High Performance Computing in Web Browsers

Henning Lohse

***Abstract*—**

***Index Terms*—HPC, JavaScript, asm.js, HTML5, Web Workers, WebRTC, DataChannel, WebCL, WebGL, Compute Shaders**

I. Introduction

TODAY'S computational applications originate from broad interdisciplinary fields. These include meteorology, astro physics and molecular biology in scientific computing, as well as fluid dynamics, crash tests and distributed databases in commercial areas, or online games for consumers.

They share high demands to the computational platform. Results are prefered to be achieved as fast, precise and energy efficient as possible or required. Large data sets have to be handled accordingly. Techniques to meet these demands have been explored and aggregated in the field of High Performance Computing. (HPC)

Runtime and storage requirements of HPC applications traditionally exceed the capabilities of a single processor or machine. Therefore, parallel computing became the major factor for performance optimizations. Today, multiple heterogeneous processors can be used in interconnected machines. When designing an HPC application, the following aspects as illustrated in Figure 1 have to be considered:

- *Processor*. Determine which application parts should be executed on a distinct processor. This can be a core of a multicore CPU.
- *Intra-node processing*. A machine node might contain multiple sockets for CPU, where each CPU provides multiple cores. The application design should benefit from synergetic effects of caching, memory access patterns and locality while minimizing communication overhead. For programming, pthreads or OpenMP can be used.
- *Inter-node processing*. To meet HPC applications' requirements, multiple nodes are usually connected via network. Inter-node communication is orders of magnitude slower then intra-node communication. While synergetic effects to be considered here are similar to intra-node, communication overhead induces a more significant penalty. sockets or MPI can be used for implementations.
- *Coprocessors*. Complementary processors like GPUs can be used to improve performance for certain types of algorithms. GPUs for example are highly optimized for SIMD workloads with preferably short runtimes per data point on a huge set of the latter. CUDA or other coprocessor-dependent libraries can be utilized.

These aspects are tightly coupled. Designs considered for each one are dependent on the others. Furthermore, when implementing the HPC application, optimizations are usually done for a specific platform or range of platforms. Applications come as compiled binaries. While these approaches provides the best performance, portability is not available or time-consuming.

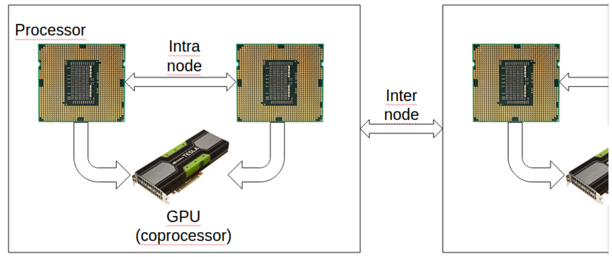

Fig. 1: HPC design aspects example

A. Motivation

Consumer electronics today are heterogeneous multiprocessor systems. Notebooks, Tablets and Smartphones are equipped with multicore CPUs, GPUs and wireless network technologies. While they run different and diverse operating systems hindering portability of native applications, each platform comes with a web browser.

Modern web browsers can be considered application platforms. They provide GUI rendering, network communication, user interaction and dynamic content control. The development of HTML5 introduced several improvements and new features, like threads or client-server sockets. Plenty of frameworks allowing developers to connect the browser frontend with a server backend exist. The JavaScript engines are constantly improved for faster script execution. And all these features work on virtually any platform with a modern web browser, resulting in inherent application portability.

The topic of this work is to present the current state of web browser technologies regarding HPC application design aspects processor, intra-node and inter-node processing. As coprocessors can be diverse and specialized, this work focuses on GPUs.

B. Section Overview

Section II gives a brief overview of the JavaScript language used in HTML scripts. Important concepts and constructs for further sections are discussed. The following section III introduces asm.js as an annotation-based JavaScript subset for performance optimizations. Intra-node processing capabilities come with HTML5 Web Workers in section IV. Section V presents WebRTC DataChannel and its inter-node processing capabilities. The state of GPUs as coprocessors using WebCL and OpenGL Compute Shaders is shown in section VI. Finally, conclusions are drawn in section VII.

II. JAVASCRIPT

JavaScript is the programming language used for scripting in HTML documents. Code written in Javascript resides in the script block of the HTML Domain Object Model (DOM), as seen in figure 2. This language's purpose is amongst others to allow dynamic reload of content on events, like clicks, to reduce bandwidth and allow user interactions. Input can also be verified and designs can be manipulated. These operations can be done by manipulating the DOM tree, yet arbitrary computations can also be done.

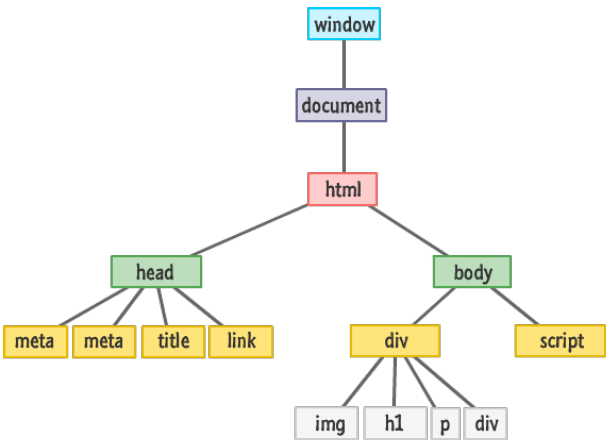

Fig. 2: HTML Domain Object Model [kirupa.com]

A. Type System

When looking at arbitrary computations, it is important to note the key characteristics of JavaScript's type system:

- *Dynamically typed*. Types of objects are resolved at runtime; static checks are not done. This means operations like late binding or downcasting are performed during execution. This allows developers to write code faster, but operations might fail at runtime due to e.g. missing operators.
- *Object-oriented*. As noted before, JavaScript utilizes objects. It is important to note that even CPU native types, like a 32 bit integer, are wrapped into JavaScript primitive type objects. While this simplifies development by adding utility functions, addtitional overhead occurs during runtime.
- *Classless*. Objects are not instantiations of pre-defined classes. They are defined as key-value pairs inside curly brackets, as seen in listing 1. This notation is refered to as the JavaScript Object notation. (JSON) It allows arbitrary hierarchical nesting of data structures, like additional objects or arrays.
- *Prototypes*. As JavaScript is classless, prototypes are used. If multiple objects with the same members are desired, a prototype object is created and copied. These copies can again be arbitrarily modified.

Listing 1: JavaScript type system example

```
var prototype = {
  "Publisher": "Xema",
  "ID": "1234-5678-9012-3456",
  "Owner": {
    "Name": "Max",
    "male": true,
    "Hobbys": ["Riding", "Golf"],
    "Age": 42,
  }
};

var copy = prototype;
copy["Currency"] = "EURO";
copy["Owner"]["Hobbys"].push("Reading");
```

B. Memory Management

JavaScript does not support explicit memory management. Functions like malloc or free, new or delete are not available. The web browser must provide a Garbage Collection mechanism. Their purpose is to automatically remove unused objects from memory. This makes development easier, as object ownership has not to be considered, and prevents memory leaks. Implementations in current web browsers are based on the mark-and-sweep algorithm. Basically, unreferenced in the sense of unreachable objects are to be removed.

Applications though may suffer from badly timed garbage collecting. JavaScript and current web browsers provide no interface to control the time a garbage collection occurs. Today's implementations are highly optimized. Still, real-time and memory intensive applications, like many HPC applications, have to be implemented carefully. Creation of too many objects should be avoided and patterns benefiting garbage collection should be used. [1]

III. ASM.JS

Text.

IV. INTRODUCTION

Text.

V. INTRODUCTION

Text.

VI. INTRODUCTION

Text.

VII. CONCLUSION

Text.

REFERENCES

[1] Mozilla Developer Network, *Memory Management*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management (05.01.2015)