5-1-2014

Variable Bounds Analysis of a Climate Model Using Software Verification Techniques

Peter Revesz
University of Nebraska-Lincoln, prevesz1@unl.edu

Robert Woodward
University of Nebraska–Lincoln, s-rwoodwa2@unl.edu

Variable Bounds Analysis of a Climate Model Using Software Verification Techniques

PETER REVESZ & ROBERT WOODWARD
University of Nebraska–Lincoln
Department of Computer Science & Engineering
Lincoln NE 68588-0115
USA
{revesz,rwoodwar}@cse.unl.edu

Abstract: Software verification techniques often use some approximation method that identifies the limits of the possible range of values that variables in a computer program can take during execution. Current climate models are complex computer programs that are typically iterated time-step by time-step to predict the next value of the climate-related variables. Since these iterative methods are necessarily computed only for a fixed number of iterations, they are unable to answer many long-range questions that may be posed regarding climate change, for example, whether there are natural fluctuations or whether a tipping point is reached after which there is no return to normal. In a departure from the usual step-by-step climate models, we propose to use software verification techniques to predict absolute bounds, i.e., predict maximum and minimum values that could ever arise according to a climate model. Hence we can answer some of the long-range climate prediction questions that were not possible to answer before.

Key–Words: Software Verification, Computing Invariant Values, Datalog, Climate Model

1 Introduction

Climate change has potentially a huge impact on life on earth because the daily operation of every living organism depends on weather and climate. Weather is the state of the atmosphere, defined in part by the temperature, humidity, wind, and precipitation, and can differ day-to-day from the result of changes in the atmosphere pressure or other phenomena. Climate, on the other hand, is the average of the weather, either by its mean or change over a time-period and area (e.g., season). Climate varies in both temporal and spatial dimensions. The variations are based on the location of the earth, altitude, or nearby geographical features nearby, such as bodies of water, and over time based on the position of the sun. Numerous studies link climate change to the increase of green house gases, including carbon dioxide and other contaminants, land-use change, and other natural and human-related factors. The ability to predict climate change is affecting the legislation of countries and their mitigation efforts around the world [1].

The predictions of the impacts of climate change rely heavily on the simulations of global climate models. Regional climate models offer a finer level of detail than the global climate models, and are sometimes used to determine the impact of climate on smaller regions. Climate models are calibrated using historical weather data. The model scenarios have been standardized by the *Intergovernmental Panel on Climate Change (IPCC)*. The IPCC was established in 1988 by the World Meteorological Organization and the United Nations Environment Programme.

In this paper, we study global climate models using the Third Assessment Report (TAR) of the IPCC [1] and a Fourth Assessment Report (AR4) which has succeeded TAR [2]. The model studied in this paper was originally based on TAR and later updated with information from AR4. The TAR and AR4 assessment reports are updated to include new information and research conducted between each assessment report.

Climate models are able to calculate the global average temperature above a baseline year. Chapter 9 of the TAR defines that climate change simulations are to be assessed over the period from 1990 to 2100. However, rather than simulate over a given bound, it is more interesting to have an absolute bound of change. The goal of this paper is to determine if there is an invariant value for the global average temperature change from a baseline using software verification techniques.

This paper is organized as follows. Section 2 gives some background information, including the Java Climate Model used in this paper. Section 3 describes the experimental design. Section 4 discusses the implications of the results. Section 5 summarizes related work. Finally, Section 6 gives some conclusions and future work.

ISBN: 978-960-474-381-0

2 Background

In this section, we give some background information on climate systems, climate models, and software verification.

2.1 Climate Systems

A climate system consists of five major components:

1. **Atmosphere**: The air and space surrounding the earth.
2. **Hydrosphere**: The water surrounding the earth.
3. **Cryosphere**: The parts of the earth where water is frozen.
4. **Land surface**: The parts of the earth covered by land.
5. **Biosphere**: The parts of the earth covered by living organisms.

Figure 1 shows the interactions of these five major components of the climate system. Each of these components is influenced by external forces, such as the sun or human activities. The components can also interact among themselves. For example, the biosphere affects the concentration of carbon dioxide in the atmosphere [1].

2.2 Climate Models

Climate models try to model the climate system and predict some values for the climate, such as the following:

- Land-surface temperature and land-surface air temperature.
- Sea-surface temperature and ocean air temperature.
- Land and sea combined temperature.
- Sub-surface ocean temperature.
- Upper air temperature.
- Snow cover, including snowfall.
- Sea-ice extent and thickness.

The IPCC's TAR outlines some of the ways to accurately predict the above values [1]. The *Wigley/Raper Upwelling-Diffusion Energy Balance Model (UD/EBM)* climate model is a simple climate model that models the differentiates the hemispheres, and the land and ocean regions in each hemisphere [3]. The model uses heat flux equations to model the transfer from one year to the next, and from one region to another. It is important to note that it must iteratively compute each year's value. It can *never* estimate a long-ranged value, which is the goal of this paper. This simple climate model must be tuned to simulate an *atmosphere-ocean coupled general circulation model (AOGCM)* and is in itself not a model [1].

The Java Climate Model (JCM)[1] implements UD/EBM and was properly tuned to match a AOGCM [4]. Rather than using direct integration to compute the values for the heat fluxes, the JCM uses an eigenvector calculation method. This method finds the exactly analytical solution, given the assumption that the non-linear fluxes change linearly within one time-step of a year [4]. Figure 2 shows the average temperature change that is estimated by the JCM until the year 2150.

The Java Climate Model (JCM) was downloaded from http://jcm.climatemodel.info/ (Version from June 2011). The SVN code repository for JCM was not operational, however, the source code was included inside of the distributed Java Archive (JAR) file. After extracting the source folders from the JAR file, a new project was created in NetBeans IDE 7.0.1

- substance.jar – included in the JCM JAR.
- lucdata.jar – included in the JCM JAR.
- labdoc.jar – included in the JCM JAR.
- match-emitdata.jar – included in the JCM JAR.
- JCM.jar – included in the JCM JAR.
- javaws.jar – included in the Jave Runtime Environment (JRE) library folder.
- Jama-1.0.2.jar – downloaded from `http://math.nist.gov/javanumerics/jama/`.

The file that calculates the global temperature at each step is 'udebclimod.java' inside of the 'adjust(rf)' method. The adjust method is called for each time step in the program. The initial values the adjust method are set up in the 'setupfluxes()' method. To obtain the initial values of the program, a breakpoint was set on the first line of the adjust method, and the program was started in debug mode. Once this line is reached for the first time, the initial values of all of the variables were determined through the debugger, and copied into the Datalog program. The adjust method is shown in Section **??**.

Note that the model seems to already have leveled off at a value around 2. However, there is no guarantee that the value is leveled off. Whether 2 is the maximum or some other value is a question that we will try to determine in this paper.

2.3 Software Verification

Verifying that a program functions correctly on a valid input is the goal of software verification. The focus of our discussion is on using constraint databases with linear constraints to verify the program state of a program. The program state is the values assigned to the variables in the program at a specific line of the program code [5].

Through the use of addition-bound matrices (ABM), we can find an over-approximation of the values a variable can take. Each variable x_i can be represented

[1] `http://jcm.climatemodel.info/`

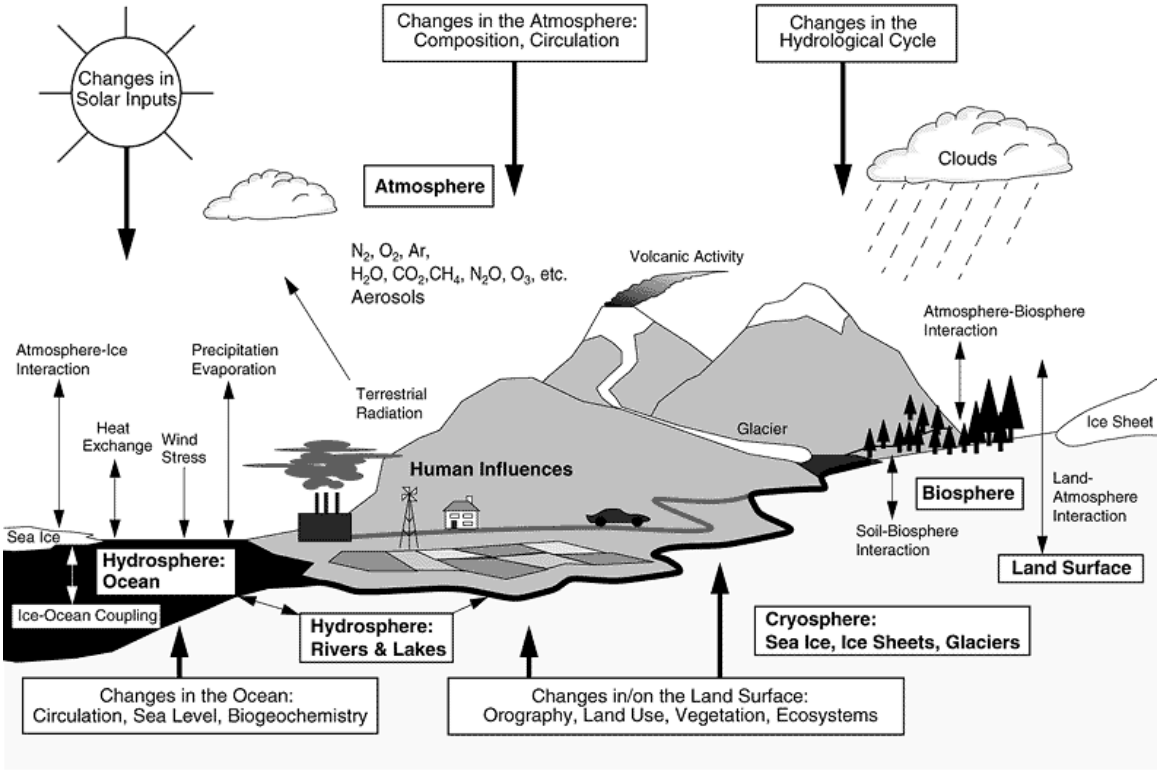

Figure 1: The climate system. Figure borrowed from [1].

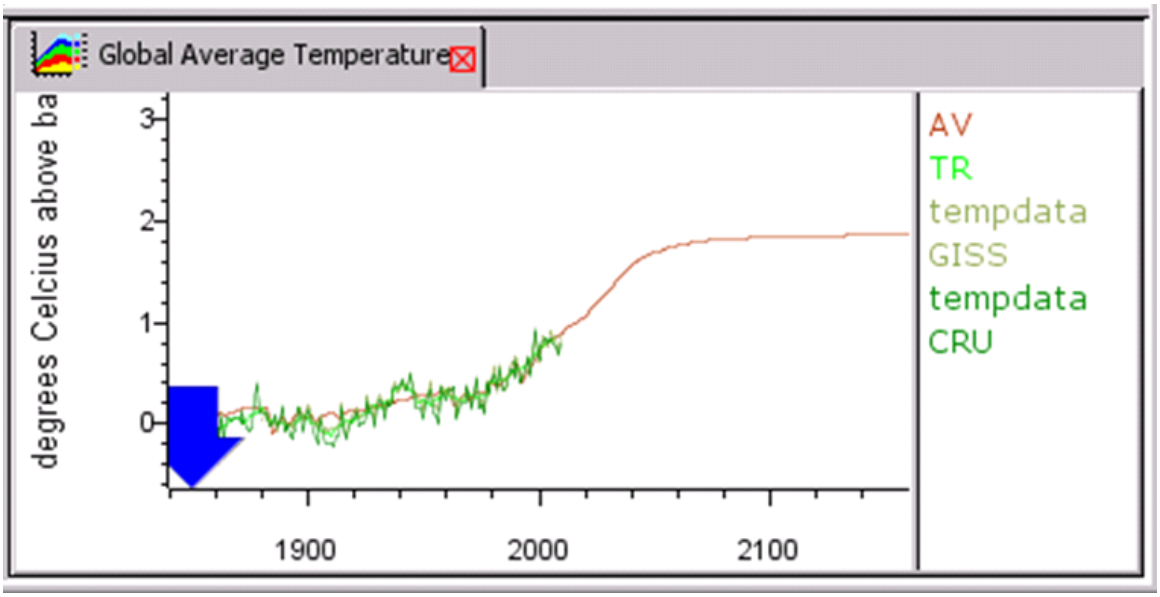

Figure 2: The global average temperature change given by the Java Climate Model.

in its positive form, $x_i^+ \equiv x_i$, and its negative form, $x_i^- \equiv -x_i$ and each expression can be represented by a conjunction of difference constraints. For example, $x_1 + x_2 \geq b$ can be represented by the difference constraint $x_1^+ - x_2^- \geq b$. The ABM is a $2n$ by $2n$ matrix (where n is the number of variables), with each row and column labeled by one of the positive or negative variables. The entry in the i^{th} row and j^{th} column is the bound of the difference. The widening operator $\triangledown$ is defined over two ABM's M and N as follows:

$$[M \triangledown N][i,j] = \left\{ \begin{array}{ll} M[i,j] & \text{if} M[i,j] \leq N[i,j] \\ -\infty & \text{if} N[i,j] < M[i,j] \end{array} \right\}$$

For each line in the code, we want to find the set of possible values for each variable. We will apply an abstract execution, which finds an over-approximation of

the values, by repetitively applying the widening operator until there is no further change. The use of the widening operator can be too loose, therefore, we can use an l-u-widening, $\diamond_{l,u}$, to get a more precise approximation. The $\diamond_{l,u}$ is defined over two ABM's M and N as follows:

$$[M \diamond_{l,u} N][i,j] = \left\{ \begin{array}{ll} M[i,j] & \text{if} M[i,j] \leq N[i,j] \\ N[i,j] & \text{if} l \leq N[i,j] < M[i,j] \\ -\infty & \text{if} N[i,j] < l \leq M[i,j] \end{array} \right\}$$

Finding the least fixed point semantics of a program will allow us to examine the program state, informing us of possible values that variables in the program could take [6].

The Management of Linear Programming Queries (MLPQ) database [7, 5] is a constraint database that implements these operations needed for software verification. The program must first be converted to Datalog, then MLPQ can compute the least fixed point semantics.

3 Experiments

The goal of the experiment is to determine an invariant value on the average temperature change above a baseline year. Typically the value of the average temperature change above a baseline year is computed between 1990 and 2100 [1]. Instead, the variant average temperature change above a baseline year will not depend on a year, but will be an upper-bound that the model can compute.

First, we discuss the experimental design, converting the Java Climate Model (JCM) code into Datalog to use with the MLPQ system, which can compute the least fixed point semantics to find the possible values the average climate temperature change. Second, we will discuss the results of the conversion.

3.1 The Experimental Design

The code was examined to determine constant values that do not change from in each iteration of the program (e.g., year-to-year). For the JCM, these values are typically the flux equations, or values that are specified by the user to adjust the model to match an AOGCM. The default values were taken in this conversion to Datalog. Once these values were all identified, because of the assumptions made by JCM to use the eigenvector calculation method instead of integration, all of the resulting equations were linear. Having linear equations was the goal of the model to study because MLPQ is a linear constraint database.

In the converted Datalog code, we refer to "line$i$" as the values of the variables at the start of line i of the program. Each line of the code was converted to Datalog to represent the change of values. Since we have only linear equations, these conversions are easy. For example, line 2 states:

```
guess=n+(n-nold)
```

and can be converted to Datalog:

```
line3(n,ndold,guess):- line2(n,ndold),
                       guess=n+(n-ndold).
```

This Datalog code states that at the start of line 3, we are taking the same state as the start of line 2, except that we have now manipulated guess to be the equation specified in the code.

The code we are converting from JCM contains two for-loops, which pose a problem for Datalog. However, each iteration of the for-loop is independent from one-another and the code can be duplicated for each iterated. Since the method in JCM is called once for each year, after computing the last line in Datalog, we create a rule for the first line that propagates the values from the last line back as input. This creates the loop in the code that can then compute the invariant bounds. After MLPQ has finished computing the values, we can look at the relation of the last line and see the bound of the global average temperature change.

3.2 The Datalog Code and The Coverter

The first step in creating the Datalog code was creating an even simpler model that assumed the ocean/land for each hemisphere took the same values. This simplification allowed the for-loops on line 5 and line 11 to be removed. However, this simplification was later removed by unrolling the content in the for-loops, once for each loop of the for-loop. The reason this simplification could be made was because the loops were independent from one another, which allowed the code to be unrolled.

Another simplification made, that is still in place, simplifies the for-loops of line 6 and 15 to only compute the first three values. Normally, these for-loops iterate over 40 values. All of the implementation details are in-place to remove this simplification. However, currently the program is not currently running in MLPQ. Therefore, expanding the Datalog program to execute more relations would be problematic when it is not already working.

The goal of the converter is three fold: 1) allowing the insertion of constants into the Datalog program, 2) convert equations for MLPQ compatibility, and 3) allow more complicated arithmetic operations on constants (i.e., multiplication).

Prior to using the converter, we inserted constants into the Datalog program by fixing the variable assignment. For example, consider the constant $x = 123$:

```
CONST_X(x) :- 123
```

ISBN: 978-960-474-381-0

We found that using constants in this fashion overcomplicated the program, and caused significant overhead. Therefore, the converter insert the constants directly into the Datalog code. Notice, we could have put the constants into the original Datalog program directly, but using a converter increases the readability of the code and gives us the ability to change the constants if required.

To accomplish multiplication, we tried to generalize an approach of multiplying two integer variables (See page 240 of [6]) to using floating point numbers, and first attempted to create a more general multiplication in Datalog as follows:

```
mult(x, y, z) :- y = 0, z = 0.
mult(x, y, z) :- y - y1 = 1,
                 z - z0 - x = 0,
                mult(x, y1, z0).
```

However, in the above multiplication, where $x \times y = z$, the value of x is allowed to be a floating point number, but y is still required to be an integer. Since in some calculations both x and y need to be floating point numbers, we utilized the converter because all of our multiplications are on constants.

The converter has three parts:

1. A set of assignments used to convert constants to floating-point numbers. These assignments are stored in the 'ASSIGNMENTS' variable in the form 'VARIABLE=NUMBER'. VARIABLE is the text to search for, and NUMBER is a floating-point number that can be positive or negative. All fractions of numbers must have a leading '0' prior to the decimal point.
2. Converts double negation into a plus, for MLPQ compatibility. (E.g., $2 - -2$ becomes $2 + 2$.)
3. Evaluates arithmetic operations on numbers that are included in square brackets [ ]. This functionality allows more advanced arithmetic operations to be applied on constants (e.g., multiplication, division).

The converter script was created to work as a UNIX shell script. The script assumes a file named 'datalog.txt' is in the same working directory as the script, and will output a file named 'datalog_convert.txt' in the same working directory as the script. To run the script, simply type './convert.sh'. Note, the script requires the proper permissions set (e.g., chmod 700). Note: when using a Windows computer and, the script might need to be converted not to have the Windows line returns (e.g., dos2unix convert.sh).

4 Discussion

One of the problems we had early on when writing the Datalog code was not always determining when we had a typing error in one of the variable names in our code. MLPQ would then, correctly, interpret in the code the mistyped variable as a 'free-variable,' one that does not have any constraints on it. This problem caused errors early on in values not being computed properly.

Another struggle was getting MLPQ to properly load the relations. Some relations would take a huge amount of time for MLPQ to compute the value of the relation, which caused the program to look like it crashed. However, after waiting patiently, the program would load the relation. One thing we did to get around this issue was tweaking the order of relations that were loaded, and optimizing code by factoring out common code and creating smaller relations.

In our tests, MLPQ returned the value of 1 for the bound on the global average temperature change as can be seen in Figure 3 for the results from the MLPQ system.

Although the idea of testing the long-range predictions of climate models using software verification is an intuitive and valid idea, the value of 1, unfortunately, should not be taken as conclusive because of the possible errors in the translation process from Java to Datalog and some simplifications we had to make to the original code. We need further tests of the algorithm to achieve confidence in its correctness. Nonetheless, this experiment shows the soundness of our approach of being able to compute invariants for climate models.

5 Related Work

The Earth System Modeling Framework (ESMF) (http://www.earthsystemmodeling.org) is another open-source framework that supports climate models of the world. This project seems to be more supported than that of the Java Climate Model, but the details of the ESMF are far more complex. Since the primary aim of our paper was only to show the feasibility of applying software-verification techniques to testing the long-range implications of climate models, we started with the simpler JCM model. Applying software verification techniques to ESMF too remains a natural next step.

To the best of our knowledge, no one has previously attempted to compute an invariant value for climate change models. In fact, the calculation of any invariant is not considered in Chapter 9 of the TAR, where all simulations arbitrarily end at 2100.

6 Conclusions

Climate models integrate physical principles with sensor data about both moving object and static climate variables. Long-range data mining about the predictions provided by these climate models is a challenging problem because current climate models use itera-

ISBN: 978-960-474-381-0

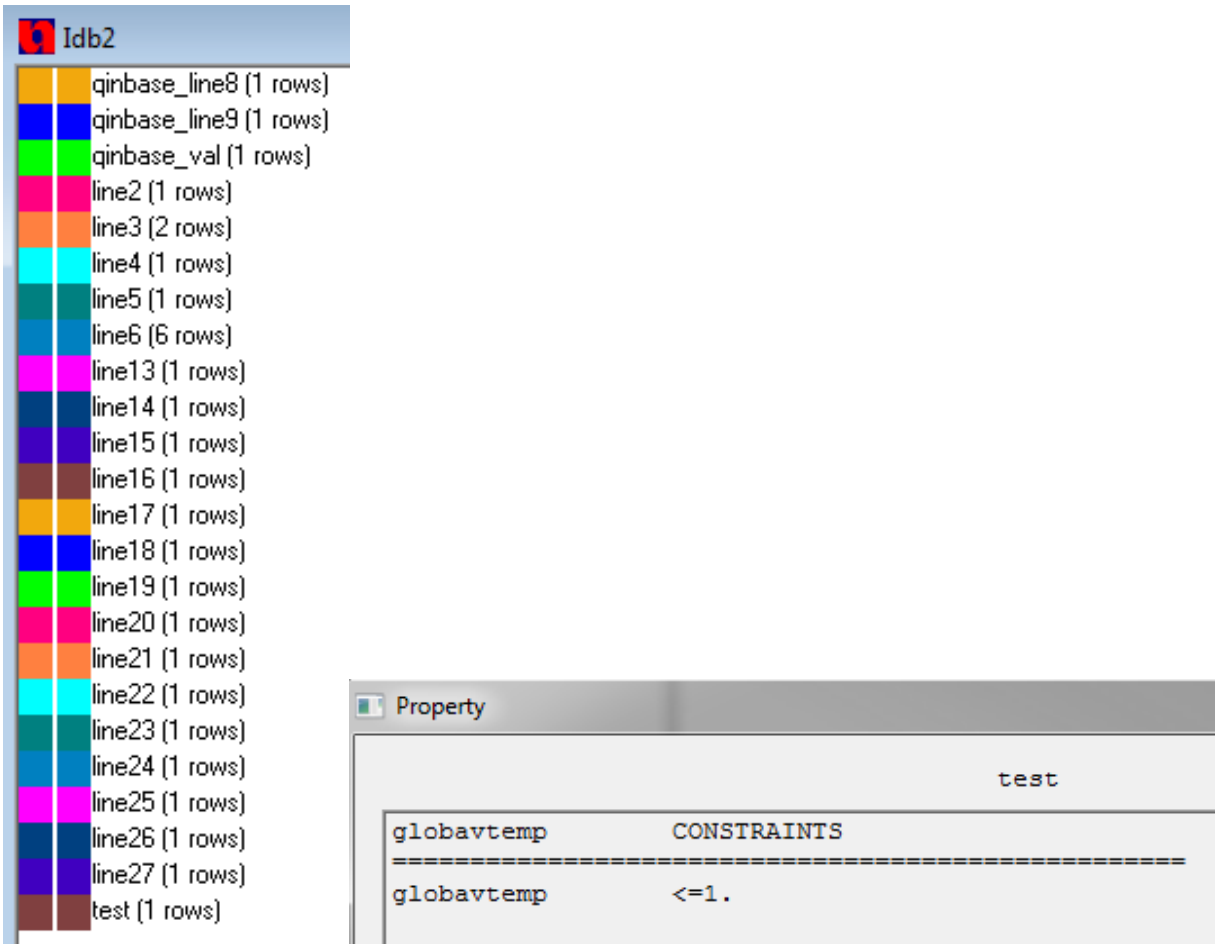

Figure 3: The relations loaded into MLPQ (left) and the resulting output of the global average temperature change above baseline of value one (right).

tive methods to compute the value of variables one year at a time. Currently these iterative methods are computed only for a fixed number of iterations. In this paper, we provided a novel method that is able to answer these challenging questions. We also gave a particular implementation using Datalog and the MLPQ system. The idea of our method is a general contribution that is applicable to other climate models too. In the future, we plan to implement more advanced climate models, such as the Earth System Modelling Framework. That model would allow certain further code optimizations because parts of the code could be factored out as separate relations to make the computation by MLPQ easier.

References

[1] *Climate Change 2001: The Scientific Basis*. Cambridge University Press, 2001.

[2] *Climate Change 2007: Synthesis Report*. Cambridge University Press, 2007.

[3] S. C. B. Raper, J. M. Gregory, and T. J. Osborn, "Use of an upwelling-diffusion energy balance climate model to simulate and diagnose A/OGCM results," *Climate Dynamics*, vol. 17, pp. 601–613, 2001.

[4] *Java Climate Model*, June 2011. [Online]. Available: http://jcm.climatemodel.info/

[5] S. Anderson and P. Revesz, "CDB-PV: A Constraint Database-Based Program Verifier," in *International Symposium on Abstraction, Reformulation and Approximation (SARA 2007)*, ser. LNCS, vol. 4612. Springer, 2007, pp. 35–49.

[6] P. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*, 1st ed. Springer, 2010.

[7] P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang, "The MLPQ/GIS constraint database system," in *ACM SIGMOD international conference on Management of data (SIGMOD 2000)*, 2000, p. 601.