

# RISC-V based computers in the data center

Bachelor Defence

Simon Vinding Brodersen, 20-06-2024

KØBENHAVNS UNIVERSITET



# Introduction

## Context

- Data centers play an ever increasing role in the IT sector.
- Large data movement between CPU and storage plane results in bottleneck for performance.
- Computational storage to reduce data movement.

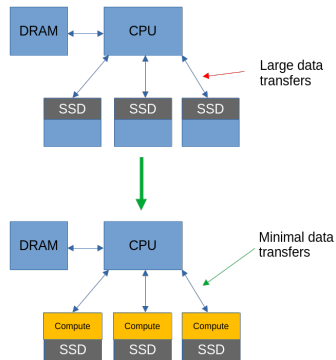


Figure: Traditional architecture vs with CSD.

# Introduction

## Problem and Approach

- What computation should be handled by a storage device?
- Is it feasible to implement such a computation on a RISC-V processor.
- Implemented following the RISC-V instruction set architecture.
- implemented on the QEMU virtual machine.



**Figure:** QEMU virtual machine. Created using Microsoft

Designer

## Background

### Accelerator-based Computer Architecture

- Off-loading the CPU.
- Optimized for distinct objectives, instead of General Purpose.
- Prominent example is the Graphical Processing Unit(GPU).

### RISC-V

- Reduced Instruction set Computing(RISC), version 5 (V).
- Open source, minimize intellectual property, reduce barrier of entry.
- Provides RV32I, RV64I and RV128I.



**Figure:** RISC-V logo. By RISC-V Foundation - Vectorised by Vulphere from <https://riscv.org/wp-content/uploads/2017/05/Tue1100-RISC-V-Foundation-Update.pdf>, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=69489388>

# Background

## QEMU

- Able to emulate both a 32-bit and 64-bit RISC-V processor.
- Made it possible to develop a binary file targeting RISC-V working on another ISA.



Figure: QEMU-logo. <https://www.logo.wine/logo/QEMU>

# Design

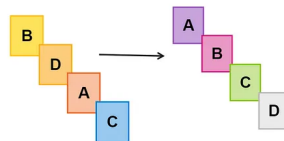
## Requirements

- Merge sort algorithm capable of running on a bare-metal RISC-V processor.
- Capable of sorting lists of varying sizes.

## Bare-metal

- Also known as Embedded system programming.
- Applications running without an underlying operating system.
- A lot of standard library implementations do not work out of the box.

## Sorting Algorithms



**Figure:** <https://medium.com/@noransaber685/understanding-sorting-algorithms-5575d52f5a18>

# Design

## Unikernel

- Bridge between a fully fledged operating system and bare-metal.
- Allows a single application to run with some OS-support.
- Nanos. Meant for virtual machines only. An alternative to Docker with Linux.

## freeRTOS

- Real Time Operating System(RTOS).
- Gives some standard library functionality.
- Allows for more platform independent applications.

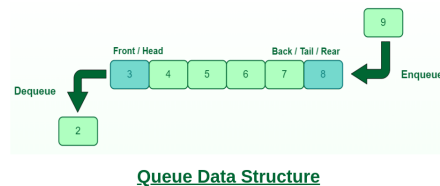
## Bare-metal

- Processor uses RISC-V ISA with extensions:
- Control and Status Register(Zicsr), Atomic Instructions(a), Integer Multiplication and Division (m), Compressed Instructions (c).
- Single vs multicore. Merge sort has a parallel structure. Parallel should in theory increase performance.

# Design

## Scheduler vs Partitioning

- Assuming cores have different execution speed, scheduler would allow for the faster cores to do more.
- Scheduler would require synchronization.
- Partitioning makes each cores amount of work deterministic.



**Figure:** A queue data structure.

<https://www.geeksforgeeks.org/queue-meaning-in-dsa/>



# Design

## Memory

- Use context switching.
- Separate core memory and thread memory.
- When I had issues with corrupted data, I would have an idea of where it was happening.

## Partitioning the list

- Partition the list following the structure of the merge tree.
- Create threads for each level of the merge tree.
- Bottom level regular sort and not merge.

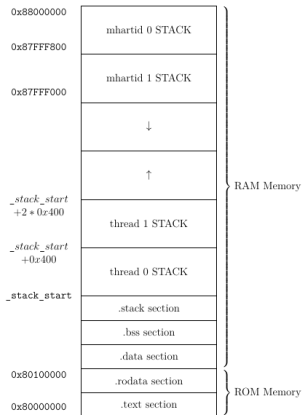


Figure: Design of memory layout.

# Implementation

## Getting to the main function

- I implemented a linker script, defining the memory section of the QEMU virt machine
- Using the Zicsr extension I am able to read the mhartid.
- Each core loops to create a stack of size STACK\_SIZE.
- Primary core goes to main function, all else to secondary\_main function.

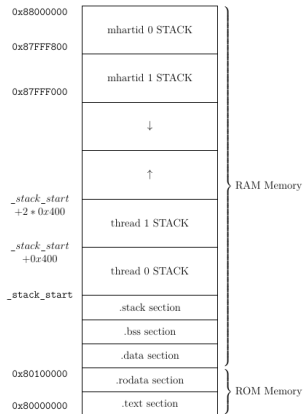
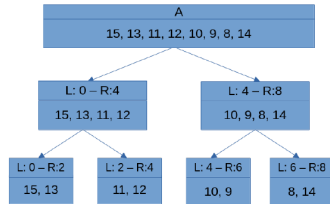


Figure: Design of memory layout.

# Implementation

## Partitioning the lists, and creating the threads

- mhartid 0 is in charge of partitioning the list.
- Each thread merges a subsection of the list.
- When the number of cores is equal to the number of elements on a level, the context is instead to mergesort that subsection.
- Each core can offset into the global THREADS list, for the next element it has to sort.
- Using the atomic extension, each core tells parent when they finish.



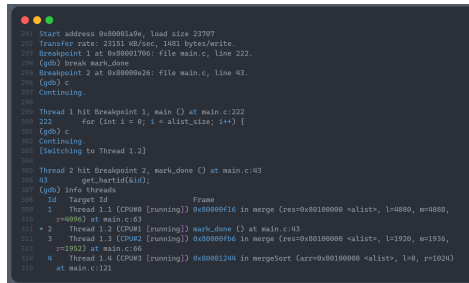
THREADS := [(A, 0, 8, MERGE), (A, 0, 4, MERGE), (A, 4, 8, MERGE),  
(A, 0, 2, MERGESORT), (A, 2, 4, MERGESORT), (A, 4, 6, MERGESORT),  
(A, 6, 8, MERGESORT)]

Figure: Design of memory layout.

# Evaluation

## Debugging

- Debugging using the riscv32-unknown-elf-gdb debugger.
- Able to see the different cores active as "Threads".
- Connects to the QEMU virtual machine remotely.



```
291 Start address 0x80001a9e, load size 23707
292 Transfer rate: 23151 KB/sec, 1481 bytes/write.
293 Breakpoint 1 at 0x80001706: file main.c, line 222.
294 (gdb) break mark_done
295 Breakpoint 2 at 0x8000e26: file main.c, line 43.
296 (gdb) c
297 Continuing.
298
299 Thread 1 hit Breakpoint 1, main () at main.c:222
300 222     for (int i = 0; i < alist_size; i++) {
301 (gdb) c
302 Continuing.
303 [Switching to Thread 1.2]
304
305 Thread 2 hit Breakpoint 2, mark_done () at main.c:43
306 43     get_hartid(&id);
307 (gdb) info threads
308 Id      Target Id      Frame
309 1      Thread 1.1 (CPU#0 [running]) 0x80000f16 in merge (res=0x80180098 <alist>, l=4880, m=4880,
310      r=8096) at main.c:63
311 * 2      Thread 1.2 (CPU#1 [running]) mark_done () at main.c:43
312 3      Thread 1.3 (CPU#2 [running]) 0x80000fb6 in merge (res=0x80180098 <alist>, l=1920, m=1936,
313      r=1952) at main.c:66
314 4      Thread 1.4 (CPU#3 [running]) 0x80001244 in mergeSort (arr=0x80180098 <alist>, l=0, r=1024)
315      at main.c:121
```

Figure: Debugging

# Evaluation

## Validation

- Tested using varying sizes.
- Redirects QEMU stdout to file and compares with python sort.
- Found that number of cores has to be less than list size.
- Number of cores has to be a power of 2. Due to the partitioning.

## Future work

- Variable thread stack size.
- Performance testing. QEMU emulates number of cores.

Lower:Upper:Number	NUM_CORES					
	2	4	8	16	32	64
-100:0:100	pass	pass	pass	pass	pass	pass
0:100:100	pass	pass	pass	pass	pass	pass
-50:50:100	pass	pass	pass	pass	pass	pass
-50:50:10	pass	pass	pass	fail	fail	fail
-1000:1000:1000	pass*	pass*	pass*	pass*	pass*	pass*
-1000:1000:20000	pass*	pass*	pass*	pass*	pass*	pass*
-1000:1000:100000	pass*	pass*	pass*	pass*	pass*	pass*

\*This run initially failed due to stack overflow. After increasing the different stack sizes following Appendix A it passed.

Figure: Lists of varying sizes tested

## Related work

- Marcelino et al. evaluate three hardware sorting algorithms implemented with Field Programmable Gate Arrays. One being the FIFO-based merge sorting machine with a merge FPGA. They find speed-ups between 1.6 and 15 times compared to a quick sort pure software.
- Jackson et al. provide a merge sort for sorting with minimal memory usage which aims to reduce the number of WRITES. They find it reduces I/Os and execution time by about 30%.
- Lobo et al. compare the performance of different types of merge sort algorithms. They theorize that the parallelized merge sort algorithm could increase the performance over a serial counterpart.

## Conclusion

- The creation of custom bare-metal solution comes with complications.
- Could be a deterrent, where alternatives might be more feasible.
- In the context of large data centers, the complexity could be worth it given large enough magnitude of demand.
- In this thesis I demonstrate that custom made solutions are feasible.
- A custom computational storage device might be both viable and advantageous. However, future research will determine whether the performance enhancement justify the investment.