MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5)

J. Michalakes

Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, Illinois, U.S.A. 60439

This paper describes MM90, a parallel regional weather model based on the Penn State/NCAR MM5. Parallelization of finite differencing, horizontal interpolation, and nesting on distributed-memory (message-passing) computers is handled transparently using the RSL library package. Fortran90 modules, derived data types, dynamic memory allocation, pointers, and recursion are used, making the code modular, flexible, extensible, and run-time configurable. The model can dynamically sense and correct load imbalances. The paper provides performance, scaling, and load-balancing data collected on the IBM SP2 computers at Argonne National Laboratory and NASA Ames Laboratory. Future work will address the impact of parallel modifications on existing modeling software; an approach using commercially available source translation software is described.

1 Introduction

Regional models of the earth's atmosphere provide high resolution for weather forecasts and climate predictions but require large amounts of computer power. The need to run larger and longer scenarios at higher resolutions with more sophisticated physics will continue to exceed capabilities of nonscalable computer architectures. This paper describes a multiyear effort at Argonne National Laboratory to develop tools and techniques for implementing regional weather models on scalable, distributed-memory computers, sometimes called massively parallel processors (MPPs), and other distributed-memory computing environments, such as networks of workstations. The focus of this effort has been the development of MM90, a parallel implementation in Fortran90 of a well-known and widely used regional weather model, the Pennsylvania State University/National Center for Atmospheric Research Mesoscale Model, MM5. This work addresses the dual and often conflicting concerns for good performance and maintainable software.

Achieving good, scalable performance requires that the resources of the parallel machine be used efficiently. Each processor operates on the data in

its local memory and exchanges data with other processors when operands are not available locally. Communication, idle time from uneven workload or processor capability, and I/O costs must be addressed to give efficient scaling (relative to perfect scaling, where execution time is inversely proportional to the number of processors). Scaling in memory is also a consideration, since it allows the aggregate memory available on MPPs to be used to run very large problems. Parallelization is described in Section 2. Section 3 describes dynamic load balancing in MM90. Performance benchmark results are given in Section 4.

Software is an issue insofar as modifications segregate the parallel code from the officially maintained version of MM5. Although many details of the parallelization have been encapsulated within a parallel library, a troublesome amount of parallel artifact remains. Thus, MM90 has evolved into a separate, nonstandard version of MM5. Section 5 describes effort now underway to employ source translation technology to make virtually all modifications for distributed-memory parallelism transparent. The result will be a true single-source expression of MM5 that provides good performance on diverse high-performance architectures.

1.1 MM5, MPMM, and MM90

The Penn State/NCAR model is a limited-area (as opposed to global) model of atmospheric systems over regions ranging from several thousand to several hundred kilometers. Originally developed in the late 1970s [1], the model is now in its fifth generation, MM5 [3]. It is a primitive-equations model employing finite differencing on an Arakawa-B grid for the computation of atmospheric dynamics (advection and diffusion) under both hydrostatic and nonhydrostatic assumptions, both of which employ time splitting. In the hydrostatic case, a split-explicit scheme is used to handle fast-moving gravity waves without reducing the overall model time step. In the nonhydrostatic case, which is fully compressible and permits sound waves, a split semi-implicit method is used. Vertical propagation of sound waves is treated implicitly, allowing the length of the split time step to be independent of vertical resolution. Initial conditions and forcing for lateral boundaries come from external sources: global atmospheric models and observations. MM5 allows multiple nested grids with two-way interaction between nest levels. Four-dimensional data assimilation (FDDA) is provided as an option for incorporating observations during a run. The model is maintained in the public domain by NCAR for internal use and by three-hundred groups at institutions worldwide for weather forecasting, regional climate prediction, air-quality, and storm research.[1]

The Massively Parallel Mesoscale Model (MPMM) [2][8] was developed by Argonne researchers beginning in 1992 on the Intel Touchstone Delta com-

[1] The MM5 source code and other information are available from http://www.mmm.ucar.edu/mm5.

Table 1
MM5 versions and options supported in parallel MPMM and MM90 codes

		MM5v1	MPMM	MM90	MM5v2
Dynamics	Hydrostatic	•			•
	Non-hydrostatic	•	•	•	•
PBL	Bulk	•			•
	Blackadar	•	•	•	•
	Burke-Thompson	•	•		•
Cumulus	Grell	•	•	•	•
	Anthes-Kuo	•	•	•	•
	Kain-Fritsch				•
	Fritsch-Chappel				•
	Betts-Miller				•
	Arakawa-Schubert				•
Radiation	Simple	•	•	•	•
	Dudhia	•	•	•	•
	CCM2				•
Moisture	Warm rain	•	•	•	•
	Mixed phase	•	•	•	•
	Grauple (GSFC)				•
	Grauple (Reisner)				•
FDDA	Analysis	•	•		•
	Observational	•			•

puter (Caltech). Development has continued on the IBM SP2 at Argonne with ports to other parallel systems: Intel Paragon, Cray T3D/E, Fujitsu AP1000, SGI Power Challenge, and networks of workstations. MM5 was parallelized using the Runtime System Library (RSL), a communication and portability library that provides high-level communication, automatic two-dimensional irregular domain decomposition, support for remapping for dynamic load balancing, automatic local/global index translation, and distributed I/O [6]. The library is tailored to the needs of finite-difference regular grid weather models with multiple nests. It uses vendor-specific message-passing layers when available (NX on the Paragon, MPL on the IBM SP2), and MPI on other platforms. RSL was developed at Argonne National Laboratory in conjunction with MPMM and MM90. It is similar to packages developed by a number of other groups [4][5][9].

MPMM and its follow-on Fortran90 implementation, MM90, are functionally equivalent in most respects to the 1995 release of MM5 (MM5v1). The parallel model is strictly nonhydrostatic, although a separate hydrostatic version of MPPM has been implemented by Y. Kim at Iowa State. Since the release of MM90, NCAR has released MM5 version 2 (MM5v2). Options in the parallel code are listed in Table 1.

MM90 uses new features in Fortran90 — dynamic memory allocation, modules, structures, pointers, and recursion — to make the model more flexi-

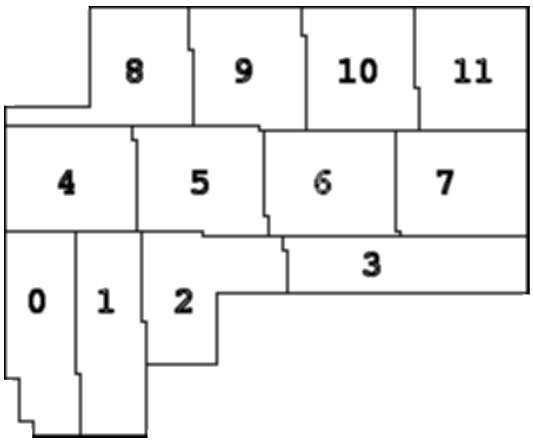


Fig. 1. Pointwise decomposition of an irregularly shaped domain over 12 processors using RSL's built-in partitioner. The number of grid-points per processor differs by no more than 2.

ble and run-time configurable. Building on this flexibility, MM90 includes advanced features such as dynamic load balancing and irregularly shaped nests, which replace overlapping nested grids in the source model. MM90 is the forecast model component of the prototype Global Theater Weather Analysis and Prediction System (GTWAPS) at the U.S. Air Force Global Weather Central (AFGWC). MPMM is used on the Cray T3D at the U.S. EPA National Environmental Supercomputing Center (NESC) to provide meteorological input for air-quality modeling. MM90 and MPMM are also used for study of regional climate at Australian National University, Iowa State University, and the Fraunhofer Institute (Germany).

1.2 Acknowledgments

Argonne researchers T. Canfield, K. Dritz, S. Hammond, I. Foster, J. Mogill, and R. Nanjundiah contributed to the design and implementation of MPMM and MM90. NCAR researchers J. Dudhia, G. Grell, and W. Kuo provided access to and assistance with MM5. Y. Kim at Iowa State University, J. Larson and D. Sitsky at Australian National University, and V. Wayland at Cray Research contributed to the code development. The U.S. Air Force and the U.S. EPA sponsored the MPMM and MM90 projects. Argonne National Laboratory (under U.S. Department of Energy contract W-31-109-Eng-38) and the NASA Numerical Aerospace Simulation Facility are gratefully acknowledged for grants of machine time and assistance.

2 Parallelization

MM5 is parallelized using two-dimensional data domain decomposition. Two- and three-dimensional data arrays containing state variables — wind velocity, temperature, moisture, and pressure — as well as diagnostic and inter-

mediate fields are partitioned in two dimensions (north/south and east/west) and the resulting subdomains are distributed over processors. Extra memory is allocated at processor subdomain boundaries for ghost points to store off-processor operands for finite differencing and horizontal interpolation.

RSL automatically decomposes a domain when it is defined or, in the case of a nest, spawned. Each domain is decomposed independently over the same set of processors; that is, each processor has a portion of every domain in the simulation. In RSL a grid point is the unit of work that is mappable to processors, and subdomains are not constrained to any particular shape. This approach supports load balancing by allowing more precise allocation of work to processors than strategies that constrain subdomains to rectangular shapes. Pointwise decomposition is especially useful if the domains themselves are not rectangular (Figure 1). Domains may be remapped at run time to address load imbalances.

Decomposing domains over processors generates the need for two types of communication, intra- and interdomain (Figure 2). *Intradomain* communication is characterized by the size and shape of the computation's stencil and is handled by a *stencil exchange* in RSL. Stencil exchanges are inherently "nearest neighbor" and therefore well suited to provide good performance on distributed-memory parallel computers. By precomputing communication schedules, RSL is able to execute an exchange with just two messages (one in each direction) per processor pair per stencil exchange, a benefit on machines where the cost of starting a message (latency) is significant (e.g., IBM SP2 and Intel Paragon). In addition, RSL allows many fields to be communicated in a single exchange, facilitating *hoisting*, the combining of exchanges to minimize the number per time step. Thus, MPMM and MM90 require only twelve exchanges per time step: four stencil exchanges during the main nonhydrostatic solver plus two exchanges for each of four minor iterations of a solver for sound waves.

Interdomain communication is necessary to parallelize nest forcing and feedback in MM5. RSL establishes logical communication streams between associated points in the parent and nest and provides routines for scattering and gathering data over these streams. As with stencil exchanges, RSL precomputes schedules so that the underlying communication is efficient.

2.1 Memory Scaling and Local Iteration

Model data structures are shrunken in the north/south and east/west dimensions, using only as much memory as needed on each processor. RSL returns the minimum local array size whenever a domain is decomposed (when it is defined or when it is remapped for load balancing). The MM90 code uses this information to dynamically allocate grid arrays using the Fortran90

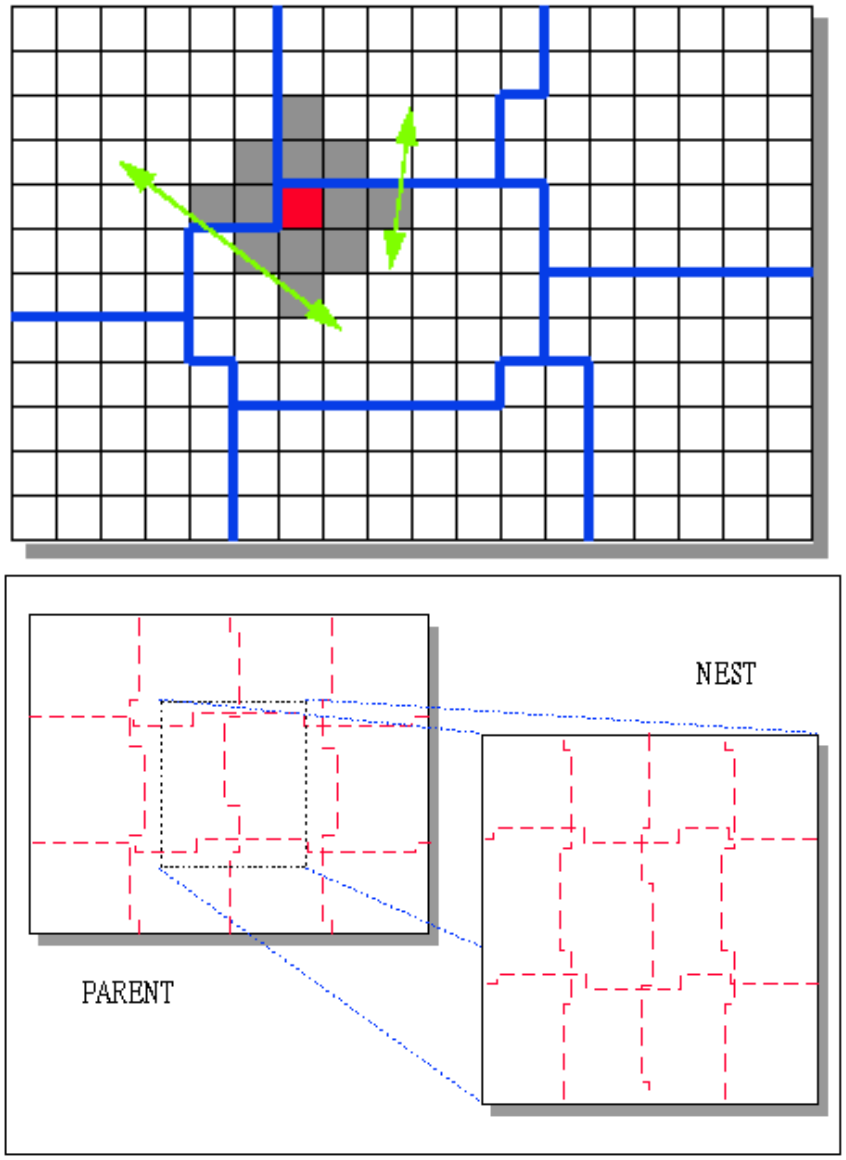


Fig. 2. Intra- and interdomain communication

ALLOCATE statement.[2]

Shrinking local data structures for memory scaling requires attention to iteration over decomposed dimensions, since the implicit identity between logical and memory indices no longer holds. The logical indices of a grid-point may not be used to access an array in local memory; nor may local indices be used to test for proximity with a domain boundary. The pointwise-decomposition strategy employed by RSL further complicates horizontal iteration. RSL provides programming constructs and run-time information for iterating over the local subdomain on each processor and for translating between logical and

[2] MPMM, the Fortran77 version, uses static memory allocation at compile time. The size information returned by RSL is used as a run-time check.

```
recursive subroutine iterate_model( d, start_time, end_time )
use domains_module
type ( domainstruct ) d          ! structure for this domain
...
do while ( d%xtime .ge. start_time .and. d%xtime .lt. end_time )
  call time_step( d )
  do kid = 1, maxkids
    if ( d%active( kid ) ) then
      call force( d, d%child(kid) )
      call iterate_model( d%child(kid), d%xtime-d%dt, d%xtime )
      call feedback( d, d%child(kid) )
    endif
  enddo
enddo
return
```

Fig. 3. Using Fortran90 recursion for iteration over domain hierarchy. Top-level call is CALL ITERATE_MODEL(MOTHER, 0., TIMAX).

memory indices.

2.2 I/O

MM5 outputs 13 three-dimensional and 13 two-dimensional fields each time it generates a history write; depending on the configuration this amounts to between five and ten Mbytes every three, six, or twelve model hours. RSL implements distributed I/O using a single-reader/single-writer strategy: one processor performs Fortran record-blocked I/O and distributes or collects the data to or from the other processors. This strategy permits MM90 to read and write standard MM5 data sets but is not scalable. Fortunately, output is infrequent and small compared with the amount of computation. Further, the cost of reading and writing the file system far exceeds the cost of interprocessor communication. Hence, addressing I/O speed on the reader/writer processor by ensuring that the processor reads and writes local disk or some other fast file system (e.g., PIOFS on the IBM SP2) has proven sufficient (Section 4). Future output requirements or increased input volume for FDDA may require revisiting this problem.

2.3 Fortran90 in MM90

MM90 employs a number of Fortran90 features — abstract data types, pointers, dynamic memory, and recursion — to achieve a more modular, flexible, and run-time configurable model suitable for distributed memory parallel computers:

– COMMON data and the myriad include files have been replaced by a Fortran90 MODULE, containing definitions of the *domain* data type and routines

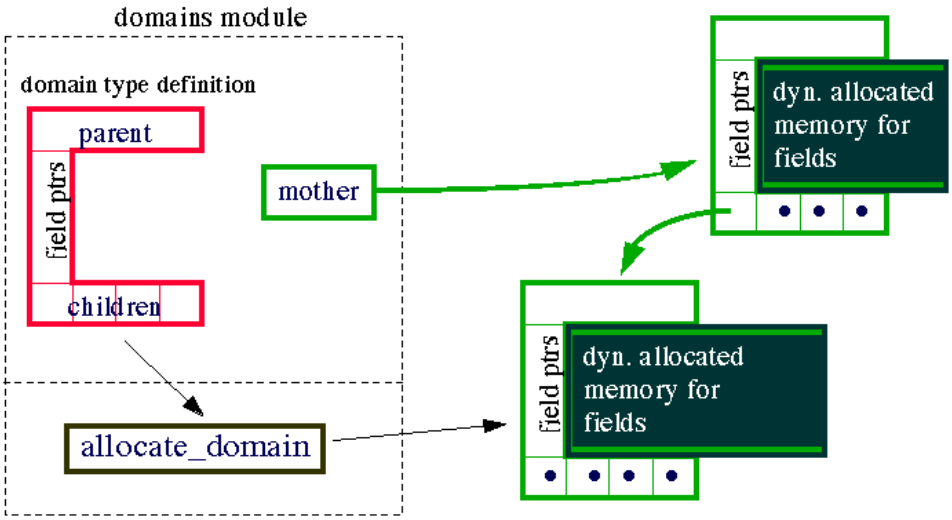


Fig. 4. MM90 nest-hierarchy implemented as nodes-and-pointers; each node is a dynamically allocated domain structure.

for manipulating (e.g., allocating or deallocating) instances of this type.

- Each domain (grid) is an instance of a dynamically allocated derived data type (structure). New nests can be activated or deactivated at any time during the run. Crucial for load balancing, data structures of an active domain can be resized if remapping causes a change in the distribution of points to processors.
- The hierarchy of nested domains is represented by a tree of domain structures connected by parent and child pointers and rooted at the global domain pointer variable MOTHER. The top level of MM90 employs recursion to traverse the hierarchy (Figure 3).
- The model is 100 percent run-time configurable. Whereas changing nesting and physics options in the MM5 code often requires recompilation, the MM90 code is configured entirely through the namelist.

The principal object in MM90 is the *domains module*. It contains the definition of a *domain structure* (a Fortran90 derived data type) and a pointer to the root of a tree of domain structures that represent the hierarchy of nesting (Figure 4). A domain structure (Figure 5) contains domain-specific scalar data and all two- and three-dimensional state arrays for the domain, defined as "deferred-shape" arrays. These have no memory until explicitly allocated. This approach allows run-time allocation of memory in the size and shape necessary for the local partition initially and during remapping.

3 Dynamic Load Balancing

Load balancing in MM90 is implemented by identifying the key computational segments of the code and then inserting instrumentation that measures the cost of computing each column in those segments. The segments are

(i) Planetary boundary layer and radiation;
(ii) Calculation of horizontal diffusion coefficients;
(iii) Horizontal and vertical advection. diffusion, cumulus, explicit moisture , convective adjustment, boundary nudging;
(iv) Split semi-implicit solver for sound waves; and
(v) Interpolation for nest boundary forcing.

Several sections known to be lightweight were omitted or combined into the ones shown.

The number of milliseconds to compute column i, j in a segment is accumulated into the corresponding entry of a two-dimensional array of timers for that segment. Periodically, a new mapping Π^{new} is computed, and the efficiency of the new mapping is compared with the old mapping. The new mapping is adopted if it improves efficiency by more than ε.

$$\frac{\sum T}{P \max_p \Pi^{new}(p,T)} > \frac{\sum T}{P \max_p \Pi^{old}(p,T)} + \epsilon$$

T denotes the array of timers with an entry for each point in the domain, and $\sum T$ is the total time for all points in the domain. $\Pi(p,T)$ denotes total time for the columns of T allocated to processor p under mapping Π. P is the number of processors. Performance of the model with dynamic load balancing is discussed in Section 4.

4 Performance

Concern for model performance falls into two categories: "raw" performance and scaling. Raw performance, or time-to-solution is usually (and justifiably) the bottom line for users. Scaling is important as a measure of machine utilization and provides the ability to increase capabilities by simply adding processors. Benchmark results for MM90 were gathered on the IBM SP2 at the Numerical Aerospace Simulation (NAS) facility at NASA Ames laboratory.[3]

[3] The NAS SP2 is composed of 160 RS/6000-590 processing nodes on a high-speed switch for message passing. Additional information is available at http://lovelace.nas.nasa.gov/Parallel/ SP2.

```
type domainstruct
  integer il,jl,jl
  integer stencil_a, stencil_b,...
  real, pointer :: psa(:,:),tga(:,:),...   ! 2d arrays
  real, pointer :: ua(:,:,:),va(:,:,:),... ! 3d arrays
  type (domainstruct), pointer : parent, child(:)
endtype domainstruct
type (domainstruct), target :: mother      ! root
```

Fig. 5. Fortran 90 code defining a skeletal domain structure

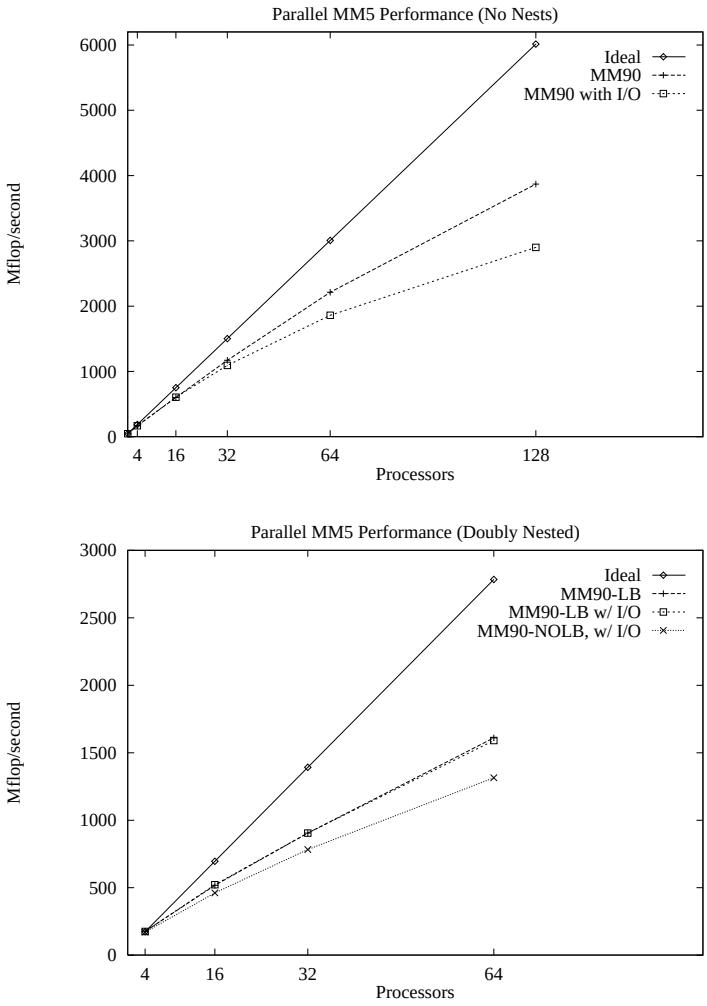
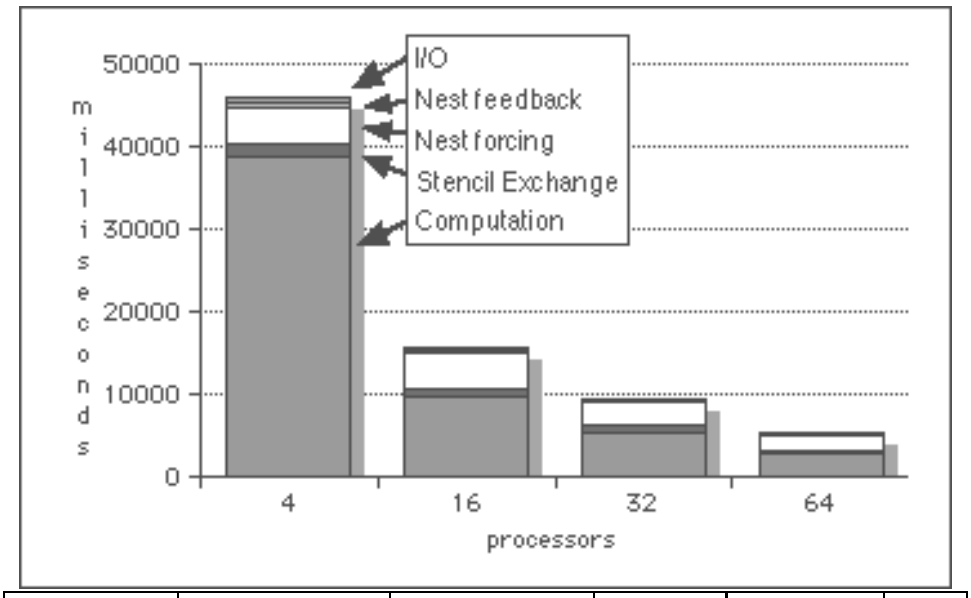


Fig. 6. Performance without and with nesting

All runs were conducted with single (32-bit) floating-point precision.

Figure 6 shows model performance in Mflop/sec for two scenarios, one without and one with nesting.

The first scenario is a single 100-km resolution domain with grid dimensions 61 by 61 horizontally and 23 levels. The run is nonhydrostatic with full physics including explicit moisture with mixed-phase ice, cumulus (Grell), radiation (Dudhia), PBL (Blackadar), and an upper radiative boundary condition. The relatively coarse resolution would not usually warrant nonhydrostatic numerics. However, for comparison with higher-resolution nested runs (and because the hydrostatic solver is not parallelized in MM90), all runs were done nonhydrostatically. The scenario, centered over the Korean peninsula is from July 1995, during Typhoon Faye. Lateral boundary conditions are read every 12 hours; output data is generated every 6 simulation hours. The floating-point rates are based on an estimated 716 Mflop/time step. The model runs at 47 Mflop/sec on 1 processor and 2.9 Gflop/sec on 128 proces-



Processors	Computation	Stencil exch.	Forcing	Feedback	I/O
4	84.7	3.0	9.5	1.7	1.1
16	61.8	7.6	27.4	2.4	.9
32	56.4	8.8	30.2	3.0	1.5
64	51.8	9.8	32.1	3.4	3.8

Fig. 7. Cost as a percentage of total run time in doubly nested runs

sors, including time spent performing I/O. A 36-hour forecast (5-minute time steps) completes in 1 hour and 50 minutes on 1 processor and in 110 seconds on 128 processors I/O comprises 7 percent of total run time on 32 processors, 14 percent on 64 processors, and 25 percent on 128 processors. The apparently high cost for I/O is explained by the fact that the total run time for a 36 hour forecast is under 2 minutes, indicating a small computational problem to begin with. Efficiency is 70 percent on 32 processors, 62 percent on 64 processors, and 48 percent on 128 processors. Since the effect of dynamic load balancing on the single-domain runs is nominal, only the non-load balancing times are shown.

The second scenario adds two nests to the original scenario: a 33-km nest within the mother domain and an 11-km nest within the 33-km nest. The nests have the same grid dimensions as the mother. Cost per five-minute time step is 8054 Mflop. The model runs at 176 Mflop/sec on 4 processors (the scenario is too large to run on a single processor) and at 1592 Mflop/sec on 64 processors; results from a 128-node run were not available. Although there are thirteen times as many iterations per five-minute time step — one at 100-km, three at 33-km, and nine at 11-km — the number of floating point operations is only 11.25 times greater. This is because radiation calculations are done at a fixed interval of thirty minutes, regardless of resolution; therefore the

ratio of non-radiation steps to costly radiation steps increases as resolution becomes more fine. Results from a 128-node run were not available. On the considerably larger doubly nested problem, I/O comprises only 1.6 percent of total run time on 32 processors and 4 percent on 64 processors. Efficiency is 74 percent on 16 processors, 64 percent on 32 processors, and 57 percent on 64 processors. A 36-hour forecast completes in 5.5 hours on 4 processors and in 36 minutes on 64 processors, including I/O.

Figure 7 shows a detailed breakdown of the component costs of the doubly nested scenario. Computation scales well, and the communication associated with stencil exchanges also stays under 10 percent; I/O is also not a major concern. The principal source of inefficiency is nest forcing. The inefficiency is not communication, however. Nest feedback, which communicates considerably more data than forcing — data from the nest interior is fed back while only the nest boundaries are forced — has only a modest share of the cost. Rather, the problem is a costly and imbalanced interpolation computation during forcing that involves the parent-domain points over the nest boundaries. We are addressing this imbalance by distributing the interpolation work more evenly over processors without migrating the points themselves, as is done to address other forms of load imbalance.

Dynamic load balancing in MM90 can handle load imbalances that result from poor guesses at initial decompositions and load distributions that change over time, as a function of either the state of the model (different paths being executed in model physics depending on the state of the atmosphere) or the state of the computing environment. Dynamic load balancing significantly improved the doubly nested NAS runs (Figure 6). Efficiency from 4 through 64 processors was 57 percent with load balancing but only 48 percent without. Nevertheless, in its current configuration, MM90 does not exhibit a high degree of physics-induced dynamic load imbalance. For this paper, a dynamic load imbalance is simulated within an eight-hour single-domain simulation using 32 processors of the Argonne SP2 to test the model's ability to rebalance the load automatically (Figure 8).

For the first 2.5 hours (Period A), the model runs normally. The maximum processor time is 15.0 seconds and the mean is 14.1 seconds, giving an efficiency of 94 percent (T_{mean}/T_{max}). At hour 2.5, a two-fold increase in load is induced within a centered circle, radius 1/5 the width of the grid. This increases maximum processor time to 35.1 seconds; the mean is 17.4 seconds (Period B). Computational efficiency drops to 50 percent. MM90 detects and corrects the imbalance at hour 3.5 by moving work off the processors in the center of the domain (Period C). As a result, efficiency rebounds: the maximum processor time is 19.6 seconds, and the mean is 18.9 seconds. Efficiency is restored to 96 percent. At hour 4.5, the induced load finishes, and the distribution again becomes imbalanced because the interior cells are now too lightly loaded (Period D). Efficiency drops to 78 percent. The remapping step at hour 5.5 corrects this and restores efficiency to 95 percent (Period E).

Full rebalancing ($T_{map} + T_{move}$) costs 12 seconds. Calculating a remapping

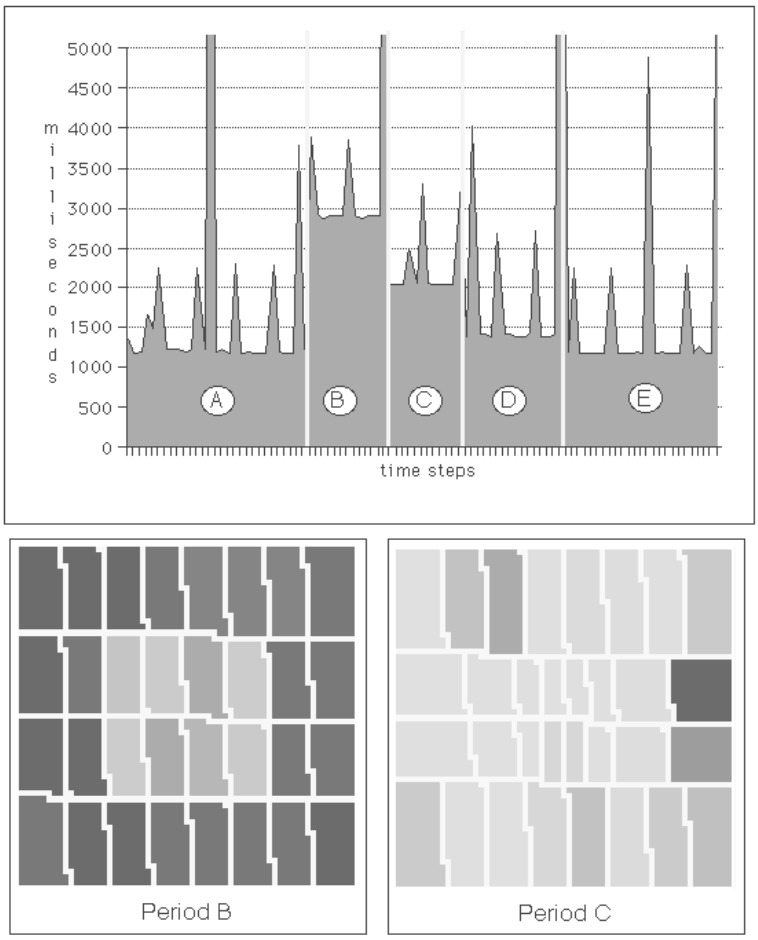


Fig. 8. Correcting an artificially induced load imbalance in an 8-hour simulation on 32 processors. The top figure is time per time step, in milliseconds; each tick on on the x-axis is an MM90 time step. The two lower figures show the decomposition and resulting processor loads for two of the periods in the run (B and C). Lighter grey denotes greater work, but the shading scales between the two periods are different. Workloads between processors in Period C are actually closer together than in Period B.

but retaining the old one (T_{map} only) costs 2.5 seconds. Continued work will focus on improving the efficiency of the mapping mechanisms and refining the performance model for dynamic load balancing in MM90.

5 Source Translation

MM90 provides both an excellent testbed for parallel computing research and a production-quality weather model suitable for meteorological research

and operational weather forecasting. However, it is a separate code from the official MM5 and has not kept up to date with the official version. Essentially finished for the Air Force at the end of 1995, MM90 was soon a version behind the NCAR model, with the release of MM5v2 in 1996.

A number of solutions have been proposed to address the problem of maintaining a model a diverse computer architectures. Alternatives include conditional compilation, parallelizing data-parallel compilers, the use of directive-driven preprocessors, writing and maintaining separate versions, or accepting suboptimal performance (or no port at all) for other architectures.

Source translation based on lexical, syntactic, and semantic analysis and transformation of a code appears a most promising strategy. Source translation can automatically detect loops over decomposed dimensions and generate the proper looping expression for the architecture, thereby facilitating local iteration for distributed memory, multi-tasking for shared-memory, and restructuring for cache-blocking or vectorization. Source translation can also automatically and interprocedurally identify data dependencies and generate efficient communication. With run-time system support (e.g., RSL), source translation can easily address irregular domains and domain decompositions for load balancing. Source translation can target different underlying run-time system libraries, as needed for a particular architecture or installation. A project involving model developers at NCAR and software experts Applied Parallel Research Inc. is underway to develop this approach for MM5 [7].

6 Conclusions

MM90 is an efficient, scalable, and largely functionally equivalent implementation of the Penn State/NCAR MM5 model. It is useful both as a testbed for parallel computing research and for operational weather forecasting and atmospheric research. It supports nesting and dynamic load balancing. It employs modules, derived data types, dynamic memory allocation, pointers, and recursion in Fortran90 to produce a more modular, flexible, extensible, and run-time configurable expression of the model. Many modifications for parallelism, such as message passing and data domain partitioning over processors, are handled transparently using RSL. The final step, source translation, will complete the task of hiding remaining parallel artifacts relating to iteration over decomposed dimensions and boundary computations.

References

[1] R. Anthes and T. T. Warner, *Develpment of hydrodynamic models suitable for air pollution and other mesometeorological studies*, Mon. Wea. Rev., 106 (1978), pp. 1045–1078.

[2] I. Foster and J. Michalakes, *MPMM: A Massively Parallel Mesoscale Model*, in Parallel Supercomputing in Atmospheric Science, G.-R. Hoffmann and

T. Kauranne, eds., World Scientific, River Edge, New Jersey, 1993, pp. 354–363.

[3] 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.

[4] R. Hempel and H. Ritzdorf, *The GMD Communications Library for Grid-oriented Problems*, Tech. Rep. GMD-0589, German National Research Center for Information Technology, 1991.

[5] S. R. Kohn and S. B. Baden, *A Parallel Software Infrastructure for Structured Adaptive Mesh Methods*, in Proceedings of Supercomputing '95, IEEE Computer Society Press, 1996.

[6] 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, December 1994.

[7] ——, *FLIC: A Translator for Same-source Parallel Implementation of Regular Grid Applications*, Tech. Rep. ANL/MCS-TM-223, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, March 1997.

[8] J. Michalakes, T. Canfield, R. Nanjundiah, S. Hammond, and G. Grell, *Parallel Implementation, Validation, and Performance of MM5*, in Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Procesors in Meteorology, World Scientific, River Edge, New Jersey, 1995, pp. 266–276.

[9] B. Rodriguez, L. Hart, and T. Henderson, *A Library for the Portable Parallelization of Operational Weather Forecast Models*, in Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology, World Scientific, River Edge, New Jersey, 1995, pp. 148–161.