FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen
Campus Jülich

Parallelsim in JavaScript Applications

Bachelorthesis

Thorben Below
Matr. No.: 3255003

Faculty of
Medical Engineering and Applied Mathematics
Applied Mathematics and Computer Science BASc

Aachen, August 2022

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

My name ____________________

signature

Diese Arbeit wurde betreut von:

1. Prüfer:	Prof. Dr. Karola Merkel
2. Prüfer:	Stephan Oehlert M. Sc.

This is the english abstract

Hier kommt die deutsche Zusammenfassung

0.1 Concurrency and Parallelism in the JavaScript Ecosystem

Moore's Law states the number of transistors doubles about every two years, though the cost of computers is halved. While this has been true for a long time, semiconductor advancement has slowed industry-wide since around 2010. A lot of the increase in compute power now comes from an increased core count in CPUs rather than a performance increase in single cores [**?**].

It follows that the importance of software that can make use of this increased number also rises. Most modern programming languages provide features that allow for concurrent execution of its code.
JavaScript is best-known as the scripting language of the Web, although it is also supported by many non-browser environments like Node.js or Deno. It is lightweight, interpreted or just-in-time compiled, prototype-based and features first-class functions.
In the past, the platforms that JavaScript ran on did not provide any thread support, so the language was thought of as single-threaded. Concurrency was achieved through the use of Callbacks and later, Promises and async/await that use the advantages of the event-based design of the language. Parallel programming was only possible in some environments by using other languages that were capable of invocing threads and interfacing with JavaScript code. Since then there have been a lot of changes to the language to support multi threading.

With the introduction of web, shared and service workers in the browser and worker threads in nodejs and deno, there are APIs to achieve real multithreading in all major environments. Communication between Threads or JavaScript contexts is done via message channels and the postMessage API. While this API allows sending JSON Objects and some builtin data types, it also has strict limitations when it comes to other objects. It is not possible to send Function Objects or preserve prototype information.[[2]](references) This greatly impacts the way multithreaded programs can be designed, especially for libraries (e. g. using polymorphy across thread bounds becomes non trivial).

The goal of this thesis is to understand the reasoning behind the restrictions regarding the communication between contexts in general and more specifically for Function objects. To answer this question, we will first give an overview of multithreading and the important concepts that go along with it. After that, the memory model underlying JavaScript and the way in which engines convert code into executable instructions will be looked at in more detail and the already existing and planned possibilities for interaction between threads will be presented. Subsequently, Function Objects (and the corresponding subtypes) will be considered in particular. In addition prototypically experiment(?) which dependencies and restrictions arise here with the transmission over thread bounds. (e. g. serialize/deserialze necessary?).

0.2 Literature

- Modern operating systems (Tanenbaum and Andrew)
- Multithreaded JavaScript - Concurrency Beyond the Event Loop (Thomas Hunter II and Bryan English)
- ecma262 specification (https://tc39.es/ecma262/)
- HTML specification - Web messaging (https://html.spec.whatwg.org/multipage/web-messaging.html)
- V8 documentation (https://v8.dev/docs)

Contents

1 Concurrency

1.1 Processes

1.2 Threads

1.3 Interprocess Communication

2 JavaScript

JavaScript (JS) is a general purpose programming language that is best-known as the scripting language for the Web and is defined by the ECMAScript Specification. More precisely by the ECMAScript Language Specification (ECMA-262) and the ECMAScript Internationalization API Specification (ECMA-402) [**?**].

2.1 Core concepts

JavaScripts core concepts include being prototype-based, supporting multiple paradigms such as object-oriented and functional programming and having higher order functions [**?**]. The following sections give a brief overview over the most important core concepts.

2.1.1 Inheritance and the prototype chain

2.1.2 Higher order functions

2.1.3 Closures

2.1.4 Strict mode

2.1.5 Event Loop

2.1.6 Memory management

2.2 Compilation

2.3 Parallelism in JavaScript

ECMAScript itself has no notion of Threads. It only has the concept of Realms and execution contexts. Instead the Engines and envrionments that JavaScript is executed in provide methods to create new execution contexts (Threads) and interact with them.

2.3.1 Browser specific

In the Browser these are different kind of Workers. Currently there are Web, Shared and Service workers. The Web Worker is akin to a "normal" Thread. It lives only as long as its parent Context and can only communicate with it and contexts that are related to the parent context.
The Shared Worker allows communication with multiple "Parent" contexts. It lives as long as atleast one parent context is still alive.
The Service Worker can persist even after the parent context is already closed. It can communicate with all other contexts that have the same origin.

2.3.2 Serverside

nodejs and deno provide Worker Threads. Theses behave similar to threads known in other programming languages.

List of abbreviations

ABM	Antibiotic/Antimycotic
$AcLDL$	acetylated low density lipoprotein
EGF	Epidermal growth factor
$EGM-2$	Endothelial cell growth medium 2
$ELISA$	Enzyme-linked immunosorbent assay
$FACS$	Fluorescence-activated cell sorting
HE	Hematoxylin and eosin
$HEPES$	4-(2-hydroxyethyl)-1-piperazineethanesulfonic acid
RNA	Ribonucleic acid
RT	Room temperature
vWf	Von Willebrand factor

List of Figures

List of Tables

Danksagung

Wenn Du hier auf Deutsch schreiben möchtest, wird durch den Befehl ’selectlanuage’ die deutsche Rechtschreibung bzw. Silbentrennung aufgerufen.

Appendix

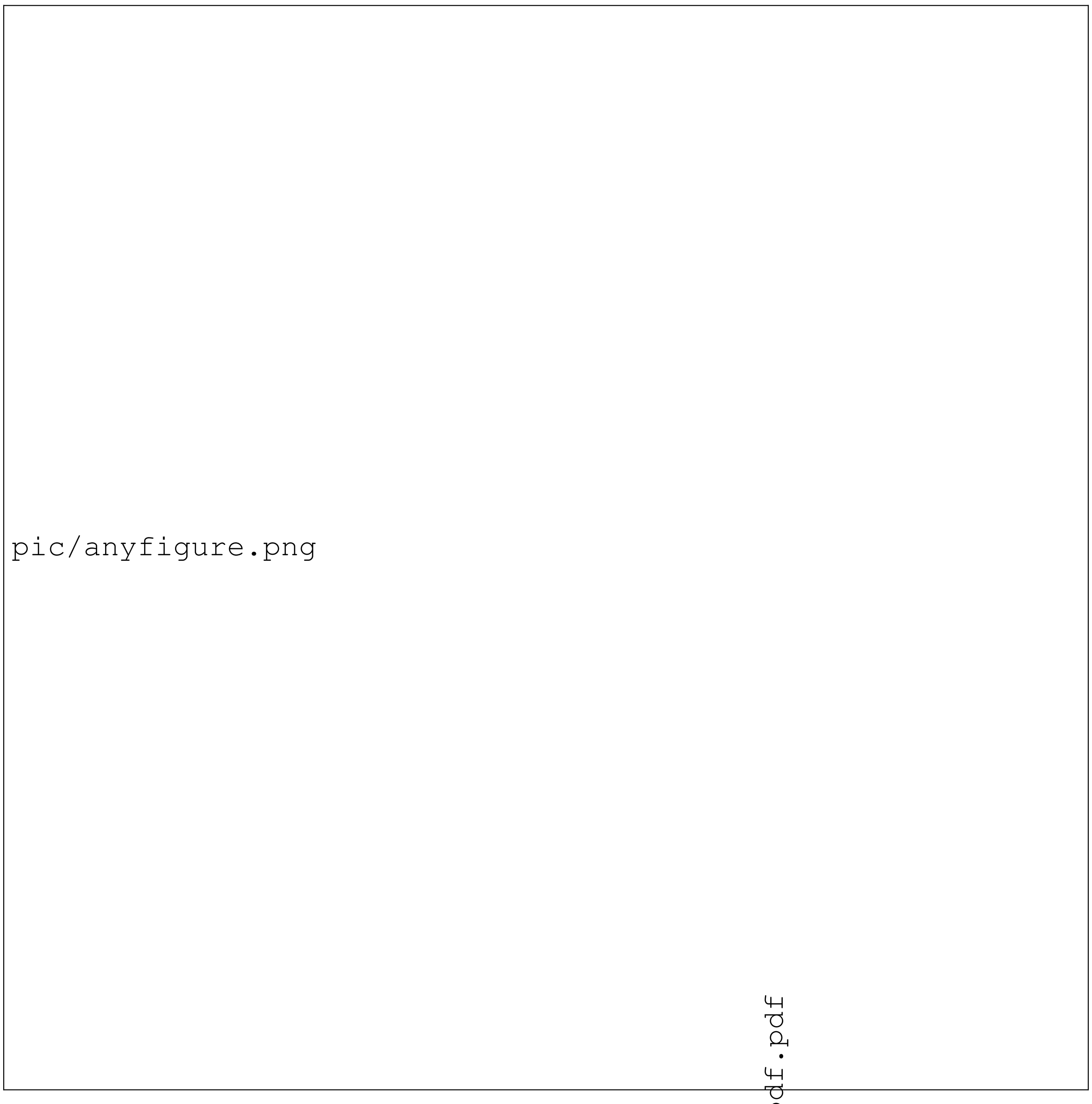

Abbildung 1: *Some cool photo taken in the lab.*

Abbildung 2: *It is possible to add pdf files as figure, while keeping your header and footer from your document. Mostly useful in the appendix.*