CS6291 Project 2: ARM vs. Thumb

Introduction

The purpose of this experiment is to find a method for determining a mixture of functions compiled into ARM and Thumb instructions that maximizes the code size reduction from Thumb instructions and the performance benefits from ARM instructions. To determine which functions should be compiled into Thumb instructions and which should be compiled into ARM instructions, not only should functions that are used the most and have the most time spent in them be considered for ARM compilation, but also the parent functions that call them. By looking at the connect components (CC) in the call graph instead of each function individually, uses, time spent, and parent functions can be evaluated all at once. By compiling into ARM CCs with the most time spent, costly switches between ARM and Thumb should be minimized while the code size remains reduced and the performance increases.

Methodology

To evaluate CCs, the call graphs for the benchmark programs were generated with the CCs being sorted in order of time spent in each of them using gprof2dot and graphviz to visualize the graph.

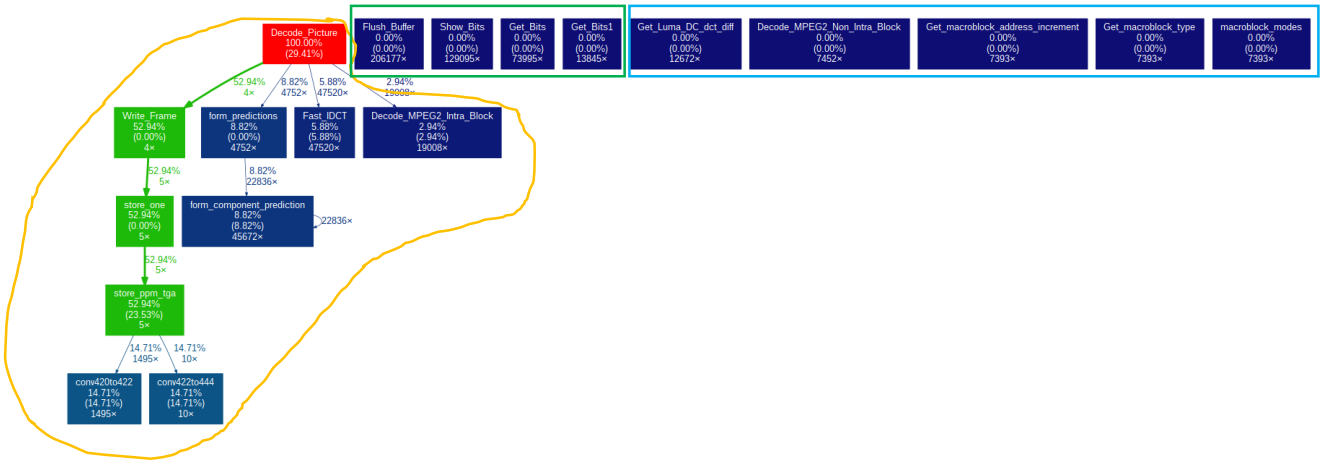


Figure 1: Part of the call graph generated by the mpeg2decode benchmark. The functions enclosed by the orange border represents the first CC. The functions enclosed by the green rectangle are the four that were compiled into ARM instructions for the second set of runs for the CC heuristic. The functions enclosed by the blue rectangle are the ones compiled into ARM instructions for the third set of runs for the CC Heuristic.

All nodes were considered and edges that accounted for less than 0.01% of the runtime were excluded to better divide the functions into CCs. Excluding parent functions that make infrequent calls is considered acceptable since there won't be frequent ARM/Thumb conversions across those edges. A variable amount of CCs were then chosen to be compiled into ARM instructions starting with the CCs with the most time spent in them. The programs were then run 10 times for each amount of CCs and input file. There were two input files used for each benchmark. For cjpeg and djpeg, a 24 MB PPM file and a 463 MB PPM file were used, and for mpeg2encode and mpeg2decode the input_base files included with the benchmark

were used as well as 4 copies of the 463 MB PPM file for the second set of test files. Each newly compiled executable was run once before the ten runs to ensure all runs were on a hot cache.

Results

The results below show the size of the four executables for complete ARM compilation, complete Thumb compilation, and when the functions in 1, 5, and 10 CCs are compiled into ARM instructions while the rest is Thumb. Below that table are the tables showing the average execution times after 10 runs of each executable in all 5 variants for each of the 2 input sets. Note that all time values have an uncertainty of ± 0.005 and the standard deviation within the 10 runs for each test is listed within the parentheses next to the time value.

Executable File Size (B)

Benchmark	Thumb	Arm	CC Count: 1	CC Count: 5	CC Count: 10
cjpeg	74764	90972	75004	75068	75292
djpeg	79408	103744	83728	83744	83840
mpeg2enc	73852	90108	82484	82616	82616
mpeg2dec	68392	80552	72776	72968	77064

Average Execution Time with Standard Deviation: 463 MB PPM File (Seconds)

Benchmark	Thumb	Arm	CC Count: 1	CC Count: 5	CC Count: 10
cjpeg	15.37 (0.14)	15.39 (0.13)	15.65 (0.40)	15.56 (0.21)	15.46 (0.07)
djpeg	9.24 (0.31)	8.52 (0.30)	8.45 (0.43)	8.42 (0.69)	8.66 (0.37)

Average Execution Time with Standard Deviation: 24 MB PPM File (Seconds)

Benchmark	Thumb	Arm	CC Count: 1	CC Count: 5	CC Count: 10
cjpeg	0.73 (0.03)	0.74 (0.03)	0.74 (0.03)	0.74 (0.03)	0.72 (0.03)
djpeg	0.42 (0.03)	0.40 (0.04)	0.43 (0.03)	0.42 (0.05)	0.42 (0.03)

Average Execution Time with Standard Deviation: Input_Base Set (Seconds)

Benchmark	Thumb	Arm	CC Count: 1	CC Count: 5	CC Count: 10
mpeg2encode	8.32 (0.09)	7.55 (0.11)	7.51 (0.10)	7.54 (0.13)	7.51 (0.13)
mpeg2decode	0.57 (0.03)	0.58 (0.04)	0.57 (0.03)	0.57 (0.03)	0.58 (0.02)

Average Execution Time with Standard Deviation: 463MB File Set (Seconds)

Benchmark	Thumb	Arm	CC Count: 1	CC Count: 5	CC Count: 10
mpeg2encode	1.74 (0.07)	1.61 (0.05)	1.64 (0.04)	1.66 (0.07)	1.64 (0.04)
mpeg2decode	0.32 (0.02)	0.32 (0.02)	0.30 (0.02)	0.33 (0.02)	0.32 (0.02)

Conclusion

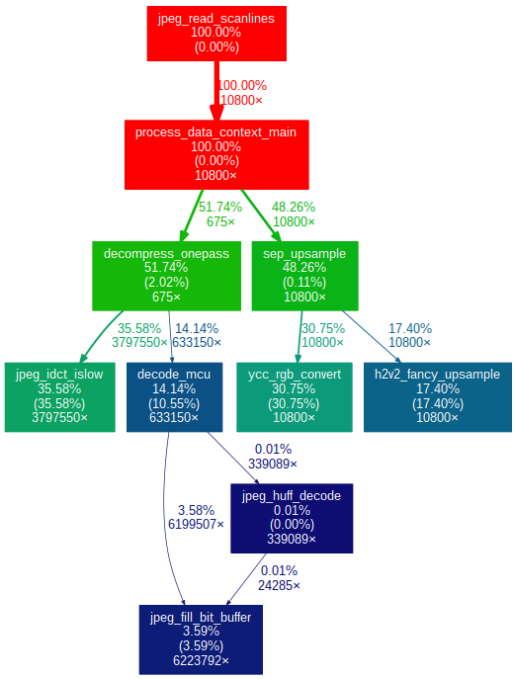
For the following tests, the average execution times with their standard deviations for each compilation mode all overlap showing no significant performance gains:

- cjpeg (463MB)
- cjpeg (24MB)
- djpeg (24MB)
- mpeg2decode (Input_Base Set)
- mpeg2decode (463MB Set)
- mpeg2encode (463MB Set)

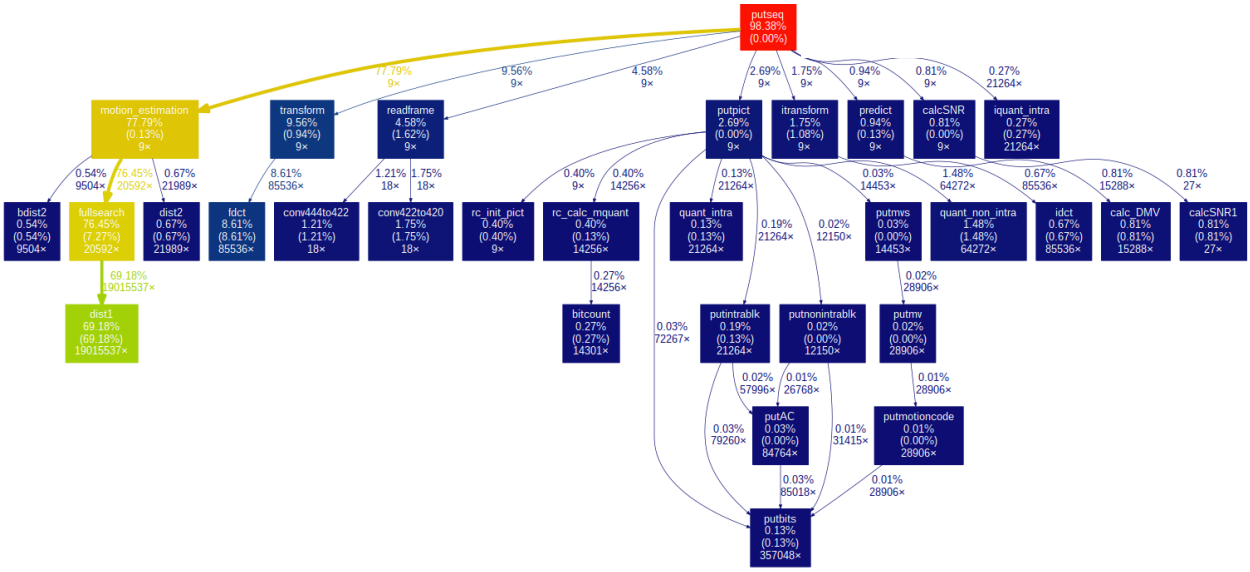
Therefore, the smallest executable, the complete Thumb compilation executable is viewed as having the maximum benefit when considering the execution times and the file size. For remaining two tests (djpeg 463MB and mpeg2encode Input_Base set), the complete Thumb compilations performed significantly worse (standard deviations did not overlap with other compilation modes), while the other did. Therefore, compiling one CC into ARM instructions yield the best performance plus size. The two tests that did see a significant performance difference were both on the larger of the two input files and sets that they were evaluated for. It stands to reason that there may be significant performance gains among the other tests if the input files and sets were larger. However, for the cjpeg and djpeg programs, working with PPM files larger than 463MB would not be a common case. The 463MB file was originally generated by converting a 10500x10500 image to PPM. In conclusion, the size difference outweighs the performance differences in most use-cases. However, when taken to extremes, using the connected components heuristic appears to keep with the same performance improvements of total ARM compilation while still reducing the overall size.

Appendix: Submitted Compilations

cjpeg is compiled completely into Thumb instructions
djpeg is in the folder djpeg-prj2 and has had the following functions compiled into ARM instructions



mpeg2encode has had the following functions compiled into ARM instructions



mpeg2decode has been compiled completely into Thumb instructions.