nbody-with-center: A (not so) new code for simulation of planetary rings

Jun Makino

Internal Seminar Dec 8, 2021

Talk plan

1. FDPS and planetary ring
2. Codes available
3. nbody-with-center
4. Development status
5. How to use
6. Command-Line Options

FDPS and planetary ring

FDPS has been used by several groups for the study of planetary rings/mooon formation

- **Michikochi and Kokubo 2017 (Thin ring)**
- **Sasaki and Hosono 2018 (Moon formation)**
- **Iwasawa et al. 2020 (Algorithm and Performance)**

Codes available

Now, Iwasawa-san added a code, planetary-ring, to samples of FDPS. This code supports

- FDPS 7.x (cylindrical coordinate)
- Massive satellites
- PIKG

So one can use this code, but it is currently a bit difficult to use/modify

- Not well documented
- Written in (too) advanced C++

nbody-with-center

- Originally written by me back in 2017 as a sample code to be used by Ishibashi-san
- Use soft-core potential + dumping force (vertical only) for physical collision
- I used this code to test version 7.x of FDPS on Fugaku
- As a result, now this code is reasonably tested and fast
- I added some useful functions so that it can be used real scentific works.

Development status

- **Performance reasonable on Fugaku and also x86 platforms**
- **Further performance optimization possible (reuse-mode, mode optimized kernel etc.)**
- **Energy check includes dissipation due to dumping term for collision**

How to use

Install on x86

```
git clone https://github.com/jmakino/nbody-with-center.git
cd nbody-with-center
```

Edit Makefile so that `PS_PATH` points to the location of FDPS source

```
make nbody-with-center-quad
./nbody-with-center-quad -i ringin_hot -T 0.5 -r 0.004 -K 128
```

For use with MPI

```
make nbody-with-center-quad-mpi
mpirun -np 4 ./nbody-with-center-quad-mpi -i ringin_hot -T 0.5 -r 0.004 -K 128
```

Command-Line Options

(For complete list see Readme.md or try -h option)

-D: Output interval for snapshot

-d: Output interval for diagnostics

-g: Inner cutoff radius for particle distribution. Particles closer to center than this radius will be removed

-i: The name of initial condition snapshot file. Format:

```
time
n
id1 mass1 x1 y1 z1 vx1 vy1 vz1
...
idn massn xn yn zn vxn vyn vzn
```

-K: Number of timesteps to resolve collison.

-M:Mass of the central planet.

-o: Name of the output directory

-p: Softening parameter

-R: Number of divisions in radial direction for MPI parallelization

-r: Physical radius of particles (shared by all particles)

-S: Vertiacal restitution coefficient

-s: Timestep

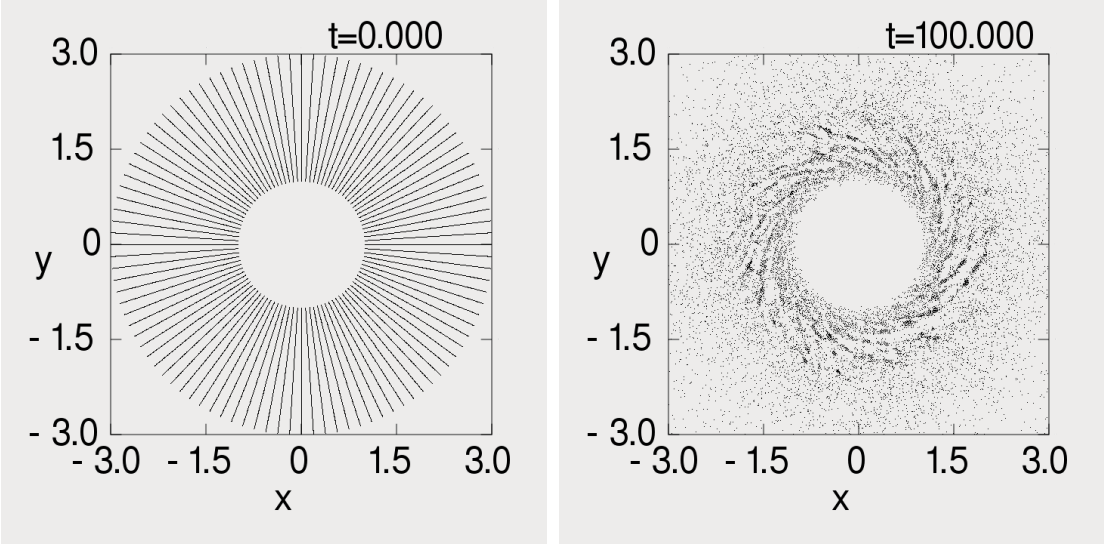
-t:Tree opening angle

-T:Simulation end time

Example calculation

```
mpirun -np 16 nbody-with-center-quad-mpi -T 1000 -r 0.01 -K 128 \
  -i moontest20k.in -o result -D 0.125 -g 0.95 -S 0.1
```

https://github.com/jmakino/nbody-with-center/blob/master/moontest20k.gif



Performance

20k particles, 512 steps, 4 core skylake notebook : 15 sec.
170k steps/second·core

Note on SIMD kernel

```
#pragma omp simd reduction(+:aix,aiy,aiz,poti,dedtdumperi)
        for(int j=0; j<nj; j++){
            PS::F64 rijx    = xix - xj[j];
            PS::F64 rijy    = xiy - yj[j];
            PS::F64 rijz    = xiz - zj[j];
            PS::F64 mjlocal = mj[j];
            PS::F64 m_r = mjlocal/ (mi+mjlocal);
            PS::F64 r2 = rijx*rijx +rijy*rijy +rijz*rijz +eps2;
            PS::F64 r2_inv;
            PS::F64 r_inv;
            PS::F64 r = sqrt(r2);
            PS::F64 dr = r_coll-r ;
            if(r_coll_sq < r2){
                r2_inv  = 1.0/r2;
                r_inv  =  sqrt(r2_inv);
                dr = 0.0;
                m_r = 0.0;
```

```
}else{
    r_inv = r_coll_inv;
    r2_inv = r_inv*r_inv;
    if(iid == jid[j]) {
        m_r=0;
        mjlocal=0;
    }
}
PS::F64 pot = r_inv * mjlocal;
aix     +=  -pot * r2_inv * rijx;
aiy     +=  -pot * r2_inv * rijy;
aiz     +=  -pot * r2_inv * rijz;
poti   +=  -pot;
PS::F64 kappacoef = kappa * m_r * dr/r;
aix += kappacoef* rijx;
aiy += kappacoef* rijy;
aiz += kappacoef* rijz;
poti += kappacoef*dr*r * 0.5;
PS::F64vec vij = pi[i].vel - pj[j].vel;
```

```
        PS::F64 vijx = vix - vxj[j];
        PS::F64 vijy = viy - vyj[j];
        PS::F64 vijz = viz - vzj[j];
        PS::F64 rv = rijx*vijx + rijy*vijy + rijz*vijz;
        if(r_coll_sq < r2) rv=0.0;

        PS::F64 etacoef = -eta * m_r * rv * r2_inv;
        aix += etacoef * rijx;
        aiy += etacoef * rijy;
        aiz += etacoef * rijz;
        dedtdumperi -= etacoef * rv*m_r;
    }
```

This version can be SIMDized with gcc 7.5. Should be okay with Fugaku. (A more complex version was okay on Fugaku but not with gcc 7.5)

- **Use only 1D arrays and Scalar variables**
- **Avoid complex and large if block**

Kernel aseembly code

Generated with gcc -S

vd instructions for zmm registers (AVX512) are emitted. Looks okay.**

Should treat the cases with physical contacts separately (at least for ring calculations. For moon calculations most particles have contacts)

```
.L18:
        vsubpd  (%r9,%rax), %zmm26, %zmm5
        addl    $1, %edx
        vmovupd (%rbx,%rax), %zmm13
        vsubpd  (%rsi,%rax), %zmm27, %zmm6
        vsubpd  (%r10,%rax), %zmm25, %zmm4
        vaddpd  %zmm24, %zmm13, %zmm7
        vmulpd  %zmm5, %zmm5, %zmm15
        vfmadd231pd     %zmm6, %zmm6, %zmm15
        vaddpd  -560(%rbp), %zmm15, %zmm1
        vdivpd  %zmm7, %zmm13, %zmm0
        vfmadd231pd     %zmm4, %zmm4, %zmm1
```

```
vsqrtpd %zmm1, %zmm2
vsubpd  %zmm2, %zmm17, %zmm31
vcmppd  $2, %zmm14, %zmm1, %k2
vcmpltpd        %zmm1, %zmm14, %k4
vmovdqu64       0(%r13,%rax), %zmm12{%k2}
vcmplepd        %zmm14, %zmm1, %k1
vpcmpq  $4, %zmm16, %zmm12, %k5{%k2}
vpcmpq  $0, %zmm16, %zmm12, %k3{%k2}
vdivpd  %zmm2, %zmm28, %zmm3
vmulpd  %zmm30, %zmm2, %zmm2
vmulpd  %zmm20, %zmm0, %zmm15
vmulpd  %zmm31, %zmm2, %zmm2
vmulpd  %zmm31, %zmm3, %zmm3
vmovapd %zmm18, %zmm31
vdivpd  %zmm1, %zmm28, %zmm7
vmulpd  %zmm19, %zmm3, %zmm3
vmulpd  %zmm0, %zmm3, %zmm3
vsqrtpd %zmm7, %zmm31{%k4}
vmulpd  %zmm13, %zmm31, %zmm13
```

```
vmovapd %zmm20, %zmm31
vdivpd  %zmm1, %zmm28, %zmm7
vmulpd  %zmm19, %zmm3, %zmm3
vmulpd  %zmm0, %zmm3, %zmm3
vsqrtpd %zmm7, %zmm31{%k4}
vmulpd  %zmm13, %zmm31, %zmm13
vmovapd %zmm20, %zmm31
vmulpd  %zmm3, %zmm2, %zmm2
vmovapd %zmm7, %zmm31{%k4}
vmovapd %zmm2, %zmm1{%k1}{z}
vmovapd %zmm2, %zmm1{%k5}
vmovapd -368(%rbp), %zmm2
vsubpd  %zmm13, %zmm2, %zmm2
vmovapd %zmm8, %zmm1{%k3}
vaddpd  %zmm2, %zmm1, %zmm1
vsubpd  (%rdi,%rax), %zmm22, %zmm2
vmovapd %zmm1, -368(%rbp)
vsubpd  (%r8,%rax), %zmm23, %zmm1
vmulpd  %zmm5, %zmm2, %zmm2
```

```
vfmadd132pd     %zmm4, %zmm2, %zmm1
vsubpd  (%r11,%rax), %zmm21, %zmm2
addq    $64, %rax
cmpl    -476(%rbp), %edx
vfmadd132pd     %zmm6, %zmm1, %zmm2
vmovapd %zmm15, %zmm1
vmulpd  %zmm7, %zmm0, %zmm1{%k4}
vmovapd %zmm31, %zmm7
vmovapd %zmm3, %zmm31{%k1}{z}
vmovapd %zmm8, %zmm0{%k3}
vmovapd %zmm3, %zmm31{%k5}
vmovapd %zmm15, %zmm1{%k5}
vmulpd  -624(%rbp), %zmm2, %zmm15
vmovapd %zmm8, %zmm31{%k3}
vmovapd %zmm8, %zmm1{%k3}
vfnmadd132pd    %zmm7, %zmm31, %zmm13
vmulpd  %zmm15, %zmm1, %zmm1
vmovapd %zmm1, %zmm3{%k1}{z}
vmovapd %zmm8, %zmm1{%k3}
```

```
vfnmadd132pd    %zmm7, %zmm31, %zmm13
vmulpd  %zmm15, %zmm1, %zmm1
vmovapd %zmm1, %zmm3{%k1}{z}
vsubpd  %zmm3, %zmm13, %zmm13
vfmadd231pd     %zmm13, %zmm6, %zmm11
vmulpd  %zmm2, %zmm0, %zmm6
vfmadd231pd     %zmm13, %zmm5, %zmm10
vfmadd231pd     %zmm13, %zmm4, %zmm9
vmovapd -432(%rbp), %zmm0
vmovapd %zmm11, -176(%rbp)
vmovapd %zmm10, -240(%rbp)
vmovapd %zmm9, -304(%rbp)
vmulpd  %zmm1, %zmm6, %zmm5
vxorpd  %zmm29, %zmm5, %zmm4
vmovapd %zmm4, %zmm4{%k1}{z}
vsubpd  %zmm4, %zmm0, %zmm2
vmovapd %zmm2, -432(%rbp)
jb      .L18
```

Summary

- **nbody-with-center is now available, with some documentation, for ring and massive disk calculations**
- **SIMDized kernel with reasonable performance, and highly scalable with FDPS 7.1.**