CS4342 Advanced Computer Architecture

Take Home Lab 4

Due – July 29 before 11:55 PM

Learning Outcomes

In this lab we will analyse the influence of cache parameters on the performance. At the end of the lab you will be able to:

- Recognize various caching options/techniques used in commercial microprocessors
- Explain the influence of architectural properties of the computer to the execution of programs written in a high-level language

Getting Started

This lab is extracted from Prof. Siniša Šegvić's lab on "Influence of computer architecture to software performance", which is available at http://www.zemris.fer.hr/~ssegvic/ar2/lab3_en.html

Each lab group consists of 2 students (you are free to select your lab buddy, but he/she should not have worked with you in previous labs).

Step 1: Determine the following cache (L1, L2, and L3) parameters of the considered computer: total capacity, line width in bytes, and associativity. On modern x86 processors this can be accomplished by directly executing the CPUID instruction (available in Unix/Linux) or by invoking the tools which employ that instruction (e.g., http://rh-software.com). For other processors, once you find the processor model, this information can be retrieved at the web pages of the manufacturer.

On Linux nodes, such information can be retrieved in virtual directories /proc/ (file cpuinfo) and /sys/devices/system/cpu/ (e.g., file cpu0/cache/index2/size). For example, you may use the following commands:

```
cat /proc/cpuinfo
cat /sys/devices/system/cpu/cpu0/cache/index2/size
```

The other option is to employ programs hwinfo, lshw lscpu, or dmidecode which are able to display more detailed information about the system.

Step 2: Write a computer program to show the memory access performance for data in main memory, L3 cache, L2 cache, and L1 cache. It is recommend to develop this program in C or C++.

Let us consider the following notations:

- s1 – capacity of L1 cache
- b1 – line width (i.e., block size) of L1 cache
- s2, b2 – analogously for cache L2

Your program should measure the average bandwidth of the byte access during many rounds of execution of the following 3 subroutines:

- Subroutine A – Increment all bytes of a memory buffer containing 2*s1 bytes in the order of increasing memory addresses
- Subroutine B – Increment each b1-th byte of a memory buffer containing 2*s1 bytes in the order of increasing memory addresses
- Subroutine C – Increment each b2-th byte of a memory buffer containing 2*s2 bytes in the order of increasing memory addresses

Subroutines B and C employ the memory buffer which is exactly twice as large as the capacity of the analysed cache (e.g., L1 or L2). By using such memory buffers we ensure that the buffer is smaller than the capacity of the memory at the next level of the memory hierarchy (if we are testing L1, the buffer is smaller than L2).

Each of these three subroutines has to be invoked many times in the loop. We see that the subroutine A generates L1 cache misses relatively seldom (once in b1 accesses). Subroutine B generates a L1 cache miss in each memory access, but a majority of these accesses shall fall inside L2. Subroutine C generates a L2 cache miss in each access.

For each subroutine, your program should determine average time of a byte access, as well as the achieved bandwidth in MB/s. Based on the obtained data, estimate the ratios of the latencies corresponding to neighbouring levels of the memory hierarchy (t(L2)/t(L1), t(RAM)/t(L2)).

Instructions:

- Before carrying out the measurements, initialize all bytes of the memory buffers to random values to be sure that the whole buffer is in cache L2 (for subroutine A), and to make the optimizing compiler think that we are actually doing something useful
- Perform the measurements by using the clock function (e.g., <time.h>)
- To make the measurements reliable, repeat the experiment enough times to make the measured time interval comparable to one second
- After the measurement (when the loop is completed) sum all bytes in the buffer and write out the result to make the optimizing compiler think that we really care about the result
- Use full compiler optimization (gcc: -O3)
- Clearly indicate the machine specification (e.g., CPU speed, CPU type, number of cores, total memory, etc.) while submitting the results
- Check whether our assumptions about cache misses and hits are correct. Comment on your observations while considering the cache architecture of the microprocessor

What to Submit

- Source codes and instructions on how to compile and run them
- Submit answers to comments in Step 2. Clearly indicate group members' index numbers and names.
- Name the .zip file as lab4_<*index no 1*>_<*index no 2*>.zip. Replace <*index no x*> with your index number.