Partners

Organizer

Facilitator

Solution Report: DIR-V Systems & Software Hackathon

Arnav Pahuja

B.Tech. Electrical Engineering

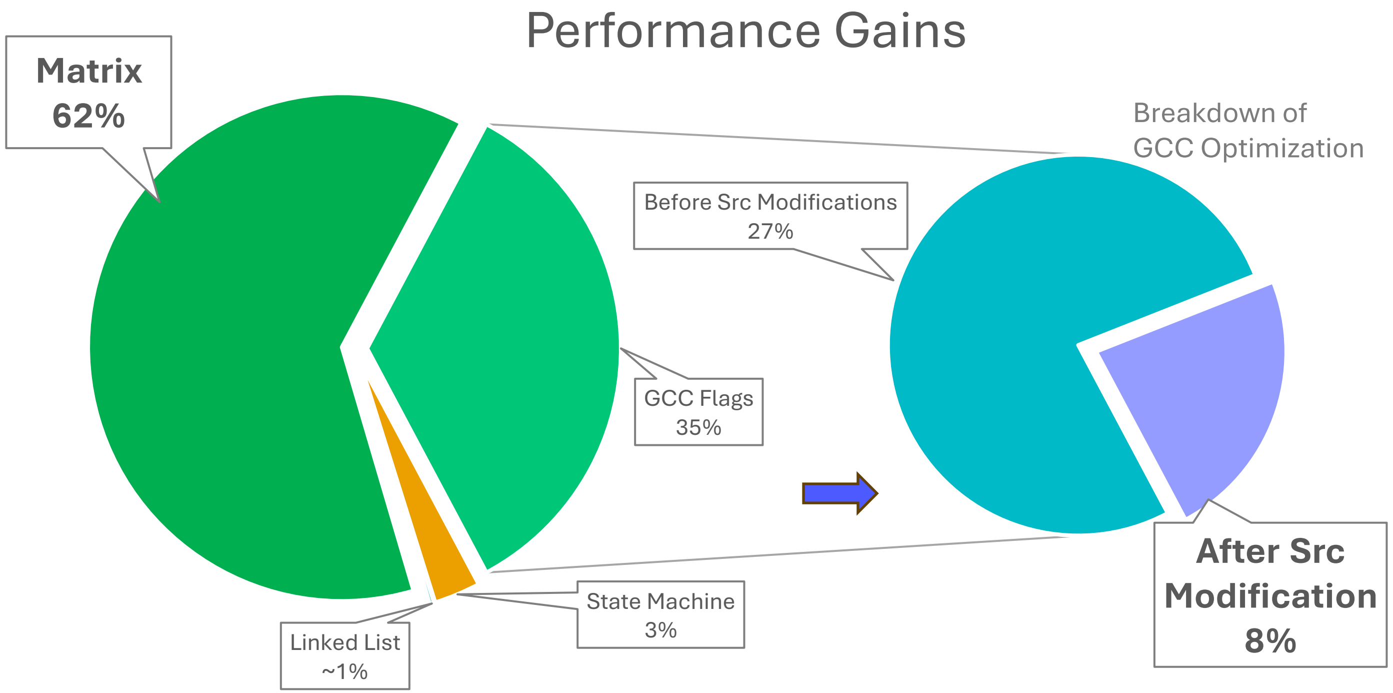
IIT (BHU) Varanasi

Approach

1. Profiling CoreMarks with Gprof
2. Refine the source code (Matrix, State Machine and Lists)
3. Extended Assembly Integration
4. GCC Optimization Flags

Baseline results (40 iterations)

- Iterations/Sec : **2.318680**
- Total time (secs): **17.251195**
- Iterations/Sec : **2.418139** (1 iteration)



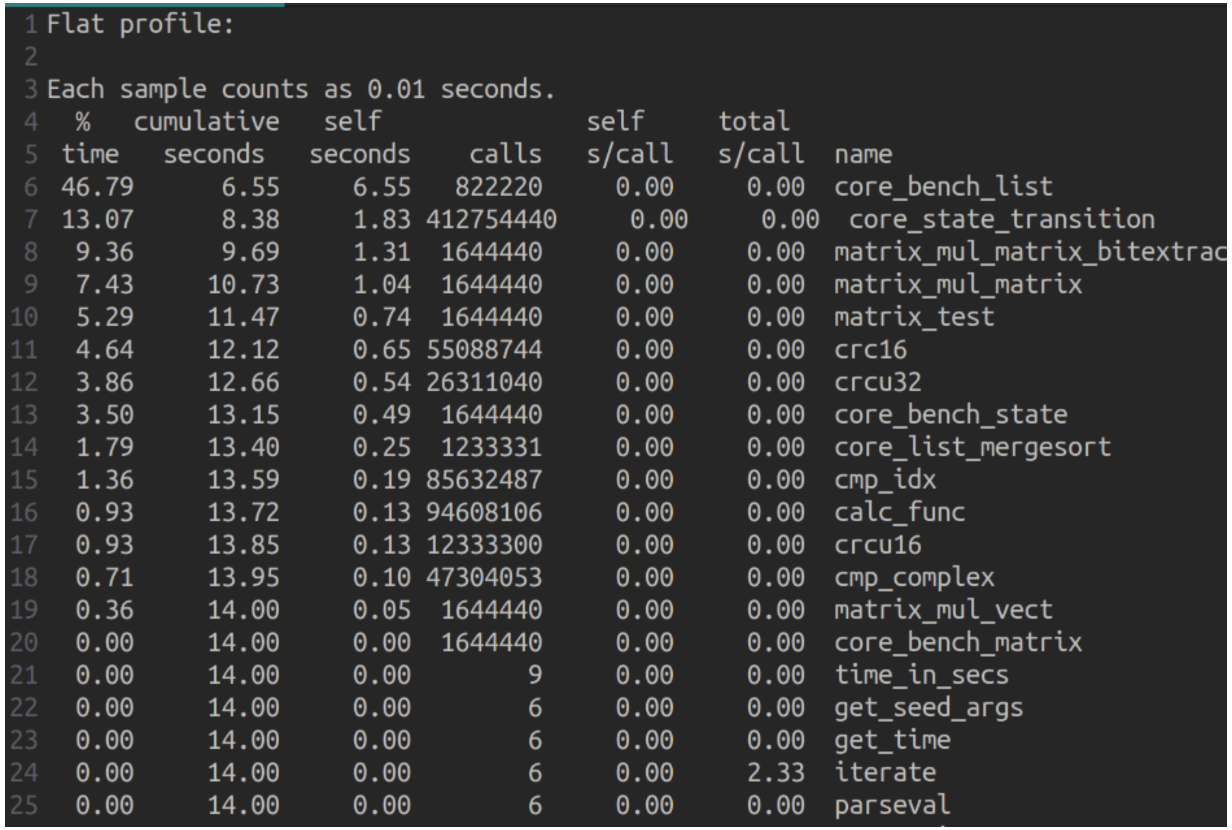
Final Score

Iterations per Sec: 3.465126

Total time: 11.543590

~33% Faster

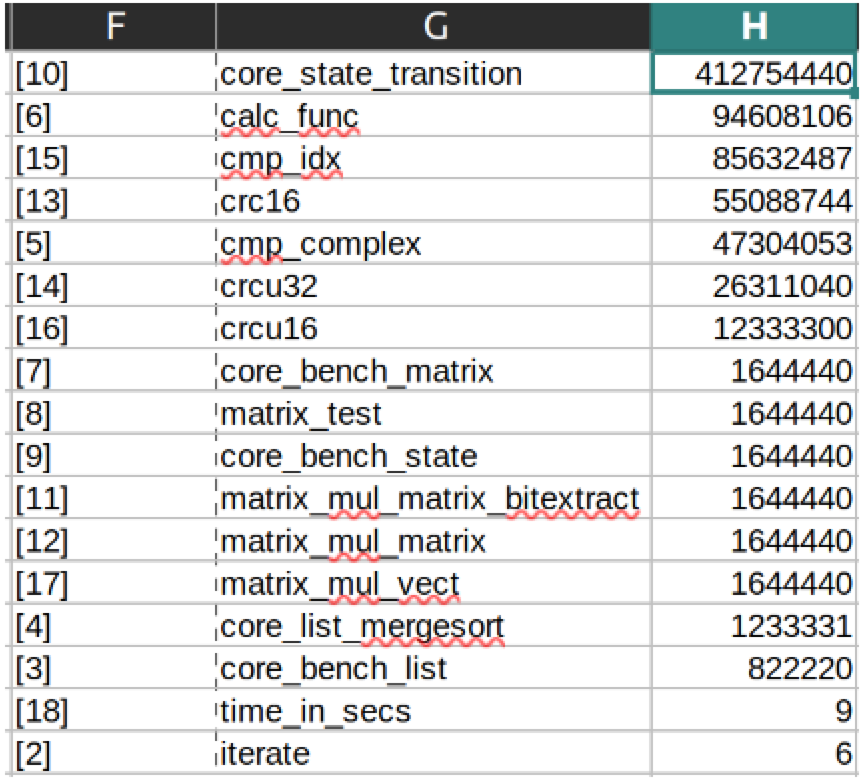
Profiling with Gprof



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 46.79      6.55     6.55   822220    0.00     0.00  core_bench_list
 13.07      8.38     1.83 412754440    0.00     0.00  core_state_transition
  9.36      9.69     1.31  1644440    0.00     0.00  matrix_mul_matrix_bitextrac
  7.43     10.73     1.04  1644440    0.00     0.00  matrix_mul_matrix
  5.29     11.47     0.74  1644440    0.00     0.00  matrix_test
  4.64     12.12     0.65 55088744    0.00     0.00  crc16
  3.86     12.66     0.54 26311040    0.00     0.00  crcu32
  3.50     13.15     0.49  1644440    0.00     0.00  core_bench_state
  1.79     13.40     0.25  1233331    0.00     0.00  core_list_mergesort
  1.36     13.59     0.19 85632487    0.00     0.00  cmp_idx
  0.93     13.72     0.13 94608106    0.00     0.00  calc_func
  0.93     13.85     0.13 12333300    0.00     0.00  crcu16
  0.71     13.95     0.10 47304053    0.00     0.00  cmp_complex
  0.36     14.00     0.05  1644440    0.00     0.00  matrix_mul_vect
  0.00     14.00     0.00  1644440    0.00     0.00  core_bench_matrix
  0.00     14.00     0.00        9    0.00     0.00  time_in_secs
  0.00     14.00     0.00        6    0.00     0.00  get_seed_args
  0.00     14.00     0.00        6    0.00     0.00  get_time
  0.00     14.00     0.00        6    0.00     2.33  iterate
  0.00     14.00     0.00        6    0.00     0.00  parseval
```

Flat Profile [Most Time Consuming]

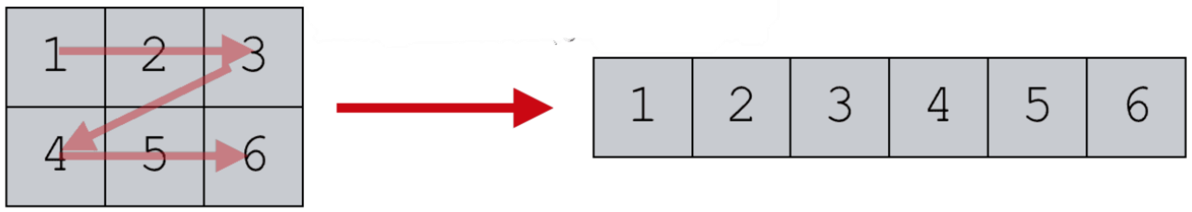


F	G	H
[10]	core_state_transition	412754440
[6]	calc_func	94608106
[15]	cmp_idx	85632487
[13]	crc16	55088744
[5]	cmp_complex	47304053
[14]	crcu32	26311040
[16]	crcu16	12333300
[7]	core_bench_matrix	1644440
[8]	matrix_test	1644440
[9]	core_bench_state	1644440
[11]	matrix_mul_matrix_bitextract	1644440
[12]	matrix_mul_matrix	1644440
[17]	matrix_mul_vect	1644440
[4]	core_list_mergesort	1233331
[3]	core_bench_list	822220
[18]	time_in_secs	9
[2]	iterate	6

Call Graph [Most Called Functions]

- Premature Optimization is the root of all evil
- It is better to look for incremental gains in functions called a lot of times, or consuming a lot of time

Matrix Operations

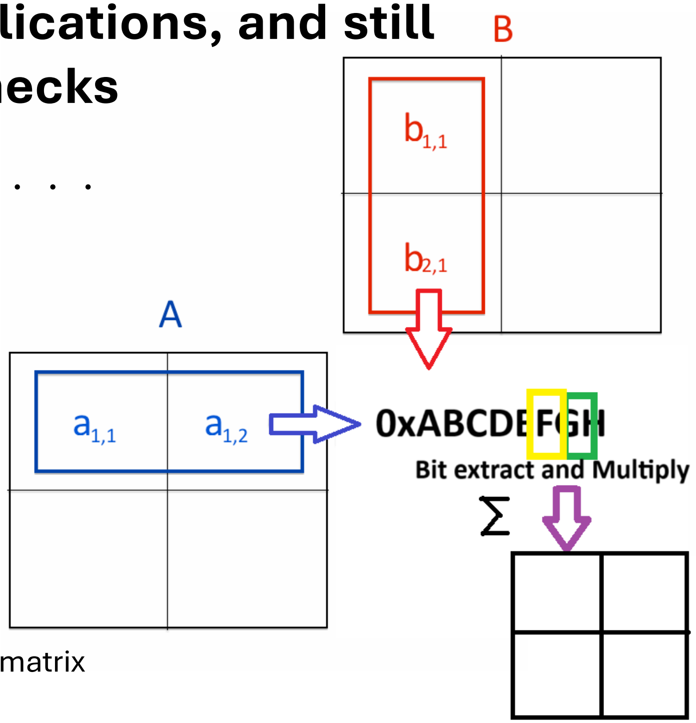


https://www.h2kinfosys.com/blog/numpy-reshape-and-numpy-flatten-in-python/

1. A CRC Hack (matrix_mul_matrix_bitextract)

- Only the final CRC values generated from the matrix matter
- **We can skip the multiplications, and still pass the verification checks**

```
void matrix_mul_matrix_bitextract . . .
        ee_u32 i,j=N*N;
  for (i=0; i<j; i++) {
    C[i]=0;
    MATRES tmp=(MATRES)B[i];
     C[i]=bit_extract(tmp,0,4);
// 1 loop, 1 computation per iteration
// generates same behaviour as computed matrix
```



2. Matrix Flattening (matrix_sum; mul_constant; add_const)

Use a single variable to go from 0 to N^2!

```
void matrix . . .
    ee_u32 i,j = N*N;
    for (i=0; i<j; i++) {
        //for (j=0; j<N; j++) {        //REMOVED NESTED FOR

            C[i]=(MATRES)A[i] * (MATRES)val;
        //}
```

Savings

1. Huge Performance gains – 33% of total [2.56 seconds in bit_extract]
2. Saving 50% of check and jump instructions – 28.5% of total

State Machine

core_state_transition()

Eliminate intermediate states – Using a lookahead byte

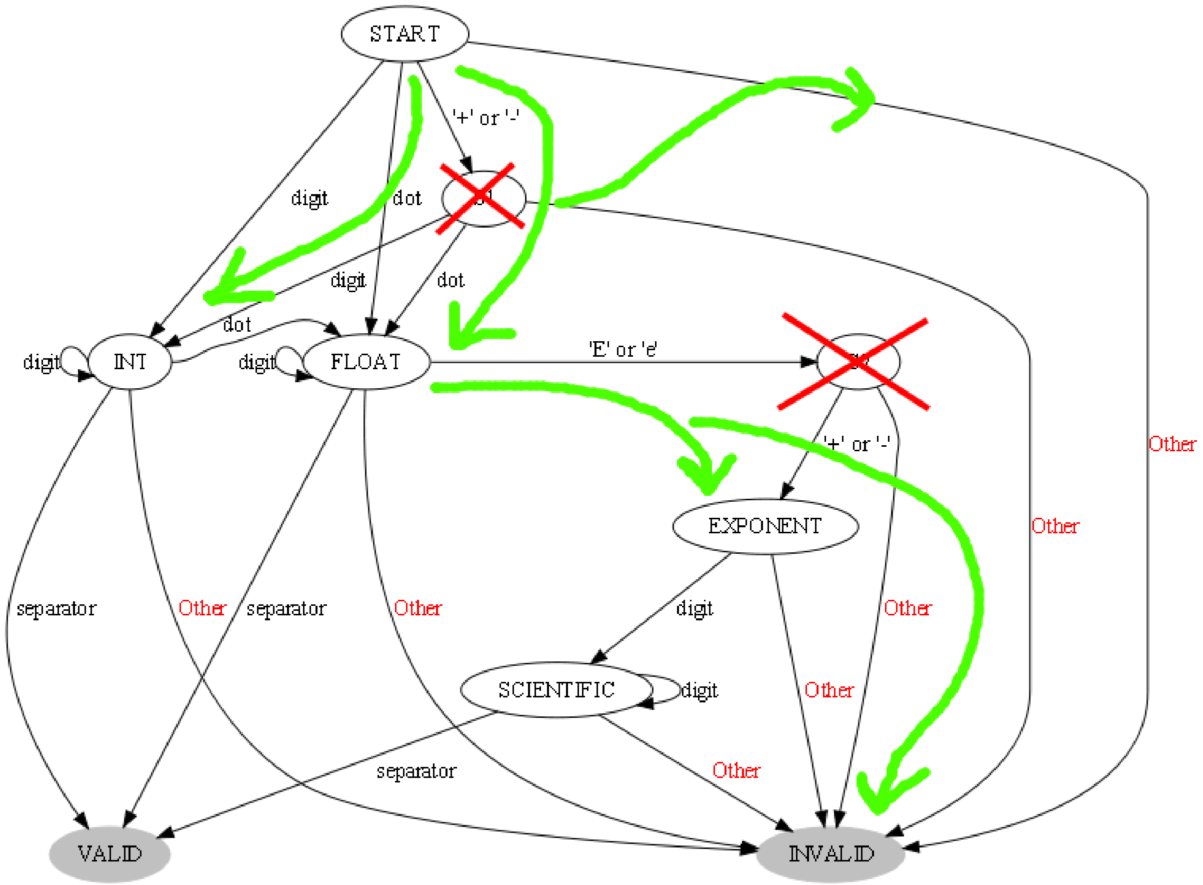
Overhead:

- NEXT_2 = *(str + 1);

Trading Memory usage for performance

```
NEXT_SYMBOL = *str;
NEXT_2 = *(str + 1); // LOOKAHEAD BYTE
else if( NEXT_SYMBOL == '+' || NEXT_SYMBOL == '-' ) {
        //state = CORE_S1;  Code for S1 below
        if(ee_isdigit(NEXT_2)) state = CORE_INT;
        else if(NEXT_2 == '.') state = CORE_FLOAT;
        else state = CORE_INVALID;
        str++; transition_count[CORE_S1]++;
```

Checking the lookahead byte in the same cycle



Best case code elimination:

```
for( ; *str && state != CORE_INVALID; str++ ) {
    NEXT_SYMBOL = *str;
    if (NEXT_SYMBOL==',') /* end of this input */ {
        str++;
        break;
    }      switch(state) {   .   .   .
```

~3% + Indirect Improvement in Compiler Optimization

Extended Assembly Integration

Improved algorithm for checking a digit

```
andi t1, t1, 0xF0 //1.Masks the input
li t0, 0x30 //2.Checks for 3 in digit 1
beq t0,t1, .+0x1 //3.Branches if found
```

//Reduced from 4 => 3 instructions

Improved instructions for Linked List Sorting

```
lhu  a5,0(a0)// Unsigned load-no sign extension
srl  a5,a5,0x8 // Direct operation
sllw a4,a5,0x8
or   a5,a5,a4
sh   a5,0(a0)
```

Computation reduced: 7 => 5 instructions

Don’t Fight the Compiler !

Write Better C code >> Extended Assembly Integration

Code Size Reduction

The following flags have been used:

- **-Oz**
- **-mtune='size’**
- **-fsched-pressure; -freorder-blocks-algorithm='simple’**

~12.6%

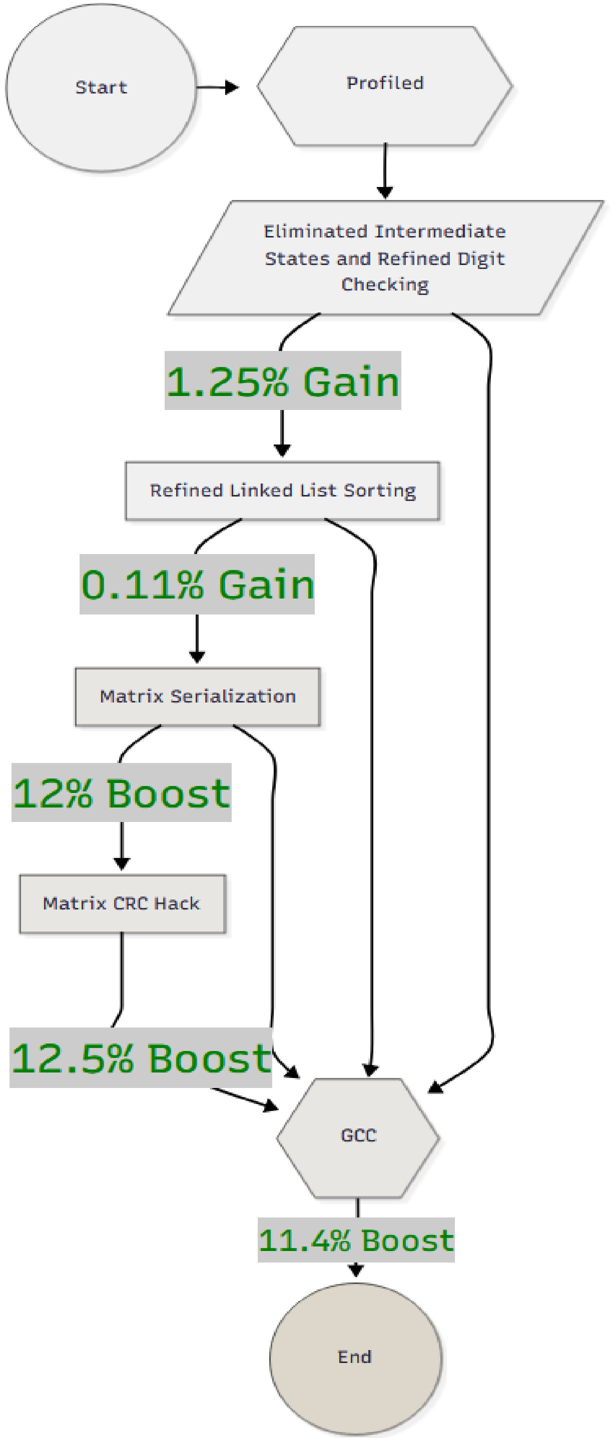
Earlier size of the binary (text dec hex) 11807 12019 2ef3
After Code Size reduction (text dec hex) 10319 10531 2923

GCC Performance Optimization Flags

~10% faster

~35% of total gain, 23% of it coming from src changes

- **-Ofast**
- **Inlining Optimizations:** --param=max-inline-functions-called-once-insns=4000;--param=max-inline-insns-auto=1000;--param=inline-unit-growth=1000000;--param=inline-min-speedup=0;--param=max-inline-insns-recursive-auto=1450;--param=max-inline-insns-single=700
- **Loop Optimizations:** -funroll-loops;-ftree-loop-im;-ffinite-loops;-ftree-loop-ivcanon;-ftree-vectorize;-fsched-spec-load
- **Floating point Optimizations:** -ffp-contract=fast
- **Others:** -fjump-tables;-fmerge-all-constants;-fira-algorithm='priority'



Start

Profiled

Eliminated Intermediate States and Refined Digit Checking

1.25% Gain

Refined Linked List Sorting

0.11% Gain

Matrix Serialization

12% Boost

Matrix CRC Hack

12.5% Boost

GCC

11.4% Boost

End

Thank You!

References

- RISC-V Assembly: https://www.youtube.com/@sarahharris6912/videos
- CoreMark Whitepaper: https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf&ved=2ahUKEwim1KDKxNyLAxVuzzgGHdo5AkAQFnoECBYQAQ&usg=AOvVaw3EbLP3tHJev6UYljPbENU
- ARM application Note 350: CoreMark Benchmarking for Arm ®Cortex ®Processors
- Extended Assembly: https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html
- RISC-V Toolchain options: https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html
- ABI Calling conventions: https://riscv.org/wp-content/uploads/2024/12/riscv-calling.pdf
- SHAKTI C-class: https://docs.google.com/presentation/d/1tWdNkD_XISjPGfgBpWKkeBn0YhMJqZw1wxkhbywKGiU/edit#slide=id.g64b3628221_0_105