Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-739: Advanced Processor Design

Assignment 1

Submission Deadline : On resuming classes

Problem Statement:

The program given below can be used to evaluate behavior of memory system. The key is having accurate timing and then having the program stride through memory to invoke different levels of the hierarchy. The first part of the code that uses a standard utility to get an accurate measure of the user CPU time; this procedure may have to be changed to work on some systems. The second part is a nested loop to read and write memory at different strides and cache sizes. To get accurate cache timing, this code is repeated many times. The third part times the nested loop overhead only so that it can be subtracted from overall measured times to see how long the accurate accesses were. Running the program in single user mode or at least without other active applications will give more consistent results.

The program assumes that program addresses track physical addresses, which true on the few machines that uses the virtually -addressed caches. In general, virtual address tend to follow physical address after rebooting. So you may need to reboot machine in order to get smooth lines in your results. Using the program answer the following:

1. What is the overall size and block size of the second level cache?
2. What is the miss penalty of the second level cache?
3. What is the associativity of the second level cache?

Program:

```c
#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#define ARRAY_MIN (1024) /* 1/4 smallest cache */
#define ARRAY_MAX (4096*4096) /* 1/4 largest cache */
int x[ARRAY_MAX]; /* array going to stride through */

double get_seconds() { /* routine to read time in seconds */
  __time64_t ltime;
  _time64( &ltime );
  return (double) ltime;
}
int label(int i) {/* generate text labels */
  if (i<1e3) printf("%1dB,",i);
  else if (i<1e6) printf("%1dK,",i/1024);
  else if (i<1e9) printf("%1dM,",i/1048576);
  else printf("%1dG,",i/1073741824);
  return 0;
}
int _tmain(int argc, _TCHAR* argv[]) {
int register nextstep, i, index, stride;
int csize;
double steps, tsteps;
double loadtime, lastsec, sec0, sec1, sec; /* timing variables */

/* Initialize output */
printf(" ,");
for (stride=1; stride <= ARRAY_MAX/2; stride=stride*2)
  label(stride*sizeof(int));
printf("\n");

/* Main loop for each configuration */
for (csize=ARRAY_MIN; csize <= ARRAY_MAX; csize=csize*2) {
  label(csize*sizeof(int)); /* print cache size this loop */
  for (stride=1; stride <= csize/2; stride=stride*2) {

    /* Lay out path of memory references in array */
    for (index=0; index < csize; index=index+stride)
      x[index] = index + stride; /* pointer to next */
    x[index-stride] = 0; /* loop back to beginning */

    /* Wait for timer to roll over */
    lastsec = get_seconds();
    do sec0 = get_seconds(); while (sec0 == lastsec);



    /* Walk through path in array for twenty seconds */
    /* This gives 5% accuracy with second resolution */
```

```
    steps = 0.0; /* number of steps taken */
    nextstep = 0; /* start at beginning of path */
    sec0 = get_seconds(); /* start timer */
    do { /* repeat until collect 20 seconds */
      for (i=stride;i!=0;i=i-1) { /* keep samples same */
        nextstep = 0;
        do nextstep = x[nextstep]; /* dependency */
        while (nextstep != 0);
      }
      steps = steps + 1.0; /* count loop iterations */
      sec1 = get_seconds(); /* end timer */
    } while ((sec1 - sec0) < 20.0); /* collect 20 seconds */
    sec = sec1 - sec0;

    /* Repeat empty loop to loop subtract overhead */
    tsteps = 0.0; /* used to match no. while iterations */
    sec0 = get_seconds(); /* start timer */
    do { /* repeat until same no. iterations as above */
      for (i=stride;i!=0;i=i-1) { /* keep samples same */
        index = 0;
        do index = index + stride;
        while (index < csize);
      }
      tsteps = tsteps + 1.0;
      sec1 = get_seconds(); /* - overhead */
    } while (tsteps<steps); /* until = no. iterations */
    sec = sec - (sec1 - sec0);
    loadtime = (sec*1e9)/(steps*csize);
    /* write out results in .csv format for Excel */
    printf("%4.1f,", (loadtime<0.1) ? 0.1 : loadtime);
    }; /* end of inner for loop */
    printf("\n");
  }; /* end of outer for loop */
  return 0;
}
```