Verification and Uncertainty Analysis of Fuel Codes Using Distributed Computing

By
D. Evens and R. Rock[*]

Abstract

Of late, nuclear safety analysis computer codes have been held to increasingly high standards of quality assurance. As well, best estimate with uncertainty analysis is taking a more prominent role, displacing to some extent the idea of a limit consequence analysis. In turn, these activities have placed ever-increasing burdens on available computing resources.

A recent project at Ontario Hydro has been the development of the capability of using the workstations on our Windows NT LAN as a distributed batch queue. The application developed is called SheepDog. This paper reports on the challenges and opportunities met in this project, as well as the experience gained in applying this method to verification and uncertainty analysis of fuel codes.

SheepDog has been applied to performing uncertainty analysis, in a basically CSAU like method, of fuel behaviour during postulated accident scenarios at a nuclear power station. For each scenario, several hundred cases were selected according to a Latin Hypercube scheme, and used to construct a response surface surrogate for the codes.

Residual disparities between code predictions and response surfaces led to the suspicion that there were discontinuities in the predictions of the analysis codes. This led to the development of "stress testing" procedures. This refers to two procedures:
- coarsely scanning through several input parameters in combination, and
- finely scanning individual input parameters.

For either procedure, the number of code runs required is several hundred. In order to be able to perform stress testing in a reasonable time, SheepDog was applied. The results are examined for such considerations as continuity, smoothness, and physical reasonableness of trends and interactions. In several cases, this analysis uncovered previously unknown errors in analysis codes, and allowed pinpointing the part of the codes that needed to be modified.

The challenges involved include the following.
- The usual choices of development language and environment had to be made.
- Significant learning curve for building Windows NT service programs.
- Activity by the distributed jobs was not permitted to interfere with the activity of the local workstation user.

---

[*] Currently: Sententia Research Inc.

- The workstation component of the system must be exceptionally robust.
- Codes to be run must be ported to Windows NT.

Some of the opportunities involved include the following.
- The amount of computing power is quite large.
- The marginal cost of utilising this computing power is very small since it makes use of unused cycles on existing hardware.
- The system is easily scalable.
- The system is easily customisable.

Results of the verification efforts will be discussed in another paper.

The Problem

Recently a project at Ontario Power Generation (then Ontario Hydro) involved the creation of a response surface to perform uncertainty analysis (UA) of fuel behaviour in an essentially CSAU-like (Reference 1) method. The procedure for building a response surface involved running the fuel code 300 times, varying the input parameters in a carefully chosen manner, then fitting an algebraic function to the results. The computing platform used for this project was an IBM RS6000 system with eight nodes. Running 300 cases sequentially on a single node required several hours.

It was noticed that, while the response surface could be made to fit the trend exceptionally well, and to closely fit the details of the results for the bulk of test cases, there were residual disparities between the code predicted values and the response surface. Since there were a total of seven parameters involved in the UA work, it was difficult to decide what was the source of these disparities. For example, it could have been that it was simply necessary to work a bit harder at constructing the response surface. Or it could have been that there was some kind of discontinuity in the code, and that no amount of effort would make a simple algebraic function response surface a good fit.

In an effort to explore this, each of the seven parameters was densely scanned. The range to scan was chosen to correspond to roughly the two sigma band of that parameter. This was chosen because UA efforts would probe a code throughout this range. A code must produce acceptable results throughout this range for methods such as CSAU to work acceptably. Each range was divided into 1000 cases. The intervals between cases were chosen to be equally probable, according to the expected probability distribution for the parameter. This was done because we already had a case generator program that produced Latin Hypercube type data sets.

At this point, a problem arose. Even though each case took only a few minutes, running several thousands of cases would produce a significant burden on the already highly loaded RS6000 system. This load would be significant both from a CPU standpoint, and from the standpoint of other resources such as disk space. Retaining the output files from several thousands of code runs would have been prohibitive. As a result, such scanning

was slow, difficult, and unpopular with the system manager and other users on the RS6000.

It was determined to perform exploratory tests with converting the analysis code to run on desktop workstation running Windows NT. As others have found (Reference 2) modern desktop workstations provide significant computing power. It was found that the typical workstation produced real time throughput of analysis cases better than those of the RS6000 system. The conclusion was that this was primarily due to the very large load on the RS6000. However, even in comparing throughput on the basis of CPU second per CPU second, a group of three or four workstations easily out produced a single RS6000 node.

This meant that the analysis could be performed on workstations. However, it was extremely inconvenient to use many desktop machines to perform such tasks. The input sets needed to be divided up and distributed to the various machines. The workstations needed to be monitored to assure their progress was adequate and correct. The results had to be collected and assembled. This process was error prone due to the numerous manual steps. It was difficult to verify this process. Running the analysis cases interfered with the normal work that workstation users wished to perform. These considerations initially prevented desktop workstations from being used for any significant amount of analysis work of this kind. It was simply too difficult.

The Solution

An effort was made to automate the process of using several desktop workstations in this manner. The application which resulted from this effort is called SheepDog. It is a scheduler utility that allows the workstations on a LAN to act as a distributed batch queue. Essentially, SheepDog is the very simplest possible of distributed parallel computing machines.

SheepDog is currently in the prototype stage and is being developed for more general use.

The SheepDog architecture is pictured in Figure 1. The application has two major parts. The first is the part installed on each workstation, and is called BQDog. Workstations on which BQDog is installed are called dogs. The second part of SheepDog is the console, or user interface, and is called BQShep. Figure 2 shows a typical screen from the BQShep portion of the application.

SheepDog is essentially a client server application. It is unusual in that there is one client, the Shep portion, and many servers, the Dog portions.

The single largest challenge to developing SheepDog was arranging that it be automatically available. This was accomplished by making BQDog a WindowsNT service program. Services are programs that WindowsNT can be directed to start automatically at time of power on, and which run whether or not any user is logged into the workstation. Services are commonly used on WindowsNT to provide a variety of

functions. For example, virus scanning is commonly performed by installing a service on each workstation. Thus SheepDog could be made available provided only that each workstation was powered on.

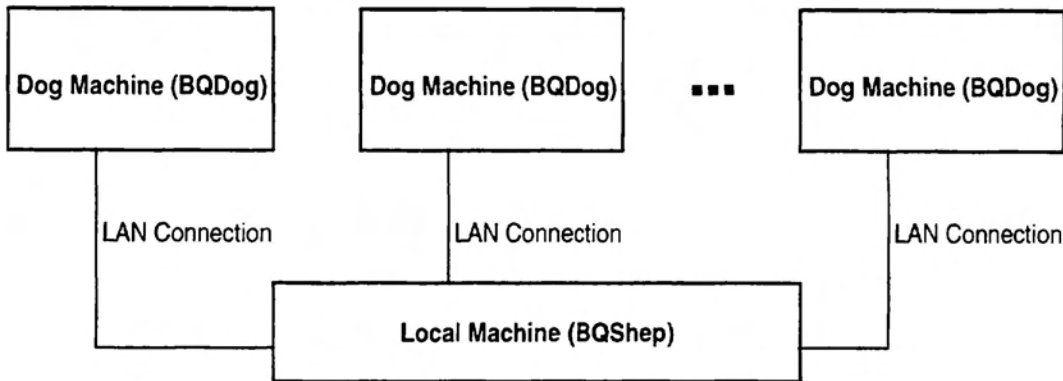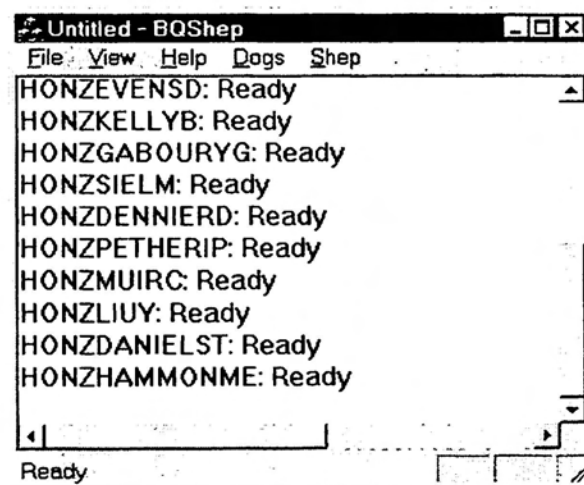## Figure 1 SheepDog Connectivity Model

Figure 2 BQShep Screen Capture

One of the overriding requirements of the SheepDog project has been that no interference with normal workstation work is permitted. Neither SheepDog itself nor the jobs run through it are permitted to hamper the normal use of workstations. This has meant several aspects of application development have received special emphasis.

- First, the program could not be allowed to leak resources. In other words, the BQDog program must properly return to the operating system any resources such as memory, file handles, process handles, etc. Since it is expected that BQDog will run on a workstation, without monitoring and without adjustment, for extended times, any such leak would eventually cause the workstation to crash. BQDog has been installed

for more than a year on ten workstations in the department, and has caused no problems.

- Second, the jobs run through SheepDog had to be executed at low priority. WindowsNT has several levels of priority, the user by default operating at a middle priority. Experience has shown that users are typically unaware that low priority jobs have been run through SheepDog, with two minor exceptions. First, they may notice their hard disk light shows activity when they are not using their machines. And second, they may notice their screen saver slows down. This is because screen savers are typically run at the same low priority as are SheepDog jobs. As a result it is typically requested of workstation users where SheepDog is installed that they set their screen savers to a mode which does not use significant CPU.

- Third, jobs run through SheepDog must not use excessive amounts of any resource. For example, programs which require large amounts of RAM, or many files to be open at the same time, or that are excessively burdensome in terms of disk use, must either not be run through SheepDog, or must be run outside of regular office hours. SheepDog jobs must also be careful to clean up after themselves.

SheepDog is, in its present form at least, tied to WindowsNT. In fact, it is not even possible to run it on any other version of windows, let alone other hardware platforms. As a result, in order to run a program through SheepDog it is necessary to port the program to run on WindowsNT. There are excellent development tools for WindowsNT. For example, there are Visual Fortran and Visual C++, which were used in the SheepDog project.

Potential Uses

The potential value of applying SheepDog is large. For example, a 266 MHz Pentium II machine is capable of maintaining about 70 million floating point operations per second (MegaFLOPS). With machines of this type, it would require approximately twenty to thirty workstations to equal the power of a typical super computer.

At present, the typical workstation is totally idle for about two thirds of each day, and significantly under used even during the heaviest of workstation use. Thus this potential computing power is, at present, nearly completely unused.

The marginal costs of using this computing power is very small. The hardware is already present, and its maintenance is already budgeted. The largest part of the marginal cost is likely to be the time and effort involved in porting analysis code to WindowsNT.

The configuration of such a system is easy to customise. It is quite possible to arrange that only certain workstations be used, that some machines not be used at some times of the day, etc. It is very easy to add or remove machines from the system. It is easy to upgrade individual machines. Thus it would be quite easy to arrange, for example, that a high priority project was granted exclusive access.

Having such a powerful computing system available makes a variety of computational techniques possible that were until recently either prohibitively expensive or prohibitively

slow. As well, certain tasks can be significantly speeded up, and automated. Here are some examples.

- Running regression tests suites to examine the consequences of changes to analysis codes. Automating such a process could allow a new version to be rapidly tested against a benchmark version.
- Running multi-parameter searches to determine limiting cases. This has the potential to remove unwarranted conservatism, while at the same time concentrating limited resources on the areas where it is possible to most improve safety margin.
- Examining code predictions to determine the optimum operation condition. By doing scans of combinations of parameters it may be possible to extend the life span of components of nuclear stations by reducing damaging conditions. For example, it might be possible to scan for combinations of parameters to reduce vibration during normal operation.
- Running model building cases. It is often desirable to scan certain modelling parameters, often in combination, to find best fits to data. This can require large amounts of computer time.
- Running verification and stability test cases.
- Performing uncertainty analysis, particularly in the manner of the CSAU method or the Bootstrap method (Reference 3).

Thus, SheepDog has the capability to decrease costs at the same time as increasing safety margin.

An Example of Use

As an example of one application of SheepDog, some verification and stability tests are discussed in the following. This method is generically referred to as "stress testing." Generically, this takes two forms.

The first form is to coarsely scan through several parameters in combination, examining the results for reasonableness and physically sensible trends. This work is reported on in another paper at this conference.

The second form of stress testing is motivated by searching for discontinuities in code output. A single input parameter is chosen. In the example presented, UO2 thermal conductivity is scanned. Its range and distribution of uncertainty is determined. In the example, the MATPRO (Reference 4) reported error in thermal conductivity is +/-0.2W/mK. The example treats this as the one sigma range in a normal distribution. A total of 1000 cases were run for each parameter in each of several different scenarios. The cases were chosen to have equal probability between them.

The code discussed in the example is a development version of ELESTRES-IST. This new version of this code has not yet been released for use in safety analysis. It is interesting to note that approximately 20,000 cases were run for this version of the code during initial stress testing. This very likely exceeded the total number of cases run through the code in its life to that point.

Figure 3 shows the results of running eleven cases through the code. Running a set of cases like this would be quite typical of the effort involved in verification or validation test runs of a code. The typical analyst would find little to be concerned about in this graph. There is some sign of non-linear behaviour in that the curve seems to have a small amount of flutter. The trend of decreasing thermal conductivity is increasing internal gas pressures, with a maximum near -0.3 W/mK. This trend seems to be reasonably well behaved over a total of six sigma of thermal conductivity, from plus three to minus three.

Figure 4 shows a very different story. Figure 4 shows the full 1000 cases of scanning thermal conductivity. This graph shows a number of troublesome features. Most obvious and dramatic is the region of severe instability with an increment near -0.2 W/mK. At its most extreme, this instability shows a discontinuity of more than 4 MPa. However, there are discontinuities and oscillatory behaviour in large portions of the curve, with one significant discontinuity occurring right at the nominal value. Such behaviour is troubling from several aspects:

- It is difficult to see how it could be physical.
- It is difficult to contemplate comparing such results with experimental data.
- It is difficult to contemplate applying a CSAU like method.

These results were brought to the attention of the code developers. On examination of the code they did indeed find a program fault. On correcting this fault, the stress test was repeated, and the results are shown in Figure 5. Once again, this is a very different result from the previous graph. The curve is now very smooth. And even the slight flutter from the coarse scan has been removed. This code is now one in which we may have a great deal more confidence. The curve is very reasonable physically. It would be direct to compare to physical measurements. And applying a CSAU like method to the uncertainty analysis of this code would be trouble free.

Stress testing through SheepDog has thus uncovered a bug in an analysis code which would have been quite difficult to uncover otherwise. As well, it has allowed specific parameter values to be determined that would manifest the bug. This has significantly aided in locating the portion of the code that needed to be changed.

Summary

SheepDog is a batch job scheduler that allows the workstations on a WindowsNT LAN to operate as a distributed batch queue. It allows calculation methods to be applied which until quite recently were prohibitively expensive or slow. It is easy to install, easy to customise, and easy to remove. Its cost of implementation is relatively small. Dramatic savings of resources and time are possible if this method is applied to code qualification, safety analysis, and uncertainty analysis activities.

Figure 3 Pressure vs. Thermal Conductivity Increment Before Bug Fix
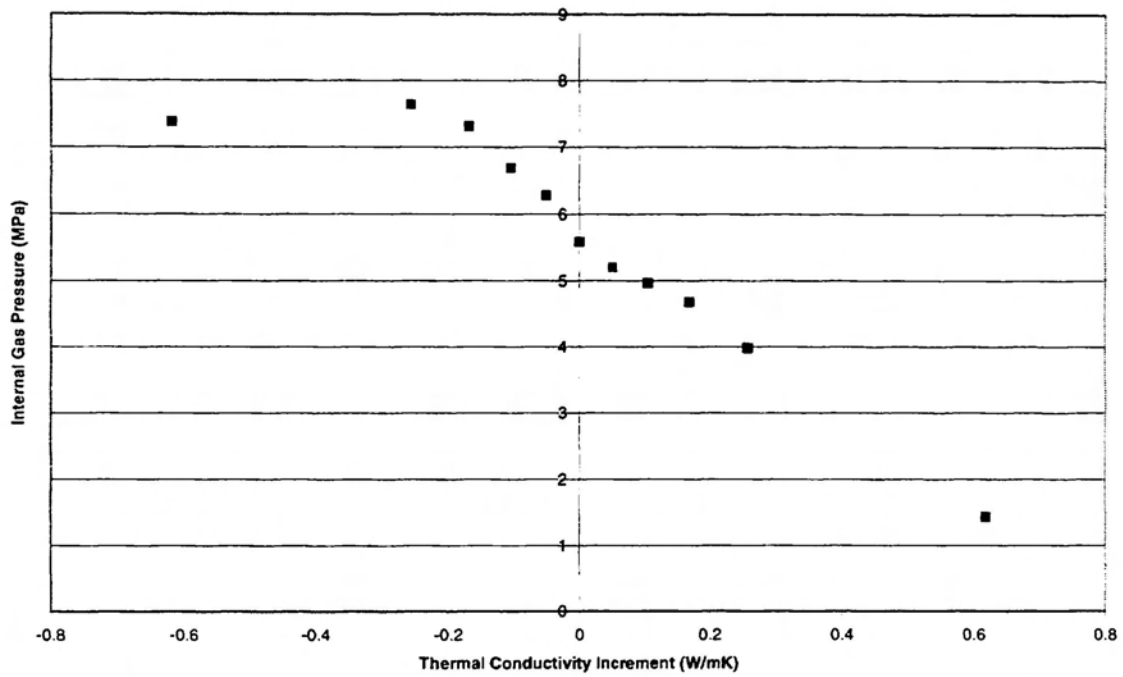

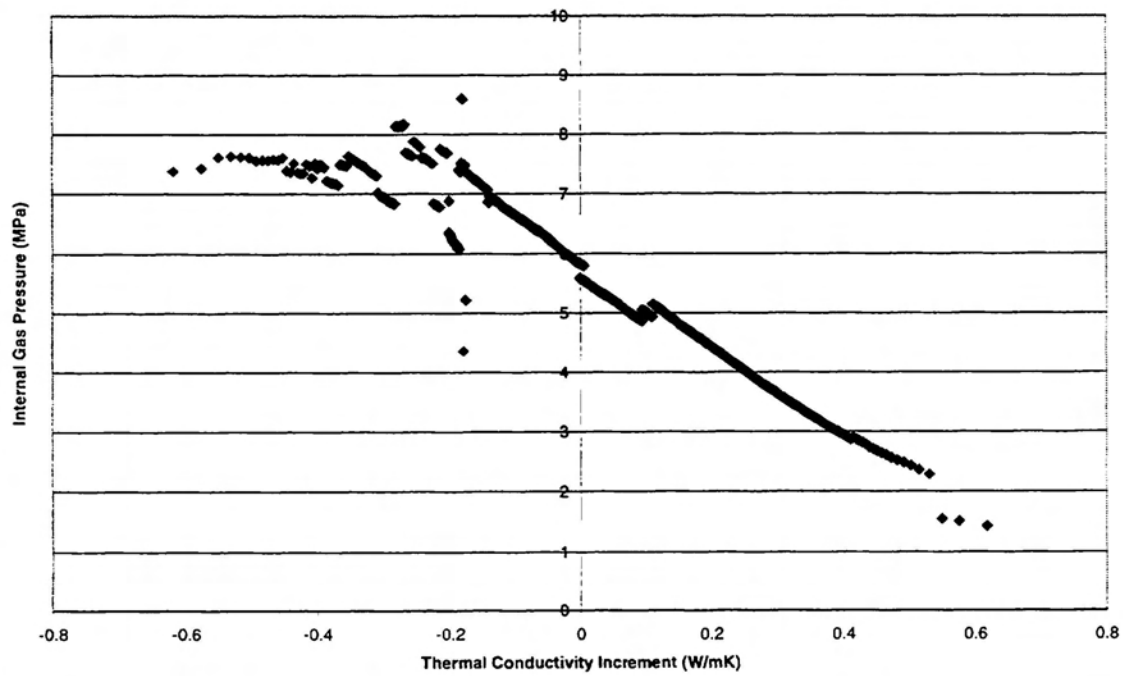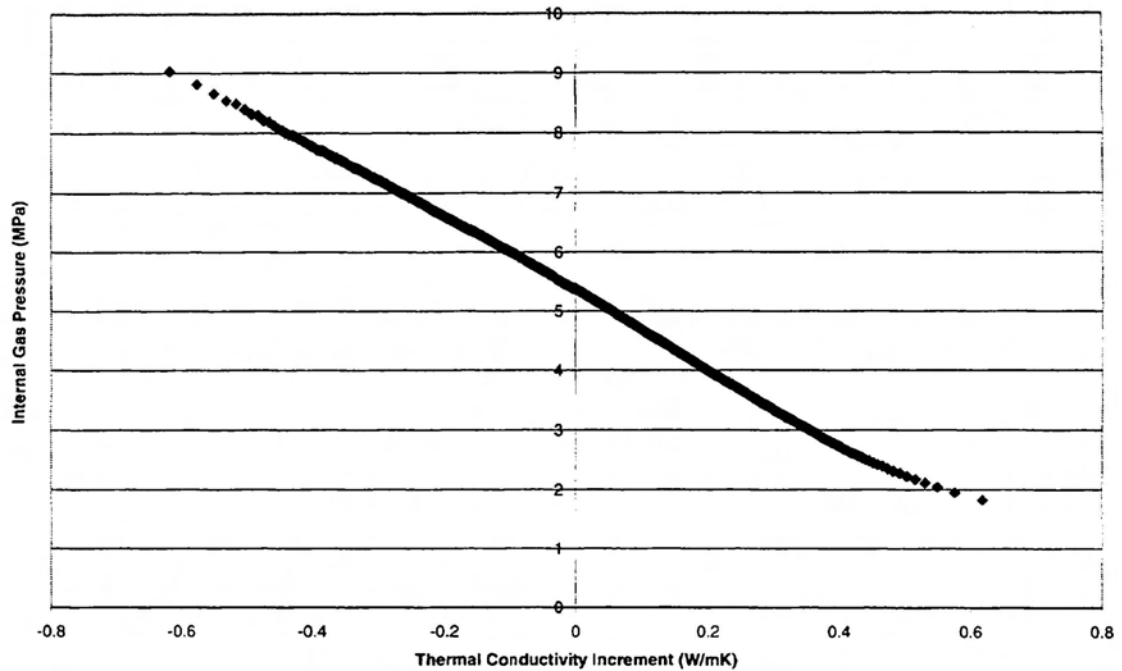**Figure 4 Pressure vs. Thermal Conductivity Increment Before Bug Fix**

Figure 5 Pressure vs. Thermal Conductivity Increment After Bug Fix

References

1. Nuclear Engineering and Design, Topical Issue on Quantifying Reactor Safety Margins, Vol. 119 (1990) No. 1.
2. Steven H. Langer, "A Comparison of the Floating-Point Performance of Current Computers", Computers in Physics, Vol. 12, No. 4, JUL/AUG 1998, pp 338-345.
3. B. Effron and R. J. Tibshirani, *An Introduction to the Bootstrap*, Chapman & Hall, 1993, ISBN 0-412-04231-2.
4. *SCDAP/RELAP5/MOD2 Code Manual, Volume 4: MATPRO - A Library of Materials Properties for Light-Water-Reactor Accident Analysis*, Edited by J.K. Hohorst, NUREG/CR-5273, 1990.