MICROBENCHMARK – C, C++, C#, JAVA

Table of contents:

1. Project Proposal

The objective of this project is to create programs that compute the different computation times needed for memory allocation, both statically and dynamically and thread management. For this reason, we are going to perform the measurements in 3 different programming languages: C, C++ and Java (plus another language, C#, if the time will permit it). After the measurements are done, we are going to generate tables and graphs based on the obtained time values and we are going to draw conclusions between the programming languages.

In other to perform the measurements we are going to use minimum 3 different operations:

- Thread creation
- Thread context switch
- Thread migration
- Thread creation comparison in C#
- Memory management/ memory allocation (static vs dynamic)

In addition to comparing these operations, another idea would be to perform the microbenchmark on both Windows and Linux and on other computers to see if there are performance differences between the operating systems and computers with different specifications and what are those differences. All the results will be put in tables and graphs and compared.

The project will come with a GUI where the user will be able to see the results of the measurements.

2. Plan

Week 2 (05.10.2020 – 11.10.2020)	- First laboratory regarding the project in which we chose the theme of the project and in which we were given details about the project and homework for the next project meeting (project proposal, plan, bibliographical study)
Week 3 (12.10.2020 – 18.10.2020)	- Starting to work on the project/homework for the next project session
Week 4 (19.10.2020 – 25.10.2020)	- Presenting the homework at the lab session, noting the feedback and starting the implementation based on the discussion - We start the implementation with coding the measurement functions for C
Week 5 (26.10.2020 – 01.11.2020)	- Continue the implementation for C, generating tables and graphs based on the measurement results - We test the program on both Linux and Windows and compare the results - If we can, we test the program on different computers and we compare the results
Week 6 (02.11.2020 – 08.10.2020)	- Discussing the progress during the project lab session, noting the feedback and implementing it - We start the program for the C++ measurements
Week 7 (09.11.2020 – 15.11.2020)	- Continue the implementation for C++ and we proceed in the same way we proceeded for C - We compare the result between C and C++
Week 8 (16.11.2020 – 22.11.2020)	- Laboratory project session – discuss the results and the implementations - We start programming the Java measurements
Week 9 (23.11.2020 – 29.11.2020)	- Continue the implementation for Java and we proceed in the same way we proceeded for C and C++ - We compare the result between the 3 programming languages, we generate tables and graphs based on the measurements and we draw conclusions
Week 10 (30.11.2020 – 6.12.2020)	- We present the results during the project lab session

	- We start implementing the code for comparing thread creation in C# - We start implementing a GUI for the project
Week 11 (07.12.2020 – 13.12.2020)	- We compare the results from thread creation in C# - We continue the GUI and add the finishing touches - We gather all the data and we add it to the final documentation
Week 12 (14.12.2020 – 20.12.2020)	- We present the project and the documentation

- The weeks marked with red are the weeks in which we have laboratory project sessions
- All of the steps will be implemented in code and written in the documentation to assure that the work regarding the documentation will not remain for the last weeks
- All the test measurements will be repeated 10-15 times. We will extract/highlight the minimum, maximum and average values
- Regarding the tests made on other computers, we will note the specifications of that computer vs the specifications of my personal computer in order to see how much those are going to affect the end results
- Another idea would be to compare the final test results obtained with some average values found on the internet

3. Bibliographical study

In order to start implementing the different programs for the project we have to first define some theoretical notions, such as: what a benchmark/microbenchmark is, how we can measure the execution time in an accurate manner, what we should measure and how.

- What is a benchmark?

  “In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it”[1].

  In other words, benchmarks are created in order to copy the behavior of a particular system, component or program for the purpose of measuring different performance times and comparing them. Using all these values and comparisons, engineers can furthermore improve the said component or application.

  There are different types of benchmarks available:

  - Real program
  - Component benchmark/microbenchmark
  - Kernel
  - Synthetic benchmark
  - I/O benchmark
  - Database benchmark
  - Parallel benchmark

Synthetic benchmarks mimic the behavior of a system/component by running specially created programs that “impose the workload on the component”[2] and are used to test individual components. At the same time, application benchmarks run real-world programs and usually give a much better real-world performance on a given system.

Even though benchmarks should focus on testing the performance of different systems and should be a guide regarding those performances, it has some challenges like the fact that the benchmarking institutions are often disregarded or do not follow scientific methods, many benchmarks focus on only one application, excluding others or users can have different perceptions of performance than the benchmarks may suggest.

There are a few vital characteristics for benchmarks[3]:

- Relevance: Benchmarks should measure important features.
- Representativeness: Benchmark performance metrics should be broadly accepted by industry and academia
- Equity: All systems should be fairly compared
- Repeatability: Benchmark results should be verifiable
- Cost-effectiveness: Benchmark tests should be economical
- Scalability: Benchmark tests should measure from single server to multiple servers
- Transparency: Benchmark metrics should be readily understandable

- What is a microbenchmark?

  A microbenchmark is a core routine that consists of a relatively small and specific piece of code in order to measure performance of a computer’s basic components. Because of this, it is often used to analyze the performance of critical code in order to see how that operation will impact the performance of the whole application or how much it takes to be executed. Microbenchmarking is not a black box but a tool to understand better the system.

  Besides this, microbenchmarking[4]:

  - provides upper performance limits for sustained performance
  - creates knowledge about performance behavior
  - helps finding performance bugs in architectures
  - provides undocumented processor performance properties
  - quantifies the cost of programming model constructs or runtime environments
  - provides input for performance models
  - helps to learn how software interacts with the hardware

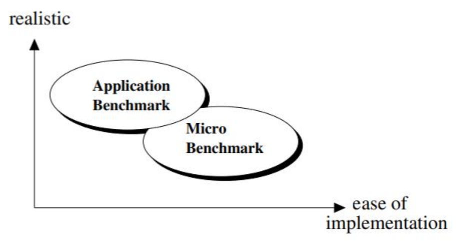


Figure 1. The relation between application level benchmarks and microbenchmarks

[5]

- Ways of measuring execution time

- In C there are four commonly used methods for measuring the execution time [6]:
    - The clock() function provided in the <time.h> header to calculate the CPU time consumed by a task. It returns clock_t type which stores the number of clock ticks. In order to obtain the time in seconds, we have to divide that value with CLOCKS_PER_EC macro, also in <time.h>
    - The time() function returns the number of seconds elapsed since the Epoch (00:00:00 UTC, January 1, 1970). It takes pointer to time_t as an argument which is usually passed as NULL and returns type_t type.
    - The gettimeofday() function returns the clock time elapsed since Epoch and stores it in the timeval structure, expressed in seconds and microseconds. It is defined in the <sys/time.h> header. The timeval structure is declared as:

      ```
      struct timeval {
      long tv_sec; //seconds
      long tv_usec; //microseconds
      };
      ```

    - The clock_gettime() function defined in <time.h> header file supports upto nanosecond accuracy. It takes 2 arguments, clock type and pointer ti timespec structure declared as:

      ```
      struct timespec {
              time_t tv_sec; //seconds
              time_t tv_nsec; //nanoseconds
      };
      ```

- In C++ time(), clock(), gettimeofday(), clock_gettime() functions in C work in C++ as well. Apart from these, in C++ we have [7]:
    - chrono::high_resolution_clock function. “Chrono library is used to deal with date and time. This library was designed to deal with the fact that timers and clocks might be different on different systems and thus to improve over time in terms of precision.chrono is the name of a header, but also of a sub-namespace, All the elements in this header are not defined directly under the std namespace (like most of the standard library) but under the std::chrono namespace.”

   - At the same time, in C++ we have a lot of different libraries that help in the microbenchmarking process [8]:
     - Hayai library
     - Celero library
     - Nonius library
     - Google Benchmark library

- In Java, for measuring execution time we can use [9]:
   - System.nanoTime() in order to measure elapsed time with nanoseconsds precision. It returns the current value of the running JVM's high-resolution time source, in nanoseconds
   - System.currentTimeMillis() runs the difference between the current time and midnight, January 1st, 1970 UTC in milliseconds. Ideally, it should be used to measure wall-clock time and nanoTime() should be used to measure the elapsed time of the program. If the elapsed time is measured with this function, the result might not be accurate
   - Instant.now() obtains the current time from the system clock. We can convert this instant to the number of milliseconds using the toEpochMilli() method. This function internally uses System.currentTimeMillis() and thus it might not be the best solution to measure the elapsed time
   - Guava's StopWatch class can be used instead of System.nanoTime() as StopWatch provides more abstraction by not exposing the absolute value returned by nano.Time() which serves no importance
   - Apache Commons Long StopWatch – we can use the StopWatch API by Apache to measure the execution time in milliseconds. To start the watch we use start() or createStarted() method and then we use getTime() to get the time between the start and the moment the method is called or we can also use stop() before getTime() to return the amount of time between start() and stop()
   - Date.getTime() in order to see the time in milliseconds since January 1st, 1970 GMT
- In C# we can also use the Stopwatch class from System.Diagnostics namespace or StopWatch GetTimestamp() method that return the current number of ticks of the underlying timer mechanism (a millisecond is formed by 10000 ticks)

- Benchmark/Microbenchmark programs [10]
- NovaBench:
   - Site: https://novabench.com
   - Small and good software for benchmarking test of your PC to test the speed and compare it with other PCs
   - On start up it displays the processor, instructions, memory and graphic details
   - We can run all tests or run without GPU test; individual tests can also be taken, like CPU test (floating point operations, integer operations, MD% hashing tests), GPU test, hardware test (RAM transfer speed and Drive Write speed test) etc.
   - Benchmarking test results are displayed showing NovaBench score, system details, RAM score, RAM speed, CPU tests score, floating point operations per second, integer operations per second, MD5 hashes generated per second,

graphics test score, 3D frames per second, hardware test score, primary partition capacity, and drive write speed. The scores can be compared, viewed in graphical form, and compared on NovaBench website, or can be saved as NBR. The results can be shared on their site, where an average score for that operation will be displayed

- SiSoftware Sandra:
  - Site: https://www.sisoftware.co.uk
  - Freeware benchmarking tool that has several tools to test benchmarks, hardware and software details
  - The user can create reports, get analysis and advice, test the stability, monitor computer environment, add module, connect to computer, VM, database and more
  - After the benchmarks, the program can display the overall score of the PC and the top ranking performing PCs
  - The processor portion holds the options to test the processor arithmetic, processor multi media, cryptography, processor financial analysis, processor scientific analysis, processor multi-core efficiency, and processor management efficiency. In case of the (GP) general purpose computing, the GP processing, GP cryptography, GP financial analysis, GP scientific analysis, GP bandwidth, GP memory latency, and GP cache bandwidth can be generated.
  - Other operations that can be measured are: memory bandwidth, cache and memory latency, cache bandwidth, memory transaction throughput, .NET arithmetic, .NET multi-media, Java arithmetic, Java Multi-Media, video shader compute, media transcode, video memory bandwidth, network(LAN, WLAN, WWAN), internet connection, and internet peerage.
  - In the hardware tab the user is provided with hardware details about hei computer and in the software tab the user can see details about the computer's software configuration

- XTREMEMARK:
  - Site: https://www.xtreme-lab.net/en/xmark.htm
  - On start-up one is required to choose different settings like threads to execute, threads priority, quantity of operations, and simple report generation options
  - The benchmark result display the test start time, test end time, threads executed, thread priority, quantity of operations, average operations per second, time taken by thread 1 and thread 2, total time spent and global time spent. The generated report can be exported as RTF, and TXT.

4. Analysis

As mentioned in the proposal of this project, the main focus of the program will be to analyze the different computation times needed for memory allocation (static + dynamic) and thread management. In order for these to happen, the project will have the next functionalities:

- Thread creation in C, C++ and Java
    - The threads will be created using the specific functions from each programming language
    - In C, for Linux we could use the POSIX functions from <pthread.h> header in order to create and manage the threads, while for Windows we can use the functions from the <windows.h> header like DWORD WINAPI ThreadFunction (LPVOID parameter) or <processthreadsapi.h> header and CreateThread function
    - In C++ we could use the class std::thread in order to create and manage threads from <thread> header
    - In Java we can use and extend the class Thread or to implement the Runnable interface
- Thread context switch in C, C++, Java
    - A context switch is the process of storing the state of a process or thread, so that it can be stored and resume execution at a later point. This allows multiple processes to share a single CPU and is an essential feature of a multitasking operating system
    - For C, we can use the <processthreadsapi.h> header and the functions from it in order to control the thread context. We can suspend the threads, switch among them, resume a thread, get its context and more using function like SuspendThread(), SwitchToThread(), ResumeThread(), GetThreadContext() etc. For inux we can use Mutex locks or Semaphores in order to manage the thread switching
    - For C++ we can use the <thread> header and std::condition_variables or std::unique_locks in order to block or start threads
    - For Java we can use functions like sleep(), wait() or notify() from the Thread class
- Thread migration in C, C++, Java
    - When a thread reaches the migration point, the OS executes the migrations which sends the thread between kernels. In other words, it moves the thread from one core’s running queue to another
    - Thread migration can be done using scheduling and the headers and functions above
- Thread creation comparison in C#
    - We compare different method for creating threads in C#
    - In C# we have two types of threads: foreground and background. The foreground thread keeps on running even if the Main thread leaves it process, or in other words its life does not depend upon the main thread. However, the background thread depends on the main thread; if the main thread finishes its process, then the background one will also end its process
    - Both types of threads can be created using Thread class from System.Threading

- Memory management/ memory allocation (static vs dynamic)
    - Memory allocation is the process of setting aside sections of memory is a program to be used to store variables and instances of classes and structures. In Static memory allocation the memory if the data is allocated when the program starts and the size is fixed. In Dynamic memory allocation we assign the memory space during the execution time or the run time.
    - In C, we can allocate memory dynamically using functions from the <stdlib.h> like malloc(), realloc(), free(), calloc() and statically (for examples using arrays of set length)
    - In C++ we can use the operator *new* to allocate space dynamically and *delete* to erase the allocated memory. Also, the malloc() function from C exists in C++. The static memory allocation is like the one in C
    - In Java, all objects are allocated dynamically. We are always passing around references to the objects. We use *new* in order to create a new object. The new created object is then allocated on the heap and a reference to it is stored in the stack (assuming that it is inside a method). This means that we can always pass objects without worrying about where they are stored. This is also true for arrays.
- Besides these functionalities, the programs will perform measurements that will be stored in a .txt, .csv or .xml file. The results will all be in seconds
- We will have a GUI from where the user will be able to choose the operation and the language and will see the results in tables and graphs. The GUI will extract the data from the generated files.
- As actors we will have the administrator/programmer that works on the code and a user that wants to test the speed of an operation

Pseudocodes:
- The ideas for the pseudocodes/implementations is no that different among the operations
- For the threads the ideas are similar; we will have a main body where we choose the number of threads to be created, the creation of the threads (or separate functions for creation outside main) and separate functions that will work on thread scheduling:

```
threadOperationFunction(); // used for scheduling – not necessary for just
                           // creation
main()
        measure start time before thread creation;
        createThread();
        threadOperationFunction(); // if needed
        measure end time;
        put total time in a file;
```

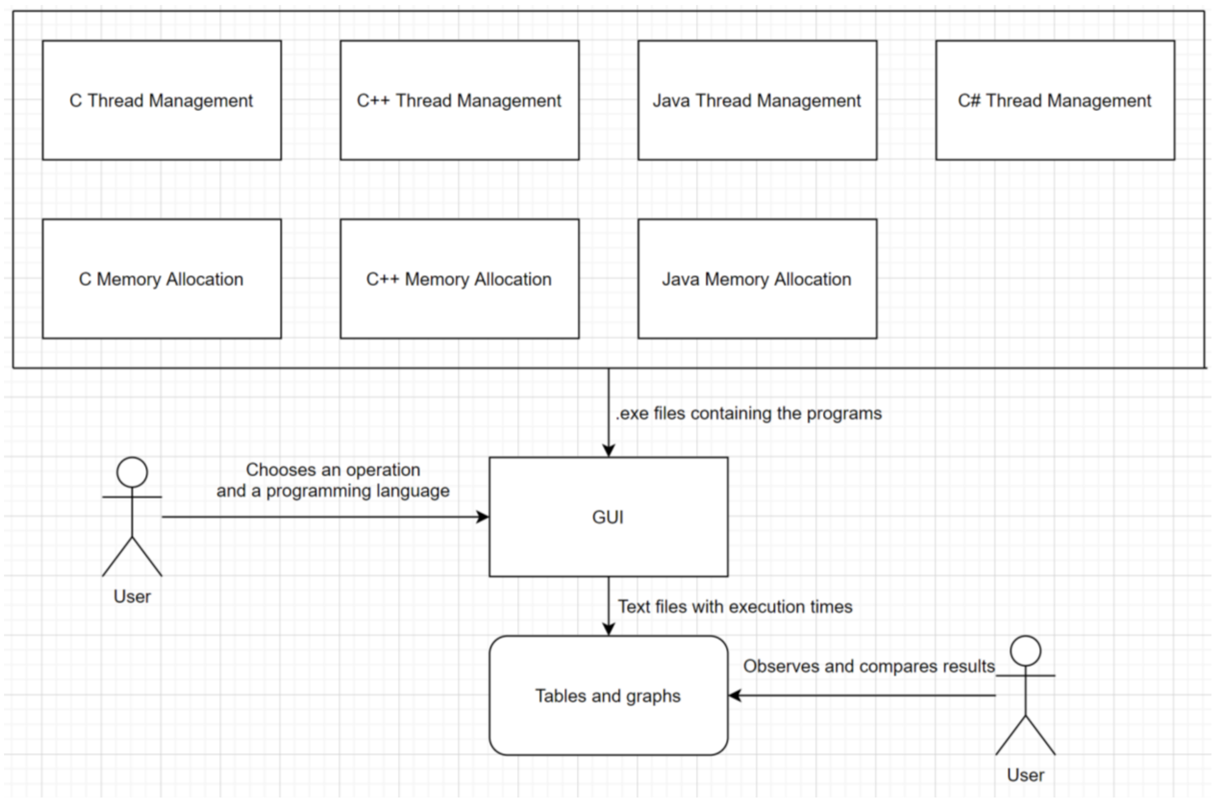
- For the memory allocation, we will allocate memory for an array statically and dynamically and fill the array with random values

```
fillArray();
main()
        create new array statically and dynamically;
        measure start time;
        fillArray();
        measure end time and put total time in a file;
```

5. Design

In the project we will have 3 (or 4, the C# one) main programs for computing the thread management operations and 3 main programs for computing the memory allocation. This is done in order to minimize the amount of files that we have. After the programs are run, the total execution times will be put in separate files. GUI will process the .exe program of every functionality and create tables and graphs. On the GUI the user will be able to select between the operations and the programming languages. Base on the graphs, the user can draw different conclusion on the used operations.

Simple schematic of how the program is going to work:



6. Implementation

6.1 Thread Creation

a) C

- For the creation of threads in C, we use the <Windows.h> library and created the threads with CreateThread() function
- Using the function show(int index) we do an simple add operation to a variable
- We wait for all the threads to be created and we close them one by one
- We measure the start and finish times using clock() and the beginning of the thread creation loop and after they were closed
- We measure 1 thread 1000 times and we print in a .txt file all the total times resulted from each measurement and at the bottom of the file we put the average time

- We use clock() in order to measure the beginning and end time

b) C++

- For C++ we use the <thread> header that contains the class std::thread
- The implementation resembles the one from C a lot, but instead of using an array of threads, we use the class std::vector in order to keep the threads
- The thread function will perform addition to a simple variable
- The thread function will display the thread, its number and the fact that it was created
- In the file we keep the measurements for creating 1 thread 1000 times and the average time

c) Java

- For Java we use a new class, MyThreadFunction that will extend the class Threads. This class will act as the function from C and C++ and perform addition to a variable
- For measuring, we use the method currentTimeMillis() from class System in order to keep track of the time in milliseconds from the creation to the end of the threads. This time will be then converted into seconds and put in a file
- We create the file using FileWriter and we write in it with BufferedWriter.
- We measure 1 thread 1000 times

6.2 Thread Context Switch

- In the context Switch operation the creation of the threads is the same as the one described in section 6.1 for each programming language
- Apart from that, we now use synchronization methods such as mutex locks in order to make sure that the second thread waits for the first one and begins only when it has ended
- The time for context switch is measured from the ending of a thread to the start of a new one using the same functions as above
- We perform 1000 measurements and the total times are kept in separate files like in the section 6.1

6.3 Thread Migration

6.4 Memory Allocation

- We will compute the allocation of the memory both statically and dynamically
- For this we will have an array of size 10000 and a pointer for which we will allocate enough memory and we will fill them with random values using a for loop
- We will measure the start time before the loop and the end time after the values are filled and we will subtract them
- The total time will be kept in different files for every programming language
- The functions for measuring the time are clock() for C and C++ and System.currentTimeMillis() for Java

6.5 GUI

7. Testing and validation

8. Conclusions

- Bibliography

1. Fleming, Philip J.; Wallace, John J, "How not to lie with statistics: the correct way to summarize benchmark results"
   - We extracted the definition of the benchmark from here

2. https://en.wikipedia.org/wiki/Benchmark_(computing)
   - We extracted further explications about benchmarks, types of benchmarks, the difficulty regarding benchmarks, their importance, what a microbenchmark is

3. Dai, Wei; Berleant, Daniel (December 12–14, 2019), "Benchmarking Contemporary Deep Learning Hardware and Frameworks: a Survey of Qualitative Metrics"
   - The principles of benchmarks

4. https://hpc-wiki.info/hpc/Micro_benchmarking
   - Uses of microbenchmarking

5. Ehliar, Andreas; Liu, Dake. "Benchmarking network processors"
   - The relation between application level benchmarks and microbenchmarks (picture)

6. https://www.techiedelight.com/find-execution-time-c-program/
   - How to measure execution time in C

7. https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
   - How to measure execution time in C/C++

8. https://www.bfilipek.com/2016/01/micro-benchmarking-libraries-for-c.html#comparison
   - C++ microbenchmarking libraries

9. https://www.techiedelight.com/measure-elapsed-time-execution-time-java/
   - How to measure execution time in Java

10. https://listoffreeware.com/list-of-best-free-benchmarking-software-for-windows/
    - Extracted information about three different benchmarking applications