ISSN : 2231-0711

A Comparative Study of Some Ingenious Sorting Algorithms on Modern Architectures

D.Abhyankar, M.Ingle
School of Computer Science, D.A. University, Indore M.P. India

***Abstract-* Heapsort, Quicksort and Mergesort are three ingenious sorting algorithms, which deserve special attention. In the past these algorithms were studied in detail, but the study was carried out on old machines and involved cache simulations. Over the years computer architecture has gone through radical changes. Therefore, a study valid on old architectures may not be valid on new architectures. The comparative study of Heapsort, Quicksort and Mergesort on modern machines with the help of latest performance analyzers is the central idea of the paper. To choose among Heapsort , Quicksort and Mergesort on modern machines, a comparative study of prior will be useful. Quicksort has been a usual choice for sorting, but some researchers suggest that accelerated Heapsort, a variant of Heapsort, is competitive enough to become a method of choice to solve sorting problem [6]. This paper examines the claim of accelerated Heapsort as a method of choice.**

1. INTRODUCTION

In Computer science one of the classic problems is sorting. Though several algorithms solve the problem, nevertheless only a few solve it efficiently. Quicksort, Heapsort and Mergesort belong to the set of those few. Heapsort, Quicksort and Mergesort are intensely competitive sorting algorithms. So in the past Scientists studied and compared these sorting algorithms, the comparisons however were theoretical and were done on old architectures. An algorithm effective on old architectures may not be effective on modern machines. A comparative study valid on old architectures may not be so on modern architectures. Moreover in past researchers did not have advanced performance analyzers to study cache miss and page faults. Consequently researchers relied on cache simulations. Therefore their results may be inaccurate. Hence It is beneficial to compare the algorithms on contemporary architectures using state of the art performance analyzers.

It has not escaped our notice that state of the art machines are multicore and if an algorithm has to be effective it should be multicore ready [13]. Future lies in parallel /multithreaded algorithms, but even then one should not forget that parallel algorithms or multithreaded algorithms will need sequential algorithms at lower level. The basic question is which sequential sorting algorithm to call at lower level. Having a slow sequential algorithm at lower level will neutralize the advantage of parallel sorting gained by multiple cores. So the question which sequential sorting is the best alternate at lower level is of paramount importance. This study tries to answer the same question.

To compare the performance of Quicksort, Heapsort and Mergesort on contemporary machines is the central idea of the paper. A good test of the algorithm's performance is its execution time. The drawback of this approach, however is that no intuition is provided as to why the execution time performance was good or bad. The reason(s) may be high instruction count, high cache miss count and High branch misprediction count. Even high pagefault count affects the performance. Earlier researchers studied the impact of these factors using cache simulation and similar techniques. Fortunately today researchers have performance analyzing softwares which are not merely effective in capturing execution time but also acquire accurate data about cache miss, branch mispredictions and page faults.

Performance analyzer AQTime obtained us reliable data about factors discussed in earlier paragraph. A researcher needs to interprete data correctly with an eye towards overall performance improvement, or researcher arrives at false conclusions. She cannot be biased in favour of a single factor. Having a focus on overall performance gain rather than being biased in the favour of a single factor is one of the key ideas of research method. Next Section focuses on the Research method. The research method is followed by a summary of past results, which is covered in Section 3. Section 4 presents the data obtained by performance analyzer on the modern computers. Finally Section 5 analyzes the data carefully and concludes the study.

2. RESEARCH METHOD

One of the key ideas of this method is to try a practical approach. Fortunately we have state of the art softwares to adopt the pragmatic approach. AQtime software was used to profile the sorting algorithms. AQtime is a profiler that obtains reliable data about elapsed time, CPU cache misses, branch mispredictions and pagefaults. Profiler helps us in understanding as to why one algorithm is slower, and why other is fast? While we prefered a practical approach, whereas previous researchers, did not enjoy the luxury of sophisticated profilers which we enjoy, relied heavily on theoretical models and cache simulations.

There are sound reasons to prefer the pragmatic approach. To appreciate the reasons we will contrast our approach with the other approaches. Majority of algorithm researchers compare the algorithmic performance on the basis of unit cost model. The RAM model is a most commonly used unit cost model in which all basic operations involve unit cost. The advantage of unit cost model is that it is simple and easy to use. Moreover it produces results which are easily comparable. However, this model does not reflect the memory hierarchy present in modern machine. It has been observed that main memory has grown slower relative to processor cycle times, consequently Cache miss penalty has grown significantly [12]. Thus good overall performance cannot be achieved without keeping cache miss count as low as possible. Since RAM model does not count cache miss, it is no longer a useful model.

Usually algorithm researchers in sorting area only count particular expensive operations. Analyses of sorting and searching algorithms, for instance, only count the number of compares and swaps. There was logic behind only counting comparison operation which was expensive in the past. That simplified the analysis and still retained accuracy since the bulk of the costs was captured, but this is no longer true because the shift in the technology renders the “expensive operations” inexpensive and vice versa. Same happened with comparison operation which is no longer expensive. Indeed it is no more expensive than adding or copy. Section 4 and 5 show how this bias towards comparison leads to incorrect conclusions. So the study favours a pragmatic approach and is not biased towards a single performance indicator. The idea is to have a fairly objective view and goal of good overall performance rather than concentrating on a single point.

3.A BRIEF REVIEW OF PAST RESULTS

Quicksort, deteriorates and takes Quadratic time in the worst case, spends a lot of time even on the sorted or almost sorted data, takes up only O(log n) amount of extra space if implemented cleverly, takes linear amount of space in the worst case otherwise. It performs a lot of comparisons even on sorted data, but swap count is low for sorted or almost sorted input [10]. Quicksort is on average blindingly fast. On average it needs only 1.3 n log n comparisons to complete the task. Even overall instruction count is comparatively low [1,2,3,4,5,7,8,9]. Quicksort, upto some extentent can be parallelized.

Heapsort, runs slower than Quicksort on average, spends a lot of time even on the sorted or almost sorted, takes up only constant amount of extra space. It, does not maintain original order of records on equal keys, does not need extra array or extra stack space. Heapsort even in the worst case offers an asymptotically optimal performance. An Heapsort variant, known as Accelerated Heapsort was developed by Karlson [11]. Literature suggests that Accelerated Heapsort is competitive enough to replace Quicksort as an algorithm of choice. It has very little inherent parallelism [6,11].

Mergesort takes extra array, and if implemented in a topdown way takes extra stack space because of recursion. Though time spent by Merge sort is asymptotically optimal, but other factors ought to be studied cautiously. Mergesort a stable algorithm maintains the original order of records on equal keys. Mergesort has an extremely low comparison count and only a few algorithms have smaller count. Easier it is devise a parallel version of Mergesort [9].

4. COMPARATIVE STUDY ON MODERN ARCHITECTURES

This study, compares Quicksort, Heapsort and Mergesort on modern computers, crosschecks whether Accelerated Heapsort(a variant of Heapsort) can compete with Quicksort on average as literature suggests. Thus to compare, algorithms were tested on Psuedorandom numbers. Following tables and figure present the average case statistics generated by the tests on 4 important performance indicators: elapsed time, CPU Cache Miss, Branch mispredictions and page faults. AQtime software was instrumental in gathering the reliable profiling data.

Table 1 Quicksort Statistics

N	Elapsed time (in ms)	Cache Miss	Mispredicted Branches	Page Faults
10000	2.17	73	97402	0
20000	4.48	50	204154	0
30000	7.10	105	315749	0
40000	9.42	173	426113	0
50000	11.84	1567	543232	0
60000	14.69	495	661780	0
70000	17.47	337	768937	0
80000	20.35	364	893583	0
90000	23.76	1167	1012280	0
100000	26.24	788	1150368	0

Table 2 Heapsort and Accelerated Heapsort Statistics

N	Heapsort				Accelerated Heapsort			
	Elapsed Time (in ms)	Cache Miss	Mispredicted Branches	Page Faults	Elapsed Time (in ms)	Cache Miss	Mispredicted Branches	Page faults
10000	2.93	182	104676	0	7.42	498	261907	0
20000	6.11	661	211068	0	13.66	616	505649	0
30000	9.60	821	324514	0	23.50	1637	668386	0
40000	12.97	1141	442950	0	30.83	2204	1021126	0
50000	16.58	1158	566733	0	37.62	2962	1313025	0
60000	20.01	1712	690736	0	46.37	3003	1614741	0
70000	23.68	1474	805584	0	51.67	2685	1869227	0
80000	27.75	1964	919547	0	62.13	8466	2187585	0
90000	31.79	2862	1052313	0	69.49	4457	2508983	0
100000	36.05	3077	1194742	0	78.49	8430	2885227	0

Table 3 Topdown Mergesort Statistics

N	Elapsed Time (in ms)	Cache Miss Count	Mispredicted Branches	Total Page Faults	Soft Memory Page Faults	Hard Memory Page Faults
10000	7.25	2253	195353	250	250	0
20000	14.70	4442	393851	521	521	0
30000	22.23	6472	594805	795	795	0
40000	30.36	8326	817774	1084	1084	0
50000	29.54	11648	1001526	1369	1369	0
60000	38.31	10898	1004099	1369	1369	0
70000	53.75	14601	1371954	1939	1939	0
80000	62.46	16487	1677569	2248	2248	0
90000	70.08	18718	1917031	2547	2547	0
100000	77.22	21626	2115426	2838	2838	0

Table 4 Bottomup Mergesort Statistics

N	Elapsed Time (in ms)	Cache Miss Count	Mispredicted Branches	Total Page Faults	Soft Memory Page Faults	Hard Memory Page Faults
10000	5.48	1570	114612	248	248	0
20000	11.28	3700	231595	520	520	0
30000	17.09	6462	357841	788	788	0
40000	23.18	7460	488235	1078	1078	0
50000	30.93	7864	615057	1362	1362	0
60000	35.28	10794	737387	1636	1636	0
70000	41.22	15715	880837	1967	1967	0
80000	47.71	20177	1002236	2236	2236	0
90000	58.36	22870	1133682	2516	2516	0
100000	63.53	22199	1259579	2824	2824	0

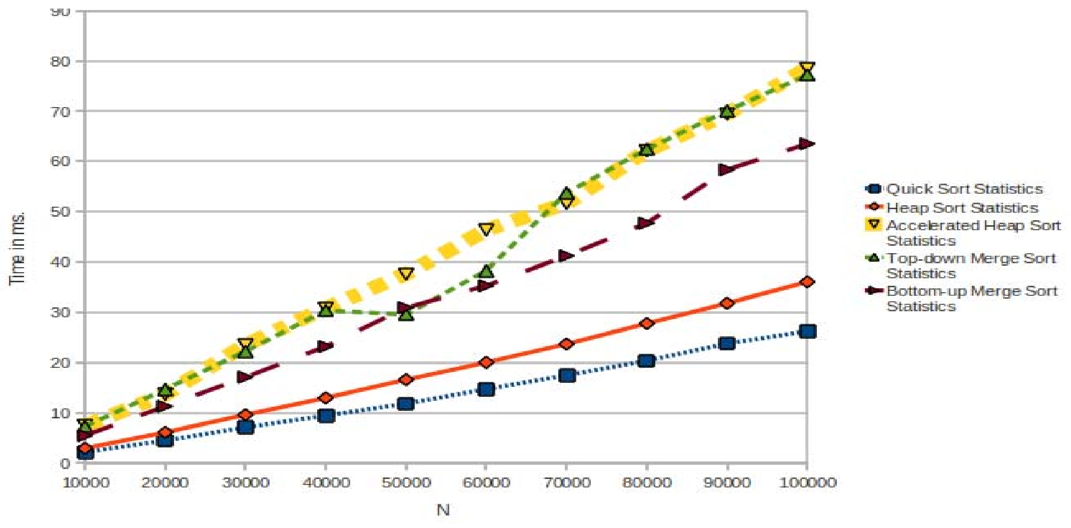

Figure 1: Comparative Statistics

5. Analysis, Results and Conclusion

Tables and Figure 1, show the results based on random input, depict the performance on 4 crucial performance indicators. Hard Page fault count is 0 for each one of the algorithms. Zero pagefault count is due to large main memory size which was not feasible earlier. Even soft page fault count is zero for each one of the algorithms except Mergesort. Soft page faults of Mergesort, indicate TLB miss, explain as to why the merge sort is slow. Quicksort beats the other algorithms in almost all entries in the table. To put it differently Quicksort suffers lesser number of cache miss and branch mispredictions. So Quicksort sorts the data in record time. Surprizingly Accelerated Heapsort is the last one to finish. Accelerated insertion sort is slow because of its higher instruction count, poor cache locality and fairly high branch mispredictions. Though comparison count of Accelerated Heapsort and Merge sort is low, overall instruction count however is high. Mergesort has two variations: Topdown mergesort and Bottomup Mergesort. Tables 3 and 4 show that bottom up mergesort runs faster than top down Mergesort. Bottom up Mergesort sorts the data efficiently because it avoids the recursive calls, whereas Top down Mergesort is on slower side because it has double recursion.

On the whole we can say that Quicksort, though a bit of gamble is still the best choice on average. Heapsort is behind Quicksort, but is able to compete with Quicksort on smaller arrays. Accelerated Heapsort is an overrated algorithm and does not accelerate at all, its acceleration is a myth created entirely by the past literature and a biase towards comparison count. This paper does not intend to unduly critisize the accelerated Heapsort, but still this paper reveals that accelerated heapsort with its sole virtue of low comparison count does not add much to the performance. Same is the case with Mergesort, which takes lot of time and space. But it is easy to create a parallel version of Mergesort.

References

[1] J. L. Bentley and M. D. Mcilroy "Engineering a sort function," Software—practice and experience, VOL. 23(11), 1249–1265 (NOVEMBER 1993).

[2] R. Sedgewick, 'Quicksort', PhD Thesis, Stanford University (1975).

[3] C. A. R. Hoare, "Partition: Algorithm 63, " "Quicksort: Algorithm 64," Comm. ACM 4(7), 321-322, 1961.

[4] D. E. Knuth, The Art of Computer Programming, Vol. 3, Pearson Education, 1998.

[5] C. A. R. Hoare, "Quicksort," Computer Journal5 (1), 1962, pp. 10-15.

[6] S. Baase and A. Gelder, Computer Algorithms:Introduction to Design and Analysis, Addison-Wesley, 2000.

[7] J. L. Bentley, "Programming Pearls: how to sort," Communications of the ACM, Vol. Issue 4, 1986, pp. 287-ff.

[8] R. Sedgewick, "Implementing quicksort Programs," Communications of the ACM, Vol. 21, Issue10, 1978, pp. 847-857.

[9]T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001.

[10] G. S. Brodal, R. Fagerberg and G. Moruz, "On the adaptiveness of Quicksort," Journal of Experimental AlgorithmsACM, Vol. 12, Article 3.2, 2008.

[11] S.Carlsson, "A variant of HEAPSORT with almost optimal number of comparisons, " Information Processing Letters Modified 24:247-250,1987.

[12] A. G. LaMarca, "Caches and Algorithms, " PhD theses University of Washington, 1996.

[13] M. Edahiro, "Parallelizing fundamental algorithms such as sorting on multi-core processors for EDA acceleration, " ASP-DAC '09 Proceedings of the 2009 Asia and South Pacific Design Automation Conference, 2009.