§62. Improvement of Calculation Efficiency of Simulation Code on Open System

Ohtani, H., Ishiguro, S., Horiuchi, R.

In the previous paper, we report how to distribute the array and free boundary condition for particles in the distributed parallel algorithm. In this paper, the efficient calculation of gather process are explained and the performance of calculation is shown.

It is difficult to calculate the charge density and current density in the vector algorithm because several particles exist in the same cell in some cases (Fig. 1(a)). In order to avoid this difficulty and to assign the particle information to different array from one another, we make working array in this calculation as follows (Fig. 1(b)):

```
!hpf$ independent
      do k=1,l
       do i=1,n(k),m
        do j=1,min(m,n(k)-i)
         ix=int(x(i,k)/dx+0.5)
         xx=x(i,k)/dx-ix
         s1=(0.5-xx)**2/2
         s2=(0.75-xx**2)
         s3=(0.5+xx)**2/2
         w(j,ix-1,k)=w(j,ix-1,k)+q*s1/dx
         w(j,ix,  k)=w(j,ix,  k)+q*s2/dx
         w(j,ix+1,k)=w(j,ix+1,k)+q*s3/dx
        enddo
       enddo
      enddo
```

However, this working array needs a lot of memory in large scale simulation. When we use the compile-directive *listvec*, we can decrease the memory of the working array and perform the vector calculation [1].

```
!hpf$ independent
     do k=1,l
       do m=-1,1
!cdir listvec
        do i=1,n(k)
         ix=int(x(i,k)/dx+0.5)
         xx=x(i,k)/dx-ix
         if(m.eq.-1) s1=(0.5-xx)**2/2
         if(m.eq. 0) s1=(0.75-xx**2)
         if(m.eq.+1) s1=(0.5+xx)**2/2
         ix=ix+m
         w(ix,k)=w(ix,k)+q*s1/dx
        enddo
       enddo
      enddo
```

Note that if the particles whose numbers are close to one another exit in the same cell (2 and 3 in Fig. 1(a)), the vector calculation is not performed but the scalar calculation is. So we should randomize the particle's number

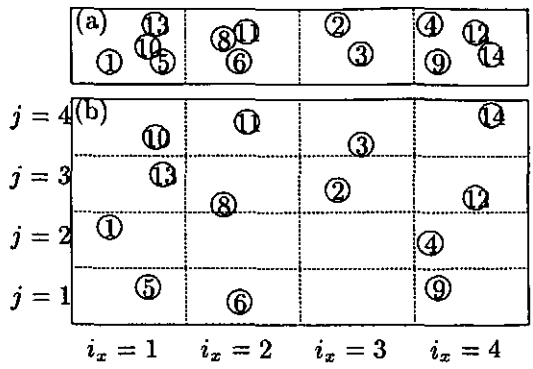

Fig. 1. Schematic illustration of gather process (a) without and (b) with working array.

sufficiently when we make the initial particle distribution.

We show the calculation performance of (a) working array case and (b) *listvec* case in Table 1. Calculation parameters are as follows; the number of particles is 240 million, the number of active particles is 96 million, the grid size is $256 \times 128 \times 128$ and the time step is 1000. The number of distributed parallel calculation (HPF processes) is 5, and the number of common parallel calculation is 32.

The vector operation ratio (v.o.r.) of the case (a) is about 0.1 point larger than that of the case (b). However the calculation performance becomes extremely good when v.o.r is larger than 99%. The vector length of case (a) becomes also longer compared with case (b). The value of floating-point arithmetic per second (FP) of case (a) is smaller than that of case (b), because the addition of working array along vector direction (j) is need in case (b) and not in case (a), and the essential calculation quantity reduces. The memory of working array of case (a) is much smaller than that of case (b), and the total memory dramatically decreases. As a consequence, the calculation time decreases in case (a) and it becomes possible to perform larger scale simulation due to using the compile-directive *listvec*.

The future works are the improvement of physics model and the modification of Poisson solver which needs a lot of memory for array long preliminary calculation.

	v.o.r.	v.l.	FP	mem	time(s)
(a)	99.34	224.91	103.78	952.5	4574.6
(b)	99.43	240.15	77.17	277.2	3343.5

Table 1. Calculation performance of (a) working array case and (b) *listvec* case. V.o.r., v.l., FP and mem show vector operation ratio (%), vector length, floating-point arithmetic per second (GFlops) and memory (Gbytes), respectively.

Reference

1) Sugiyama, T. et al.: IPSJ **45**, 171 (2004).