Link to repo = https://github.com/scandum/wolfsort

The workload I chose is a benchmark of a sorting algorithm that the owner of the repository had made. The sorting algorithm in mind is called **wolfsort**, it is a hybrid sorting algorithm that combines properties of radix sort, quicksort and merge sort. The workload runs a benchmark that compares wolfsort to other hybrid sorting algorithms.

Downloading and running the workload

To download the repository, run the following command

```
git clone https://github.com/scandum/wolfsort
```

After cloning the repository, you can run and compile the benchmark as such:

```
cd wolfsort/src
gcc -O3 bench.c
./a.out
```

To compile it into a RISC-V binary run the following compile line

```
riscv64-linux-gnu-gcc -O3 -static bench.c -o bench.rv
```

The bash script I used to run the workload is the following code

```
ESESC_BIN=${1:-../main/esesc}

export ESESC_ReportFile="part2Report"
export ESESC_BenchName="./wolfsort/src/bench.rv"
if [ -f $ESESC_BIN ]; then
  $ESESC_BIN
else
  $ESESC_BenchName
fi
exit 0
```

Initial Results

The initial run of the benchmark showed an IPC of 0.39, and a total instruction count of 362,993,011 instructions, with a running time of 676 seconds. The full report given by esesc can be seen below

```
**************************************************************************************************
# File  : esesc_part3Report1.xT6pxO : Wed Nov 30 16:22:46 2022
**************************************************************************************************
Sampler 0 (Procs  0)
         Rabbit Warmup  Detail  Timing  Total  KIPS
  KIPS   94029    N/A   251972    606   10775
  Time     5.9%   0.0%    1.8%   92.2%          : Sim Time (s) 676.409 Exe  551.860 ms Sim (1700MHz)
  Inst   51.8%    0.0%   43.0%    5.2%          : Approx Total Time 10642.763 ms Sim (1700MHz)
**************************************************************************************************
Proc : Delay : Avg.Time :  BPType           :  Total  :        RAS        :       BPred       :        BTB        :  iBTB    :  BTAC    : WasteRatio  : MPKI
   0 :     3 :   12.809 :  2bit             :  89.73% :  99.99% of  8.36% :  88.28% of 87.64% :  88.15% of 49.47% :   0.00% :   0.00% :   0.00%         23.50
   0 :     4 :   12.809 :  2level           :  89.73% :   0.00% of  0.00% :  88.69% of 90.80% :  69.19% of  0.09% :  49.38% : (  3.22% fixed) :        23.50
--------------------------------------------------------------------------------------------------------------------------------------
Proc :  nCommit   :   nInst   :  AALU   :  BALU   :  CALU   :  LALU   :  SALU   :  LD Fwd :    Replay    : Worst Unit  (clk)
   0 : 362993011  : 362993052 :  47.87% :  24.80% :   0.05% :  14.37% :  12.91% :  0.02%  :      N/A     :   0.00
--------------------------------------------------------------------------------------------------------------------------------------
Proc  IPC     uIPC    Active        Cycles    Busy   LDQ   STQ  IWin   ROB  Regs    IO  maxBr  MisBr Br4Clk brDelay
   0   0.39   0.39   1.00         938162540   19.3   0.0  51.5   0.2   0.0   1.0   0.0   0.0   0.0    0.0    8.8
**************************************************************************************************
Cache          Occ  AvgMemLat MemAccesses    MissRate      (  RD ,    WR,    BUS)
IL1(0)         0.0  2.0       262853301    0.0%    0.0%    (100.0%,  0.0%,  0.0%)   29.8    0.0 GB/s
--------------------------------------------------------------------------------------------
DL1(0)         0.0  98.7      99024207    14.0%   53.7%    ( 77.7%, 44.3%,  0.0%)   11.2    0.0 GB/s
--------------------------------------------------------------------------------------------
L2(0)          0.0  186.3     13904016    92.4%   92.4%    (  7.6%,  0.0%,  0.0%)    1.6    0.0 GB/s
Memory(0)      0.0  60.0      13220519     0.0%    0.0%    (100.0%,100.0%,  0.0%)    1.5    0.0 GB/s
**************************************************************************************************
```

Finding the bottleneck

By looking at the report, it was clear that the L2 cache was underperforming heavily with a 90% miss rate on both types of misses. I double-checked that this was the case by testing changes with the L1 cache since it also had a relatively high miss rate at 50%. Some of the changes I tried were increasing both block size and associativity of the L1 cache, these changes resulted in little to no speedup.