IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol-6, Special Issue-3, Apr 2018 E-ISSN: 2347-2693

Performance Evaluation of Lazy-Funnelsort Algorithm on Multicore System

Riaz Ahmed[1*], Lalitsen Sharma[2]

[1*]Department of Computer Sciences & IT, University of Jammu, Jammu, J&K, India
[2] Department of Computer Sciences & IT, University of Jammu, Jammu, J&K, India

[*]*Corresponding Author riaz.mirza.11@gmail.com*

Available online at: www.ijcseonline.org

***Abstract*—** Sorting is one of the basic problems that have been extensively studied. There are many sorting algorithms which are efficient in computation but cannot use cache efficiently. Efficient use of cache is an important factor for determining the performance of an algorithm. Cache-oblivious algorithms are designed that are both work and cache efficient. The aim of this paper is to evaluate the performance of cache-oblivious sorting algorithm called Lazy Funnelsort on multicore processors. The evaluation is made against the well known fast sorting algorithms: quick sort, merge sort on multicore processor machine. The experiments are conducted against different input sizes and number of processing cores and threads using Intel Cilk Plus, which is extension to C and C++ to express task and data parallelism. The performance of algorithms is examined in terms of execution time, speedup, efficiency and scalability. The results show that parallel implementation of Lazy Funnelsort is better than its sequential implementation and also scalable on multiple cores. Though it has been outperformed by quick sort and merges sort algorithms but shows moderate promise as a parallel algorithm.

***Keywords*—**Cache-oblivious, Funnelsort, Performance, Speedup, Efficiency, Cache-miss-ratio

I. INTRODUCTION

Since the multicore has become the default configuration of almost all computers, every year manufacturing companies are producing chips with increased number of cores/threads in order to increase its performance potential. Multicore in its simplest form is a collection of cores sharing an arbitrary large main memory containing data and featuring one or more level of caches which could be either private [1] or shared among all the cores [2]. In this era of multicore, research is on the way to develop parallel and multithreaded algorithms so as to utilize the performance potential of multicore hardware. As the multicore hardware is growing fast, writing efficient and portable algorithms for exploiting modern hardware has become important. In recent past, a number of algorithms have been developed by researchers which has optimal efficiency and are also oblivious to multicore parameters like No. of cores, No. of levels of cache or their size, block size, etc. e.g. in [3-12].

Sorting is one of the core steps of a number of algorithms. There are number of sorting algorithms which are computationally efficient but are not cache efficient. Cache efficiency is an important factor for algorithmic performance. Lazy-Funnel sort [13] in one of the sorting algorithms which uses cache efficiently. It is the modified version of Funnel sort algorithm introduced in [14]. This algorithm is portable as well as cache-oblivious as it does not contain any tuning parameter like cache size, cache line length etc. in its code for performance improvement. The Lazy-Funnel sort algorithm breaks down the input problem into independent sub problems very easily, so it has the big potential for parallelizing.

This paper deals with the performance evaluation of multithreaded Lazy-Funnel sort algorithm on dual-core and quad-core processor machines with varying input sizes and threads. The performance in terms of efficiency and cache miss ratio of Lazy-Funnel sort are calculated and analyzed. The evaluation is made against quick sort and merge sort algorithms.

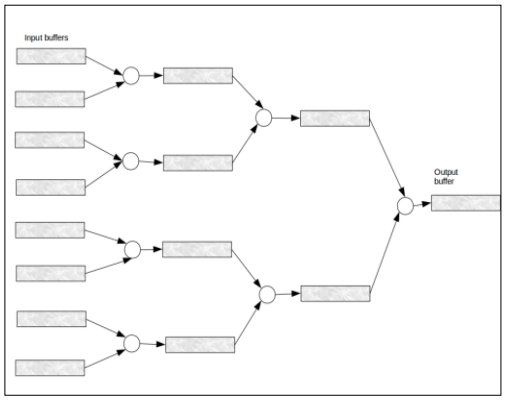
The rest of the paper is organized as follows: Section II briefly described the Lazy-Funnel sort Algorithm. Earlier work is briefed in section III. Methodology of research and experimental environment is described in section IV. Section V presents the results and analysis of algorithms and conclusion is drawn in section VI.

II. LAZY-FUNNELSORT

Lazy-Funnel sort [13] is similar to merge sort [15] except the process of merging which is performed by a device called k-merger. In order to sort n elements, the Lazy-Funnel sort

algorithm firstly split input list into $K=n^{1/3}$ segments, then recursively sort each segment and finally merges these sorted segments using k– merger. A k-merger is a merge tree consisting of k-1 binary mergers. The structure of k-merger for use in Lazy-Funnel sort was described in [13]. A k-merger is laid out recursively in memory in order to achieve input-output efficiency.

The base of the algorithm is binary merger which intakes two sorted arrays, merges them and outputs a sorted array. An 8-merger



consisting of 7 binary mergers is shown in Fig. 1.

Figure 1. An 8-merger consists of 7 binary mergers

A binary merge tree is formed using binary mergers in which mergers are at node and buffers are at edges. The arrays that to be merged are at leaves of the merge tree. A merger is called recursively to perform merge steps till its both input arrays are empty or output buffer is full. The Procedure to fill an array is shown in Fig.2 [16].

```
Procedure FILL (node x) {
if (both inputs =non-empty) {
  while (output buffer of x = not full) {
    if (left input's head < right input's head) {
        move left input's head to output buffer
        if (left input buffer= empty)
                        FILL(x's left child)
    }else {
                    Move right input's head to output
                    If (right input buffer= empty)
                        FILL(x's right child )
    }
}else if (only one input buffer= empty){
        move max (other's input elements) to the output
       if (input buffer= empty)
                        FILL(x's corresponding child)
}else
          return
```

Figure 2. The Lazy Fill Algorithm

III. RELATED WORK

The first study of sorting algorithms in cache-oblivious settings on microprocessor with a single core and multiple levels of caching appeared in [14] where two cache-oblivious sorting algorithms: a new funnel sort and a distribution based sorting were described and analyzed in Ideal Cache Model [14]. The performance of these algorithms was optimal both in running time and cache complexity without knowing the machine parameters in their codes. Later on, the simplified version of funnel sort called Lazy-Funnelsort was introduced in [13]. The performance of Lazy-Funnelsort was empirically evaluated in [16] where the efficiency of this algorithm remained competitive with other fast distribution based sorting algorithms: merge sort, quick sort and cache-aware sorting.

IV. METHODOLOGY

This section explains the experimental environment including methodologies of implementation and measurement as follows:

A. Experimental Setup

We performed our experiments on two multicore systems having different specifications as shown in Table-1.

Table 1. Specifications of Experimental Machines

Model	Intel Core I-5-240M	Intel Core I-7-240M
CPU	2.50 GHZ	3.40 GHZ
CORES	2	4
THREADS	4	8
L1 CACHE	32 KB	256 KB
L2 CACHE	256 KB	1024 KB
L3 CACHE	3072 KB	8192 KB
MEMORY	4 GB	4 GB
OPERATING SYSTEM	LINUX 64-BIT	LINUX 64-BIT

B. Implementation and measurement

All programs are implemented using Intel Cilkplus, which simplifies the programming efforts for shared memory multiprocessor systems. The three keywords: *cilk_for*, *cilk_spawn* and *cilk_sync a*re used to express parallelism in Intel cilkplus. The for loop is parallelized by using cilk_for keyword, cilk_spawn specifies that no function can execute in parallel with the remainder of the calling function and all spawned functions must complete before execution continues is specified by cilk_sync keyword. These keywords provide opportunities for parallelism but the Intel cilkplus runtime determines which portion of the program actually run in parallel using efficient work-stealing scheduler.

We used *srand* function of C library to generate random list of values. The C library function *gettimeofday* is used to measure wall clock time as our performance matrix. We performed experiments for more than 30 times and the lowest value amongst all runs was recorded.

The efficiency of algorithms is calculated by using the formula:

$$\text{Efficiency (E)} = SU \ / \ K \quad ---(1)$$

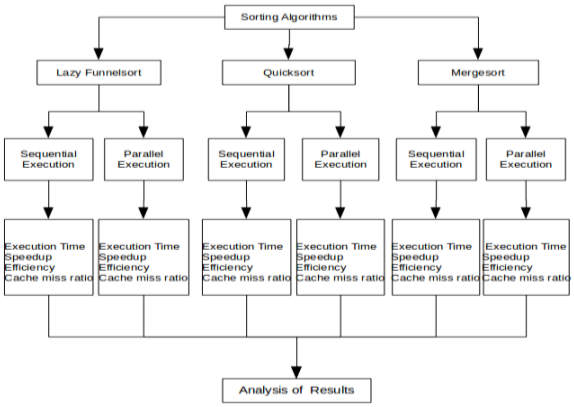
where, SU is speedup and K is the number of threads.

The speedup of algorithms is calculated by using the formula:

$$\text{Speedup (SU)} = T1 / Tn \quad – \quad (2)$$

where, T1 is sequential execution time and Tn is execution time with n number of processing threads.

Cache performance is measured using perf 4.10.17 Linux tool [22] to measure cache-miss-ratio of L1 data cache for sorting algorithms.



The overview of proposed work is shown in Figure 3.

Figure 3. Overview of proposed work

V. RESULTS AND DISCUSSION

Here we present experimental results and analysis. To evaluate the performance, a number of experiments are performed both on dual-core and quad-core machines with varying input sizes. The sizes of tested lists are $(10)^2$, $(10)^3$, $(10)^4$, $(10)^5$, $(10)^6$ and $(10)^7$. The execution time of sorting algorithms with varying input sizes and with available threads both on dual-core and quad-core machines are recorded. The efficiency and speedup of sorting algorithms is calculated using the formulas of equations 2 & 1 respectively.

A. *Pereformance on Dual-Core Machine*

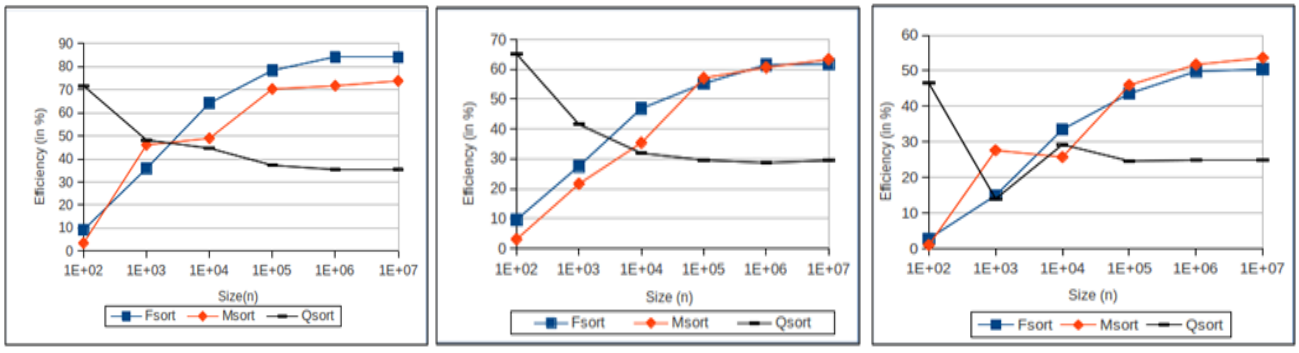
The calculated efficiency of sorting algorithms when executed on dual-core machine with 2, 3 and 4 threads (K) and varying input sizes (n) is shown in Fig.4. It is observed that:

- For small input values up to $(10)^3$ and with K=2, 3 & 4, the performance of quick sort algorithm is better than other algorithms.

For input size above $(10)^3$, the performance of Lazy-Funnel sort is best except the case when $n > (10)^5$ and K=4 where efficiency of Lazy-Funnel sort is just underneath the merge sort algorithm.

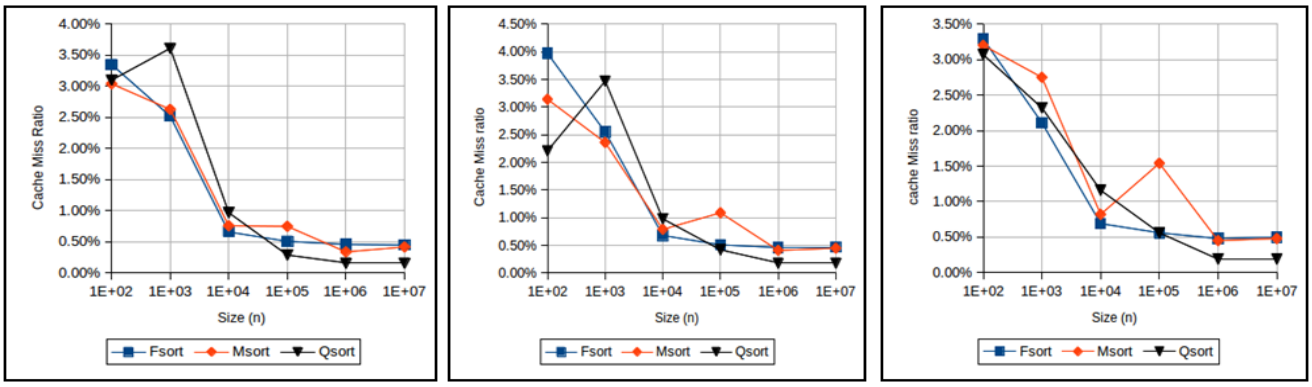
The following observations are made in all cases of experiments:

- In 61% of all cases, the performance of Lazy-Funnel sort remained better than merge sort algorithm.
- In 72% of all cases, the performance of Lazy-Funnel sort remained better than quick sort algorithm.
- The Lazy-Funnel sort showed equal performance in 17% and 6% of all cases when compared with merge sort and quick sort algorithms respectively.



a) With K = 2 a) With K = 3 c) With K = 4

Figure **4**. Efficiency of Sorting Algorithms on Dual-Core Machine



a) With K = 2 b) With K = 3 c) With K = 4

Figure 5. Cache Miss Ratio of Sorting Algorithms on Dual-Core Machine

- In 78% of all cases, the performance of Lazy-Funnel sort remained equal to or better than merge sort or quick sort algorithms.
- In 56% of all cases, the performance of Lazy-Funnel sort remained equal to or better among all algorithms.
- In 95% of all cases, the performance of Lazy-Funnel sort remained in top two among all algorithms.

The cache performance of sorting algorithms is evaluated on dual-core machine using perf suit tool. The cache miss ratio of level one data cache is recorded by executing sorting algorithms with varying input sizes (n) and threads (K) and shown in Fig. 5.

The following observations are made in all cases of experiments:

- In 50% of all cases, the Lazy-Funnel sort has lesser number of cache misses than merge sort algorithm.
- In 33% of all cases, Lazy-Funnel sort has lesser number of cache misses than quick sort algorithm.
- The Lazy-Funnel sort have equal number of cache misses in 17% and 6% of cases when compared with merge sort and quick sort algorithms respectively.
- In 67% of all cases, the Lazy-Funnel sort have equal to lesser number of cache misses than merge sort algorithm.
- In 39% of all cases, Lazy-Funnel sort has equal to or lesser number of cache misses than quick sort algorithm.
- In 44% of all cases, the cache misses of Lazy-Funnel sort remained equal to or lesser among all algorithms.

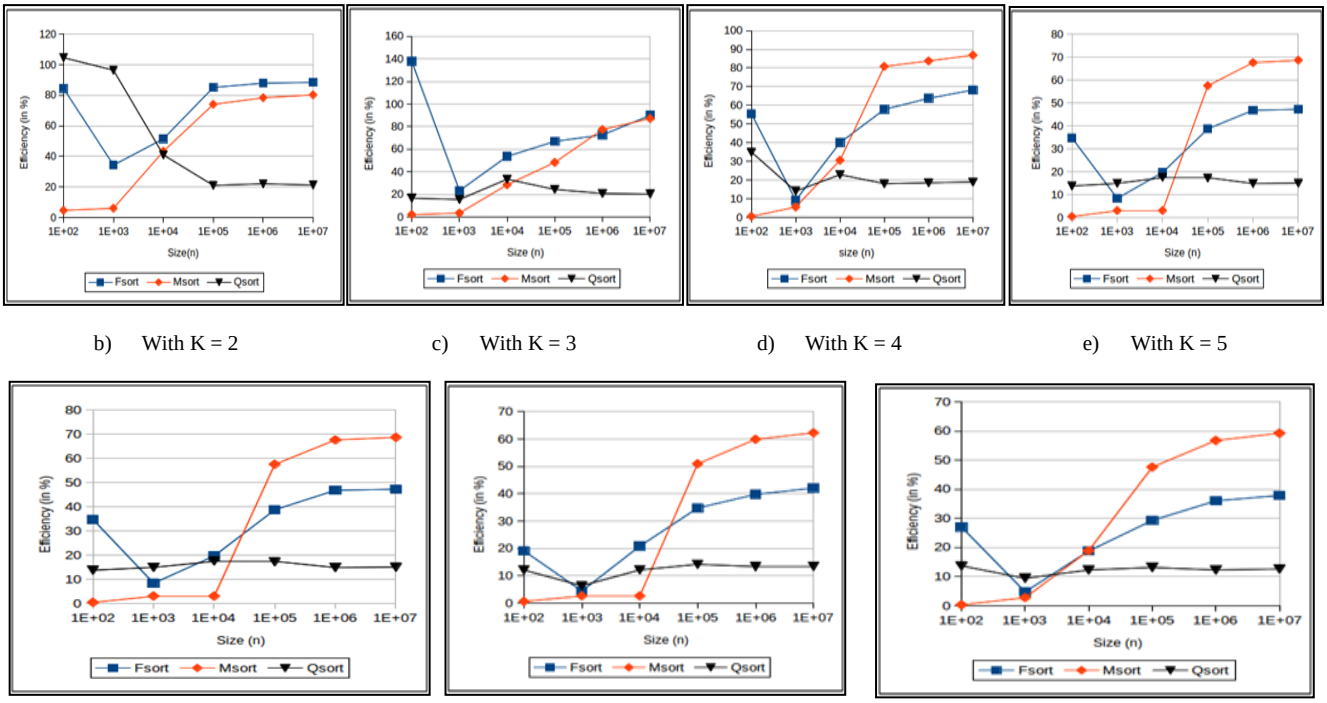
B. Pereformance on Quad-Core Machine

The calculated efficiency of sorting algorithms when executed on quad-core machine with 2, 3, 4 5, 6 7 and 8 threads (K) and varying input sizes (n) is shown in Fig.6. It is observed that:

- For small input values up to (10)4 and with all cases of K, the performance of Lazy-Funnel sort is better except in few cases where quick sort algorithm outperforms it.
- For input size above $(10)^4$, the performance of Lazy-Funnel sort is underneath the merge sort algorithm.

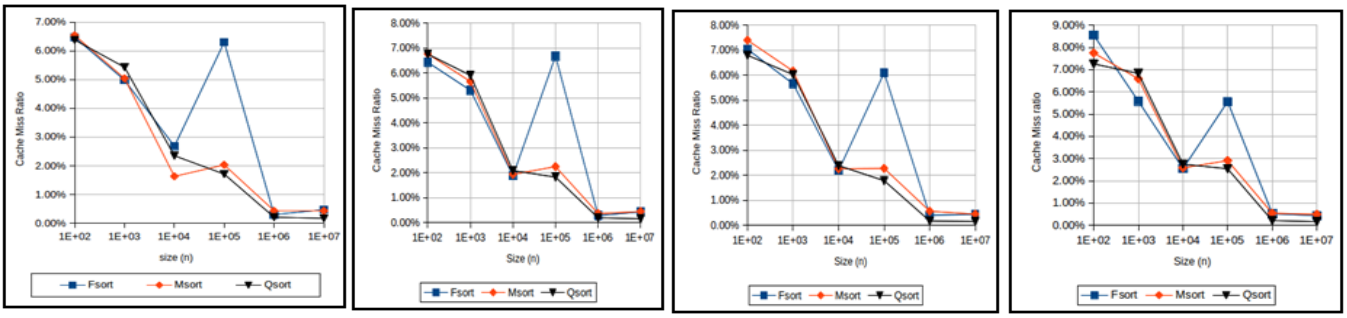
The following observations are made in all cases of experiments:

- In 60% of all cases, the performance of Lazy-Funnel sort remained better than merge sort algorithm.
- In 83% of all cases, the performance of Lazy-Funnel sort remained better than quick sort algorithm.
- The Lazy-Funnel sort shows equal performance in 7% and 5% of all cases when compared with merge sort and quick sort algorithms respectively.
- In 67% of all cases, the performance of Lazy-Funnel sort remained equal to or better than merge sort algorithm.
- In 88% of all cases, the performance of Lazy-Funnel sort remained equal to or better than quick sort algorithm.
- In 59% of all cases, the performance of Lazy-Funnel sort remained equal to or better among all algorithms.
- In almost all cases, the performance of Lazy-Funnel sort remained in top two among all algorithms.

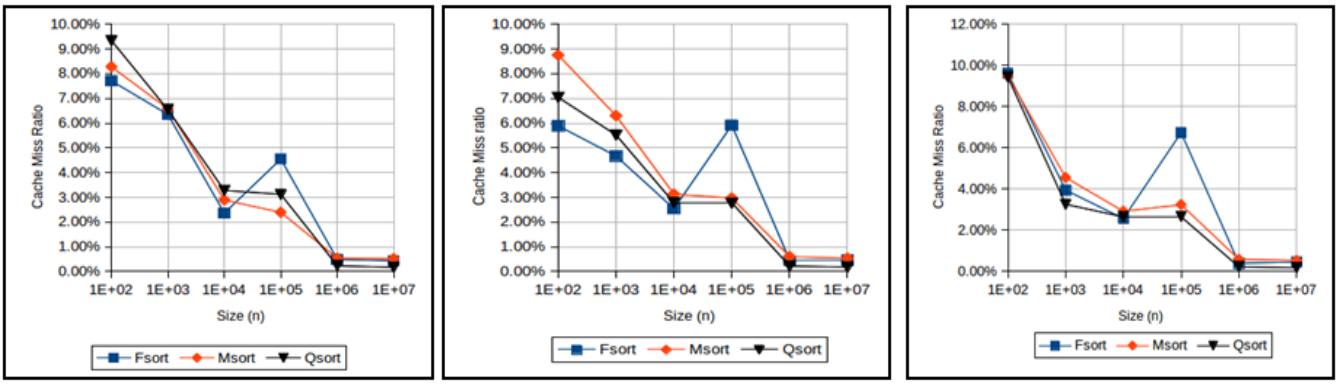


b) With K = 2 c) With K = 3 d) With K = 4 e) With K = 5

Figure 6. Efficiency of Sorting Algorithms on Quad-Core Machine



a) With K = 2 b) With K = 3 c) With K = 4 a) With K = 5



e) With K = 6 f) With K = 7 g) With K = 8

Figure 7. Cache Miss Ratio of Sorting Algorithms on Quad-Core Machine

The cache performance of sorting algorithms is evaluated on quad-core machine using perf suit tool. The cache miss ratio of level one data cache is recorded by executing sorting algorithms with varying input sizes (n) and threads (K) and shown in Fig. 7. The following observations are made in all cases of experiments:

- In 43% of all cases, the Lazy-Funnel sort has lesser number of cache misses than merge sort algorithm.
- In 38% of all cases, Lazy-Funnel sort has lesser number of cache misses than quick sort algorithm.
- The Lazy-Funnel sort have equal number of cache misses in 33% and 14% of cases when compared with merge sort and quick sort algorithms respectively.
- In 78% of all cases, the Lazy-Funnel sort have equal to lesser number of cache misses than merge sort algorithm.
- In 52% of all cases, Lazy-Funnel sort has equal to or lesser number of cache misses than quick sort algorithm.
- In 47% of all cases, the cache misses of Lazy-Funnel sort remained equal to or lesser amongst all algorithms.

VI. CONCLUSION and Future Scope

The performance of cache-oblivious sorting algorithm called Lazy-Funnel sort has been empirically studied with respect to efficiency and cache misses. The efficiency is evaluated by executing the multithreaded Lazy-Funnel sort program both on dual-core and qua-core processor machines with varying input sizes and threads. It is seen in the experimental studies that on dual-core machine, the performance of Lazy-Funnel sort remained equal to or greater than merge sort or quick sort algorithms in 78% of all cases. It is also shown that in 56% of all cases, the Lazy-Funnel sort outperformed all algorithms. In general, it is shown that Lazy-Funnel sort remained at top two algorithms in 95% of all cases as for as performance is concerned on dual-core machine. On quad-core machine, in 67% and 88% of all cases the performance of Lazy-Funnel sort remained equal to or better than merge sort and quick sort algorithms respectively. It is also shown that in 59% of all cases, the Lazy-Funnel sort outperformed all algorithms. In general, it is shown that Lazy-Funnel sort remained on top two algorithms in almost all cases as for as performance is concerned on quad-core machine. The cache miss ratio is evaluated through use of hardware performance counter both on dual-core and qua-core processor machines. It is also observed from the experiments that performance of sorting algorithms is influenced by cache performance.

From our study, we arrived at the conclusion that the Lazy-Funnel sort shows fairly good parallel performance as its efficiency remained in top two algorithms in almost all cases of experiments performed both on dual-core and quad-core processor machines. The reason for good performance of Lazy-Funnel sort is due to the fact that it has better data locality. The poor performance in some cases is due to the parallel scheduling overheads.

We have performed experiments on dual-core and quad-core systems but the work could be extended too many core systems. In this paper, we have evaluated one cache parameter of level 1 data cache and in future we aimed to evaluate other cache parameters like L2, L3, dTLB, etc

REFERENCES

[1] L. Arge, M. Goodrich, M. Nelson, and N.Sitchinava, *"Fundamental parallel algorithms for private-chip multiprocessors"*, In the Proceedings of the 20th ACM SPAA, pp. 197-206, 2008.

[2] G. Blelloch and P. Gibbons, "*Effectively sharing a cache among threads*", In the Proceedings of the 16th ACM SPAA, pp. 235-244, 2004.

[3] U. A. Acar, G. E. Blelloch, and R. D. Blumofe, *"The data locality of work stealing"*, Theory of Computing Systems, vol. 35, pp.3, 2002.

[4] G. Bilardi, A. Pietracaprina, G. Pucci, and F. Silvestri, *"Network-oblivious algorithms"*, In the Proceedings of the 21st IEEE IPDPS, 2007.

[5] R. A. Chowdhury and V. Ramachandran, *"The cache-oblivious Gaussian elimination paradigm: Theoretical framework, parallelization and experimental evaluation"*, In the Proceedings of the 19th ACM SPAA, pp. 71–80, 2007.

[6] R. A. Chowdhury and V. Ramachandran, *"Cache-oblivious dynamic programming"*, In the Proceedings of the 17th ACM-SIAM SODA, pp. 591–600, 2006.

[7] G. Blelloch, R. Chowdhury, P. Gibbons, V. Ramachandran, S. Chen, and M. Kozuch, *"Provably good multicore cache performance for divide-and-conquer algorithms"*, In the Proceedings of the SODA, pp. 501–510, 2008.

[8] M. Frigo and V. Strumpen, *"The cache complexity of multithreaded cache oblivious algorithms"*, Theory Compute. Syst., vol. 45, no. 2, pp. 203–233, 2009.

[9] R. Cole and V. Ramachandran, *"Resource oblivious sorting on multicores"*, In the Proceedings of the ICALP, Track A, 2010.

[10] Richard Cole and Vijaya Ramachandran, *"Efficient Resource Oblivious Algorithms for Multicores with False Sharing"*, In the Proceedings of the IEEE 26th IPDPS, pp. 201 – 214, 2012.

[11] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava, *"Fundamental parallel algorithms for private-cache chip multiprocessors"*, In the Proceedings of the ACM SPAA, pp. 197–206, 2008.

[12] G. Belloch, P. Gibbons and H. Simhadri, *"Brief announcement: Low depth cache-oblivious sorting"*, In the Proceedings of the ACM SPAA, ACM, 2009.

[13] G. S. Brodal and R. Fagerberg, *"Cache-oblivious distribution sweeping"*, In the Proceedings of the 29th International Colloquium on Automata, Language and Programming, ICALP, vol. 2518, Springer, New York, pp. 426-438, 2002.

[14] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran, *"Cache-oblivious algorithms"*, In the Proceedings of the 40th IEEE FOCS , pp.285-297, 1999.

[15] A. Aggarwal and J. S. Vitter, *"The input/output complexity of sorting and related problems"*, Comm. ACM 31, 9, 1116-1127, 1988.

G. S. Brodal, R. Fagerberg and K. Vinther, *"Engineering Cache-Oblivious Sorting Algorithm"*, Journal of Experimental Algorithmics, vol. 12, ACM, New York, 2008

