OpenCL Programming

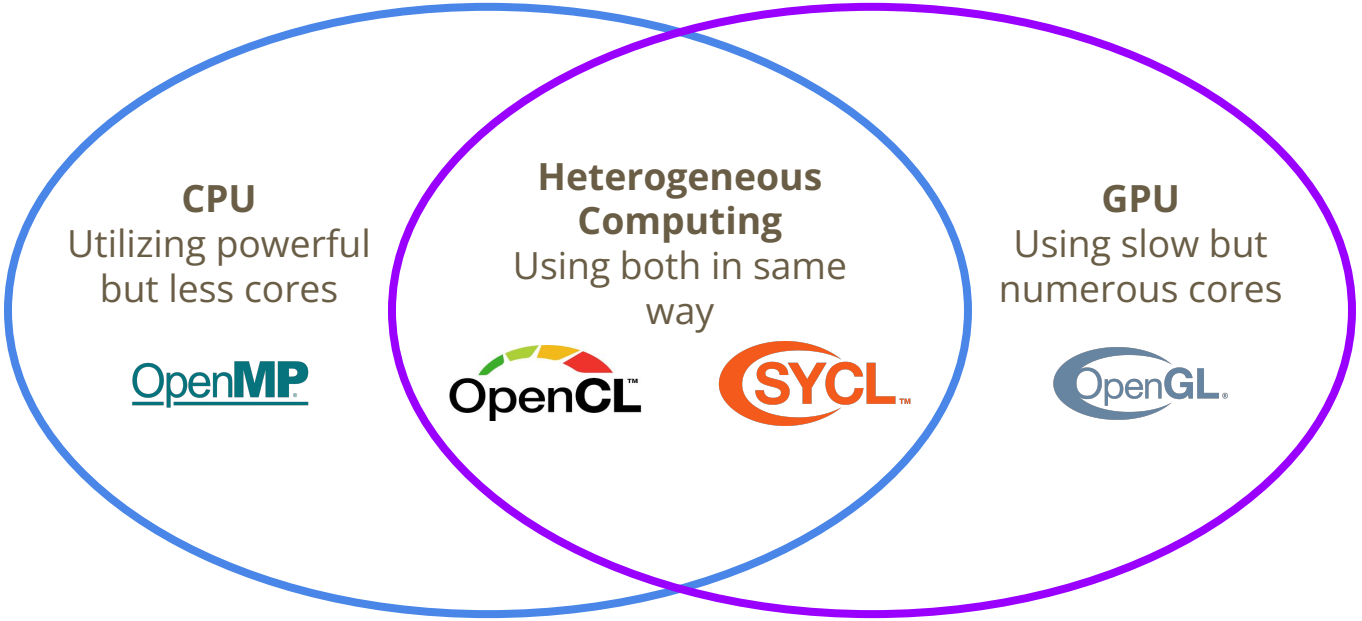
By Joel Jacob

Acknowledgement of Country

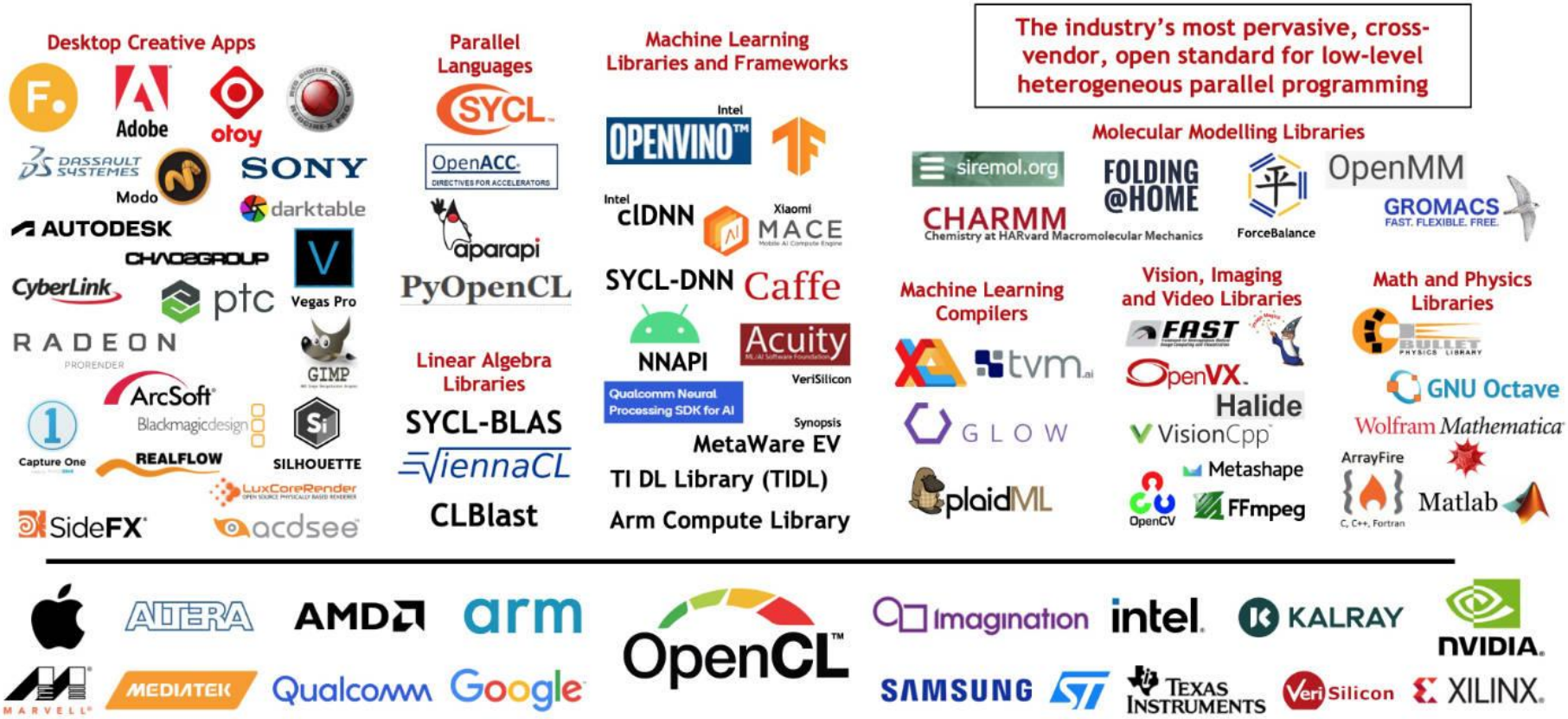
We respectfully acknowledge the Traditional Owners of the land on which we are meeting and pay our respects to Elders past, present, and emerging. We extend that respect to all Aboriginal and Torres Strait Islander peoples today.

Heterogeneous Computing with OpenCL

Open Computing Language is an open, royalty-free standard for cross-platform, parallel programming of diverse accelerators of CPUs, GPUs and others.



The Khronos Group and OpenCL Applications



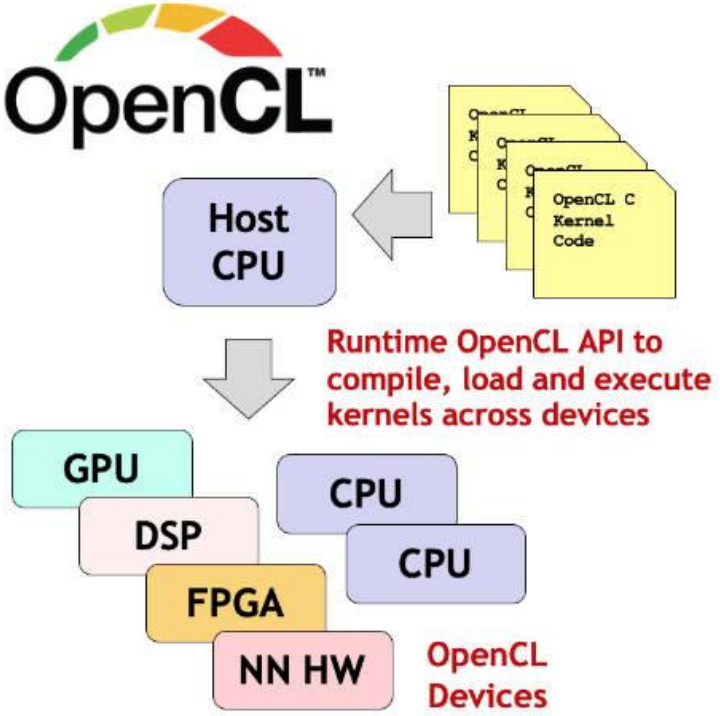
OpenCL Platform Model

Host and Device model
- A host (CPU) connected to one or more OpenCL devices

OpenCL devices (Accelerators):
- A collection of one or more compute units (cores)
- GPUs, FPGAs, TPU, Other CPUs

A Compute Unit
- Composed of one or more processing elements
- Processing elements execute code as SIMD or SPMD

Command Queue:
- Host submits commands for execution on the devices asynchronously



Complements GPU-only APIs
Simpler programming model
Relatively lightweight run-time
More language flexibility, e.g., pointers
Rigorously defined numeric precision

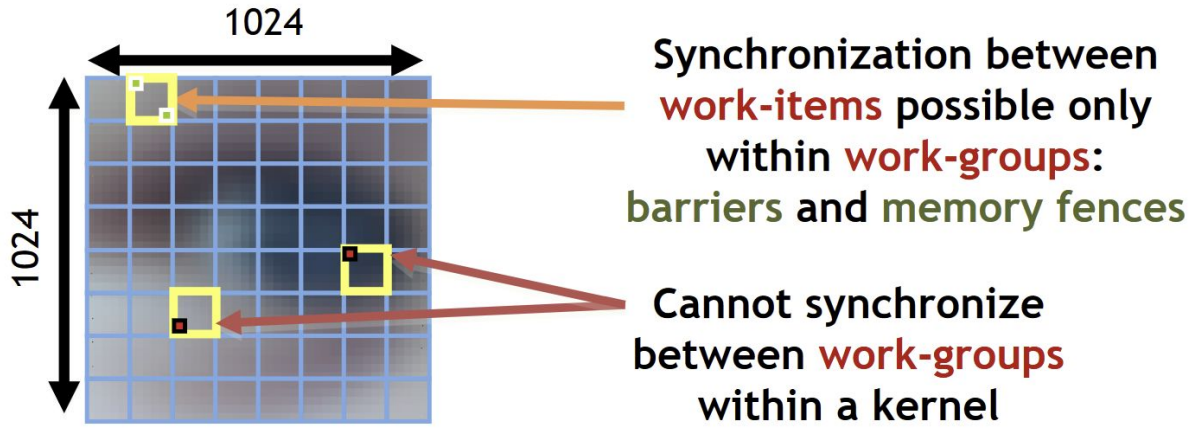
Dimensions and Kernels

Kernels

- Replace loops with functions (a kernel, work item, thread) executing at each point in a problem domain
- Is compiled at runtime, written in subset of C99
- Eg. In 1024 × 1024 Matrix multiplication, there are 1,048,576 kernel execution

Dimensions

- Global Dimensions: 1024 x 1024 (whole problem space)
- Eg. Local Dimensions: 128 x 128 (executed together, work-group, thread block)



Memory Model

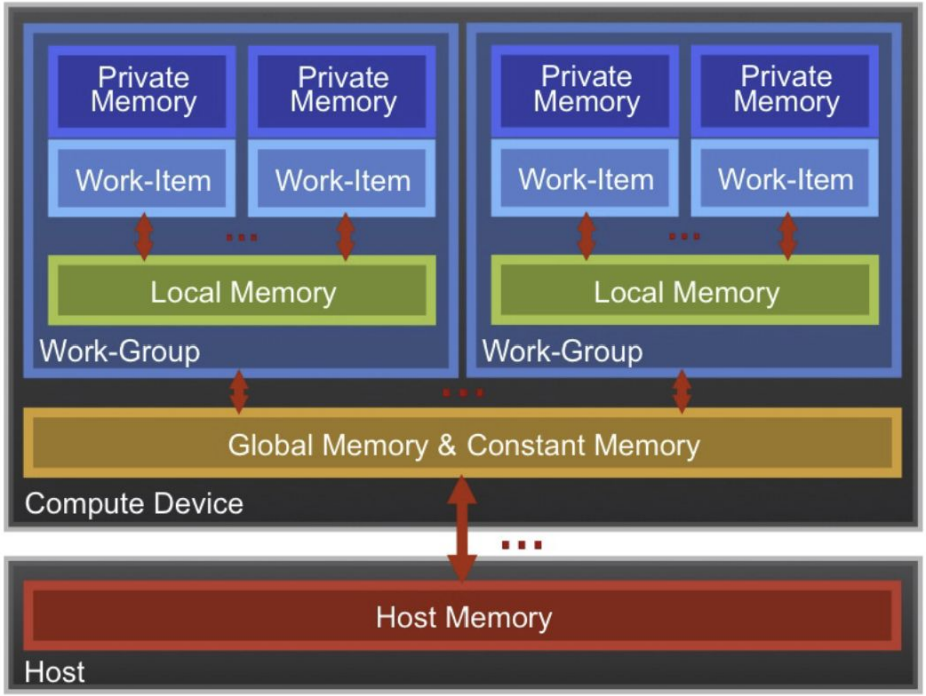
Explicit memory management between host to devices and cores.

Levels of memory are:

- Private Memory: Per work-item
- Local Memory: Shared within a workgroup
- Global/Constant Memory: Not synchronized
- Host Memory: On the CPU

Must move data from host to global to local and back

Device memory is unprotected (unreported segfaults)



Coding in OpenCL (Matrix Multiplication)

- OpenCL programming API is available officially in C99 and C++14, while the kernel language is a subset of C99.
- No need for SDK, only the header files and link libraries to develop OpenCL on most platforms.

Kernel For Matrix Multiplication

```
__kernel void matrix_mul(
    __global float *A, __global float *B,
    __global float *C, const int N)
{
    int i = get_global_id(0);
    int j = get_global_id(1);
    float sum = 0.0f;
    for (int k = 0; k < N; k++)
    {
        sum += A[i * N + k] * B[k * N + j];
    }
    C[i * N + j] = sum;
}
```

Sequential Matrix Multiplication (C++)

```
void matrix_mul_seq(const std::vector<float> &A,
const std::vector<float> &B, std::vector<float> &C,
const int N)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            float sum = 0.0f;
            for (int k = 0; k < N; k++)
            {
                sum += A[i * N + k] * B[k * N + j];
            }
            C[i * N + j] = sum;
        }
    }
}
```

Running the Code

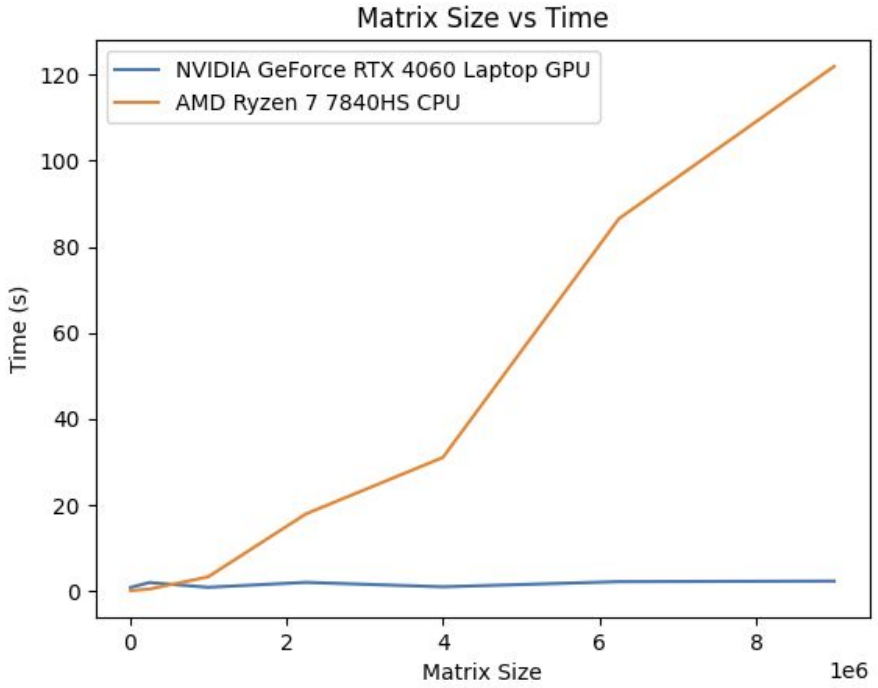
For the c++ host program to run the kernel code:

- Choose device to run (GPU)
- Create context (environment with defined synchronization and memory management)
- Create command queue (Commands for device (kernel execution, synchronization, and memory operations) are submitted here)
- Read the kernel code and add it to a program object.
- Compile the program object at runtime.
- Allocate global memory on device buffer memory for A, B and C
- Write matrices A and B to device memory
- Setting the kernel argument to the buffer memory
- Using 0 dimensional range, execute the kernel
- Read the C matrix back from device to host memory

```
// Get the most performant device
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);
cl::Platform platform = platforms.front();
std::vector<cl::Device> devices;
platform.getDevices(CL_DEVICE_TYPE_GPU, &devices);
cl::Device device = devices.front();

// Create a context and command queue
cl::Context context(device);
cl::CommandQueue queue(context, device);
std::string kernel_code; // This is GEMM1 kernel

sources.push_back({kernel_code.c_str(), kernel_code.length()});

cl::Program program(context, sources);

// Build the program
program.build("-cl-std=CL1.2");

// Create a kernel
cl::Kernel kernel(program, "matrix_mul");

cl::Buffer buffer_A(context, CL_MEM_READ_ONLY, N * N * sizeof(float));
cl::Buffer buffer_B(context, CL_MEM_READ_ONLY, N * N * sizeof(float));
cl::Buffer buffer_C(context, CL_MEM_WRITE_ONLY, N * N * sizeof(float));

// Write the input buffers
queue.enqueueWriteBuffer(buffer_A, CL_TRUE, 0, N * N * sizeof(float), A.data());
queue.enqueueWriteBuffer(buffer_B, CL_TRUE, 0, N * N * sizeof(float), B.data());

// Set the kernel arguments
kernel.setArg(0, buffer_A);
kernel.setArg(1, buffer_B);
kernel.setArg(2, buffer_C);
kernel.setArg(3, N);

// Execute the kernel
cl::NDRange global(N, N);
queue.enqueueNDRangeKernel(kernel, cl::NullRange, global);

// Read the output buffer
queue.enqueueReadBuffer(buffer_C, CL_TRUE, 0, N * N * sizeof(float), C.data());
```

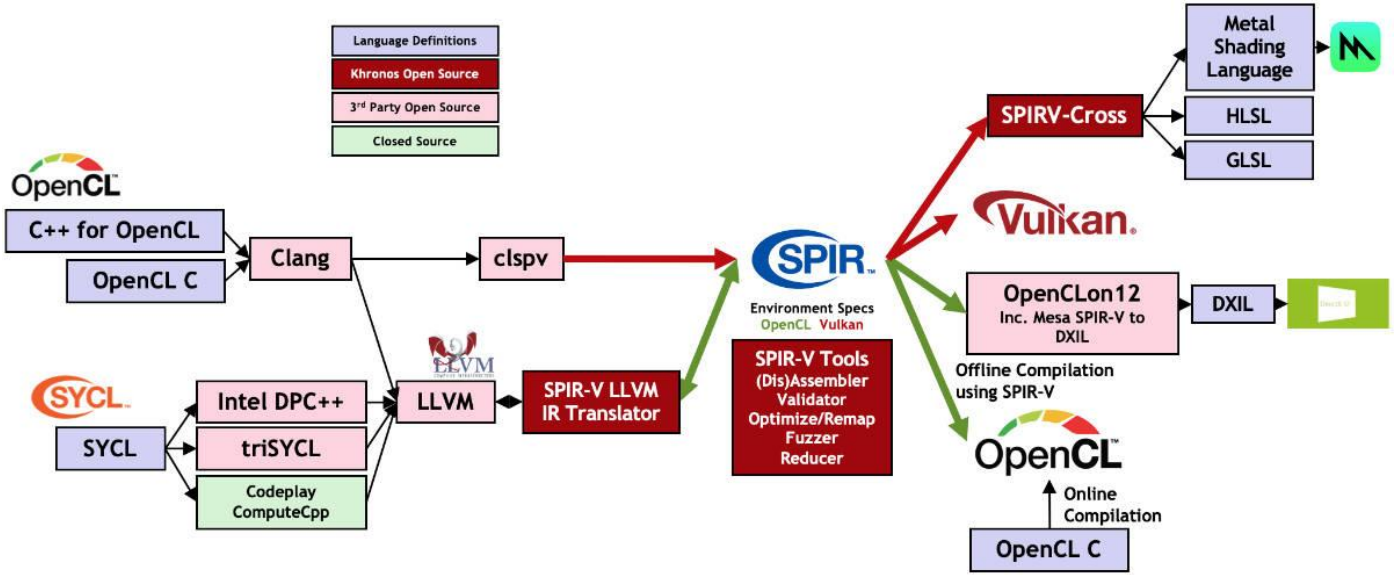
Performance and Improvements

- 20.62 times faster than sequential method
- Offers significant speedups on large matrices
- Performance gain depends on hardware
- CPU still faster on small matrices because of GPU memory transfer overhead
- Naive kernel, not utilizing the local memory or work groups



OpenCL Development and 3.0

- OpenCL 3.0 released on September 30th, 2020
- Subgroups are part of core specification, allowing finer-grained parallelism.
- Extensions for Asynchronous Data Copies in Embedded systems
- Can cross compile to various other runtimes like Vulkan which enable OpenCL applications to be deployed onto platforms that do not have available native OpenCL drivers.



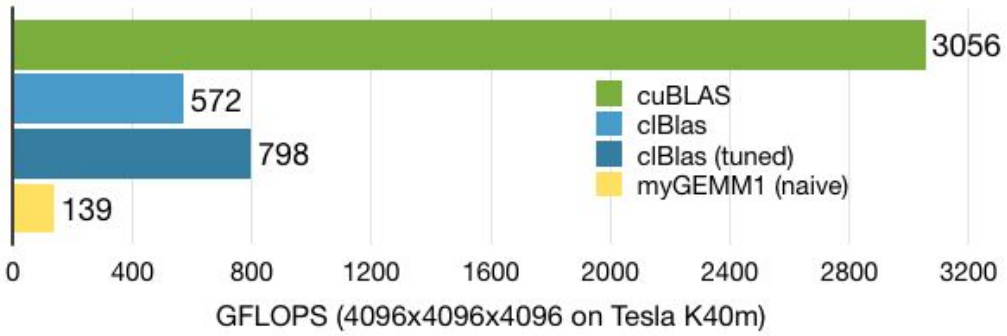
Strengths and Weaknesses

Strengths

- Cross-platform, open standard.
- Supports a wide range of hardware and link libraries are available by default on all OSs.
- Can combine with other programming models (OpenMP or MPI)
- Huge ecosystems with directives with OpenCL like OpenAcc which uses OpenCL kernels

Limitations

- Complex memory management.
- Performance varies by hardware and lags behind CUDA.
- Documentation, Tooling and library support not as mature as CUDA.
- Modern SYCL alternatives available



References

Lopez-Novoa, U. (2012, October 14). *Introduction to OpenCL* [Slide show]. SlideShare. https://www.slideshare.net/slideshow/introduction-to-opencl/14719571

OPENCL - the open standard for parallel programming of heterogeneous systems. (2013, July 21). The Khronos Group. https://www.khronos.org/opencl/

Thompson, J. A., & Schlachter, K. (2012). *An introduction to the OpenCL programming model*. https://www.semanticscholar.org/paper/An-Introduction-to-the-OpenCL-Programming-Model-Thompson-Schlachter/c5c925a729a6691da121eed06dc94f27a9e8d3dc

OpenCL testing repository available: https://github.com/debugst1ck/opencl-introduction