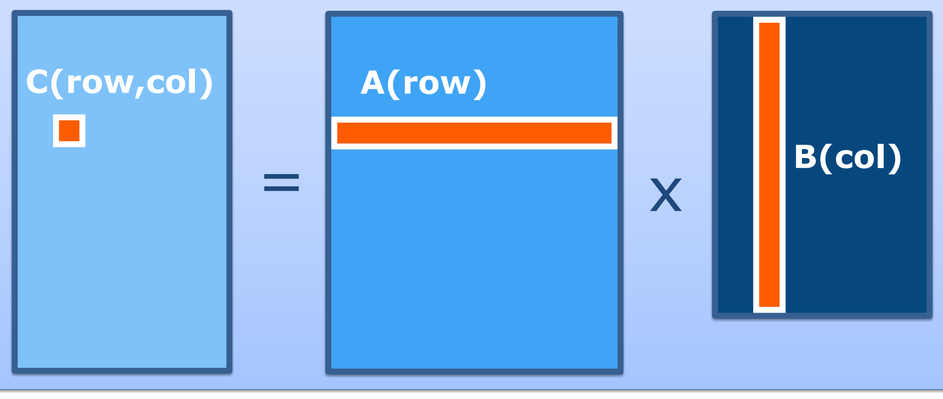
# Accelerating SGEMM with Subgroups

## Algorithm Overview:

Matrix Multiplication: Compute C = A x B

- A = M rows by K cols
- B = K rows by N cols
- C = M rows by N cols




## Naïve Implementation:

Each Work Item computes one element of C:

Inner Loop:

- 2x reads, 1x multiply, 1x add
- Too many reads per multiply and add

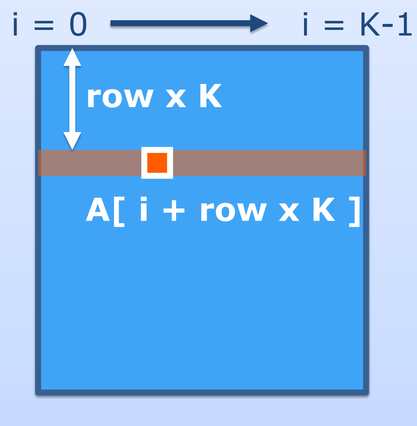
```
__kernel void naive(
    __global float* C,
    __global const float* A,
    __global const float* B,
    int K )
{
    int col = get_global_id(0);
    int row = get_global_id(1);
    int N = get_global_size(0);
    float sum = 0.0f;
    for( int i = 0; i < K; i++ ) {
        sum += A[ i + row * K ] *
               B[ col + i * N ];
    }
    c[ col + row * N ] = sum;
}
```

## Memory Access Patterns:

Assume 1D Work Group:

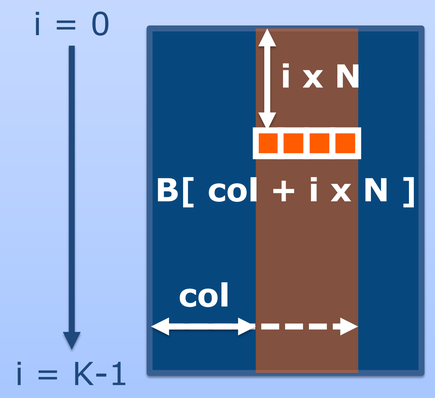
Accesses to Matrix A are "Uniform":

```
int row = get_global_id(1);
...
for( int i = 0; i < K; i++ ) {
    ...
    A[ i + row * K ] *
}
```




Accesses to Matrix B are "Consecutive":

```
int col = get_global_id(0);
...
for( int i = 0; i < K; i++ ) {
    ...
    B[ col + i * N ];
}
```



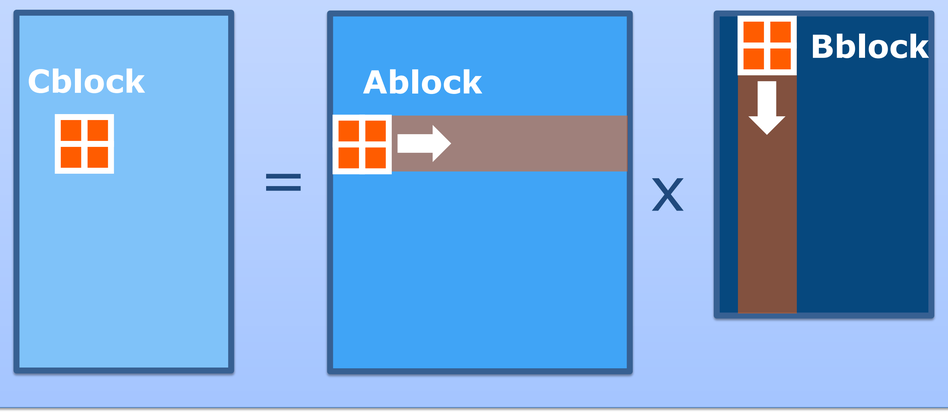

## Block Implementation:

Each Work Group computes a Block of C from Blocks of A and B:

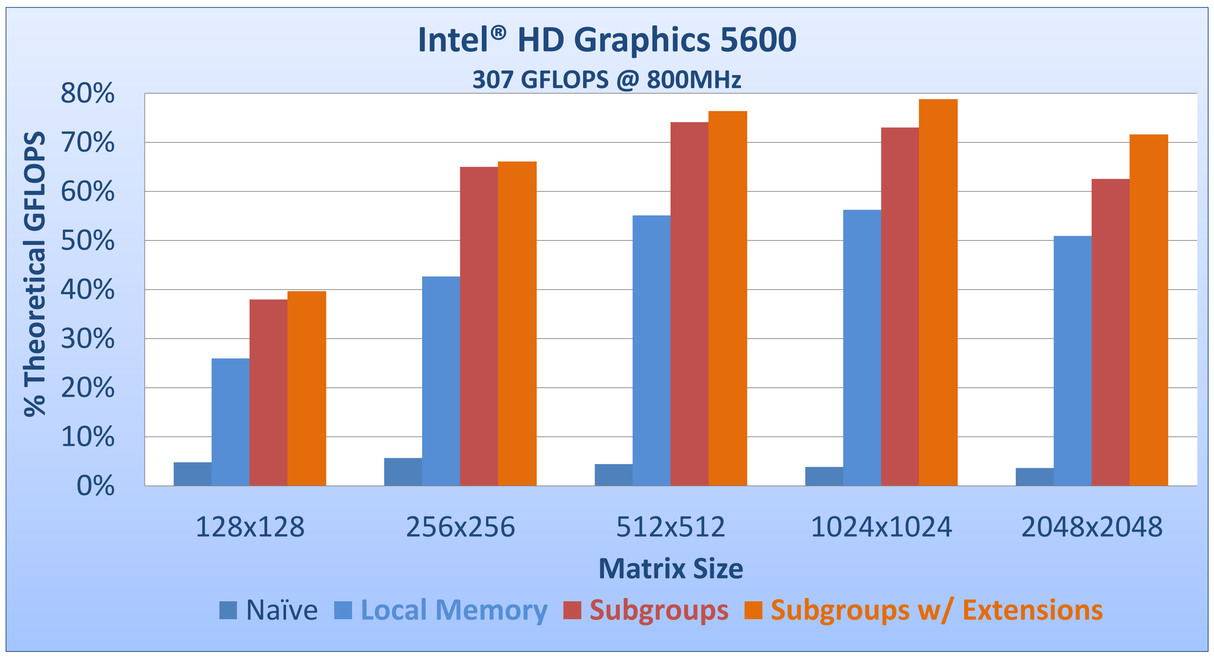
- Blocks enable re-use
- Read and **`broadcast()`** elements of A
- Fewer reads per multiply and add

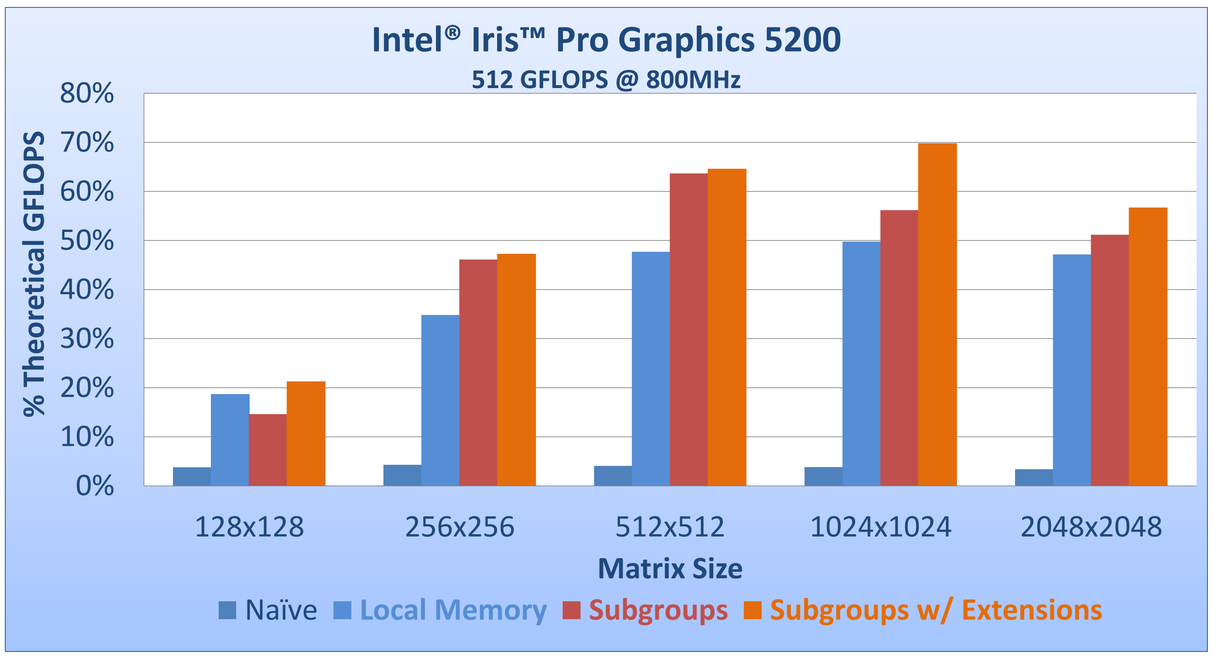
Implementation Choices:

- Use Shared Local Memory (OpenCL 1.2)
- Use Subgroups *(New in OpenCL 2.0!)*
- Use **`cl_intel_subgroups`** *(New in 2015!)*




## Results:




By Ben Ashbaugh, Intel® Corporation

IWOCL 2015

Stanford University, USA | 12-13 May