Current status of FDPS and Future plans

Jun Makino
R-CCS Particle Simulator Research Team/Kobe University

FDPS team and collaborators

R-CCS
(Particle Simulator Research Team)
Daisuke Namekata
Miyuki Tsubouchi
Jun Makino

Kobe U
Masaki Iwasawa
Kentaro Nomura

JAMSTEC
Natsuki Hosono

U. Tokyo
Ataru Tanikawa
Long Wang
Yota Ishigaki

NSC in Wuxi
(Implementation on Sunway TaihuLight)
Zhao Liu
Haohuan Fu
Guangwen Yang

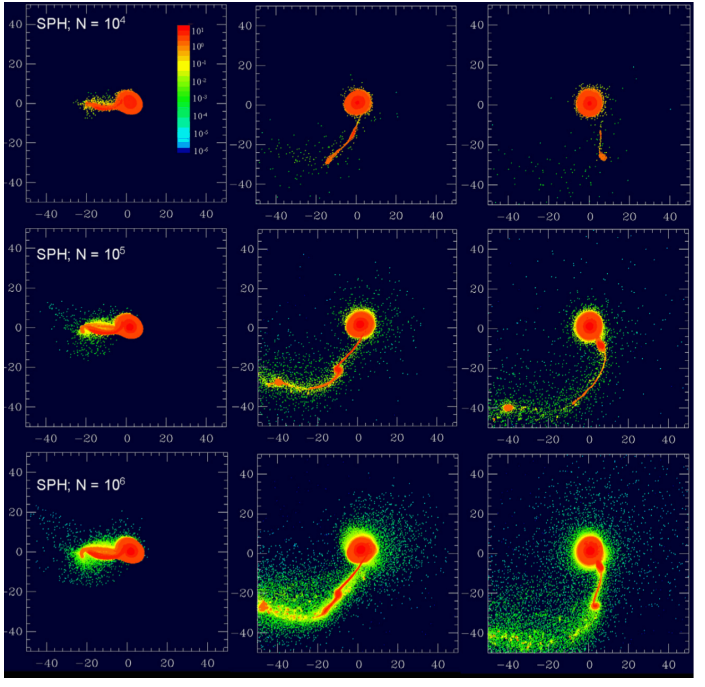
and many others

Talk plan

1. What we want to do when writing particle-based simulation codes. (or any other large-scale HPC code)
2. What should be done?
3. Design of FDPS
4. Performance
5. Future plan

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)

What we want to do

- **Write a simple program expressing the numerical scheme used**
- **Run it on notebooks, desktops, clusters and large-scale HPC platform**

What we are doing now

- rewrite the entire program using MPI to make use of multiple nodes.
- apply complicated optimizations to hide interprocessor communications.
- rewrite data structure and loop structure to make efficient use of data caches.
- rewrite inner loops and data structure to let compilers make use of SIMD instruction sets.
- apply machine-specific optimizations or write codes using machine-specific languages (Cu**, Open**).

Existing efficient codes

Astrophysics

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

Molecular Dynamics
GENESIS, GROMACS, LAMMPS, Modylas, NAMD, and several others

Developers need to write codes for domain decomposition, particle move, and interaction calculation.

Our solution

If we can develop a program which can generate a highly optimized MPI program for

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

for a given particle-particle interaction, that will be the solution.

Design concept

- API defined in C++
- Users provide
  - Particle data class
  - Function to calculate particle-particle interaction

  Our program generates necessary library functions. Interaction calculation is done using parallel Barnes-Hut tree algorithm
- Users write their program using these library functions.

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

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**
- **As of version 3.0 (released 2016) GPUs can be used and user programs can be in Fortran (and now also pure C).**
- **Versions 4.0 and 5.0 offers many performance improvements.**

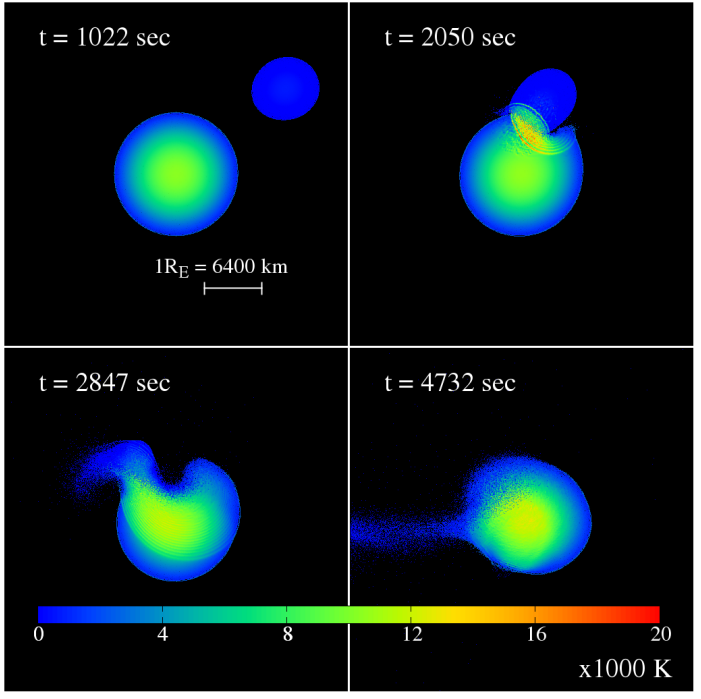
Tutorial
FDPS Github: https://github.com/FDPS/FDPS

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

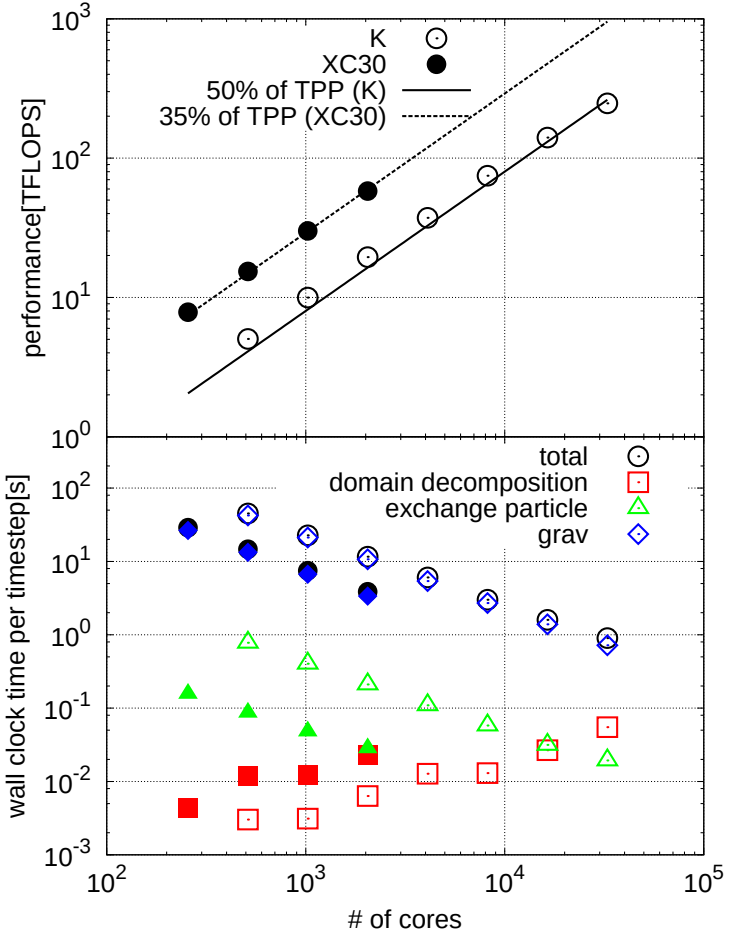
Example of calculation



Giant Impact calculation (Hosono et al. 2017, PASJ 69, 26+)
Figure: 9.9M particles
Up to 2.6B particles tried on K computer

animation Terrestrial magma ocean origin of the Moon, Hosono et al. Nature Geoscience volume 12, 418423(2019)

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

Preparation for Fugaku and next-generation platforms

Compared to K computer, future machines will have

- **Larger number of cores, more FPUs per core**
- **(relatively) weak main memory**
- **(relatively) weak network**

Our software should be ready for such machines to be useful.

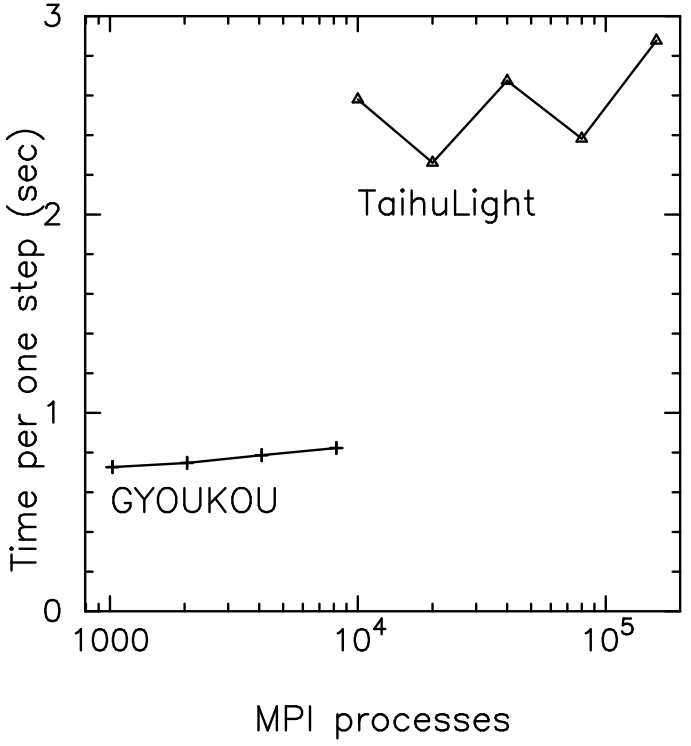
How to be “future-proof”?

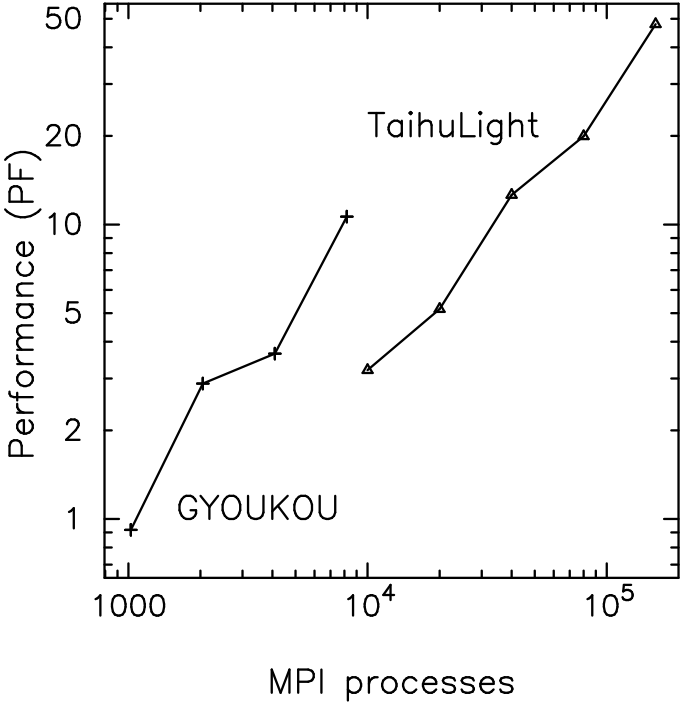
- **One possibility: try to use machines with weak main memory and weak network.**
- **Examples as of 2015-2020: Sunway Taihulight and PEZY-SC2(GYOUKOU)**

Our current implementation

- Elimination of all-to-all communications and other operations with the cost proportional to the number of nodes.
- Use the same “interaction list” for multiple timesteps (similar to “bookkeeping” or “pairlist” method
- Minimize the main memory access within FDPS (tree construction etc)
- Minimize internode communication
- manual tuning of interaction kernels

Achieved performance





30-40% of the theoretical peak on both machines.
10M particles/MPI process
Planetary ring simulation

Future plan

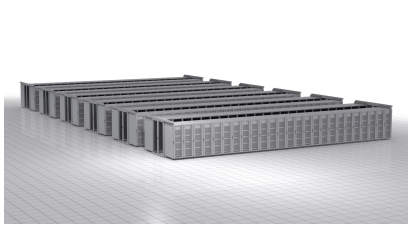
Current status of FDPS:

- **Users do not need to parallelize their codes by themselves. FDPS does it.**
- **Optimization of the interaction kernel should still be done by users.**

What should be done

User: writes high-level description of interaction kernel
FDPS: generates optimized code for supported architectures

- **Fugaku**
- **x86**
- **NVIDIA/AMD GPGPUs**
- **MN-Core**
- **(FPGA)**
- **...**

Example of high-level description

```
EPI F32vec xi:pos
EPI F32    eps2i:eps2
EPJ F32vec xj:pos
EPJ F32    mj:mass
EPJ F32    eps2j:eps2
FORCE F32vec f:acc
FORCE F32 phi:pot
rij = xi - xj
r2 = eps2i + eps2j + rij*rij
rinv = rsqrt(r2)
mrinv = mj*rinv
f -= mrinv*rinv*rinv*rij
phi -= mrinv
```

Example of high-level description (2)

```
EPI U32    ti:type
EPI F32vec ri:pos
EPI F32vec vi:vel
EPI F32    hi:h

EPJ U32    tj:type
EPJ F32vec rj:pos
EPJ F32vec vj:vel
EPJ F32    mj:mass
EPJ F32    uj:eng
FORCE F32 dens:dens
FORCE F32 pres:pres
FORCE F32 gradh:gradh
FORCE F32 divv:divv
FORCE F32 rotv:rotv
F32 gamma
function gradW(dr,h)
  r = sqrt(dr*dr)
  u = r*inv(h)
  p1u = 1.0f - u
  h2 = h*h
  h5 = h2*h2*h
  coeff = 1155.0f*inv(12.5663706144f*h5)
  p1u2 = p1u*p1u
  p1u5 = p1u2*p1u2*p1u
  return - coeff*p1u5*(1.0f + 5.0f*u)
end
```

```
function W(r,h)
  u = r*inv(h)
  p1u = max(0.0f, 1.0f - u)
  coeff = 495.0f*inv(100.530964915f*h*h*h)
  p1u2 = p1u*p1u
  p1u6 = p1u2*p1u2*p1u2
  return coeff*p1u6*(1.0f + u*(6.0f + 11.6666666667f*u))
end

function dwdh(r,h)
  u = r*inv(h)
  p1u = max(0.0f, 1.0f - u)
  h2 = h*h
  coeff = 165.0f*inv(100.530964915f*h2*h2)
  p1u2 = p1u*p1u
  p1u5 = p1u2*p1u2*p1u
  return -coeff*p1u5*(9.0f + u*(45.0f + u*(-5.0f - 385.0f*u)))
end
g1u = gamma - 1.0f
dr = ri - rj
dv = vi - vj
if tj > 0.0
  rij = sqrt(dr*dr)
else
  rij = 2.0f*hi
endif
```

```
wij = W(rij,hi)
dens += mj*wij
pres += g1u*mj*uj*wij
dwdh_ij = dwdh(rij,hi)
gradh += g1u*mj*uj*dwdh_ij
divv -= g1u*mj*uj*(dr*dv)*gradW(dr,hi)
tmp.x = dv.y*dr.z - dv.z*dr.y
tmp.y = dv.z*dr.x - dv.x*dr.z
tmp.z = dv.x*dr.y - dv.y*dr.z
rotv -= g1u*mj*uj*tmp
```

What need to be done

- (For CPUs) AoS/SoA conversion, use of SIMD intrinsics, loop fission, stripmining, unrolling, software pipelining...
- (For GPUs and other accelerators) generation of kernel code and codes for communication between CPU and GPU.

Current status:

- Automatic generation for Fugaku is working.
- The generated code is currently slightly (10-15%) slower than our best hand-optimized code.

Summary

- FDPS is a framework which helps the development of scalable high-performance particle-based simulation codes
- It can be used with arbitrary particle structure and particle-particle interaction function
- Efficiency of application written using FDPS is very high on large-scale machines including K and Sunway Taihulight. (30-50% of the peak)
- We are working on automatic generation of highly optimized kernel for particle-particle interactions, and that for Fugaku is almost ready
- FDPS will be used in various projects on Fugaku