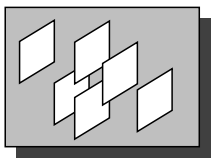


Northeast Parallel Architectures Center

The Parallelization of a Weather Prediction Model

Gregor von Laszewski
gregor@nova.npac.syr.edu

Technical Report

SCCS 533

Science and Technology Center
111 College Place
Syracuse, NY 13244-4100
Tel.: (315) 443-1722, 1723; Fax: (315) 443-1973

Contents

Chapter 1

Introduction

Recent devastating events caused by storms show how important the development of a reliable storm prediction system is. One such effort was started at the Center for Analysis and Prediction of Storms (CAPS) in Oklahoma. This institute tries to develop techniques for the practical prediction of weather phenomena on scales ranging from a few kilometers to hundreds of kilometers.
The use of a prediction model is possible in near future because a network of 175 Doppler radars will be installed around the U.S in order to gather the necessary initial data for the model. The initialization is an important part of this application since the precise initial data leads to more accurate predictions. It is easy to imagine that this data accumulation leads to huge storage requirements. One way to deal with this problem is to distribute the prediction of a storm at the area of interest. Since, many sites have different computers it is useful to develop a highly portable source code. Another important requirement is that a modification of the program should be possible with moderate effort in order to incooperate new computational schemes determining the modeling equations.
Naturally a storm has to be predicted as fast and long as possible with very high accuracy in order to avoid damages by invoking early prevention methods. This can be achieved by, e.g. using supercomputers for solving the modeling equations.
This report describes the parallelization of a weather prediction code using the

dataparallel programming scheme. In addition, it is shown how to obtain a version for message passing computers while using High Performance Fortran. Benchmarking for the dataparallel program is done on a CM5 with 32 nodes and vector units.

1.1 The Advanced Regional Prediction System

The program for the prediction of storms developed by CAPS is called *Advanced Regional Prediction System* (ARPS). In this section the features of the ARPS code important for a parallelization are summarized. In addition a small example is used to show the modularity introduced by the *operator* model.
The following design characteristics for the ARPS code are important for a possible parallelization and the portability onto many different machines:

Flexibility: The code is modular so that it is easy to modify it.

Portability: The code is written in Fortran 77.

Support:
- Documentation is provided for the source code and for the user.
- The CAPS project is an ongoing project and support by the authors is given for questions arising not covered in the documentation.

1.2 Modularity of the Model

In ARPS the differential equations are described by their operators. This makes the extension of the modeling equations used in ARPS easier. We illustrate the operator scheme for the the simple scalar conservation law:

$$\frac{\delta T}{\delta t} = -\frac{\delta(uT)}{\delta x} - \frac{\delta(vT)}{\delta y} - \frac{\delta(wT)}{\delta z}$$

To solve this equation the term can be expanded in the usual way as shown in [?].

```
do k=1,nz
  do j=1,ny
    do i=1,nx
```

```
          T(i,j,k,future) = T(i,j,k,past)
                   -.5 * rdx * dt *
                     u(i,j,k,now) * (T(i+1,j,k,now)+T(i,j,k,now))
                    -u(i-1,j,k,now) * (T(i+1,j,k,now)+T(i-1,j,k,now))
                   -.5 * rdx * dt *
                     v(i,j,k,now) * (T(i,j+1,k,now)+T(i,j,k,now))
                    -v(i,j-1,k,now) * (T(i,j+1,k,now)+T(i,j-1,k,now))
                   -.5 * rdx * dt *
                     w(i,j,k,now) * (T(i,j,k+1,now)+T(i,j,k,now))
                   - w(i,j,k-1,now) * (T(i,j,k+1,now)+T(i,j,k-1,now))
      end do
    end do
  end do
```

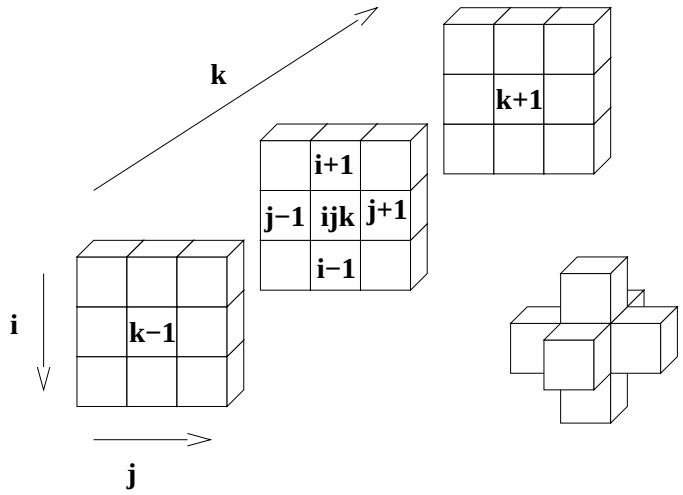


Figure 1.1: Data dependency in the calculation of the conservation law

In case new equations and parameters are added it is necessary to put considerable amount into rewriting the program. In addition, the parallelization of this code fragment is difficult due to data dependencies inherented between variables in the loop construct. The dependencies are shown in Figure 1.1 and can be drawn in form of a stencil. Since the sequential loop is traversed in a particular order it is not straight forward how to parallelize the code. To avoid

this problem the introduction of a small set of operators is useful. Let avg_d and dif_d be the counterparts of the spatial average and spatial derivative in direction $d \in \{x, y, z\}$. Let *aamult* denote the element wise multiplication of two matrices. Than the equation of the conservation law can be expressed as shown in Figure 1.2.

```
begin parallel
  temp1 ← avg_x(T)
  temp2 ← avg_y(T)
  temp3 ← avg_z(T)
end parallel

begin parallel
  temp1 ← aamult (U, temp1)
  temp2 ← aamult (U, temp2)
  temp3 ← aamult (U, temp3)
end parallel

begin parallel
  temp1 ← dif_x(temp1)
  temp2 ← dif_y(temp2)
  temp3 ← dif_z(temp3)
end parallel

do k=1,nz in parallel
  do j=1,ny in parallel
    do i=1,nx in parallel
      T(i,j,k,future) = T(i,j,k,past)
                      - 2 × dt × (temp1(i,j,k,now)
                      +  temp2(i,j,k,now)
                      +  temp3(i,j,k,now))
    end do
  end do
end do
```

Figure 1.2: Procedural formulation of the conservation law

The pseudo code for the three operators used is given in the appendix. Parallelism occurs in three ways:

1. Each operator can work in parallel on the three temporary arrays.

2. There are no dependencies in the do loop. Therefore, the do loop is easily parallelizable.

3. The operators avg, dif, and aamult, the multiplication of two matrices element by element, can be parallelized easily.

Chapter 2

Parallelization

To avoid mistakes done in previous attempts to parallelize the ARPS code a more solid software engineering approach is followed (Figure 2.1).

Analysis In the analysis phase the existing ARPS versions have been evaluated carefully for a parallelization. (See Sections 2.1,2.2)

Definition In consideration of the problem analysis the tasks are defined to complete the project (See Sections 2.1,2.2).

Design The parallel code and supporting algorithms are designed (See Sections 2.1,2.2, 2.3, Chapters 3, 4).

Testing The program is tested and compared with the sequential program. Benchmarking is done with real storm data used in weather prediction codes (See Chapter 5).

2.1 Analysis of the Parallel ARPS Code (Version 3.0)

The ARPS code version 3.0 has been previously parallelized for the DECmpp available at NPAC [?]. The development time for the parallelization of the code took two semesters. Unfortunately, the coding did not pass the testing

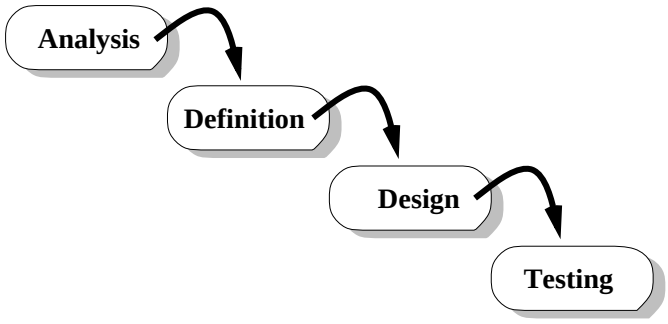


Figure 2.1: Software engineering phases

phase due to programming errors. In order not to loose the time already spend in this project the initial task included to eliminate the errors and generate benchmarking results.
While trying to debug the code using the DECmpp it was quickly obvious that it is difficult to use this machine for the testing (debugging) phase. Problems due to the choice of the system and due to lack of documentation of the parallelized code occurred.

System Oriented Problems

1. A complete compilation of the 69000 lines of code took 2 hours and 30 minutes. Changing only one file and generating the executable took inbetween 15 to 30 minutes.

2. Including routines for printing variable values at different times during the computation caused runtime errors due to memory limitations of the system.

Software Oriented Problems After carefully analysis of the parallel code the following problems became obvious:

- One third of the original code has not been parallelized. The missing code handles the boundary conditions and is the most complicated one to

parallelize. The missing calls resulted in floating point errors.

- In order to avoid the floating point errors a false initialization has been introduced in the code. Finding this particular line in the 69000 lines of code could only be done with the help of a line by line comparison between the parallel and the sequential code. The comparison took almost a week.
- After parallelizing the remaining code it stated out that the program computed the wrong results due to the elimination of some parameters used in procedure calls of the original program. The correction of the code would effect many procedure calls.

At this time it became clear that the best approach would be to start the parallelization completely new and disregard all work spend on this project before. The motivation for this being:

1. The original sequential code is working and available without modifications.
2. A new version of the sequential software has been available already since half a year.

Since there was only limited man power available a new parallelization strategy was necessary. The analysis of the sequential code showed that

- the procedures are similar in their structure,
- variable names are used consistently,
- common variables have been avoided,
- the main computation is done in do loops with few data dependencies.

These properties show that in a parallelized code most of the compiler directives and parallelizable structures will be similar. To make use of this fact the development of an interactive parallelization tool became obvious. This tool should eliminate the time spend while rewriting similar parts of the code over and over again.
In addition a version using High Performance Fortran (HPF) should be generated. In order to do this a common set of directives between the different Fortran 90 variants are specified.

2.1. ANALYSIS OF THE PARALLEL ARPS CODE (VERSION 3.0)

2.2 Parallelization of the ARPS Code (Version 3.1)

In this section the analysis of the sequential ARPS code (version 3.1) and resulting consequences for a parallelization of this code are presented.
The ARPS 3.1 code is quite large with about 69000 lines of code divided in about 150 procedures. The Figure 2.2 shows the main procedures in the calling tree. The complete calling tree is given in the appendix.

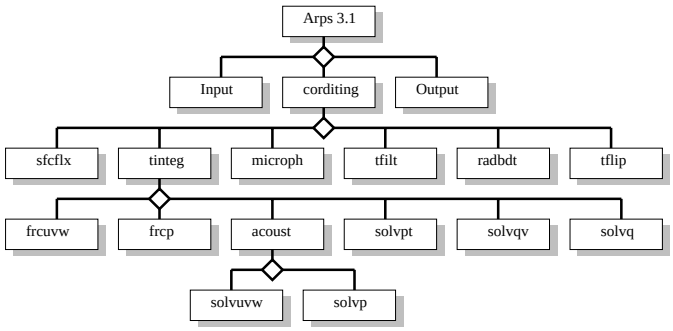


Figure 2.2: Partial program tree of the ARPS 3.1 code

A program profile of the sequential code on a SUN SPARC workstation gives valuable insight in the behavior of the code. For the profile a prediction time of 6.00 seconds is used, while choosing a domain size of $32 \times 32 \times 32$. The most time consuming routines are listed in the Tables 2.1 and 2.2. The total time to generate the profile is 425 seconds while the optimized run takes 27 seconds on the same workstation. The time listed under mcount (50%) is used for the bookkeeping of the profile. It is obvious by the 124 million calls to the multiplication that a fast multiplication routine is necessary in order to execute the program efficiently. It is important to vectorize the constructs including the multiplications efficiently in order to gain high performance. From the subroutines the routine *uvwrho* has the longest execution time. This routine should be parallelized more carefully. The other routines are more or less equal important.

Since the code is so big input and output routines are not parallelized[1].
The important data structures used in the ARPS code are one, two, three, and four dimensional arrays (Figure 2.3). The model domain is essentially a volume in a three dimensional geometrical domain. The fourth dimension is used to store values at different times. Three time points are considered, namely: *past*, *present*, and *future*. The variables important for the domain decomposition are listed in Figure 2.4 and 2.5. These variables must be distributed onto the parallel machine. For a detailed explanation of the semantic of these variables we refer to the ARPS manual [**?**]. As mentioned before most array elements neighboring each other in the index space are data dependent to each other. This means that for an update of the array element $a(i,j,k)$ the 6 neighboring elements $a(i \pm 1, j, k)$, $a(i, j \pm 1, k)$, and $a(i, j, k \pm 1)$ are necessary. Because of the operator model all elements $a(i,j,k)$ can be calculated at the same time. For certain boundary conditions data dependencies between opposite boundary elements exist resulting in more complex data dependencies.

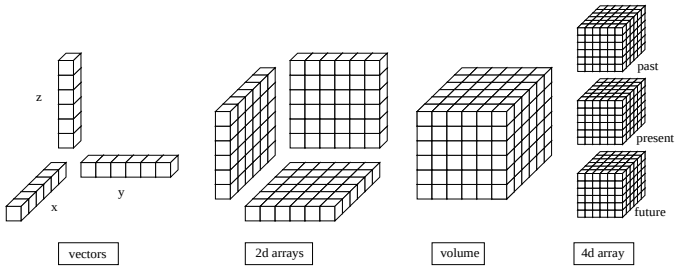


Figure 2.3: Main data structures used in the ARPS code in respect to the three dimensional model domain

In order to enable the integration of compiler directives for the CM5, the DECmpp and HPF the parallelization of the code is done in four steps:

- Data distribution
- Loop vectorization

[1] At present, the input routine has been parallelized to about 90%

- Array segmentation
- Data layout directives

Data Distribution First, for each variable a special directive is assigned telling whether the array is a parallel vector, a parallel array, a parallel volume, or a parallel four dimensional array. The directives listed below are used to indicate the property of the arrays. The comments behind a particular directive indicate the semantic of the directive as used in the code. The programmers task is to associate each variable to one of these classes. Is the variable not used in parallel no association is necessary.

`PU_Array(name)`	*A simple array*
`PU_2dE(name)`	*A 2d array at the east boundary*
`PU_2dW(name)`	*A 2d array at the west boundary*
`PU_2dS(name)`	*A 2d array at the south boundary*
`PU_2dN(name)`	*A 2d array at the north boundary*
`PU_2dSurface(name)`	*A 2d array at the surface*
`PU_Volume(name)`	*A 3d array representing the model domain*
`PU_4d(name)`	*A time dependent 3d array*

This directives will be substituted by the particular machine or language directive.

Loop Vectorization Second, the loops used in the subroutines should be vectorized. The following example demonstrates this issue. Let the original code fragment look like:

```
do i=1,n
   a(i) = b(i) + 1
end do
```

Than the modified code is:

```
a(1:n) = b(1:n) + 1
```

Array Segmentation Third, for particular versions of data parallel Fortran it is not possible to use pointers to indicate the beginning of a array segment passed to subroutines. They have to be substituted by their particular array segments. The following example illustrates the work to be done. Let the original code fragment look like:

```
real a    (nx,ny,nz,nt)
integer time
...
call routine (a(1,1,1,time),nx,ny,nz)
...
```

Than the modified code is:

```
real a    (nx,ny,nz,nt)
...
call routine (a(:,:,:,time),nx,ny,nz)
...
```

Using different machines can complicate this issue even more. While the above construct is allowed on the CM5 it is not possible on the DECmpp. Using the DECmpp as a target machine one has to copy first the array segment into a temporary array. This temporary array is than passed as parameter to the procedure. Another way to handle this problem would be to use common variables, but this contradicts the design principle of the modularity used for the ARPS code. The example shown bellow explains which additional lines of code have to be included for the parallelization on the DECmpp. These additional statements in the ARPS code lead to about 10% efficiency loss. Let the original code fragment look like:

```
real a    (nx,ny,nz,nt)
integer time
...
call routine (a(1,1,1,time),nx,ny,nz)
...
```

Than the modified code is :

```
real a    (nx,ny,nz,nt)
real temp (nx,ny,nz)
...
temp = a(:,:,:,time)
call routine (temp,nx,ny,nz)
a(:,:,:,time) = temp
...
```

Data layout directives Once all the tasks mentioned above are completed the data layout directives have to be substituted by valid compiler directives suitable for the target machine of choice. Here possible layout directive substitutions for the DECmpp and the CM5 are given. The directive substitutions for HPF are introduced later since they need a more detailed discussion (Chapter 4).

CM5 data layout directives In the LAYOUT directive of the CM5 *:news* means that the particular array axis is mapped on the machine in such a way that computations in this direction are executed in parallel. The directive *:serial* means that computations in this direction is done serially.

```
PU_Array(name)     ← CMF$  LAYOUT name (:news)
PU_2dE(name)       ← CMF$  LAYOUT name (:news,:news)
PU_2dW(name)       ← CMF$  LAYOUT name (:news,:news)
PU_2dS(name)       ← CMF$  LAYOUT name (:news,:news)
PU_2dN(name)       ← CMF$  LAYOUT name (:news,:news)
PU_2dSurface(name) ← CMF$  LAYOUT name (:news,:news)
PU_Volume(name)    ← CMF$  LAYOUT name (:news,:news,:news)
PU_4d(name)        ← CMF$  LAYOUT name (:news,:news,:news,:serial)
```

DECmpp data layout directives Here ONDPU means that the arrays are mapped in the from compiler suggested way onto the parallel processors. Since the DECmpp is anyway not the main target machine for this study we leave a more efficient mapping for future activities.

```
PU_Array(name)     ← ONDPU name
PU_2dE(name)       ← ONDPU name
PU_2dW(name)       ← ONDPU name
PU_2dS(name)       ← ONDPU name
PU_2dN(name)       ← ONDPU name
PU_2dSurface(name) ← ONDPU name
PU_Volume(name)    ← ONDPU name
PU_4d(name)        ← ONDPU name
```

2.3 Summary

We summarize the most important issues between the two target machines of choice.

DECmpp

- There exists a 4d to 3d transformation problem while passing arrays. The compiler is not able to recognize if pointers instead of array segments are passed to a procedure.
- The compilation takes a long time. At this time it is not clear if it is a problem caused by the system or the compiler.
- Using the print statement while doing extensive debugging leads to an out of memory error message.

CM5

- The compilation time is much faster than on the DECmpp.
- Printing variable values during extensive debugging is possible.
- The order of the parallel array is essential for a high performance. The serial layout should be to the far left of an array. Nevertheless, in the way the program is written it is to the far right. This indicates a lack of a compiler directive in CMF where the order of the array index can be permuted arbitrarily.
- Double precision is about 15% faster than single precision.
- The system at NPAC is heavily loaded so that it might happen that the program does not execute successfully since the memory is too small for the applications executed in timesharing mode.
- Benchmarking is only possible when the machine is empty. But there exists no time slot when the machine runs in single user or in batch mode.

Limited Resources

- Semiautomatic translation is necessary in order to outcome the limited man power.

```
c     TIME DEPENDENT VARIABLES
      real, array(nx,ny,nz,nt) u,v,w    !  Total velocities in m/s
      real, array(nx,ny,nz,nt) ptprt    !  Perturbation potential temperature
                                        !  From that of base state atmosphere (K)
      real, array(nx,ny,nz,nt) pprt     !  Perturbation pressure from that
                                        !  Of base state atmosphere (Pascal)
      real, array(nx,ny,nz,nt) qv       !  Water vapor specific humidity
      real, array(nx,ny,nz,nt) qc       !  Cloud water mixing ratio
      real, array(nx,ny,nz,nt) qr       !  Rain water mixing ratio
      real, array(nx,ny,nz,nt) qi       !  Cloud ice mixing ratio
      real, array(nx,ny,nz,nt) qs       !  Snow mixing ratio
      real, array(nx,ny,nz,nt) qh       !  Hail mixing ratio
      real, array(nx,ny,nz) km          !  The turbulent mixing coefficient for
                                              !  momentum m^2/s
```

```
c     BASE STATE VARIABLES
      real, array(nx,ny,nz) ubar     !  Base state u-velocity m/s
      real, array(nx,ny,nz) vbar     !  Base state v-velocity m/s
      real, array(nx,ny,nz) ptbar    !  Base state potential temperature (K)
      real, array(nx,ny,nz) pbar     !  Base state pressure (Pascal).
      real, array(nx,ny,nz) rhobar   !  Base state air density kg/m^3
      real, array(nx,ny,nz) qvbar    !  Base state water vapor specific humidity
```

```
c     ARRAYS RELATED TO MODEL GRID DEFINITION
      real, array(nx) x           !  The coordinates of the physical and
      real, array(ny) y           !  computational grid.
      real, array(nz) z
      real, array(nx,ny,nz) zp    !  The physical height coordinate defined at
                                  !  w-point of the staggered grid.
      real, array(nx,ny) hterain  !  The height of the terrain.
      real, array(nx,ny,nz) j1    !  Coordinate transform Jacobian defined as -d zp/d x
      real, array(nx,ny,nz) j2    !  Coordinate transform Jacobian defined as -d zp/d y
      real, array(nx,ny,nz) j3    !  Coordinate transform Jacobian defined as -d zp/d z
```

```
c     PURE WORK ARRAYS THAT DO NOT CARRY PHYSICAL MEANING IN THE
CODE
      real, array(nx,ny) temxy_   !  2-D temporary array
                                  !  where _ = 1,2,3,4
      real, array(nx,ny,nz) tem_  !  Temporary work array.
                                  !  where _ = 1,2,3,4,5,6,7,8,9,10,11,12,13
```

Figure 2.4: Most of the domain variables on which calculations are performed

`real, array(ny,nz) pdteb`	!	*T-tendency of pprt at e-boundary* $\frac{Pascal}{s}$
`real, array(ny,nz) ptdteb`	!	*T-tendency of ptprt at e-boundary* $\frac{K}{s}$
`real, array(ny,nz) qcdteb`	!	*T-tendency of qc at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) qhdteb`	!	*T-tendency of qh at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) qidteb`	!	*T-tendency of qi at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) qrdteb`	!	*T-tendency of qr at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) qsdteb`	!	*T-tendency of qs at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) qvdteb`	!	*T-tendency of qv at e-boundary* $\frac{1}{s}$
`real, array(ny,nz) udteb`	!	*T-tendency of u at e-boundary* $\frac{m}{s^2}$
`real, array(ny,nz) vdteb`	!	*T-tendency of v at e-boundary* $\frac{m}{s^2}$
`real, array(ny,nz) wdteb`	!	*T-tendency of w at e-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) pdtnb`	!	*T-tendency of pprt at n-boundary* $\frac{Pascal}{s}$
`real, array(nx,nz) ptdtnb`	!	*T-tendency of ptprt at n-boundary* (K/s)
`real, array(nx,nz) qcdtnb`	!	*T-tendency of qc at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) qhdtnb`	!	*T-tendency of qh at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) qidtnb`	!	*T-tendency of qi at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) qrdtnb`	!	*T-tendency of qr at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) qsdtnb`	!	*T-tendency of qs at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) qvdtnb`	!	*T-tendency of qv at n-boundary* $\frac{1}{s}$
`real, array(nx,nz) udtnb`	!	*T-tendency of u at n-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) vdtnb`	!	*T-tendency of v at n-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) wdtnb`	!	*T-tendency of w at n-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) pdtsb`	!	*T-tendency of pprt at s-boundary* $\frac{Pascal}{s}$
`real, array(nx,nz) ptdtsb`	!	*T-tendency of ptprt at s-boundary* $\frac{K}{s}$
`real, array(nx,nz) qcdtsb`	!	*T-tendency of qc at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) qhdtsb`	!	*T-tendency of qh at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) qidtsb`	!	*T-tendency of qi at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) qrdtsb`	!	*T-tendency of qr at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) qsdtsb`	!	*T-tendency of qs at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) qvdtsb`	!	*T-tendency of qv at s-boundary* $\frac{1}{s}$
`real, array(nx,nz) udtsb`	!	*T-tendency of u at s-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) vdtsb`	!	*T-tendency of v at s-boundary* $\frac{m}{s^2}$
`real, array(nx,nz) wdtsb`	!	*T-tendency of w at s-boundary* $\frac{m}{s^2}$
`real, array(ny,nz) pdtwb`	!	*T-tendency of pprt at w-boundary* $\frac{Pascal}{s}$
`real, array(ny,nz) ptdtwb`	!	*T-tendency of ptprt at w-boundary* $\frac{K}{s}$
`real, array(ny,nz) qcdtwb`	!	*T-tendency of qc at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) qhdtwb`	!	*T-tendency of qh at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) qidtwb`	!	*T-tendency of qi at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) qrdtwb`	!	*T-tendency of qr at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) qsdtwb`	!	*T-tendency of qs at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) qvdtwb`	!	*T-tendency of qv at w-boundary* $\frac{1}{s}$
`real, array(ny,nz) udtwb`	!	*T-tendency of u at w-boundary* $\frac{m}{s^2}$
`real, array(ny,nz) vdtwb`	!	*T-tendency of v at w-boundary* $\frac{m}{s^2}$
`real, array(ny,nz) wdtwb`	!	*T-tendency of w at w-boundary* $\frac{m}{s^2}$

Figure 2.5: The domain variables for the boundaries

Table 2.1: Sequential profile of the ARPS 3.1 code

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
50.1	213.46	213.46				mcount
13.4	270.70	57.24	124816095	0.00	0.00	.mul[1]
7.2	301.26	30.56				mul_12bit[1]
5.8	325.84	24.58				mul_8bit[1]
2.8	337.90	12.06				mul_4bit[1]
1.9	345.98	8.08	17	475.30	871.29	uvwrho
1.1	350.67	4.69	12	390.83	3367.74	solvuvw
0.9	354.53	3.86	35	110.29	203.35	difx
0.9	358.38	3.85	36	106.94	199.85	difz
0.9	362.17	3.79				zero_divide [1]
0.9	365.95	3.78	35	108.00	201.06	dify
0.9	369.71	3.76	26	144.62	231.85	avgz
0.7	372.60	2.89	29	99.66	193.23	aamult
0.6	375.18	2.58	1	2580.01	4698.80	initdvr
0.5	377.10	1.92	12	160.00	500.81	stepw
0.4	378.96	1.86				next4[1]
0.4	380.78	1.82	12	151.67	260.85	stepv
0.4	382.59	1.82				moncontrol[1]
0.4	384.40	1.81	12	150.83	882.83	divgs
0.4	386.10	1.70	51	33.33	33.33	flzero
0.4	387.73	1.63	7	232.86	382.46	difzz
0.4	389.26	1.53	8	191.25	331.09	tswap
0.4	390.78	1.52	12	126.67	235.85	stepu
0.3	392.18	1.40	12	116.67	216.33	pdivrg
0.3	393.52	1.34	12	111.67	190.91	stepp
0.3	394.85	1.33	1	1330.01	2279.92	inibase
0.3	396.15	1.30	11	118.18	209.63	avgx
0.3	397.27	1.12	1	1120.00	2407.55	revap
0.3	398.38	1.11	11	100.91	192.35	avgy
0.3	399.47	1.09	7	155.71	297.27	difxx
0.2	400.53	1.06	1	1060.00	1797.50	stress
0.2	401.58	1.05	1	1050.00	5001.36	cmix2uvw
0.2	402.60	1.02	7	145.71	287.27	difyy
0.2	403.61	1.01	5	202.00	2156.92	advcts
0.2	404.60	0.99	2	495.00	1427.86	tmixq
0.2	405.58	0.98	1	980.00	1769.05	deform
0.2	406.55	0.97	1	970.00	1595.99	buoycy
0.2	407.51	0.96	1	960.00	2065.80	satadj
0.2	408.45	0.94	4	235.00	419.91	divgsg
0.2	409.37	0.92	1	920.00	8597.91	microph
0.2	410.24	0.87	1371	0.63	0.63	write[1]
0.2	411.04	0.80	1	800.00	1315.44	qrfall
0.2	411.82	0.78				mul_16bit[1]

[1]Is a system intern call

Table 2.2: Sequential profile of the ARPS 3.1 code

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
0.2	412.60	0.78	1	780.00	1519.20	cftmix
0.2	413.36	0.76	1	760.00	1788.66	tmixpt
0.2	414.09	0.73	1	730.00	1779.45	stabnsq
0.2	414.77	0.68	1	680.00	1122.14	autocac
0.2	415.43	0.66	3	220.00	424.15	satmr
0.1	416.03	0.60	12	50.00	79.65	lbdtuvw
0.1	416.61	0.58	3	193.33	325.41	stepq
0.1	417.18	0.57	1	570.00	1502.86	tmixqv
0.1	417.70	0.52	3	173.33	1491.37	cmix2q
0.1	418.17	0.47	348511	0.00	0.00	r_pow[1]
0.1	418.59	0.42				normal[1]
0.1	418.96	0.37	1	370.00	665.09	jacob
0.1	419.32	0.36	1	360.00	26337.97	frcuvw
0.1	419.66	0.34	1	340.00	695.75	rdmpuvw
0.1	419.99	0.33				finite[1]
0.1	420.30	0.31	1	310.00	1624.28	cmix2pt
0.1	420.61	0.31	1	310.00	458.12	chkstab
0.1	420.91	0.30	12	25.00	37.93	bcw
0.1	421.19	0.28	1	280.00	56810.41	acoust
0.1	421.44	0.25	1	250.00	417.37	rdmppt
0.1	421.68	0.24	1	240.00	398.49	steppt
0.0	421.89	0.21	460	0.46	0.46	ioctl
0.0	422.09	0.20	12	16.67	24.94	bcp
0.0	422.29	0.20	12	16.67	25.12	bcu
0.0	422.47	0.18	12	15.00	23.45	bcv
0.0	422.64	0.17	1	170.00	2694.05	advv
0.0	422.80	0.16	104788	0.00	0.00	r_exp[1]
0.0	422.95	0.15	1	150.00	886.31	inigrd
0.0	423.08	0.13	1	130.00	2705.87	advu
0.0	423.21	0.13	1	130.00	2793.60	advw
0.0	423.34	0.13	1	130.00	2859.18	frcp
0.0	423.47	0.13	1	130.00	3229.11	mixqv
0.0	423.59	0.12	348511	0.00	0.00	pow_rr[1]
0.0	423.71	0.12	1	120.00	833.23	divgw
0.0	423.82	0.11	1	110.00	823.45	divgu
0.0	423.92	0.10	30805	0.00	0.02	x_putc[1]
0.0	424.02	0.10	4	25.00	36.92	bcsclr
0.0	424.11	0.09	104448	0.00	0.00	Fsqrt[1]
0.0	424.20	0.09	1	90.00	3803.27	advpt
0.0	424.29	0.09	1	90.00	803.45	divgv
0.0	424.37	0.08	3	26.67	39.82	latbdtq
0.0	424.45	0.08	1	80.00	2558.94	advp
0.0	424.53	0.08	1	80.00	97.47	bdtu
0.0	424.60	0.07	104788	0.00	0.00	Fexp[1]
0.0	424.67	0.07	1	70.00	106.87	bdtp

[1] Is a call to an internal routine

Chapter 3

Semiautomatic Translator

This chapter describes the interactive parallelization tool[3]. It is embedded in the standard editor *emacs*. The parallelization technique is customized for the ARPS code. Since the interactive shell is very easy to modify it can be used as a base for the parallelization of other applications.
The semiautomatic translator is especially from interest because of the introduction of the new Fortran standard. One of the most important features of Fortran 90 is the ability of using vector constructs instead of simple loops used in Fortran 77. As mentioned before these vector constructs enable to express parallelism in an easy straight forward way. Dataparallel compilers developed for different machines make use of this vector constructs and generate parallel code for these machines.
While transferring Fortran 77 programs to the new standard Fortran 90 one faces the problem that most real life applications consist of many thousands of Fortran lines. It would be a tedious task to handcode such large codes without the support of a software tool. In case of the ARPS code the development time could be reduced considerably while using the interactive parallelization tool. In addition we present a way to write an intermediate code which can be easily transferred to either CM Fortran, DECmpp Fortran, or High Performance Fortran.

3.1 The Parallelization Tool in Context

Figure 3.1 shows the interactive parallelization tool in context to existing hardware architectures and High Performance Fortran (HPF). First, the Fortran 77 program is changed in such a way that do loops are vectorized and the corresponding layout directives for the array distribution are added to the code. This is done by problem specific layout directives.
After including the directives one has the choice of a direct mapping to the machine or transfer the program further to Fortran 90D (HPF). The advantage of using HPF will be more obvious in future when more vendors provide this new Fortran as standard for their machines.

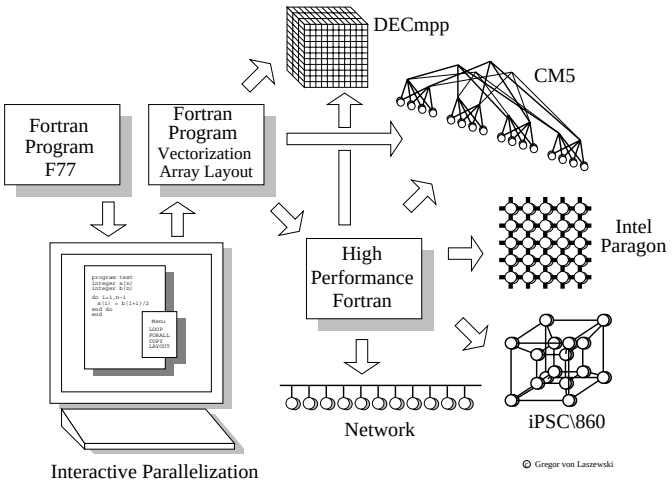


Figure 3.1: The interactive Parallelization Tool in context

We chose the ARPS code for parallelization and targeted the following languages

- CM Fortran,
- DECmpp Fortran,
- and Fortran 90D (HPF).

The parallelization of the code has been done in two independent groups. While the first group consisting of two members used 6 month for the parallelization the second group consisting of only one programmer did the work in one semester.

3.2 Menu Interactions

The editor emacs is used to invoke the interactions with the help of popup menus under X-windows. Emacs is available for a huge number of different machines ranging from mainfraims to Workstations and PC's. Because of its availability and its portability we chose emacs as the frontend to the interactive parallelization tool. One major advantage of emacs is its extendability while using keyboard macros and invoking shell scripts on specially marked program regions. In the following list we give some of the advantages which made us choosing emacs as the frontend.

- It is free.
- It is portable.
- It is available.
- Many users already use it.
- A X11 interface is available.
- A mouse interface is available.

The forthcoming sections show how the output on the monitor looks like before and after invoking a particular mouse action. We use a simple example program from the ARPS code which is shown in the first Figure.

The Menu

The menu is invoked while pressing the

CTRL key and the right mouse button

at the same time. A menu pops up as shown in the next Figure. Each item in the menu invokes a program on a marked region. The beginning of a region is marked with the help of

CTRL - space

The end of the region is marked by the actual cursor position. In the Figures the begin of the region is displayed with the help of the @ character and the end with ▮. The menu contains

- The simple copying of a program part.
- Transforming a loop to a Forall loop.
- Transforming a loop to a vector construct.
- Determining the variable layout.

These tools are sufficient to parallelize the ARPS code.

```
    SUBROUTINE AAMULT(a, b, nx, ny, nz,
                      ibgn, iend, jbgn, jend, kbgn, kend, ab)
    implicit none
    integer nx, ny, nz        ! Number of grid points in 3 directions
    real a (nx, ny, nz)         ! Input array 1
    real b (nx, ny, nz)         ! Input array 2
    integer ibgn, iend, jbgn, jend, kbgn, kend
    real ab(nx, ny, nz)         ! Product array
    integer i, j, k

    DO 100 k = kbgn, kend
    DO 100 j = jbgn, jend
    DO 100 i = ibgn, iend
      ab(i, j, k) = a(i, j, k) * b(i, j, k)
100   CONTINUE
    END
```

Emacs: SimpleFortranProgram.f


```
    SUBROUTINE AAMULT(a, b, nx, ny, nz,
                      ibgn, iend, jbgn, jend, kbgn, kend, ab)
    implicit none
    integer nx, ny, nz        ! Number of grid points in 3 directions
    real a (nx, ny, nz)         ! Input array 1
    real b (nx, ny, nz)         ! Input array 2
    integer ibgn, iend, jbgn, jend, kbgn, kend
    real ab(nx, ny, nz)         ! Product array
    integer i, j, k

    DO 100 k = kbgn, kend
    DO 100 j = jbgn, jend
    DO 100 i = ibgn, iend
      ab(i, j, k) = a(i, j, k) * b(i, j, k)
100   CONTINUE
    END
```


Menu
Simple Loop Translation
Forall Loop Translation
Copy Text Segment
F90 Variable Layout

Emacs: SimpleFortranProgram.f

3.3 Variable Layout

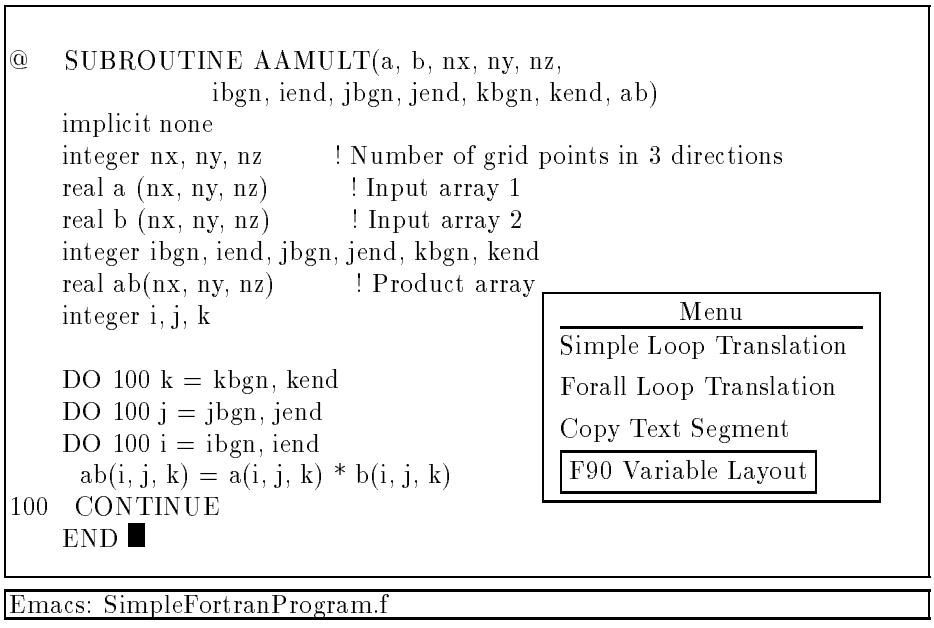
One of the most important tasks of the parallelization is to determine the variable layout. For our example we realize that the domain is a three dimensional array representing a volume. Therefore, we design a layout macro for volumes. This macro will be later on substituted with an appropriate compiler directives. As example we use here the CM5. The Volume directive would be translated into the following compiler directive:

CMF$ LAYOUT name (:news,:news,:news)

Note also that some comment lines will be inserted into the program. These comment lines make it easier to search for parallel variables. This is especially useful if the variables should not be laid out in parallel. We follow the strategy:

Deleting is easier and less time consuming than inserting.

In case a variable should be serial the inserted layout statements can be deleted. Let the following Figure represent the state before invoking the *Variable Layout* command:



The next Figure represent the state after invoking the command:

```
      SUBROUTINE AAMULT(a, b, nx, ny, nz,
                        ibgn, iend, jbgn, jend, kbgn, kend, ab)
      implicit none
      integer nx, ny, nz             ! Number of grid points in 3 directions
      real a (nx, ny, nz)            ! Input array 1
cgvl <real> <a> <nx, ny, nz> ! Parameters : 3
PU_Volume(a)
      real b (nx, ny, nz)            ! Input array 2
cgvl <real> <b> <nx, ny, nz> ! Parameters : 3
PU_Volume(b)
      integer ibgn, iend, jbgn, jend, kbgn, kend
      real ab(nx, ny, nz)            ! Product array
cgvl <real> <ab> <nx, ny, nz> ! Parameters : 3
PU_Volume(ab)
      integer i, j, k
      DO 100 k = kbgn, kend
      DO 100 j = jbgn, jend
      DO 100 i = ibgn, iend
        ab(i, j, k) = a(i, j, k) * b(i, j, k)
100   CONTINUE
      RETURN
      END
```

Emacs: SimpleFortranProgram.f

3.4 Simple Loop Translation

Many programs use simple loops with no data dependency between the left and right hand side. This enables one to substitute the loop with a vector construct available in Fortran 90. The loop translation consists the following phases:

1. Detect how many loops are nested in each other.
2. Detect the loop indices and their ranges.
3. Substitute in each line of the loop block the loop indices with the correct range values.

Let the following Figure represent the state before invoking the *Simple Loop Translation* command:

```
    SUBROUTINE AAMULT(a, b, nx, ny, nz,
                      ibgn, iend, jbgn, jend, kbgn, kend, ab)
    implicit none
    integer nx, ny, nz        ! Number of grid points in 3 directions
    real a (nx, ny, nz)        ! Input array 1
    real b (nx, ny, nz)        ! Input array 2
    integer ibgn, iend, jbgn, jend, kbgn, kend
    real ab(nx, ny, nz)        ! Product array
    integer i, j, k

@   DO 100 k = kbgn, kend
    DO 100 j = jbgn, jend
    DO 100 i = ibgn, iend
      ab(i, j, k) = a(i, j, k) * b(i, j, k)
100 CONTINUE
    END
```

```
        Menu
Simple Loop Translation
Forall Loop Translation
Copy Text Segment
F90 Variable Layout
```

Emacs: SimpleFortranProgram.f

The next Figure represent the state after invoking the command:

```
      SUBROUTINE AAMULT(a, b, nx, ny, nz,
                        ibgn, iend, jbgn, jend, kbgn, kend, ab)
      implicit none
      integer nx, ny, nz        ! Number of grid points in 3 directions
      real a (nx, ny, nz)         ! Input array 1
      real b (nx, ny, nz)         ! Input array 2
      integer ibgn, iend, jbgn, jend, kbgn, kend
      real ab(nx, ny, nz)         ! Product array
      integer i, j, k
c.f77
c       DO 100 k = kbgn, kend
c       DO 100 j = jbgn, jend
c       DO 100 i = ibgn, iend
c         ab(i, j, k)  =  a(i, j, k) * b(i, j, k)
c100    CONTINUE
c.f77
c.f90
        ab(ibgn:iend, jbgn:jend, kbgn:kend)  =
     :      a(ibgn:iend, jbgn:jend, kbgn:kend)
     :    * b(ibgn:iend, jbgn:jend, kbgn:kend)
c.f90
      RETURN
      END
```

Emacs: SimpleFortranProgram.f

To make the statements included by the translator more transparent the old statements are commented out in the program and are located between the comment

c.77

The new code for this line is surrounded by the comment lines

c.90

3.5 Forall Loop Translation

Often a simple loop translation can not be applied because the loop indices are incremented by a constant. Two possibilities exits to correct this problem:

1. modify the index boundaries for the loop variable
2. use the more powerful FORALL construct.

Here the FORALL translation is chosen. It contains the following phases:

1. Detect how many loops are nested in each other.
2. Detect the loop indices and their range.
3. Construct the FORALL statement with ranges.
4. Write each line of the loop block with the appropriate FORALL statement and range information.

Let the following Figure represent the state before invoking the *FORALL Loop* command:

```
      SUBROUTINE AAMULT(a, b, nx, ny, nz,
                        ibgn, iend, jbgn, jend, kbgn, kend, ab)
      implicit none
      integer nx, ny, nz        ! Number of grid points in 3 directions
      real a (nx, ny, nz)         ! Input array 1
      real b (nx, ny, nz)         ! Input array 2
      integer ibgn, iend, jbgn, jend, kbgn, kend
      real ab(nx, ny, nz)         ! Product array
      integer i, j, k

@     DO 100 k = kbgn, kend
      DO 100 j = jbgn, jend
      DO 100 i = ibgn, iend
        ab(i, j, k) = a(i+1, j, k) * b(i, j-1, k)
100   CONTINUE
      END
```

```
        Menu
Simple Loop Translation
Forall Loop Translation
Copy Text Segment
F90 Variable Layout
```

Emacs: SimpleFortranProgram.f

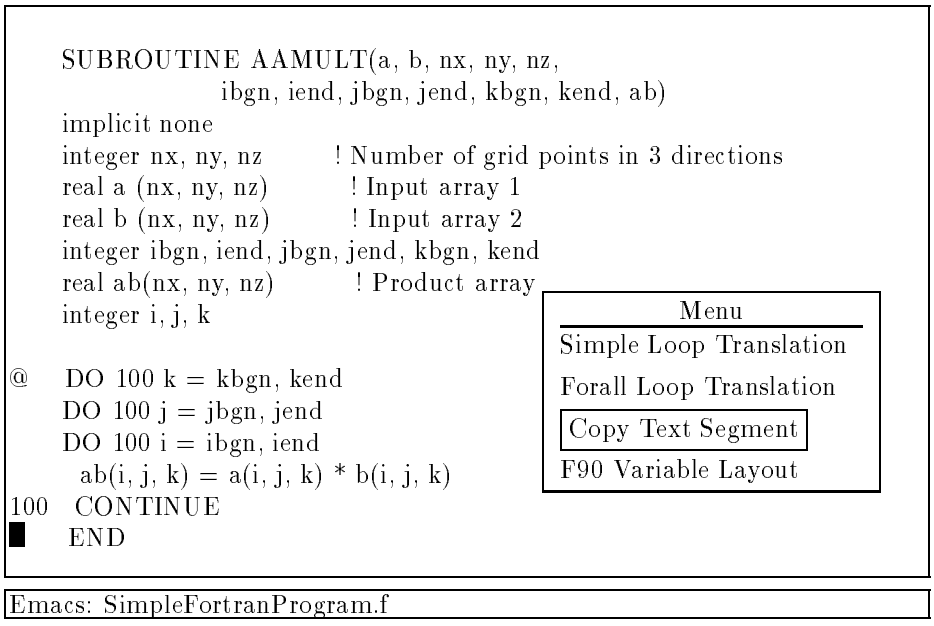
The next Figure represent the state after invoking the command:

```
      SUBROUTINE AAMULT(a, b, nx, ny, nz,
                        ibgn, iend, jbgn, jend, kbgn, kend, ab)
      implicit none
      integer nx, ny, nz        ! Number of grid points in 3 directions
      real a (nx, ny, nz)         ! Input array 1
      real b (nx, ny, nz)         ! Input array 2
      integer ibgn, iend, jbgn, jend, kbgn, kend
      real ab(nx, ny, nz)         ! Product array
      integer i, j, k
c.f77
c       DO 100 k = kbgn, kend
c       DO 100 j = jbgn, jend
c       DO 100 i = ibgn, iend
c         ab(i, j, k)  =  a(i+1, j, k) * b(i, j-1, k)
c100    CONTINUE
c.f77
c.f90
       FORALL (k = kbgn:kend, j = jbgn:jend, i = ibgn:iend)
      :   ab(i, j, k)  =  a(i+1, j, k) * b(i-1, j, k)
c.f90
      RETURN
      END
```

Emacs: SimpleFortranProgram.f

3.6 Duplicating Text Segments

Sometimes more complicated parallelization methods have to be used. in this case it is useful to copy particular parts of the program and modify them by hand. Let the following Figure represent the state before invoking the *Copying* command:



The next Figure represent the state after invoking the command:

```
      SUBROUTINE AAMULT(a, b, nx, ny, nz,
     *                  ibgn, iend, jbgn, jend, kbgn, kend, ab)
      implicit none
      integer nx, ny, nz        ! Number of grid points in 3 directions
      real a (nx, ny, nz)         ! Input array 1
      real b (nx, ny, nz)         ! Input array 2
      integer ibgn, iend, jbgn, jend, kbgn, kend
      real ab(nx, ny, nz)         ! Product array
      integer i, j, k
c.f77
c     DO 100 k = kbgn, kend
c     DO 100 j = jbgn, jend
c     DO 100 i = ibgn, iend
c       ab(i, j, k) = a(i, j, k) * b(i, j, k)
c100  CONTINUE
c.f77
c.f90
      DO 100 k = kbgn, kend
      DO 100 j = jbgn, jend
      DO 100 i = ibgn, iend
        ab(i, j, k) = a(i, j, k) * b(i, j, k)
100   CONTINUE
c.f90
      RETURN
      END
```

Emacs: SimpleFortranProgram.f

Chapter 4

ARPS and HPF

This chapter shows how to implement the Advanced Regional Storm Prediction System (ARPS) in Fortran 90D/HPF[4]. The modification from Fortran 90 to HPF can be done very quickly because of findings from the program analysis described in Chapter 2. First, we will introduce the Fortran 90D constructs necessary for the implementation of ARPS. Than we show in detail what we have to change in the Fortran 90 program. Using HPF has the advantage that the program will be portable on many machines. Many vendors are currently developing HPF compilers for their computers.

4.1 Fortran90D

Many scientific applications use multi dimensional arrays on which computations are performed. To write efficient parallel programs while using message passing the data decomposition is an important issue. To obtain speedup via parallel execution it is necessary to decompose the array and perform the computations in parallel on separate processing units.
First, one has to decide how the arrays should be aligned with respect to each other, both within and across the array dimensions. The purpose of the alignment is to reduce the data movement while performing the calculation. The alignment is basically independent of the actual machine layout. Second, these arrays are distributed onto the machine. This includes issues dealing with min-

imizing the data movement while maintaining load balance. The Figure 4.1 describes the strategy used in specifying Fortran90D programs:

1. Find the structure of the problem domain used in the original program.
2. Define an appropriate template for the data alignment. The template can be viewed as logical data or memory space.
3. Define an appropriate distribution of the template to a logical processor space.
4. A mapping to the specified hardware architecture is created by the compiler.

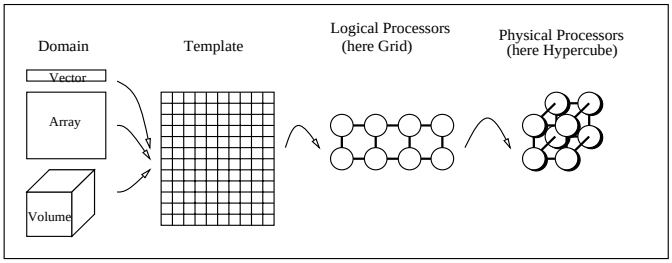


Figure 4.1: Mapping the Domain onto the Processing Elements

4.1.1 The Language Directives

We explain the Fortran D directives important for the implementation for the ARPS code with the help of a small example where two arrays with different array indices are added and stored in a third array:

```
REAL, ARRAY (N): A, B, AB
DO i=1,N-1
   AB(i) = A(i+1) + B(i)
END DO
```

Figure 4.2 shows a possible decomposition and its alignment which is advantages for a parallel execution. Here `M` specifies the number of processors used for the parallel execution. Inspecting the data dependencies between the arrays `AB`, `A`, and `B` we see that these arrays can be accessed in arbitrary order. This enables one to use a `FORALL` loop. The complete Fortran 90D code fragment is shown bellow:

```
REAL, ARRAY(N) A,B
TEMPLATE vector(N)
DISTRIBUTE vector(BLOCK(M))
ALIGN A(i+1) with vector(i)
ALIGN B(i) with vector(i)
FORALL i=1,N-1
   AB(i) = A(i+1) + B(i)
END FORALL
```

The `DISTRIBUTE` statement specifies in which way the data is distributed onto the templates. For the ARPS code we need only the BLOCK distribution. That means that the vector is divided into chunks of N/M and than each is mapped onto one processor. For other choices we refer to [2].

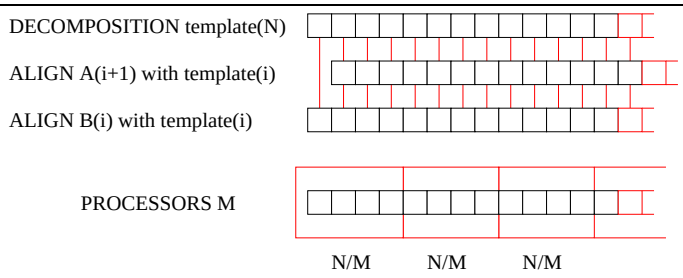


Figure 4.2: Example of a decomposition and an alignment

In addition to the distribution directives, parallel *forall* loops, *independent* loops, and *reduction* operators are supported. The Figure 4.3 explains the different semantic of a forall and independent loop. In a forall loop each statement is parallelized separately over the given index set. In an independent loop the statements in the loop are considered as a block and are executed as a block in

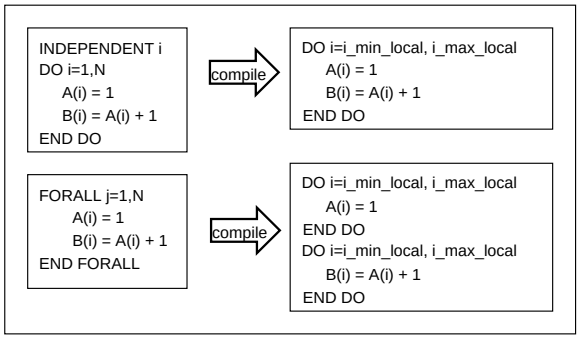


Figure 4.3: Example of a forall and independent loop

parallel over the given index set. A more detailed description of the language extension can be found in [1, 2].

4.1.2 The Compiler Phases

Figure 4.4 shows the outline of the Fortran90D/HPF compiler. In the first phase of the compilation a parse tree is generated from a syntactically correct Fortran90D/HPF program. In addition *array assignments* and *where* constructs are transformed into equivalent *forall* statements such that the following phases of the compiler deal only with forall constructs.
In the *partitioning phase* the data distribution directives *decomposition, distribute* and *align* are recognized and the appropriate code is generated for the data partitioning and their distribution onto the processors. In the *sequentialization phase* all parallel constructs in the node program are sequentialized since they can be efficiently executed on a single processor. In case communication from one processor to another is necessary this code is inserted during the *communication detection phase* of the compilation. The final code is generated in the last phase of the compilation. following the Single Program Multiple Data (SPMD) paradigm. The program is naturally divided into code fragments containing local computation, where operations of a processor are performed on data stored in its own local memory, and global communication, where data is

transferred among processors.

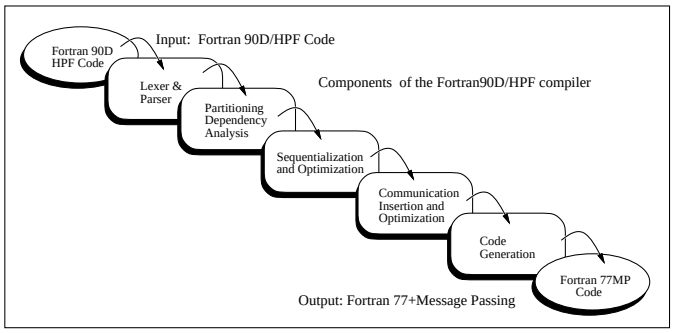


Figure 4.4: The main components of the Fortran90D compiler

4.1.3 Templates for the ARPS Code

In this section we specify the basic code needed for a Fortran D implementation. As mentioned before the variables in the ARPS code consist of one, two, three, and four dimensional arrays. The model domain is essentially a volume in a three dimensional geometrical domain.

We assume there are procs_x, procs_y, procs_z processors in x, y, and z direction. The size of the model domain is specified by nx, ny and nz. Naturally we divide arrays with a dimension in x direction in procs_x blocks. In the same way it is done for the y and z direction. This results in a total of procs_x $\times$ procs_y $\times$ procs_z processors. Thus we define the templates for the ARPS code as shown in Figure 2.3. The following code fragment specifies the templates used in the Fortran D code of ARPS.

```
INTEGER procs_x, procs_y, procs_z
INTEGER nx, ny, nz
TEMPLATE Volume(nx,ny,nz)
TEMPLATE x-vector(nx)
TEMPLATE y-vector(ny)
TEMPLATE z-vector(nz)
TEMPLATE xy-boundary(nx,ny)
```

Templates for the ARPS Code

```
TEMPLATE xz-boundary(nx,nz)
TEMPLATE yz-boundary(ny,nz)
```

Now it is easy to distribute these templates on the processors by using the appropriate number of processors in a particular direction of the model domain. We choose a block distribution because the algorithms are based on nearest neighbor communication in the three dimensional space (See the Program Analysis in Chapter 2).

```
DISTRIBUTE Volume      (BLOCK(procs_x), BLOCK(procs_y), BLOCK(procs_z))
DISTRIBUTE x-vector    (BLOCK(procs_x), *             , *              )
DISTRIBUTE y-vector    (*             , BLOCK(procs_y), *              )
DISTRIBUTE z-vector    (*             , *             , BLOCK(procs_z))
DISTRIBUTE xy-boundary (BLOCK(procs_x), BLOCK(procs_y), *              )
DISTRIBUTE xz-boundary (BLOCK(procs_x), *             , BLOCK(procs_z))
DISTRIBUTE yz-boundary (*             , BLOCK(procs_y), BLOCK(procs_z))
```

Now the four dimensional arrays can be laid out in the following way:

```
REAL, ARRAY(nx,ny,nz,nt) u
ALIGN u with Volume        ! Array collapse in the fourth dimension
```

Note that there exists an array collapse in the fourth dimension. That means The elements $a(i,j,k,t)$ with $t \in \{present, past, future\}$ are mapped onto the same template position. The three dimensional arrays are mapped in the following way onto the template

```
REAL, ARRAY(nx,ny,nz) xyz
ALIGN xyz with Volume
```

Two dimensional arrays are mapped according to their dimensional specification

```
REAL, ARRAY(nx,ny) xy
REAL, ARRAY(nx,nz) xz
REAL, ARRAY(ny,nz) yz
ALIGN xy with xy-vector
ALIGN xz with xz-vector
ALIGN yz with yz-vector
```

Vectors with dimension nx, ny, or nz are mapped as follows

```
REAL, ARRAY(nx) x
```

```
REAL, ARRAY(ny) y
REAL, ARRAY(nx) z
ALIGN x with x-vector
ALIGN y with y-vector
ALIGN z with z-vector
```

With this scheme the variables shown in Figure 2.4 and 2.5 are distributed with the help of the code fragment shown in Figure 4.5 and 4.6 onto a parallel machine.

4.2 Summary

This chapter shows how to use the Fortran D directives to implement the ARPS code for the Fortran 90D/HPF language. Since HPF is expected to be soon standard on a variety of massively parallel machines the ARPS code is already ported to these machines.
For each dimension of the physical model domain a particular number of processors is used. Therefore, the code can be easily modified for different machine sizes. Furthermore, one can explore in future experiments the influence of the memory hierarchy on the speed of the data excess while modifying the number of processors in a particular dimension. This issue is in detail explained in [5].
We could show that a high level language support tool for MIMD machines as introduced in [5] is not necessary. The implementation of the ARPS code with Fortran 90D showed that the language consist of enough expressionistic power to implement the problem.

```
c     TIME DEPENDENT VARIABLES
      ALIGN u,v,w  with Volume !  Total velocities in m/s
      ALIGN ptprt  with Volume !  Perturbation potential temperature
                               !  From that of base state atmosphere
      ALIGN pprt with Volume   !  Perturbation pressure from that
                               !  Of base state atmosphere
      ALIGN qv with Volume     !  Water vapor specific humidity
      ALIGN qc     with Volume !  Cloud water mixing ratio
      ALIGN qr     with Volume !  Rain water mixing ratio
      ALIGN qi     with Volume !  Cloud ice mixing ratio
      ALIGN qs     with Volume !  Snow mixing ratio
      ALIGN qh     with Volume !  Hail mixing ratio
      ALIGN km        with Volume !  The turbulent mixing coefficient for
                                  !  momentum m^2/s
```

```
c     BASE STATE VARIABLES
      ALIGN ubar   with Volume !  Base state u-velocity m/s
      ALIGN vbar   with Volume !  Base state v-velocity m/s
      ALIGN ptbar  with Volume !  Base state potential temperature (K)
      ALIGN pbar   with Volume !  Base state pressure (Pascal).
      ALIGN rhobar with Volume !  Base state air density kg/m^3
      ALIGN qvbar  with Volume !  Base state water vapor specific humidity
```

```
c     ARRAYS RELATED TO MODEL GRID DEFINITION
      ALIGN x with x-vector
      ALIGN y with y-vector
      ALIGN z with z-vector
      ALIGN zp with Volume !  The physical height coordinate defined at
                           !  w-point of the staggered grid.
      ALIGN hterain with xy-boundary !  The height of the terrain.
      ALIGN j1 with Volume !  Coordinate transform Jacobian defined as -δzp/δx
      ALIGN j2 with Volume !  Coordinate transform Jacobian defined as -δzp/δy
      ALIGN j3 with Volume !  Coordinate transform Jacobian defined as -δzp/δz
```

```
c     PURE WORK ARRAYS THAT DO NOT CARRY PHYSICAL MEANING IN THE
CODE
      ALIGN temxy_  with xy-boundary !  2-D temporary array
                                 !  where _ = 1,2,3,4
      ALIGN tem_with Volume !  Temporary work array.
                                 !  where _ = 1,2,3,4,5,6,7,8,9,10,11,12,13
```

Figure 4.5: Most of the domain variables on which calculations are performed

ALIGN pdteb	with yz-boundary !	*T-tendency of pprt at e-boundary* $\frac{Pascal}{s}$
ALIGN ptdteb	with yz-boundary !	*T-tendency of ptprt at e-boundary* $\frac{K}{s}$
ALIGN qcdteb	with yz-boundary !	*T-tendency of qc at e-boundary* $\frac{1}{s}$
ALIGN qhdteb	with yz-boundary !	*T-tendency of qh at e-boundary* $\frac{1}{s}$
ALIGN qidteb	with yz-boundary !	*T-tendency of qi at e-boundary* $\frac{1}{s}$
ALIGN qrdteb	with yz-boundary !	*T-tendency of qr at e-boundary* $\frac{1}{s}$
ALIGN qsdteb	with yz-boundary !	*T-tendency of qs at e-boundary* $\frac{1}{s}$
ALIGN qvdteb	with yz-boundary !	*T-tendency of qv at e-boundary* $\frac{1}{s}$
ALIGN udteb	with yz-boundary !	*T-tendency of u at e-boundary* $\frac{m}{s^2}$
ALIGN vdteb	with yz-boundary !	*T-tendency of v at e-boundary* $\frac{m}{s^2}$
ALIGN wdteb	with yz-boundary !	*T-tendency of w at e-boundary* $\frac{m}{s^2}$
ALIGN pdtnb	with xz-boundary !	*T-tendency of pprt at n-boundary* $\frac{Pascal}{s}$
ALIGN ptdtnb	with xz-boundary !	*T-tendency of ptprt at n-boundary* (K/s)
ALIGN qcdtnb	with xz-boundary !	*T-tendency of qc at n-boundary* $\frac{1}{s}$
ALIGN qhdtnb	with xz-boundary !	*T-tendency of qh at n-boundary* $\frac{1}{s}$
ALIGN qidtnb	with xz-boundary !	*T-tendency of qi at n-boundary* $\frac{1}{s}$
ALIGN qrdtnb	with xz-boundary !	*T-tendency of qr at n-boundary* $\frac{1}{s}$
ALIGN qsdtnb	with xz-boundary !	*T-tendency of qs at n-boundary* $\frac{1}{s}$
ALIGN qvdtnb	with xz-boundary !	*T-tendency of qv at n-boundary* $\frac{1}{s}$
ALIGN udtnb	with xz-boundary !	*T-tendency of u at n-boundary* $\frac{m}{s^2}$
ALIGN vdtnb	with xz-boundary !	*T-tendency of v at n-boundary* $\frac{m}{s^2}$
ALIGN wdtnb	with xz-boundary !	*T-tendency of w at n-boundary* $\frac{m}{s^2}$
ALIGN pdtsb	with xz-boundary !	*T-tendency of pprt at s-boundary* $\frac{Pascal}{s}$
ALIGN ptdtsb	with xz-boundary !	*T-tendency of ptprt at s-boundary* $\frac{K}{s}$
ALIGN qcdtsb	with xz-boundary !	*T-tendency of qc at s-boundary* $\frac{1}{s}$
ALIGN qhdtsb	with xz-boundary !	*T-tendency of qh at s-boundary* $\frac{1}{s}$
ALIGN qidtsb	with xz-boundary !	*T-tendency of qi at s-boundary* $\frac{1}{s}$
ALIGN qrdtsb	with xz-boundary !	*T-tendency of qr at s-boundary* $\frac{1}{s}$
ALIGN qsdtsb	with xz-boundary !	*T-tendency of qs at s-boundary* $\frac{1}{s}$
ALIGN qvdtsb	with xz-boundary !	*T-tendency of qv at s-boundary* $\frac{1}{s}$
ALIGN udtsb	with xz-boundary !	*T-tendency of u at s-boundary* $\frac{m}{s^2}$
ALIGN vdtsb	with xz-boundary !	*T-tendency of v at s-boundary* $\frac{m}{s^2}$
ALIGN wdtsb	with xz-boundary !	*T-tendency of w at s-boundary* $\frac{m}{s^2}$
ALIGN pdtwb	with yz-boundary !	*T-tendency of pprt at w-boundary* $\frac{Pascal}{s}$
ALIGN ptdtwb	with yz-boundary !	*T-tendency of ptprt at w-boundary* $\frac{K}{s}$
ALIGN qcdtwb	with yz-boundary !	*T-tendency of qc at w-boundary* $\frac{1}{s}$
ALIGN qhdtwb	with yz-boundary !	*T-tendency of qh at w-boundary* $\frac{1}{s}$
ALIGN qidtwb	with yz-boundary !	*T-tendency of qi at w-boundary* $\frac{1}{s}$
ALIGN qrdtwb	with yz-boundary !	*T-tendency of qr at w-boundary* $\frac{1}{s}$
ALIGN qsdtwb	with yz-boundary !	*T-tendency of qs at w-boundary* $\frac{1}{s}$
ALIGN qvdtwb	with yz-boundary !	*T-tendency of qv at w-boundary* $\frac{1}{s}$
ALIGN udtwb	with yz-boundary !	*T-tendency of u at w-boundary* $\frac{m}{s^2}$
ALIGN vdtwb	with yz-boundary !	*T-tendency of v at w-boundary* $\frac{m}{s^2}$
ALIGN wdtwb	with yz-boundary !	*T-tendency of w at w-boundary* $\frac{m}{s^2}$

Figure 4.6: The domain variables for the boundaries

Chapter 5

Benchmarking ARPS on the CM5

The CM5 is used for benchmarking the dataparallel implementation of the ARPS code. The machine available at the Northeast Parallel Architectures Center at Syracuse University consists of 32 nodes with a SPARC 1 scalar CPU. Each node has a memory of 32MB and four vector units. The processors are connected with the help of a quad-tree topology. The peak performance is estimated to be 3 GFLOPS.

To compare the execution time in a forthcoming study between a version using message passing vs. a data parallel program, as introduced in this paper, timers are set as shown in the calling tree (Figure 5.1, 5.2, and 5.3). To be consistent with previously published results a common initial data set is used. The data is stored in the so called *sounding* file. The sounding file *may20.snd* contains one dimensional sounding data from a supercell storm which took place at 20 May, 1977 in Ft. Sill, OK. From this data the next 6.0 seconds are predicted. To exclude all possible interferences caused by the timesharing system the runs have been done on an empty machine.

Timers in the Calling Tree

To measure the time for the execution of some routines the CM timer functions timer_start, timer_clear and timer_stop have been used. For the sequential program this timer calls have been reimplemented. 64 timers are usable. A particular timer can be accessed in the following way (Figures 5.1 – 5.3):

(=0) is equivalent to a clear the timer with number 0

(+0) is equivalent to start or better continue the timer 0

(-0) is equivalent to stop or better to interrupt the timer 0

The timers are called outside the subroutine to be measured. For example, the function `call f(a,b,c)` is timed in the following way:

```
call timer_start(n)
call f(a,b,c)
call timer_stop(n)
```

5.1 Timing Results

For the timings the program has been compiled for the CM5 with the compiler options *-O -vu* to enable optimization and the usage of the vector units. While the elapsed time on the CM5 is 13.086 seconds the CM busy Time is 10.194 seconds. The timings of the routines are compared to the timings one can obtain while running the program on a single workstation. Here a SPARC 1 is used[1]. On this machine the code did run 24.71 seconds using all possible compiler optimization options. As stated in [?] the execution of the code on a single node of the CM5 is about 28 seconds.
It is obvious that the speedup gained for the particular procedures is different. The overall speedup is about 2.5 while the speedup for the routine *microph* is about 9. The explanation for this difference is quite easy.

[1] Th name of the machine is lambda.npac.syr.edu

Table 5.1: Comparison of the timers between a single workstation and a CM5 with 32 nodes

Timer	Procedure	Sequential ARPS CPU Time in seconds	Parallel ARPS CPU Time in seconds	Speedup
1	tinteg	19.983	7.995	2.499
2	microph	4.250	0.458	9.279
3	tfilt	0.000	0.000	0.000
5	frcuvw	5.616	1.100	5.105
6	frcp	0.417	0.175	2.383
7	nestbdt	0.000	0.000	0.000
8	acoust	9.866	4.854	2.033
9	solvpt	1.250	0.464	2.694
10	solvqv	0.950	0.439	2.164
11	solvq	1.850	0.962	1.923
12	mixuvw	3.567	0.466	7.655
13	advuvw	1.567	0.467	3.355
14	coriol	0.000	0.000	0.000
15	buoycy	0.400	0.161	2.484
16	solvuvw	7.233	3.224	2.243
64	Total	24.716	10.194	2.425

- First, the data distribution is done on the three spatial dimensions. The time axis has been kept serially.
- Second, CM Fortran is more efficient when the serial dimensions are on the left and not on the right as it is in the current parallelized code.
- Third, vectorization is done on the procedural level. It is to be expected while using inlining of code the performance will increase.

This results in high speedups for low level functions with many computations like the subroutine *microph* but to little speed up for higher level routines like *tinteg*. To speed up the computation even further one should change the array layout in such a way that the serial array dimension is to the right. In addition, we notice that the boundary conditions are handled in two dimensional arrays. It would be good to place them in a three dimensional array and do the calculations

on the elements of the boundary. The execution in double precision leads to a performance gain from about 15%. The limited time and man power did not allow to work on this issues. Preliminary, results to be published in [?] indicate that these additional improvements will reduce the running time to about 4 seconds.

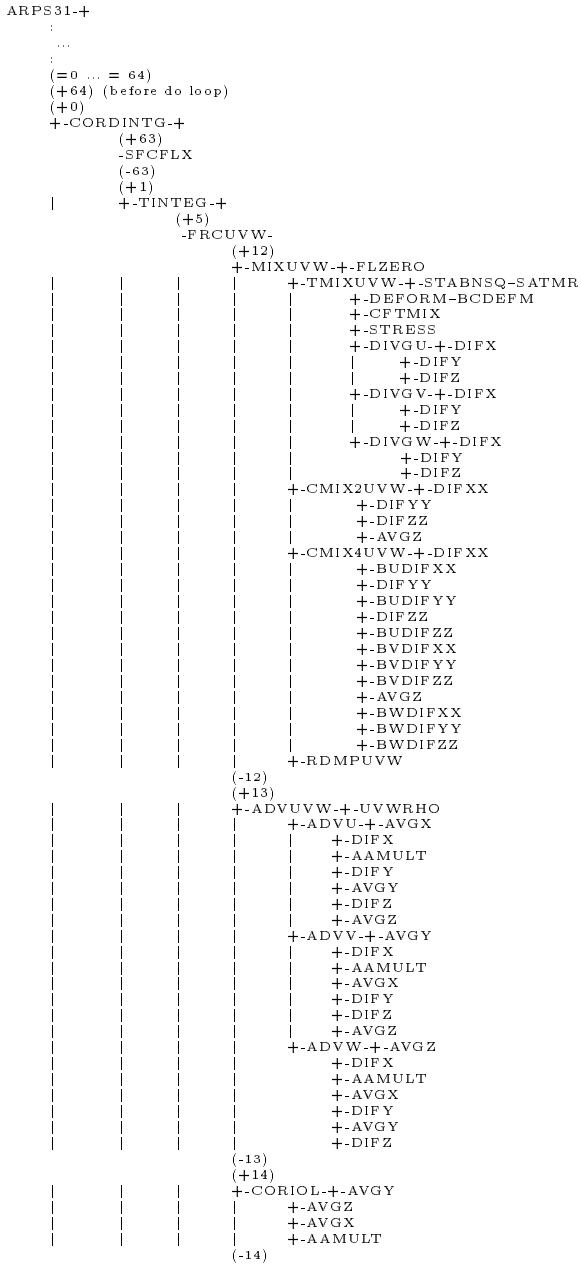


Figure 5.1: The timer for the benchmark, Part a

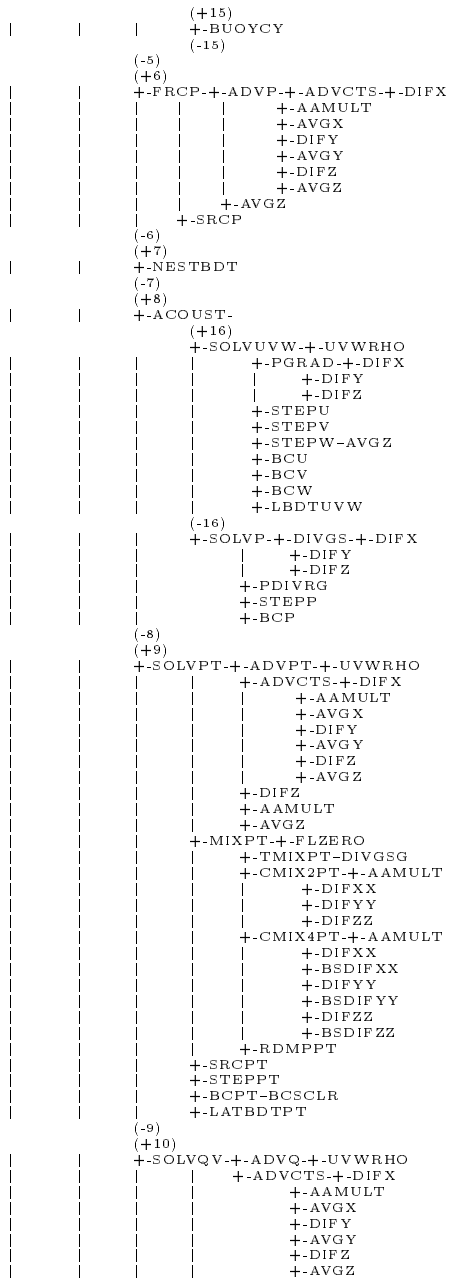


Figure 5.2: The timer for the benchmark, Part b

```
|       |       |       +-MIXQV-+-FLZERO
|       |       |       |     +-TMIXQV-DIVGSG
|       |       |       |     +-CMIX2Q-+-AAMULT
|       |       |       |     |        +-DIFXX
|       |       |       |     |        +-DIFYY
|       |       |       |     |        +-DIFZZ
|       |       |       |     +-CMIX4Q-+-AAMULT
|       |       |       |     |        +-DIFXX
|       |       |       |     |        +-BSDIFXX
|       |       |       |     |        +-DIFYY
|       |       |       |     |        +-BSDIFYY
|       |       |       |     |        +-DIFZZ
|       |       |       |     |        +-BSDIFZZ
|       |       |       +-STEPQ
|       |       |       +-BCQ-BCSCLR
|       |       |       +-LATBDTQ
                (-10)
                (+11)
|       |       +-SOLVQ-+-ADVQ-+-UVWRHO
|       |       |        |     +-ADVCTS-+-DIFX
|       |       |        |              +-AAMULT
|       |       |        |              +-AVGX
|       |       |        |              +-DIFY
|       |       |        |              +-AVGY
|       |       |        |              +-DIFZ
|       |       |        |              +-AVGZ
|       |       |        +-MIXQ-+-FLZERO
|       |       |        |      +-TMIXQ-DIVGSG
|       |       |        |      +-CMIX2Q-+-AAMULT
|       |       |        |      |        +-DIFXX
|       |       |        |      |        +-DIFYY
|       |       |        |      |        +-DIFZZ
|       |       |        |      +-CMIX4Q-+-AAMULT
|       |       |        |      |        +-DIFXX
|       |       |        |      |        +-BSDIFXX
|       |       |        |      |        +-DIFYY
|       |       |        |      |        +-BSDIFYY
|       |       |        |      |        +-DIFZZ
|       |       |        |      |        +-BSDIFZZ
|       |       |        +-STEPQ
|       |       |        +-BCQ-BCSCLR
|       |       |        +-LATBDTQ
                (-11)
        (-1)
        (+2)
|       +-MICROPH-+-AUTOCAC
|       |         +-REVAP-SATMR
|       |         +-QRFALL
|       |         +-SATADJ-SATMR
        (-2)
        (+3)
|       +-TFILT-ASELIN
        (-3)
        (+4)
|       +-RADBDT-+-BDTU
|       |        +-BDTV
|       |        +-BDTP
        (-4)
|       +-TFLIP-TSWAP
|       +-CHKSTAB (outcommented)
(-0)
+-OUTPUT   (outcommented)
:
 ...
:
+-CHKSTAB   (outcommented)
:
 ...
:
+-(GRAFCLOSE) (outcommented)
(-64) (after do loop)
```

Figure 5.3: The timer for the benchmark, Part c

Chapter 6

Summary

This study shows that it is a big software engineering challenge to parallelize a huge code like the ARPS code. In a limited time it is only possible while using a semiautomatic translation tool. For the ARPS code it is sufficient to have translator functions for

1. the data layout,
2. forall loops,
3. and loop vectorization.

This is motivated by the observation that

- the procedures are similar in their structure,
- variable names are used consistently,
- common variables have been avoided,
- the main computation is done in do loops with few data dependencies.

In addition it is shown that the introduction of problem specific compiler directives determining the layout of a variable are a most helpful while adapting the code from one machine to the other. This includes even the possibility to transfer the CAPS code to HPF.

The benchmarking results show clearly that the current version of CM Fortran has to be improved while dealing with serial and parallel dimensions in an array. There should be no need for the programmer to change the code in such a way that the serial dimension is placed to the far left. This could be done easily by the compiler. One way to deal with this problem is to introduce a new directive PERMUTE which enables the permutation of the subscription indices to a variable without actually rewriting the code. This is also discovered independently in [?]. The following example illustrates this issue:
Assume the following code:

```
      REAL a(nx,ny,nz,nt)
CMF   LAYOUT a(news:,news:,news:,serial:)
CMF   PERMUTE a(4,1,2,3)
      a(i,j,k,t) = 5
```

The PERMUTE directive transforms the code into:

```
      REAL a(nx,ny,nz,nt)
CMF   LAYOUT a(serial:,news:,news:,news:)
      a(t,i,j,k) = 5
```

With this additional directive it is to be expected that the running time of the parallel program can be improved by the factor of 2 as demonstrated in [?]. This permutation is not necessary for HPF. In a forthcoming study a comparison between the data parallel and the message passing version of the ARPS code will be done.

Acknowledgment

I thank Kim Mills, Gang Cheng for their valuable discussions while parallelizing the CAPS code and Rahul Bhargava to make his initial attempt to parallelize the ARPS 3.0 code available to me.

Bibliography

[1] Bozkus, Z., Choudhary, A., Fox, G., Haupt, T., and Ranka, S. Fortran 90D/HPF Compiler for Distributed Memory MIMD Computers. Tech. rep., Syracuse University, Rice University, 1990.

[2] Fox, G., Hiranandani, S., Kennedy, K., Koebel, C., Kremer, U., Tseng, C.-W., and Wu, M. Fortran D Language Specification. Tech. Rep. Rice COMP TR90079, SCCS42c, Syracuse University, Rice University, 1990.

[3] von Laszewski, G. Customized Interactive Parallelization of Fortran 77 Programs. Tech. Rep. SCCS 510, Northeast Parallel Architectures Center at Syracuse University, July 1993.

[4] von Laszewski, G. Implementing the Advanced Regional Prediction System (ARPS) with Fortran D. Tech. Rep. SCCS 492, Northeast Parallel Architectures Center at Syracuse University, June 1993.

[5] von Laszewski, G. The MIMD Volume Distributor for Multigrid Methods. Tech. Rep. SCCS 478, Northeast Parallel Architectures Center at Syracuse University, April 1993.

Chapter 7

Appendix

7.1 Program Tree

```
ARPS31-+-INITIAL-+-INITPARA-+-STRLNTH
       |         |          +-PRTPARA-+-GTLOGFN
       |         |                    +-STRLNTH
       |         +-INIT0-FLZERO
       |         +-INIGRD-JACOB
       |         +-INITVAR-+-INIBASE-ZPROFIL-+-SOUNDG-GETQVS
       |                   |                 +-SNDINTRP-INTE1D
       |                   +-INITDVR-RANARY-RAN3
       |                   +-INITBDT-ZEROBDT
       |                   +-RSTIN-+-JACOB
       |                   |       +-CPYARY
       |                   |       +-A3DMAX0
       |                   +-EXTINIT-+-JACOB
       |                             +-CPYARY
       |                             +-A3DMAX0
       +-INITOUT-+-A3DMAX0
       |         +-MAXMIN-A3DMAX
       |         +-ENERGY-+-AAMULT
       |         |        +-UVWRHO
       |         +-BASPRT-WRIGAR-OUTARR
       |         +-FMTPRT-+-WRIGAR-OUTARR
       |         |        +-PLTARY-PARRAY
       |         +-GTBASFN
       |         +-DTADUMP-+-BINDUMP-KNTARY
       |         |         +-ASCDUMP-KNTARY
       |         |         +-HDFDUMP-+-(DSSDIMS)
       |         |         |         +-KNTARY
       |         |         |         +-HDFGDMP-+-(DSSDIST)
       |         |         |         |         +-(DSSDISC)
       |         |         |         +-EDGFILL
       |         |         |         +-(DSSDAST)
       |         |         |         +-(DSADATA)
       |         |         +-PAKDUMP-+-KNTARY
       |         |         |         +-MKHEAD-TRNCHAR
       |         |         |         +-A3DMAX0
       |         |         |         +-EDGFILL
       |         |         |         +-PACKDAT
```

```
|       |       |       +-MKLABEL-+-TRNCHAR
|       |       |               +-TRNREAL
|       |       +-SVIDUMP-+-(GRAFOPEN)
|       |       |         +-(GRAFDEFNGRID)
|       |       |         +-(GRAFWRITEGRIDPOINT)
|       |       |         +-(GRAFDEFNSCALAR)
|       |       |         +-(GRAFDEFNVECT)
|       |       |         +-CVTTIM
|       |       |         +-(GRAFTIMESTART)
|       |       |         +-(GRAFTIMESTEP)
|       |       |         +-(GRAFNEWFRAME)
|       |       |         +-(GRAFWRITESCALARPOINT)
|       |       |         +-(GRAFENDFRAME)
|       |       +-BN2DUMP-KNTARY
|       +-GTDMPFN-CVTTSND
+-CORDINTG-+-SFCFLX
|          +-TINTEG-+-FRCUVW-+-MIXUVW-+-FLZERO
|          |        |        |        +-TMIXUVW-+-STABNSQ-SATMR
|          |        |        |        |        +-DEFORM-BCDEFM
|          |        |        |        |        +-CFTMIX
|          |        |        |        |        +-STRESS
|          |        |        |        |        +-DIVGU-+-DIFX
|          |        |        |        |        |       +-DIFY
|          |        |        |        |        |       +-DIFZ
|          |        |        |        |        +-DIVGV-+-DIFX
|          |        |        |        |        |       +-DIFY
|          |        |        |        |        |       +-DIFZ
|          |        |        |        |        +-DIVGW-+-DIFX
|          |        |        |        |                +-DIFY
|          |        |        |        |                +-DIFZ
|          |        |        |        +-CMIX2UVW-+-DIFXX
|          |        |        |        |          +-DIFYY
|          |        |        |        |          +-DIFZZ
|          |        |        |        |          +-AVGZ
|          |        |        |        +-CMIX4UVW-+-DIFXX
|          |        |        |        |          +-BUDIFXX
|          |        |        |        |          +-DIFYY
|          |        |        |        |          +-BUDIFYY
|          |        |        |        |          +-DIFZZ
|          |        |        |        |          +-BUDIFZZ
|          |        |        |        |          +-BVDIFXX
|          |        |        |        |          +-BVDIFYY
|          |        |        |        |          +-BVDIFZZ
|          |        |        |        |          +-AVGZ
|          |        |        |        |          +-BWDIFXX
|          |        |        |        |          +-BWDIFYY
|          |        |        |        |          +-BWDIFZZ
|          |        |        |        +-RDMPUVW
|          |        |        +-ADVUVW-+-UVWRHO
|          |        |        |        +-ADVU-+-AVGX
|          |        |        |        |      +-DIFX
|          |        |        |        |      +-AAMULT
|          |        |        |        |      +-DIFY
|          |        |        |        |      +-AVGY
|          |        |        |        |      +-DIFZ
|          |        |        |        |      +-AVGZ
|          |        |        |        +-ADVV-+-AVGY
|          |        |        |        |      +-DIFX
|          |        |        |        |      +-AAMULT
|          |        |        |        |      +-AVGX
|          |        |        |        |      +-DIFY
|          |        |        |        |      +-DIFZ
|          |        |        |        |      +-AVGZ
|          |        |        |        +-ADVW-+-AVGZ
|          |        |        |                +-DIFX
|          |        |        |                +-AAMULT
|          |        |        |                +-AVGX
```

```
|       |     |     |          +-DIFY
|       |     |     |          +-AVGY
|       |     |     |          +-DIFZ
|       |     |     +-CORIOL-+-AVGY
|       |     |     |        +-AVGZ
|       |     |     |        +-AVGX
|       |     |     |        +-AAMULT
|       |     |     +-BUOYCY
|       |     +-FRCP-+-ADVP-+-ADVCTS-+-DIFX
|       |     |      |     |         +-AAMULT
|       |     |      |     |         +-AVGX
|       |     |      |     |         +-DIFY
|       |     |      |     |         +-AVGY
|       |     |      |     |         +-DIFZ
|       |     |      |     |         +-AVGZ
|       |     |      |     +-AVGZ
|       |     |      +-SRCP
|       |     +-NESTBDT
|       |     +-ACOUST-+-SOLVUVW-+-UVWRHO
|       |     |        |         +-PGRAD-+-DIFX
|       |     |        |         |       +-DIFY
|       |     |        |         |       +-DIFZ
|       |     |        |         +-STEPU
|       |     |        |         +-STEPV
|       |     |        |         +-STEPW-AVGZ
|       |     |        |         +-BCU
|       |     |        |         +-BCV
|       |     |        |         +-BCW
|       |     |        |         +-LBDTUVW
|       |     |        +-SOLVP-+-DIVGS-+-DIFX
|       |     |        |       |        +-DIFY
|       |     |        |       |        +-DIFZ
|       |     |        |       +-PDIVRG
|       |     |        |       +-STEPP
|       |     |        |       +-BCP
|       |     +-SOLVPT-+-ADVPT-+-UVWRHO
|       |     |        |       +-ADVCTS-+-DIFX
|       |     |        |       |        +-AAMULT
|       |     |        |       |        +-AVGX
|       |     |        |       |        +-DIFY
|       |     |        |       |        +-AVGY
|       |     |        |       |        +-DIFZ
|       |     |        |       |        +-AVGZ
|       |     |        |       +-DIFZ
|       |     |        |       +-AAMULT
|       |     |        |       +-AVGZ
|       |     |        +-MIXPT-+-FLZERO
|       |     |        |       +-TMIXPT-DIVGSG
|       |     |        |       +-CMIX2PT-+-AAMULT
|       |     |        |       |         +-DIFXX
|       |     |        |       |         +-DIFYY
|       |     |        |       |         +-DIFZZ
|       |     |        |       +-CMIX4PT-+-AAMULT
|       |     |        |       |         +-DIFXX
|       |     |        |       |         +-BSDIFXX
|       |     |        |       |         +-DIFYY
|       |     |        |       |         +-BSDIFYY
|       |     |        |       |         +-DIFZZ
|       |     |        |       |         +-BSDIFZZ
|       |     |        |       +-RDMPPT
|       |     |        +-SRCPT
|       |     |        +-STEPPT
|       |     |        +-BCPT-BCSCLR
|       |     |        +-LATBDTPT
|       |     +-SOLVQV-+-ADVQ-+-UVWRHO
|       |     |        |      +-ADVCTS-+-DIFX
|       |     |        |      |        +-AAMULT
```

```
|       |      |      |          +-AVGX
|       |      |      |          +-DIFY
|       |      |      |          +-AVGY
|       |      |      |          +-DIFZ
|       |      |      |          +-AVGZ
|       |      |      +-MIXQV-+-FLZERO
|       |      |      |       +-TMIXQV-DIVGSG
|       |      |      |       +-CMIX2Q-+-AAMULT
|       |      |      |       |       +-DIFXX
|       |      |      |       |       +-DIFYY
|       |      |      |       |       +-DIFZZ
|       |      |      |       +-CMIX4Q-+-AAMULT
|       |      |      |                +-DIFXX
|       |      |      |                +-BSDIFXX
|       |      |      |                +-DIFYY
|       |      |      |                +-BSDIFYY
|       |      |      |                +-DIFZZ
|       |      |      |                +-BSDIFZZ
|       |      |      +-STEPQ
|       |      |      +-BCQ-BCSCLR
|       |      |      +-LATBDTQ
|       |      +-SOLVQ-+-ADVQ-+-UVWRHO
|       |      |       |     +-ADVCTS-+-DIFX
|       |      |       |              +-AAMULT
|       |      |       |              +-AVGX
|       |      |       |              +-DIFY
|       |      |       |              +-AVGY
|       |      |       |              +-DIFZ
|       |      |       |              +-AVGZ
|       |      |       +-MIXQ-+-FLZERO
|       |      |       |      +-TMIXQ-DIVGSG
|       |      |       |      +-CMIX2Q-+-AAMULT
|       |      |       |      |        +-DIFXX
|       |      |       |      |        +-DIFYY
|       |      |       |      |        +-DIFZZ
|       |      |       |      +-CMIX4Q-+-AAMULT
|       |      |       |               +-DIFXX
|       |      |       |               +-BSDIFXX
|       |      |       |               +-DIFYY
|       |      |       |               +-BSDIFYY
|       |      |       |               +-DIFZZ
|       |      |       |               +-BSDIFZZ
|       |      |       +-STEPQ
|       |      |       +-BCQ-BCSCLR
|       |      |       +-LATBDTQ
|       +-MICROPH-+-AUTOCAC
|       |         +-REVAP-SATMR
|       |         +-QRFALL
|       |         +-SATADJ-SATMR
|       +-TFILT-ASELIN
|       +-RADBDT-+-BDTU
|       |        +-BDTV
|       |        +-BDTP
|       +-TFLIP-TSWAP
|       +-CHKSTAB-+-MAXMIN-A3DMAX
|                 +-FMTPRT-+-WRIGAR-OUTARR
|                 |        +-PLTARY-PARRAY
|                 +-GTDMPFN-CVTTSND
|                 +-DTADUMP-+-BINDUMP-KNTARY
|                 |         +-ASCDUMP-KNTARY
|                 |         +-HDFDUMP-+-(DSSDIMS)
|                 |         |         +-KNTARY
|                 |         |         +-HDFGDMP-+-(DSSDIST)
|                 |         |         |         +-(DSSDISC)
|                 |         |         +-EDGFILL
|                 |         |         +-(DSSDAST)
|                 |         |         +-(DSADATA)
```

```
|             |       +-PAKDUMP-+-KNTARY
|             |       |       +-MKHEAD-TRNCHAR
|             |       |       +-A3DMAX0
|             |       |       +-EDGFILL
|             |       |       +-PACKDAT
|             |       |       +-MKLABEL-+-TRNCHAR
|             |       |                 +-TRNREAL
|             |       +-SVIDUMP-+-(GRAFOPEN)
|             |       |         +-(GRAFDEFNGRID)
|             |       |         +-(GRAFWRITEGRIDPOINT)
|             |       |         +-(GRAFDEFNSCALAR)
|             |       |         +-(GRAFDEFNVECT)
|             |       |         +-CVTTIM
|             |       |         +-(GRAFTIMESTART)
|             |       |         +-(GRAFTIMESTEP)
|             |       |         +-(GRAFNEWFRAME)
|             |       |         +-(GRAFWRITESCALARPOINT)
|             |       |         +-(GRAFENDFRAME)
|             |       +-BN2DUMP-KNTARY
|             +-(GRAFCLOSE)
+-OUTPUT-+-RSTOUT-+-CVTTSND
|        |        +-CPYARY
|        +-MAXMIN-A3DMAX
|        +-ENERGY-+-AAMULT
|        |        +-UVWRHO
|        +-FMTPRT-+-WRIGAR-OUTARR
|        |        +-PLTARY-PARRAY
|        +-GTDMPFN-CVTTSND
|        +-DTADUMP-+-BINDUMP-KNTARY
|                  +-ASCDUMP-KNTARY
|                  +-HDFDUMP-+-(DSSDIMS)
|                  |         +-KNTARY
|                  |         +-HDFGDMP-+-(DSSDIST)
|                  |         |         +-(DSSDISC)
|                  |         +-EDGFILL
|                  |         +-(DSSDAST)
|                  |         +-(DSADATA)
|                  +-PAKDUMP-+-KNTARY
|                  |         +-MKHEAD-TRNCHAR
|                  |         +-A3DMAX0
|                  |         +-EDGFILL
|                  |         +-PACKDAT
|                  |         +-MKLABEL-+-TRNCHAR
|                  |                   +-TRNREAL
|                  +-SVIDUMP-+-(GRAFOPEN)
|                  |         +-(GRAFDEFNGRID)
|                  |         +-(GRAFWRITEGRIDPOINT)
|                  |         +-(GRAFDEFNSCALAR)
|                  |         +-(GRAFDEFNVECT)
|                  |         +-CVTTIM
|                  |         +-(GRAFTIMESTART)
|                  |         +-(GRAFTIMESTEP)
|                  |         +-(GRAFNEWFRAME)
|                  |         +-(GRAFWRITESCALARPOINT)
|                  |         +-(GRAFENDFRAME)
|                  +-BN2DUMP-KNTARY
+-CHKSTAB-+-MAXMIN-A3DMAX
|         +-FMTPRT-+-WRIGAR-OUTARR
|         |        +-PLTARY-PARRAY
|         +-GTDMPFN-CVTTSND
|         +-DTADUMP-+-BINDUMP-KNTARY
|         |         +-ASCDUMP-KNTARY
|         |         +-HDFDUMP-+-(DSSDIMS)
|         |         |         +-KNTARY
|         |         |         +-HDFGDMP-+-(DSSDIST)
|         |         |         |         +-(DSSDISC)
|         |         |         +-EDGFILL
```

```
|       |       |       +-(DSSDAST)
|       |       |       +-(DSADATA)
|       |       +-PAKDUMP-+-KNTARY
|       |       |       +-MKHEAD-TRNCHAR
|       |       |       +-A3DMAX0
|       |       |       +-EDGFILL
|       |       |       +-PACKDAT
|       |       |       +-MKLABEL-+-TRNCHAR
|       |       |                 +-TRNREAL
|       |       +-SVIDUMP-+-(GRAFOPEN)
|       |       |       +-(GRAFDEFNGRID)
|       |       |       +-(GRAFWRITEGRIDPOINT)
|       |       |       +-(GRAFDEFNSCALAR)
|       |       |       +-(GRAFDEFNVECT)
|       |       |       +-CVTTIM
|       |       |       +-(GRAFTIMESTART)
|       |       |       +-(GRAFTIMESTEP)
|       |       |       +-(GRAFNEWFRAME)
|       |       |       +-(GRAFWRITESCALARPOINT)
|       |       |       +-(GRAFENDFRAME)
|       |       +-BN2DUMP-KNTARY
|       +-(GRAFCLOSE)
+-(GRAFCLOSE)
```

7.2 Operators

```
SUBROUTINE AAMULT (a,b,ab)
do k=kbgn,kend
  do j=jbgn,jend
    do i=ibgn,iend
      ab(i,j,k)=a(i,j,k)*b(i,j,k)
    end do
  end do
end do
END
```

```
SUBROUTINE AVGX (a, onvf, aavg)
IF ( onvf.eq.1) THEN
  iright = 0
  ileft = -1
ELSE
  iright = 1
```

```
  ileft = 0
 END IF
 do k=kbgn,kend
 do j=jbgn,jend
 do i=ibgn,iend
  aavg(i,j,k)=(a(i+iright,j,k)
:            +a(i+ileft ,j,k))*0.5
 end do
 END
```

```
 SUBROUTINE DIFX (a, onvf, adifx)
 IF ( onvf.eq.1) THEN
  iright = 0
  ileft = -1
 ELSE
  iright = 1
  ileft = 0
 END IF
 dxinv = 1.0/dx
 DO 100 k=kbgn,kend
 DO 100 j=jbgn,jend
 DO 100 i=ibgn,iend
  adifx(i,j,k)=(a(i+iright,j,k)-a(i+ileft ,j,k))*dxinv
100  CONTINUE
 RETURN
 END
```

7.3 Availability

Since the code is about 69000 lines long it would be unhandy to provide the complete source code in the appendix. The code is available via anonymous ftp at NPAC.

The site name is *FTP.NPAC.SYR.EDU*. This archive site is provided by the Northeast Parallel Architectures Center at Syracuse University. Access is allowed 24 hours a day, seven days a week. All transfers will be logged with your host name and email address. If your FTP client crashes or hangs shortly after login, try using a dash (-) as the first character of your password. This will turn off the informational messages that may be confusing your ftp client.
This FTP server will handle compression on the fly. If you append a .Z (or .gz) to a filename, it will be compressed (or gzipped) before transmission; if you omit one of these suffixes, the file will be sent uncompressed. The server also understands the suffixes .tar, .tar.Z, and .tar.gz, which can be used to retrieve entire directories as well as individual files.
If you have any problems or questions, the archive maintainers can be reached by electronic mail at the address *ftp@ftp.npac.syr.edu*.
The directory in which the parallel ARPS code for the CM5 is stored is:

/pub/gregor

The file to be downloaded is

arps31.parallel.tar.Z

A makefile is included. For technical reports please contact the author at gregor@npac.syr.edu or lsd@npac.syr.edu.