Thesis Proposal:

Implementing Parallelisations in a Fuzzy Qualitative Reasoning Engine

Allan M. Bruce

e-mail: abruce@csd.abdn.ac.uk

March 25, 2005

Contents

List of Figures

List of Tables

1 Thesis

Qualitative Reasoning is an area of Artificial Intelligence which has been well researched [1–5]. Until recently QR implementations were developed sequentially and thus could not take advantages of multiple computers or processors. Platzner & Rinner [6–8] looked into increasing the efficiency of QSIM [2] - a popular QR system. They achieved this by porting QSIM to C but also noticed that stages of the *Qualitative Analysis* process benefited from being parallelised since the way that data was generated could be partitioned. This means that one data unit can be processed independently of all others hence can be run in a parallel manner. Their results were encouraging but there were a few drawbacks with the work. The parallelisations were only introduced for the two stages, and QSIM itself is limited dues to using only one derivative per variable. This document details the proposal that it is possible to parallelise the whole process using data partitioning and introducing novel algorithm parallelisations.

With the price of computer hardware falling and the development of faster internet communications, web services are becoming increasingly popular. A parallelised QR system will be able to benefit from this new technology and thus could be used as a web service which shall allow QR to be more accessible.

2 Motivation

There are several domains in which QR has been used [9–13] but one which has not been explored as frequently is that of planning. Planning is thought to be an area that would be interesting to use as a domain for QR (such as the model-based planner, Excalibur [11]) and this was the original intention for the project. However, a suitable implementation of a Qualitative Reasoner was not available, so it was decided to implement one based on Morven (formerly known as Mycroft [4]). During background research it was discovered that Platzner & Rinner came up with an excellent idea of exploiting parallelisations and investigated using a dedicated parallel QSIM machine to speed up execution. They found that the Tuple-filter stage of Qualitative Analysis used data that allowed partitioning so that each unit could be processed independently of the others. This allows each process to be run in parallel thereby achieving a speedup. They also describe parallelising the process of generating state data by partitioning the search space into smaller sub-searches. Their results were interesting but there were limitations with their system. The implementation was described as portable however their design required a dedicated hardware system consisting of several FPGAs and DSPs[1] acting as co-processors which were programmed to speed up execution of heavy calculations. This means that in order to run their system, this same machine must be used which is not desirable. Platzner & Rinner did not investigate parallelising the *Transition Analysis* phase which is required for simulation. QSIM allows qualitative behaviours to be constructed from models, however QSIM itself has limitations in that it only reasons with one derivative per variable and this derivative can only be decreasing, steady or increasing. Also, QSIM uses crisp symbolic quantity spaces, using fuzzy quantities allows behaviours to be simulated in which quantities can be represented with different precision and uncertainty. Overcoming these limitations is the main motivation for developing a novel architecture for a new

[1] Texas Instruments TMS320C40

Operation	Result	Conditions
$-n$	$(-d, -c, \delta, \gamma)$	all n
$\frac{1}{n}$	$\left(\frac{1}{d}, \frac{1}{c}, \frac{\delta}{d(d+\delta)}, \frac{\gamma}{c(c-\gamma)}\right)$	$n >_0 0, n <_0 0$
$m + n$	$(a + c, b + d, \tau + \gamma, \beta + \delta)$	all m, n
$m - n$	$(a - d, b - c, \tau + \delta, \beta + \gamma)$	all m, n
$m \times n$	$(ac, bd, a\gamma + c\tau - \tau\gamma, b\delta + d\beta + \beta\delta)$	$m >_0 0, n >_0 0$
	$(ad, bc, d\tau - a\delta + \tau\delta, -b\gamma + c\beta - \beta\gamma)$	$m <_0 0, n >_0 0$
	$(bc, ad, b\gamma - c\beta + \beta\gamma, -d\tau + a\delta - \tau\delta)$	$m >_0 0, n <_0 0$
	$bd, ac, -b\delta - d\beta - \beta\delta, -a\gamma - c\tau + \tau\gamma)$	$m <_0 0, n <_0 0$
	$m = [a, b, \tau, \beta], n = [c, d, \gamma, \delta]$	

Table 1: Arithmetic primitives used in FuSim

QR implementation.

Having a parallel QR package has several advantages. The execution time should be reduced, therefore allowing larger problems to be solved in less time which in turn should make QR more applicable to an increased number of problems. Making an abstract architecture which would be portable and scalable would also allow web services or similar technologies to be used thus making JMorven available for use in other systems, e.g. diagnosis, planning and learning. Doing so would also make QR more available and would potentially make it a more popular tool.

Implementing parallelisations in a QR system is not a trivial task, especially when trying to make the system portable and scalable. As mentioned earlier, Platzner & Rinner used data partitioning to implement parallelisations however this is not possible with all stages in the QR process. The problem is then to develop novel algorithms which allow the calculations to be parallelised while conserving the integrity of shared data. Another problem to overcome is the amount of memory required to run a parallel implementation, especially when running on a single machine with multiple processors.

The main focus of the project has shifted to investigate how parallelisations can benefit a qualitative reasoning engine.

3 Related Research

The work undertaken so far is largely based on an existing Qualitative Reasoning package which in turn takes influences from a number of QR systems. These are detailed in the following subsections along with a brief discussion of some work already completed on parallelisations by other researchers.

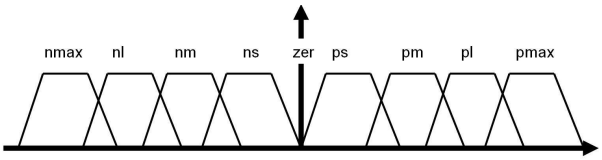

Figure 1: Fuzzy quantity space showing nine quantities.

3.1 QSIM

One of the most popular qualitative reasoning systems is Qualitative Simulation (QSIM) developed by Kuipers [2]. QSIM is a constraint based QR system using *Qualitative Differential Equations* to specify the constraints. Variables in QSIM are represented by a <*qmag, qdir*> pair where *qmag* denotes the qualitative magnitude of the variable which consists of either a landmark value or an interval within the given range. The rate of change of the variable is expressed by *qdir* which can take one of the three values, *inc* for increasing, *dec* for decreasing or *std* for steady. The *Constraint-filter* is used to ensure the qualitative states created are consistent with the constraints. A pairwise Waltz-filter [14] is used to ensure the model is consistent across all constraints which discards any tuples conflicting across any pair of constraints. To allow QSIM to simulate a model's behaviour over time, transition rules are used describing how variables transit depending on their magnitude and direction.

3.2 FuSim

QSIM was the main inspiration for the development of a Fuzzy Qualitative Simulation package called FuSim [3]. Fuzzy reasoning deals with uncertainty whereas QR deals with imprecision therefore combining the two approaches is thought to increase the scope of such a system. FuSim represents fuzzy numbers as paramaterised four tuples as detailed below:

$$\mu_a(x) = \begin{cases} 0 & x < a - \alpha \\ \alpha^{-1}(x - a + \alpha) & x \in [a - \alpha \quad a] \\ 1 & x \in [a \quad b] \\ \beta^{-1}(b + \beta - x) & x \in [b \quad b + \beta] \\ 0 & x > b + \beta \end{cases}$$

These fuzzy four tuples are used to create *Fuzzy Quantity Spaces*. A quantity space in FuSim is a set of overlapping four tuples which span a finite range as shown in figure 1. An α-cut is used in FuSim to aid simulation - this is where a fuzzy quantity space is converted to a non-overlapping crisp quantity space by selecting a 'typical' membership value, α.

To use fuzzy four tuples the standard arithmetic operators have been defined as shown in table 1. Once these arithmetic operators have been applied to fuzzy numbers, the resulting propagated value needs to be mapped back

to a relevant quantity space (the predicted values). The *Approximation Principle* is used to do this, which merely states that any quantities in the quantity space overlapping the propagated fuzzy value are an approximation to it. Obviously some values may be more suited to this approximation, so a *distance metric* is used to determine how close the approximation is for a given propagated value mapped back into the quantity space. This technique also allows the prioritisation of quantities which is utilised during simulation.

FuSim uses a similar variable representation to QSIM. However the derivative is not restricted to three values as in QSIM, it can instead take on any value in the quantity space specified (including a specific one just for that derivative if desired). This allows the model to be analysed more precisely over time and also allows FuSim to make temporal calculations. Due to the finite number of quantities in each quantity space, FuSim creates a state to describe the model at a given time. A behaviour is described as a set of these states in a tree with each node representing a valid state and each edge a valid transition.

FuSim, like QSIM, is a non-constructive QR system, i.e. the algorithm uses the transition rules to determine the set of successor values and then filters these with the constraints and the pair-wise filter.

3.3 Morven

Morven [4], formerly known as Mycroft, is a qualitative reasoning framework built on ideas developed in FuSim and adding several novel features. Morven uses a constructive approach to qualitative analysis which lends itself better toward simulation and helps reduce spurious behaviour generation. Due to its being constructive, Morven requires that constraints are causally ordered. This is where a constraining variable must not appear before it has been constrained in the ordering.

One major limit of QSIM and FuSim was the use of only one derivative per variable. Morgan [15] introduced the concept of *Vector Envisionment* [16] which allows a non-fixed number of derivatives to be used, including reasoning purely with the magnitude of a variable. Morven built on this and introduced *Fuzzy Vector Envisionment* [4] which reasons about a non-fixed number of fuzzy derivatives. With the inclusion of multiple derivatives, a system has to incorporate a method to be able to constrain these extra derivatives. *Differential Planes* [17] are used to add these extra constraints. Differential planes also have the advantage that the model complexity may be reduced for higher derivatives if the extra detail is not required.

3.4 Parallel QSIM

One disadvantage of Qualitative Reasoning is that current implementations are not very efficient [6] and can take a long time to analyse the behaviour of complex models. Platzner and Rinner decided to try and optimise the popular package QSIM to make it more efficient and therefore appeal to a wider audience. QSIM was originally developed in LISP so their first optimisation achieved was when porting QSIM to C. They found that this typically decreased execution time by approximately three to four times for models running on the C version over the LISP version on the same hardware setup. These results were encouraging so they sought more optimisations.

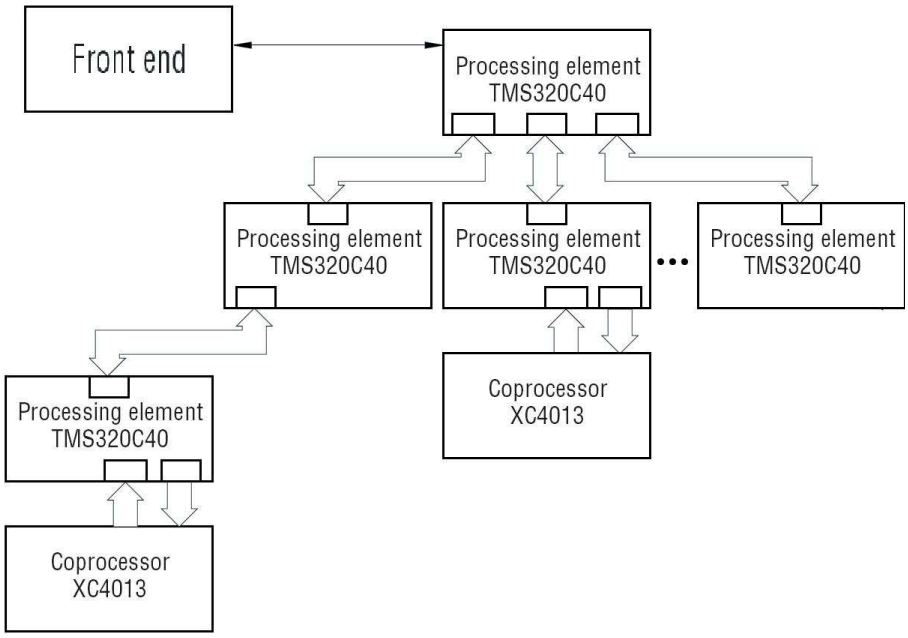

Figure 2: The Architecture of Parallel QSIM.

The Constraint-filter iterates through all tuples from the constraints and only consistent and valid tuples remain. Each constraint is considered in turn and for every possible combination of values a variable may take, the Tuple-filter checks for consistency with the single constraint. If a combination is found to be inconsistent that tuple is discarded, however if the tuple is found to be consistent then the Tuple-filter does not discard it. The Tuple-filter and the Waltz-filter together make up the Constraint-filter.

The form-all-states stage takes all valid tuples from the constraint filter and generates all possible unique states for the set of constraints.

Platzner and Rinner decided to use a dedicated hardware machine which would execute the large number of instructions more quickly than a standard processor. During the design and implementation of the tuple filter, they found that the tuples could be filtered independently of each other. This would mean that all tuples could be constrained in their own parallel stage which would speed up execution greatly. The Tuple-filter was redesigned for the dedicated parallel hardware platform using up to seven DSP co-processors as shown in figure 2 [6].

The form-all-states stage was also parallelised, but using a different technique which partitions the search space into smaller sub-searches.

The results were positive offering a good performance increase. As mentioned earlier, the basic C implementation was approximately three to four times faster than the original LISP version. While using the dedicated hardware parallel system, a further speed increase of up to five times was observed (when using seven processors). These results were very good, although there are a few drawbacks with their design: The implementation was suited only for a dedicated hardware system using these DSP chips, which limits the user base considerably. Also, the implementation was only tested on up to seven parallel units therefore a speedup

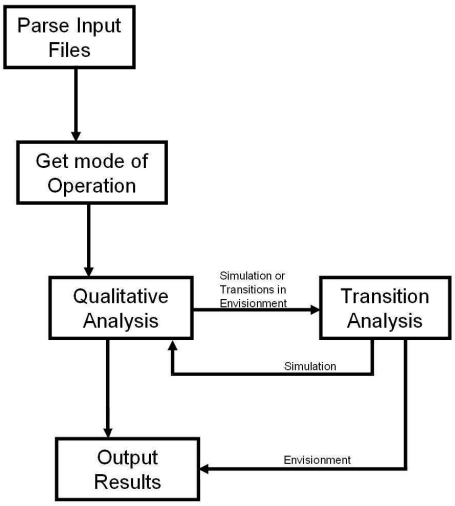

Figure 3: Flow Chart of JMorven.

model cannot be obtained to show the benefits of parallelisation on a large scale. As mentioned in section 2, QSIM has the limitations of only reasoning about one derivative and uses crisp quantities. A new parallel QR engine was designed and implemented to overcome these limitations.

4 Approach

JMorven is a Java implementation of a Qualitative Reasoning engine called Morven and completely re-written with a novel parallel architecture. The new architecture is abstract and scalable allowing JMorven to make use of multi-threaded machines, multi-processor systems or to run in distributed computing environments. Being written in Java, JMorven is also very portable which increases the potential number of users. The design allows the number of parallel units to be specified at run-time making optimal use of the resources available. During the design of JMorven, all stages of Qualitative Analysis have been identified as parallelisable. These are detailed below. Figure 3 shows a flow diagram of how JMorven is executed and figure 4 shows a detailed diagram of the parallel architecture of JMorven.

JMorven differs from its predecessor in that it uses a non-constructive algorithm, thus it uses a similar process to QSIM and FuSim during Qualitative Analysis.

As with Morven, JMorven uses Qualitative Differential Equations across Differential Planes to specify models. For example, the following differential equation

$$V' = q_i - q_o$$

is specified as follows for use with JMorven:

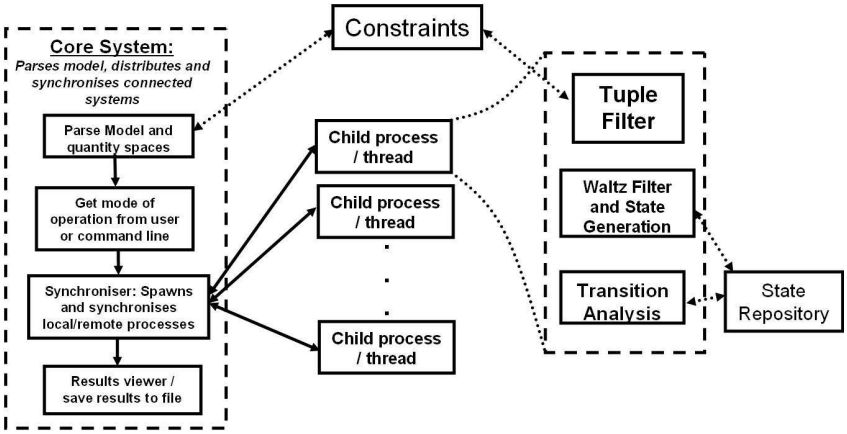

Figure 4: The JMorven Architecture.

Constraint: sub (dt 1 V) (dt 0 qi) (dt 0 q0)

The first keyword after 'Constraint:' specifies the type of constraint. The subtraction constraint requires three variables to be specified - the result, the variable to subtract from and the variable to subtract. Variables are specified by (dt DERIV VARNAME) where DERIV specifies the order of the derivative for the variable (zero denotes the magnitude of the variable) and VARNAME specifies which variable is in the constraint. All constraints are specified per differential plane allowing the detail of higher order derivatives to be reduced for speed of execution. Since JMorven uses *Qualitative Vectors* (which allows a variable number for the maximum order of derivative for a variable), the number of the derivatives in the constraint are not restricted, however every derivative must be constrained from zero to the maximum order specified.

JMorven also uses Fuzzy Quantity Spaces to specify the fuzzy quantities within the system. An example quantity space would be specified in JMorven as follows:

n-max	-1	-1	0	0.1
n-large	-0.9	-0.75	0.05	0.15
n-medium	-0.6	-0.4	0.1	0.1
n-small	-0.25	-0.15	0.1	0.15
zero	0	0	0	0
p-small	0.15	0.25	0.15	0.1
p-medium	0.4	0.6	0.1	0.1
p-large	0.75	0.9	0.15	0.05
p-max	1	1	0.1	0

Quantities are specified by the following: QNAME $a\ b\ \alpha\ \beta$, where a, b, α & β form the fuzzy four-tuple as detailed in section 3.2. JMorven also uses the Approximation Principle to map propagated values from the constraints back

into the relevant quantity space.

Details of how each stage of Qualitative Analysis has been parallelised so far are discussed in the following sub-sections along with a note about Transition Analysis.

4.1 Tuple-filter

The Tuple-filter was parallelised in a similar manner to that of Platzner and Rinner where each tuple can be filtered independently of the others. JMorven iterates through each constraint in turn obtaining a set of valid tuples from each. For example, for a single tank with an inflow and outflow, one of the constraints is:

$$V' = q_i - q_o$$

which states that the rate of change of the volume of water is equal to the difference between the inflow and outflow. This constraint only reasons about the first derivative of V and the zeroth derivatives of q_i and q_o. A valid tuple from this constraint may include:

[V' q_i q_o] = [*p-small p-large p-medium*]

A number of valid tuples will be produced for each constraint in the model. Since each constraint can be filtered independently, they can be executed in their own parallel unit. JMorven creates a new thread for each constraint. If there are more constraints than the maximum number of available threads, JMorven queues constraints until a thread becomes available. A diagram of how the Tuple-filter is parallelised is shown in figure 5.

4.2 Waltz-filter

After the Tuple-filter there are a large set of constraint-tuples, and each constraint-tuple is a vector of valid fuzzy quantities similar to the one shown above. To ensure that the tuples are consistent over all constraints, a Waltz-filter is used. which involves pairing all possible constraint-pairs that are adjacent (two constraints are said to be adjacent if they each share a common derivative of a variable). Each pairing then iterates through all possible tuples and discards those that are not common to both. For example, if we have constraints C1 and C2 from the single tank example with tuples as shown:

C1: [V' q_i q_o] = [*p-small p-large p-medium*]
C2: [V q_o] = [*p-medium p-medium*]

where C2 is the one to one mapping representing a monotonic increasing function between V and q_o. The only common variable to both of these constraints is q_o which is consistent for the given values (*p-medium* in both constraints) therefore the filter would keep this pair, however if C2 = [*p-large p-large*] then the pair would be discarded as q_o would be inconsistent across the pair of constraints.

Each pair of constraints can be executed independently of each other, therefore can be parallelised. One way to implement the Waltz-filter in parallel is to first create an exhaustive list of all possible pairs of constraints for

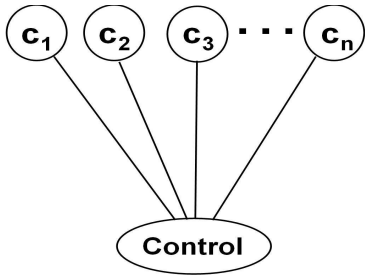

Figure 5: The Tuple Filter in parallel.
C_x denotes each constraint in the system which is filtered in its own thread

the model, and then create a new thread for each pair (queuing pairs if the maximum number of threads has been reached until one becomes free as before). JMorven incorporates the Waltz-filter in the State-generation stage as explained in the next section.

4.3 State-generator

The State-generator is the most computationally expensive stage of Qualitative Analysis. This is the process of iterating through each set of tuples and creating unique states for every combination of variables' derivatives possible. JMorven combines the Waltz-filter described above within this stage to optimise performance. To create these unique states, JMorven uses a recursive technique to enumerate the constraints. All of the tuples within a constraint are considered in turn. If the Waltz-filter discards the tuple, the next tuple is considered. If there are no more tuples left in the current constraint the iteration stops and the end of this recursion is met allowing the previous constraint's iteration to continue. If and when the Waltz-filter finds a consistent tuple, the next constraint is considered from its first tuple (unless the current constraint is the last one). If this is the last constraint and a tuple is consistent then a unique state is created. This procedure carries on until there are no more tuples left in the first constraint. The following pseudo-code shows the process described above:

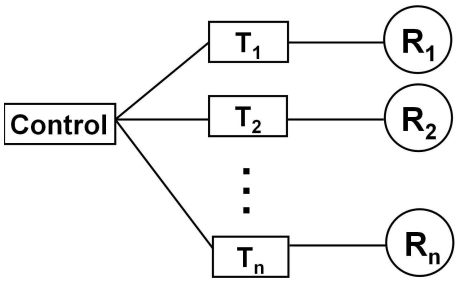

Figure 6: The State Generator in parallel.
T_x shows each thread which executes the Recursive function R_x

```
constraint c=0
function:  recurse(int c)
{
  iterate t tuples in constraint c
  {
    if tuple t is consistent
    {
      if c is last constraint
        create unique state
      else
        recurse(c+1) //next constraint
```

To parallelise this stage, the first recursive step is broken down into an iterative step, and each iteration is spawned in its own thread (note that all tuples in the first constraint are valid since no other constraints have been set, therefore there is no need to check the validity of these tuples). The iteration is shown below:

```
iterate i through tuples in c=0
{
  c = 1
  recurse(c)
}
```

This allows the state-generation to run in parallel as shown in figure 6. The first constraint is chosen to be the one which has the number of tuples closest to the number of available threads. It can be seen that a filter is included in the State-generation which negates the need for a separate Waltz-filter, leading to a decrease in execution time.

4.4 Transition Analysis

The Transition Analysis (TA) phase involves determining how qualitative states transit between one another. This is done by following *Transition Rules* which assume that all transitions are continuous [18]. A preliminary version of the TA phase has been implemented in JMorven in parallel however the benefits of the parallelisations are not yet apparent. Optimising these parallelisations will form part of future work still to be completed.

5 Deliverables

By the end of the project it is proposed that a fuzzy qualitative reasoning engine, called JMorven, will be fully developed using parallel optimsations in as many stages as possible. Parallelisations will be in the form of data partitioning where possible and novel algorithms will be used when this is not possible or when new algorithms will provide a greater benefit from parallelisations. JMorven will be implemented to take advantage of web services and GRID technology (such as the Globus Toolkit [19]). Web services provide a convenient and popular interface to applications and servers and distributed computing environments allow the sharing of resources to be accessed easily. These benefits should make JMorven more applicable to a wider array of problems and should become a more popular tool for analysing systems that can be modelled qualitatively.

6 Evaluation

To test JMorven and its performance benefits, a set of models will be tested and the results will be analysed against existing behaviours from its predecessor, Morven. Testing JMorven on multiprocessor machines and distributed computing environments will show the benefits of a parallel architecture and should show that this is a viable means of decreasing execution times for QR systems in the future. As a web service, JMorven will allow a means of testing QR on some example domains, e.g. planning or learning.

7 Work Completed So Far

So far, the main architecture for JMorven has been implemented and all stages have been successfully parallelised, however some of the benefits are not yet apparent in the Transition Analysis phase. The output from JMorven is consistent with its predecessor so assumed to be correct. The benefits from parallelisations offer a good speed increase - approximately halving execution time when using four processors over one.

So far, JMorven only runs on single machines with one or more processors. JMorven was executed on a ten processor SUN server running Solaris 5.8 with Sun Java 1.4.2_03. Two test models were used. For testing the Tuple-filter a coupled tanks model (see fig. 7) with two inputs and two outputs was used as shown below (only the first differential plane):

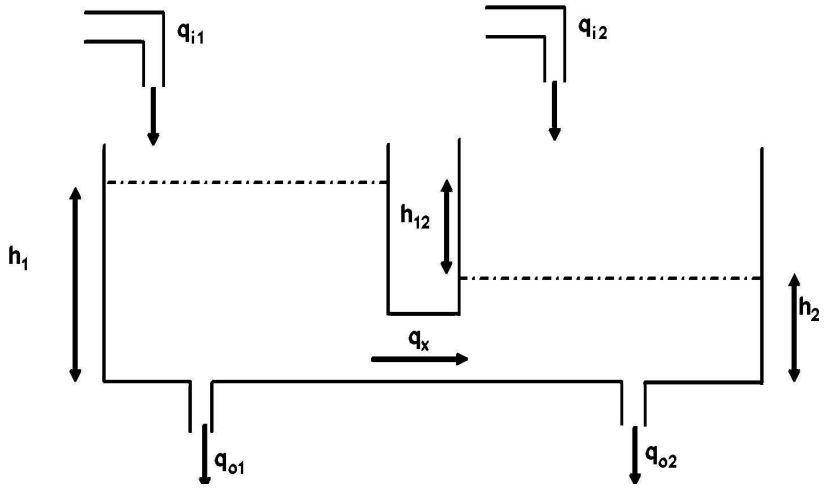

Figure 7: Coupled tanks model.

Two tanks of water with heights h_1, h_2 and there difference h_{12}. Two inflow taps q_{i1}, q_{i1} and two outflow plugs q_{i1}, q_{i1} determine the flow in and out of the tanks and the cross-flow q_x describes the flow between them.

Constraint: sub (dt 0 h12) (dt 0 h1) (dt 0 h2)
Constraint: func (dt 0 qx) (dt 0 h12)
Constraint: func (dt 0 qo2) (dt 0 h2)
Constraint: func (dt 0 qo1) (dt 0 h1)
Constraint: sub (dt 0 q1flow) (dt 0 qi1) (dt 0 qx)
Constraint: add (dt 0 q2flow) (dt 0 qi2) (dt 0 qx)
Constraint: sub (dt 1 h1) (dt 0 q1flow) (dt 0 qo1)
Constraint: sub (dt 1 h2) (dt 0 q2flow) (dt 0 qo2)

The *sub* and *add* constraints are organised by having the result in the first variable specified. The *func* constraint is a qualitative function where values can be mapped from the left variable to the right variable which allows many types of function to be implemented. For such models, these merely define the monotonic increasing function (M^+).

For testing the State-generator a coupled tanks model was also used, but with only one input (to tank A) and one output (from tank B) as described by the following constraints:

Constraint: func (dt 0 qo) (dt 0 h2)

Constraint: func (dt 0 qx) (dt 0 h12)

Constraint: sub (dt 0 h12) (dt 0 h1) (dt 0 h2)

Constraint: sub (dt 1 h1) (dt 0 qi) (dt 0 qx)

Constraint: sub (dt 1 h2) (dt 0 qx) (dt 0 qo)

The quantity spaces used for both consisted of nine fuzzy intervals. Each model was run ten times for each number of threads, and results show the speedup. The results of the Tuple-filter are shown in figure 8. It is clear to see that there is a benefit from the parallelisations. There was quite a large error in times recorded for the Tuple-filter - this is due to the very small execution time. The Tuple-filter takes well under one second to complete for this model. Using six threads over one almost decreases the execution time by a factor of two, which is less than expected. This is probably due to the very small amount of time taken for the Tuple-filter. When the time taken is this small, the overhead of creating and killing threads becomes apparent. A much more complex model should show larger benefits from the parallelisations, and will form part of future work.

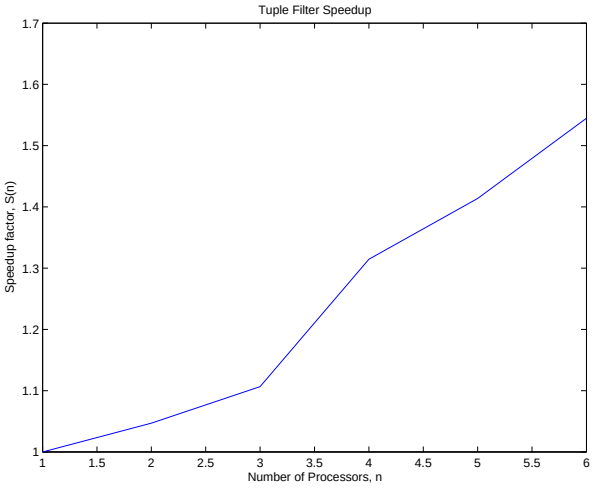

Figure 8: Execution times of the Tuple-Filter.
Shows speedup factor of Tuple-filter for multiple numbers of processors using a coupled tanks qualitative model with two inputs and two outputs.

The State-generator results are shown in figure 9. The benefit of the parallelisations is apparent for a smaller number of threads - using four threads over one almost halves the running time of the State-generator. This benefit is less obvious when using a larger number of threads. This is thought to be due to the model used since the State-Generator splits the first constraint into threads. The first constraint chosen may have only a few valid tuples when recursed to the next constraint. For example, if there are ten valid tuples in the first constraint (C0) then ten threads will be spawned and each thread starts the recursion in constraint C1. One of these threads might not have

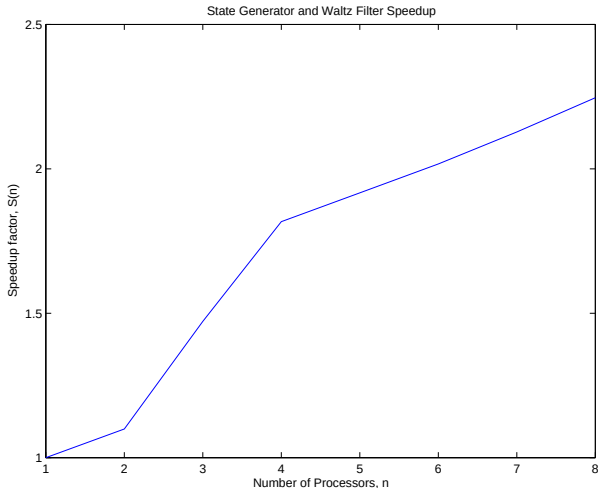

Figure 9: Execution times of the Waltz-filter and State-generator.
Shows speedup factor of State Generator and combined Waltz-filter for multiple processors of threads for a coupled tanks qualitative model with one input and one output.

any consistent tuples in C1 therefore would terminate after a very short time. However one of the other threads may have several valid tuples in the constraint C1 and would require recursion to the next constraint for each of them therefore this thread may take substantially longer to execute. This behaviour would not be as apparent when using models with a greater number of constraints. Another possibility for the benefits to be less apparent than expected might be due to the amount of semaphoring required to protect the data from corruption when accessing it from more than one thread simultaneously. For a distributed computing environment, semaphoring would not be required as all data would be copied for each process, which should allow more benefit from the parallelisations.

The parallelisations still have to be optimised - it is hoped that close to linear speedup [20] [2] will be observed when the project is completed. Although JMorven is functional, there are some advanced features which are in mid-development. These include the use of *Auxiliary Variables* and a distance metric. The use of auxiliary variables allows a variable to be used during constraint filtering but is not mapped backed to any quantity space, which helps reduce spurious state generation. A distance metric determines how close a calculated variable is to the quantity from the quantity space when mapped back using the approximation principle. This allows behaviours to be prioritised and can suggest more likely behaviours when simulating.

JMorven has not been implemented as a web service yet. It is proposed to implement JMorven as a web service, or use existing GRID technology.

[2] Linear speedup is where the execution time is decreased n times when run on n processors

8 Workplan

The last thirteen months of the project will be used to finish implementing the features discussed in section 7 and to writing up the final thesis. The following workplan shows how this time is intended to be broken down.

- Incorporate auxiliary variables in a parallel manner (1 month)
- Incorporate distance metric in a parallel manner (0.5 months)
- Parallelise Transition Analysis (1 month)
- Test more complex models and record execution times (0.5 months)
- Optimise parallelisations (1 month)
- Incorporate more advanced simulation engine (1.5 months)
- Create JMorven web-service (0.5 months)
- Test web-service with planning and learning (1 month)
- Writing up time (6 months)

References

[1] K. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, December 1984.

[2] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, September 1986.

[3] Q. Shen and R. Leitch. Fuzzy qualitative simulation. *IEEE Transactions on Systems, Man and Cybernetics*, 23(4):1038–1061, July-August 1993.

[4] G. M. Coghill. *Mycroft: A Framework for Constraint-based Fuzzy Qualitative Reasoning*. PhD thesis, Heriot-Watt University, September 1996.

[5] D. S. Weld and J. de Kleer. *Readings in Qualitative Reasoning about Physical Systems*, volume 1. Morgan Kaufmann Publishers, Inc., 1990.

[6] M. Platzner and B. Rinner. Parallel qualitative simulation. *Simulation Practice and Theory - International Journal of the Federation of European Simulation Societies*, 5(7-8):623–638, 1997.

[7] M. Platzner and B. Rinner. Toward embedded qualitative simulation. *IEEE Intelligent Systems*, 15(2):62–68, March-April 2000.

[8] M. Platzner and B. Rinner. Design and implementation of a parallel constraint satisfaction algorithm. *International Journal in Computers and Their Applications*, 5(2):106–116, June 1998.

[9] J. de Kleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):91–130, April 1987.

[10] G. M. Coghill, S. M. Garrett, and R. D. King. Learning qualitative metabolic models. In R Lopez de Mantaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 445–449.

[11] B. Drabble. Excalibur: A program for planning and reasoning with processes. *Artificial Intelligence*, 62(1):1–40, July 1993.

[12] U. E. Keller. *Qualitative Model Reference Adaptive Control*. PhD thesis, Heriott-Watt University, September 1999.

[13] K. de Koning, B. Bredeweg, J. Breuker, and B. Wielinga. Model-based reasoning about learner behaviour. *Artificial Intelligence*, 117:173–229, March 2000.

[14] D. Waltz. *Understanding Line Drawings of Scenes with Shadows*. McGraw-Hill, New York, 1975.

[15] A. Morgan. *Qualitative Behaviour of Dynamic Physical Systems.* PhD thesis, University of Cambridge, 1988.

[16] G. M. Coghill. Vector envisionment of compartmental systems. Master's thesis, University of Glasgow, April 1992.

[17] M. Wiegand. *Constructive Qualitative Simulation of Continuous Dynamic Systems*. PhD thesis, Heriot-Watt university, May 1991.

[18] F. Cellier. *Continuous System Modelling*. 1991.

[19] Globus. http://www.globus.org/, 2005.

[20] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation*. Oxford University Press, Oxford, UK, 1995.