Journal of Student Research
Volume 9 Issue 2 (2020)
Research Article

Accelerating and Improving Simulation Performance in Communication Systems Modeling through Parallel Computing and Clustering

Kacper Kapuśniak[1], Tanmay Nautiyal[2] and Ryan Grammenos[1]

[1]Department of Electronic and Electrical Engineering, University College London, United Kingdom
[2]Department of Computer Science, University of Hong Kong, Hong Kong
DOI: https://doi.org/10.47611/jsr.v9i2.829

ABSTRACT

Existing 5G communication systems suffer from two major problems: the need for better spectrum efficiency and lesser adjacent channel interference. Thus, development of novel waveform techniques to overcome these problems is a major topic of research amongst scholars and it requires carrying out Monte Carlo simulations in MATLAB© (by MathWorks) to measure the Bit Error Rate (BER) of these communication models. As most of these simulations require millions of computations, they take a significantly long time to run (for example, days) as they run on single-core machines and carry out the computations serially. The main objective of this research is to reimplement current scripts using various parallel computing techniques in MATLAB to study which one is the best suited for this particular type of simulations while also scaling these scripts onto a multi-core cluster to further improve the execution time.

Introduction

The mass adoption of smartphones has led to an increased demand for faster wireless connectivity at an unprecedented scale. In order to meet this demand, the 5th generation of communication systems (commonly known as 5G) make use of Non-orthogonal transmissions, which are unable to use current detection methods due to interference from adjacent streams. To overcome this obstacle, 5G systems require the development of new novel waveform techniques, which improve spectral efficiency while maintaining acceptable Bit Error Rates (BER) [1]. In this work, we consider one example of a 5G waveform, known as Spectrally Efficient Frequency Division Multiplexing (SEFDM).

In order to measure the performance of various waveform techniques, Monte Carlo simulations are carried out in MATLAB to measure their BER. Two of these techniques which are of particular interest are Zero-Forcing (ZF) and Sphere Decoding (SD), with Regular Sphere Decoding yielding much lower BERs but consequently taking significantly longer in processing the signal matrix compared to ZF. As most of these

computations are carried out serially and only make use of single cores on machines, they can potentially take days to run a single simulation.

MATLAB includes a Parallel Computing Toolbox, which has various features and functions that allow independent tasks to run in parallel rather than serially on multi-core machines, thus speeding up data-intensive computations [2]. Since, each iteration of the simulation is independent of each other, it is well suited for making use of parallel computing techniques. MATLAB also offers a Parallel Server, which allows a user to scale up their code and run it on clusters, thus leading to further improvement in execution speeds. The University College London (UCL) Myriad cluster is configured with the Parallel Server, allowing users to install the cluster profile on their devices and submit jobs remotely.

Thus, this paper explores the use of specific parallel computing features offered by the MATLAB toolbox (namely Parfor, Single Program Multiple Data and Parfeval) to find which one is best suited to this particular task. Subsequently the approach is then scaled onto the UCL cluster to make full use of parallel computing functionality available.

Existing Work

Since the introduction of the parallel computing toolbox in MATLAB in 2004, a number of research papers have discussed its implementation to speed up performance of simulations. Many papers have shown the parallel programming advantage over the traditional serial programming [3]. Although many of these papers focus on Monte Carlo simulations [4] as a whole, little research has been conducted on 5G communications systems related simulations specifically. Even the MATLAB Documentation includes an extensive guide to speeding up Monte Carlo simulations [5], but has limited suggestions for the improvement in BER simulation [6]. Most of the techniques it suggests are limited to breaking down the problem into smaller parts to be parallelised based on the Energy-per-bit-to-noise-spectral-density ratio (EbNo), which is not efficient as each EbNo simulation takes a different amount of time to run. Also, the guide's focus is limited to the parfor technique and machines with four cores, making its approach inefficient on clusters as the number of EbNo to be simulated is usually small (usually in the range of 8 dB to 12 dB), thus leaving a lot of cores unused and decreasing efficiency. The improvement in execution time is limited to three times faster than a non-parallelized version of the script, which is not very significant as the scripts still take multiple days to run. Keeping these in mind, the focus of our research is going to be on further breaking down the problem into smaller parts, experimenting with other parallel computing techniques and designing the script while taking multicore clusters into consideration in order to maximise performance and further reduce execution times.

Theory

A. 'Parfor'

Parfor loop, or the parallel for loop, executes iterations in parallel using multiple cores in a pool. It executes in a non-deterministic order and automatically allots tasks to cores when they become free, making the implementation simple

and efficient, which makes it a viable candidate for this study [7]. Also, the original code designed to run on single-core machines uses 'for' loops for the core processing of their simulations thus, making the implementation of 'parfor' loops quite straightforward in this case.

B. 'SPMD'

'SPMD' stands for single program, multiple data [8] and is a parallel technique which simply executes the code between 'spmd' and 'end' in parallel on workers of the parallel pool. While other techniques like 'parfor' are restricted to operations in a loop, 'spmd' can parallelise an entire workflow of any type. However, 'spmd' requires manually and explicitly dividing the work between the workers by using 'labindex' [9], thus increasing the complexity of implementing parallelization. Furthermore, 'parfor' is optimised for loops and thus, usually gives better results for the same code executed with 'spmd'. On the other hand, where the explicit division of the work between the workers may be used, the 'spmd' typically outperforms 'parfor' [10].

C. 'Parfeval'

The 'parfeval' function allows for the parallel execution of functions asynchronously in a parallel pool. The function can request execution of the function in the parallel pool without blocking MATLAB operations, thus leaving other workers free to be allotted to other tasks [11]. This offers numerous advantages as we may parallelize unrelated but equally complex operations in MATLAB and solve them simultaneously, which is not possible with the 'parfor' loop. Although 'parfeval' is more complex in terms of implementation, it serves as a viable option as it also allows the parallelisation to be optimised specifically for each code.

Research Methodology and Approach

The BER performance of the current MATLAB scripts was used as a benchmark to verify the correct functionality of the newly modified scripts designed to run on parallel computing clusters. The scripts follows the format of a standard Monte Carlo simulation, with certain modifications made to achieve the desired results. There are two functions which support the script, RegCSD, which is the Complex Sphere Decoder, and Enum, which essentially computes the position of the variables in the constellation. There is also another 'for' loop nested within the main loop, which essentially further breaks down each simulation and computes the BER for each block of SEFDM symbols.

A closer look at the code reveals that there are three main variables which affect the execution time of the code. Timed simulations are then carried out to observe the exact relationship between these factors and the execution time:

1) Number of sub-carriers ('n'): The execution time increases exponentially as n increases.
2) 'Eb_No': 'Eb_No' shares a linear relationship with respect to the computation time.
3) Number of SEFDM symbols ('NoBlocks'): The computation time is directly proportional to the number of SEFDM symbols and shares a linear relationship with it.

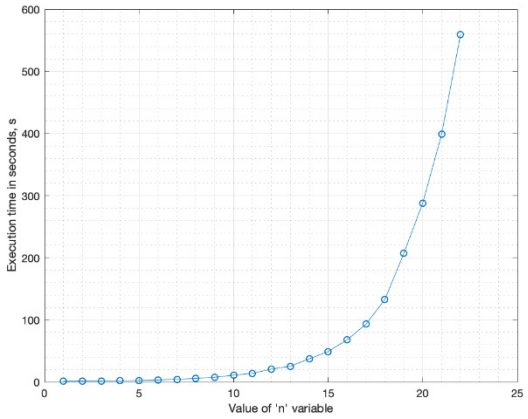


Figure 1: Execution time versus number of subcarriers on a single-core machine (EbNo = 8, NoBlocks = 300)

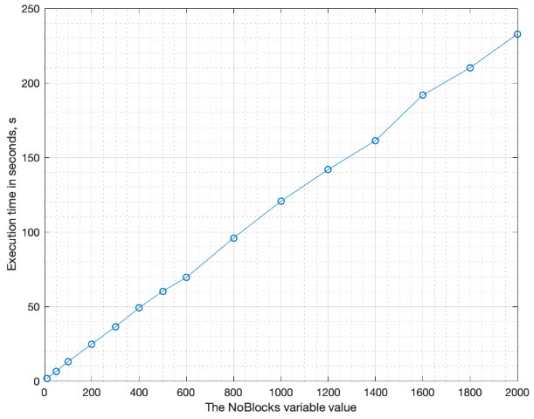


Figure 2: Execution time versus number of SEFDM symbols on a single-core machine (n = 12, EbNo = 8)

On using the MATLAB profiling tool on the code, it was discovered that 99% of the execution time was taken by the RegCSD function. The code profiling of the RegCSD function itself suggested that the significant execution time of the function is not due to any specific time-consuming operations, but due to a large number of times the function is called (which for 24 sub-carriers is as high as 200 million according to the profiler). Thus, it was concluded that the most appropriate approach for parallelisation is by splitting the various iterations amongst the cores in the most efficient way possible and only two operations met the criteria: the individual iterations of each 'EbNo' simulation, and the iterations for the transmission of SEFDM symbols within. Both iterations were using the 'for' loop and the latter was nested within the former.

A. Implementing 'Parfor'

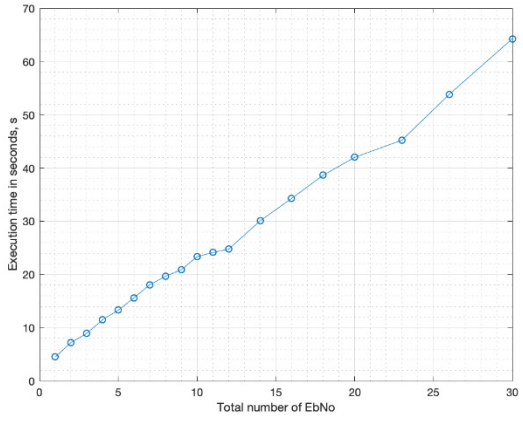
In case of nested 'for' loops, since both loops cannot be reimplemented with 'parfor' [11], the approach is to implement them separately for the 'outer' loop and 'inner' loop and use the version which shows a greater improvement.

The use of outer 'parfor' suffered from a number of drawbacks, mainly because it was limited by the number of 'EbNo' simulations to be run, which was usually lesser than the number of workers and thus, left any extra workers unused.

Also, since the Regular Sphere Decoder is sensitive to noise levels, the execution time is inversely proportionate to 'EbNo' (the higher the 'EbNo', the shorter the execution time). Thus, iterations take unequal amounts of time to complete and the time taken by each batch of computations is equal to the time taken by the longest iteration. Thus, resources were not being fully utilized in the outer 'parfor' iteration.

Implementing 'parfor' on the inner loop avoids the drawbacks of implementing 'parfor' on the outer loop, as the number of iterations always exceeded the available number of workers, as the 'NoBlocks' usually has values more than 300.

Figure 3: Execution time versus the total number of EbNo on a single-core machine (n = 12, NoBlocks = 8)



This is because

$$NoBlocks = \frac{10 * (Desired\ BER)^{-1}}{n * bits/symbol}$$

where n>20, bits/symbol = 2, Desired BER=10^(-3), leaving NoBlocks ≈ 250

Also, since the time taken by each iteration of the inner loop for an 'EbNo' were roughly the same, all the resources are utilised more efficiently and the time taken does not depend on the longest outer loop iteration but on the sum of the times taken by all iterations, which is considerably lesser due to better utilisation of resources.

Thus, it is observed that the 'inner parfor' is the most efficient implementation of 'parfor' in terms of computation time and thus, further tests are conducted on it after scaling it up to a cluster.

B. Implementing 'SPMD'

As parallelisation has a significant overhead attached to it, it is important to use 'spmd' only on the part of the code which will show improved timings in parallel. As established by the MATLAB profiler, the most time-consuming operations take place in the inner 'for' loop and especially in the RegCSD function. The code outside the main 'for' loop executes in minimal time, which means that parallelizing the code outside the 'for' loops would not improve the performance and might actually slow it down, because of the overheads [12]. That is why our approach is to again parallelize only the 'for' loops region, which means that we are essentially just using 'spmd' instead of 'parfor' loop for the same part of the code. In theory, although this should give better results than the original code, this version will still be slower than 'parfor', as the 'parfor' loop is optimised to minimise overheads in loops. Additionally, 'parfor' automatically allots workers with the next job from the queue when it finishes the previous job, thus making better use of the workers. 'SPMD' on the other hand requires jobs to be submitted manually in groups, the size of which depends on the number of workers being used. Thus, even if a worker finishes a job, it has to wait for the most time consuming task in the batch to finish, spending a significant amount of time in the idle state .

This increases the execution time and thus, we predict that the 'parfor' will outperform the 'spmd', and the tests that are carried out reveal the same.

C. Implementing 'Parfeval'

As established before, parallelizing the iterations through the number of blocks is more efficient than parallelizing the iterations through the number of 'EbNo'. For this reason, we implement 'parfeval' in place of the inner loop instead. The advantage of that solution is that the region is not blocked which means that the last operations for the previous 'EbNo' iteration may be executed at the same time as the first operation for the next 'EbNo' iteration, whereas using 'parfor' loop we have to wait until all operations in the previous 'EbNo' are completed, possibly leaving some workers unused for some time. On the other hand, in the 'parfeval' the impact of the overhead may be more significant than in 'parfor'.

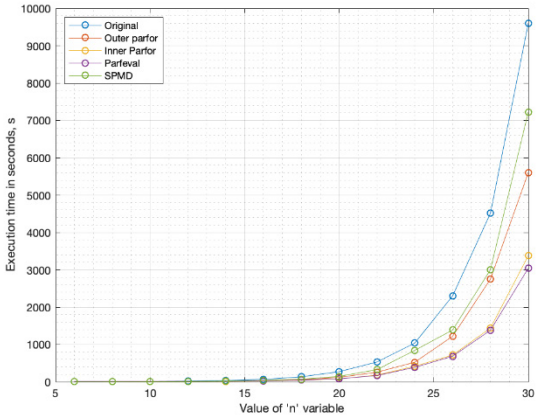


Figure 4: Comparison of the original, parfor, parfeval and 'SPMD' versions of the script (local tests)

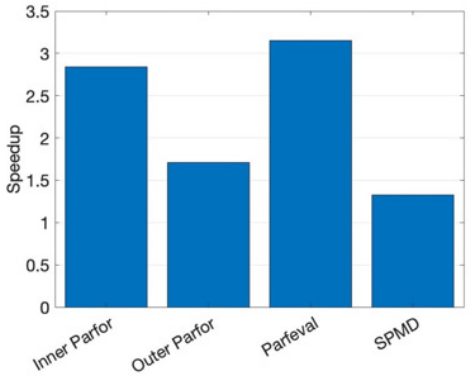


Figure 5: Improvement of different versions of the script on the 4 core machine for n=30 (local tests)

Results and Analysis

After testing all possibilities, we conclude that the most suited techniques for this scripts are the 'inner parfor' loop and 'parfeval,' depending on the value of n. Thus, the next step is to scale up the computations to a cluster and see which version performs better.

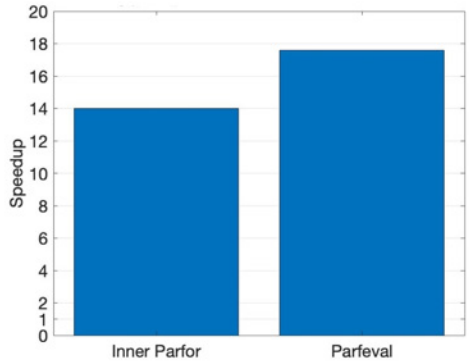


Figure 6: Comparison of the different scripts on the cluster

As 'inner parfor' performed better for smaller values of 'n' than 'parfeval', the two were then scaled up onto the UCL Myriad Cluster, making use of 30 workers at the same time to study which version performs better with more cores. Although both versions showed improvement in comparison to the original script,

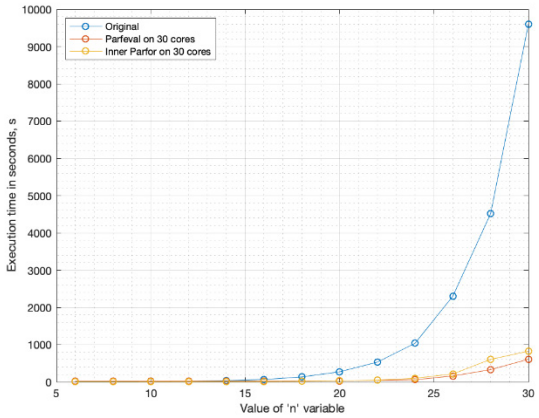


Figure 7: Improvement of different versions of the script on the 30 core cluster

'parfeval' outperformed the 'inner parfor' by significant margins for the target range of 'n' (n>30), executing in approximately 4/5 of the time taken by 'parfor'.

Conclusion

As a final benchmark for measuring the improved results, we ran the original script on the local machine and the 'parfeval' script on the cluster for n=40, which is a significantly heavy task in terms of computations. The original version ran for around six days, while the new version was able to perform the same computations in less than eight hours. These results will not only enable the team to significantly speed up

the pace of the research, but they also go a long way in demonstrating the extraordinary power of parallel computing.

Original Script on PC	New Script on Cluster
505720 seconds ≈ 140 hours	28745 seconds ≈ 8 hours

(For n=40, m=4, EbNo_final=8, NoBlocks=300, alpha=0.8)

Essentially, we were able to speed up the simulations by **seventeen times** in our tests and since the function of time taken by the simulations is exponential, the improvement only increases for larger initial variables. The improvements are of over **one order of magnitude**, and are **six times faster** than the results of the approach recommended by the MATLAB documentation for BER simulations. All in all, the results showed that for this particular case, the 'parfeval' function serves as the most efficient solution as it makes maximum utilisation of all cores in the machine.

Technical specifications

Local Tests

Processor: Intel Core i7-4770 CPU@3.40 GHz x 8
Memory: 32GB
GNOME : Version 3.28.2
Operating System: CentOS Linux 7 (64-bit)
Disk 512GB
MATLAB Version: MATLAB R2018b

Cluster tests

All nodes:
2 x Intel Xeon Gold 6140 18C 140W 2.3GHz Processor (36 cores total)
12 x 16GB TruDDR4 RDIMM (192 GB total) *2TB
7.2K SATA HDD
Mellanox ConnectX-5 EDR/100Gb IB single port VPI HCA
12 x 16GB TruDDR4 RDIMM (192 GB total)
MATLAB Version: MATLAB R2018b

Only standard H node:
Lenovo SD530 Standard Nodes
Only type J GPU nodes:
Lenovo SD530 GPU Nodes

2 x nVidia Tesla P100 Adapter

Acknowledgements

The authors acknowledge the use of the UCL Myriad High Throughput Computing Facility (Myriad@UCL), and associated support services, in the completion of this work. The authors would also like to thank UCL's Laidlaw and Undergraduate Research Opportunities Scheme (UROS) programmes which helped fund the research that led to these results. Finally, the authors are grateful to Dr Tongyang Xu, member of the Information and Communication Engineering (ICE) group in the department of Electronic and Electrical Engineering at UCL, for providing the original MATLAB script used to carry out Monte Carlo BER simulations.

References

[1] Jayawardena, C., & Nikitopoulos, K. (2018). Massively Parallel Detection for Non-Orthogonal Signal Transmissions. *2018 IEEE Globecom Workshops (GC Wkshps).*

[2] Moler, C. (2007). Parallel MATLAB: Multiple Processors and Multiple Cores. *Technical Articles And Newsletters, 91467v00.*

[3] Abhay B. Rathod, Rajratna Khadse, & M Faruk Bagwan. (2014). SERIAL COMPUTING vs. PARALLEL COMPUTING: A COMPARATIVE STUDY USING MATLAB. *International Journal Of Computer Science And Mobile Computing, 3(5).*

[4] Ge, H., & Asgarpoor, S. (2011). Parallel Monte Carlo simulation for reliability and cost evaluation of equipment and systems. *Electric Power Systems Research, 81(2),347-356.doi:10.1016/j.epsr.2010.09.012*

[5] MathWorks. (n.d.). Improving Performance of Monte Carlo Simulation with Parallel Computing - MATLAB & Simulink. Retrieved June 17, 2019, from https://in.mathworks.com/help/finance/improving-performance-of-monte-carlo-simulation-with-parallel-computing.html

[6] MathWorks. (n.d.). BER Simulations with Parallel Computing Toolbox - MATLAB & Simulink. Retrieved June 18, 2019, from https://in.mathworks.com/help/comm/examples/ber-simulations-with-parallel-computing-toolbox.html

[7] Luszczek, P. (2009). Parallel Programming in MATLAB. *The International Journal Of High Performance Computing Applications, 23(3).*

[8] Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999, "single program multiple data", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 17 December 2004.

[9] Sharma, G., & Martin, J. (2008). MATLAB®: A Language for Parallel Computing. *International Journal Of Parallel Programming, 37(1), 3-36. doi: 10.1007/s10766-008-0082-5*

[10] Varsamis, D., Talagkozis, C., Mastorocostas, P., Outsios, E., & Karampetakis, N. (2012). The performance of the MATLAB Parallel Computing Toolbox in specific problems. *Advances In Information Science And Applications, 1.*

[11] The MathWorks, Inc. (2019). *Parallel Computing Toolbox™ User's Guide (7th ed.).*

[12] CAO Jun-jun, FAN Shan-shan, & YANG Xuan. (2012). Spmd Performance Analysis With Parallel Computing of Matlab. *2012 Fifth International Conference On Intelligent Networks And Intelligent Systems.*