Parameter Exploration in Science and Engineering using Many-Task Computing

David Abramson, Blair Bethwaite, Colin Enticott, Slavisa Garic, Tom Peachey

Abstract—Robust scientific methods require the exploration of the parameter space of a system (some of which can be run in parallel on distributed resources), and may involve complete state space exploration, experimental design or numerical optimization techniques. Many-Task Computing (MTC) provides a framework for performing robust design because it supports the execution of a large number of otherwise independent processes. Further, scientific workflow engines facilitate the specification and execution of complex software pipelines, such as those found in real science and engineering design problems. However, most existing workflow engines don't support a wide range of experimentation techniques, nor do they support a large number of independent tasks. In this paper we discuss Nimrod/K - a set of add in components and a new run time machine for a general workflow engine, Kepler. Nimrod/K provides an execution architecture based on the tagged dataflow concepts developed in 1980's for highly parallel machines. This is embodied in a new Kepler 'Director' that supports many task computing by orchestrating execution of tasks on on clusters, Grids and Clouds. Further, Nimrod/K provides a set of 'Actors' that facilitate the various modes of parameter exploration discussed above. We demonstrate the power of Nimrod/K to solve real problems in cardiac science.

Index Terms— scientific workflows, parameter exploration, Kepler, many-task computing.

1 Introduction

In-silico design has become common practice in research and industry. Computer simulations are routinely used in engineering design allowing for complex structures to be simulated with significant accuracy. For example, fluid flow, crack propagation and electronic simulations are routinely used in product design and research. In-silico design has enormous impact because it allows exploration before real experiments are conducted, cutting design times dramatically, and also enabling the consideration of dangerous or impractical real world experiments. Parameter exploration methods make it possible to evaluate many different designs fairly cheaply by performing multiple runs with different parameter values, and thus there is interest in developing tools and techniques that support this idea.

Many-Task Computing (MTC) provides a general framework for executing a large number of otherwise independent processes on high throughput parallel engines [42]. It provides a perfect environment to perform parameter exploration because the simulations often involve the execution of different independent scenarios.

In the past few years, scientific workflows have emerged as a powerful technique for specifying and executing complex in-silico experiments. Whilst there are many engines available, almost all share the ability to script data manipulation and computational pipelines that invoke a series of steps, often utilizing distributed resources. Pipeline steps can range from those that perform simple computations locally, to accessing external instruments and databases, through to significant simulations on remote parallel machines.

Most workflow engines, on the other hand, are designed for generalised application. They include components (which we will refer to as actors in this paper) that perform a wide range of activities. Most engines implement a data-flow execution mechanism that supports some amount of parallel execution. However, in general, most of them lack:

1. Methods that facilitate parameter exploration; and
2. A MTC execution mechanism that supports high throughput evaluation of designs.

In this paper we discuss the development of actors that support parameter exploration, and consider a range of different design techniques – from complete enumeration to targeted search. Moreover, we discuss techniques that add MTC to existing engines. We have implemented a number of new actors in an existing workflow system called Kepler [6][7][20][34][35]. When coupled with a new parallel execution mechanism (called TDA) that supports MTC, we can explore multiple designs in parallel using clusters, Grids and Clouds. We demonstrate the power of a new tool called Nimrod/K using case studies in quantum chemistry and cardiac science.

2 Parameter Exploration

In this section we introduce some of the ideas that underpin parameter exploration, and show that these map nicely onto a Many-Task Computing approach. We then introduce the Nimrod tool family, and discuss how prior work supports parameter exploration. This section provides background required for sections 3 and 4.

• *All authors are with Monash University, Faculty of IT. E-mail: firstname.surname@ infotech.monash.edu.au.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

2.1 Techniques

Parameter exploration involves considering more than a single data point in analysing a system. Whilst this seems like an obvious statement, it is remarkable how many scientific experiments (especially in-silico ones) only consider a single, or small number of data points. This may occur because the user may simply assume that the system is well behaved, or because any more rigorous exploration may be prohibitively expensive or infeasible.

At one end of the parameter exploration spectrum is complete enumeration, in which parameter ranges are broken into discrete steps, and the direct product of values is generated and explored. Complete enumeration often generates too many designs, and thus there is interest in techniques that reduce the number of scenarios evaluated in a controlled way. As early as 1935 Fisher [23] designed more efficient procedures, thereby starting the now huge field of experimental design [13]. With the advent of in-silico experiments, there arose the opportunity for much larger experiments, exploring more parameters and more values for each parameter. Such experiments will still benefit from careful selection of parameter combinations to run; good experimental design facilitates processing of the results, revealing for example which parameter interactions significantly affect outputs [44][22].

One commonly used technique, Fractional Factorial Design (FFD) [39], expresses a model in terms of each of the parameters and combinations of those parameters. It expresses a numerical output as constant, plus a linear combination of the parameter values, followed by secondary terms that concern all combinations of two parameters, tertiary terms that concern all combinations of three parameters, etc. Using this approach, it is possible to increase the resolution of the model until all combinations of all parameters are considered – this is the equivalent of complete enumeration. FFD assumes that higher order combinations have little effect on the output of a model, and thus these can be removed. The technique uses a variety of visualizations that clearly show the effects of the various parameters, and this makes it possible to prune the search space in a controlled way.

Both complete enumeration and FFD generate the scenarios statically – that is, they decompose the parameter values (and ranges) into discrete instances, and send these off for evaluation as a separate phase. When used for in-silico design, such a technique works well with commodity schedulers such as PBS, Condor, etc, where the jobs can be placed in a queue for execution.

An alternative to generating all scenarios up front is to use an iterative algorithm that evaluates the quality of solutions before choosing a new set of scenarios. This mode of operation is well suited to automatic optimization, in which the goal is to minimise or maximise some objective cost function. Various optimization algorithms can be employed – ranging from heuristics that understand the design space, through to generic algorithms such as Genetic Algorithms [46][43][1][40][41]. Optimization of this type opens up a new range of problems called inverse problems, in which the parameter values are computed to minimise the difference between some model output and a known solution. Inverse problems are common, and are also difficult to solve in general. Accordingly, it is desirable to have a range of search algorithms available. Software has been developed to assist in such optimization searches, such as LMS Optimus [32], Nimrod/O [40] and Matlab Simulink [33].

2.2 Nimrod's approach to parameter exploration

The Nimrod tool family automates the techniques discussed in the previous section using many-task computing (MTC) [42]. It targets computational models that are time consuming, and executes these on a range of platforms from high-end workstations to large compute clusters. Traditionally, Nimrod consists of three related tools:

- Nimrod/G, which performs complete enumeration [5];
- Nimrod/E, which performs fractional factorial design [39]; and
- Nimrod/O, which performs optimization [1].

Figure 1 shows the relationship between these three tools, and how they are exposed to users by a Web portal. They are configured by a description called a plan file. A plan file describes the parameters of interest, their types, values and ranges, and also a set of tasks that tell the system how to execute the model. Plan files are declarative, and typically very small. They use a specially formulated syntax that describes an experiment.

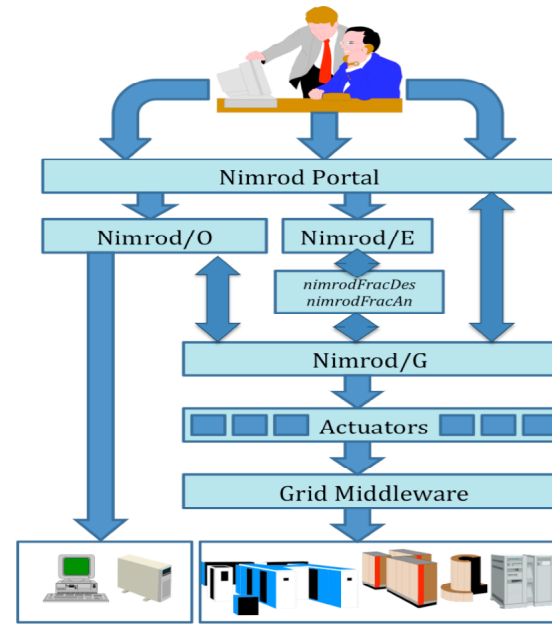

Figure 1– Nimrod tool chain [3]

As well as functioning as a user level tool, Nimrod/G contains an API that allows other applications to schedule and execute jobs. This API, used by Nimrod/E and Nimrod/O, makes it possible to evaluate models on demand. Whilst Nimrod/E generates a set of jobs statically, Nimrod/O generates them on the fly using a range of non-linear optimization algorithms. After each batch of jobs is executed, the objective cost function values are returned to Nimrod/O, which then decides on the next set to explore. All tools exploit parallel and distributed resources

by leveraging a range of Grid middleware – the most mature and commonly available interface is based on the Globus Toolkit [24][25].

Nimrod/G and Nimrod/E generate as many jobs as possible based on the resolution of the parameter specifications. Nimrod/E implements the Fractional Factorial design discussed in 2.1. It allows a user to specify the model resolution, that is, the number of parameters combined at a time. The techniques for doing this are discussed in more detail in [39]. Nimrod/O, on the other hand, uses a range of non-linear optimization methods to explore the design space. Each of these attempts to minimize or maximise some objective cost function as computed by the application, and range from various gradient descent techniques to genetic algorithms [40]. Nimrod/O generates jobs in parallel batches, using parallel versions of various search algorithms where possible [1][40][41]. Nimrod/O also executes multiple searches concurrently in order to handle non-linear problems that have multiple local minima. As a consequence, all three tools generate multiple tasks. A variety of schedulers allocate jobs to resources, ranging from a first-come-first-serve heuristic that simply allocates jobs to the next available resource, through to sophisticated schedulers based on a computational economy [14].

These traditional Nimrod tools have been applied to a very wide range of applications [29]. For example, one quantum chemistry application generated tens of thousands of compute jobs [50][51] and ran across the IEEE Supercomputing meeting in 2003. Whilst most experiments are completed in the order of days or weeks, Nimrod has successfully completed an environmental modelling experiment that lasted for about 6 months [36]. This required solutions to fault tolerance and data movement to be developed, as discussed in [11], because the infrastructure varied considerably across such a long period. Nimrod has been applied on a few thousand distributed hosts [12], however, the currently implementation does not scale to larger configurations because of technical restrictions. These limitations are currently being addressed by a new set of backend services.

In this paper we discuss the design of a new tool in the Nimrod family, Nimrod/K. Nimrod/K supports the execution of workflows but is combined with the parameter exploration ideas developed in the earlier Nimrod tools. Thus, Nimrod/K incorporates experimental design algorithms from Nimrod/E, and optimization algorithms from Nimrod/O. The current implementation uses Nimrod/G to provide backend compute services much as shown in Figure 1. Nimrod/K is addressed in more detail in section 4.

3 Workflow Engines

A number of groups have developed scientific workflow engines that allow a user to compose a complex virtual application based on pre-existing, in some case, legacy components [28][35][7][6][31][19][30][47][18][21][52][53][54][56].

In this model, components typically take input and produce output as part of a pipeline. The workflow system schedules the computations on the most appropriate (or selected) resource only when the inputs are available. Likewise, when the output is produced, it is forwarded to the next computation in the pipeline. Scientific workflows have been applied to diverse fields such as Computational Chemistry [7], Ecology [6] and Bioinformatics [31]. In this paper we focus on Kepler, although it shares many features with other workflow engines.

3.1 Parameter Exploration and MTC

While most workflow engines support some form of Many-Task Computing, most of them are remarkably static – that is, the graphs that are generated do not typically change size at run time and thus the level of parallelism cannot be varied easily. This static code generation strategy means there are usually only two ways to specify parallel activity. Either,

1) build a very large static Directed Acyclic Graph (DAG) with cloned sub-graphs for each parallel activity; or
2) write a workflow that contains logic to spawn parallel activities.

The first of these means that the amount of parallelism cannot change once the graph has started execution, which is restrictive. Further, if there is a lot of parallelism, the graph can become very large and difficult to manage. For example, a workflow that wishes to process the contents of a database, in parallel, must be replicated many times to allow them to run in parallel. The underlying graph can become very large, but it also contains a high degree of redundancy since the same operations are applied to each database element. This is the approach taken with tools like APST [15] and Pegasus/DAGMAN [18]. The second alternative supports dynamic parallelism, but pushes the onus onto the programmer to add the task management logic. For example, users must explicitly code for parallel execution, and incorporate loops that process the contents of the database, provide mechanisms that spawn independent calculations and then synchronize the results. This process significantly complicates the task of writing a workflow. In spite of this, the approach is typical of systems such as Kepler, for example.

An alternative to static code generation is to support the dynamic spawning of sub-graphs. This is the approach effectively taken in Nimrod/K, Swift [57] and the most recent additions to Taverna [38]. These systems are simpler to use, because the task management code is embedded in a middleware layer, and they can scale from small to large amounts of parallelism very quickly and dynamically.

Almost no workflow engines support parameter exploration methods explicitly. One exception is Geodise [55], which does recognize the need for optimization algorithms in in-silico workflows. It does this by exposing "optimisation services in a flexible, generic interface that can be easily integrated into various environments and used to compose different optimisation workflows." [55]. In Geodise, optimization algorithms are exposed as Web services, which can be invoked by a workflow system, or

any other programming language that can execute Web service calls. Apart from Geodise, few other projects have integrated parameter exploration primitives in the workflow tools, and none have done it as completely as Nimrod/K.

3.2 Kepler

Kepler builds upon the Ptolemy II framework [30] developed at the University of California, Berkeley. Ptolemy II is a Java-based software framework with a graphical user interface called Vergil. The Ptolemy II project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components.

The focus of the Ptolemy II system is to build models of systems based on the assembly of pre-designed components. These components are called actors:

"An actor is an encapsulation of parameterized actions performed on input data to produce output data. An actor may be state-less or state-full, depending on whether it has internal state. Input and output data are communicated through well-defined ports. Ports and parameters are the interfaces of an actor. A port, unlike methods in Object-Oriented designs, does not have to have call-return semantics. The behaviors of a set of actors are not well-defined without a coordination model. A framework is an environment that actors reside in, and defines the interaction among actors." [30].

The interaction styles of actors are captured by Models of Computation (MoC). A MoC defines the communication semantics among ports and the flow of control and data among actors. "Directors" are responsible for implementing particular MoCs, and thus they define the "orchestration semantics" of the workflow. Simply by changing the director of a workflow, one can change the scheduling and overall execution semantics of a workflow, without changing any of the components or the network topology of the workflow graph. Two directors that are commonly used for Grid programming are the Process Networks (PN) and the Synchronous Data Flow (SDF) directors, which are based on Kahn Process Networks.

A Process Network is a directed graph, comprising a set of nodes (processes) connected by a set of directed arcs (representing FIFO queues). Each process executes as a standalone sequential program and is wrapped as a Ptolemy II actor. The one-way FIFO channels are used for the communication of processes and each channel can carry a possibly infinite sequence (a stream) of atomic data objects (tokens). Since channels have, in principle, unbounded capacity, writes to channels are non-blocking, while reads are blocking [30]. The SDF domain is a dataflow-based execution model in which a sequential execution order of actors can be statically determined prior to execution. This results in execution with minimal overhead, as well as bounded memory usage and a guarantee that deadlock will never occur.

Kepler extends PtolemyII with a significant number of actors aimed particularly at scientific applications, e.g., for remote data and metadata access, data transformations, data analysis, interfacing with legacy applications, web service invocation and deployment, provenance tracking, etc. Kepler also inherits from Ptolemy the actor-oriented modeling paradigm.

In spite of its significant power, Kepler, and many other current workflow systems, do not support dynamic parallel execution. Thus, as discussed, users must explicitly code a workflow to cause it to execute elements in parallel – either by replicating the workflow statically, or adding looping constructs that scatter and gather threads. Both of these techniques significantly complicate the workflow and obscure the underlying business logic. In the next section we discuss a tagged dataflow architecture, and show that it provides a much richer execution environment for parallel workflows.

4 Nimrod/K: Parameter Exploration with MTC

Nimrod/K combines scientific workflows with the parameter exploration techniques and Many-Task Computing. It builds on Kepler and Ptolemy as core engines, but augments them with

1. A new mechanism for spawning and executing many independent tasks; and
2. Actors that facilitate parameter exploration techniques.

Figure 2 shows a workflow as envisaged at design time, showing three actors connected by arcs. Tokens move between the actors conveying data, but the execution semantics are described by the choice of director. Importantly, for many scientific applications no workflow "code" needs to be developed. Instead, users are able to assemble a pipeline of pre-existing actors to implement the required logic. Kepler has an very large actor library, with components that perform functions ranging from data manipulation to explicit computation. Nimrod/K inherits these advantages directly from Kepler.

Figure 2 – Static workflow of 3 actors

4.1 Nimrod's approach to MTC (TDA)

As discussed, Ptolemy directors control the way a workflow is executed. Kahn Process Networks (PN) and Synchronous Data Flow (SDF) are two of the more common directors used in scientific workflows. Both of these execute when data, passed as tokens, is available on actors' inputs. SDF calculates a schedule statically, whereas PN supports bounded queues of tokens and leaves actors in a ready state until tokens arrive. While powerful, neither of these directors exploit parallel platforms in a way which supports parameter sweeps and search. For example, SDF only fires one actor at a time, regardless of how many tokens are active in a workflow. Thus, even when multiple tokens are queued, the workflow cannot exploit this concurrency. PN, which does allow some inter-actor concurrency, only runs one copy of each actor at a time. However, we want to generate a large number of scenar-

ios and execute these in parallel. Accordingly, we want the number of instances of any given actor to increase dynamically as more data tokens are sent to it.

To solve this problem, we created a new director called TDA. TDA implements a Tagged-Dataflow Architecture originally designed for the MIT, RMIT and Manchester data flow machines [27][2][9]. TDA augments data tokens with a tag, or colour field[4]. When multiple tokens of different colours are available on the inputs of an actor, multiple independent instances of the actor are invoked. Thus, by controlling the colouring and de-colouring of token, the workflow can scale to use an arbitrary number of processors. Figure 3 shows the flow of tokens in this workflow when only one token is present on the arcs – 3(a) shows a token on the input of actor 2, and 3(b) shows the token emitted by actor 2 after it has executed. Figure 4 shows the flow of multiple coloured tokens on arcs – 4(a) shows 3 tokens on the input to actor 2, and 4(b) shows the multiple copies of that actor running in parallel.

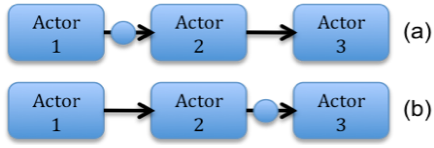

Figure 3 – Tokens in static workflow

The TDA implementation leverages a feature of Ptolemy that "clones" actors. The TDA maintained a set of queues for each actor. When multiple tokens of different colour arrive on an actor's input, TDA creates new copies of the actor using cloning. These clones are destroyed after execution, which allows the graph to dynamically expand and contract as the concurrency changes. More details are discussed in Sections 4.3 and 4.4.

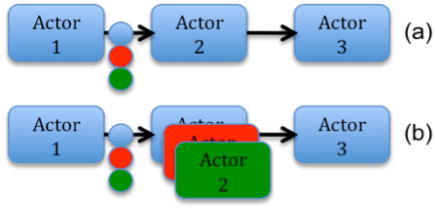

Figure 4 – Tokens in dynamic workflow

4.2 Parameter Exploration Actors

Three new families of actors have been created to support the parameter exploration methods discussed in Section 2.

4.2.1 Nimrod/G Actors

As discussed, Nimrod/G takes a direct product parameter values and computes a set of scenarios. Parameter ranges and values are normally provided in a plan file, although they can be added dynamically through the Nimrod/G API. In Nimrod/K, we have created a new actor that takes the same parameter descriptions used in plan files, and it emits a series of tokens – one per instance.

Figure 5 shows the way this actor is instantiated in a workflow, and also shows the configuration of the parameter values. In this case, ParameterSweep is linked to a MatlabExpression actor that executes a Matlab program. The tokens that flow into the Matlab actor contain actual values for the various parameters – in this case 9 different variables. This actor works with the current Ptolemy directors, however, when used in combination with the TDA, parallel execution of the different parameter combinations allows the workflow to execute efficiently on the Grid.

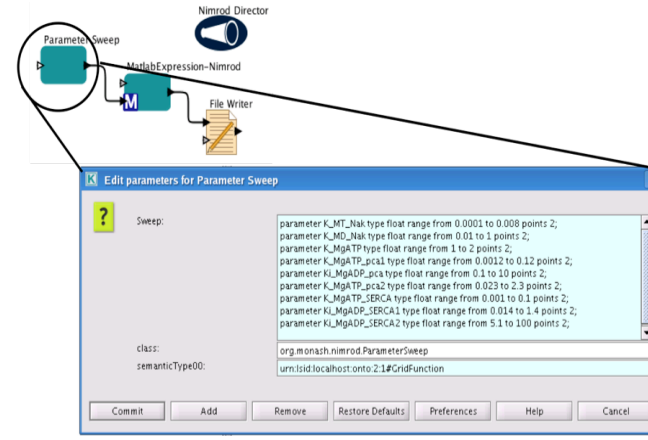

Figure 5 – Nimrod/G actors and configuration control

4.2.2 Nimrod/E Actors

Nimrod/E is similar to Nimrod/G in that it generates a set of experiments. However, it uses the Fractional Factorial Design algorithm to decide which combinations to explore. These are produced by the ExperimentalDesign actor. The plan file syntax for Nimrod/E is a superset of Nimrod/G, and contains additional statements that control the resolution of the model. In addition to the ExperimentalDesign actor, an EffectsEstimator actor computes the effects of the various parameter combinations. These are streamed to two different graphing actors, one that generates a Daniel plot, and another that generates a Lenth plot. These two diagrams show the effects of various parameter combinations in a graphical form. Lenth plots show the effects of various parameter combinations. Parameter combinations, which are shown as individual dots, are plotted using the Y axis range to indicate their contribution. Thus, dots that lie close to the X axis have little effect on the output. In the Daniel plots, the results are displayed using a cumulative graphing technique. Here, insignificant combinations cluster on a straight line, and significant ones appear off the line. Using these two forms of display, it is very easy to spot the significant combinations. Figure 6 shows a workflow that uses all these actors. Figure 7 shows examples of Daniel and Lenth plots.

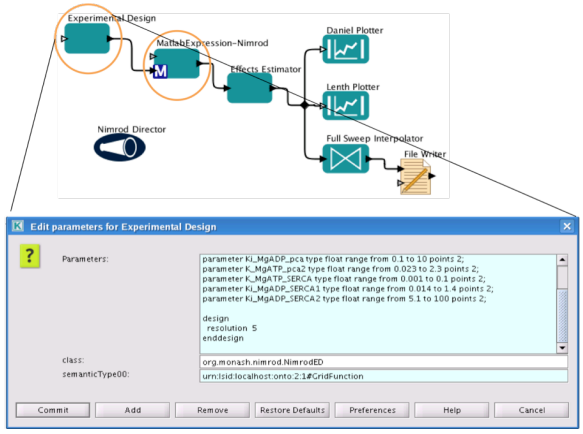

Figure 6 – Experimental Design actors

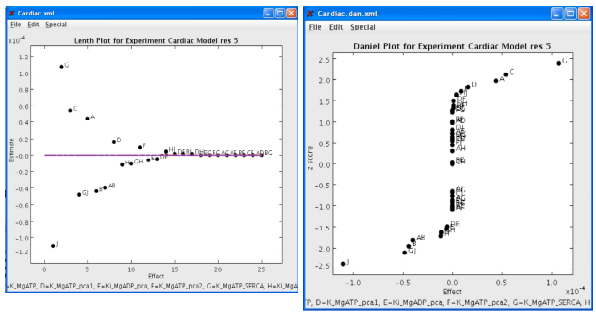

Figure 7 (a) Lenth Plot (b) Daniel Plot

4.2.3 Nimrod/O Actors

Optimization algorithms combine well with workflows because they usually involving repetitive looping in which results are passed from one iteration to the next. Implementing Nimrod/O functionality within Kepler exposes the components of the optimization process, giving the user a clear view of the data flows and facilitating substitution of these components. Accordingly, we have built a set of actors that support the generic components of an optimization algorithm, and also specific actors that implement specific algorithms.

Figure 8 shows a generic optimization algorithm and the actors that support it. First, the domain of the search is defined by specification of the input parameters using a similar method to Nimrod/G plan file syntax. Within this domain the next component will select starting points for the multiple optimizations. Each starting point will begin an optimization run using the specific optimizer actor. This actor generates sets of points that are sent for evaluation, a "batch of jobs" in Nimrod/O's terminology. Running the models represented by these jobs produces numerical results that are used to decide the next generation of points. This cycle continues until some convergence criterion has been achieved whereupon a final point is forwarded as the result of the search. Figure 8 also shows a component that evaluates constraint conditions imposed by the user. This calculates the margin by which a search point violates any constraints, and imposes penalties accordingly. If the constraint is a "hard" one then the point is completely forbidden, obviating the need for model evaluation.

The existing implementation of Nimrod/O supports a variety of optimization methods, namely gradient descent, simplex, genetic algorithms and simulated annealing.

4.3 Nimrod/K Design

In Section 3 we introduced the way Kepler (and the underlying Ptolemy) supports the use of different execution models with the same workflow. One of the reasons we have based our implementation on Kepler is that this important separation of concerns makes it possible to introduce a new execution mechanism based on the tagged dataflow execution model whilst leveraging the existing infrastructure. Accordingly, we have implemented a new Director called the Tagged Dataflow Architecture (TDA). The TDA builds on the existing SDF and PN directors but tags (or colours) tokens to distinguish different threads of execution. We envisage the idea could be added to other workflow engines with an ability to add new execution mechanisms. The TDA director supports existing 'legacy' actors in the PtolemyII project and the newer actors developed for Kepler, making a diverse range of actors available.

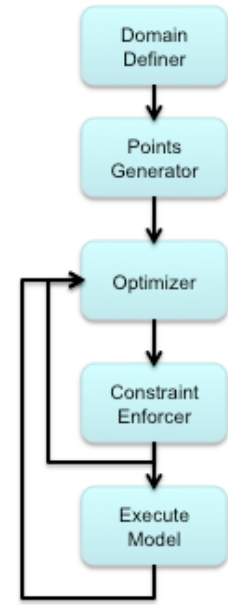

Figure 8 – Generic optimization actors

In the TDA, threads are identified by adding a unique tag to each token. These tag values are manipulated by a set of special tag manipulation actors, although the tag flow implementation is usually hidden from workflow designers. Even though some of Ptolemy II data types can be large complex structures, the implementation minimises memory usage by using pointers to the structure containing each token in the thread. This means the implementation scales well as the number of threads increases.

Underlying the SDF and PN directors are FIFO queues located on each of the inputs on an actor. Normally, when multiple tokens arrive on an input port, they are queued and can be processed when the actor is available. The

TDA director follows the same procedure, but with a separate queue for tokens with different tags. This means that multiple tokens with the same tag value queue up, whereas tokens with different tag values can be consumed in parallel. Multiple actors cannot read from the same queue because they only read from queues with the same tag value assigned to them, and no two actors are given the same tag value. This gives the ability to copy actors (PtolemyII calls this cloning) and invoke them using different tags simultaneously. Importantly, this approach requires no changes to existing actors, which are usually unaware that they have been cloned.

We offer three methods to assign and modify tags within a workflow. The first two are aimed at the workflow creator who wishes to parallelize execution of a workflow. The third is aimed at the actor developer who wishes to develop new actors that support parallel threads.

The first method is transparent to workflow designers, and involves a number of new actors discussed to date. The Parameter Sweep actor shown in Figure 9 generates tokens that are already tagged. One could imagine a small set of such actors, such as reduction operators, etc. Clearly, this method is preferred because the parallelism is implicit in the workflow specification, and no explicit tag manipulation is required. Thus, this method makes it possible to take existing workflows and parallelise them without modification to the core logic. The second method provides a special tagging actor that adds a tag to a given token. This actor has two inputs, one of which takes a tag and the other takes an arbitrary token. The output of the actor is a token with its tag set.

The third method, which is directed at actor developers, provides an API that allows tagging, retagging or removing the tag of a token. This means that a developer can write complex actors that abstract tag management from the workflow creator, but still expose sophisticated thread management techniques. Likewise, we have provided two methods to access the values assigned to the tag from both the workflow design and from within an actor's code. First, when an actor is invoked, the tag value of its input tokens is added to the actor's parameter scope making it available in the actor's parameter options for a workflow creator to use. Second, we provide API functions so that actor developers can also access this information.

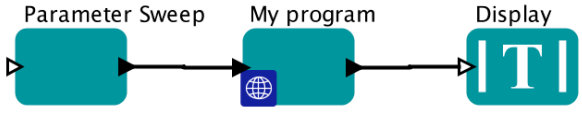

Figure 9 – a simple Kepler workflow

Figure 9, shows a simple parameter sweep experiment executing over an application called "My program". In this example, the parameter sweep actor produces a number of tokens, each containing one of the parameter combinations (stored in a record). When using the TDA director, these tokens are also tagged with the value of the record. For example, if the value of the record is "X=1, Y=1", then the value of the tag is "X=1, Y=1". The "My program" actor accepts one token and produces one token as output, whilst the Display actor displays any tokens it receives. We will now compare the different behaviour of the SDF, PN and TDA directors as they orchestrate this simple workflow.

The SDF director [34] is a fully synchronised and pre-scheduling MoC. It executes its actors in turn until every actor reports it is unable to execute. The PN director, on the other hand, [34] activates every actor simultaneously and blocks an actor when there are no tokens available. Therefore, when executing the workflow in Figure 9 under the PN director, all actors are activated simultaneously and the "My program" and the Display actors are blocked immediately. As the parameter sweep actor produces tokens, the "My program" actor proceeds to process the tokens one at a time. Likewise the Display actor displays the tokens as they arrive. As it turns out, both the SDF and the PN directors behave in similar ways for this simple workflow.

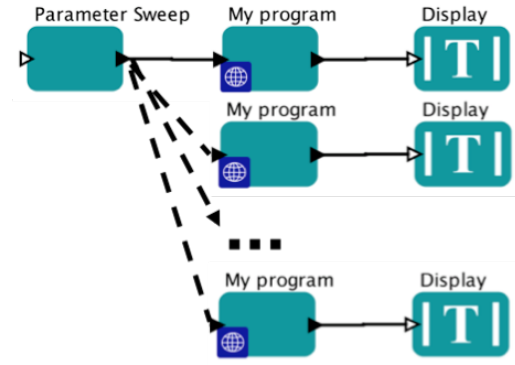

Figure 10 – The TDA director executing the simple workflow

Whilst the TDA director uses the same requesting technique as SDF, it also has the ability to create new threads for each of the tokens that it generates. In this example, the TDA director creates a single thread for the parameter sweep actor, which sequentially sends tokens to the "My program" actor. However, since each output token from the parameter sweep actor is tagged with a different value, multiple copies of "My program" execute in parallel. Importantly, the output of the "My Program" actor is tagged with the same value as its input, and thus, each "My program" thread communicates with a different Display actor thread. The effect of this is that the TDA director creates a separate thread for each of the parameter combinations and, as shown in Figure 10, they each display in parallel. It is important to highlight that Figure 10 depicts the run-time behaviour of the workflow, whereas, Figure 9 shows the workflow as designed.

A more complex workflow would probably gather or reduce the outputs of "My Program" together into a single thread, possibly reordering them into a single queue based on their tag values. This is analogous to a scatter-gather operation in a vector processor or a parallel loop instruction in a programming environment, and achieves the same sort of parallel execution.

Comparing directors shows the concurrency benefit of

using the TDA director and how actors remain unmodified in all environments. Many of the Grid actors in Kepler do not provide information on the consumption and firing rate of tokens. In the example given in Figure 9, it may not be clear how many tokens the "My program" actor needs to consume nor how many tokens it will eject for each firing and if these numbers remain consistent. As discussed in [34], the SDF director determines a schedule for a workflow at the start of the execution. Because of this, it is unable to handle non-deterministic actors that change their token usage behaviour during the execution while the PN director adapts to the changing state. In this sense, the TDA director is similar to the PN as its schedule is not fixed.

State fullness is another attribute of an actor that needs to be considered. State fullness is where an actor holds information from a previous firing making the order of tokens entering the actor important. This can also happen indirectly through shared resources. These types of actors are safe using the SDF and PN directors as there is only one copy of the actor on the workflow. These actors are supported with the TDA director, but may be of limited value as they may not be able to be cloned because of concurrency issues, and thus, should be used with caution. The TDA environment allows both the actor developers and the workflow designers to specify if and when an actor can be cloned. Issues relating to state and parallelism will be discussed in further work.

4.4 Implementation Details

Figure 11 shows the structure of the TDA director implementation. There are three main components: the token scheduler, the actor manager and the director. The token scheduler's role is to decide which tag an actor should process next. The actor manager is responsible for the execution of an actor, and for maintaining all of its copies. Like all directors in PtolemyII and Kepler, the TDA director maintains the execution of the workflow and reports to the workflow's calling function.

The token scheduler is responsible for deciding the actor firing order and the assignment of tokens to actor clones. The scheduler is notified of all token, movements and is responsible for deciding the number of actor copies that should be cloned at any time. This dynamic scheduling is different from the static scheduler used by the SDF director and the blocking method used in the PN director. We have also added a actor base class with an API that allows the scheduler to query the actor's resource requirements. A scheduler needs to consider resource availability to improve its scheduling even when using local resources. For example, a workflow may run a CPU intensive simulation package such as MATLAB which might be executed on a multi-processor. In this scenario, it is important for the scheduler to know how many MATLAB computations it can execute concurrently. Further to this, MATLAB might appear more than once in the workflow which needs consideration when scheduling these computations concurrently. It is the scheduler's responsibility to coordinate shared resources across the workflow.

The prototype token scheduler is based on a simple outstanding request queue where an entry is added when a token arrives on any actors' input ports. The entry holds a reference to the actor and a reference to the token's tag's value, and ensures that all outstanding requests are processed before terminating the workflow. Storing references to these objects does not create inconsistency issues because actors are never changed during execution and tags are immutable in this environment. Only one outstanding request is kept per actor and tag combination to avoid unneeded multiple requests that will increase the queue size. This has the most noticeable effect when an actor requires more than one token to fire, but will only need a single firing request. Storing requests this way reduces the memory requirement of the queue because the actor only needs to be flagged once to execute. When an actor finishes firing and reports that it can fire again, the token scheduler decides whether to requeue this actor and tag combination by checking if there are any outstanding tokens. The only exception are actors that are defined as "source" actors which always generate a new request after a successful fire until the actor reports it is no longer able to fire. Our prototype token scheduler currently clones as many actors as required to consume all outstanding tokens at that time. However, this strategy may generate a large demand on resources including memory, in both the number actors and the number of active tokens, and on shared resources. Similar issues have been addressed in the past with the k-bounded loops work of Culler [16] and we plan to integrate some of these ideas in the near future.

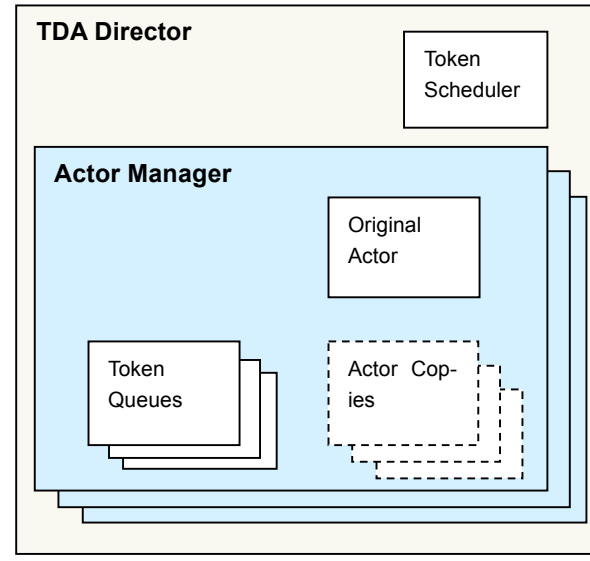

Figure 11 - Object associations

Actor managers are responsible for maintaining the actors on the workflow. There is one actor manager for each actor on the workflow. They provide all the functions required to maintain an actor, including functions that manage actor cloning, token inputs and outputs, initialisation and cleaning up. Importantly, actors do not require modification to work under the TDA director because the actor manager performs all the functions required. These

functions include the interface for a scheduler to query an actor's resource needs. The actor manager adds the tag's content (assigned by the token scheduler) to the actor's parameter scope and it ensures outgoing tokens have the correct tag for the actors that do not manage the tag itself.

The TDA director is invoked in the same way SDF and PN directors. Importantly, it is built on the same director interface as SDF and PN, and this allows it to interact seamlessly with the Ptolemy environment. It also has the functions to maintain synchronisation betweens the threads in its environment.

An important design feature of Kepler (and Ptolemy) is the use of multiple directors on the same workflow by using a hierarchical layout. This is implemented using special "Composite Actors" that supports nesting of directors. Just as PN directors allow an SDF controlled sub-graph actor in its workflow, a TDA controlled sub-graph can also be part of a PN controlled workflow. Further to this, it is possible to have an SDF sub-graph inside a TDA workflow. There are rules governing which directors can be nested as a sub-graph on a workflow, which are explained in [34]. The TDA director closely resembles PN director on which combinations are possible with one exception, it is possible to have TDA controlled sub-graph as part of an SDF controlled workflow. This is because the TDA director does not block on reads and can release control back to the SDF director when no internal actor is active. To complicate the issue, placing a PN director inside a TDA controlled workflow blocks the TDA director from releasing control back to an outlying SDF controlled workflow. This is because PN directors do not relinquishing execution until it is told there are no more tokens at which point it terminates completely. We are currently exploring the semantics of nesting TDA with other directors and will produce a set of templates with clearly defined semantics for workflow developers.

5 Case Studies

In this section we illustrate the three design techniques in Nimrod/K using two real world case studies in quantum chemistry and cardiac modelling. In 5.1 we describe the science behind these studies; more to provide a real world motivation and some context than to expose the details of the science. Case study 1 (Section 5.2) discusses a quantum chemistry calculation and demonstrates how the new TDA director implements a complete parameter sweep. Case studies 2 and 3 (Sections 5.3 and 5.4) implement various searches across cardiac models. Specifically, case study 2 demonstrates the optimization actors whilst case study 3 shows the use of the experimental design actors. In order to provide a high throughput Grid, the computations are performed on a distributed Grid of Linux clusters, as shown in Table 1.

Machine	# Cores	Hardware	Location
East	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	Monash University, Melbourne
	64	Intel(R) Xeon(R) 5160 @ 3.00GHz	
West	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	Deakin University, Geelong
South	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	RMIT, Melbourne

Table 1 – Properties of Grid Testbed

5.1 Case study science

5.1.1 Quantum Chemistry

This experiment concerns quantum chemical models – based on the Schrödinger equation – of assemblies of atoms as found in large biomolecules. To date, it has been impossible to use quantum methods on such large molecules because the computational time becomes prohibitive.

This work involves a set of approximations that replace large multi-atom systems with a single "pseudo" atom that has special theoretical properties that make it an analogue for the larger system. The solution involves computing a potential surface for the pseudo atom, called a pseudo-potential, and using this in the quantum calculations. A pseudo-potential does not describe a real chemical system, but can be used to approximate, and replace, a much larger molecular system. The result is that the quantum chemical calculations are much simpler, and faster, than if the original molecule had been used, opening the possibility of using quantum methods on very large systems. In this case study we need to explore a range of parameters that characterize the pseudo potential, and thus a parameter sweep is used to run across a large number or independent scenarios. The details can be found in [50][51].

5.1.2 Cardiac Science

The electrical activity of the heart is based on cycles of ion transfer via cell surface membrane. There has been much research on developing robust and accurate models of myocyte behaviour so that in-silico experiments can be performed on complete cardiac systems. The research has significance from basic science through to therapeutic drug design.

Cardiac excitation-contraction coupling is a sequence of well-orchestrated events. Both excitation and contraction processes and their interactions are required for an integrative model of cardiac electromechanical interactions [48]. A number of mathematical models have been developed to study excitation-contraction coupling in ventricular cardiac cells. Recently, our colleagues in the cardiac mechanics lab at UCSD integrated additional ionic-metabolic equations [8][37] into the Shannon et al. four compartment cell model [48]. The updated ionic model is more complex due to the multiple domains and increased number of unknown model parameters, so it implies a less stable set of ordinary differential equations systems. For this reason, any information gained based on this model can yield a significant level of misunderstanding if one wants to use it investigate the excitation-contraction coupling in ventricular myocytes.

As a result of this integration, we are interested in two

different experiments. First, we want to calibrate the new model so that it matches real physiological data. Second, we want to explore the sensitivity of the model to the parameters, and determine whether there are some combinations that are more important than others. Accordingly, the first experiment is performed with Nimrod/OK (case study 2) and the second with Nimrod/EK (case study 3).

5.2 Nimrod/K experiment: Case Study 1

The workflow shown in Figure 12 depicts the computation, which involves the execution of the GAMESS quantum chemistry package [45] a number of times, across a search space defined by 4 parameters (A, B, C and D). Here parameters A, B, C and D are to be determined that will provide the best fit between the real atomic system and target properties of the pseudo atom. The experiment used the results of the very large sweep over the A, B, C, D parameter space.

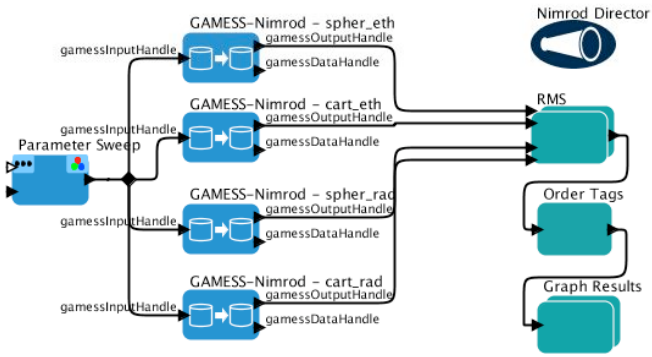

Figure 12 – GAMESS Workflow

The workflow shows 4 separate executions of GAMESS for each parameter set. The Parameter Sweep actor computes all combinations of the A, B, C and D, and builds a record for each of these combinations. They are used as inputs to the GAMESS actors, and the outputs are sent into an actor that computes the Root Mean Square error of the cost function. This is a measure of how well a particular pseudo potential surface fits calculations of a real molecule. The results are re-ordered and plotted using a Graph actor.

In the experiment we executed the workflow with three different Directors – SDF, PN and TDA. We highlight the simplicity of changing the execution semantics – all we had to do to change them was to swap out one director on the Vergil canvass and replace it with another one. Figure 13 shows the performance of the workflow under each of the Directors running on the testbed shown in Table 1. All compute resources have little communication latency (<1ms) and the data sets were very small and such had a negligible affect on the experiment time. The SDF director only executed one job at a time, as expected, and took a total of 15:34:20. The PN director was able to exploit the static parallelism in the workflow, and executed up to 4 jobs at a time, taking 05:18:38. The TDA director was able to execute as many jobs together as there were processors available, and took a total of 00:9:45. Thus, the PN ran approximately 3 times faster than the SDF, and the TDA ran nearly 96 times faster than the SDF. The reason the PN times are not exactly 4 times faster is because the director waits on all 4 GAMESS-Nimrods to finish before submitting the next 4, and since each of them takes a different amount of time, time is lost in synchronising them. The graph in Figure 13 shows the number of jobs running at any time. The top trace shows that the TDA Director peaks at 352 jobs, which is the maximum number of available processors on the testbed at the time. The variation in the number of processors is because there is a variation in job execution times. Thus, whilst the shorter jobs had completed by approximately 4 minutes, a smaller number of longer jobs continued to run for the remainder of the time. This behaviour meant that the speedup was less that the peak number of processors. The bottom trace shows the execution profile for the SDF and PN directors. It is clear that the SDF director can only utilize one processor, and the PN uses 4.

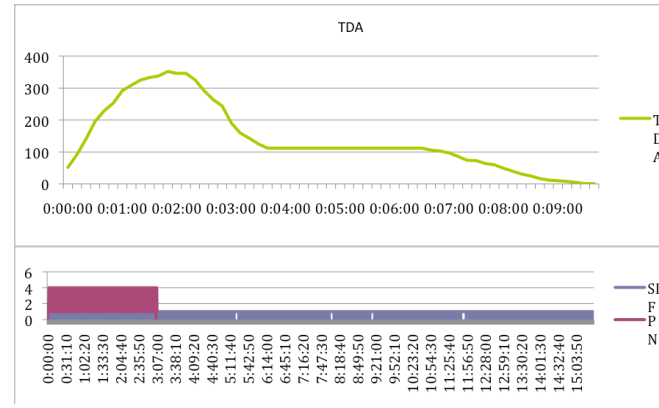

Figure 13 – performance results – TDA top and PN/SDF bottom.

5.3 Nimrod/OK Experiment: Case Study 2

This case study performs an optimization run on the cardiac model discussed in Section 5.1.2 to tune its parameters. After adding the new channel and metabolic factors for both normal condition and ischemia, we confirm the results against physiological data. We then stabilize the model for a heart rate of 0.5Hz and a longer performance duration (3 min). In both, the focus is on calcium transient concentrations, and action potential shape and duration, to determine the stability and accuracy of our model. In order to validate the model, some final outputs are compared against experimentally known data [37][48]. This involved exploring a range of numbers for nine input metabolic constants and selecting the combination that produces the closest output to the desired value.

This experiment performed simplex searches over the nine-dimensional space, using the workflow shown by Figure 14.

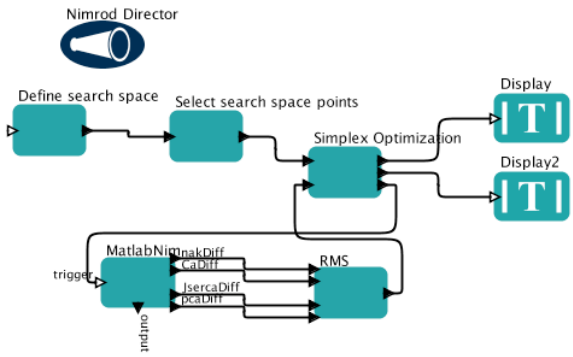

Figure 14 – Optimization run in Nimrod/OK.

128 starting points are randomly selected within the domain. Table 2 summarises the results of these searches, showing the best, worst and average search results. We also show the number of jobs and batches required to achieve the best and worst results, and the average number of jobs and batches. The best search gives a considerably better (smaller) result than the average, at the expense of more computational effort as shown by the numbers of jobs and batches of jobs. This shows the utility of multiple searches.

Job executions averaged 3m 21s, but with substantial parallelism the full experiment was completed in 7h 37m. Figure 15 plots the number of jobs executing over the duration of the experiment. Initially the simplex method evaluates the objective at the vertices of the starting simplex. In this case each of the simplexes has ten such vertices, so the experiment began with 1280 jobs. Thereafter, most iterations require just four new evaluations, so the load dropped to 512 jobs, except for the occasional peak where a new simplex was required. As searches completed the load dropped in stages; the final longest running optimization required only four processors but dominated the experiment wall clock time.

Index	Objective	Jobs	Batches
Min	0.00042	154	37
Average	0.0015	142	35
Worst	0.0038	34	7

Table 2 – results for searches over the full domain

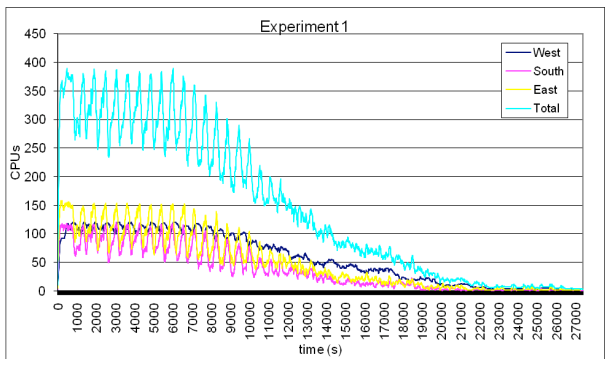

Figure 15 – job concurrency for full domain search

5.4 Nimrod/EK Experiment: Case Study 3

The main goal of this experiment was to examine the changes in which total ionic flux(s) in the compartment of interest may play the largest role in determining specific model outputs (action potential (AP) and Ca transients in the four cell sub-domains) in rabbit ventricular cardiac cells. We use Nimrod/EK to perform a partial parameter sweep and measure the effects of each of the parameter combinations, as summarised by the workflow in Figure 16. This workflow shows how the new actors can be combined to both explore the design space and present the results to the user by a range of different plotting actors.

In agreement with experiment [48], results suggest that small changes in L-type calcium and sodium/calcium exchanger total fluxes in the junctional cleft may significantly affect the cell function. Specifically, Figure 17 shows a Lenth plot for the AP duration. Here, there are few combinations that actually exhibit any significant effect because most of them are clustered around the X axis. This is reenforced in the Daniel plot in Figure 18, in which significant combinations appear off the line, and those on the line are insignificant. The same effects are apparent in the cytosolic Ca-peak in Figures 19 and 20. Surprisingly, the studies suggested that small changes in the junctional chloride/calcium flux also may have prominent effect on the model outputs. New experiments need to be perfomed to test this hypothesis. The fact that both the action potential duration and cytosolic calcium peak are impacted by the same fluxes distributions strengthen further our findings.

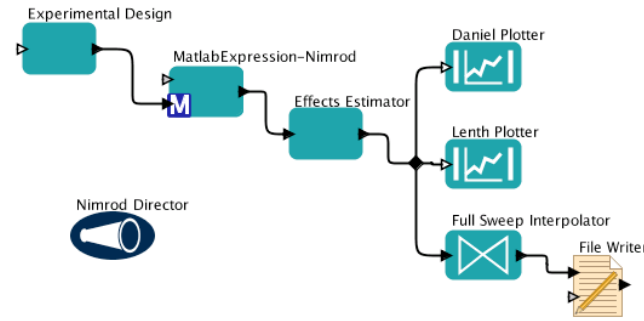

Figure 16 – workflow using Nimrod/EK

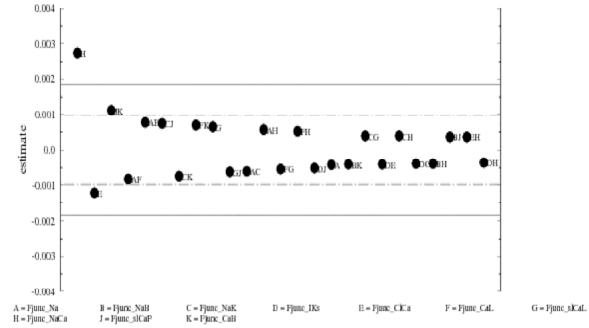

Fig. 17 Lenth plot for AP duration

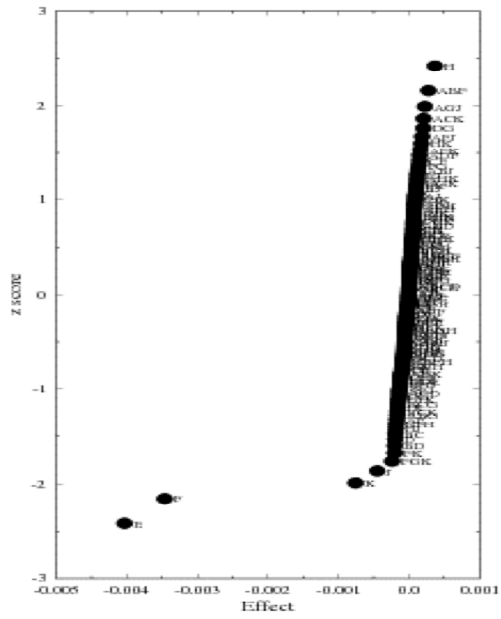

Fig. 18 Daniel plot for AP duration comparison

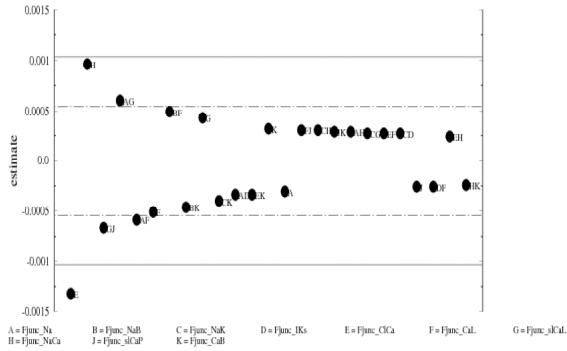

Fig. 19 Lenth plot for cytosolic Ca-peak comparison

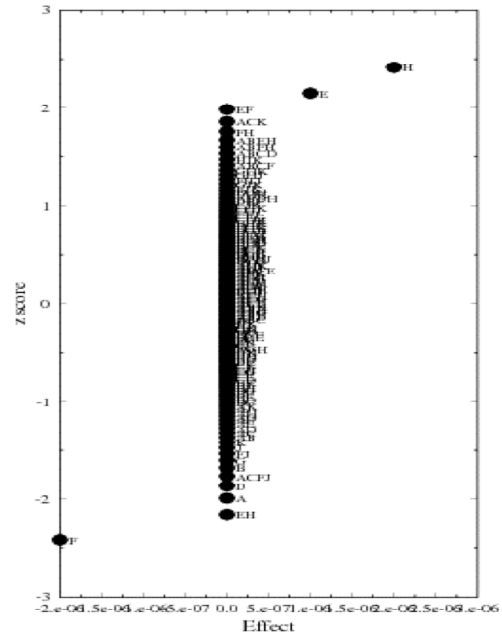

Fig. 20 Daniel plot for cytosolic Ca-peak comparison

CONCLUSIONS

In this paper we have shown it is possible to perform parameter exploration with workflow tools, and have discussed the addition of two key new ideas. First, we need to be able to run workflows using the MTC paradigm, and have shown that the tagged-dataflow architecture can achieve this. The scheme is based on mechanisms that were originally designed for parallel dataflow computers. When used in this mode, they allow a workflow to dynamically spawn threads of computation transparently from the workflow design, and this greatly simplifies the work for the user over existing workflow systems. Second, we have added new actors for specifying various types of parameter exploration, ranging from complete enumeration to optimization and fractional factorial design. We have discussed the implementation of these ideas in a prototype built on top of the Kepler workflow engine, called Nimrod/K.

We have demonstrated the applicability of Nimrod/K by various case studies in quantum chemistry and cardiac science. These have shown that we can explore a large parameter space using a complete sweep within Nimrod/K; solve a complex inverse problem framed as an optimization problem using Nimrod/OK; and explore role of parameter combinations using Nimrod/EK.

Importantly, Nimrod/K is an active and ongoing research project. We are currently exploring a number of research themes using this as a base. First, we are building a data transport framework that removes the need for users to explicitly manage data staging and movement in their workflows. When combined with the Nimrod/K engine, this will provide a very powerful distributed computing platform. Second, we are designing and building a range of scheduling heuristics that work with the data transport and parallel computing models discussed [10][49]. Third, we are building interfaces with powerful display technologies, and these allow a user to be immersed in the optimization process.

ACKNOWLEDGMENT

This project is supported by the Australian Research Council under the Discovery grant scheme We acknowledge our colleagues in the MeSsAGE Lab at Monash University; Saleh Amirriazi, Anuska Michailova and Ramya Chitters from the Cardiac Mechanics Lab at UCSD; and Wibke Sudholt from the Baldridge group at the University of Zurich. We also acknowledge the US NSF funded PRIME project, (supporting Amirriazi and Chitters) and thank colleagues, in particular Peter Arzberger, for their strong support.

REFERENCES

[1] Abramson D, Lewis A, Peachey T, Fletcher, C., "An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics", SuperComputing 2001, Denver, Nov 2001

[2] Abramson, D and Egan, G, "The RMIT Dataflow Computer: A Hybrid Architecture", The Computer Journal, Vol 33, No 3, June 1990, pp 230 - 240.

[3] Abramson, D., Bethwaite, B., Enticott, C., Garic, Slavisa and Peachey, T. "Parameter Space Exploration using Scientific workflows", ICCS 2009, SU Center for Computation & Technology, Baton Rouge, Louisiana, U.S.A, May 25-27, 2009.

[4] Abramson, D., Enticott, C and Altinas, I. "Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows", IEEE Supercomputing 2008, Austin, Texas, November 2008.

[5] Abramson, D., Giddy J., and Kotler, L. "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid," In Int'l. Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico, May 2000. http://www.csse.monash.edu.au/~davida/nimrod/.

[6] Altintas, I. Berkley, C. Jaeger, E. Jones, M. Ludäscher B. and Mock, S. "Kepler: Towards a Grid-Enabled System for Scientific Workflows," in the Workflow in Grid Systems Workshop in GGF10 - The 10th Global Grid Forum, Berlin, March 2004.

[7] Altintas, I., Birnbaum, A., Baldridge, K., Sudholt, W., Miller, M., Amoreira, C., Potier Y. and Ludaescher, B. "A Framework for the Design and Reuse of Grid Workflows" Intl. Workshop on Scientific Applications on Grid Computing (SAG'04), LNCS 3458, Springer, 2005

[8] Amirriazi S., Chang S., Peachey T., Abramson D. and Michailova A., Optimizing Cardiac Excitation-Metabolic Model By Using Parallel Grid Computing, Biophysics 2008, Long Beach, California, February 2008

[9] Arvind, and Nikhil R.S. "Executing a Program on the MIT Tagged-Token Dataflow Architecture", IEEE Transactions on Computers, Vol. 39, No. 3 (1990)

[10] Ayyub, S. and Abramson, D. "GridRod - A Service Oriented Dynamic Runtime Scheduler for Grid Workflows". 21st ACM International Conference on Supercomputing, June 16-20, 2007, Seattle.

[11] Ayyub, S., Abramson, D., Enticott, C., Garic, S., Tan, J. "Fault-tolerant execution of large parameter sweep applications across multiple VOs with storage constraints", Concurrency and Computation: Practice and Experience. Currently Online, 25/8/2008.

[12] Bethwaite, B, Abramson, D and Buckle, A. "Grid Interoperability: An Experiment in Bridging Grid Islands", PRAGMA Workshop at e-Science, 2008, 4th IEEE International Conference on e-Science, Indiana, Dec 8th – 12th, 2008.

[13] Box, G. E., Hunter W. G. and Hunter, J. S. "Statistics for Experimenters", Wiley, 1978.

[14] Buyya, R. and Abramson, D. "The Nimrod/G Grid Resource Broker for Economic-based Scheduling", in Market-Oriented Grid and Utility Computing (Wiley Series on Parallel and Distributed Computing), Editors Rajkumar Buyya and Kris Bubendorfer, ISBN-13: 978-0470287682, November 2009.

[15] Casanova H. and Berman, F. "Parameter Sweeps on The Grid With APST", chapter 26. Wiley Publisher, Inc., 2002. F. Berman, G. Fox, and T. Hey, editors.

[16] Culler. D. "Managing Parallelism and Resources in Scientific Dataflow Programs", PhD thesis, Department of Electrical Engineering and Computer Science,Massachusetts Institute of Technology, March 1990.

[17] Deelman E., Blythe J., Gil Y., Kesselman C., Mehta G., Patil S., Su M, Vahi K., Livny M., "Pegasus : Mapping Scientific Workflows onto the Grid", Across Grids Conference 2004, Nicosia, Cyprus.

[18] Deelman E., Blythe J., Gil Y., Kesselman C., Mehta G., Patil S., Su M, Vahi K., Livny M., "Pegasus : Mapping Scientific Workflows onto the Grid", Across Grids Conference 2004, Nicosia, Cyprus

[19] e-Science Grid Environments Workshop, e-Science Institute, Edinburgh, Scotland, May 2004, http://www.nesc.ac.uk/esi/events/.

[20] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., "Taming Heterogeneity---the Ptolemy Approach," Proceedings of the IEEE, v.91, No. 2, January 2003.

[21] Fahringer, T. Jugravu, A. Pllana, S. Prodan, R. Jr, C. S. and Truong H. L., "ASKALON: a tool set for cluster and Grid computing", Concurrency and Computation: Practice and Experience, 17:143-169, Wiley InterScience, 2005.

[22] Fang, K.-T., Li, R. and Sudjianto A., "Design and Modelling for Computer Experiments", Chapman and Hall/CRC, 2006.

[23] Fisher, R. A. "The Design of Experiments", Oliver and Boyd, 1935.

[24] Foster I. and Kesselman, C. "Globus: A Metacomputing Infrastructure Toolkit," Int'l J. of Supercomputer Applications, vol. 11, no. 2, 1997, pp. 115-128.

[25] Foster, I., "Globus Toolkit Version 4: Software for Service-Oriented Systems," Conference on Network and Parallel Computing, 2005.

[26] Goderis, A., Brooks, C., Altintas, I., Lee, E., Goble. C. "Composing Different Models of Computation in Kepler and Ptolemy II". 2007 Proceedings, International Conference on Computational Science (ICCS), April, 2007.

[27] Gurd J.R. and Watson I. "Data Driven System for High Speed Parallel Computing (1 & 2) Computer Design, vol.9 nos.6 & 7, June & July 1980, pp. 91-100 & 97-106.

[28] http://kepler-project.org

[29] http://messagelab.monash.edu.au/EScienceApplications

[30] http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm

[31] http://taverna.sourceforge.net

[32] http://www.lmsfrance.fr/LMS-Releases-OPTIMUS-22-For-Automated-Design-Space-Exploration

[33] http://www.mathworks.co.uk/access/helpdesk/help/toolbox/sldo/ug/brzapme-1.html

[34] Lee, E. et al, "Overview of the Ptolemy Project," Technical Memorandum UCB/ERL M01/11, University of California, Berkeley, March 6, 2001.

[35] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao J. and Zhao, Y. "Scientific Workflow Management and the Kepler System", Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, 2005.

[36] Lynch, A. H., D. Abramson, K. J. Beringer, and P. Uotila (2007), Influence of savanna fire on Australian monsoon season precipitation and circulation as simulated using a distributed computing environment, Geophys. Res. Lett., 34, L20801, doi:10.1029/2007GL030879.

[37] Michailova A., Lorentz W., McCulloch A., Modelling Transmural Heterogeneity of KATP current in Rabbit Ventricular Myocytes, AJP-Cell Physiology, 293 (2007), pp C542-C557.

[38] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., Li, P., Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics Journal, 20(17):3045–3054, 2004.

[39] Peachey, T. C., Diamond, N. T., Abramson, D. A. Sudholt, W., Michailova, A. Amirriazi, S., Fractional Factorial Design for Parameter Sweep Experiments using Nimrod/E, Journal of Scientific Programming, Volume 16, Numbers 2,3, 2008.

[40] Peachey, T., Abramson, D. and Lewis, A. Model Optimization and Parameter Estimation with Nimrod/O, The International Conference on Computational Science, May 28-31, 2006, University of Reading.

[41] Peachey, T., Abramson, D., Lewis, A., Kurniawan, D. and Jones, R. Optimization using Nimrod/O and its Application to Robust Mechanical Design, PPAM 2003, Czestochowa, Poland, Lecture Notes in Computer Science, Volume 3019 / 2004, pp. 730 – 737, September 7-10, 2003.

[42] Raicu, I., Foster, I., Zhao. Y. "Many-Task Computing for Grids and Supercomputers", IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08), 2008.

[43] Ravindran A., Ragsdell, K. M., Reklaitis G. V., "Engineering Optimization (Second Edition)", Wiley, 2006.

[44] Santner, T., Williams B. and Notz, W. "The Design and Analysis of Computer Experiments", Springer Verlag, New York, 2003.

[45] Schmidt, M. W., Baldridge, K. K., Boatz, J. A., Elbert, S. T., Gordon, M. S., Jensen, J. H., Koseki, S., Matsunaga, N., Nguyen, K., Su, S., Windus, T.L, Dupuis, M. and Montgomery, J. A. General atomic and molecular electronic structure system, J. Comput. Chem., v. 14, 1993, pp. 1347-1363;

[46] Schwefel, H. P. "Numerical Optimization of Computer Models,", New York : Wiley, 1981.

[47] Scientific Data Management Framework Workshop, Argonne National Labs, August 2003. http://sdm.lbl.gov/~arie/sdm/SDM.Framework.wshp.htm

[48] Shannon T. R., Wang, F. Puglisi, J. Weber C. and Bers, D. M. A Mathematical Treatment of Integrated Ca Dynamics within the Ventricular Myocyte, Biophysical Journal, 87 (2004), pp 3351-3371.

[49] Smanchat, S., Indrawan, M., Ling, S., Enticott, C. and Abramson, D. "Scheduling Multiple Parameter Sweep Workflow Instances on the Grid", IEEE e-Science 2009, Dec 9 – 11th, Oxford, UK.

[50] Sudholt, W., Baldridge, K. K., Abramson, D., Enticott C. and Garic, S. Applying Grid Computing to the Parameter Sweep of a Group Difference Pseudopotential, in Computational Science – ICCS 2004: 4th International Conference, Kraków, Poland, June 6-9, 2004, Lecture Notes in Computer Science, v. 3036, 2004, pp. 148-155.

[51] Sudholt, W., Baldridge, K. K., Abramson, D., Enticott C. and Garic, S. Parameter Scan of an Effective Group Difference Pseudopotential Using Grid Computing, New Generation Computing, v. 22, 2004, pp. 125-136.

[52] Tannenbaum, T. Wright, D. Miller, K. and Livny. M. "Condor - A Distributed Job Scheduler." Beowulf Cluster Computing with Linux, The MIT Press, MA, USA, 2002.

[53] Taylor, I. Shields, M. and Wang. I. "Resource Management of Triana P2P Services", Grid Resource Management, Kluwer, Netherlands, June 2003.

[54] von Laszewski, G. Amin, K. Hategan, M. Zaluzec, N. J. Hampton, S. and Rossi. A. "GridAnt: A Client-Controllable Grid Workflow System", In 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Big Island, Hawaii: IEEE CS Press, Los Alamitos, CA, USA, January 5-8, 2004.

[55] Xue G., Song W., Cox S.J., and Keane A.J., Numerical Optimisation as Grid Services for Engineering Design, Journal of Grid Computing, Vol.2 No.3, 2004, pp223-238.

[56] Yu J., and Buyya, R., A Taxonomy of Workflow Management Systems for Grid Computing", Journal of Grid Computing, Springer Press, New York, USA.

[57] Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Raicu, I., Stef-Praun, T. and Wilde, M. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation", IEEE Workshop on Scientific Workflows 2007.

David Abramson (Member IEEE CS) is a full Professor of Computer Science at Monash University Clayton Australia, and was department chair from 1997 to 2002. Before joining Monash University in 1997, he has held appointments at Griffth University, CSIRO, and RMIT. At CSIRO he was the program leader of the Division of Information Technology High Performance Computing Program, and was also an adjunct Associate Professor at RMIT in Melbourne. He has held senior management positions in the Co-operative Research Centre for Intelligent Decisions Systems and the Co-operative Research Centre for Enterprise Distributed Systems. He has chaired a number of international conferences, and has published over 120 papers and technical documents. He has given seminars and received awards around Australia and internationally and has received over $8 million in research grants. His current interests are in high performance computer systems design and software engineering tools for programming parallel and distributed supercomputers.

Blair Bethwaite is a research scientist and system administrator in the Monash eScience and Grid Engineering (MeSsAGE) Lab where he has worked since 2006. He holds a BCS from Monash University and is currently working towards an MIT. He has co/authored several publications in his time at MeSsAGE Lab and worked on a wide variety of software projects. His current research interests are in cloud computing and large scale distributed workflow systems.

Colin Enticott has been a research scientist with Monash eScience and Grid Engineering Laboratory (MeSsAGE Lab) working on Grid related experiments for 8 years. Colin has collaborated on eScience related projects varying across fields such as bio-chemistry, astrophysics and complex systems. For the last two years he has been working on his PhD which is related to Grid computing and workflow engines

Slavisa Garic is a research scientist in the Monash eScience and Grid Engineering Lab. He completed an Honours Degree in Computer Science at Monash University in 2002. Since then he has worked on the development of Nimrod/G toolkit and has been the core developer of the Nimrod back-end services within the MeSsAGE Lab. He has extensive experience in working with current Grid middleware toolkits as well as relational database systems such as PostgreSQL.

Tom Peachey has spent several decades working at various universities in Melbourne, Australia. He specializes in applicable mathematics, those parts of mathematics with immediate use in science, engineering and business. As a programmer he was lead developer in several major projects: a model of photo-voltaic installations that is used for sizing such in that industry, COPPS, a model of telecommunications traffic used in Australia and elsewhere for planning changes to telephony infrastructure, Hebat, a multimedia presentation language and interpreter used commercially in technical learning. Tom is currently Senior Reseach Fellow in the MeSsAGE Lab eScience research centre at Monash University.