Mining Software Repository

Mesuring the Power Consumption In C and C++

Mohannad Aljaser
Department of Computer Science and Software Engineering
University of Victoria
Victoria, Canada
Hano@UVic.ca

***Abstract*—With advances in computer programming languages, there has been more pressure on software developers to optimize power consumption, as the amount of energy consumed by computer systems can be lowered through the use of more efficient algorithms and software. Due to the widespread use of C/C++ for application development, we conducted a study to determine which language provides more power efficiency. Our approach focuses on analyzing the consumption of energy by comparing the output of two dictionaries, g++ and gcc (g++ represents the C++ dictionary and the gcc represents the C dictionary). The process of how power consumption was measured, the way results were gathered is described in methodology. Our results showed a significant difference between the two dictionaries based upon energy consumption. Results showed that the C language is lighter and takes fewer watts of energy than does the C++ language.**

Keywords—Power Consumption; C; C++; Watts Up? Pro; Energy.

I. INTRODUCTION

In past, computer manufacturers and software engineer's primary objective was to yield faster computers and software systems, but nowadays this concept has completely changed: the extensive use of powerful computing devices is giving a real tough time to hardware manufacturers and software developers as well. Moreover, in the past much of the software development has been done in C/C++ language. Due to the large amount of developers using these languages, we are going to study the efficiency of these languages by checking the amount of power consumed. For our research purpose, we have a developed a custom code snippet, which is provided with this paper. The C and the C++ versions of our code were run on the Linux machine located at University of Victoria. For accuracy purposes, we ran it several times and recorded the processing time for each iteration during the process, with the help of a dedicated device called Watts up? Pro. After gathering the sufficient amount of data, we then applied data analytics to observe the behavior of program for both languages. The graphs, for illustration purpose, are given in the paper.

II. BACKGROUND

Although software developers recognize the need for more efficient algorithms and software, they lack the tools to pinpoint energy-hungry sections in their code. Therefore, they must rely on their intuition when trying to optimize their code for energy consumption. Every day, new solutions are being proposed for these challenging problems. Giovanni De Michli proposed a solution that focused on algorithm changes, data representation changes and instruction level optimization [1]; Ibrahim and Markus Rupp attempted to lower energy consumption through optimization of the source code and the compiler base [2].

III. PROBLEM STATEMENT

Energy consumption can be regarded as one of the primary constraints for any system, and it matters greatly in the case of battery-powered devices. As upgrades are made in the code of a script/program, the probability of battery drain can either be increased or decreased depending on the change that is made. Our goal in this research is to discover the power consumption of C and C++ by amending our code snippets and analyzing the behavior of efficiency against the power consumption. However, while CPU usage is relatively easy to measure, the same cannot be said of power usage [1]. Measurement of power consumption requires dedicated devices to monitor electrical pulses, which reflect the software's power usage. With the use of the Watts Up? Pro device, our study focused on the following research question:
RQ. What is the most efficient dictionary, between C and C++, regarding energy consumptions?

IV. SOLUTION

In contrast to these studies [1][2], we developed an approach that involved making changes to a single code script and observing the subsequent differences in energy consumption between the C and C++ versions of the code as they were executed. To address these problems, we investigated the impacts of applying code optimization techniques on the power consumption of the algorithm scripts written purely in C/C++. Our proposed solution works in three phases. First, the algorithm was designed with the small level of operations,

some tests were applied, and the changes in the energy usage were noted. In this phase, only the minor changes in the codes were done more explicitly by setting the array. We started by writing a code snippet in C and C++ that created three empty arrays of size 10K, 20K and 30K, respectively. Next, we filled those arrays by generating random numbers in them. Finally, we sorted each of those arrays in ascending order as separate observations. We then repeated the process several times and recorded the consumption measurements, including the time, to an external CSV file.

V. METHODALGY

The methodology used in our research is described as following:

1- Write a code script in C/C++ that sorts the array for different sizes.
- Make empty arrays of different sizes (30K, 20K, 10K).
- Fill those arrays with random numbers.
- Sort the array in ascending order.

2- Run the Linux terminal and terminate all other applications working in the background.
3- Execute the program with gcc and g++ and record the watts using Watts up? Pro.
4- Make changes in the array for different sizes.
5- Compile the gathered data and visualize it to draw conclusions.

After creating the dataset, we collected and analyzed data on the behavior of the program. Our goal was to find the energy consumption in C/C++ coding script on its different optimization level. For this purpose, the code snippet written in C/C++ (Table 1.1) that was executed with changes in the number of elements in the array. We used the same script for C and C++ but compiled and ran them with different dictionaries in Linux [3].

```
int main()
{
   const int _size = 30000;
   double Array[30000];
   //srand(10);
   for (int k = 0; k<_size; k++)
   {
      Array[k] = rand() % 10000;
   }
   int temp;
   int count = 0;
   while (count < 6)
   for (int i = 0; i<_size; i++)
   {
      for (int k = 0; k<_size - 1; k++)
      {
         if (Array[k]>Array[k + 1])
         {
            temp = Array[k + 1];
            Array[k + 1] = Array[k];
            Array[k] = temp;
         }
      }
      count++;
      random_shuffle(std::begin(Array), std::end(Array));
   }

   return 0;
}
```

Table 1.1: This table shows the code snippet used to get our data.

As we used arrays of varying sizes for different optimization levels, Table 1.2 shows how these iterations were made. Different numbers of elements for the array were used to show the consumption of watts on different levels of code to compare both dictionaries on different levels, which means the number of watts consumed by the array varied with the size of the array as it used different numbers of flags (if condition) for each array size.

No. Of Iterations	Level of optimization (based upon the array size)
30 K	Array size is used, to compare both of the dictionaries. This is the more optimized level of code, as it takes the less number of energy watts as compared to the other size of array
20 K	Both dictionaries will be used to execute this program based upon this level of optimization. It's the middle level of optimization.
10 K	This is the un optimized code, which takes the most number of the energy watts.

Table 1.2

We used the GNU Compiler Collection, version 3.0, provided by Linux to run the program on different dictionaries. First, we ran the Linux terminal, which enabled us to compile and execute the code, and calculated the number of watts by terminating everything else running in the background. Watts up? Pro gave the number of watts, 32, when the system was idle and only the Linux terminal was running.

VI. RESULTS

After executing the program with different array sizes, we observed significant changes in the array consumption. For an array of 10K, the watts consumed rose from 32W to 1000W, while the watts consumed by a 20K array reached to 800W and those consumed by a 30K array to 400W. This data showed that the energy consumption depends on the software and motivated us to compare it by using different dictionaries. The reason behind the significant differences in each level of array size is because the number of "if" conditions execute for each level.

Every level of array size (30K, 20K, and 10K) was executed 30 times, and each time Watts up? Pro was used to calculate the changes in the watts. Our scale for comparing the dictionaries in each level was the median value. The median was measured for C++ and C for each level of iteration, and

the data were compared to determine which dictionary takes more watts.
Below are the scatter plots and boxplots for each level of array size, which were further analyzed to identify the dictionary that consumes the most watts.
Below are the scatter plots and boxplots for each level of array sizes, which are further analyzed to get the dictionary which consumes more watts than the other one

Figures 1.2 and 1.3 shows the scatter plot for the code with 30K iterations for the C++, and C (gcc) dictionaries.
.

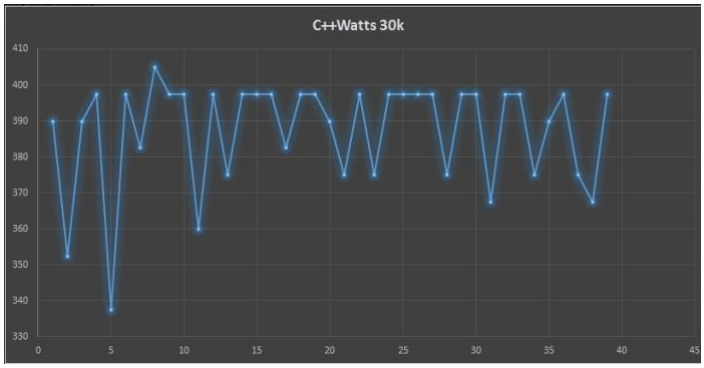


Figure 1.2: The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C++ code for 30K iterations, with the maxima at 405 and minima at 338 watts consumed by the program

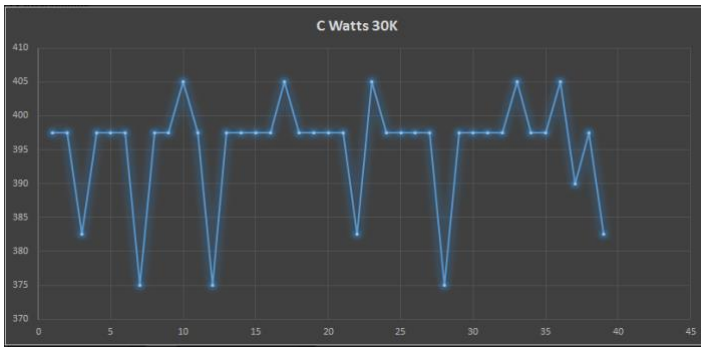


Figure 1.4: The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C code for 30K iterations, with the maxima at 405 and minima at 375 watts consumed by the program

Above are the figures for the C++, and C code with 30K iterations. Below, Figures 1.4 and 1.5 shows the boxplot for the code with 30K iterations for the C++ and C (gcc) dictionaries.

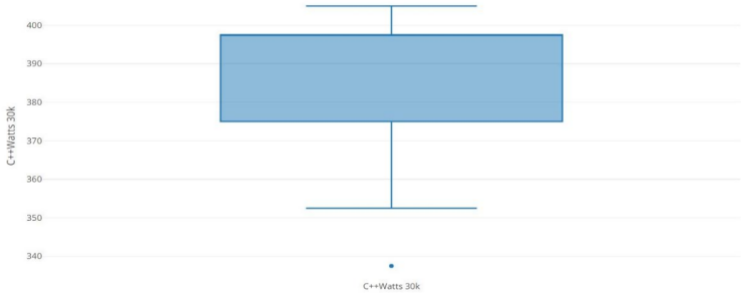


Figure: 1.4: This figure shows the boxplot of consumed watts by the C++ code with 30K iterations.

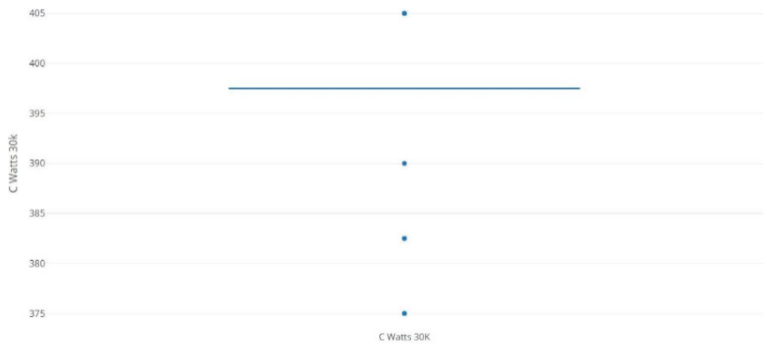


Figure 1.5: This figure shows the boxplot of consumed watts by the C code with 30K iterations.

Analyzing the median value of the C and C++ dictionaries shows that the C++ dictionary had a lower median value (397) as compared to the C dictionary (399). Thus, there was no significant difference between the median values of the languages. To show that the execution time did not affect the energy consumption, we ran the array sizes code at least 60 seconds, which helped us to bring them to the same level concerning time.

Comparing the results for the dictionaries, it was apparent that we needed to work further with more optimized code to get more meaningful results. To further analyze the behavior of the dictionaries, we reduced the number of elements in the array to 20K, but we ran the program for 60 seconds as we did for the 30K iterations. During the whole period, the code ran 30 times, and every time it sorted the array and then shuffled it back until the time is passed. The flag (the if condition in the code) was executed more than it was executed for the 30K iterations. Eventually, it consumed more energy than it did in the previous analysis.

Below, figure 1.6 and 1.7 shows the scatter plot for the code with 20K iterations for the C++, and C (gcc) dictionaries.

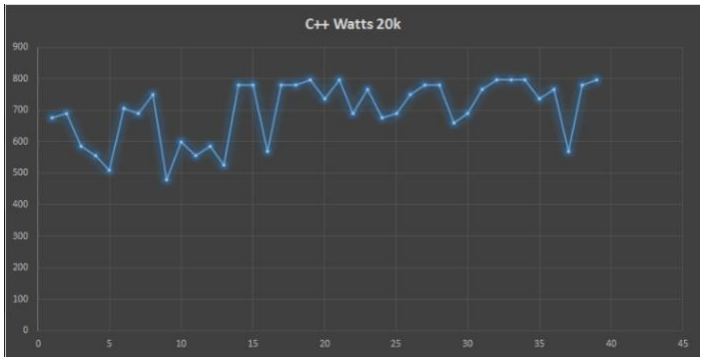
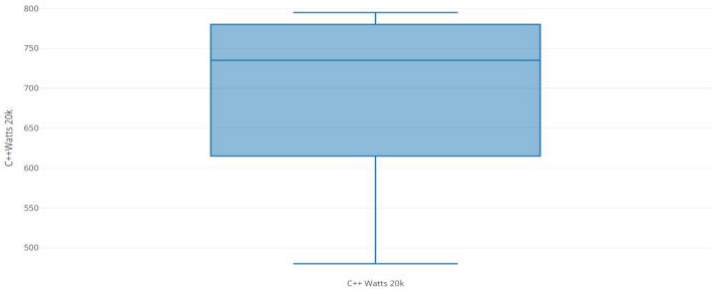


Figure 1.6: The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C++ code for 20K iterations, with the maxima at 800 and minima at 490 watts consumed by the program.

Figure 1.7: The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C code for 20K iterations, with the maxima at 800 and minima at 490 watts consumed by the program..

To analyze the gcc dictionary behavior on the same code, we scattered the data and drew the boxplot for the consumed watts to analyze it.

Below, Figures 1.8 and 1.9 shows the boxplot for the code with 20K iterations for the C++ and C (gcc) dictionaries.



This figure shows the boxplot of consumed watts by the C++ code with 20K iterations.

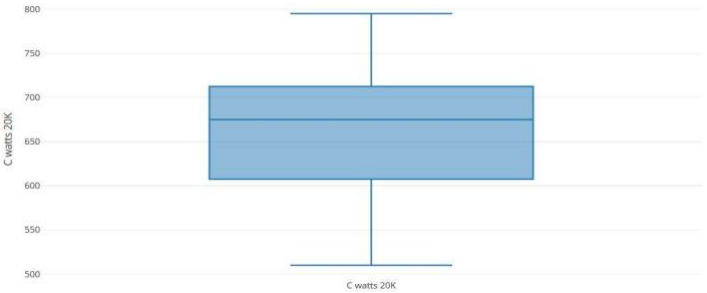


Figure 1.9: This figure shows the boxplot of consumed watts by the C code with 20K iterations.

Analyzing the median value of the C and C++ dictionaries, the C++ dictionary had a lower median value (735) as compared to the C dictionary (675). The C++ dictionary consumed more watts than did the C. Before jumping to any conclusions, we decreased the size of the array to 10K, which required more watts than both optimizations as it also executed for one more time.

Figures 2.1 and 2.2 show the scatter plot for the code with 10K iterations for the C++ and C (gcc) dictionaries.

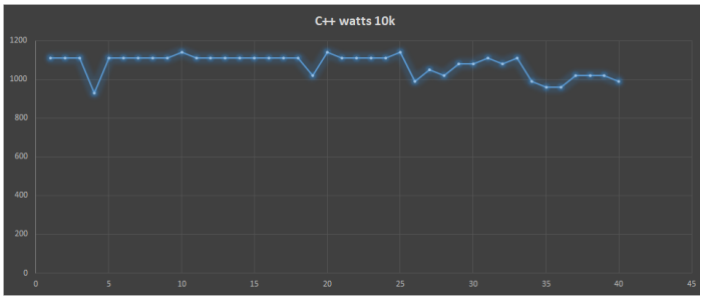


Figure 2.1: The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C++ code for 10K iterations, with the maxima at 1140 and minima at 890 watts consumed by the program.

To compare it with the C 10K watts we will analyze the behavior of it as well.

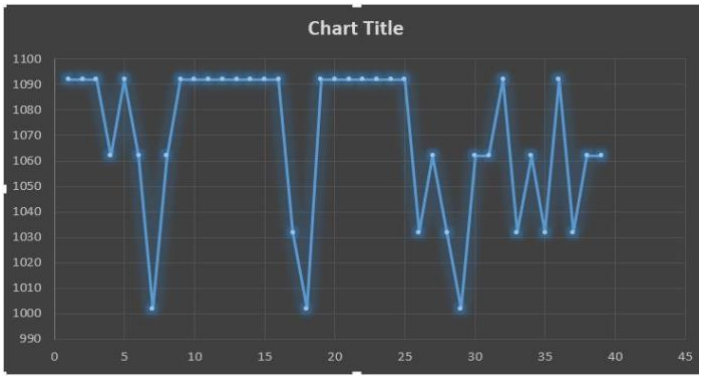


Figure 2.2 The x-axis shows the number of iterations and y is for the number of watts for each iteration. This figure shows the scatter plot of the calculated watts for the C code for 10K iterations, with the maxima at 1140 and minima at 1050 watts consumed by the program.

Figures 2.3 and 2.4 show the box plot for the code with 10K iterations for the C++ and C dictionaries

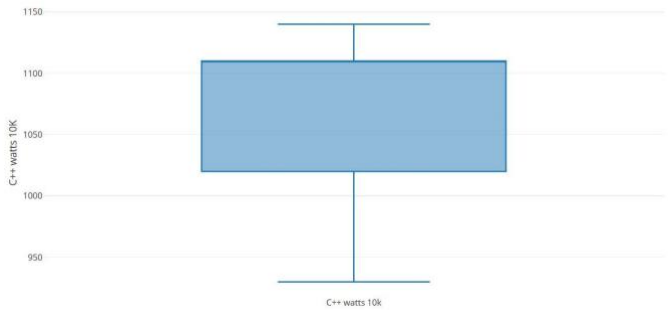


Figure 2.3: This figure shows the boxplot of consumed watts by the C++ code with 10K iterations.

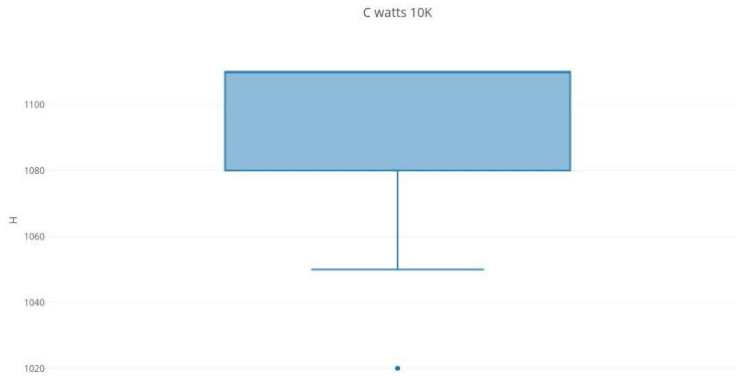


Figure 2.4: This figure shows the boxplot of consumed watts by the C code with 10K iterations.

Analyzing the median value of the C and C++ dictionaries, C++ has a lower median value (1110) as compared to the C dictionary (792), which means again we have C as the more efficient regarding energy consumption

VII. DISCUSSION

Although it is the hardware that consumes energy, the software can have a significant influence on energy consumption, just as a driver who operates a car influences its fuel consumption. Our methodology can be a good complement for C/C++ programmers who seek to reduce the power consumption of their programs. Developers should take into account the power consumption of both languages when working on an enterprise scale. Other developers will be able to leverage our study to carefully select their development language if they are concerned about power usage.

VIII. CONCLUSION

In this study, we proposed a methodology for comparing the dictionaries of two programming languages, C and C++, to find out if C or C++ is more efficient in regard to power consumption. Using the given code snippet in Table 1.1 and following the given methodology, anyone can easily produce the data and compare the results. In our study, we observed the effects of the custom-made script and compared the results of one with the other. We found that C++ consumes slightly more energy than C, although the differences were not significant. Based on our results, we can say that the difference may be negligible in small programs; however, in the case of large-scale projects used in the enterprise sector,

power consumption matters significantly. The longer it takes a function to perform, the higher its energy consumption.

ACKNOWLEDGMENT

I would like to express my most profound appreciation to all those who provided me the opportunity to complete this report. A special gratitude to Dr. Daniel German, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project notably in writing this report.

REFERENCES

[1] T. Simunic, L. Benini, G. De Micheli and M. Hans, "Source code optimization and profiling of energy consumption in embedded systems", *ACM*, 2000. [Online]. Available: https://dl.acm.org/citation.cfm?id=501831. [Accessed: 12- Dec- 2017].

[2] M. E.A. Ibrahim and M. Rupp, "Embedded Systems Code Optimization and Power Consumption", *Publik.tuwien.ac.at*, 2013. [Online]. Available: https://publik.tuwien.ac.at/files/PubDat_217753.pdf. [Accessed: 12- Dec- 2017].

[3] "G++ and GCC - Using the GNU Compiler Collection (GCC)", *Gcc.gnu.org*, 2017. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-6.2.0/gcc/G_002b_002b-and-GCC.html. [Accessed: 17- Dec- 2017].