COP701 - Assignment 3 (Part 1)
WebAssembly

Nilaksh Agarwal 2015PH10813
Navreet Kaur 2015TT10917

Hardware Specifications:

Model Name:	MacBook Pro
Processor Name:	Intel Core i7
Processor Speed:	3.3 GHz
Number of Processors:	1
Total Number of Cores:	2
L2 Cache (per Core):	256 KB
L3 Cache:	4 MB
Memory:	16 GB

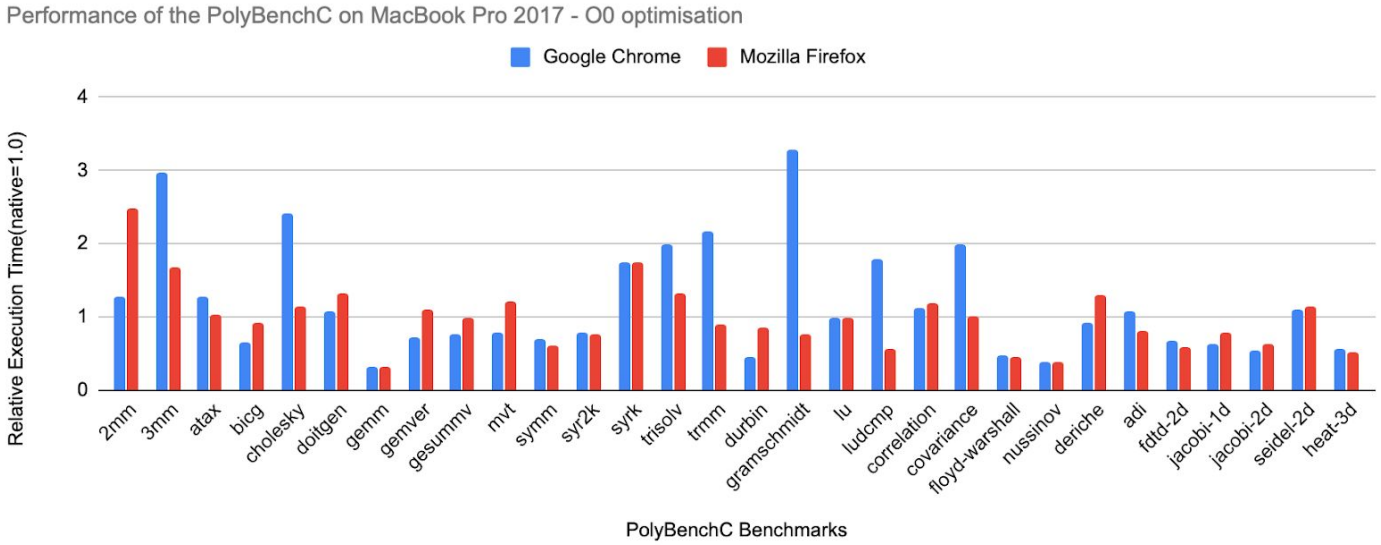
Browsers Used:

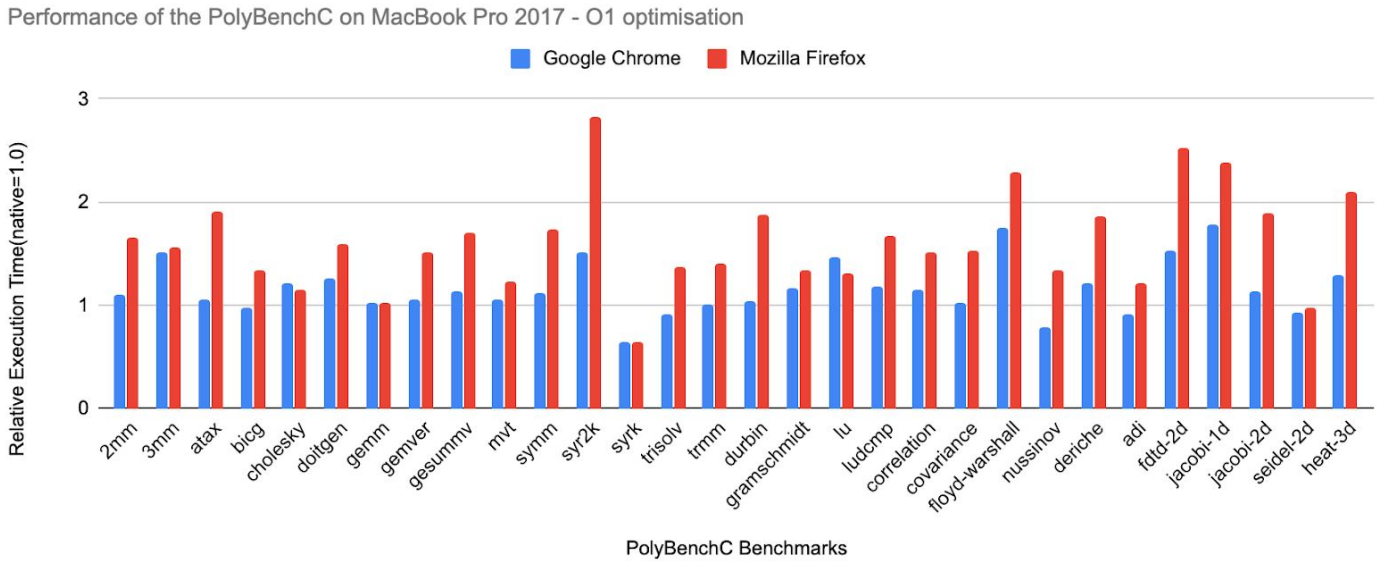
Google Chrome
Mozilla Firefox

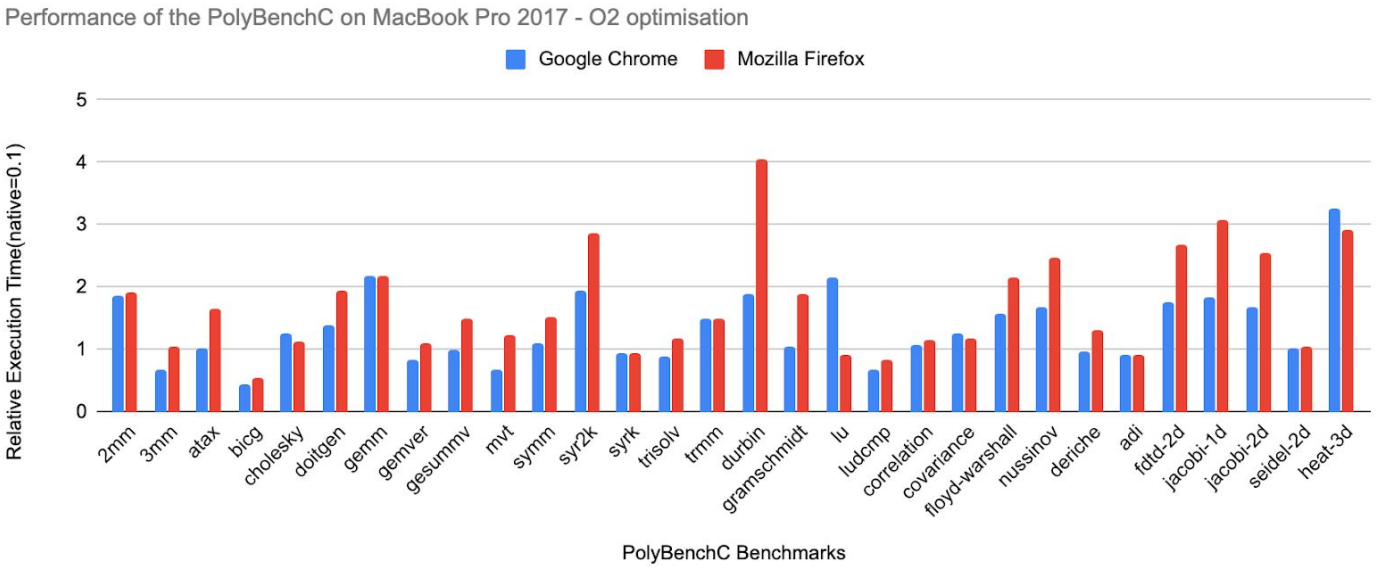
Scripts written:

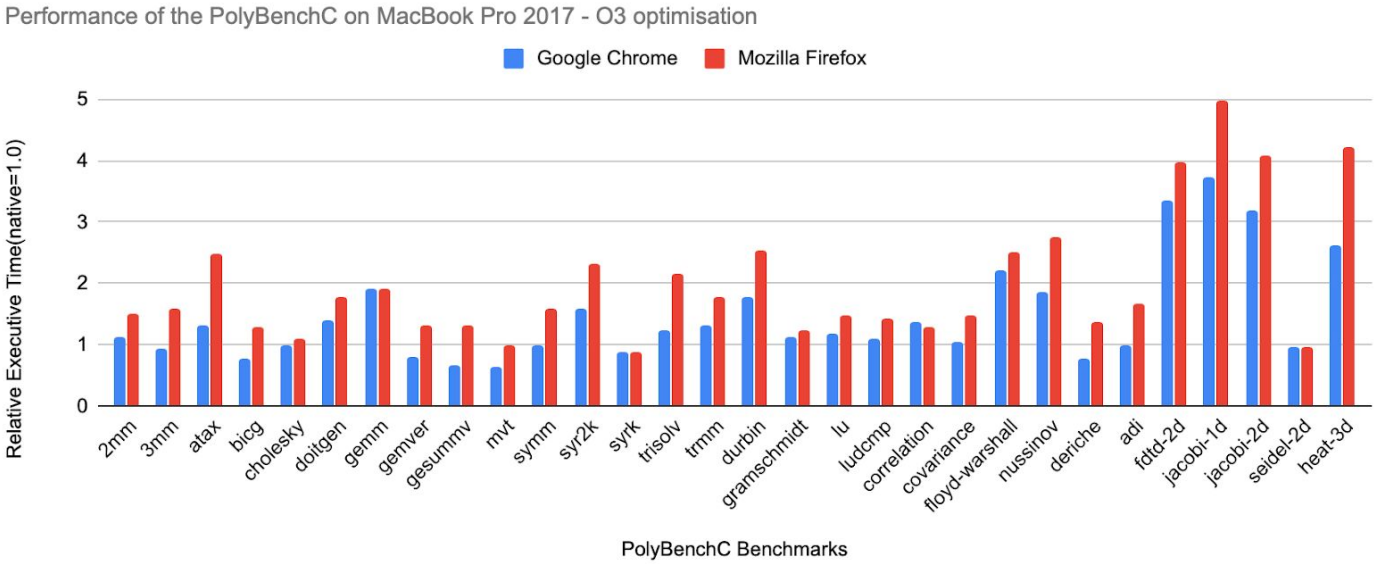
Generating executables (both gcc and emcc) with and without timing analysis
Running and timing native code
Generating Instruction mix, dynamic instruction count and hot code in wasabi and pin tools

Experiment 1- Relative Execution Time





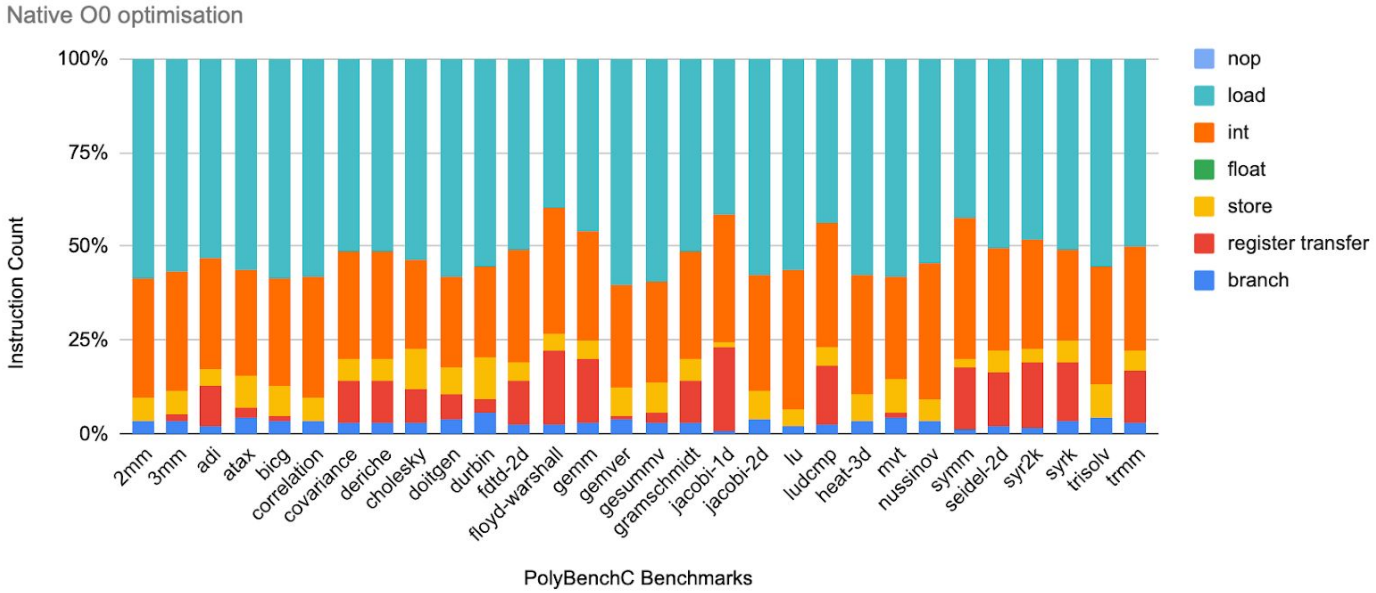


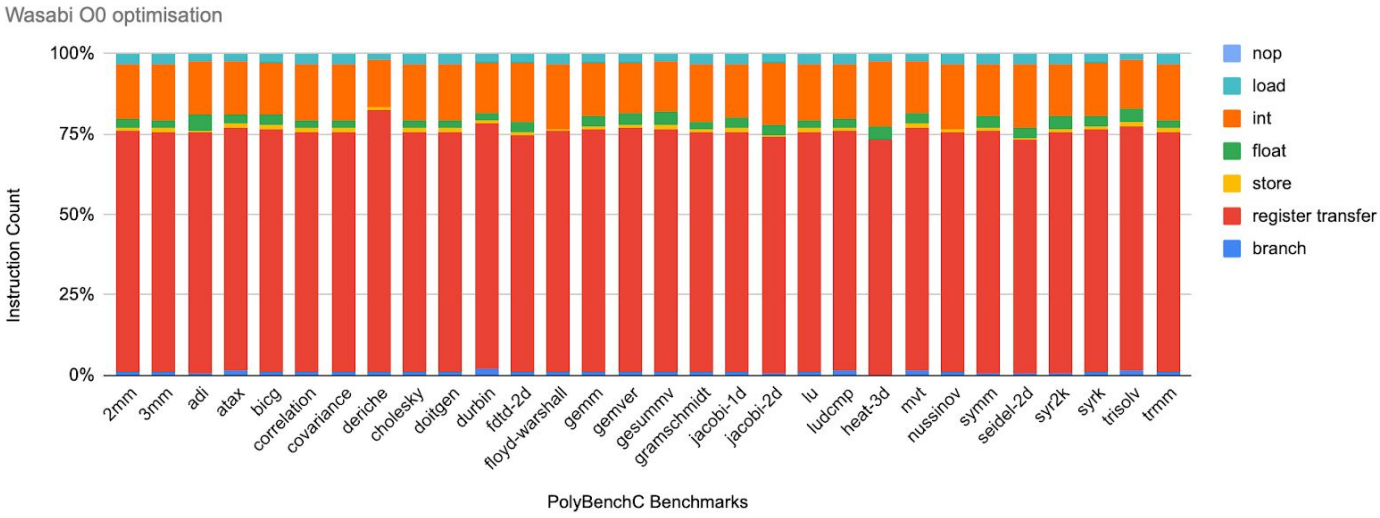


From these graphs, we see that Firefox performs better for unoptimized (-O0 mode). However, Chrome performs consistently better for all 3 stages of optimization.

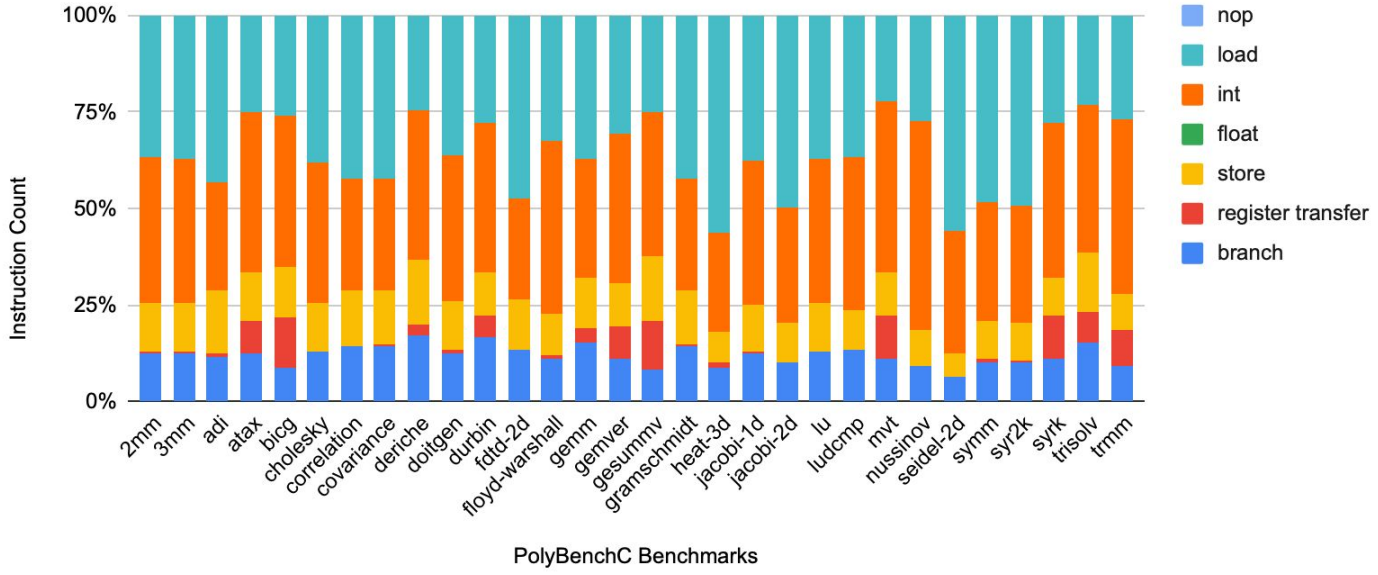
Also, we notice that the unoptimized code is faster through WebAssembly than locally. However, this doesn't hold for optimized code as the gcc/g++ optimization techniques are far better than emcc. Also being a client-side interface further adds to overheads in the optimized stage.

Experiment 2 - Instruction Mix Analysis

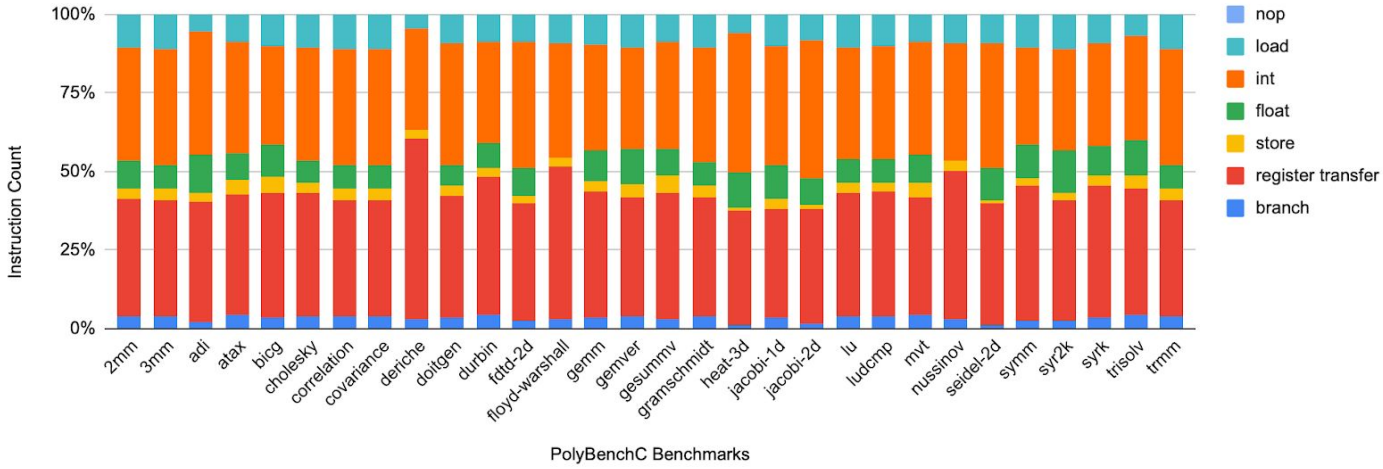




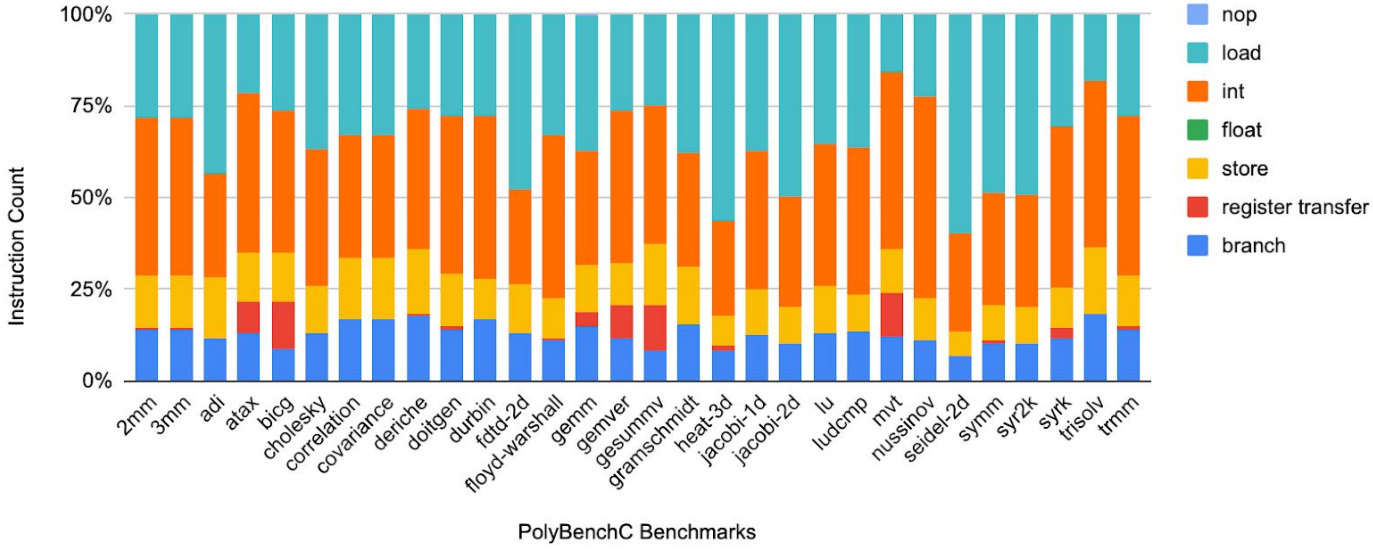
Native O1 Optimisation



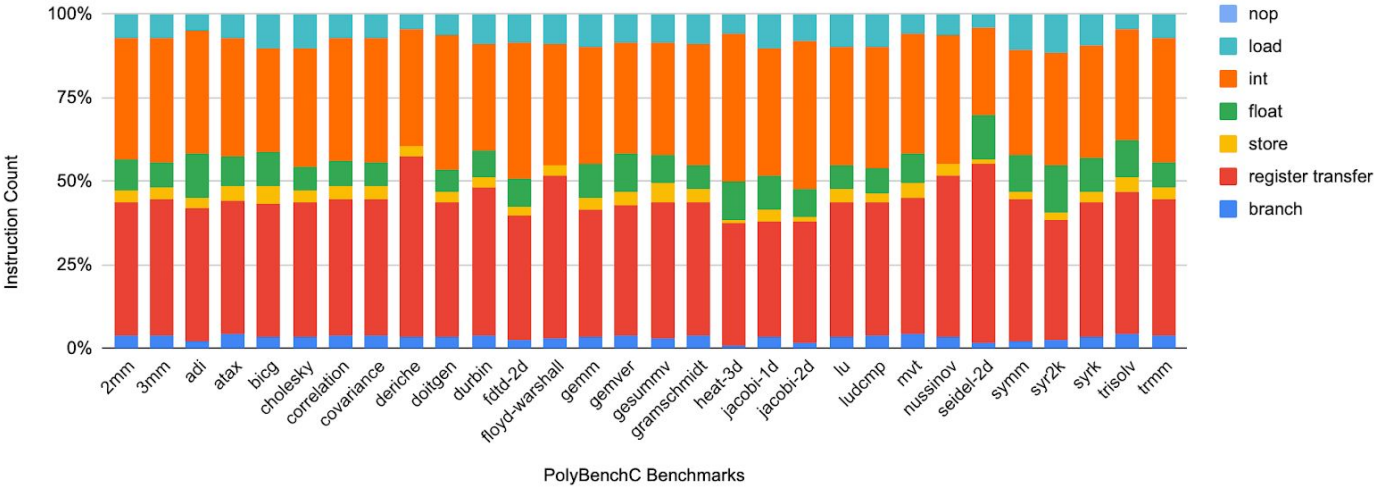
Wasabi O1 optimisation

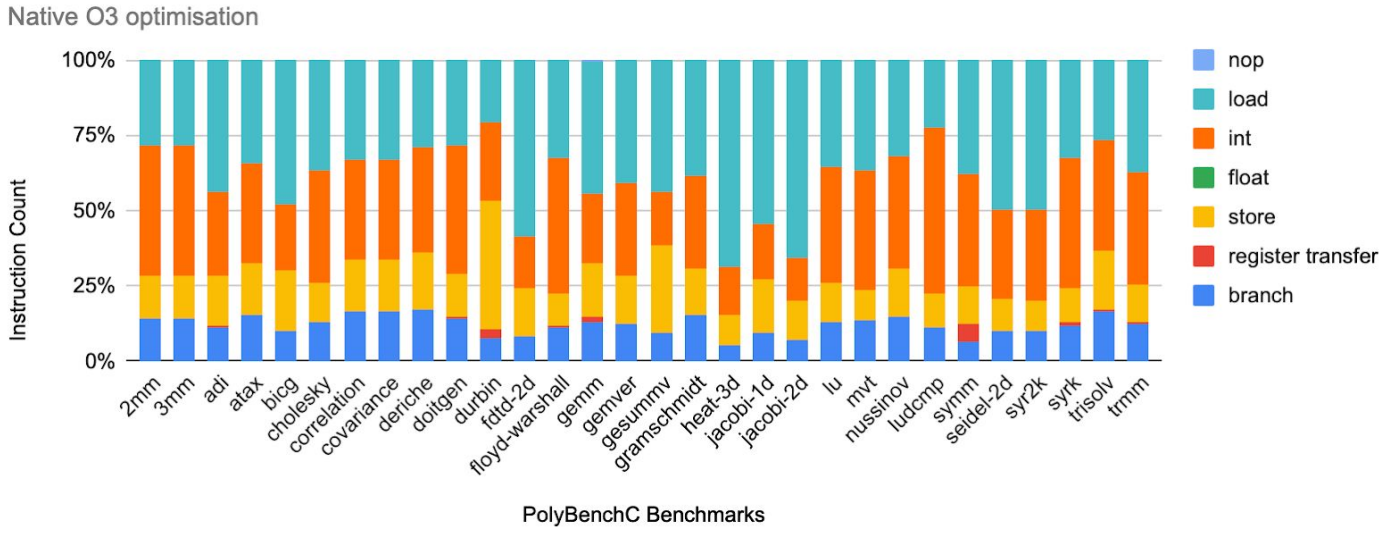


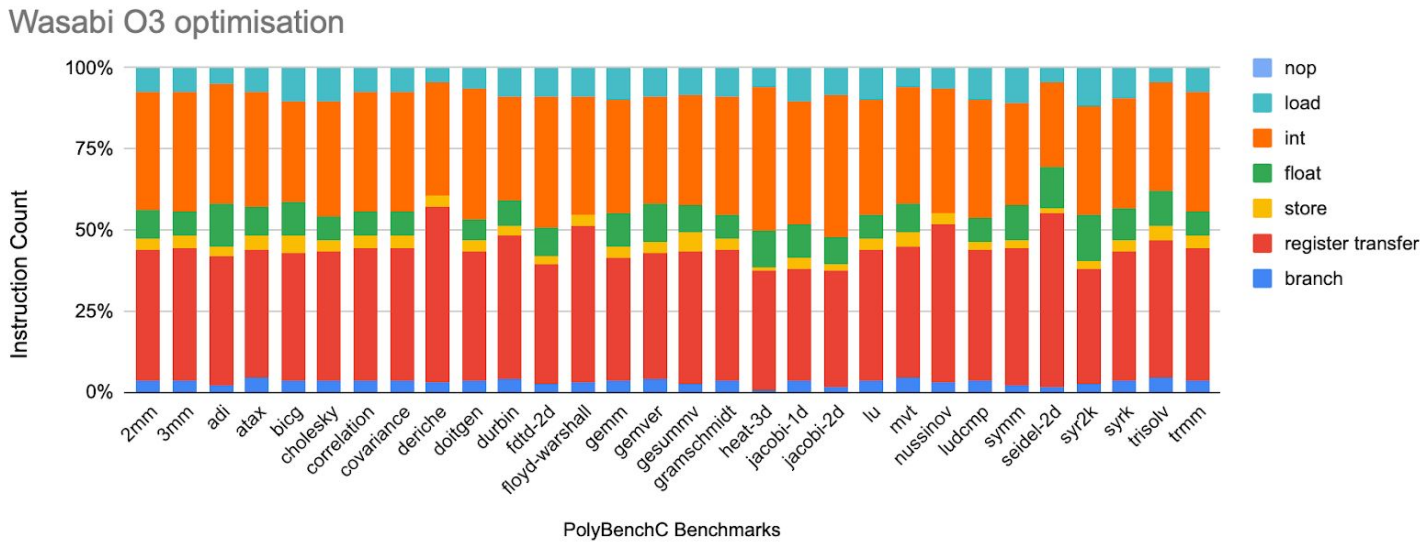
Native O2 Optimisation



Wasabi O2 optimisation

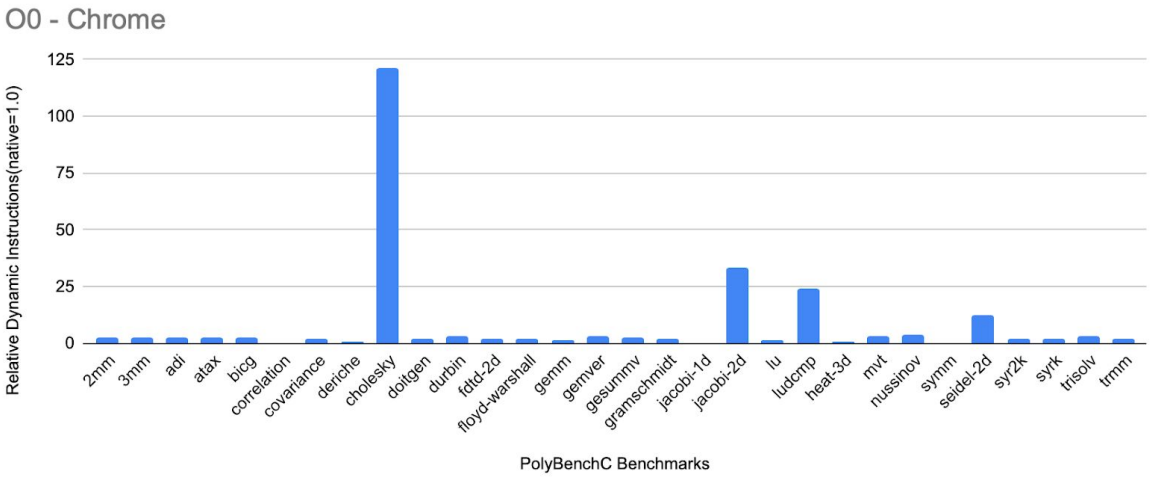


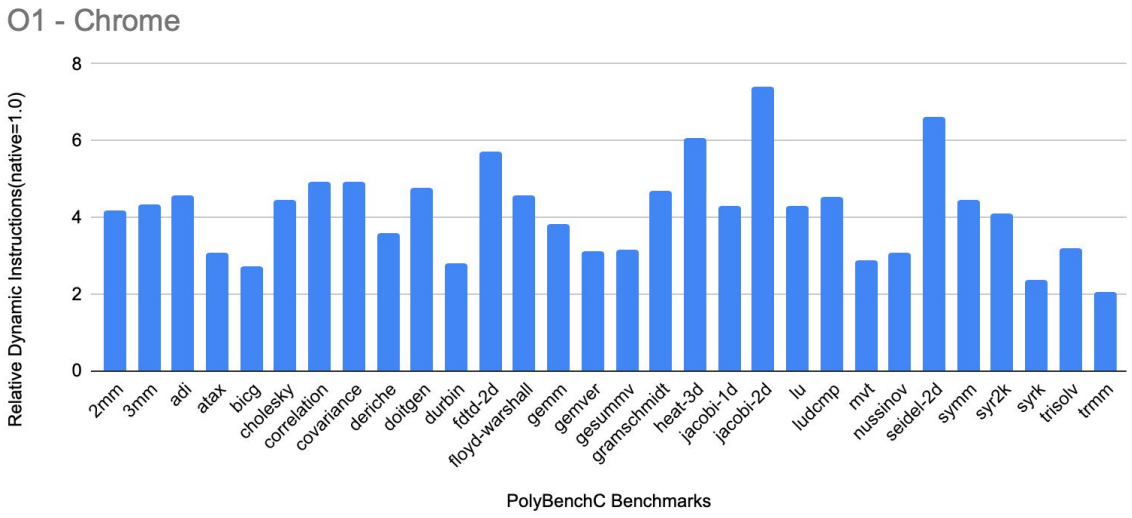


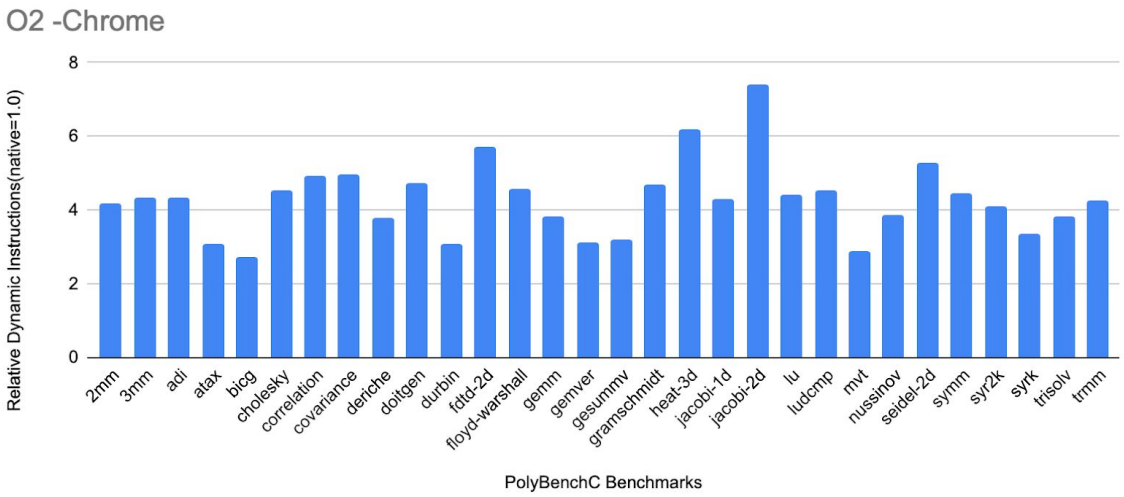


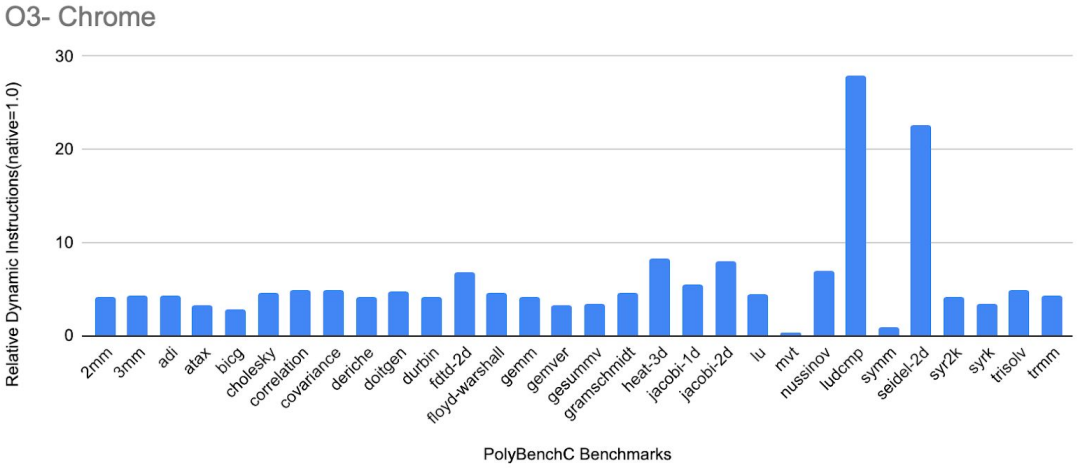
Across all 4 optimization stages, we observe that the native compilers tend to favour load and store instructions whereas the WebAssembly compiler prefers register transfer. Since primarily WA works on the client side, it doesn't have complete access to the storage systems, and hence relies on register transfer for most of its computations. Google Chrome and Firefox, being web browsers, suffer from poor instruction selection, hence resulting in larger code.

Experiment 3 - Instruction Count



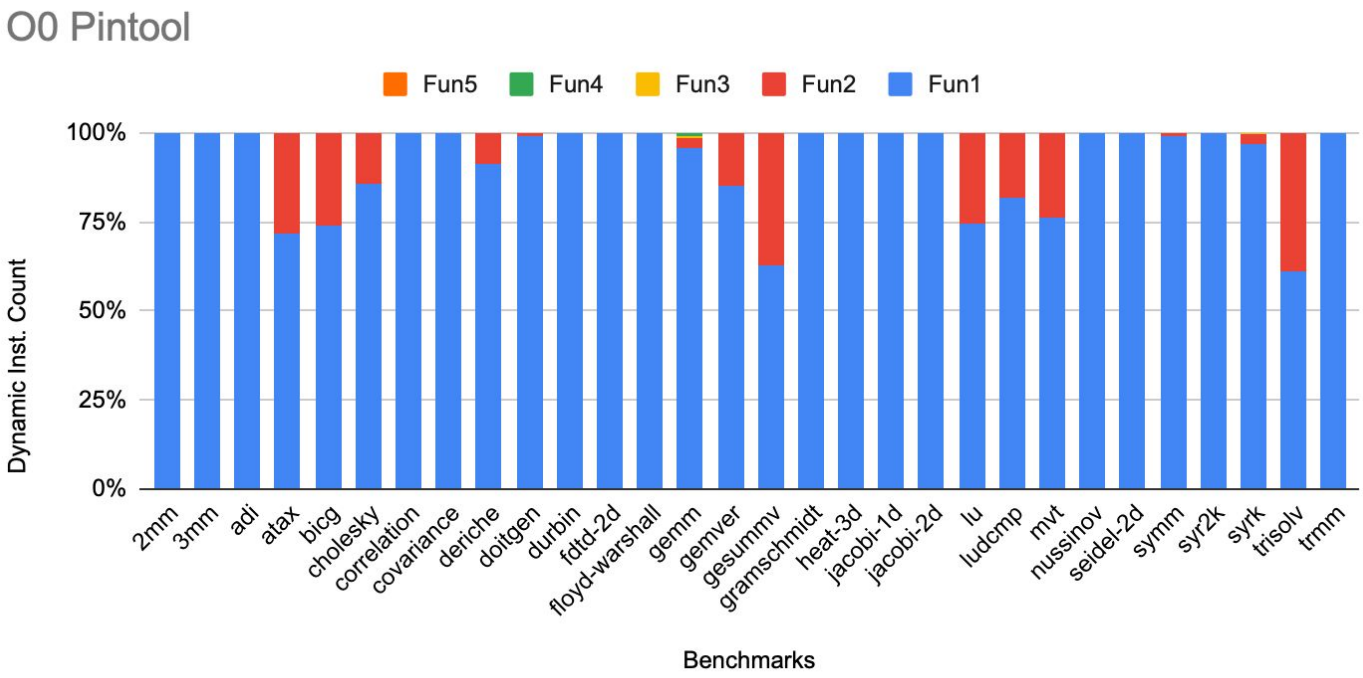


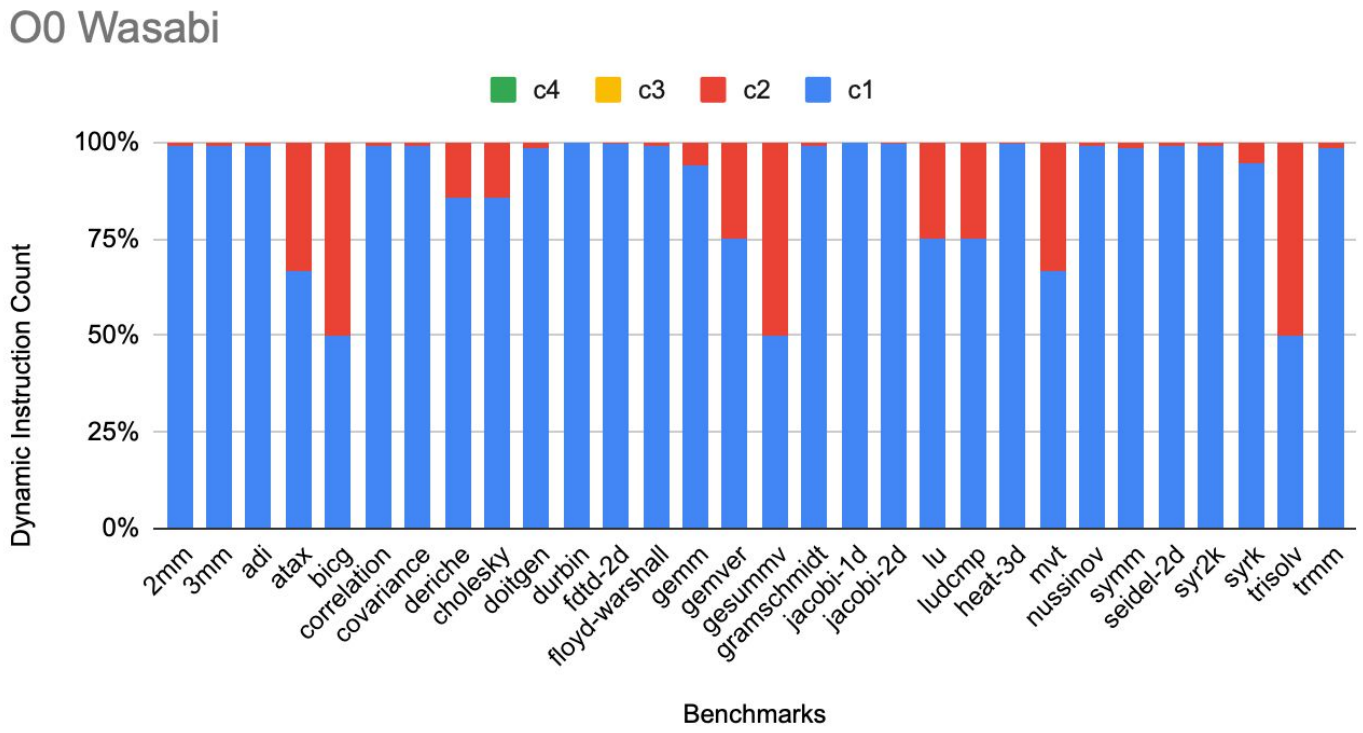




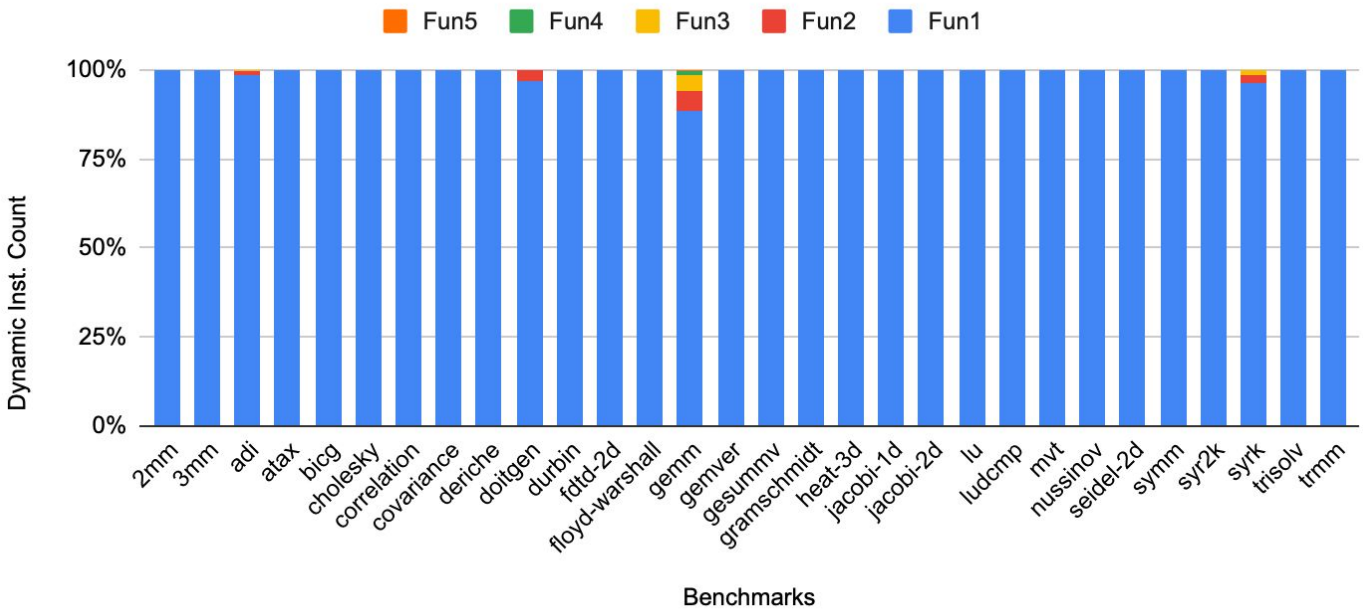
Here, we notice, apart from a few outliers, the instructions are scaled up uniformly in the WebAssembly code as compared to native compilers. Generally by a factor of 5. This is due to the multiple overheads that WA needs to handle due to its virtualization and abstraction from the OS.

Experiment 4 - Hot Code

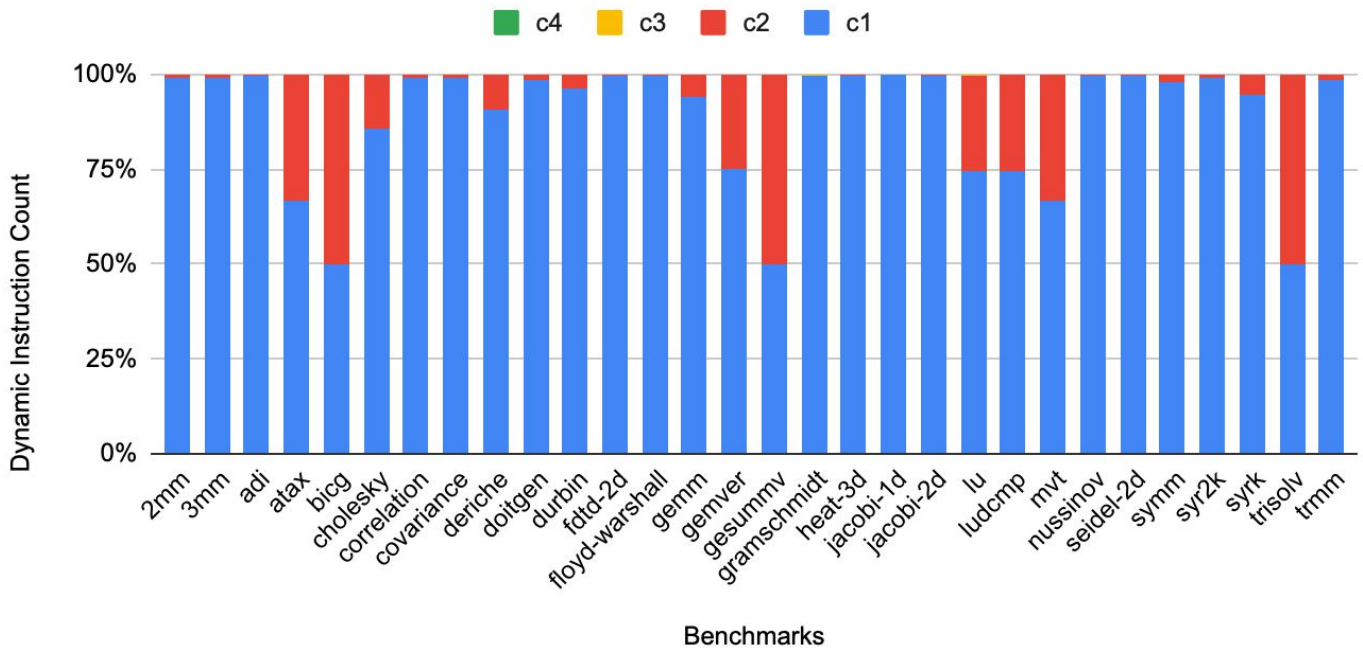




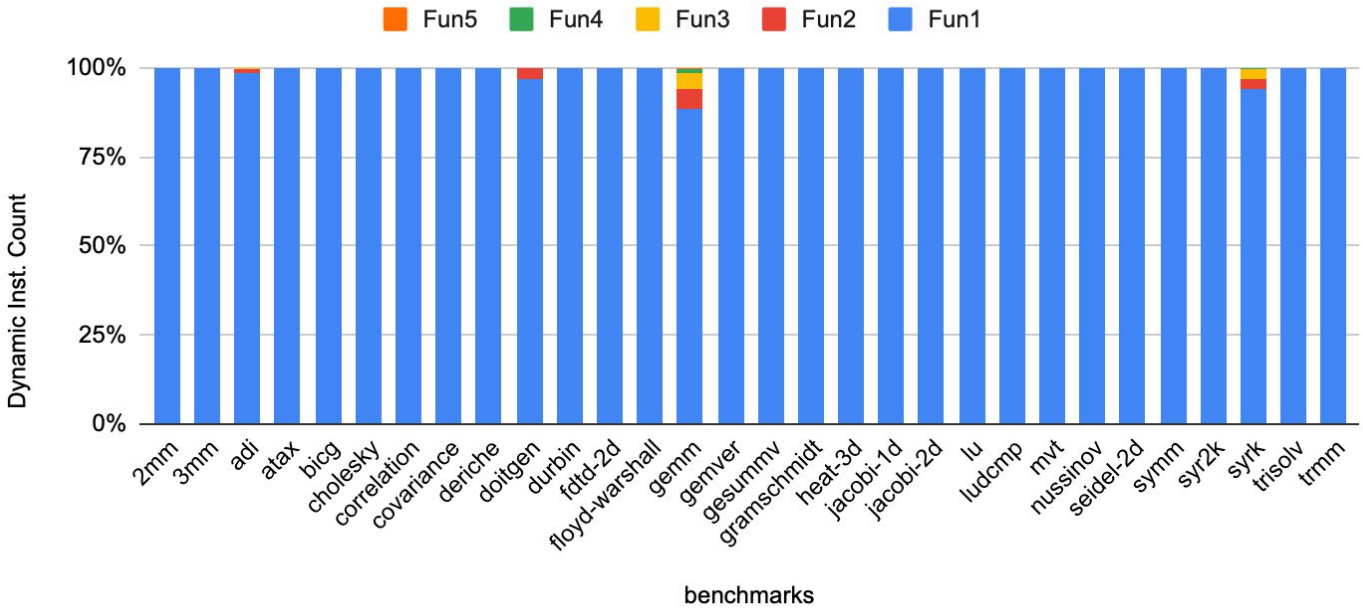
O1 Pintool



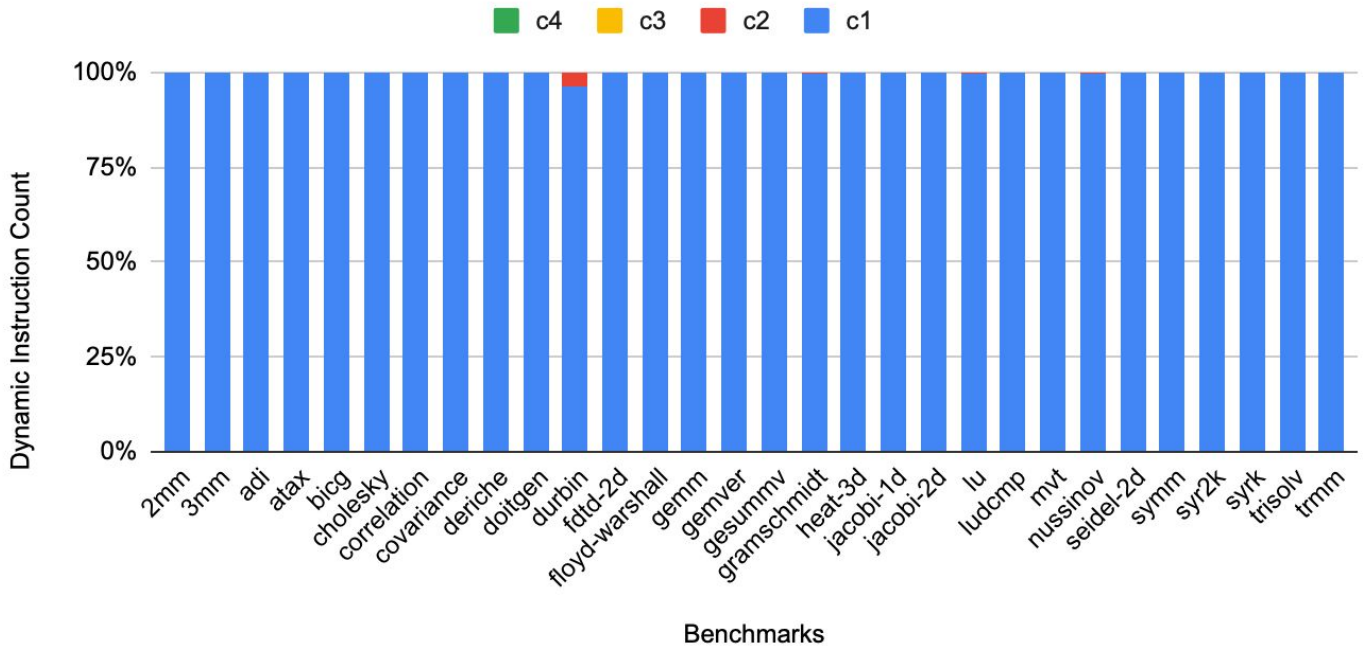
O1 Wasabi



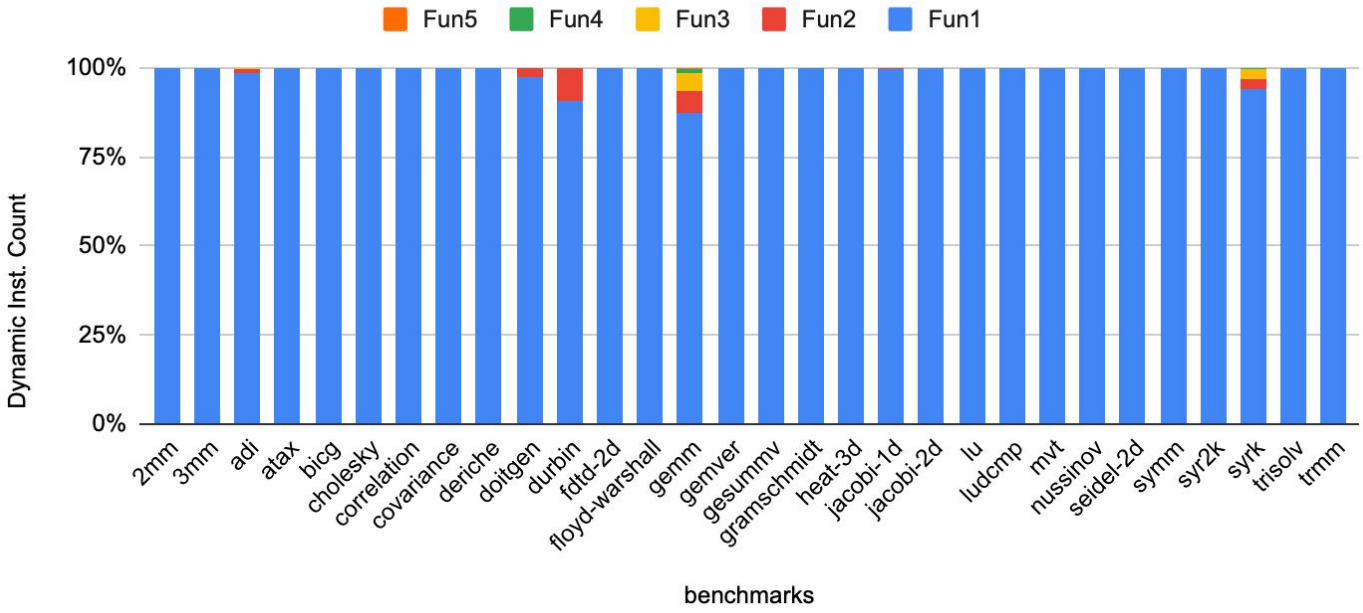
O2 Pintool



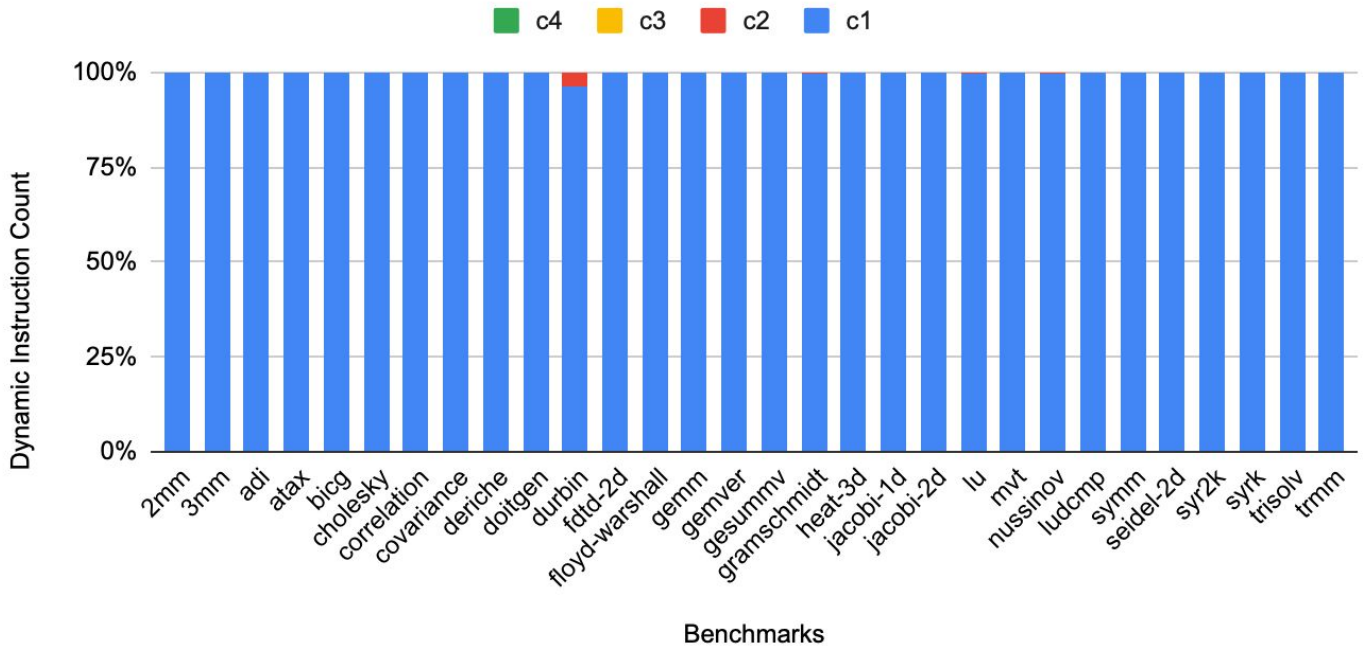
O2 Wasabi



O3 Pintool



O3 Wasabi



Here we notice that for most functions the instructions are concentrated in one function for the native compiler (Generally 90%+ of the instructions mix). However, for the WebAssembly code, there are multiple functions with comparable magnitudes of instructions count that are executed frequently. Again, due to poor code selection in Chrome and Firefox, the dynamic count of function execution increases.

***Note**: All the data used for preparing the graphs is available in this sheet*