# HP-SEE
# Tiled matrix multiplication (using shared memory)

www.hp-see.eu

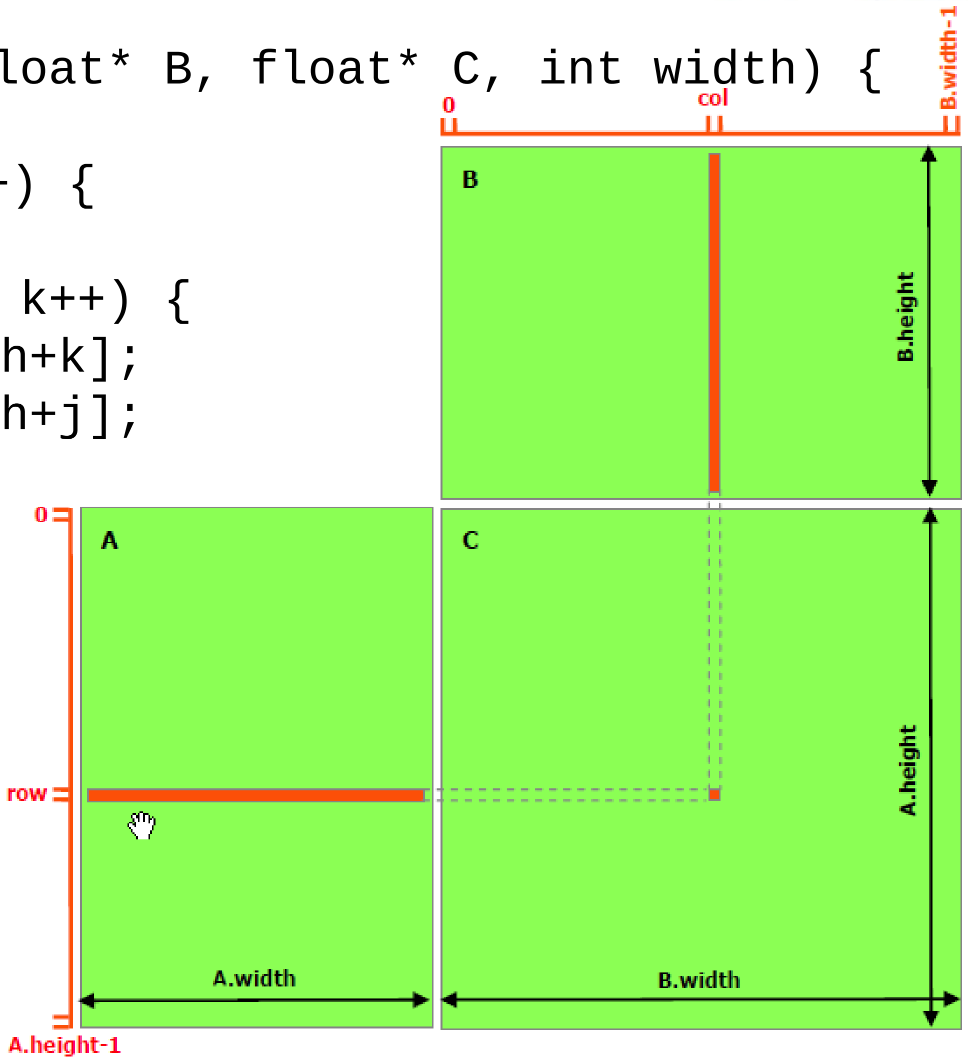
**Petar Jovanović**
**Scientific Computing Laboratory**
**Institute of Physics Belgrade**
**petarj@ipb.ac.rs**

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

The HP-SEE initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 261499

# Matrix Multiplication: Simple host version in C

```
void matrixMulOnHost(float* A, float* B, float* C, int width) {
    for (int i=0; i<width; i++) {
        for (int j=0; j<width; j++) {
            double sum = 0;
            for (int k=0; k<width; k++) {
                double a = A[i*width+k];
                double b = B[k*width+j];
                sum += a*b;
            }
            C[i*width+j] = sum;
        }
    }
}
```

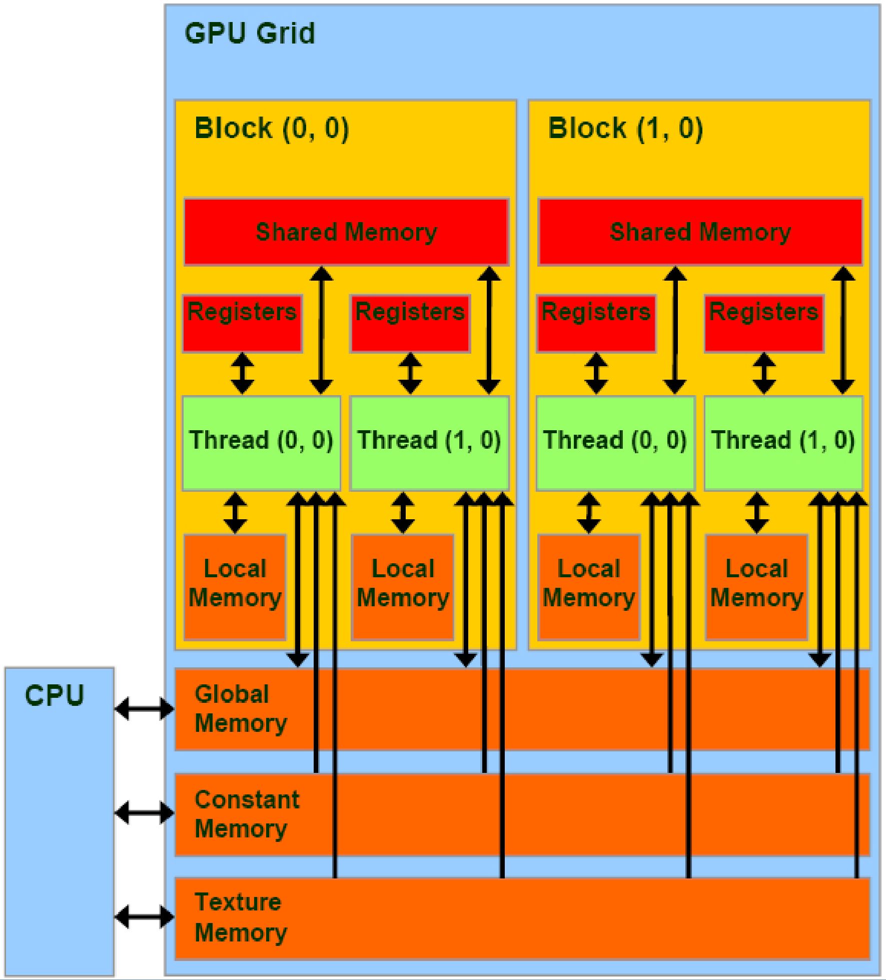


# Simple Matrix Multiplication Kernel

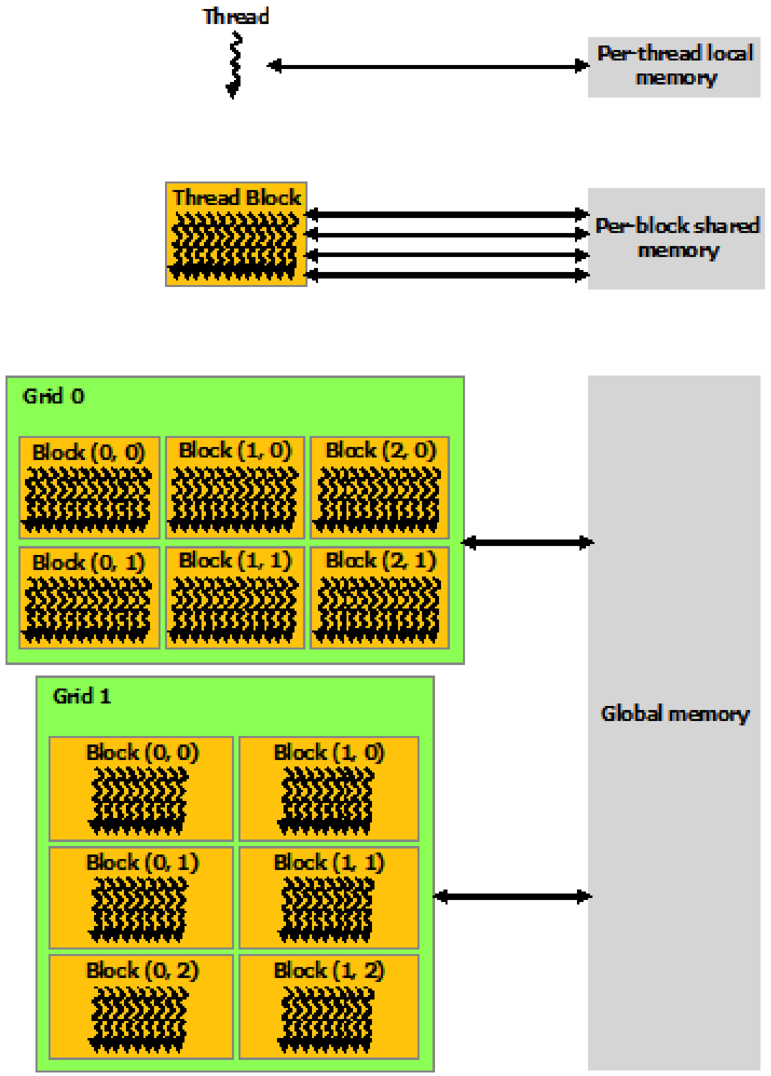
```
__global__ void matrixMulKernel(float* A, float* B, float* C,
                                                    int width) {
    int row = blockIdx.y*blockDim.y+threadIdx.y;
    Int col = blockIdx.x*blockDim.x+threadIdx.x;

    if ((row<width) && (col<width)) {
        float tmp = 0;
        for (int i=0; i<width; i++)
            tmp += A[row*width+i]*B[i*width+col];
        C[row*width+col] = tmp;
    }
}
```

# CUDA Memory Reminder(1)

- Local memory & registers
  - Small
  - Accessed by one core/thread
  - Low latency (~1 cycle)
- Shared memory
  - Not large (16 KB)
  - Low latency (~5 cycles)
  - Shared between cores/threads within a thread block
- Global memory
  - Large (256mb+)
  - High bandwidth (100 GB/s)
  - High latency (~500 cycles)
- Constant memory
  - Read only, low latency, shared by all threads



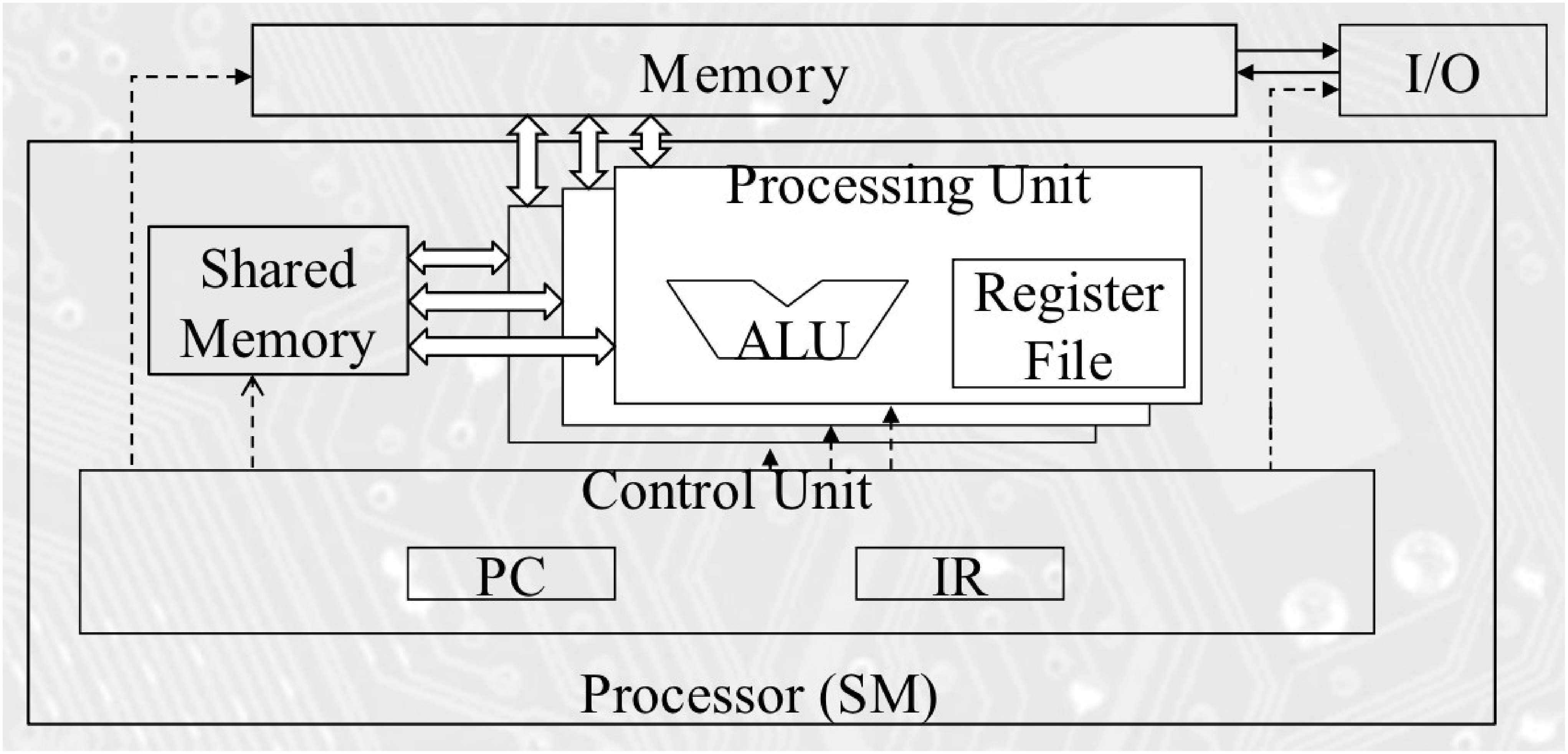
# CUDA Memory Reminder(2)



## Shared Memoru in CUDA

- A special type of memory whose contents are explicitly declared and used in the source code
  - One in each SM
  - Accessed at much higher speed than global memory
  - Still accessed by memory instructions
  - A form of scratchpad memory

# Hardware View



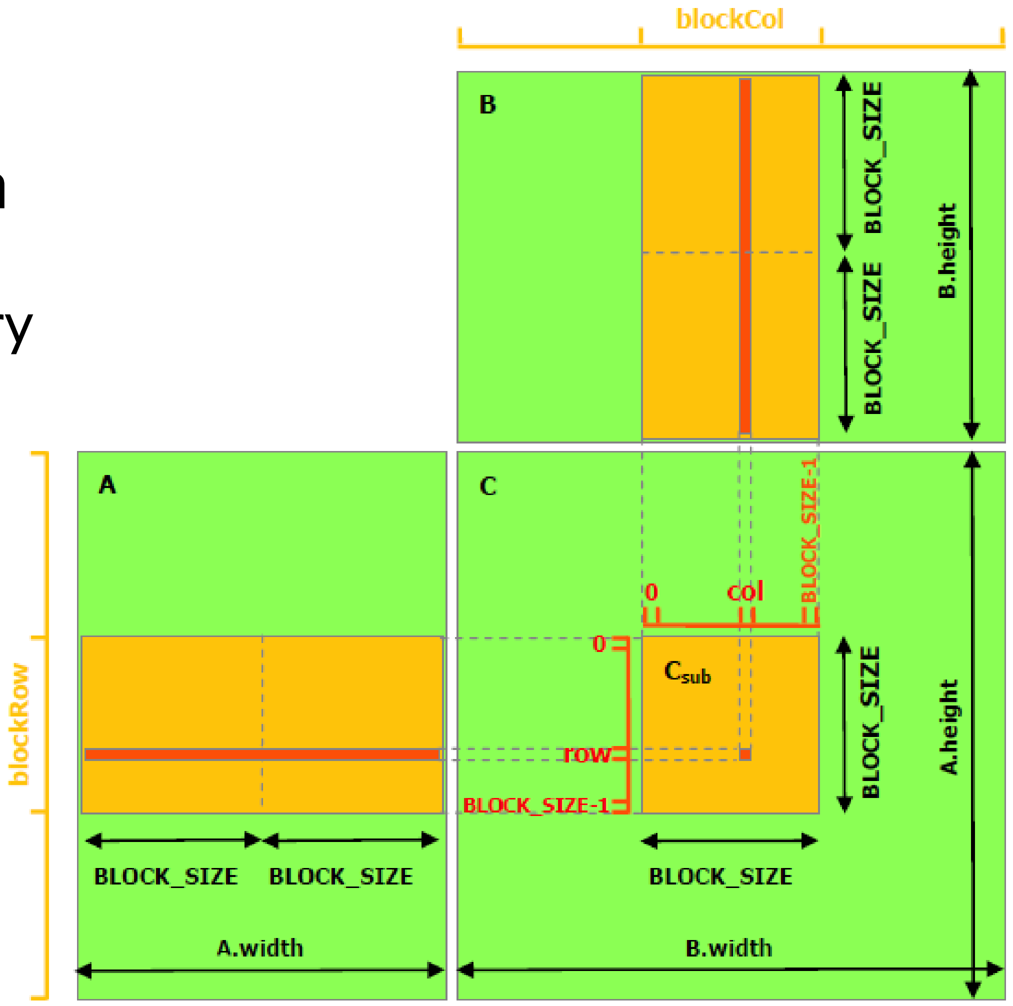
# CUDA Variable Type Qualifiers

| Variable declaration | Memory | Scope | Lifetime |
|---|---|---|---|
| `int localVar;` | register | thread | thread |
| `__device__ __shared__ int sharedVar;` | shared | block | block |
| `__device__ int globalVar` | global | grid | application |
| `__device__ __constant__ int constantVar;` | constant | grid | application |

- `__device__` is optional when used with `__shared__` or `__constant__`
- Automatic variables reside in registers
  - Except per-thread arrays, which are in global memory

# Common Programming Strategy

- Partition data into tiles that fit into shared memory
- Each thread block handles a tile by:
  - Loading it into shared memory using multiple threads
  - Performing the computation on the tile
  - Copying results back from shared to global memory



# Tiled Matrix Multiplication Kernel

```
__global__ void matrixMulKernel(float* dA, float* dB, float* dC,
                                                  int width) {
    __shared__ float dsA[TILE_WIDTH][TILE_WIDTH];
    __shared__ float dsB[TILE_WIDTH][TILE_WIDTH];

    int bx = blockIdx.x, by = blockIdx.y;
    int tx = threadIdx.x, ty = threadIdx.y;

    int row = by*TILE_WIDTH+ty;
    int col = bx*TILE_WIDTH+tx;
    float tmp = 0;

    for (int i=0; i<width/TILE_WIDTH; i++) {
        dsA[ty][tx]=dA[row*width+i*TILE_WIDTH+tx];
        dsB[ty][tx]=dB[(i*TILE_WIDTH+ty)*width+col];
        __synchtreads();
        for (int j=0; j<TILE_WIDTH; j++)
            tmp += dsA[ty][j]*dsB[j][tx];
        __synchthreads();
    }
    dC[row*width+col] = tmp;
}
```

# Size Considerations

- Each thread block should have many threads
  - For 16x16 threads, each block performs 512 loads for 8192 mul/add operations
  - For 32x32 threads, we have 2048 loads and 65536 mul/add operations
- Each SMP in Fermi has 16 or 48 KB shared memory
  - Size is implementation dependent
  - TILE_WIDTH=16 takes 2*256*4B=2KB of shared memory
    - Can have up to 8 thread blocks executing
  - TILE_WIDTH=32 takes 8KB of shared memory per block, which allows for 2 or 6 blocks to be active at the same time