CS301-HPC Lab 1 (January 20, 2020)

Deadline: 27th Jan

CPU architecture, Vector Triad, Measuring Performance and Code Optimization Techniques

Items to be included in the report and submitted through Moodle : B, D, E

A) Understanding your CPU architecture using `lscpu`

The goal of this exercise is to explore and understand the hardware architecture in a given Linux Computer.

1. Open the Terminal
2. Type the command **`lscpu`**
3. Several important parameters will be listed.

The following are some important parameters to take note of:

- Architecture
- Byte Order
- CPU(s)
- Model Name
- L1, L2, L3 cache

B) Vector triad benchmarking

As seen in the previous exercise, the CPU memory is divided into 4 parts:

1. L1 cache
2. L2 cache
3. L3 cache
4. RAM Memory

These are listed in the order of the bandwidth/ latency at which they operate.

The goal of this exercise is to explore the impact of the memory hierarchy on the computation speed of a CPU. For this, we will use the Vector Triad benchmarking.

This benchmark consists of performing a vector sum between 3 vectors and storing it in another vector. Consider 4 vectors **A**, **B**, **C**, **D** of size **N** each.

Pseudocode for Vector triad is as follows:

```
for i=1 to N
A[i] = B[i] + C[i]*D[i]
```

For benchmarking purposes we will measure the time taken by the vector triad for different problem sizes **N**.

```
minSize = 2^8
maxSize = 2^29
total = maxSize
for size =minSize; size<maxSize; size*=2
      RUNS = total/size //initialise arrays
      double A[size], B[size], C[size],
            D[size]
      start_time = clock()
      for j=1 to RUNS
            for i=1 to size
                  A[i] = B[i] + C[i]*D[i]
      end_time = clock() - start_time
      throughput = (sizeof(double)*2*total)/end_time
      print size, throughput
```

Using python/matlab/gnuplot make the necessary plots and investigate the following:

- Make the necessary modifications in the code to measure the run-time accurately. What are the couple of clock functions that can be used to measure the elapsed time? Understand how the clock works and how it is used to measure runtime.
- Make modifications in the code to approximately measure the time taken by the computation part.
- Make modifications in the code to approximately measure the time taken by the data access part.
- How does your measured performance compare with the peak-theoretical performance of the computing system?
- Can you show which operation is more expensive, addition or division by making some modifications in the code? Provide explanations for your result.
- Is it possible to fix the clock speed to 2.0 GHz? If yes, can you interpret the observable changes to the results in throughput?

C) Profiling of the code (gprof):

`gprof` calculates the amount of time spent in each routine. Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle.
Necessary for gprof

- Have profiling enabled while compiling the code
- Execute the program code to produce the profiling data
- Run the gprof tool on the profiling data file (generated in the step above).

Step:1 Compile your code normally using -pg flag

Step:2 Run the code

This will create gmon.out file Step:3 $gprof

./a.out gmon.out

as a test code, you can use any code .

D) Which optimization strategies would you suggest for the piece of pseudo-code below?

```
double mat[N][N], s[N][N], val;
int i, j, v[N];
//
// ... v[] and s[][] --□ assumed to contain valid data
//
for(i=0; i<N ; ++i) {
  for(j=0; j<N; ++j) {
    val = (double)(v[i] % 256);
    mat[j][i] = s[j][i]*(sin(val)*sin(val)-cos(val)*cos(val));
  }
}
```

Do not make any major assumptions about the size of N.

- Does the optimization strategy depend on the problem size?
- Can you predict an upper performance limit on the given lab system?

E) High Performance Matrix Multiplication on a CPU

Goal is to optimize matrix multiplication to run on a single core and understand how fast we can perform important linear algebra kernels/ routines.

We will start with matrix multiplication which is a level-3 BLAS subroutine (https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms)

Matrix-matrix multiplication computes C = C + A*B, where C is m-by-n, A is m-by-k and B is k-by-n.

Assume we have two levels of the memory hierarchy, fast (cache) and slow (RAM), and that all data initially resides in slow memory.

we can count

m = number of memory references to slow memory needed just to read

the input data from slow memory, and write the output data back

c = number of floating point operations

CMA = c/m = average number of flops per slow memory reference

The significance of CMA : for each word/ byte read from slow memory (the expensive operation), one can hope to do at most CMA ratio operations on it (on average) while it resides in fast memory. The higher the CMA is, the more the algorithm will operate at the top of the memory hierarchy, where it is most efficient.

Some important points/ assumptions to be taken into account as discussed during the lecture, particularly during Block Matrix Multiplication lecture:

1. There are just two levels in the hierarchy, fast and slow.
2. The small, fast memory has size “c” bytes, where c << n^2, so we can only fit a small part of an entire n-by-n matrix, but c >= 4*n, so we may fit several whole rows or columns depending on the size of the matrix.
3. In the code - Each word/ element is read from slow memory individually (as discussed, in practice, larger groups of words/ elements are read, such as cache lines or memory pages).

In this assignment, we will consider 2 implementations of matrix multiply and compute CMA for each and also do memory/ computational complexity analysis as performed during the last lecture.

Algorithm 1: Simple matrix multiply (Unblocked) C=A*B

Look into the pattern - in this case, the innermost loop is doing a dot product of row i of A and column j of B.

What is the CMA ratio in this case?

for a standard code and for matrix size of nxn

CMA=(2*n^3)/(n^3 + 3*n^2) ~ around two (2)

m = # slow memory access = n^3 read each column of B n times

+ n^2 read each row of A once for each i,

and keep it in fast memory during

the execution of the two inner loops

+ 2*n^2 read/write each entry of C once

= n^3 + 3*n^2

Algorithm 2: Square blocked matrix multiply (also called 2D blocked MM)

Now consider C to be an N-by-N matrix of n/N-by-n/N subblocks Cij, with A and B similarly partitioned. As discussed during the lecture.

Show that : CMA in this case is n/N.

Compare the computational throughput of the above two matrix multiplication algorithms using several computational experiments with different problem sizes (changing the size of the matrix, taking into account the Cache sizes and RAM size) and support your observations quantitatively.

Assignment Updated (3rd February, 2020)

To multiply two matrices, we use 3 nested loops: Perform the following numerical experiments, report your observations and support it with **<u>quantitative justifications</u>** in terms of cache performance, data access pattern, reuse of cached data, cache blocking, temporal/ spatial locality of memory accesses etc.

1. Try all 6 different loop orderings. Which ordering perform best for (a) 512 by 512 and (b) 2048-by-2048 matrices? Which ordering is the worst? Support your observations in terms of striding through the matrices with respect to the innermost loop?
2. Why does performance drop for large values of matrices (start from 256 by 256 and try till 2048 by 2048)? How much is the drop in terms of compute throughput?
3. What is the effect on performance if elements of the matrices are changed from integer to double precision? Perform a quantitative analysis for the case 1024 by 1024 and 2048 by 2048.

F) Perform a basic matrix transposition.

Now implement a single level of cache blocking for the same. i.e. Loop over all matrix blocks and transpose each into the destination matrix.

Try block sizes of 2-by-2, 32-by-32, 128-by-128, 512-by-512. Which performs best on the 2048-by-2048 matrix? Which performs worst?

All major queries in the lab to be addressed via the following discussion board (requires login for viewing/ posting queries/ answers):

http://letshpc.herokuapp.com/discussionboard