Matrix Multiplication using OpenCL

Leiming Yu

3-16-2015

Previous Sections

- Introduction to Parallelism
  - CPU vs. GPU
- OpenCL compilation and framework
- OpenCL Case Study: vector add
  - Create objects, e.g., platform, context, program
  - Write kernel
  - Transfer data
  - Profile kernel execution

Current Section

- Matrix Multiplication (MM) Background
- MM on GPU
  - Memory
  - Execution
  - Validation
  - Performance Benchmarking

Matrix Multiplication

"Dot Product"

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \times \begin{bmatrix}7 & 8\\ 9 & 10\\ 11 & 12\end{bmatrix} = \begin{bmatrix}58 & \\ & \end{bmatrix}$$

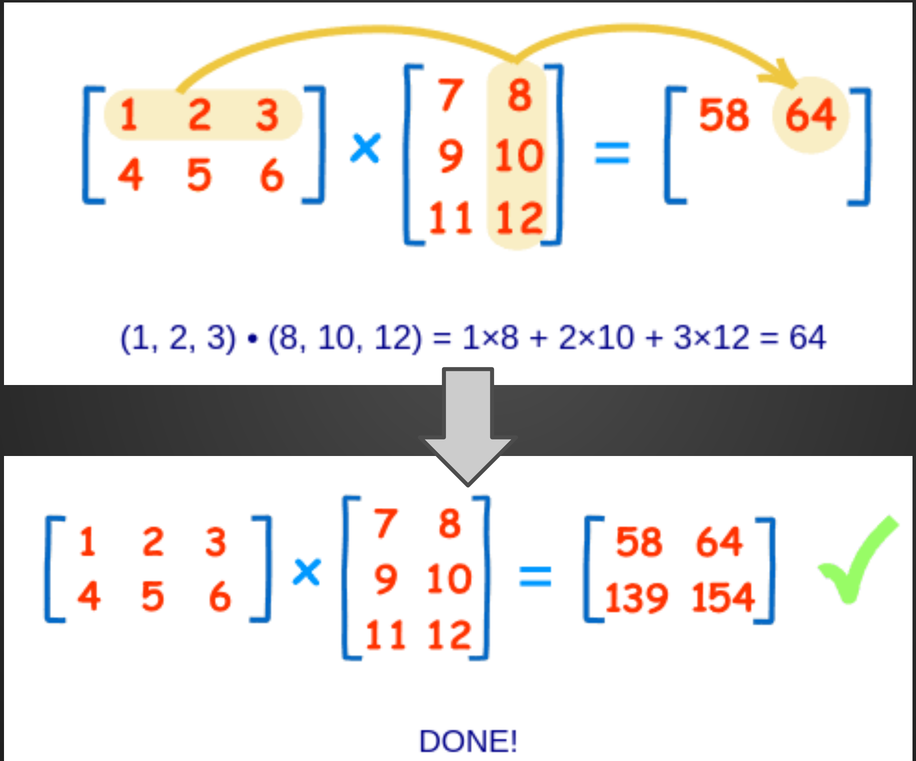
The "Dot Product" is where we **multiply matching members**, then sum up:

$$(1, 2, 3) \bullet (7, 9, 11) = 1\times7 + 2\times9 + 3\times11 = 58$$

We match the 1st members (1 and 7), multiply them, likewise for the 2nd members (2 and 9) and the 3rd members (3 and 11), and finally sum them up.

http://www.mathsisfun.com/algebra/matrix-multiplying.html

Matrix Multiplication



http://www.mathsisfun.com/algebra/matrix-multiplying.html

An Example

Beef pies cost **$3** each

Chicken pies cost **$4** each

Vegetable pies cost **$2** each

	Mon	Tue	Wed	Thu
Beef	**13**	**9**	**7**	**15**
Chicken	**8**	**7**	**4**	**6**
Vegetable	**6**	**4**	**0**	**3**

$$\begin{bmatrix}\$3 & \$4 & \$2\end{bmatrix} \times \begin{bmatrix}13 & 9 & 7 & 15\\ 8 & 7 & 4 & 6\\ 6 & 4 & 0 & 3\end{bmatrix} = \begin{bmatrix}\$83 & \$63 & \$37 & \$75\end{bmatrix}$$

$\$3\times13 + \$4\times8 + \$2\times6$

http://www.mathsisfun.com/algebra/matrix-multiplying.html

More about MM

Order of Multiplication

In arithmetic we are used to:

$$3 \times 5 = 5 \times 3$$

(The Commutative Law of Multiplication)

But this is **not** generally true for matrices (matrix multiplication is **not commutative**):

$$AB \neq BA$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}$$

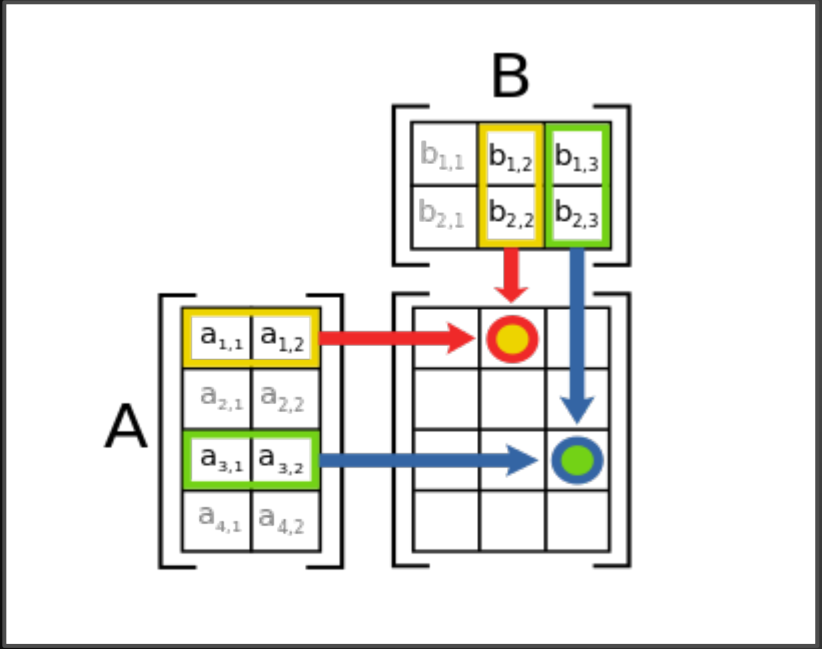
$$\begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 7 & 10 \end{bmatrix}$$

Other interesting terms

- Identity Matrix
- Inverse of a Matrix
- Determinant of a Matrix

http://www.mathsisfun.com/algebra/matrix-multiplying.html

MM on CPU




```
for (i=0;i<N;i++)
   for (j=0;j<N;j++)
      C[i,j] = 0;
      for (k=0;k<N;k++)
         C[i,j] += A[i,k]*B[k,j];
```

Code Exercise

- CPU implementation of MM

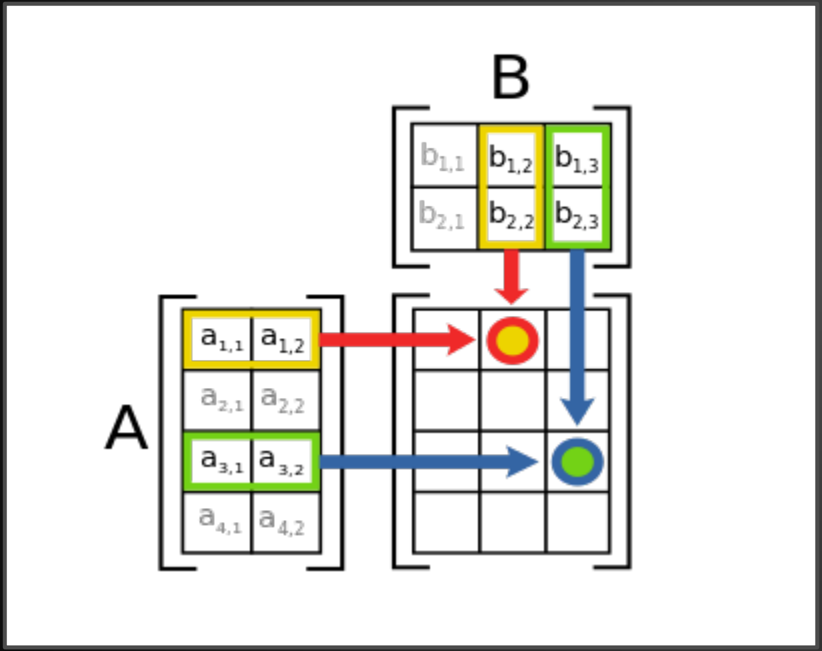
Naive MM on GPU

- Launch 2D Grid
- Customize the 2D workgroup size
- Store data in global/device memory
- Each thread computes one element of the output matrix

Code Exercise

- CPU implementation of MM
- Naive MM implementation on GPU

Problem with naive version



$A_{1,1}\ A_{1,2}\ B_{1,1}\ B_{2,1} \Rightarrow C_{1,1}$

$A_{1,1}\ A_{1,2}\ B_{1,2}\ B_{2,2} \Rightarrow C_{1,2}$

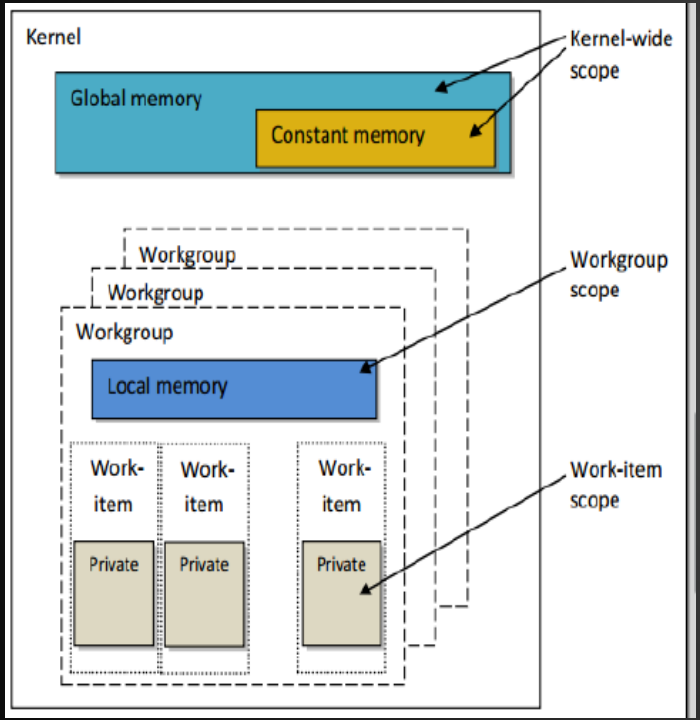
$A_{1,1}\ A_{1,2}\ B_{1,3}\ B_{2,3} \Rightarrow C_{1,3}$

$A_{2,1}\ A_{2,2}\ B_{1,1}\ B_{2,1} \Rightarrow C_{2,1}$

$\vdots \qquad \vdots$

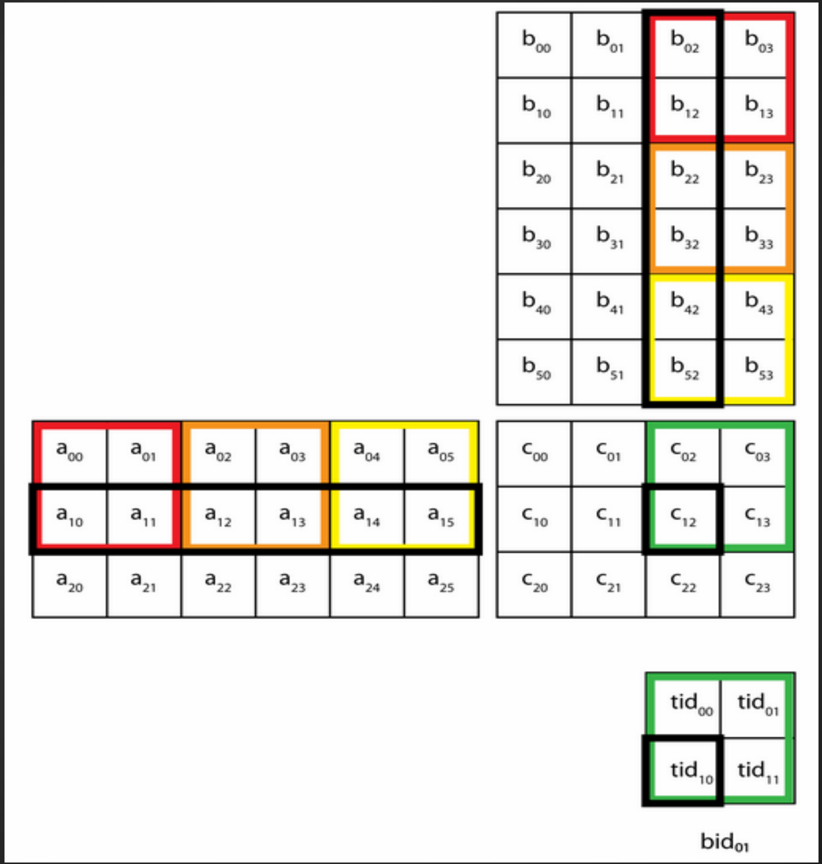
Data reuse

Solution



- Local memory for data reuse
- Chunk the matrix into sub-matrices for parallel computation and good data locality

Scheme



b00 b01 b02 b03
b10 b11 b12 b13
b20 b21 b22 b23
b30 b31 b32 b33
b40 b41 b42 b43
b50 b51 b52 b53
a00 a01 a02 a03 a04 a05
a10 a11 a12 a13 a14 a15
a20 a21 a22 a23 a24 a25
c00 c01 c02 c03
c10 c11 c12 c13
c20 c21 c22 c23
tid00 tid01
tid10 tid11
bid01

Code Exercise

- CPU implementation of MM
- Naive MM implementation on GPU
- Optimized MM
    - Tiled computation
    - Local Memory

Congratulations!

You are leveled up ^~^