ISSN:0975-9646

Evaluation of Power Consumption of Modified Bubble, Quick and Radix Sort, Algorithm on the Dual Processor

Ahmed M. Aliyu[*1] Dr. P. B. Zirra[*2]
[1]*Post Graduate Student*
[*1,2], *Computer Science Department, Adamawa State University, Mubi Nigeria*

Abstract **- Sorting is one of the fundamental operations in computer science. Sorting refers to the arrangement of data in some given order such as increasing or decreasing order, with numerical data or alphabetically, with character data [7]. There are many sorting algorithms. All sorting algorithms are problem specific. The selection of any of these algorithms depends on the properties of data and operations performed on them. This study is intended to determine the performance of dual Processors on three (3) sorting algorithms (Bubble, Quick and Radix) and measure the power consumption on execution of each of these algorithms. The three algorithms (Bubble, Quick and Radix sorts) were selected and modified, by infusing random number generation and power estimation procedures into each and then determine its order against the unmodified algorithms. The time taken to sort the generated random integers was then used to estimate the power consumed by the processor. The study revealed that radix sort takes less time and consume less power when performing the sorting; and these were ranked in ascending order for clarity and easy identification of better performance.**

Keywords – Bubble Sort, Radix Sort, Quick Sort, Big O, Dual Processor, Processor

I. INTRODUCTION

Sorting is one of the most important and well-studied problems in computer science. The problem of sorting is a problem that arises frequently in computer programming. Many different sorting algorithms have been developed and improved to make sorting fast. Some sorting algorithms are simple and intuitive such as bubble sort while others such as quick sort are complicated but produce very fast results. The commonly used algorithms can be divided into two classes by the complexity of their algorithm. Algorithmic complexity is generally written in a form known as Big-O notation, where the O represent the complexity of an algorithm and a value n represent the size of the set the algorithm is run against. The two classes of sorting algorithm are $O(n^2)$, which include the bubble sort, insertion sort, selection sort, and shell sort; and O(n log n), which include the heap sort, merge sort and quick sort. There is direct correlation between the complexity of an algorithm and its relative efficiency. There is no one sorting method that is best for every situation. Some of the factors to be considered in choosing a sorting algorithm include the size of the list to be sorted, the programming effort, the number of words of main memory available, the size of disk or tape units, the extent to which the list is already ordered, the distribution of values and the power consumption rate of the processor per instruction set.

II Statement of the Problem

Power consumption of software is becoming an increasingly important issue in designing mobile embedded systems where batteries are used as the main power source. As a consequence, a number of promising techniques have been proposed to optimize software for reduced Power consumption. Such low-power software techniques require a Power consumption model that can be used to estimate or predict the Power consumed by software Therefore there is a need to design a new algorithm that would consume less power by the Microprocessor when running program instructions.

[10] observed that bubble and quick sort takes less Processor's time and consumed as little as 18 watts of Power in a single pass of sorting 500 stored random numbers in a text file.

This study is intended to determine the performance of dual Processors on various sorting algorithms by infusing some factors such as random number generation and power estimation module into existing algorithms which would help in measuring the power consumption on execution of each of these algorithms.

II AIM AND OBJECTIVES OF THE STUDY

The aim of this research work is to evaluate the Power consumption of sorting algorithms on Dual Processors in line with other specific objectives, which include:

(i) Determine which among (Radix Sort, Quick Sort, and Bubble Sort) sorting algorithm run faster.
(ii) Determine which among the Radix Sort, Quick Sort and Bubble Sort algorithm consume less or more Power on usage by the dual processors.
(iii) Develop software that would implement the proposed modified sorting algorithms.

III SCOPE OF THE STUDY

In this study the researcher had concentrate on sorting items in an array in memory using comparison sorting (because that's the only sorting method that can be easily implemented for any item type, as long as they can be compared with the less-than operator).

The following sorting algorithms had been evaluated with power consumption measurement algorithm infused into each. Radix Sort, Quick Sort, and Bubble Sort.

C++ Programming language was chosen for this research because it offers high-level programming structure with low-level features.

IV RELATED WORKS

A common misconception is that a radix sorting algorithm either has to inspect all the characters of the input or use an inordinate amount of extra time or space; however with a careful implementation efficient implementations are possible as shown by several researchers [3]

[2] investigated the performance of a number of string sorting algorithms. And they concluded that radix sorting algorithms are much faster than the most frequently used comparison-based algorithms. On the average Adaptive radix sort was the fastest sorting algorithm.

Reference [1] proposes a Modified Pure Radix Sort for Large Heterogeneous Data Set. In their research they discussed the problems of radix sort, and presented new modified pure radix sort algorithm for large heterogeneous data set. They optimize all related problems of radix sort through this algorithm.

Reference [11] illustrated the importance of reducing misses in the standard implementation of least-significant bit first in (LSB) radix sort, these techniques simultaneously reduce cache and TLB misses for LSB radix sort, all the techniques propose yield algorithms whose implementations of LSB Radix sort & comparison-based sorting algorithms.

[3] explained the Communication and Cache Conscious Radix sort Algorithm (C3-Radix sort). C3-Radix sort uses the distributed shared memory parallel programming Models.

[9] propose the high-performance parallel radix sort and merge sort routines for many-core GPUs, taking advantage of the full programmability offered by CUDA. Radix sort is the fastest GPU sort and merge sort is the fastest comparison-based sort reported in the literature. For optimal performance, the algorithm exploited the substantial fine grained parallelism and decomposes the computation into independent tasks.

[9] suggested an optimization for the parallel radix sort algorithm, reducing the time complexity of the algorithm and ensuring balanced load on all processor.

[8] in their work “parallel Quicksort algorithm Part 1 - Run time analysis” stated that Quicksort being such a popular sorting algorithm, there have been a lot of different attempts to create an efficient parallelization of it.

Another approach by [5] has been to Multiply each sequence to be sorted into blocks that can then be dynamically assigned to available processors . However, this method requires extensive use of atomic FAA2 which makes it too expensive to use on graphics processors.

According to [7] the Bubble Sort algorithm works by continually swapping adjacent array elements until the array is in sorted order. Every iteration through the array places at least one element at its correct position.

Although algorithmically correct, [8] observed that Bubble Sort is inefficient for use with arrays with a large number of array elements and has a $O(n^2)$ time complexity.

V METHODOLOGY

A. Method of data collection

For the purpose of this research work, all reference materials are collected from reputable Journals, textbooks and Internet. While the test data for the program would be randomly generated from a random program to be designed and incorporated into the final software.

B. Propose Radix Sort Algorithm

Algorithm radixSort(a, first, last, maxDigits)
// Generate Random Numbers
1: int min = 1;
2: int max = n;
3: int a = rand(min, max);
// Sorts the array of positive decimal integers a[first..last] into ascending order;
// maxDigits is the number of digits in the longest integer.
4: Timercount() = 0
5: for (i = 1 to maxDigits)
6: { Clear bucket[0], bucket[1], . . . , bucket[9]
7: for (index = first to last)
8: { digit = ith digit from the right of a[index]
9: Place a[index] at end of bucket[digit]
}
10: Place contents of bucket[0], bucket[1], . . . , bucket[9] into the array a
}
Timercount() = Timercount + 1
// Power Estimation
11: Begin
12: Compute the Time taking to Sort the Random numbers
13: Compute the Time (t) taking to Sort the Random numbers in an Array (CPU Time)
14: Multiply the Energy by the Time used in sorting Array of Data using Equation (2)
15. end

The complexity of the proposed selection sort algorithm can be determined as follows:

Let $c1$ to $c15$ represent the Cost of executing each instruction in the algorithm and n be the running time; then it can be derived that the total cost is:

$$T(n) = c1 + c2 + c3 + c4 + c5 \times n^2 + c6 + c7 \times n^2 + c8 + c9 + c10 + c12 + c13 + c14 + c15$$

Dropping all the Constant terms we have

$$T(n) = c5 \times n^2 + c7 \times n^2$$
$$= n^2(c5 + c7$$
$$T(n) = n^2$$

Therefore the Order of the Proposed Radix Sort algorithm is $O(n^2)$

C. Propose Quick Sort Algorithm

Algorithm quickSort(a, first, last)
// Sorts the array elements a[first] through a[last] recursively.
// Generate Random Numbers
1: int min = 1;
2: int max = n;
3: int a = rand(min, max);
4: Timercount = 0
5: if (first < last)
6: { Choose a pivot
7: Partition the array about the pivot

8: pivotIndex = index of pivot
9: quickSort(a, first, pivotIndex-1) // sort Smaller
10: quickSort(a, pivotIndex+1, last) // sort Larger
11: Timercount() = Timercount + 1
// Power Estimation
12: Begin
13: Compute the Time (t) taking to Sort the Random numbers in an Array (CPU Time)
14: Multiply the Energy by the Time used in sorting Array of Data using Equation (3)
15: Multiply the Energy by the Time used in sorting
16. end }

The complexity of the proposed selection sort algorithm can be determined as follows:

Let $c1$ to $c16$ represent the Cost of executing each instruction in the algorithm and n be the running time; then it can be derived that the total cost is:

$$T(n) = c1 + c2 + c3 + c4 + c5 \times n^2 + c6 + c7 + c\,c9 + c10 + c12 + c13 + c14 + c15 + c16$$

Dropping all the Constant terms we have

$$T(n) = c5 \times n^2$$

$$T(n) = n^2$$

Therefore the Order of the Proposed Quick Sort algorithm is $O(n^2)$

D. Propose Bubble Sort Algorithm

Bubble (Data, N)
// Here DATA is an array with N elements. This algorithm sorts the elements in //DATA.
// Generate Random Numbers
1: int min = 1;
2: int max = n;
3: int a = rand(min, max);
4: Timercount() = 0
5: Repeat Steps 6 and 7 for K = 1 to N-1
6: Set PTR := 1 [Initialize pass pointer PTR]
7: Repeat while PTR <= N-K : [Execute Pass]
(a) If DATA[PTR] > DATA[PTR+1], then
Interchange DATA[PTR] and DATA[PTR+1]
End if
(b) Set PTR := PTR+1 [End of inner loop]
[End of step 1 outer loop]
8: Timercount() = Timercout() + 1
9: Exit.
// Power Estimation
10: Begin
11: Compute the Time (t) taking to Sort the Random numbers in an Array (CPU Time)
12: Multiply the Energy by the Time used in sorting Array of Data using Equation (3)
13: Multiply the Energy by the Time used in sorting
14: end

The complexity of the proposed selection sort algorithm can be determined as follows:

Let $c1$ to $c14$ represent the Cost of executing each instruction in the algorithm and n be the running time; then it can be derived that the total cost is:

$$T(n) = c1 + c2 + c3 + c4 + c5 \times n^2 + c6 + c7 \times n^2 + c9 + c10 + c12 + c13 + c14$$

Dropping all the Constant terms we have

$$T(n) = c5 \times n^2 + c7 \times n^2$$

$$T(n) = n^2(c5+c7)$$

Therefore the Order of the proposed bubble Sort algorithm is $O(n^2)$

VI. RESULTS

The results obtained from the implementation of three selected algorithms (Bubble, Radix, and Quick) sorts using input array size of 250, 500, 750 and 1000 as extracted from Run program are shown in Table 1.1 and Table 1.2. The tables show the summary of the running time and Power Consumption respectively.

Table 1.1: Running Time Analysis of the Sorted data

Data element	Bubble Sort	Radix Sort	Quick Sort
250	0.099533	0.0730675	0.0626768
500	0.106327	0.0765998	0.0951078
750	0.140579	0.0681788	0.114954
1000	0.203767	0.0659055	0.156125

Table 1.2: Power Consumption Analysis of the Sorted data

Data element	Bubble Sort	Radix Sort	Quick Sort
250	0.0020736	0.00123843	0.00108063
500	0.00221515	0.0012983	0.00202357
750	0.00255598	0.00131113	0.00212877
1000	0.00377346	0.00122047	0.00283864

Table 1.1 and Table 1.2 can be graphically represented as shown in Figure 1.1 and Figure 1.2 below.

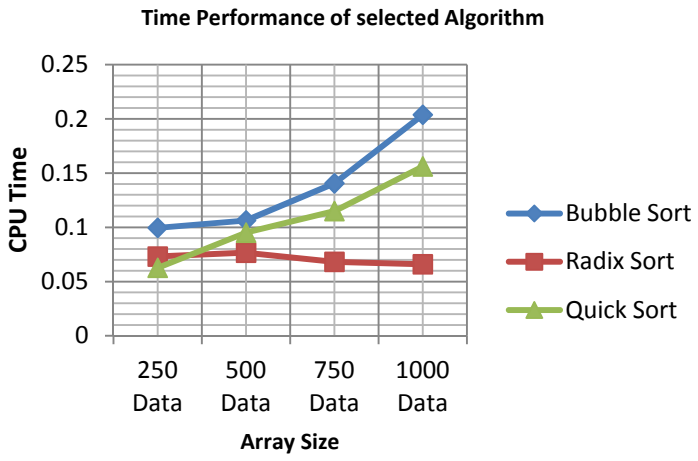

Figure 4.1: Graph showing CPU Time Measurement.

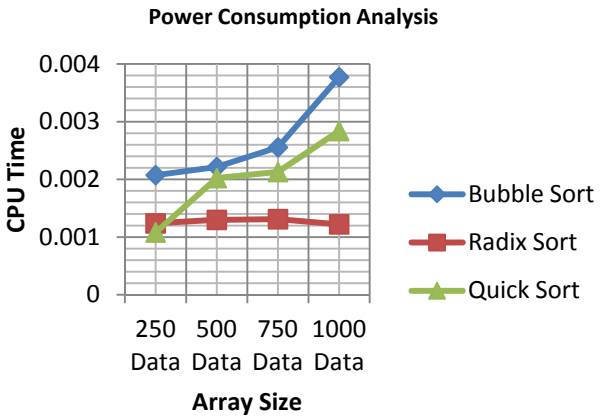

Figure 4.2: Graph showing Power Consumption

VII. DISCUSSION

Table 1.1 shows summary of the running time of the selected sorting algorithm of the study; it is obtained by running each of the selected algorithms for different input elements size. It can be observed from the table that when you increase the input size of the array, there is a significance increase in the running time thus; indicating that the size of an array affects the performance of bubble sort, Radix sort and quick sort algorithms; which are all of $O(n^2)$, where increasing the input size increase the running time. And this is in conformity with the assertion of [7] and other scholars who state that bubble sort is inefficient for use with large array, though the running time on dual processor is better compared to those run on single Processor.

It can then be deduced that increasing the size of the array by a factor (i.e. 250 elements) increases the running time and this is a true behavior of quadratic Sort of $O(n^2)$ therefore the study revealed that Bubble, Radix and Quick sort are quadratic in nature.

In terms of speed on a dual processor, Radix Sort tend to perform better than the remaining sorting Algorithm under study, from Table 1.1 it can be seen that for a large data array of 1000 elements the algorithms under study can be ranked in terms of speed as follows (Radix, Quick, and Bubble Sort) with the Radix taking the lead and this is in line with the work of [9] who's result shows that Radix sort and merge sort routine perform better on many-core CPU.

A table 1.2 show that the time complexity of sorting algorithms may affect the Power usage greatly since the time is proportional to energy consumption. The result shows that algorithms with time complexity O (n^2) may consume a lot more power than algorithms with time complexity O (n log n), as shown in same figure 1.2

The Result from Table 1.2 shows that for 500 data array, a bubble sort would consume 0.140579 Watts, Contrary to [10] whose study shows that for 500 data array, 18 watts of power was consumed on single core processor, amounting to 99.2% improvement when compared to what was obtained in this study. Similarly comparing same with result obtained by [11] whose study shows a power consumption of 0.2521 watts on running a bubble sort algorithm with 500 array size of random number, this shows an improvement of 44.2% when compared with what is obtained in this study when run on dual processor. Similarly the Algorithms under study can be ranked according to the power consumption if we take results for 1000 array elements from low to higher power consumption from Table 1.2 as follows Radix, Quick Sort, and Bubble Sort, indicating that Radix sort performs better in terms of speed and power consumption.

VIII. CONCLUSION

This study discusses comparison based sorting algorithms. It analyses the performance of these algorithms for different number of elements .It then concludes that Radix sort shows better performance than the rest while bubble sort efficiency drastically reduces with an increase in data size suggesting that bubble sort is not suitable for sorting large data array. It is clear that all the sorting techniques are not that popular for the large arrays because as the arrays size increases both timing is slower. But generally the study indicate that Radix Sort have faster CPU time as well as less Power Consumption on Dual Processor even though it have the upper bound running time $O(nlogn)$.

RECOMMENDATIONS

From the result obtained from this study it is recommended that for any programmer engaging in developing a commercial or personal software special consideration should be given to the type of algorithm to be used after all other factors were considered Since time is directly proportional to the Power consumption, this research reveal that some algorithm run faster than others and this means those that have low CPU time tend to have less power consumption. Among the three (3) algorithms studied, Radix Sort tend to have favorable running time of order $O(nlogn)$ and optimal power consumption of less than a watt Power drain on dual processor. Based on these the following recommendations were made:

1. The Study should serve as a reference manual for program developers interested in Power management of Software.
2. For researchers interested in Algorithms efficiency measurement.

REFERENCE

[1] Avinash, S. & Anil, K. S.. Modified Pure Radix Sort for Large Heterogeneous Data Set *IOSR Journal of Computer Engineering (IOSRJCE)*, 2012.

[2] Arne, A. A. & Stefan, N. S.. Implementing Radix Sort. *ACM Journal of Experimental Algorithmics*, 1998.

[3] Daniel, N. & Joseph, H.. CC-Radix: a Cache Conscious Sorting Based on Radix Sort,11th IEEE Conference on Parallel, Distributed & Network-BasedProcessing(Euro-PDP'03) ,2003.

[4] David, M. W. Powers, *Parallel Unification: Practical Complexity*, Australasian Computer Architecture Workshop, Flinders University, 1992.

[5] David, M. W. *Powers Parallelized Quicksort and Radixsort with OptimalSpeed* Proceedings of International Conference on Parallel Computing Technologies, 2006.

[6] Gret, Z. & Evans, D. J. (2006). Adaptive Bitonic Sorting. An Optimal Parallel Algorithm for Shared-Memory Machines. *SIAM Journal Computing*, 182:216.

[7] Knuth, D.E. *The Art of programming- Sorting and Searching*. 2nd Edn., Addison Wesley Press, London 780-781, . 1988.

[8] Macllory, H., Norton, A., & John, T. R. Parallel Quicksort Using Fetch-And Add.IEEE Trans. Comput., 133-138, 1993.
[9] Nadathur, S.. Designing Efficient Sorting Algorithm for manycoregpus",23rd *IEEE International Parallel and Distributed Processing Symposium*, 2009.
[10] Oyelami, O. M.). Improving the Performance of Bubble Sort using a Modified Diminishing Increment Sorting. Scientific Research and Essay 1992-2248 *Academic Journals*, 2009.
[11] Pallav, C.. Performance Comparison of Distributive and Merge sort as External Sorting Algorithms, *Journal of Systems Software*, 2007.
[12] Rajeev, R. & Naila, R.. Adapting Radix Sort to the Memory Hierarchy , *Journal of Experimental Algorithmic* (JEA), pp.7-9, 2011.
[13] Tiwari, V., Malik S. & Wolfe A. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems,pp:437-445, 1994.