PARALLEL IMPLEMENTATION, VALIDATION, AND PERFORMANCE OF MM5

J. Michalakes, T. Canfield, R. Nanjundiah, and S. Hammond
Mathematics and Computer Science Division,
Argonne National Laboratory,
9700 S. Cass Ave., Argonne, Illinois 60439, U.S.A.

G. Grell
NOAA Forecast Systems Laboratory,
325 Broadway, Boulder, Colorado 80303, U.S.A.

Abstract

We describe a parallel implementation of the nonhydrostatic version of the Penn State/NCAR Mesoscale Model, MM5, that includes nesting capabilities. This version of the model can run on many different massively parallel computers (including a cluster of workstations). The model has been implemented and run on the IBM SP and Intel multiprocessors using a columnwise decomposition that supports irregularly shaped allocations of the problem to processors. This stategy will facilitate dynamic load balancing for improved parallel efficiency and promotes a modular design that simplifies the nesting problem. All data communication for finite differencing, inter-domain exchange of data, and I/O is encapsulated within a parallel library, RSL. Hence, there are no sends or receives in the parallel model itself. The library is generalizable to other, similar finite difference approximation codes. The code is validated by comparing the rate of growth in error between the sequential and parallel models with the error growth rate when the sequential model input is perturbed to simulate floating point rounding error. Series of runs on increasing numbers of parallel processors demonstrate that the parallel implementation is efficient and scalable to large numbers of processors.

1. Introduction

Computer simulation for weather forecasting is computationally demanding, especially at very fine resolutions and over very large domains. Massively parallel processors and networks of high-performance RISC-based workstations are a cost-effective and scalable alternative to vector/shared-memory supercomputing technology for weather forecasting. This paper describes the distributed-memory parallel implementation of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5) for use as a real time weather forecasting system for the United States Air Force. We first discuss requirements for the model and modeling system. Next, we describe the approach to implement the parallel model. This approach employs RSL, a parallel library package designed to hide low-level details of the parallel implementation and that is tailored to regular-grid finite-difference codes with mesh refinement. The parallelized model is validated with respect to the source model by comparing model output data plots and by measuring error growth rates in the source and parallel models. Performance results obtained running the parallel model on an IBM Scalable POWERparallel system are presented.

2. Model and System Requirements

MM5 is a regional weather model for prediction on domains ranging from several thousand kilometers down to several hundred (or fewer) kilometers. The model may be run at a resolution of as low as 1 km. Domains are uniform rectangular grids representing three-dimensional regions of the atmosphere. The horizontal coordinate system is equally spaced geographically and the model uses the Arakawa-B gridding scheme. The vertical coordinate system is σ surfaces, with layers distributed more closely nearer the surface (23 layers in the current model). For this implementation, atmospheric dynamics is nonhydrostatic and uses finite-difference approximation. Physics includes the Blackadar high-resolution planetary boundary layer scheme, the Grell cumulus scheme, explicit moisture with treatment of mixed-phase processes (ice), shallow convection, dry convective adjustment, and the Dudhia long- and short-wave radiation scheme [1].

*This work was supported by the Applied Mathematical Sciences subprogram of the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38. Corresponding author: John Michalakes, MCS Division, Argonne National Laboratory, Argonne, Illinois 60439, U.S.A. Email: *michalak@mcs.anl.gov.*

The requirements of the modeling system are that it provide high resolution over a very large domain with fast time to solution. A simple model of computational cost is

$$S = \frac{1}{t_{sol}} \cdot \frac{t_{sim}}{3\frac{sec}{km} \cdot r} \cdot c \cdot F$$

where S is the required floating-point operations per second, t_{sol} is the time to solution in seconds, t_{sim} is the length of the simulation in seconds, r is the resolution in kilometers, c is the number of cells in the domain, and F is the number of floating-point operations to compute one time step for one cell. The desired resolution is 10 km over a square geographical domain that is 1500 nautical miles (approximately 2700 km) on an edge. The desired time to solution for a 36-hour simulation is one hour. Based on a measured 8420 floating-point operations per three-dimensional grid point per time step, the problem as specified is essentially intractable, requiring a sustained computational rate of 17 billion floating-point operations per second. Therefore, we employ mesh refinement in the parallel model to focus the more costly 10 km resolution over a much smaller area of interest. Within the larger 2700 km by 2700 km domain, we instantiate a 10 km nested domain covering an area about 500 km on an edge. The rest of the larger domain is calculated at 30 km resolution, reducing the computational requirements for the large domain by a factor of 27 (9-fold fewer cells, and a 3-fold increase in the length of the time step, based on the MM5's nesting ratio of 3). This sacrifice of resolution over some of the original domain reduces the performance requirement to 760 million floating-point operations per second, a considerable savings.

3. Parallelization

Parallel efficiency is the degree to which a program achieves ideal speedup: an increase in computational speed proportional to the number of processors. A number of factors including load imbalance and communication overhead affect parallel efficiency. Load imbalance is an uneven distribution of work between processors, which causes less heavily loaded processors to reach a synchronization point in the code sooner and then wait for more heavily loaded processors. Load imbalance can be corrected by moving work between processors, as long as the cost to redistribute the work and the cost of possibly increased communication do not outweigh the benefits of the more efficient distribution. Communication overhead is the cost in time of sending and receiving messages between processors. Typically, the cost to initiate a message is equivalent to the cost to send hundreds of four-byte words once startup has occurred. This is fairly high latency. Therefore, there is an advantage to sending messages that are as large as possible, by blocking (aggregating) into single messages smaller messages that can be sent at the same time. Communication cost can also be improved by using asynchronous communication. Processors need stop only to initiate a send or a receive and may then go forward with computation that does not depend on the data that is being communicated. Thus, at least some of the cost is hidden by doing useful work at the same time.

Message-passing code and the additional code required to block messages into large buffers, to hide communication cost with asynchronous messages, and to redistribute work for load balancing increase the complexity of writing parallel codes. Compilers that can automatically parallelize codes for distributed-memory computers are a potentially important future technology, but not one that is workable at the present time. Therefore, data movement between processors in an massively parallel processor (MPP) or a cluster of workstations must, at some level, be explicitly coded using *send*, *receive*, and other system calls that implement message passing on the parallel machine. The coding process is manual and therefore more prone to error; it may also introduce machine dependencies that hinder portability; and it certainly introduces distributed-memory-specific code. The RSL library on which the parallel MM5 is implemented addresses these problems by encapsulating low-level messaging operations within higher-level functions that are specific to the application.

3.1 *RSL*

RSL is a run-time system and library to support parallelization of grid-based finite-difference weather models with a large nondynamic computational component (physics) and supporting mesh refinement. The RSL interface to the parallel machine is abstract and high level, thereby simplifying the programming task. All details of the underlying message passing — buffer allocation, copying, routing, and more complicated tasks such as asynchronous communication — are encapsulated within high-level routines for stencil exchange or moving forcing data between domains for nesting.

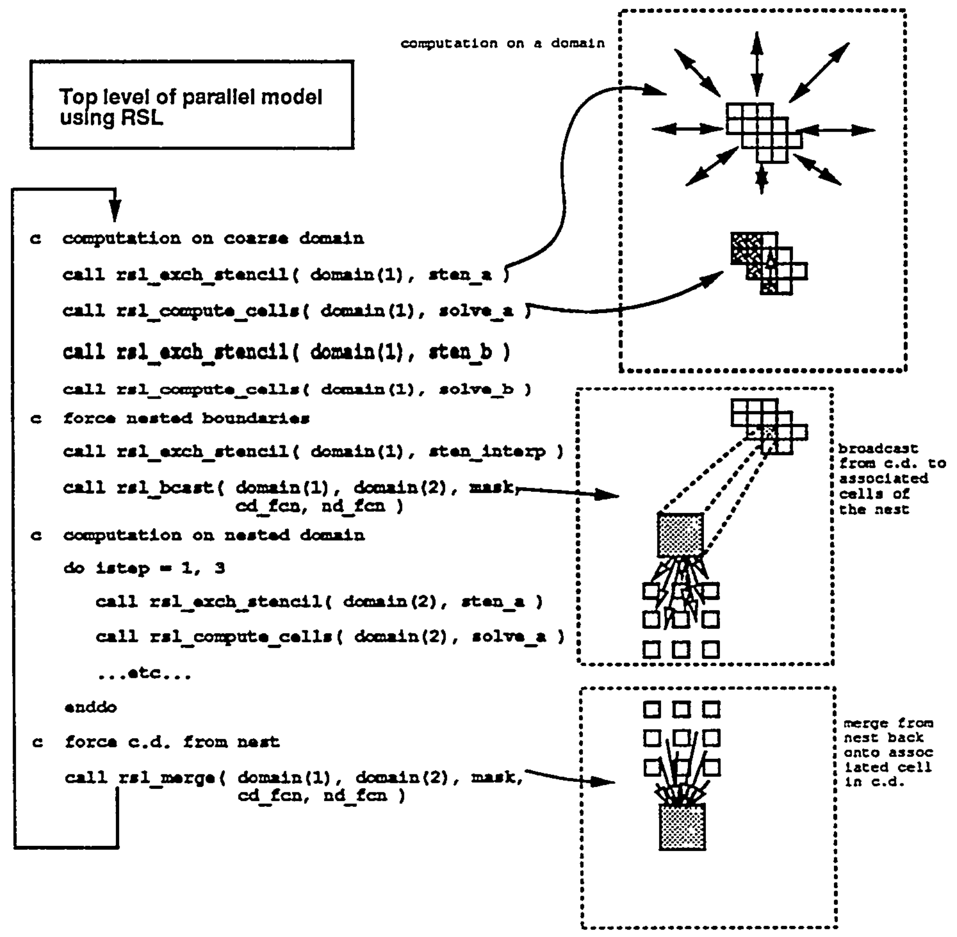
By focusing on a type of application, RSL can be lightweight and efficient, imposing little additional overhead or wasted capability. Intra-domain communication (messages required to satisfy data dependencies arising from stencils for finite differencing and interpolation on a horizontally decomposed domain) and inter-domain communication (messages required to communication forcing data between domains, i.e., between a parent domain and a nest) are specified abstractly as *stencils* and *broadcast/merges*, which RSL converts to the appropriate low-level communications between processors. RSL employs a technique called run-time compilation of communication schedules. Once a stencil is defined by specifying the model variables and stencil points involved for a transfer, the stencil is then compiled. During stencil compilation, RSL precomputes the interprocessor communication schedule (i.e., the sequence and contents of messages) to satisfy the stencil. Thus, when the exchange occurs during the model run, very little additional overhead is required to pack and deterministically exchange messages between the processors.

RSL supports a logical view of the model domain as an aggregation of *column processes*, each of which is a one-dimensional (vertical) expression of the model code for a single i, j mesh point. Expressed in this way, the model is said to be *column callable.* The columnwise expression of the model is more natural with respect to model column-physics, and it supports a more modular approach to mesh refinement via nesting. It also provides for small units of work (single columns) and allows for irregularly shaped allocations of work to processors, thereby facilitating load balancing.

Additional information on RSL can be found in [3].

3.2 *Parallel MM5*

Figure 1 illustrates the top-level structure of the parallel model once the code has been converted so that the model may be called separately for each column of the grid. At the start of a new time step, data is exchanged in the call to RSL_EXCH_STENCIL using the communication schedule that was previously defined and run-time-compiled for the stencil `sten_a`. This satisfies the horizontal data dependencies by updating ghost regions around the processor's local partition of data, which may be irregularly shaped. Next, RSL_COMPUTE_CELLS is called and applies the first phase (`solve_a`) of the model computation to each locally stored grid cell. The RSL exchange and compute cells are called in pairs for each phase in the computation


Top level of parallel model using RSL
computation on a domain
broadcast from c.d. to associated cells of the nest
merge from nest back onto associated cell in c.d.

```
c  computation on coarse domain
   call rsl_exch_stencil( domain(1), sten_a )
   call rsl_compute_cells( domain(1), solve_a )
   call rsl_exch_stencil( domain(1), sten_b )
   call rsl_compute_cells( domain(1), solve_b )
c  force nested boundaries
   call rsl_exch_stencil( domain(1), sten_interp )
   call rsl_bcast( domain(1), domain(2), mask,
                   cd_fcn, nd_fcn )
c  computation on nested domain
   do istep = 1, 3
      call rsl_exch_stencil( domain(2), sten_a )
      call rsl_compute_cells( domain(2), solve_a )
      ...etc...
   enddo
c  force c.d. from nest
   call rsl_merge( domain(1), domain(2), mask,
                   cd_fcn, nd_fcn )
```

Figure 1: Top level parallel driver for an MM5 time step with nest interactions

for a coarse domain, `domain(1)`, until it is time to provide forcing data to a nested domain, `domain(2)`. This is accomplished using RSL_BCAST and appropriate masking and interpolation routines that are passed as arguments. Data for the nested domain cells is transferred, across processor boundaries when necessary, and the computation of the time step on the nest begins. The same sequence RSL of stencil exchanges and compute calls occur, but with a different domain descriptor. Finally, data is merged back onto the coarse domain and the computation of the next time step begins.

The example in Figure 1 is simplified to show the relationship between a parent and a nest. In the actual parallel MM5 code, a stack is maintained to support a pseudo-recursive approach that permits nesting to arbitrary depth.

4. Validation

The validation effort for the parallel model involved determining that the model is correct with respect to the original version of the code. The source model has undergone validation with respect to observed phenomena and is, for this effort, assumed a priori to be correct.

Side-by-side plots of instantaneous model output fields provide a first indication of correctness. Figure 2 shows a comparison of the U/V streamline plot at ground level between the parallel code and the original model after three hours. All gross structure of the function within the original model is reproduced in the parallel code.

A more rigorous verification of correctness with respect to the original model is being conducted using the error-growth analysis technique of Rosinski and Williamson [4]. The technique involves measuring the growth in error in selected output fields between two runs of the sequential model, in which the second run has been initialized with data that has been perturbed by flipping the lowest-order bit in each floating-point value, thereby simulating the effect of rounding error. This measurement is compared with the error growth measured between the unperturbed sequential model and the parallel model. If the deviation with the parallel model is similar to that of the perturbed sequential model, the difference is no worse than what is expected from floating-point roundoff.

5. Performance

Figure 3 and Table 1 shows the result of a series of runs on the large IBM

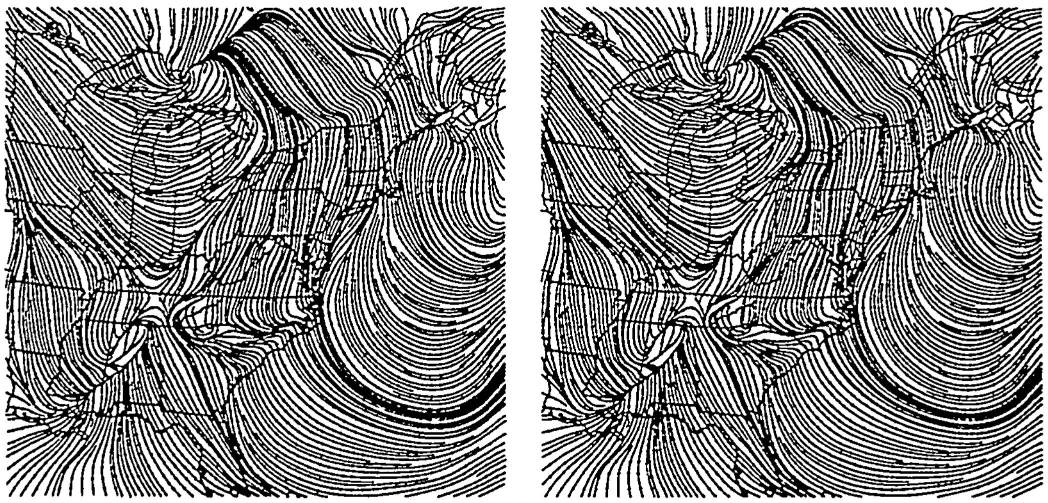


Figure 2: Plots of horizontal wind streamline plot three hours into the simulation (120 time steps) for a parallel MM5 run on a cluster of IBM RS/6000 workstations (left) and the original code run on a single RS/6000 workstation (right)

Scalable POWERparallel system at Argonne and on a 14-processor SP2. The larger machine is an SP1 with SP2 communication hardware (upgrade of the machine with SP2 processors is anticipated). It consists of 128 processors, each the equivalent of an IBM RS/6000 model 370 workstation with a 62.5 MHz clock, a 32-kilobyte data cache and a 32-kilobyte instruction cache. Theoretical peak performance of each workstation is 125 Mflop/second (one 64-bit floating-point add and one floating-point multiply in each clock cycle). In practice, each processor can achieve between 15 and 70 Mflop/second on Fortran codes. Each processor has 128 Mbytes of memory and 1 Gbyte of local disk. The message-passing hardware is a high-speed Omega switch providing 70μsec latency and 35 Mbytes/second bandwidth [2][5]. The smaller machine, an SP2 acquired specifically for this project, consists of 14 processors, each equivalent to an RS/6000 model 390 workstation, which has

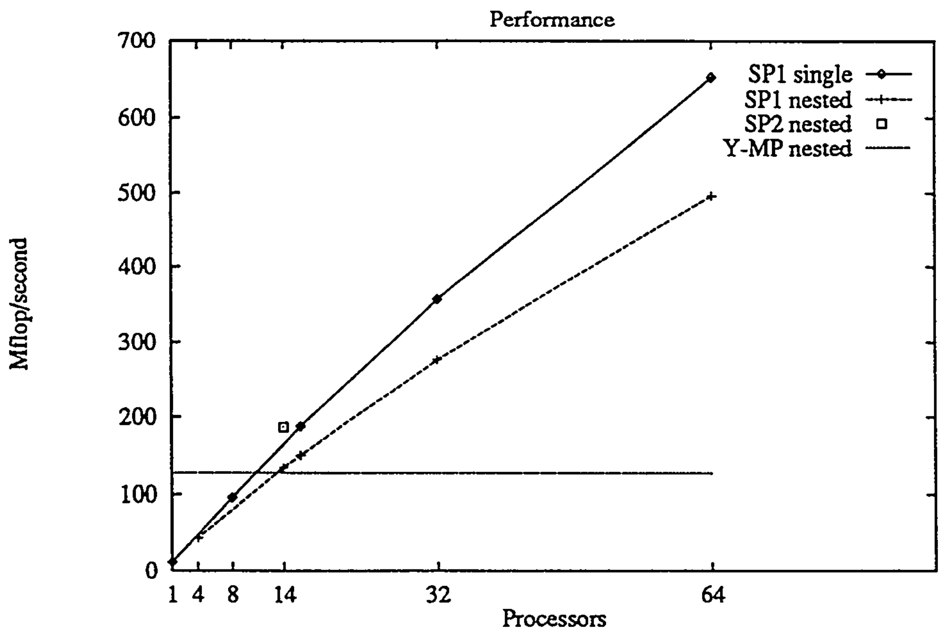


Figure 3: Performance with a 30-kilometer domain (91x91x23) by itself and with one 10-kilometer nest (52x52x23) on IBM SP1. Also, performance for a 14-processor run of the nested code on the IBM SP2 is shown. The horizontal line at 118 Mflop/second is the performance of one processor of a Cray Y-MP on the nested problem.

additional floating-point hardware for better performance.

The model was run with and without a nest. As expected, the nested code runs less efficiently because of the additional cost of interpolation between domains and because of inter-domain communication in the parallel model. Nevertheless, the nested code is still the most cost-effective way to achieve 10 km resolution over at least part of the modeled area.

6. Conclusion

The ten-month effort to produce a scalable parallel implementation of a real-time weather forecasting model has been completed. The model runs efficiently and generates meteorologically valid data; nesting is used to achieve high resolution. Future work will address dynamic load-balancing strategies

Table 1: MM5 performance on a single domain and with one nest on the IBM SP1, the SP2, and on 1 processor of a Cray Y-MP (shown for comparison). The performance is shown as seconds per 90-second time step, as the ratio of simulated time to real time, and as a floating-point rate. The single-domain problem seemingly exhibits perfect speedup between 1 and 8 processors on the SP1. In actuality the single-node performance is likely suppressed — perhaps because of a processor memory/cache effect. The floating-point rates are computed based on 3175 Mflop/time-step for the nested code and 1604 Mflop/time-step for the single domain code.

Platform	Procs.	sec/ts	sim:wall	Mflop/sec
Single domain: 91x91x23, 30 km				
IBM SP1	1	134.0	0.7	12
	8	16.7	5.4	96
	16	8.5	11.0	188
	32	4.5	20.0	356
	64	2.5	37.0	652
IBM SP2	1	82.0	1.1	20
	14	6.8	13.2	236
91x91x23, 30 km domain with 52x52x23 10 km nest				
IBM SP1	4	72.3	1.2	44
	14	23.5	3.8	135
	16	21.0	4.2	151
	32	11.5	7.8	276
	64	6.4	14.1	496
IBM SP2	14	17.0	5.3	187
Cray	1	26.8	3.4	118

that exploit the ability to migrate columns at run time, other performance tuning to achieve larger percentages of theoretical peak performance on the individual processors, irregularly shaped and dynamically moving nested domains, tools for automated columnwise decomposition of existing models, data assimilation, and scalable parallel I/O.

Acknowledgements

We acknowledge Ying-Hwa Kuo, Jimy Dudhia, David Gill, and Sue Chen of the NCAR Mesoscale and Microscale Meteorology Division for providing the original model, data sets, and expertise. We also acknowledge the contribution of Ian Foster of the Mathematics and Computer Science Division at Argonne, whose work formed the basis for the current work.

References

[1] G. A. Grell, J. Dudhia, and D. R. Stauffer, *A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5)*, Tech. Rep. NCAR/TN-398+STR, National Center for Atmospheric Research, Boulder, Colorado, June 1994.

[2] W. Gropp, E. Lusk, and S. Pieper, *Users Guide for the ANL SPx*, Tech. Rep. ANL/MCS-TM-198, Mathematics and Computer Science Division, Argonne National Laboratory, 1994. (http://www.mcs.anl.gov/sp1/guide-r2/guide-r2.html).

[3] J. Michalakes, *RSL: A Parallel Runtime System Library for Regular Grid Finite Difference Models Using Multiple Nests*, Tech. Rep. ANL/MCS-TM-197, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1995 (in press).

[4] J. M. Rosinski and D. L. Williamson, *On the accumulation of rounding errors in a global atmospheric model*, NCAR preprint, (1994).

[5] C. B. Stunkel, D. G. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao, *The SP1 High Performance Switch*, in Proceedings of the Scalable High Performance Computing Conference, IEEE Computer Society Press, Los Alamitos, California, 1994, pp. 150–156.