Profiling the carbon footprint of performance bugs

Iztok Fister Jr., Dušan Fister, Vili Podgorelec, Iztok Fister

University of Maribor

ACDSA, February 2024

Presentation agenda

Motivation
Understanding Carbon Footprint
Performance bugs
Experiments and results
Discussion and future work

Motivation

- Identifying the performance bugs in software or inappropriate implementation that can have a great influence toward the carbon emission.
- Investigating the influence of the iteration performance bugs on the increased carbon emission by results of four simulation programs developed in C++ programming language.
- Searching for methods of how to measure the influence of performance bugs.

Contributions of this study

The main contributions of the study are as follows:

- Simulating the performance bugs by four different algorithms written in C++.
- Measuring an increasing carbon footprint caused by the simulation.
- Showing that the proposed measuring methods can be applied to estimate the increased carbon footprint due to the simulated performance bugs.

Understanding Carbon Footprint

- A **carbon footprint** quantifies the amount of greenhouse gases (e.g., carbon dioxide and methane) generated by our activities.
- This measurement is typically expressed in units of CO_2 per unit (CO_2e).
- Virtually everything in the world contributes to carbon emissions, each having its distinct carbon footprint.

Measuring the Carbon Footprint of Computers

- Determining the energy consumption of computers involves both software and hardware tools.
- Existing software tools primarily focus on measuring power consumption in laptops running on a battery.
- However, for desktop or server machines, the current solution involves using an electronic power meter plugged into the mains socket.
- In our study, we evaluated the effectiveness of the **powertop** utility as a representative software tool and the AVHzY ct-3 **power meter** as a hardware tool.

Exploring **powertop** utility

- The **powertop** serves the purpose of analyzing and managing power consumption specifically on laptops operating on battery power.
- The utility provides detailed reports on various aspects, including the estimated discharge rate, and statistics related to processors, devices, kernel, timer, and interrupt handler behavior.
- Additionally, **powertop** allows us to dynamically adjust certain kernel parameters to optimize battery life, providing a practical approach to power management.

AVHzY CT-3 **power meter**

- The AVHzY CT-3 **power meter** measures electrical power in Watts, specifically tailored for computer power assessments.
- Connected to the electricity network via a 230 V plug adapter with a USB port, it measures the connected consumer through an output USB port.
- Notably, the **power meter** features a USB-C output port, allowing direct connection to a personal computer. This facilitates the installation of powerful PC software for simultaneous monitoring of power consumption.

Performance bugs 1/2

- Performance bugs typically increase the time complexity of the algorithm due to coding the inefficient code chunks.
- Obviously, this inefficient coding is programmed by the programmer unintentionally, but has a crucial impact on the performance behavior of the algorithm.

Performance bugs 2/2

- In order to show, how the performance bugs affect the performance of the algorithm, different applications of the vector class in C++ were taken into consideration.
- In line with this, the following four different algorithms were developed:
  - Vector (i.e., the simulation of the performance bug),
  - Raw (i.e., avoiding the performance bug version I),
  - Array (i.e., avoiding the performance bug version II),
  - Double linked List (i.e., avoiding the performance bug version III).

Experiments and results

The purpose of our experimental work was to show what amount of carbon footprint can be expected due to performance bugs. In line with this, the four implemented algorithms (i.e., Vector, Raw, Array and List) [1] were compared according to their carbon footprint, measured on three different computer platforms, i.e.,:

- with software tools on a laptop,
- with a power-meter on a Raspberry Pi 3,
- with a power-meter on an Apple iPad.

[1] `https://codeberg.org/firefly-cpp/green-ict-benchmarks`

Measuring the carbon footprint on a laptop

Algorithm	Init DR [W]	Running DR [W]	Alg. DR [W]	Time [sec]	Energy [Wh]	Carbon footprint [CO_2e/Wh]
Vector	9.288	15.953	6.665	198.496	22.051	14.333
Raw	9.010	16.142	7.132	195.606	23.250	**15.113**
Array	9.532	16.221	6.689	177.381	19.776	12.854
List	7.850	15.138	7.288	16.081	1.953	1.270

Measuring the carbon footprint on a Raspberry Pi

Carbon footprint	Vector	Raw	Array	List
Average inactivity	3.299	3.242	3.273	3.637
Total strain	12.860	63.659	27.496	8.211
Algorithm strain	9.561	60.417	24.223	**4.574**

Measuring the carbon footprint on a Apple iPad

Carbon footprint	Vector	Raw	Array	List
Average inactivity	6.087	6.121	5.808	5.975
Total strain	0.855	1.289	0.637	0.824
Algorithm strain	0.123	0.516	0.317	**0.075**

Discussion

- It turns out, that the vector in C++ is sensitive on performance problem only, when the data class is used a lot of the time.
- Interestingly, sequential usage of the array data structure by the algorithm **Array** was even more efficient than using the built-in functions by the algorithm Vector, when the enormous actions were applied on the class.
- As expected, the algorithm **List**, using the classical implementation of **Double linked list**, outperformed all the other algorithms.

Questions