Recent development of FDPS

Jun Makino
R-CCS Particle Simulator Research Team

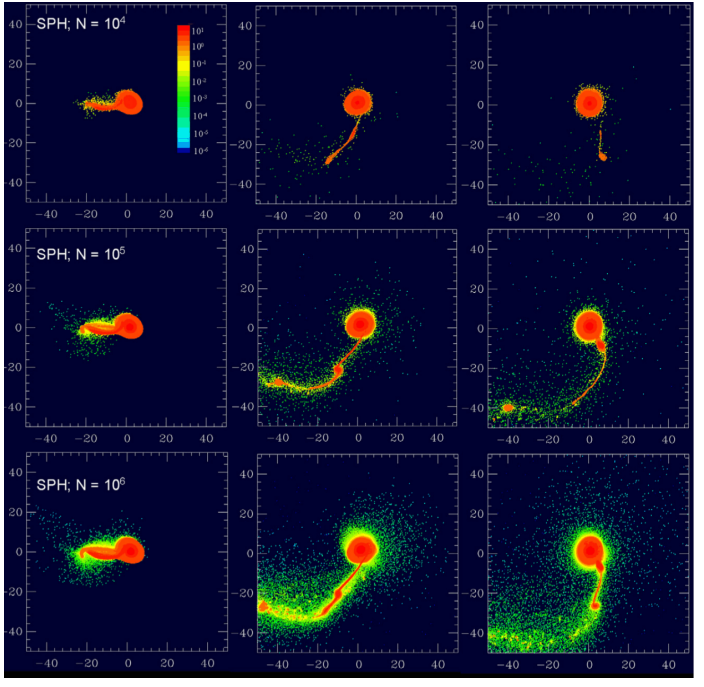
R-CCS Cafe Sept 6. 2021

Talk plan

1. What is FDPS?
2. Basic functions and the history of additions
3. Recent additions/improvements
4. Summary

What we want to do

- We want to try large simulations.
- Computers (or the network of computers...) are fast enough to handle hundreds of millions of particles, for many problems.
- In many fields, largest simulations still employ 1M or less particles....



(example: Canup+ 2013)

Why?

Writing parallel codes which run efficiently on modern supercomputers has become too difficult.

- **Just to make a parallel code working is not easy**
- **We need to implement an efficient domain decomposition which achieves very good load balancing and at the same time minimizes communication.**
- **For long-range interactions, we need to implement complex schemes such as Particle-Mesh Ewald, FMM, TreePM, P^3M or PM^3.**
- **We need to take advantage of SIMD instructions, and keep doing so for new versions of them (SSE, AVX, AVX2, AVX512, SVE, SVE2, ...) or some other vendor-specific languages such as Cuda.**

But what we can do?

Traditional ideas:

- **Hope that parallelizing compilers will solve all problems.**
- **Hope that big shared memory machines will solve all problems.**
- **Hope that parallel languages (with some help of compilers) will solve all problems.**

But...

- **These hopes have never been.....**
- **Reason: low performance. Only approaches which achieve the best performance on the most inexpensive systems have survived.**

Then what can we really do?

1. **Accept the reality and write MPI programs and do optimization**
   Limitation: If you are an ordinary person the achieved performance will be low, and yet it will take more than infinite time to develop and debug programs. Your researcher life is likely to finish before you finish programming.
2. **Let someone else do the work**
   Limitation: If that someone else is an ordinary person the achieved performance will be low, and yet it will take more than infinite time and money.

- **Neither is ideal**
- **There are exceptions**

Exceptional Products

Within Astrophysics

- **pkdgrav (Quinn et al. 1997)**
- **Gadget (Springel et al. 2001)**
- **GreeM (Ishiyama et al. 2009)**

Molecular Dynamics: Gromacs, LAMMPS, NAMD, Genesis, Modylas, ...

CAE: commercial codes

Problems with exceptional products

- They are designed for their specific problems. They do not solve your problem.
- You can choose:
  - Solve what these programs can solve
  - Do small experiments on single node (possibly with multiple GPUs)
  - Devote your life to the development of good scalable code for your problem

Not quite ideal. Major obstacle for the advance of computational science

Is there any solution?

- **Ideally, what we want is one simulation code which can handle all possible particle-based simulations on all possible computer systems.**
- **This is certainly impossible. No single group can implement**
  - **all possible problems**
  - **all necessary integration methods**
  - **etc etc...**

One might be able to:

- **Provide templates, (or DSL) using which researchers describe the problems they want to solve.**
- **Then the “compiler” generate the reasonably scalable and efficient code to solve that particular problem.**

To be more specific:

Particle-based simulations includes:

- **Gravitational many-body simulations**
- **molecular-dynamics simulations**
- **CFD using particle methods(SPH, MPS, MLS etc)**
- **Meshless methods in structure analysis etc (EFGM etc)**

Almost all calculation cost is spent in the evaluation of interaction between particles and their neighbors (long-range force can be done using tree, FMM, PME etc)

Our solution

Users specify the problem by providing the definition of particle class and the particle-particle interaction function.

Our software generates functions for

- domain decomposition (with load balance)
- particle migration
- interaction calculation (and necessary communication)

that can be called from user-written programs.

Actual “generation” is done using C++ templates.

FDPS (Framework for Developping Particle Simulators)

Initial release

Iwasawa+2016 (PASJ 2016, 68, 54+arxive 1601.03138)

- **Publicly available**
- **A single user program can be compiled to single-core, OpenMP parallel or MPI parallel programs.**
- **Parallel efficiency is very high**

Tutorial
FDPS Github: https://github.com/FDPS/FDPS

Getting FDPS and run samples

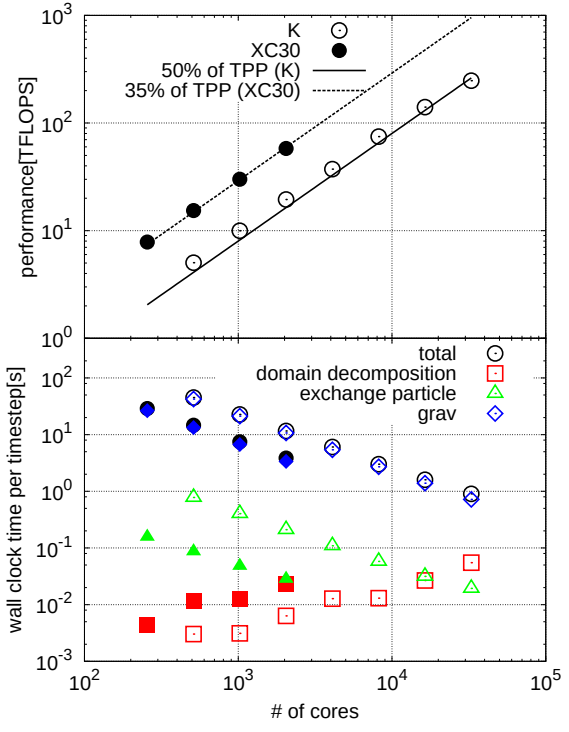
```
> git clone git://github.com/FDPS/FDPS.git
> cd FDPS/sample/c++/nbody
> make
> ./nbody.out
```

To use OpenMP and/or MPI, change a few lines of Makefile

Important points for users

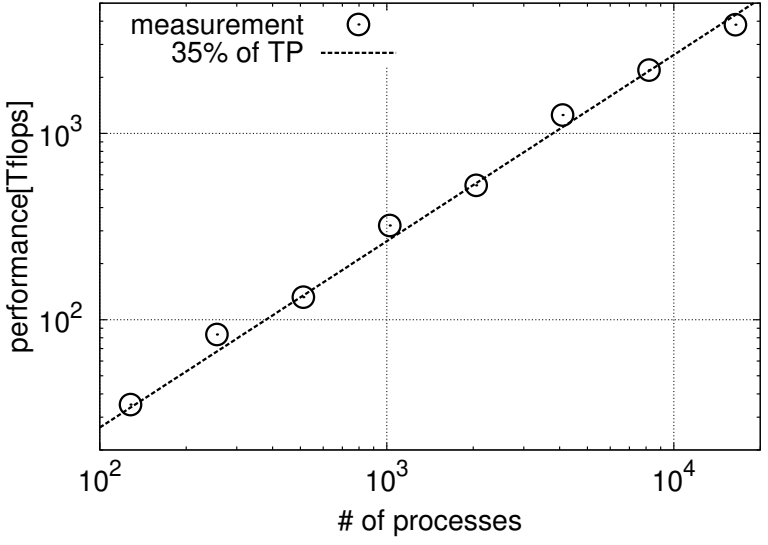
- **Users who know nothing about MPI can still write their programs using FDPS and compile them to highly scalable and efficient MPI programs.**
- **Single source code can be used on all platforms, from single PC to entire Fugaku (currently accelerators still need additional care).**

Performance examples



Strong scaling with 550M particles
Measured on both K computer and Cray XC30 at NAOJ
Gravity only, isolated spiral galaxy
scales up to 100k cores
30-50% of the theoretical peak performance

Performance (and tuning) of FDPS on Sun-way TaihuLight



Nearly 4PF on 1/10 of TaihuLight (Implemented many of the new algo-rithms which will be discussed later today)

History since 2016

- 2016/1 V2.0: supper of accelerators such as GPGPU
- 2016/12 V3.0: Fortran API
- 2017/11 V4.0: Several new functions
- 2018/11 V5.0: C API
- 2020/8 6.0: addition of PIKG automatic kernel generator
- 2021/8 7.0: Support of polar/cylindrical coordinates, support of multiple FDPS instances, improved communication algorithms

More to come (already in experimental code) : hiding communication, support of Particle-Mesh Multipole Method.

Today we discuss some of additions in Version 5.0 and later.

C API

- FDPS is written in C++, to take advantage of its template functions. r
- Not every researchers are familiar with C++. Some uses Fortran (for good reasons), some C, and some other languages (Julia, Rust, Crystal, ...)
- We now provide C API (accepts C struct and interaction functions with C bindings), which can be used to develop APIs in other languages.
- Fortran API was developped earlier. C API is actually more like exposing the internals of Fortran API

PIKG

- **What has been lacking in FDPS: Automatic way to generate high-performance interaction kernels.**
- **Parallelization is taken care by FDPS, but SIMD tuning has been the responsibility of the users.**

PIKG generates high-performance kernels for a variety of architectures (now AVX2, AVX512, SVE and Cuda) for you.

PIKG example

From FDPS sample code in sample/c++/nbody

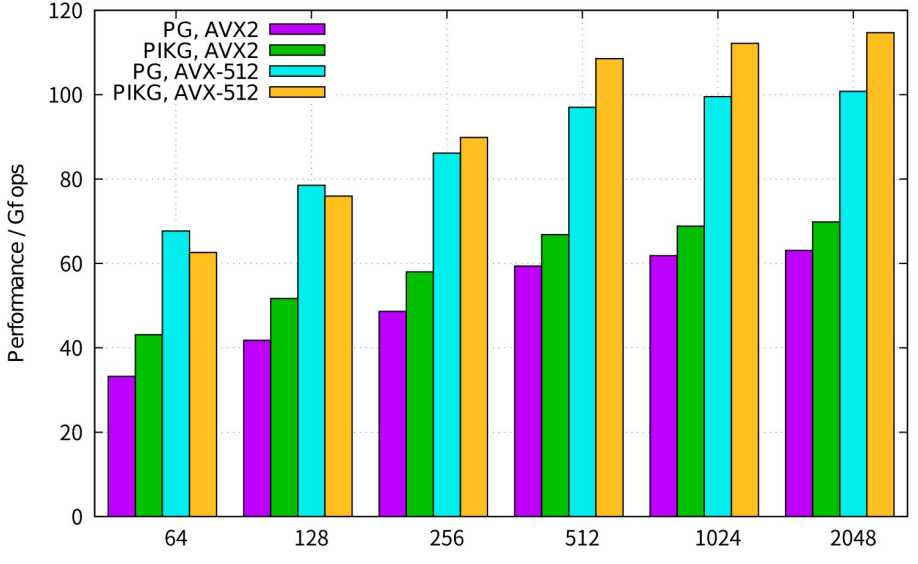
```
EPI F32vec xi:pos    #i-particle variable type, name, name in FDPS particle class
EPJ F32vec xj:pos    #j-particle variable type, name, name in FDPS particle class
EPJ F32    mj:mass   #j-particle variable type, name, name in FDPS particle class
FORCE F32vec acc:acc #name of variable  in the struct to return result
FORCE F32    pot:pot #name of variable  in the struct to return result

F32 eps2             #shared variable (like static member)

rij    = xi - xj     #vector arithmetic operations are pre-defined
r2 = rij * rij + eps2
r_inv  = rsqrt(r2)   #inverse-square-root and many mathmatical functions are predefined
r2_inv = r_inv * r_inv
mr_inv  = mj * r_inv
mr3_inv = r2_inv * mr_inv
acc -= mr3_inv * rij # FORCE variables are used to accumulate results
pot -= mr_inv
```

For complete documentation and source, visit https://github.com/FDPS/PIKG

PIKG performance example



Comparison with Phantom GRAPE(PG) (by Dr. Yoshikawa of Tsukuba)

AVX2 and AVX512

64-2048: Number of particles to share the same interaction list

Performance on Fugaku

Asymptotic single core performance of interaction kernels

Kernels	# of operations	asymptotic speed	efficiency
Gravity(monopole)	27	35.8 Gflops	27.9 %
Hydro density/pressure	67	32.2 Gflops	25.2 %
Hydro force	80	21.6 Gflops	16.8 %
Hydro time deriv	103	27.7 Gflops	21.7 %

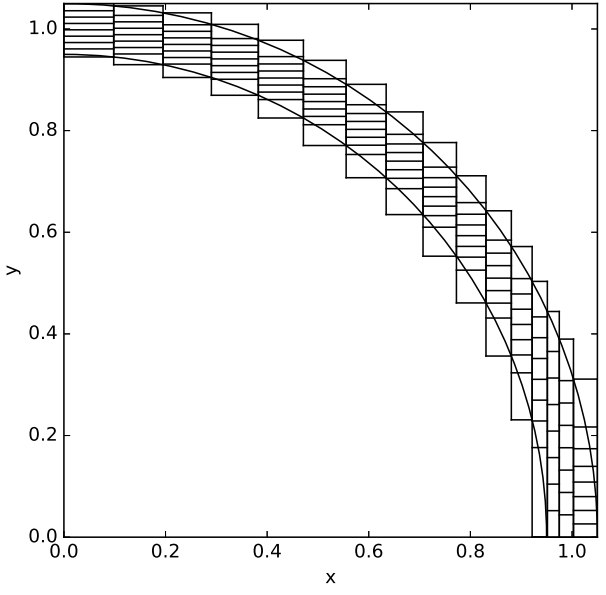
For comparison, the world's fastest gravitational interaction kernel on A64fx (K. Nitadori, 2021) https://www.slideshare.net/RCCSRENKEI/12-a2021-249556559

63.6GF (89.5GF with 38 operations/interaction)

Our current approach is based on loop fission. Nitadori's relies on software pipelining and ... manual rewrite of generated assembly code.

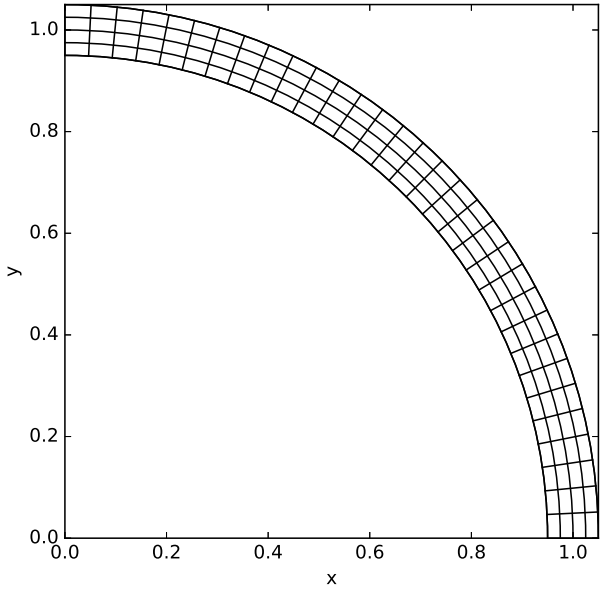
(On Intel Xeon the performance gap is not this large)

Polar coordinates — the problem



The Cartesian tree and domain decomposition in Cartesian coordinate are not ideal for narrow rings. For domain decomposition, some domains can have extremely long shape (for example containing two areas one with $y > 0$ and the other with $y < 0$.

Polar coordinates — why we need it



If we can use polar (or cylindrical) coordinates, the shape of domains are always good.
We need only one change in FDPS, and several changes in the user code.

Polar coordinates for tree and domain decomposition

- use cylindrical coordinates (or for wide disks, polar coordinates with $\log r$ for radius) for domain decomposition and tree
- For tree traversal, the distance is calculated as the usual Cartesian distance, even though actual coordinates are polar, with the periodic correction for θ.
- Since what we need is the opening criterion and not the exact distance, this scheme works fine.
- The interaction calculation is done in the original Cartesian coordinates.

Change in FDPS

Introduce the following template parameter for TreeForForce class:

```
enum CALC_DISTANCE_TYPE{
    CALC_DISTANCE_TYPE_NORMAL = 0,
    CALC_DISTANCE_TYPE_NEAREST_X = 1,
    CALC_DISTANCE_TYPE_NEAREST_Y = 2,
    CALC_DISTANCE_TYPE_NEAREST_XY = 3,
    CALC_DISTANCE_TYPE_NEAREST_Z = 4,
    CALC_DISTANCE_TYPE_NEAREST_XZ = 5,
    CALC_DISTANCE_TYPE_NEAREST_YZ = 6,
    CALC_DISTANCE_TYPE_NEAREST_XYZ = 7,
};
```

(Not sure why we need this in addition to `enum BOUNDARY_CONDITION`**...)**

Change in the user code

- **FullParticle should contain positions in two coordinates. Polar one should be used for getPos.**
- **MomentMonopole, MomentQuadrupole, SPJMonopole and SPJQuadrupole classes need to be replaced with the user-defined ones with two positions in two coordinates.**
- **positions in polar coordinates should be updated before domain decomposition and tree updates.**
- **For collision detection, care should be taken on which distance is used when. (treewalk and force calculation uses different distances)**

Example (particle class)

(will soon appear in https://github.com/jmakino/nbody-with-center. A different version will be in sample/c++/planetary-ring)

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec pos_car;
    PS::F64vec vel;
    ...
    PS::F64vec getPos() const {
        return pos;
    }
```

```
void ctod()
{
    const auto cth = cos(pos.x);
    const auto sth = sin(pos.x);
    const auto r = pos.y;
    const auto pos_x = r*cth;
    const auto pos_y = r*sth;
    pos_car= PS::F64vec(pos_x, pos_y,
}
```

(Probably the use of `pos` and `pos_polar` would be better...)

Example (main part of user code)

```
PS::DomainInfo dinfo;
...
dinfo.setBoundaryCondition(PS::BOUNDARY_CONDITION_OPEN);
dinfo.setPosRootDomainX(-MY_PI, MY_PI);
...
using Tree_t = PS::TreeForForce<PS::SEARCH_MODE_LONG_SCATTER, FPGrav, FPGrav,
              FPGrav, MyMomentMonopole, MyMomentMonopole,
              MySPJMonopole, PS::CALC_DISTANCE_TYPE_NEAREST_X>;
...
Tree_t tree_grav;
tree_grav.calcForceAllAndWriteBack(CalcForceEp<FPGrav>,
           CalcGravitySp<MySPJMonopole>, system_grav,  dinfo);
```

Moment classes and interaction calculation functions should also be changed.

LET communication method

- **MPI_ALLTOALLV has been used for the exchange of the "local essential tree" (the information of the tree of sender domain necessary in the receiver domain)**
- **MPI_ALLTOALLV can be very time-consuming for large number pf MPI processes**
- **Added other ways of LET exchange.**

Use `PS::TreeForForce::setExchagneLETMode` **to select.**

MPI communicator

- **Can set MPI communicators for domain, particlesystem and tree classes.**
- **Can have multiple systems in one MPI run.**

Up to this function is in FDPS 7.0

hiding communication

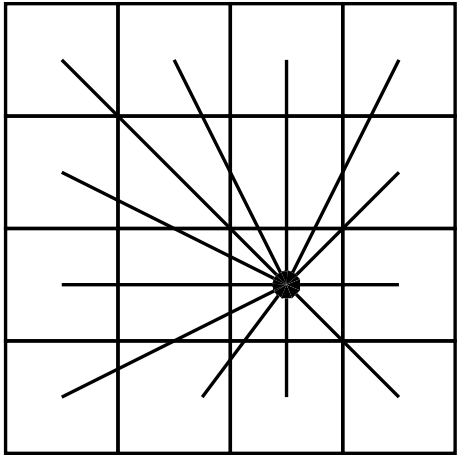
- **Split interaction calculation to internal and external contributions.**
- **External contribution is much less expensive than internal contribution, at least when the number of particles is large.**
- **Do all communications for external interaction calculation while performing internal calculation**
- **Quite important on Fugaku**

Particle-Mesh Multipole Method(1)

"The best from the Two Worlds"
Basic idea (Nitadori 2014)

- **Stop making high-level nodes for FMM at certain level and apply (conceptually) $O(N^2)$ calculation at that level.**
- **Using the fact that this $O(N^2)$ operation can be regarded as convolution of multipole expansions and Green's functions, apply FFT and reduce calculation cost to $O(N \log N)$**

Particle-Mesh Multipole Method(2)



Conceptually, local expansion L of cell i, j can be obtained as convolution of multipole expansion of other cells and Green's function

$$L_{i,j} = \sum_{k,l} M_{k,l} G_{i-k,j-l}$$

This can be done using FFT

Advantage of PM^3 and current status

Advantages:

- **Over PME: Higher accuracy with less calculation cost (existing drawback same as that of FMM, though)**
- **Over FMM: natural periodic boundary, elimination of steps with low parallel efficiency. Open boundary might be possible with Vico-Greengard-Ferrando method.**

Current status:

Implemented on experimental code for Fugaku. Will be available on the next update.

Summary

- **We have been improving FDPS, and hopefully will be doing so even after our team in R-CCS disappear.**
- **New features: Automatic interaction kernel generation for AVX(s), SVE and Cuda, Many optimizations for Fugaku**
- **Things to come: communication hiding, PM^3 etc.**