ITC 3/48
Journal of Information Technology and Control
Vol. 48 / No. 3 / 2019
pp. 487-498
DOI 10.5755/j01.itc.48.3.23696

The Analysis of Energy Performance in Use Parallel Merge Sort Algorithms

Received 2019/06/26 | Accepted after revision 2019/08/30

http://dx.doi.org/10.5755/j01.itc.48.3.23696

The Analysis of Energy Performance in Use Parallel Merge Sort Algorithms

Zbigniew Marszałek
Institute of Mathematics, Silesian University of Technology, ul. Kaszubska 23, 44-100 Gliwice, Poland

Corresponding author: zbigniew.marszalek@polsl.pl

The issue of productivity and energy is an important objective of the optimization of parallel applications. The size of the problem for a large number of data on multiprocessor platforms forces the use of parallel algorithms. Efficient management of large memories using modern processors in Big data processing requires innovative techniques and efficient algorithms. For years have found the results of tests conducted on methods for use in various computing environments and improvements. This article shows the energy consumption analysis by parallel sorting algorithms. Sort algorithms are used in information systems and databases, to select and organize the information. The subject of this article is research into energy consumption and computational complexity for parallel sorting methods by merging compared to classic methods. The tests carried out confirm the reduction of energy consumption by using parallel sorting algorithms. The presented parallel fast sort and parallel modified merge sort for large task dimensions have less power consumption than classic methods and can be used successfully in NoSQL databases.

KEYWORDS: power-aware testing, functional power component, parallel algorithm, data sorting, data mining, analysis of computer algorithms.

1. Introduction

Analysis of energy consumption by algorithms is an important element taken into account in the design of computer systems. Power analysis can be performed by using the measurement methods or based on simulation. In the case of computer algorithms it is more convenient to perform a simulation of energy consumption. Simulators processor down-level and estimated average power consumption were presented in [4, 5, 10].

Information technology is growing every year. Data processing systems in the NoSQL databases require efficient sorting algorithms with low energy con-

sumption. Basic sorting algorithms, such as quick sort, sorting through packing and sorting by merging have been described in many works, among other things in [3, 13, 19]. Parallelized algorithm for fast sort [16] and parallelized merge sort algorithm [17] play a special role in increasing the efficiency of the ordering data sets, and reducing the energy consumed by the processors. Most of the work on energy efficiency [1, 9, 12, 18] assumes that energy consumption is independent of the size of the problem. In [15], an analysis of the memory impact on the energy efficiency of parallel computation is presented. This article attempts to determine the power consumption of parallel sorting methods by merging. The analysis carried out shows the cost-effectiveness of using parallel sorting methods for large task dimensions.

To compare the energy efficiency of the parallel sorting methods, a classic fast sorting algorithm, a trigeminal heap algorithm, and a non-recursive merge sorting algorithm were taken. The quick sort algorithm divides the sorted string into two substrings and moves the elements of the string so that the first substring is less than or equal to the middle element and the elements in the second substring are larger than the midpoint. The recursive split process is executed for the ordering of the string. The introduction of a median value for divisions exchange was presented in [21]. This method was tested on various architectures in [22]. Heap sort uses the three level structure of data storage, where introduced relations between the following levels influence the speed of sorting. Each change in the structure requires the procedure to insert elements into the heap. Mathematical models of the relations between levels of the heap were discussed in [2, 8]. The non-recursive version of the sort algorithm iterates through the string elements, enlarges the size of the merged string in each iteration twice. Sublinear merging was presented in [3]. Parallel computing issues and their use are described in [20]. The Parallel Fast Sort Algorithms and Parallel Modified Merge Sort Algorithm [16, 17] methods are shown to reduce the energy consumption of large data sets.

2. Materials and Methods

Work on reducing the power consumption of microprocessors has been carried out for many years [4, 5, 10]. Gupta [10] shows a method of measuring the power consumption by means of voltage measurement during instructions by the microprocessor and defines the energy usage in the clock cycle by $\frac{1}{2}\max(v_l)^2 - \frac{1}{2}\min(v_l)^2$. In this method, the voltage is measured on the resistors in the power circuit. Similarly, Chen [4] determines the maximum energy consumption of a processor in a measurement cycle. To improve performance and to reduce power consumption in cloud computing infrastructures at work, [18] apples the parallel bi-objective genetic algorithm. The computation cost of a task and the parameter ω decision-making that specifies the processor power consumption is a function of the $V^2 \cdot \omega$. The optimization of scientific workflows in heterogeneous computing environments and energy consumption is subject to study at work [9]. The energy consumption model used the clock frequency of the main module. Heuristic algorithm [1] in the calculation of energy efficiency in the data center resources reallocated to Quality of Service (QoS) takes into account the number of virtual machines and clock frequency. This work in the energy consumption model of data collations using the described algorithms assumes the frequency of the main processor and the number of cycles needed to complete the task. Research on the relationship between memory accesses, bank conflicts, thread multiplicity, and instruction-level parallelism in comparison-based sorting algorithms for Graphics Processing Units (GPUs) are described in [11]. In [6], the implementation of image transformation algorithm on mobile devices using the approximate computation methods is described. The proposed algorithm allows to save 6% of energy when transforming the greyscale images using the *Twirl* effect. The efficiency of energy consumption function hash is the subject of tests [7]. Research shows that the most efficient algorithm for cryptographic applications is SVI and crc16 for non-cryptographic applications. The proposed solutions allow to save up to 29% of energy on hashing operations.

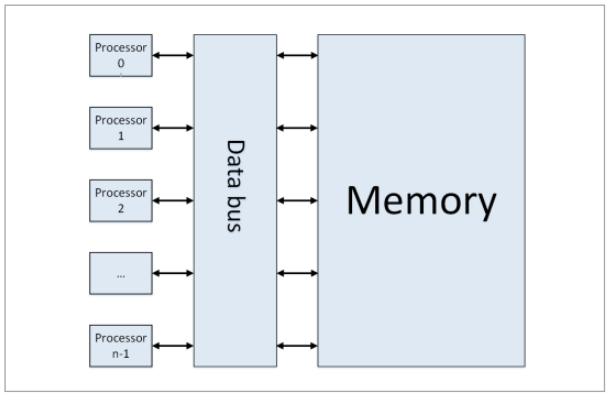
Commonly used method in information systems to organize data sets is called quick sort. However, this method is sensitive to the critical settings of the sorted string and is not easily performed on many processors available on modern computers. Alternatively, for sorting large data sets the sort method is used by the merge described in [3, 19]. This paper will compare the power consumption of the processor to the

classic sorting methods: quick sort, trigeminal heap sort and merge sort with parallel methods: fast sort [16] and modified merge sort [17].

2.1. Parallel Random Access Machine

For the analysis of parallel sorting algorithm it is convenient to use the parallel machine model the PRAM (parallel random access machine) as shown in Figure 1.

Figure 1
A sample schema of the Parallel Random Access Machine



Depending on the processor's access method, four types of PRAM machines are specified in memory:

1. Exclusive read exclusive write (EREW)
2. Concurrent read exclusive write (CREW)
3. Exclusive read concurrent write (ERCW)
4. Concurrent read concurrent write (CRCW)

The first type of machine the PRAM allows to read write memory only one processor. The second type provides reading memory through any processor, but wringing at the same time can run only one processor. The third type allows to read every memory cell by only one processor and multiple processors can read a memory cell at the same time. The third type of PRAM machine is not intended to be practical and is not considered in theory.The fourth type allows to access memory using any processor. The second of the presented models reflects the architecture of the modern computer and is practically possible to implement.

2.2. Statistical Research on Algorithm Performance

Let $a_1, a_2, ..., a_n$ denote disordered number of distributed results of methods should be described with sufficient details to allow others to observe some features of A, and let $\bar{a}$ be arithmetic mean of these results

$$\bar{a} = \frac{a_1 + a_2 + \cdots + a_n}{n}. \quad (1)$$

Standard deviation describes the formula

$$\sigma = \sqrt{\frac{(a_1 - \bar{a})^2 + (a_2 - \bar{a})^2 + \cdots + (a_n - \bar{a})^2}{n-1}}, \quad (2)$$

where n is the number of elements in the sample, $a_1, a_2, ..., a_n$ are values of the random variable in the sample and $\bar{a}$ is the arithmetic mean of the sample. In order to find the most efficient algorithm is an analysis of average time for sufficiently large set of data. The analysis for sorting time was carried out in 100 benchmark tests for each of the fixed dimension of the task on the input. The stability of the algorithm is best described on the basis of the coefficient of variation. The coefficient of variation is a measure that allows to determine value of diversity in the examined population. It is determined by the formula

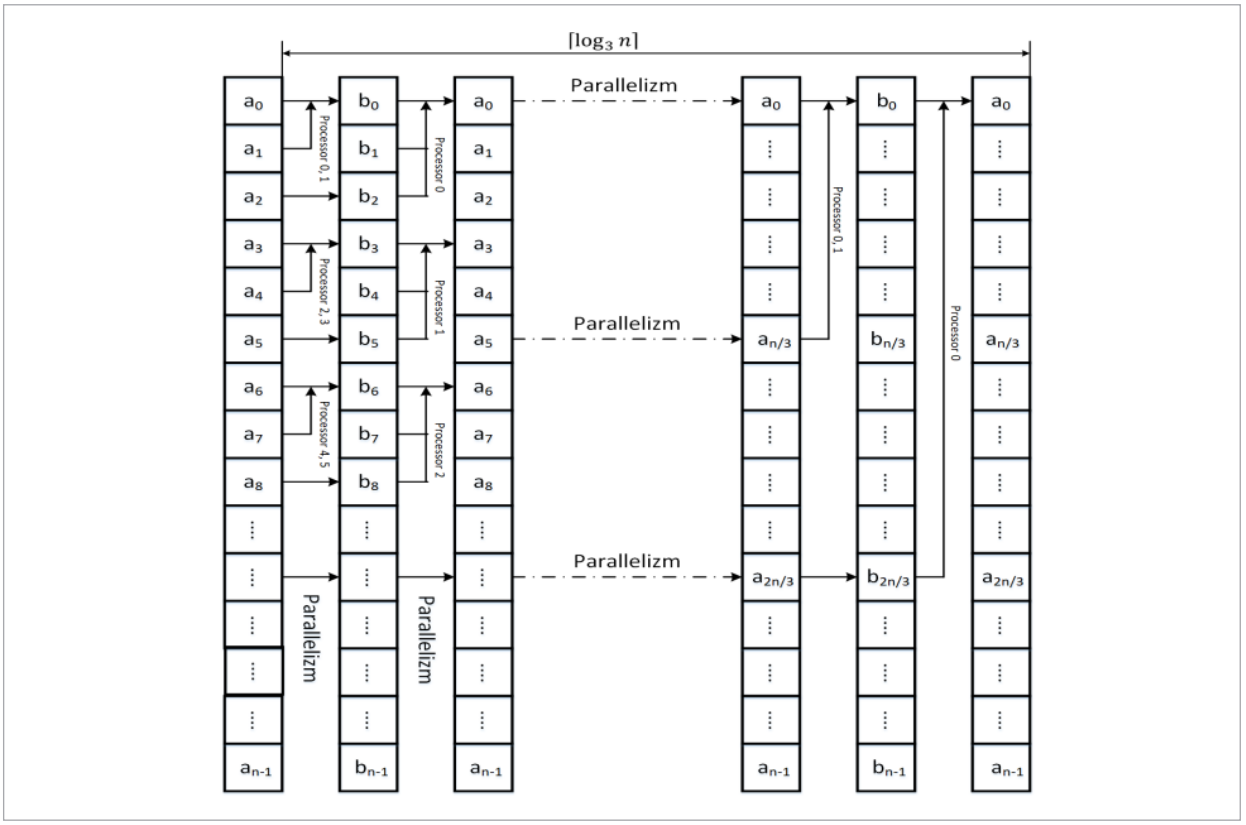
$$V = \frac{\sigma}{\bar{a}}, \quad (3)$$

where $\bar{a}$ is the arithmetic mean (1) and σ is the standard deviation (2). The coefficient of variation reflects the stability of the method in a statistical sense. Benchmark tests of the newly proposed method for sorting sets were taken for 100, 1.000, 10.000, 100.000, 1.000.000, 10.000.000 and 100.000.000 elements on the input. The results are presented in graphs and discussed in the following sections.

3. Parallel Fast Sort Algorithm

Sorting large data sets requires high-speed parallel methods of low complexity. In [16], the parallel sorting method that allows the use of high-capacity modern processors is described. Consider parallelization of the fast sort algorithm. Each string sort step merges all three ordered numeric strings into one sorted numeral string in Figure 2. In the first stage, the first two pairs of strings are merged into the temporary array, and then the string stored in the temporary array with the third string written to the input array is merged. In

Figure 2
Parallel fast sort algorithm



the first step, $n/3$ independently working processors merge each pair of strings, where n is the dimension of the sort tasks. In the next steps, the number of independently working processors decreases three times, and only one processor is involved in the last step.

Theorem 1. Parallel Fast Sort Algorithm using $n/3$ processors has time complexity

$$T_{max} = \frac{5}{2}n - 2\log_3 n - \frac{5}{2} \tag{4}$$

PROOF

The proof is conducted for the $n = 3^k$, where $k = 1, 2, \ldots$.

Due to the fact that three ordered t-elements can merge into one sorted string by doing no more than $5t - 2$ comparisons, then for $t = 1$ in the first step each $n/3$ processors will perform no more than $5 \cdot 1 - 2 = 3$ comparisons. All string merge time is a single processor operation time. In each step t, the algorithm using $n/3^t$ processors will merge the three 3^{t-1} element strings by doing no more than than $5 \cdot 3^{t-1}-2$ comparisons. The sum of all comparisons performed by the algorithm is

$$\sum_{t=1}^{k}(5 \cdot 3^{t-1} - 2) = \tag{5}$$

$$(5-2) + \cdots + (5 \cdot 3^{k-1} - 2) = \tag{6}$$

$$= (5 \cdot 1 + 5 \cdot 3 + \cdots + 5 \cdot 3^{k-1}) - \tag{7}$$

$$-2 \cdot k = \tag{8}$$

$$= \frac{5 \cdot 3^k - 5}{2} - 2\log_3 n = \tag{9}$$

$$= \frac{5}{2}n - 2\log_3 n - \frac{5}{2}. \tag{10}$$

Figure 3
Pseudocode parallel fast sort algorithm

```
Start
Load table a
Load dimension of table a into n
Create an array of b of dimension n
Set options for parallelism to use all
processors of the system
Set the size of the merged string to one
and remember in tt
While tt is less than n then do
Begin
  Calculate the number of three consecutive
  merged strings and remember in it
  Parallel for each processor j at index
  greater or equal 0 and less than it do
  Begin parallel for
     Merge three consecutive strings
     allocated for the processor J
     using an array of b
  End of the parallel for
  Multiply variable tt by three
End
Stop
```

Linear time complexity of the algorithm is the result of separate work processors in each iteration of the merge strings. The number of possible to use processors merging strings reduces in each iteration and only one processor can be used in the last step. To increase the number of processors participating in the calculation would use a parallel merge algorithm, which is the subject of research. The algorithm presented in Figure 3 uses the maximum number of processors available on the system.

The presented method was implemented in C# Visual Studio Ultimate 2013. The algorithm uses a parallel loop, which reduces the created program code, because there is no need to create separate tasks, run them, and wait for them to finish. The study was conducted on 100 input samples randomly generated for each dimension of the task. Tests were carried out on Intel i7-7700HQ as describes in Table 1.

The purpose of analysis and comparison is to reveal how the parallel calculations affect the CPU power consumption of sorting large data sets. For the benchmark, input samples of 100, 1.000, 10.000, 100.000, 1.000.000, 10.000.000 and 100.000.000 elements were applied.

Two types of energy consumption are distinguished, namely, static and dynamic energy consumption.

Table 1
The specification of the Intel Core i7-7700HQ Processor

Technical Specifications	Intel® Core™ i7-7700HQ Processor
Product Collection	7th Generation Intel® Core™ i7 Processors
Code Name	Products formerly Kaby Lake
Processor Number	7-7700HQ
Lithography	14 nm
Processor Base Frequency	2.80 GHz
Max Turbo Frequency	3.80 GHz
Cache	6 MB
TDP	45 W
Configurable TDP-down	35 W
Number of cores	4
Number of threads	8

Static power consumption is the result of Microsoft optimizing the application. Dynamic energy consumption is calculated by subtracting from the total energy consumption of the static energy processor. It is noted that the total power consumption of the processor reflects the energy consumption of the constant and can be taken to compare algorithms.

The sorting algorithms have a time complexity of $O(n)$ or $O(n\log_2 n)$, where $\log_2 n$ is the number of algorithm runs with linear time complexity. For a linear algorithm, the power consumption of the processor is constant and the number of CPU cycles needed to perform Algorithm determines energy consumption.

Each sorting operation by examined methods was measured in time [ms]. The average power dissipated in the high complexity workload of all cores (Thermal Design Power) for the i7-7700HQ processor is 45W. The Configurable TDP-down power dissipation for processor i7-7700HQ defined by Intel is 35 W. The use of Configurable TDP-down is typically executed by the system Microsoft to optimize power and performance, and TPD-down has been accepted for calculation.

These results are averaged for 100 sorting samples. The benchmark comparison for parallel fast sort algorithm is described in Table 2, and Figures 4 and 6. The analysis of the results confirms the hypothesis that adding each successive processor reduces energy consumption. More visible changes are seen between 2 and 4 processors usage. With each processor added to the system, the power consumption decreases. This confirms the theoretical time complexity of the algorithm in Theorem 1.

An analysis of the variation coefficients shows the stability of the parallel method of fast sort for large

Table 2
Energy consumption parallel fast sort algorithm in [Ws]

Energy consumption – average time sorting for 100 samples in [Ws]				
Elements	1 – processor	2 - processors	4 – processors	8 – processors
100	0.259693	0.249606	0.245371	0.230685
1 000	0.266371	0.252864	0.252819	0.239208
10 000	0.335657	0.302518	0.288938	0.267067
100 000	1.096057	0.824436	0.771877	0.706828
1 000 000	9.436844	5.703065	4.931087	4.350560
10 000 000	101.9322	57.91858	49.60860	39.98709
100 000 000	1151.145	617.5842	497.3152	391.3391

data sets. Some variations in stability of the algorithm for small inputs are due to the fact that the system exceed sorting algorithm.

4. Parallel Modified Merge Sort Algorithm

A parallel modified sorting algorithm by merging four strings is presented in [17]. To merge four strings, logical indexing of the processors participating in the sort has been applied. In each step, sorting methods that independently operate the processors merge all ordered pairs of strings saved from the input array and write the result into a temporary array. It then

Figure 4
Comparison of energy consumption for parallel fast sort algorithms [Ws]

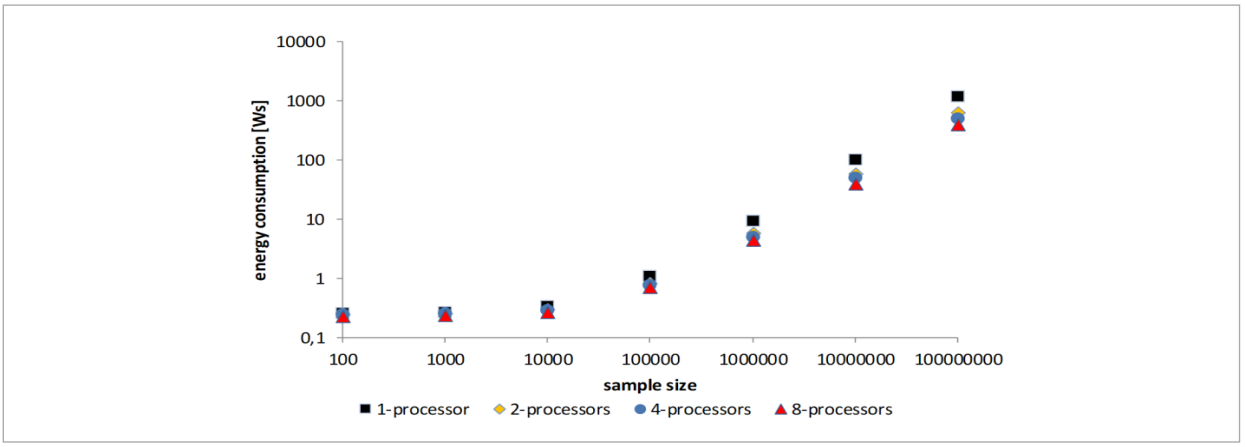


Figure 5
A comparison of the efficiency of the method in power consumption, using multiple

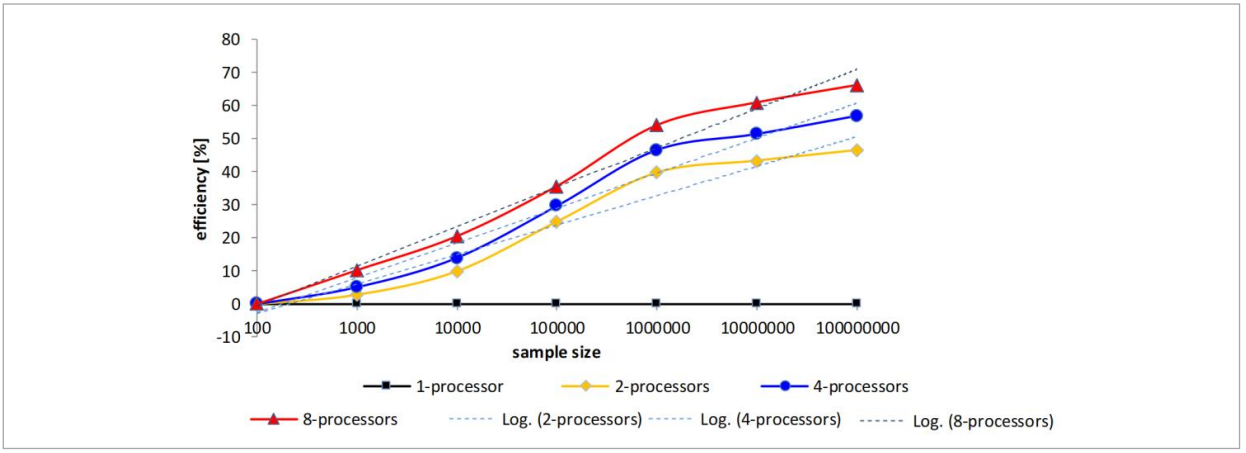
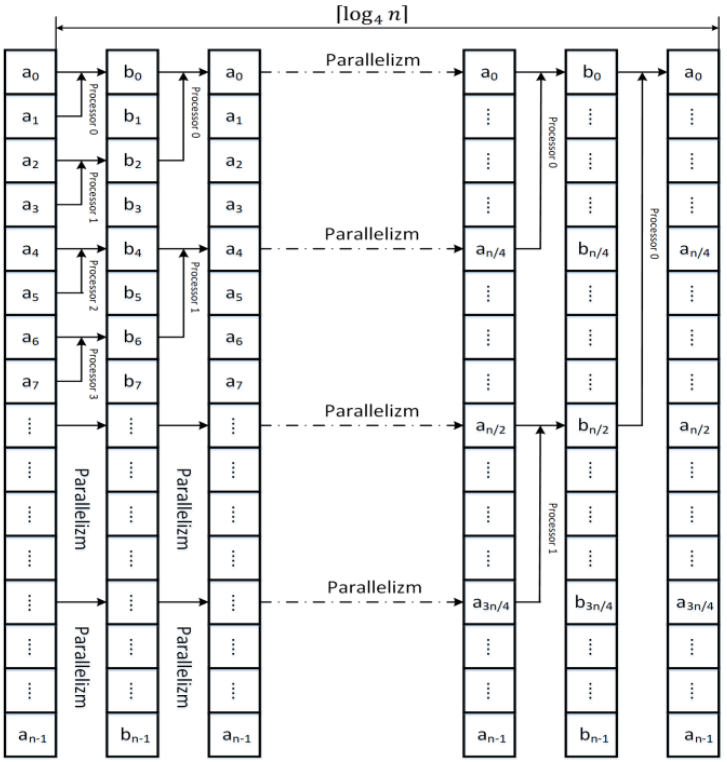


Figure 6
Parallel modified merge sort algorithms



merged all ordered pairs from the temporary array into the input array. The index of merged string shall be the same as the index of the first element of the first string. The entire proses merge all pairs of numeric strings by the parallel loop by independently working processors is the end of the merge cycle both in a temporary array and in the input array. The duration of the parallel loop can be defined as the time the longest working processor which is involved in the merging of strings. The way that strings are merged by independent processors is shown in Figure 6.

Theorem 2. Parallel Modified Merge Sort Algorithm using $n/2$ processors has time complexity:

$$T_{max} = 2n - \log_2 n - 2\,. \tag{11}$$

PROOF

The proof is conducted for the $n = 4^k$, where $k = 1, 2, \ldots$.

Due to the fact that two ordered t - elements can merge into one sorted string by doing no more than $2t - 1$ comparisons, then for $t = 1$ in the first step each $n/2$ processors will perform no more than $2 \cdot 1 - 1 = 1$ comparisons. All string merge time is a single processor operation time. In each step of t, the algorithm first merges the $n/2^{2t-1}$ -element strings into a temporary array, and then merge the $n/2^{2t}$ element strings from the temporary array into the input array by doing no more than $2 \cdot 2^{2t-2} + 2 \cdot 2^{2t-1} - 2$ comparisons. The sum of all comparisons performed by the algorithm is

$$\sum_{t=1}^{k} [(2 \cdot 2^{2t-2} - 1) + \\ + (2 \cdot 2^{2t-1} - 1)] = \tag{12}$$

$$= 2 \sum_{t=1}^{k} (2^{2k-2} + 2^{2k-1}) - 2k = \tag{13}$$

$$= 2\sum_{t=1}^{k} 2^{2k-2} + 2\sum_{t=1}^{k} 2^{2k-1} - 2k = \quad (14)$$

$$= 2(1 + 2^1 + 2^2 + \cdots + 2^{2k-1}) - 2k = \quad (15)$$

$$= 2(2^{2k} - 1) - 2k = \quad (16)$$

given that

$$2k = \log_2 n \quad (17)$$

and

$$2^{2k} = n \quad (18)$$

As a result of substitution

$$= 2n - \log_2 n - 2\,. \quad (19)$$

The presented method was implemented in C# Visual Studio Ultimate 2013. Statistical surveys for samples range from 100 to 100 million elements, increasing the dimension 10 times for each subsequent. Each sorting operation by examined methods was measured in time [ms]. The average power dissipated in the high complexity workload of all cores (Thermal Design Power) for the i7-7700HQ and TPD-down has been accepted for calculation energy consumption. The algorithm presented in Figure 7 uses the maximum number of processors available on the system.

Statistical surveys are averaged for 100 sorting samples. The benchmark comparison for parallel modified merge sort algorithm is described in Table 3, and Figures 8-9. As with the parallel fast sort algorithm, the analysis of the parallel results of the modified sorting algorithm by merging shows that each processor added reduces power consumption. By adding a larger number of processors, the changes are clearly visible. With each processor added, the efficiency of the algorithm increases, which confirms the theoretical complexity of the algorithm contained in Theorem 2.

An analysis of the variation coefficients shows the stability of parallel modified merge sort, which improves sorting for large data sets.

Figure 7

Pseudocode parallel modified merge sort algorithms

```
Start
Load table a
Load dimension of table a into n
Create an array of b of dimension n
Set options for parallelism to use all
processors of the system
Set the size of the merged string to one
and remember in tt
While tt is less than n then do
Begin
   Calculate the number of consecutive pairs of
   strings to merge in array a and remember in it1
   Parallel for each processor jj at index
   greater or equal 0 and less than it1 do
   Begin parallel for
      Merge pairs of strings from array a
      allocated for the processor jj
      by saving the merged strings in array b
   End of the parallel for
   Multiply variable tt by two
   If tt greater than n then do
        Remember n in tt
   Calculate the number of consecutive pairs of
   strings to merge in array b and remember in it1
   Parallel for each processor jj at index
   greater or equal 0 and less than it1 do
   Begin parallel for
       Merge pairs of strings from array b
       allocated for the processor jj
       by saving the merged strings in array a
   End of the parallel for
   Multiply variable tt by two
End
Stop
```

NOTE: If there is only one string to merge it, it is rewritten to the second array.

Table 3

Energy consumption parallel modified merge sort algorithm in [Ws]

Energy consumption – average time sorting for 100 samples in [Ws]				
Elements	1 – processor	2 – processors	4 – processors	8 – processors
100	0.3119445	0.302785	0.2850365	0.287763
1 000	0.318682	0.3114545	0.303832	0.296408
10 000	0.3927525	0.35687	0.333672	0.311886
100 000	1.1110715	0.8203335	0.718957	0.6298375
1 000 000	8.977948	5.1638685	4.000066	3.4158845
10 000 000	96.4112905	52.5186145	39.1911415	32.556293
100 000 000	1074.068723	568.6885755	426.01256	351.97273

Figure 8
A comparison of energy consumption for parallel modified merge sort [Ws]

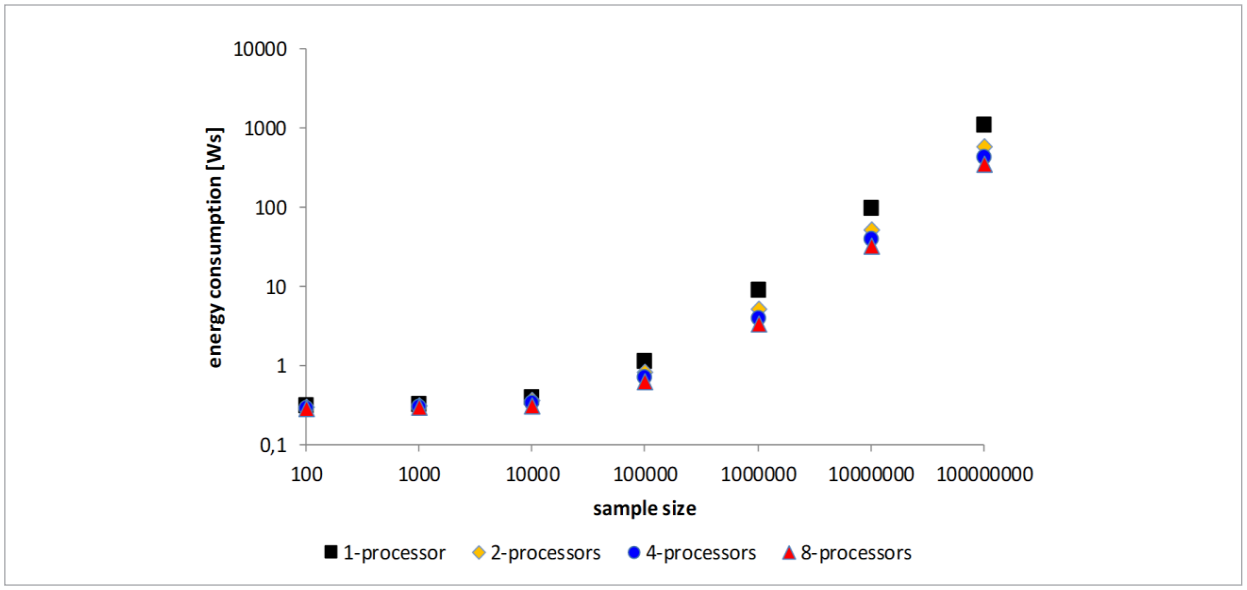
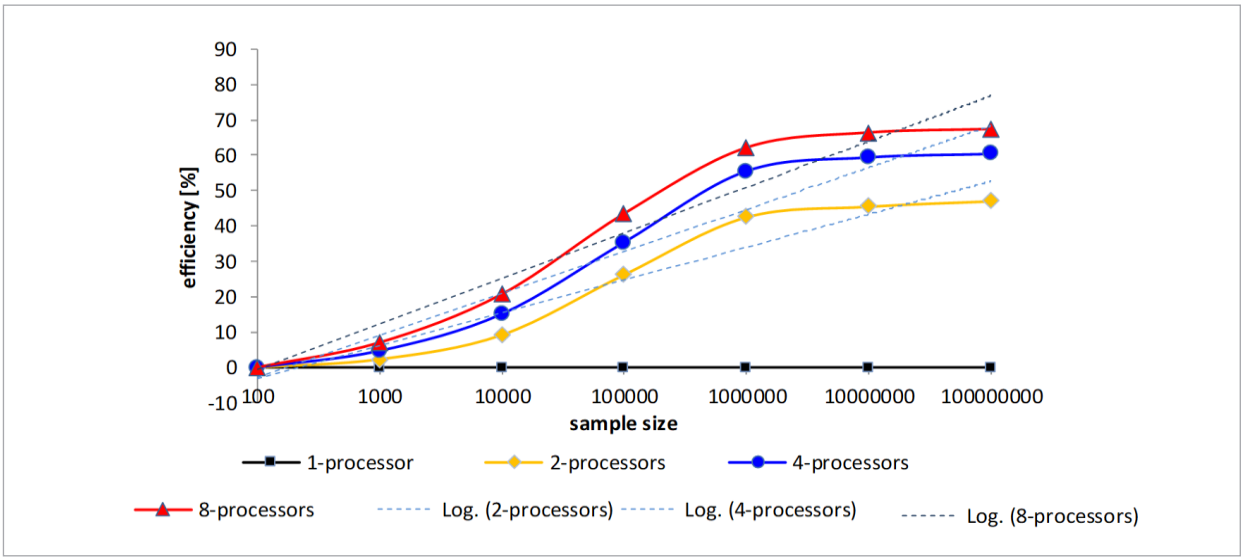


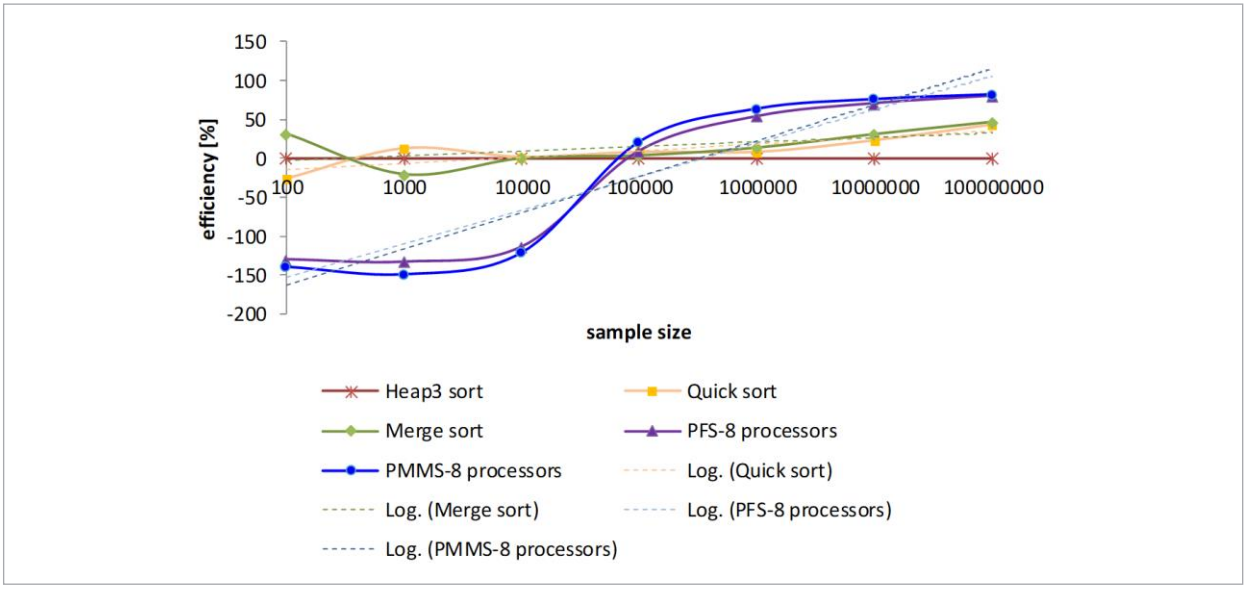
Figure 9
A comparison of the efficiency of the method in power consumption, using multiple processors in [%]



5. Analysis and Compression of Energy Consumption

An analysis of energy consumption is an important element in determining the efficiency of algorithms for sorting large data sets in the NoSQL databases. The reduction of power consumption when increasing the number of processors is visible both for the parallel fast sort parallel and the parallel modified merge sort algorithm, which is exposed Efficiency in Figures 5 and 9. Comparison of energy consumption

Figure 10
A comparison of sorting time for trigeminal heap sort, quick sort, merge sort on 1 processor with parallel fast sort and parallel modified merge sort on 8 processors



by parallel fast sorting and parallel modified merge sorting algorithms running on 8 processors with algorithms running on a single processor such as quicksort, merge sort, trigeminal heap sort is preferable for Parallel algorithms and is shown in Figure 10. The comparison based on trigeminal heap sort algorithm and received a percentage difference in energy consumption [Ws] compared to other algorithms. Efficiency means reducing the percentage of energy consumption in Figure 10.

The static analysis carried out in Figure 10 shows that the parallel methods running on multiple processors are more efficient than classical sorting methods and have less power consumption in sorting large data sets. The results obtained show that when sorting small data sets, it is better to use classical sorting methods that run on a single processor than the parallel sorting methods. This is because parallel methods that run on multiple processors require that the program is loaded into memory. They also require memory reservations for the processors to be used, and that processor activity synchronization actions are performed to prevent deadlock. The advantage of parallel methods is only visible for the day dimension greater than 100000. Time complexity of the parallel merge sorting methods is $O(n)$ and cannot be reduced by simply adding processors to the sorting algorithm. Reducing the time complexity of parallel sorting algorithms by merging is possible by applying a parallel method to merge numeric strings. However, it should be expected that the use of more processors in the string merging process will degrade energy consumption and such algorithms would be less efficient in terms of energy consumption [16, 17].

6. Final Remarks

The article presents the analysis of power consumption by a parallel fast sort algorithm and parallel modified merge sorting algorithm for large data sets in the NoSQL databases. The proposed method is based on a model of PRAM (Parallel Random Access Machine) that allows the efficient access to read and write information in the memory cell for each single processor. In the article, a theoretical analysis of the efficiency and practical verification of power consumption algorithms was presented. The practical realization of the algorithms was done in C# MS Visual 2015 on Intel i7 7700hq.

The tests demonstrate stability of the methods and confirm theoretical time complexity. Comparison tests have shown that a parallel fast sort and a parallel modified merge sort are more efficient than other sorting methods, especially for big data sets. The presented methods for large task dimensions have less power consumption than classic methods and can be used successfully in NoSQL databases.

In the future research is planned a further development in sorting performance. The research will involve developing the power consumption parallel merge string algorithm.

Acknowledgement

The author would like to acknowledge the contribution to this project from the Rector of the Silesian University of Technology under the grant for perspective professors no. 09/RGH18/0035 and pro-quality grant no. 09/010/RGJ19/0042

References

1. Beloglazov, A., Abawajy, J., Buyya, R. Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. Future Generation Computer Systems, 28(5), 2012, 755-768. https://doi.org/10.1016/j.future.2011.04.017
2. Ben-Or, M. Lower Bounds for Algebraic Computation Trees. In Proceedings of 15th ACM Symposium. Theory of Computing; ACM Press: New York, NY, USA, 1983, 80-86. https://doi.org/10.1145/800061.808735
3. Carlsson, S., Levcopoulos, C., Petersson, O. Sublinear Merging and Natural Merge Sort. International Symposium on Algorithms, Springer, Berlin, Heidelberg, 1990, 251-260. https://doi.org/10.1007/3-540-52921-7_74
4. Chen, R., Y., Owens, R., M., Irwin, M., J., Bajway, R., D., S. Validation of an Architectural Level Power Analysis Technique. Proceedings of the 35th Annual Design Automation Conference, ACM, 1998, 242-245. https://doi.org/10.1145/277044.277106
5. Chang, N., Kim, K., Lee, H., G. Cycle-Accurate Energy Measurement and Characterization with a Case Study of the ARM7TDMI [Microprocessors]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2002, 10(2), 146-154. https://doi.org/10.1109/92.994992
6. Damasevicius, R., Ziberkas, G. Energy Consumption and Quality of Approximate Mage Transformation. Elektronika ir Elektrotechnika, 2012, 120, 4, 79-82. https://doi.org/10.5755/j01.eee.120.4.1459
7. Damasevicius, R., Ziberkas, G., Stuikys, V., Toldinas, J. Energy Consumption of Hash Functions. Elektronika ir Elektrotechnika, 18, 10, 81-84. https://doi.org/10.5755/j01.eee.18.10.3069
8. Doberkat, E.-E. Inserting a New Element Into a Heap. Numerical Mathematics, 1981, 21(3), 255-269. https://doi.org/10.1007/BF01941462
9. Fard, H., M., Prodan, R., Barrionuevo, J., J., D., Fahringer, T. A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2012, 300-309. https://doi.org/10.1109/CCGrid.2012.114
10. Gupta, S., Najm, F., N. Energy-Per-Cycle Estimation at RTL. IEEE Proceedings, 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477), 1999, 121-126. https://doi.org/10.1145/313817.313894
11. Karsin, B., Weichert, V., Casanova, H., Iacono, J., Sitchinava, N. Analysis-Driven Engineering of Comparison-Based Sorting Algorithms on GPUs. Proceedings of the 2018 International Conference on Supercomputing, ACM, 2018, 86-95. https://doi.org/10.1145/3205289.3205298
12. Kessaci, Y., Melab, N., Talbi, E., G., A. Pareto-Based Metaheuristic for Scheduling HPC Applications on a Geographically Distributed Cloud Federation. Cluster Computing, 2013, 16(3), 451-468. https://doi.org/10.1007/s10586-012-0210-2
13. Kushagra, S., López-Ortiz, A., Munro, J., Qiao, A. Multi-Pivot Quicksort: Theory and Experiments. Proceedings of the Meeting on Algorithm Engineering & Experiments, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2016, 47-60. https://doi.org/10.1137/1.9781611973198.6
14. Manumachu, R. R., Lastovetsky, A. Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy. IEEE Transactions on Computers, 2017, 67(2), 160-177. https://doi.org/10.1109/TC.2017.2742513
15. Marszalkowski, J., M., Drozdowski, M., Marszalkowski, J. Time and Energy Performance of Parallel Systems

with Hierarchical Memory. Journal of Grid Computing, 2016, 14(1), 153-170. https://doi.org/10.1007/s10723-015-9345-8

16. Marszałek Z. Parallelization of Fast Sort Algorithm. International Conference on Information and Software Technologies, Springer, Cham, 2017, 408-421. https://doi.org/10.1007/978-3-319-67642-5_34
17. Marszałek, Z. Parallelization of Modified Merge Sort Algorithm. Symmetry, 2017, 9(9), 176. https://doi.org/10.3390/sym9090176
18. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., C., Talbi, E., G., Zomaya, A., Y., Tuyttens, D. A Parallel Bi-Objective Hybrid Metaheuristic for Energy-Aware Scheduling for Cloud Computing Systems. Journal of Parallel and Distributed Computing, 2011, 71(11), 1497-1508. https://doi.org/10.1016/j.jpdc.2011.04.007
19. Paira, S., Chandra, S., Alam, S. K. S. Enhanced Merge Sort-A New Approach to the Merging Process. Procedia Computer Science, 2016, 93, 982-987. https://doi.org/10.1016/j.procs.2016.07.292
20. Połap, D., Kęsik, K., Woźniak, M., Damaševičius, R. Parallel Technique for the Metaheuristic Algorithms Using Devoted Local Search and Manipulating the Solutions Space. Applied Sciences, 2018, 8(2), 293. https://doi.org/10.3390/app8020293
21. Rauh, A., Arce, G., A Fast Weighted Median Algorithm Based on Quick Select. Proceedings of the IEEE 17th International Conference on Image Processing, Hong Kong, China, 2010, 105-108. https://doi.org/10.1109/ICIP.2010.5651855
22. Tsigas, P., Zhang, Y. A Simple, Fast Parallel Implementation of Quick Sort and Its Performance Evaluation on SUN Enterprise 10000. IEEE Proceedings of Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2003, 372-381. https://doi.org/10.1109/EMPDP.2003.1183613