Problem set 1

TMA4280 Introduction to Supercomputing

Sigve Sebastian Farstad

January 10, 2014

1 Exercises

Exercise 1. Consider the maximum and minimum numbers derived in (6)-(7). How many digits should we include in each of these numbers?

Solution:
Roughly 7.22 digits, or "about 7 digits", as the course material puts it.

Exercise 2. Find the binary floating point representation of the decimal number 4.25 in single precision.

Solution:

Sign	Exponent								Mantissa																			
0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Exercise 3. How many digits of accuracy does a floating point number in double precision have?

Solution:
The mantissa in the IEEE754 double precision float is 52 bits wide. This means that the significand precision is 53 bits, which is gives roughly 15.95 digits of accuracy.

Exercise 4. Propose a way to avoid the above limitation.

Solution:
This exercise is about the limitation of traditional for-loops that can only run `MAX_INT` number of times. If program requires a for-loop to run longer than an int counter can count, several techniques can be used to work around the issue:

- The for-loop can use a wider fixed-size integer representation, or even an integer of arbitrary width.
- The for-loop can be partially unrolled, if the counter variable is only a couple of bits too short.
- The programmer can use nested for-loops to achieve a larger maximum loop count.

Exercise 5. Let c be a scalar (a floating point number), let $\vec{x}$, $\vec{y}$, and $\vec{z}$ be vectors, each comprising n floating point numbers, and let $\mathbf{A}$ be an $n \times n$ matrix. How many floating point operations does it take to perform the following basic linear algebra operations: $\vec{z} = \vec{x} + c\vec{y}$ (1)? What about the matrix-vector product $\mathbf{y} = \mathbf{Ax}$ (2)?

Solution:
Computing (1) requires n operations for calculating $c\vec{y}$, and then n further operations for adding the result to $\vec{x}$. This means that computing (1) takes a total of $2n$ floating-point operations.

Computing (2) requires $n \times n$ dot products of vectors of size n. A single dot product of two vectors of size n requres n multiplications and $n-1$ additions. This means that the computation of (2) requires $(n \times n)(n + n - 1) = 2n^3 - n^2$ floating point operations.

Exercise 6. Let $\mathbf{A}$ be an $n \times n$ matrix, and $\vec{x}$ and $\vec{b}$ be two vectors of length n. Assume that we want to solve the linear system of equations

$$\mathbf{A}\vec{x} = \vec{b} \tag{3}$$

using Gaussian elimination. Assume further that the matrix $\mathbf{A}$ is dense, meaning that we need to store all the n^2 entries in the matrix. What is (approximately) the largest equation system we can solve (i.e., the largest number of n we can use) and still be able to fit the whole problem in the main memory, which we assume is 1 Gbyte?

Solution:
Assuming 32-bit single-precision floating point numbers, the largest n is given by

$$n_{largest} = \lfloor \sqrt{1Gbyte/32bits} \rfloor = 15811$$

meaning that the largest equation system we can solve is one with no more than approximately 15800 equations with the approximately 15800 unknowns.

2 Code

Attached is the source code for a program written in C which calculates

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

where

$$\mathbf{A} = \begin{bmatrix} 0.3 & 0.4 & 0.3 \\ 0.7 & 0.1 & 0.2 \\ 0.5 & 0.5 & 0.0 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}.$$

The code is organized into a light-weight matrix library, and a short main function that uses the library to calculate the correct answer.