OpenCL consists of two parts: a C/C++-callable API and a C-ish programming language.

The OpenCL programming language can run on NVIDIA GPUs, AMD GPUs, Intel CPUs, Intel GPUs, mobile devices, and (supposedly) FPGAs (Field-Programmable Gate Arrays).

But, OpenCL is at its best on compute devices with large amounts of data parallelism, which usually implies GPU usage.

You break your computational problem up into lots and lots of small pieces. Each piece gets farmed out to threads on the GPU.

Each thread wakes up and is able to ask questions about where it lives in the entire collection of (thousands of) threads. From that, it can tell what it is supposed to be working on.

OpenCL can share data, and interoperate, with OpenGL.

There is a JavaScript implementation of OpenCL, called WebCL.

There is a JavaScript implementation of OpenGL, called WebGL.

WebCL can share data, and interoperate, with WebGL.

The GPU does not have a stack, and so the OpenCL C-ish programming language cannot do recursion and cannot make function calls. It also can’t use pointers.
OpenCL wants you to break the problem up into Pieces

```c
void ArrayMult( int n, float *a, float *b, float *c)
{
    for ( int i = 0;  i < n;  i++ )
        c[i] = a[i] * b[i];
}
```

If you were writing in C/C++, you would say:

```c
kernel void ArrayMult( global float *dA, global float *dB, global float *dC)
{
    int gid = get_global_id ( 0 );
}
```

Think of this as having an implied for-loop around it, looping through all possible values of gid.

OpenCL also supports Vector Parallelism

- An OpenCL program can be vector-oriented, meaning that it can perform a single instruction on multiple data values at the same time (SIMD).
- Vector data types are: char, int, float, where n = 2, 4, 8, or 16.

```c
float4 f, g;
f = (float4)( 1.f, 2.f, 3.f, 4.f );
float16 a16, x16, y16, z16;
f.x = 0.;
f.xy = g.zw;
x16.s89ab = f;
float16  a16  =  x16 * y16  +  z16;
```

(Note: just because the language supports it, doesn’t mean the hardware does.)

Computes Units and Processing Elements are Arranged in Grids

- A GPU Device is organized as a grid of Compute Units.
- Each Compute Unit is organized as a grid of Processing Elements.
- So in NVIDIA terms, their Turing GPU has 68 Compute Units, each of which has 64 Processing Elements, for a grand total of 4,352 Processing Elements.

Work-Groups are Arranged in Grids

- The GPU’s workload is divided into a Grid of Work-Groups.
- Each Block’s workload is divided into a Grid of Work-Items.

OpenCL Software Terminology:

- Work-Groups and Work-Items are Arranged in Grids.
- An OpenCL program is organized as a grid of Work-Groups.
- Each Work-Group is organized as a grid of Work-Items.
- In terms of hardware, a Work-Group runs on a Compute Unit and a Work-Item runs on a Processing Element (PE).
- One thread is assigned to each Work-Item.
- Threads are swapped on and off the PEs.

OpenCL Memory Model
Threads can share memory with the other Threads in the same Work-Group.
Threads can synchronize with other Threads in the same Work-Group.
Global and Constant memory is accessible by all Threads in all Work-Groups.
Global and Constant memory is often cached inside a Work-Group.
Each Thread has registers and private memory.
Each Work-Group has a maximum number of registers it can use. These are divided equally among all its Threads.

```
status = cl_device_id device;
// find out how many devices are attached to each platform and get their ids:
status = clGetDeviceIDs( platform, CL_DEVICE_TYPE_GPU, 1, &device, NULL );
```

```
status = clGetDeviceIDs( platform, CL_DEVICE_TYPE_ALL, numDevices, devices, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```

```
status = clGetPlatformIDs( numPlatforms, platforms, NULL );
```
Querying the Device, II

```
clGetPlatformInfo(platform, CL_PLATFORM_PROFILE, size, str, NULL);
class cl_device_type
clGetPlatformInfo(platform, CL_PLATFORM_VERSION, size, NULL);
class cl_device_id
clGetPlatformInfo(platform, CL_PLATFORM_EXTENSIONS, size, NULL);
class cl_device_type

void printDeviceInfo(char *device)
```

Steps in Creating and Running an OpenCL program

1. Program header
2. Allocate the host memory buffers
3. Create an OpenCL context
4. Create an OpenCL command queue
5. Allocate the device memory buffers
6. Write the data from the host buffers to the device buffers
7. Read the kernel code from a file
8. Compile and link the kernel code
9. Create the kernel object
10. Setup the arguments to the kernel object
11. Enqueue the kernel object for execution
12. Read the results buffer back from the device to the host
13. Clean everything up
```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h> // for timing
#include "cl.h"

1. .cpp Program Header

```
Enqueuing Works Like a Conveyer Belt

Write Buffer dC
Execute Kernel
Write Buffer dB
Write Buffer dA

Whopp-a, whopp-a

The .cl File

```
kernal
void
ArrayMult( global const float *dA, global const float *dB, global float *dC )
{
    int gid = get_global_id( 0 );
}
```

Which dimension's index are we fetching?
gid = which element we are dealing with right now.

Since this is a 1D problem, X is the only index we need to get.

.OpenCL code is compiled in the Driver...

Application Program

OpenCL code in a separate file

OpenCL code is compiled in the Driver...

Applicaiton Program

OpenCL Driver does the Compile and Link

(... just like OpenGL's GLSL Shader code is compiled in the driver)

Application Program

GLSL shader code in a separate file

GLSL Driver does the Compile and Link

7. Read the Kernel Code from a File into a Character Array

```
const char *CL_FILE_NAME = { "arraymult.cl" };
...
FILE *
fp = fopen( CL_FILE_NAME, "r" );
if( fp == NULL )
{
    fprintf( stderr, "Cannot open OpenCL source file '%s'
    return 1;
}
// read the characters from the opencl kernel program:
FILE *
fp = fopen( CL_FILE_NAME, "r" );
if( fp == NULL )
{
    fprintf( stderr, "Cannot open OpenCL source file " file "", CL_FILE_NAME );
    return 1;
}
```
Something new: Intermediate Compilation

- You pre-compile your OpenCL code with an external compiler
- Your OpenCL code gets turned into an intermediate form known as SPIR-V
- SPIR-V gets turned into fully-compiled code at runtime

Advantages:
1. Software vendors don’t need to ship their OpenCL source
2. Syntax errors appear during the SPIR-V step, not during runtime
3. Software can launch faster because half of the compilation has already taken place
4. This guarantees a common front-end syntax
5. This allows for other language front-ends

How does that array-of-strings thing actually work?

```c
char *ArrayOfStrings[3];
ArrayOfStrings[0] = …one commonly-used function…;
ArrayOfStrings[1] = " . . . another commonly-used function. . . ";
ArrayOfStrings[2] = " . . . the real OpenCL code . .. ";
cl_program program = clCreateProgramWithSource( context, 1, (const char **) ArrayOfStrings, NULL, &status );
```

These are two ways to provide a single character buffer:

```c
char *buffer = " . . . the entire OpenCL code . . . ";
cl_program program = clCreateProgramWithSource( context, 1, (const char **) &buffer, NULL, &status );
```

Why use an array of strings to hold the OpenCL program, instead of just a single string?

1. You can use the same OpenCL source and insert the appropriate ‘#defines’ at the beginning
2. You can insert a common header file (= a .h file)
3. You can simulate a “#include” to re-use common pieces of code

8. Compile and Link the Kernel Code

```c
// create the kernel program on the device:
char *strings[1];
strings[0] = "ProgramText; 
cl_program program = clCreateProgramWithSource( context, 1, (const char **)strings, NULL, &status );
```

1. Software vendors don’t need to ship their OpenCL source
2. Syntax errors appear during the SPIR-V step, not during runtime
3. Software can launch faster because half of the compilation has already taken place
4. This guarantees a common front-end syntax
5. This allows for other language front-ends

9. Create the Kernel Object

```c
cl_kernel kernel = clCreateKernel( program, "ArrayMult", &status );
```

10. Setup the Arguments to the Kernel Object

```c
status = clSetKernelArg( kernel, 0, sizeof(cl_mem), &dA );
status = clSetKernelArg( kernel, 1, sizeof(cl_mem), &dB );
status = clSetKernelArg( kernel, 2, sizeof(cl_mem), &dC );
```

kernel void ArrayMult( global const float *dA, global const float *dB, global float *dC )
size_t globalWorkSize[3] = { NUM_ELEMENT, 1, 1 };
size_t localWorkSize[3]   = { LOCAL_SIZE,       1, 1 };
Wait( cmdQueue ); // will be covered in the OpenCL event notes

double time0 = omp_get_wtime( );
status = clEnqueueNDRangeKernel( cmdQueue, kernel, 1, NULL, globalWorkSize, localWorkSize, 0, NULL, NULL );
Wait( cmdQueue ); // will be covered in the OpenCL event notes

double time1 = omp_get_wtime( );

## Enqueue the Kernel Object for Execution

### Work-Groups, Local IDs, and Global IDs

"NDRange Index Space" can be 1D, 2D, or 3D. This one is 1D.

$$\text{# GlobalIndexSpaceSizeWorkGroups} = \frac{\text{GlobalIndexSpaceSize}}{\text{WorkGroupSize}}$$

$$\text{# GlobalIndexSpaceSizeWorkGroups} = \frac{5 \times 4}{20} = \frac{20 \times 12}{4 \times 3}$$

### Work-Groups, Local IDs, and Global IDs

"NDRange Index Space" can be 1D, 2D, or 3D. This one is 2D.

### Work-Groups, Local IDs, and Global IDs

"NDRange Index Space" can be 1D, 2D, or 3D. This one is 3D.

### Figuring Out What Thread You Are and What Your Thread Environment is Like

- `uint get_work_dim();`
- `size_t get_global_size( uint dimindx );`
- `size_t get_global_id( uint dimindx );`
- `size_t get_local_size( uint dimindx );`
- `size_t get_local_id( uint dimindx );`
- `size_t get_num_groups( uint dimindx );`
- `size_t get_group_id( uint dimindx );`
- `size_t get_global_offset( uint dimindx );`

0 ≤ dimindx ≤ 2

### 12. Read the Results Buffer Back from the Device to the Host

status = clEnqueueReadBuffer( cmdQueue, dC, CL_TRUE, 0, dataSize, hC, 0, NULL, NULL );

Want to block until done?

```
13. Clean Everything Up

clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(cmdQueue);
clReleaseMemObject(dA);
clReleaseMemObject(db);
clReleaseMemObject(dC);
delete[] hA;
delete[] hB;
delete[] hC;

Do this because we created these arrays with new.

Array Multiplication Performance: What is a Good Work-Group Size?

Writing the .cl Program's Binary Code

size_t binary_sizes;
status = clGetProgramInfo(Program, CL_PROGRAM_BINARY_SIZES, 0, NULL, &binary_sizes);
size_t size;
status = clGetProgramInfo(Program, CL_PROGRAM_BINARY_SIZES, sizeof(size_t), &size, NULL);
unsigned char *binary = new unsigned char [size];
status = clGetProgramInfo(Program, CL_PROGRAM_BINARIES, size, &binary, NULL);
FILE *fpbin = fopen("particles.nv", "wb");
if(fpbin == NULL){
    fprintf(stderr, "Cannot create 'particles.bin'
    }else{
        fwrite(binary, 1, size, fpbin);
        fclose(fpbin);
    delete[] binary;
}

char *options = {""};
status = clBuildProgram(program, 1, &device, options, NULL, NULL);
if(status != CL_SUCCESS){
    size_t size;
    clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0, NULL, &size);
    cl_char *log = new cl_char[size];
    clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, size, log, NULL);
    fprintf(stderr, "clBuildProgram failed:
    %s
    
    delete[] log;
}

char *strings[1];
strings[0] = clProgramText;
cl_program program = clCreateProgramWithSource(context, 1, (const char **)strings, NULL, &status);
delete[] clProgramText;

Importing that Binary Code back In:

Instead of doing this:
char *strings[];
cl_program program = clCreateProgramWithSource(context, 1, (const char **)strings, NULL, &status);
delete [] clProgramText;

You would do this:
unsigned char byteArray[numBytes];
cl_program program = clCreateProgramWithBinary(context, 1, &device, &numBytes, &byteArray, &binaryStatus, &status);
delete [] byteArray;

And you still have to do this:
char *options[""];
status = clBuildProgram(program, 1, &device, options, NULL, NULL);
if(status != CL_SUCCESS){
    size_t size;
    clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, 0, NULL, &size);
    cl_char *log = new cl_char[size];
    clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, size, log, NULL);
    fprintf(stderr, "clBuildProgram failed:
    %s
    
    delete[] log;
}