Literrature review

Contents

Introduction

In order to keep up with the evolution and continuous growth of the Internet, web technologies have been undergoing significant upgrades. Since 2007, the World Wide Web Consortium (W3C) has been working on a major update of the core language of the web that renders and displays all web contents. This is known as the 5th revision of Hyper Text Markup Language (HTML5). [1]

HTML's outstanding performance concerning power consumption and security made it the most likely candidate for cross-platform applications. However the slow performance of JavaScript in performing dynamic operations is a serious limiting factor to wider use. [2] Improving the efficiency of JavaScript is an active field of research. The discoveries made in the field of distributed computing look encouraging.

This paper provides a review of the literature on HTML5's WebSocket protocol and its use of distributed computing. It is organized as follows. Section one focuses on the improvement brought by OpenCL to distributed computing in browsers. Section two studies more in depth HTML5's WebSocket protocol.

1 OpenCL

1.1 JavaScript

Historically web browsers have always relied on JavaScript to provide dynamic content and user interfaces. JavaScript provides a portable way to distribute applications. However as a downside, it does not exploit task-parallel and data-parallel computing capabilities available on modern GPUs and CPUs. [3]

Since the most recent web applications are supposed to be capable of running rich media and graphics applications, several researches for accelerating JavaScript have initated. The performance issue is becoming more and more critical because of the appearance of very high resolution displays such as Full HD and Ultra HD. It is because higher resolution of images requires much higher computing capability. [4]

OpenCL is one of the most promising technologies for JavaScript acceleration. [2]

1.2 History of OpenCL

Open Computing Language (OpenCL) by contrast with JavaScript is explicitly designed for parallel computing. [3]

Historically, the first technology to take advantage of the massive parallel nature of GPUs was Open Graphic Library (OpenGL). OpenGL is an application programming interface (API) for rendering 2D and 3D vector graphics. Through the insertion of little pieces of C-like codes in shader, programmers realized graphic processing units (GPUs) could also be used for general programming. This became known as General Purpose computation on GPUs (GPGPU). [5]

However, shader can only provide limited features for such applications. As the need for more complex applications arose Apple proposed the Khronos Group to develop a more general framework OpenCL. [5] OpenCL is a low-level API accelerating applications with task-parallel or data-parallel computations in a heterogeneous computing environment. [6] Indeed OpenCL allows not only the usage of CPUs but also any processing devices like GPUs, DSPs, accelerators and so on.

OpenCL doesn't guarantee a particular kernel will achieve peak performance on different architectures. The nature of the underlying hardware may make induce different programming strategies. Multi-core CPU architecture is definitely the more popular. But the recent specification published by Khronos [7] to take GPU computing to the web is bound to raise programmers interest toward GPUs architecture.

1.3 Hardware architecture

1.3.1 CPUs architecture

Modern CPUs are typically composed of a few high-frequency processor cores. CPUs perform well for a wide variety of applications, but they are optimal for latency sensitive workloads with minimal parallelism. However, to increase performance during arithmetic and multimedia workloads, many CPUs also incorporate small scale use of single-instruction multiple-data (SIMD). SIMD can not be programmed directly using plain C. It requires to call vectorized subroutines. Routines supported by OpenCL. [6]

1.3.2 GPUs architecture

Contemporary GPUs are composed of hundreds of processing units running at low frequency. As a result GPUs are able to execute tens of thousands of threads. It is this ability which makes them so much more effective then CPUs in a highly parallel environment. Some research even claims a speedup in the order of 200x over JavaScript. [4]

The GPU processing units are typically organized in SIMD clusters controlled by single instruction decoders, with shared access to fast on-chip caches and shared memories. Massively parallel arithmetic-heavy hardware design enables GPUs to achieve single-precision floating point arithmetic rates approaching 2 trillions of instructions per second (TFLOPS). [6]

Although GPUs are powerful computing devices, currently they still often require management by a host CPU. Fortunately OpenCL was designed to be used in heterogeneous environment. It abstracts CPUs and GPUs as compute devices. After which, applications can query device attributes to determine the properties of the available compute units and memory systems. [6]

All the same, even if OpenCL's API hides the trickiest part of parallel programming a good understanding of the underlying memory model of OpenCL leads to more efficient codding.

1.4 OpenCL's memory model

1.4.1 Plateform model

CPU and GPU are called compute devices. A single host regroups one or more compute devices and has its own memory. Each compute device is composed of one or more cores also called compute units. Each compute unit has its own memory and is divided into one or more single-instruction multiple-data (SIMD) threads or processing elements with its own memory. [5]

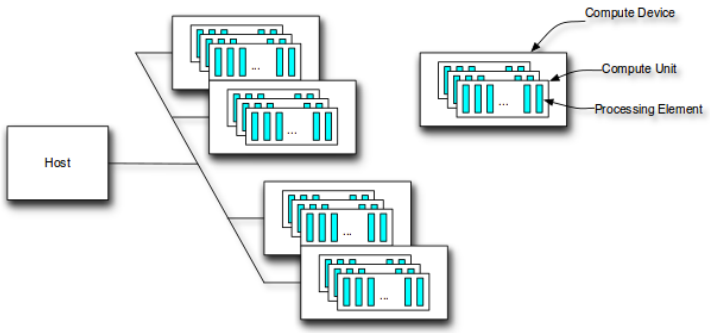

Figure 1: OpenCL platform model [5].

1.4.2 Memory model

OpenCL defines 4 types of memory spaces within a compute device. A large high-latency global memory corresponding to the device RAM. This is a none cached memory where the data is stored and is available to all items. A small low-latency read-only constant memory which is cached. A shared local memory accessible from multiple processing elements within the same compute unit and a private memory accessible within each processing element. This last type of memory is very fast and is the register of the items. [6] The last and least used kind of memory is texture memory, which is similar to global memory but is cached. [5]

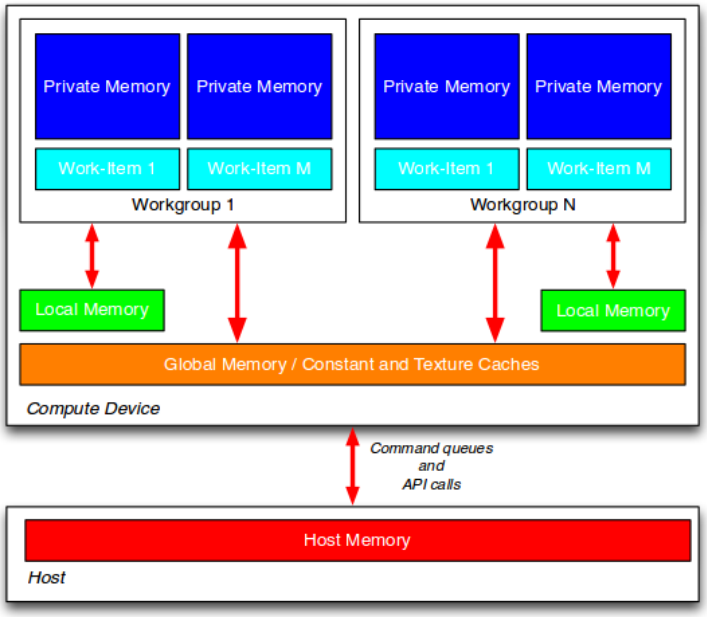

Figure 2: OpenCL memory model [5].

In conclusion, OpenCL provides a fairly easy way to write parallel code but to reach an optimal performance / memory access trade off programmers must choose carefully in where to save their variables in memory space.

1.4.3 Global and local IDs

Finally, at an even lower level, work-items are scheduled in work groups. This is the smallest unit of parallelism on a device. Individual work-items in a work group start together at the same program address, but they have their own address counter and register state and are therefore free to branch and execute independently. [5]

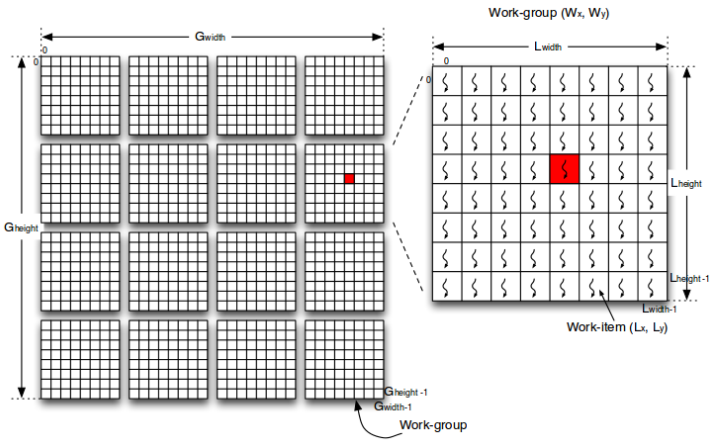

Figure 3: Global and local IDs for a 2D problem [5] .

On a CPU, operating systems often swap two threads on and off execution channels. Threads (cores) are generally heavyweight entities and those context switches are therefore expensive. By comparison, threads on a GPU (work-items) are extremely lightweight entities. Furthermore in GPUs, registers are allocated to active threads only. Once threads are complete, its resources are de - allocated. Thus no swapping of registers and state occurs between GPU threads. [5]

As can be deduced from this section in the underlying memory model, OpenCL is a fairly low-level API. Moreover the programming language used is a derivate of the C language based on C99. A language web developers will most likely be unfamiliar with. But Khronos anticipated this and developed the web computing language (WebCL). [7]

1.5 WebCL

WebGL and WebCL are JavaScript APIs over OpenGL and OpenCL's API. [7] This allows web developers to stay in an environment they are used to. [3]

In the first place, OpenCL was developed because of web browser's increasing need for more computational power. A necessity which arose from heavy 3D graphics applications such as on-line games and augmented reality. However, OpenCL doesnt provide any rendering capability, it only processes huge amounts of data. [5] That is why OpenCL was designed for inter-operation with OpenGL. WebCL/WebGL interoperability builds on that available for OpenCL/OpenGL.
WebCL provides an API for safely sharing buffers with OpenCL. This buffer is inside the GPU which avoids the back and forth copy of data when switching between OpenGL and OpenCL processes. [8]

GPU computing is quite a new notion. But it is a fast evolving field of research. Single GPUs are not enough anymore, the trend is moving towards GPU clusters.

1.6 GPU clusters

Most OpenCL applications can utilize only devices of the hosting computer. In order to run an application on a cluster, previously the program needed to be split to take advantage of all devices. Virtual OpenCL (VirtualCL) is a wrapper for OpenCL. VirtualCL provides a platform where all the cluster devices are seen as if located on the same hosting node. Basically, the user starts the application on the master node then VirtualCL transparently runs the kernels of the application on the worker nodes. Applications written with VirtualCL don't only benefit from the reduced programming complexity of a single computer, but also from the availability of shared memory and lower granularity parallelism. [9]

This first section dedicated to OpenCL introduced how web browsers intend to improve their computational capacities. The second section is on WebSockets. WebSockets is one of the major improvement of brought by HTML5. It provides a revolutionary communication way between client and server. It is also probably the most resource consuming protocol of HTML5 and thus one of the reasons OpenCL and WebCL were developed.

2 WebSocket protocol

2.1 Client server communications

The web has been largely built around the request/response paradigm of HTTP. A client loads up a web page and then the communication stops, nothing happens until the user clicks onto the next page.

With AJAX the web started to be more dynamic. The web browser could ask for an update of the pages, but it was still the same scheme: the browser polled from the server. Updates were either triggered by the user or by a periodic polling from the server. [10]

Later pseudo-push technologies like Comet enabled the server to send data to the client in the very moment when it knows that new data is available. The technique used to create the illusion that the server initiated the connection is called long polling. The goal is to keep the an HTTP connection open so that the server can send informations as soon as they are available.

However these technologies are only a temporary solution, because they carry the overhead of HTTP [10]. Every time a HTTP request is made a bunch of headers and cookie data are transferred to the server. For contemporary applications like on-line games latency is crucial, hence the need of creating a low latency connection that can support transactions initiated by either the client or server. [11]

WebSockets provide a new protocol between client and server which runs over a persistent TCP connection. Through this open connection, bi-directional, full-duplex messages can be sent between the single TCP socket connection. [12]

2.2 How to establish a WebSocket connection

For a client to establish a WebSocket connection, the first step is a process known as the WebSocket handshake. The client sends a regular HTTP request to the server informing him he wishes to establish a WebSocket connection. [13]

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
Host: websocket.example.com
Upgrade: websocket
```

If the server supports the WebSocket protocol, it sends a header in response.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 5 May 2014 04:04:22 GMT
Connection: Upgrade
Upgrade: WebSocket
```

After the completion of the handshake the WebSocket connection is active and either the client or the server can send data. The data is contained in frames, each frame is prefixed with a 4-12 bytes to ensure the message can be reconstructed. [11]

Once the WebSocket communication is over, the channel can be closed by either the client or the server.

Conclusion

HTML5 updates came just in time to answer the rising growth of the Internet. One of the most interesting new features is WebSocket. WebSocket is bringing real time communication to the web. However improving HTML capabilities brought up new problems. JavaScript, the scripting language in charge of rendering web content slowly reached its limits. Hence new technologies to make use of parallel computing and GPU computing have been developed. OpenCL and its JavaScript API WebCL is the most promising one.

My thesis aims to explore the extent to which WebSockets are improved by the use of distributes computing techniques.

References

[1] Robin Berjon. Html 5.1 working draft. *W3C*, february 2014. URL `http://www.w3.org/TR/html51/`.

[2] Myeongjin Cho. Web based image processing using javascript and webcl. 2014. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6776030`.

[3] Benedict R. Gaster. Heterogeneous computing with opencl. page 255, October 2012.

[4] Eero Aho. Towards real-time applications in mobile web browsers. *Master thesis in computing science*, 2012. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6507030`.

[5] Mikal Bourges-Svenier. Graphics programming on the web webcl course notes. *Siggraph*, 2012. URL `http://khronosgroup.github.io/siggraph2012course/WebCL/WebCL%20Course%20Notes.pdf`.

[6] John E. Stone. Opencl: A parallel programming standard for heterogeneous computing systems. *Comput Sci Eng.*, May 2010. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2964860/pdf/nihms220212.pdf`.

[7] Peter Bright. Khronos publishes a range of specs to take gpu computing to the web, c++. *ars technica*, March 2012. URL `http://arstechnica.com/information-technology/2014/03/khronos-publishes-a-range-of-specs-to-take-gpu-computing-to-the-web-c/`.

[8] Won Jeon. Webcl for hardware - accelerated web applications. *Advanced Technology Lab Samsung Information Systems America*, 2013. URL `http://dev.bowdenweb.com/html/api/webcl/webcl-for-hardware-accelerated-web-applications-dev-track-008.pdf`.

[9] A. Barack. The virtualcl cluster platform. *Mosix company.* URL `http://www.mosix.cs.huji.ac.il/vcl/VCL_wp.pdf`.

[10] Malte Ubl. Bringing sockets to the web. *html5 rocks*, October 2010. URL `http://www.html5rocks.com/en/tutorials/websockets/basics/`.

[11] Matt West. An introduction to websockets. *treehouse blog*, October 2013. URL `http://blog.teamtreehouse.com/an-introduction-to-websockets`.

[12] Joe Hanson. What are websockets ? *PubNub blog*, September 2013. URL `http://www.pubnub.com/blog/what-are-websockets/`.

[13] Nikolai Qveflander. Pushing real time data using html5 web sockets. *Masters Thesis in Computing Science*, August 2010. URL `http://www.diva-portal.org/smash/get/diva2:354621/FULLTEXT01.pdf`.