11 ARPS Performance and Optimization on Single and Multiple Processor Systems

11.1. Introduction

The principal goal of CAPS-supported research in high-performance computing and parallel processing is to attain problem sizes and model execution rates with ARPS appropriate for operational storm-scale NWP and associated research and development. To achieve this rather ambitious goal, several approaches to parallelism are being investigated, ranging from the use of large, loosely-coupled networks of stand-alone workstations to shared-memory vector/scalar computers to distributed-memory parallel and massively parallel processors. As a complete application developed with modern software engineering strategies, ARPS serves as a unique testbed for MPP compilers and translation tools and has fostered important communication and collaborative studies among meteorologists, computer scientists, applied mathematicians, and electrical engineers throughout the world. Experience gained through this type of interaction has led to improvements in MPP tools, and will hopefully drive the creation of new tools appropriate for broad classes of application codes.

In this chapter we present performance statistics for ARPS on single and multiple processor systems, discuss basic strategies for code optimization, and describe future plans in high-performance computing.

11.2. General Performance Statistics

During the code development and testing process, ARPS has been run on as many different computers as possible to ensure portability, to identify peculiarities associated with certain compilers and operating systems, and to investigate various types of parallel implementation strategies (*e.g.*, data-parallel, message-passing). Because much of this work is published in the available literature with specific references provided at the end of this chapter, we seek here to condense this and related information by providing an overview of model efficiency on selected computers, tips for optimizing ARPS on vector and RISC machines, and strategies for running the model effectively on test and production problems. *The most recent timing statistics and related information can be obtained on the World Wide Web at URL address http://wwwcaps.uoknor.edu/ARPS/ARPS.html.*

Recent versions of ARPS have been run on the following machines:

• Connection Machine CM-200 (64K, 16K)
• Connection Machine CM-5 (1024 nodes)
• Cray Y-MP/8
• Cray-2/4-512
• Cray C90/16
• Cray Y-MP/J916
• Cray T3D/256 using PVM/MPI
• Alliant FX/80
• Convex C3840
• IBM RS6000/530H
• IBM RS6000/590H
• IBM SP-1 and SP-2 using PVM/MPI
• Parallel Cluster of IBM RS6000's
• VAX/VMS 6520 w/Vector Processor
• DEC ALPHA 3000 Model 800
• Sun SparcStation 10 Model 51
• SGI Power Challenge XL

A rough estimate of the memory required by ARPS 4.0 for these and other machines can be obtained using the following simple equations:

$$\text{Memory Required (in bytes)} \approx (nx)\,(ny)\,(nz)\,(284)$$

$$\text{Memory Required (in 64-bit Cray words)} \approx (nx)\,(ny)\,(nz)\,(71)$$

In order to provide the user with some feeling for execution times on typical machines, we ran benchmark cases *without I/O* to establish a baseline of ARPS' performance. The results, presented below in Table 11.1, were obtained using the following parameters in the model, repeated here from the log file so the user can conduct similar experiments (note that the array dimensions used are nx = 67, ny = 67, and nz = 35.):

```
&comment_lines
   nocmnt    =  2,
   cmnt(01)  = 'ARPS 4.0.21',
   cmnt(02)  = 'nx= 67, ny= 67, nz= 35',
 &END

 &jobname
   runname   = 'may20, V.4.0.22, May 20',
 &END

 &model_configuration
   runmod    =    1,
 &END

 &initialization
   initopt   =    1,
   inibasopt =    1,
   viniopt   =    1,
   ubar0     =           .0000,
   vbar0     =           .0000,
   pt0opt    =    1,
   ptpert0   =         4.0000,
   pt0radx   =      10000.0000,
   pt0rady   =      10000.0000,
   pt0radz   =       1500.0000,
   pt0ctrx   =      48000.0000,
   pt0ctry   =      16000.0000,
   pt0ctrz   =       1500.0000,
   buoyopt   =    1,
   sndfile   = 'may20.snd',
   rstinf    = 'arps40.rst003600',
   inifmt    =    1,
   inifile   = 'arps40.bin003600',
   inigbf    = 'arps40.bingrdbas',
   initime   = '1977-05-20.21:00:00',
 &END

 &terrain
   ternopt   =    1,
   mntopt    =    1,
   hmount    =       5000.0000,
   mntwidx   =      10000.0000,
   mntwidy   =      10000.0000,
   mntctrx   =      10000.0000,
   mntctry   =      10000.0000,
   terndta  = 'arpstern.data',
 &END

 &grid
   dx        =       1000.0000,
   dy        =       1000.0000,
   dz        =        500.0000,
   strhopt   =    0,
   dzmin     =        500.0000,
   zrefsfc   =           .0000,
   dlayer1   =           .0000,
   dlayer2   =     100000.0000,
   strhtune  =          1.0000,
   zflat     =     100000.0000,
   ctrlat    =         36.0000,
   ctrlon    =       -100.0000,
   umove     =           .0000,
   vmove     =           .0000,
 &END

 &projection
   mapproj   =    0,
   trulat1   =         36.0000,
   trulat2   =         36.0000,
   trulon    =       -100.0000,
   sclfct    =      .10000E+01,
 &END

 &timestep
   dtbig     =          6.0000,
   tstart    =           .0000,
   tstop     =        600.0000,
 &END

 &acoustic_wave
   vimplct   =    1,
   ptsmlstp  =    0,
   csopt     =    1,
   csfactr   =           .5000,
   csound    =        150.0000,
   tacoef    =           .6000,
   dtsml     =          1.0000,
 &END

 &numerics
   madvopt   =    2,
   sadvopt   =    2,
 &END

 &boundary_condition_options
   lbcopt    =    1,
   wbc       =    4,
   ebc       =    4,
   sbc       =    4,
   nbc       =    4,
   tbc       =    1,
   bbc       =    1,
   rbcopt    =    4,
   c_phase   =        300.0000,
   rlxlbc    =           .5000,
   pdetrnd   =    1,
 &END

 &exbcpara
   exbcname  = 'arpsexbc',
   tinitebd  = '1977-05-20.21:00:00',
   tintvebd  =      3600,
   ngbrz     =         5,
   brlxhw    =      .23000E+01,
   cbcdmp    =      .33333E-02,
   cbcmix    =      .10000E-02,
 &END

 &coriolis_force
   coriopt   =    1,
   coriotrm  =    1,
 &END
```

```
&turbulence
  tmixopt   =    4,
  trbisotp  =    1,
  prantl    =          1.0000,
  tmixcst   =           .0000,
  tmixcst1  =           .1000,
  tmixcst2  =           .9300,
  kmlimit   =           .5000,
&END

&computational_mixing
  cmix2nd   =    1,
  cfcm2h    =      .00000E+00,
  cfcm2v    =      .40000E-03,
  cmix4th   =    1,
  cfcm4h    =      .10000E-02,
  cfcm4v    =      .00000E+00,
&END

&divergence_damping
  divdmp    =    1,
  divdmpnd  =           .0500,
&END

&rayleigh_damping
  raydmp    =    1,
  cfrdmp    =           .0033,
  zbrdmp    =      12000.0000,
&END

&asselin_time_filter
  flteps    =           .1000,
&END

&microphysics
  moist     =    1,
  mphyopt   =    1,
  cnvctopt  =    0,
  wcldbs    =      .50000E-02,
  confrq    =        600.0000,
  idownd    =    1,
&END

&surface_physics
  sfcphy    =    1,
  landwtr   =    0,
  cdmlnd    =      .30000E-02,
  cdmwtr    =      .10000E-02,
  cdhlnd    =      .30000E-02,
  cdhwtr    =      .10000E-02,
  cdqlnd    =      .21000E-02,
  cdqwtr    =      .70000E-03,
  pbldopt   =    1,
  pbldpth0  =       1400.0000,
  sflxdis   =    1,
  sfcdiag   =    1,
&END
```

```
&soil_ebm
  sfcdat    =    1,
  styp      =    3,
  vtyp      =   10,
  lai0      =           .3100,
  roufns0   =           .0100,
  veg0      =           .3000,
  sfcdtfl   = 'arpssfc.data',
  soilinit  =    1,
  ptslnd0   =        293.0000,
  ptswtr0   =        288.0000,
  tsoil0    =        297.0000,
  wetsfc0   =      .25000E+00,
  wetdp0    =      .25000E+00,
  wetcanp0  =      .00000E+00,
  soilinfl  = 'arps40.soilinit',
  dtsfc     =          6.0000,
&END

&grdtrans
  cltkopt   =   0,
  tceltrk   =       120.0000,
  tcrestr   =      1800.0000,
  grdtrns   =   0,
  chkdpth   =      2500.0000,
  twindow   =       300.0000,
&END

&output
  dirname   = './',
  thisdmp   =       3600.0000,
  exbcdmp   =    0,
  hdmpfmt   =    1,
  filcmprs  =    0,
  basout    =    0,
  grdout    =    0,
  varout    =    1,
  mstout    =    1,
  iceout    =    0,
  trbout    =    0,
  sfcout    =    1,
  rainout   =    0,
  landout   =    1,
  tfmtprt   =       3600.0000,
  trstout   =       3600.0000,
  tmaxmin   =        300.0000,
  tenergy   =        300.0000,
  imgopt    =    0,
  timgdmp   =         60.0000,
  pltopt    =    0,
  tplots    =        600.0000,
&END

&debug
  lvldbg    =    0,
&END
```

Table 11.1. Timing statistics of ARPS 4.0 on selected platforms with no I/O for a 600 second simulation (100 timesteps).

Machine	Memory requirement	Compiler options *	Time (s)
IBM RS/6000 590[1]	44.64 MB	xlf -O3 -qstrict	2843.6
Sun Sparc 1000[2]	43.2 MB	f77 -fast	1.0×10^4
SGI R4000[3]	44.0 MB	f77 -O3	9050.9
Cray C90 1 processor[4]	11.26 MW 45 MB	cf77 -Zv -wf "-i operat3d.f"	256.7 (327 MFLOPS)

Notes on the above systems:

[1] Processor clock speed of 66 MHz
Theoretical top speed of 266 MFLOPS
Level 2 cache not supported.
128 MB 4-way interleaved memory, however, only two banks used.

[2] Processor speed of 40 MHz
Level 2 cache (1MB) with non-interleaved memory of 236 MB.

[3] Processor clock speed of 100 MHz.
Level 2 cache (1MB) with two-way interleaved memory of 96 MB.

[4] Processor clock speed of 250 MHz
Theoretical peak speed of 1 GFLOPS

* Options set by makearps. The makearps command for all machines except the Cray was **makearps** *-opt 3 arps40*. On the Cray, the command was **makearps** *-opt 3 -inline arps40.*

Note that ARPS 4.0 achieves 327 MFLOPS on a single processor of a Cray C90 for 100 time steps in a 67×67×32 domain using most of the physics. There are about 5850 floating point operations per grid point per time step. To evaluate maximum achievable speeds, we ran the above case using the explicit vertical solver on a 120×120×35 domain, for which ARPS achieved 457 MFLOPS on a single processor of the C90.

11.3. Basics of Optimization on Uniprocessor Systems

ARPS was designed not to run optimally on a given platform, but rather with general efficiency and ease of use so that optimization could be rapidly and effectively achieved once the target machine is known. It is therefore important for users to know which parts of the code are most computationally intensive and how to improve model efficiency on the most common architectures.

In this section we present statistics of the time consumed by the more important processes in ARPS using the same case as above but on an IBM RS/6000 Model 590 with 256 megabytes of memory in single-precision (32 bit) mode. This type of information, available as a runtime performance or profiling option on most machines, is presented in caller/callee chart fashion and *does not include I/O timing*. The total time at the top can be used to calculate the percentage associated with each process, and the individual timings can be used to obtain the percentage of time required for a given process. Although the behavior will vary across machines, one can expect the timings to be accurate to within 5 - 10%.

```
Total time: 3180.6
      initial: 1.7
      cordting: 2697.4
            tinteg: 2521.9
                  frcuvw: 488.0
                        mixuvw: 389.5
                              tmixuvw: 335.6
                                    cftmix: 113.0
                              cmix4uvw: 24.6
                              cmix2uvw: 13.2
                        advuvw: 71.6
                  frcp: 25.4
                  frcpt: 108.4
                        mixpt: 75.0
                              tmixpt: 49.0
                              cmix4pt: 7.5
                              cmix2pt: 3.2
                        advpt: 32.4
                  smlstep: 1327.6
                        solvuv: 292.1
                              pgrad: 138.9
                        slovwpex: 1013.6
                              wcontra: 127.5
                              tridiag: 111.3
                        solvpt: 5.0
            sfcflx: 2.6
            solvq: 199.3
                  mixq: 193.8
                        tmixq: 143.9
                              trbflxs: 168.0
                        cmix4q: 22.5
                        cmix2q: 9.7
            solvqv: 98.5
                  mixqv: 68.9
            microph: 144.6
                  revap: 48.3
                  satadj: 34.2
                  autocac: 30.7
                  qrfall: 18.4

            microph_ice:  889.32
```

We also conducted timing analyses on the Cray C90 to test the effects of different compiler options on code performance. This test case used is slightly different from the above in that it used the implicit vertical solver; nevertheless, one can anticipate similar behavior for other cases:

1. Warm Rain.
   Total time: 214 sec.
   Execution Speed: 354 MFLOPS.

2. Warm Rain, Code inlined with -*inline* option **
   Total time: 203 sec.
   Execution Speed: 373 MFLOPS.

3. Ice Microphysics.
   Total time: 684 sec.
   Execution Speed: 153 MFLOPS.
   Ice code Speed: 52 MFLOPS.

4. Ice Microphysics with -*ice* option *
   Total time: 316 sec.
   Execution Speed: 333 MFLOPS.
   Ice code Speed: 293 MFLOPS.

** The -*inline* option is provided on machines which support cross-file inlining. Currently, the file *operat3d.f* is provided for inlining.

* The -*ice* switches on the -*aggress* option for the Cray CF77 compiler. It helps vectorize loops which in normal cases would not vectorize.

In optimizing ARPS, one should focus attention on those routines and processes that use the most CPU time. Unfortunately, optimization is a nonlinear process, *i.e.*, improving the efficiency of one routine may degrade the efficiency of another. Further, due to variations in compiler technology among vendors, a code that runs efficiently on one platform may not run efficiently on another. Because it is impossible to deal with the specifics of each machine and compiler option, we present below the 10 most important considerations (in arbitrary order) when optimizing ARPS on either RISC/scalar or vector machines. Parallel code optimization, which is very hardware- and vendor-specific, is a much more complicated task and thus will not be addressed here.

1. Use large values for *nx* and *ny* on vector machines and parallel processors. For example, on the Cray C90, which has 128-word vector units, *nx* and *ny* could each be set to 128 (avoid using values near this number, *e.g.*, 125, 133). Values of *nx* and *ny* around 30 or 40, for example, will run much less efficiently on the C90. For parallel machines, performance is usually greatest for large-memory configurations.
2. Inline the subroutines in *operat3d.f* (on the Cray, use the *-inline* option when running **makearps**). This can also be done by hand, and neighboring loops can be joined (fused) as appropriate.
3. ARPS spends a significant amount of time in the routines in *solv3d.f*. Thus, attention should be paid to optimizing these routines before most others.
4. Convert variable *j3* (Jacobian) to *j3inv* (*i.e.*, *j3inv* = 1.0/*j3*) during initialization and computation. Future versions of ARPS will use this approach.
5. Use optimized mathematical libraries to improve performance of standard Fortran functions, *e.g.*, max.
6. Replace portions of ARPS code, *e.g.*, routine TRIDIAG, with a tridiagonal solver from a library optimized for the particular machine that you are using.
7. Use lookup tables to replace exponentiation.
8. Use compiler options that maximize your particular hardware features, *e.g.*, cache sizing, memory interleaving.
9. Set the values of *dx* and *dy* so as to maximize the value of *dtsml*.
10. Choose input parameters that reduce the frequency and complexity of runtime diagnostic calculations and I/O.

11.4. Economizing Computer Time Through a Judicious Choice of Model Parameters

A common question put forth by ARPS users is: “how can I adjust the domain size, timestep, and grid spacing so as to make the model run as economically as possible for my particular problem?” Of these three parameters, the grid spacing is typically the most important and is determined by the problem under consideration. For example, one nominally requires 1 km horizontal resolution to capture the dynamically important features (*e.g.*, mesocyclone, gust front, updraft and downdraft) of a supercell thunderstorm, whereas resolution as coarse as 15-30 km may be adequate for mesoscale weather (*e.g.*, fronts, drylines) in which explicit resolution of convection is unnecessary. Thus, one has only a limited amount of flexibility in the choice of spatial resolution.

Once the grid spacing is chosen, the timestep is for the most part determined as well through the so-called CFL criterion (see Chapter 4), which states that the model timestep is directly proportional to the grid spacing and inversely proportional to the speed of the fastest wave or signal in the model. Doubling the horizontal resolution in a 3-D model leads to a factor of 8 increase in execution time (twice the number of points in the two-dimensional horizontal plane and a factor of two decrease in the timestep). The penalty becomes a factor of 16 if the resolution is doubled in all three dimensions.

The final parameter - model domain size, is typically determined by available computer memory as well as a desire to maintain adequate distance between the features of interest and the boundaries, even when the boundaries are forced externally. Larger domain sizes obviously require more memory. Thus, a reduction in domain size is often the best way to economize in model simulations, particularly in tests made prior to full production jobs.

11.5. Implementation of ARPS on Parallel Systems: Strategies

From the beginning of the CAPS program, it was felt that parallel computers would provide the only means by which storm-scale numerical prediction could be accomplished in a realtime operational setting. Further, we believed that only through the use of parallel computers could scientific problems heretofore beyond our grasp finally be addressed. It is with these two points in mind that CAPS chose to place considerable emphasis on parallel computing in its design strategy for ARPS.

Parallelism may be exploited at several levels in NWP models like ARPS, ranging from local parallelism involving loops within the code to global parallelism involving subroutines and tasks. The work here addresses the issue of parallelism at the global level by treating ARPS as a coarse-grained application. Within this context, two general types of decompositions are possible: functional and spatial (also called domain) (Johnson *et al.*, 1994).

Functional decomposition involves performing parallel calculations of independent processes of a program on separate processors. This decomposition potentially requires a large amount of communication among processors because each prognostic equation is a function of most or all of the other prognostic variables. Thus, the updated prognostic fields must be transferred from the processors they are being calculated on to all other processors needing them after each timestep. The effectiveness of functional decomposition also depends on the number of independent processes, and the relative workload for each process. In a sophisticated atmospheric model, few processes are independent of the others. Programming techniques, such as the sharing of temporary work arrays among processors, often treat even independent processes as sequential. Such is the case for ARPS 4.0 and subsequent releases. Furthermore, computational load imbalances occur, resulting in poor concurrency because some processors remain idle waiting for others to catch up.

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; no global information is required at any particular grid point and inter-processor communications are required only at the boundaries of the subdomains. The inner border of a subdomain requires the outer border of the adjacent subdomain during a timestep because of spatial finite differences (Fig. 11.1). The outer border data resident in the local memories of adjacent processors are supplied by passing messages between processors.

Even though all processors run the same code, domain decomposition is subject to load imbalances when the computational load is data dependent. This is most often true for spatially-intermittent physical processes such as condensation, radiation, and turbulence in atmospheric models. However, when these processes are written such that the calculations at all points are identical, independent of what is actually occurring (as is the case for microphysical processes in ARPS version 3.0 and beyond), the difference in work load due to data-dependent branching is often small relative to the total task, and in most cases this degradation in concurrency and parallel performance is inconsequential.

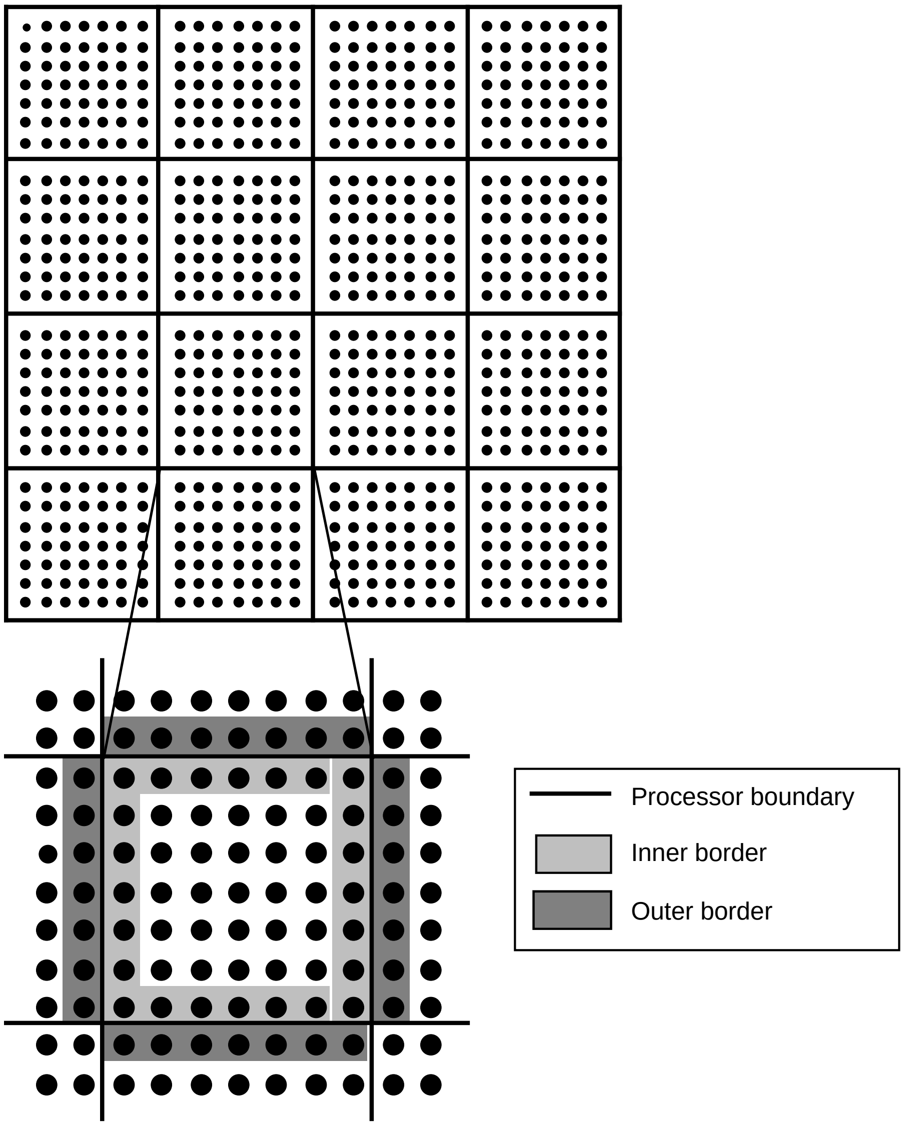

Figure 11.1. Spatial domain decomposition of ARPS grid. Each square in the upper panel corresponds to a single processor with its own memory. The lower panel details a single processor. Grid points having a white background are updated without communication, while those in dark stippling require information from neighboring processors. To avoid communication in the latter, data from the outer border of neighboring processors (light stippling) are stored locally. (adapted from Fox *et al.,* 1988).

The choice of decomposition strategy depends on the target multiprocessor architecture and the type of application. For current distributed memory systems, domain decomposition is a more effective parallel processing approach for ARPS, as described above and elaborated upon further below.

Several strategies exist within the domain decomposition paradigm for dividing the model into subdomains. Figure 11.2 shows possible decompositions for a three-dimensional grid.

Assuming no other complicating factors, a logical strategy is to partition in a way that minimizes the surface area of each subdomain relative to its volume. This keeps the computation-to-communication ratio high. The most natural decomposition from a coding standpoint is a one-dimensional decomposition in the z-direction, *i.e.*, assigning a fixed number of consecutive x-y planes to each processor (Fig. 11.2a). However, for most NWP grids, the computational domain is usually a parallelepiped with the least number of points in the z-direction (*i.e.*, the atmosphere is effectively confined in the vertical, with the lateral extent chosen at the user's discretion). Further, one-dimensional decomposition in the z-direction would be even less efficient if vertical integrations or sums, such as those involving column-wise physics or the hydrostatic relation, are central to the model solution process. In the newest version of ARPS, a vertically implicit scheme has been applied to the acoustic modes, thereby coupling the calculations vertically. This suggests that one-dimensional decompositions, with cuts made in either the x- or y-directions (Fig. 11.2b, c), are better choices.

Two-dimensional decomposition involves decomposing the computational domain in two coordinate directions simultaneously (Fig. 11.2d). The decision between a one-or two-dimensional formulation depends, in part, upon the characteristics of the communication network linking the processors. A one-dimensional decomposition will have longer but fewer messages than its two-dimensional counterpart, and thus may be more appropriate for a communication network having high message startup costs relative to transmission costs. A two-dimensional decomposition will transfer fewer data, at least for nearly square grids, despite a greater number of messages. Thus, for a nearly square grid, which is the most likely case for NWP problems, a two-dimensional decomposition may be the best choice when network transmission rates (hardware and software) are relatively slow.

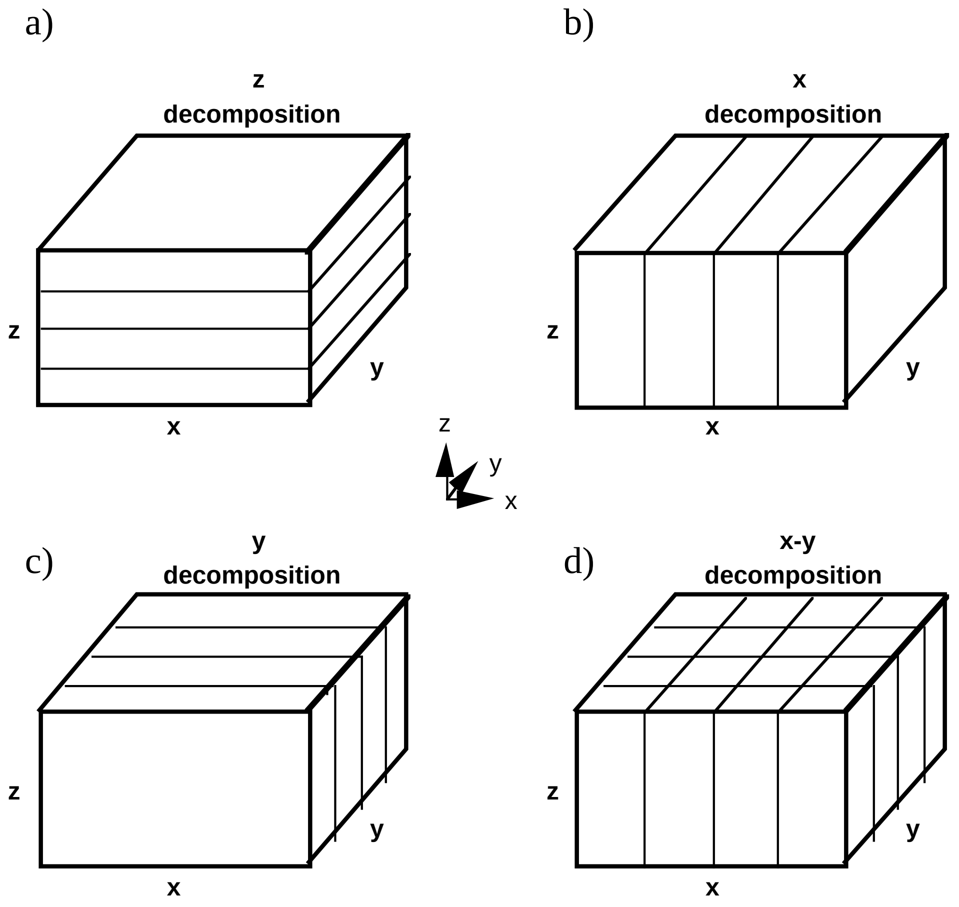

Figure 11.2. One- and two-dimensional domain decomposition strategies for a three-dimensional grid.

11.6. Implementation of ARPS on Parallel Systems: Timing —

We have evaluated ARPS on numerous parallel architectures, and in this chapter we compare and contrast our experience using four separate approaches: 1) A data-parallel approach in which serial Fortran-77 is translated into machine-specific parallel Fortran using the Fortran-P translator; 2) Explicit message-passing using a portable communications library, Parallel Virtual Machine (PVM), on a networked cluster of IBM RISC (reduced instruction set chip) System 6000 workstations; 3) A data-parallel approach in which serial Fortran-77 is translated into Connection Machine Fortran (CMF) on the CM-5 using the Thinking Machines CMAX translator; and 4) Explicit message-passing applied to the original Fortran-77 ARPS on the CM-5 and Intel Delta, representing the first step toward a port using Fortran-D/High-Performance Fortran.

CAPS has been fortunate to collaborate with a number of outstanding scientists in its parallel computing efforts, and the first such collaboration is with Prof. M. O'Keefe and colleagues at the Army High Performance Computing Research Center (AHPCRC), University of Minnesota. CAPS helped support the refinement of Fortran-P, a stylized subset of Fortran-77 that allows the coding of self-similar applications programs, *i.e.*, those having the property that the algorithm applied to the global data domain is the same as that applied to each sub-domain in a spatial decomposition. The current version of Fortran-P generates machine-specific Fortran from Fortran-77. In so doing, Fortran-P automatically converts a Fortran-77 source code to Fortran-90 and inserts data layout directives that describe the spatial decomposition of the model domain across the processing array.

In our first tests using Fortran-P, the 3-D ARPS code was translated and executed on the CM-5. As with earlier version of CM Fortran, high virtual processor ratios had to be used to achieve good performance. Since ARPS uses three boundary points (one on the left and two on the right) due to grid staggering (see Figure 11.1), each subdomain also required 3 boundary points. Because of ARPS memory requirements (approximately 60 full-domain-sized 3-D arrays), the subdomains needed to be small to avoid memory limitations on the CM-5. A reasonable size turned out to be 5^3 real points per subdomain, or 8^3 total points per subdomain. This means that only $5^3/8^3$ = 24.4 % of the memory used was for real points, the remainder associated with the fake boundary zones. This memory inefficiency, associated with large VP ratios, resulted in speeds only 25% of real-time on 32-nodes of a 544-node CM-5 for a problem size of 40x40x40 grid points. A 128-node run using 80x80x80 points ran at about the same speed relative to real-time.

As mentioned earlier, CM Fortran compiler versions prior to CMF 2.1 Beta 1.0 (which was unfortunately not available for our preliminary study conducted almost two years ago) yielded poor memory efficiency and hence poor computational efficiency as well. The alpha version of the Fortran-P translator did not generate the appropriate loop form for the CM Fortran 2.1 Beta 1.0 compiler; this has been corrected in the beta Fortran-P translator. Early results with the CMF 2.1 Beta 1.0 compiler are very encouraging. In particular, high subgrid ratios become much less important and replication of subgrids on processors (which leads directly to memory inefficiency) was not critical to performance. We have also employed an early version of the beta Fortran-P translator to generate Fortran-90-styled loops (instead of the serial DO loop structure employed in the alpha version) and used the Cray FPP in-liner to reduce the significant subroutine overheads present in ARPS due to its operator-based style. These translation improvements led to significant performance increases; early results indicate that a 40×40×40 grid size now runs at twice the speed of the weather on 32-nodes, a factor of 8 increase in speed over our earlier results.

Our second collaboration involved several scientists at Thinking Machines Corp., principally Drs. G. Sabot and S. Wholey, and has led directly to improvements in a commercial tool that is now available internationally. Sabot’s group developed the first general-purpose commercial translation tool, known as CMAX, for the Connection Machine, the purpose of which is to map Fortran-77 codes to CM Fortran. Sabot and colleagues became interested in using ARPS as a testbed for CMAX mainly because of its clean style and modern Fortran constructs. With nominal effort, they successfully translated the entire 65,000 lines of ARPS 3.1 into CM Fortran and produced solutions that agreed with our validation suite results. This code conversion effort, described in Sabot *et al.* (1993), was one of five finalists for the 1993 Gordon Bell Prize in High Performance Computing. It validates our initial decision to produce a model that can easily be ported to MPP systems, and indicates that ARPS will be able to achieve the domain sizes, spatial resolutions, and performance deemed necessary for operational storm-scale prediction.

Because the singular goal of parallel computing is a reduction in wallclock time, with speedups linearly proportional to the number of processors a desired achievement, we have collected a number of timings from the CMAX-translated version of ARPS 3.1 for a 1 hour simulation of a supercell storm (Figure 11.3). As described by Droegemeier *et al.* (1994), the translated code shows a consistent, nearly linear speedup for all problem sizes, implying excellent scalability even when using several hundred processors (similar scalability is anticipated with ARPS 4.0). Most importantly, we were able to run our “ultimate” domain of size 1024x1024x32 points on the 1024 node CM-5 at Los Alamos National Laboratory, at a rate about 6 times slower than realtime. We believe that an additional factor of 4 to 6 improvement in performance can be achieved with further optimization. If future hardware improvements result a factor of 50 to 100 increase in *sustained* speed, which seems likely in the next few years, we should easily reach of our goal of running 50 to 100 times faster than the weather using 1000 x 1000 x 40 gridpoints with a uniform horizontal resolution of 1 km.

It is important to recognize that high-end computers are very expensive and unlikely to be available for dedicated use in operational storm-scale prediction. During the past 4 years, we have therefore placed considerable emphasis on the use of workstations and workstation clusters to achieve good price-performance with ARPS, and were one of the first, if not the first, to port a cloud-scale model to a workstation cluster (ARPS 2 in 1990).

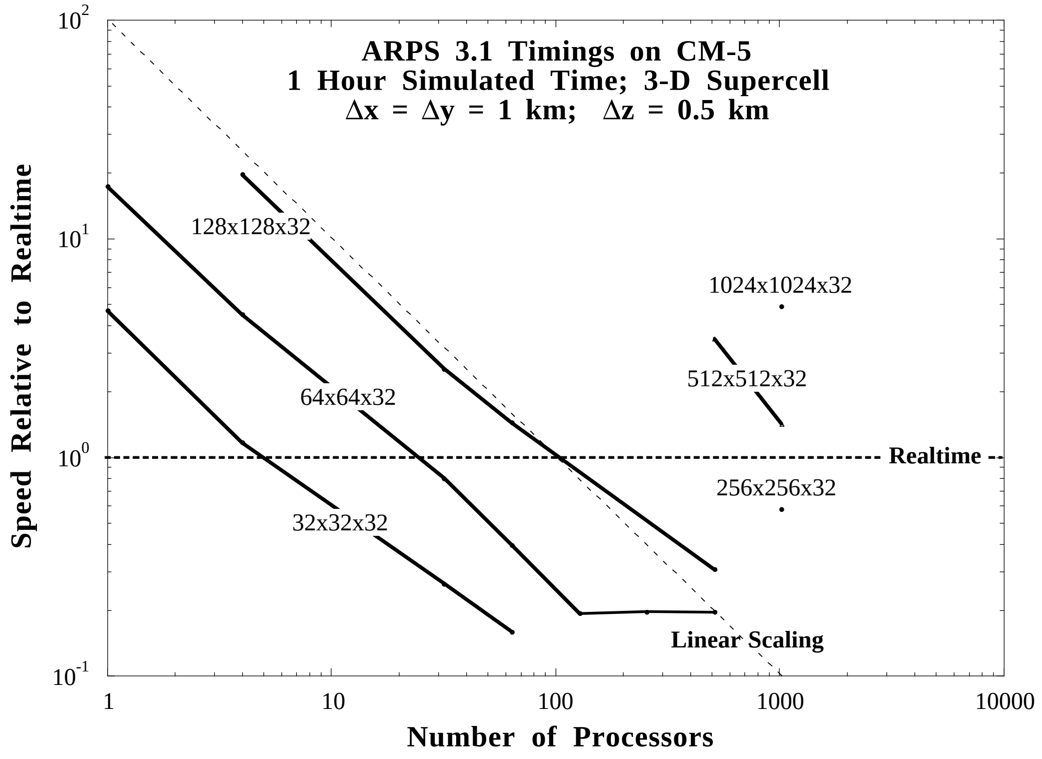

Figure 11.3. Speed relative to real-time for simulating 1 hour of supercell storm evolution on the Connection Machine CM-5 as a function of the number of processors. Each connected line represents a single problem size, and points below the horizontal dashed line represent model runs that execute faster than real-time. The diagonal dashed line represents linear (optimal) scaling. From Droegemeier *et al*. (1995).

With that as a preface, our third collaboration is with Dr. K. Johnson, formerly of the Supercomputer Computations Research Institute at Florida State University and now with the National Weather Service. This effort involves implementing ARPS on workstation clusters using the Parallel Virtual Machine (PVM) message passing library (Johnson *et al*., 1994). The results of several tests, described in the journal article just cited, show that the biggest obstacle to achieving significant speedups with ARPS (and, most likely, with other models of similar type) when run on a workstation cluster is communication associated with transferring data to adjacent processors. Next in importance is the effect of redundant calculations, followed by computations for generating the initial fields and synchronization between processors due to load imbalances. These effects are tied to the serial portion of the program, and limit speedup in a fashion analogous to the serial portion of Amdahl's law. Remedies to these problems are discussed below.

Inter-processor communication time can be reduced by choosing appropriate decompositions and using faster networks. Some decomposition strategies have advantages over others when considering communication costs. As mentioned above, decomposing in two or more dimensions leads to less data transfer than in one dimension. A rule of thumb is to select subdomains that minimize the surface area while maximizing the volume, thereby increasing the computation-to-communications ratio. For highly non-isotropic computational domains, decomposing along the larger dimensions leads to greater efficiency.

Faster communication networks (including both hardware and software) are likely to produce the greatest speedup improvements on workstation clusters. Increasing the communication speed by a factor of 100 in the analytic performance model suggests an increase in speedup by a factor of two over the Ethernet results. Software/hardware network systems with this speed will be available for clusters in the near future, and considerable improvement in network speed is also anticipated on MPPs, though at a significantly greater cost than for clusters of loosely-coupled processors.

The issue of load balancing across processors did not arise in the parallel experiments conducted here because all subdomains were of equal size and no data-dependent process existed; thus, all processors, being of equal speed, performed equivalent amounts of computation.

Our final collaboration is with the Northeast Parallel Architectures Center at Syracuse University. This effort, led by Dr. K. Mills, has resulted in a Fortran-77-plus-message-passing version of ARPS 3.1 known as mpARPS (Lin *et al.*, 1993). By overlapping communication and computation, mpARPS achieves a greater speedup on the Intel Delta system than on the CM-5, with a price-performance ratio 5 times greater than for the CMAX version. More specifically, for mpARPS (CMAX-ARPS) on the Delta (CM-5), the best rate measured with 512 processors is 1.6 (0.4) million gridpoint updates per timestep per second, with a maximum problem size of 20 (8) million points. For the 128x128x32 point domain using 32 processors, mpARPS on the Delta was 2.4 times faster than CMAX-ARPS on the CM-5.

In numerical weather prediction, the megaflop is a relatively meaningless measure of code performance unless examined in light of one important factor: the ability of a model to stay ahead of the evolving weather. Based on ideas originally proposed by Dr. Ken Johnson, we present in Fig. 11.4 a nomograph of the ratio of dedicated wall clock time (DWT) to model simulated time (ST) which, when plotted against model memory usage (equivalently, domain size), allows one to project future hardware (code) requirements based on anticipated code (hardware) performance. DWT is simply the elapsed wall-clock time when the machine is dedicated to a single user or job. If the ratio of DWT to ST is less than unity, the model runs faster than the weather. CAPS believes that a 1000×1000×20 km or so domain at a uniform horizontal grid spacing of 1 km is required for regional-scale prediction. As a result, from the figure below it is clear that sustained performance of 1 teraflop will be required to predict the weather 100 times faster than it occurs. This means that a 4 hour forecast could be produced in slightly less than two and a half minutes, neglecting time-consuming tasks such as data ingest, quality control, assimilation, post-processing and dissemination, *etc*. Although Figure 11.4 was drafted based on ARPS 3.0 timing, ARPS 4.0 with exactly the same configuration yields similar results.

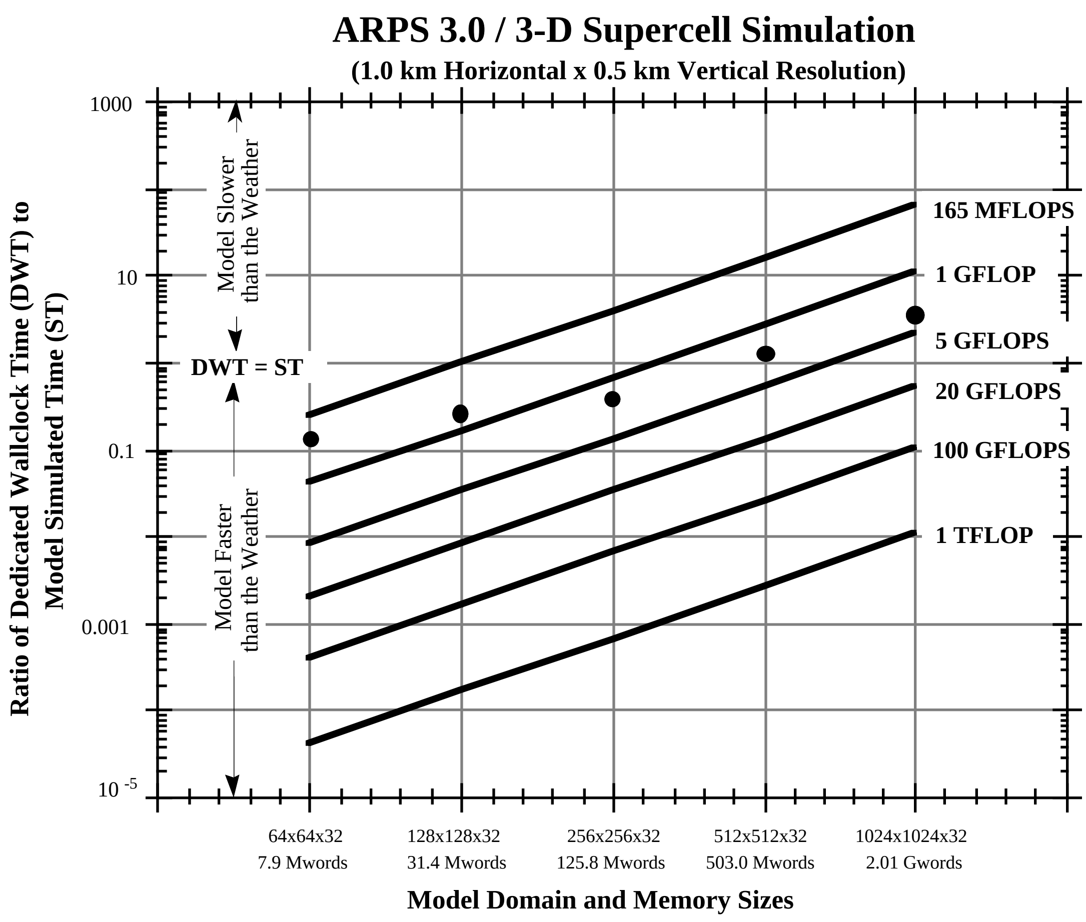

Figure 11.4. Nomograph of the ratio of dedicated wallclock time (DWT) to simulated time (ST) plotted against domain size for ARPS 3.0. The diagonal lines represent sustained model performance in megaflops, gigaflops, and teraflops. When DWT=ST, ARPS is running as fast as the weather evolves. The bold dots indicate actual ARPS performance for each of the five domain sizes shown. From Droegemeier *et al.* (1995).

Within CAPS, we are focusing our MPP efforts on ports of ARPS 4.0 to the Cray T3D using a set of translation tools built in-house using the data parallel paradigm offered by the Cray CRAFT Fortran compiler. Because ARPS is continually evolving, we require the ability to parallelize new versions of the code with minimal investment of user time. Creating a message-passing version manually is cumbersome and requires porting and maintaining multiple versions of the code. The data parallel port using automatic translation tools is clearly a more attractive alternative, particularly since the HPF standard will allow the latest version of the code to be ported to current and future parallel machines with limited modifications to the translation tool.

With the official release of ARPS version 4.0, a PVM version is included. The commands to compile, link and execute this version can be found in Chapter 3.

For more information about various parallel versions and current porting efforts, please send e-mail to arpsuser@uoknor.edu or reference our World Wide Web page at http://wwwcaps.uoknor.edu/ARPS/ARPS.html.

11.7. Publications Describing the Construction, Efficiency, Operational Application, and Parallelization of ARPS

The following publications (in alphabetical/chronological order) contain information on the construction, efficiency, and parallelization of ARPS model on a number of platforms. Reprints and preprints can be obtained by sending a request to *arpsuser@uoknor.edu*.

Chrisochoides, N., K. K. Droegemeier, G. Fox, K. Mills, and M. Xue, 1993: A methodology for developing high performance computing models: Storm-scale weather prediction. *Proc., Society for Computer Simulation Multiconference*, March 29-April 1, Arlington, VA.

Droegemeier, K. K., M. Xue, K. Johnson, K. Mills, and M. O’Keefe, 1993: Experiences with the scalable-parallel ARPS cloud/mesoscale prediction model on massively parallel and workstation cluster architectures. *Parallel Supercomputing in Atmospheric Science*, G.R. Hoffman and T. Kauranne, Eds., World Scientific, 99-129.

Droegemeier, K. K., M. Xue, K. Johnson, M. O’Keefe, A. Sawdey, G. Sabot, S. Wholey, N.T. Lin, and K. Mills, 1995: Weather prediction: A scalable storm-scale model. Chapter 3 (p. 45-92) in *High Performance Computing*, G. Sabot (Ed.), Addison-Wesley, Reading, MA, 246pp.

Droegemeier, K. K., M. Xue, A. Sathye, K. Brewster, G. Bassett, J. Zhang, Y. Liu, M. Zou, A. Crook, V. Wong, and R. Carpenter, 1996: Realtime numerical prediction of storm-scale weather during VORTEX '95, Part I: Goals and methodology. Preprint to be presented at the *18th*

Conf. on Severe Local Storms, 15-20 Jan., San Francisco, CA, Amer. Meteor. Soc.

Janish, P.R., K. K. Droegemeier, M. Xue, K. Brewster, and J. Levit, 1995: Evaluation of the advanced regional prediction system (ARPS) for storm-scale modeling applications in operational forecasting. *Proc., 14th Conf. on Wea. Analysis and Forecasting,* 15-20 Jan., Dallas, TX., Amer. Meteor. Soc., 224-229.

Johnson, K.W., J. Bauer, G.A. Riccardi, K. K. Droegemeier, and M. Xue, 1994: Distributed processing of a regional prediction model. *Mon. Wea. Rev.*, **122**, 2558-2572.

Lin, N.-T., K. Mills, Y.-C. Chen, K. Droegemeier, and M. Xue, 1993: A message passing version of the Advanced Regional Prediction System (mpARPS). 17 pp. (Internal report, available from CAPS.)

Purnell, D.K., M. J. Revell, and P. N. McGavin, 1995: Field object design of a numerical weather prediction model for uni- and multiprocessors. *Mon. Wea. Rev.*, **123**, 401-429.

Sathye, A., G. Bassett, K. Droegemeier, and M. Xue, 1995: Towards operational severe weather prediction using massively parallel processors. *Int. Conf. on High Performance Computing*, New Dehli, India, Tata McGraw-Hill Publishing Company, Limited.

Sawdey, A., M. O'Keefe, O. Meirhaeghe, M. Xue, and K. Droegemeier, 1993: Conversion of the ARPS 3.0 mesoscale weather prediction code to CM-Fortran using the Fortran-P translator. Preprint 93-089, Army High Performance Computing Research Center, University of Minnesota, 7pp.

Xue, M., K. Droegemeier, and V. Wong, 1995: The Advanced Regional Prediction System and Realtime storm-scale weather prediction. Preprints, *Int. Workshop on Limited-Area and Variable Resolution Models*. Beijing, China, World Meteor. Org., 7pp.

Xue, M., K. Brewster, F. Carr, K. Droegemeier, V. Wong, Y. Liu, A. Sathye, G. Bassett, P. Janish, J. Levit and P. Bothwell, 1996: Realtime numerical prediction of storm-scale weather during VORTEX '95, Part II: Operations summary and example predictions. Preprint to be presented at the *18th Conf. on Severe Local Storms*, 15-20 Jan., San Francisco, CA, Amer. Meteor. Soc.