RSL: A PARALLEL RUNTIME SYSTEM LIBRARY FOR REGIONAL ATMOSPHERIC MODELS WITH NESTING

JOHN G. MICHALAKES*

Abstract. RSL is a parallel runtime system library developed at Argonne National Laboratory that is tailored to regular-grid atmospheric models with mesh refinement in the form of two-way interacting nested grids. RSL provides high-level stencil and interdomain communication, irregular domain decomposition, automatic local/global index translation, distributed I/O, and dynamic load balancing. RSL was used with Fortran90 to parallelize a well-known and widely used regional weather model, the Penn State/NCAR Mesoscale Model.

Key words. Weather modeling, parallel computing, mesh refinement, dynamic load balancing.

1. Introduction. Models of the earth's atmosphere were among the first applications for supercomputers and continue to push the limits of available resources today [3]. Dynamic models of the atmosphere are used for forecasting and climate prediction. Such models may be categorized as global and regional. Global models provide relatively low-resolution predictive capabilities and are crucial to providing large-scale long-range simulations. Regional models provide higher (and more costly) resolution over a limited area for modeling effects of complex terrain, simulating high-gradient features such a fronts, and "downscaling" — generating high-resolution input for other simulations such as atmospheric chemistry models.

The application of adaptive mesh refinement to regional weather models is an area of active research. The problem for weather models is the representation of small-scale features (clouds, complex terrain) in large-scale atmospheric flows while conserving computation [2]. The solution is nesting — the ability to create or delete finer subgrids in a background mesh to obtain a given level of accuracy with a minimum number of grid points. The Penn State NCAR Mesoscale Model (MM5), for example, uses an approach involving quasi-uniform grids [8]; that is, the model domain is divided into tiles, which are then further divided as necessary to provide higher resolution, preserving the alignment and orientation of grid points. The realization of "adaptive" mesh refinement in MM5 and most other regional weather models is primitive, however. It is adaptive only in the sense that the location of nests may be determined a priori and statically by the person configuring the model run. Nests in NCAR model may also move over the course of a simulation, but only following a scripted set of translations, not in response to any dynamically detected increases in local gradients or error terms as the model runs.

* Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439.

Efficient parallelization involves decomposing the two horizontal dimensions of the model domain over processors, implementing communication between processors, adjusting iteration to compute only over the local subdomain in each processor's memory, and load balancing. In the case of large, preexisting models, parallelization must not hinder understandability, maintainability, and portability of the code. At first, few tools existed beyond low-level message-passing libraries. Over time, however, many groups have developed libraries that roll in other necessary functionality: library-level support for data domain decomposition and computation over distributed domains [1][9][18], mesh refinement, and load balancing [10][17]. The Runtime System Library [12], was developed at Argonne the course of a research effort to address load imbalance and nesting in the development of regional weather models.

In addition to providing higher-level communication constructs, stencil exchanges and broadcast-merges for nesting, RSL offers other advanced features: pointwise decomposed, irregularly shaped processor subdomains, dynamic remapping of work to processors for load balancing, and support for irregularly shaped nests. Earlier concerns that RSL required more dramatic modifications to existing codes for column callability have been addressed in the current version, without sacrificing RSL's unique ability to efficiently support irregularly shaped processor decomposition.

RSL has been used to parallelize MM5, the fifth-generation Pennsylvania State University/National Center for Atmospheric Research Mesoscale Model. RSL is also one component of an effort to enable "same-source" parallelization of large existing atmospheric codes; the other component is application-specific Fortran source translation software [13]. The combination hides parallel infrastructure in these codes and allows a single version of a model source code to run efficiently on diverse computer architectures.

Section 2 describes characteristics of the type of application for which RSL is targeted. Section 3 discusses parallelization issues and how these are addressed in RSL. Section 4 presents the MM5 parallelization as a case study, with particular emphasis on dynamic load balancing and advanced nesting options. Section 5 briefly describes source translation and its relationship to RSL in supporting a same-source approach to parallelization of regional atmospheric models.

2. Model Characteristics. Finite-difference models of dynamical systems are widespread in atmospheric and other sciences. The models typically consist of a two- or three-dimensional gridded domain representing the model state—velocity, temperature, and pressure, for example. Most generally, a domain is initialized and then integrated forward over a series of time steps. Boundary input and model output are performed periodically, as follows:

Domain definition and initialization.
Loop over time.

If it is time, acquire new boundary data.
Advance domain state by one time step.
If it is time, perform model output.
End loop.

At the beginning of the simulation, the model domain is defined in terms of its size, shape, and allocation in memory, and the initial state of the model is input or otherwise obtained. Lateral boundary conditions, also may be input periodically over the course of the simulation. During each time step, the state of the model for the next time step is computed for each grid point by evaluating the state at the point and some stencil of nearest-neighbor grid points.

$$\begin{array}{lllllll} X^{new}_{i,j} & = & & & c_1X_{i+i,j} & + & \\ & & c_2X_{i,j-1} & + & c_3X_{i,j} & + & c_4X_{i,j+1} \\ & & & + & c_5X_{i-i,j} & & \end{array}$$

The exact shape and number of points in a stencil depend on the order of the finite-difference method and on the gridding scheme used. Interpolation will also involve a stencil.

Accurate resolution of weather phenomena improves with scale-appropriate resolution. However, as fineness of resolution increases, so does computational cost because of the added number of grid points and the smaller time step. Nesting is used to increase resolution over portions of a domain. Nesting is accomplished by positioning a higher-resolution domain within a coarser domain and exchanging forcing and feedback data between the two:

Parent domain definition and initialization.
Nested domain definition and initialization.
Loop over time.
Advance parent domain one time step.
Transfer parent domain state data to force the nest.
Loop over nest time steps.
Advance nested domain one time step.
End loop.
Transfer nested domain state data back to parent domain.
If it is time, perform model output for both parent and nest.
End loop.

The parent domain advances one time step; then data in the region of the nest is transferred from the parent to the nest. The model iterates over the smaller nested domain time steps, bringing it forward to the same time level as the parent. Finally, nested domain data is transferred back onto the region of the parent domain, and the next time step commences.

Nested domains may themselves have nests, allowing simulations to reach arbitrarily fine resolutions within the limits of the particular dynamics and physics in the model.

3. Parallelization. Parallelizing a model on a distributed-memory parallel computer involves defining, decomposing, and allocating memory for the model domains; iteration over decomposed dimensions; local-global index translation; interprocessor communication; load balancing, nesting; and I/O. RSL provides support for each of these tasks.

3.1. Domain Definition, Decomposition, and Allocation. Domains are defined by describing their size, shape, and parentage to RSL. For rectangular domains, size and shape are specified by giving the number of rows and columns. For irregularly shaped domains, size and shape are specified by giving the outline of the domain, that is, by listing the coordinates of the vertices of the irregularly shaped domain's enclosing polygon. A domain may be any nonzero size provided it is totally enclosed by its parent domain (in the case of nest), within the limits of physical memory. A nest is always defined as the child of a parent domain, and parentage remains fixed for the duration of the nest. Multiple nested domains may be defined within a parent. There must always be a top-level mother domain that is defined first and only once. The mother domain is always rectangular and has no parent.

Decomposition of a domain maps each grid cell of the domain to a processor. All domains in a model are defined over the same set of processors. Viewed another way, each processor has a piece of every domain in the model. RSL automatically decomposes domains when they are defined or remapped. RSL's default algorithm divides the domains into partitions with the number of points as close to equal as possible. Each point of the domain can be allocated independently, allowing irregularly shaped processor subdomains. Domains may be redecomposed at any point during a run. The user may specify alternative decomposition algorithms.

Allocation pertains not to the domain itself but rather to the two- and three-dimensional arrays that store the state and intermediate variables used in the model. For a given decomposition, the arrays associated with a domain require a certain amount of memory on each processor. RSL does not actually allocate the arrays associated with a domain. Rather, it makes the size information available to the program. This size information may be used to allocate memory dynamically or simply to provide a means for checking that static sizes are large enough for a decomposition.

3.2. Local Iteration and Computation. Since a processor computes only the points that are stored locally, a mechanism is needed for keeping track of a processor's local allocation in the parallel code. RSL assumes the responsibility for keeping track of the points that are local on each processor and for directing iteration over those points. A number of mechanisms are provided. RSL may actually control the iteration by applying model routines that the user provides as functional pointers, or it may simply make the partition information available to control iteration that is specified explicitly in the user program. Macros are provided to facilitate the expression of decomposed loops using RSL. The macros may be programmed manually or generated automatically by using a special purpose preprocessor or precompiler, such as the Fortran Loop and Index Converter (FLIC) [13].

3.3. Local and Logical Index Correspondence. Under the single-address space memory model, the indices of a point in the logical domain are identical to its array indices, so that the indices may be used interchangeably. Decomposition and shrinking of local data structures on processors break this relationship: the index of a point in a local processor's memory is almost never the logical index of the point in the global domain. Therefore, the relationship between the local array indices and logical coordinates must be explicitly established and maintained.

RSL automatically computes and makes available to the program both sets of indices. The indices in local data structures are used whenever a local array is referenced in the code. No assumptions can be made by the program about the actual value of these local indices except that a point i is always adjacent to the points $i-1$ and $i+1$ in a given dimension. A corresponding set of global indices are used for determining the position of a point within the logical domain, for example, when testing for proximity with a boundary.

3.4. Interprocessor Communication. Model computations that involve data from neighboring cells or from cells that exist on another domain will require communication if the cells reside on a different processor. To avoid complicated, error-prone, and potentially less efficient message-passing code in the model, RSL provides high-level communication mechanisms for handling the types of data dependency found in finite-difference models with nests. The *stencil* provides intradomain communication for finite-differencing and interpolation. The *broadcast-merge* provides communication for exchanging data between domains for nesting.

*Intra*domain communication resolves the nearest-neighbor data dependencies associated with finite differencing and horizontal interpolation. The set of neighboring points that have data needed for a computation is called a stencil. Under RSL, stencils are defined by specifying the points of the stencil and the fields (model variables) that should be exchanged on each of the points. Stencils are used in *stencil exchanges*: transfers of data from remotely stored points into extra cells of the local array that have been allocated around the partition. This padding is known as the "halo" or "ghost" region of an array. RSL automatically determines the size and shape of the ghost region for each defined stencil. During a stencil exchange, the needed data is automatically buffered on the sender and unbuffered on the receiver; hence, each stencil exchange involves only one message sent and one message received for each processor pair in the exchange, minimizing the latency cost of the transfer.

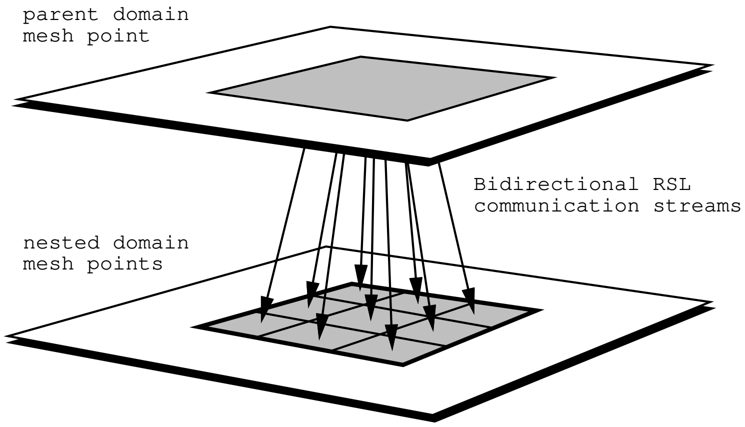

FIG. 3.1. *A parent domain cell and nine nested domain cells covering the same geography at different resolutions. The domains exchange data over communication streams.*

*Inter*domain communication transfers the forcing or feedback data between a parent domain and a nest. At the time a nest is created, RSL establishes a link between each parent domain point and the points in the nest it overlays (Figure 3.1). The links are logical and do not depend upon on what processor a parent or nested domain point resides. Downward forcing, from parent to nest, involves a logical broadcast from a parent domain point to the nest points that are linked to it. Upward forcing involves a merge along the same links but in the opposite direction.

Incidentally, RSL permits the ratio of nested to parent points to vary in each horizontal dimension (but always ≥ 1).

3.5. Load Balancing. Load imbalance occurs when some processors have more work to do than others. Processors that finish first idle, reducing performance relative to the ideal (in which all processors are kept busy). The ratio of actual performance to ideal performance is called the *efficiency*. Inefficiency from load imbalance may result from (1) an uneven initial distribution of domain points to processors — especially if the number of processors does not evenly divide the number of rows or columns; (2) reduced amounts of work in the boundary points of a domain; (3) dynamic conditions in the simulation itself that cause computations to be performed in some sections of the domain but not in others; or (4) different processor speeds or task loads in a heterogeneous or multiuser computing environment. RSL addresses this problem by supporting optimal decompositions of points to processors, whether or not the decomposition results in rectangular processor subdomains, and by providing a mechanism for distributing and redistributing domain cells between processors. Implementing irregularly shaped processor decompositions would be prohibitively complicated in an explicit message-passing code or using High

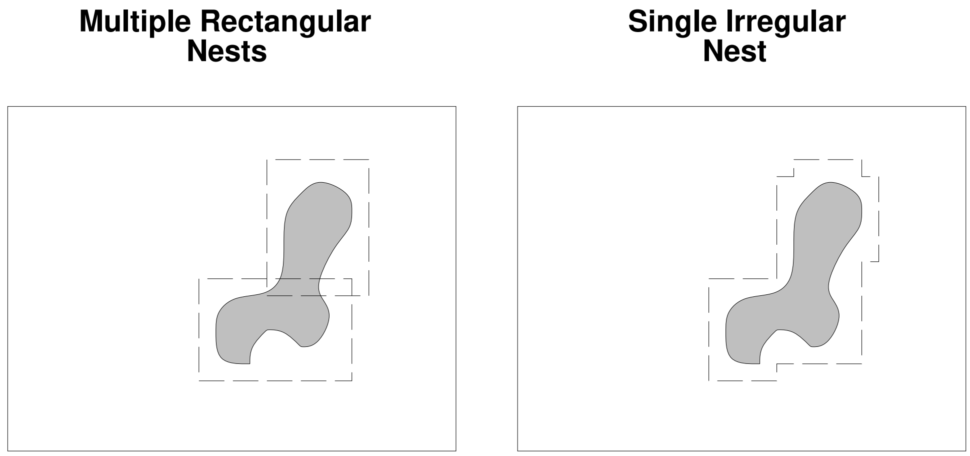

FIG. 3.2. *A single irregularly shaped nest fits a feature of interest in a simulation more closely and without additional code to handle the overlap region that occurs when two rectangular nests are used.*

Performance Fortran, which supports only regular decompositions of work to processors. However, RSL supports this automatically, transparently, and with little additional overhead.

3.6. Irregularly Shaped Nests. Models that support nested domains may also allow multiple overlapping nests so that a user can overlay a number of rectangular nests to closely fit a feature of interest in the simulation, such as a weather front or a region of complicated terrain. RSL supports multiple domains on a nest level, but the user can avoid writing complicated "overlap" code by specifying, instead, an irregularly shaped nested domain whose shape is a union of rectangles to fit the feature of interest (Figure 3.2). Control flow of the model is also simplified by eliminating nest overlapping in favor of irregularly shaped domains. The nesting hierarchy becomes strictly tree-shaped, since only parent-to-nest (not nest-to-nest) data dependency relationships need to be supported.

3.7. Input and Output. Reading data from a serial data set onto distributed domains, and outputting distributed data to serial data sets, requires communication between processors and may also introduce a serial bottleneck in the parallel code. RSL provides routines that read and write sequential Fortran data sets, automating the distribution of array elements to processors on input and the collection of array elements from processors on output. The parallel implementation of MM5, for example, is able to read and write serial MM5 data sets.

Although RSL manages the complicated task of decomposing serial input and recomposing serial output on the fly, the mechanism employed is currently "single reader, single writer"; that is, one processor reads and writes the data to files and sends and receives messages to the other processors. This aspect of the system is currently nonscalable. However, since atmospheric codes such as MM5 generate output at a low frequency relative to the amount of computation that occurs between outputs, the single-reader, single-writer mechanism has not been a serious problem in the work with MM5. Implementing a scalable yet portable solution to parallel I/O is an issue that will be addressed in future implementations of RSL.

4. MM5. The PSU/NCAR MM5 models limited-area atmospheric systems ranging from several thousand kilometers to several hundred. It is a primitive-equations model employing finite differencing for atmospheric dynamics and has a rich complement of physics parameterization packages: solar radiation, cumulus, moisture physics, and boundary layer physics [7]. It allows multiple grids for nesting high-resolution computations over regions of interest in a simulation with two-way interaction between nest levels. Four-dimensional data assimilation, in the form of Newtonian nudging, is provided to allow the incorporation of observational data to refine a forecast at run time. Uses include weather forecasting, regional climate predication, air quality research, and basic atmospheric research. The model, which dates back to the late 1970s, is now in its fifth generation and is maintained in the public domain by NCAR in cooperation with a large, active, and institutionally diverse user community.

RSL was developed in the course of producing the Massively Parallel Mesoscale Model (MPMM), a Fortran77-based implementation of MM5 [5][15]. This version employed static memory allocation and a simplified nesting scheme: there could be only one nest per nest-level. The model configuration needed to be specified at compile time. MPMM also allowed static load balancing in which a pointwise decomposition of the domain could be specified once at the beginning of the run. Subsequent effort to employ advanced features of Fortran90, including modules, derived data types, and dynamic memory allocation, and recursion, led to development of MM90 [14], a more modular, flexible, and run-time configurable code than MPMM. The added flexibility also enabled dynamic load balancing and irregularly shaped nests.

4.1. Dynamic Load Balancing in MM5. Overhead for parallel computation must be kept low relative to the amount of useful computation. Load imbalance is a source of inefficiency that results in some fraction of available processing power being lost as lightly loaded processors wait for more heavily loaded processors to finish. Load imbalance in atmospheric codes comes from a number of sources [11] [16] [4].

- The number of processors may not evenly divide the data domain.
- Domain boundaries entail less work than the interior.
- Model physics (the parameterization of solar radiation, cloud processes, boundary layer physics, etc.) can perform different amounts of computation depending on the local state of the model in an area

of the grid.

- Nesting in multiple-grid models may induce imbalances associated with the forcing and feedback between domains.
- The processors of the parallel computer may not be uniform in their computational power.
- Some processors may be running other users' jobs.

Using mechanisms in the RSL library, MM90 addresses load imbalance using

- instrumentation to monitor the amount of work performed in each grid-column of each domain (grid) in the simulation,
- run-time remapping grid-columns to other processors to adjust for load imbalances that are detected, and
- pointwise irregular decomposition of the two horizontal domain dimensions to allow for greater precision in mapping work to processors than in traditional patchwise decomposition.

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 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 over the course of the model run, the timer arrays are collected into a global array of timers for all columns and all segments, and then this array is redistributed among the processors. A new mapping is computed using the timing information, and the efficiency of the new mapping is compared with the efficiency of the old mapping and adopted if it improves efficiency by more than an epsilon.

Remapping work to processors involves determining which columns stay and which columns are to be moved to a different processor, packing up the state data for the columns to be moved into messages, sending the messages to their destinations, and unpacking them into the data structures on the destination processor. In addition, it may be necessary to resize the local data structures on a processor to accommodate an influx of columns from other processors. Five steps are involved:

- Construct an RSL state vector for the current (old) decomposition.
- Decompose the domain using the newly computed mapping.
- Construct an RSL state vector for the newly installed decomposition.
- Reallocate memory.
- Remap.

A state vector is an RSL message definition that contains a list of all the fields that make up the state for a grid-column. RSL will use the state vector for packing and unpacking messages containing the state data to be moved. State vectors are described by making a series of calls to the RSL library, passing information about the size, shape, and location in memory of the state arrays.

A new decomposition is put into effect by passing a function to compute the new mapping to RSL. The library includes a built-in mapping function, but it considers only the unweighted number of cells per processors — the function MM90 provides weights the cells with the timing information gathered on the most recent series of time steps. The MM90 mapping function computes the new mapping using the following algorithm:

1. Compute T, the sum of the times for all the columns in the domain.
2. Divide the m dimension of the domain into *plat* parts, each containing cells whose individual timings sum as closely as possible to $\frac{T}{plat}$.
3. Divide each of the partitions from Step 2 along the n dimension into *plon* parts, each containing cells whose individual timings sum as closely as possible to $\frac{T}{plat \cdot plon}$.

Once it has generated the new decomposition, the MM90 mapping function compares the new mapping with the current one. If the new decomposition is adopted, MM90 allocates a new domain structure to hold the remapped data. A new RSL state vector is described, identical to the previous one except that it is associated with the fields in the newly allocated domain data structure. It remains only to effect the remapping:

```
CALL RSL_REMAP_STATE(D)
```

RSL compares the old and new mappings and generates lists of moves of grid points between processors. From these lists, information in the first state vector is used to pack columns into messages; the second state vector is used to unpack the messages at their destinations. Finally, the old structures are deallocated, and the model resumes time stepping under the new mapping.

Figure 4.2 shows the cost for the series of time steps in a single domain (no nest) 32-processor run into which an artificial load was induced at hour 2.5. Prior to this, efficiency as measured by the sum of the times for all points divided by the maximum processor time is 94 percent (Period A). As a result of the imbalance, efficiency falls to 50 percent (Period B). Model performance drops sharply until the next load-balancing step at hour 3.5. The resulting remapping restores efficiency to 96 percent (Period C). The cost for each remapping is approximately 18 seconds.

The usefulness of MM90 load-balancing is demonstrated in a production setting. As part of the U.S. Air Force Global Theater Weather Analysis and Prediction System (GTWAPS) [19], MM90 produces twice-daily real-time forecasts at 10-km resolution covering a 1000-km by 1000-km domain centered over Bosnia. The hardware platform, IBM SP2 with fourteen 66 Mhz Power2 processors, was recently upgraded with the addition of six 120 Mhz Power2-SuperScalar (P2SC) processors, yielding higher aggregate computing power, but in an unbalanced configuration. The MM90 code automatically adapted by loading more points onto the faster P2SC nodes (Figure 4.1), yielding a decrease in execution time from 2.3 seconds per time step to 2.0 seconds per time step (these are averages over a 36-hour

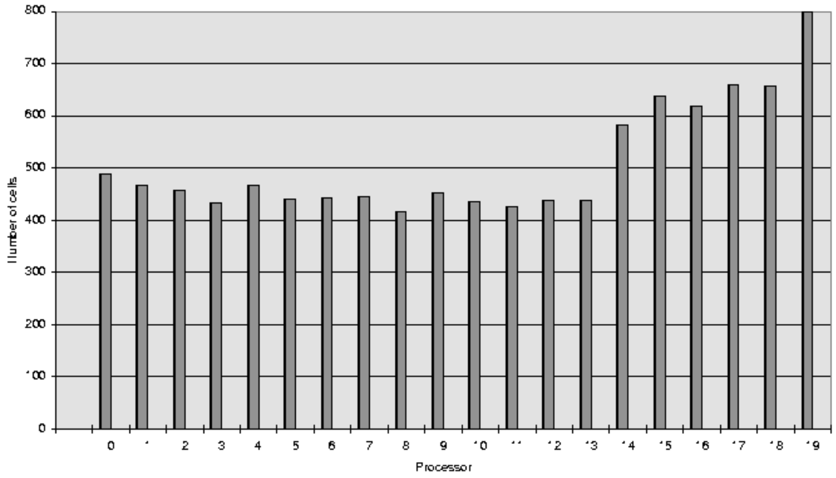

FIG. 4.1. *Distribution of 10201 grid-cells over twenty processors of the IBM SP2 at Air Force Global Weather Central, Offut AFB, Nebraska. The first fourteen are Power2 nodes and the remaining six nodes have faster Power2 Super-Scalar processors. MM90 automatically adjust the distribution of points to processors to give more work to the faster nodes.*

simulation and include the cost of model initialization, I/O, and the cost of load-balancing itself). This amounts to a 20-minute savings in the time to run the forecast, from 2.76 hours to 2.4 hours.

4.2. Irregularly shaped nests. MM5 allows overlapping of rectangular nests to cover irregularly shaped features such as mountains or evolving fronts. However, this entails redundant computation in the overlap areas, it requires additional code to maintain a consistent solution between the overlapping nests, and it adds to the complexity of model control flow, since there are child-child data relationships at the same level in the nesting hierarchy (compared with the simpler situation where there exist only parent-child data relationships). Using the RSL library, MM90*i*, an experimental version of the MM90 code, permits a single irregularly shaped nest to take the place of a number of overlapping domains. Adapting the model for irregularly shaped nests entails modifying boundary tests to allow for irregularly shaped boundaries, and structuring loops to iterate over irregularly shaped domains. Since MM90 is already computes over irregularly shaped processor subdomains for load balancing, only irregular boundary treatment needed to be added. For a given grid point, RSL provides the application with information on distance to the nearest boundary. Code macros such as:

```
IF (DOT_GRID_INTERIOR(2)) THEN . . .
```

take the place of boundary tests in the model and expand to use the RSL-

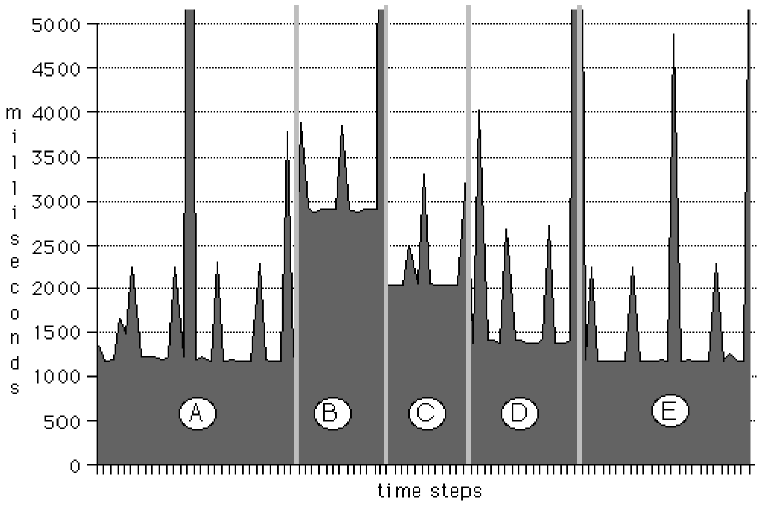

FIG. 4.2. *Series of time steps from an 8-hour run with an artificially induced load between 2.5 and 4.5 hours. Load balancing takes occurs at 3.5 hours, reducing the increase in half. The load balancing itself required about 18 seconds each time it was invoked.*

supplied irregular boundary information when an irregularly shaped domain is specified.

Figure 4.3 shows the decomposition of an irregularly shaped domain used for a climate simulation [6]. The nest, at 5-km resolution, is irregularly shaped and fitted over a region of the Alps (map not shown). The irregular domain requires only 6972 horizontal grid cells, compared with 9592 cells for the enclosing rectangle, for a savings of roughly 27 percent. Here, RSL's ability to specify irregular processor subdomains is especially advantageous (Figure 4.4).

5. Source Translation. Parallelizing an existing atmospheric model involves two main tasks: implementing coherency and modifying the code to operate over separate distributed memories. On distributed memory message-passing architectures, the first task requires detailed knowledge of the code and involves a high level of conceptual effort, but the resulting impact on the source code is small, especially if a library such as RSL is used. The second task, adapting the numerous loops throughout the code and correctly handling global/local index translations entails much more mechanical work involves considerably more mechanical effort, but is conceptually straightforward. The second task is thus ideally suited to automation, through the use of source translation tools. To be effective a source translator must be able to infer the greatest amount of information

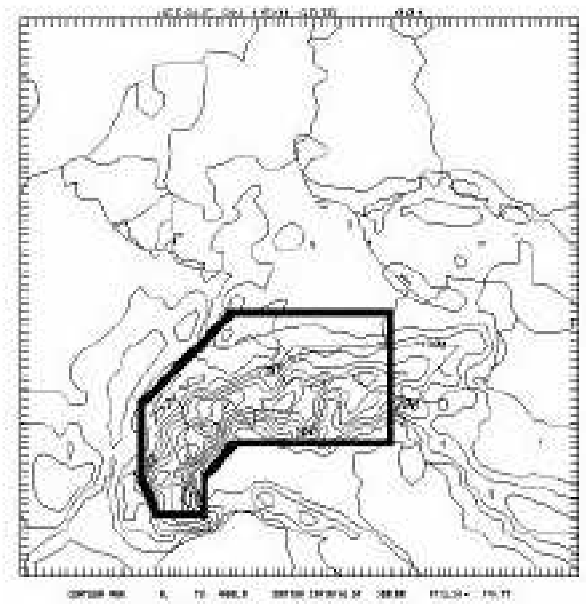

FIG. 4.3. *A domain for simulating regional climate. A 5-km resolution irregularly shaped domain specified over a section of the Alps, within a larger 15-km resolution rectangular domain covering central Europe.*

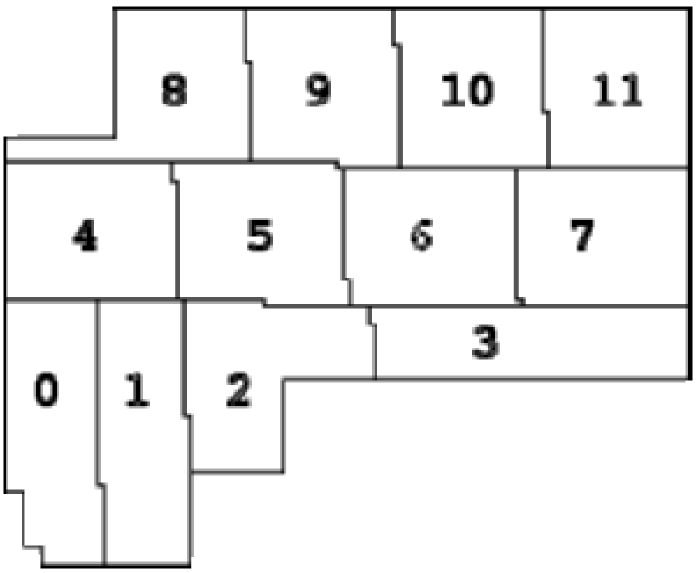

FIG. 4.4. *Pointwise decomposition over 12 processors of the irregularly shaped Alpine domain. With RSL's built-in partitioner, the number of grid-points per processor differs by no more than 2.*

about the code with the least amount of input. Tools that perform simple lexical translation — macro preprocessors, for example — are insufficient, since they require macros or directives to be inserted throughout the code. Source translation based on lexical, syntactic, and semantic analysis of a code, on the other hand, can be virtually directiveless because the translator — effectively a Fortran compiler front-end — has complete information on the loop and data structures of the code already, without the need for directives.

FLIC [13] is a prototype source translator tailored to regular grid atmospheric models and is being used to develop a same-source parallel option in MM5 for integration into the official NCAR version of the code later this year. FLIC takes as input the list of defined constants that are used to declare the decomposed dimensions of model arrays (e.g. "MIX" for the north/south dimension and "MJX" for the east/west dimension) and from this correctly infers

- all loops over decomposed dimensions,
- all indices that must be treated as global, and
- all indices that must be treated as local.

Further, the amount of information to direct the translation is small enough to include on the command line for the tool.

6. Conclusion. The runtime system library RSL has been developed for implementing existing or new weather models on distributed memory parallel machines. It supports dynamic, fine-grained parallel decomposition of multiple nested domains and provides high-level communication routines for intra- and interdomain communication. Mechanisms provided support dynamic load balancing and irregularly shaped nests, which can take the place of multiple overlapping nests. Used in conjunction with source translation tools, RSL enables implementation of atmospheric models so that virtually all architecture-specific aspects of the parallelism may be hidden, permitting one source code to be used on diverse computing architectures.

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.

REFERENCES

[1] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997. To appear (also Argonne National Laboratory Mathematics and Computer Science Division preprint P634-0197).

[2] N. Chrisochoides, K. Droegemeier, G. Fox, K. Mills, and M. Xue, *A Methodology for Developing High Performance Computing Models: Storm-Scale Weather Prediction*, in High Performance Computing, 1993: Grand Challenges in Computer Simulation, The Society for Computer Simulation, 1993.

[3] J. Drake and I. Foster, *Introduction to the Special Issue on Parallel Computing in Climate and Weather Modeling*, Parallel Computing, 21 (1995), pp. 1539–1544.

[4] R. Ford, D. Snelling, and A. Dickinson, *Controlling Load Balance, Cache Use and Vector Length in the Unified Model*, 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. 195-205.

[5] 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.

[6] G. Grell, A. Pfeiffer, L. Schade, and J. Michalakes, *Regional and Local Climate Modeling in Bavaria: Verification with Lightning Statistics*, presented to Sixth Annual MM5 Users Group Workshop, NCAR, Boulder, Colorado, July 22-24, 1996.

[7] 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.

[8] W. Gropp and D. Keyes, *Semi-structured refinement and parallel domain decomposition methods*, in Unstructured Scientific Computation on Scale Multiprocessors, P. Mehrotra, J. Saltz, and R. Voigt, eds., 1990, pp. 187-203.

[9] 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.

[10] 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.

[11] J. Michalakes, *Analysis of Workload and Load Balancing Issues in the NCAR Community Climate Model*, Tech. Rep. ANL/MCS-TM-144, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, January 1991.

[12] ———, *Runtime System Library for Parallel Finite Difference Models with Nesting*, Tech. Rep. ANL/MCS-TM-197, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, March 1997.

[13] ———, *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.

[14] ———, *MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5)*, to appear in Parallel Computing (also Argonne National Laboratory preprint ANL/MCS-P659-0597, 1997).

[15] 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 Processors in Meteorology, World Scientific, River Edge, New Jersey, 1995, pp. 266-276.

[16] J. Michalakes and R. Nanjundiah, *Computational Load in Model Physics of the*

Parallel NCAR Community Climate Model, Tech. Rep. ANL/MCS-TM-186, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, January 1994.

[17] M. Parashar and J. C. Browne, *Distributed dynamic data-structures for parallel adaptive mesh-refinement*, Proceedings of the International Conference for High Performance Computing, 1995, pp. 22-27.

[18] 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.

[19] K.L. Simunich and S. Pinkerton and J. Michalakes *System Implementation for Global Theater Weather Analysis and Prediction System (GTWAPS)*, in proceedings of the 13th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology. Long Beach, CA, February 1997, pp. 217-220.