#4 Assignment - Implementing Scientific Computations - CMPT 383

Luiz Fernando Peres de Oliveira - 301288301 - lperesde@sfu.ca

I. Introduction

Numerical methods for scientific problems, especially in engineering and science, are frequently related to solving problems for large matrices (such as image processing). Many of the most efficient algorithms for large-scale matrix computations are based on approximations of the given matrix by small matrices (e.g. kernel convolution and Petrov-Galerkin projections).

Traversing matrices is one of the heaviest numerical computations as, for a given matrix **m** with *r* rows and *c* columns, it has known time complexity of $O(r \times c)$. Imagine that we have a bitmap image **img** of size 800×600 with *RGB* colors, for example. Now imagine that we want to convert **img** into its grayscale representation. In that case, it would take a time of $800 \times 600 \times 3$ (we multiply by 3 because we must consider the space occupied by red, $green$ and $blue$ values of each pixel) to traverse and convert every RGB pixel of **img** into its grayscale representation.

The work below will discuss and illustrate how efficiently different paradigms use matrices to solve a given problem. The paradigms used will be *(Functional Programming (Haskell), Object-oriented Programming (Java), Scripting Language (Python), Imperative Programming (Go) and Procedural Programming(C))*. For each paradigm here discussed, the programs are expected to vary in time complexity to traverse and apply map functions to matrices for large inputs as well as vary in resources (memory) allocation, as some paradigms might take more memory than others.

Furthemore, it will be implemented, evaluated and discussed the map functions **add**, **multiply** and **custom** for each paradigm/language described above. Each one of these map functions take a *matrix* **m** as input and return a *matrix* **m'** as output. They can be represented mathematically as:

- **add**: $c_{ij} = a_{ij} + b_{ij}$
- **multiply**: $c_{ij} = a_{ik} \times b_{kj}$
- **custom**: $c_{ij} = max(a_{ij}, b_{ij}) + a_{ij}^2 + b_{ij}^2$

Finally, the map functions described above will be compared against a data d starting from a matrix m of size 10×10 and will then be increased of an order of magnitude 100×100 until we notice that the target code takes longer to execute.

II. Related Work, Background and Applications

Representation Theory is the field of study that uses matrices representations for problem solving [19]. All finite groups can be represented as matrices, and, therefore, matrices are a very useful tool for studying finite groups.

There are numerous applications in which matrices are a useful for scientific programming. For instance, matrices are largely used in the field of linear systems as a mean to represent and manipulate finite dimensional vector spaces. Matrices are are also very important in the field of linear algebra which is one of the most crucial tools in math. Some real-world applications for matrices are:

- **Computer Graphics and Image Processing**: Your computer graphics are represented by matrices. The major task of the *GPU* is to calculate more than a billion matrix operations per second. Also, reflection and distortion effects applied on images such as the Gaussian and Sobel operators are represented by matrices as well as the images themselves.
- **Linear systems arithmetic**: Linear systems take advantage of matrix arithmetic. For instance, error-correcting codes and linear differential equations.
- **Supports Graph Theory**: Graphs may be abstracted as matrices.
- **Probability and Statistics**: The field of probability and statistics use matrix representations such as probability vectors of a neural system and Markov Chains on stochastic matrices.
- **2D and 3D Geometry**: Matrices

That is to say, scientific programming is related to many fields and therefore it is not possible to describe completely its current *state of the art*. Current work on scientific programming using matrices focus in Computer Graphics, Geometry and problem solving in general. For more details on current work on scientific programming with matrices, please check [21], [22], [23] and [24]. Note that the given references are only examples and therefore, they do not represent all works on the large field of scientific programming).

III. Implementation

Object-oriented Programming - Java

The way Object-oriented programming paradigm behaves regarding scientific programming can be related to the Object-oriented language we are using, because such paradigm is often implemented in multi-paradigm languages. For example, *Java, Erlang, Objective-C and Pony* will most likely have different ways and use different methods for processing numerical computations. That happens because these languages are not only Object-oriented, but also

imperative (for Java and Objective-C) and Process-Calculus-based(Erlang and Pony), a paradigm very close to Functional Programming, where "everything" is a process rather than a function. That is to say, Java will be representative language for this category, but what is true to Java might **not** be true to other different *OOP languages*.

Java is one of the *C-family* languages[11] and for this reason, scientific programming in pure Java can be **very** similar to scientific programming in C (not accounting direct access to pointers).

Java has the benefit of having multiple frameworks for numerical computations, such as *COLT, JLAPACK, ND4J, Matrix toolkit Java* and others. However, I chose to write numerical computations in pure Java for this assignment, as I would like to compare it with C (because the code is very similar, but there might be differences in runtime because of the different compilers for these languages and because Java's bytecode will be interpreted by the JVM).
On the code below, you will see how one would represent the map functions **add**, **multiply** and **custom** in Java.

```java
public class Matrix {
    private int[][] a;
    private int[][] b;
    private int[][] c;

    public int[][] getA() {
        return this.a;
    }

    public void setA(int[][] a) {
        this.a = a;
    }

    public int[][] getB() {
        return this.a;
    }

    public void setB(int[][] b) {
        this.b = b;
    }

    public int[][] getC() {
        return this.a;
    }

    public void setC(int[][] c) {
        this.c = c;
    }

    public int[][] add(int[][] a, int[][] b) {
        int[][] c = new int[a.length][a[0].length];
        for (int i = 0; i < a.length; i++)
            for (int j = 0; j < a[0].length; j++)
                c[i][j] = a[i][j] + b[i][j];
        return c;
    }

    public int[][] multiply(int[][] a, int[][] b) {
        int[][] c = new int[a.length][a[0].length];
        for(int i = 0; i < a.length; i++)
            for (int j = 0; j < b[0].length; j++)
                for (int k = 0; k < a[0].length; k++)
                    c[i][j] += a[i][k] * b[k][j];
        return c;
    }

    public int[][] custom(int[][] a, int[][] b) {
        int[][] c = new int[a.length][a[0].length];
        for(int i = 0; i < a.length; i++)
            for (int j = 0; j < b[0].length; j++)
                c[i][j] = Math.max(a[i][j], b[i][j]) +
                          Math.pow(a[i][j], 2)       +
                          Math.pow(b[i][j], 2)
        return c;
    }
}
```

The previous Java code will traverse matrices as fast as C would do and in fact (with limitations on the JVM), it is very hard to distinguish the differences between that code and a given C code for the same task. You can of course see that the Java code is very verbose regarding such simple functions. In order to execute the code, we will need to have a *Java class* that implements the *main* function. In this way, our class will be instantiated into an object of type **Matrix** and only them, after setting it up properly, we would be able to use all the power of this language. Also, this Java code allows us to use custom sizes for our data d described on the introduction of this work.

Scripting Language - Python As said before on previous assignments, scripting languages usually run slower than compiled languages (because of the time that is spent on just-in-time translations). Python has a slight advantage on other scripting languages because it uses many libraries written in pure C, boosting the process of interpreting programs.

Python is multi-paradigm and allows **imperative programming** as well, just like *Java and C* and is reasonable to think that it will be one of best languages for this assignment. One point is that it is well known as a scientific programming language with numerous libraries out there, such as *NumPy* (which is mostly written in C). Lastly, Python allows "hacks" for performance when dealing with matrix handling. Although, for this assignment, it will be used standard Python with no libraries as requested.

Compared to other languages, Python is the one of the ones with clearest code and helps us understand why the scientific community enforce the use of this tool. The code below represents how one would implement the map functions **add**, **multiply** and **custom** in Python.

```python
import sys

def add(a, b):
  for i in range(len(a)):
    for j in range(len(a[0])):
      c[i][j] = a[i][j] + b[i][j]
  return c

def multiply(a, b):
  for i in range(len(a)):
    for j in range(len(b[0])):
      for k in range(len(a[0])):
        c[i][j] += a[i][k] * b[k][j];
  return c

def custom(a, b):
  for i in range(len(a)):
    for j in range(len(a[0])):
      c[i][j] = (max(a[i][j], b[i][j]) +
                 pow(a[i][j], 2)       +
                 pow(b[i][j], 2))
  return c
```

Python is one of the most beautiful languages out there, in my opinion. Compared to the Java code, its code is about *50%* smaller and is much simpler to read. However, it has some problems as well. In pure Python, without *NumPy*, Python was extremely slow. We shall see more on that on our Results.

Procedural Programming - C

Procedural programming, as its name suggests, bases itself in procedure, functions or sub-routines calls (not to be confused with mathematical functions related to functional programming). It also follows the imperative paradigm as it makes explicit reference to changing-state, performs a series of well-structured sequential steps and may have side effects. Systematically, it embraces a succession of statements and functions to perform a computational task or program. Pascal, C and Go are example of procedural languages. For this assignment, however, *C* was picked because I believe that it will be one of the fastest languages in this assignment (if not the fastest). C has complete power over the machine and its imperative/procedural structures allow programmers to design code as well as one would do in pure Assembly.

On the code below, you will see how one would represent the map functions **add**, **multiply** and **custom** in C.

```c
#include <math.h>

void add(int** a, int** b, int** c, int s)
{
  for (int i = 0; i < s; i++)
    for (int j = 0; j < s; j++)
      c[i][j] = a[i][j] + b[i][j];
}

void multiply(int** a, int** b, int** c, int s)
{
  for(int i = 0; i < s; i++)
    for (int j = 0; j < s; j++)
      for (int k = 0; k < s; k++)
        c[i][j] += a[i][k] * b[k][j];
}

void custom(int** a, int** b, int** c, int s)
{
  for(int i = 0; i < s; i++)
    for (int j = 0; j < s; j++)
      c[i][j] = max(a[i][j], b[i][j]) +
                (int)pow(a[i][j], 2)  +
                (int)pow(b[i][j], 2);
}
```

Functional Programming - Haskell

Functional programming in general appears to have many beautiful ways of traversing matrices, given (in theory) the lack of side effects and its foundation of applying mathematical functions to the real world, however, in practice, this may not work as beautiful as it seems because a bigger problem may be divided into many small problems (function calls, as high order functions are always broken down in other smaller functions), which may impact on the speed of a given program, once that it would be faster if the process was done linearly[5] (the way that imperative languages would try to solve this category of problems). Haskell will be the representative of the Functional Programming paradigm for this assignment.

On the code below, you will see how one would represent the map functions **add**, **multiply** and **custom** in Haskell.

```haskell
import Data.List

add :: (Num a) =>
  [[a]] -> [[a]] -> [[a]]
add = zipWith(zipWith (+))

multiplication :: (Num a) =>
  [[a]] -> [[a]] -> [[a]]
multiplication a b =
 [[sum $ zipWith (*) ar bc |
  bc <- (transpose b)] |
  ar <- a]

custom :: (Num a) =>
  [[a]] -> [[a]] -> [[a]]
custom a b = zipWith (+)
             (max a b)
             (zipWith (+)
                      (a**2)
                      (b**2))
```

Imperative Programming - Go Differently from the Functional Programming, Imperative programming is a programming paradigm that uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform.

Go is a multiparadigm language and it was chosen as the Imperative Programming representative because of its Imperative/Procedural/Object-oriented nature. In this way, it will be nice to compare against another OO language (Java) and another Procedural language (C). Go syntax includes changes from C aimed at keeping code concise and readable.

Because Go is Imperative, changing states and side effects are strongly encouraged in order to run it faster.

Being able of optimizing and giving the change to programmers to change the behaviour of an algorithm with extern data (side effects) can be good and bad. It might be a positive point regarding its flexibility, but on the other side, it also makes systems more complex, being more open to human errors.

The code below represents the map functions **add**, **multiply** and **custom** in Go.

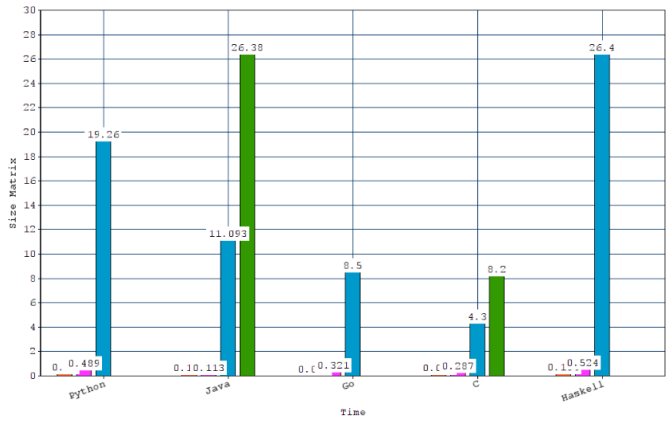
```go
func add (a *matrix) (b *matrix) (c *matrix) {
  for i, m3x := 0, 0; i < len(a.ele); i += a.stride {
    for m2r0 := 0; m2r0 < b.stride; m2r0++ {
      c.ele[m3x] = a.ele[m3x] + b.ele[m3x]
      m1x++
    }
  }
  return c
}

func multiply(a *matrix) (b *matrix) (c *matrix) {
  for i, m3x := 0, 0; i < len(a.ele); i += a.stride {
    for m2r0 := 0; m2r0 < b.stride; m2r0++ {
      for m1x, m2x := m1c0, m2r0; m2x < len(b.ele); m2x += b.stride {
          c.ele[m3x] += a.ele[m1x] * b.ele[m2x]
          m1x++
      }
      m3x++
    }
  }
  return c
}

func custom(a *matrix) (b *matrix) (c *matrix) {
  for i, m3x := 0, 0; i < len(a.ele); i += a.stride {
    for m2r0 := 0; m2r0 < b.stride; m2r0++ {
      c.ele[m3x] = (max(a.ele[m3x],b.ele[m3x]) +
                    pow(a.ele[m3x], 2) +
                    pow(b.ele[m3x],2))
      m1x++
    }
  }
  return c
```

IV. Results

The results below represent the time execution for the languages here compared. The evaluation was on done a **sparsed** and **dense** matrices **m** and data d:

- 10×10
- 100×100
- 1000×1000
- 10000×10000
- 100000×100000

The time execution is as described on the Graph below:



We can see that the smaller the bar, the faster the language is. On the discussion section will be given details about the time execution of each language/paradigm.

V. Discussion

After evaluating all languages, **C**, **Go** and **Java** have better performance than **Haskell** and **Python** for this assignment. That happens because these three languages base themselves on the imperative paradigm and are able to perform fast scientific computations, whereas *Haskell* will be constrained by the functional programming paradigm and *Python* is a Script Language.

Before evaluating these languages, I thought that *Python* would be one of the fastest ones (together with C and Go) as it has libraries developed in *C*, making it as fast as *C*. But in practice, it did not happen. Python (pure) was the slowest language and did not complete more than 60% of the benchmarks.

That being said, in spite of being very reliable (related to code safety and complexity), *Haskell* was the second slowest languages. In my preview assignments, I said that it would most likely run slower than the others, proving that Functional Programming Languages run slower than Imperative ones, in general.

Regarding *Java*, the *JVM* warm up gave a huge boost to the execution of the scientific computations here evaluated.

To conclude, the rank of languages ended up with:

- **1st place: C**
C is evaluated as the best one (in comparison with the other ones in this assignment) because of its native advantages over the other languages. C is also one of the oldest languages amongst this set as has proven that it is indeed very optimized.

- **2nd place: Go**
Go is the newest and one of most respected languages amongst this set of languages. It also has very fast libraries. Its position is due to the fact that it is done by Google researches and they based on C (and other languages) to develop it.

- **3rd place: Java**
The *JVM* is slow, but not so slow. There still some advantage because of its *JVM's* warm-up. Some people say that scientific programming in Java is actually possible and there are a lot of support for it on its community, which was proved in this assignment.

- **4th place: Haskell**
I also believe strongly that *Haskell* will took the second last position because of the reasons discussed before on this assignment.

- **5th place: Python**
Python is the slowest languages (in comparison with the other ones in this assignment) because of its interpreted properties. Although it has many libraries written in C, having advantage over other languages, when you program in pure Python, it did not help the execution of the program.

REFERENCES

[1] "IWASEP", 2017. [Online]. Available: http://www4.ncsu.edu/ ipsen/ps/slides_iwasep.pdf. [Accessed: 28- Oct- 2017].

[2] "Freund_HLA", 2017. [Online]. Available: https://www.math.ucdavis.edu/ freund/freund_HLA.pdf. [Accessed: 28- Oct- 2017].

[3] "Big O notation", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Big_O_notation. [Accessed: 28- Oct- 2017].

[4] "Tweag I/O - Enter the matrix, Haskell style", Tweag.io, 2017. [Online]. Available: https://www.tweag.io/posts/2017-08-31-hmatrix.html. [Accessed: 28- Oct- 2017].

[5] "Functional vs. Imperative Programming", Ryanhmckenna.com, 2017. [Online]. Available: http://www.ryanhmckenna.com/2014/11/functional-vs-imperative-programming.html. [Accessed: 28- Oct- 2017].

[6] "hmatrix", 2017. [Online]. Available: http://dis.um.es/ alberto/material/hmatrix.pdf. [Accessed: 28- Oct- 2017].

[7] "For scientific computing, is Java useful in a way that C or Python aren't? - Quora", Quora.com, 2017. [Online]. Available: https://www.quora.com/For-scientific-computing-is-Java-useful-in-a-way-that-C-or-Python-arent. [Accessed: 28- Oct- 2017].

[8] "Scientific Computation", Introcs.cs.princeton.edu, 2017. [Online]. Available: https://introcs.cs.princeton.edu/java/90scientific/. [Accessed: 28- Oct- 2017].

[9] "Java And Cpp Platforms For Scientific Computing", 2017. [Online]. Available: https://inside.mines.edu/ dhale/papers/Hale06JavaAndCppPlatformsForScientificComputing.pdf. [Accessed: 28- Oct- 2017].

[10] P. Knoll and S. Mirzaei, "Scientific computing with Java", 2017.

[11] "List of C-family programming languages", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/List_of_C-family_programming_languages. [Accessed: 28- Oct- 2017].

[12] "Scientific Computing Tools for Python SciPy.org", Scipy.org, 2017. [Online]. Available: https://www.scipy.org/about.html. [Accessed: 28- Oct- 2017].

[13] "1.1. Python scientific computing ecosystem Scipy lecture notes", Scipy-lectures.org, 2017. [Online]. Available: http://www.scipy-lectures.org/intro/intro.html. [Accessed: 28- Oct- 2017].

[14] "A Primer on Scientific Programming with Python", 2017. [Online]. Available: https://hplgit.github.io/primer.html/doc/pub/half/book.pdf. [Accessed: 28- Oct- 2017].

[15] "Fortran", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Fortran. [Accessed: 28- Oct- 2017].

[16] "Programming in FORTRAN", Chem.ox.ac.uk, 2017. [Online]. Available: http://www.chem.ox.ac.uk/fortran/fortran1.html. [Accessed: 28- Oct- 2017].

[17] "Scientific computings future: Can any coding language top a 1950s behemoth?", Ars Technica, 2017. [Online]. Available: https://arstechnica.com/science/2014/05/scientific-computings-future-can-any-coding-language-top-a-1950s-behemoth/. [Accessed: 28- Oct- 2017].

[18] Sun.stanford.edu, 2017. [Online]. Available: http://sun.stanford.edu/ keiji/pixelfrt.for. [Accessed: 28- Oct- 2017].

[19] W. matrices?, "What is the usefulness of matrices?", Math.stackexchange.com, 2017. [Online]. Available: https://math.stackexchange.com/questions/160328/what-is-the-usefulness-of-matrices. [Accessed: 12- Nov- 2017].

[20] M. DeHaan, "Matrix Mathematics: How Do We Use Matrices In Day-to-Day Life?", Decoded Science, 2017. [Online]. Available: https://www.decodedscience.org/practical-uses-matrix-mathematics/40494. [Accessed: 12- Nov- 2017].

[21] Paul Gillespie, Giulio Casini, Hayley Iben, and James F. O'Brien. "Simulation of Subseismic Joint and Fault Networks Using a Heuristic Mechanical Model". Subseismic Scale Reservoir Deformation, 459:14, April 2017.

[22] Rahul Narain, Rachel A. Albert, Abdullah Bulbul, Gregory J. Ward, Martin S. Banks, and James F. O'Brien. "Optimal Presentation of Imagery with Focus Cues on Multi-Plane Displays". ACM Transactions on Graphics, 34(4):59:112, August 2015. To be presented at SIGGRAPH 2015, Los Angeles.

[23] Eric Kee, James F. O'Brien, and Hany Farid. "Exposing Photo Manipulation from Shading and Shadows". ACM Transactions on Graphics, 33(5):165:1165:21, September 2014. Presented at SIGGRAPH 2014.

[24] Michael W. Tao, Jitendra Malik, and Ravi Ramamoorthi. "Sharpening Out of Focus Images using High-Frequency Transfer". Computer Graphics Forum (Eurographics 2013), 2013.