USER-FRIENDLY SCHEDULING TOOLS FOR LARGE-SCALE SIMULATION EXPERIMENTS

Heath A. James
Ken A. Hawick
Chris J. Scogings

Institute of Information and Mathematical Sciences
Private Bag 102-904 NSMC
Massey University, Albany
Auckland, 1311, NEW ZEALAND

ABSTRACT

Planning and steering numerical experiments that involve many simulations are difficult tasks to automate. We describe how a simulation scheduling tool can help experimenters submit and revoke simulation jobs on the basis of the most up to date partial results and resource estimates. We show how ideas such as pre- and post-conditions; interrupt handling; rapid experiment schema creation; and sparse parameter cross-products can be used to make a generalisable and user-friendly scheduling toolset. We describe our prototype in the context of typical long-running computational experiments of a complex networks simulation problem.

1 INTRODUCTION

Although scientists tend to write up reports on experimental studies as though the experiments were carefully planned and were carried out in a sequential and organised manner, this is not the way experimental science is actually done. We know from our own experiences (Hawick, Epperson, Windsor, and Rainey 1990, Epperson, Hawick, Windsor, and Rainey 1990) and of others (Feynman, Leighton, and Hutchings 1997, Medawar 1988), that both real and numerical experiments are almost never carried out in the neat structure that best fits the report.

In practice scientists interact very strongly with experimental results as they come in and the particular sample space investigated will be varied in direction and level of detail, as can be guided from the partial results already obtained. This "computational steering" (Smarr and Catlett 1992) is vital to obtain useful results both from the perspective to posing specific and finite experimental questions, but also from the practicalities of their being limited resources. Generally speaking if an investigation or experiment has a set of input search parameters, it is neither practical nor desirable to carry out a brute force search of the space spanned by a full flat cross-product of all the parameters. Some areas of the search space are flat and will not contribute insights to the posed questions. Not all parts of the space are worth searching in equally fine resolution. Indeed many of the numerical experiments that involve questions of power laws and other scaling phenomena need to be searched using a multi-scale or logarithmic resolution. A large part of the experimenter's skill is being able to manage, with finite resources, the sparse searching of the problem parameter search-space.

Despite the almost ubiquitous nature of computing in all fields of science and engineering, the problem of automating experimental science is still considerable. There are many *ad hoc* tools that help an experimenter and his team, such as spreadsheets, automated data collection systems and tools to word process the report. However, the goal of a fully integrated experimental management suite remains elusive. One approach that might be favoured by a traditional software vendor would be to try to produce a monolithic software product that meets the needs of at least some specific experimenter scenarios. Another approach, which we favour, is to look at how existing software tools and technologies can be better organised into a recognisable kit of parts that can be put together in a customisable manner for an experiment and team scenario.

To this end, in this paper we describe some issues that we have found important in managing large scale numerical simulation experiments and some of the tool components and interoperability protocols that we have used. In particular we focus on the issues of scheduling simulations that involve large and complex multi-scale parameter spaces, where the experimenter wants to make best use of limited time and computational resources. An adaptive and **user-friendly** simulation scheduling and management system is necessary. In Section 2 we illustrate the issues facing a computational scientist based on our experiences with a long running investigation of phase transitions in a complex

networks problem. A key issue we have identified is the need to adapt the schedule when partial information accrues. We want an experiment scheduler that incorporates and combines the best estimates on progress so far and **allows the experimenter to change his mind** – adapting the parameter search space on the basis of the most up-to-date information. In Section 3 we review traditional and current approaches to scheduling and their limitations. We present an initial scheduler developed for our problem in Section 4 and then a prototype solution based on various user-friendly programming concepts such as: high level pattern matching; interrupt handling; graphical interfaces; and integration scripting technologies in Section 5. Finally in Section 6 we suggest some ways whereby these ideas could be adapted to manage the scheduling of other custom numerical simulation experiments.

2 THE PROBLEM DOMAIN

We have battled for almost three years with naive solutions to managing experiments that are comprised of multiple independent parameters. Most of our initial solutions have been based on shell scripts that execute jobs with parameters drawn from a pre-set range. This paper presents a GUI-based solution to the composition and management of complex, sparse, cross-products of parameters in a local or LAN-based environment.

Our GUI prototype allows an experimenter to specify pre- and post-conditions for elements of an experiment and how successful experimental results are to be staged and incorporated into a well-managed experiment. Furthermore, the solution allows individual job elements of an experiment to be suspended or stopped, and the bulk experiment's configuration to be saved and later re-started.

We use our recent study of the Ising model (Hawick and James 2006), as the motivating experiment for this paper. The objective of our study is to identify the so-called "critical temperature" of the system. Briefly, each experiment in our programme involves the Monte Carlo sampling of a large number of configurations of an Ising system that can be represented, in part, by the following parameters:

- temperature, T
- number of lattice sites (N as a function of dimension, d, and lattice size in each dimension, L) $N = L^d$
- lattice perturbation percentage, p
- random number generator (RNG) seed, s

In fact, the simulation program we have developed for this experiment can take up to 17 different command-line parameters at once to produce different simulation outputs. However, a typical experiment may be posed as such:

fixed	$d = 5$, *spin states* $= 2$, *rewiring model* $= 1$, *repetitions* $= 10$
experiment set	$L = [15, 16, 17]$
experiment set	$[p, [T]] = [0.0100, [8.76, 8.85, 0.001]]$
experiment set	$[p, [T]] = [0.0200, [8.78, 8.85, 0.001]]$
experiment set	$[p, [T]] = [0.0400, [8.79, 8.87, 0.001]]$

This experiment is interpreted as such: dimension of the system, number of spin states, the rewiring model and number of times to repeat the experiment are all fixed for the experiment as a whole; for each of the L values, perform each $p \times T$ parameter product produced by the range given. For example, for $p = 0.0100$, $8.76 \le T \le 8.85$ using a delta of 0.001. Thus, the above experiment will result in $3 \times (91 + 71 + 81) \times 10 = 7290$ individual experimental jobs which must be scheduled and successfully run to complete the experiment. It must be noted that *each* experiment is assumed to have a unique RNG seed; this is crucial to achieve statistically significant results when repeating experiments with the same input parameters. This experiment is quite complex, but in our experience is not atypical. The system we describe in this paper tries to address this complexity. The structure of a typical experiment is then shown graphically in Figure 1.

We have discussed the task of managing the individual data files of this magnitude elsewhere for the purposes of quality control and also analysis (James and Hawick 2005). In this paper we focus on the ability to *specify* and *steer* simulation experiments in real-time.
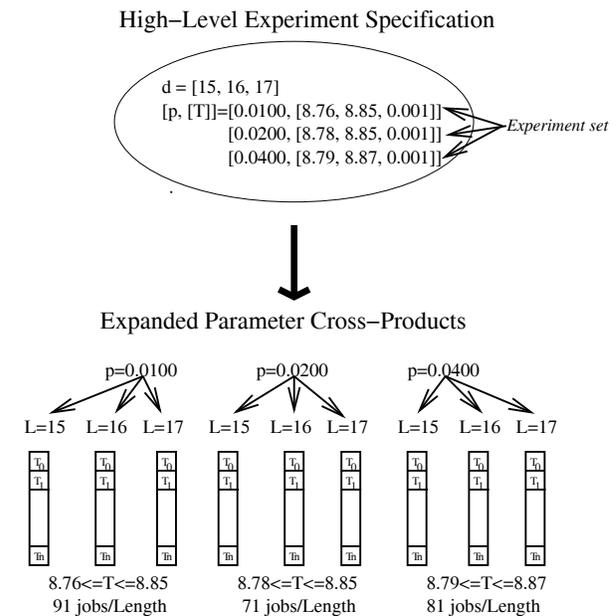

Figure 1: Structure of an experiment. Each experiment consists of a set of job elements created by the (potentially sparse) cross-product of input parameters.

Let us consider, for example an average high-end workstation that has four processors available for the user. Let us further assume, for simplicity, that the user decides to reserve one of those processors for doing interactive work, leaving three to run the experiment. A naive method of partitioning the jobs between the processors would be to assign each of the above $[p, [T]]$ sets to a separate processor. However, our simulation, like others, exhibits a phase transition in the execution time: when T is below or above the critical temperature the execution time is much faster than when T is approximately the critical temperature (James and Hawick 2007). Unfortunately we do not know the critical temperature *a priori*: in practice we observe that by simply dividing the work up based on either p or T we always end up with a significant fraction of our available processors still working and the remainder idle, waiting for us to analyse the results before submitting a new set of jobs.

3 EXISTING SYSTEMS

When one considers how to best schedule jobs across either a local set of processors or a cluster of closely-connected processors there are a number of important factors to consider. To us, the most significant factor is the ease with which we can submit the job to the scheduler. The next most significant factor is the reliability of the scheduler – that is our confidence that the experiment will complete.

An Internet search will reveal a lack of research or commercial products to manage simulation experiments. There are some specialised tools available (for example, Angel 1996), but these require one to write their code in a particular programming language or environment. Also available are so-called "wet-science" experiment managers (for example, STARLIMS Corporation 2007).

One general tool that is available for creating and controlling a set of parametrised experiments is Nimrod (Abramson, Sosic, Giddy, and Hall 1995). This tool allows an experimenter to specify a *single* experiment consisting of a cross-product of parameters and monitor its progress. Unfortunately it does not allow for the case where the parameters themselves are comprised of a "holey" or sparse cross-product (for example see the different T ranges in the experiment specified in Section 2). Treating the cross-product as dense can be extremely wasteful of resources and research effort. Thus, if an experimenter were content to run each experiment in isolation then it would be a perfectly adequate solution. Later versions of Nimrod (Abramson, Giddy, and Kotler 2000) have been extended to operate on the burgeoning Grid (Foster and Kesselman 1999). Other software (for example, FlowGrid Consortium 2004), has been created for running CFD applications across the Grid. Systems such as Persson, Grimm, and Ng (2006) allow users to computationally steer simulations to trade off optimisation objectives.

Thus, there are no real candidates for creating and controlling experiments which feature multiple "holey" cross products of parameters. As such we have written our own manager and associated schedulers.

We have experimented with using fully-fledged cluster scheduling and queue management software, such as PBS (Altair Grid Technologies 2007) (and its variants) and XGrid (Apple Computer Inc 2007) on our local processors. Anecdotally we have found these products a non-trivial burden to set up and administer, and the apparatus required to batch submit thousands of copies of anything but the simplest jobs can become quite complicated.

The main drawback to using cluster queue manager software on even our local machines is the case in which we need to revoke the experiment (or part thereof) after it has started. This is a non-trivial exercise in most scheduling software, even with a graphical front-end to the cluster such as provided by `xpbsmon` (Altair Grid Technologies 2007): one must semi-manually work out which jobs are no longer required and remove them from the execution queues. Only some queuing systems provide automatic clean-up scripts.

4 AN INTERIM SOLUTION

If one is developing a user-friendly scheduler that is only intended for use with a single application then it is easy to construct a multi-threaded program in, say, Python (Python Software Foundation 2007) that will maintain a queue of experimental jobs and allocate each job to an available processor in a first-come first-serve fashion. The basic pseudo-code for this program is shown in Algorithm 1.

Algorithm 1 Scheduler pseudocode

---
  while more job parameter sets **do**
    Read job parameter set
    Expand list of jobs
    Check to see whether it is necessary to run each job
  end while
  Create N queues, one for each processor, as a thread
  for each queue **do**
    if non-empty queue of jobs **then**
      take job from front of queue
      inspect pre-conditions
      run job
      inspect post-conditions
    end if
  end for
Ensure: appropriate cleanup on interrupt

We specify the job elements for our Ising code by an $n-$tuple consisting of the command-line string used to invoke the simulation and another command to run the post-conditions, as such:

```
cmdstring1 = "<pre-conditions>"
cmdstring2 = "<run experiment>"
cmdstring3 = "<post-conditions>"
tuple = filenameLog, cmdstring1, \
        cmdstring2, cmdstring3
q.put_nowait(tuple);
```

Each of the processor threads exists in a simple loop while there are jobs remaining in the queue:

```
def worker():
 while True:
   # item[0] filename mask
   # item[1] pre-conditions
   # item[2] runs experiment
   # item[3] post-conditions
   item = q.get()
   commands.getoutput(item[1])
     # pre-conditions
   commands.getoutput(item[2])
     # run experiment
   commands.getoutput(item[3])
     # quality control and
     # compress the output
   q.task_done()
   print "Jobs remaining in queue: "+
     str(q.qsize())
```

We have introduced exception handler code that enables us to not only stop the scheduler on demand, but to also kill any running experiment processes and execute their post-condition code. Typically the post-condition code we are interested in involves the testing of an output file either for a set number of lines or the presence of certain strings. If these features are not present in the output file then it should either be deleted or at least moved to a temporary place for human inspection.

5   A MORE GENERAL SOLUTION

We have since generalised the scheduler system developed for the application described in Section 2. The new GUI-based prototype scheduler is applicable to multiple different types of of simulation experiment while still retaining the most important design feature: allowing running tasks to be cleanly killed while allowing for inherent variations in simulation run-time to be "smoothed out". We chose to implement the prototype using Python's Tcl/Tk interface module, Tkinter. This system allows us to efficiently specify "simulations on demand" (Fox, Williams, and Messina 1994).

The architecture of our generalised scheduler is shown in Figure 2. The system is broken into two main components: the Experimental Manager and the Scheduler/Dispatcher.
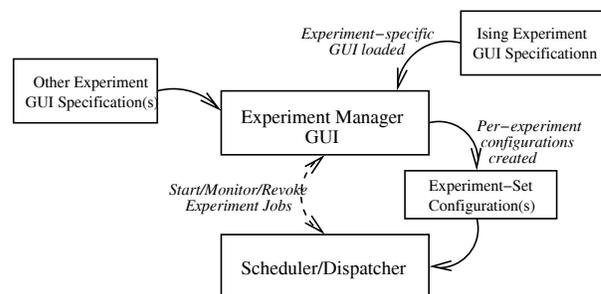

Figure 2: Architecture of our generalised prototype. Each experiment is specified by a customised GUI specification which tells the Experiment Manager how to display the Experimental Parameters. Once the user submits an experiment a configuration file containing the experiment details is created and then passed to the Scheduler. The user has the ability to control the Scheduler from the Experiment Manager module.

The Manager is generic: it requires a per-experiment GUI/Experiment specification file to specify the details of an experiment. After an experiment is specified a configuration file is created to save the details; this is, in a sense, passed to the Dispatcher for processing. The Manager is then able to monitor the Dispatcher and request updates on the experiment's progress or revoke submitted jobs.

Our current prototype, shown in Figure 3 allows the experimenter to specify a set of pre- and post-conditions that must be satisfied for the experiment to be deemed successful. At present the conditions may only be selected from a list of generic choices such as "require a file", "space available", "length of a file" and "occurrences of a symbol in a file". In order to specify which file the operations apply we allow the experimenter to either use a concrete filename (e.g. `rng.cfg`) or to use general patterns (e.g. `ising-*-*-*.log`). We are mindful of UML's Object Constraint Language extension (Object Management Group (OMG) 2006) and are using this for a reference for future prototypes.

The figure also shows a basic experiment posed through the management system. In fact this experiment is the same as discussed in Section 2. One can see three sets of $P - [T]$ variables that must be explored to complete this experiment. Of particular note is the fact that the start and end temperatures for each P value overlap but are not the same.

When the experimenter submits the experiment to the system they are presented with a monitor interface that shows the different parameter sets, the number of jobs this expands into (see Figure 1) and also what job each processor is currently running. In addition, the user has the option to update the monitor interface to show the percentage that each parameter set has been processed. This is shown in Figure 4.
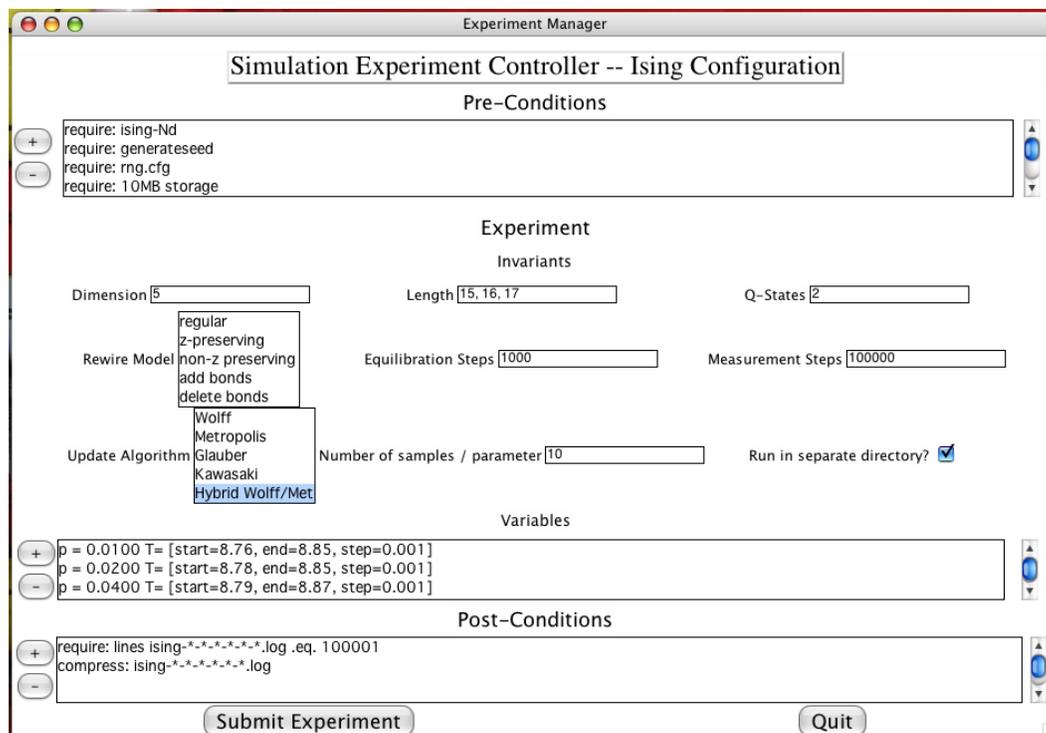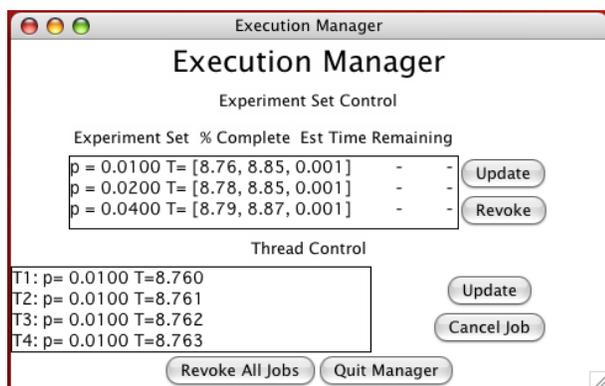
Figure 3: Our GUI front-end to the scheduler

Figure 4: Manager interface

It is not unusual for experimenters to perform partial analysis of results as they become available. In our case this is a simple task that is implemented by yet more scripts. Furthermore, it is not unusual, especially in the early stages of posing an experiment to perform a coarse-grained parameter sweep to identify the areas of parameter-space of interest. In these cases it is often useful to be able to computationally steer the simulation towards parameters of greater interest. For example, if one realises the area of interest is in the first half of a T set's range then there is little point in processing the second half of the set. For this reason the user is given the opportunity to steer their experiment through the revocation of either complete parameter sets or canceling individual jobs running on processors. The

use of post-conditions allows a canceled job to be cleaned up and the next job started by the system without leaving "dangling" files.

We are currently experimenting with efficient ways of specifying the files in the case that a complete experimental set needs to be revoked. Currently under consideration is the use of revocation-masks for the Scheduler/Dispatcher.

We intend that future versions of our scheduling prototype will include the ability to use a simple web-services interface to allow the remote creation and steering of simulation experiments. This was an idea that we successfully used to control the automatic processing of large amounts of satellite data held in an online data repository (James and Hawick 1998).

6 DISCUSSION AND CONCLUSIONS

It is of course an ambitious goal to construct a toolset that really automates the highly humanised process of formulating a research question and a series of experiments to address it. Nevertheless we believe it is possible for a set of tool components to be suitably integrated into an interoperable suite that **supports** this human activity. An experiment might start off described in a very vague natural language form, but as we have discussed, once the parameter set can be identified and quantified, the matter of managing a set of overlapping and "holey" cross-product parameter spaces is a suitable target for automatic tool support.

A quantifiable experiment scenario might be to locate the value of some parameter such as temperature or pressure where an effect, once identified, occurs. Another scenario might be to push known parameter regions outwards to confirm that an effect does or does not still occur at extreme conditions. Our numerical phase transition simulations fall into the former category. Many materials science experiments involving real physical systems fall into the latter.

Experimental science is limited by the sheer **complexity** of parameter spaces and closely interlinked parameters as well as the resource cost of searching large parameter spaces. We believe this applies equally to the arena of computational simulation science.

We have identified what we believe are the main issues for a practical set of experiment management tools. The first of these concerns the need to handle complex sparse parameter spaces, particularly with multi-scale parameters. Other issues include: the ability to revoke or refine the set of simulation parameters dynamically; the need for heuristics and other techniques to allow resource estimates and partial results to be used to guide the experiments; an ability to integrate relatively simple tool components with new ones or with off the shelf systems. The ability to adapt is probably the main failing of most existing simulation scheduler systems. We believe there is great scope for a toolset, based around the prototype ideas we present, that could be customised semi-automatically for a wide range of specific simulation experiments.

REFERENCES

Abramson, D., J. Giddy, and L. Kotler. 2000, May. High performance parametric modeling with nimrod/g: Killer application for the global grid? In *Proc. Int. Parallel and Distributed Processing Symp (IPDPS)*, 520–528.

Abramson, D., R. Sosic, J. Giddy, and B. Hall. 1995, August. Nimrod: A tool for performing parameterised simulations using distributed workstations. In *Proc. 4th IEEE Symp. High Performance Distributed Computing*. Virginia.

Altair Grid Technologies Available at <http://www.openpbs.org> last visited 2007. Portable Batch System.

Angel, M. 1996. Conducting experiments with experiment manager. In *Proc 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 535–541.

Apple Computer Inc Available at <http://developer.apple.com/hardwaredrivers/hpc/xgrid_intro.html> last visited 2007. Xgrid: High performance computing for the rest of us.

Epperson, J. E., K. A. Hawick, C. G. Windsor, and V. S. Rainey. 1990, Nov. A small angle neutron scattering investigation of phase separation in an Fe-Cr-Ni alloy.

In *Proc. Materials Research Society Meeting*. Boston, USA: Materials Research Society. Symposium F.

Feynman, R. P., R. Leighton, and E. Hutchings. 1997. *Surely You're Joking, Mr. Feynman! (Adventures of a Curious Character)*. ISBN 0393316041: W. W. Norton & Company.

FlowGrid Consortium 2004. FlowGrid Project available from <http://www.unizar.es/flowgrid/>.

Foster, I., and C. Kesselman. (Eds.) 1999. *The grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-475-8.

Fox, G. C., R. D. Williams, and P. C. Messina. 1994. *Parallel computing works!* Morgan Kaufmann Publishers, Inc. ISBN 1-55860-253-4.

Hawick, K. A., J. E. Epperson, C. G. Windsor, and V. S. Rainey. 1990, Nov. Chemical phase separation in binary Iron-Chromium alloys. In *Proc. Materials Research Society Meeting*. Boston, USA: Materials Research Society. Symposium F.

Hawick, K. A., and H. A. James. 2006. Ising model scaling behaviour on z-preserving small-world networks arxiv:cond-mat/0611763.

James, H. A., and K. A. Hawick. 1998, February. A web-based interface for on-demand processing of satellite imagery archives. In *Proc. 21st Australasian Computer Science Conf. ACSC'98*, ed. C. McDonald, Volume 20 of *Australian Computer Science Communications*: Springer-Verlag Pte Ltd.

James, H. A., and K. A. Hawick. 2005. Scientific data management in a grid environment. *J. Grid Computing* 3:39–51.

James, H. A., and K. A. Hawick. 2007. Computational data structures for high-performance complex network-based small-world simulations. *submitted to ACM Transactions on Modeling and Computer Simulation*.

Medawar, P. 1988. *Memoir of a thinking radish*. ISBN 0192829834: Oxford University Press.

Object Management Group (OMG) 2006, June. Object Contraint Language Specification 2.0 available from <http://www.omg.org>.

Persson, A., H. Grimm, and A. Ng. 2006. On-line instrumentation for simulation-based optimization. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto.

Python Software Foundation 2007. The python programming language.

Smarr, L., and C. E. Catlett. 1992, June. Metacomputing. *Communications of the ACM* 35 (6): 44–52.

STARLIMS Corporation 2007. STARLIMS Laboratory Information Management System. available from <http://www.starlims.com>.

BIOGRAPHIES

HEATH JAMES is a Senior Lecturer in Computer Science at Massey University (Auckland Campus). He has a PhD in Distributed and High Performance Computing from the University of Adelaide. He is a member of the ACM and Society for Modelling and Simulation International. His main research interests are Data Management Issues in Large-Scale Complex Systems Simulations.

KEN HAWICK holds the Chair of Computer Science at Massey University, in New Zealand. Professor Hawick has a PhD in Computational Physics from the University of Edinburgh and is a Member of the IEEE, IEEE Computer Society, ACM, SIAM, the Institute of Physics, the British Computer Society, and is a Fellow of the Royal Meteorological Society, an Associate Member of the Chartered Management Institute and is a member of the Royal Society of New Zealand. Hawick's main research interests are in studying emergent behaviour in complex systems and developing quantitiative metrics for emergence and complexity.

CHRIS SCOGINGS is a Senior Lecturer in Computer Science at Massey University (Albany Campus). He is interested in Artificial Life and Simulation.