

ActiveSheets: Super-Computing with Spreadsheets

David Abramson †, Paul Roe §, Lew Kotler ‡ and Dinelli Mather 8

† School of Computer Science and
Software Engineering
Monash University,
CLAYTON, VIC 3168, Australia

‡ Australian Radiation Protection and
Nuclear Safety Agency
Lower Plenty Road
YALLAMBIE VIC 3085, Australia

§ School of Computer Science and Software Engineering
Queensland University of Technology,
Gardens point Campus
Brisbane QLD 4000, Australia

8 School of Business Systems
Monash University,
CLAYTON, VIC 3168, Australia

ABSTRACT

Spreadsheets are important business tools. Increasingly they are being used for simulation e.g. to perform risk analysis. Such tasks have far greater computational demands than traditional spreadsheet bookkeeping applications. In this paper we show how spreadsheets can support supercomputing. This is achieved without requiring the spreadsheet user to have specialist tools or knowledge. The key technical innovation is a mechanism enabling the concurrent evaluation of spreadsheet functions. Furthermore the mechanism does not require modification of the standard spreadsheet evaluation engine.

INTRODUCTION

Spreadsheets are popular because they are easy to use and modify, and they support numerical data analysis without programming. A feature of modern spreadsheets is their capability to be extended; in particular custom functions may be incorporated into them, for example complex simulation functions and computational experiments. For these reasons spreadsheets make an ideal front-end for numeric simulations since they support pre and post processing of simulation data without the need for programming.

Since simulations are often computationally intensive they are good candidates for parallel evaluation. Whilst there have been many attempts to accelerate various aspects of simulation programs through parallelisation, an easy way to expose parallelism is to execute the simulation of several different scenarios concurrently. High performance distributed computers provide an excellent platform for this style of execution, because independent simulations can be run on a number of different machines concurrently. We have significant experience with a custom interface that

supports this functionality with a research tool called Nimrod [2][1] and its commercial counterpart, EnFuzion[20]. These tools allow us to distribute independent executions of a single simulation in order to perform complex scenarios analysis. They manage the generation of work, distribution of the computation and gathering of the results. The output is very often read back into a spreadsheet for analysis and visualisation once it has been returned from the distributed computers. Spreadsheets allow a range of tools to be used to display results, such as statistical analysis and charting utilities. Consequently, the idea of using a spreadsheet to generate the runs, perform the simulations and gather the results within the one application is attractive because it obviates the user from running multiple independent packages. Complex structures can be established easily, for example, the output of one simulation may be processed in the spreadsheet, and then sent as input to another one. However, for this to work in a spreadsheet setting, parallel evaluation of spreadsheets is required. Unfortunately existing spreadsheets, such as Microsoft Excel [14], have in-built sequential calculation mechanisms. Thus, they are unable to evaluate multiple cells concurrently using the standard execution mechanism in the cell calculation engine.

This paper describes how a conventional spreadsheet may be evaluated in parallel without modifying the core execution engine. The parallel evaluation mechanism is encoded using custom functions, and works in conjunction with the standard built in sequential evaluation mechanism of spreadsheets. The paper begins with a discussion on how to perform high performance simulation on distributed computers. We then discuss how an existing sequential evaluation mechanism can be modified to work in parallel, some implementation issues, followed by a few case studies that illustrate the power of such a system.

HIGH PERFORMANCE SIMULATION

Simulation is widely used in many fields. It allows a user to explore a wide design space without the need to build real world models. Simulation techniques vary from continuous simulations, discrete event simulation and mathematical modelling of real world phenomena using systems of equations (such as partial differential equations). Whilst simulations for simplified models can be quite fast, they are often very slow for accurate simulation using realistic problem sizes. It is not surprising, therefore, that High Performance Computing (HPC) has been used extensively in support of simulation.

A number of different methods have been used in the past to accelerate simulation using parallel machines. For the most part, these consist of parallelising the core algorithms used in the simulation. Because the implementation techniques vary so widely, there is no common method of application. For example, simulations that solve partial differential equations usually use a parallel decomposition of the finite difference representation of the problem [13]. This means that sections of the domain are calculated in parallel, and boundary values are transmitted between the processes as required. Simulations that involve ordinary differential equations may require different solution techniques, and these are, accordingly, parallelised differently. On the other hand, parallel discrete event simulation has received a great deal of attention over the years [21][10][7][19][12][11][18]. Since it is inherently a sequential process, it has been hard to achieve good parallel speedup even though the underlying problem being solved may be highly concurrent. The most successful parallel techniques have involved speculative execution of the event list with subsequent back-off operations when the simulation proceeds incorrectly.

Another technique that has been largely ignored is the area of scenario analysis. In this mode, the same simulation model is run repeatedly using different parameters [5]. Such a technique can be parallelised easily, by simply running the different scenarios concurrently. Whilst the technique is simple, it achieves close to perfect speedup with very little effort. This mode of execution is supported by the tools we have developed, namely Nimrod [1][2][3][4] and EnFuzion [20].

When a user describes an experiment to Nimrod and EnFuzion, they develop a *declarative* “plan” file which describes the parameters, their default values, and the commands necessary for performing the work. The system then uses this information to transport the necessary files and schedule the work on the first available machine.

When the user invokes Nimrod on a workstation, the machine becomes known as the “root” machine because it

controls the experiment. When the dispatcher executes code on remote platforms, each of these is known as a computational “node”. Thus a given experiment can be conducted with one root and multiple nodes, each of a different architecture if required.

However, both Nimrod and EnFuzion expect the user to specify the scenarios they wish to explore using a dedicated user interface. Further, they rely on the user to produce programs that aggregate the results, and if these are to be processed by a spreadsheet tool, they must be imported manually. ActiveSheets is a system that automates this by combining all the functions of Nimrod into the spreadsheet tool itself. Importantly, we rely on an API for EnFuzion that allows it to provide the backend computation platform, whilst the frontend is provided by a commercial spreadsheet tool, in this case, Microsoft Excel.

PARALLEL EVALUATION OF SPREADSHEETS

The mechanism for the parallel evaluation of spreadsheets is based on the dataflow model of computation [6][9]. An expression such as $(1+2)*(3+4)$ may be naturally represented as a graph as shown in Figure 1.

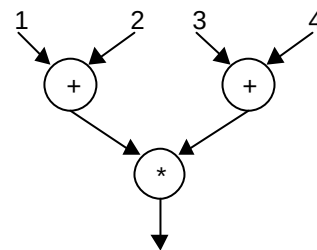


Figure 1 – Dataflow graph exhibiting parallelism

Nodes represent operators; once an operator has all arguments it may be evaluated generating a result which is output. Thus evaluation may be viewed as data flowing along arcs between operators. Operators may evaluate in parallel subject only to arguments being available. This expression can be represented in a spreadsheet as illustrated in Figure 2.

In ordinary spreadsheets cells are always evaluated sequentially. However, if the values in row 1 change we require that cells A2 and C2 are evaluated in parallel, just as the dataflow graph in Figure 1 supports parallel evaluation. Using custom functions, rather than built-in ones like “+”, a spreadsheet may be made to behave in this way even though its built evaluation mechanism is sequential.

	A	B	C	D
1	1	2	3	4
2	=A1+B1		=C1+D1	
3	=A2+C2			

Figure 2 – Spreadsheet representation of Figure 1.

The goal is to evaluate separate functionally independent cells in a spreadsheet in parallel. In particular the goal is to evaluate in parallel the custom functions (e.g. representing a simulation) in such cells. The key to evaluating an otherwise sequential spreadsheet in parallel is to adopt a two stage evaluation process for custom functions. Note that the scheme we are proposing will not work with built-in functions in parallel; however, this is not a serious drawback since most of the time this would not be desirable. In order to trick the otherwise sequential execution mechanism, a custom function sends its arguments together with a representation of the function to a backend computer for parallel evaluation. A table is used to store the current state of the cell, which can be one of the following states: unevaluated, under evaluation or evaluated. The backend may distribute the calculation to any one of a number of machines. The custom function in the spreadsheet returns before evaluation has completed. It returns a *distinguished* error or *undefined* value, which will prevent the cell from being interpreted as a valid value. *By returning immediately without waiting for the custom function's result other custom functions may be evaluated, giving rise to parallel evaluation.* The rest of the spreadsheet is evaluated in the same way driven by the standard built in evaluation¹ mechanism. The parallel evaluation of the spreadsheet is automatic subject only to functional dependencies between spreadsheet cells.

Once results have been calculated on the backend they are sent to the spreadsheet. The results are collected for insertion into the spreadsheet. However the results are not inserted into the spreadsheet directly, rather they are stored in a table so that subsequent re-evaluation of the destination cells will discover them. Re-evaluation of the spreadsheet is forced programmatically. The “second time around” a result is found in the table representing the true value of the function; this is then returned as the function's result.

IMPLEMENTATION ISSUES

The implementation comprises an add-in for Microsoft Excel (version 97 and 2000). Excel is highly customisable through

¹ The terms evaluation and calculation are used interchangeably

OLE automation; this is used to implement the two phase evaluation strategy which gives rise to parallel evaluation.

Custom functions are implemented in Visual Basic for Applications (VBA) as stubs [15]. These stubs marshal arguments and function identifiers to an ActiveX DLL (COM component). For a given function evaluation, the ActiveX component either sends a request to the back-end machine to evaluate the function or extracts the function's result from a table. In the latter case the result will have been placed there by the backend machine, in response to the first call for the function's evaluation. The key to the implementation is the table that stores the state of custom function evaluation requests. Effectively it serves as a huge cache. Currently the user decides on which custom functions are to be evaluated in parallel on the backend machine.

The implementation supports both inter-cell parallelism and intra-cell parallelism. For example consider a formula such as “=foo(A1)+bar(B1)” where “bar” and “foo” are custom functions, the system will support the parallel evaluation of “foo(A1)” and “bar(B1)”.

All backend aspects of the system have been factored into an ActiveX control known as the “adaptor”. This enables the system to utilise different backends by simply selecting different adapters. Currently we use a manual builder tool to build the stub code, however this is dependent on the backend. Where metadata is available, the stub code can be automatically generated.

An important goal, which has been realised, is that the system coexists with Excel without requiring any modifications or restrictions to its use. The system may be incorporated into an existing spreadsheet without requiring the spreadsheet to be rewritten – all that is required is for the parallel functions to be available in the backend machine.

CASE STUDIES

In this section we examine two case studies. These are problems that have been solved previously using a combination of EnFuzion and spreadsheets. Whilst we have not re-run the experiments using ActiveSheets, the studies highlight the power of using a single spreadsheet tool for all of the tasks.

Ionisation Chamber Simulation

The Australian Radiation Protection and Nuclear Safety Agency (ARPANSA) provides the Primary Australian Standards for certain Radiological quantities such as Air Kerma, which essentially quantifies photon energy deposition in terms of ionization in air. The calibration of medical equipment used for diagnosis and/or treatment

against these standards ensures that in any location the radiation doses received by patients is consistent with that received in other locations in Australia and overseas.

An ionization chamber essentially isolates a certain volume of air and measures the ionization within that volume. However, the physical process of isolating a volume of air modifies the original photon and electron spectrum entering the volume. Thus, if the chamber is to act as a primary standard for calibration purposes, it is necessary to correct the measured ionization for the spectral perturbations due to the physical presence of the chamber.

The nature of the problem is such that not all correction factors can be determined experimentally. Within the field of radiation dosimetry, Monte-Carlo simulation programs, such as the EGS4 [16], have become an acceptable method of simulating the response of ionization chambers to photons and electrons and thus an alternative means of determining the appropriate correction factors. The absolute accuracy of the EGS4 package for simulating ion chambers is quoted at about 1% but relative accuracy is actually better than this.

One important correction factor for the primary standard, in the standardization of photons emitted by a cobalt-60 teletherapy source, is the extent of photon attenuation and scattering in the front wall of the chamber. Detailed measurements of chamber response as a function of front wall thickness have been performed and a quantitative model of interactions within the front wall used to determine the appropriate correction factors. These detailed measurements provide an interesting basis for comparison with Monte-Carlo, as not only the chamber but also the cobalt-60 photon spectrum are to be simulated. The calculations reported here concern the simulation of the chamber response as a function of front wall thickness. Accordingly, we need to run the model many times with different random number seeds and different front wall thicknesses.

Figure 3 shows the spreadsheet required to perform this calculation, including a chart showing the results. It is not possible to fully illustrate the complexity of this spreadsheet in the paper, however, it occupied 7 work sheets and contained many complex calculations and MS macros. Prior to the development of ActiveSheets, it was necessary for the results to be computed separately using EnFuzion, and for these to be transferred into the spreadsheet. With ActiveSheets, this could all be performed in the one spreadsheet, reducing the scope for error considerably.

Simulating Injecting Drug Users

Another project being conducted by the School of Business Systems at Monash University concerns the simulation of

injecting drug users. Injecting drug use facilitates the transmission of numerous blood borne viruses such as The Human Immuno-deficiency Virus (HIV) and Hepatitis C Virus (HCV). Whilst these viruses are prevalent within the injecting drug using community, the process of transmission of the viruses from an infected individual to a susceptible individual is determined by various behavioural factors of the users as well as clinical factors of the viruses. From a public health perspective, intervention policies need to be directed at changing the injecting behaviour that facilitates this transmission. Given the inability to carry out controlled experiments in this field as well as the need to deal with a large number of parameters a simulation model makes a very effective analysis tool.

Considering the spread of HCV among injecting drug users (IDUs), some of the parameters in question are the prevalence of the virus, the infectivity of the virus, the frequency of injecting as well as the dynamics of the injecting process. A micro simulation model that enables researchers to create various cohorts of IDUs with differing injecting practices and prevalence levels and then experiment with various intervention policies is extremely useful. Given that spreadsheets are universally used across most disciplines, providing a spreadsheet interface to such a simulation program makes the model easier to use as well as more transparent to the user.

The results shown in Figure 4 were generated from a series of simulation designed to assess the impact of interventions that would result in changes to various parameters. They were gathered from the EnFuzion runs and then placed in a spreadsheet for charting and display.

Again, like the previous study, the work was originally performed using a conventional spreadsheet model and the runs were conducted using EnFuzion. However, ActiveSheets would have allowed us to combine these into one spreadsheet that computed and charted the results.

These two case studies illustrate that it is natural to consider the direct execution of a simulation from a spread sheet, since quite complex spread sheets have been created to specify the scenarios under consideration. The architecture that has been developed would allow these complex experiments to be created without complex external data manipulation, or for the need to use multiple tools.

RELATED WORK

There are a few commercial products for performing parallel Monte Carlo simulation using spreadsheets. However these are restricted to Monte Carlo simulation applications constructed using the supplied modeling functions. Palisade

has a product called @RISKAccelerator™; this supports the parallel evaluation of @RISK simulations [17]. Decisioneering has a similar product called Crystal Ball Turbo™ which parallelises their Monte Carlo simulation addin for Excel (Crystal Ball) [8]. However, neither system supports the general parallel evaluation of spreadsheet functions, as does ActiveSheets.

CONCLUSIONS

This paper has shown through some case studies that super computing using spreadsheets is highly desirable. To this end a mechanism for parallelising traditional spreadsheets has been defined. The mechanism is based on the idea of dataflow and is particularly useful in that it does not require modification to spreadsheet applications. The parallel spreadsheet tool interfaces with the popular parallel programming tools Nimrod and Enfuzion.

ACKNOWLEDGMENTS

This work has been sponsored by the Co-operative Research Centre for Distributed Systems Technology (DSTC), under the Australian Government CRC program. We wish to thanks Jon Giddy and Toby Chittenden for their contribution to the development of Active Sheets and Nimrod.

REFERENCES

- [1] Abramson , D., Sosic , R., Giddy , J., Cope , M. "The Laboratory Bench: Distributed Computing for Parametised Simulations", 1994 Parallel Computing and Transputers Conference, Wollongong, Nov 94, pp 17 - 27.
- [2] Abramson D., Sosic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [3] Abramson, D. "Parametric Modelling: Killer Apps for Linux Clusters", The Linux Journal, #73, May 2000, pp 84 – 91.
- [4] Abramson, D., Giddy, J. and Kotler, L. "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), pp 520- 528, Cancun, Mexico, May 2000.
- [5] Abramson, D., Power, K., and Sosic, R., "Simulating Computer Networks using Clusters of PCs", HPC-TelePar'99 at the 1999 Advanced Simulation Technologies Conference (ASTC'99), April 11-15, San Diego, California, USA.
- [6] Arvind, Bic, L. and Ungerer, T. "Evolution of Data-flow Computers", Chapter 1, "Advanced Topics in Dataflow Computing", Prentice Hall, 1991.
- [7] B.J. Overeinder; L.O. Hertzberger and P.M.A. Sloot, "Parallel Discrete Event Simulation", In W.J. Withagen, editor, The Third Workshop Computersystems, Faculty of Electrical Engineering, Eindhoven University, pp. 19-30. Eindhoven, The Netherlands, May 1991
- [8] Decisioneering Inc. <http://www.decisioneering.com/>
- [9] Dennis, J. "The Evolution of "Static" Data-flow Architectures", Chapter 2, "Advanced Topics in Dataflow Computing", Prentice Hall, 1991.
- [10] Ferscha Alois, Tripathi Satish, "Parallel and Distributed Simulation of Discrete Event Systems", TCS-TR-3336, University of Maryland, August 1994.
- [11] H. Rajaei and R. Ayani, Design Issues in Parallel Simulation Languages, IEEE Design & Test of Computers, Dec. 1993, pp. 52-63.
- [12] H.T. Mouftah, Distributed Discrete Event Simulation for Communication Networks, IEEE Journal on Selected Areas in Communication, Vol. 8, No. 9, pp.1723-1734, 1990.
- [13] K. Motegi and S. Watanabe. A Parallel Scheduling of Multi-Step Diakoptics for Three Dimensional Finite Difference Method. IEICE Trans. Fundamentals, Vol.E76-A, No.10, pages 1822 - 1829, October 1993.
- [14] Microsoft Corporation, Excel User's Guide.
- [15] Microsoft Corporation, Visual Basic for Applications.
- [16] Nelson, W.R., Hirayama H. and Rogers, D.W.O. "The EGS4 Code System", Stanford Linear Accelerator Center Report SLAC-265 (1985)
- [17] Palisade Corporation, <http://www.palisade.com/>
- [18] Paul Wonnacott and David Bruce, "Comparing the performance of a prototype implementation of the APOSTLE discrete event simulation language for shared and distributed memory multiprocessors", pp. 30-36 in Proc. Object-Oriented Simulation 1998, San Diego, California, 11-14 January 1998.
- [19] Richard Fujimoto, Parallel Discrete Event Simulation, Communications of the ACM, October 1990.
- [20] Turbo Linux Inc, EnFuzion, <http://services.turbolinux.com/shop/prod.php?id=enf>
- [21] Yi-Bing Lin, Bellcore Paul A. Fishwick, , "Asynchronous Parallel Discrete Event Simulation" CS-TR-95-005, University of Florida, 1999

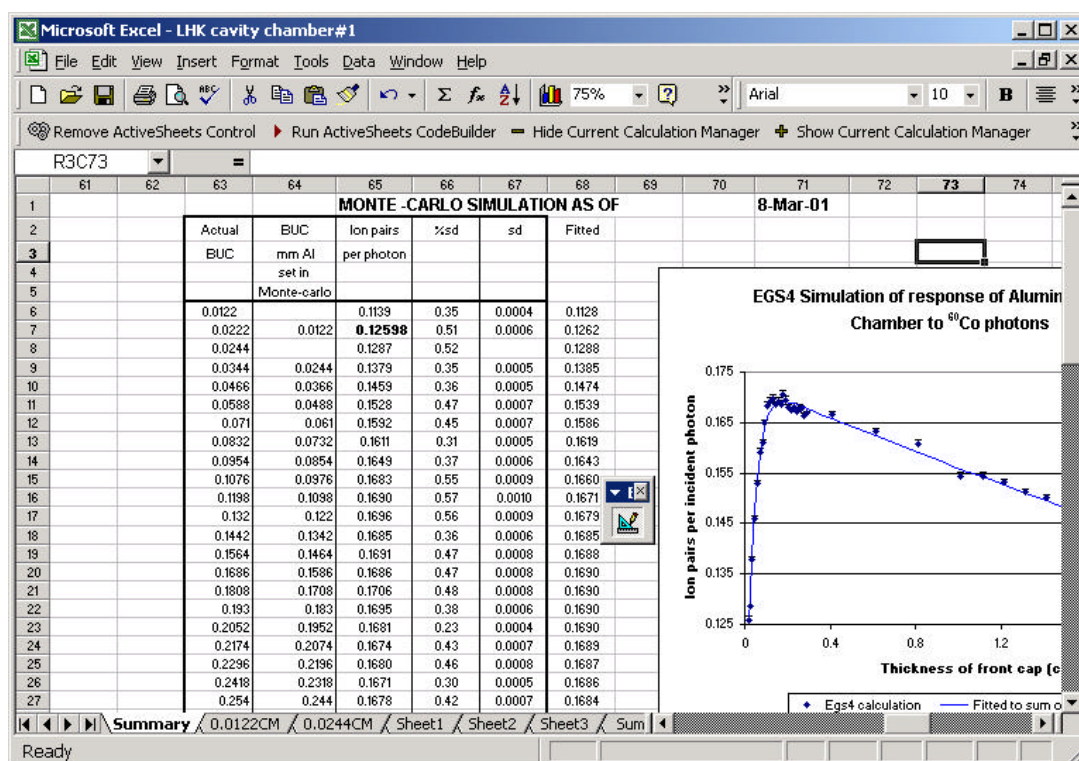


Figure 3 – Spreadsheet for ionisation chamber experiments

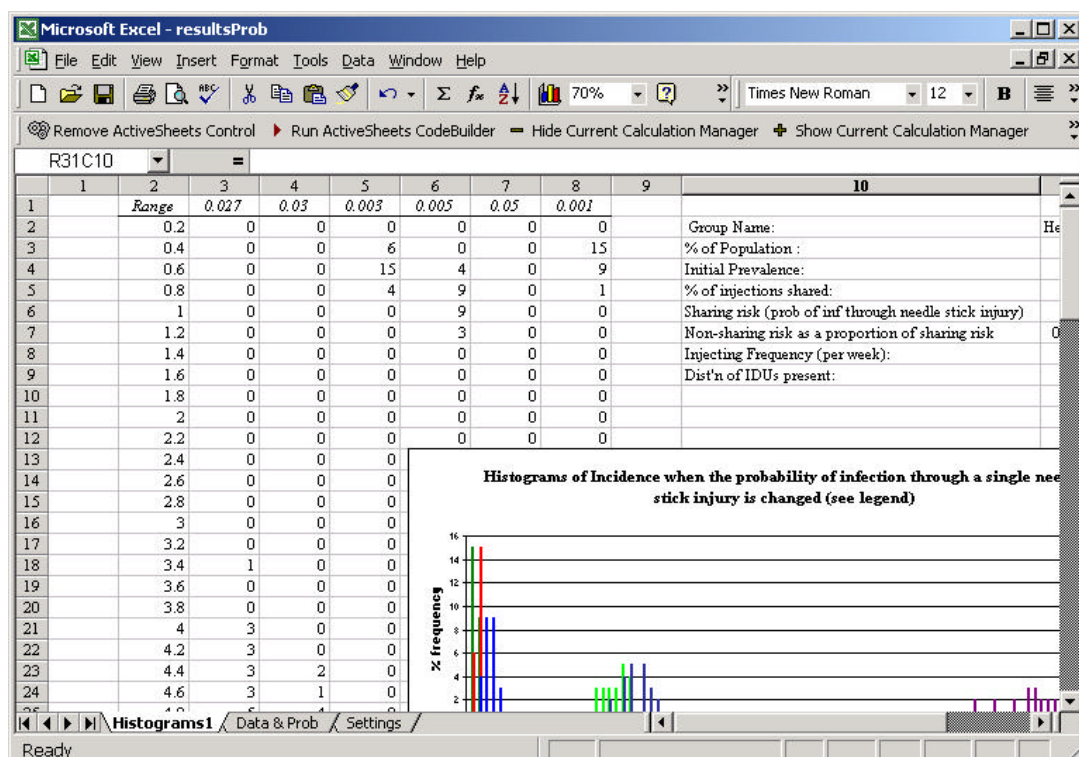


Figure 4- Spreadsheet for drug injecting simulation