Delft University of Technology
Software Engineering Research Group
Technical Report Series

Improving the Performance of a Large Scale Spreadsheet: A Case Study

Alaaeddin Swidan, Felienne Hermans and Ruben Koesoemowidjojo

Report TUD-SERG-2016-003

TUD-SERG-2016-003

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`http://www.se.ewi.tudelft.nl/techreports/`

For more information about the Software Engineering Research Group:
`http://www.se.ewi.tudelft.nl/`

Note: Accepted for publication in the Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering, 2016, IEEE Computer Society.

Improving the Performance of a Large Scale Spreadsheet: A Case Study

Alaaeddin Swidan
Delft University of Technology
Delft, Netherlands
Email: alaaeddin.swidan@tudelft.nl

Felienne Hermans
Delft University of Technology
Delft, Netherlands
Email: f.f.j.hermans@tudelft.nl

Ruben Koesoemowidjojo
BitBrains IT Services
Amsterdam, Netherlands
Email: ruben.koesoemowidjojo@bitbrains.nl

***Abstract*—Spreadsheets are used extensively for calculations in several domains, especially in finance and insurance. Spreadsheets offer a clear benefit to their users: they are an easy to learn application in which to express their business needs, however, there are downsides too. Like software, spreadsheets can have a long life span in which they are used by several people. This leads to maintainability issues, including errors, but also often to issues with performance. In this paper we present a case study in which a model for *shortfall calculations*, originally implemented in a spreadsheet, was adapted to run on an HPC cluster. We present the design, analysis and implementation of the solution which clearly improved the performance of the spreadsheet, with a factor of 50 in some cases. We subsequently reflect on challenges related to reverse engineering, testing and scalability. Finally, we identify opportunities that would provide automatic support to refactoring, dependency recognition and performance profiling in future spreadsheet optimization projects.**

I. Introduction

Research shows the prevalent use of spreadsheets in various business domains in general, and financial services in particular [1], [2], [3]. Spreadsheet usage ranges from trivial data manipulation to complex modeling and simulation [3]. A spreadsheet model in principle is easy to create: domain experts convey their ideas into the intuitive UI, allowing for immediate results to show, and thus providing the ability to react with a business decision such as buying or selling certain stock. Within organizations, spreadsheets are used for long periods of time [2], which could lead to a spreadsheet model growing in size and complexity. Increased size and complexity could affect the performance of the spreadsheet, hampering immediate output and thus inhibiting one of the strongest features of spreadsheets [2]

To investigate the issues around spreadsheet performance in practice, we present a case study of one large-scale spreadsheet model which suffered from performance problems. The spreadsheet was developed at a major insurance company in the Netherlands. It was used to support risk analysts in comparing pension investments and obligations. The spreadsheet suffered from severe performance issues, as the execution regularly exceeded 10 hours. This situation was inconvenient to the business operations, as the time-consuming runs of the spreadsheet caused additional difficulties in its maintenance and usability for the analysts. To address these problems, a parallel-based solution was applied to the spreadsheet, based on a Microsoft HPC cluster. In this paper, we highlight the steps followed to improve the performance of the spreadsheet.

The contributions of this paper are: (i) providing an industrial investigation into addressing performance issues related to simulation models in spreadsheets, (ii) identifying the challenges that we faced throughout the project, and how they were overcome. Finally, (iii) we introduce possible enhancements, which we foresee could mitigate the risks in similar projects.

II. Background and motivation

Simulation models in general can be either deterministic or stochastic. In deterministic models, input values are set before the run, and calculations are built to produce an output accordingly. Running a deterministic model for many times will produce the same result in each run. Opposite to that is the stochastic modeling, where the knowledge of the inputs is neither complete nor certain. In a stochastic model, some of the inputs are chosen randomly, and usually independently, from within a range of distributed values. This is a common modeling approach used in the financial service-providers, banks and insurance companies that highly depend on live information of stocks. In each run of the model, a different set of inputs is used, leading to unique and discrete results. One of the most popular stochastic modeling techniques for numerical analysis is the Monte Carlo simulation [4]. As illustrated in Figure 1, Monte Carlo simulations aim to evaluate a specific calculation, a function, depending on a group of input parameters that are discrete, independent and randomly evaluated from a range of possible values. In Monte Carlo, the main calculation function is executed for a predefined number of iterations $\boldsymbol{n}$. Each iteration's result is recorded, and participates in presenting the final outcome to the user in the shape of graphs, aggregations and probability analysis.

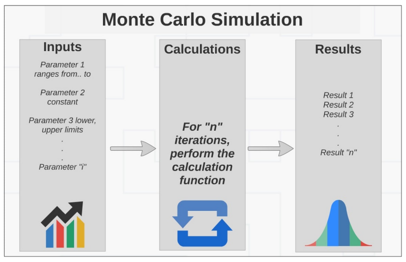

Fig. 1: Monte Carlo Simulation Summary

A. Motivation

Monte Carlo simulations by nature suffer from performance issues. This is usually the case since analysts and domain experts prefer to run their simulations for as many iterations as possible. By this, they aim to produce more precise and reliable results. However, this leads to a high utilization of computing resources, and thus a growth in the runtime of the model relative to the number of iterations. The overall health of the model is also affected, because its maintenance becomes an issue. In our case for instance, modifying the model meant that a simple debugging, of any change, would take tens of hours to finalize.

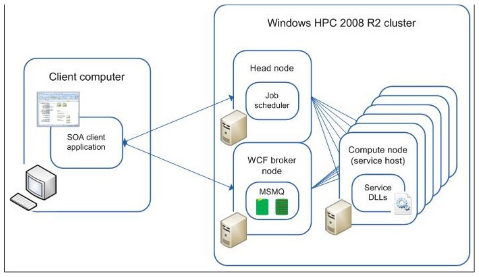

Fig. 2: Microsoft HPC Standard Package [10]

III. Context

The company at which the spreadsheet was developed, is one of the largest financial service providers in the Netherlands, with millions of customers and thousands of employees. In the next subsections we provide information about the use case and aspects of the spreadsheet model. From now on, the company will be referred to as the insurance company.

A. Shortfall Analysis

The investigated spreadsheet model is called the shortfall risk calculator (SFC), and contains an application of a Monte Carlo simulation. A shortfall analysis aims to numerically predict the deficit of the pension liabilities of a company, compared to their stock market capitalization [5]. This is implemented through a stochastic evaluation of the expected returns of the company's investments in equity markets for example, while simultaneously comparing it to the probabilities of the pension obligations at a certain point in the future. An SFC model usually involves a high level of uncertainty and deals with continuously changing data, such as the interest rate, stock compounded market rate, and mortality intensity of a person in a certain age [5].

In our case, the SFC model was implemented using Excel, and used to run it on the common enterprise PCs. The problem is that the model usually took many hours to complete, sometimes days. Worse case, the model would crash unexpectedly and the analysts had to run it again. The performance of the model caused a major problem to the risk analysis department by delaying the decision making process. Moreover, this made it difficult to maintain the spreadsheet model. For example to add a new functionality, debug or test any modification, meant that a new run will start, which then will take another tens of hours to complete.

B. Features of the Model

In this section we present the metrics of the original spreadsheet model and the contained VBA code. For that, we used our research spreadsheet analyzer [6], and a leading software tool for VBA code quality [7]. These metrics are shown here to illustrate the model features, but were not used by the developers in the case.

The SFC model in our case, features one core spreadsheet which is the shortfall calculator. The core spreadsheet depends on three external spreadsheets for providing input data and configurations. It produces data into three other spreadsheets. The core spreadsheet contains 12 worksheets, and a total of 3,329 nonempty cells. Within these cells, 329 contain formulas of which 72 are unique. The formulas used have low level of complexity, which is indicated by the number of cells (16) containing functions such as VLOOKUP, OFFSET, and IF.

The VBA project in the core spreadsheet has 37 source files with more than 7,000 LoC. The code contains 23 modules, 146 methods, and 58 user-defined types. We generated the standard complexity metrics for methods and types, finding the maximum Cyclometic Complexity (CC) of 386 paths for the `main()` method. Another 16 methods are labeled as too complex by the software tool we used, since they have CC value above 20 paths.

IV. The Case Study

As an HPC solution provider, our industry partner BitBrains [8] was approached by the insurance company to improve the spreadsheet's performance. In collaboration with GridDynamics [9], the solution we provided is based on a framework called the HPC for Excel Acceleration Toolkit (HEAT). HEAT, developed by GridDynamics, is built on top of a standard Microsoft HPC cluster [10], and allows for the spreadsheet calculations to be offloaded into the HPC nodes (Figure 2). The project was carried out in three phases that included: setting up the design based on parallelism (Section IV.A), followed by analyzing the spreadsheet model to decompose its components (Section IV.B), and implementing needed changes to the original model (Section IV.C), leading to clear improvements in the model performance as presented in the evaluation (Section IV.D).

A. Design

In determining the design, the aim was to handle two major requirements: (i) the parallelization of the model, and (ii) the scalability of the provided solution.
The parallelization of the model was achieved through a map-reduce derived algorithm. However, a limitation in Excel prevented user-defined types from being communicated between HPC nodes. In VBA, a user-defined type is a data structure that is defined by the keyword `Type`, and contains a combination of built-in types. Since the model contained more than 50 user-define types (Table I), the solution was to serialize them into built-in types before sending to the cluster nodes, and to deserialize them back to the original types on receiving.
The second requirement considered in the design was the

TABLE I: VBA Code Metrics extracted from the spreadsheet model

VBA Code Analysis								
Size	# of modules	23	**Complexity**	Cyclomatic Complexity for Methods	Max=386, Average=14.36	**External Dependency**	# of projects invoked	2
	# of methods	146		Cyclomatic Complexity for Types	Max=546, Average=69.3		# of methods invoked	65
	# of types	58		Methods too complex (Cyclomatic Complexity 20)	17 (11.6% coverage)			
	lines of code (total)	7,372		Methods too big (LoC = 30)	53 (36.3% coverage)		# types used	14
	comments coverage	12.79% (1081 LoC)		Types with too many fields (fields 20 and not enumerated)	5			

scalability. The insurance company's aim was to evaluate as large as 30K scenarios in the future, and this would bring up two issues: a high memory utilization, and an increased time in splitting the input data. Consequently, the original model was redesigned, and that included some code refactoring, such as minimizing the creation of temporary arrays inside the loops.

B. Analysis

In the analysis phase, the spreadsheet model was analyzed to determine what methods to parallelize and how. The analysis consisted of three operations:

a) Performance profiling: The profiling was done by instrumentation, where a time-logging library was called on each VBA method's entrance and exit. Three methods were the most time-consuming, one of them was the `main()` method.

b) Complexity evaluation: Through standard tools, the complexity and size of the VBA project were measured. The `main()` method was the largest and most complex.

c) Dependency analysis: This was done by studying the input and output data flows of the methods, with the aim of avoiding parallelizing a method with high level of data coupling. The original VBA code was written in a way that introduces a considerable amount of data coupling between the methods, where the `main()` method exhibited the most dependency on other methods.

As a direct outcome of the analysis, various loops in a number of VBA methods were candidates for parallelization. Of which, the `main()` method was the most feasible since it contained the most time-consuming code. Important to note here, the analysis process was difficult, since the software engineers lacked the needed domain knowledge of the model. As such, information were regularly required from the model owners at the insurance company (discussed further in the Challenges, section VI.A).

C. Implementation

The implementation comprised rewriting parts of the VBA code, and adding other parts (i) to apply parallelization and (ii) to deliver a scalable solution.

The parallelization targeted various loops in the `main` method, and the focus was on splitting the arrays used in these loops' iterations. The mapping for an array was done by dividing the total number of scenarios by the number of cluster cores, consequently creating split copies of that arrays. On the other hand, the reduce process was performed based on each function's requirements: some arrays were averaged or summed, others were concatenated. To ensure successful parallelization, dependencies between the methods were minimized. In this manner we refactored, when applicable, code parts that include passing parameters by reference, or modifying the passed parameters inside a function's body.

Because of the parallel approach, serialization and deserialization code was developed to allow the communication of each user-defined type in the model. This is due to a limitation in Excel, which forbade the proper transfer of user-defined types between the cluster nodes (described in section IV.A).

Finally, to ensure scalability both in terms of the number of scenarios and the number of cores, re-design of the model in some loops was implemented. For instance the creation of temporary arrays inside the loops was not allowed. In addition, the array splitting and copying were moved to the cluster instead of the client for some loops, and some data structures were made smaller through using data types with smaller sizes.

D. Evaluation

As a result of the parallelization approach and the modifications to the model, the run times showed clear improvements. In this section we present (i) the performance figures of the various runs of the model, (ii) the verification of the model's health throughout the migration process.

a) Performance Results: There were two factors that directly affected the performance: the number of scenarios in the model and the number of cores utilized. First, Figure 3 shows the effect of increasing the scenarios on the model's performance, while the number of cores was fixed: a single core for the sequential original model, versus a 24-core cluster for the parallel modified model. When the original model was considered, the runtime increased exponentially as we increased the number of scenarios. In the meanwhile, the modified model runtime increased as well, but in an almost linear manner. In Figure 3-b, we see that the modified model in parallel performed 39 and 52 times faster than the original model, for 480 and 960 scenarios respectively. To run 960 scenarios for instance, the original model required 729 minutes (more than 12 hours), while the modified model completed the run in 14 minutes on the 24-core cluster. Figure 4 illustrates the effect of increasing the number of cores on performance, while the number of scenarios is fixed. For 1008 scenarios, the original model needed more than 15 hours to complete. For the parallel runs, started by 8-cores and increased gradually up-to 48 cores, the runtime goes down almost linearly, from 105 minutes to 33 minutes respectively.

b) Model Accuracy: The accuracy of the obtained results was verified throughout the migration of the model. The verifications were carried out by the insurance company's team, who manually compared the outcomes of the calculations between the original and the modified models.

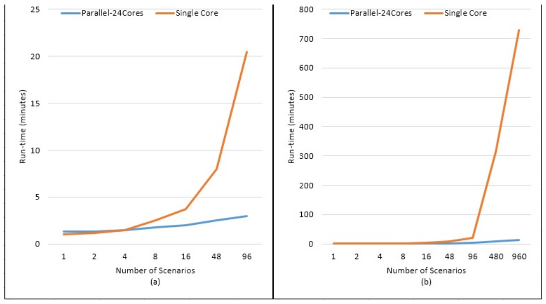

Fig. 3: Results: Effect of increasing the number of scenarios on the performance of the original model and the modified model. (a) On a small scale: up to 96 scenarios. (b) On a larger scale: up to 960 scenarios

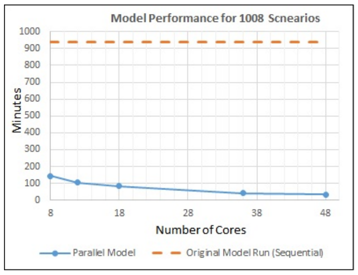

Fig. 4: Results: Effect of increasing the number of cores on the performance of the parallel model

In one case for example, the results were slightly different, and an investigation showed that a conditional statement was not migrated correctly. However, the outcome of the modified model was more accurate than before, due to running more simulations, which is expected in a Monte Carlo simulation.

c) Migration Cost: According to the project plan sheet, the migration was finalized in 431 working hours. The recorded work included development of the new model and changes to the HEAT, testing and verification, bug fixing, and writing documentation. 93 hours of the recorded hours were related to providing support for the insurance company's employees, helping them through the running of the model.

E. Maintainability of the Modified Model

The insurance company wanted to ensure business continuity. They required that the modified model is easily changed and maintained by their end-users, without continuously referring to the engineers in BitBrains. To achieve this, the HEAT provides an option for the end-users to run the model sequentially on a local machine. This is important to test the introduced changes, before the model is run on the cluster. Furthermore, the engineering team performed several hand-over sessions, and provided thorough documentation materials, that detailed how to modify the core parallel part of the code.

V. Related Work

Related to our study is the work of Pichitlamken *et al.* [11], where they provided a prototype that is similar to the one we presented. The prototype, presented as an Excel add-in, offloads the calculations of a spreadsheet model to a computer grid. However, the two solutions still differ. The solution in [11] built its own resource management and scheduling unit, based on FCFS algorithm. In our solution, HEAT exploits the well-established Microsoft HPC services for these functionalities. Additionally, the solution they provided as a prototype, was not validated on a large-scale spreadsheet, or in an industrial setup. In another related work, Abramson *et al.* in [12] presented ActiveSheets, a solution designed to evaluate the parallel user-defined functions of a spreadsheet in the background. ActiveSheets is different to our solution by targeting independent user-defined functions, and the lack of an industrial application. Another related approach is ExcelGrid [13], in which a middle ware tool was designed and built to connect an Excel spreadsheet with either a private or a public cluster. Even though ExcelGrid applies a parallel solution to distribute work on a cluster, it does not target spreadsheet models built completely inside Excel, as it uses the spreadsheet as a means for providing input data to another software. To the best of our knowledge, our paper presents the first industrial and large-scale case, involving the parallelization of a spreadsheet model.

VI. Lessons Learned

In the above, we described the steps used by BitBrains and GridDynamics, to improve the performance of a spreadsheet-based model. This was done by creating a design based on a software framework, that allows Excel spreadsheets to run in parallel on an HPC cluster. Subsequently, the model was analyzed by the software developers to identify candidate methods for parallelization. Following that, the solution was implemented by modifying parts of the the methods, and adding other code parts to ensure a successful integration between the spreadsheet and the HPC cluster. In this section, we reflect on the project, identifying challenges that arose along the way to the final implementation. Subsequently, we introduce possible opportunities that can be addressed in future work, to minimize the efforts in similar spreadsheets projects.

A. Challenges

Throughout the phases of the solution, a number of issues proved to be challenging. We foresee that these challenges will also be faced in future projects that optimize other spreadsheet models. Therefore, the followings are our insights into some of the challenging issues.

a) Reverse Engineering and Restructuring: The analysis of the spreadsheet model was a difficult process. Four weeks were needed to reverse engineer the data flows and the algorithm of the original spreadsheet model. The difficulty originated from the fact that the developers lacked financial domain knowledge, which required holding several meetings with the model owners. These meetings hampered a speedy development process. On the other hand, the implementation phase involved applying several refactorings to the VBA code, as well as rewriting and adding some components to the HEAT framework, for the sake of fitting all the parts together. These recursive and continuous processes of reverse engineering, restructuring and integration development were vital to prepare the spreadsheet model for the parallel execution, without any unwanted result. These preparations however were done manually, neither planned nor systematic, which was time consuming.

b) Preserving the Model's Functionality: The transformation process is risky; each modification applied to the original model compromises the accuracy of the output results, thus forcing continuous validations throughout the process. These validations, in our case, were performed manually and remotely by the insurance company's team. With the lack of domain knowledge added, It was difficult for the developers in some cases to track the root causes of the variances, causing some delays to the implementation process.

c) Scale of Improvement: When to consider the performance improvement *enough* in a similar project? This question was not answered in our case. The computational power was increased gradually to see the reflection on performance. However, we believe that a turnover point exists, at which the model's run may take longer time. This would happen since the increase of the computation nodes means the increase of the number of intermediate results to be reduced. Depending on the size of the data structures used, the intermediate data may grow rapidly to utilize more memory, increasing the processing time. Therefore, the computing resources that would achieve the optimal performance is not previously defined, and is challenging to measure.

B. Opportunities

In this section we present areas for researchers to investigate in future. The aim is to provide tools and solutions to issues that could hinder the implementation of similar spreadsheets projects. We identify the following opportunities, categorized per phase of the solution:

1) Design: The solution was primarily based on parallelization of the spreadsheet model, by offloading the core computational parts to the underlying cluster. An improvement to the design is possible by accompanying the core parallel solution with a concurrent approach. For example, we envision a new design that includes a step related to detect and refactor spreadsheet smells that may cause performance issues, such as long calculation chains [14].

2) Analysis: In analyzing a spreadsheet model, software engineers could benefit from having tools in the following areas:

a) Dependency Recognition: To analyze the dependency between the VBA methods, the developers carried manual checks to the data flows and algorithms of the model. A systematic approach, through a tool for example, is needed to identify dependencies inside a spreadsheet model. This would allow the developers and spreadsheet users to decide possible parallel sections in their spreadsheet easily, and more quickly.

b) Performance Profiling and Hot-spot Analysis: In the same manner to the dependency checks, profiling of the VBA methods was done manually by writing code to log the run times, and upon that deciding the major functions. A spreadsheet model consists of different parts (both formulas and VBA code), that can consume time and resources. The manual process is error-prone and time consuming, thus a tool that detects code runtime , and links it directly to the associated formula or VBA method, would produce more assured results.

3) Implementation: The development cycles in the solution involve repetitive work that has room for improvement, in the form of automation or semi-automation of some processes. The developers identified a predefined set of code behaviors that lead to heavy dependencies between functions, such as the passing parameters by reference, and the modification of parameters inside the function body. A tool can be developed to detect and refactor the code to eventually remove, or reduce, the dependencies.

Finally, the solution presented is fully customized towards a specific spreadsheet model. This suggests that for each new spreadsheet model (new in terms of the simulation type, or the used data structures and algorithms) the same procedure shall be followed again. An opportunity to improve would be widening the targeted models, by specializing in a specific category of financial simulations, such as Monte Carlo. The solution would automatically analyze any model that fits the simulation criteria, based on a predefined set of rules and steps, reducing the amount of repetitive work in future projects.

REFERENCES

[1] R. Baxter, "Enterprise spreadsheet management: A necessary good," in *Proceedings of the EuSpRIG 2007 Symposium*, 2007, p. 7. [Online]. Available: http://arxiv.org/abs/0908.1584

[2] F. F. J. Hermans, *Analyzing and visualizing spreadsheets*. TU Delft, Delft University of Technology, 2013.

[3] G. J. Croll, "The importance and criticality of spreadsheets in the city of london," in *Proceedings of the EuSpRIG 2005 Symposium*, 2005. [Online]. Available: http://arxiv.org/abs/0709.4063

[4] J. Wittwer, "Monte carlo simulation basics," 2004. [Online]. Available: http://www.vertex42.com/ExcelArticles/mc/MonteCarloSimulation.html

[5] A. Orlando and M. Politano, "Pension funds risk analysis: stochastic solvency in a management perspective," *Problems and Perspectives in Management*, vol. Volume 8, no. Issue 3, 2010.

[6] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," Delft University of Technology, Software Engineering Research Group, Tech. Rep., 2014.

[7] Vbdepend.com, "Vbdepend :: Achieve higher vb6/vba code quality," 2015. [Online]. Available: http://www.vbdepend.com/Metrics#MetricsOnMethods

[8] Bitbrains.nl, "De bits en de brains achter uw bedrijfskritische applicaties - bitbrains," 2015. [Online]. Available: http://bitbrains.nl

[9] Griddynamics.com, "Blog — grid dynamics." [Online]. Available: http://www.griddynamics.com/solutions/excel-hpc.html

[10] Technet.microsoft.com, "How hpc services for excel work." [Online]. Available: https://technet.microsoft.com/en-us/library/ff877825%28v=ws.10%29.aspx?f=255&MSPPError=-2147217396#BKMK_soa

[11] J. Pichitlamken, S. Kajkamhaeng, P. Uthayopas, and R. Kaewpuang, "High performance spreadsheet simulation on a desktop grid," *Journal of Simulation*, vol. 5, no. 4, pp. 266–278, 2010.

[12] D. Abramson, P. Roe, L. Kotler, and D. Mather, "Activesheets: Super-computing with spreadsheets," in *2001 High Performance Computing Symposium (HPC01), Advanced Simulation Technologies Conference*. Citeseer, 2001, pp. 22–26.

[13] K. Nadiminti, Y.-F. Chiu, N. Teoh, A. Luther, S. Venugopal, R. Buyya, and B. Street, "Excelgrid: A .net plug-in for outsourcing excel spreadsheet workload to enterprise and global grids," in *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004*, 2004.

[14] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, vol. 20, no. 2, pp. 549–575, 2014.

TUD-SERG-2016-003
ISSN 1872-5392