An Argument for Thread-Level Speculation and Just-in-Time Compilation in the Google's V8 JavaScript Engine

Jan Kasper Martinsen and Håkan Grahn
Blekinge Institute of Technology
Karlskrona,Sweden
{jkm,hgr}@bth.se

Anders Isberg
Sony Mobile
Lund, Sweden
Anders.Isberg@sonymobile.com

ABSTRACT

Thread-Level Speculation can be used to take advantage of multicore architectures for web applications. We have implemented Thread-Level Speculation in the state-of-the-art JavaScript engine V8 instead of using an interpreted JavaScript engine. We evaluate the implementation with the Chromium web browser on 15 popular web applications for 2, 4, and 8 cores. The results show that it is beneficial to combine Thread-Level Speculation and Just-in-time compilation and that it is possible to take advantage of multicore architectures while hiding the details of parallel programming from the programmer of web applications.

1. IMPLEMENTATION OF THREAD-LEVEL SPECULATION IN GOOGLE'S V8 JAVASCRIPT ENGINE

The main idea of Thread-Level Speculation (TLS) in JavaScript for web applications is to try to execute JavaScript function calls as threads. Like the implementation in [1] we support nested speculation. V8 employs Just-in time compilation (JIT) but global variables are not compiled into the code that are to be executed. Instead these variables are accessed from function calls in the compiled code, such as *StoreIC_Initialize* and *LoadIC_Initialize*. This means that the global JavaScript stack and the native stack are accessed separately. We save the JavaScript stack right before we speculate, in case of a rollback. On a rollback the function is already ready for reexecution since a function get compiled, it is placed in a cache. However, rollbacks are relatively rare, as we see in [1]. When the function return, we commit the values back to its parent.

Since we do not add features to the native code, and since all JavaScript global variables are accessed through external functions, we do not need to reinterpret the code upon rollbacks, instead we simply execute the compiled function. The result is that the execution time for rollbacks decreases with V8.

CF'14, May 20-22 2014, Cagliari, Italy
Copyright 2014 ACM 978-1-4503-2870-8/14/05
http://dx.doi.org/10.1145/2597917.2597950 ...$15.00.

2. RESULTS

There are a large number of JavaScript function calls in web applications, and each function call is small in terms of the number of instructions executed. It does not improve the execution time that they are executed as native code instead of being interpreted, since the number of JavaScript lines is often very small, reuse of compiled code are rare and improved execution time does not outweigh the compilation time. This is not an argument against JIT, but it is an argument against the benchmarks, which suggested that JIT compilation for JavaScript should be designed in this way.

The average speedups for 2, 4 and 8 cores are 1.2, 2.9 and 4.7 respectively and the maximum and minimum speedup times, are 2.0 and 0.02 for 2 cores, 4.3 and 1.0 for 4 cores and 6.5 and 2.9 for 8 cores.(In Figure 1). Therefore we need more than 2 cores to take advantage of TLS in combination with JIT in web applications.

amazon (2) is twice as fast on 2 cores since this use case executes the largest maximum number of threads and has the largest average number of threads for this web application. When we log into *Amazon*, there is a personalization in terms of client side functionality which increases the use of JavaScript. This gives us the possibility to find many events / functions to speculate on, and we know from previous results that there will be little dependencies between such functions. We also see that this is the result of the sum of previous uses of *Amazon*, which makes *Amazon* to use JavaScript according to the previous uses.

Wikipedia has the slowest execution time with TLS+JIT. The JavaScript that is executed in this use case is limited in terms of number of lines of JavaScript code. However, we do not know this when we enter the web application, so we still try to speculate aggressively. This means that we set up the entire TLS, with thread-pool etc, for a use case where it turns out that there are few JavaScript function calls to speculate on, which means that the costs of setting up the system to use TLS will outweigh the gain in execution time by speculating on function

For 4 cores the performance is doubled for 43 out of 45 use cases. The two use cases where it is less than twice as fast, are *bing (1)* and *wikipedia (1)*. Both of these are the front page of the web application with a small number of executing threads. There is little interaction in these use cases and there is a need for some interaction to take advantage of JavaScript with TLS, as interaction allows us to execute more event generated JavaScript functions. For half of the use cases, the speedup is three folded with 4 cores. If we look

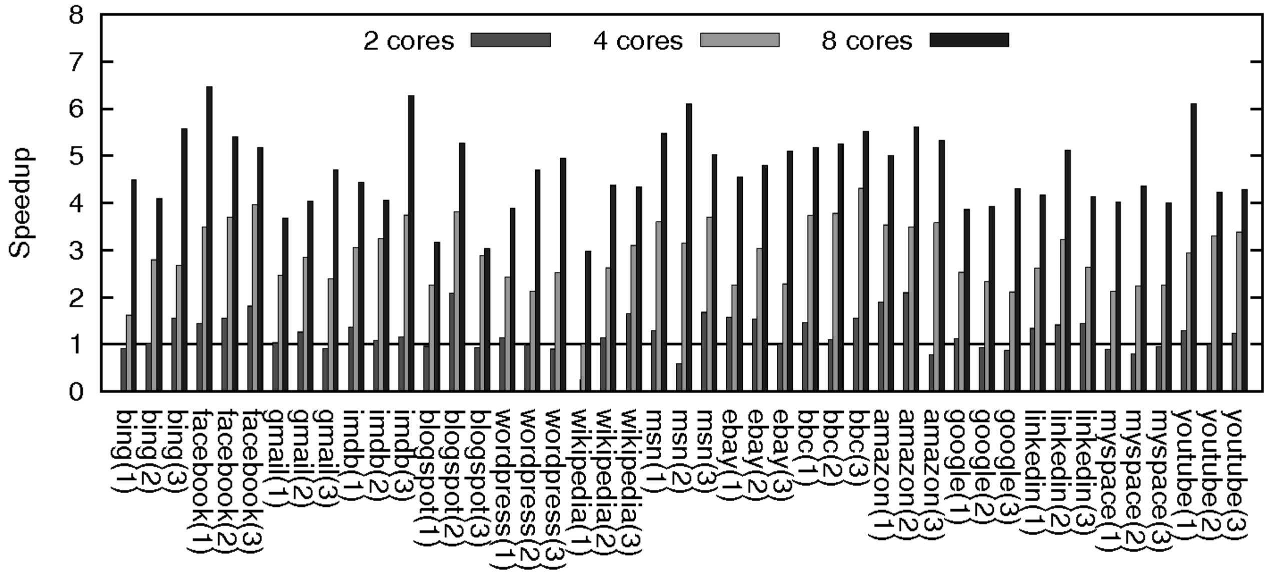

Figure 1: The speedup for 2, 4 and 8 cores on 45 use cases in 15 web applications.

at the use cases, this applies in general to certain JavaScript intensive web applications (such as *Amazon*, *BBC*, *MSN*, *Imdb*, *Facebook*). It also, in general applies to use cases 2 and 3, where there is more interaction than for use case 1 (for instance in use case 3 we search for one of the author of this paper).For the *bbc (3)* use case the execution time is four times the execution time on 4 cores, this seems to be the "news" page that is rapidly updated thanks to JavaScript with "news tickers". These are quite independent of one another which make speculation successful.

For 8 cores, we are able to at most six double the execution speed for the use cases *Facebook*, *Imdb*, *msn* and *Youtube*. This improvement is found in both use case 2 or 3 (except for *Facebook* and *Youtube*).

Increased interactivity in the web application increases the speedup with TLS, as this will increase the number of events, which in turn increases the number of executed JavaScript function calls. We also see that the speedup and number of cores ratio is the highest with four cores, this can be understood by that there is a limit to the number of events/function calls in each event, however the amount of interaction increases as the use cases become more advanced.

The average maximum number, the average number of threads and the distance between them is: 204, 123 and 81 (Figure 2). There is an average difference of 40% between the maximum number of threads and the average number of threads, which shows that the functions are short lived. Deviations are *blogspot(3)*, *wikipedia(1)* and *msn(2)* where the number of speculations are 5 and 17.

For *msn(2)*, the number of speculations are relatively close to the average number of speculations for the other use cases, but several of the functions are very large in terms of instructions to be executed. This makes the average number of functions executing low compared to the maximum number of functions executing. The functions in *msn* have a low number of writes, which in turn makes them easy to speculate on. This could be caused by the *msn* use case since it has several "tickers" and the functionality in terms of JavaScript is events which are repeatedly called. As we see in Figure 1 this creates one of the largest speedups which are over 6 times faster than the sequential execution time of V8 with 8 cores.

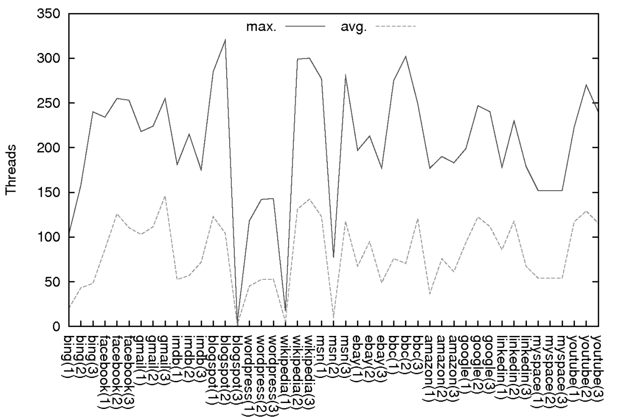

Figure 2: The maximum and the average number of threads during execution.

3. CONCLUSION

The programming model in web applications makes TLS appropriate for taking advantage of hardware with more than 2 cores and certain features in V8 are very useful for TLS.

We present the first implementation of TLS+JIT, and shown that this yields significant speedups without any changes to the JavaScript code. We strongly believe that combining TLS+JIT is a very promising approach to enhance the performance of JavaScript in web applications.

4. REFERENCES

[1] J. K. Martinsen, H. Grahn, and A. Isberg. Using speculation to enhance javascript performance in web applications. *IEEE Internet Computing*, 17(2):10–19, 2013.