Enabling OpenCL Acceleration of Web Applications

Tasneem Brutch
Samsung Electronics

Khronos WebCL Working Group Chair

LinuxCon
Sept. 17, 2013, New Orleans, LA
Outline

- WebCL tutorial, demos and coding examples
- WebCL use case: Big Data Visualization (presentation and demo)
- WebCL open API standardization: Status and roadmap
- Security requirements and provisions & WebCL Kernel Validator
- OpenCL to WebCL translator utility
- Questions and Answers
WebCL Overview

- **High performance:**
  - Enables acceleration of compute-intensive web applications, through high performance parallel processing.

- **Platform independent:**
  - Portable and efficient access, to heterogeneous multicore devices.
  - Provides single and coherent standard across desktop and mobile devices.

- WebCL (Web Computing Language) is a JavaScript binding to OpenCL

- Being standardized by the Khronos standards organization
Motivation

• Enable new mobile apps with high compute demands:
  - Big Data Visualization
  - Augmented Reality
  - Video Processing
  - Computational Photography
  - 3D Games

• Popularity of web-centric platforms
OpenCL Features Overview

- C-based cross-platform programming interface
- Kernels: Subset of ISO C99 with language extensions
- Run-time or build-time compilation of kernels
- Well-defined numerical accuracy: IEEE 754 rounding with specified max. error
- Rich set of built-in functions: cross, dot, sin, cos, pow, log, ...
- Accelerated mathematical operations (fast-math)
OpenCL Platform Model

• A host is connected to one or more OpenCL devices

• OpenCL device:
  - A collection of one or more **compute units**
    - (≈ cores)
  - A compute unit is composed of one or more **processing elements**
    - (≈ threads)
  - Processing element code execution:
    - As SIMD or SPMD
OpenCL Execution Model

- **Kernel:**
  - Basic unit of executable code, similar to a C function
  - Data-parallel or task-parallel

- **Program:**
  - Collection of kernels, and functions called by OpenCL kernels
  - Analogous to a dynamic library (run-time linking)

- **Command Queue:**
  - Applications queue kernels and data transfers
  - Performed in-order or out-of-order

- **Work-item:**
  - An execution of kernel by a processing element (~ thread)

- **Work-group:**
  - Collection of related work-items executing on a single compute unit (~ core)
OpenCL Memory Model

**Memory types:**
- Private memory (blue)
  - Per work-item
- Local memory (green)
  - At least 32KB split into blocks
  - Available to a work-item in a given work-group
- Global/Constant memory (orange)
  - Not synchronized
- Host memory (red)
  - On the CPU

**Memory management is explicit:**
- Application must move data from host → global → local and back
OpenCL Kernels

- **OpenCL kernel**
  - Defined on an N-dimensional *computation domain*
  - Executes a kernel at each point of the computation domain

### Traditional Loop

```c
void vectorMult(
    const float* a,
    const float* b,
    float* c,
    const unsigned int count)
{
    for(int i=0; i<count; i++)
        c[i] = a[i] * b[i];
}
```

### Data Parallel OpenCL

```c
kernel void vectorMult(
    global const float* a,
    global const float* b,
    global float* c)
{
    int id = get_global_id(0);
    c[id] = a[id] * b[id];
}
```
Executing OpenCL Programs

1. **Query host** for OpenCL devices
2. **Create a context** to associate OpenCL devices
3. **Create programs** for execution on one or more associated devices
4. **Select kernels to execute** from the programs
5. **Create memory objects** accessible from the host and/or the device
6. **Copy memory data** to the device as needed
7. **Provide kernels to command queue** for execution
8. **Copy results** from the device to the host
Programming WebCL

- Initialization
- Creating kernel
- Running kernel
- **WebCL image object creation**
  - From Uint8Array
  - From `<img>`, `<canvas>`, or `<video>`
  - From WebGL vertex buffer
  - From WebGL texture
- **WebGL vertex animation**
- **WebGL texture animation**
Nbody Simulation

- Calculates the positions and velocities of $N$ particles and animates them
- Two simulation modes:
  - JavaScript and WebCL
- Two drawing modes:
  - JavaScript and WebGL with 2D/3D rendering option
- For 1024 particles, WebCL gets 20~40x faster simulation time on Mac.

http://www.youtube.com/watch?v=F7YSQxz3j7o
Deform

- Calculates and renders transparent and reflective deformed spheres in skybox scene.
- Performance comparison on Mac:
  - JS: ~1 FPS
  - WebCL: 87-116 FPS
- http://www.youtube.com/watch?v=9Ttux1A-Nuc
WebCL Initialization

```javascript
var platforms = webcl.getPlatforms();
var devices = platforms[0].getDevices(WebCL.DEVICE_TYPE_GPU);
var context = webcl.createContext({ WebCLDevice: devices[0]});
var queue = context.createCommandQueue();
```
Creating WebCL Kernel

```html
<script id="squareProgram" type="x-kernel">
  __kernel square(__global float* input,
                  __global float* output,
                  const unsigned int count)
  {
    int i = get_global_id(0);
    if(i < count)
      output[i] = input[i] * input[i];
  }
</script>

<var programSource = getProgramSource("squareProgram"); //JavaScript function using DOM APIs
  var program = context.createProgram(programSource);
  program.build();
  var kernel = program.createKernel("square");
</script>
```
Running WebCL Kernels

```javascript
... var numBytes = Float32Array.BYTES_PER_ELEMENT * count;
var inputBuf = context.createBuffer(WebCL.MEM_READ_ONLY, numBytes);
var outputBuf = context.createBuffer(WebCL.MEM_WRITE_ONLY, numBytes);

var data = new Float32Array(count);
// populate data ...
//Second arg indicates blocking API
queue.enqueueWriteBuffer(inputBuf, true, bufferOffset = 0, numBytes, data);

kernel.setKernelArg(0, inputBuf);
kernel.setKernelArg(1, outputBuf);
kernel.setKernelArg(2, count, WebCL.KERNEL_ARG_INT);

var workGroupSize = kernel.getWorkGroupInfo(devices[0], WebCL.KERNEL_WORK_GROUP_SIZE);
var localWorkGroupSize = NULL; // automatically determine how to breakup global-work-items
queue.enqueueNDRangeKernel(kernel, [count], [workGroupSize], localWorkGroupSize);

queue.finish(); // This API blocks
queue.enqueueReadBuffer(outputBuf, true, offset = 0, numBytes, data); // Second arg indicates blocking API
</script>
WebCL Image Object Creation

From Uint8Array ()

<script>
  var bpp = 4;  // bytes per pixel
  var pixels = new Uint8Array(width * height * bpp);
  var pitch = width * bpp;
  var clImage = context.createImage(WebCL.MEM_READ_ONLY,
                                     {channelOrder:WebCL.RGBA, channelType:WebCL.UNORM_INT8,
                                      width:width, height:height, pitch:pitch } );
</script>

From <img> or <canvas> or <video>

<script>
  var canvas = document.getElementById("aCanvas");
  var clImage = context.createImage(WebCL.MEM_READ_ONLY, canvas);
  // format, size from element
</script>
WebCL: Vertex Animation

- Vertex Buffer Initialization:

```javascript
var points = new Float32Array(NPOINTS * 3);
var glVertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, points, gl.DYNAMIC_DRAW);
var clVertexBuffer = context.createFromGLBuffer(WebCL.MEM_READ_WRITE, glVertexBuffer);
kernel.setKernelArg(0, NPOINTS, WebCL.KERNEL_ARG_INT);
kernel.setKernelArg(1, clVertexBuffer);
```

- Vertex Buffer Update and Draw:

```javascript
function DrawLoop() {
    queue.enqueueAcquireGLObjecents([clVertexBuffer]);
    queue.enqueueNDRangeKernel(kernel, [NPOINTS], [workGroupSize], null);
    queue.enqueueReleaseGLObjecents([clVertexBuffer]);
    gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.drawArrays(gl.POINTS, 0, NPOINTS);
    gl.flush();
}
```
WebCL: Texture Animation

- Texture Initialization:

```html
<script>
var glTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, glTexture);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
var clTexture = context.createFromGLTexture2D(WebCL.MEM_READ_WRITE, gl.TEXTURE_2D, glTexture);
kernel.setKernelArg(0, NWIDTH, WebCL.KERNEL_ARG_INT);
kernel.setKernelArg(1, NHEIGHT, WebCL.KERNEL_ARG_INT);
kernel.setKernelArg(2, clTexture);
</script>
```

- Texture Update and Draw:

```html
function DrawLoop() {
    queue.enqueueAcquireGLObjects([clTexture]);
    queue.enqueueNDRangeKernel(kernel, [NWIDTH*NHEIGHT], [workGroupSize], null);
    queue.enqueueReleaseGLObjects([clTexture]);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, glTexture);
    gl.flush();
}
</script>
```
Interactive Big Data Visualization

*Powered by WebCL*

To enable interactive exploration of big data!

Leo Meyerovich, Matthew Torok,
Eric Atkinson, Rastislav Bodik

*UC Berkeley Parallel Computing Lab*
Interactive Big Data Visualization

BIG DATA

INTERACTIVE

EFFICIENT
PRODUCTIVE

VISUALIZATIONS
DEMO: D3.js

D3.js
Circle Packing Demo

- 100 Nodes
- 1000 Nodes
- 10000 Nodes
- 20000 Nodes
DEMO: Superconductor

Superconductor
Sunburst Graph Demo
Demos

- Automatic parallelization
- Superconductor on three data sets:
  - 10,000 nodes: 71 fps
  - 100,000 nodes: 61 fps
  - 1,000,000 nodes: 11 fps
- JavaScript: 1,000 nodes at 27 fps
  - 100x difference
Four Tasks of Visualization

- Data
- Stylize
- Focus of WebCL use case development
- Render

Challenge: Scalable language support for each one?
Web Programming for Big Data Visualization

- **Data** => **JSON** (JavaScript Object Notation)
- **Custom styling** => **CSS selectors**
- **Simple scripting** => **JavaScript**

**Layout:**
- HTML (paragraph, table, ...)
  with JavaScript (calendar, ...): too slow
- New Domain Specific Language (DSL) for custom layouts
  - Simple but optimizable
Opportunity: Deploy Parallel JS Libraries

- OpenCL
- WebCL
- WebGL
- Web Workers

JavaScrip VM

- Parsing
- Styling
- Layout
- Rendering

shared memory
Parallel Runtime Architecture

- Parser.js
- Selectors (CL)
- Layout (CL)
- Renderer (GL)
- superconductor.js
- data, styling, widgets, scripts, ...

Multithreaded

JavaScript VM

GPU
class HBox:

\[ w := \text{left}.w + \text{right}.w \]

1. Local, single-assignment
2. Compiler automatically parallelizes
Layout Compiler: Scheduling + Code Generation

COMPILER

1. static scheduler
2. Flattening

custom layouts

traversals over data

Parser.js
Selectors (CL)
Layout (CL)
Renderer (GL)

JavaScript VM

superconductor.js
Layout Scheduler Finds Parallel Breadth-First Traversals

Traversal sequence valid for any data set:

Output as WebCL
Compiler handles these low-level optimizations.
WebCL Big Data Vis. Use Case: Summary

- Big data visualization language
- Orders-of-magnitude speedup over plain JavaScript
- Approach:
  - GPU compiler
  - GPU rendering library

WebCL Standardization,
Implementation & Conformance Testing
WebCL API Standardization

• Standardized, portable and efficient access to heterogeneous multicore devices from web content
• Define ECMAScript API for interaction with OpenCL
• Designed for security and robustness
• WebCL 1.0 based on OpenCL 1.1 Embedded Profile:
  - Implementations may utilize OpenCL 1.1 or 1.2
• Interoperability between WebCL and WebGL through a WebCL extension
• Initialization simplified for portability
• WebCL kernel validation
WebCL 1.0 Kernels

- Kernels do not support structures as arguments
- Kernels name must be less than 256 characters
- Mapping of CL memory objects into host memory space is not supported
- Binary kernels are not supported in WebCL 1.0
- Support for OpenCL extensions:
  - OpenCL Extension dependent functions may not be supported, or may require translation to WebCL specific extension.
- No 3D image support in WebCL 1.0:
  - May change in future WebCL versions
- Enhanced portability and simplified initialization:
  - Some device attributes may not be visible.
  - Code strongly bound to these attributes may not be translated
- Parameter mapping:
  - Certain OpenCL parameters may not directly carry over to WebCL, or a subset of OpenCL parameters may be available in WebCL.
WebCL API

- Window and WorkerUtils implements WebCLEnvironment
  - Declares WebCL webcl object (if not there, then WebCL is not supported)

- 3 Layers: Platform, Compiler, Runtime
WebCL API

- API mimic object-oriented nature of OpenCL 1.1 in JavaScript
- WebCL object defines all WebCL constants
- WebCL may support the following extensions
  - GL_SHARING — interoperability with WebGL
  - KHR_FP16 — 16-bit float support in kernels
  - KHR_FP64 — 64-bit float support in kernels
- HTML data interoperability
  - `<canvas>`, `<image>`, and `ImageData` sources can be bound to `WebCLBuffer` and `WebCLImage`
  - `<video>` tag can be bound to a `WebCLImage`
WebCL 1.0 API: Current Status

- **WebCL 1.0 API definition:**
  - Working Draft Publicly available at [www.khronos.org/webcl](http://www.khronos.org/webcl)
  - Expected WebCL1.0 specification release: 1H, 2014

- **Strong interest in WebCL API:**
  - Participation by Browser vendors, OpenCL hardware/driver vendors, application developers
  - Non-Khronos members engaged through public mailing list: [public_webcl@khronos.org](mailto:public_webcl@khronos.org)
WebCL Conformance Testing

• WebCL Conformance Framework and Test Suite (WiP)

• Scope
  - Full API coverage
    - Input/output validation
    - Error handling checks for all methods in the API
  - Security and robustness related tests
    - Two OpenCL 1.2 robustness and security extensions ratified in 2012
      - Context termination: clCreateContext() with CL_CONTEXT_MEMORY_INITIALIZE_KHR
      - Memory initialization: clAbortContextKHR (cl_context)
    - Conformance tests related to WebCL Validator completed
      - Kernel validator enforces prevention against out of bounds memory accesses
  - WebCL requires a conformant underlying OpenCL on the host system
    - Tests will not re-run OpenCL conformance test suite in JavaScript

• Conformance tests for OpenCL 1.2 security and robustness extensions
  - Contributed by Samsung and reviewed by WebCL members
WebCL Conformance Testing: Status

- **Clean up of existing tests**
  - Defined and deployed coding style guide
  - Removed code duplication
  - Consolidated redundant tests
  - Shared kernel sources between tests where appropriate possible
  - Compiled common routines into helper files (e.g. webcl-utils.js)

- **Improved conformance framework structure**
  - Aligned test framework with WebGL conformance test suite

- **Completion of API unit tests/conformance tests**
  - ~80% WebCL APIs are covered today (WiP)

- **Redesigned start page**
  - Support for running the test suite on any OpenCL platforms/devices available on the host machine

- **Available on GitHub:** [https://github.com/KhronosGroup/WebCL-conformance/](https://github.com/KhronosGroup/WebCL-conformance/)

- **Tracked by a “meta” bug on Khronos public bugzilla:**
  - [https://www.khronos.org/bugzilla/show_bug.cgi?id=793](https://www.khronos.org/bugzilla/show_bug.cgi?id=793)
WebCL API

Security and Robustness
Security and Robustness

Security threats addressed:
- Prevention of information leakage from out of bounds memory access
- Prevent denial of service from long running kernels

Security requirements:
- Prevent out of range memory accesses
  - Prevent out of bounds memory accesses
  - Prevent out of bounds (OOB) memory writes
  - Return a pre-defined value for out of bound reads
  - Implementation of WebCL Kernel Validator
- Memory initialization to prevent information leakage
  - Initialization of private and local memory objects
- Prevent potential Denial of Service from long running kernels
  - Ability to terminate contexts with long running kernels
- Prevent security vulnerabilities from undefined behavior
  - Undefined OpenCL behavior to be defined by WebCL
  - Robustness through comprehensive conformance testing
WebCL API Security: Current Status

- Cross-working group security initiative
  - Engaged with OpenCL, WebGL, OpenGL, and OpenGL-ES WGs
- Ratification of OpenCL 1.2 robustness/security extensions
  - OpenCL Context Termination extension
  - OpenCL Memory Initialization extension
- WebCL Kernel Validator
  - Open source WebCL kernel validator development Phase-I completed:
  - Accessible from https://github.com/KhronosGroup/webcl-validator
  - Tracked by a “meta” bug on Khronos public bugzilla:
    - http://www.khronos.org/bugzilla/show_bug.cgi?id=875
WebCL Kernel Validator

- **Protection against out of bounds memory accesses:**
  - The memory allocated in the program has to be initialized (prevent access to data from other programs)
  - Untrusted code must not access invalid memory

- **Memory initialization:**
  - Initializes local and private memory (in case underlying OpenCL implementation does not implement memory initialization extension)

- **Validator implementation:**
  - Keeps track of memory allocations (also in runtime if platform supports dynamic allocation)
  - Traces valid ranges for reads and writes
  - Validates them efficiently while compiling or at run-time
OpenCL to WebCL Translator Utility
OpenCL to WebCL Translator Utility

- **OpenCL to WebCL Kernel Translator**
  - Input: An OpenCL kernel
  - Output: WebCL kernel, and a log file, that details the translation process.
  - Tracked by a “meta” bug on Khronos public bugzilla:
    - [http://www.khronos.org/bugzilla/show_bug.cgi?id=785](http://www.khronos.org/bugzilla/show_bug.cgi?id=785)

- **Host API translation** (WiP)
  - Accepts an OpenCL host API calls and produces the WebCL host API calls that will eventually be wrapped in JS.
  - Provides verbose translation log file, detailing the process and any constraints.
  - Tracked by a “meta” bug on Khronos public bugzilla:
    - [http://www.khronos.org/bugzilla/show_bug.cgi?id=913](http://www.khronos.org/bugzilla/show_bug.cgi?id=913)
WebCL Prototypes & Experimental Integration into Browser Engines
WebCL Prototype Implementations

• Nokia’s prototype:
  - Firefox extension, open sourced May 2011 (Mozilla Public License 2.0)
  - [https://github.com/toaarnio/webcl-firefox](https://github.com/toaarnio/webcl-firefox)

• Samsung’s prototype:
  - Uses WebKit, open sourced June 2011 (BSD)
  - [https://github.com/SRA-SiliconValley/webkit-webcl](https://github.com/SRA-SiliconValley/webkit-webcl)

• Motorola Mobility’s prototype:
  - Uses Node.js, open sourced April 2012 (BSD)
  - [https://github.com/Motorola-Mobility/node-webcl](https://github.com/Motorola-Mobility/node-webcl)
WebCL on Firefox

- **Firefox add-on**
  - Android should be doable with reasonable effort – any volunteers?
  - Source code: [https://github.com/toaarnio/webcl-firefox](https://github.com/toaarnio/webcl-firefox)
  - Main limitation: Cannot share textures/buffers with WebGL

- **Special Firefox build with integrated WebCL**
  - Browser internal APIs allow WebGL resource sharing
  - Currently Work-in-progress

http://webcl.nokiaresearch.com/SmallPT/webcl-smallpt.html  
http://fract.ured.me/
Samsung’s WebCL Implementation

- Samsung provided an open-source version of their first prototype WebCL implementation for WebKit in July 2011 (BSD license).
- Since Q4 of 2012 the prototype has been through a major refinement effort, including:
  - Addition of a general purpose abstraction layer for Compute languages, called `ComputeContext`.
  - Class redesign and clean up
  - Layout tests support
  - Kept in sync with the latest spec
- Project currently being hosted on GitHub
  - [https://github.com/SRA-SiliconValley/webkit-webcl](https://github.com/SRA-SiliconValley/webkit-webcl)
  - [(https://code.google.com/p/webcl/ is obsolete)](https://code.google.com/p/webcl/ is obsolete)
WebCL API Summary

- WebCL is a JavaScript binding to OpenCL, allowing web applications to leverage heterogeneous computing resources.

- WebCL enables significant acceleration of compute-intensive web applications.

- WebCL API is designed for security.
Resources and Contacts

• WebCL Resources:
  - WebCL distribution lists: public_webcl@khronos.org, webcl@khronos.org
  - Samsung WebCL prototype: [https://github.com/SRA-SiliconValley/webkit-webcl](https://github.com/SRA-SiliconValley/webkit-webcl)
  - Motorola WebCL prototype: [https://github.com/Motorola-Mobility/node-webcl](https://github.com/Motorola-Mobility/node-webcl)

• Contacts:
  - Tasneem Brutch (t.brutch@samsung.com), WebCL Working Group Chair
  - Tomi Aarnio (tomi.aarnio@nokia.com), WebCL Specification Editor
  - Mikael Bourges-Sevenier (msevenier@aptina.com), WebCL Specification Editor
Questions and Answers
Thank You!