Project 2 : Common SubExpression Elimination

Question 1 - Implementation

There are 4 different optimization techniques which are implemented as a part of this project.

Dead Code Elimination:

The optimization removes instructions which have definitions but there aren't any uses associated with it inside a basic block. The instructions inside a function of a module is being traversed and if there are any uses for the value computed in the instruction, it is being removed from the program. It removes the instructions which have no uses only if it is one of the below instruction types.
Add, FNeg, FAdd, Sub, FSub, Mul, FMul, UDiv, SDiv, FDiv, URem, SRem, FRem, Shl, LShr, AShr, And, Or, Xor, GetElementPtr, Trunc, ZExt, SExt, FPToUI, FPToSI, UIToFP, SIToFP, FPTrunc, FPExt, PtrToInt, IntToPtr, BitCast, AddrSpaceCast, ICmp, FCmp, PHI,Select, ExtractElement, InsertElement, ShuffleVector,ExtractValue, InsertValue.

CSE Simplify and Eliminate:

This optimization focusses to simplifying constant expressions. The instructions inside a function of a module is being traversed and if they can be simplified, the instructions which use the result of this simplified instruction are being modified to use the value of the simplification.

In the elimination step, the instruction is compared with the child instructions in the same basic block and in the basic blocks dominated by the parent instructions for common sub expressions. If they are common, it replaces these with the parent instruction's value. There are some instructions which shouldn't be eliminated in this step which are described below.
Load, Store, Call, PHI, Alloca. Ret, Br, FCmp, ICmp, ExtractValue.

Redundant Load Eliminate:

This optimization passes, iterates through the instructions, for each function in a module and removes the load instructions which are redundant inside a basic block. Each child Load instruction inside a basic block is being compared to be non volatile, have the same type and have the same address. Then the subsequent child load instruction can be eliminated and use the value of the parent load in the instructions which use those. This optimization is not performed when there is a store or call instruction in between two loads.

Redundant Store Eliminate:

This optimization step performs store to load forwarding and removes redundant store instructions. It iterates through the instructions, for each function in a module and redundant instructions inside a basic block. If there Store followed by a non volatile load instruction pointing to the same address and having the same type, then all uses of the child load instruction is replaced with the value being stored and the child load is removed.

If there are two consecutive store instructions having the same type and pointing to the same address, then the old store instruction can be removed. But if there is a load or a store or a call in between two subsequent stores or between a load and store, then the optimization is not performed.

Order of optimization:

The dead code elimination pass is performed first to remove all the useless instructions inside a basic block. Now the Redundant Load Elimination pass is performed followed by Store to load forwarding and redundant store elimination. This pass removes all the unnecessary loads and stores. Now that loads and stores are removed and instructions which use them are replaced with values from those, we can perform CSE Simplify to simplify the expressions followed by CSE Eliminate which eliminates Common Subexpressions.

Note:

The call instructions have not been considered in redundant stores, store to load forwarding and redundant load optimization step to pass the statistics result in gradescope. The reason being that whenever there is a call instruction between subsequent loads or subsequent stores then optimization 2 and 3 are not being performed. To pass the statistics test case in gradescope, the call function in line number 499 and 572 as a part of the else if statement has been removed.

Question 2 - Run Data

Instruction count

Category	**CSE only**	**M2RCSE**
adpcm	406	241
arm	707	381
basicmath	556	326
bh	3069	1882
bitcount	616	431
crc32	141	83
dijkstra	319	226
em3d	1151	614
fft	704	394
hanoi	90	51

hello	4	2
kmp	518	342
l2lat	88	57
patricia	1031	658
qsort	140	92
sha	603	377
smatrix	283	209
sql	167725	104858
susan	11771	6538

Execution time

Category	**CSE only**	**M2RCSE**
adpcm	2.17	2.05
arm	0.0	0.0
basicmath	0.08	0.04
bh	0.91	1.09
bitcount	0.16	0.23
crc32	0.23	0.1
dijkstra	0.05	0.04
em3d	0.42	0.41
fft	0.04	0.1
hanoi	1.94	1.73
kmp	0.13	0.17
l2lat	0.03	0.04
patricia	0.16	0.07
qsort	0.02	0.1
sha	0.04	0.04

smatrix	2.86	3.34
sql	0.0	0.0
susan	0.66	1.19

Total Number of Loads

Category	**CSE only**	**M2RCSE**
adpcm	111	15
arm	210	49
basicmath	149	24
bh	764	196
bitcount	137	51
crc32	33	8
dijkstra	92	47
em3d	353	107
fft	201	38
hanoi	25	6
hello	-	-
kmp	148	57
l2lat	17	5
patricia	346	137
qsort	35	13
sha	173	42
smatrix	72	33
sql	52320	16368
susan	3972	1029

Total Number of Stores

Category	CSE only	M2RCSE
adpcm	81	7
arm	116	18
basicmath	100	12
bh	494	142
bitcount	92	18
crc32	29	4
dijkstra	51	24
em3d	192	43
fft	102	24
hanoi	16	4
hello	1	-
kmp	71	20
l2lat	14	1
patricia	108	30
qsort	16	4
sha	98	28
smatrix	31	10
sql	21872	5840
susan	1429	156

Optimization Data

Dead Code Elimination

Category	CSE only	M2RCSE
adpcm	1	2
arm	-	-
basicmath	1	1
bh	-	-
bitcount	1	1
crc32	-	-
dijkstra	-	-
em3d	-	3
fft	-	-
hanoi	-	-
hello	-	-
kmp	-	-
l2lat	-	-
patricia	-	-
qsort	1	1
sha	-	-
smatrix	-	-
sql	183	188
susan	-	-

CSE Simplify

Category	CSE only	M2RCSE
adpcm	1	2

arm	19	21
basicmath	6	6
bh	1	1
bitcount	1	2
crc32	-	-
dijkstra	-	-
em3d	13	14
fft	-	7
hanoi	1	1
hello	-	-
kmp	2	2
l2lat	-	-
patricia	3	7
qsort	-	-
sha	2	2
smatrix	-	-
sql	667	857
susan	2	16

CSE Elimination

Category	**CSE only**	**M2RCSE**
adpcm	-	4
arm	21	29
basicmath	10	16
bh	103	174
bitcount	3	10

crc32	-	-
dijkstra	-	7
em3d	3	56
fft	20	51
hanoi	1	1
hello	-	-
kmp	14	38
l2lat	-	1
patricia	13	76
qsort	4	10
sha	17	46
smatrix	7	20
sql	1396	7104
susan	247	1219

Redundant Load Elimination

Category	**CSE only**	**M2RCSE**
adpcm	1	-
arm	30	-
basicmath	13	-
bh	75	2
bitcount	20	-
crc32	3	-
dijkstra	3	2
em3d	21	10
fft	10	-

hanoi	4	-
hello	-	-
kmp	20	-
l2lat	6	3
patricia	24	-
qsort	3	-
sha	32	-
smatrix	24	5
sql	4165	125
susan	384	24

Store to Load forwarding

Category	**CSE only**	**M2RCSE**
adpcm	10	-
arm	7	-
basicmath	5	1
bh	53	-
bitcount	18	-
crc32	1	-
dijkstra	-	-
em3d	45	-
fft	5	-
hanoi	-	-
hello	-	-
kmp	5	1
l2lat	2	-

patricia	8	-
qsort	-	-
sha	6	-
smatrix	1	1
sql	2549	91
susan	217	2

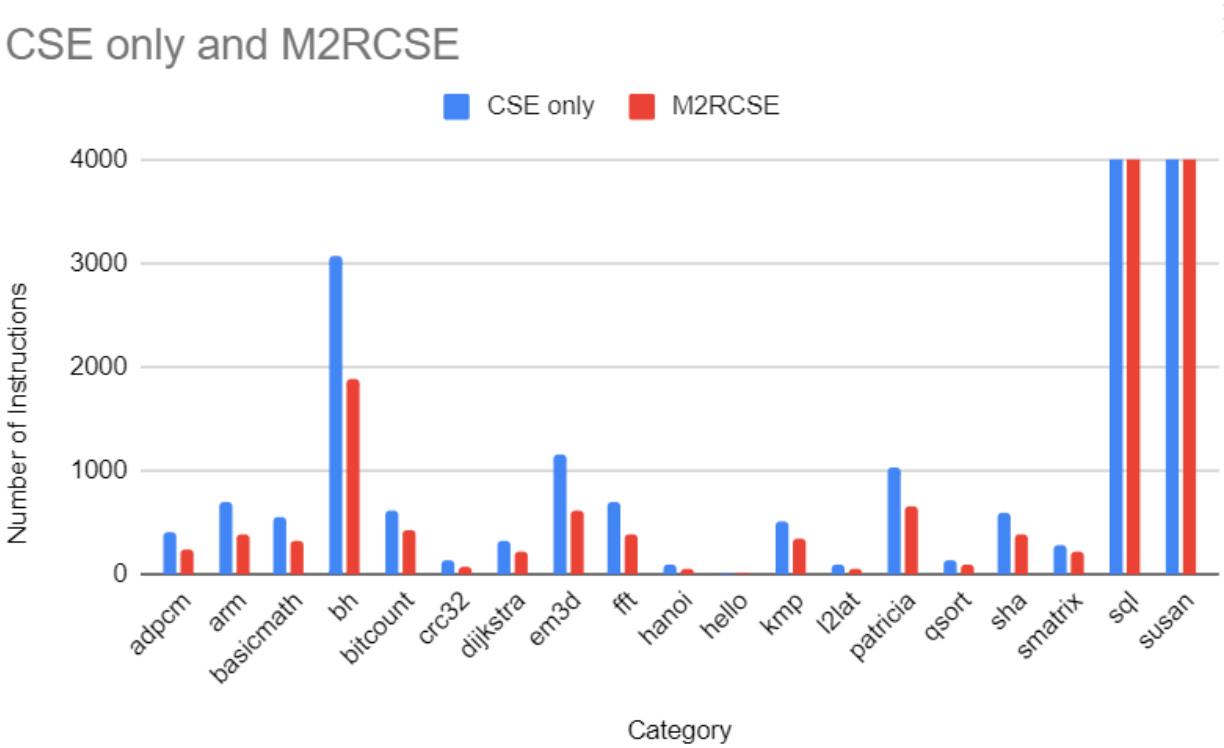
Redundant Store Eliminate

Category	**CSE only**	**M2RCSE**
adpcm	-	-
arm	-	-
basicmath	-	-
bh	-	-
bitcount	6	-
crc32	-	-
dijkstra	-	-
em3d	-	-
fft	-	-
hanoi	-	-
hello	-	-
kmp	-	-
l2lat	1	-
patricia	-	-
qsort	-	-
sha	1	-
smatrix	-	-

sql	26	3
susan	9	1

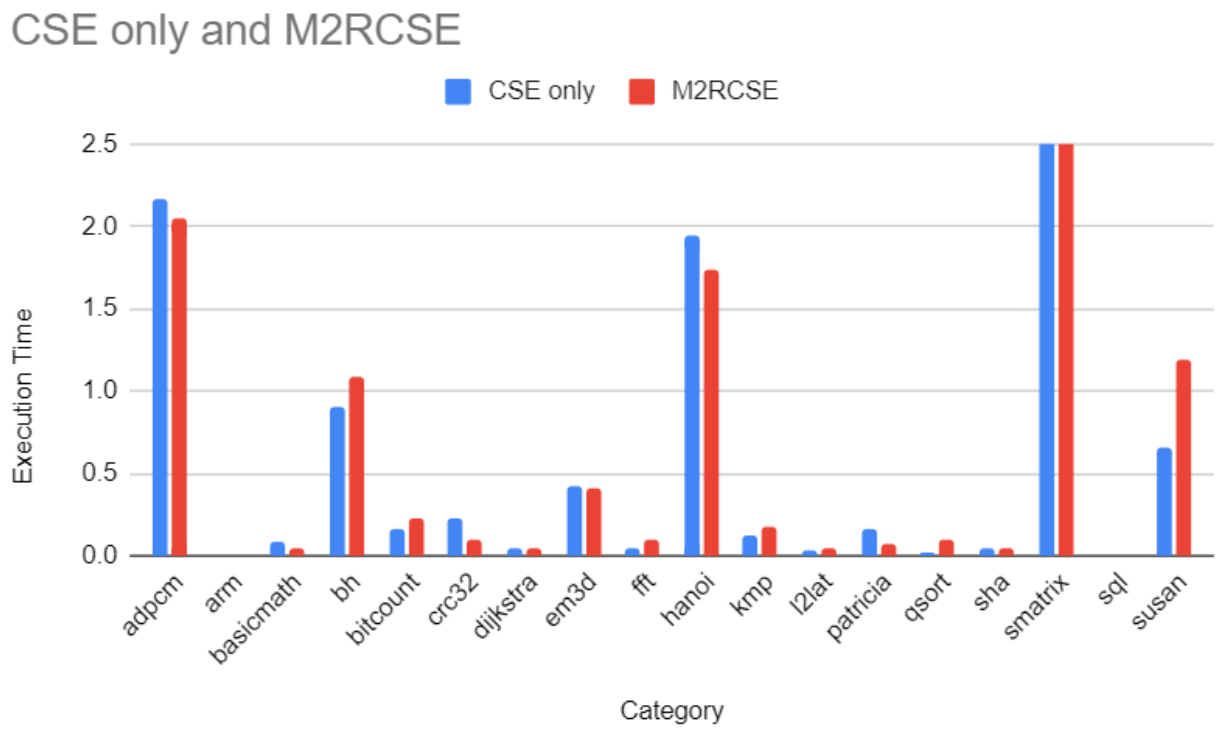
Question 3 - Performance comparison between 2 configurations

Plot for number of final instructions after optimization for various benchmarks



The above plot compares the number of instructions in the program after performing the optimization for each category of benchmarks. We can see that for all the cases the final instruction count for mem2reg followed by CSE gives lesser value compared to final instruction count achieved only using CSE.
The reason being that, mem2reg optimizations bring most of the memory instructions into simple register operations due to which the CSE optimization pass finds out a broader set of instructions which can be optimized. If we don't perform this mem2reg first then there will be a lot of unnecessary loads and stores which cannot be identified by CSE since they won't be directly available.

Plot of Execution time after optimization for various benchmarks



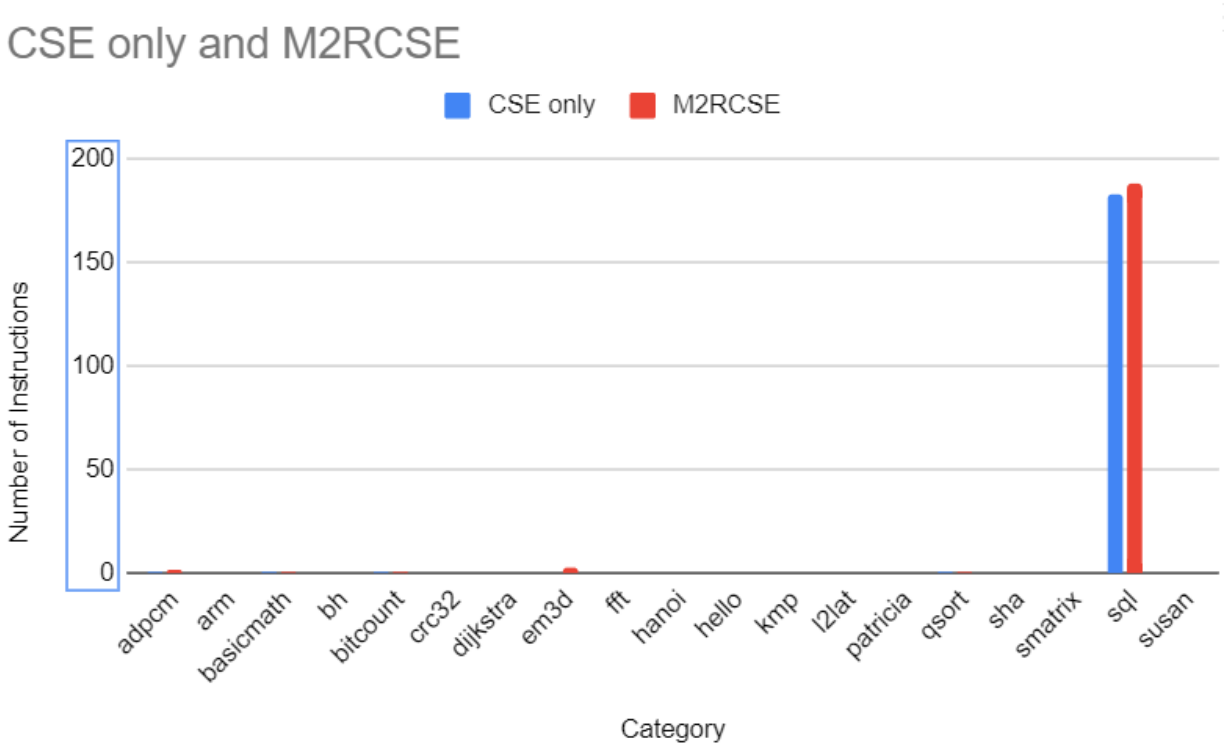
The above plot compares the execution time of various benchmarks when performing just the CSE pass and mem2reg along with CSE pass. We can note that for most of the instructions the execution time is reduced for mem2reg along with CSE pass compared to just the CSE pass. The reason being that, mem2reg pass initially reduces unnecessary memory operations thereby providing a more obvious representation for the CSE to perform optimization. Without the mem2reg pass the CSE won't be able to perform optimizations which are not explicitly available in the program thereby increasing the execution time of the program. Thus mem2reg along with CSE reduces more instructions and thereby has lesser program execution time compared to just the CSE.

Impact on loads and stores

We can also see that mem2reg optimization along with CSE further reduces the memory instructions compared to with just CSE.

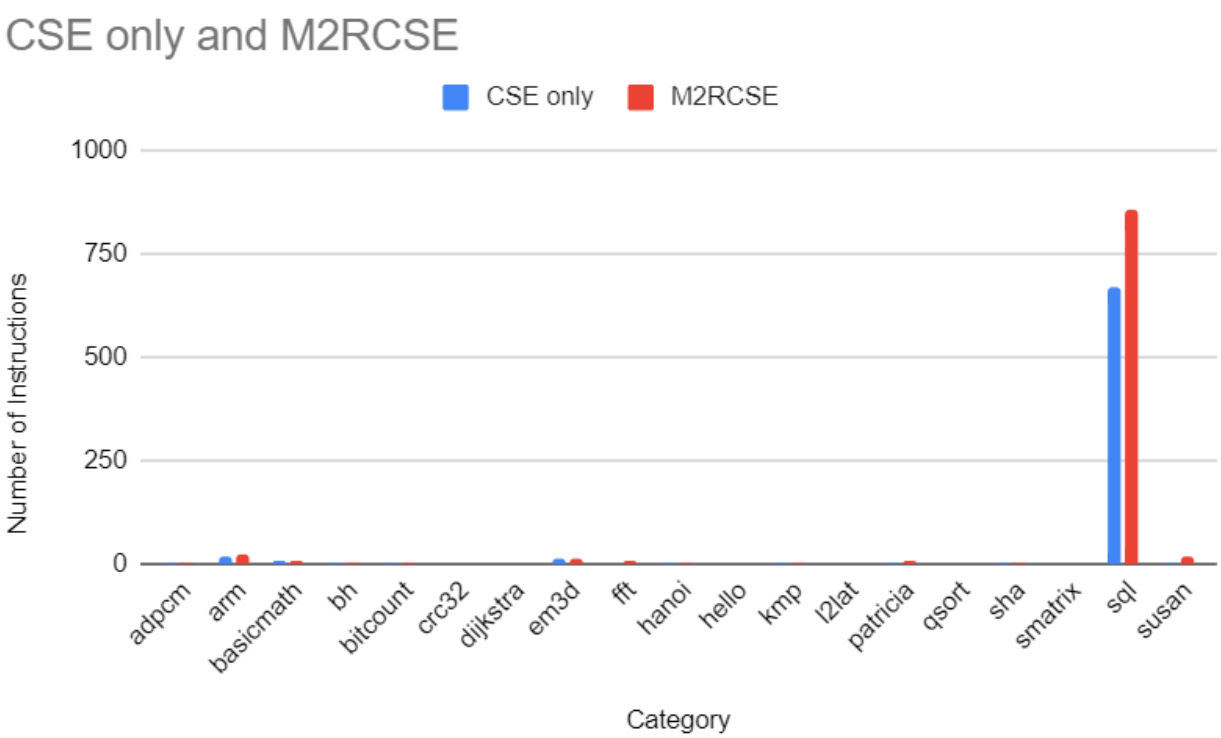
Question 4 - Performance comparison between 2 configurations for various optimizations.

Optimization 1 - Dead Code Elimination



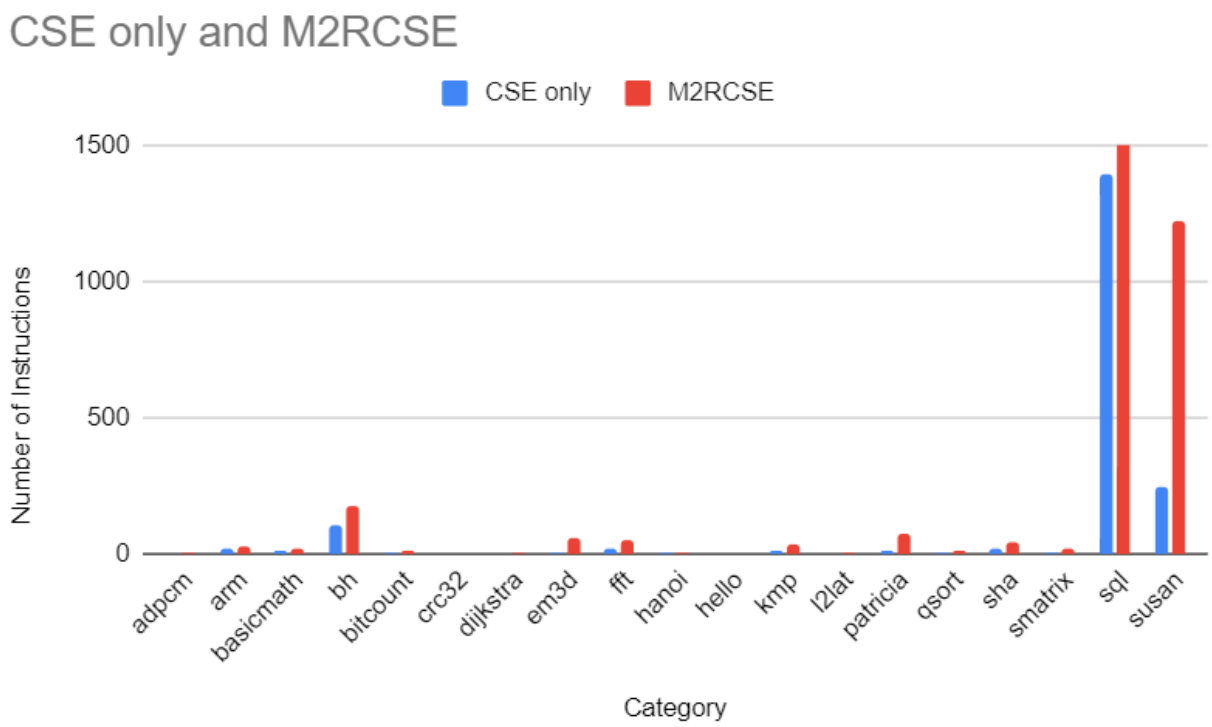
In this plot we can find out that the basic dead code elimination performs optimization in some benchmarks indicating that those benchmarks have explicit dead code inside a basic block. To get more performance, we can add aggressive dead code elimination which checks outside the basic block of instruction as well.
Since Mem2Reg optimization provided a register centric program, this optimization can easily identify dead codes compared to just applying dead code optimization.

Optimization 1.1 - CSE Simplify



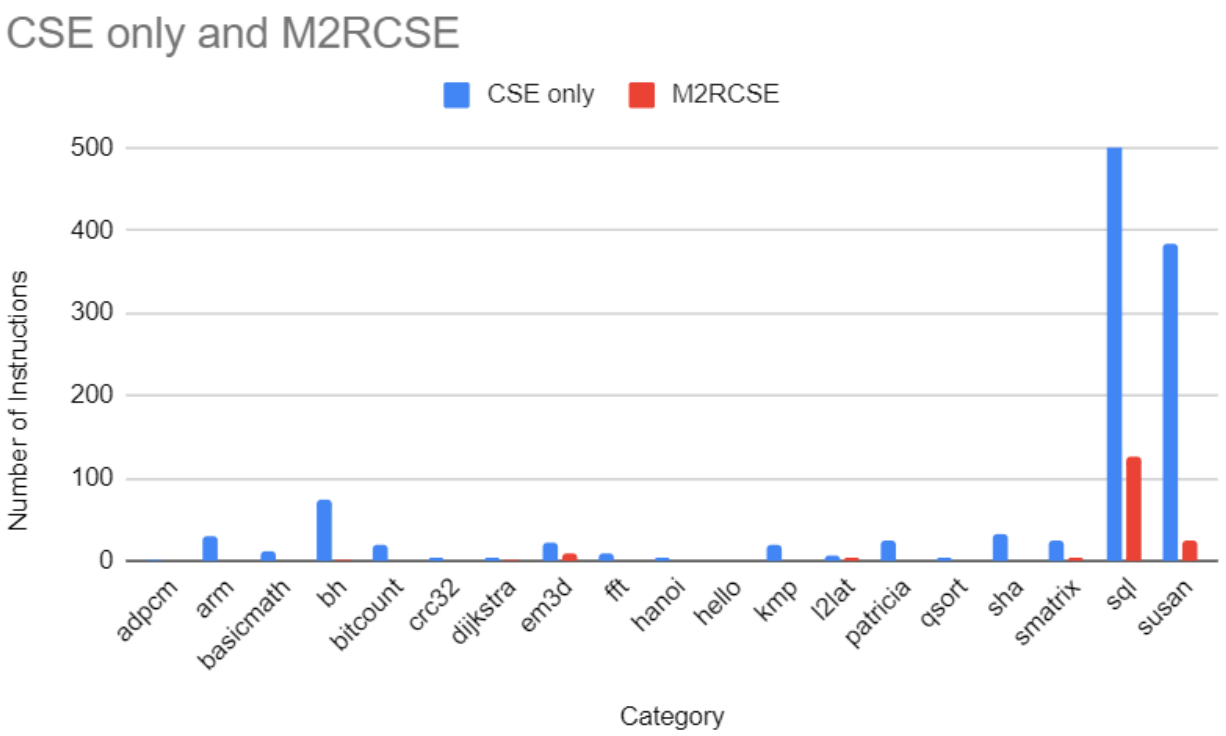
Since Mem2Reg simplifies the program to mostly register operations, the CSE Simplify can identify explicit expressions and apply the optimization.

Optimization 1.2 - CSE Elimination



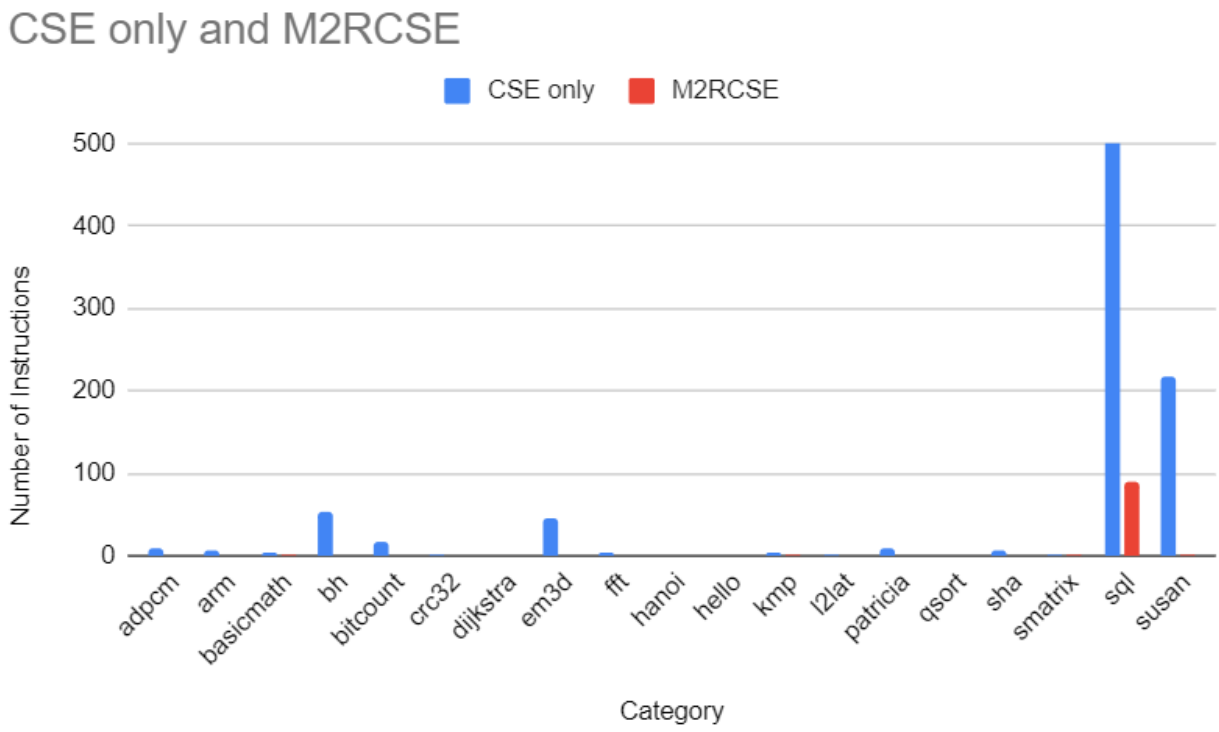
The mem2reg pass along with CSE produces better results as there are more possible instructions which can be optimized by CSE which are explicitly visible.

Optimization 2 - Redundant Load Elimination



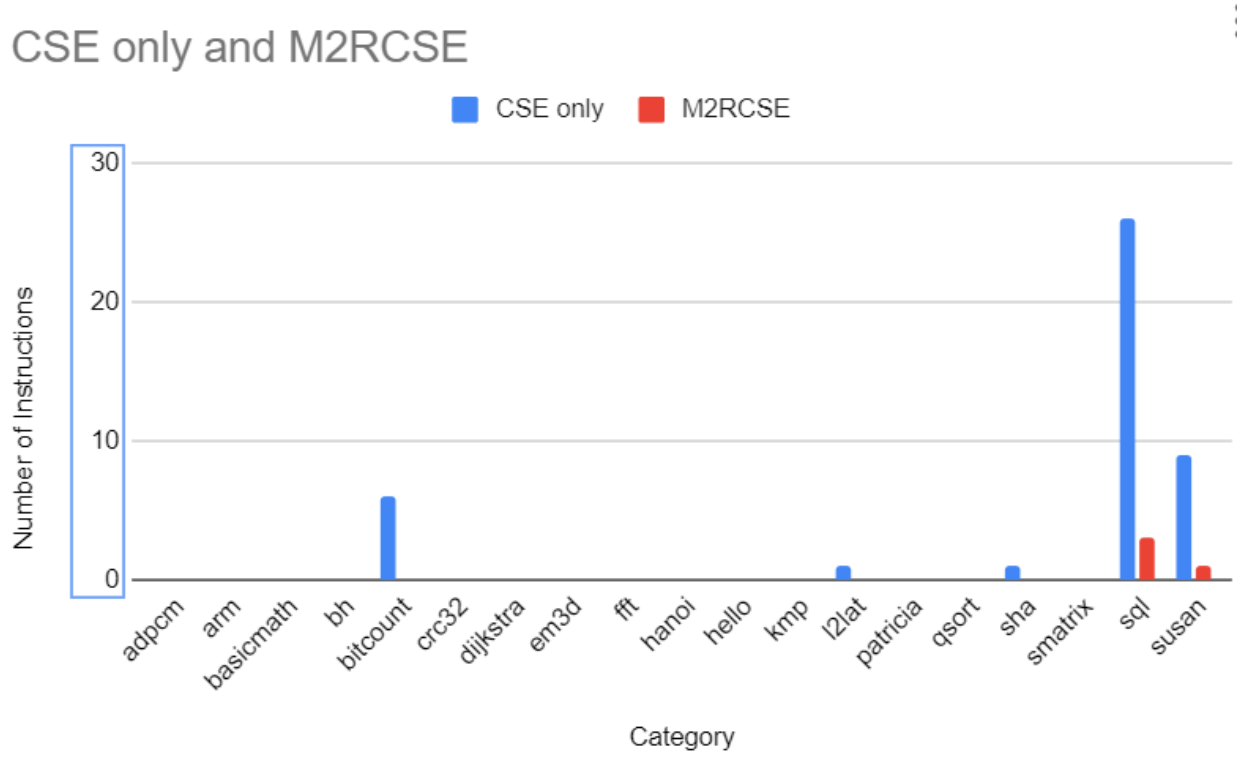
Here we can see that not much of work has been done by redundant load elimination as most of the work is done by Mem2Reg which converts memory operations to register operations.So we can see the results only if CSE is applied without Mem2Reg.

Optimization 3.1 - Store to Load forwarding



Here we can see that not much of work has been done by store to load forwarding as most of the work is done by Mem2Reg which converts memory operations to register operations.So we can see the results only if CSE is applied without Mem2Reg.

Optimization 3.2 - Redundant Store Elimination



Here we can see that not much of work has been done by redundant store elimination as most of the work is done by Mem2Reg which converts memory operations to register operations.So we can see the results only if CSE is applied without Mem2Reg.