Examination of the Weather Research and Forecast (WRF) Model on Heterogeneous Distributed Systems

Wei Huang, Jennifer Abernethy
National Center for Atmospheric Research
University of Colorado at Boulder

Abstract

The Weather Research and Forecast model (WRF) is a numerical weather prediction model suitable for research and operations. WRF is designed to run on a variety of platforms, either serially or in parallel, with or without multi-threading. Currently, a model forecast can be run on only one system even though the choice of system is highly flexible. In this paper, we describe our implementation of WRFv1.3 using TCP/IP that allows division of a model domain between two heterogeneous systems distributed across a network. This option can help current and potential WRF users utilize all the computing power available to them; it allows users to run larger and/or higher resolution domains than would be feasible on only one of their systems. Forecast results are equivalent between distributed and non-distributed runs on the same domain with the same initial conditions. We examine the communication costs and their impact on performance, explore how to reduce those costs, and discuss next steps in development.

Introduction

The Weather Research and Forecast (WRF) model project is a multi-institutional effort to develop an advanced mesoscale forecast and data assimilation system that is accurate, efficient, and scalable across a wide range of problems and over a host of computer platforms. Institutions participating in the development include research centers, government laboratories, and universities. A main focus of the WRF effort has been to develop and implement a software infrastructure that facilitates modular, flexible, reusable and portable software [1].

This project is the combination of two class projects completed in the Computer Science Department at the University of Colorado-Boulder: the first by Huang and Abernethy in the Advanced Operating Systems course (CSCI5573, Fall of 2003), and the second by Huang in the High Performance Computing course (CSCI7000, Spring 2004). Huang and Abernethy implemented WRFv1.3 to run across two heterogeneous systems; each system ran a single thread. Huang further developed that project into a fully parallel distributed heterogeneous implementation. We believe the ability to pool together resources to study larger domains and/or higher resolution applications will be useful to researchers who currently may be limited by the performance of their fastest system. Our distributed implementation would allow a researcher, for instance, to take his current domain, increase its resolution, and run a forecast across an SMP system and a Linux cluster, or between two workstations with different operating systems. In this paper, we will discuss the implementation details of the prototype, evaluate performance, and examine communication protocol issues involved. In addition, we will discuss the usefulness of WRF as a distributed application in regards to performance, and future work.

Implementation

WRF is a finite difference model that uses an Arakawa "C" grid. Time integrations for each grid point are done using data on its own grid point and data from its neighbors. The simplest decomposition is to split the model domain into two, as shown in Fig. 1. To make things even easier, assume we always split the domain in the horizontal (x-direction), since currently this is the standard decomposition used by WRF. Data values at the division boundary must be communicated at every time step when we integrate forward in time. For this project, we set the boundary values to be communicated as 5 columns. This translates into a 120pt stencil, which is a stencil size that is included in the WRF code and may be used in the future; until then, we are safely padding our boundaries.

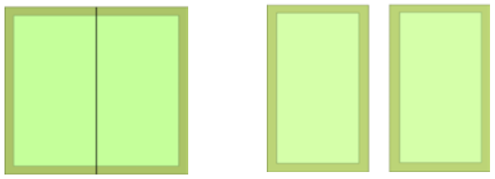

Figure 1. The picture on the left shows a full domain. The picture on the right shows our distributed domain decomposition, where the domain is split between two systems, and there is a halo region for each sub-domain.

Since our project deals with developing a distributed heterogeneous application, we will assume that we are using two different systems which can differ in architecture and/or operating system. Specifically, we assume the two systems can not both run the same executable; of course, our code will work with those that can. In either case, the model must be compiled separately on both systems. Using a run-time namelist variable, one system is designated as the server, and the other is designated as the client (the current prototype requires a compile-time specification of the server hostname, also). Functionally, the only difference between client and server is that the client makes the initial contact with the server to begin communication. We used TCP sockets for communication to ensure reliable packet transfer. Once the socket is established, the client and server are simply partners in data exchange. To prevent network blocking, the code is organized such that for each timestep, the client always sends its data before listening for incoming data, and the server must receive data from the client before sending its own data.

In each time integration step, we need to exchange every model variable at the boundary. The variables needing to be exchanged include temperature, pressure, humidity, wind components (3 in total), and others. This prototype exchanges the data for one variable as a single message. There are about 230 variables that need to be exchanged in a large time step (1 large step includes 4 short time steps), and thus about 230 messages per step. The code checks the floating point representation of each machine, and if necessary, handles conversion between little endian and big endian floating point representations [2].

As shown in Fig. 1, if both sides are running with a single thread, it is straightforward that at each time step these two sides exchange data at their adjacent boundary. However, if one or both systems are running in parallel with MPI, the communication will become more complicated. Ideally, if both sides have the same number of processors at the boundary, we could connect as pairs the processors assigned adjacent grid points across the boundary. In reality, the two systems may have different numbers of processors at the boundary, and the grid points assigned to one processor on system 1 may belong in halo regions of more than one processor on system 2. Even if both systems have the same number of processors on the boundary, some systems may only have one node capable of network communication. Thus, we assume that there is only one I/O node on each system that can handle network communication (Figure 2). With this assumption, on each individual system, the I/O node needs to collect all the boundary data for the data exchange. After exchanging data with the other system, the I/O node must distribute the data it received back to the nodes responsible for those boundary grid points.

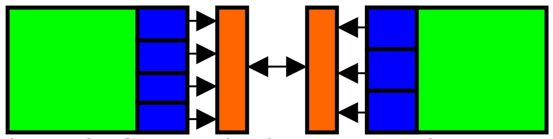

Figure 2. Communication between 2 systems when both side have multiple processors (in this figure, the left hand side has 4 processors at the border, and the right hand side has 3 processors).

Experiments

The case we have chosen for our preliminary experiments is the idealized super-cell (em_quarter_ss in the WRF distribution). As it is an ideal case, it is easy to generate initial conditions and lateral boundary conditions. We ran forecasts with a 2km resolution and grid dimension of 80x70x40. The grid dimensions on each sub-domain are 40*40*40. There are ten grid 'columns' overlapping (in N/S direction, per vertical level): the 'west' sub-domain columns 31-40 are duplicates of the 'east' sub-domain columns 1-10. The system computing the west sub-domain sends its column 31-35 to the east sub-domain columns 1-5. The east sub-domain system sends its columns 6-10 to the west sub-domain's columns 36-40.

We did initial testing on a four processor Compaq Tru64 system running Unix v5.1, and a system with four 1.9Ghz Intel Zeon processors running Linux Fedora Core release 1, both within the NCAR/MMM administrative domain and connected via 100Mbps Ethernet. We have tested in three basic settings:

1. Single processor to single processor (each system only has a single thread);
2. Multiple processors on both systems, with the same number of processors on each side;
3. Multiple processors on both systems, but each system has a different number of processors on the shared boundary.

For comparison, we include 'local' runs, where the same code was executed on a single system. While the forecasts could not be compared bit-for-bit because of the different floating point handling/storage on different architectures, qualitatively the forecasts were the same (no visual discrepancies).

Table 1. Average Wall Time for a Model Timestep

Experiment	*Time(seconds)*
Local, single processor	1.046
Local, 4 processors	0.567
Distributed, 1processor to 1 processor	6.549
Distributed, 4 processors to 4 processors	4.532
Distributed, 4 processors to 2 processors	4.689

Table 1 shows the average wall time for a model integration step. For these preliminary results, we used 10-timestep runs and averaged the times after discarding the largest and smallest step times. The experiments were run after-hours when there was low network traffic and system usage, but it was not officially dedicated processor time on either machine. While these results can't be used as official performance numbers, we feel they can represent a typical usage scenario.

Discussion

It is apparent that the packaging and communication involved in distributing the domain across a network significantly increases the forecast time. This was not unexpected. We believe the value of this project lies in the ability to pool together resources; we warn against holding your breath for the results.

The problem for the single processor to single processor run came from the compile option we used. Here, we still compiled with MPI. Therefore, there is a huge amount of redundant copy, as we still use the collecting and distributing strategy. In real application, we do not need to do this anymore in this situation.

Our preliminary experiments show that we can run WRF across distributed, heterogeneous systems. However, the performance needs more improvement. The major issue for the distributed heterogeneous application is the communication cost. The communication cost includes two parts: latency and bandwidth. With the development of faster networks, bandwidth will increase. We can do some things ourselves to decrease the total latency cost more immediately, though. The first one is to reduce the amount of data transported. In these preliminary experiments, we have exchanged five columns of data. With careful analysis and arrangement, we probably can cut the columns exchanged to three (the minimum necessary for correctness). The second one is to reduce exchange time to eliminate some system latency. There are two ways to decrease exchange times; one is to transport a larger message size by packing data together. The other one is to make the exchange zones wider, and then exchange data not every integration time step, but put a few steps together.

Related Work

There has been some work in using WRF and other atmospheric models in distributed computing, but all projects, to our knowledge, differ from ours in certain fundamental ways. Two of the projects are spearheaded by NOAA's Forecast Systems Laboratory. One, an initial application in a larger-scale project to develop NOAA's computational grid, employs the prototype of the grid between PMEL and FSL to (loosely) couple ROMS and WRF[3]. Data between the models are exchanged over the grid at the necessary timesteps. In that project,WRF still runs on a single system and receives boundary condition data over the grid. In contrast, ours is not a model coupling implementation but a way to run the WRF model by itself, dividing the domain among separate systems and communicating subdomain boundary conditions at each timestep. A second FSL project, the WRF portal, aims to streamline the process of testing thousands of

WRF variants to determine the best choices for a 6-member ensemble [4]. The portal will allow researchers to use one site to submit and manage jobs and access data from multiple WRF runs on systems all over the country. Again, our project is concerned with distributing a single WRF forecast among different systems, not managing the simultaneous execution of multiple forecasts.

Conclusion

We have added a new ability to the WRF model to allow the model run across distributed, heterogeneous systems connected by a network (currently, there are no plans to incorporate this functionality into a WRF release). By splitting the model domain into two parts, we allow the two parts to run on different architectures (platforms), and on each system the model can still run in parallel if there is more than one processor available. Our implementation produces qualitatively the same forecast as the WRFv1.3 release.

The main improvement opportunity is to decrease latency. We can (with some revision and addition to the code) send one large message per timestep, or increase the boundary zone in order to exchange data less often.

Future Work

A logical next step to this proof-of-concept implementation is to use the Globus Toolkit software package [5] to build the participating systems into a grid and use MPICH-G2 to handle the message passing between heterogeneous systems in different administrative domains [6]. Essentially, it uses MPI for intra-machine message passing and TCP sockets for inter-machine communication, which is virtually the same as our design. Currently, we are learning and setting up both software packages, and we aim to run WRF distributely across a prototype grid and compare the performance of both implementations.

Acknowledgments

We would like to thank David Gill for his advice, suggestions, and help setting up and using the idealized supercell case for our testing. We also want to thank John Michalakes for helpful discussions.

References

[1] Foster, I. And J. Michalakes, Parallel Supercomputing in Atmospheric Science, World Scientific, River Edge, NJ (1993), pp. 354-363.

[2] IEEE standard for floating point numbers. http://www.psc.edu/general/software/packages/ieee/ieee.html

[3]Govett, M., D. Schaffer, A. Hermann, C. Moore. Using the TeraGrid for NOAA Scientific Computing: FY2003 NOAA NPCC Funded Project. Available from http://www-ad.fsl.noaa.gov/ac/schaffer/ngi.html

[4]Govett, Mark. A WRF Portal for Model Test and Verification. May 2003. Available from http://www-ad.fsl.noaa.gov/ac/WRFGridPortal.html

[5] http://www-unix.globus.org/toolkit/

[6]Karonis, N., B. Toonen, I. Foster. MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. Journal of Parallel Distributed Computing, 63(2003), 551-563.