Using the parallel version of the ARPS

Adwait Sathye
Center for Analysis and Prediction of Storms
University of Oklahoma
asathye@tornado.ca.s.ou.edu

1 Introduction

Moving a sequential code to a parallel version invariably leads to a significant amount of changes to the original structure and content. We have tried to automate the process, and shield the code changes from the end user. This document details the changes made to the ARPS code, how to 'make' the model, the extra tools necessary to split & combine the history dumps and other data files, and simple queue scripts to submit the code on the Cray T3D (in PSC) and the IBM SP-2 (in Maui HPCC). **Note:** The code compilation and job submission is to be done on the front end machine (mario, in case of the PSC C90) when working with the T3D.

2 Changes to the ARPS code

All the code changes, in this released version, are handled by a translator, that will convert the sequential ARPS code into parallel form. The translator uses the **cMP** comments embedded in the code, so the user should not delete or modify these lines in any form. The translator will convert the original files and add a _PlatformMachine extension. The Platform could be pvm/mpi (for now) and the Machine could either be t3d or the sp2 (e.g. the translated version of tinteg3d.f is called tinteg3d_pvmt3d for PVM execution on the t3d. Similarly the arps executable is called arps40_pvmt3d). For cluster execution, the Machine part is ignored (i.e. there will only be a Platform extension).

Note: If you are going to make changes to the code, make those changes to the original ARPS source, or they will be lost in a subsequent translation. A couple of 'rules of thumb'

- If you add any error checks with STOP, add in similar code from the **arps40.f** file. This should include a call to stop the parallel execution.
- As far as possible, do not change any **boundary condition** code without checking with one of us.
- In the **I/O** routines, if you add new variables, dont forget to add the same variables to the split & join source files.

The rest of the section details the changes made to the code, to make it parallel. These will remain comment statements unless you create the parallel version. **Note:** All the changes are shown along with the original file name.

- **Node Initialization** (whoami etc.) **arps40.f** Some code has been added to restrict multiple messages.
- **Model initialization**
  - **init3d.f:** has changes to INIGRD which calculate the local grids position in the global grid. Some boundary condition code in JACOB

- **initpara3d.f:** The 0'th processor reads in the input file name and broadcasts it to other processors. Only the 0'th processor will print the output log and input description.

- **Model Input/Output**
  - **out3d.f:** No significant changes
  - **outlib3d.f:** Both A3DMAX0 and A3DMAX calculate their own max's and min's and communicate with other processors to calculate global max's and min's. All output and input data files have a _ suffix followed by the processors location in the processor grid e.g. the first processors output(and input) files will have a _0101 suffix.
  - **dump3d.f/read3d.f/exbcio3d.f/rst3d.f:** The Cray T3D does not support IEEE single precision format, so every processor writes in COS format. We have programs to combine these output files and write them in single precision IEEE_sp format. Only the binary output format is supported at the moment. Any changes to the output sequence or new variables in the input/output stream will need a corresponding change to the split & join programs. More on this in the 'how to run' section

- **Boundary Conditions**

  Boundary values (only North, East, West and South) are exchanged with the neighboring processors every small time step. We added some communication space (look at `par.inc, par1d.inc`, and `par2d.inc`) to store values to be sent/ received, and some code was added to copy data to/from the corresponding data arrays. Modifying boundary condition code will be harder for a person not familiar with parallel computing, and it would be best if you contact us (at the arpsuser@tornado.caps.ou.edu address) before making any changes.

  - **advct3d.f**
  - **bc3d.f**
  - **bcdif3d.f**
  - **rst3d.f**
  - **soilebm3d.f**
  - **tke3d.f**
  - **init3d.f**

- **Miscellaneous Files**

  These files contain minor changes to the original code.

  - **exbc3d.f**
  - **exbcio3d.f**
  - **iolib3d.f**
  - **setbdt3d.f**
  - **tinteg3d.f**
  - **soildiag3d.f**
  - **energy3d.f:** Contains calls to a subroutine to calculate global sums of variables

- **Other files**

  The rest of the files in the ARPS model have not been modified so you can make changes to those files with no side effects on the parallel execution of the code.

3 Setting up the model

This part contains the external changes to the ARPS model. Perhaps the most significant, at least the one with the most effect to ARPS'ers, is the changes to `dims.inc`. The values in `NX, NY` in `dims.inc` now reflect the size of the grid per processor. The values `NPROCX` and `NPROCY` indicate the number of processors in the `X` and `Y` direction. The total number of processors used in that run are indicated by `NPROCX * NPROCY`. To calculate the size of your entire grid,

$$global\_x\_size = (nx - 3) * nprocx + 3$$

and

$$global\_y\_size = (ny - 3) * nprocy + 3$$

The `nz` value is not affected (its 2 dimensional decomposition along the `X` and `Y` dimensions).

It might just be easier to select your final grid size and then calculate

$$nx = (global\_x\_size - 3)/nprocx + 3$$

and

$$ny = (global\_y\_size - 3)/nprocy + 3$$

You can set the values of `nprocx` and `nprocy` depending on the size of your model, and the number of jobs in the MPP queue. As a rule of thumb, the not-optimized-for-MPP code performs at 11MFLOPS per processor (on the T3D) , or 32 processors equal 1 C90 processor, dedicated. It might be a good idea to use 16 processors for smaller models and 64 for the larger cases. Its usually harder to get 128 & more processors. Also, the total number of processors have to be a power of 2 (on the T3D, not on the SP-2).

A new file, `par.inc`, has been added to set up the sizes of the communication buffers. As far as possible, try to keep

$$NX\_MX = NX$$

and

$$NY\_MX = NY$$

though it wont hurt if those values are larger than `nx, ny` We have added a subroutine to check these values at run time, and the program will stop if those `nx` and/or `ny` are set to be larger than the values in this file.

Also, if you will be using external boundary data to initialize/nudge the model, do not forget to set the values of `nxebc` and `nyebc` (in `exbc.inc`) to be the same as `nx` and `ny`

4 Compiling the model

The compile script for the parallel model has been merged into makearps. To create a parallel version for the machine you are working on, type `makearps arps40_pvm`. This will create not only the parallel version, but also the translator, and utilities to split data files (splitfiles) and join the output history dumps (joinfiles). On the Cray T3D and the IBM SP2, we add a Machine extension (i.e. for the T3D, the names change to arps40_pvmt3d, joinfiles_t3d and splitfiles_t3d). Since the header files, and libraries, are platform dependent and their locations depend on where the system administrator placed them, we have added two new options to makearps, `mpp_inc` and `mpp_lib`. The `mpp_inc` specifies the location of the include files (e.g. fpvm3.h, mpif.h etc.), while `mpp_lib` specifies the location of the library (e.g. libmpi.a, libfpvm3.a etc.) files.

5 Running ARPS on parallel machines

5.1 Queue Scripts

The Cray uses Network Queueing System (NQS) to schedule jobs, while the IBM SP-2 uses Loadleveller. Most supercomputers have an interactive job limit, and queueing systems allow you to run larger and longer jobs in a non-interactive manner. The scripts are similar to shell scripts with extra keywords (like QSUB in the case of NQS) to define your job limits and the execution environment.

A typical NQS header will look like

```
#QSUB -lM 7Mw -lT 15000
#QSUB -q mpp
#QSUB -l mpp_p=16
#QSUB -l mpp_t=9000
#QSUB -s /usr/local/bin/tcsh
#QSUB -eo
#QSUB -o $AFS/arps40.log
```

The `mpp_p` sets the number of processors, while `mpp_t` sets the computation time limit for each individual processor. The `-q mpp` will work only on the PSC NQS.

A sample Loadleveller script looks like

```
#!/bin/csh
#
# initialdir = /u/asathye/ebc
# input = input.name
# output = arps40.output
# error = arps40.error
# class = Medium
# job_type = pvm3 (or parallel for MPI jobs)
# min_processors = 16
# max_processors = 16
# requirements = (Adapter == "hps_user")
# parallel_path = /u/asathye/pvm3/bin/RS6K/
#
# notify_user = asathye@tornado.gcn.uoknor.edu
# queue
```

However, a program called 'xloadl' can help you construct a script with a X interface (Select the File option). Also, you can find more info on Loadleveller at

`http://www.mhpcc.edu/training/workshop/html/loadleveler/LoadLeveler.html`

5.2 Splitting the data files

If you are planning to use any external data files, you will have to split those files before you execute the parallel version. The following programs are available to split the necessary data files. The **splitfiles** tool will scan the input file for files that need to split, and will split those files (once again, the splitfiles is called splitfiles_t3d and splitfiles_sp2 on the T3D and SP2 respectively). **Note: Some systems might have a limit on the number of open files. Please check/set your environment limits, via the `limit` command, before you run the model**.

```
splitfiles < arps40.input
```

5.3 Running the model

To execute, say a 16 processor job on the Cray-T3D, do

```
echo arps40.input >!  input.name
arps40_pvmt3d -npes 16 < input.name >!  output
```

We had to change from feeding the input file to the program to feeding a file with the name of the input file in it. In the above example, the file, `input.name` should contain the name of the actual input file, for instance `arps40.input`.

On the SP-2, we follow the typical PVM style. First copy the executable to the /pvm3/bin/RS6K/ directory. Then start the master process.

```
master &
```

You can execute the MPI version with

```
echo arps40.input >!  input.name
mpirun -np 16 arps40_pvm < input.name >!  output
```

5.4 Joining the data files

The new tool joinfiles, will combine all the history dumps at the end of the run. Similar, to joinfiles, it will scan the input file for the runname, and the frequency of history dumps, and combine the history dumps. **Note: Some systems might have a limit on the number of open files. Please check/set your environment limits, via the `limit` command, before you run `joinfiles`.**

```
joinfiles < arps40.input
```

A A few technical details about the port

- Two dimensional decomposition along the horizontal
- Size 1 “halo” or “fake” zone
- Communication only during boundary exchanges every time step
- Flexible ownership paradigm. Use SPMD, but allow for easy conversion to Master-Slave, if need be
- Even if Master-Slave paradigm was used, reduce I/O control in the Master region
- Use a hierarchical message pattern for global reduction operations
- Use barriers only during initialization and completion
- Write simple tools to allow automatic generation of the communication code. The ARPS is an evolving model and we wanted to incorporate the latest changes into the parallel code

B Architectural differences between the C90 and the T3D

- Processor
  - C90: Cray proprietary. Capable of 1GFLOPS

- T3D: DEC Alpha EV4. 150MFLOPS
- SP2(Wide): IBM Power2. 266 MFLOPS

- Cache
  - C90: 256 Words for instructions. Data is read through vector pipelines into vector registers (8 registers each containing 64 words).
  - T3D: 1KW for instructions; 1KW for data.
  - SP2(Wide): 256Kb data cache

- Memory
  - C90: Shared memory. All processors are hooked up to a common memory. Memory divided into banks.
  - T3D: Physically distributed, logically shared memory. All processors have their own local memories, but each processor can access any memory element in the machine. Memory divided into pages, each 4KW in size.
  - SP2: Physically distributed memory machine.

- Network
  - C90: crossbar (shared memory).
  - T3D: Two processing elements (PE's) make up a node. Each node is connected to 6 neighboring nodes via a network switch which can transfer data at the rate of 150MB/s, in six directions, simultaneously. The nodes are connected as a 3D Torus (3D grid with all edge points connected to their counterparts; sort of like a hardware interpretation of the reflective boundary condition).
  - SP2: Similar to Omega Network. 4x4 crossbar switching elements. 16 nodes per frame.

- Performance inhibitors
  - C90: Code is not vectorized. Large stride. Code does not fit in instruction cache. I/O in (computation) do loops.
  - T3D: Small cache size. Using the division operator. Low small step size value causes frequent communication.
  - SP2(wide): Similar to the T3D except that the SP2 wide node has a large cache.

C Location of the code

- ftp://ftpcaps.ou.edu/pub/ARPS/arps40.official.tar.Z

D Job submission and status information

- Queue submission
  - T3D: Use `qsub` to submit a NQS job
  - SP2: Use `llsubmit` to submit a Loadleveler job

- Job status
  - T3D: Use `qstat -a` to check on your jobs status. The prefix `qm` indicates a mpp job. Alternately, you can type `qstat` and look at the jobs under `mpp`. Use `mppstat` to figure out the number of free PE's if you are planning to run an interactive job or use `mppstat -q` to check your jobs status in the execute queue.
  - SP2: `llq` will indicate your jobs current status.