BENCHMARK - MICROBENCHMARK (C++, C, JAVA)

1. Project Proposal

The main objective of the project is to clearly show the difference between the computation time that is needed for different kind of operations written in 3 different programming languages. The device on which the tests will be performed is a laptop using Windows 10 as the Operating System and Intel Core i5 8th generation as the processor. The programming languages in which the tests will be written are: C++, C and Java. Throughout the project, we will implement at least 3 operations from below:

- Memory allocation (static and dynamic)
- Memory access (static and dynamic)
- Thread creation
- Thread context switch
- Thread migration

All the results will be documented in tables and based on them, we will generate some graphs that will clearly show the differences between the 3 programming languages. Looking at both the tables and graphs, we are going to write some observations and, at the end, draw some conclusions. The operations using C++ and C languages will be written in Visual Studio 2019 and the operations using Java language will be written in IntelliJ IDEA 2019.3 .

In order to be more user friendly, I'm considering implementing for this project a GUI.

2. Plan

05.10.2020 – 18.10.2020:

In the first week, we had to choose the theme of the project we are going to work on throughout the whole semester. I've chosen the benchmark – microbenchmark (C, C++, C#, Java) project. Further, we were given as a homework, to start working on the project (project proposal, plan, bibliography study).

19.10.2020 – 01.10.2020:

At the lab session, I will present the documentation I've written so far and I will note the feedback I'll receive. I will ask some questions in order to make sure that everything is in line before I'm going to work further. I will make changes in the documentation, if needed, and after that I will start working on the implementation of the project.

Next, I will study more about microbenchmarks and I will write the next chapters of the documentation: Design and Analysis.

<u>02.11.2020 – 15.11.2020:</u>

I will present the evolution of the project and I will thoroughly note the feedback I will receive. Further, I will ask any question I have regarding the implementation of the project.

In the beginning, I will write the implementation in C++ language. I will document every result in a table where I will also have the minimum value, the maximum value and the average value. After repeating the tests for 10-15 times, I will generate the graphs that are going to show the “conclusion” of the experiments. I will also keep in mind every problem I’ve encountered and ask for advise the lab assistant.

<u>16.11.2020 – 29.11.2020:</u>

At the laboratory session, I will discuss together with the lab assistant the results I’ve documented using C++ language. In order to be sure everything is fine, I will notify every error or odd thing I’ve encounter when running the tests. If everything is in order and nothing major needs to be changed, I will be continuing the project development with the implementation in C language. As I’ve said before, I will document everything using tables and graphs and draw conclusions.

Keeping in mind that I already have the previously measurements, I will make a comparison between C++ and C based on the tables & graphs resulted in both experiments.

<u>30.11.2020 – 07.12.2020:</u>

At this lab, I will present the documented measurements in C# and a short comparison between the two languages used so far (C++, C). Noting the feedback I receive, I will move onto the last part of the implementation of the project, the implementation in Java. When finished, I will make a comparison between C++ and Java, another one between C# and Java and a final, more comprehensive, between all of them.

<u>08.11.2020 – 14.12.2020:</u>

In the last week before the final presentation of the project, I will finish documenting the results obtained in Java language and the comparisons made so far. I will make some last changes, if needed, then move on onto the final part of the whole project: the GUI. […]

<u>14.12.2020 – 20.12.2020:</u>

As we’ve approached the final week, we are going to make a presentation for the project implemented. Alongside the project will be a documentation where every information regarding the steps made and algorithms used will be included. Everything will be thoroughly documented.

3. Bibliography study

What is a benchmark?[1]

If referred in computing, a benchmark can be defined as the act of running a computer program/a set of programs/other operations as a mean of assessing the relative performance of an object. This is done by running standard tests and trials multiple times against it. It is commonly associated with assessing performance characteristics of computer hardware. It can be also applicable to software (i.e: benchmarks that are run against compilers or database management systems). Benchmarks represents a method for comparing the performance of various subsystems across different chip/system architectures.

What is the purpose of a benchmark?[2]

Because it became more difficult to compare the performance of different computer systems only looking at their specifications, tests that allowed the comparison to be possible were developed. *"For example, Pentium 4 processors generally operated at a higher clock frequency than Athlon XP or PowerPC processors, which did not necessarily translate to more computational power; a processor with a slower clock frequency might perform as well as or even better than a processor operating at a higher frequency"*

By discovering what is the best performance being achieved, the information can then be used to identify gaps.

What are the types of benchmarks?[3]

Based on the levels of performance they measure, they can be grouped into 2 levels: Component-level Benchmarks and System-level Benchmarks. Further, based on their compositions, they can be categorized into 2 types: Synthetic Benchmarks and Application Benchmarks.

Component-level Benchmarks: test a specific component of a computer system (i.e: video board, audio card, the microprocessor, etc.). Examples of such benchmarks: SPECweb96 -> measures the web server performance; GPC -> measures graphics performance for displaying 3D images.

System-level Benchmarks: evaluate the overall performance of a computer running real programs or applications. These types of benchmarks are useful if we want to compare systems of different architectures. Examples: SYSmark/NT 4.0 -> measures the performance of computer running popular business applications under Windows NT 4.0; TPC-C -> measures the performance of transaction processing system like

Synthetic Benchmarks: were created by combing basic computer functions in proportions that developers think they will indicate the performance capabilities of a machine under test.

These types of benchmarks try to match the average frequency of operations and operands of a large set of programs. Synthetic benchmarks are component-level-benchmarks. Examples: WinBench 97 -> measures the performance of a PC’s graphics, disk, processor, video and CD-ROM subsystems in Windows environment; MacBench 97 -> measures the processor, floating-point, graphics, video and CD-ROM performance of a MAC OS system

Application Benchmarks: employ actual application programs. Most application benchmarks are system-level benchmarks and they measure the overall performance of a system. If the application benchmark is run, it will test the contribution of each component of the system to the overall performance. Because these types of benchmarks are usually large and difficult to execute, they aren’t considered useful for measuring future needs. Examples: Winstone 97 -> test a PC’s overall performance when running Windows-based 32-bit business applications; SYSmark/NT 4.0.

What is a microbenchmark?[4]

To be bold, a microbenchmark measures the performance of something “small” (for example, a system call to a kernel of an operating system). To be more academic, a proper definition of microbenchmark would sound like: a small artificial benchmark designed to test the performance of some specific piece of code. The performance tested can be the elapsed time, rate or operations, bandwidth, latency, etc.

What are the dangers of microbenchmarking?

The main danger is that people can use whatever results they obtain from the microbenchmarking in order to dictate optimizations. Further, compiler optimizations can skew the results. Another danger is that the microbenchmarking results are influenced by other softwares running on the same system.

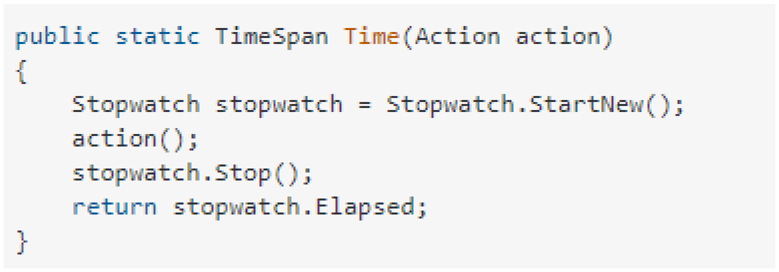
Which are the factors that affect the performance?[5]

Some factors that can affect the performance are: program input, version of the program, version of compiler, optimizing level of compiler code, version of operating system, amount of main memory, number and types of disks, version of the CPU, etc.

The difficulty in microbenchmarking is to really measure what you are interested in because the things measured are usually very small, correct timing is a problem. For example, when implementing a microbenchmark in a programming language, one must assure that the language doesn't add overhead that influences the results. Overhead, in computer science, is any combination of excess or indirect computation time, memory, bandwidth or other resources that are required to perform a specific task. Thus, it is recommended to use available tools/benchmarks which make it easier to produce meaningful results.[6]

3.1. C [7][8]

In C, the exact time measurement for performance testing can be done using the **Stopwatch** class as it provides a set of methods and properties that one can use to accurately measure elapsed time. This class can be seen wrapped in a helpful method as seen below.



```
public static TimeSpan Time(Action action)
{
    Stopwatch stopwatch = Stopwatch.StartNew();
    action();
    stopwatch.Stop();
    return stopwatch.Elapsed;
}
```

When using the **Stopwatch** class, we nee to take into account that modern CPUs have multiple cores, large caches, instruction pipelines and many other stuff that affect the run time of an algorithm in a particular test scenario. Thus, we need to prevent switching between CPU cores and processors, as switching dismisses the cache, etc. and has a huge impact on the test. We can do this by setting **ProcessorAffinity** mask of the process. Further, we must prevent other threads from using the CPU core and we can achive this by setting out process and thread priority. Finally, we need to run some warmup tests to stabilize the results. Depending on the system, the warmup tests will be ran for a certain time. For most of them, around 1000-15000 milliseconds are enough. An example where the elapsed time was computed correctly can be seen at [8].

3.2. C++ [9]

In C++, one option of measuring the execution times for a portion of C++ code is to use the classes that are available as part of the C++ 11's standard library. Such class is the **high_resolution_clock** class that is defined in the standard **<chrono> header**. This class represents a clock with the highest precision (equivalently, the smallest tick period) available on a given platform.

This class exposes the "**now**" **method**, which returns a value that corresponds to the call's point in time. This method is a static method, thus there is no need to create a new instance of the **high_resolution_clock** class. It is defined under the **std::chrono** namespace. We'll need to invoke the method twice: one time at the beginning portion of the code and the second time at the end of that portion. By doing this, we record the start and finish time of the execution.

In order to reduce that clutter and increase code readability, we can use the class **duration**. This class represents a time interval and of the same std::chrono namespace. Combining these 2 classes, we can get the elapsed time. An instance of this class using the start and finish time previously recorded:

```
std::chrono::duration<double> elapsed = finish - start;
```

Once that's in, we can invoke the **count method** of the duration class to get the elapsed time measured in seconds:

```
std::cout << "Elapsed time: " << elapsed.count() << " s\n";
```

We can also take into account, when calculating time in C++ with Windows high-resolution time stamps, a couple of Windows APIs such as **QuerryPerformanceCounter.** It can be wrapped in a convenient **C++ Stopwatch class** and its usage is very simple. Because I've already discussed the usage of this class, I'm going to skip explaining all over again.

3.3. Java [10][11][12]

When measuring time in Java, one might try to do it by using **currentTimeMillis()** or **nanoTime()**. Even though the code makes perfect sense, its results may and will be innacurate. **System.currenttimeMillis()** measures wall-clock time that may change for many reasons. **nanoTime()** is used to measure elapsed time and it will be returned in nanoseconds. We need to convert the nanoseconds accordingly (from nanoseconds in milliseconds, we divide the result by 1.000.000). However, this method doesn't guarantee nanosecond resolution (how often the value is updated).

Moving on to libraries, Apache Commons Lang provides the **StopWatch** class that can be used to measure elapsed time. This class was explained previously at page 3. An example of using the **StopWatch** class in Java:

```
class TimeTest2 {
   public static void main(String[] args) {

      Stopwatch timer = new Stopwatch().start();

      long total = 0;
      for (int i = 0; i < 10000000; i++) {
         total += i;
      }

      timer.stop();
      System.out.println(timer.getElapsedTime());
   }
}
```

For proper benchmarking, instead of measuring the time manually, we can use a framework like the **Java Microbenchmark Harness (JMH)**. **JMH** takes care of the things like JVM warmup and code-optimization paths, making benchmarking as simple as possible.

A useful example where the elapsed time was measured correctly can be seen at [12]

4. Analysis

4.1 Memory allocation and memory access

Memory access, in computer science, is the operation of reading or writing stored information. Memory allocation is the process of setting aside sections of memory in a program to be used to store variables, and instances of structures and classes. There are two basic types of memory allocation: static and dynamic. The static memory allocation is when you declare a variable or an instance of a structure or class. When you do this, for that object, the operating system allocates memory. You can access that block of memory using the name declared for that object. On the other side, dynamic memory allocation is when you make usage of new operator or with a call to the malloc function. By doing this, the operating system allocates a block of a memory of the appropriate size while the program is running. The block of memory is designated and a pointer to the block is returned.

Memory allocation: In C, we can allocate memory dynamically using functions such as malloc(), calloc(), realloc(), etc. that are found in the <stdlib.h>. In C++, the memory is allocated dynamically using the "new" operator which will allocate space dynamically. In Java language, all objects are allocated dynamically. In order to create an object, we use the "new" operator.

Memory access: In C++ and C, I've used memmove function. This function copies n bytes from memory area source, that is passed as the second parameter, to memory area destination, that is passed as the first parameter. In Java, I've used Arrays.copyOf. This function copies the specified array to the specified location or pads with zero in order to make sure the copy has the specified length.

All the results will be stored in .csv files, in milliseconds. For the memory part of the project, I'm going to repeat the operation 100000 times, make the average and then gather 1000 such averages in those csv files.

4.2 Thread

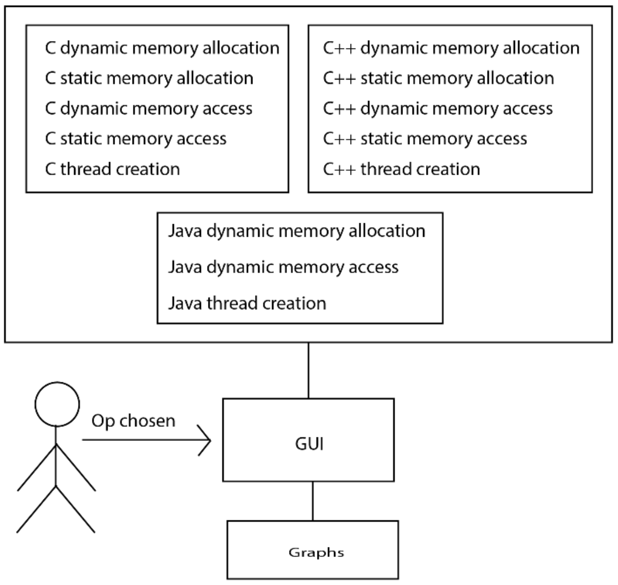
Thread context switch is the process of storing the state of a thread so that it can be restored and resume execution at a later point. In thread context switch, the virtual memory space remains the same. Context switching are usually computationally intensive, switching in and out of the operating system kernel along with the cost of switching out of the registers is the largest fixed cost of performing a context switch. [15]

For threading part of the project, I will use a simple function that multiplies a number with itself.

4.3 Pseudocode

- memory allocation:
```
start counting
create the array, statically and dynamically
stop counting
```
- memory access:
```
start counting
perform the operation that access the memory
stop counting
```
- thread creation
```
start counting
ThreadFunction function = new ThreadFunction()
Thread thread1 = new Thread(function)
thread.start();
stop counting
```

5. Design


C dynamic memory allocation
C static memory allocation
C dynamic memory access
C static memory access
C thread creation
C++ dynamic memory allocation
C++ static memory allocation
C++ dynamic memory access
C++ static memory access
C++ thread creation
Java dynamic memory allocation
Java dynamic memory access
Java thread creation
Op chosen
GUI
Graphs

This project will have a dedicated GUI (graphical user interface) that will enable the user to choose between the operation that is going to be displayed. The logic behind the GUI can be found in "main.py". The first window that will open when the user runs de app will be a window with 4 buttons: memory allocation, memory access, thread creation and close. For both memory allocation and access, the user can choose between statically allocated/accessed memory. If the user presses Thread Creation, a graph will open immediately. When the button close is pressed, the current window will disappear.

6. Implementation

We've used the Stopwatch header in order to measure the elapsed time in C++, System.currentTimeMillis() in Java and time header in C, for all the operations.

6.1 Memory allocation:

We define an array of 333, for both statically and dynamically allocations. We repeat this allocation 100000 times and make the average. In order to construct the graphs, we repeat computing the average in the manner presented previously 1000 times. Java only has dynamically allocated memory.

6.2 Memory access:

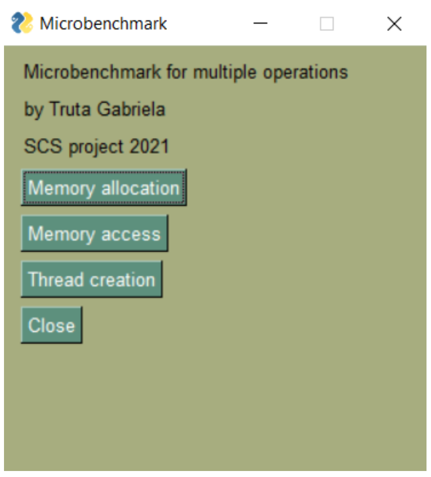
We define an array of 333 positions that holds the value of i*i where i is the index. Then we define a new array of the exact same size and then perform the operation of memmove, in order to access the memory. This is done both for C and C++. In Java, you can access memory only dynamically. In Java, a similar operation was performed, Array.copyOf. The steps in Java are the same as described moments ago.

6.3 *Thread Creation:*

For thread creation, I've chosen the same function to the one that I've mentioned before. A simple function that multiplies a certain number with itself. We loop only for 1000 times in order to get the average value as this operation requires more resources. We repeat this loop 100 times and store the values in csv files.

6.4 *GUI*

This graphical user interface was made in Python using PyCharm IDE. Below are the snippets of the application:

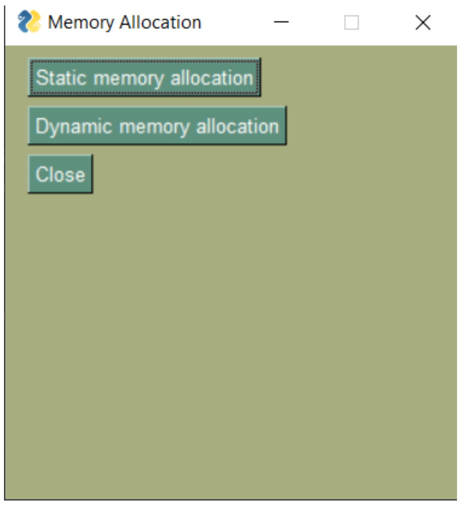


Microbenchmark for multiple operations

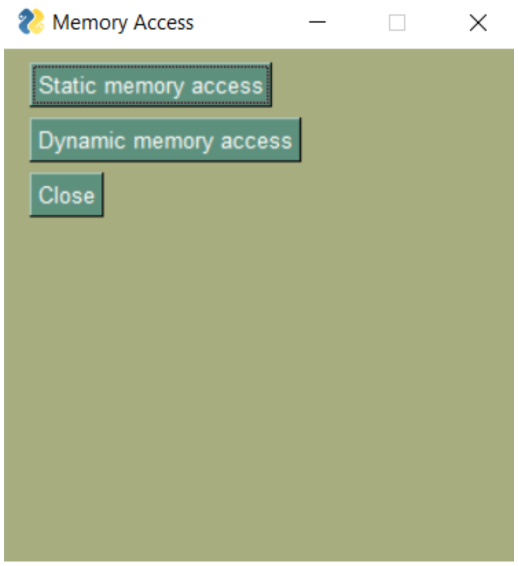
by Truta Gabriela

SCS project 2021

If the first button is pressed, another window will open with 2 buttons, one for dynamic allocation and one for static allocation. Java has only dynamic allocation, thus the graph for static allocation will only contain 2 columns, one for C and another for C++.

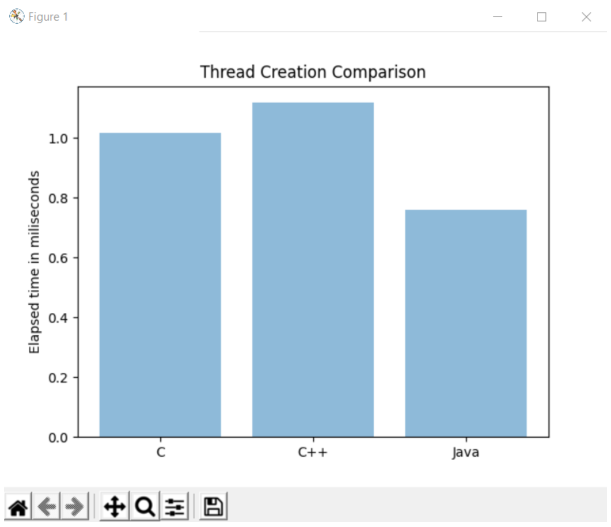


For the static memory access button, a similar window to the previous one will open:



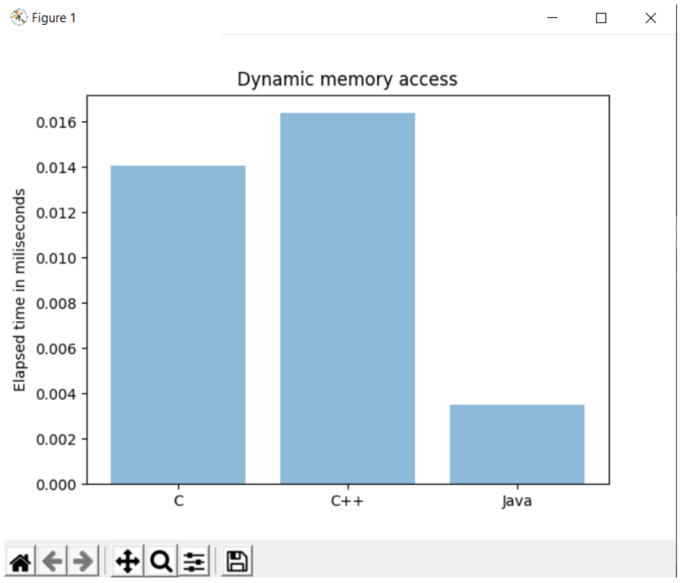
7. Testing and validation

Thread Creation:



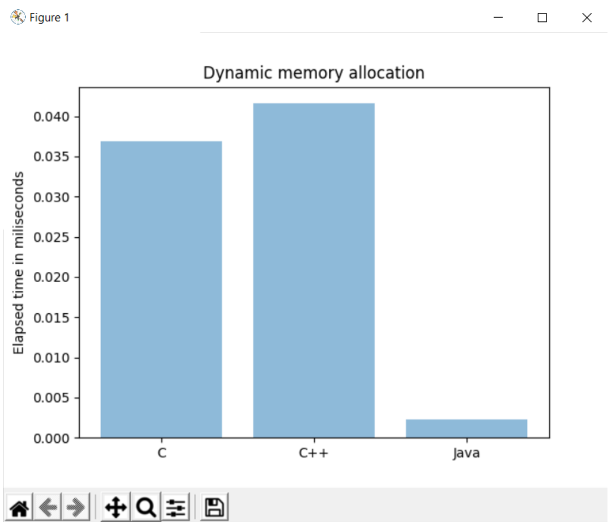
C: 1.013; C++: 1.118; Java: 0.756

Dynamic memory access:



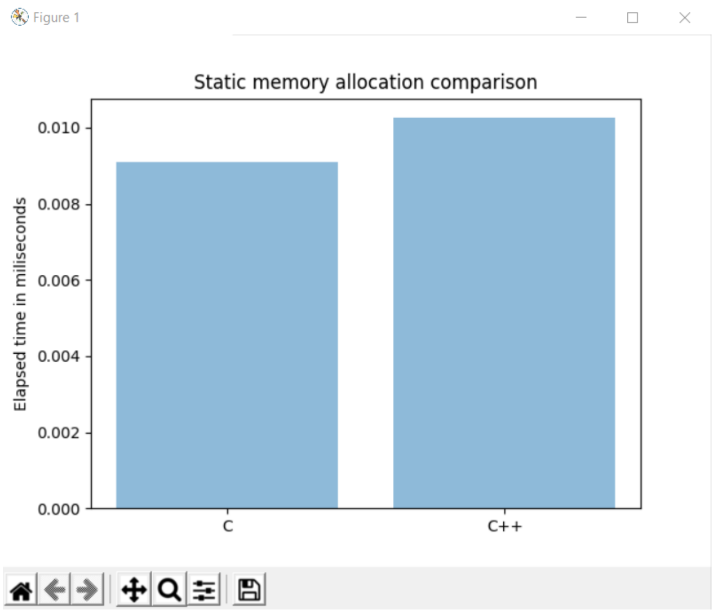
C: 0.0138, C++: 0.0163, Java: 0.00345

Dynamic memory allocation:



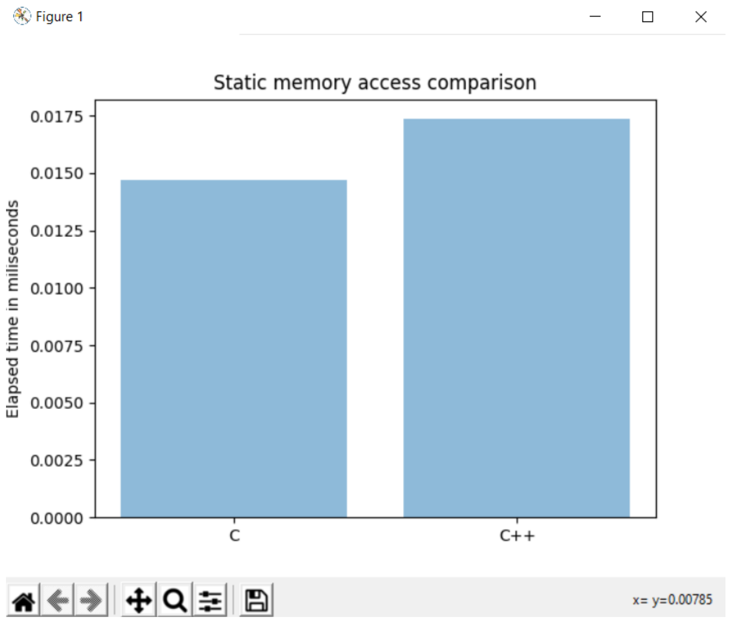
C: 0,0369, C++: 0,0416, Java: 0.021

Static memory allocation:



C: 0.0097, C++: 0.01026

Static memory access:



C: 0.01475, C++: 0.01727

8. Errors

- the values stored as csv file has an automatic scientific notation from excel that cannot be turned off, thus the graphs where the average value is written in that manner will not be reliable.
- noob errors such as: divided with 1000 instead of 10000, forgot to free a dynamically allocated array, etc.
- Intellij IDE doesn't work properly and uses more memory than needed (I've tried solving this problem by uninstalling the app, didn't work out; perhaps a windows reinstallation would be the solution, but I didn't have time to do it)
- Because of Intellij's memory error, some results might have been influenced
- I've ran the test countless of times to see where they would stabilize and ended up remarking something strange. When testing the thread creation part, both for C and C++, I've obtained results somewhere around 2,30ish ms. However, after a few hours of running the test multiple times, the results suddenly changed and stabilize around 0,8 ms, without modifying the code at all. Repeating the same thing, the results came back at around 2,32 ms and I cannot find an explanation for this.
- Surely, the processes that are running in the background influences the results. I've restarted my laptop multiple times after some strange occurs in the results to be sure that the RAM memory is free. In addition, I've turned off everything that I had open at that time, leaving only the IDE to run the tests.

Bibliography

[1] https://en.wikipedia.org/wiki/Benchmark_(computing)

[2] https://stackoverflow.com/

- What is a microbenchmark?

[3] Augusto Vega, Pradip Bose and Alper Buyuktosunoglu, *Rugged Embedded Systems - Computing in Harsh Environments* (2017)

[4] https://hpc-wiki.info/hpc/Micro_benchmarking

[5] https://www.cs.umd.edu/users/meesh/cmsc411/website/projects/morebenchmarks/measure.html

- 4.3 Reporting Performance

[6] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*

[7] https://stackoverflow.com/questions/969290/exact-time-measurement-for-performance-testing

[8] https://www.codeproject.com/Articles/61964/Performance-Tests-Precise-Run-Time-Measurements-wi

[9] https://www.pluralsight.com/blog/software-development/how-to-measure-execution-time-intervals-in-c--

[10] https://www.baeldung.com/java-measure-elapsed-time

[11] https://www.baeldung.com/java-microbenchmark-harness

[12] https://github.com/jawb-software/template-jmh-benchmark/blob/simple/src/main/java/de/jawb/jmh/benchmark/example/SimpleBenchmark.java

[13] https://www.cs.uah.edu/~rcoleman/Common/C_Reference/MemoryAlloc.html

[14] https://www.w3resource.com/python-exercises/data-structures-and-algorithms/python-search-and-sorting-exercise-4.php

[15] https://en.wikipedia.org/wiki/Context_switch

[16] https://en.wikipedia.org/wiki/Bubble_sort

[17]http://www.techthings.ca/Coding/elementaryprograms/Perfect_Square_Program_Algorithm.pdf

[18] https://www.geeksforgeeks.org/check-if-a-number-is-perfect-square-without-finding-square-root/