Energy Footprint report

Calvin Roth

December 2022

Abstract

This work investigates the energy requirements of optimization algorithms. Power usage is a becoming a real budgetary constraint to high performance large computing. This project demostrates that the simplex algorithm and the interior point method in addition to the differences in practical runtime also vary in how much energy they use per unit time.

1 Introduction

The power consumption of complex computer systems poses a problem for designers and maintains of this systems. The power usage of this large systems should raise the concern of scientists who wish to be environmentally green when possible but also more pragmatically these large energy draws can prove to be a real constraint due to the heat they generate and the budget they demand.

To see just how much power is needed for these system we use the website top500[12] which compiles a monthly list of supercomputers ranked by how many petaflops/s they are capable of on linear algebra systems, conjugate gradient problems, and also how much wattage they use. For example, at number 1 Oakridge national lab has a system,Frontier - HPE Cray, that is capable of 1,102 petaflops/s on the linear algebra problem set. But it gobbles electricity at the rate of 21 mega Watts[12]. To get an idea for how much power this is reporting suggests that 1 megawatt/hour will sustain 400-900 households' needs/hour[2]. So running this Oakridge system at maximum power is an equivalent power need to powering approximately 8000-19000 households. This is not a negligible operating cost. Especially with the raise of machine learning models proving to be important for national labs and commercial companies concerns about the energy needs of large scale computation is a relevant set of problems.

This leads to some natural questions for the programmers. Is there anything they can reduce the strain on system by modifying their code. There are a lot of natural questions to ask such as how does parallelism impact power, could we downclock a system to reduce the overall load but have longer to wait for an answer, what are the differences in using cpus and gpus, and many more. The question of interest here is does the choice of algorithm make an impact? Specifically, this work will look at power draw per second. This makes comparisons between different algorithms with different runtimes more interesting. For a large enough problem, quick sort should use less electricity than bubble sort because it will finish much faster. But it is not obvious which will use more power per unit time. In fact, it seems counter intuitive to suggest they should be different or that any two algorithms ran on the same math utilizing parallelism in similar ways should be different because the view of the computer as "uniform" worker seems natural.

2 Related Work

Previous work has actually shown this is a gap that exists. Much of the literature about energy efficiency is concerned with performance on GPUs, FPGAs, and/or distributed systems. This paper is only concerned with the GPU performance only and only the parts that can be extrapolated to a single cpu as that is what this project tests with. Of particular inspiration was the project TEE-ACM2 showed that matrix chain multiplication algorithms does not have uniform power demands to such an extent that the optimal matrix chain multiplication while faster draws more power than the a specialized algorithm they made to optimize power[9]. The core thesis of their work is that not all operations demand the same power. In particular, floating point operations consume significantly less power than operations involving the memory on the computer such as loading and storing storing to such an extent at it's core their algorithm attempts to minimize the number of these operations.

Memory performance is a fairly well known factor in energy footprint. In High-performance Energy-efficient Recursive Dynamic Programming with Matrix-multiplication-like Flexible Kernels, Tithi et. al.[**tithi2015high**] were interested in optimizing dynamic programming problems mention this phenomenon. The machine learning community[4][3] seems to be independently aware that "the energy-hungry data accesses to cache introduce high energy consumption"[13]

The conjecture based on this previous work is that the relative power cost of algorithms is predicted by their cache behavior. In this paper we test this theory on the interior point method and the simplex method.

3 Testing Framework

In order to test the electricity usage of different algorithms in a reliable way we need to do a few things. Tests should be conducted on medium to large problems and these problems and solutions should be replicatible. Next, we need a way to measure the power usage of a problem and only the part of the problem related to solving the problem not auxillary tasks like creating the constraint matrix. Finally, sophisticated implementations of algorithms should be used instead of homegrown implementations. This last point is because there is little point in comparing naive unoptimized variants of the algorithms. Each of these problems is addressed in sequence below.

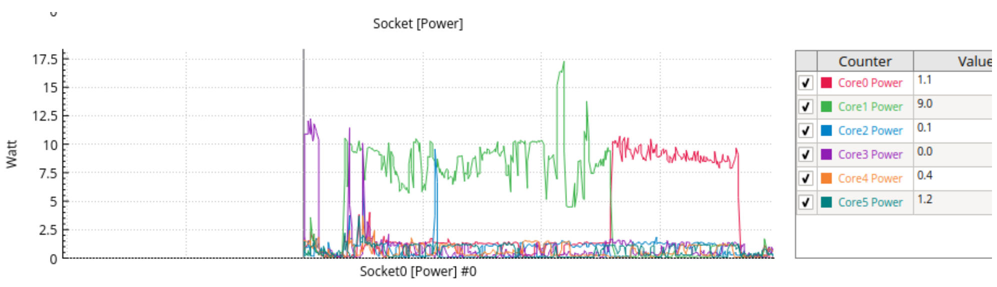
For this set of experiments we do the computation in a Julia language notebook. This has an advantage over doing it as a single program because we separate actions we want into cells. For instance model create and optimization are easily separated. This is important because unlike timing of functions there is not an option to start monitoring energy usage in a program. The monitoring of energy use is provided by a different external program.

Monitoring the energy is a logistically annoying challenge of finding the right tools. All the tools to the best of my knowledge depend on the architecture and system you are using. Specifically, one has Intel VTune[10] for intel cpus, AMDμProf [1] for AMD products, and the NVIDIA toolkit for their gpus[5]. There does not seem to be a universal way to do this. My computer has an AMD cpu so we will be using the AMDμProf tool. The main hypothesis of this work is that one way to explain differences in algorithms is by differences in cache performance. So we need a tool to monitor cache misses as well. This is actually fairly easy to accomplish. We can use the Perf linux tool and call it in the program to get statistics of interest.

The solver used in this experiment is the JuMP(Julia Mathematical Programming)[6] interference with the HiGHS solver[8] This setup allows an setup of the problem and configuration of algorithm parameters. Many parameters such as the maximum number of iterations or tolerance requested are not relevant here but important one is the option to force it to use a particular solver between the choice of the Dual Simplex method or the Interior Point Method.

3.1 A note on parallelism

Right now, parallel implementations are not being tested. There is a parallel solver in the HiGHS package but it seems to be getting quite poor performance in terms of speed. This raises the concern that it isn't a good implementation and therefore not interesting enough to report on. I hope to investigate other solvers and look for one that lets me set the method used or is at least clear what algorithm it uses, gives me control over the parallelism, and works well. If I'm lucky it will be capable of utilizing the GPU on my pc.



		Counter	Value
✓	🟥	Core0 Power	1.1
✓	🟩	Core1 Power	9.0
✓	🟦	Core2 Power	0.1
✓	🟪	Core3 Power	0.0
✓	🟧	Core4 Power	0.4
✓	🟦	Core5 Power	1.2

Figure 1: Power consumption over time for the Simplex solving at 8000x8000 system

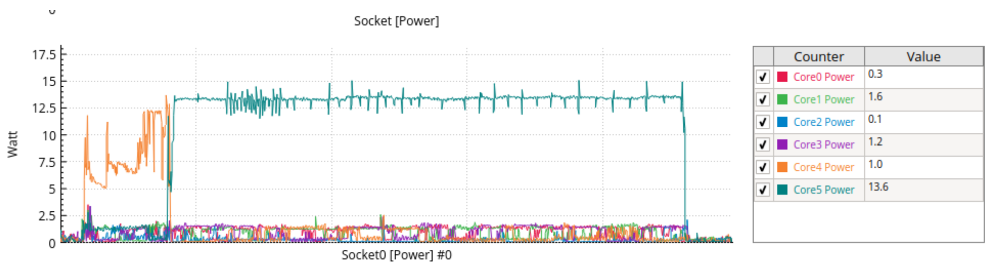
Qi Huangfu presents a good survey of implementations both parallel and serial of the simplex method[7]. Parallel implementations of the simplex method on dense data are relatively boring in the sense that algorithm doesn't really change just the operations are performed in parallel. Their work devotes a lot of time to parallel simplex method on sparse problems which has more novel options available to maximize performance.

4 Results

All tests were done using an AMD Ryzen 5 5600X 6-Core Processor. Each cpu is equip with an L1-cache of 192 kilobytes. To test the behavior of the Simplex and Interior Point method both an ran on medium sized linear programming problem. This Problem will be of the form

In particular, A, b,c are randomly filled with values from 0 to 1. This already implies that the problem is feasible because $x = 0$ will be a solution. For this work, the problem size was chosen to be 8000 variables and 8000 constraints. Furthermore, although randomly populated it is random in a set way based on a seed. That is both methods are solving the exact same problem data wise.

Notice for both there is one one core that is doing significantly more work that others but that others are doing a none zero amount of work. This small amount of work corresponds to background processes of the system, when these tests were ran care was taken to have the minimal set of applications open simply visual studio code and the AMDμProf program. Detailed results were collected and are available on github. This work will just address



	Counter	Value
✓	🟥 Core0 Power	0.3
✓	🟩 Core1 Power	1.6
✓	🟦 Core2 Power	0.1
✓	🟪 Core3 Power	1.2
✓	🟧 Core4 Power	1.0
✓	🟩 Core5 Power	13.6

Figure 2: Power consumption over time for the Interior Point method solving an 8000x8000 system.

the key insights from the data. Visually, we see that the Simplex method uses somewhere between 7.5 and 10 Watts in a typical moment this is in contrast to the interior point method which uses a very consistently around 13.5 watts. The author is unable to explain the higher variance seen on the simplex method especially the short duration jump to nearly 17.5 watts. Notice also that it is not always the same core doing the computation. Occasionally, a switch is made as seen by a difference in color. This is a decision that the operating system makes but as each core is identical should have not impact on the validity of the results. This analysis is visual but AMDμProf is perfectly capable of reporting the results in the form a CSV.

The simplex method took 1428 iterations and an the interior point method uses 28 iterations but each is significantly slower than 1 simplex iteration.

5 The Cache Behavior

This leaves a mystery yet to be solved. That being why should the interior point algorithm be more costly in terms of kilowatt hours than the simplex method? This section will attempt to give a reasonable hypothesis that could be tested in future work. The hypothesis that I had starting this project was that differences in cache behavior can induce differences in energy cost. The reason this hypothesis is reasonable is based on work concerning other algorithms. Specifically, the TEE-ACM2(Transformations for Energy Efficient Accelerated Chain Matrix Multiplication)[9] found that in the problem of matrix chain multiplication that it was storage operations such as loading and storing variables are very costly.

So what differences do we see in memory usage between these problems. The perf linux command provides a means to get cache statistics of the interior point method and the simplex method. The two metrics of prominence will be the number of loads of data into the L1 cache and the number of L1 data load cache misses both per unit time. We have to do this comparison per unit time because the meaning of raw number of cache loads is meaningless if one algorithm is quicker than the other.

Below are the results from this experiments on a linear programming problem with 8000 constraints and 8000 variables.

Optimizer	runtime	Loads(Mil)	Load Misses(Mil)	Loads/sec	% Load Misses
Simplex	16.44	70,611	5,010	4295	7.10
IPM	102	608981	53413	5970	8.77

We see that the the number of memory loads a second and % of the time we have a cache miss is consistent with the hypothesis that more memory operations is related to power usage of a program. It seems insufficient without further testing to conclude the cahce behavior is the cause of power usage. Two conceivable ways to increase the confidence that there is a causal relationship between is to test any number of reasonable hypotheses. Another way would be to alter either of the interior point solver's or simplex method's base code to use cache differently. If the hypothesis is truly right an algorithm that makes more economical use of cache should use less electricity. Ideally, the change is an improvement that uses data in smarter because we would like a better algorithm in addition to knowing this relationship between energy and memory. But if one is satisfied with just the relationship it should be easier to make a deliberately poor use of memory to further test this theory.

6 Conclusion

In this work it was shown that for the Simplex method and the Interior Point differ in the way they use electricity. In particular, consistent with the hypothesis set forward it seems that the reason why they differ is how they use memory as measured by L1-Cache loads. This raises interesting questions such as what happens when we look at the same problem on GPUs and can we find a way to implement the Simplex Method and/or the interior point method to use memory and theefore power in a more efficient way.

References

[1] *AMDuPROF*. Dec. 2022. URL: `https://developer.amd.com/amd-uprof/#userguide`.

[2] Bob Bellemare. *What is a megawatt*. 2012.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 367–379.

[4] Tianshi Chen et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning". In: *ACM SIGARCH Computer Architecture News* 42.1 (2014), pp. 269–284.

[5] *Cuda Toolkit 11.7 downloads*. Oct. 2022. URL: `https://developer.nvidia.com/cuda-downloads`.

[6] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320. DOI: `10.1137/15M1020575`.

[7] Qi Huangfu. "High performance simplex solver". In: (2013).

[8] Qi Huangfu and JA Julian Hall. "Parallelizing the dual revised simplex method". In: *Mathematical Programming Computation* 10.1 (2018), pp. 119–142.

[9] Maxim Moraru et al. "Transformations for Energy Efficient Accelerated Chain Matrix Multiplication (TEE-ACM 2)". In: *Supercomputing*. 2022.

[10] James Reinders. *VTune performance analyzer essentials*. Vol. 9. Intel Press Santa Clara, 2005.

[11] Jesmin Jahan Tithi et al. “High-performance energy-efficient recursive dynamic programming with matrix-multiplication-like flexible kernels”. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 303–312.

[12] *top500*. URL: `https://www.top500.org/`.

[13] Mingyu Yan et al. “Hygcn: A gcn accelerator with hybrid architecture”. In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2020, pp. 15–29.