CS231, DLDCA Lab, Lab 10

Goals

1. Understanding the performance effects of the memory hierarchy
2. Writing a program to guess the machine's cache size

Instructions

1. These exercises are to be done individually.
2. While you are encouraged to discuss with your colleagues, do not cross the fine line between discussion *to understand* versus discussion as a *short-cut* to complete your lab without really understanding.
3. Create a directory called <rollno>-<labno>. Store all relevant files to this lab in that directory.
    a. In the exercises, you will be asked various questions. Note down the answers to these in a file called “answers.txt”.
    b. In some parts of the exercises, you will have to show a demo to a TA; these are marked as such. The evaluation for each lab will be in the subsequent lab, or during a time-slot agreed upon with the TAs. For this evaluation, you need to upload your code as well.
4. Before leaving the lab, ensure the following:
    a. You have marked attendance on SAFE
    b. You have uploaded your submission on BodhiTree, and downloaded and checked if the submissions is right
5. Things to ensure during TA evaluation of a particular lab submission:
    a. The TA has looked at your text file with the answers to various questions
    b. The TA has given you marks out of 10, and has entered it in the marks sheet
6. You have to use the MIPS conventions, unless mentioned otherwise.
7. **House points:** Some questions carry house points. For these, you are allowed to work along with others, but only if *all* of those working together have completed the lab. (You can work on it individually, even before finishing the lab completely, but only if you care about house points more than your own marks!). All questions related to house points have to be shown to the instructor (Bhaskar).

Effect of memory hierarchy on the performance of matrix multiplication

- Read the 'pitfall' mentioned in page 551 of the textbook (3rd edition). This talks about the effect of the memory hierarchy on something similar to a matrix multiplication algorithm.

- In this exercise, you have to first implement a regular 3-nested-loop matrix multiplication algorithm in C-language. Write a file called **mat-mult.cc**. The program should declare 3 static 2-D arrays of type double. Call these arrays A, B, and C. Use a #define for the SIZE of the 2D square arrays.

- A is initialized to the zero matrix, B to 2I (twice the identity matrix), and C to a matrix whose first row is [0, 1, 2, ..., SIZE-1] and all other rows are 0.

- Within the 'main' of the program, write the 3-nested-loop matrix multiplication algorithm. The heart of the loop will be something like:

  A[i][j] = A[i][j] + B[i][k]*C[k][j]

- In this loop, A and B are accessed row-wise, while C is accessed column-wise. Now, C-language allocates memory in row-major order (i.e. one row after another). So column-wise array access has little locality of reference, while row-wise access has good locality of reference.

- Compile the program and run it, for SIZE=500. Estimate the time taken to run it, using the 'time' command. Run it 5 times and take the median value as the run-time.

- You are now going to make the program more efficient by first computing another matrix D, which is the transpose of the C matrix, to take advantage of locality of array access. Note that you will still be computing A=B*C, but will use the fact that transpose(transpose(D)) = C..

- **Demo to TA [3 marks]:** Make the modification as suggested above, and save it in a file called **mat-mult-opt.cc**. Compile and run the modified program **mat-mult-opt.cc**. Compute the median run time of 5 runs, as earlier. Compare the run-time with the earlier run-time. Do you get a 'feel' for the effect of caches on your program's performance?

Confirming the effect of the memory hierarchy, using dineroIV

- In this exercise, you are going to produce (an approximate) dineroIV trace file of the execution of **mat-mult.cc** and **mat-mult-opt.cc**. We are going to ignore all instruction references (i.e. only deal with a D-cache), and also ignore all data references other than the 4 arrays: A, B, C, and D (i.e. ignore i, j, k).

- Modify both programs (**mat-mult.cc** and **mat-mult-opt.cc**) to make them print the addresses of the data they refer during their execution. Print in a format which can be understood by dineroIV. Take care to print both reads as well as writes. (You can assume the specific manner of C-language expression evaluation; it is quite obvious).

- ***IMPORTANT NOTE 1:*** *Do NOT print the output to a file, it would be too huge. You have to redirect the output of your program to dineroIV directly using a UNIX pipe.*

- ***IMPORTANT NOTE 2:*** *Change the SIZE of the array to 100x100 (from 500x500 earlier), else your simulation will run for very long.*

- ***Hint:*** *You may want to print the index of the outer most loop 'i' for each iteration. This is so that you know that your program is making progress. Note that you have to print to STDERR so that the input to dineroIV is not affected.*

- **Demo to TA [3 marks]:** Run the modified programs. Get the output file and feed it as input to dineroIV with the following configuration: 16-word blocks, 128KB data cache size, direct-mapped, write-back with write-allocate. Are you able to bolster your observations regarding program performance from the previous exercise?

Writing a program to guess the size of the cache

- In this exercise, you will write a clever program which will try to guess the size of the machine's cache. The idea is as follows. Declare a large array and access its elements. Compute the average access time. Once the size of the array exceeds the cache size, you will see a significant jump in the average access time (due to the miss penalty). With this, you can guess the cache size.

- ***Hints:*** *In my solution, I used an int array. I had to arrange to access elements such that there was no spatial locality. This I did by accessing elements 97 words away from one another (and using %guess_size to arrange wrap-around). I multiplied guess_size by 2 each time. I also had to adjust the averaging loop somewhere between 100 and 10000, depending on the guess_size. With this arrangement, I was able to guess the L2 cache size. Other similar solutions may work for you.*

- You may have to use the function gettimeofday() to do time measurements within your program.

- ***WARNING 1:*** *Make sure to print anything* ***after*** *you have done all time computations necessary. This is because printing itself takes a very long time (several milli-seconds) and will disrupt any time measurements you may make within your program if you print anything inbetween.*

- ***WARNING 2:*** *You need to pay careful attention to the types of various variables: int, long, float, double, etc. This could be a source of bug in this program.*

- ***WARNING 3:*** *This is a tough problem. I had a tough time working out the hints/warnings I have given above. Hopefully with the hints it would be easier for you. But in any case, try this problem last.*

- **Demo to TA [4 marks]:** Implement a C program using the above idea, called **guess-cache-size.cc**. Plot the above-mentioned graph, and explain how you have guessed the cache size.

Writing a program to guess the size of the cache

- **20 house points:** The book shows some performance studies comparing the performance of radix sort with that of quick-sort. Implement both sorting algorithms and generate graphs similar to that in the textbook. Use the dineroIV cache simulator as necessary, in a manner similar to how we have used it above. You will get upto 20 house points, depending on the level of clarity in what you show.

- **20 house points:** If you are able to guess the L1 cache size using a program.