Application and Performance Analysis of Cache Effected Merge Sort Algorithms

Fatih Tekbacak*,[1], Ilker Korkmaz[2], Orhan Dagdeviren[1] and Senem Kumova Metin[2]
[1]Dept. of Comp. Engineering, Izmir Institute of Technology
[2]Faculty of Eng. and Comp. Sciences, Izmir University of Economics

Abstract- One of the most significant factors that affect run time of a program is the cache behavior. Especially, the efficient use of cache in the programs that process huge data with an iterative manner is very important. In this paper, our objective is to measure the cache effectiveness of sorting algorithms. To test sorting algorithms, a suitable cache configuration is determined while using cache effected merge sort algorithms in Valgrind simulator. Our motivation is to investigate algorithm performances by experiencing and comparing them on Level-1 and Level-2 caches.

Index Terms- merge sort, cache effectiveness, simulation.

I. Introduction

SORTING is a very common operation in computer science. Most familiar example is binary search to find key data within a sorted list. While the run time complexity of binary search algorithm is *O(logn)*, the random search algorithms in a list has *O(n)* like complexities. If search operation is the primary concern, sorted list will be necessity. However, sorting operation has also storage cost. Although the first considered cost is generally complexity, the implementation of a sorting algorithm can be based on a few different paradigms depending on the system. One of the important parameters in algorithms running with huge data set is especially cache usage. In this study, our aim is to run cache effected solutions for merge sort algorithms on a Pentium 4 architecture and compare their performances on cash use.

II. Related Work

Sorting algorithms have a great effect on runtime in their operations. Therefore, qualifying these algorithms has an effect that mostly decreases run time. While one of the most effective ways of decreasing run time is to decrease the instruction number, primary concern of most researchers is to rearrange algorithms in such a way that reduces instruction count traditionally. But previous researches showed that decreasing instruction count doesn't supply enough improvement when it has a positive effect on run time.

Differently from this traditional approach, the reality of cache usage in sorting algorithms was stated firstly in LaMarca and Ladner [1] based on their mergesort, tiled mergesort, and multi mergesort implementations. Xiao, Zhang, and Kubricht [2] contributed to the literature with their tiled mergesort with padding and multi mergesort with TLB padding algorithms. An algorithm considering cache properties and cache size will reduce memory access rate or cache miss rate. Basic mergesort is a comparative sort algorithm with *O(nlogn)* complexity and designed as *divide and conquer* paradigm. Tiled mergesort puts initial data to two different sub arrays and sort these arrays between each other. Multi mergesort, different from tiled mergesort, merges all sub arrays in one operation. Tiled mergesort with padding organizes data locations and aims to decrease the collision miss rate. Multi mergesort with TLB padding has a goal to reduce TLB misses created by multi mergesort algorithm. These algorithms can be simulated by Valgrind [3], a program designed to manage the code management and thread defects automatically, and Cachegrind, a Valgrind tool that finds the miss rates regarding to program simulating Level-1 and Level-2 caches.

III. Experiments

A. Method

In order to understand the cache performances on sorting algorithms, the basic variations of merge based algorithms as base mergesort, tiled mergesort, multi mergesort, and tiled mergesort with padding, are examined in Valgrind simulator. In our tests, two-level cache structure of Northwood core Pentium 4 processor architecture [4] is simulated. Level-1 cache is configured as 4-way associative with a total capacity of 8 K, which has a line size of 64 Bytes; Level-2 cache is used as 8-way associative with a capacity of 256 K, and with the line size of 128 Bytes. Random data sets of 1 K to 4096 K are used as inputs to the implementations. Each different experiment is conducted for 5 times and the average results are noted regarding to accuracy.

The data miss rates of Level-1 and Level-2 caches are compared for each algorithm to understand the cache performances. Moreover, the running times of the implementations are measured. Although all mergesort implementations have a time complexity of *O(nlogn)*, the more miss in the last level of a cache the more access to the memory, and so the more time actually.

B. Simulation Results and Evaluation

All misses of data in both levels of caches are measured since any miss in data cache is important on the performance. In the experiments, the total miss number on all sorting process of an input data is measured and the miss rate is calculated as the number of misses per one data element. The miss rates in Level-1 and Level-2 data caches on different sizes of input are depicted in Fig. 1 and Fig. 2 respectively. The misses in the instruction cache are not under debate in this study.

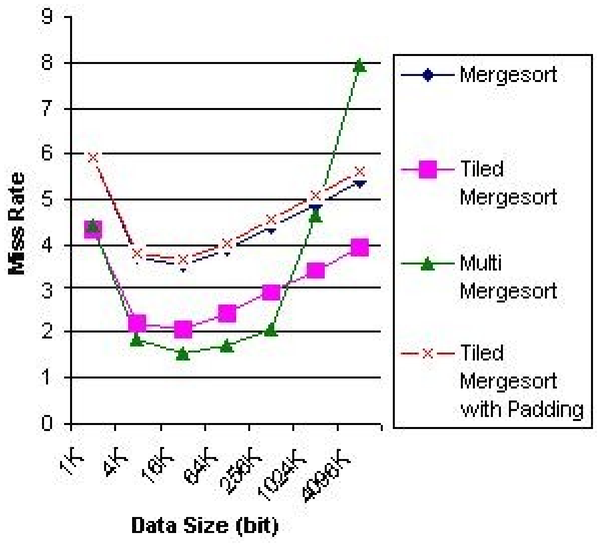

Fig. 1. Level-1 cache miss rates (number of misses per one element) of different sizes of input data for different merge sort algorithms.

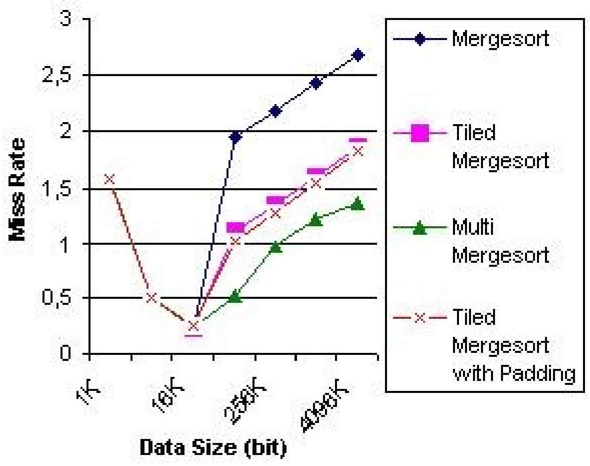

Fig. 2. Level-2 cache miss rates (number of misses per one element) of different sizes of input data for different merge sort algorithms.

As seen in Fig. 1, multi mergesort algorithm has a better performance for data sizes less than 1 M. The reason is that it gains the advantage of temporal locality due to the use of a priority queue utility structure. However, tiled mergesort is more scalable. On the other side, any miss on Level-1 cache leads to the access to Level-2 cache, which is slower but larger than Level-1. As in Fig. 2, multi mergesort overcomes the Level-2 miss possibilities with better miss rates.

Fig. 3 shows the running times (in seconds) of the algorithms. Although they have similar results for inputs less than 256 K, multi mergesort has the worst time when data enlarges 256 K threshold. The reason is the high total miss rates (Level-1 plus Level-2) of multi mergesort algorithm for input size larger than 256 K. Any miss means an access to the other storage area in a slower level, and so any miss leads to extra time in the process. Therefore, according to the input data size, multi or tiled variations of merge sort algorithm may be elected as the winners of performance results

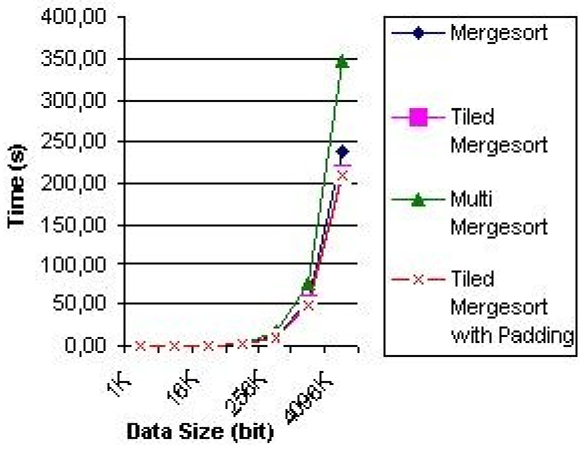

Fig. 3. Running times of different merge sort implementations on the same two-level cache architecture simulations.

IV. Discussion And Future Work

In this paper, the effect of performance increment for active merge sort algorithms, which previously detailed in [1], and [2], are simulated within Valgrind [3] platform. The experiment results are presented. As depicted in Fig. 3, the best runtime performance was obtained by tiled mergesort with padding.

Cache design parameters include cache size, line size, assocativity, and multi-level structure. According to these arguments, the quicksort algorithm has also to be covered. Different input data distributions can cause different miss rates. For example, tiled mergesort has better performance than multi mergesort except poisson distribution. We could not indicate such experimental results here due to the lack of space in this paper. As a future plan, different distributions may be examined for quicksort algorithm and the results can be compared with the ones pointed in our work. Quicksort needs an analysis for cache parameters while we know that quicksort has a better practical running time than mergesort.

V. Acknowledgment

The authors gratefully acknowledge the contributions of Prof. Dr. Mehmet Emin Dalkılıç for his advice on studying the cache performance subject.

VI. References

[1] A. LaMarca and R. E. Ladner, "The influence of caches on the performance of sorting", *Journal of Algorithms 31(1)* (1999), pp. 66-104.
[2] L. Xiao, X. Zhang, and S. A. Kubricht, "Improving memory performance of sorting algorithms", *ACM Journal on Experimental Algorithmics 5(3)* (2000), pp. 1-22.
[3] http://valgrind.org/
[4] http://en.wikipedia.org/wiki/Pentium_4#Northwood