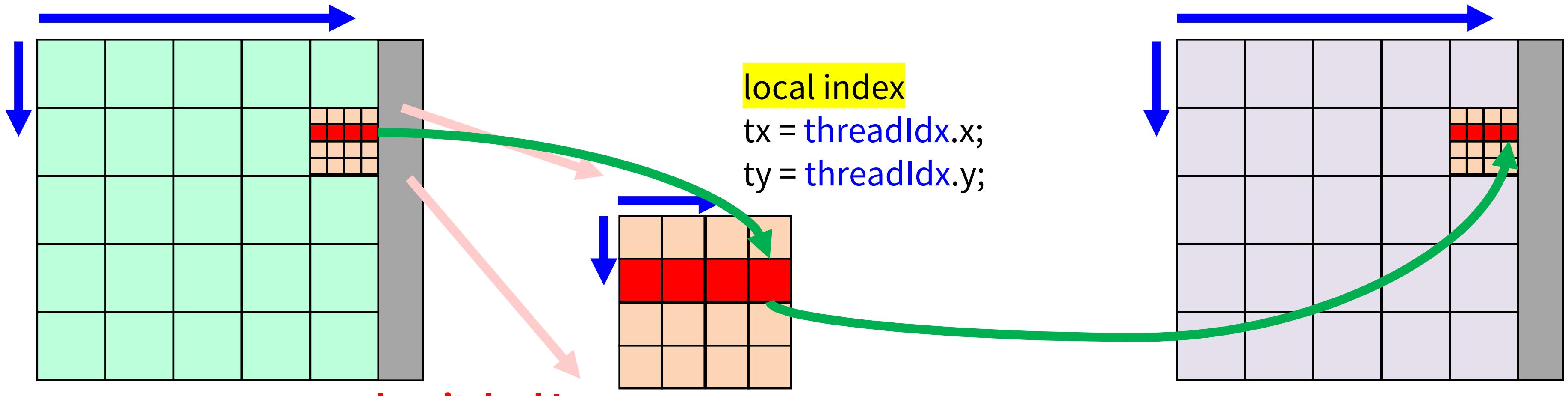
local index
tx = threadIdx.x;
ty = threadIdx.y;

grid = a 2D layout of blocks

block = a 2D layout of threads

thread block in **shared memory**

Matrix Copy – Tiled Approach



global index
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;

local index
tx = threadIdx.x;
ty = threadIdx.y;

global index
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;

may be pitched !

may be pitched !

2D matrix
in the **global memory**

2D sub-matrix
in the **shared memory**

another 2D matrix
in the **global memory**

© Illustration by biztripcru@gmail.com

Tile Size

- **for simplicity, we assume square matrices**
- **tile : 정사각형 모양을 선호**

```
__shared__ float s_mat[32][32];
```

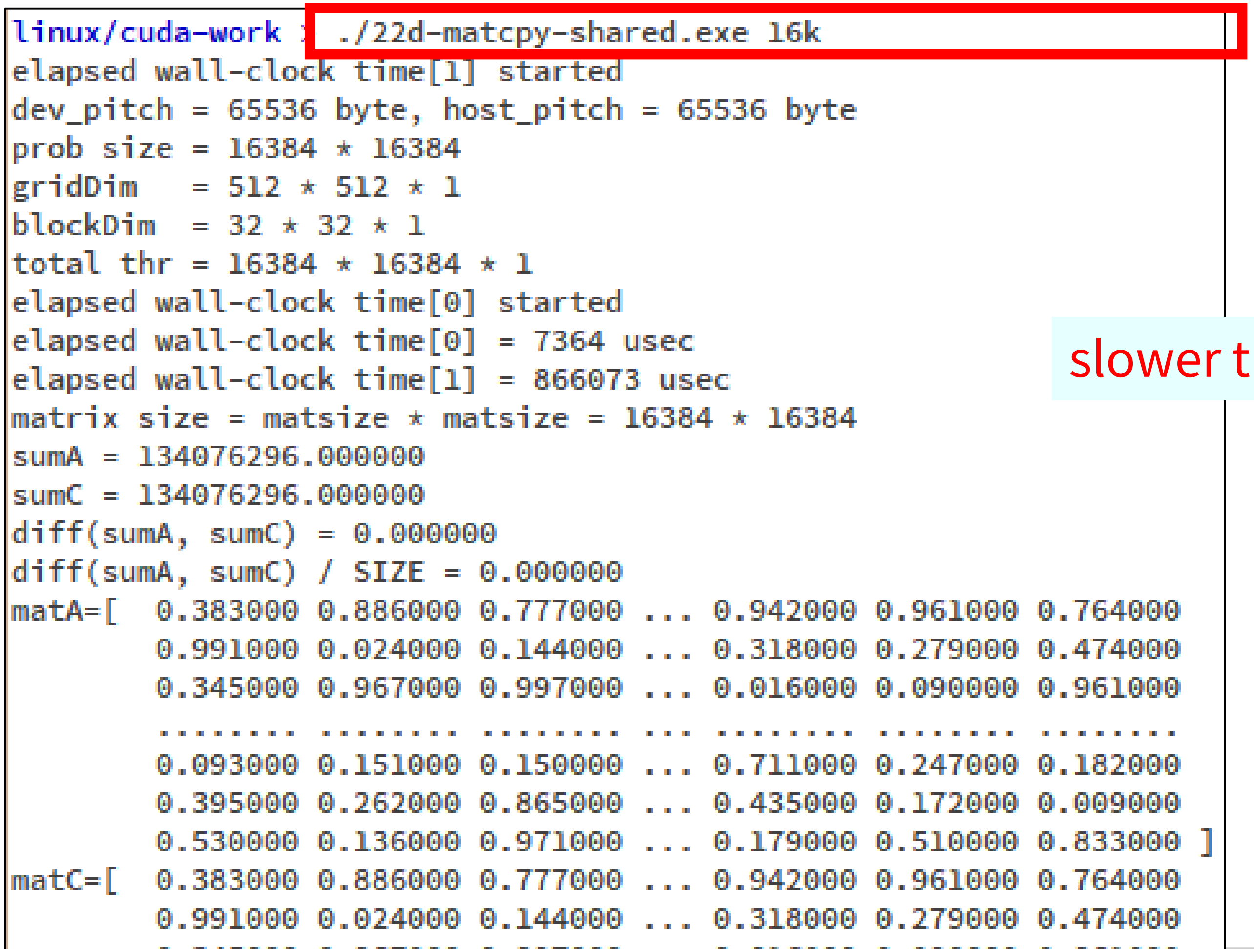
  - 32 x 32 = 1,024
  - 1,024 = maximum number of threads in a thread block
  - 32 = a single warp
- **old code 에서는 16x16 또는 32x16 사용**
  - 이유: maximum number of threads in a thread block **was 512**
  - 16 x 16 = 256
  - 32 x 16 = 512 → 1개의 thread 에서 2개씩 처리 → 32x32 로 작동

matcpy-shared.cu

```
// CUDA kernel function
__global__ void kernelMatCpy( float* C, const float* A, unsigned matsize, size_t pitch_in_elem ) {
   __shared__ float s_mat[32][32];
   register unsigned gy = blockIdx.y * blockDim.y + threadIdx.y; // CUDA-provided index
   if (gy < matsize) {
      register unsigned gx = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
      if (gx < matsize) {
         register unsigned idx = gy * pitch_in_elem + gx; // in element
         s_mat[threadIdx.y][threadIdx.x] = A[idx];
         __syncthreads();
         C[idx] = s_mat[threadIdx.y][threadIdx.x];
      }
   }
}
```

matcpy-shared.cu 실행 결과

- 7,364 usec for simple 16k x 16k matrix copy, shared memory (GeForce RTX 2070)



```
linux/cuda-work : ./22d-matcpy-shared.exe 16k
elapsed wall-clock time[1] started
dev_pitch = 65536 byte, host_pitch = 65536 byte
prob size = 16384 * 16384
gridDim   = 512 * 512 * 1
blockDim  = 32 * 32 * 1
total thr = 16384 * 16384 * 1
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 7364 usec
elapsed wall-clock time[1] = 866073 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076296.000000
diff(sumA, sumC) = 0.000000
diff(sumA, sumC) / SIZE = 0.000000
matA=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
matC=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
```

CPU version	434,394 usec
memcpy version	449,107 usec
CUDA naïve copy	6,728 usec
CUDA shared mem copy	7,364 usec

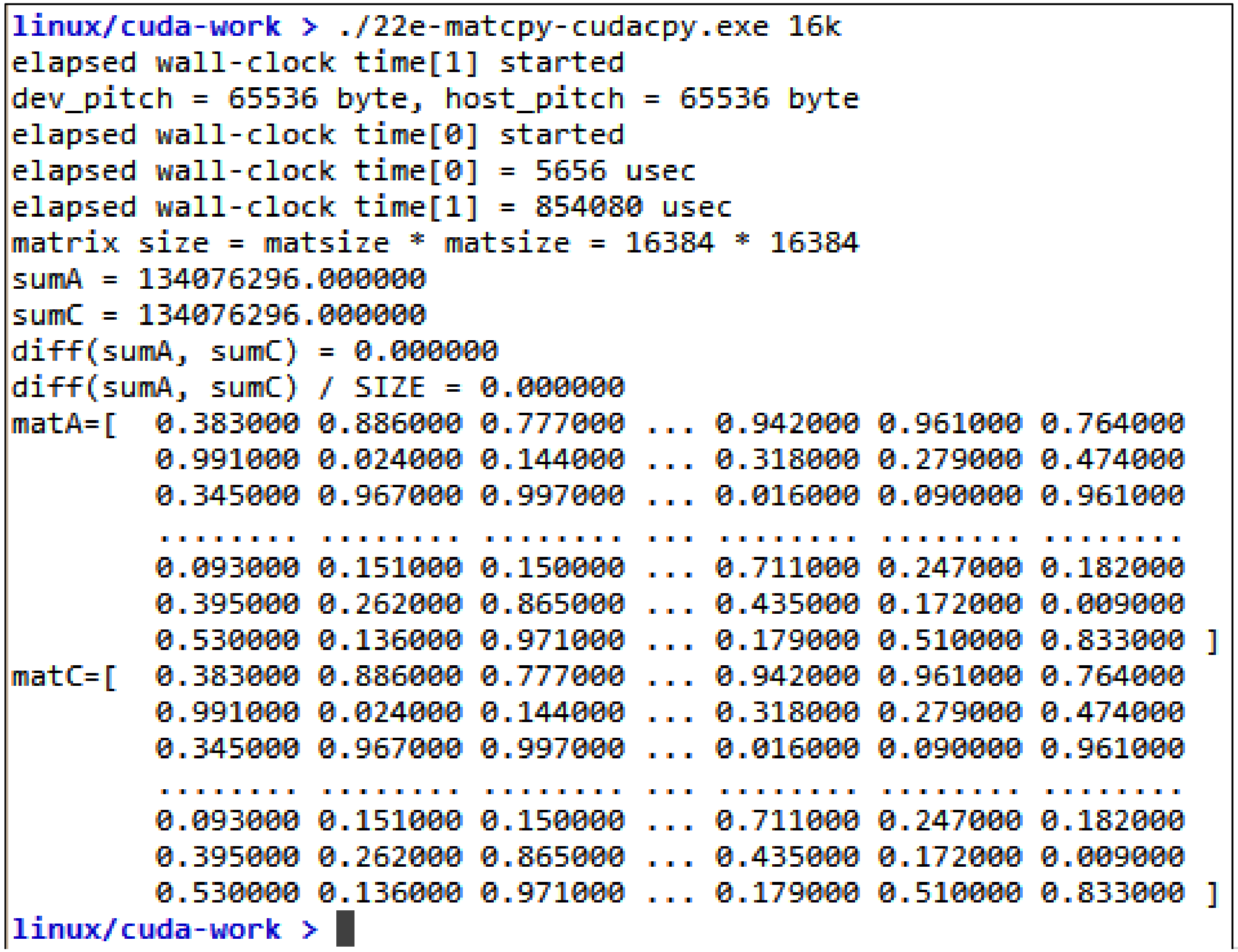
slower than global memory version !

matcpy-cudacpy.cu

```
int main(const int argc, const char* argv[]) {
   . . .
   // CUDA kernel call
   ELAPSED_TIME_BEGIN(0);
   cudaMemcpy2D( dev_matC, dev_pitch, dev_matA, dev_pitch,
                                                                  matsize * sizeof(float), matsize, cudaMemcpyDeviceToDevice);
   cudaDeviceSynchronize();
   ELAPSED_TIME_END(0);
   CUDA_CHECK_ERROR();
   . . .
}
```

matcpy-cudacpy.cu 실행 결과

- **5,656 usec for simple 16k x 16k matrix copy, cudaMemcpy2D( )** **(GeForce RTX 2070)**



```
linux/cuda-work > ./22e-matcpy-cudacpy.exe 16k
elapsed wall-clock time[1] started
dev_pitch = 65536 byte, host_pitch = 65536 byte
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 5656 usec
elapsed wall-clock time[1] = 854080 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076296.000000
diff(sumA, sumC) = 0.000000
diff(sumA, sumC) / SIZE = 0.000000
matA=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
matC=[  0.383000 0.886000 0.777000 ... 0.942000 0.961000 0.764000
        0.991000 0.024000 0.144000 ... 0.318000 0.279000 0.474000
        0.345000 0.967000 0.997000 ... 0.016000 0.090000 0.961000
        ........ ........ ........ ... ........ ........ ........
        0.093000 0.151000 0.150000 ... 0.711000 0.247000 0.182000
        0.395000 0.262000 0.865000 ... 0.435000 0.172000 0.009000
        0.530000 0.136000 0.971000 ... 0.179000 0.510000 0.833000 ]
linux/cuda-work > 
```

CPU version	434,394 usec
memcpy version	449,107 usec
CUDA naïve copy	6,728 usec
CUDA shared mem copy	7,364 usec
CUDA memcpy2D	5,656 usec

내용 contents

- **matrix copy – theoretical limit**

CPU version	434,394 usec
memcpy version	449,107 usec
CUDA naïve version – global memory	6,728 usec
CUDA shared memory version	7,364 usec
CUDA memcpy2D	5,656 usec

Matrix Copy

행렬 복사

폰트 끝단 일치 → 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
본고딕 Noto Sans KR

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
Source Sans Pro

Mathematical Notations O(n log n)
Source Serif Pro