WASM vs Javascript vs C++

Final Assignment

COMP 7615

Peyman Tehrani Parsa, A00922386

Table of Contents

Hypothesis

WebAssembly or wasm is a new binary format that runs on all modern browsers. Since it's release multiple videos have come out showcasing how wasm out performs Javascript in embedded applications like unity, autocad and, face-detection. However not many benchmarks can be found showing the performance of common algorithms and how they achieved their results. I set out to show how much faster wasm is to Javascript in three algorithms, merge-sort, radix-sort and, calculating the 94th Fibonacci number.

Method

First I started by programing the same algorithms in Javascript and WebAssembly. For Fibonacci I implemented a non-recursive algorithm for calculating the 94th number. I decided on the 94th because after that an overflow would happen due to the max size of a unsigned 64-bit integer. For Merge-sort I would order an int array of 242880 unique numbers. Finally for radix I would order an int vector of 1000000 unique numbers. All tests repeat their algorithms ten times to get the largest delta between execution times.

Building

For convenience a bash file called "make" will compile all the C++ files to WebAssembly but if you wish to run it on your own files, the commands I used are listed below and what they do.

C++

[g++ -O3 x.cpp -o x.o]

WASM

[emcc x.cpp #input file
--emrun \ #generate Module to enable capture of stdout, stderr and exit().
-O2\ #various JavaScript-level optimizations and LLVM -O3 optimizations
-s WASM=1\ #Default for compiling WebAssymbly
-s ALLOW_MEMORY_GROWTH=1\ #total amount of memory used to change depending on the demands of the application
-Wall\ #Compiling warnings enable
-s MODULARIZE=1\ #minifies globals to names that might conflict with others in the global scope
-s ASSERTIONS=1\ # enable runtime checks for common memory allocation errors (e.g.

writing more memory than was allocated). It also defines how Emscripten should handle errors in program flow.
-o x.js] outputfile

Testing

For convenience a bash file called “test” will run a all benchmarks and print the output to terminal.

Before you start testing first make sure that cgmemtime is installed for CPU time and memory usage measurements and node to run WebAssembly+Javascript. Download links are provided in tools section.

For testing WebAssembly I tested two different methods for running main function. First method I will call ‘EMSC index’ is loading the Javascript and WebAssembly exported by Emscripten than running the main function. Second method is running node directly on the exported Javascript file. Due to the programs compiled containing a main method, it will automatically run. Note that this will not work if you have functions other than main.

Tests

1. For the javascript tests run ‘cgmemtime node script.js’
2. For C++ test run ‘cgmemtime binary’
3. ‘EMSC index’ method run ‘cgmemtime node index.js’. An index file is provided in the source files for each algorithm.
4. EMSC js method run ‘cgmemtime node test.js’. A test.js file is provided in the source files for each algorithm.

Conclusion

In all my tests wasm beat plain Javascript in all execution time; At its slowest still being x4 faster. I used two methods to use wasm files. The first being calling node directly on the compiled javascript file created by Emscript. That javascript file as all the assertions needed to execute a C/C++ file that has include statements.

Getting the current compile is heavily flag dependent

Tools

- Emscripten SDK is required for c/cpp to wasm compilation read how to install it here
- Node is required to run javascript in terminal
- cgmemtime used to measure CPU and memory usage

Full test results

Fibonacci

	CPU Time (s)	Memory (KiB)
JAVASCRIPT	0.022	28748
GCC C++	0.001	3084
EMSC index	0.022	32428
EMSC js export	0.005	29996
*WASM direct	0.051	32412

*Possible because it did not require any libraries

Merge sort

	Time (s)	Memory (KiB)
JAVASCRIPT	0.263	93432
GCC C++	0.189	3120
EMSC index	0.546	34052
EMSC js export	0.017	29604

Radix sort

	Time (s)	Memory (KiB)
JAVASCRIPT	6.429	141056

GCC C++	0.582	15052
EMSC index	2.729	60700
EMSC js export	0.022	29604

Hardware

OS: Ubuntu 18.10 x86_64

Kernel: 4.18.0-11-generic

CPU: Intel i7-7700K (8) @ 4.500GHz

GPU: NVIDIA GeForce GTX 1070

Memory: 32124MiB