Optimization of KINETICS Chemical Computation Code

Cristina Donastorg
NASA Jet Propulsion Laboratory, California Institute of Technology SURF Summer Intern, Pasadena, CA, 91125

NASA JPL has been creating a code in FORTRAN called KINETICS to model the chemistry of planetary atmospheres. Recently there has been an effort to introduce Message Passing Interface (MPI) into the code so as to cut down the run time of the program. There has been some implementation of MPI into KINETICS; however, the code could still be more efficient than it currently is. One way to increase efficiency is to send only certain variables to all the processes when an MPI subroutine is called and to gather only certain variables when the subroutine is finished. Therefore, all the variables that are used in three of the main subroutines needed to be investigated. Because of the sheer amount of code that there is to comb through this task was given as a ten-week project. I have been able to create flowcharts outlining the subroutines, common blocks, and functions used within the three main subroutines. From these flowcharts I created tables outlining the variables used in each block and important information about each. All this information will be used to determine how to run MPI in KINETICS in the most efficient way possible.

I. Background

For over thirty years NASA JPL has been creating a code called KINETICS, which calculates molecular abundances in planetary atmospheres. The idea is that molecules can be observed in different planets’ atmospheres and by comparing them to models this can be indicative of conditions in the atmospheres. The calculations done by KINETICS include both the chemistry and transport processes in the atmosphere e.g. Winds.

As the different pieces of the code have been compiled together in Fortran it became clear that the program could be more efficient. Therefore, there has recently been a push to integrate Message Passing Interface (MPI) into the program in order for the computation process to become more optimized. MPI is a programming method that makes it possible to run multiple processes, or segments of code, simultaneously instead of one at a time. One of the more appreciable differences between a regular, series computational program and one that runs parallel processes using MPI is the amount of time the server spends solving the problem it was given. With a program as vast and as complicated as KINETICS if it were run in series it would take an extremely long time to run, not to mention the
stress that would be put on the physical hardware itself. Many of the subroutines within the KINETICS code have already been converted to an MPI format, however there is still a lot of inefficiency in the program.

In order to make the program more efficient my task was to outline all the variables called from three of the major subroutines. The purpose of this is to be able to more easily determine which variables need to be scattered and when that needs to occur. Being able to scatter only certain variables and at the right time can drastically increase the efficiency of the program.

II. Tasks

My main task for the first few weeks of the internship term was to become familiar with KINETICS. Once I knew the basics my task for the remainder of the term was to dissect the main subroutines to understand how the many variables are used.

A. MPI for the FORTRAN Programmer

The first thing I had to do was read the manual entitled “MPI for the FORTRAN Programmer” which describes MPI in relation to the KINETICS code. This manual helped me to understand MPI and the commands that are found throughout KINETICS. For example, shown below are two examples of methods to parallelize code that I learned about through the manual. The first method is to have all the processes run the entire program. The second method is the have just a single process running, break up sections of the program to be calculated by multiple processes, and then to gather the results into a single process. The second method is the one that is widely used within KINETICS.

B. KINETICS Subroutines: MARCH, RAD, and JALT

After I read through the manual I began going through the KINETICS code itself using the text editor Emacs. There were three main subroutines that I needed to evaluate: MARCH, RAD, and JALT. Each of these subroutines has a separate part in performing the overall chemistry modeling of the program.
KINETICS begins the model with starting conditions of parameters, such as the density, pressure and temperature structure of the atmosphere and also the initial abundances of the molecules. Then the model integrates over time to compute the way that the abundances of molecules in the atmosphere change. These integrations are done by the MARCH subroutine.

RAD calculates the radiation field in the atmosphere. This is necessary because the Sun’s ultraviolet (UV) field will irradiate molecules near the surface of the atmosphere, and this will dissociate them. Transport processes can then move the dissociation products lower down into the atmosphere, where they are protected from the UV field and can reform molecules.

JALT calculates the photodissociation rates, or the rates that compounds are broken down by photons, as a function of position in the atmosphere. Because different molecules dissociate at different wavelengths, the model is given the full spectrum of the Sun. Since the dissociation cross-sections as a function of wavelength it calculates the dissociation rates as a function of wavelength.

One of the deliverables required from each of the main subroutines was a flowchart showing how all the subroutines, common blocks, and functions in MARCH, RAD, and JALT fit together within the subroutine. These flowcharts can be found in Appendix 1. The other deliverable was a table listing how every variable called by the different segments of the subroutine was defined, how it was used internally and externally, and whether or not the variable needed to be scattered before the call to the subroutine and gathered when returned to CHIEF.

1. **Process for outlining the variables**

The first step in outlining the variables called in MARCH, RAD, and JALT was to find where each variable was initialized. Depending on whether the variable was initialized internally in CHIEF or externally determines whether the variable is scattered, or broadcast from a higher subroutine to a lower one, or not. For example, if a variable was calculated in INPUT and saved as an AIPT then it will need to be scattered. If it were calculated in CHIEF or within the subroutine in question then there is no need to scatter that variable, or else the program is less efficient.

The next step was to determine how the variable was used within the subroutine and also outside of it. If the variable was changed within the subroutine and then used externally as an input then the variable was to be gathered, that is, it needed to be sent to original placeholder of the variable and the change needed to be received. If the variable was not used externally at all or was not altered then gathering would be unnecessary.
After all this information was collected for all the variables in tables they were sorted so that variables that either need to be scattered, gathered, or both were placed at the top of the chart. This will make it easier for future programmers to implement the changes that need to be made.

### III. Future Work

The work that is still required to make KINETICS more efficient is to expand on the information about how the variables were initially defined. Also, whether or not the variables are updated elsewhere needs to be documented, as well as the flags for when the update takes place. All of this information will be used to determine where the variables to be scattered will eventually be broadcast to the correct subroutine.

#### Appendix 1: Flowcharts

**MARCH Flowchart**
Acknowledgments

Acknowledgments are due to my mentors Karen Willacy and Mark Allen. Also to the Summer Undergraduate Research Fellowships (SURF) program for giving me this opportunity and to the California Institute of Technology and William Carey International University for housing me over the summer.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by SURF and the National Aeronautics and Space Administration.