# CprE 381 - Computer Architecture and Assembly Level Programming Fall 2017 Lab-8

**INTRODUCTION:**  
In this lab, you will use the **sim-cache** simulator from the SimpleScalar toolset to compare the performance (Miss Rates) of various cache models. You will then use the **Cacti** tool to compare the power requirements of these configurations. This is a two-week, individual lab.

Before you begin, create a new lab directory in your home directory, name it **lab8**.

**Note: Sample commands for sim-cache given in this report assume that you are executing them from the /usr/local/ss/simplesim-3.0 directory.**

You can use **Putty** to access the remote servers like you did for labs 2, 3, 4 or use the Windows provided **Remote Desktop Connection** utility. The latter gives you a GUI. Both are available from the Start Menu. If you use the **Remote Desktop Connection** utility, open up a terminal after you have successfully connected to the server. You may connect to any of the Linux remote servers (linux-1, linux-2,linux-3 or linux-4).   
  
When you open up Putty or Remote Desktop Connection utility, type in for example***linux-1.ece.iastate.edu*** into the host name field if you want to connect to linux-1, and click Open.   
(Click YES if a pop-up window appears asking for key authentication).   
Enter your login credentials.

**Part 1**

Navigate to SimpleScalar installation directory using   
 **cd /usr/local/ss/simplesim-3.0**

You will find the simulator ***sim-cache*** inside this directory( invoke **ls** to view contents of the directory).  
The next step is important, and you **ONLY NEED TODO THIS ONCE**. We need the simulator to generate a configuration file that you can modify to run your tests. To generate this file, run the following:  
  
**./sim-cache -dumpconfig<path/to/lab8/dir>/config.cfg tests-pisa/bin.little/test-math**You now have a file in your directory named **config.cfg.** You will edit this configuration file for different cache configurations. Open up the file (using **vi** or **nano** or an editor of your choice). The file looks something like this:

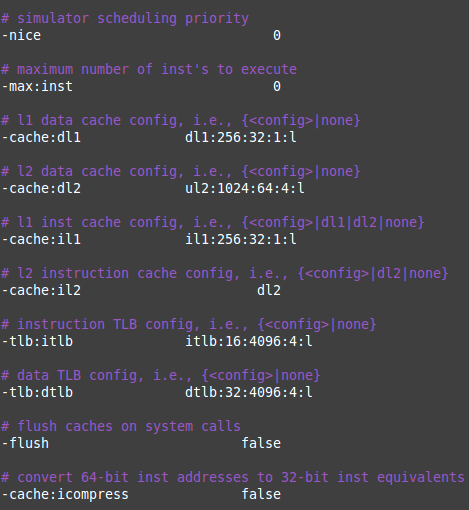
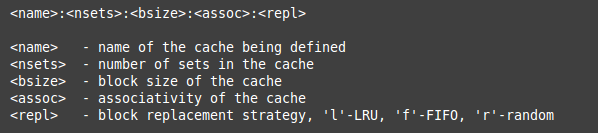


Figure.1

The file tells you in the comments exactly how you can configure each component. For example, the Level2 Data cache (shown as **dl2**) can be configured to **none** meaning that it does not exist or to **<config>.** The fields in **<config>** are:

  
 Figure.2  
  
For example, dl2:4096:32:1:l means a Level2 D-cache with 4096 sets, 32 bytes block size, Associativity = 1 and LRU replacement policy. And if you wish to disable Level2 D-cache, you would replace **dl2:4096:32:1:l** with **none.**

Let’s get started! We want to see variations in Miss Rates with varying #Sets and Associativity.

**A.**First, test the Level-1 D-cache. Disable the level-2 D-cache, level-1 and level-2 I-cache using the **none** attribute. For the level-1 D-cache, set the block size to 32bytes, Replacement policy to LRU, Associativity and #Sets as follows:  
  
(i) Vary Associativity as 1-way,2-way,4-way,8-way.  
(ii) Vary #Sets as 16,32,64,128,256.  
  
Run **sim-cache** on the **compress95**benchmark to test your cache. The benchmark is located inside the ***Benchmarks*** directory within **/usr/local/ss**. The benchmark might take well over a minute (or more depending on the server load) to execute, so please be patient! The command to run the simulator is:  
  
**./sim-cache -config<path/to/lab8/dir>/config.cfg ../Benchmarks/compress95.ss < ../Benchmarks/compress95.in**

For each, note down the **Miss Rates** for L1 D-cache to fill the table below**.**

**Miss Rates (Data Cache)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Sets\Associativity | 1-way | 2-way | 4-way | 8-way |
| 16 sets |  |  |  |  |
| 32 sets |  |  |  |  |
| 64 sets |  |  |  |  |
| 128 sets |  |  |  |  |
| 256 sets |  |  |  |  |

**B.**Repeat **A** for Level-1 I-cache. This time, disable both levels of D-cache, level2 I-cache and vary configurations for level-1 I-cache.

**Miss Rates (Instruction Cache)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Sets\Associativity | 1-way | 2-way | 4-way | 8-way |
| 16 sets |  |  |  |  |
| 32 sets |  |  |  |  |
| 64 sets |  |  |  |  |
| 128 sets |  |  |  |  |
| 256 sets |  |  |  |  |

Plot graphs for Miss Rates v/s Associativity (for #sets=16 and 128), and Miss Rates v/s #Sets (for 1 way and 4way Associativity) using MS Excel, Matlab or any other software for **A** and **B**. Submit them with your report.

**Part 2:**

In this part, we are going to see the effect of **Spatial locality** on Cache performance.  
Write a C program that populates two linear arrays (say X and Y)of length 1024000 each with numbers of **type int** in the range of 0 to 100, adds them and stores the result in a third array (Z).  
  
*(****Note:*** *declare your array variables as global variables, else you might overflow the stack)*  
**A.**Populate the arrays X and Y with random numbers, add X, Y and store the result in Z. (run a loop to compute Z[current index] = X[current index] + Y[current index])   
  
Compile using:

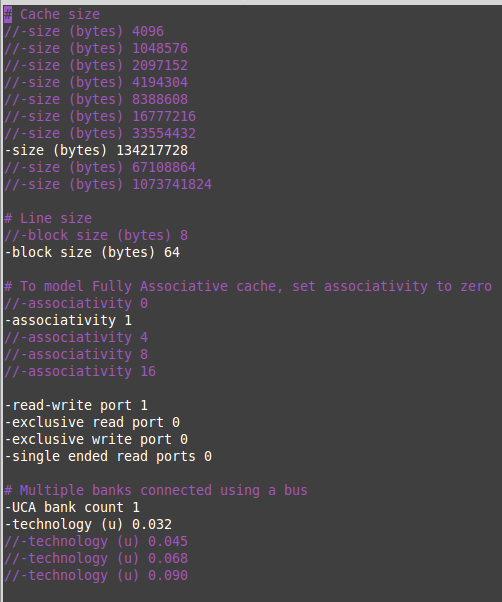
**../bin/sslittle-na-sstrix-gcc -O0 <path to lab8 directory>/<your .c file> -o <path to lab8 directory>/<executable name>**  
  
Now run **sim-cache** on the executable and find the **Miss rates** for Data Cache. In your config.cfg file, disable both level-2 caches like Part 1, use 64 sets, 4-way associativity, 64 byte block size for both dl1 and il1).

**./sim-cache –config<path/to/lab8/dir>config.cfg<path/to/lab8/dir>/<executable>**

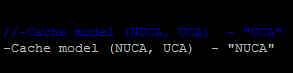
**B.**Edit your code to add random elements of X and Y and store them in the third matrix sequentially (the only difference from part **A** is that you are adding elements from random locations in arrays X and Y). Repeat the compilation and simulation steps from **A.**You have been provided with a template file***programtemplate.c***in the zip folder, use that for reference.   
  
Report the Miss Rates noted in **A** and **B.** Give your inference below. Submit your source codes for parts **A** and **B.**

**Part 3:**

Different cache configurations have varying power requirements. Modern day computing requires processors to be power efficient while keeping up with the computing needs. In this section, we will model various cache configurations and see how their **energy requirements** change.  
  
We will use **Cacti6** tool for the task. Cacti is another command line tool that models cache and memory access time, cycle time, area, leakage, and dynamic power. It is intended for use by computer architects to better understand the performance trade-offs inherent in memory system organizations.  
  
Cacti6.tgz has been provided to you in the zip file. Copy it to your **lab8** directory and extract it. This will create a **cacti6** directory. **Cd** into it andrun **make.**  
  
 **cd <path/to/your/lab8/dir>/cacti6/** **make**  
  
Cacti uses a configuration file to model caches, but you do not have to generate it. It is available to you as **cache.cfg,** you will find it in **cacti6** directory**.**The contents of the file look something like this:

  
 Figure.3

This configuration file gives you options for different parameters; you can uncomment the one you want to use or edit it with your own values. We will deal with cache size, block size, and associativity.   
Open up the **cache.cfg** file and look for the following:  
  
  
Change **“NUCA”** to “**UCA**”. We will not deal with Non-Uniform Cache Architecture(**NUCA**) in this lab. Edit the Block size, Associativity and Cache size as instructed below.Leave everything else to default.  
For a block size to **32bytes**find the Dynamic Read Energy and Access Times for the following:  
  
(i) Vary Associativity as 1-way,2-way,4-way,8-way   
(ii) Vary #Sets as 128,256,512,1024



Cacti uses the Cache size, Block size and Associativity as inputs, unlike sim-cache which takes #sets. Hence, for a given #sets, calculate the cache size using the formula:  
  
 ***Cache Size = Block Size \* # Sets \* Associativity.***

To run cacti, make sure you are inside the Cacti installation directory (cacti6) and that you have saved your configuration file. Then run the following:  
  
 ***./cacti***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Sets\Associativity | 1-way | 2-way | 4-way | 8-way |
| 128 sets |  |  |  | Not Supported |
| 256 sets |  |  |  |  |
| 512 sets |  |  |  |  |
| 1024 sets |  |  |  |  |

Read Energy (nJ)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Sets\Associativity | 1-way | 2-way | 4-way | 8-way |
| 128 sets |  |  |  | Not Supported |
| 256 sets |  |  |  |  |
| 512 sets |  |  |  |  |
| 1024 sets |  |  |  |  |

Access Time (ns)

Plot graphs for Read Energy v/s Cache size (for 2 different #sets, say 128 and 512), and Access Times v/s Cache size (for 2 different Associativities, say 1-way and 4-way) using MS Excel, Matlab or any other software. Submit them with your report.

**Part 4:**

To close the gap further between the fast clock rates of modern processors and the increasingly long time required to access DRAMs, most microprocessors support an **additional level of caching**. This second-level cache is normally on the same chip and is accessed whenever a miss occurs in the primary cache. In this section, we will see the effect of multi-level caches on cache performance.  
  
Edit your configuration file according to the worst performing configuration (highest Miss Rate) from **Part1** for D-cache. Now remove the **none** attribute from the level2 D cache (**dl2)** and add the following configuration to it:  
  
 -cache:dl2 dl2:2048:64:4:l

Save the configuration file. Now run the simulator exactly the way you did in **Part1.**Note the **Number of Misses** for L1 Data Cache and L2 Data cache. Say the L1 cache access latency is 1 cycle, L2 is 4cycles and Main Memory is 100 cycles.   
State the approximate % gain in Access time (in terms of #cycles) using the following formula:  
  
tm = MM latency, t1 = L1 latency, t2 = L2 latency  
m1 = #L1 misses, m2 = #L2 misses  
  
%gain = {m1 \* (tm+t1)–[m2 \*(t1+t2 + tm) + (m1 – m2) \* (t1+t2)] } / {m1 \*(t1+tm) } \*100

**Part 5:**

Caches are very small compared to Main Memory. Hence, old data must be replaced when new data needs to be brought into the cache. Many replacement algorithms exist but they perform differently for different types of datasets/programs. No single replacement policy works equally well in every situation. Some of the common replacement policies used are LRU (Least Recently Used), FIFO (First In First Out) and Random. The program that you wrote for **Part2** was fairly data intensive, each array being roughly a million elements long. Let us test the policies on it.

Use the cache configurations from **Part2** but change the Replacement policy field of **<config>** (in **config.cfg** file) to **LRU, FIFO** and **Random (**see figure2). For each policy, run **sim-cache** on the program from **Part2 A** and compare their performance based on Miss Rates. Write your inference (similarities in performance, which policy does better than the other(s) etc.).

**Part 6: (optional)**The virtual address to physical address translation operation is a critical operation that sits in the way of the CPU accessing the cache. If every request made by the processor for a memory location required multiple accesses to Main Memory to read the PTE(page table entries) and additional accesses to fetch the requested data, then that would cost our processor hundreds of cycles! Hence high performance processors include a translation look-aside buffer(TLB).

A TLB is a small cache (associative memory) that holds the PTE ( page table entries) of recently accessed virtual pages. Exploiting locality of reference, a TLB can save a lot of time by substituting PTE lookup.  
Use the configurations from **Part1 A** to configure your Data TLB. Observe how the dtlb Miss rate varies with varying Associativity and #Sets (use tables from Part1A as well).  
  
**Submission**All submissions are through blackboard. You may edit this document with your answers or create a separate document. If you are submitting multiple files, name them correctly and put everything into a single zip file.