**Bits Please**

Computer Organization Project

Coordinated by: Made by:

Alexandru TOPIRCEANU Adina IORGA

Amira IORGOVAN

Andreea MIHOC

Bianka NAGY

Anita PATRUT

Paul PETRISOR

Table of Contents

[1. Introduction 3](#_Toc9459630)

[What is BitsPleaseBenchmark? 3](#_Toc9459631)

[Why BitsPleaseBenchmark? 3](#_Toc9459632)

[2. Design and implementation 3](#_Toc9459633)

[How did we do it? 4](#_Toc9459634)

[3. State of the Art 6](#_Toc9459635)

[4. Usage 7](#_Toc9459636)

# Introduction

### What is BitsPleaseBenchmark?

BitsPleaseBenchmark is a benchmark with a friendly User Interface which is used for testing the performance of the Central Processing Unit(CPU) and also the performance of the Hard Disk Drive(HDD). The functionalities used for testing the CPU are:

* Compression test - using Huffman compression;
* String sorting - using Quick sort;
* Prime numbers- using the classic method;
* Digits of Pi- using Machin’s formula;
* Threading - computing the square root using Newton’s method.

The functionalities used for testing the HDD are:

* Threads

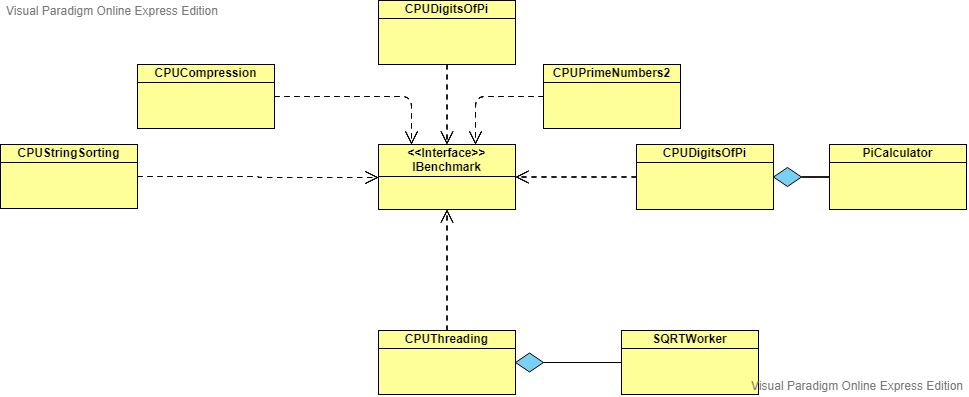
### Why BitsPleaseBenchmark?

BitsPleaseBenchmark is not a completely new benchmark. It uses features from other benchmarks, all combined in a single application. The aim is to test five different functionalities of the CPU by using algorithms which are easy to understand for everyone.

# Design and implementation

BitsPleaseBenchmark has the following features:

* Friendly User Interface;
* Database for the results of the benchmark, updated in real-time;
* Portability.



### How did we do it?

The benchmark is measuring the execution time for performing one or all of the operations. For calculating the score, we have used the following formulas:

These scores were calculated in a for loop. After that, for an average result, we divided the resulting scores as you can see below:

Then we calculate the mean of these scores and we obtain the final score:

The application is using the five functionalities mentioned before:

* Compression test - using Huffman compression;

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT.

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol. The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted..

Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code (sometimes called "prefix-free codes", that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol). Huffman coding is such a widespread method for creating prefix codes that the term "Huffman code" is widely used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

* String sorting- using Quick sort

Quicksort is an efficient sorting algorithm, serving as a systematic method for placing the elements of a random access file or an array in order. It is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation is defined. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting. It is very similar to selection sort, except that it does not always choose worst-case partition.

Mathematical analysis of quicksort shows that, on average, the algorithm takes O(n log n) comparisons to sort n items. In the worst case, it makes O(n2) comparisons, though this behavior is rare.

* Prime numbers

A primality test is an algorithm for determining whether an input number is prime. Among other fields of mathematics, it is used for cryptography. Unlike integer factorization, primality tests do not generally give prime factors, only stating whether the input number is prime or not. Factorization is thought to be a computationally difficult problem, whereas primality testing is comparatively easy (its running time is polynomial in the size of the input).

Our prime number test is based on the formula, which gets a number x of numbers as input, and checks from until, if x is divisible by . All these operations are made inside a for loop. If,, then is a prime number.

* Digits of Pi- using Machin’s formula

The algorithm for computing the digits of PI uses Machin’s formula:

where the development in power series of arctan is:

In writing the code is being used the BigDecimal class which can save the digits until the computer runs out of memory. The total number of decimal digits of PI is obtained depending on the parameter received from the initialize() method.

* Threading – computing the square root using Newton’s method.

This branch of BitsPleaseBenchmark is implemented by creating a Worker, SQRTWorker, where Newton’s method is used for computing the square root of a number. The Newton method comprises of the idea: one starts with an initial guess which is reasonably close to the true root, then the function is approximated by its [tangent line](http://en.wikipedia.org/wiki/Tangent_line) (which can be computed using the tools of [calculus](http://en.wikipedia.org/wiki/Calculus)), and one computes the *x*-intercept of this tangent line (which is easily done with elementary algebra). This *x*-intercept will typically be a better approximation to the function's root than the original guess, and the method can be [iterated](http://en.wikipedia.org/wiki/Iterative_method).

To the worker are sent the limits of an interval on which the square root of each number will be calculated. The CPUThreading class splits the interval [1;10M] in smaller interfaces proportional to the number of threads used and sends through different threads those intervals to the worker to compute the square root.

The number of threads sent is varying from 1 to 16, for consecutive powers of 2, giving different times that will be used in computing the partial score belonging to this benchmark.

The application runs from the class UI which is a Jframe.

# State of the Art

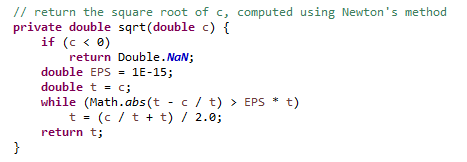
We have got inspired from some well known benchmarks, such as: wPrime and CPUBench. From the wPrime we have taken only the idea of testing the CPU by using Machin’s formula to calculate the digits of PI and Newton’s method for obtaining the square root of a number.

CPUBench – is a CPU benchmark from which you can choose what benchmark algorithms to be used for measuring the performance of the CPU. A feature that we have used from CPUBench is the fact that it combines many benchmark algorithms even though we have implemented five of them. The second feature we have used for our own benchmark is a database updated in real-time.

wPrime – is a benchmark that calculates the square roots of the first 32 or 1024 million integers. To run the benchmark select one of the first options (“Run 32M” or “Run 1024M”). After the benchmark is finished you can view your scoring time in the “View Scores” button. The score is based on the total benchmark time, and you can either save it to file or upload it online. The interface is based on a simple and minimalistic window, where wPrime gets started by retrieving hardware information from your PC, thus revealing the CPU specs, core, revision, clock, cache, VCore, FSB, motherboard model and vendor, together with the memory type, size, clock and timings.

wPrime uses a recursive call of Newton's method for estimating functions, with f(x)=x2-k, where k is the number we're sqrting, until Sgn(f(x)/f'(x)) does not equal that of the previous iteration, starting with an estimation of k/2. It then uses an iterative calling of the estimation method a set amount of times to increase the accuracy of the results. It then confirms that n(k)2=k to ensure the calculation was correct. It repeats this for all numbers from 1 to the requested maximum.

We used another implementation of Newton’s method:



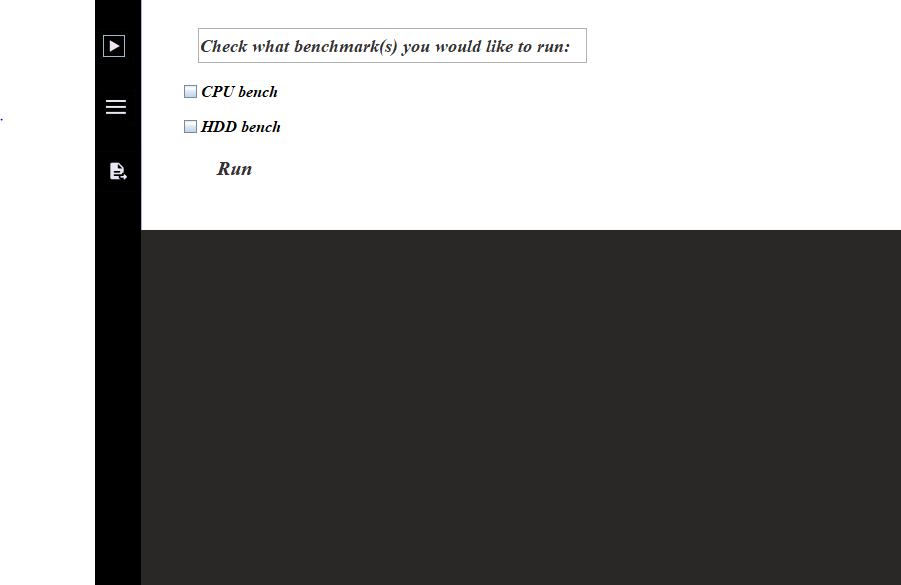
**Advantages:**

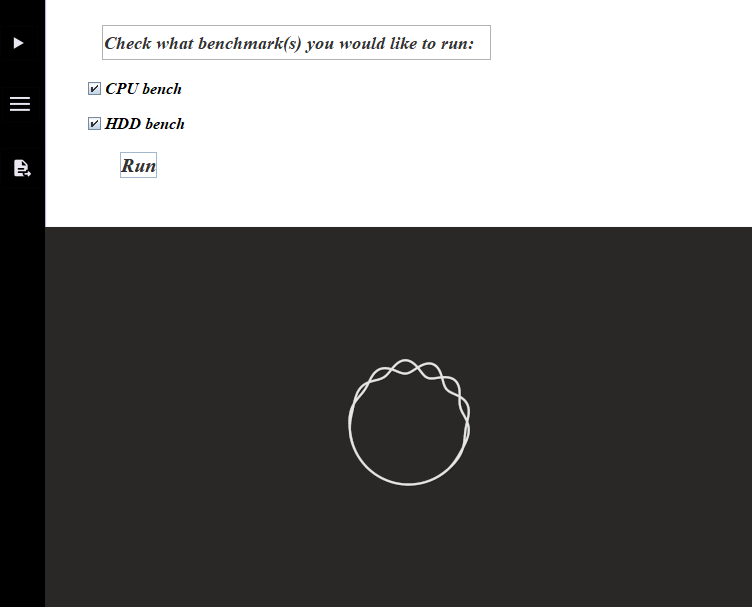
* Friendly User Interface
* Easy to use
* Real-time updated database
* Five benchmarking algorithms for the CPU

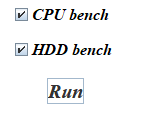
**Disadvantages:**

* Fewer benchmarking algorithms for testing the HDD

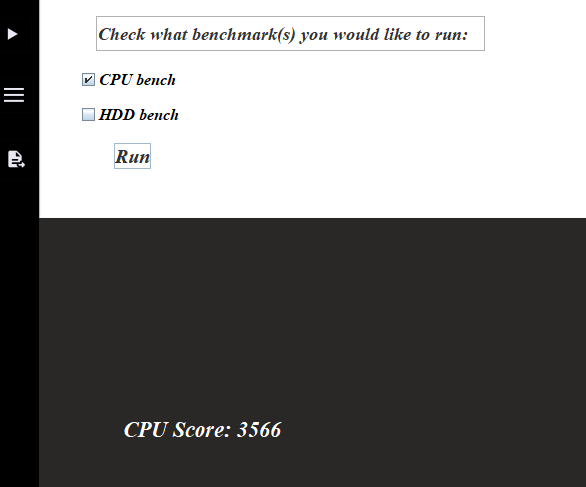
# Usage

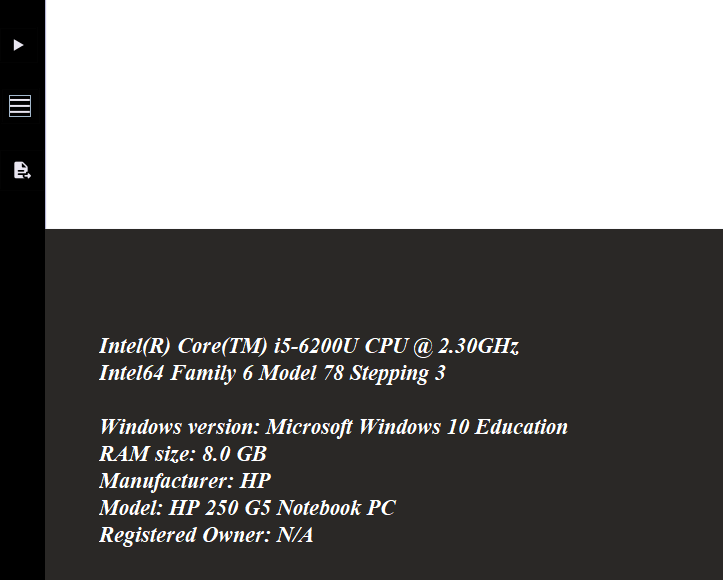
BitsPleaseBenchmark is easy to use for anybody. It has a friendly interface with clear text messages. First of all, you have to click on the run button on the left, so you can choose what you want to benchmark. Your pc’s CPU, its HDD or both. After checking the little boxes, click on the Run button, so the program can start benchmarking.

For example, I decided to benchmark both the CPU and the HDD as it shows:

While the program is working, a circle will be shown on the bottom half of the window. In the top left corner we can see three buttons. The first one is the Run button. You have to click it, so your choices will appear and you can choose what you want to benchmark.

When the bench finishes, the score of your computer’s CPU will be shown:

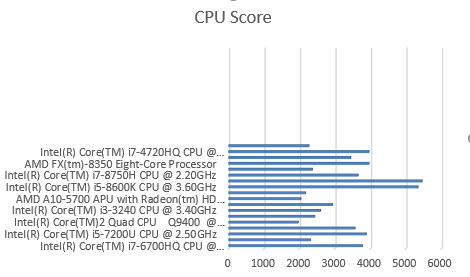


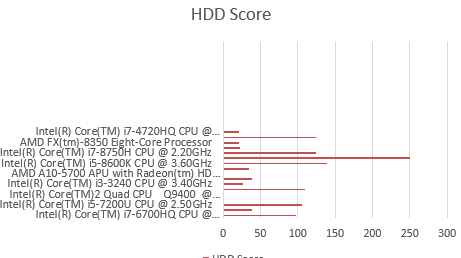
 The second button on the left, will show you some information about your computer, such as the model of the CPU, RAM, OS, the model of your computer and so on.

The third, and also the last button, is for submitting your score to a real-time database, where the scores of the computers are stored. After a successful submit, a message will appear under the button.

After finishing the benchmark a score is given depending on the execution time. The results are stored in an online database in real-time.

1. **Results**





1. **Conclusions**

What did we learn from this project?

1. We can use PI for many more calculations, not just for the radius
2. How to make a executable file from a working source code
3. How hard is to make a simple application
4. Teamwork
5. Details are very important and hard to handle
6. How to organize our work and source code

What we…?

1. Thought was hard: Making the interface, connecting the interface to the core part, database connection and maintenance
2. Enjoyed: Working with friends, learning new things
3. Hated: Fixing the errors
4. Liked: The satisfaction of the final result
5. Disliked at project: Writing the documentation

We think that there is nothing to be changed in these labs. This kind of teaching is very interactive, thank you for showing us that learning can be fun even at university.