Contemporary Engineering Sciences, Vol. 10, 2017, no. 19, 945 - 952
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ces.2017.79120

On the Parallel Implementation of Best Fit Decreasing Algorithm in Matlab

Dimitris Varsamis

Department of Informatics Engineering
Technological Educational Institute of Central Macedonia - Serres
62124, Serres, Greece

Abstract

In this work a parallel implementation of Best Fit Decreasing algorithm in Matlab is presented. The propose of this work is twofold: (a) the reduction of the execution time and (b) the optimal partition of dataset for the minimum cost in results of algorithm. Specifically, a function for the partition of dataset is presented. Additionally, a function for BFD algorithm is developed, which meets all the requirements for parallel computations in Matlab. Finally, the performance tests for the computation times and the results of BFD algorithm respect to the number of dataset are illustrated.

Keywords: Bin-packing problem, BFD algorithm, Parallel processing, Matlab

1 Introduction

Given a set of numbers, and a fixed bin capacity, the bin-packing problem is to assign each number to a bin so that the sum of all numbers assigned to each bin does not exceed the bin capacity. An optimal solution to a bin-packing problem uses the fewest number of bins possible. Optimal bin packing one of the classic NP-complete problems [1].

A approximation algorithm for bin-packing problem is known as Best Fit Decreasing (BFD) [2], [3]. It sorts the elements in decreasing order of size, but

puts each element into the fullest bin in which is fits. It can be implemented by keeping the bins sorted in increasing order of their remaining capacity, and placing each element into the first bin in which it fits.

The execution of BFD algorithm requires large computational cost [4], that leads researchers to implement BFD using parallel techniques. Matlab is a software tool that supports, conveniently, numerical computations and parallel techniques [5], [7], [8]. Additionally, Matlab supports parallel computations either in cluster of computers or in multicore CPUs [6].

In section 2, the parallelization of BFD algorithm with an illustrative example is presented. This implementation consists of two parts. The first one is the partition of dataset and the second one is the parallel execution. Finally, in section 3 the implemented algorithms are tested for performance. The tests include both serial and parallel implementations of BFD algorithm using Matlab software tools and commands.

2 The parallelization of BFD algorithm

In this section we illustrate an example for the parallelization of the Best Fit Decreasing Algorithm. We add a part of code to partition the data into two and four groups and to apply the Best Fit Decreasing Algorithm to them. Each group can be apply in one core or computer machine, the so-called workers, with the result that a set of data is assigned to each one worker when executed at the same time in order to reduce execution time.

Suppose the following numbers are 20 object sizes that we want to place in bins of the same capacity with size 10, using as few as possible. Also the wastage (unused space in bins) must be the minimum. The sizes of the objects are sort.

9 8 8 6 5 5 5 4 4 4 4 4 4 3 3 2 1 1 1 1

First, by running the algorithm in data before partitioning (serial execution), we get the following results as shown in Figure 1. We notice that the wastage size is 8 and the number of used bins are 9. Gray marked are full bins and no color marker bins is not full.

2.1 Data Partition

Each group of the data partition must be a uniform distribution so that the mean and standard deviation of the group data are similar to those before the partition. The code is automated to partition the data according to the number of workers and is in MATLAB function as follows:

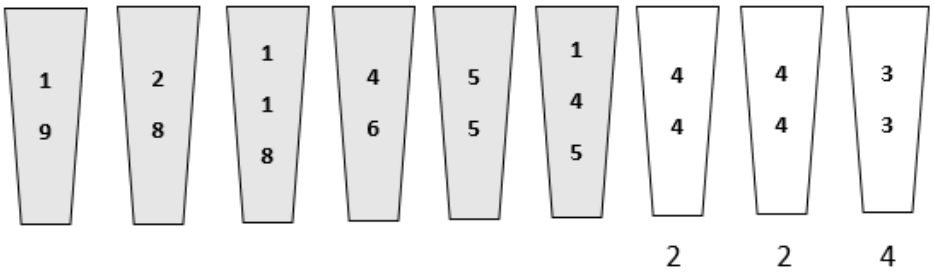

Figure 1: Serial Execution of BFD

```
function a=data_partition(labs,package)
n=length(package);
for i=1:labs
    for j=1:n/labs
        k=j+1;
        a(i,j)=package(labs*(k-1)-mod(i,labs));
    end
end
```

New partitioning data (case of 2 new tables) form initial data result as follows, placing the first element in the new table A, the second element in the new table B, the third element in the new table A, the fourth element in the second new table B, and so on.

For the data of above example in case of two partitions we have

9	8	8	6	5	5	5	4	4	4	4	4	4	3	3	2	1	1	1	1
A	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*

with mean and Standard deviation respectively $\bar{x} = 4.1$, $s = 2.35975$. Thus, the table A is

$$9 \quad 8 \quad 5 \quad 5 \quad 4 \quad 4 \quad 4 \quad 3 \quad 1 \quad 1$$

with $\bar{x}_A = 4.4$, $s_A = 2.59058$ and the table B is

$$8 \quad 6 \quad 5 \quad 4 \quad 4 \quad 4 \quad 3 \quad 2 \quad 1 \quad 1$$

with $\bar{x}_B = 3.8$, $s_B = 2.20101$.
The above partition is middle representational because the mean and the standard deviation of each group has values with little difference.

With the same way we partition the data in 4 groups. For the data of above example in case of two partitions we have

9	8	8	6	5	5	5	4	4	4	4	4	4	3	3	2	1	1	1	1
A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*

with mean and Standard deviation respectively $\bar{x} = 4.1$, $s = 2.35975$. Thus, the table A is

$$9 \quad 5 \quad 4 \quad 4 \quad 1$$

with $\bar{x}_A = 4.6$, $s_A = 2.88097$ the table B is

$$8 \quad 5 \quad 4 \quad 3 \quad 1$$

with $\bar{x}_B = 4.2$, $s_B = 2.58843$ the table C is

$$8 \quad 5 \quad 4 \quad 3 \quad 1$$

with $\bar{x}_C = 4.2$, $s_C = 2.58843$ and the table D is

$$6 \quad 4 \quad 4 \quad 2 \quad 1$$

with $\bar{x}_D = 3.4$, $s_D = 1.94935$.
The above partition is low representational because the mean and the standard deviation of each group has values with significant difference.

2.2 Parallel Execution

In parallel execution we apply the serial BFD Algorithm in each group. The advantage of this method is the reduction of execution time. The disadvantage is the worst results instead to serial execution of the BFD algorithm.

In case of 2 partitions in above example, we apply the BFD Algorithm in table A and table B and we get the following results as shown in Figure 2. The

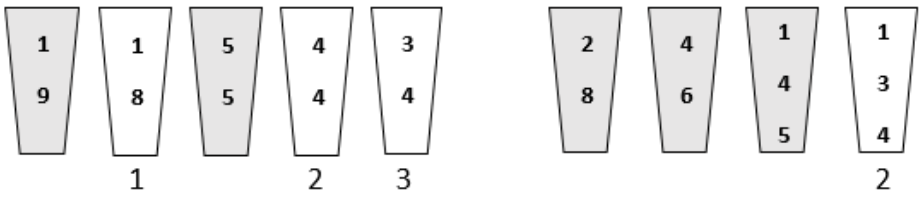

Figure 2: Parallel Execution of BFD Algorithm in two cores

wastage (unused space in bins) in table A is 6 and the number of used bins are 5. The wastage (unused space in bins) in table B is 2 and the number of used bins are 4.

Thus, in parallel execution of BFD in two cores we have total wastage 8 and the number of used bins are 9.

In case of 4 partitions in above example, we apply the BFD Algorithm in table A, table B, table C and table D and we get the following results as shown in Figure 3.

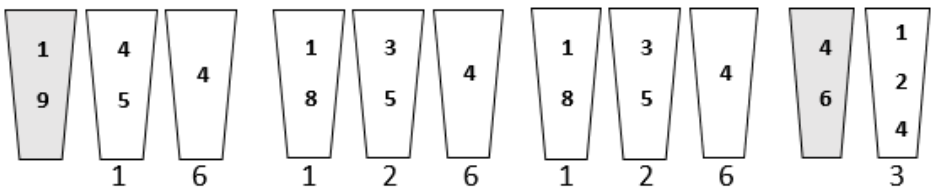

Figure 3: Parallel Execution of BFD in four cores

The wastage (unused space in bins) in table A is 7 and the number of used bins are 3. The wastage (unused space in bins) in table B is 9 and the number of used bins are 3. The wastage (unused space in bins) in table C is 9 and the number of used bins are 3. The wastage (unused space in bins) in table B is 3 and the number of used bins are 2.

Thus, in parallel execution of BFD in two cores we have total wastage 28 and the number of used bins are 11.

The total results are shown in Table 1

Table 1: Comparison of Serial and Parallel Execution

	Serial	Parallel (2cores)	Parallel (4cores)
Total Wastage	8	8	28
Total number of used Bins	9	9	11

Consequently, we see that as the number of groups in which data is divided increases, both the wastage and the number of used bins increase. This is dues to the pattern of partition of the groups and to the individual statistical values that have the data. The more representative the partition of the groups, the less variations in the results will be. It is important to note that the volume of data used to analyze the example is very small. Experimental measurements respect to execution time, wastage and number of used bins will then be presented.

3 Performance tests and Results

The performance tests are implemented in an efficient computing system with the following characteristics: CPU Intel Xeon E5640 64x 2.67GHz (multicore) and RAM 16GB. Additionally, for the accuracy of the performance tests, the execution time of the tests are calculated with the formula is given by

$$Time = \frac{t_1 + t_2 + t_3 + \ldots + t_{12} - t_{Max} - t_{Min}}{10}$$

where $t_i (i = 1, ..12)$ is the execution time of each run with the same data and parameters [9].

The BFD algorithm is implemented in Matlab and the parameters of the BFD algorithm are the following: (a) *mat*, the matrix of dataset and (b) $C = 100$, the length of bin. Each object of dataset is a random integer number between 1 and 100. The corresponding data are created using the build-in function of Matlab `randi()`.

The performance tests are implemented respect to the number of objects in dataset and respect to the number of cores. In particular, the values of parameters are $n = 2^{14}, 2^{16}, 2^{18}$ and $cores = 1, 2, 4, 8$.

In case of $cores = 1$ and $n = 2^{14}$ we have serial implementation with mean and Standard deviation respectively

$$\bar{x} = 50.47894 \ , \quad s = 28.79547$$

In case of $cores = 2$ and $n = 2^{14}$ we have parallel implementation (partition in 2 groups) with mean and Standard deviation respectively

$$\begin{aligned} \bar{x}_1 &= 50.48168 \ , \quad s_1 = 28.79616 \\ \bar{x}_2 &= 50.47619 \quad , \quad s_2 = 28.79654 \end{aligned}$$

The above partition is high representational because the mean and the standard deviation of each group has almost similar values.

In case of $cores = 4$ and $n = 2^{14}$ we have parallel implementation (partition in 4 groups) with mean and Standard deviation respectively

$$\begin{aligned} \bar{x}_1 &= 50.48754 \quad , \quad s_1 = 28.79745 \\ \bar{x}_2 &= 50.48291 \quad , \quad s_2 = 28.79803 \\ \bar{x}_3 &= 50.47583 \quad , \quad s_3 = 28.79840 \\ \bar{x}_4 &= 50.46948 \quad , \quad s_4 = 28.79856 \end{aligned}$$

The above partition is high representational because the mean and the standard deviation of each group has almost similar values.

The same statistical results have in case of $cores = 8$ and for $n = 2^{16}$, $n = 2^{18}$, respectively.

The results in terms of wastage are shown in table 2, in terms of number of used bins are shown in table 3 and in terms of execution time are shown in table 4, respectively.

4 Conclusion

From the aforementioned analysis it becomes evident that the parallel implementations of BFD lead to ameliorated performance in terms to execution

Table 2: The wastage of BFD vs cores and n

n	1 core	2 cores	4 cores	8 cores
2^{14}	1036	1136	1236	1236
2^{16}	4219	4319	4319	4319
2^{18}	14320	14420	14420	14620

Table 3: The number of used bins of BFD vs cores and n

Dataset	1 core	2 cores	4 cores	8 cores
2^{14}	8227	8228	8229	8229
2^{16}	33041	33042	33042	33042
2^{18}	132897	132898	132898	132900

Table 4: The execution times of BFD vs cores and n

Dataset	1 core	2 cores	4 cores	8 cores
2^{14}	0,2308	0,4359	0,3633	0,6056
2^{16}	13,6948	3,6948	1,3277	0,9903
2^{18}	204,9967	53,2644	17,0844	7,3111

time. The parallel implementations approach the results of serial implementation. The differences in results (wastage and number of used bins) between serial and parallel implementation is up to 1% while the differences in execution time is at least 70%.Thus, the parallel implementation improves the execution time of BFD algorithm in large dataset.

It needs to be noticed that both serial and parallel Matlab implementations run on the same computing system, with the same resources. The parallel algorithms take advantage of the resources of the computing system, namely the cores of CPU.

Acknowledgements. The authors wish to acknowledge financial support provided by the Research Committee of the Technological Education Institute of Central Macedonia, under grant SAT/IC/18072017-112/10.

References

[1] M. R. Garey, D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness,* W. H. Freeman & Co., New York, NY, USA, 1990.

[2] R. E. Korf, A New Algorithm for Optimal Bin Packing, *Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada,* (2012), 731-736.

[3] A. N. Zehmakan, Bin Packing Problem: Two Approximation Algorithms, *CoRR,* (2015), abs/1508.01376.

[4] D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, R. L. Graham, Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms, *SIAM J. Comput.,* **4** (1974), 299-325. https://doi.org/10.1137/0203025

[5] P. Luszczek, Parallel Programming in MATLAB, *International Journal of High Performance Computing Applications,* **23** (2009), no. 3, 277-283. https://doi.org/10.1177/1094342009106194

[6] C. Moler, Parallel MATLAB: Multiple Processors and Multiple Cores, *The MathWorks News & Notes,* (2007).

[7] G. Sharma, J. Martin, MATLAB: A Language for Parallel Computing, *International Journal of Parallel Programming,* **37** (2009), no. 1, 3-36. https://doi.org/10.1007/s10766-008-0082-5

[8] D. N. Varsamis, C. Talagkozis, P. A. Mastorocostas, E. Outsios, N. P. Karampetakis, The performance of the MATLAB Parallel Computing Toolbox in specific problems, *Federated Conference on Computer Science and Information Systems (FedCSIS), Wroclaw, Poland,* (2012), 587-593.

[9] D. N. Varsamis, P. A. Mastorocostas, A. K. Papakonstantinou, N. P. Karampetakis, A parallel searching algorithm for the insetting procedure in Matlab Parallel Toolbox, Advanced Information Science and Applications Volume I, *18th Int. Conf. on Circuits, Systems, Communications and Computers (CSCC 2014),* 2014, Santorini Island, Greece, (2014), 145-150.

Received: October 11, 2017; Published: October 25, 2017