Rethinking RAM: Testing alternative models of computation

David Lachut
dlachut1@umbc.edu

Kaustav Lahiri
klahiri1@umbc.edu

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County

15 May 2013

Abstract

Modern computers do not access memory in constant time, though the random access machine model of computation assumes they do. A new model of computation that reflects the reality of expensive memory access might allow more accurate algorithm analysis. This paper experimentally compares the random access machine to the new virtual address translation model and concludes that the random access machine is a more accurate, more usable tool for analyzing algorithms.

Keywords RAM Model, computational models, benchmarks, algorithms

1 Introduction

The Random Access Machine (RAM) model of computation is widely used for analyzing the performace of algorithms. Some, such as Jurkiewicz and Mehlhorn [5] question whether this model accurately represents the complexities of modern hardware. They propose the Virtual Address Translation (VAT) model of computation to account for added complexities, such as the memory address translation needed to refference ever larger amounts of system memory in modern computers.

This paper analyzes a set of algorithms using both models, VAT and RAM, and presents the results of benchmarking these algorithms on real hardware. The goal of this is to then use the results of the analysis and benchmarks to determine which model more accurately reflects the real hardware.

The novel contribution includes not only the benchmarking of these procedures, but also the comparative analysis of the two models. This work is significant because it could have shown the benefit to adopting a new model of computation into standard use and because it does confirm the continued viability of the standard RAM model.

This paper is organized as follows: Section 2 presents the paper's motivation. Section 3 addresses previous work. Section 4 details how the benchmarks and analyses were done.

Section 5 presents the results. Section 6 discusses the results, selecting the more accurate model. Section 7 and 8 address the potential for future work and conclude the paper.

2 Motivation

Analysis of algorithms on abstract machines serves as a crucial element of computer science, which allows incredible increases in the speed and utility of computer programs [4]. Until now, computer scientists have used the RAM and the EM models to perform algorithmic analysis accurately. But Jurkiewicz and Mehlhorn observe discrepancies with some experimental findings and attempt to push forward the VAT model to account for these discrepancies [5].

Before algorithm analysts shift to this new model, they must carefully verify its claims and correctness, it's utility. This is where Jurkiewicz and Mehlhorn's contribution falls short. Their paper mentions that experimental findings differ with the theoretical predictions of the RAM model and claims to solve these difficulties with the VAT model, but it does not clearly present these findings. Further, the number of algorithms studied in the paper is limited. For these reasons, someone must further test the model and expand the set of tested and analyzed algorithms, comparing the analysis with benchmarks, before analysts can incorporate this new model into their work.

More accurate algorithm analysis would help maintain the pace of innovation enjoyed by computer science for a generation. If it could be verifed by experimental comparison that the new VAT model be more accurate, then this would serve as an incentive to researchers to use the proposed model for more accurate estimations of complexities of algorithms.

3 Previous Work

Most work on algorithms tends to ignore the cost of virtual address translation [4], though researchers recognize these costs [5,6]. Virtualized memory is not accounted for in the standard random access machine model of computation, which treats all memory accesses as constant-time operations. Most algorithm analysis uses this RAM model. Papers like "The cost of virtualization" by Ulrich Drepper [6] and other materials like "AMD64 Architecture Programmer's Manual Vol 2" [2] describe the implementation of virtual memory and its associated costs during translation; but no study has tried to develop a model that considers these costs for algorithmic analysis prior to the recently proposed VAT model.

Keeping this trend in mind, it is not surprising that there has been little related work that carefully verifies the experimental results and strengthens the new model that accounts for these costs. This paper's contribution, experimentally examining and comparing the proposed model to the older model, begins to fill this gap.

4 Methods

Three components are necessary to determine which model of computation yields a more accurate analysis of the performance of algorithms running on real hardware. First, there must be implementations of algorithms running on real hardware. Running time benchmarks establish a ground truth for doing comparisons. Second, the algorithms must be analyzed according to the new VAT model. And third, the algorithms must be analyzed using the RAM model. The authors selected five well-studied algorithms which have been previously analyzed with the RAM model.

4.1 Benchmarking

To generate a standard for comparison, the authors selected a set of five algorithms: Binary Search, Heapsort, Insertionsort, Quicksort, and Permute. These five are well understood, have a variety of running times, and should be straightforward enough to analyze on the new model.

The algorithms were coded in C++ and compiled with GCC and Make. They were run on a VirtualBox virtual machine, with Debian GNU/Linux for 32 bit x86 processors. The virtual machine was hosted by 64 bit Windows 7 on a 2.0 GHz Core i7 processor. The physical machine has 8 GiB of memory and four processor cores, of which 2.5 GiB and 2 cores were allocated to the virtual machine. Find a link to the source code in the appendix.

The benchmark program ran each procedure 10 times on each input size. The smallest input to each procedure was an integer array of length 1. The input doubled in size after every tenth run. The maximum size input array to Binary Search was 536870912 32 bit integers, the 2 GiB size reaching the limits of the machine's memory capacity. The maximum size input array to Insertionsort was 1048576 32 bit integers. The execution time on any larger input to Insertionsort became excessive as it surpassed 4 billion microseconds (over 1 hr and 6 minutes) and overflowed the 32-bit integer used by the C standard library to track the time. The other three procedures had maximum size input arrays of 268435456 32 bit integers, each array taking up 1 GiB of system memory.

The benchmark program timed each run of each procedure on each size of input. The raw results are available at the link provided in the appendix. The next section of the paper provides a summary presentation.

4.2 VAT Model Analysis

As mentioned in the previous sections, the VAT model introduces the concept of virtual memory to the existing RAM machines. The motivation behind using the VAT model is multiprocessing where several programs are executed concurrently on the same machine. In this scenario the VAT model provides each of the concurrently running programs with a linear address space with non negative indices. But these addresses are virtual and are simulated with one physical memory. This implies that to get the actual physical memory location, some translation from the virtual address space to the physical address space is required which in turn adds some costs to the algorithmic complexity. Costs are also associated with page faults and translation lookup buffer (TLB) misses.

The main data structure that is used for the translation is a tree with outdegree K and the translation process is a walk in this tree. The tree is also referred as the page table (consisting of entries that map virtual to physical addresses). The leaves of the tree store indices of physical pages and the offset determines the cell in the physical address.

The translation process is done by a Translation Cache (TC) residing in the RAM which stores some nodes of the translation tree. The TC is changed by insertions and evictions and follows efficient replacement strategies[5]. To translate a virtual address to a physical address we start from the root node of our tree and continue traversing the nodes as mentioned in the virtual address and stop when we reach a leaf. Therefore translating a virtual address requires access to the nodes of the translation path in the TC in the correct order. The length of the translation is the number of insertions performed during translation and the cost of the translation is r times the length. An elaborate explanation of the translation tree can be found in the appendix section of our reference paper.[5]

For simplicity, we consider the virtual memory of a single program. Now for analyzing the translation cost of algorithms as a function of problem size n and memory m, we consider $m = \Theta(n)$ and we assume.[5]

- $rd \leq P$ the cost of moving a single translation path to the TC is no more than the size of a page, i.e., if at least one instruction is performed for each cell in a page, the cost of translating the index of the page can be amortized.
- $K \geq 2$ the fanout of the translation tree is at least two.
- $\frac{m}{P} \leq Kd \leq \frac{2m}{P}$ the translation tree suffices to translate all addresses but is not large. As a consequence $\log(\frac{m}{P}) \leq d\log(K) = Kd \leq 1 + \log(\frac{m}{P})$ and hence $\log(k)(\frac{m}{P}) \leq d \leq \frac{1}{k}(1 + log(\frac{m}{P}))$.
- $d \leq W$ the translation cache can hold at least one translation path.

This is only a summary of the model, and any seeking further explanation ought to consult Jurkiewicz and Mehlhorn [5].

5 Results

This section presents the results of the experimental and analytical work. The results of RAM model analysis are in the first subsection for reference. The second subsection contains the results of analysis under the VAT model. The results of the implementation and measurement reside in the third section.

5.1 RAM

This paper selected five reference algorithms, with running times well studied under the RAM model. The following table presents those running times. These will be used later to make comparisons. Where there is a difference in average and worst case with Quicksort, the average case running time is the tested case.

Algorithm	Average Case	Worst Case
BinarySearch	$O(\log(n))$	$O(\log(n))$
Heapsort	$O(n\log(n))$	$O(n\log(n))$
Insertionsort	$O(n^2)$	$O(n^2)$
Permute	$O(n)$	$O(n)$
Quicksort	$O(n\log(n))$	$O(n^2)$

It is left as an excersize for the skeptical reader to look these up in any algorithms reference text.

5.2 VAT

The VAT model presents more of a challenge than checking a reference text. In fact, the difficulty of using this model is a major barrier to its usefulness. One must imagine that its results must be superior indeed for an algorithm analyst to find it worth the effort to use.

Jurkiewicz and Mehlorn provided analyses for two of the algorithms [5]. Those results are here:

Algorithm	VAT Complexity
BinarySearch	$O\left(\frac{r}{k}\log(\frac{2n}{P})\log(\frac{2dn}{W})\right)$
Heapsort	$O\left(r(d+\frac{pn}{P})\right)$

Where r is the unit cost of translation (length/number of insertions in the TC). d is the length of translation. W represents the nodes of the translation tree that are in the TC. P is the page size. And, n is the input size.

Extending on the work that Jurkiewicz and Mehlorn do to give bounds on random access and sequential access, one can give VAT analyses for Permute and for Insertionsort.

Algorithm	VAT Complexity
Sequential Access	$O\left(2P+\frac{2n}{d}\right)$
Random Access	$O\left(\frac{r}{k}n(1+log(n/P))\right)$

Permute consists of simultaneous random and sequential accesses to an array. This, then, makes the sequential accesses non-sequential and random. So, its time can be bounded by $O\left(2\frac{r}{k}n(1+log(n/P))\right)$. This becomes $O\left(\frac{r}{k}n(1+log(n/P))\right)$.

Insertionsort consists of nested sequential accesses to an array. This makes the inner accesses truly sequential and the outer accesses random, yielding $O\left((2P+\frac{2n}{d})\frac{r}{k}n(1+log(n/P))\right)$.

From these results, one would expect the following behaviours in an experiment where n is changed and other variables held constant:

Algorithm	VAT Complexity
BinarySearch	$O\left(\log^2(n)\right)$
Heapsort	$O\left(n\right)$
Permute	$O\left(n\log(n)\right)$
Insertionsort	$O\left(n^2\log(n)\right)$

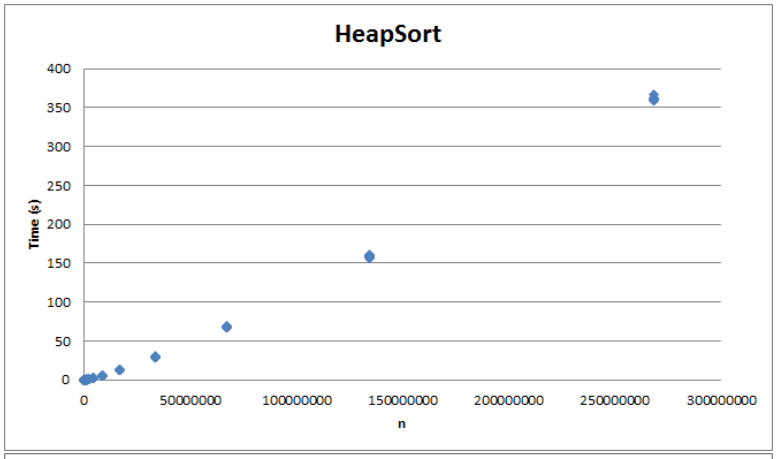
The authors were unable to analyze Quicksort by the submission deadline.

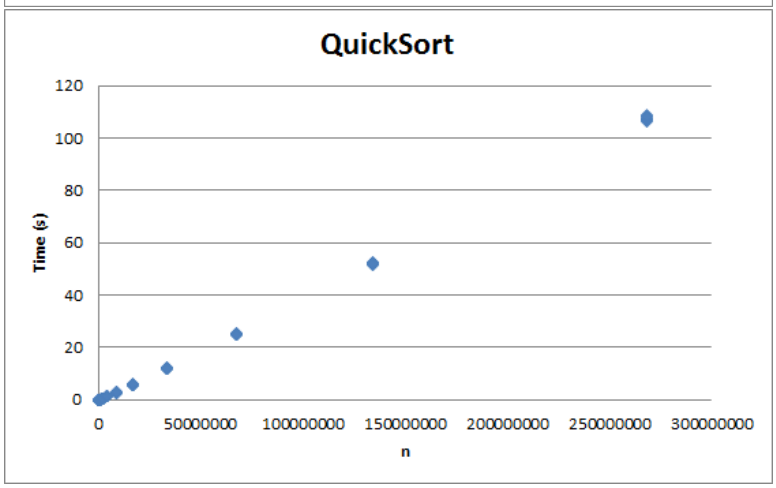
5.3 Benchmarks

Benchmarking the algorithms on a modern computer is a key step in determining whether VAT or RAM more accurately models a computer for the purposes of algorithm analysis.

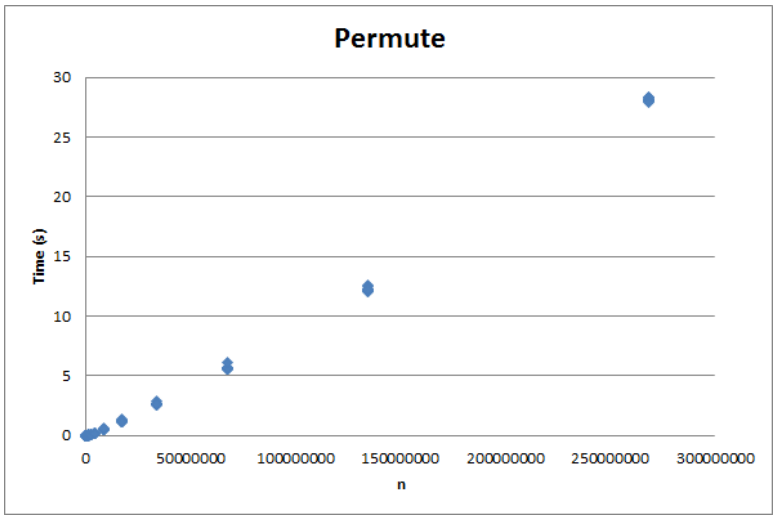
BinarySearch of an integer array was the first algorithm measured. Reviewing the results of timing, one would see all search times as 0.000000 seconds. The BinarySearch procedure executed too quickly, even on a 2 GiB array, to produce a result within the precision of the program's measurement.

The sorting algorithms were executed on randomly generated integer arrays, and the permute procedure was executed on a sorted array to yield a randomized array of integers. The accompanying scatter plots show the execution times of each procedure at each size input.

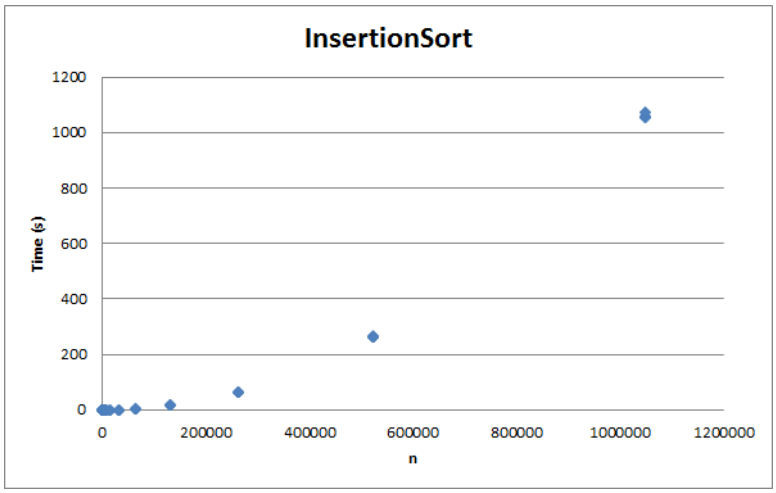




The Heapsort and Quicksort times were consistent with $O(n \log(n))$ runtimes. Heapsort has a much higher absolute running time and other behaviour consistent with that algorithm requiring a lengthier procedure.



The Permute procedure shows linear time behaviour.



Insertionsort shows a $O(n^2)$ running time that one sees very clearly on the above scatterplot.

6 Discussion

The results of the experiments come out very clearly from the data. This section compares the RAM model to the benchmarks, then the VAT model to the benchmarks, and lastly the VAT model to the RAM model to determine which model is superior.

RAM and Benchmarks As noted in the previous section, the running times of Heapsort and Quicksort are consistent with $O(n \log(n))$ behaviour, Insertionsort with $O(n^2)$ behaviour, and Permute with linear behaviour. These results are the same as predicted by the RAM model.

VAT not Necessary The results of the benchmarks are not consistent with Jurkiewicz and Mehlhorn. They presented, as part of the motivation for their new model, a graph showing large differences between the RAM-predicted time complexities and the actual running times of a set of algorithms [5]. In particular, Permute, Heapsort, and Binary-Search were shown in that paper to significantly depart from the RAM model at input sizes near 2^{21}. Benchmarks done on a virtual machine could show this discrepancy even more clearly, as the virtual address translation would need to be done on both the virtual and the physical machines.

The benchmarks described above present a different story. This paper tests those three algorithms at input sizes well above 2^{21} and finds no noticeable departure from the RAM model which would necessitate a new model of computation.

Recommendation This paper cannot recommend the VAT model for use at this time. The experimental evidence points to the continuing utility and validity of the venerable RAM model of computation. The VAT model is far more difficult to use than is the RAM model, while it simultaneously fails to give a more accurate prediction of the running times of the tested algorithms. Therefore, the RAM model should continue to be used by most analysts.

7 Future Work

This paper leaves open some possibilities for future work. In the first place, mathematical models are always up for refinement. No mathematical construct has been found that can perfectly simulate the physical universe. Thus, there is always room for improvement to any model.

With respect the the analysis and conclusions given in this paper, one might see that the analysis was conducted on consumer grade, personal computer hardware. This analysis may not necessarily apply to situations involving big data, servers, cloud computing, or some new technology. Perhaps there is not a single mathematical model to precisely describe every arrangement. These areas are worth further study.

8 Conclusions

The aim of this paper was to experimentally compare two models of computation, the Random Access Machine model and a new model devised by Jurkiewicz and Mehlhorn that incorporates the costs of Virtual Address Translation. This paper began by introducing the contenders, explained the motivation to look for a new model of computation, and noted the relevant prior work. The paper continued by describing a method for comparing the two models and presenting the results of that work. Having measured the running times of a set of algorithms and compared the results with the predictions of the models, this paper determined that the incumbent RAM model is superior to the new VAT model.

While it would have been exciting to validate this brand new model, to potentially contribute to a small revolution in the field, it is comforting to know that the widely used RAM model has earned its place by continually demonstrating its validity and utility.

9 References

1. N. Rahman, "Algorithms for harware caches and TLB," *Lecture Notes in Computer Science*, vol. 2625, pp. 171–192, 2003.

2. Advanced Micro Devices, *AMD64 Architecture Programmer's Manual*, vol. 2: System Programming. 2010.

3. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. New York: Elsevier, 2012.

4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, Massachusettes: The MIT Press, 2009.

5. T. Jurkiewicz and K. Mehlhorn, "The cost of address translation," presented at the ALENEX13, New Orleans, Louisiana, 2013, pp. 148–163.

6. U. Drepper, "The Cost of Virtualization," ACM Queue, vol. 6, no. 1, pp. 28–35, 2008.

7. R. K. Ahuja and J. B. Orlin, "Use of Representative Operation Counts in Computational Testing of Algorithms," *INFORMS Journal on Computing*, vol. 8, no. 3, pp. 318–330, 1996.

8. U. Drepper, "What every programmer should know about memory, Part 1 [LWN.net]," 21-Sep-2007. [Online]. Available: http://lwn.net/Articles/250967/. [Accessed: 27-Feb-2013].

Appendix

All source code, benchmarks, and report drafts can be found at:
https://github.com/dslachut/adv-algo-project

Item	Address
Benchmarking Source Code	adv-algo-project/tree/master/prof++/
Benchmarking Executable	adv-algo-project/blob/master/prof++/Debug/prof++
Benchmark Results	adv-algo-project/tree/master/benchmarks
Final Report	adv-algo-project/tree/master/Final

: