Data Movement Complexity vs Time Complexity: a Case Study on Algorithmic Analyses as Predictors of Performance

Abstract:

The rapid advancements in computer science and technology over the past few decades have given a rise to a paradigm shift in computing: it is no longer seen as mere calculation, but also the processing of large volumes of data whether it be through large databases, massive machine learning models. Thus new theoretical foundations are being laid to accommodate this shift, which necessitates the need for evaluating them. This work will look to explore the soundness of Data Movement Complexity, a memory aware framework for algorithm analysis, by seeing if it can be used as a predictor for performance of algorithms on real machines.

Introduction:

We currently live in a technology and data driven world. Innovations over the past few decades such as the internet, machine learning, and cryptocurrency have necessitated the widespread use of technology in the form of computational power and data storage. Likewise, some of our most costly expenditures stem from these two things. However, compared to other scientific domains, computer science and software engineering are relatively immature fields and have not even been around for a century. Likewise a lot of the theoretical foundations which were built early on, while still important, have lost a lot of their relevance and applicability. A great example of this is time complexity, an analytical method to measure how many “steps” or operations an algorithm requires. This measure, for a lot of computing’s history, has been able to reflect reality as counting the number of operations in an algorithm can be mapped to how many steps it takes a processor to execute it. Although computing systems employ a wide range of engineering techniques to improve performance, such as pipelining and instruction reordering, it did not fundamentally change what the framework was looking to measure.

With all that said, the current workloads computer systems and applications encounter are now data driven. And in correspondence to this, time complexity is becoming more and more out of touch with reality, as it does not take into account the cost of moving memory/data to and from the processor to actually perform some computation. A recently developed framework for algorithm analysis, Data Movement Complexity, proposed by Smith et al attempts to rectify this problem. The model builds on top of time complexity by assigning each access to a memory location a cost based on its location in an abstract memory hierarchy [1]. The result is a form of analysis, which, similar to time complexity, is portable across machines and can derive insights at an algorithmic level [1]. That being said, unlike time complexity, the measure of Data Movement Complexity is completely abstract [1]. As mentioned previously, time complexity can be thought of as counting operations. On the other hand, the cost Data Movement Complexity assigned to each operation is not strictly defined by some actual machine, but is an abstract cost called reuse distance [1]. Reuse distance essentially measures how many unique accesses occurred in between a reuse of some data [2]. Although a distribution of reuse distances is convertible to miss ratio curve (cache performance), it is only one factor in program performance, and Data Movement Complexity is only a partial reuse distance distribution [1]. Moreover, Data Movement Complexity amortizes the differences in reuse distances by taking the square root of them when associating cost [1]. And so that begs the following questions:

RQ1: Does Data Movement Complexity have any correlation with performance of algorithms on real machines, either in terms of runtime or the energy consumption?

RQ2: Is Data Movement Complexity a better predictor of runtime than time complexity?

Since Data Movement Complexity is characterizing the cost of moving data around, if the framework is sound, it logically follows that Data Movement Complexity should be able to predict performance and/or energy cost to some degree. Before the rest of the outline of the plan moves further, it should be noted that operations and data movement are not completely orthogonal measures. That is, the number of operations will typically influence the number of data accesses. This reasoning leads to the following hypotheses:

H1: There is a moderate, positive correlation between Data Movement Complexity and runtime, but a stronger correlation with respect to energy consumption.

H2: The correlation between Data Movement Complexity and runtime will be stronger than the correlation between time complexity and runtime.

With that established, and combining a lot of the points mentioned previously, it makes intuitive sense that there would be a correlation with both respect to energy and performance: Data Movement Complexity is essentially making up for the gaps in time complexity with a cost model [1]. The engineering tricks which software and hardware employ also have a lot to do with prefetching and accessing data in parallel, which would increase performance in terms of time, but would not hide energy consumption. Of course, this reasoning hinges on the idea that reuse distance is both a good metric for measuring cache performance in general and a valid metric for measuring data movement costs. Ding and Zhong demonstrate that profiling of reused distances can predict program locality (efficiency of data caching) [7]. Similarly Arafa et Al showed that reuse distance analysis can predict cache performance on GPUs with accuracy rates ranging from 85% to 95% [4]. With all that said, it is also hard to equate reuse distance with cache performance in modern computers with multi-chip and multi-threaded processes [3]. It is usually not the case that a singular program is what is running on a computer, usually lots of processes are running concurrently both in the background and foreground. Despite these fallbacks, Jiang et al demonstrate that these losses in applicability can be recuperated by using probabilistic models to characterize the behavior of multi-chip processors [3]. Hence why I posit a correlation with respect to speed, but a stronger one with respect to energy cost. This work will not be concerned with exactly how to extend Data Movement Complexity, but rather experiment with and observe the merits in its current form.

Methods:

In order to explore this question and hypothesis, the plan is to first replicate the results presented in the original paper and then move on to comparison with algorithm implementations on several machines. And then the next step will be to establish the correlation, or lack thereof, between Data Movement Complexity and performance (with respect to both energy and speed). Luckily, the proofs used employ empirical validation of reuse distances, and so I will recreate the observed reuse distances of the algorithms analyzed and see if they match up with what is presented in the paper.

For the experimentation, one consideration is which programming language will be used. The plan is to instrument the algorithms presented in the paper — naive matrix multiplication, Strassen's algorithm, and recursive matrix multiplication — using C. The reasoning for this is that the algorithms used do not have any abstractions, so it is best to use low level programming languages such as C. This is because there is less overhead performance overhead for memory management in comparison to popular languages with automatic memory management (garbage collection) such as Java or Python. A language with garbage collection would also add variation in the experiment itself, as the programmer has no control over when it happens.

Since the data being sampled is energy consumption and runtime, the implementations of the aforementioned algorithms will be run on several machines. In order for the results to be generally representative, specific machines will not be cherry picked. Instead, I will be randomly sampling (without replacement) from the set of 100 best selling laptops on Amazon, Ebay, and Bestbuy. It would not make much sense to sample from the entire space of all laptops ever created, and this is a simple way to filter out machines which are irrelevant due either being old or not used by many. I say "set" because there is likely lots of overlap on these lists, and so laptops which appear multiple times will only be in the sample space once. Thus the population is the performance of these algorithms on popular laptops. 10 laptops from this population will be picked, and since this is at most 10 percent of the population size, the sample will be random. For each laptop the option with the most RAM and HDD will be picked to avoid memory bottlenecks as much as possible and to allow for more data collection. The choice of using laptops is mostly for budget constraints, and there is no reason that the correlation should be significantly different between laptops and desktops. An empirical validation of this intuition will be left for future work. Lots of people also make customized desktops, and so sampling from pre-built desktops would not be very representative, and on the other hand, building lots of desktops would create more variation in addition to taking more time. One concern is what will be done for the operating system, and if the system comes with Windows or macOS, then we will keep it that way, otherwise we will install Fedora Linux 36. Again this creates some variation across machines, but we are not too concerned with cross machine comparisons, and instead it is satisfactory if the machines themselves are sound (and if they are bestsellers then they are not likely to have problems which hinder simply running a program).

As for data collection, measuring execution time is simple and can be measured with the C library ‘time.h’. Measuring energy consumption will be a lot trickier, especially when considering multiple OS platforms, different machine hardwares, etc. Lots of the literature is concerned with reducing energy consumption at scale, rather than for individual programs or machines. However specific tooling does exist, especially for low level languages such as C: Santos et al present a tool which utilizes LLVM tooling and AST analysis to automatically instrument energy consumption [6], and this will be used for the experiment. This methodology is convenient and sound in the context of this experiment because usually energy profiling requires work on both the side of software and hardware [5]; By using this tooling and a low level language to accommodate it, measurement variation in terms of modifying hardware directly is avoided. On top of this, all laptops will not use an IDE for the code, and will be run through the terminal with no other applications open. As for the actual input size of the experimentation, in order to get as much data as possible each algorithm on each laptop will increase input size (by varying steps depending on the algorithm) until the laptop is unable to compute some input size, or until it takes a week long to compute it.

Analysis and Discussion:

Building on the expectations set in the hypotheses, it is expected that the correlations will vary noticeably, but perhaps not significantly, across the machines. And for all machines we will plot time complexity versus runtime, time complexity versus energy consumption, Data Movement Complexity versus runtime and Data Movement Complexity versus energy consumption, as well as make one plot with all the points. Because a “perfect” predictor of performance or energy consumption would have a 1:1 mapping between the prediction and reality, we can use linear regression to measure correlation. I also will plot the factors of differences between x and y at each point and compare the rates of growth of the factors between the time complexity and Data Movement Complexity plots. Correlations can be somewhat sensitive given the data collected and so while it is expected that one will exist, I believe that the comparison between the rates of growth of factor differences will be much more compelling. I believe the factors of differences, for both energy and speed, will grow slower for Data Movement Complexity in comparison to time complexity.

This statistical result would suggest Data Movement Complexity is a better predictor of performance with respect to both energy consumption and runtime. Likewise, if this conjecture is true, then this would confirm our hypotheses for general commodity laptops, and further exploration would be needed to confirm the hypotheses against heavy duty hardware, supercomputers, and computing on large clusters. If the results are not as expected, this would call for a further investigation into explaining the performances on each laptop and why it does not correlate to Data Movement Complexity, as well as experimental replication to further confirm the result. It would not however undermine the framework for analysis entirely, similar to how time complexity is still widely used in the computer science community, as it is still designed to derive insights at an algorithmic level. Both time complexity and Data Movement Complexity do claim to be purely performance predictors, it is just that the former posits that better performance prediction could be a possible benefit. Although this methodology would eliminate experimental variation, not all of it is gone. This is because we can't control things like OS context switching and differences in hardware design across the machines which Data Movement Complexity does not take into account. This emphasizes the need for replication by other groups and further investigation on other forms of computing

Budget and Planning:

Since data collection does not involve tampering with hardware, it is completely safe and there is no need to keep it confidential and will be published along with the other results. Moreover because we only choose 10 laptops, we will not be drying out the market for the sake of this research. As for the cost, the laptops in each of the lists mentioned range from 200 to 2500 dollars, with most in the mid thousands. As the cost depends on the sample, nothing is entirely certain, although a budget of 22,000 dollars would be extremely safe (to also account for the power bill). The other parts of the research have no cost at all.

In terms of time, the actual instrumentation should not take too long, as the tooling which will be used is automatic and the same programs will be used across machines. The data collection can also be done in parallel, and so I expect that it will take 4 weeks. The budget will be used entirely at the beginning when laptops are ordered. The only potential bottleneck will be the replication of the results in the original paper, which I expect to take 2 weeks if it goes smoothly. Lots of these components can be done concurrently, on a rolling basis as the laptops come in and so the total expected time for the research is 2 months, including synthesization and analysis of the results.

References

[1] W. Smith, A. Goldfarb, and C. Ding. Beyond time complexity: data movement complexity analysis for matrix multiplication. In Rauchwerger, L., Cameron, K. W., Nikolopoulos, D. S., and Pnevmatikatos, D. N. (eds.), Proceedings of the International Conference on Supercomputing, pp. 32:1–32:12. ACM, 2022.

[2] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation techniques for storage hierarchies," in IBM Systems Journal, vol. 9, no. 2, pp. 78-117, 1970, doi: 10.1147/sj.92.0078.

[3] Y. Jiang, E.Z. Zhang, K. Tian, X. Shen. (2010). Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors?. In: Gupta, R. (eds) Compiler Construction. CC 2010. Lecture Notes in Computer Science, vol 6011. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-11970-5_15

[4] Y. Arafa, G. Chennupati, A. Barai, A. -H. A. Badawy, N. Santhi and S. Eidenbenz, "GPUs Cache Performance Estimation using Reuse Distance Analysis," 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019, pp. 1-8, doi: 10.1109/IPCCC47392.2019.8958760.

[5] R. Ge, X. Feng, S. Song, H. -C. Chang, D. Li and K. W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," in IEEE Transactions on

[6] M. Santos, J. Saraiva, Z. Porkoláb, & D. Krupp. (2017). Energy Consumption Measurement of C/C++ Programs Using Clang Tooling. SQAMIA.

[7] C. Ding and Y. Zhong. 2003. Predicting whole-program locality through reuse distance analysis. SIGPLAN Not. 38, 5 (May 2003), 245–257. https://doi.org/10.1145/780822.781159