Distributed Processing of a Regional Prediction Model

KENNETH W. JOHNSON

Supercomputer Computations Research Institute, The Florida State University, Tallahassee, Florida

JEFF BAUER

Academic Computing and Network Services, The Florida State University, Tallahassee, Florida

GREGORY A. RICCARDI

Department of Computer Science and Supercomputer Computations Research Institute, The Florida State University, Tallahassee, Florida

KELVIN K. DROEGEMEIER AND MING XUE

Center for Analysis and Prediction of Storms and School of Meteorology, University of Oklahoma, Norman, Oklahoma

(Manuscript received 7 July 1993, in final form 14 March 1994)

ABSTRACT

This paper describes the parallelization of a mesoscale–cloud-scale numerical weather prediction model and experiments conducted to assess its performance. The model used is the Advanced Regional Prediction System (ARPS), a limited-area, nonhydrostatic model suitable for cloud-scale and mesoscale studies. Because models such as ARPS are usually memory and CPU bound, the motivation here is to decrease the computer time required for running the model and/or increase the size of the problem that can be run. A domain decomposition strategy using a network of workstations produced a significant decrease in elapsed time and increase in problem size relative to a single-workstation run. The performance of the resulting program is described by derived formulas (collectively known as a performance model), which predict the execution time and speedup for different numbers of processors and problem sizes. The interprocessor communication speeds are shown to be the major obstacle to achieving full processor use. The effect of faster communication networks on parallel performance is predicted based on this performance model. Parallelization experiments using the ARPS code were run on a cluster of IBM RS6000 workstations connected via Ethernet. The message-passing paradigm implemented here made use of the library of routines from the Parallel Virtual Machine software package.

1. Introduction

This paper describes the parallelization of the Advanced Regional Prediction System (ARPS) model of the Center for Analysis and Prediction of Storms (CAPS) (CAPS 1992; Droegemeier et al. 1992). The ARPS is a limited-area, nonhydrostatic model suitable for cloud-scale and mesoscale studies. Models such as ARPS are usually CPU and memory bound; they require fast CPUs to complete a run in a reasonable time and require large amounts of memory (real or virtual) to accommodate the problem size. The motivation here is to use parallel systems with these models to decrease the computer time required for running the model and/or increase the size of the problem that can be run. The specific goals are to 1) determine how adaptable the ARPS code is to run on a distributed system, 2) quantify the parallel performance, and 3) detect bottlenecks in the ARPS code and the hardware that prevent improved parallel performance.

Parallelization experiments using the ARPS model were run on a compute cluster. A compute cluster is a system of networked computers, in this case workstations, having distributed memory. This computer system was chosen since it is an emerging technology that is becoming increasingly available to researchers because of its price/performance ratio (Buzbee 1993). Parallel processing architectures are typically classified as either single instruction multiple data (SIMD) or multiple instruction multiple data (MIMD). Architecturally, compute clusters may be classified as distributed memory MIMD (dMIMD) systems. It is believed that much of what is learned with this type of system can be carried over to massively parallel processors (MPP) having distributed memory architectures.

Parallel performance of the ARPS code is assessed through numerical experiments and the use of an ana-

Corresponding author address: Dr. Kenneth W. Johnson, Supercomputer Computations Research Institute, 400 Dirac Science Library, The Florida State University, Tallahassee, FL 32306-4052.

lytic performance model. The analytic performance model is developed to analyze and predict the performance of the parallel code. The analytic performance model is developed to analyze and predict the performance of the parallel code. The performance model includes the effects of extra calculations and message passing required by the parallel algorithm. The performance model and the experimental results suggest that significant speedups, relative to a single-CPU system, can be achieved when the ARPS is run on a compute cluster. The greatest obstacle to improved performance is interprocessor communications.

In this paper details of the ARPS model and the parallel algorithm are described in section 2. Hardware and software applied to the parallelization experiments are described in section 3. Section 4 contains descriptions of the analytic performance model. Results of the parallelization experiments are discussed in section 5. Future work is described in section 6.

2. The numerical prediction model and its parallelization

The ARPS model is a fully three-dimensional, nonhydrostatic code designed for the prediction of mesoscale to convective-scale weather (Droegemeier et al. 1991; CAPS 1992; Droegemeier et al. 1992). It was developed at CAPS at the University of Oklahoma. ARPS was designed as a scalable code that takes advantage of the parallelism inherent in the equations of fluid flow. This parallelism can be distributed by the user in many ways given knowledge of the target architecture. The code is highly modular and makes use of discrete operators to perform Eulerian derivatives and averages. In this manner, the mathematical structure of the continuous equations is at least partly preserved, providing for ease of learning, modification, and debugging.

The governing equations are the three-dimensional, nonhydrostatic Navier–Stokes equations augmented by equations for pressure, potential temperature, water vapor, cloud water, rainwater, cloud ice, hail, and snow. These equations in continuous space are shown in appendix A. To allow for stretched grids and terrain, the governing equations are transformed from Cartesian space to a curvilinear space. All computations are done in the curvilinear space where the grid mesh is uniform and orthogonal. Space derivative terms are approximated with second-order quadratically conservative finite differences on the staggered Arakawa C grid. Time marching is done using the leapfrog scheme. For additional details on ARPS version 2.0, see Droegemeier et al. (1991).

The ARPS model can be decomposed to run across multiple processors using either the functional or domain-decomposition approaches. Functional decomposition involves solving each prognostic and/or diagnostic equation on a separate processor. This decomposition potentially requires a large amount of communication between processors. This is because the updated prognostic fields must be transferred from the processors they are being calculated on to all other processors needing them after each time step. Domain decomposition involves assigning subdomains of the full computational grid to separate processors and solving all prognostic and/or diagnostic equations for that subdomain on that processor. Using this type of decomposition, interprocessor communications are required only at the boundaries of the subdomain. The choice of which decomposition to use must be made based on the target multiprocessor architecture. Functional decomposition seems least desirable on most shared memory MIMD and dMIMD architectures. Memory contention may be an issue for the former, whereas communications in the form of message passing may be a bottleneck for the latter. However, a functional decomposition of the ARPS code is the easiest to implement on dMIMD systems. This simply requires that, during a given time step, each governing equation be evaluated over the entire domain on its own dedicated processor. At the conclusion of a time step, each processor sends its results to and receives results from all the other processors. Since all prognostic variables for the entire computational domain must be sent to each processor after each time step, the ratio of computation to communication time is low for this decomposition. This leads to small speedups at best when implemented on a dMIMD system as used here.

Domain decomposition of the ARPS code is a more effective parallel processing approach for current distributed memory architectures. It requires breaking the computational domain into subdomains in one or more of the coordinate directions. Each subdomain is assigned to a processor that does the time-stepping calculations in the subdomain. The ARPS model employs an explicit solution technique. Therefore, as pointed out by Kauranne (1990), no global information is required at any particular grid point. The only additional information needed by a subdomain during a time step is the inner border of adjacent subdomains. These grid-point values are needed to evaluate derivative terms in the prognostic equations at points on the borders of a subdomain using spatial finite differences (Fig. 1). These data are resident and updated in the local memory of adjacent processors. They can be copied to the memory of the current processor using message passing techniques to be discussed shortly.

Several strategies exist for dividing the model grid into subdomains. Figure 2 shows possible decompositions for a three-dimensional grid. Assuming no other complicating factors, a logical strategy is to partition in such a way as to minimize the surface area of each subdomain relative to its volume (Fox et al. 1988). This keeps the computation-to-communication ratio high. The most natural decomposition from a coding standpoint is a one-dimensional decomposition in the z

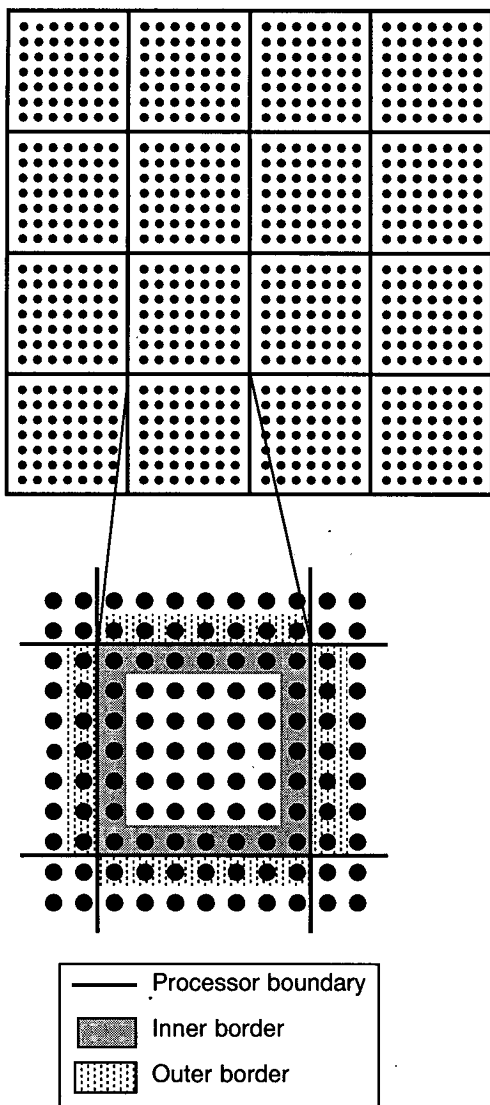

FIG. 1. Domain decomposition of grid showing inner and outer borders of processors. Finite differencing on inner borders requires information from adjacent processors (adapted from Fox et al. 1988).

direction, that is, assigning a fixed number of consecutive $x-y$ planes to each processor (Fig. 2a). However, for most NWP grids, the computational domain is usually a parallelepiped with the least number of points in the z direction. Further, one-dimensional decomposition in the z direction would be even less efficient if vertical integrations or sums, such as those involving columnwise physics, are a part of the model. This suggests that a more efficient decomposition is a one-dimensional decomposition with cuts made in the x or y directions (Figs. 2b,c) to minimize communications.

Two-dimensional decomposition involves decomposing in two coordinate directions simultaneously (Fig. 2d). The decision to use a one-dimensional or two-dimensional decomposition depends, in part, upon the characteristics of the communication network linking the processors. A one-dimensional decomposition will have longer but fewer messages than a two-dimensional decomposition. Thus, it may be more appropriate for a communication network with high message start-up costs relative to transmission costs. A two-dimensional decomposition will transfer fewer total data, at least for nearly square grids, despite a greater number of messages. Thus, for a square grid, which is the most likely case for NWP problems, a two-dimensional decomposition may be the best choice when network transmission rates are relatively slow. The work presented in this paper examines both types of decompositions.

The same code that runs on a uniprocessor was used here on each processor with some modifications. These modifications included changes to loop indices and array sizes, and the addition of message-passing calls. The present code permits one- or two-dimensional decomposition in any of the coordinate directions.

The loop index and array size changes were needed to account for the storage of the outer borders required by each subdomain. Consider a one-dimensional decomposition in the x direction (Fig. 2b). For a computational domain of $N_x \times N_y \times N_z$ points, the N_x vertical planes are partitioned into P subdomains, each containing $(N_x/P) + 2$ contiguous $y-z$ planes. The two extra $y-z$ planes in each subdomain contain inner border data from adjacent subdomains. To accommodate these data, the dimensions of the arrays in local memory of each processor are extended by 2 in the direction of the decomposition.

A scatter–gather routine was added to the model code to accommodate the transfer of a processor's bor-

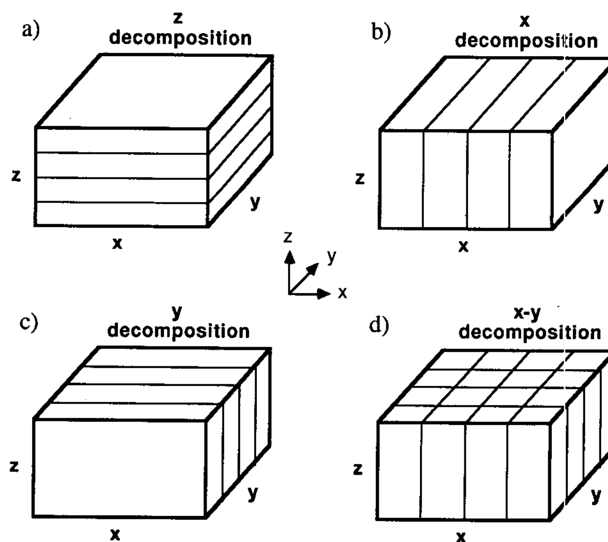

FIG. 2. One- and two-dimensional domain decomposition strategies for a three-dimensional grid.

der data to its neighbors. Regardless of the type of decomposition, each processor has code that enables it to determine the identity of its adjacent processors. The gather routine collects a given processor's inner border data and sends it to the adjacent processors. This is done by filling a buffer array with an appropriate plane of data and sending the array as a single message. The scatter routine receives the messages from the adjacent processors and scatters that data to the current processor's outer border memory locations.

The ARPS was coded to allow for the grid to change shape with time. This gives it the potential to have higher-resolution grids in regions where the fields have large gradients. But, time-dependent grid changes require the calculation of Jacobians at each grid point. This is because the model equations use the contravariant form of the velocity components (see appendix A), which are related to the physical velocity components by means of the Jacobian of transformation. Thus, the Jacobian calculations are duplicated on the outer borders of each domain by the two processors "sharing" these border data. The Jacobians could be precalculated at the beginning of each time step and sent to each processor. However, to save communication costs, they are not sent from adjacent processors but are recalculated when needed.

No explicit synchronization calls were added to the parallelized code. Synchronization occurs indirectly because of the message passing. Whenever a processor gets ahead of the calculations being done on adjacent processors, it has to wait until those processors can complete their calculations and send their border values to their neighbors.

As mentioned above, columnwise physics, and physics in general, could be a factor in the selection of the type of decomposition. A potential disadvantage of domain decomposition is the possibility that one or more processors will finish their work ahead of the rest (e.g., if only part of the domain contains clouds). This is known as the load-balancing problem. Thus, although each processor runs identical code, each may execute a different number of instructions due to data-dependent branching (e.g., branching associated with condensation, radiation, and turbulence). Processors having to perform these additional calculations would lag behind those that do not need to perform them. Thus, after a time step, processors not doing microphysical calculations, for example, would have to wait until the other processors finished to exchange inner border data with neighboring processors.

The issue of load balancing across processors did not arise in the parallel experiments conducted here. This is because all the subdomains were of equal size and, thus, had the same amount of computation since all microphysical processes were turned off for these experiments. Also, all the node machines were dedicated and of equal speed.

3. Hardware and software

The ARPS parallelization experiments were run on a cluster of IBM RS6000 workstations connected via Ethernet. The workstation cluster is described more fully in appendix B. All machines were run in a dedicated mode to eliminate competition for cycles by other users. The host machine ran the initialization portion of the code and sent appropriate subdomains of the base state and prognostic fields to the nodes. The node machines carried out the time stepping for their associated subdomains.

Parallel processing generally requires special software. A code written in Fortran for a sequential processor usually will not run on a multiprocessor system without modification either by the user or the computer. Generally, it is easier to develop parallel software for shared rather than for distributed memory machines. A principal reason is that software technology for shared memory systems is fairly well advanced, whereas compiler technology for dMIMD systems is still in its infancy. For example, existing compilers generally do not automatically partition data among processors for distributed memory MIMD systems. For systems such as compute clusters, problem decomposition requires communication between processors via message passing, which entails the insertion of subroutine calls in the source code. For the dMIMD work reported here, several message-passing libraries were available. The Parallel Virtual Machine (PVM) package (Sunderam 1990) was selected. It supports process creation, message passing, and synchronization between processors through use of user interface primitives. Libraries that interface with Fortran are available and were used.

4. Predicting parallel performance

Many factors influence the performance of an NWP model in a multiprocessor environment. The task of analyzing and predicting NWP model behavior can be aided by the use of analytic performance models. Performance models are mathematical expressions that predict the execution time of a computer program. They are generally functions of the operation count of the algorithms used in the code, the communication time, and constants related to machine and network characteristics. The advantage of a performance model is that predictions of program performance due to system changes can be made, allowing one to forecast how hardware changes could improve code execution. The disadvantage is that the performance models are not as accurate as actual measurements since estimates of actual system values are required to develop and validate the performance model.

Two analytic performance models for the ARPS code were derived to predict elapsed wall-clock time. One performance model describes the behavior of the sequential model, while the other describes the behavior of the parallel code.

a. Sequential performance model

First, consider the performance model for the sequential code. The ARPS can be viewed as having two segments: an initialization portion, and an iteration portion. The initialization portion reads in data and initializes arrays, while the iteration portion carries out the time stepping. The total sequential execution time T_{seq} can be expressed as

$$T_{seq} = T_{init} + T_{iter}, \tag{1}$$

where T_{init} is the execution time for the initialization portion and T_{iter} for the iteration portion of the code. As shown in appendix C, this can be written for a run of N_t time steps on an $N_x \times N_y \times N_z$ grid as

$$T_{seq} = (174N_xN_yN_z + 1574N_xN_yN_zN_t)t_{calc}, \tag{2}$$

where t_{calc} is the typical time to do a generic calculation such as an add or multiply. It is obvious from (2) that the iterative part of the code, represented by the second term of (2), dominates the total execution time and increases linearly with the number of iterations.

b. Multiprocessor performance model

An analytic performance model for the dMIMD processing paradigm using P processors also can be derived. The following discussion is for a grid decomposed in the x direction as described previously (Fig. 2b). Thus, there are P subdomains of size $N_xN_yN_z/P$. Similar results can be obtained when decomposing into slabs in the other directions, or even into cubelike subdomains resulting from two-dimensional decomposition. The model also can be applied to subdomains of unequal size, although for simplicity, that is not presented here.

A performance model for the parallel execution of ARPS must consider three issues not germane for the sequential runs: nonparallelizable code in the model, communications overhead, and software overhead due to redundant decomposition calculations. The initialization routines constitute the nonparallelizable portion of the code. Although in principle this part of the code can be parallelized, it was not done here since this part of the model often changes from case to case. Only the iterative part of the model is considered for parallelization here.

Communications overhead result from the transfer of data to the outer borders of the subdomains from adjacent processors. Software overhead results from duplicated calculations of Jacobians on the outer borders of each subdomain (Fig. 1). The Jacobian terms are needed for differencing and averaging in the direction of the decomposition (for the examples to be shown here, both horizontal differencing and averaging in the x direction).

The parallelized code's total execution time when run on P processors T_P consists of the computational time T_{calc} and the communication time T_{comm}. When the computations and communications cannot be overlapped (as is the case for these experiments), the expression for total execution time is

$$T_P = T_{calc} + T_{comm}. \tag{3}$$

Each of these terms has several components.

The code's computational time T_{calc} has an initialization component T_{init} and an iteration component T_{iter} like the sequential code. Also included is a redundant calculation time T_{dup} resulting from the parallelization of ARPS. Thus, the computational time T_{calc} can be expressed as

$$T_{calc} = T_{init} + \frac{T_{iter}}{P} + T_{dup}. \tag{4}$$

As shown in appendix C, this can be expressed, for a run of N_t time steps on an $N_x \times N_y \times N_z$ grid decomposed in the x direction, as

$$T_{calc} = \left(174N_xN_yN_z + 1574\frac{N_xN_yN_z}{P}N_t + 1338N_yN_zN_t\right)t_{calc}. \tag{5}$$

The code's total communication time T_{comm} when run on P processors consists of an initialization communication time T_{base} and an iterative communication time T_{itcom}. Time T_{base} is required to send base-state fields and initial values to each processor. Time T_{itcom} is required to send data to the outer borders of adjacent processors in the iterative part of the code. The expression for total communication time is thus

$$T_{comm} = T_{base} + T_{itcom}. \tag{6}$$

The communication time required to transfer a packet of information from one processor to another depends on the start-up time, the size of the message, and the transmission rate. The start-up time is assumed to be independent of message size. With values for these parameters, communication times may be calculated in the performance model.

Initial communication in the ARPS model involves sending appropriate subdomains of the base-state fields, coordinate arrays, and prognostic fields to the local memory of each processor. Node-to-node communication during the iterative part of model execution requires only that data on the inner borders of a subdomain of a given processor be transferred to adjacent processors after each time step. When these factors are taken into account, the total communication time T_{comm} can be expressed as (see appendix C)

$$T_{comm} = (P-2)\left[35t_{start} + 41N_yN_z\left(\frac{N_x}{P}+2\right)t_{trans}\right] + 2\left[35t_{start} + 41N_yN_z\left(\frac{N_x}{P}+1\right)t_{trans}\right] + 2N_tt_{start} + 22N_yN_zN_tt_{trans}, \tag{7}$$

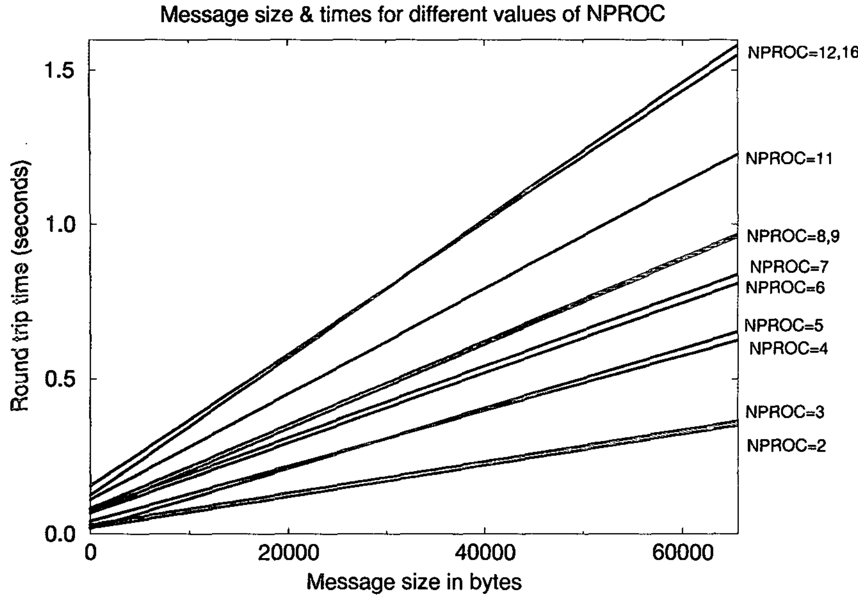

FIG. 3. The effect of message size and number processors on communication time when using Ethernet.

where t_{start} is a start-up time, and t_{trans} a transmission rate. The derivation of (7) assumes no bottlenecks or slowdowns due to all the processors sending their messages simultaneously. With this assumption, P does not appear in the last two terms of (7), which represent the iterative communication time T_{itcom}. Thus, communication time during the iterative part of a model run is independent of the number of processors used. If this assumption is not valid, T_{itcom} would be an explicit function of P. Consequences of this are addressed later.

The time needed to initialize the ARPS model and send base-state and initial fields to the nodes generally will be small compared to that of the iterative portion of the code. Under this assumption the first term in (5) and the first two terms in (7) are small and can be dropped. This allows the analytic performance model to be written as

$$T_P = T_{\text{calc}} + T_{\text{comm}}$$
$$= \left(1574 \frac{N_x N_y N_z}{P} N_t + 1338 N_y N_z N_t\right) t_{\text{calc}} + 2N_t t_{\text{start}} + 22 N_y N_z N_t t_{\text{trans}}. \quad (8)$$

Useful measures of performance in addition to execution time include speedup and efficiency. It is generally thought that elapsed wall-clock time is the most important metric for NWP and forecasting (Droegemeier et al. 1992). However, speedup and efficiency can be used to determine how effectively the computers are being used. Speedup S is the ratio of the execution time of a sequential run to the execution time of a P processor run. Efficiency is the ratio of the speedup to the number of processors. The maximum speedup possible on P processors is P. The analytic model of speedup for this problem is

$$S = \frac{T_{\text{seq}}}{T_P}$$
$$= \left[\frac{1}{P} + \frac{1338}{1574}\frac{1}{N_x} + \frac{1}{1574}\frac{1}{N_x N_y N_z} \times \left(\frac{t_{\text{start}}}{t_{\text{calc}}} + 11 N_y N_z \frac{t_{\text{trans}}}{t_{\text{calc}}}\right)\right]^{-1}. \quad (9)$$

The first term in (9) gives the speedup due to grid decomposition, while the second term arises due to the duplicate computations. The remaining terms are related to communication between processors. The first two terms in parentheses give the communication time for sending data to adjacent processors. Note that (9) is similar to Amdahl's law, where the first term is analogous to the time for parallel processing and the other terms are analogous to the time for sequential (scalar) computation. From (9), it is seen that the effects of duplicate calculations on speedup decrease as the number of points in the direction of decomposition (here, the x direction) increase for a fixed number of processors. This is because the ratio of computation to duplicate computation increases. The effect of communications on speedup varies according to grid size and the ratio of computational speed to communications speed. As the grid size increases and/or the computation-to-communication ratio increases, so does the speedup for a fixed number of processors. Thus, speedup is limited primarily by the computation-to-communication ratio.

The above discussion of speedup focused on problems of fixed size and, thus, fixed-size speedup. Fixed-size speedup is determined by holding fixed the problem size (i.e., the number of grid points) and varying the number of processors, the goal being to solve the fixed-size problem faster. As the number of processors increases, the amount of work done by each processor decreases while the communication time remains constant, thus decreasing the computation-to-communication ratio. In contrast, a problem can be scaled up by increasing the problem size as processors are added. This leads to a scaled speedup where the goal is to solve the largest problem within a given time. In this case, the amount of work done by each processor remains constant, independent of the number of processors added. In this way the computation-to-communication ratio remains constant.

Use of the performance models requires that the variables t_{calc}, t_{start}, and t_{trans} be determined either from system hardware characteristics or measurements. Here they were determined empirically. The value for t_{calc} was determined from single processor runs of the code. The values of t_{start} and t_{trans} were determined by making multiprocessor runs mimicking the communications pattern of the decomposition. Messages of various sizes were sent back and forth between processors, and round-trip times for the message passing were measured (Fig. 3). Predicted timings using these values are given in the next section.

When using Ethernet for communications, as was done here, the communication time is not independent of the number of processors. As the number of processors trying to send messages to adjacent processors increases, the time for a message to get from the sending processor to the receiving processor also increases. Experimentation has verified that t_{start} and t_{trans} are functions of P when using Ethernet. The magnitude of the communication-time increase, as a function of P, was determined by sending messages between increasing numbers of processors and measuring the time the messages took to make a round-trip. As seen in Fig. 3, the greater the number of processors trying to communicate simultaneously, the greater the time for a message to travel between processors. The slopes and intercepts of the curves in Fig. 3 were determined and used as part of the communications model within the analytic performance model.

5. Parallel performance of the ARPS model, version 2.0

a. Parallelization experiments

Parallelization experiments were run for functional and domain decomposition strategies using the ARPS model code. The functional decomposition experiments showed little speedup and, in some cases, a slowdown. As discussed in section 2, this poor performance is due to the large amount of data that must be transferred to each processor during each time step. Thus, this decomposition was not considered further.

Parallelization experiments using one- and two-dimensional domain decomposition were run to examine fixed-size speedup and scaled speedup. Single-dimension decomposition experiments were done with decomposition in the x direction and in the z direction. For the x-decomposition experiments, two different values of N_y were used. Subdomain sizes with $N_y = 31$ ranged from $2 \times 31 \times 32$ to $16 \times 31 \times 32$ points per processor, resulting in grid sizes up to $256 \times 31 \times 32$ points for 16 processors. With $N_y = 62$, subdomain size ranged from $4 \times 62 \times 32$ to $8 \times 62 \times 32$ points, producing grid sizes up to $128 \times 62 \times 32$ points for 16 processors. For the z decompositions, only fixed-size grids of $32 \times 31 \times 32$ and $64 \times 62 \times 32$ total points are considered. The $32 \times 31 \times 32$ grid was decomposed with subdomain sizes ranging from $32 \times 31 \times 2$ to $32 \times 31 \times 16$ points. The $64 \times 62 \times 32$ grid was decomposed with subdomains of $64 \times 62 \times 4$ and $64 \times 62 \times 2$ points. The x-decomposition cases with $N_y = 62$ and the z-decomposition cases were done principally to explore communication issues.

The subdomain sizes of $16 \times 31 \times 32$ and $8 \times 62 \times 32$ points described above represent the largest subdomain that will fit in the memory of each processor, 15 872 grid points. Subdomains larger than this will cause time consuming paging of data in and out of memory. However, the larger memory of the host machine allowed sequential jobs to run with grids as large as $32 \times 62 \times 32$ points.

Two-dimensional decomposition experiments were done with decomposition in the x and v directions. Subdomain sizes of $16 \times 16 \times 31$ and $8 \times 8 \times 31$ points per processor were used. Subdomain arrangements were restricted to those producing whole grids with square dimensions. This restriction, although arbitrary, is reasonable since whole grids with rectangular dimensions are more efficiently decomposed with one-dimensional decompositions in the direction of the long axis. The two-dimensional experiments used either 4 or 16 processors, resulting in total grid sizes of up to $64 \times 64 \times 31$ points.

The time step, grid increment, and number of time steps were held constant for all decomposition experiments, regardless of the number of grid points used. The time step was 0.15 s (note that ARPS 2.0 does not employ a mode-splitting time integration scheme as will later versions of ARPS—hence, the small time step), the grid increment was 500 m, and the number of time steps was 100. Thus, as the number of points increased in a given direction, so did the physical length of the grid in that direction. This was done to prevent computational stability issues due to the Courant–Friedrichs–Lewy (CFL) condition from complicating the study of parallel performance. The CFL criterion states that $\Delta t \leqslant \alpha \min(\Delta x, \Delta y, \Delta z)$, where α

TABLE 1. Wall-clock time in seconds for a 1D decomposition in the x direction (N_y = 31). The results of the fixed-size problems follow the diagonals from top left to bottom right. The results of the scaled-size problems follow along the rows from left to right.

Subdomain size per node (number of points per node)	Number of processors 1	2	4	8	16
32 × 31 × 32 (31 744)	839				
16 × 31 × 32 (15 872)	411	459	514	568	665
8 × 31 × 32 (7936)	203	256	282	358	413
4 × 31 × 32 (3968)	102	155	188	246	278
2 × 31 × 32 (1984)	58	104	138	187	204

is a proportionality constant, the Courant number, that depends upon the speed of the fastest wave. Keeping the same physical domain size while adding points would result in a higher-resolution grid. Due to the CFL criterion, this would require a shorter time step and, thus, more iterations of the model to achieve the same forecast time. The impact of this criterion on wall-clock time and speedup will be discussed later in this section.

Wall-clock times for the x-decomposition experiments are shown in Tables 1 and 2. Tables 3 and 4 show timings for the z decompositions and the two-dimensional decompositions, respectively. The results of the scaled-size problems follow along the rows of Tables 1, 2, and 4. The fixed-size problems, where the size of the physical domain remains constant, follow the diagonals of the tables from top left to bottom right. For example, the case of a 16 × 31 × 32 grid on a sequential processor (Table 1) is the same size as the 8 × 31 × 32 points per processor problem using two processors. Similarly, the 16 × 31 × 32 points per processor problem on 8 processors is the same size as the 8 × 31 × 32 problem on 16 processors.

b. Fixed-size experiments

The fixed-size runs showed a speedup with increasing numbers of processors (subdomains). The results of the fixed-size experiments for one-dimensional decomposition in the x direction are shown in Fig. 4 for three different grid sizes: 32 × 31 × 32, 64 × 62 × 32, and 96 × 93 × 32. Shown are the measured timings obtained by running the code on the compute cluster, and predicted timings obtained from the analytic performance model. The predicted timings agree closely with the measured timings. The slight difference between the timings is most likely due to the inaccuracies involved in obtaining the fitted curves in Fig. 3, which are used in the performance model. The timings for the 32 × 31 × 32 case show a result similar to that predicted by Amdahl's law in that the timing curve appears to approach an asymptote.

TABLE 2. Wall-clock time in seconds for a 1D decomposition in the x direction (N_y = 62). The results of the fixed-size problems follow the diagonals from top left to bottom right. The results of the scaled-size problems follow along the rows from left to right.

Subdomain size per node (number of points per node)	Number of processors 1	2	4	8	16
32 × 62 × 32 (63 488)	1584				
16 × 62 × 32 (31 744)	864				
8 × 62 × 32 (15 872)	410	513	563	695	800
4 × 62 × 32 (7936)	210	316	366	428	551

TABLE 3. Wall-clock time in seconds for a 1D decomposition in the z direction.

Total grid size (number of points)	Number of processors 1	2	4	8	16
32 × 31 × 32 (31 744)	839	434	271	189	183
64 × 62 × 32 (126 976)				825	667

Timings for the 64 × 62 × 32 and 96 × 93 × 32 grids suggest an asymptotic behavior similar to that for the 32 × 31 × 32 case (Fig. 4). Again, predicted versus measured timings are in close agreement. The experiments for the 64 × 62 × 32 grid could not use fewer than 8 processors due to memory constraints. Similarly, experiments for the 96 × 93 × 32 grid case could not be run with fewer than 16 processors.

The fixed-size speedup determined from these runs is 4.11 using 16 processors and 32 × 31 × 32 total grid points (Fig. 5), and 2.19 using 8 processors and 16 × 31 × 32 total grid points. The fixed-size speedup of 4.11 is 25.7% of the possible linear speedup. Under ideal conditions with no limiting factors, the wall-clock time would have been 52 s instead of the measured 204 s. Analysis of the 152-s loss shows that 125 s are due to interprocessor communication, 21 s are due to duplicate calculations, and 6 s are due to initialization. This sort of loss for fixed-size problems, where communication is responsible for the biggest loss and duplicate computations for the next biggest loss, is typical.

The effect of communication can be further illustrated by comparing calculation time with communication time as a function of the number of processors (Fig. 6). It is obvious that the calculation times (solid

TABLE 4. Wall-clock time in seconds for a 2D decomposition in x and y directions.

Subdomain size per node (number of points per node)	Number of processors 1	4	16
16 × 16 × 31 (7936)	178	241	431
8 × 8 × 31 (1984)	44	92	192

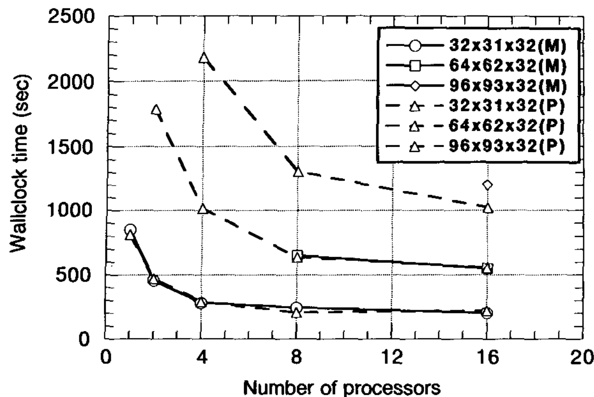

FIG. 4. Measured (M) and predicted (P) wall-clock time for fixed-size problem.

curves) decrease with increasing numbers of processors since the number of points per subdomain also decreases. However, because of the dependence of the communication time on the number of processors, the communication time increases. As stated earlier, this is a characteristic of using Ethernet for the communication network. Faster communication networks would permit the use of more processors before this effect would become significant.

The effect of various decompositions on communication time can be shown by comparing the wall-clock times for two decompositions, where the subdomains are of equal size but different dimension. Consider the results of a decomposition using a $16 \times 31 \times 32$ point subdomain, shown in the second row of Table 1, versus that using a subdomain of $8 \times 62 \times 32$ points, as shown in the third row of Table 2. The grid size is the same for corresponding entries in each of the two rows, yet the wall-clock times for the cases where $N_y = 62$ are higher than for the cases where $N_y = 31$. This is due to the greater communication time needed to transfer the larger shared subdomain boundaries. In the $N_y = 62$ case, the shared data planes are twice as large, resulting in twice the communications cost.

Greater communication efficiency can be realized when decomposing along the larger dimension of a nonsquare grid. This is typically the grid situation for most NWP grids where the horizontal dimension is greater than the vertical. A horizontal decomposition is also most efficient when processes in the model require vertical integration (e.g., radiation). The x-decomposition results for the fixed-size problem shown above can be compared with a z decomposition to illustrate this point. For example, for a $64 \times 62 \times 32$ grid, compare the 8-processor x decomposition (row 3, column 4 in Table 2) with the z decomposition (row 2, column 4 in Table 3). Similarly, for a $64 \times 62 \times 32$ grid, compare the 16-processor x decomposition (row 4, column 5 in Table 2) with the z decomposition (row 2, column 5 in Table 3). As expected, for the same size grid, the wall-clock time for the grids decomposed in the z direction are greater than those decomposed in the x direction. This is because of the additional communication time in the former.

Wall-clock times for the two-dimensional decomposition experiments are smaller than for comparable one-dimensional decomposition experiments with equal numbers of processors and grid points (Table 4). For example, the wall-clock time for the case of $32 \times 32 \times 31$ total grid points over 16 processors ($8 \times 8 \times 31$ subdomains) is 192 s. This is 6% faster than the one-dimensional decomposition run. This speedup results from shorter message lengths that, in turn, result in less communication time and fewer duplicate computations.

c. Scaled-size experiments

The scaled-speedup experiments used one- and two-dimensional domain decompositions similar to the fixed-size experiments. The size of the subdomain on each processor was held fixed as more processors were added. Consider a one-dimensional decomposition in the x direction for the case of an $8 \times 62 \times 32$ gridpoint subdomain with up to 16 processors, giving a total grid size of $128 \times 62 \times 32$ points. The results for this case are shown in Figs. 7 and 8. The predicted values in these figures were obtained from the analytic performance model by using the appropriate values for the whole domain size (N_x, N_y, N_z) and the number of processors P. The speedup is then the ratio of sequential execution time to parallel execution time as previously defined. In Fig. 7 the predicted timings agree closely with the measured timings. As with the fixed-size case, the discrepancies between timings relate to the inaccuracies involved in obtaining the fitted curves in Fig. 3. As seen in Fig. 7 and the second row of Table 2, the wall-clock time increases as the number of processors

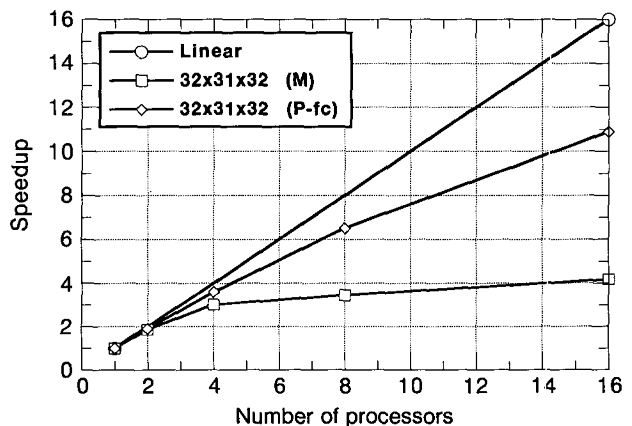

FIG. 5. Measured (M) and predicted (P-fc) speedup for fixed-size problem. The predicted speedup (P-fc) is for a network 100 times faster than Ethernet.

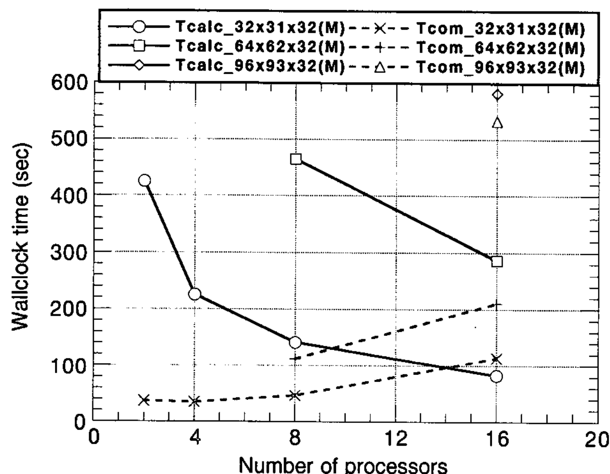

FIG. 6. Analysis of calculation versus communication time for fixed-size problem.

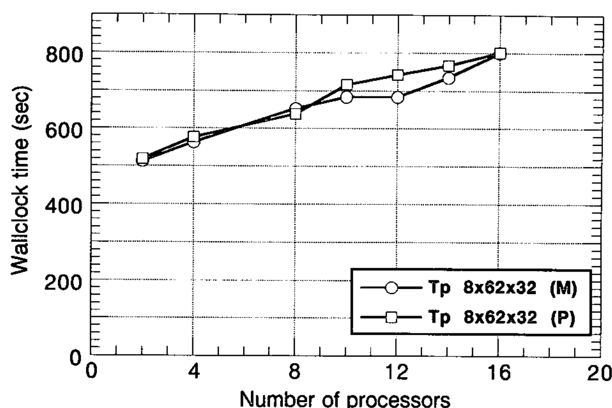

FIG. 7. Measured (M) and predicted wall-clock time when using Ethernet (P) and a "fast communication network" (P-fc) 100 times faster than Ethernet.

increases. Theoretically, the time to run the ARPS model should not change with the number of processors P unless the nonparallelizable part of the code depends upon P. For this decomposition, the nonparallelized initialization portion of the ARPS is a function of grid size and thus a function of the number of processors. Also, the communication time is a function of the number of processors (as shown in Fig. 3). Thus, these two factors contribute to an increase in wall-clock time as the number of processors increases. This is evident in both the measured and predicted timings. The scaled-problem speedup of 8.20 for the case of $128 \times 62 \times 32$ total points on 16 processors is 51.2% of the possible linear speedup of 16. The 390-s difference between the ideal wall-clock time and the measured time is attributable primarily to communication costs. Of the total wall-clock time of 800 s, interprocessor communications take 319 s, duplicate calculations take 43 s, and initialization takes 28 s.

The scaled-sized experiments using two-dimensional decomposition showed behavior similar to that of the one-dimensional decompositions. However, for similar size grids, the wall-clock times for the two-dimensional decompositions were smaller. For example, the case of a $32 \times 31 \times 32$ grid decomposed in the x direction over 4 processors had a wall-clock time of 282 s (row 3, column 3 in Table 1) versus 241 s for the two-dimensional decomposition of a $32 \times 32 \times 31$ gridpoint run on 4 processors (row 1, column 1 in Table 4). This difference is primarily the result of shorter message lengths. This leads to less communication time even though there are more messages in the latter. Also, as mentioned earlier, duplicate calculations are also a factor in the decreased time. Shorter messages imply that the length of the subdomain borders is also shorter, leading to fewer grid points where duplicate calculations are required.

Besides those discussed above, the factor that can most affect wall-clock time and limit the scalability of models like the ARPS is the CFL criterion. As described earlier, this criterion relates the dependence of the number of time steps to grid spacing. The fixed-size and scaled speedup experiments carried out here all used grids with the same grid spacing. In practice, it is more likely that one would wish to decrease the grid spacing to increase the model's resolution as the problem size is scaled up. However, changes in grid spacing can have an important deleterious effect on execution times adding considerably to the execution time of a problem because additional time steps are needed to reach the same forecast time.

The analytic performance model was used to estimate the wall-clock time of the ARPS for a scaled-up problem, while the physical domain size was held constant. These estimates are plotted in Fig. 9. The curve for the measured values, labeled (M), is the same as that in Fig. 7. Scaling up a problem in the x direction,

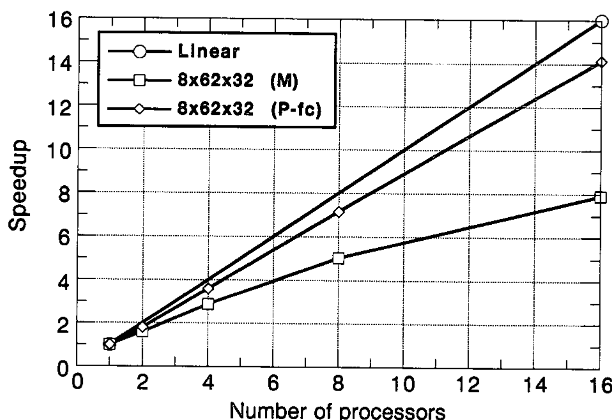

FIG. 8. Measured (M) and predicted (P-fc) speedup for scaled problem. The predicted speedup (P-fc) is for a "fast communications network" 100 times faster than Ethernet.

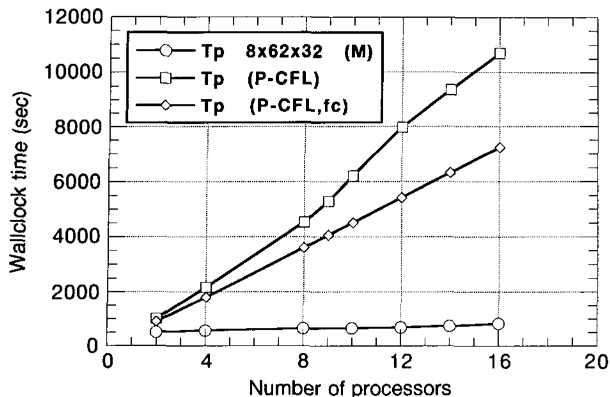

FIG. 9. Measured (M) wall-clock time and predicted wall-clock time for scaled problem when CFL condition is considered. Projections using Ethernet (P-CFL) and a "fast communication network" (P-CFL, fc) 100 times faster than Ethernet are shown.

for example, with 16 processors results in a grid with 16 times more points in that direction over the same physical length. As a result, the grid spacing is one-sixteenth of what it is for a single processor. Thus, 16 times more iterations of the model are needed to reach the same forecast time, as illustrated by the curve labeled "P-CFL" in Fig. 9.

6. Summary and conclusions

A finite-difference cloud model, the Advanced Regional Prediction System, was implemented on a compute cluster. Effects of grid decomposition strategies and communication speeds on model speedup were examined. Speedups were shown to be most affected by communication speeds. It was shown that the interprocessor communications time can be reduced by choosing appropriate decompositions and using faster networks. Generally, communication speeds much greater than that of the Ethernet are required to obtain significant model speedups.

Speedups can be enhanced by choosing appropriate decompositions. In general, decomposing along the x and/or y directions as opposed to the z direction is most effective since the former are usually the longest dimensions in NWP problems. Two-dimensional decompositions in these directions can be even more effective.

The measured and predicted timings for the parallelized code run on the compute cluster may seem disappointing at first glance. The best speedup achieved was 9.89 for a scaled problem using 16 processors. However, the speedup results presented here represent a worst-case scenario for the measured and predicted values for several reasons. First, a typical problem probably would require thousands of time steps as opposed to the 100 time steps used here. This would further amortize start-up costs due to initialization. Second, all measured and predicted timings were based on Ethernet network speeds. The projections from the analytic performance model (Figs. 5 and 8) suggest that speedups could double if gigabit network speeds (i.e., speeds 100 times faster than Ethernet) are used. Networks with this speed will be available in the near future. Third, the Ethernet communication times were shown to be a function of the number of processors, and the projections from the analytic performance model using fast communications assumed this dependence. However, communication times will be virtually independent of the number of processors with higher-speed networks. Thus, the projected model performance probably is an underestimate.

It was shown that the wall-clock time of scaled problems also can be greatly affected by the CFL criterion. This is due to the additional time steps needed to satisfy the criterion. However, even when the CFL criterion is considered, significant speedups can be achieved using faster communication networks.

Current wall-clock times of the ARPS on the compute cluster are a factor of 4 slower than running the same size problem on a single processor of a Cray Y-MP. Thus, even using faster networks, at least another factor of 2 in speedup will be needed if clusters of workstations are to compete with present-day supercomputers. This does not, however, detract from the effectiveness of using clusters as platforms for developing and testing MPP codes. Further, a factor not brought out by timing and speedup figures is that large memory is easier to provide with a compute cluster than for a sequential computer. Jobs that are too large to fit in the memory of a sequential machine or that would page in and out of memory may be distributed over 2 or more processors and run without paging. This could lead to significant speedups.

Several issues remain to be addressed for the parallelization of the ARPS model. The parallelized code must actually be run on a compute cluster with faster communications when it becomes available to verify projected model performance. The issue of load balancing across processors needs to be studied, especially as it is affected by microphysical calculations. There also is the issue of whether codes parallelized to run on a compute cluster can be easily ported to MPPs. These issues will be the focus of future work.

Acknowledgments. The authors wish to acknowledge Dennis Duke, Tom Green, Randy Langley of the Supercomputer Computations Research Institute and Neil Lincoln of SSESCO for their helpful suggestions during this research, and Karen Johnson for comments on the manuscript. Portions of this research were conducted with the support of the Supercomputer Computations Research Institute, which is partially funded by the U.S. Department of Energy through Contract DE-FC05-85ER250000. G. Riccardi was supported by the Office of Naval Research Contract N00014-93-1-0463. K. Droegemeier and M. Xue were supported by Grant ATM88-09862 from the National Science Foun-

dation to the Center for Analysis and Prediction of Storms, University of Oklahoma.

APPENDIX A

ARPS Model Equations

The ARPS model, as used in the parallelization experiments, was version 2.0. This model is three-dimensional, compressible, and nonhydrostatic. The continuous form of the model equations are transformed from Cartesian space (x, y, z) to a curvilinear system (ξ, η, ζ) to include terrain effects. In ARPS version 2.0, the physical grid is allowed to change with time; therefore, the coordinate transformation Jacobians are time dependent.

The zonal momentum equation is

$$\frac{\partial}{\partial t}(\rho^* u) = -\left[\frac{\partial}{\partial \xi}(\rho^* u U^c) + \frac{\partial}{\partial \eta}(\rho^* u V^c) + \frac{\partial}{\partial \zeta}(\rho^* u W^c)\right] + u\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c) + \frac{\partial}{\partial \zeta}(\rho^* W^c)\right]$$
$$-\left[\frac{\partial}{\partial \xi}(\bar{p} J_{\eta\zeta}^{yz}) + \frac{\partial}{\partial \xi}(p' J_{\eta\zeta}^{yz})\right] - \left[\frac{\partial}{\partial \eta}(\bar{p} J_{\zeta\xi}^{yz}) + \frac{\partial}{\partial \eta}(p' J_{\zeta\xi}^{yz})\right] - \left[\frac{\partial}{\partial \zeta}(\bar{p} J_{\xi\eta}^{yz}) + \frac{\partial}{\partial \zeta}(p' J_{\xi\eta}^{yz})\right]$$
$$+ \rho^* f v + G^{1/2} D_u. \quad \text{(A1)}$$

The meridional momentum equation is

$$\frac{\partial}{\partial t}(\rho^* v) = -\left[\frac{\partial}{\partial \xi}(\rho^* v U^c) + \frac{\partial}{\partial \eta}(\rho^* v V^c) + \frac{\partial}{\partial \zeta}(\rho^* v W^c)\right] + v\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c) + \frac{\partial}{\partial \zeta}(\rho^* W^c)\right]$$
$$-\left[\frac{\partial}{\partial \xi}(\bar{p} J_{\eta\zeta}^{zx}) + \frac{\partial}{\partial \xi}(p' J_{\eta\zeta}^{zx})\right] - \left[\frac{\partial}{\partial \eta}(\bar{p} J_{\zeta\xi}^{zx}) + \frac{\partial}{\partial \eta}(p' J_{\zeta\xi}^{zx})\right] - \left[\frac{\partial}{\partial \zeta}(\bar{p} J_{\xi\eta}^{zx}) + \frac{\partial}{\partial \zeta}(p' J_{\xi\eta}^{zx})\right]$$
$$- \rho^* f u + G^{1/2} D_v. \quad \text{(A2)}$$

The vertical momentum equation is

$$\frac{\partial}{\partial t}(\rho^* w) = -\left[\frac{\partial}{\partial \xi}(\rho^* w U^c) + \frac{\partial}{\partial \eta}(\rho^* w V^c) + \frac{\partial}{\partial \zeta}(\rho^* w W^c)\right] + w\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c) + \frac{\partial}{\partial \zeta}(\rho^* W^c)\right]$$
$$- \frac{\partial}{\partial \xi}(p' J_{\eta\zeta}^{xy}) - \frac{\partial}{\partial \eta}(p' J_{\zeta\xi}^{xy}) - \frac{\partial}{\partial \zeta}(p' J_{\xi\eta}^{xy}) + \rho^* B + G^{1/2} D_w, \quad \text{(A3)}$$

where the thermal and water buoyancy B is given by

$$B = g\left\{\left(1 + \frac{\theta'}{\bar{\theta}}\right)\left(1 - \frac{p'}{\gamma \bar{p}}\right)\left[\frac{1.0 - \bar{q}_v/(\epsilon + \bar{q}_v)}{1.0 - q_v/(\epsilon + q_v)}\right]\left(\frac{1 + \bar{q}_v}{1 + q_v + \Sigma\, q_{\text{liquid+ice}}}\right) - 1\right\}. \quad \text{(A4)}$$

The thermodynamic energy equation is

$$\frac{\partial}{\partial t}(\rho^* \theta) = -\left[\frac{\partial}{\partial \xi}(\rho^* \theta U^c) + \frac{\partial}{\partial \eta}(\rho^* \theta V^c) + \frac{\partial}{\partial \zeta}(\rho^* \theta W^c)\right]$$
$$+ \theta\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c) + \frac{\partial}{\partial \zeta}(\rho^* W^c)\right] + G^{1/2} D_\theta + \text{sources/sinks}. \quad \text{(A5)}$$

The pressure equation is

$$\frac{\partial}{\partial t}(\rho^* p') = -\left\{\frac{\partial}{\partial \xi}[\rho^*(\bar{p} + p')U^c] + \frac{\partial}{\partial \eta}[\rho^*(\bar{p} + p')V^c] + \frac{\partial}{\partial \zeta}(\rho^* p' W^c)\right\}$$
$$+ \left(\frac{\rho^*}{g^{1/2}}\right)^2 g w + (\bar{p} + p')\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c)\right] + p' \frac{\partial}{\partial \zeta}(\rho^* W^c)$$
$$- \bar{\rho}^2 \bar{c}^2\left[\frac{\partial G^{1/2} U^c}{\partial \xi} + \frac{\partial G^{1/2} V^c}{\partial \eta} + \frac{\partial G^{1/2} W^c}{\partial \zeta}\right] + \bar{\rho}\, \rho^* \bar{c}^2\left(\frac{1}{\theta}\frac{d\theta}{dt} - \frac{1}{E}\frac{dE}{dt}\right). \quad \text{(A6)}$$

The water substance equations are

$$\frac{\partial}{\partial t}(\rho^* q_\Psi) = -\left[\frac{\partial}{\partial \xi}(\rho^* q_\Psi U^c) + \frac{\partial}{\partial \eta}(\rho^* q_\Psi V^c) + \frac{\partial}{\partial \zeta}(\rho^* q_\Psi W^c)\right] + q_\Psi\left[\frac{\partial}{\partial \xi}(\rho^* U^c) + \frac{\partial}{\partial \eta}(\rho^* V^c) + \frac{\partial}{\partial \zeta}(\rho^* W^c)\right] + G^{1/2}(D_{q\Psi} + S_{q\Psi}), \quad \text{(A7)}$$

where $\Psi = v$ (vapor), c (cloud), r (rain), i (cloud ice), h (hail), and s (snow). In the equations above, u, v, and w are the Cartesian velocity components in the zonal, meridional, and vertical directions, respectively; p is pressure; θ is potential temperature; q is water substance; D is subgrid-scale mixing terms; ρ is density; and $\rho^* = \bar{\rho} G^{1/2}$, where the overbar indicates the 3D base state and the prime indicates departure from the base state.

These equations use the contravariant form of the velocity defined as

$$U^c = \frac{uJ_{\eta\zeta}^{yz} + vJ_{\eta\zeta}^{zx} + wJ_{\eta\zeta}^{xy}}{G^{1/2}}$$

$$V^c = \frac{uJ_{\zeta\xi}^{yz} + vJ_{\zeta\xi}^{zx} + wJ_{\zeta\xi}^{xy}}{G^{1/2}}$$

$$W^c = \frac{uJ_{\xi\eta}^{yz} + vJ_{\xi\eta}^{zx} + wJ_{\xi\eta}^{xy}}{G^{1/2}}, \quad \text{(A8)}$$

where U^c, V^c, W^c are the contravariant velocity components in the ξ, η, and ζ directions, respectively. The J's are the Jacobians of transformation and $G^{1/2}$ is the three-dimensional Jacobian of transformation. It is terms involving the Jacobians that lead to the need for redundant calculations on the outer boundaries of the subdomains.

The discrete version of the model uses quadratically conservative second-order finite differences in time and space. The grid system is the Arakawa C grid.

APPENDIX B

Compute Cluster

The workstation cluster consisted of 17 IBM RS6000 workstations connected via Ethernet. The host machine was a model 530 with 128 MB of memory, while the node machines were identical model 320Hs, each with 32 MB of memory. The clock speed for the model 530 and the model 320H is 25 MHz. The machines were placed on a separate strand of the Ethernet to minimize network traffic not associated with the parallelization experiments.

APPENDIX C

Performance Models

a. Sequential performance model

The performance model for the sequential code of the ARPS treats the code as having an initialization portion and an iteration portion. The total sequential execution time T_{seq} can be expressed as

$$T_{\text{seq}} = T_{\text{init}} + T_{\text{iter}}, \quad \text{(C1a)}$$

where T_{init} is the execution time for the initialization portion and T_{iter} for the iteration portion of the code. This may be approximated by

$$T_{\text{seq}} = (N_{\text{init}} + N_{\text{iter}})t_{\text{calc}}, \quad \text{(C1b)}$$

where N_{init} is the number of floating point operations in the initialization portion of the code, N_{iter} is the total number of floating point operations for the iterative portion of the code, and t_{calc} is the typical time to do a generic calculation such as an add or multiply. For the ARPS code, N_{init} and N_{iter} for a run of N_t time steps on an $N_x \times N_y \times N_z$ grid are

$$N_{\text{init}} = (71 \text{ adds} + 97 \text{ multiplies} + 5 \text{ divides} + 1 \text{ exponential})N_xN_yN_z \quad \text{(C2)}$$

and

$$N_{\text{iter}} = (725 \text{ adds} + 805 \text{ multiplies} + 44 \text{ divides})N_xN_yN_zN_t. \quad \text{(C3)}$$

If the add, multiply, divide, and exponential operations are grouped together, the total number of operations for a sequential run of the code on a uniprocessor computer for N_t time steps on an $N_x \times N_y \times N_z$ grid is

$$N_{\text{seq}} = N_{\text{init}} + N_{\text{iter}} = 174N_xN_yN_z + 1574N_xN_yN_zN_t. \quad \text{(C4)}$$

It then follows that the model for sequential execution time (C1b) can be rewritten as

$$T_{\text{seq}} = (174N_xN_yN_z + 1574N_xN_yN_zN_t)t_{\text{calc}}. \quad \text{(C5)}$$

b. Multiprocessor performance model

An analytic performance model for the parallelized version of ARPS using P processors is derived for a grid decomposed in the x direction as shown in Fig. 2b. Thus, there are P subdomains of size $N_xN_yN_z/P$.

The parallelized code's total execution time when run on P processors T_P consists of the computational time T_{calc} and the communication time T_{comm}. Assuming computations and communications cannot be overlapped, the expression for total execution time is

$$T_P = T_{\text{calc}} + T_{\text{comm}}. \quad \text{(C6)}$$

The code's computational time T_{calc} has an initialization component T_{init} and an iteration component T_{iter} like the sequential code. Also included is a redundant calculation time T_{dup} resulting from the parallelization of ARPS. Thus, the computational time T_{calc} can be expressed as

$$T_{calc} = T_{init} + \frac{T_{iter}}{P} + T_{dup}. \quad \text{(C7a)}$$

As with (C1) this may be approximated by

$$T_{calc} = \left(N_{init} + \frac{N_{iter}}{P} + N_{dup}\right) t_{calc}, \quad \text{(C7b)}$$

where N_{dup} is the total number of floating point operations for redundant calculation of Jacobian terms. The operations counts, N_{init} and N_{iter} in (C7b), are the same as for (C1b). For a run of N_t time steps on an $N_x \times N_y \times N_z$ grid decomposed in the x direction,

$$N_{dup} = 2(295 \text{ adds} + 374 \text{ multiplies}) N_y N_z N_t. \quad \text{(C8)}$$

If the add, multiply, divide, and exponential operations are grouped together in (C2), (C3), and (C8), then the total number of operations for a parallel run of the ARPS code on P processors for N_t time steps on an $N_x \times N_y \times N_z$ grid is

$$N_P = N_{init} + \frac{N_{iter}}{P} + N_{dup}$$
$$= 174 N_x N_y N_z + 1574 \frac{N_x N_y N_z}{P} N_t + 1338 N_y N_z N_t. \quad \text{(C9)}$$

Note that this is an overestimate since the first and last subdomains need only one extra data plane. Thus, for those subdomains, the coefficient in (C8) would be 1 instead of 2. It then follows from (C7b) and (C9) that the computational time expressed in terms of t_{calc} is

$$T_{calc} = \left(174 N_x N_y N_z + 1574 \frac{N_x N_y N_z}{P} N_t + 1338 N_y N_z N_t\right) t_{calc}. \quad \text{(C10)}$$

The code's total communication time T_{comm} when run on P processors consists of an initialization communication time T_{base} and an iterative communication time T_{itcom}. Time T_{base} is required to send base-state fields and initial values to each processor. Time T_{itcom} is required to send data to the outer borders of adjacent processors in the iterative part of the code. The expression for total communication time is thus

$$T_{comm} = T_{base} + T_{itcom}. \quad \text{(C11)}$$

The communication time required to transfer a packet of information from one processor to another depends on the start-up time, the size of the message, and the transmission rate. The start-up time is assumed to be independent of message size. With values for these parameters, communication times may be calculated in the performance model.

Initial communication in the ARPS model involves sending appropriate subdomains of the base-state fields, coordinate arrays, and prognostic fields to the local memory of each processor. In ARPS version 2.0, there are 7 base-state fields, 6 coordinate gridpoint fields, and 11 prognostic fields. Subdomains of the base fields and two time levels of the coordinate fields and prognostic fields have to be sent from the host to the appropriate nodes before time integration can begin. Thus, T_{base}, for a decomposition in the x direction, can be expressed in terms of a start-up time t_{start} and a transmission rate t_{trans} as

$$T_{base} = (P-2)\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 2\right) t_{trans}\right] + 2\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 1\right) t_{trans}\right]. \quad \text{(C12a)}$$

Node-to-node communication during the iterative part of model execution requires only that data on the inner borders of a subdomain of a given processor be transferred to adjacent processors per time step. For the decomposition illustrated here, the amount of data transferred between processors depends on the grid size of a vertical plane, $N_y \times N_z$, and the number of prognostic fields—in this case 11. Thus, the maximum number of words to be transferred per border is $11 N_y N_z$. The initialization communication time T_{itcom} can be written as

$$T_{itcom} = 2(t_{start} + 11 N_y N_z t_{trans}) N_t. \quad \text{(C12b)}$$

Thus, the total communication time T_{comm} is

$$T_{comm} = (P-2)\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 2\right) t_{trans}\right] + 2\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 1\right) t_{trans}\right] + 2 N_t t_{start} + 22 N_y N_z N_t t_{trans}. \quad \text{(C13)}$$

The results above allow the analytic performance model to be written as

$$T_P = T_{calc} + T_{comm}$$
$$= \left[174 N_x N_y N_z + 1574 \frac{N_x N_y N_z}{P} N_t + 1338 N_y N_z N_t\right] t_{calc} + (P-2)\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 2\right) t_{trans}\right] + 2\left[35 t_{start} + 41 N_y N_z \left(\frac{N_x}{P} + 1\right) t_{trans}\right] + 2 N_t t_{start} + 22 N_y N_z N_t t_{trans}. \quad \text{(C14)}$$

REFERENCES

Amdahl, G. M., 1967: Validity of the single-processor approach to achieving large-scale computer capabilities. *AFIPS Conf. Proceedings,* **30,** Reston, VA, 483–485.

Buzbee, B., 1993: Workstation clusters rise and shine. *Science,* **261,** 852–853.

CAPS, 1992: *ARPS Version 3.0 User's Guide.* Center for Analysis and Prediction of Storms, University of Oklahoma, 183 pp. [Available from the Center for the Analysis and Prediction of Storms, Univ. of Oklahoma, Norman, OK].

Droegemeier, K. K., M. Xue, P. V. Reid, J. Straka, J. Bradley III, and R. Lindsay, 1991: The Advanced Regional Prediction System (ARPS) version 2.0, Theoretical and numerical formulation. CAPS Report No. CAPS91-001, 55 pp.

——, ——, K. W. Johnson, K. Mills, and M. O'Keefe, 1992: Experiences with the scalable-parallel ARPS cloud/mesoscale prediction model on massively parallel and workstation cluster architectures. *Parallel Supercomputing in Atmospheric Science. Proc. Fifth Workshop on the Use of Parallel Processors in Meteorology,* G.-R. Hoffman and T. Kauranne, Eds., World Scientific Publishing, 99–129.

Fox, G. C., M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, 1988: *Solving Problems on Concurrent Processors.* Vol. I. *General Techniques and Regular Problems.* Prentice Hall, 592 pp.

Kauranne, T., 1990: Asymptotic parallelism of weather models. *The Dawn of Massively Parallel Processing in Meteorology,* G.-R. Hoffman and D. K. Maretis, Eds., Springer-Verlag, 303–314.

Sunderam, 1990: PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience,* **2,** 315–339.