UDC 004.4

WEB ASSEMBLY – THE WAY TOWARDS THE NEW HORIZONS OF PERFORMANCE

MAKSIM BALABASH, DMITRY PIATKIN
Polotsk State University, Belarus

We consider two sets of implementations of the algorithm to find Fibonacci numbers. One of them is represented by usual JavaScript functions, the second one is written in C and converted into the WebAssembly module. This will compare the performance of wasm and JS in solving similar tasks.

We will explore three approaches of finding the Fibonacci numbers. The first uses a cycle. The second uses recursion. The third is based on the technique of memoization. All of them are implemented in JavaScript (Fig. 1) and in C (Fig. 2).

```
function fibo...s (num) {
  var a = ..., b = 0, temp;
  while (num ... 0){
    temp = a;
    a = a + b;
    b = temp;
    ...--;
  }
  return b;
}

const fiboJsRec = (num) => {
  if (num <= 1) return 1;
  return fiboJsRec(num - 1) + fiboJsRec(num - 2);
}

const fiboJsMemo = (num, memo) => {
  memo = memo || {};
  if (memo[num]) return memo[num];
  if (num <= 1) return 1;
  return memo[num] = fiboJsMemo(num - 1, memo) + fiboJsMemo(num - 2, memo);
}
module.exports = { fiboJs, fiboJsRec, fiboJsMemo };
```

Fig. 1. Javascript functions implementation

```
int fibonacci(int n) {
  int a = 1,
  int b = 1;
  while (n-- > 1) {
    int t = a;
    a = b;
    b += t;
  }
  return b;
}

int fibonacciRec(int num) {
  if (num <= 1) return 1;
  return fibonacciRec(num - 1) + fibonacciRec(num - 2);
}

int memo[10000];
int fibonacciMemo(int n) {
  if (memo[n] != -1) return memo[n];
  if (n == 1 || n == 2) return 1;
  else {
    return memo[n] = fibonacciMemo(n - 1) + fibonacciMemo(n - 2);
  }
}
```

Fig. 2. C functions implementation

We will not discuss the details of implementation here, we have another main goal. Before proceeding to a practical example, let us elaborate on the features of technologies relevant to the research.

WebAssembly is an initiative to create a safe, portable and fast to download and use a code format suitable for the Web. WebAssembly is not a programming language. This is the compilation target, which has text and binary format specifications. This means that other low-level languages, such as C / C ++, Rust, Swift, and so on, can be compiled into WebAssembly. WebAssembly gives access to the same API as browser JavaScript, seamlessly integrated into the existing stack of technologies. This distinguishes wasm from something like Java applets. The architecture of WebAssembly is the result of teamwork of the community, in which developers of all leading web browsers took part. Emscripten is used to compile the code into the WebAssembly format.

Emscripten is a compiler from the LLVM bytecode in JavaScript. That is, using it you can compile into JavaScript programs written in C / C ++ or any other languages, the code on which you can convert to LLVM. Emscripten offers a set of APIs for porting code in a format suitable for the web. This project has been around for many years, mostly it uses to transform games into their browser versions. Emscripten allows you to achieve high performance due to the fact that it generates code that meets the standards of Asm.js, which is lower, but recently it was successfully equipped with WebAssembly support.

Asm.js is a low-level, optimized subset of JavaScript that provides linear access to memory using typed arrays and supporting annotations with information about data types. Asm.js makes it possible to improve the performance of solutions. This is also not a new programming language, therefore, if the browser does not support it, Asm.js-code will be executed as normal JavaScript, that is, it will not be possible to get performance gains from its use.

Let's turn the program written in C into a wasm format. In order to do this, use the option to create standalone WebAssembly modules. With this approach, the output of the compiler is only a file with the WebAssembly code, without additional .js files.

This approach is based on the concept of additional modules (side module) Emscripten. It makes sense to use such modules, since they, in essence, are very similar to dynamic libraries. For example, system libraries do not automatically connect to them, they are some self-contained blocks of code issued by the compiler.

After receiving the binary file, we need only upload it to the browser. In order to do this, the WebAssembly API provided a WebAssembly level object, which contains the methods needed to compile and create an instance of the module.

ArrayBuffer. The buffer contains initial binary data of fixed length. You can not execute them directly, which is why in the next step the buffer is passed to the WebAssembly.compile method, which returns WebAssembly.Module, an instance, in the end, you can create using WebAssembly.Instance.

Let's use our wasm-module to test its performance and compare it with the speed of JavaScript. The number 40 will be fed to the input of the functions under study.

Testing results: • JS x 8,605,838 ops / sec ± 1,17% (55 runs are selected) • JS recursive x 0.65 ops / sec ± 1.09% (6 samples) • JS memoization x 407,714 ops / sec ± 0,95% (sampling of 59 runs) • Native loop x 11,166,298 ops / sec ± 1,18% (sample 54 samples) • Recursive recursive x 2.20 ops / sec ± 1.58% (sample of 10 runs) • Primary memorialization x 30,886,062 ops / sec ± 1,64% (56 runs are selected) • Fastest: Collecting Memory • Slowest: JS recursive

It is noticeable that the wasm-code obtained from the C programs (in the test output it is designated as "Native") is faster than the similar code written in ordinary JavaScript ("JS" in the test output). At the same time, the fastest implementation was the wasm function of searching for Fibonacci numbers, which uses the technique of memoization, and the slowest is the recursive function in JavaScript.

Research Outcomes: • The best performance in C implementation is 375% faster than the best implementation on JS. • The fastest option on C uses memoization. On JS is an implementation of an algorithm using a loop. • The second highest performance in C is still faster than the fastest version on JS. • The slowest implementation of the algorithm in C by 338%

REFERENCES

1. WebAssembly project on GitHub [Electronic resource] / GitHub - web-based hosting service for version control using git. – Mode of access: https://github.com/WebAssembly/design/blob/master/JS. – Date of access: 15.02.2018.
2. Emscripten project on GitHub [Electronic resource] / GitHub - web-based hosting service for version control using git. – Mode of access: https://github.com/kripken/emscripten/wiki/Linking. – Date of access: 10.02.2018.

3. WebAssembly official website [Electronic resource]. – Mode of access: http://webassembly.org/docs/semantics. – Date of access: 10.02.2018.
4. WebAssembly official Developer's Guide [Electronic resource] / Wikipedia – The Free Encyclopedia. – Mode of access: http://webassembly.org/getting-started/developers-guide. – Date of access: 20.01.2018.