Parallelize Bucket and Bubble Sort Algorithms Using Message Passing Interface (MPI)

المعالجة المتوازية لخوارزميتي الترتيب الفقاعي والدلو باستخدام واجهة تمرير الرسائل

Naeem Ali Askar نعيم علي عسكر
College of Basic Education / University of Duhok
Bilal Abdul rahmanTuama بلال عبد الرحمن طعمة
College of Applied Science/University of Samarra
Naeem.askar@uod.ac Bilal.at@uosamarra.edu.iq

Abstract:

Sorting has been a profound area for the algorithmic researchers and many resources are invested to suggest more works for sorting algorithms. For this purpose, many existing sorting algorithms were observed in terms of the efficiency of the algorithmic complexity. In this paper we implemented the bucket and bubble sort algorithms using Message Passing Interface (MPI) approach. The proposed work tested on two standard datasets with different size. The main idea of the proposed algorithm is distributing the elements of the input datasets into many additional temporary sub-arrays according to a number of characters in each word We implemented MPI using Intel core i3 ,(4

CPUs) . Finally, we get the data structure effects on the performance of the algorithm for that we choice the second approach.

Keywords: Bucket sort, MPI, sorting algorithms, parallel computing.

الخلاصة

يعد ترتيب البيانات وفرزها من المجالات الواسعة والمهمة للباحثين، وقد تم استثمار العديد من الموارد لاقتراح وتطوير الخوارزميات الخاصة بترتيب البيانات، ولذلك تم ملاحظة الكثير من الخوارزميات المعقدة والتي تحتاج لوقت وجهد اكثر لتنفيذها. في هذا البحث قمنا بتطبيق واختبار خوارزميتي الترتيب الفقاعي والدلو على مجموعتين من البيانات النصية (كل مجموعة بحجم مختلف) باستخدام واجهة تمرير الرسائل. الفكرة الرئيسية هي تقسيم عناصر البيانات المدخلة الى مجموعة من المصفوفات الفرعية طبقا لعدد الحروف في كل كلمة من النص وتنفيذها بتقنية واجهة تمرير الرسائل باستخدام وحدة معالجة مركزية تتكون من اربع معالجات بالنهاية تم الحصول على هيكل بيانات بترتيب جديد معتمدا على اداء هاتين الخوارزميتين معا.

1. INTRODUCTION

Sorting is one of the most common operations perform with a computer. Basically, it is a permutation function which operates on elements [4]. In computer science sorting algorithm is an algorithm that arranges the elements of a list in a certain order. Sorting algorithms are taught in some fields such as Computer Science and Mathematics. There are many sorting algorithms used in the field of computer science. They differ in their functionality, performance, applications, and resource usage[2].We are going to give a brief introduction of the most popular sorting algorithms.

2. Bucket and Bubble Sort

Bucket sort: is a sorting algorithm that runs in linear time.it works by partitioning an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. The function should sort the character by word length and by alphabetically order using provided data. Otherwise we also provide time taken to read the character array. The concept of bucket sort[5]:

Divide the data into number of buckets according to length of alphabets.

Sorting each buckets separately using one of sort algorithms.

Parallel bucket sort works as follows:

1. Set up an array of initially empty "buckets."
2. Scatter: Go over the original array, putting each object in its bucket.
3. Sort each non-empty bucket.
4. Gather: Visit the buckets in order and put all elements back into the original array.

Bubble sort algorithm: is the oldest, the simplest and the slowest sorting algorithm in use having a complexity level of O(n2). Bubble Sort Algorithm is a simplest and the slowest sorting algorithm in use having a complexity level of O(n2), that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items

and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort [1].

The sequential version of the bucket sort algorithm is considered to be the most inefficient sorting method in common usage. The performance of the sequential code for two datasets which has been tested for 5 times to get the average is shown in table 1.

Table 1: Sequential for Small Data and Big Data

		Execution time					
Sequential		1	2	3	4	5	Average
	Small Data	6.40	6.45	6.68	6.79	6.47	6.558
	Big Data	562.23	587.31	501.87	537.12	517.87	541.28

3. Methodology

There are two types of text file dataset have been provided in this paper (HAMLET, PRINCE OF DENMARK by William Shakespeare), which are different in size and length. The first dataset is equal (190 KB) and the second one is equal (1.38 MB) taken from (http://www.booksshouldbefree.com).

Pre-processing

We sort the datasets using the Bucket sort algorithm in three phases. In the first phase we are removing / ignoring

the special characters from the text file. In the second phase we divided the text file to number bucket based on the length of characters, all shorter words come be for longer words. In the third phase we sort each vector of string by arranging in the alphabetic order using the bubble sort algorithm. The data set has been tested 5 times to get the average. The sequential performance has been implemented on Dell, Windows 7 Home Premium 64-bit with Intel core i3, (4 CPUs), and 4 GB of RAM.

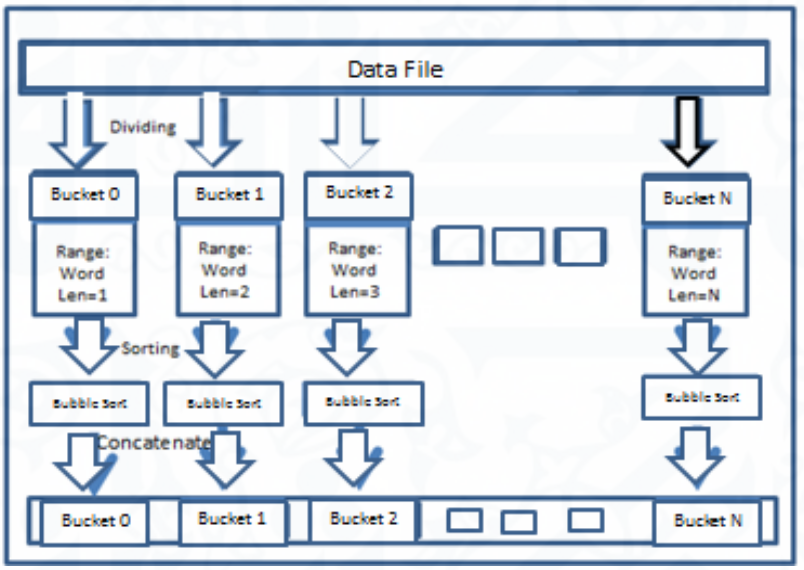

Figure 1: The concept of Bucket Sort

4. Message Passing Interface (MPI)

Message Passing Interface (MPI) is a specification for a standard library for message passing that was defined by the MPI Forum, a broadly based group of parallel computer vendors, library writers, and application specialists(William Gropp). MPI is a language-independent communications

protocol used to program parallel computers. Both point-to-point and collective communication are supported[6]. MPI is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computingtoday [1].

The MPI specification has been implemented for a wide range of computer systems from clusters of networked workstationsrunning general purpose operating systems (UNIX, Widows NT) to high performance computer systems such as CRI T3E, CRI C90, SGI Power Challenge, Intel Paragon, IBM SPI, SP2 etc.

Different versions of MPI are used such as: MPI 1.0, MPI 1.2, MPI 2.0, MPI 1.3, MPI 2.1, and MPI 2.2, these versions have different functionality and facilities[7].

MPI Implementation

We are implementing MPI code using Scatter/ gather and master /slave communications where the first thing we have calculates the size of chunks equal the length of the array divided by the number of processors ”Whenever the chunk size small the parallel time is better” . After that the master will send the data to the slaves and each slave will do a bubble sort on own data. At the end of this step the slave will send the sorted buck to the master, master will marge the buckets into one data structure and then send the completed sorted file (see table 2).

Table 2: The design of MPI

```
MPI_Scatter (Data, elements_per_proc, MPI_CHAR, local_array,
elements_per_proc,MPI_CHAR, 0, MPI_COMM_WORLD);

----------------------------------------------------------------------

MPI_Isend (buckets[i]->Data,1,MPI_CHAR,buckets[i]->next, 0,
MPI_COMM_WORLD);

----------------------------------------------------------------------

MPI_Recv (&local_array , 1,MPI_CHAR,MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
&status);

----------------------------------------------------------------------

MPI_Gather(&buckets,1,MPI_INT,array,1,MPI_INT,0,MPI_COMM_WORLD)
```

A parallel program is intrinsically more complex than its serial counterpart. To write an efficient and scalable parallel program, one must understand the behavior and performance of the program. Methods of analysis for our parallel design are based on Amdahl's law and Gustafson's law.

Amdahl's Law: The existence of non-parallelizable (sequential fraction) computations limits the potential benefit of parallelization[3].

Gustafson-BarsisLaw:Problems with large, repetitive data sets can be efficiently parallelized. It showed that Amdahl's law is invalid for cases where the problem size could be increase and the regularity of the problem could be used to deploy as many processors as the problem needed[3].

Speedup (S) = Ts / Tp

Where Ts is the sequential time and Tp represents the parallel time

Efficiency = S / P

Where S is the speedup and P represents the number of the processors from the system.

Table 3: Execution time, Speedup and Efficiency for Small and Big data

	Small data			Big data		
No. of Processor	Execution time	Speedup	Efficiency	Execution time	Speedup	Efficiency
1	10.23	1	100	745.574	1	100
2	8.05	1.271	63.55	505.506	1.474	73.7
4	5.892	1.736	43.4	321.428	2.319	57.975
6	5.11	2.002	33.3667	278.5	2.676	44.6
8	3.922	2.608	32.6	244.946	3.043	38.038
10	3.831	2.670	26.7	239.326	3.115	31.15
16	3.731	2.742	17.137	233.921	3.187	19.919

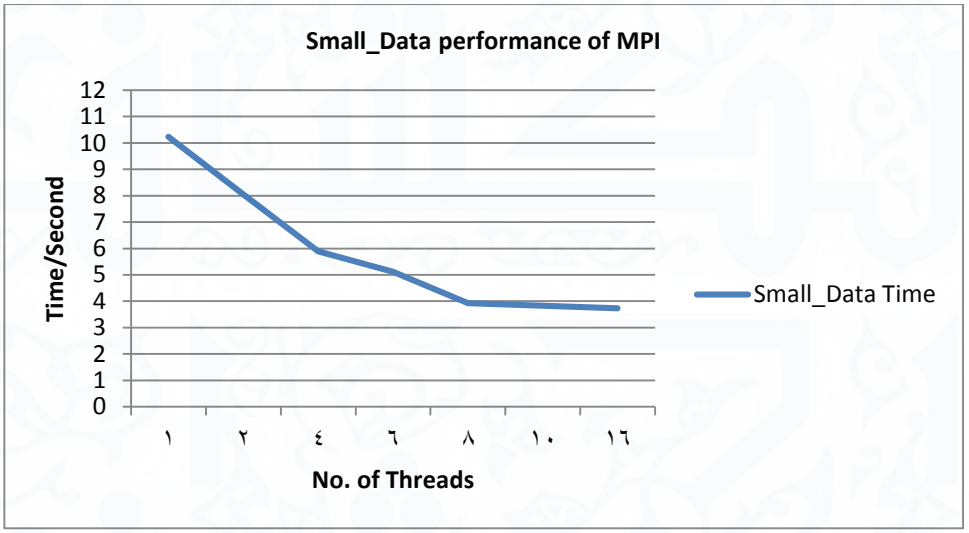

Figure 2: shows the performance of MPI with Small Data

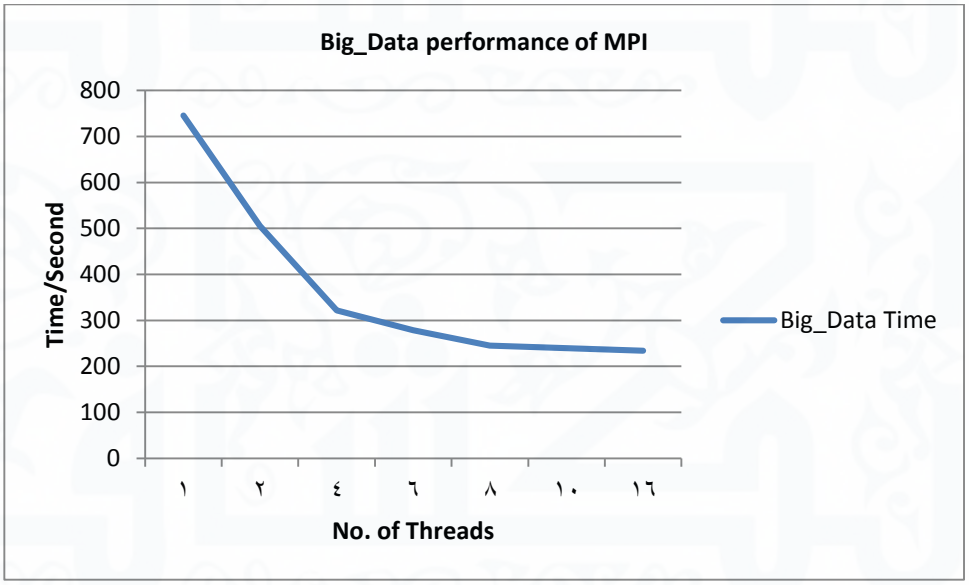

Figure 3: shows the performance of MPI with Big Data

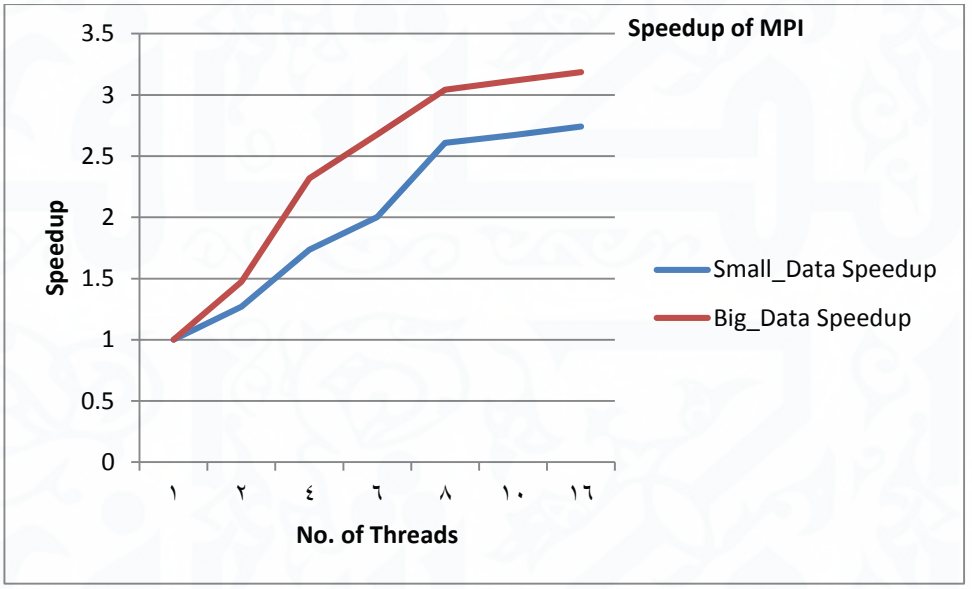

Figure 4: Shows the speedup of Small and Big Data using MPI

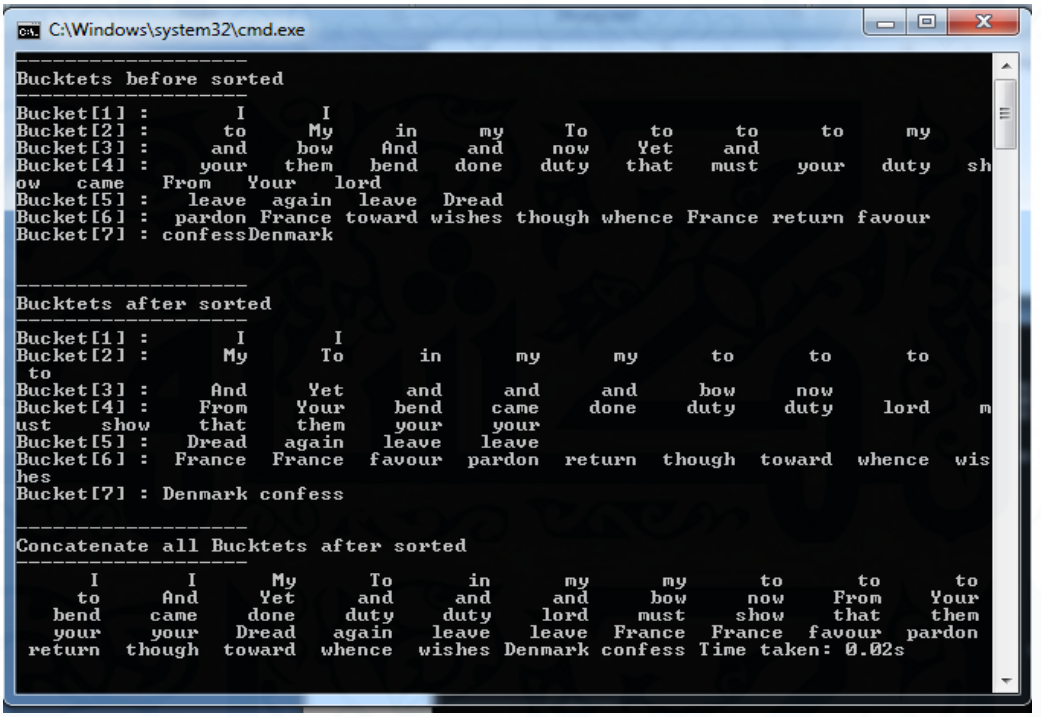

Figure 5: showed the result of bucket sort after sorting by word length for each bucket

6. Conclusions

In this paper we implemented the bucket sort algorithm using massage passing interface (MPI). The proposed work tested on two standard datasets (text file) with different size taken from (HAMLET, PRINCE OF DENMARK by William Shakespeare), (http://www.booksshouldbefree.com/). We implemented MPI using Dell, Operating System: Windows 7 Home Premium 64-bit Processor: Intel core i3 ,(4 CPUs), memory: RAM 4 GB. We were finding the data structure effects on the performance. In MPI, increasing the number of threads it will increase speedup, because the approach of MPI which implements here (Bucket and bubble sort).

REFERENCES

[1] Boris V. Protopopov, A., A Multithreaded Message Passing Interface (MPI) Architecture, Parallel and Distributed Computing,1998, pp. 1-30.

[2] Altukhaim, S. , Bubble Sort Algorithm, Florida Institute of Technology, 2003.

[3] AnanthGrama, Anshul Gupta, George Karypsis and Vipin Kumar, Introduction to Parallel Computing Second Edition, Addison Wesley, 2003

[4] Rahim Rashidy, S.Y. , Parallel Bubble Sort Using Programming Paradigm. IEEE,2011.

[5] T Rożen, K Boryczko, W Alda, GPU bucket sort algorithm with applications to nearest-neighbour search, 2008

[6] William Gopp, E.L., A High-performance, Portable Implementation of the MPI Message Passing Interface Standard.

[7] ZaidAbdi, Kadhim Al-Attar, ..Parallelize Bubble and Merge Sort Algorithms Using Message Passing Interface (MPI), 2014.