Laboratory Assignment 1
Exploring the Impact of Cache Hierarchy on Processor Performance

Lecturer: Angelos Arelakis
TAs: Celestine Valan Moses, Chen Hongguang, Konstantinos Ioannis Sotiropoulos Pesiridis, Mohammad Ali Maleki, Neethu Bal Mallya

Last Updated: 12 September 2025
Original Contributors: Mehrzad Nejat, Mohammad Waqar Azhar, Per Stenstrom © 2021

Learning outcome: The objective of this laboratory assignment is to understand the impact of cache design on overall processor performance. You will use the SimpleScalar toolset [1] and the MiBench benchmark suite [2] to perform the experiments. At the end of this assignment, you should know how to:

- measure and report the performance of a cache model
- compare the relative performance of various cache models
- evaluate the impact of cache parameters on cache performance

Suggested reading: It is strongly recommended that you go through the following study material ([1] and [2] are available to be downloaded **Canvas → DAT105 Laboratory Assignments** under *“**Link to the Required Files**”*) before you start this assignment:

1. D. Burger and T. Austin, “*The SimpleScalar Tool Set Version 2.0*”, University of Wisconsin-Madison, Computer Sciences Department, Technical Report 1342, 1997.
2. M. R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge and R.B. Brown, “*MiBench: A free, commercially representative embedded benchmark suite*”, In Proceedings of IEEE International Workshop on the Workload Characterization (WWC-4), pages 3 – 14, 2001.

Evaluation: You must write a report following the specific requirements listed after each task. The report must not exceed 2 pages.

Simulation Environment

As depicted in Figure 1, the simulation environment basically consists of the following sections:

- **Benchmark**: This is a test program that is executed on the simulated processor. It can have its own inputs and outputs. You can run any program on the simulated processor; but to have a standard base for comparison, people usually use programs and input sets from well-known benchmark suites.
- **Simulator**: This is the simulator program. It is executed on a real computer referred to as the **host machine**. The host machine can be your laptop, a local computer in the Lab, or a server. The simulator is a software model of a computer system and runs as software on the host machine. It includes probes for measurements while simulating the execution of benchmark on the computer system model being simulated with the specified configuration. The number of executed instructions, the number of loads and stores, and the average cycles per instruction are just a few examples of many parameters that the simulator can measure.
- **Configuration File**: The architecture simulator uses this file to set the configuration and parameters of the processor and cache model in simulation. This is not the only method for modifying the processor and cache model. You can use parameter-specific command-line options or change the simulator code. But we will not use these methods in this lab. We provide a base configuration file. You can create different processor and cache models by modifying this file.
- **Results**: The simulator eventually reports the results in some output files. After simulating different processor models, you can compare these results to evaluate the performance of the models when running a specific benchmark program with a specific input set.

In summary, you must remember that you are running a benchmark program on a simulated computer system model with the specified configuration on a real computer (host machine). Therefore, for example you have **two different run times**: *program runtime* on the simulated processor, and runtime of the simulation – the *simulation time* – on the host machine. Simulation time is usually several orders of magnitude larger than program run time.

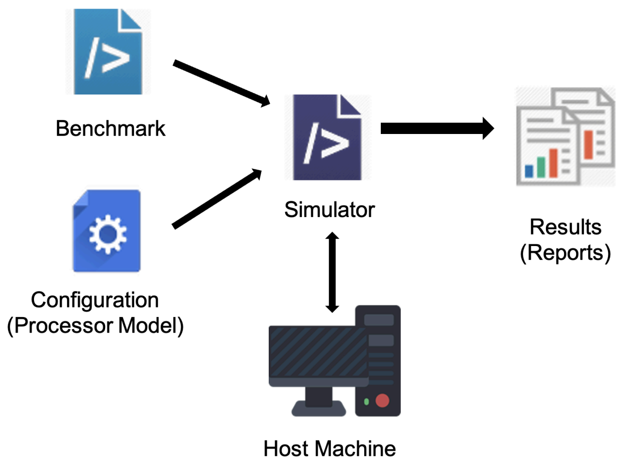

Figure 1: Simulation Environment

Running a simulation

Your host machines: You must run the simulator in **a Linux system**. You can use your own machine or connect remotely to the StuDAT computing service from your personal computer. For platform-specific instructions, see the official guide in https://chalmers.topdesk.net, which covers access from Windows, macOS or Linux.

Important: StuDAT only supports RDP connections, not SSH. When you close the remote desktop window, your session ends immediately, and any running simulations will be terminated. Make sure you have time to complete your simulations before starting them. **You can't just launch one and come back later**.

Your simulator and configuration file: To begin the lab, download the simulator package from **Canvas → DAT105 Laboratory Assignments** under ***“Link to the Required Files”***. Select *“Simulator for Lab 1 and Lab 2”* which downloads the archive **simhome.tar.gz**.

After extracting this file with `tar -xvzf simhome.tar.gz`, you will see three main folders:

- The `apps` folder contains five MiBench benchmark applications. We are using the small input sets to reduce the simulation time.
- The `bin` folder holds two SimpleScalar simulator executables:
  - `sim-profile` is used for fast functional simulation of the benchmark applications to provide statistics on instruction mix. We will not use this simulator in this lab; but you can try it yourself.
  - `sim-outorder` performs detailed architectural simulation of an out-of-order processor. This is the main simulator that you will use.
- The `configs` folder is where you place different processor models and simulation results are stored in the corresponding sub-folders.

Besides these folders, you can find the following files as well:

- `base1.txt`: Base configuration file
- Two scripts for running out of order simulator and profiling simulator: `runsim_sim` and `runsim_profile` respectively. Open these scripts and see how each simulation is initiated. You can pass the name of the configuration as a command line argument to the script. You can deactivate any benchmark application by simply changing the corresponding lines to comment.
  - `./runsim_sim $DIR $CFG`, where **$DIR** is the name of the directory inside `./configs` where the configuration file is located, and **$CFG** is the name of the configuration file **without** `.txt`, e.g., `./runsim_sim Lab1-Task1 base1`
  - For running `runsim_profile`, simply create a directory named `profile` inside `./configs` and execute `./runsim_profile` from the command line.
- `stats_parser`: a bash script to help you parse the simulation stats more easily.
  - Copy the `stats_parser` file to `simhome/`
  - Run `./stats_parser $DIR $CFG`, e.g., `./stats_parser Lab1-Task1 base1`

Remember, after each simulation, two different sets of outputs can be generated: ***Application outputs*** in the same location of the application executables, and ***simulation outputs*** in the same location as the configuration files. These locations are set in the script.

Your benchmarks: Each student group will focus ONLY on 3 out of 5 MiBench benchmarks you see in the `apps` folder. Please note the benchmarks based on your ***GroupID*** as follows:

*If your **GroupID** ends in ....*	*List of applications*
0	dijkstra, gsm-untoast, jpeg-cjpeg
1	dijkstra, gsm-untoast, qsort
2	dijkstra, gsm-untoast, stringsearch
3	dijkstra, jpeg-cjpeg, qsort
4	dijkstra, jpeg-cjpeg, stringsearch
5	dijkstra, qsort, stringsearch
6	gsm-untoast, jpeg-cjpeg, qsort
7	gsm-untoast, jpeg-cjpeg, stringsearch
8	gsm-untoast, qsort, stringsearch
9	jpeg-cjpeg, qsort, stringsearch

Hints/Tips:

- Remember to deactivate the 2 benchmarks that are not relevant to your group in the `runsim_sim` script.
- If you are not familiar with Linux scripts, you should check at least some basic structures and simple examples on the internet. It will not take a lot of your time, but it can be a powerful tool.
- If you are familiar with linux scripts, you can easily alter the provided scripts as you wish. But make sure to sustain the correct order of the arguments passed to the simulator and keep a copy of the original scripts.
- Since you'll be running multiple (and repetitive) SimpleScalar simulations (in Lab 1 and Lab 2 tasks), it would be efficient if you could automate the tasks by *writing scripts* to generate the different configs, run multiple simulations, and/or read the relevant results from the output Stats. This can save you a lot of time! :)

Task 1: Evaluating the base configuration

In this assignment, you will study a very simple system presented in Figure 2. Usually, there are two or three levels of cache between the processor and the main memory. One reason is that bigger caches come with longer access time and each level tries to hide the access time of the next level.

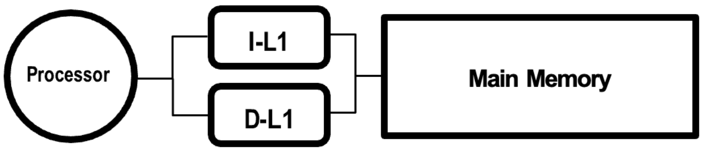

Figure 2: Architecture of the base system

However, to keep the analysis tractable while capturing the essentials, we begin with a minimal, representative system. Hence, here we only have one level of cache. But we keep the instruction and data caches separated as they are in most processors for the first-level cache (denoted I-L1 and D-L1).

We know that the execution time of a code can be modeled as:

$$\textit{Execution\_Time} = \text{IC x CPI x } T_{cycle} \qquad (1)$$

where IC is the instruction count, CPI is Cycles Per Instruction and T_{cycle} is the cycle time[1]. Because the processor frequency is constant in our case, we can remove cycle time from both sides of the equation and express all the times in processor cycles. Furthermore, since we run the same program on different computer system models with the same ISA, IC will be the same. Therefore, we can divide both sides by IC and consider only CPI – Cycles Per Instruction. This is especially useful when comparing the performance of multiple applications with different numbers of instructions. This means CPI could be used as an important parameter for measuring the performance of a processor.

CPI can be divided into two components[2]:

$$\text{CPI} = \text{CPI}_0 + \text{MPI x MP} \qquad (2)$$

Here MPI and MP stand for number of Misses Per Instruction and Miss Penalty in cycles, respectively. This is a simple model that applies to the system in Figure 2. The miss penalty component in (2) corresponds to the extra waiting time when accessed data or an instruction is not found in the data and instruction cache, respectively, and the processor needs to bring it from main memory. If we remove this component from CPI, what remains (we call it CPI_0) is the number of cycles per instruction related to processor operations and even if we had a perfect cache with no misses, it would not be affected.

[1] The inverse of processor frequency

[2] Usually in a multi-level cache and when cache access time is longer than one cycle, the second component of this equation is expressed in more details that include the multiplication of accesses per instruction to each cache or memory by the corresponding access time.

Now, let's start by simulating the base model and measuring its performance. **Note:** It takes several minutes to run a benchmark. Do not sit and wait for an experiment to finish. Go ahead and plan for the next experiment:

- Place the provided `base1.txt` configuration file in the corresponding folder in `simhome/configs`. Open it and see how different processor configurations are set in this file. Especially, check the configurations for caches. Use the SimpleScalar user guide [1] to understand the parameters. You will change these parameters to create new processor models.
- Execute `runsim_sim` for the base model[3].
- Read the results from `Stats_*.txt` and fill out the first five rows of Table 1.
- We need to establish CPI_0 for each benchmark. How would you configure the simulator to get CPI_0 for each benchmark?
- Simulate the ideal model and collect the CPI results as an estimation of CPI_0 and fill out the corresponding row in Table 1.
- Using CPI_{base} and CPI_0, calculate the upper bound on speedup (SP_{ideal}). Use the following formula for calculating the speed up of any configuration *x*:

$$SP_x = CPI_{base} / CPI_0 \qquad (3)$$

Table 1: Simulation results for base configuration

Application	*<application A>*	*<application B>*	*<application C>*
Instruction Count[4]			
Execution Time in cycles			
CPI_{base}			
MPI I-L1			
MPI D-L1			
CPI_0			
SP_{ideal}			

Report Requirements:

1. **Table 1 completion**
   a. Include the completed Table 1 with all measurements
   b. Explain how you configured the simulator to obtain CPI_0, what parameters did you change to create the "ideal" memory system?
2. **Memory subsystem impact analysis**
   a. If you could only optimize one cache to improve the performance, which cache would you choose?
   b. Calculate "Memory Stall %" for each application (add this row to Table 1).
   This represents the fraction of execution time spent waiting for memory.

   Hint: use CPI_{base} and CPI_0 to derive this percentage.

[3] All the files that need to be executed, should have executable permissions.
[4] Use number of committed instructions

Task 2: Optimization of the Caches

You have studied a base processor model in Task 1. In this task, the goal is to find an **instruction cache** configuration that improves performance for your benchmarks. The organizational parameters[5] that you can change are **cache size**, **associativity** and **block size**. However, increasing the cache size or associativity comes with longer cache hit time (in cycles) as shown in Table 2[6]. For example, whereas the cache hit time for a direct-mapped 4-KB cache is 1 cycle, the cache hit time for a 4-way associative 16-KB cache is 3 cycles.

Table 2: Cache access latency in cycles for different sizes and associativities

		Size			
		4 KB	8 KB	16 KB	32 KB
Associativity	1	1	1	2	2
	2	1	2	2	3
	4	2	2	3	3
	8	2	3	3	4

To do this, you need to devise a methodology to understand the impact each cache organization parameter has on the performance of the processor. It is important that you change only one parameter at a time to isolate the effect of other parameters. For example, if you change associativity and cache size together and observe some performance variations, you cannot determine how much each parameter in isolation impacts on the result. Every time you want to move to the next step in the simulations, create a new folder and configuration file in the `configs` folder. You can start by copying `base1.txt` and then modifying it.

Report requirements:

1. **Present your optimization goal**
   a. Which aggregate metric will you use to find the best configuration across your applications?
   b. Why would simply summing the total execution cycles across all applications not be a fair comparison?
2. **Methodology description**
   a. Define your parameter space and calculate the total design space size. How many configurations would you need to test for exhaustive search?
   b. How will you navigate this design space?
   c. Instructions usually just flow, right? One follows the other. Does this mean a bigger block size is always the right answer for an instruction cache?

[5] In the configuration file, cache size is not directly available. It is the multiplication of number of sets, associativity and block size. Therefore, **you must be careful when you want to change each parameter**. The cache configuration `<config>` is formatted as follows: `<name>:<nsets>:<bsize>:<assoc>:<repl>`. Check the simulator user guide for more details.

[6] Don't miss to update the cache access latency in the config file according to Table 2.

3. **Results presentation**
   a. Present two tables showing your aggregate metric: one for block size = 32B, another for block size = 128B. Each table should have cache sizes as columns and associativities as rows (similar to Table 2).
   b. Look at your tables. Does the best size/associativity combo change when you change the block size? In other words, are they independent?
   c. Report your optimal configuration: size, associativity, block size, and achieved aggregate metric.
   d. Pick one of your applications. Find the best cache configuration for only that application. Is this application-specific optimal config the same as the aggregate optimal configuration?