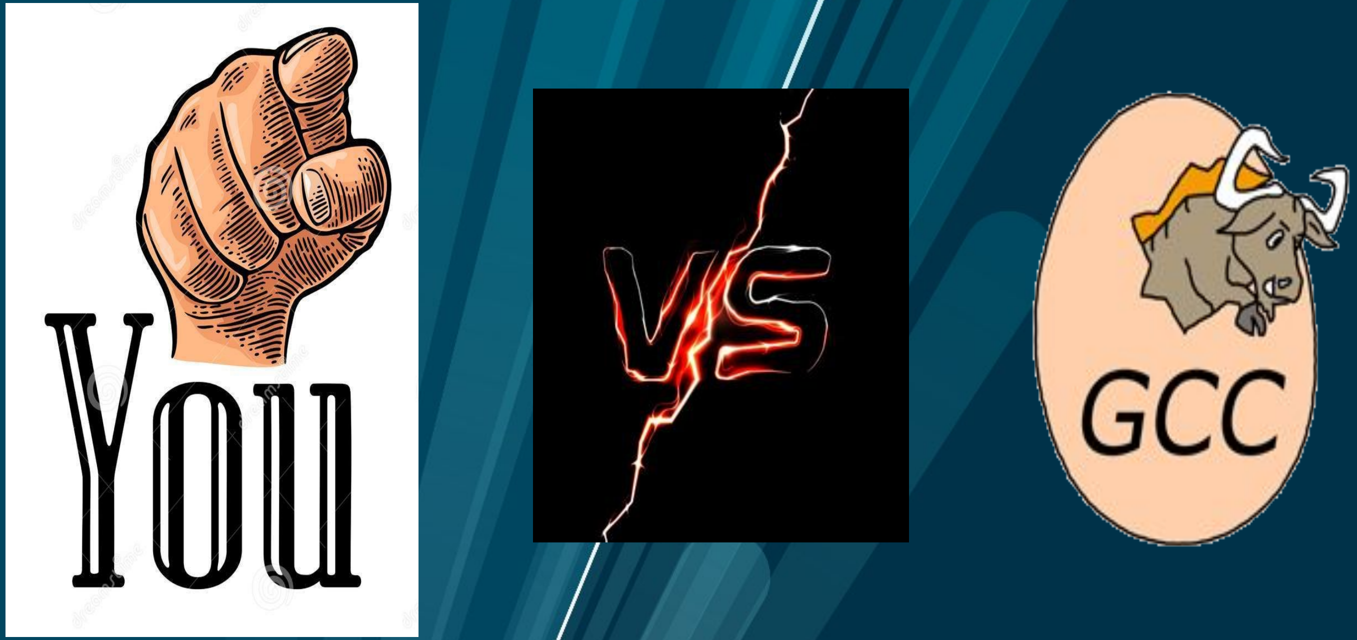
UiT The Arctic University of Norway

Mandatory assignment 1

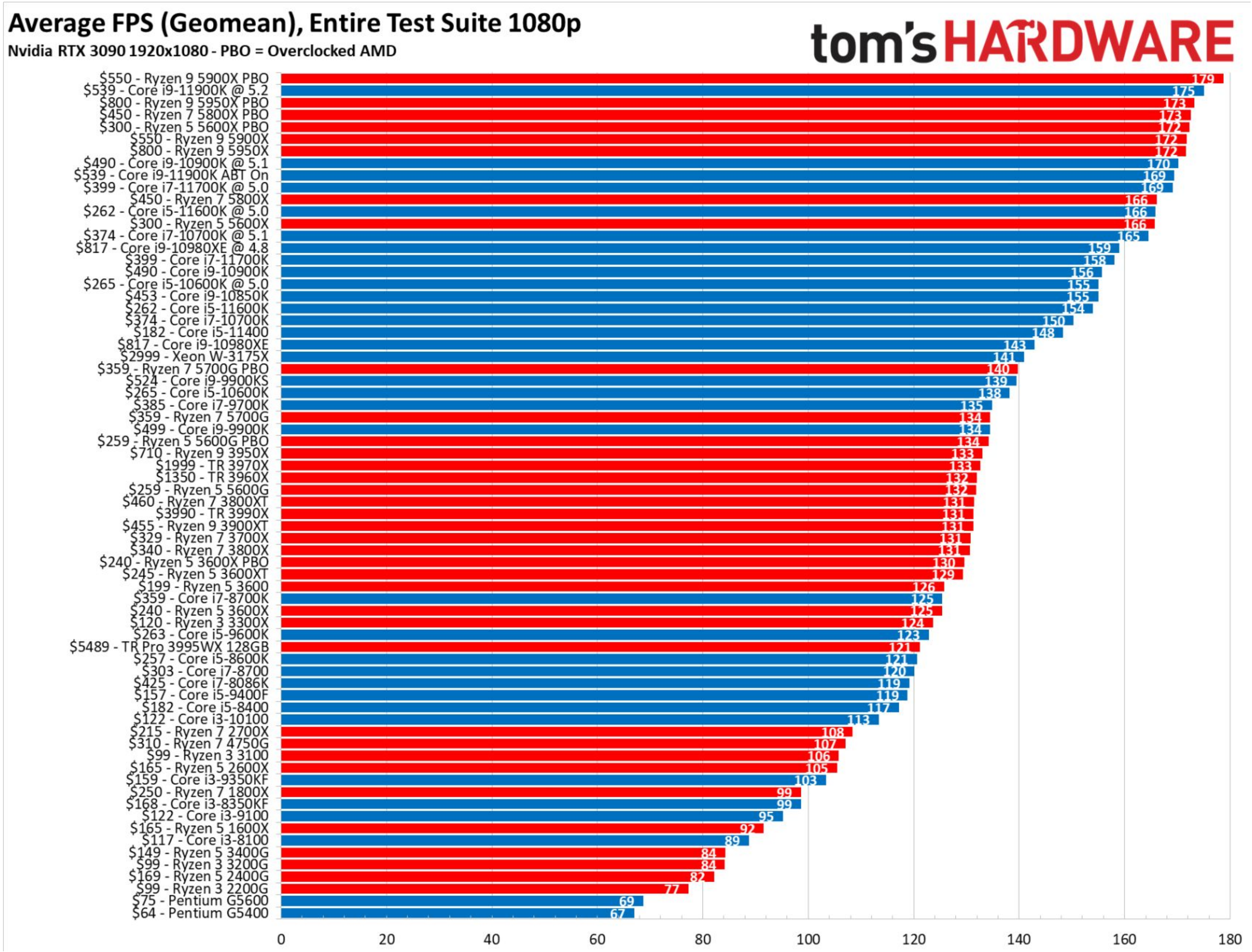
INF-2200 Fall 2023



Lars Ailo Bongo (larsab@cs.uit.no)

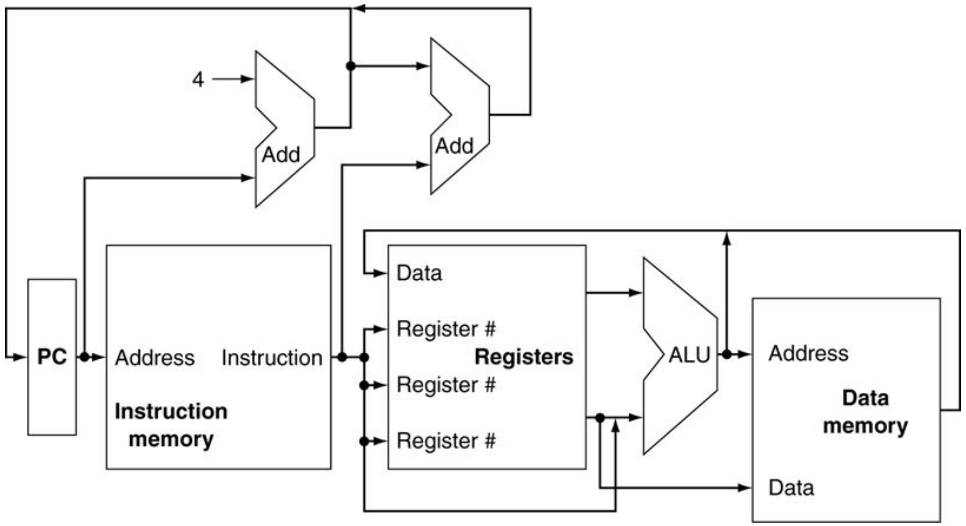
21.08.23

1. Select a benchmark
2. Identify hotspot
3. Implement hotspot in assembly and C
4. Compare performance
5. ~~Write (a good) report~~
   Produce a very good report using ChatGPT

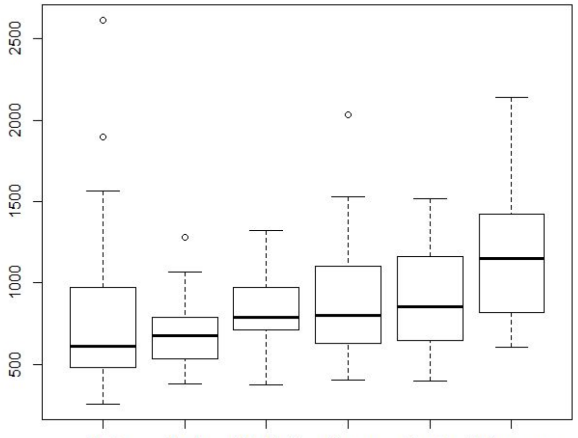


Average FPS (Geomean), Entire Test Suite 1080p
Nvidia RTX 3090 1920x1080 - PBO = Overclocked AMD
tom's HARDWARE
$550 - Ryzen 9 5900X PBO 179
$539 - Core i9-11900K @ 5.2 175
$800 - Ryzen 9 5950X PBO 173
$450 - Ryzen 7 5800X PBO 173
$300 - Ryzen 5 5600X PBO 172
$550 - Ryzen 9 5900X 172
$800 - Ryzen 9 5950X 172
$490 - Core i9-10900K @ 5.1 170
$539 - Core i9-11900K ABT On 169
$399 - Core i7-11700K @ 5.0 169
$450 - Ryzen 7 5800X 166
$262 - Core i5-11600K @ 5.0 166
$300 - Ryzen 5 5600X 166
$374 - Core i7-10700K @ 5.1 165
$817 - Core i9-10980XE @ 4.8 159
$399 - Core i7-11700K 158
$490 - Core i9-10900K 156
$265 - Core i5-10600K @ 5.0 155
$453 - Core i9-10850K 155
$262 - Core i5-11600K 154
$374 - Core i7-10700K 150
$182 - Core i5-11400 148
$817 - Core i9-10980XE 143
$2999 - Xeon W-3175X 141
$359 - Ryzen 7 5700G PBO 140
$524 - Core i9-9900KS 139
$265 - Core i5-10600K 138
$385 - Core i7-9700K 135
$359 - Ryzen 7 5700G 134
$499 - Core i9-9900K 134
$259 - Ryzen 5 5600G PBO 134
$710 - Ryzen 9 3950X 133
$1999 - TR 3970X 133
$1350 - TR 3960X 132
$259 - Ryzen 5 5600G 132
$460 - Ryzen 7 3800XT 131
$3990 - TR 3990X 131
$455 - Ryzen 9 3900XT 131
$329 - Ryzen 7 3700X 131
$340 - Ryzen 7 3800X 131
$240 - Ryzen 5 3600X PBO 130
$245 - Ryzen 5 3600XT 129
$199 - Ryzen 5 3600 126
$359 - Core i7-8700K 125
$240 - Ryzen 5 3600X 125
$120 - Ryzen 3 3300X 124
$263 - Core i5-9600K 123
$5489 - TR Pro 3995WX 128GB 121
$257 - Core i5-8600K 121
$303 - Core i7-8700 120
$425 - Core i7-8086K 119
$157 - Core i5-9400F 119
$182 - Core i5-8400 117
$122 - Core i3-10100 113
$215 - Ryzen 7 2700X 108
$310 - Ryzen 7 4750G 107
$99 - Ryzen 3 3100 106
$165 - Ryzen 5 2600X 105
$159 - Core i3-9350KF 103
$250 - Ryzen 7 1800X 99
$168 - Core i3-8350KF 99
$122 - Core i3-9100 95
$165 - Ryzen 5 1600X 92
$117 - Core i3-8100 89
$149 - Ryzen 5 3400G 84
$99 - Ryzen 3 3200G 84
$169 - Ryzen 5 2400G 82
$99 - Ryzen 3 2200G 77
$75 - Pentium G5600 69
$64 - Pentium G5400 67
0 20 40 60 80 100 120 140 160 180

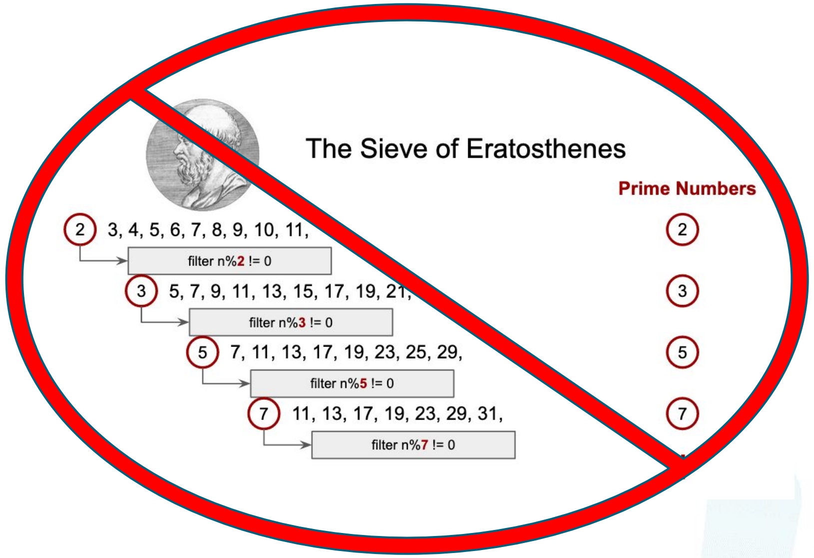
Benchmark should be relevant, realistic, and repeatable



Relevant (for this course):
-CPU (no floating point)
-Memory
~~-I/O~~
~~-Multi-core~~



Repeatable:
-Two runs should give similar results
-You need to think about the methodology

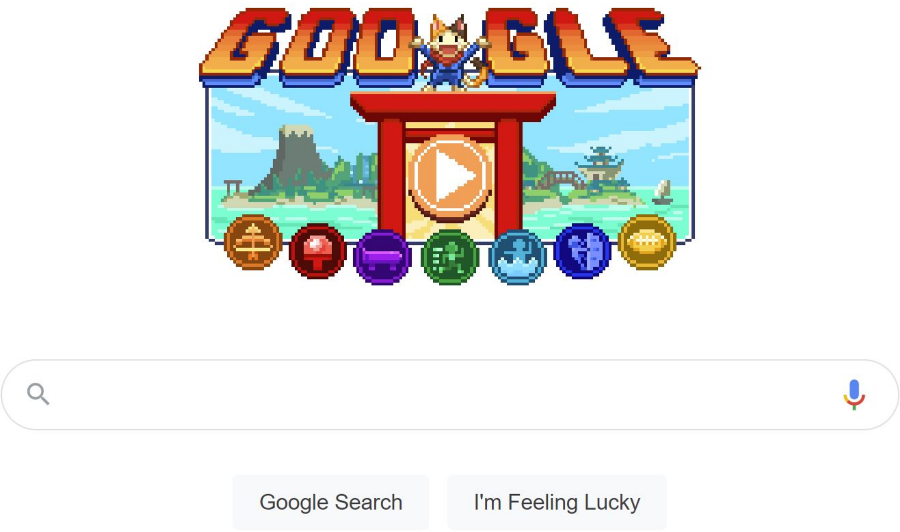


Realistic:
-Not toy example
-Should have some real-world usage
-Also **think of dataset size**

Find something interesting that is open source

and that is:

- CPU bound
- Sequential (not multi-threaded)
- Not floating point



Create a microbenchmark by identifying and re-implementing hotspots in the code

Li..	Source	CPU Time: Total
570	else if (tmax.z < tmax.y) {	0.269s
571	cur = g->cells[voxindex];	
572	while (cur != NULL) {	0.007s
573	if (ry->mbox[cur-<obj->id] != r	2.058s
574	ry->mbox[cur->obj->id] = ry->	0.604s
575	cur->obj->methods->intersect(	0.687s
576	}	
577	cur = cur->next;	0.423s
578	}	
579	curvox.z += step.z;	0.019s
580	if (ry->maxdist < tmax.z || curvo	0.011s
581	cur->obj->methods->union(cur->o	0.612s

Implement a simplified version of this part

How many times is the loop executed?
How big is the data structure?

Also need to setup the data structures, etc

Implement the micro-benchmark in assembly

...and code to setup data structures, check for correctness, etc

Estimate theoretical time for your micro-benchmark

For example: time = instructions x time-per-instruction

How many instructions are executed?

What is the time per instruction on your CPU?

Note! this can become very complicated.

Document your assumptions in your report.

Develop and describe an experiment methodology such that others can reproduce your results

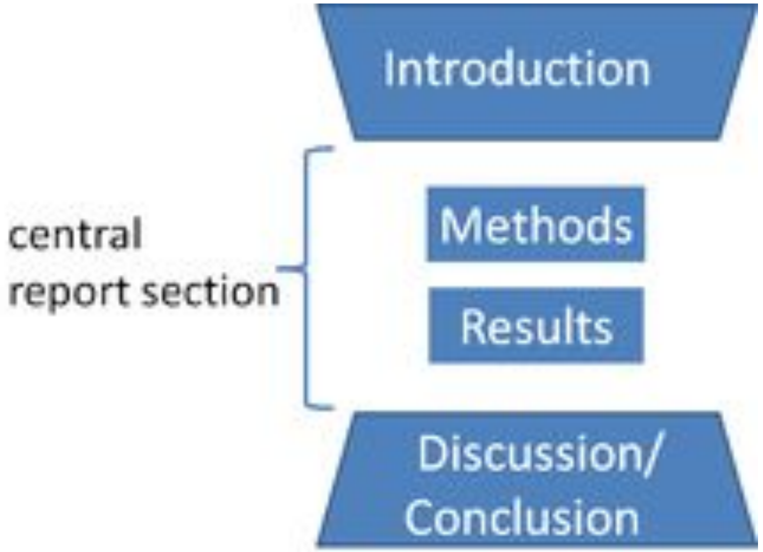
- What are the specifications of the computer you are using?
- What is the state of the computer when you run the experiments?
- How big are your dataset(s)?
- How many iterations?
- How many times do you run the test?
- How do you measure time?
- …

- Tip: you should read how others do and describe such experiments!

Compare the runtime of your assembly code vs gcc compiled C code

- Which one is faster?
  - Why?
  - (do not forget to report variation)
- How does it compare to your theoretical results?
  - If different; why?
- Any other strange results?
  - You may not be able to explain everything you observe at this stage in the course

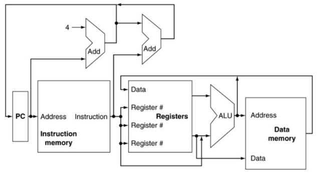
~~Write~~ Produce a good IMRaD report using ChatGPT



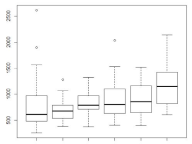
Introduction - WHY?

- Why is your benchmark interesting?
- Who uses it? What is it used for?
- What are the performance requirements for the benchmark?
- What is known about the benchmark performance?
- What does your results show?
- What did you learn from this work?
- ...

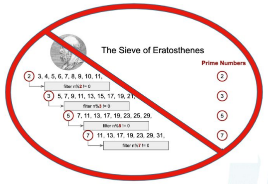
Benchmark should be relevant, realistic, and repeatable



Relevant (for this course):
-CPU (no floating point)
-Memory
-~~I/O~~
-~~Multi-core~~



Repeatable:
-Two runs should give similar results
-You need to think about the methodology



Realistic:
-Not toy example
-Should have some real-world usage
-Also think of dataset size

Methods - HOW?

- Describe your benchmark
- Describe your approach for calculating theoretical performance
- Describe your experimental methodology

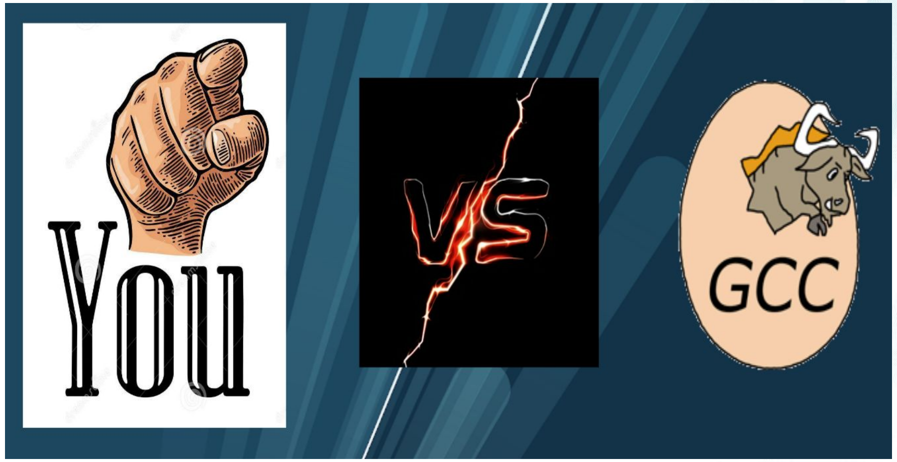
Develop and describe an experiment methodology such that others can reproduce your results

- What are the specifications of the computer you are using?
- What is the state of the computer when you run the experiments?
- How big are your dataset(s)?
- How many iterations?
- How many times do you run the test?
- How do you measure time?
- …

- Tip: you should read how others do and describe such experiments!

Results - WHAT?

- Provide your calculation and measurements



Discussion - SO WHAT?

- What did you learn?

Compare the runtime of your assembly code vs gcc compiled C code

- Which one is faster?
  - Why?
  - (do not forget to report variation)
- How does it compare to your theoretical results?
  - If different; why?
- Any other strange results?
  - You may not be able to explain everything you observe at this stage in the course

Some example papers

https://www.iscaconf.org/isca2023/program/

Use of ChatGPT (or something similar)

- You are required to use ChatGPT when writing the report
  - And also document how you have used ChatGPT
- You are still responsible for the content
  - You need to read, edit, verify, and decide how to use the text in the report
- We have never done this before

LA provide a structure for an IMRaD report

Certainly! The IMRaD format stands for **Introduction, Methods, Results, and Discussion**. This format is primarily used in scientific research papers to present research findings in a clear and structured manner. Here's a breakdown of each section and what it should include:

Title

- Should be concise and informative about the main topic of the study.

Abstract

- A brief summary of the entire study, including the aim, methods used, main results, and conclusion.

---

1. Introduction

- **Background:** Provide context to the study. Discuss the broader issue or problem area.
- **Objective:** Clearly state the research question or hypothesis. Describe what you aim to achieve or prove.
- **Rationale:** Justify why the study is important and relevant.

LA write an introduction for a cpu microbenchmark report

Introduction

In the rapidly evolving world of computer technology, the central processing unit (CPU) remains a cornerstone of computing performance [illegible] manufacturers continue to push the bou[illegible]ies of processing [illegible]es, professionals, researchers, an[illegible]asts alike need to ev[illegible]e real-world performance [illegible]devices beyond just the pro[illegible]ecifications. Microb[illegible]ks, specialized tests desig[illegible]obe individu[illegible]ions of a CPU, offer a granular insight int[illegible]forma[illegible]acteristics and potential bottlenecks of these[illegible] This report presents findings from a series of microbe[illegible] conducted on the latest CPUs in the market. By focusi[illegible] tasks rather than composite benchmarks, w[illegible]provi[illegible]iled understanding of specific performanc[illegible]s, facilitating [illegible]d decisions for hardware purchase[illegible]zations, and advan[illegible] in design methodologies.

This introduction sets the stage by highlighting the importance of the CPU in the realm of computing, the significance of microbenchmarking, and what readers can expect from the report. Adjustments can be made based on the specifics of the microbenchmarks or the targeted audience.

LA write a motivation for using quick sort as a CPU microbenchmark

Motivation for Using Quick Sort as a CPU Microbenchmark

The efficacy and performance of central processing units (CPUs) in real-world scenarios often hinge on their capability to execute common algorithms efficiently. Among these, sorting algorithms play a pivotal role due to their widespread application in various domains, from database management to graphics rendering.

Quick Sort, a widely acclaimed and frequently implemented sorting algorithm, is particularly interesting for several reasons:

1. **Diverse Operations:** Quick Sort's recursive divide-and-conquer approach offers a comprehensive blend of operations — from comparisons and swaps to function calls, which can effectively challenge and hence reflect a CPU's multifaceted performance.
2. **Memory Access Patterns:** The algorithm's access patterns bring out the CPU's cache management proficiency. Effective memory handling during the frequent and sporadic data accesses of Quick Sort is paramount, providing insights into cache hits, misses, and memory latency.
3. **Branch Prediction:** Quick Sort's inherent conditional checks, especially during the partitioning phase, test a CPU's branch prediction capabilities. Efficiently predicting the outcomes of these conditions can substantially influence the overall execution speed.
4. **Optimizations and Variants:** Over the years, several optimized versions of Quick Sort have been introduced. Using it as a benchmark allows for assessing how well a CPU performs under varying algorithmic improvements, from pivot selection strategies to tail call optimizations.
5. **Real-world Relevance:** Given Quick Sort's ubiquitous presence in software, its performance on a CPU can directly translate to

Regenerate

T Certainly! Here's a revised version that breaks down the information into clear steps for better readability:

Microbenchmarking Assignment: Comparing Assembly Coding vs. Compiler Performance

In this assignment, you will assess the effectiveness of your assembly coding skills in comparison to a compiler. Here's a step-by-step breakdown:

1. Choose a Program:

- **Objective:** Develop a microbenchmark.
  - **Definition:** A microbenchmark is a condensed version of a larger program, usually containing just its primary loop.
  - **Criteria:**
    - **Realistic:** Must represent an actual program.
    - **Relevance:** Should stress the CPU. Thus, it needs to be CPU-bound, not I/O-bound.
    - **Consistency:** Should produce the same results in repeated executions.
    - **Simplicity:** For this assignment, opt for a single-threaded microbenchmark, avoiding floating point operations.
    - **Dataset:** Ensure you have, or create, an appropriate dataset for the microbenchmark.

LA how can the above text be misunderstood

T While the revised text provided offers clearer structure and instructions, there are still areas where misunderstandings might arise. Let's point out some potential areas of ambiguity or confusion:

1. **Microbenchmark Definition:**
   - The term "microbenchmark" may still be unfamiliar to some, even with the definition provided. Without examples or more explicit criteria, users might still be unsure about how to identify or create one.
2. **Realistic Criterion:**
   - The requirement for the microbenchmark to be "realistic" and "represent an actual program" is subjective. What constitutes "realistic" can vary widely among different users.
3. **Dataset for the Microbenchmark:**
   - The instruction suggests that users should "ensure they have, or create, an appropriate dataset," but it doesn't guide them on what makes a dataset appropriate or how to create one.
4. **Hotspot Identification:**
   - While the term "hotspot" is defined, some users might not know how to identify it in practice, especially if they're unfamiliar with profilers or performance analysis.
5. **Code Implementation:**
   - The instruction does not specify which assembly language or which compilers to use, which could lead to variations in outcomes or approaches.
   - The term "necessary C-code" is broad, and without specific guidance or examples, users might be uncertain about the

Stop generating

LA provide c code for quicksort benchmarking

T Certainly! Here's a simple QuickSort benchmarking in C:

c Copy code

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function for the QuickSort
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

LA provide realistic data for quicksort when used by a RDBMS

T In a Relational Database Management System (RDBMS), data is organized into tables with rows and columns. When applying a sorting algorithm like QuickSort within an RDBMS, the sorted data will typically involve complex records, not just simple integers or strings. This is a simplification of a more complex scenario where RDBMS might use optimized sort algorithms tailored to their specific needs.

Let's design a simplistic example. Imagine we have a table named `employees` with the following columns:

- `employee_id` (integer, primary key)
- `first_name` (string)
- `last_name` (string)
- `salary` (float)

Here's some sample data for this table:

plaintext Copy code

```
employee_id | first_name | last_name | salary
------------|------------|-----------|-------
1           | John       | Doe       | 55000.50
2           | Jane       | Smith     | 60000.75
3           | Alice      | Johnson   | 49000.00
4           | Bob        | White     | 65000.25
...         | ...        | ...       | ...
```

If we were to apply QuickSort within an RDBMS context, it might be for operations like sorting employees by their `last_name` or `salary`. For benchmarking QuickSort in this context, we'd need:

1. A realistic dataset with many records, possibly using data generation tools or real-world data dumps (sanitized of personal informati[illegible])
2. A sorting criteria or key.

Stop generating

LA write quicksort in assembly

T Writing QuickSort in assembly is quite an involved task due to the low-level nature of assembly language. Implementing QuickSort in assembly will also depend on the specific architecture you're targeting (e.g., x86, ARM).

For the sake of this example, let's consider a pseudo-assembly for the x86 architecture. This wil[illegible] a complete or strictly accurate representation, but it will g[illegible] idea:

assembly Cop[illegible]

```
section .data

section .bss

section .text
    global _start

_start:
    ; Assuming our data is [illegible]ry and we have th[illegible]
    ; rsi points to the [illegible] the array
    ; rdi points to t[illegible]f the array
    call quickSort
    ; ... (other opera[illegible]ions like exit)

quickSort:
    ; Check if low < high
    cmp rsi, rdi
    jge endQuickSort

    ; Partition the array
    mov rdx, rdi  ; Pass the high index to the partition
    call partition
```

Fallacies and pitfalls

1. Using text editors that were state-of-the-art in the 80s
2. Using passive voice
3. Not reading enough textbooks/ papers
4. Repeating what can be read in the textbook or assignment text
5. Starting too late writing the report

Practicalities

- This is a group assignment
- 64-bit Intel assembly (AT&T syntax)
- Benchmark will be used in oblig 3
- Deliverables and report as specified in assignment text