A Performance Study of Some Sophisticated Partitioning Algorithms

D.Abhyankar
School of Computer Science
Devi Ahilya University
Indore, M.P.

M.Ingle
School of Computer Science
Devi Ahilya University
Indore, M.P.

***Abstract*— Partitioning is a central component of the Quicksort which is an intriguing sorting algorithm, and is a part of C, C++ and Java libraries. Partitioning is a key component of Quicksort, on which the performance of Quicksort ultimately depends. There have been some elegant partitioning algorithms; Profound understanding of prior may be needed if one has to choose among those partitioning algorithms. In this paper we undertake a careful study of these algorithms on modern machines with the help of state of the art performance analyzers, choose the best partitioning algorithm on the basis of some crucial performance indicators.**

Keywords- Quicksort; Hoare Partition; Lomuto Partition; AQTime.

I. Introduction

Partitioning is undoubtably a core part of the Quicksort on which the performance ultimately depends. Quicksort is a leading and widely used sorting algorithm. For instance C, C++ and Java libraries use Quicksort as their sorting routine. The Partitioning is a key component of the Quicksort and selection algorithm. There are several partitioning algorithms that accomplish the task, but only a few deserve special attention. Hoare, Lomuto, Modified Lomuto and Modified Hoare are those few selected partition algorithms. This paper carries out an in depth study of the selected partitioning algorithms. The important question is as to which partitioning algorithm is superior so that we can call the superior algorithm in sorting routine. This study attempts to answer the same question. In past Scientists studied and compared these algorithms; the comparisons however were theoretical and were made on old architectures. An algorithm effective on old architectures may not be effective on modern machines. A 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. Calling 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 option at lower level is of paramount importance. Literature suggests that Quicksort offers the most effective answer at least today. If the Quicksort is lower level sequential sorting algorithm, then the very next question is which Partitioning algorithm we should choose. This study is going to solve the same question.

To study the performance of selected partitioning algorithms on contemporary machines is the central idea of the paper. A fair test of the algorithm's performance is its execution time; however the drawback of this approach 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 page fault 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.

II. Literature Review

In the past researchers did not enjoy the luxury of sophisticated profilers which we enjoy now. Instead they relied heavily on theoretical models and cache simulations. 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. Analysis of sorting and searching algorithms, for instance, only counts the number of comparisons and swaps. There was exquisite 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 not expensive anymore. Indeed it is no more expensive than addition or copy. Hence the study favours a practical 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 performance indicator.

Literature reveals that every partitioning algorithm incurs (n-1) comparisons, where n is total number of elements in the array[1, 2, 3, 4, 5, 6, 7, 8, 9]. Partitioning algorithms differ in swap count or data transfer operations. Hoare partition & Modified Hoare partition algorithms lead to adaptiveness of swap count / data transfer operation count. In the worst case, for Hoare and Modified Hoare algorithms swap count/ data transfer count is approximately(n/2), whereas for Lomuto and Modified Lomuto swap count/data transfer count is approximately (n)[14].

III. PERFORMANCE STUDY ON MODERN ARCHITECTURES

This paper studies the performance of Hoare partition, Lomuto partition, Modified Hoare partition and modified Lomuto partition on contemporary computers. Thus to study algorithms were tested on Pseudorandom numbers using state of the art Machines. Experiments were performed on state of the art COMPAQ PC which was equipped with Windows Ultimate operating system. Following tables and figure present the average case statistics generated by the tests on 3 important performance indicators: elapsed time, CPU Cache Miss, Branch mispredictions. AQtime software was instrumental in gathering the reliable profiling data. Elapsed time given in the table is in milliseconds.

TABLE I: STATISTICS OF LOMUTO PARTITION

Lomuto Partition			
N	***Elapsed Time***	***CPU MisPredicted Branches***	***CPU Cache Misses***
10000	7.52	280360	435
20000	16.83	596571	857
30000	28.49	998545	1778
40000	30.43	1372734	3651
50000	50.65	1708752	3132
60000	52.82	2064278	7865
70000	68.02	2660145	3406
80000	72.52	2990947	5161
90000	86.64	3270575	6328
100000	103.15	3815563	6324

TABLE II: STATISTICS OF MODIFIED LOMUTO PARTITION

Modified Lomuto Partition			
N	***Elapsed Time***	***CPU MisPredicted Branches***	***CPU Cache Misses***
10000	1.57	66588	82
20000	2.99	148860	60
30000	4.65	222592	171
40000	6.46	312973	269
50000	8.2	387050	447
60000	10.27	477689	422
70000	11.75	569962	590
80000	13.32	637354	1206
90000	15.32	741244	206
100000	17.62	806429	798

TABLE III: STATISTICS OF HOARE PARTITION

Hoare Partition			
N	***Elapsed Time***	***CPU MisPredicted Branches***	***CPU Cache Misses***
10000	3.52	165923	745
20000	7.55	351172	650
30000	12.26	530738	615
40000	16.04	708813	1139
50000	20.53	890882	982
60000	24.95	1141577	1566
70000	31.46	1303499	4242
80000	33.96	1561376	1686
90000	38.36	1680148	3812
100000	44.58	1862329	2100

TABLE: IV: STATISTICS OF MODIFIED HOARE PARTITION

Modified Hoare Partition			
N	***Elapsed Time***	***CPU MisPredicted Branches***	***CPU Cache Misses***
10000	1.46	83287	45
20000	3.08	173563	520
30000	4.71	267342	319
40000	6.28	369235	234
50000	7.87	463831	245
60000	9.55	575380	1014
70000	11.17	725573	714
80000	12.92	724582	785
90000	14.22	829880	1489
100000	16.15	908749	1091

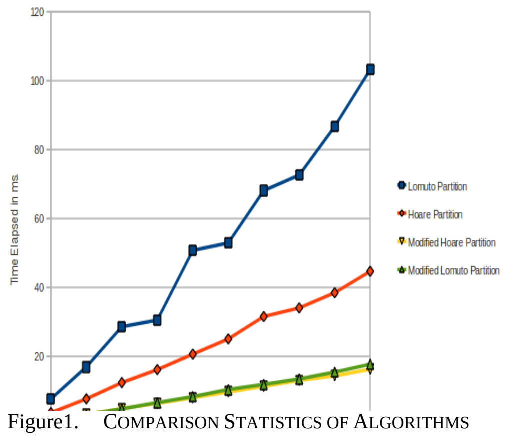

Figure1. COMPARISON STATISTICS OF ALGORITHMS

IV. ANALYSIS, RESULTS AND CONCLUSION

Tables and Figure 1, show the results based on random input, depict the performance on 3 crucial performance indicators. Since Page fault count was 0 for each one of the algorithms, it was not shown explicitly in the tables. Zero page fault count is due to large main memory size which was not feasible earlier. Modified Hoare partition outperforms the other algorithms in almost all entries in the table. Modified Lomuto is the second one to finish and is not too behind. Modified Lomuto is followed by Hoare partition which in turn is followed by Lomuto which is the last one to complete. It is easy to see that among the studied algorithms the one with the better cache miss count is usually the first one to complete the partitioning. Lomuto algorithm and Hoare algorithm are slow because of their higher instruction count, poor cache miss count and fairly high branch misprediction count. The interesting question that emerges is why Modified Hoare and Modified Lomuto have lower cache miss count whereas others have cache miss count on higher side. The intuitive reason is that instruction cache miss count is likely to go down as overall instruction count and code size goes down. If we can keep data cache miss count in check then overall cache miss count will be low. Same seems to have happened with Modified Hoare and Modified Lomuto partitioning algorithms.

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.

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

[12] 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.

[13] D. Abhyankar and M. Ingle, "Engineering of a Quicksort Partitioning Algorithm," Journal of Global Research in Computer Science Vol. 2, No. 2, 2011.

[14] V. Aho, J. E. Hopcroft, J.D. Ulman, "The Design and Analysis of Computer Algorithms, "Addison-Wesley, 1974.

[15] V. Aho, J. E. Hopcroft, J.D. Ulman, "Data Structures and Algorithms, "Addison-Wesley, 1983.

[16] J. L. Bentley, "Writing Efficient Programs, " Prentice-Hall, 1982.

[17] G. Brassard and P. Bratley, "Fundamentals of Algorithmics, "Prentice Hall, 1996.

[18] R. Mansi, "Enhanced Quicksort Algorithm, " The International Arab Journal of Information Technology, Vol. 7, No. 2, April 2010