Computer architecture

Homework week 6

Instructions

Submit by e-mail to the lecturer, as a PDF document with your name and student ID near the beginning. You can work in groups of 2; however, use different groups than week 3, 4 and 5. Use the English language. Deadline: Oct 14th, 23:59.

Overview

On most computing systems nowadays the memory hierarchy is organized as a layering of caches: L1 next to the processor, then L2, then optionally L3, then memory.

However due to chip defects, variation in manufacturing, or outright manufacturer fraud, the *actual* characteristics of the caches may not match the *advertised* characteristics. For example a cache avertised to contain a capacity of 2MiB may only have 1.5MiB available due to a bank defect.

Here comes science! The purpose of this assignment is to devise an empirical experiment that would estimate practically the usable L1 and L2 cache sizes in your work computer.

Question 1 (2pt)

Consider in the following C function, which computes for each value in its output `B` the average of the "surrounding" points in `A`:

```
void convo(int A[N][M], int B[N][M])
{
  for (i = 1; i < N-1; ++i)
   for (j = 1; j < M-1; ++j)
    B[i][j] = (A[i][j]*4 + A[i-1][j] +
       A[i+1][j] + A[i][j-1] + A[i][j+1])/8
}
```

NB: `N` and `M` can be defined statically on the compilation command line using eg. `-DN=10 -DM=10`.

1) how many memory accesses are performed by each inner iteration? Assume all temporary/intermediate results fit into registers.

2) assuming a cache line length of 64 bytes and 32-bit `int`, how many cache lines are accessed by each inner iteration? You can assume that M >= 16.

Question 3 (5pt)

Consider the following test program:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

/* .. (place the source of convo here) .. */

int A[N][M] = { 1 }, B[N][M] = { 1 };
int main(int argc, char** argv)
{
   int L, i;
   struct timeval tv1, tv2;

   L = atoi(argv[1]);
   gettimeofday(&tv1, 0);
   for (i = 0; i < L; ++i)
   {
      convo(A, B);
      convo(B, A);
   }
   gettimeofday(&tv2, 0);

   printf("T = %f seconds\n",
          (float)((tv2.tv_usec - tv1.tv_usec)*1e-6
                + (tv2.tv_sec - tv1.tv_sec)));

   return 0;
}
```

Note
`N` and `M` can be defined statically on the compilation command line using eg. `-DN=10 -DM=10`.

1) Determine the number of memory loads (#R) and stores (#W) performed by this program, as functions of `N`, `M` and `L`.

2) Select one real computer of your own choosing. Report its *advertised* cache sizes.

3) Using this computer, complete the following table experimentally:

N	**M**	**L so that 10s < T < 50s**	**#R**	**#W**	**T(min at L cst)**	**#R/T**	**#W/T**
30	30						
100	100						
1000	1000						
10000	10000						

Note

To complete each row this table, proceed as follows:

a. compile your program for the given values of `N` and `M`.

b. run your program with incrementing values of `L` until the reported value of `T` is larger than 10s (if possible lower than 50s).

c. Fill in this value of `L` in the table.

d. Using the same values of `N`, `M` and `L`, run the program 3-5 times and determine the lowest value of `T`. Fill this in the table.

e. compute the other columns.

Hint

Clever use of a scripting language is recommended.

Note

Avoid using a virtual machine (eg Linux VM on OS X), because this will make your measurements unclear.

3) Are all rates of #R/T and #W/T equal? Why?

Question 4 (3pt)

1) Using a binary search (keep N = M), determine the problem sizes that cause significant variations in #R/T and #W/T.

   It is recommended (but not required) that you generate a graph to document your search.

2) Conclude.