

Acceleration of the KINETICS Integrated Dynamical/Chemical Computational Model using MPI

Max Grossman¹, Karen Willacy², Mark Allen³
NASA Jet Propulsion Lab, Pasadena, CA, 91125

Abstract

Understanding the evolution of a planet's atmosphere not only provides a better theoretical understanding of planetary physics and the formation of planets, but also grants useful insight into Earth's own atmosphere. One of the tools used at JPL for the modeling of planetary atmospheres and protostellar disks is KINETICS. KINETICS can simulate years of complex dynamics and chemistry. At the moment, KINETICS is in use by or planned to be used for:

1. The ExoMars mission to study the Martian atmosphere in 2016.
2. Modeling of the composition of the protostellar disks which lead to star formation.
3. Examination of pollutants in Earth's atmosphere.
4. Modeling of Titan's atmosphere

Because of the complexity of KINETICS, it requires large amounts of computation to run simulations at a meaningful level of detail. Using MPI (Message Passing Interface), a popular multi-core programming model, will permit more useful experiments to be run in less time, accelerating the discovery process. The advantages of MPI lie in its exposure of communication, allowing programmers to manually tune the communication pattern of an application. MPI will be used with the existing FORTRAN code to parallelize and accelerate performance-critical sections of code in KINETICS.

I. Introduction

The understanding of the formation of a planetary body's atmosphere is important for both practical applications and pure science. By studying the composition of the atmosphere of Mars (for example), it is possible to not only gain a better theoretical understanding of planetary physics and the formation of planets but also gain insight into Earth's own atmosphere. However, the study of a body of particles as large as a planet or moon's atmosphere is not a simple task, and must be accomplished primarily using computer simulations.

To be able to trust the results of a simulation, it is necessary to represent the state of a massive body of molecules as accurately as possible. Doing this is extremely demanding in terms of both communication and computation. Therefore, it is not feasible to solve perfect chemical and dynamic models for an atmosphere. Rather, approximations must be used. This leaves us with a computationally difficult task rather than a computationally impossible task.

To run atmospheric models and other financial, medical, and scientific applications as demanding as it, multi-core programming is a necessity. Multi-core programming, as opposed to sequential programming, uses many processing units executing independent instruction streams concurrently in order to complete a task quicker than a single, faster processing unit. This ability to perform simultaneous tasks is extremely useful in scientific computation, and allows applications to be significantly accelerated. One of the tools used at JPL for the modeling of planetary atmospheres and protostellar disks is KINETICS. KINETICS allows the user to provide an initial input file containing information on the body being simulated and take the body of interest through years of complex dynamics and chemistry. At the moment, KINETICS is in use by or planned to be used by:

¹Summer Intern, Astrophysics and Space Sciences, NASA Jet Propulsion Lab, Pasadena, CA

²Research Scientist, Astrophysics and Space Sciences, NASA Jet Propulsion Lab, Pasadena, CA

³Research Scientist, Astrophysics and Space Sciences, NASA Jet Propulsion Lab, Pasadena, CA

1. The ExoMars mission to study the Martian atmosphere in 2016.
2. Modeling of the composition of the protostellar disks which lead to star formation.
3. Examination of pollutants in Earth's atmosphere.

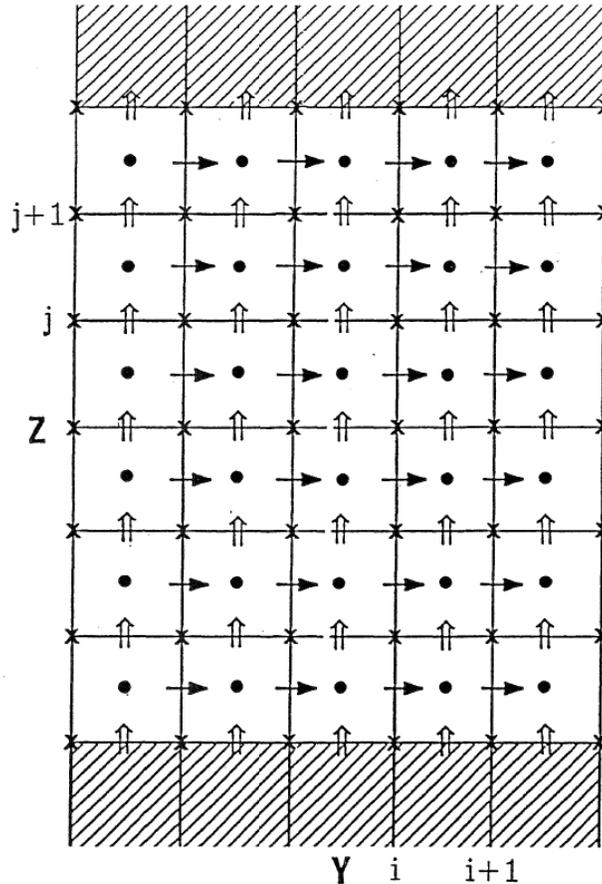


Figure 1: 2D dynamics simulated by KINETICS.

While KINETICS has years of development behind it (or because of the work that has been invested in it) it requires huge amounts of computation time to run simulations at a meaningful level of detail. Modifying the FORTRAN code base to take advantage of multiple cores will allow more useful experiments to be run in less time, accelerating the discovery process. To do this, MPI (Message Passing Interface) will be used, a common parallel programming model. The advantages of MPI lie with its exposure of communication to the user. Communication of data between physically separate computation units is often the most significant source of performance degradation in multi-core applications. MPI ensures that no data is shared between processors which is not explicitly communicated, and in this way allows the programmer to tune the communication patterns and reduce total communication. As a result, MPI also places considerably more burden on the programmer to optimize the application than other parallel programming models, such as OpenMP or Java Threads. MPI will be used with the existing Fortran code to parallelize critical sections of code in KINETICS which consume the largest amount of processor time.

II. Methods

MPI is a parallel programming model commonly used in cluster computing where several machines with multiple processor are connected by high latency, low bandwidth interconnect. The advantage of MPI is that it exposes any

and all communication to the programmer, allowing them to manually manage any data transmitted. This comes at the cost of more programming burden being placed on the programmer, in that any shared data must be explicitly communicated, but provides the parallel programmer with more power to optimize the communication patterns of their application.

An MPI program is just a collection of processes, each running identical code and only differentiated by their process IDs. These processes can communicate with each other whether using TCP. Because the TCP protocol is used for communication, from the programmers point of view there is no difference between transmitting data to another process on the same rack or in another room.

The code transformations necessary to move from sequential FORTRAN code to parallel MPI/FORTRAN code are often extensive and included:

1. Removal of COMMON blocks. COMMON blocks provide variables which can be accessed from anywhere in an application. While they can be useful, they tend to make code less readable and finding inputs and outputs of subroutines a greater challenge. In order to aid in the analysis of the code base, any sections which were to be parallelized had references to COMMON blocks removed.
2. Loop transformations. The most common form of parallelization is across iterations in a loop. So long as the work done in each iteration is independent of the others, this can be a simple and very beneficial parallelization technique. However, most code (particularly legacy code) was not designed with parallelization in mind, but rather expressed how the programmer originally understood the concepts. It was necessary to transform many of the loops in KINETICS to 1) create loops which would contained sufficient work as to be beneficially parallelizable, and 2) ensure each iteration of these loops were independent and could be parallelized.
3. Insertion of MPI calls. For this program to be parallelized, many calls to communication subroutines had to be inserted in the original FORTRAN code. These can be split into two categories. First, MPI_BCAST calls were inserted in order to transmit to all MPI processes the current state of the program and ensure that they all had the correct inputs when entering a parallel section of code. Second, MPI_RECV and MPI_SEND calls were inserted after those parallel sections of code in order to gather the results back to a master process, which would then go on to use those results for additional, sequential computation.

All of the above steps required considerable by-hand analysis of the FORTRAN code for data dependencies.

When parallelizing an application like KINETICS, it is crucial to maintain consistency with the original source code. When working with scientific code which will be used to expand the understanding of Earth and other planets, this becomes even more crucial. The primary output of KINETICS consists of a text file with information about each time step in the simulation, including concentrations of certain elements at different altitudes, latitudes, and longitudes. In order to validate the output of this code the Unix diff utility was used to compare the contents of a control output file and the output of the altered code. When the same compiler was used, no differences in these files were tolerated. Different compilers with identical code did cause differences in the output, so some tolerance for changes had to be made in those situations.

When first analyzing the KINETICS code, performance profiling using gprof and visualization tools was done in order to identify those sections of code which were taking up the most execution time, and therefore whose acceleration could have the largest impact on total run time.

In addition to the work done on the actual code, it was also necessary to include the ability to use MPI in the build process. KINETICS supports several different compilers which can be used to build KINETICS. This is useful and necessary in order to distribute KINETICS, as not every researcher will have access to the same architectures or compilers depending on licensing issues. Therefore, it was necessary to insert a new option which included the MPI compiler and the arguments to that compiler which would be necessary to produce results compatible with the others. This process was a lesson in the inconsistencies which different compilers can have, and the numerical effect they can have on the output of an application. Simply switching from the Absoft Fortran compiler to the Intel

Fortran compiler resulted in significant changes in the output of KINETICS, as a result of some aggressive optimizations on behalf of Intel. However, the Intel compiler actually picked up on some compile time errors which Absoft did not. Adding floating-point accuracy options to the Intel compilation brought the two compilers into close agreement, but this entire task taught us something about how carefully different compilers should be used and the skepticism their results should be treated with.

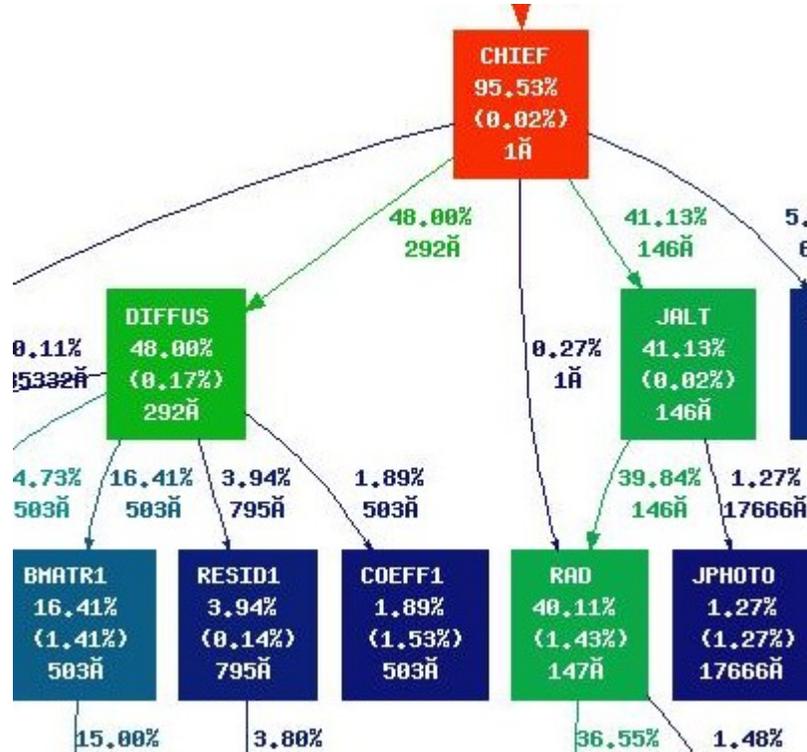


Figure 2: An example of the output of a profiling session on one of the sample data sets.

III. Results

One result of this internship is the composition of a manual on the use of MPI. This manual includes:

1. An overview of MPI. Targeted at those with no experience with MPI or parallel programming, this overview tries to relate MPI to more commonly understood features of computing to make MPI a more approachable programming model.
2. Detailed information on useful MPI subroutines.
3. More advanced performance concerns which are important to keep in mind when building an MPI application, as well as how to resolve them.
4. Compilation and Execution of an MPI program, taking the reader through the steps to build and run a simple MPI application.
5. A walk-through of the parallelization of matrix-matrix multiply using MPI.

The goal of this manual was to provide scientists (non-programmers) with the ability to parallelize their code.

Titan (1D) Execution Time

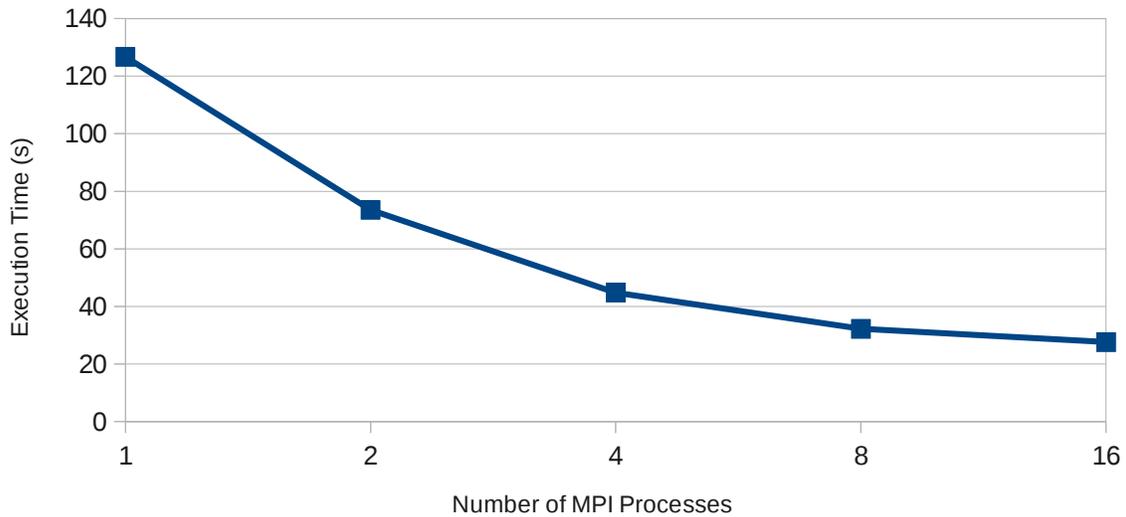


Figure 3: Execution time of the parallelized MPI program using the Titan 1D data set and varying the number of MPI processes.

In order to test the performance of the parallelized code, a data set representing a 1D column of the Titan atmosphere was used. Timing was achieved using the Unix time utility.

In Figure 3, we see performance gain of the KINETICS code from 1 process to 16 processes, achieving an eventual speedup of 4.5x (parallel efficiency of 29%). While the acceleration is significant, the gains are not efficient or linear. There are two explanations for this:

1. Amdahl's Law. Amdahl's Law places an upper bound on the amount of speedup that can be expected from parallelizing sections of code within a program. Because only two functions, RAD and MARCH, are being parallelized in this execution of KINETICS, only a certain percentage of the total execution time is being done in parallel. Even given perfect speed up of those sections and infinite processors, there is a theoretical lower bound on the measured execution time, which would be the time which the sequential sections take. As more of the code is parallelized this lower bound on execution time and upper bound on speedup become more beneficial, but time constraints dictated that no additional work could be done.
2. Communication. As more processes are added, more data needs to be transmitted. Rather than sending an array from process 0 to process 1 with 2 processes, it would be necessary to transmit that array to 15 other processes when using 16 processes. This places more load on the available bandwidth and can affect the observed speed up. Again, due to time constraints more efficient data transfer patterns could not be implemented, which would limit the impact of this problem.

IV. Conclusion

This paper demonstrated that there are significant improvements to be gained from parallelizing the KINETICS code. Using MPI to limit communication and parallelize performance-critical sections of code produced speedup of 4.5x.

While promising results were obtained, the difficulty of working with legacy code which was never designed for parallelization represented a serious bottleneck in this project. The loop structure of the code did not lend itself to parallelization, and required some major code transformations to be done and validated. The use of Fortran common blocks made it difficult to analyze data dependencies between subroutines. Even the compile process presented a

challenge as we learned that different compilers treat errors in different ways, produce slight differences with completely identical code, and rely in different libraries or syntax. Overall, this added overhead constituted the bulk of the work done while the actual parallelization tasks were relatively simple, straightforward, and quick.

Given more time, considerably better performance could be achieved. The use of more complex and efficient data transfer patterns could be implemented, following a policy of copy early and overlapping communication with computation. More of the source code could be parallelized, increasing the upper bound on speed up while potentially eliminating the need to copy certain data. Additionally, more testing could be completed to further certify the robustness of the parallel code.

The manual written as part of this work will hopefully be useful for future parallelization efforts on the KINETICS code and other scientific codes which lend themselves to parallelization.

Acknowledgments

I would like to first thank Karen Willacy. Her constant aid and attention throughout this project was incredibly helpful in understanding and analyzing the code. She was always willing to explain the functionality of a subroutine when my lack of knowledge got in the way.

I would also like to thank Mark Allen. His experience with KINETICS helped to prod me towards the subroutines which were most open to parallelization, and would have the greatest impact on execution time.

Beyond work, Karen and Mark both made me feel welcome at JPL and provided support even when the results were less than promising. It was a pleasure working with both of them.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the NASA Undergraduate Student Research Program and the National Aeronautics and Space Administration.

References

1. “Vertical Transport and Photochemistry in the Terrestrial Mesosphere and Lower Thermosphere (50-120 km)”. M. Allen, Y. Yung, and J. Waters. *Journal of Geophysical Research*, Vol. 86, No. A5, Pages 3617-3627. May 1, 1981.
2. “Two-Dimensional Atmospheric Transport and Chemistry Model: Numerical Experiments With a New Advection Algorithm”. R. Shia, Yuk Lung Ha, Jun-Shan Wen, and Yuk L. Yung. *Journal of Geophysical Research*, Vol. 95, No. D6, Pages 7467-7483. May 20, 1990.
3. “Modeling Exoplanet Atmospheres”. Gilster, Paul. <http://www.centauri-dreams.org/?p=1318>. June 21, 2007
4. “MPI: A Message Passing Interface Standard”. The Message Passing Interface Forum. <http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/mpi-report.htm>. June, 1995.
5. “Deuterium Chemistry in Protoplanetary Disks”. Karen Willacy, William Langer, and Geoff Bryden. *Planet Quest: Exoplanet Exploration*.