Chapter 7

Application of a parallel processing

In this section, we will describe the computation method of ***CReSS*** with a parallel computing.

The parallel program of ***CReSS*** is fundamentally designed for running on a multiple memory type parallel computer. The parallel program is written with Message Passing Interface (MPI). ***CReSS*** is available to almost all parallel computers, because the MPI is a defactor standard library. Using some workstations, ***CReSS*** can also be performed in a cluster environment.

7.1 Technique for parallelization

7.1.1 Two-dimensional domain decomposition

CReSS is fundamentally designed for a multiple memory type parallel computer. A parallel program is available for not only large multiple memory type parallel computers but also cluster systems of workstations and PC-UNIX computers.

It is easy to use the parallel processing by adopting two-dimensional domain decomposition in horizontal, because a spacial difference in horizontal is explicitly calculated in ***CReSS***. Various physical processes, such as cloud microphysics, can be fundamentally calculated only with the physical quantities in the caluculating cell, there are some references between upper and lower cells. A z^* coordinate system (ζ system) makes parallelization easy that a coordinate system.

Firstly, we show the advantage of parallel processing by adopting two-dimensional domain decomposition in horizontal.

- It is easy to establish the boundary condition, because coordination system is z^*-system (ζ-system) has no loss of data near the surface.
- It is easy to understand the technique of parallelization intuitively.
- It is available to use the same function on nodes except the node included the boundary condition.
- The load balance is high. (That is, a waiting time for other node's computation would be short.)
- Parallelization efficiency for a calculation with a number of grid points would be high.
- As a calculation domain for each node is larger, data communication between each node is relatively smaller.

We show a schematic illustration of two-dimensional domain devomposition as follows.

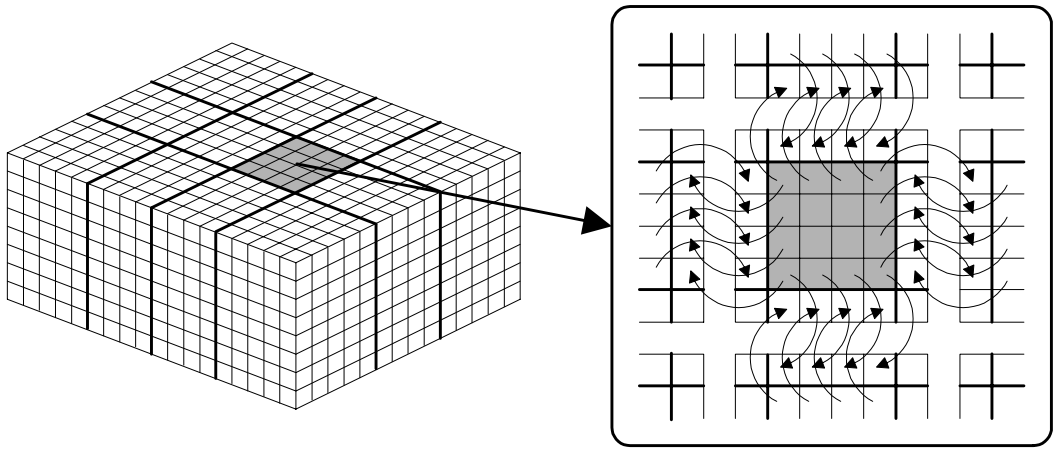

Figure 7.1. Schematic illustration of two-dimensional domain decomposition and exchange of the value in the halo region

Figure 7.1 shows a schematic illustration of the parallel computing in the case of calculating with the second-order precision. In the case of adopting central difference approximation with the second order precision, six grid points must be referred for a certain grid point. Considering the horizontal two-dimensional domain decomposition, there is no problem of reference in the vartical direction. In horizontal ,however, when the approximate solution of the grid point on the boundary of decomposition is

calculated,the reference between nodes is needed. Then showing the enlarged figure on right side, we set the halo region in each node, and it is enable to obtain the same result as by one node on the computational domain computed by decomposed nodes had the halo region. For that purpose, communication between the nodes are performed only for getting the essential data from neighboring the node whenever the value in the halo region is needed (The detail is shown in subsection 7.1.2)

In addition, ***CReSS*** is available to compute with fourth-order precision. For that purpose, it is necessary to communicate much the value which is needed by doubling a halo region like the case of second order precision. It is possible to change the precision of the differential equation by the same method, for example, sixth-order, eighth-order, ... and N-th-order.

7.1.2 Example of parallel computing

In this subsection, we describe the communication between halo regions explained in subsection 7.1.1.

The source code is written with Message Passing Interface (MPI). The MPI is the interface specifications for the call form of various communication routines required for a multiple memory type parallel computer and the function and subroutines of C and Fortran. But it is interface specification to the last, and the concrete application method is not determined.

As for the details about MPI, it is good to refer to the following homepages.

http://www.mpi-forum.org

For communicating by MPI such as figure 7.1, it is essential to repeat a shift in all directions except the boundary plane. Figure 7.2 is the example of a shift to the halo region of east side from the halo region of west side except the boundary plane. In this figure, *nipe* is numbers of node in the east-west direction and *ipe* is node number.

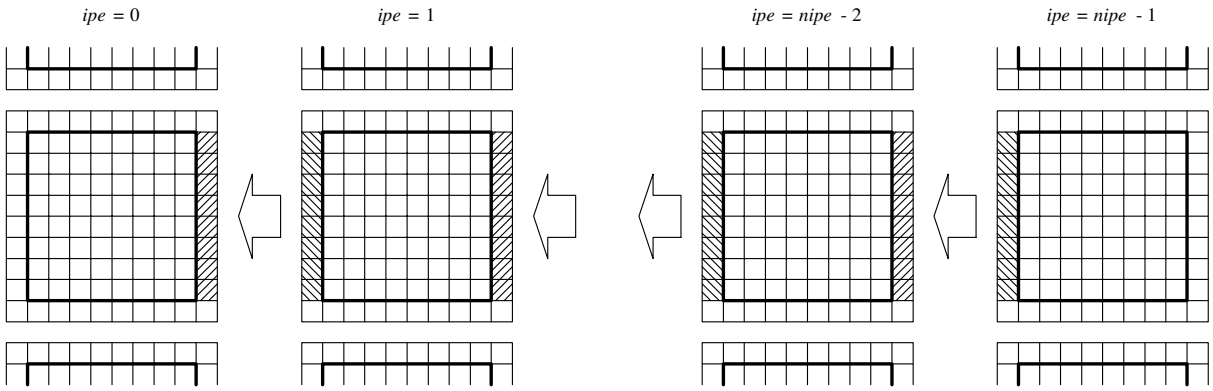

Figure 7.2. This is a example of a shift to the halo region of east side from the halo region of west side.

The source code of the MPI parts in ***CReSS***is shown as following. The part of it is shown here, so for understanding the detail it is necessary to refer `exchansn.f` and `exchanwe.f` in the directory `Src`.

```
      ! A node at the westernmost end has no transmitting buffer
      if(ipe.eq.0) then
        dst=mpi_proc_null
      end if

      ! A node at the easternmost end has no transmitting buffer
      if(ipe.eq.nipe-1) then
        src=mpi_proc_null
      end if

      ! A setup of transmitting buffers other than a node at the westernmost end
      if(ipe.ne.0) then

        do 140 k=kbmin,kbmax
        do 140 j=jbmin,jbmax
          ib=(k-kbmin)*jbmxn1-jbmin1+j

          sbuf(ib)=var(iwsend,j,k)

140     continue

      end if

      ! The call of the MPI routine for transmission and reception
      call mpi_sendrecv(sbuf,siz,mpi_real,dst,tag,rbuf,siz,mpi_real,src,
     .                  tag,mpi_comm_world,stat,ierr)

      ! A setup of receiving buffers other than a node at the easternmost end
      if(ipe.ne.nipe-1) then

        do 170 k=kbmin,kbmax
        do 170 j=jbmin,jbmax
          ib=(k-kbmin)*jbmxn1-jbmin1+j

          var(ierecv,j,k)=rbuf(ib)

170     continue

      end if
```

Although the shift is mounted by `mpi_sendrecv` here, it is better not to mount this by `mpi_send` and `mpi_recv`. Although it is not impossible, various programming can be considered and may carry out a deadlock depending on the case. Moreover, by the formal specification of the MPI, the guarantee of operation of a program will be impossible, since it seems that it has not defined whether a deadlock is carried out by this or it does not carry out.

7.2 Test of the parallel program

7.2.1 Inspection of coincidence of a calculation result

Since parallel programming adopted in ***CReSS*** should just do a shift fundamentally shown in the previous section, the result performed using several nodes must be completely same as the result performed using one node (in calculation for the whole average value, it is not necessarily to get complete agreement. Since it does not apply operation to the value in shifting, disagreement cannot happen.). We checked the agreement by ***CReSS*** as follows.

When several nodes are used for calculation, a result is outputted from every node.

ex. : *exprim*`.dmp`*xxxxxx*`.pe0000.bin` ∼ *exprim*`.dmp`*xxxxxx*`.pe`*yyyy*`.bin`

For comparing to the result by one node and that by several nodes, we gather the above data files up to one file. (A post processor "unite" can gather the output files up to a binary unformated file with direct access.) The file was compared with the result of calculation with one node. If the files of both one node and several nodes are text format, we can confirm their complete agreement as the following command.

ex. : `% diff` *exprim*`.dmp`*xxxxxx*`.united.txt` *exprim*`.dmp`*xxxxxx*`.txt`

As a result of executing this command, if nothing happens (0 byte of file will be made if a file is outputted to a standard input/output), the result must completely be in agreement. *exprim* of file name is an experiment name (see section 9.1).

We perform this verification to various experiments with ***CReSS***, and there is no problem in the parallelization.

7.2.2 Efficiency of parallel processing

Finally, the efficiency of parallel processing is shown by a comparison of calculations with the various numbers of processing elements (PEs). The performance of parallel processing of ***CReSS*** was tested by a simulation which had $67 \times 67 \times 35$ grid points and which was integrated for 50 steps on HITACHI SR2201.

As increase of the numbers of PEs, the calculation time decreased almost linearly (Fig. 7.3). The results of the test showed a sufficiently high performance of the parallel computing of ***CReSS***. In figure 7.4, it can see that the efficiency of parallel processing is falling little by little. When the number of PEs was 32, the efficiency decreased significantly. Because the number of grid points was too small to use the 32 PEs, the communication between PEs became relatively large for the present verification setting. This result of the verification shows that the efficiency of parallel processing of CReSS is higher when each PE shares larger number of grid points. The calculation with a lot of PEs is more useful for larger calculation.

However, there is a problem. Especially, it is becoming more remarkable in the case of using the option for cloud microphysics. Because the program for cloud microphysics has so many `if` sentences. For example, when a node without clouds must wait for the processing of the node with clouds, efficiency would be small. In the present condition, ***CReSS*** does not have advanced parallel processing changed from the first setup to the more efficient setup in the calculation.

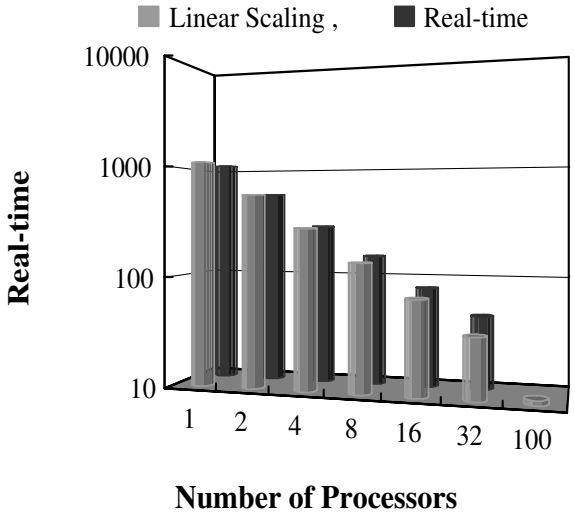

Figure 7.3. Time for parallel processing from a test experiment [s]

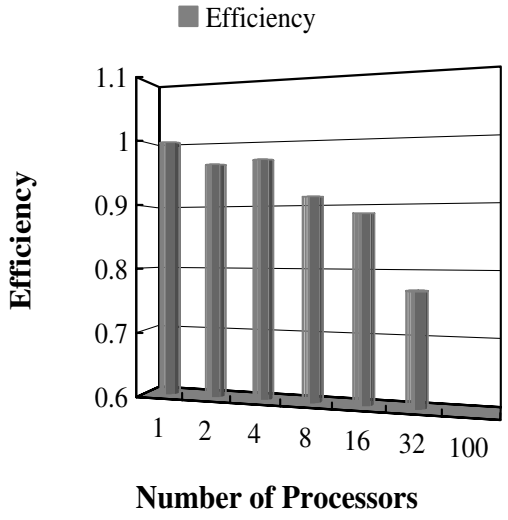

Figure 7.4. Efficiency of parallel processing from a test experiment

The values of efficiency in the graph is somewhat wavy in figure 7.4, since it is based on the difference between the ways of a decomposition on two-dimensional domain decomposition. (For example, when a calculation is performed with four nodes, there is the ways of dividing of 1×4, 2×2, or 4×1.) However, it is sufficient for seeing the tendency of the efficiency of parallel processing.