On the Cache Performance of Time-Efficient Sorting Algorithms

Ilker Korkmaz[1], Senem Kumova Metin[2], Orhan Dagdeviren[3], Fatih Tekbacak[4]

[1]Izmir University of Economics, Department of Computer Engineering, Izmir, Turkey
[2]Izmir University of Economics, Department of Software Engineering, Izmir, Turkey
[3]Izmir University, Department of Computer Engineering, Izmir, Turkey
[4]Izmir Institute of Technology, Department of Computer Engineering, Izmir, Turkey
[1,2,3]Ege University, International Computer Institute, Izmir, Turkey

[1]ilker.korkmaz@ieu.edu.tr, [2]senem.kumova@ieu.edu.tr,
[3]orhan.dagdeviren@izmir.edu.tr [4]fatihtekbacak@iyte.edu.tr

Abstract. The cache configuration is an important concern affecting the performance of sorting algorithms. In this paper, we give a performance evaluation of cache tuned time efficient sorting algorithms. We focus on the Level 1 and Level 2 cache performance of memory tuned versions of both merge sort and quicksort running on data sets having various sizes and probability distributions. The algorithms are simulated in different cache configurations, including a Pentium 4 processor configuration, by using Valgrind simulator. Besides, we propose a new merge sort algorithm design which aims to decrease run time by reducing Level 1 cache misses.

Keywords*: merge sort, quicksort, cache effected sorting, simulation*

1 Introduction

Sorting operation [1-13] is commonly used in mathematics and computer sciences as well. Sorting functions are generally implemented as top-level definitions to be called whenever required in any software program or mathematical application. There are two main performance issues for a sorting algorithm: time and space complexity. Although there is a theoretical trade-off between those two issues, practical sorting implementations try to balance the effects and provide efficient running time results. There may be some different hardware configurations for the sorting codes to be executed in more performance-effective manner. In this concept, cache configuration is an important concern affecting the performance of sorting algorithms.

In this paper, we introduce a comprehensive performance evaluation of cache tuned time efficient sorting algorithms. We cover the Level 1 and Level 2 cache performance of memory tuned versions of both merge sort [2] and quicksort [8] running on data sets having various sizes and probability distributions. The algorithms are simulated in different cache configurations, including a Pentium 4 processor configuration [14]. The data and instruction miss counts on Level 1 and Level 2 caches are measured with the time consumption for each algorithm by using Valgrind simulator [15]. Besides, we give the design of a new proposed merge sort algorithm which aims to decrease run time by reducing Level 1 misses.

2 Related Work

Sorting algorithms effect run time and efficiency of operations in which they are used. If the algorithms are qualified, it may result with a decreased running times. The traditional method which is accepted to be the most effective way of qualifying algorithms to reduce run time is decreasing the number of instructions in the algorithm. Although reducing the instruction number of any algorithm has a positive effect on its running time, it is not possible to state that it supplies enough improvement for sorting algorithms. The approach in which the reality of cache usage in sorting algorithms is considered to reduce running time of sorting operations was stated firstly by LaMarca and Ladner [9-11], restudied by Xiao et al. [12].

Basic merge sort algorithm which is designed as divide and conquer paradigm is a comparative sort algorithm with O(nlogn) complexity. Multi merge sort and tiled merge sort algorithms take account of cache properties and reduce running time by reducing the memory access rates or cache miss rates. Multi merge sort algorithm merges all sub arrays in one operation. In tiled merge sort algorithm, the data is divided in two sub arrays and the arrays are sorted between each other. Conceptually based on those algorithms, Xiao et al. [12] proposed their padding versions. Tiled merge sort with padding organizes data locations to decrease the collision miss rate. Multi merge sort with TLB padding aims to reduce TLB misses created by multi merge sort algorithm. As the initials of cache effected quicksort algorithms, LaMarca and Ladner [9-11] proposed memory tuned quicksort and multi quicksort. To further improve the quicksort performance, Xiao et al. [12] examined flash quicksort and inplaced flash quicksort, which are basically some kinds of integration of flash sort and quicksort algorithms. For the procedural details of these cache effected merge sort and quicksort algorithms, related studies [9-12] can be dissected.

In a general cache configuration design, main properties to decide on are basically as follows: capacity, line size, associativity, multi-level design. Considering those configuration issues, cache effected algorithms can be simulated by Valgrind [15] and Cachegrind which is a Valgrind tool that detects miss rates by simulating Level 1 and Level 2 caches.

3 Experiments

3.1 Approach

To investigate the cache performances and effect of cache usage on merge sort and quicksort derivations, the following algorithms are examined in in Valgrind simulator: base merge sort, tiled merge sort, multi merge sort, and tiled merge sort with padding; base flash sort, flash quicksort, inplaced flash quicksort, and memory tuned quicksort. We simulated the algorithms in 2-level cache with different configurations.

As a top-view prescription of our approach, we have 3 phases: cache configuration, simulation of the corresponding sorting algorithms within the selected configuration

with different data set distributions, simulation of the algorithms with Pentium 4 cache configuration.

Each different experiment is repeated 5 times and the average values are used to evaluate the corresponding algorithm. The data miss rates of Level 1 and Level 2 caches of each algorithm are used to observe the cache performances. The running times of the implementations are also measured. It is expected that although all merge sort implementations have the same time complexity of O(nlogn), the algorithms, which have more misses in the last level of the cache and have more accesses to the main memory, result with higher values of running times. It is also expected that flash sort supported quicksort algorithms run faster than merge sort variations.

3.2 Simulations and Results

Determining the configuration. In the cache configuration phase, we first arranged the appropriate capacities of our Level 1 and Level 2 caches in 1-way associative design through simulating the base merge sort algorithm for random input data sets with *N:*{1024, 10240, 102400, 1024000}. 10 different configurations with the following set of parameters are simulated: *parameterSet*:{ *capacityL2:*{256 KB, 512 KB}*, capacityL1:*{32 KB, 64 KB, 128 KB}*, lineSize:*{64 B, 128 B} }. Among them, a configuration with L2 of 512 KB, L1 of 128 KB, and line size of 128 B is chosen according to its fastest average run time result per input element. Then, this configuration is examined with the following set associativity values: {1-way, 2-way, 4-way, 8-way, full-associative}. Among them, 4-way set associative cache configuration is selected according to its fastest average run time and smallest average miss rate results.

Simulations of various sorting alternatives with selected configuration. In the second phase, we first simulated base merge sort, tiled merge sort, and multi merge sort with different sizes of random data sets to compare the performance results; then we compared the best (according to miss rates) two among those merge sort algorithms with several input data sets of various distributions.

Base merge sort, tiled merge sort and multi merge sort algorithms are utilized on the selected cache configuration. Table 1 gives L1 and L2 miss rates and sorting run times (in microseconds) per element for random data sets with N=1024, 10240, 102400, 1024000. L1 misses in Table 1 include both instruction and data misses.

In Table 1, it is shown that miss rates are lower in tiled merge sort and multi merge sort compared to base merge sort as expected. Multi merge sort algorithm gives the lowest miss rate in randomly distributed data set. In order to examine the effect of distribution of data sets according to possible miss rates, the two best of three algorithms in Table 1, tiled merge sort and multi merge sort algorithms, are applied to randomly, equilikely, poisson and zero distributed data sets. Zero distribution refers the distribution in which all data values are equal to 0 (zero) and equilikely distribution is the uniform distribution. Since there is not enough space in this paper to present the detailed measurement results, only a concise summary of the results is given as

follows: Multi merge sort gives better run time results only in poisson distribution and for all other distributions tiled merge sort generates better run times.

Table 1. The results of several merge sort algorithms on selected cache configuration

		1024	10240	102400	1024000	Average Miss Rate and Run Time
base merge sort	**L2 miss rate**	0.1	0	0.1	0.1	0.08
	L1 miss rate	0.27	0.2	0.3	0.3	0.27
	Run Time	31.22	27.84	34.05	38.96	33.02
tiled merge sort	**L2 miss rate**	0.1	0	0	0	0.03
	L1 miss rate	0.27	0.1	0.1	0.1	0.14
	Run Time	32,74	23,74	33.99	41.96	33.11
multi merge sort	**L2 miss rate**	0.1	0	0	0	0.03
	L1 miss rate	0.27	0.1	0	0.1	0.12
	Run Time	32.28	28.95	46.32	65.32	43.22

Besides, we simulated the quicksort and flash sort variations to compare the cache effected quicksort algorithms. It is expected that flash sort supported quicksort algorithms run faster than merge sort variations. The comparisons are evaluated regarding to cache miss rates and run times of the algorithms.

Flash sort, flash quicksort, inplaced flash quicksort and memory tuned quicksort algorithms are simulated on randomly distributed data sets. Fig. 1 and Fig. 2 include the sorting time and miss rate results of the simulations for several different sorting algorithms on randomly distributed different sized data sets, respectively.

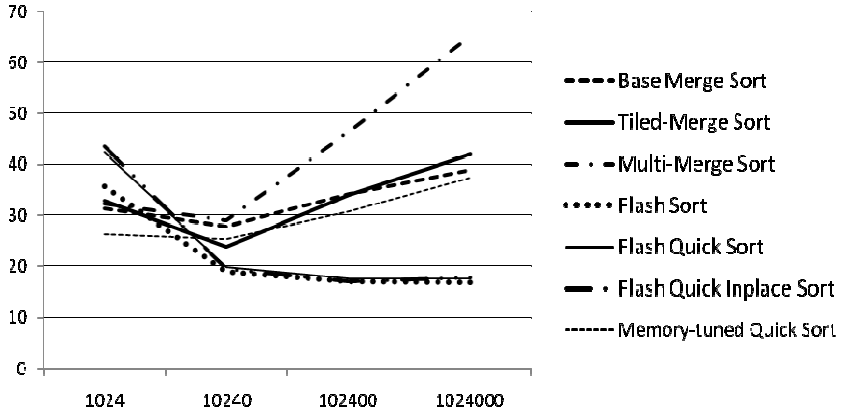

Fig. 1. Sorting run times (in microseconds) per input element of various sorting algorithms

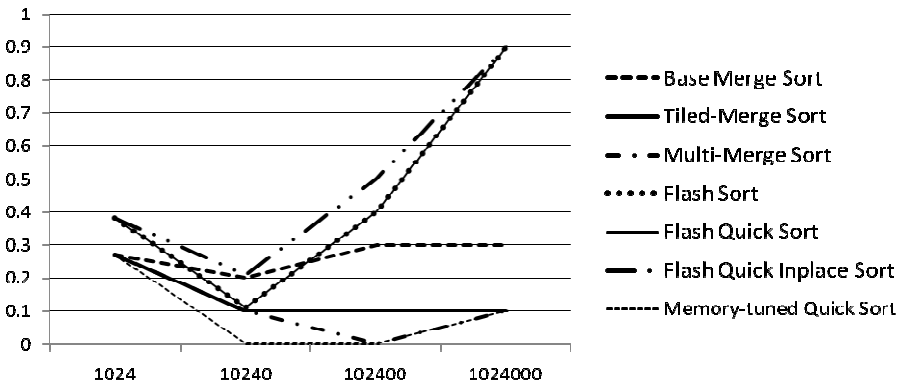

Fig. 2. Cache miss rates of various sorting algorithms

In Fig. 1, the vertical axis is the sorting time in microseconds per one input element; whereas in Fig. 2, the values of the vertical axis refer to miss rates for corresponding size of data sets. The horizontal axis is the corresponding total element size of input data set. As mentioned in the study of Xiao et al. [12], although flash quicksort algorithm produces high miss rates (shown in Fig. 2), the sorting running time is lower compared to other quicksort alternatives in large data sets (shown in Fig. 1). Flash sort and flash quicksort algorithms have same results. It is also shown in Fig. 2 that memory tuned quicksort algorithm gives the lowest miss rates as expected.

A more realistic case. In the last experimentation step, a more realistic case, 2-level cache structure of Northwood core Pentium 4 processor architecture [14] is simulated. Level 1 cache is configured as 4-way set associative with a total capacity of 8 K, which has a line size of 64 Bytes; Level 2 cache is used as 8-way set associative with a capacity of 256 K, and with the line size of 128 Bytes. Base merge sort, tiled merge sort, multi merge sort and tiled merge sort with padding algorithms are simulated with several data sets of 1 K to 4 M. The detailed simulation results can be found in related previous study of authors [13]. Among those merge sort alternatives, multi merge sort algorithm offers better miss rate performances for data sizes less than 1 M; however, tiled merge sort is more scalable for larger data sets; on the other side, the best run time performance is obtained by tiled merge sort with padding algorithm.

4 New Proposed Algorithm

According to our simulation results, we may conclude that the most important factor which effects the running time is L1 cache miss count. From our measurements, we obtained that tiling method performs well with regarding L1 cache misses. In this section, we propose some procedural modifications on tiled merge sort algorithm to reach better performance by decreasing L1 cache miss rates. To further improve the performance, we propose to make the following operations instead of using basic merge sort algorithm after tiling operation:

Assume that we have *m* arrays each with size *s* that equals to half of the size of the L1 cache. Also assume that these arrays are numbered from 0 to *m*-1. If we merge and sort all arrays into the array 0, then array 0 has the first sorted portion of the array. We make the same operations for the other arrays then the target array becomes sorted. We need to make *m*(*m*-1)/2 merge operations totally. In this way, we aim to achieve merging operation on L1 data cache to reduce read misses and we aim to avoid write misses which occur while writing to the target array. The code for our new proposed algorithm is given below where this algorithm can be integrated with tiled merge sort.

New Proposed Algorithm

```
begin
  w=min;
  m=n/w;
  source=a;
    for (i=0;i<m-1;i++){
      k=i*m;
      for (j=(i+1);j<m;j++){
        j1=k;
        jj2=j2+w;
        j2=j*w;
        jj3=j2+w;
        while ((j1<jj2)&&(j2<jj3)){
          if (source[j1]<=source[j2])
            j1++;
          else{
            temp=source[j1];
            source[j1]=source[j2];
            source[j2]=temp;
            j2++;
          }
        }
        if (j1==jj2){
          while (j2<jj3)
            source[j1++]=source[j2++];
        }
        if (j2==jj3) j1=jj2;
      }
    }
  if (target==a){
    for (i=0; i<n; i++)
      a[i]=b[i];
  }
end
```

5 Conclusions

In this paper, we first provided a survey of cache effected sorting algorithms. Secondly, we provided simulation results including basic merge sort algorithm, tiled merge sort algorithm, multi merge sort algorithm and tiled merge sort with padding algorithm using Valgrind simulator. We also simulated those various merge sort algorithms on different input data set distributions as random, equilikely, poisson, and zero distributions. As a general conclusion of those simulation experiments among merge sort variations, tiled merge sort performance seems appropriate based on the average of sorting time and cache miss rate results.

It's known that basic quicksort wins against basic merge sort in general runnig time. To observe the performance differences between alternative quicksort and flash sort variations, we simulated the miss rates and running times of flash sort, flash quicksort, inplaced flash quicksort, and memory tuned quicksort. Among them flash sort has greater miss rates, however it leads better running times than quicksort,

We also provided the simulation results of various merge sort alternatives based on Pentium 4 configuration. The instruction miss rate on L1 cache performance is nearly same for merge sort algorithms. Tiled merged algorithm performs best among other algorithms considering L1 data cache miss rate. Although multi merge sort algorithms performs best considering L2 data cache miss count, it has the worst performance when L1 data cache count is considered thus its overall run time performance is worst among other merge sort algorithms. Overall, tiled merge sort with padding algorithm performs best in run time. Overall findings in our simulations are consistent with the findings and evaluations pointed in the studies by LaMarca and Ladner [9-11], and Xiao et al. [12].

We also proposed a new algorithm to further reduce the wall clock time and showed its design in this paper. Our work is ongoing and we are planning to implement the proposed algorithm and compare with the other merge sort algorithms.

Acknowledgment. The authors thank Prof. Dr. Mehmet Emin Dalkilic for his advises to focus them on cache effected sorting area.

References

1. Demuth, H. (1956). Electronic Data Sorting. PhD thesis, Stanford University.
2. Knuth, D. E. (1973). The Art of Computer Programming, Addision Wesley, Reading, MA.
3. Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001) Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03293-7.
4. Goodrich, M. T. and Tamassia, R. (2002). 4.5 Bucket-Sort and Radix-Sort. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons. pp. 241–243.
5. Han. Y. (2002) Deterministic sorting in O(n log log n) time and linear space. *In Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, Montreal, Quebec, Canada.

6. Thorup. M. (2002) Randomized Sorting in O(n log log n) time and Linear Space Using Addition, Shift, and Bit-wise Boolean Operations. *Journal of Algorithms*, 42(2): 205-230.
7. Han Y. and Thorup, M. (2002). Integer Sorting in O(n sqrt (log log n)) Expected Time and Linear Space. *In Proceedings of the 43rd Symposium on Foundations of Computer Scienc,* 16-19 November, IEEE Computer Society, Washington, DC.
8. Hoare, C. A. R. (1962). Quicksort, *Oxford The Computer Journal*, 5(1).
9. LaMarca, A. and Ladner, R. E. (1996) The Influence of Caches on the Performance of Heaps, *Journal of Experimental Algorithms*, 1.
10. LaMarca A. and Ladner, R. E. (1997) The Influence of Caches on the Performance of Sorting, *In Proceedings of the 8th Annual ACM Symposium on Discrete Algorithms*, SODA 97, New Orleans.
11. LaMarca, A. and Ladner, R. E. (1999) The influence of caches on the performance of sorting", *Journal of Algorithms* 31(1): 66-104.
12. Xiao, L., Zhang, X. and Kubricht, S. A. (2000) Improving memory performance of sorting algorithms, *ACM Journal on Experimental Algorithmics,* 5(3): 1-22.
13. Tekbacak, F., Korkmaz, I., Dagdeviren, O. and Metin, S.K. (2010)Application and Performance Analysis of Cache Effected Merge Sort Algorithms", *In Proceedings of the 4th International Student Conference on Advanced Science and Technology*, ICAST 10, Izmir, May 2010.
14. http://www.intel.com/products/desktop/processors/pentium.htm
15. http://valgrind.org/

Biographies

Ilker Korkmaz – Ilker Korkmaz received the BSc. degree in Electrical and Electronics Engineering and MSc. degree in Computer Science from Ege University. He is a PhD candidate at International Computer Institute at Ege University. He is also an instructor at the Department of Computer Engineering at Izmir University of Economics. His research interests include computer networks, network security, and wireless sensor networks.

Senem Kumova Metin – Senem Kumova Metin received her BSc. degree in Electrical and Electronics Engineering from Ege University and MSc. and PhD degrees from International Computer Institute at Ege University. She is currently working as a lecturer in Izmir University of Economics. She is mainly interested in natural language processing applications and computational linguistics.

Orhan Dagdeviren – Orhan Dagdeviren received the BSc. and MSc. degrees in Computer Eng. from Izmir Institute of Technology. He received PhD degree from Ege University International Computing Institute. He is an assistant professor in Izmir University, Department of Computer Engineering. His interests lie in the computer networking, distributed systems and wireless sensor networks areas.

Fatih Tekbacak – Fatih Tekbacak received the BSc. and MSc. degrees in Computer Engineering from Izmir Institute of Technology. He is a PhD candidate in Computer Engineering at Ege University. He is also a research assistant in Izmir Institute of Technology. His interests lie in the software agents, access control of role based agents on multi agent systems and semantic web.