CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

Matrix Transpose

전치 행렬 구하기

내용 contents

- **matrix transpose problem**
  - host version
  - CUDA naïve version – global memory
  - CUDA shared mem, naïve version
  - CUDA shared mem, optimized version
  - CUDA shared memory, bank conflict resolved version

Matrix Transpose 전치 행렬 Problem

- **$C = A^T = A^{tr}$**
  - for simplicity, we assume **square matrices**
  - $[\, c_{i,j} \,] = [\, a_{j,i} \,]$

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix} \Longrightarrow \mathbf{C} = \begin{bmatrix} a_{0,0} & a_{1,0} & a_{2,0} & \cdots & a_{n-1,0} \\ a_{0,1} & a_{1,1} & a_{2,1} & \cdots & a_{n-1,1} \\ a_{0,2} & a_{1,2} & a_{2,2} & \cdots & a_{n-1,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{0,n-1} & a_{1,n-1} & a_{2,n-1} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

A[y][x] = A[y * WIDTH + x]

C[y][x] = A[x][y] = A[x * WIDTH + y]

© Illustration by biztripcru@gmail.com

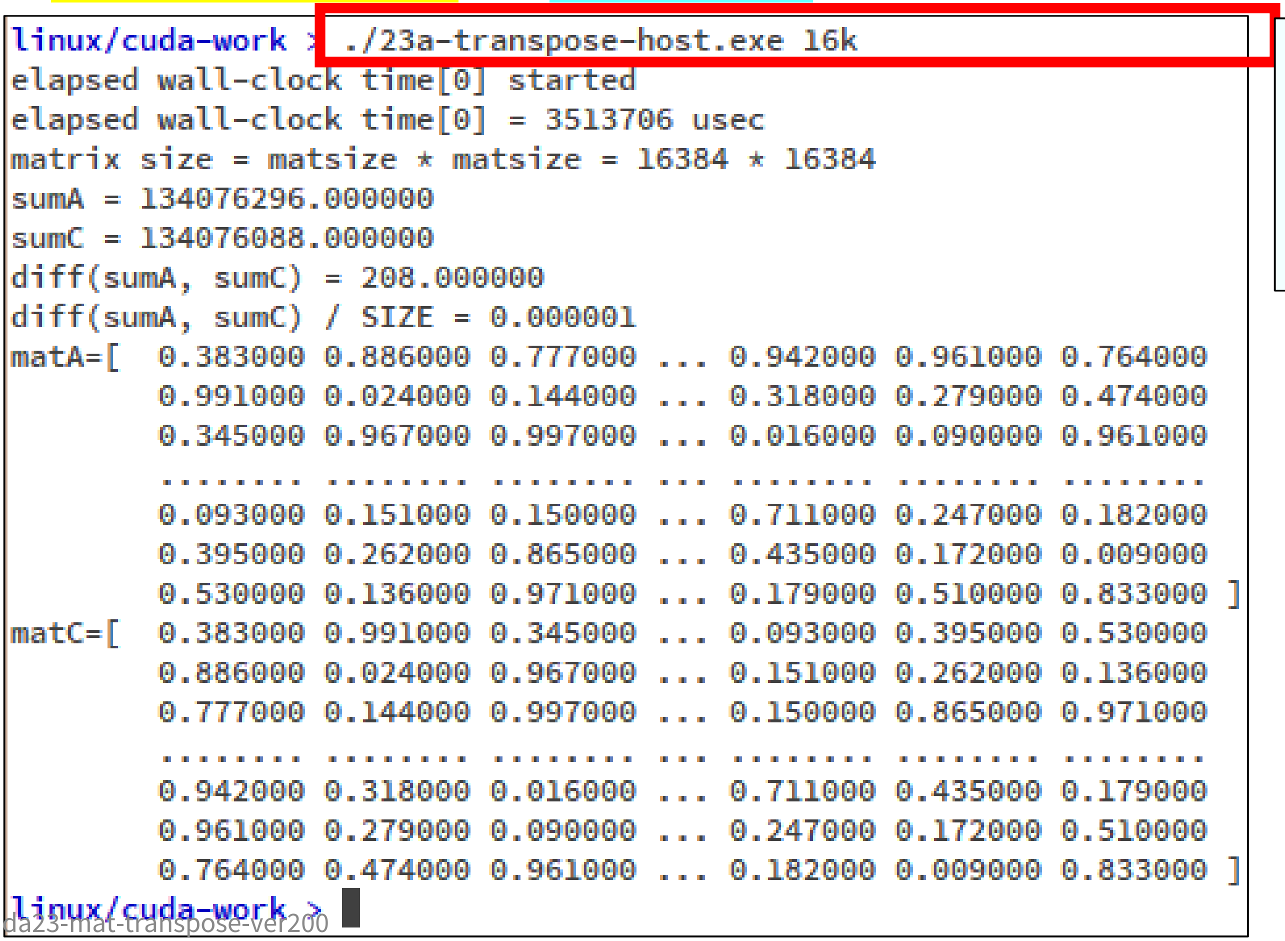
transpose-host.cpp

```
// 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: matrix transpose
  for (register unsigned y = 0; y < matsize; ++y) {
    for (register unsigned x = 0; x < matsize; ++x) {
      unsigned indA = y * matsize + x; // convert to 1D index
      unsigned indC = x * matsize + y; // convert to 1D index
      matC[indC] = matA[indA];
    }
  }
  . . .
}
```

transpose-host.cpp 실행 결과

- 3,513,706 usec for 16k x 16k matrix transpose (Intel Core i5-3570)



```
linux/cuda-work > ./23a-transpose-host.exe 16k
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 3513706 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076088.000000
diff(sumA, sumC) = 208.000000
diff(sumA, sumC) / SIZE = 0.000001
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 0.991000 0.345000 ... 0.093000 0.395000 0.530000
        0.886000 0.024000 0.967000 ... 0.151000 0.262000 0.136000
        0.777000 0.144000 0.997000 ... 0.150000 0.865000 0.971000
        ........ ........ ........ ... ........ ........ ........
        0.942000 0.318000 0.016000 ... 0.711000 0.435000 0.179000
        0.961000 0.279000 0.090000 ... 0.247000 0.172000 0.510000
        0.764000 0.474000 0.961000 ... 0.182000 0.009000 0.833000 ]
linux/cuda-work > 
```

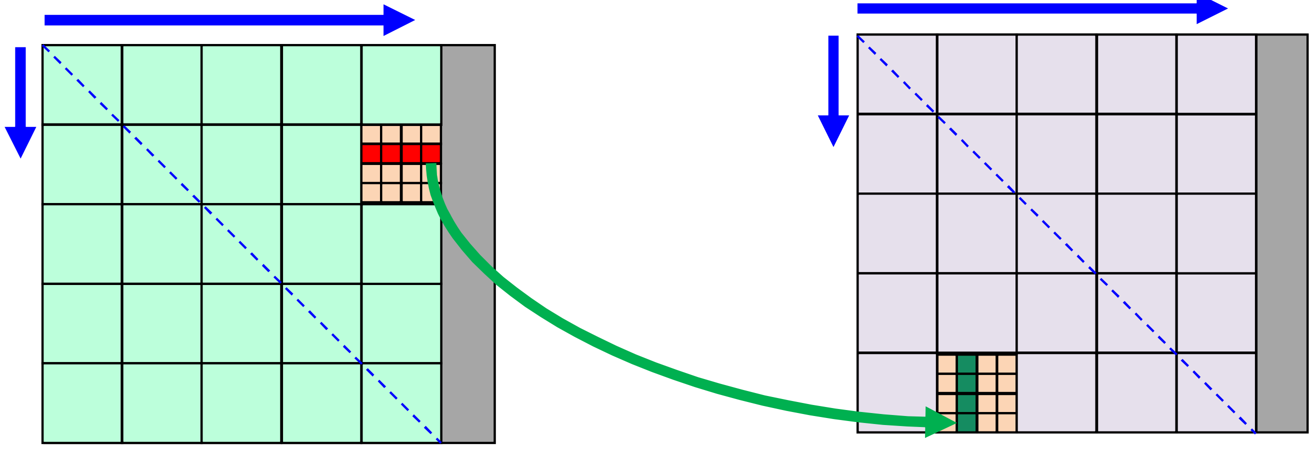
CPU version	431,431 usec
memcpy version	450,472 usec
CUDA naïve copy	6,613 usec
CUDA shared mem copy	7,209 usec
CPU matrix transpose	3,513,706 usec

Matrix Transpose – CUDA version

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

global index : x and y swapped !
gx = blockIdx.y * blockDim.y + threadIdx.y;
gy = blockIdx.x * blockDim.x + threadIdx.x;

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



may be pitched !

may be pitched !

2D matrix
in mathematics

2D matrix
in the **global memory**

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

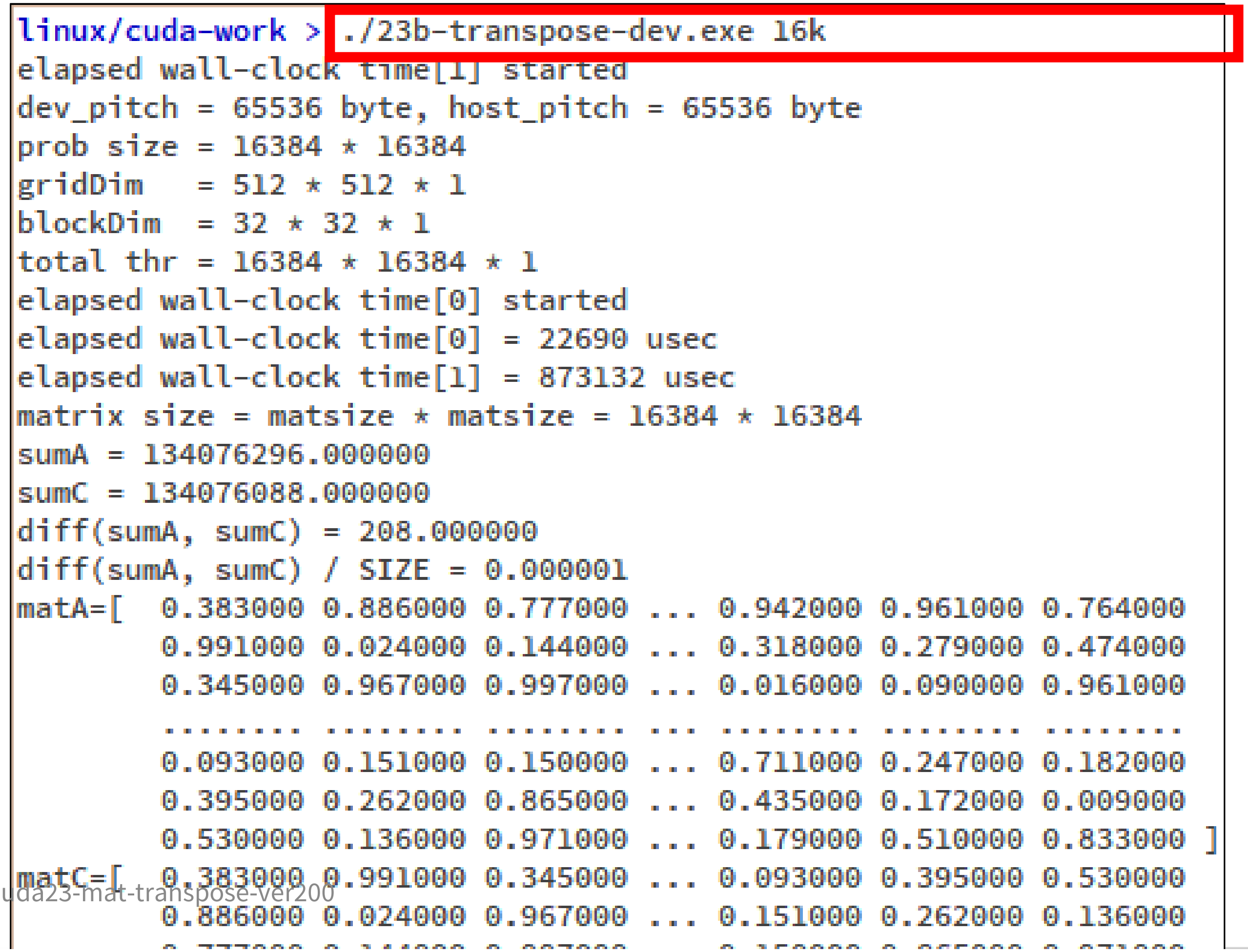
© Illustration by biztripcru@gmail.com

transpose-dev.cu

```
// CUDA kernel function
__global__ void kernelMatTranspose( float* C, const float* A, unsigned 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 idxA = gy * pitch_in_elem + gx;
      register unsigned idxC = gx * pitch_in_elem + gy;
      C[idxC] = A[idxA];
    }
  }
}
```

transpose-dev.cu 실행 결과

- 22,690 usec for 16k x 16k matrix transpose (GeForce RTX 2070)



```
linux/cuda-work > ./23b-transpose-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] = 22690 usec
elapsed wall-clock time[1] = 873132 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076088.000000
diff(sumA, sumC) = 208.000000
diff(sumA, sumC) / SIZE = 0.000001
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 0.991000 0.345000 ... 0.093000 0.395000 0.530000
        0.886000 0.024000 0.967000 ... 0.151000 0.262000 0.136000
        0.777000 0.144000 0.997000 ... 0.150000 0.865000 0.971000
```

CPU version	431,431 usec
memcpy version	450,472 usec
CUDA naïve copy	6,613 usec
CUDA shared mem copy	7,209 usec
CPU matrix transpose	3,513,706 usec
CUDA global memory	22,690 usec

Matrix Transpose – Tiled Approach

global index

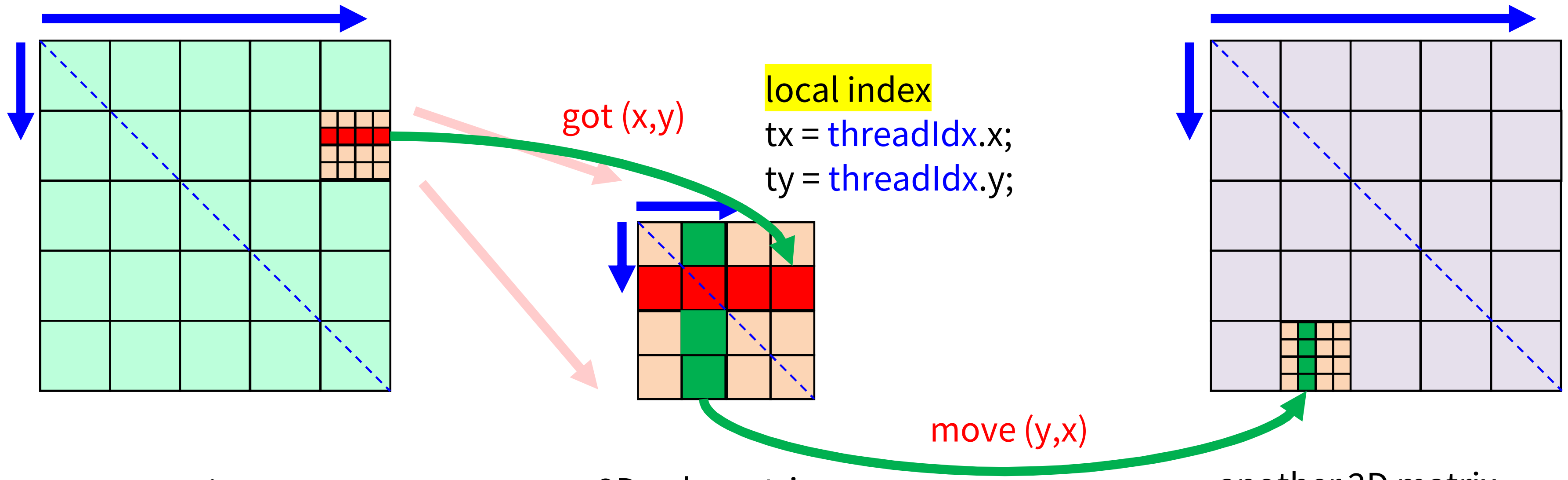
```
gx = blockIdx.x * blockDim.x + threadIdx.x;
gy = blockIdx.y * blockDim.y + threadIdx.y;
```

local index

```
tx = threadIdx.x;
ty = threadIdx.y;
```

global index : x and y swapped !

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



2D matrix
in the **global memory**

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

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

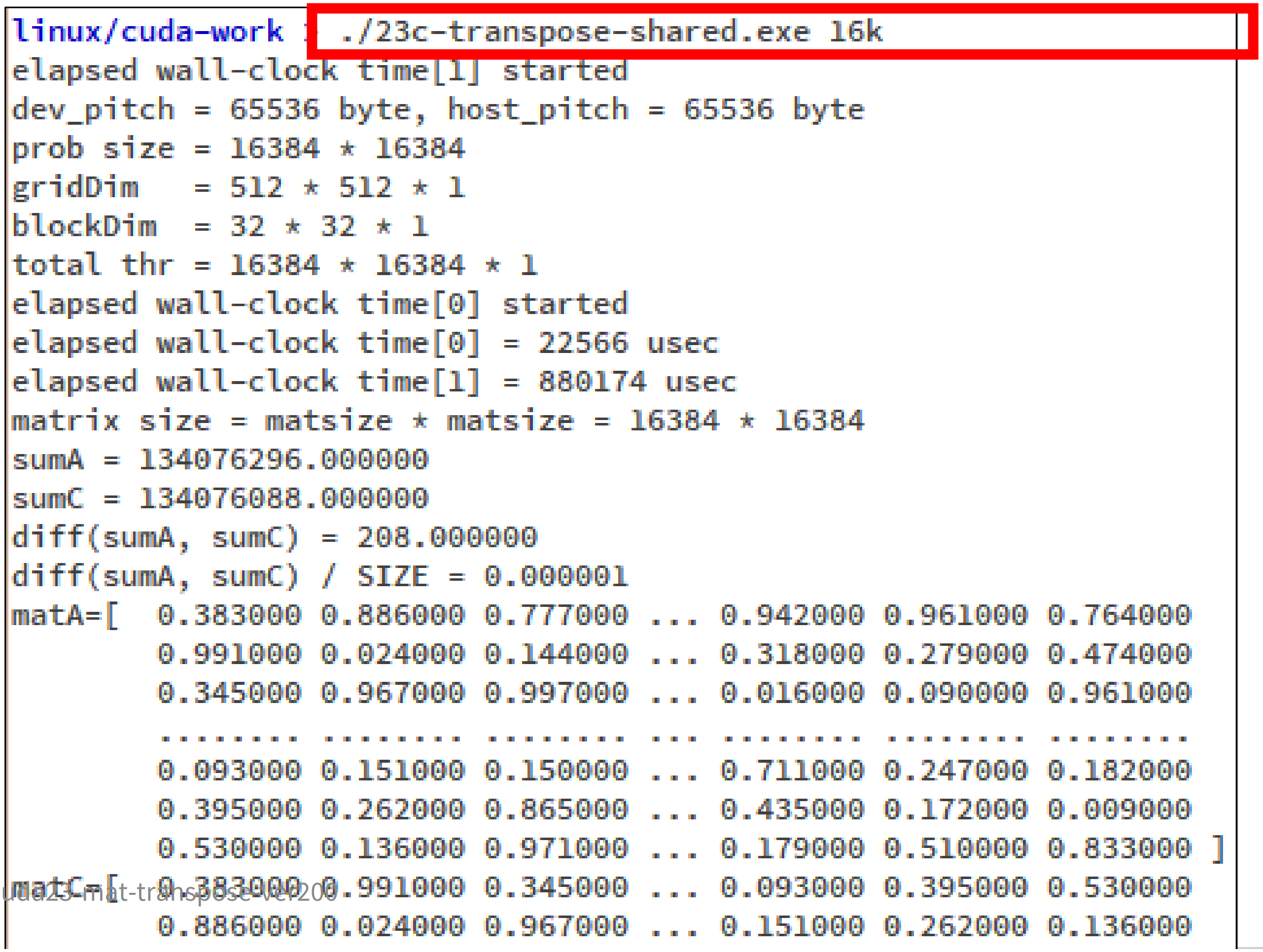
© Illustration by biztripcru@gmail.com

transpose-shared.cu

```
// CUDA kernel function
__global__ void kernelMatTranspose( float* C, const float* A, unsigned matsize, size_t pitch_in_elem ) {
    __shared__ float mat[32][32];
     // pick up for the shared memory
    register unsigned gy = blockIdx.y * blockDim.y + threadIdx.y; // CUDA-provided index
    register unsigned gx = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
    if (gy < matsize && gx < matsize) {
        register unsigned idxA = gy * pitch_in_elem + gx;
        mat[threadIdx.y][threadIdx.x] = A[idxA];
    }
    __syncthreads();
    // transposed position
    if (gy < matsize && gx < matsize) {
        register unsigned idxC = gx * pitch_in_elem + gy;
        C[idxC] = mat[threadIdx.y][threadIdx.x];
    }
}
```

transpose-shared.cu 실행 결과

- 22,566 usec for 16k x 16k matrix transpose (GeForce RTX 2070)



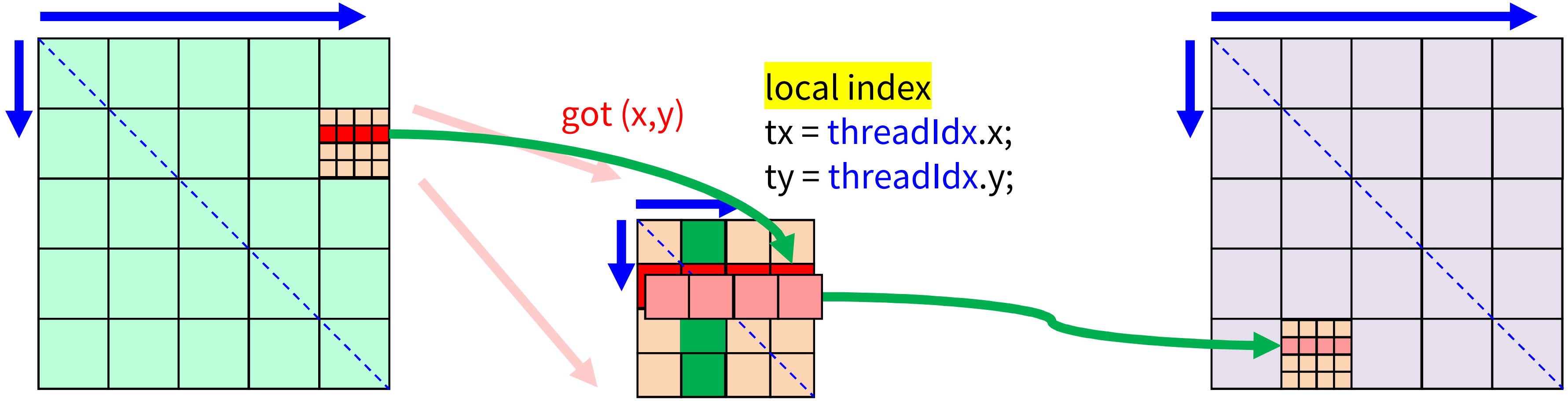
```
linux/cuda-work ] ./23c-transpose-shared.exe 16k
elapsed wall-clock time[1] started
dev_pitch = 65536 byte, host_pitch = 65536 byte
prob size = 16384 * 16384
gridDim   = 512 * 512 * 1
blockDim  = 32 * 32 * 1
total thr = 16384 * 16384 * 1
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 22566 usec
elapsed wall-clock time[1] = 880174 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076088.000000
diff(sumA, sumC) = 208.000000
diff(sumA, sumC) / SIZE = 0.000001
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 0.991000 0.345000 ... 0.093000 0.395000 0.530000
        0.886000 0.024000 0.967000 ... 0.151000 0.262000 0.136000
```

CPU version	431,431 usec
memcpy version	450,472 usec
CUDA naïve copy	6,613 usec
CUDA shared mem copy	7,209 usec
CPU matrix transpose	3,513,706 usec
CUDA global memory	22,690 usec
CUDA shared, naïve	22,566 usec

Matrix Transpose – Tiled Approach

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

global index : block swapped !
gx = blockIdx.y * blockDim.y + threadIdx.x;
gy = blockIdx.x * blockDim.x + threadIdx.y;



got (x,y)

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

move (y,x)

2D matrix
in the **global memory**

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

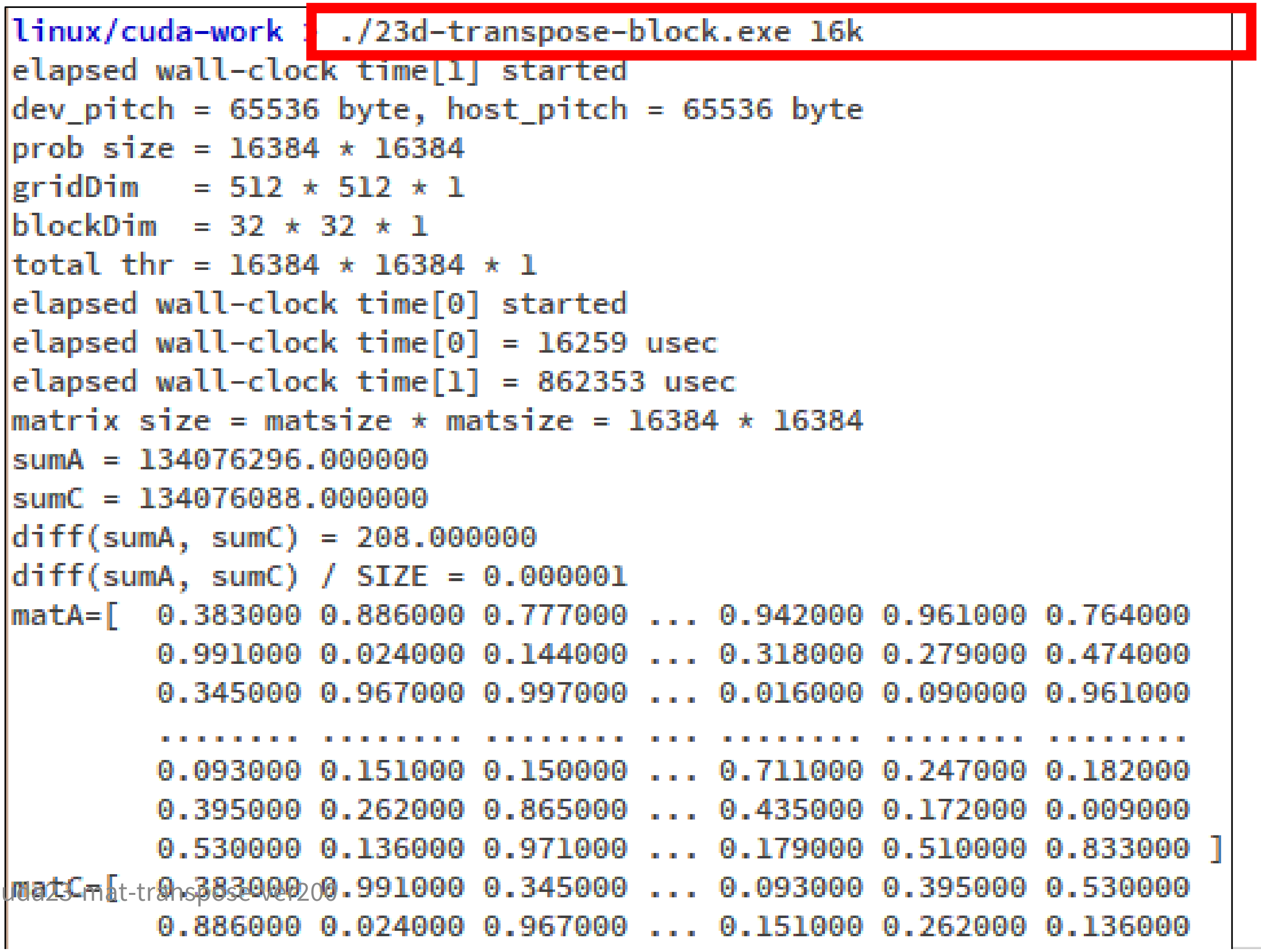
another 2D matrix
in the **global memory**

transpose-block.cu

```
// CUDA kernel function
__global__ void kernelMatTranspose( float* C, const float* A, unsigned matsize, size_t pitch_in_elem ) {
  __shared__ float mat[32][32];
  // pick up for the shared memory
  register unsigned gy = blockIdx.y * blockDim.y + threadIdx.y; // CUDA-provided index
  register unsigned gx = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
  if (gy < matsize && gx < matsize) {
    register unsigned idxA = gy * pitch_in_elem + gx;
    mat[threadIdx.y][threadIdx.x] = A[idxA];
  }
  __syncthreads();
  // transposed position
  gy = blockIdx.x * blockDim.x + threadIdx.y; // CUDA-provided index
  gx = blockIdx.y * blockDim.y + threadIdx.x; // CUDA-provided index
  if (gy < matsize && gx < matsize) {
    register unsigned idxC = gy * pitch_in_elem + gx;
    C[idxC] = mat[threadIdx.x][threadIdx.y];
  }
}
```

transpose-block.cu 실행 결과

- 16,259 usec for 16k x 16k matrix transpose (GeForce RTX 2070)

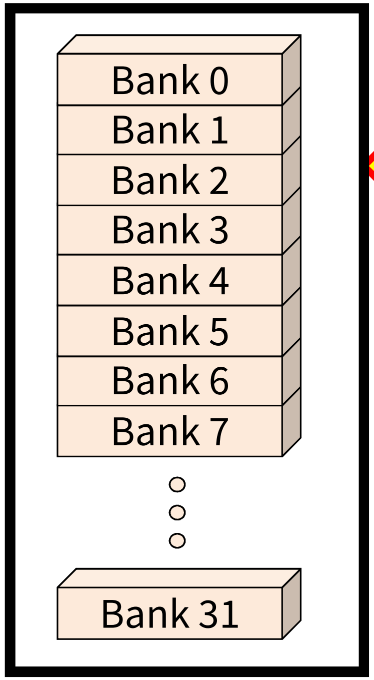


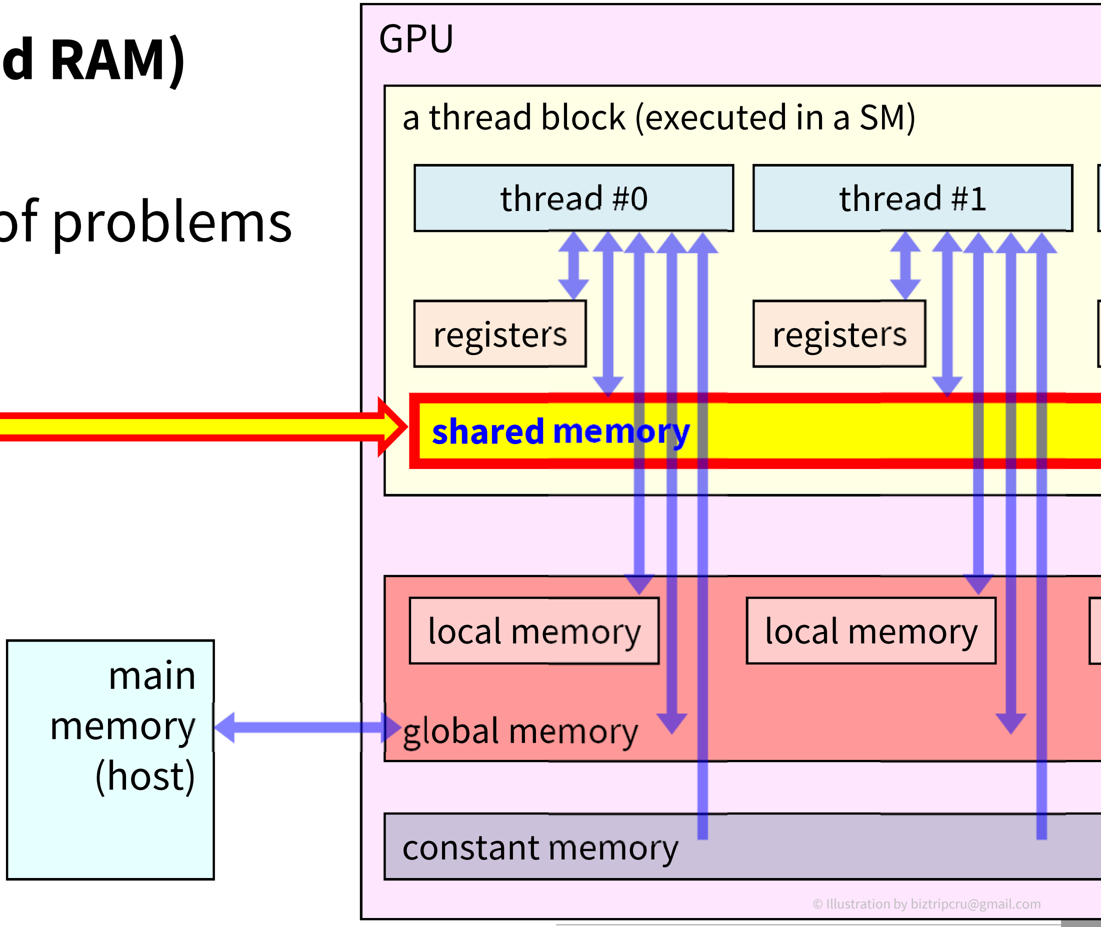
```
linux/cuda-work ] ./23d-transpose-block.exe 16k
elapsed wall-clock time[1] started
dev_pitch = 65536 byte, host_pitch = 65536 byte
prob size = 16384 * 16384
gridDim   = 512 * 512 * 1
blockDim  = 32 * 32 * 1
total thr = 16384 * 16384 * 1
elapsed wall-clock time[0] started
elapsed wall-clock time[0] = 16259 usec
elapsed wall-clock time[1] = 862353 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076088.000000
diff(sumA, sumC) = 208.000000
diff(sumA, sumC) / SIZE = 0.000001
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 0.991000 0.345000 ... 0.093000 0.395000 0.530000
        0.886000 0.024000 0.967000 ... 0.151000 0.262000 0.136000
```

CPU version	431,431 usec
memcpy version	450,472 usec
CUDA naïve copy	6,613 usec
CUDA shared mem copy	7,209 usec
CPU matrix transpose	3,513,706 usec
CUDA global memory	22,690 usec
CUDA shared, naïve	22,566 usec
CUDA shared, optimized	16,259 usec

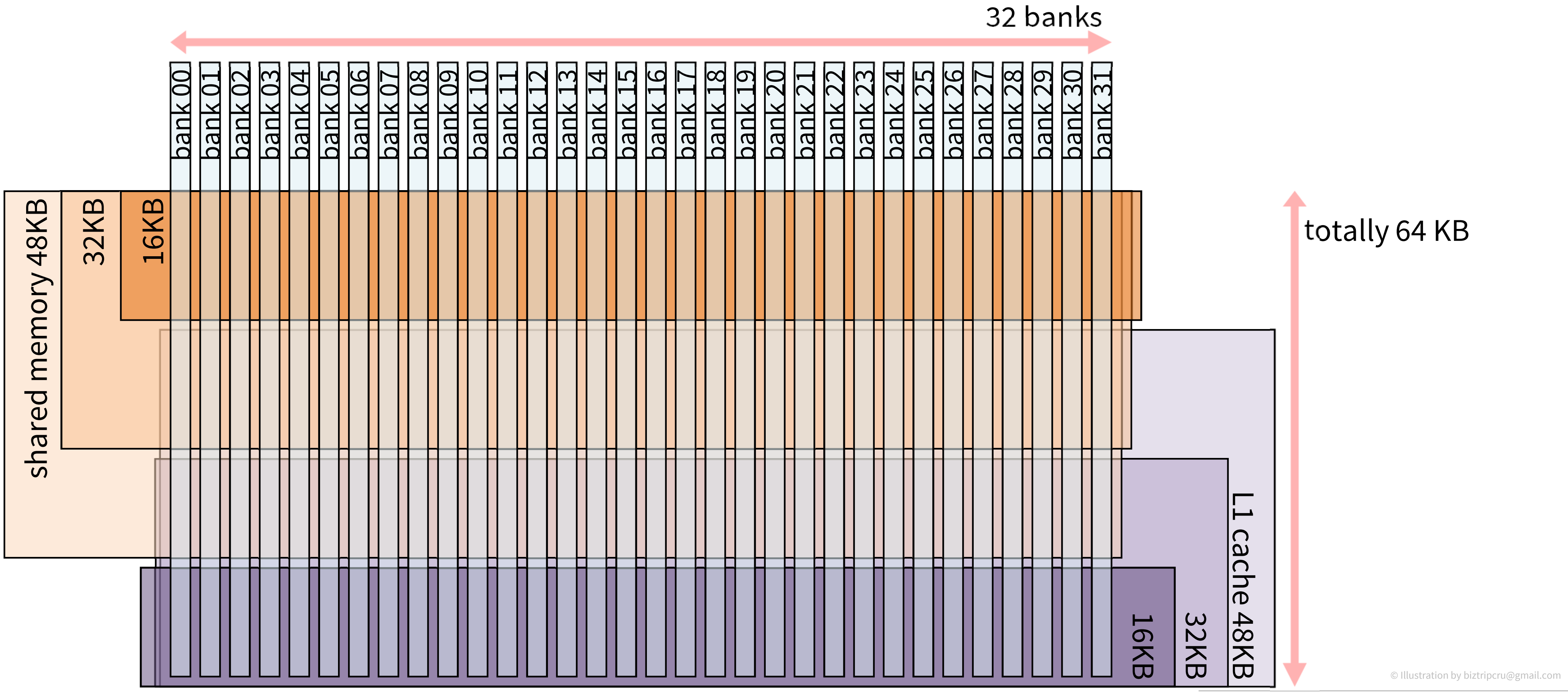
Shared Memory Handling

- **shared memory : SPRAM (scratch-pad RAM)**
  - DRAM memory 와 다른 특성 !
  - they are **banked** ! → another kind of problems





Shared Memory의 32 bank 구조



© Illustration by biztripcru@gmail.com

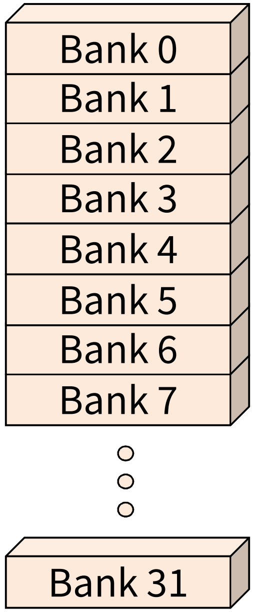
Banked Memory : Best Case

```
__shared__ float a[65536];
```

- thread 0 reads a[0] ← Bank 0
- thread 1 reads a[1] ← Bank 1
- thread 2 reads a[2] ← Bank 2
- thread 3 reads a[3] ← Bank 3
- thread 4 reads a[4] ← Bank 4
- …
- thread 31 reads a[31] ← Bank 31

- 32 threads get the result in a single cycle !

- Bank 0: a[0], a[32], a[64], …
- Bank 1: a[1], a[33], a[65], …
- Bank 2: a[2], a[34], a[66], …
- Bank 3: a[3], a[35], a[67], …
- Bank 4: a[4], a[36], a[68], …
- Bank 5: a[5], a[37], a[69], …
- Bank 6: a[6], a[38], a[70], …
- …
- Bank 31: a[31], a[63], a[95], …



© Illustration by biztripcru@gmail.com

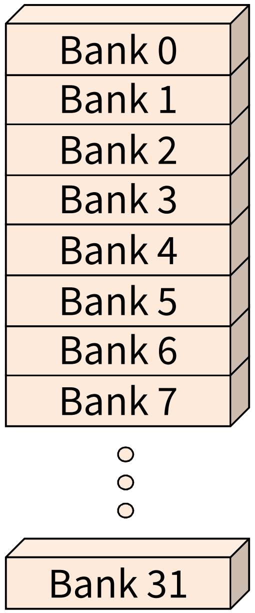
Banked Memory : broadcast case

```
__shared__ float a[65536];
```

- **thread 0 reads a[0] ← Bank 0**
- **thread 1 reads a[0] ← Bank 0**
- **thread 2 reads a[0] ← Bank 0**
- **thread 3 reads a[0] ← Bank 0**
- **thread 4 reads a[0] ← Bank 0**
- **…**
- **thread 31 reads a[0] ← Bank 0**

- **Everybody wants a[0]**
- **Bank 0 broadcasts a[0] in a single cycle.**

- **Bank 0: a[0], a[32], a[64], …**
- **Bank 1: a[1], a[33], a[65], …**
- **Bank 2: a[2], a[34], a[66], …**
- **Bank 3: a[3], a[35], a[67], …**
- **Bank 4: a[4], a[36], a[68], …**
- **Bank 5: a[5], a[37], a[69], …**
- **Bank 6: a[6], a[38], a[70], …**
- **…**
- **Bank 31: a[31], a[63], a[95], …**



© Illustration by biztripcru@gmail.com

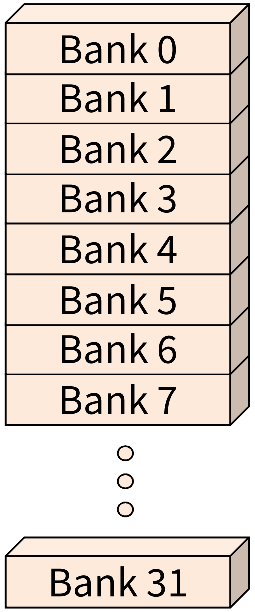
Banked Memory : serialized case

```
__shared__ float a[65536];
```

- **thread 0 reads a[0] ← Bank 0**
- **thread 1 reads a[32] ← Bank 0**
- **thread 2 reads a[64] ← Bank 0**
- **thread 3 reads a[96] ← Bank 0**
- **thread 4 reads a[128] ← Bank 0**
- **...**
- **thread 31 reads a[992] ← Bank 0**

- **Bank 0 returns a[0], a[32], a[64], . . . , a[992] serially, in 32 cycles.**

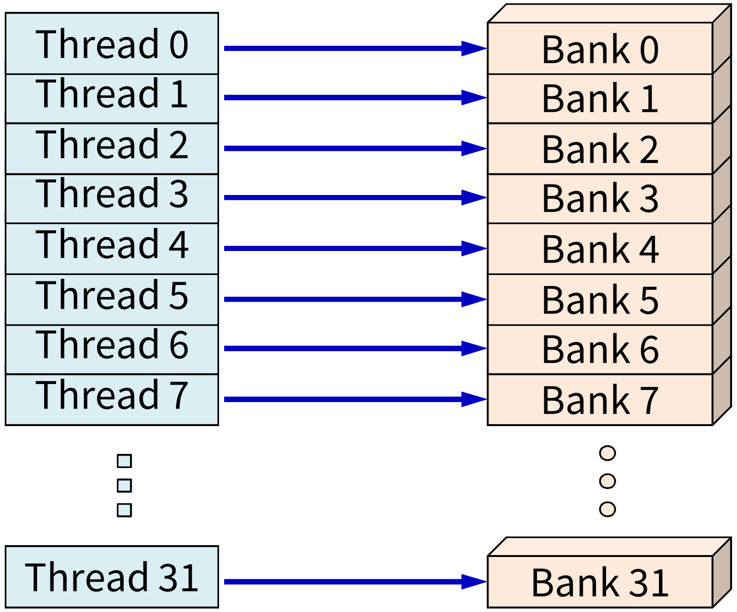
- **Bank 0: a[0], a[32], a[64], ...**
- **Bank 1: a[1], a[33], a[65], ...**
- **Bank 2: a[2], a[34], a[66], ...**
- **Bank 3: a[3], a[35], a[67], ...**
- **Bank 4: a[4], a[36], a[68], ...**
- **Bank 5: a[5], a[37], a[69], ...**
- **Bank 6: a[6], a[38], a[70], ...**
- **...**
- **Bank 31: a[31], a[63], a[95], ...**



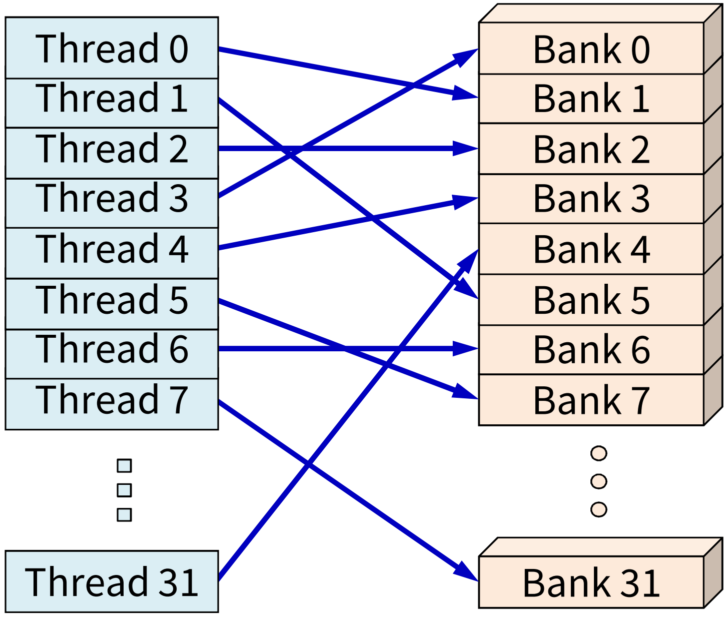
© Illustration by biztripcru@gmail.com

Bank Addressing Examples

- No Bank Conflicts



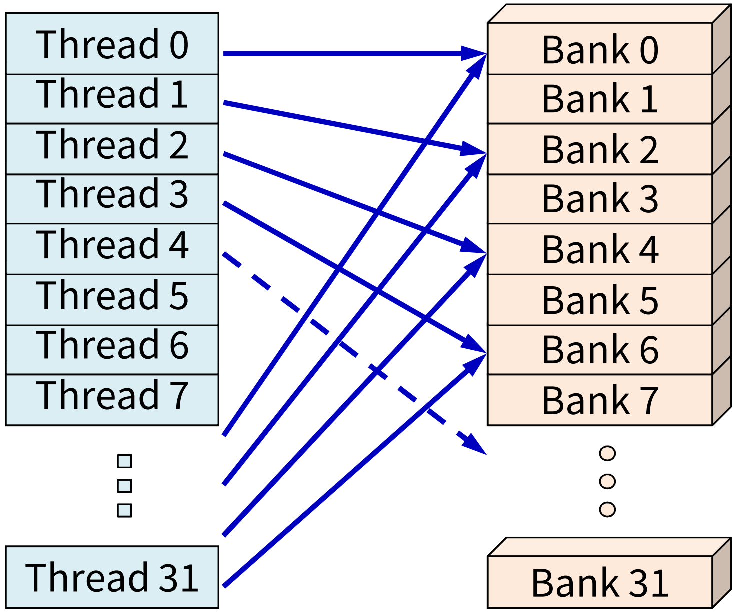
- No Bank Conflicts



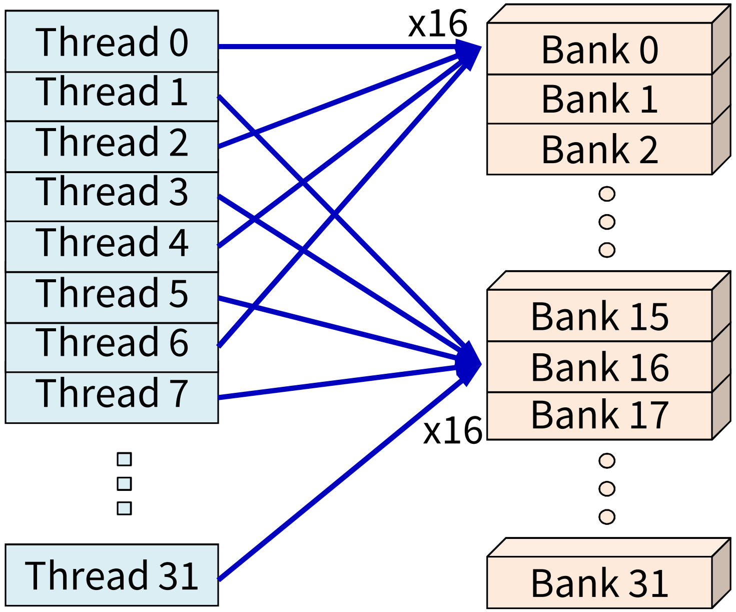
© Illustration by biztripcru@gmail.com

Bank Addressing Examples

- **2-way Bank Conflicts**
  - stride 2 case: a[2*tx]

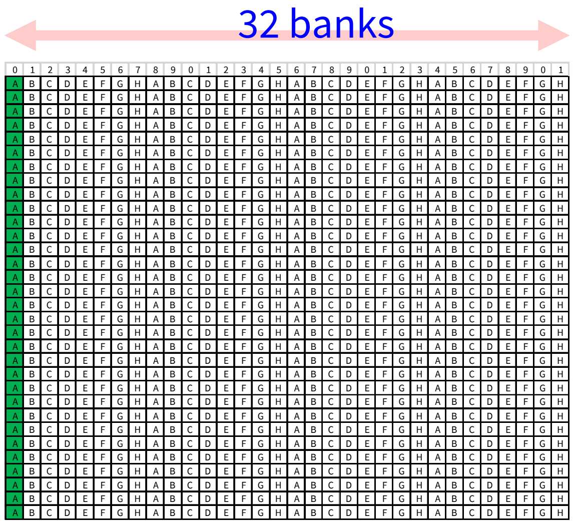


- **16-way Bank Conflicts**
  - even or odd case: a[16*tx]



© Illustration by biztripcru@gmail.com

Bank Conflict: 2D Case

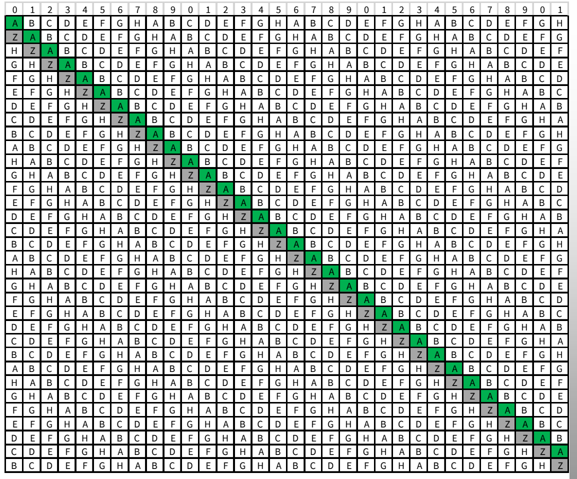


```
__shared__ float  mat[32][32];
 mat[threadIdx.y][threadIdx.x] = . . . ; // no bank conflict
. . . = mat[threadIdx.x][threadIdx.y]; // bank conflict
```

- **threadIdx.x = 0, . . ., 31, for a specific threadIdx.y,**
  - mat[0][ty], mat[1][ty], . . . , mat[31][ty] : 모두 1개의 bank에 몰림
- **해결책?**

```
__shared__ float  mat[32][32+1];
```

  - mat[0][ty], mat[1][ty], . . . , mat[31][ty] : **완벽히 분산됨**!



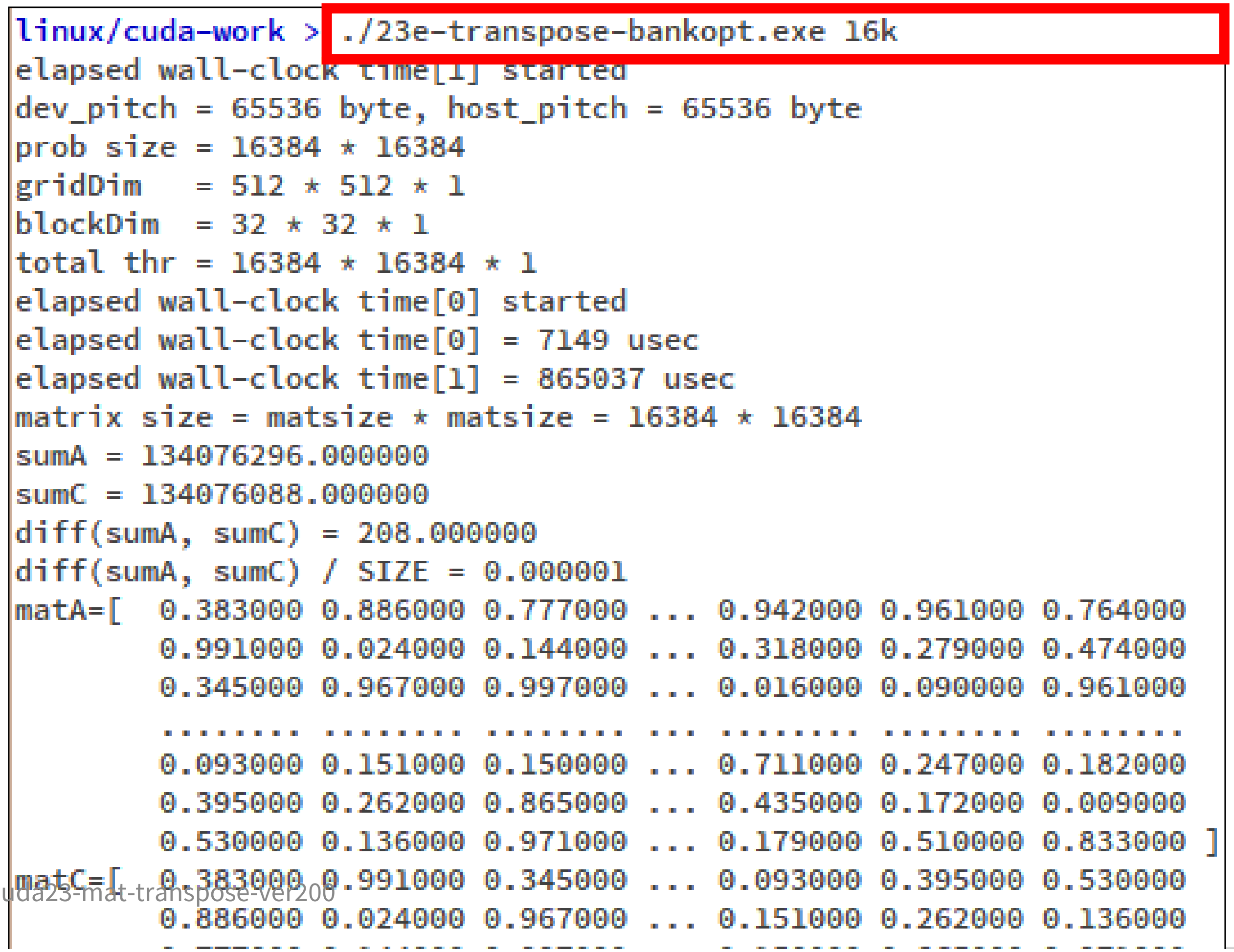
© Illustration by biztripcru@gmail.com

transpose-bankopt.cu

```
// CUDA kernel function
__global__ void kernelMatTranspose( float* C, const float* A, unsigned matsize, size_t pitch_in_elem ) {
   __shared__ float mat[32][32 + 1];
   // pick up for the shared memory
   register unsigned gy = blockIdx.y * blockDim.y + threadIdx.y; // CUDA-provided index
   register unsigned gx = blockIdx.x * blockDim.x + threadIdx.x; // CUDA-provided index
   if (gy < matsize && gx < matsize) {
      register unsigned idxA = gy * pitch_in_elem + gx;
      mat[threadIdx.y][threadIdx.x] = A[idxA];
   }
   __syncthreads();
   // transposed position
   gy = blockIdx.x * blockDim.x + threadIdx.y; // CUDA-provided index
   gx = blockIdx.y * blockDim.y + threadIdx.x; // CUDA-provided index
   if (gy < matsize && gx < matsize) {
      register unsigned idxC = gy * pitch_in_elem + gx;
      C[idxC] = mat[threadIdx.x][threadIdx.y];
   }
}
```

transpose-bankopt.cu 실행 결과

- 7,149 usec for 16k x 16k matrix transpose (GeForce RTX 2070)



```
linux/cuda-work > ./23e-transpose-bankopt.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] = 7149 usec
elapsed wall-clock time[1] = 865037 usec
matrix size = matsize * matsize = 16384 * 16384
sumA = 134076296.000000
sumC = 134076088.000000
diff(sumA, sumC) = 208.000000
diff(sumA, sumC) / SIZE = 0.000001
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 0.991000 0.345000 ... 0.093000 0.395000 0.530000
        0.886000 0.024000 0.967000 ... 0.151000 0.262000 0.136000
```

CPU version	431,431 usec
memcpy version	450,472 usec
CUDA naïve copy	6,613 usec
CUDA shared mem copy	7,209 usec
CPU matrix transpose	3,513,706 usec
CUDA global memory	22,690 usec
CUDA shared, naïve	22,566 usec
CUDA shared, optimized	16,259 usec
CUDA sh mem, bank optimized	7,149 usec

Shared Memory 특징

- **사용 목표:**
  - inter-thread communication within a block
  - cache data to reduce global memory accesses
- **주의: shared memory is banked**
  - only matters for **threads within a warp**
  - bank conflict 상황은 피해야
- **best performance 예측 방법:**
  - 모든 read/write 시의 index 를 threadIdx.x 로 변경
  - bank conflict 를 완전히 피했으므로, best performance 가능
  - 실제 구현과 속도 비교 가능

내용 contents

- **matrix transpose problem**

host version	3,513,706 usec
CUDA naïve version – global memory	22,690 usec
CUDA shared mem, naïve version	22,566 usec
CUDA shared mem, optimized version	16,259 usec
CUDA shared memory, bank conflict resolved version	7,149 usec

Matrix Transpose

전치 행렬 구하기

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