ENCE464 Assignment 2: Computer Architecture

Group 13: Jack Duignan, Isaac Cone, Daniel Hawes

The result of running the completed program over a range of cube sizes for 300 iterations using 20 threads can be found in Figure 1.

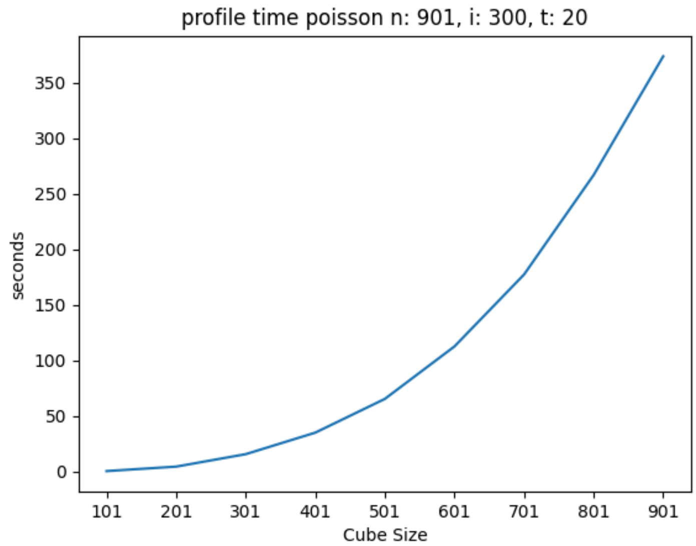

Figure 1: A complete run of the firmware across all cube sizes with 300 iterations and 20 threads.

Architecture Overview

The Central Processing Unit (CPU) described in this section is the AMD Ryzen 9 6900HX. Released in 2022, this CPU 8 identical cores with 2 threads per core for a total of 16 logical cores. The CPU uses the x86-64 instruction set architecture. The CPU structure is shown in Figure 2.

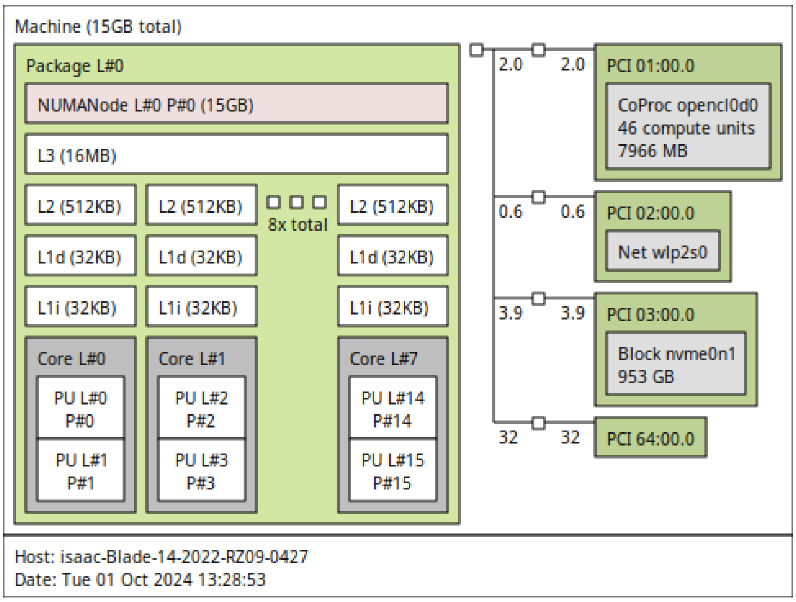

Figure 2: Central Processing Unit (CPU) architecture for the x86-64 AMD Ryzen 9 6900HX.

Cores

Memory

Instruction Set

big picture

ALU

FPL

Cache

Intruction decodeing etc.

Multithreading - easy - Daniel

row selection memcopy barrier

Cache - hard

Profiling - easy - Jack

Profiling was used through-out all stages of this projects development. This was done to identify which areas of the program where the slowest and how often these slow areas where called. From these results optimisations where made to the code to reduce execution time. When selecting areas of the code to optimise the sections called most often were prioritised as these give a larger performance benefit then optimising slower less frequent functions. To make profiling easier the various components of the code where compartmentalised into functions, while this does add some execution time (due to stack overheads) it allows the profiling tool gprof to provide more granular results.

Profiling was conducted on both optimised and non-optimised code to gain a wholistic understanding of the programs execution. A breakdown of the execution times and call counts for a non-optimised run of the program with a 201 node cube over 300 iterations using 20 threads can be seen in Table 1. The result of profiling using final optimisation parameters (-03) on the same cube size as before can be found in Table 2.

Table 1: GProf results for a non-optimised run of the program with 201 nodes 300 iterations and 30 threads.

Function	**Call Count**	**Time per call (ms)**
`poisson_iteration_inner_slice`	5965	1.25
`memcopy_3D`	5977	0.61
`apply_von_neuman_boundary_slice`	5956	0.05
Barrier waits cumulative	11945	0
Setup	0	0

Table 2: GProf results for a O3 optimised run of the program with 201 nodes 300 iterations and 30 threads.

Function	**Call Count**	**Time per call (us)**
`poisson_iteration_inner_slice`	5958	676.40
`memcopy_3D`	5971	410.32
`apply_von_neuman_boundary_slice`	5926	37.12
Barrier waits cumulative	N/A	N/A
Setup	0	0

The results found in Table 1 and 2 show that in both runs the largest time cost is the iteration over the inner slice of the cube. This is expected as it performs the majority of the floating point operations in the software. The next highest excution time is the application of the Von Neumann boundary.

In earlier iterations of the program the Von Neumann boundary was called at every inner loop of the main poission iteration. Based on profiling the team was able to identify this as a bottle neck as it is unessasary to call this for all if the inner nodes and move the updates to its own self contained iteration that only iterates over the outside nodes. Another example of profiling helping in optimisation of code is with the barrier waits that are used to synchronise the threads. Originally the team hypothesised that these waits would greatly increase the execution time as threads take different amounts of time to complete due to cpu allocation and the way the cube nodes are divided amongst them. By profiling the code with these barriers implemented it was discovered as can be seen in Table 1 that the barrier waits do not add any appreciable execution time and in the optimised version of the code seen in Table 2 are even expanded out of there respective functions and executed in the code itself with no function call overhead. Without profiling this would have been much harder to identify and solve.

- Python script
- gprof outputs and how they were used

Compiler Optimisation - easy

Individual Topic 1 Jack Duignan - Branch Prediction

- WHat is branch Prediction
- Branch prediction errors
- Changing of program
- Results of changing

Individual Topic 2 Isaac Cone - GPU

Individual Topic 3 Daniel Hawes - SIMD

References