Computer Architecture Homework #1: Addressing Arrays

June 22, 2020

Abstract

One common task for computers is multiplication of matrices; it also serves as an excellent example for learning how a computer actually executes programs. This is the first in a series of homeworks that should make that clear.

Today's homework consists of three parts: a little systems work, a little math, and a little architecture work.

The systems work to be done is just in preparation for the homework to be done later in the semester.

1. Install the `spim` simulator for the MIPS microprocessor. There are several versions available; you may install any version that works on your PC. *We will be using this in class next time! You must have it installed and running by then!*

---

1 Math

Okay, here's the math part:

This is the first of a string of three homeworks. There will be follow-on homework from this next week, so you must have this done by then!

Take the two matrices:

$$A = \begin{pmatrix} 1 & 0 & 3.14 & 2.72 \\ 2.72 & 1 & 0 & 3.14 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \tag{1}$$

$$B = \begin{pmatrix} 1 & 1 & 0 & 3.14 \\ 0 & 1 & 3.14 & 2.72 \\ 0 & 1 & 1 & 0 \\ 4 & 3 & 2 & 1 \end{pmatrix} \tag{2}$$

Do the following:

1. Find the matrix product AB. Do this by hand, and show your work.
2. Count
   (a) the number of real (floating point) multiplications necessary, and
   (b) the number of real (floating point) additions necessary.
3. Express
   (a) the number of real (floating point) multiplications necessary, and
   (b) the number of real (floating point) additions necessary

   as a function of N for multiplying two $N \times N$ matrices.
4. Write *pseudocode* for a program to multiply two $N \times N$ matrices.

2 Architecture

Okay, the architecture work: understanding just a little about how memory is laid out and how addresses are calculated. Assume all of the values in these arrays are 64-bit floating point values (doubles).

1. First, for a simple vector, by hand: If the *base address* of an array `double vector[100]` is $0x40000$,
   (a) How many bytes of memory does this array consume?
   (b) What is the address of `vector[0]`?
   (c) What is the address of `vector[1]`?
   (d) What is the address of `vector[10]`?
   (e) What is the address of `vector[99]`?
2. Now, begin with something like this:

```
main()
{
  double vector[100];
}
```

   Extend that program to print the answers to the same set of questions. Include your code and your output.

3. Next, for a 2-D array, by hand: If the *base address* of an array `double array[1024][1024]` is $0x40000$,
   (a) How many bytes of memory does this array consume?
   (b) What is the address of `array[0][0]`?
   (c) What is the address of `array[0][1]`?
   (d) What is the address of `array[1][0]`?
   (e) What is the address of `array[1][1]`?
   (f) What is the address of `array[7][7]`?
   (g) What is the address of `array[1023][1023]`?
4. Actually, although C is excellent at handling 1-D arrays of data, 2-D and 3-D are a little tricky, and there are numerous ways to do them. The closest thing there really is to "native" 2-D arrays requires that they be *statically declared*, so we'll work with that. Now, begin with something like this:

```
main()
{
  double array[1024][1024];
}
```

Assume that the layout of this array in memory is *row major*. Extend that program to print the answers to the same set of questions. Include your code and your output.