CrowdCL

Web-Based Volunteer Computing with WebCL

Tommy MacWilliam, Cris Cecka

Computer Science
Institute for Applied Computational Science
School of Engineering and Applied Sciences
Harvard University

September 11, 2013
Volunteer Computing

- "Donation" of CPU cycles to scientific problems
  - Folding@home
    - 300,000 contributors... right now.
    - 5 PetaFLOPS sustained
  - SETI@home
    - 3 million participants

- PrimeGrid, GPUGRID, NFS@Home, NSA@Home (j/k)
High Throughput Science
High Throughput Science
High Throughput Science

The right tool...
High Throughput Science
Goals

- Bring volunteer computing to the web browser
  - “Volunteer"
  - Reduce downloading/installing friction.
  - Web-browser as a high-performance distributed computing platform.
- Develop robust library for GPU computing in Javascript.
  - Enable GPU development and metaprogramming on the web.

“Windows and Linux present a near-infinite combination of hardware, software, and drivers that would not be encountered in a local setting. This means that a significant amount of time is spent dealing with incompatibilities when the clients are developed, and every time a new version of the operating system is shipped such as Windows 7, or the latest version of a Linux distribution.”

— Beberg et al. *Folding@home: Lessons From Eight Years of Volunteer Distributed Computing*
WebCL

- Experimental cross-platform JS binding for OpenCL
- Available for Firefox, WebKit, and Node.js

- API is verbose, procedural, and difficult to use
Contributions

- **KernelContext, KernelUtils**
  - PyCUDA inspired abstraction layer for WebCL

- **CrowdCL**
  - Framework for developing and deploying high performance, web-based volunteer computing projects.

- Application to existing crowd-generated data project.
- Comparison with existing cross-platform solutions
KernelContext

- Abstraction layer for WebCL inspired by PyCUDA
  - Minimizes WebCL/OpenCL boilerplate
  - OpenCL kernels are first-class citizens
  - Lazy evaluation utilizes the OpenCL command queue.
var ctx = new KernelContext;
var source_str = "__kernel void FN_NAME(...) {...}"
var kernel = ctx.compile(source_str, 'FN_NAME');

var data = new Uint32Array(10);
var d_data = ctx.toGPU(data);
kernel({local: 32, global: 32}, d_data);
ctx.fromGPU(d_data, data);
Dynamically generate kernels following common patterns

- `mapKernel`, `reduceKernel`
  - Generate a re-usable map or reduce kernel
  - "Templated" on map/reduce operation
  - Hides complexity – job size, multiple launches, etc.

- `map`, `reduce`
  - Generate and launch a single-use map or reduce kernel
var ctx = new KernelContext;
var util = new KernelUtils(ctx);

var a1 = new Uint32Array(10);
var result = util.map('x', 'x[i] + 1', a1);

var a2 = new Uint32Array(100000);
var result = util.map('x', 'x[i] * 0.43', a2);
var ctx = new KernelContext;
var util = new KernelUtils(ctx);

var a1 = new Uint32Array(10);
var sum1 = util.reduce('a + b', a1);
var max1 = util.reduce('(a > b) ? a : b', a1);

var a2 = new Uint32Array(100000);
var prd2 = util.reduce('a * b', a2);
var min2 = util.reduce('(a < b) ? a : b', a2);
```javascript
var sum_kernel = util.reduceKernel(Uint32Array, 'a + b');
var max_kernel = util.reduceKernel(Uint32Array, '(a > b) ? a : b');

var a1 = new Float32Array(100000);
var d_a1 = ctx.toGPU(a1);

var sum2 = sum_kernel(d_a1);
var max2 = max_kernel(d_a1);
```
Built on KernelContext to provide a re-usable framework for volunteer computing applications

- CrowdCLient
  - Client library – generate results via WebCL
- CrowdServer
  - Server library – collect results, aggregate data
CrowdCL Architecture

Client

- Javascript
  - CrowdCLient
    - KernelContext
    - WebCL

OpenCL

- CPU
- Devices

Web Browser

Network

CrowdServer

MongoDB

MySQL
CrowdClient

- Execute code in the background of a web page
- Send batched results to CrowdServer

- Acts like a Thread class:
  - Define a `run` method that generates results for a problem
  - API to `pause`, `resume`, and `sleep` execution.
CrowdServer

- RESTful Node.js application to aggregate CrowdClient results
- Supports both MongoDB and MySQL to store data
Thomson Problem

- Thomson problem: nonlinear optimization problem, useful in many problems in biology, math, physics, and computer science

- Lowest energy configurations of $N$ repelling charges on a sphere
  - Force (gradient) and energy require $O(N^2)$ computation.
  - Number of local minima grows exponentially with $N$
Thomson Problem

Let \( \omega_N = x_1, \ldots, x_N \) with \( \|x_i\| = 1 \) and

\[
E_s(\omega_N) = \sum_{x,y \in \omega_N, x \neq y} \frac{1}{\|x - y\|^s}
\]

Gradient descent:

- Compute the gradient (force on each point \( x \in \omega_N \))

\[
G(\omega_N)[x] = \sum_{y \in \omega_N, y \neq x} \frac{x - y}{|x - y|^3}
\]

- Compute a (heuristic) step-length

\[
ds := f(\omega_N, G(\omega_N))
\]

- Update all points in \( \omega_N \) and renormalize

\[
x := x + ds \cdot G(\omega_N)[x]
\]

\[
x := \frac{x}{\|x\|}
\]
NVIDIA Tesla K20

![Bar chart showing interactions per second vs. number of particles, N. The chart compares Javascript, Opt Java, and WebCL.](chart.png)
Kernel Performance (NVIDIA 320M)

- Force
- Update
- Reduction

Runtime (milliseconds)

Number of particles, $N$

- 512
- 1024
- 2048
- 4096
- 8192
- WebCL only available on Firefox 19 + plugin
- Nice, big security issues for general deployment
Thank you

https://github.com/tmacwill/webcl-kernelcontext
https://github.com/tmacwill/crowdcl