Talk in
Parallel Computing

RheinAhr Campus

OpenCL

A brief introduction for NVIDIA CUDA programmers

Jan Beneke

16.01.2012

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Motivation
Advantages of GPGPU

- Runs parallelized code on GPUs
- Speeds up existing applications
- Uses ordinary and „cheap“ hardware accessible to almost every user
- Even supported by modern handheld devices
- Easy to code with dedicated language extensions and frameworks
- But: Difficult to create efficient code

Motivation
NVIDIA CUDA

Pro
- Easy to learn language extension for using GPGPU
- Intuitive framework
- Many extensions provided by NVIDIA (CUFFT, CUBLAS, ...)
- Integrated graphics language support (OpenGL, Direct 3D)
- Cross-platform (Microsoft Windows, Linux, Apple Mac OS X)
- Free of charge

Contra
- Only NVIDIA hardware suported
- Only GPUs supported
- Proprietary product/ Closed source

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Introducing OpenCL
Open Computing Language

- Language extension for accessing heterogeneous computational devices

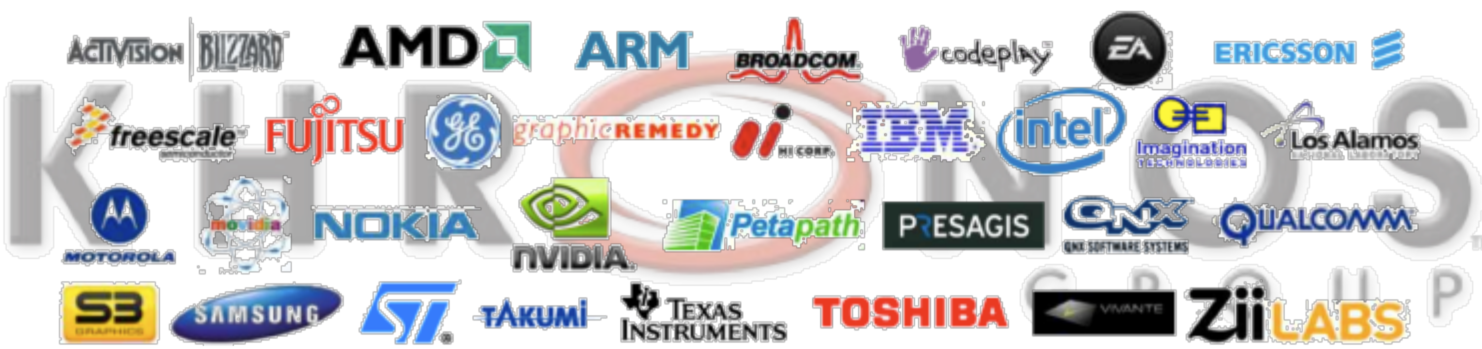
- Supportes parallel execution on single or multiple devices of many vendors
  - GPUs (NVIDIA, AMD/ ATI, ...)
  - CPUs (Intel, AMD, ...)
  - Accelerator cards/ Server blades (IBM, ...)

- Desktop, server farm and hand-held profiles

- Integrates OpenGL and Microsoft Direct 3D graphics APIs

Introducing OpenCL
An open standard

- Vendor neutral
- Specifications under review by Khronos OpenCL working group
  - Krohnos also reviews standards like OpenGL, WebGL, Open AL, ...
- Standard based on proposal by Apple and developed with industry leaders



Source: http://www.khronos.org

- First released and natively integrated in Mac OS X since Snow Leopard (2009)

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Design Goals
Heterogeneous World

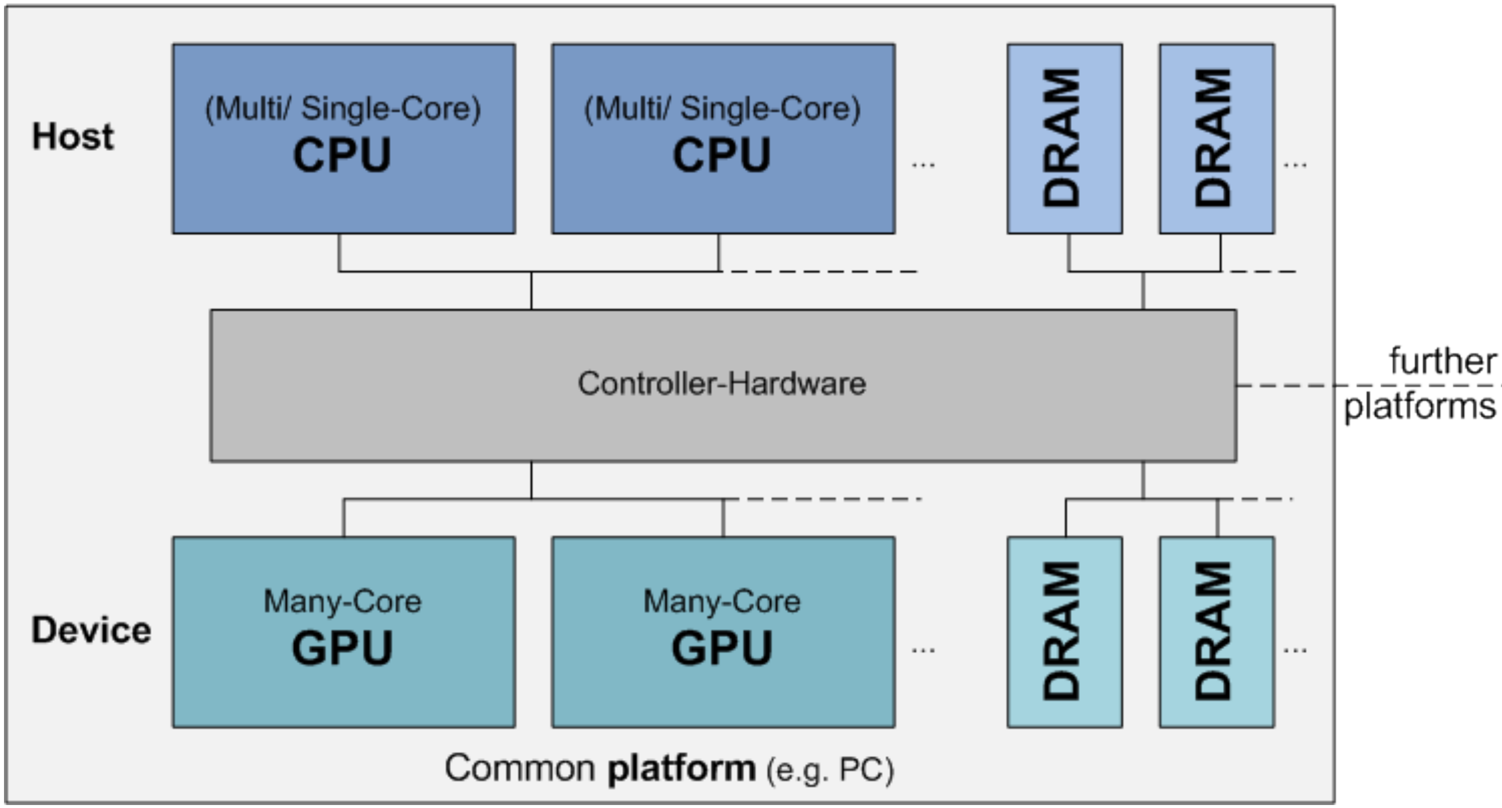
- A modern platform includes:
  - One or more CPUs
  - One or more GPUs
  - (DSPs)
  - (Accelerator cards)

➔ Use them as peers.

➔ <u>One</u> portable program uses <u>all</u> available resources.

Design Goals
Heterogeneous World

- A common platform (PC or server blade):



Design Goals
Becoming a useable standard

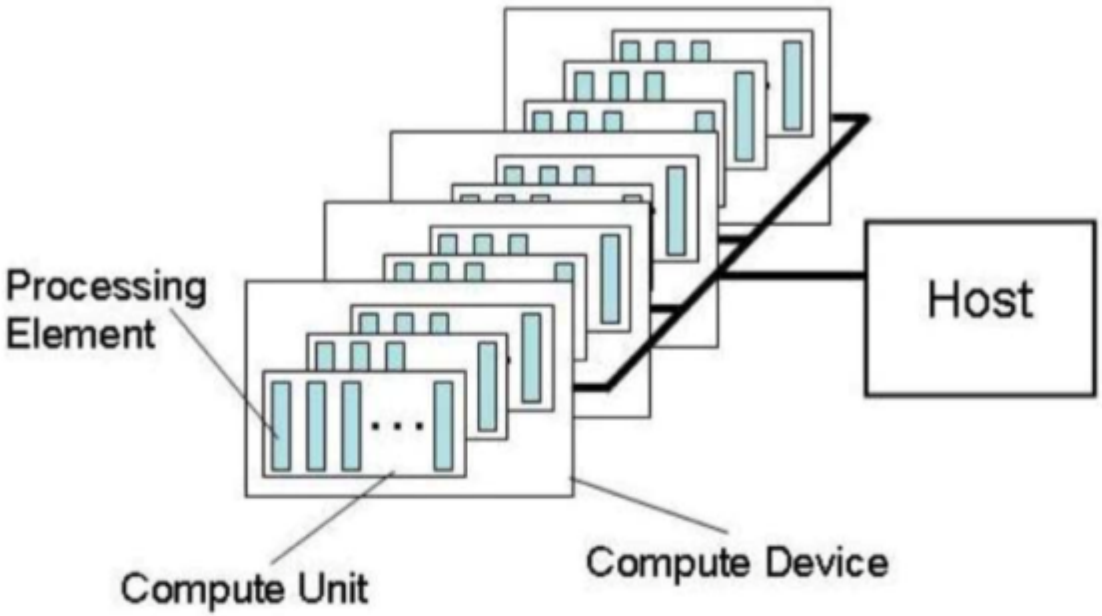
- Efficent parallel model
  - Based on ISO C99
  - Abstract the underlaying hardware
  - Additional work-items and workgroups, vector types, synchronization, address space qualifiers
  - Built-in functions for image manipulation, work-item manipulation, math routines, ...
- Specify accuracy of floating-point computations (IEEE 754)
- Support of most future hardware

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Hardware Model
Abstract Platform Model

- One host and multiple computing devices
  - Each device contains computing units
    - Each unit is devided into processing elements

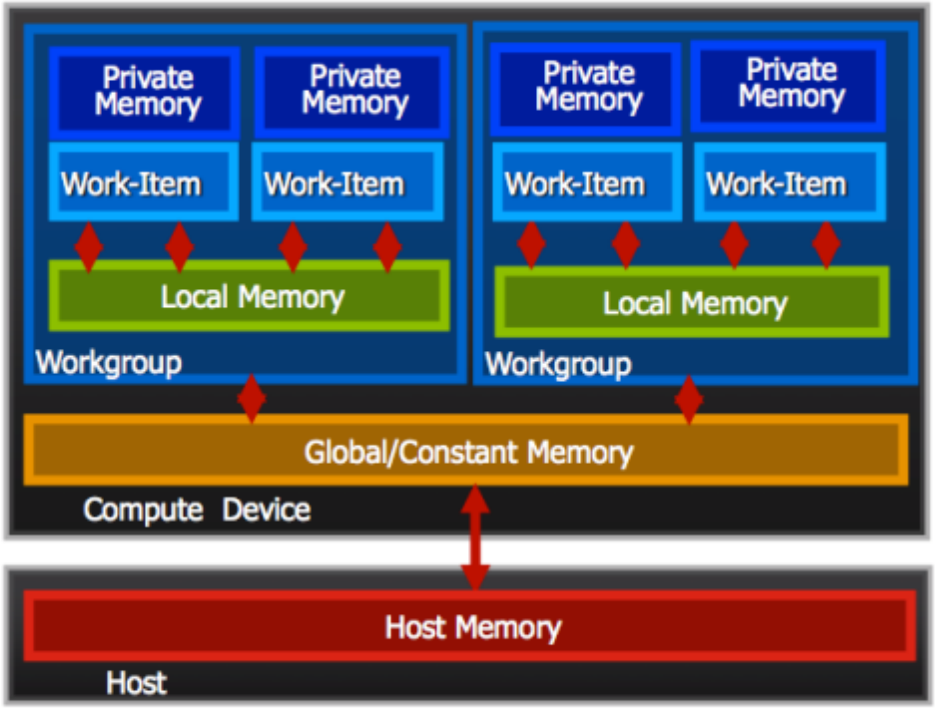


Source: http://www.khronos.org

Hardware Model
Abstract Memory Model

- Hardware is abstracted by a consitend shared memory model (like CUDA):

- **Private Memory**
  - Per work-item
- **Local Memory**
  - Shared in a workgroup
- **Global/ Constant Memory**
  - All workgroups on one device
- **Host Memory**
  - On host level (CPU)



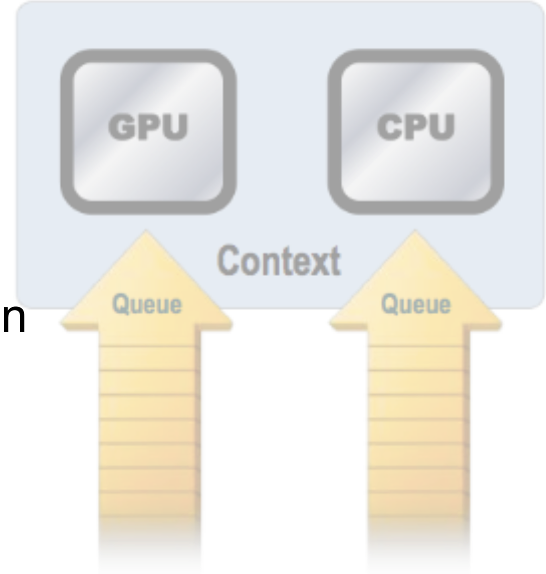
Source: http://www.khronos.org

- Independent of based hardware.

- Explicit memory model:
  - Mostly move data from host ➔ global ➔ local ➔ ... and back

Execution Model
The OpenCL application

- The application runs on a host, that submits work to the compute devices

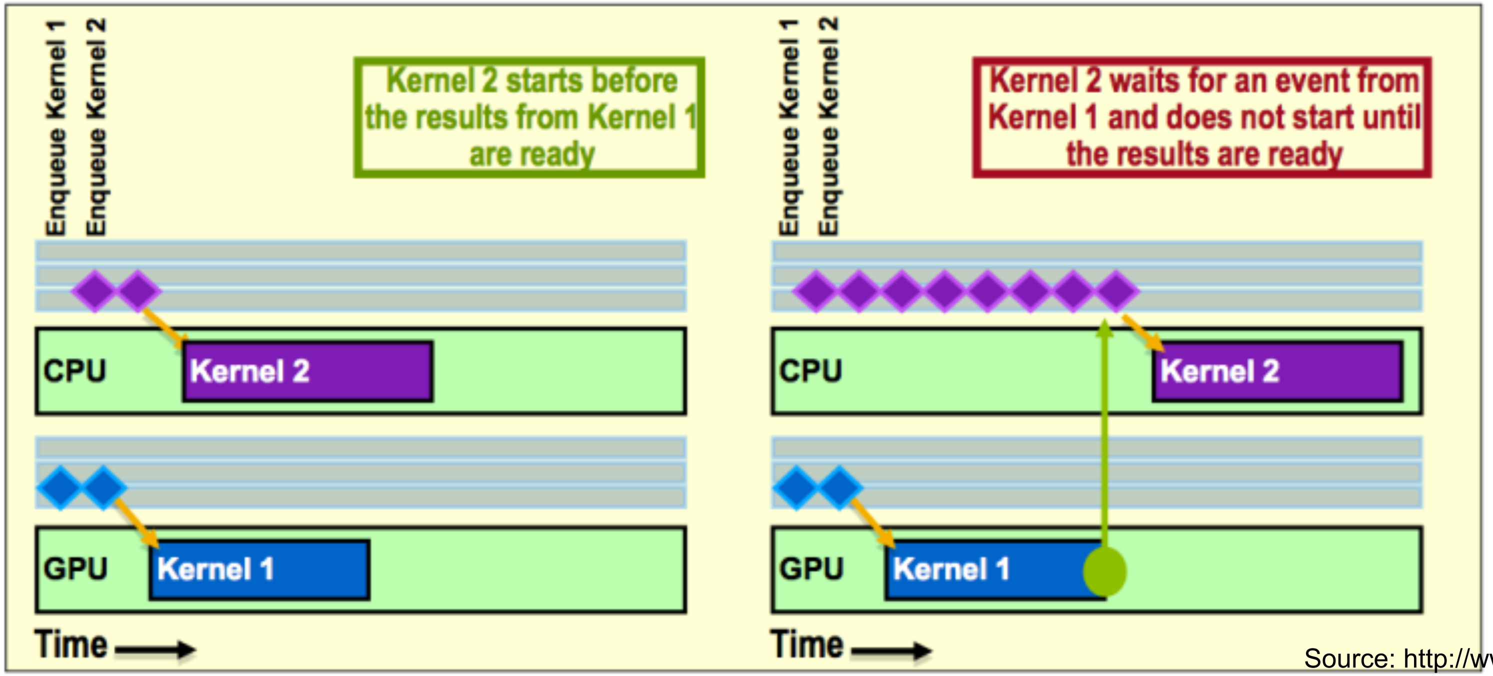


  - Work-item
    - Basic unit on OpenCL device
  - Kernel
    - The code for a work item - an extended C function
  - Command Queue
    - Queues Kernels/ synchronizing - event handling
  - Program
    - Collection of kernels and other functions - analogous to a dynamic library
  - Context
    - The environment to execute the work-items in, inculdes devices and their memories and command queues

Source: http://www.khronos.org

Execution Model
Synchronization

- Applications queue compute kernel execution instances
- Events can be used to synchronize kernel executions between queues
- Example: 2 queues on seperate devices



Source: http://www.khronos.org

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Software Stack

- Platform layer
  - Query and select computing devices
  - Initialize devices
  - Create work contexts and command queues

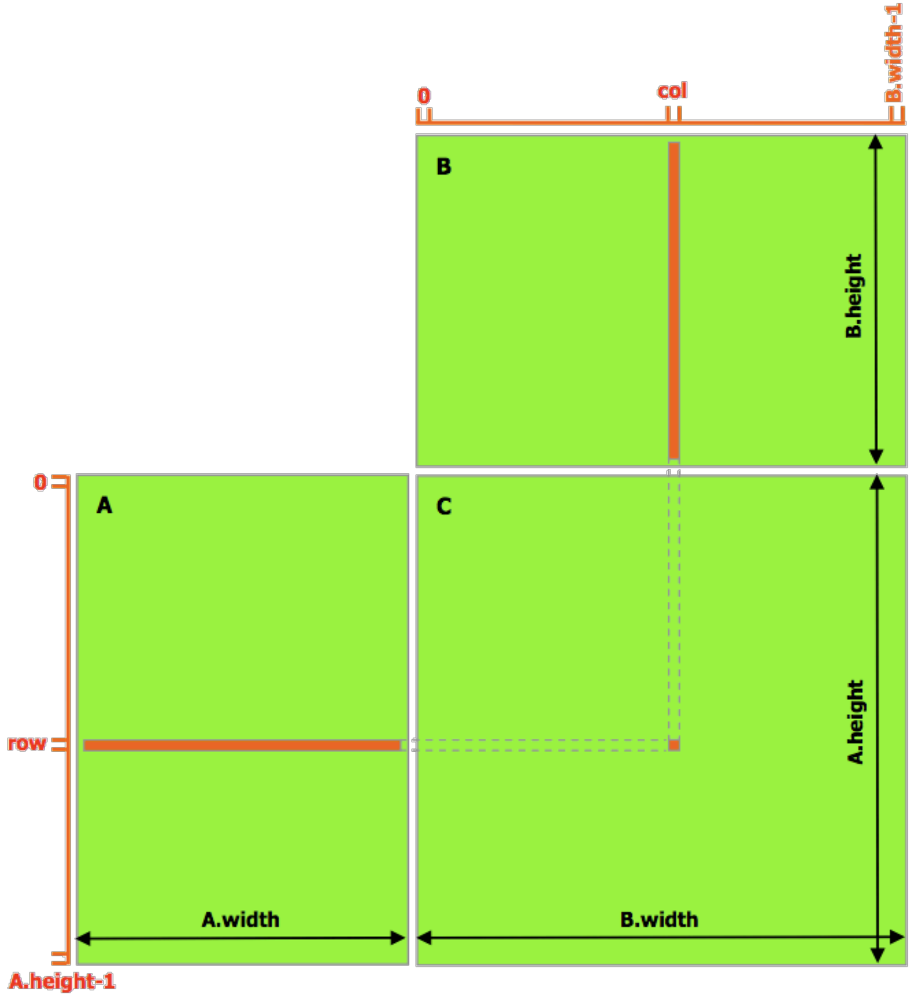
- Runtime
  - Ressource management
  - Execute kernels

- Compiler
  - Compile and build compute program executable (everything but the host code) at runtime

Outline

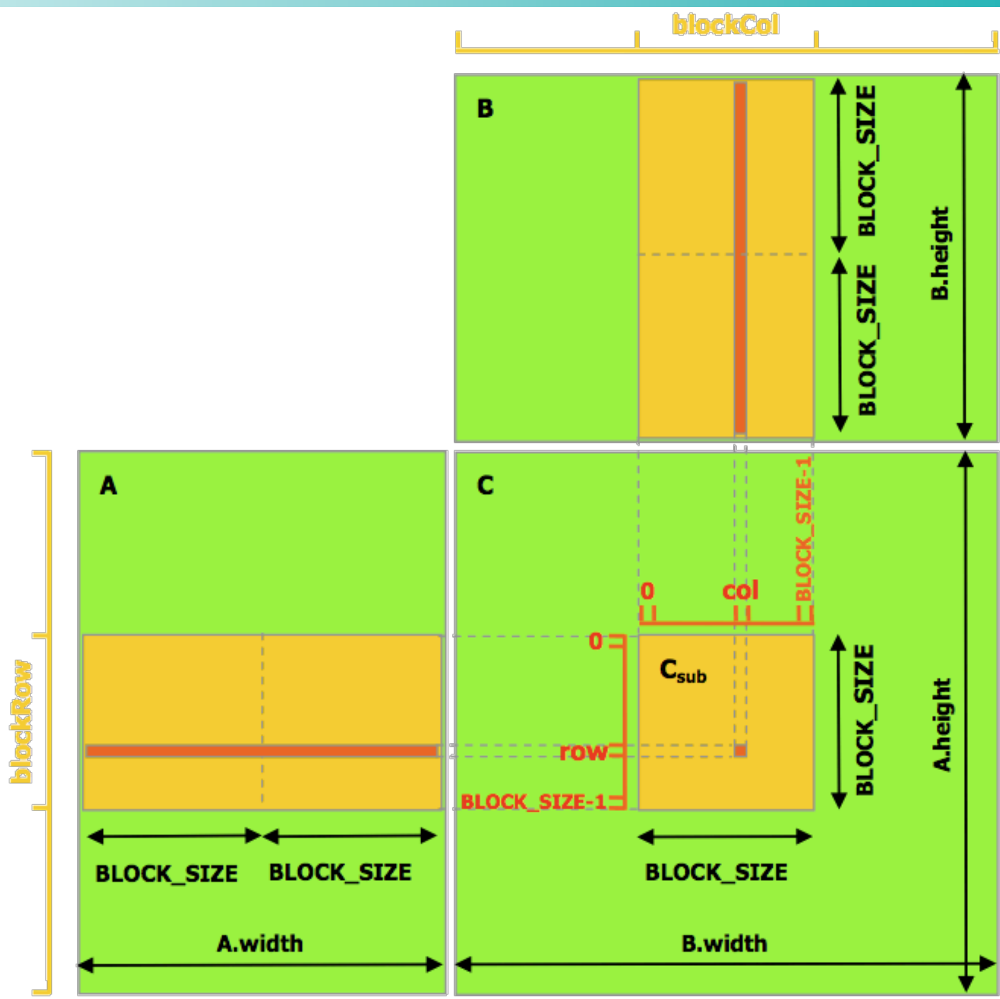
- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Known Example
Matrix Multiplication



Source: http://www.nvidia.com

Known Example
Matrix Multiplication



Source: http://www.nvidia.com

Known Example Matrix Multiplication

- Code is devided into
  - Host API Code (matrixMul.cpp)
  - OpenCL Kernel Code (matrixMul.cl)

- Kernel code will be familiar to CUDA programmers

Source: http://www.khronos.org

Known Example
Compute Kernel Code

```
// Workgroup (thread block) dimensions
#define BLOCK_SIZE 16

// Fast element access
#define AS( i, j) As[ j + i * BLOCK_SIZE]
#define BS( i, j) Bs[ j + i * BLOCK_SIZE]
```

matrixMul.cl

Known Example
Compute Kernel Code

```
__kernel void matrixMul(
        __global float* C, __global float* A, __global float* B,
        __local float* As, __local float* Bs, int widthA, int widthB)
{
    int bx = get_group_id( 0);                 // Workgroup (block) index
    int by = get_group_id( 1);                 // Workgroup (block) index

    int tx = get_local_id( 0);                 // Work-item (thread) index
    int ty = get_local_id( 1);                 // Work-item (thread) index

    int aBegin = widthA * BLOCK_SIZE * by; // Sub-matrix A start
    int aEnd   = aBegin + widthA - 1;      // Sub-matrix A end
    int aStep  = BLOCK_SIZE;               // Sub-matrix A stepsize

    int bBegin = BLOCK_SIZE * bx;          // Sub-matrix B start
    int bStep  = BLOCK_SIZE * widthB;      // Sub-matrix B end


    float Csub = 0.0f;                     // Result of block

    // . . .
```

matrixMul.cl

Known Example
Compute Kernel Code

```
    // . . .
    // Loop over A and B to compute the block sub-matrix
    for( int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep)
    {
        // Each thread loads one element of the mat. from device to shared memory
        AS( ty, tx) = A[ a + widthA * ty + tx];
        BS( ty, tx) = B[ b + widthB * ty + tx];

        // Synchronize to make sure the matrices are loaded
        barrier( CLK_LOCAL_MEM_FENCE);

        // Multiply the two matrices
        #pragma unroll
        for( int k = 0; k < BLOCK_SIZE; ++k)
            Csub += AS( ty, k) * BS( k, tx);

        // Synchronize to make sure that the preceding
        barrier( CLK_LOCAL_MEM_FENCE);
    }
    // Write the block sub-matrix to device memory
    C[ get_global_id( 1) * get_global_size( 0) + get_global_id( 0)] = Csub;
}
```

matrixMul.cl

Known Example
Host API code

```
#include <CL/opencl.h>


// Matrix dimensions
int WA = 32 * BLOCK_SIZE;          // Input matrix A width
int HA = 128 * BLOCK_SIZE;         // Input matrix A height
int WB = 32 * BLOCK_SIZE;          // Input matrix B width
int HB = WA;                       // Input matrix B height
int WC = WB;                       // Resulting matrix C width
int HC = HA;                       // Resulting matrix C height

int main( int argc, char** argv)
{
    cl_context GPUContext;                  // Device context handle
    cl_kernel matMulKernel;                 // Kernel handle
    cl_command_queue commandQueue;          // Command queue handle
    cl_program program;                     // Program handle
    cl_device_id deviceID = NULL;           // IDs of all OpenCL devices

    // . . .
```

matrixMul.cpp

Known Example
Host API code

```
// Allocate memory for matrizes on host level
float* hA = (float*) malloc( WA * HA * sizeof(float));
float* hB = (float*) malloc( WB * HB * sizeof(float));
float* hC = (float*) malloc( WC * HC * sizeof(float));

// Fill arrays with random numbers
for(int i = 0; i < WA * HA; i++)
    hA[i] = rand() / (float) RAND_MAX;

for(int i = 0; i < WB * HB; i++)
    hB[i] = rand() / (float) RAND_MAX;

// Get OpenCL devices
clGetDeviceIDs( NULL, CL_DEVICE_TYPE_GPU, 1, &deviceID, NULL);

// Create OpenCL context
GPUContext = clCreateContext( 0, 1, &deviceID, NULL, NULL, NULL);

// Create command queue
commandQueue = clCreateCommandQueue( GPUContext, deviceID,
                                     CL_QUEUE_PROFILING_ENABLE, NULL);
// . . .
```

matrixMul.cpp

Known Example
Host API code

```
// Load OpenCL kernel source code from source file
size_t programLength;
char* source = loadProgSource( SOURCE_PATH, "", &programLength);

// Create OpenCL program
program = clCreateProgramWithSource( GPUContext, 1, (const char**) &source,
                      &programLength, NULL);

// Build OpenCL program
clBuildProgram( program, 0, NULL, "", NULL, NULL);

// Create OpenCL kernel
matMulKernel = clCreateKernel( program, "matrixMul", NULL);


// . . .
```

matrixMul.cpp

Known Example
Host API code

```
// Create OpenCL buffer pointing to the host memory
cl_mem hABuffer = clCreateBuffer( GPUContext, CL_MEM_READ_ONLY |
                      CL_MEM_USE_HOST_PTR, WA*HA*sizeof(float), hA, NULL);

// Create buffers (memory on device) for matrizes
cl_mem dA = clCreateBuffer( GPUContext, CL_MEM_READ_ONLY,
                      WA*HA*sizeof(float), NULL, NULL);

// Copy from host to device, now
clEnqueueCopyBuffer( commandQueue, hABuffer, dA, 0, 0, WA*HA*sizeof(float),
                      0, NULL, NULL);

// Create buffer on device, it will be initialized from the host at first use
cl_mem dB = clCreateBuffer( GPUContext, CL_MEM_READ_ONLY|
                      CL_MEM_COPY_HOST_PTR, WB*HB*sizeof(float), hB, NULL);

// Create buffer for resulting matrix
cl_mem dC = clCreateBuffer( GPUContext, CL_MEM_WRITE_ONLY,
                      WC*HA*sizeof(float), NULL,NULL);

// . . .
```

matrixMul.cpp

Known Example
Host API code

```
// Set the argument values for the kernel
clSetKernelArg( matMulKernel, 0, sizeof(cl_mem), (void *) &dC);
clSetKernelArg( matMulKernel, 1, sizeof(cl_mem), (void *) &dA);
clSetKernelArg( matMulKernel, 2, sizeof(cl_mem), (void *) &dB);
clSetKernelArg( matMulKernel, 3, BLOCK_SIZE*BLOCK_SIZE*sizeof(float), 0);
clSetKernelArg( matMulKernel, 4, BLOCK_SIZE*BLOCK_SIZE*sizeof(float), 0);
clSetKernelArg( matMulKernel, 5, sizeof(cl_int), (void *) &WA);
clSetKernelArg( matMulKernel, 6, sizeof(cl_int), (void *) &WB);

// Execute Multiplication in parallel
size_t localWorkSize[]  = { BLOCK_SIZE, BLOCK_SIZE};
size_t globalWorkSize[] = { WC, HA};

// Launch kernel
// Multiplication - non-blocking execution:  launch and push to device
    clEnqueueNDRangeKernel( commandQueue, matMulKernel, 2, 0, globalWorkSize,
             localWorkSize, 0, NULL, NULL);

// Sync to host
clFinish( commandQueue);

// . . .
```

matrixMul.cpp

Known Example
Host API code

```
    // Non-blocking copy of result from device to host
    clEnqueueReadBuffer( commandQueue, dC, CL_FALSE, 0, WC*HA*sizeof(float),
                         hC, 0, NULL, NULL);

    // Release mem objects (device memory)
    clReleaseMemObject( hABuffer);
    clReleaseMemObject( dA);
    clReleaseMemObject( dC);
    clReleaseMemObject( dB);

    // Clean up OpenCL resources
    clReleaseKernel( matMulKernel);
    clReleaseCommandQueue( commandQueue);
    clReleaseProgram( program);
    clReleaseContext( GPUContext);

    // Clean up host memory
    free( hA); free( hB); free( hC);

    // Exit application
    return( 0);
}
```

matrixMul.cpp

Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Benchmark
Test System

Intel Core i3-2100

2 cores (2 Threads each) @ 3.1 GHz - 3MB cache - 8GB DDR3 (1333MHz)
max. $52\ GFLOPS/s$ (LINPACK)

MSI N560GTS-Ti Twin Frozr II/OC

NVIDIA GeForce GTX 560 Ti – 384 cores @ 880Mhz – 1024MB GDDR5 (4200MHz)
max. $1.253\ GFLOPS/s$ (approximated)

Compiled with Microsoft Visual Studio 2008 Professional x86 under Microsoft Windows 7 Professional SP1 x64 against

- OpenCL 1.2 (NVIDIA GPU Computing SDK 4.0 x86)
- NVIDIA CUDA 4.0 x86
- NVIDIA CUBLAS 2.0 (NVIDIA GPU Computing SDK 4.0 x86)
- OpenMP 2.0 (Microsoft Visual Studio 2008 x86)

Benchmark
Matrix Multiplication

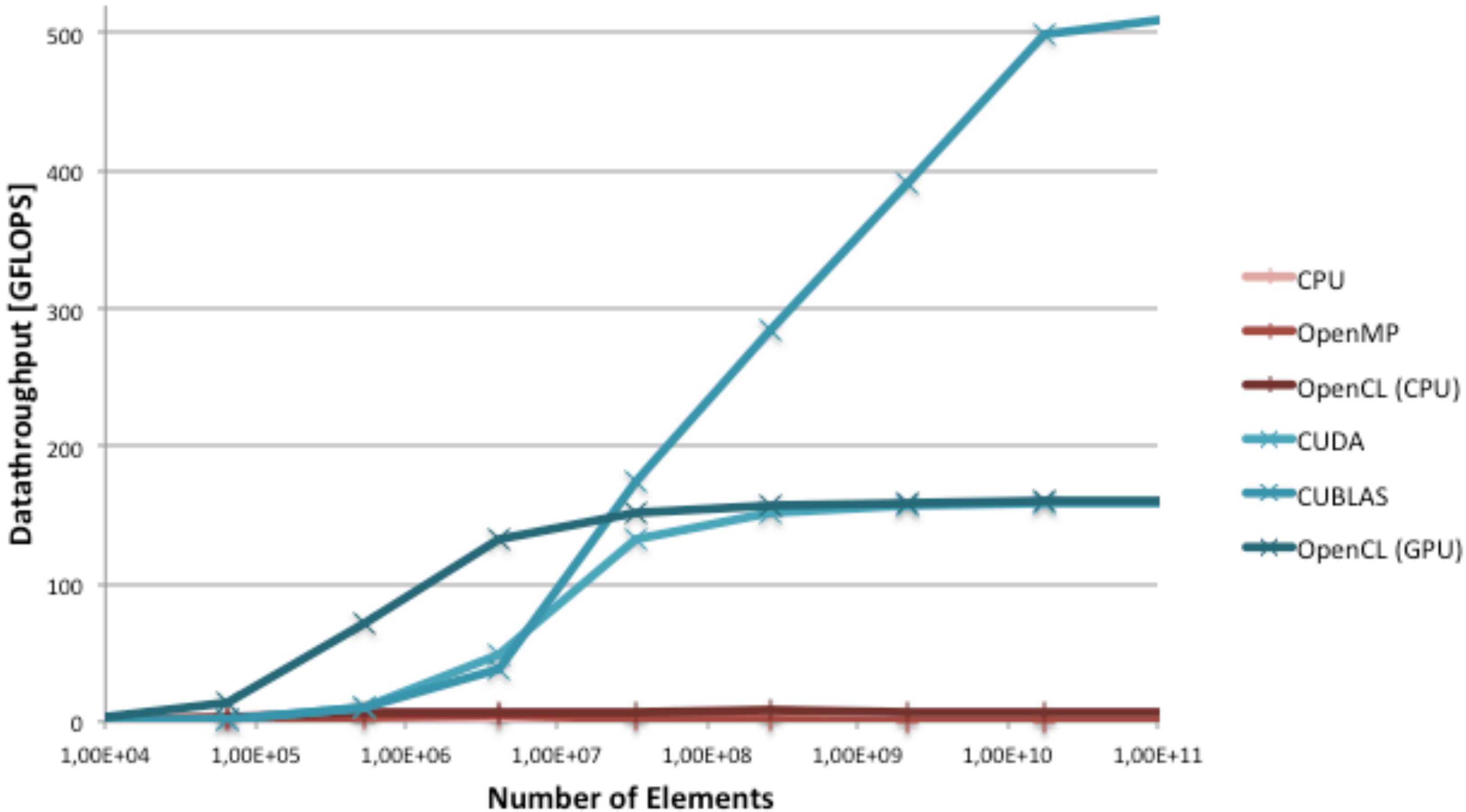
$$A^{2048\times 8192}\cdot B^{4096\times 2048}=C^{4096\times 8196}$$

$$\Rightarrow 137.4\cdot 10^{9}\ \text{elements to compute}$$

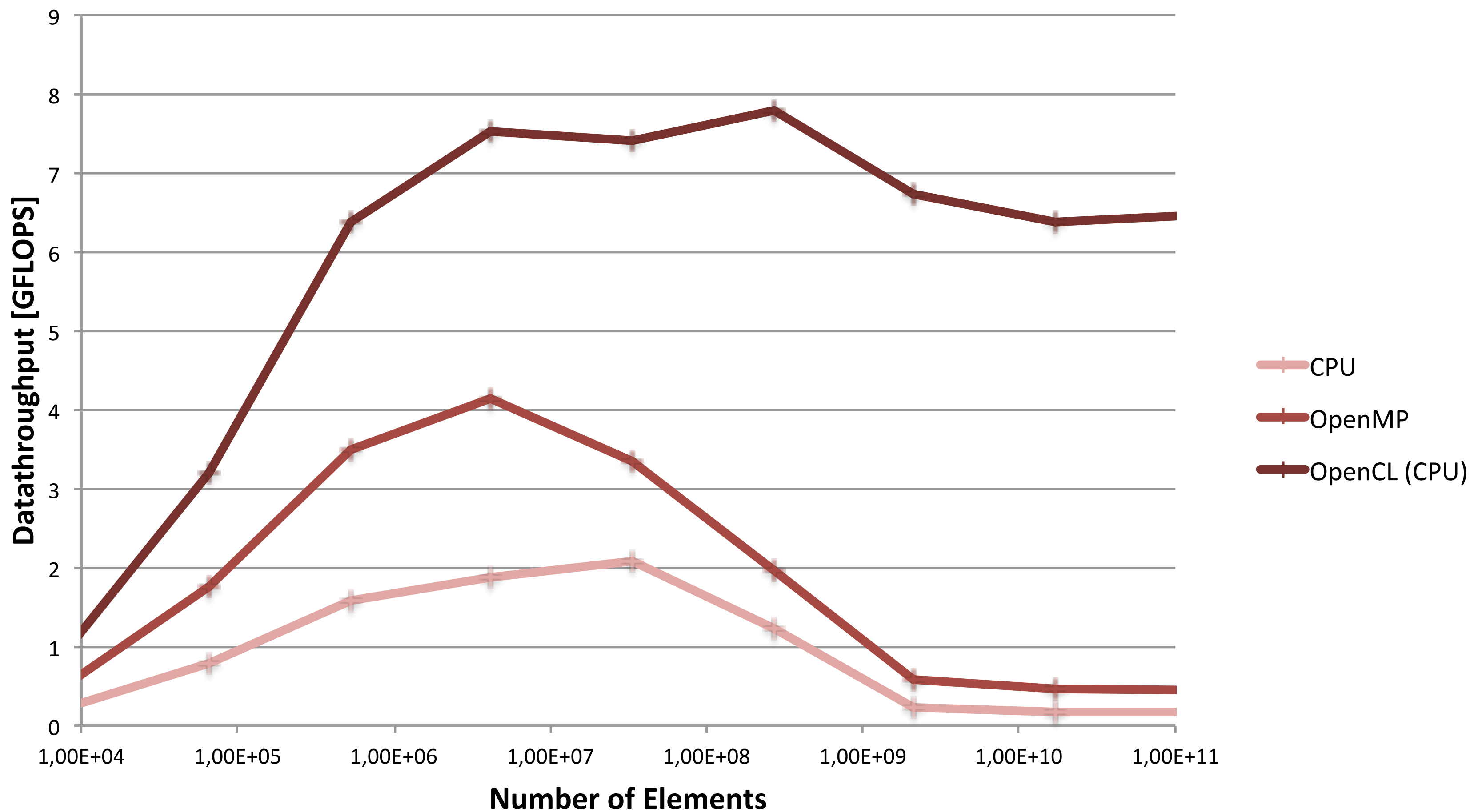
		Execution Time [s]	**Throughput [$GFLOPS$ / s]**
CPU	Single threaded	818.13	0.16
	OpenMP	460.05	0.30
	OpenCL 1.2	**21.76**	**6.32**
GPU	NVIDIA CUDA 4.0	**0.87**	**158.23**
	OpenCL 1.2	**0.86**	**159.72**
	NVIDIA CUBLAS 2.0	0.24	580.90

Average after 100 iterations

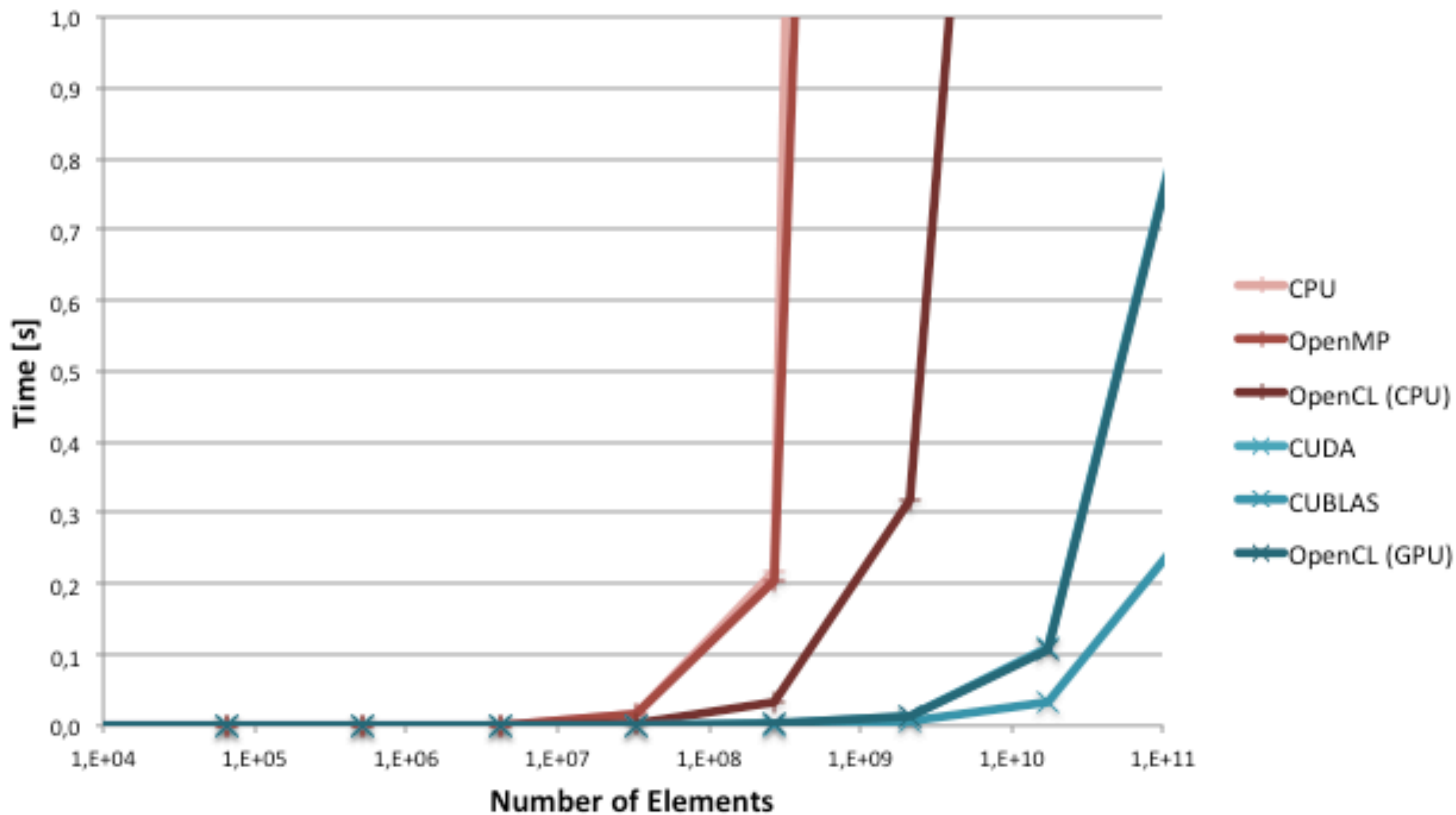
Benchmark
Matrix Multiplication



Benchmark
Matrix Multiplication



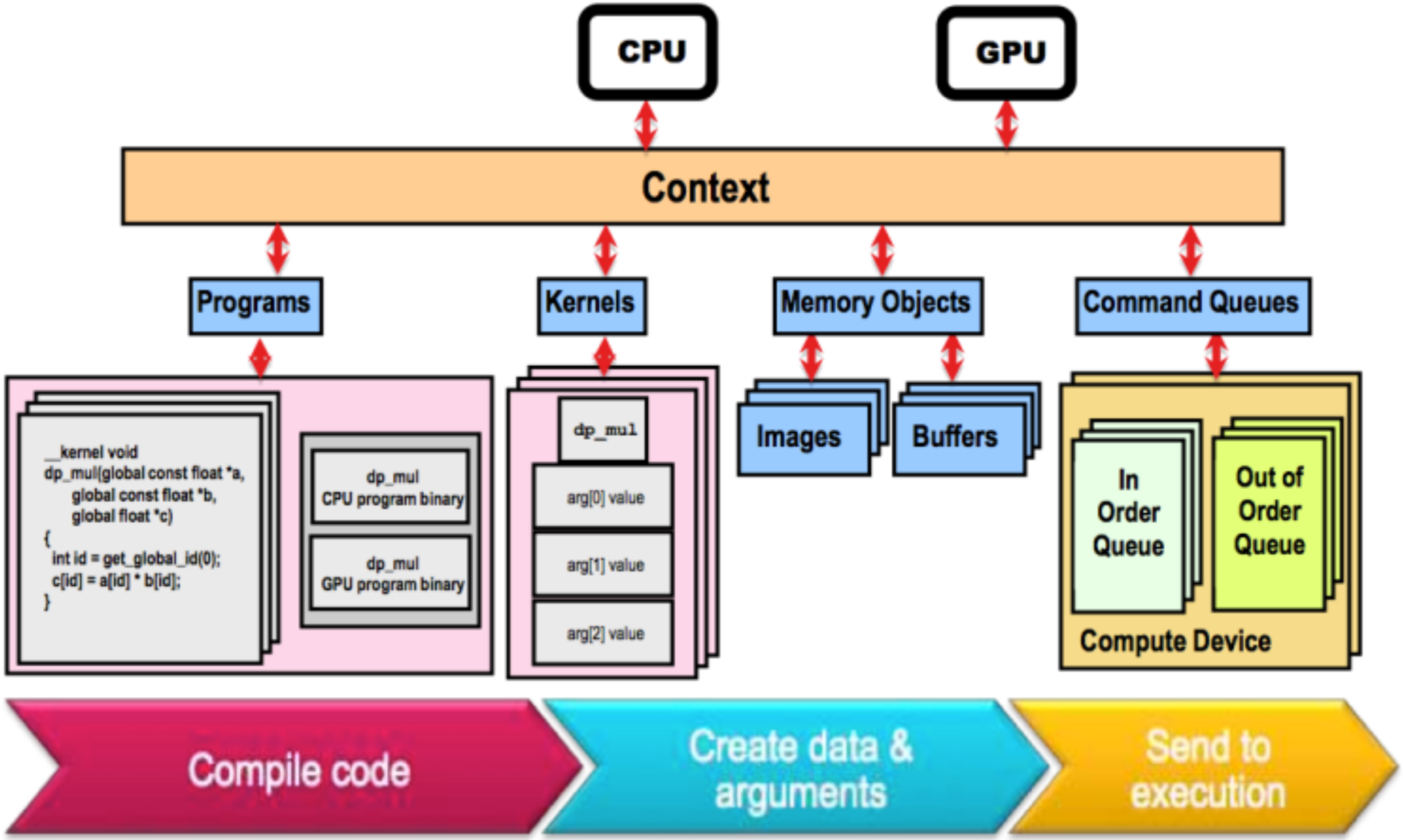
Benchmark Matrix Multiplication



Outline

- Motivation
- Introducing OpenCL
- Design Goals
- Hardware/ Execution Model
- Software Stack
- Known Example
- Benchmark
- Summary

Summary
The OpenCL workflow



Source: http://www.khronos.org

Summary
OpenCL vs. CUDA

	OpenCL 1.2	NVIDIA CUDA 4.0
Free of charge	✓	✓
Vendor independent/ Multiple SDK-vendors	✓	
Open standard	✓	
Use different types of heterogeneous hardware	✓	
Cross-platform	✓	✓
Interoperability with graphics libraries	✓	✓
Included libraries for common tasts		✓
Peer-to-peer-comunication between devices		✓
Global virtual addressing space		✓

Summary
Conclusion

OpenCL:

- More complicated to code and redistribute
- Using different types of devices
- Vendor independend
- Outstanding performance on many CPUs

NVIDIA CUDA:

- More enhanced
- Already widely spread
- Deliveres well-engineered libraries for many tasks

Thank you for your attention.

Source code at http://janbeneke.de/ParComp

References

- Khronos Group khronos.org/opencl/
- NVIDIA developer.nvidia.com/opencl/
- Intel software.intel.com/en-us/articles/vcsource-tools-opencl-sdk/
- AMD/ ATI developer.amd.com/zones/OpenCLZone/