Debugging the performance problems of FDPS on Supercomputer Fugaku

Jun Makino

Internal Seminar Sept 22, 2021

Talk plan

1. Problems of FDPS-based programs on Fugaku
2. Result of (somewhat) detailed measurements
3. Details of problems and solutions
4. Current status
5. Summary

Problems of FDPS-based programs on Fugaku

There seem to be two problems

- **Runs very slowly on Fugaku**
- **Stops seemingly randomly after relatively small number of timesteps on Fugaku**

In today's talk I'll focus on the first problem, since the code I used to evaluate the behavior of Fugaku turned out to run stably for hundreds of thousands of steps.

Motivation

Understand why the performance of FDPS-based program (for example GPLUM) is not very good on Fugaku.

- **GPLUM performance on Cray XC40 (Skylake): ~ 20 sec with 1040 cores and 10^6 particles, 1 Kepler time (~ 400 steps) (largest number of cores tested)**
- **on Fugaku: ~ 15 sec with 64 nodes (longer for more nodes)**
- **Not that Fugaku is slow, but we want to improve the calculation speed using more nodes.**

Measurement/Debugging setup

- **Code to measure:** `nbody-with-center` **(https://github.com/jmakino/nbody-with-center) My version of planetary ring code. Now uses cylindrical coordinate of FDPS.**
- **To use simple code which I fully understand.**
- **Number of nodes: 1k-4k**
- **Number of Particles: 2-200M**
- $\theta = 0.5$**, ring width=0.1, domain decomposition/4steps**

First measurement

Calculation time per one timestep

Item	time(ms)
Total	105
Domain Decomposition	1.5
Exchange Particle	10.5
Force calculation	54
Others	38

- Everything looks verrrry sloooow. (we are using an extremely large number of cores though: around 50k)
- This is measurement done in the caller of FDPS functions, not in FDPS. Should use the internal timeprofile class as well.

Changes made at this stage

- **Exchange_LET communication mode changed to `P2P_EXACT`**
- **interaction calculation function SIMDized (not using PIKG yet, though).**

External times

Item	time(ms)
Total	59
Domain Decomposition	1.7
Exchange Particle	8.4
Force calculation	44
Others	4.6

Internal times

Item	time(ms)
Domain Decomposition	3.6
Exchange Particle	22
Local/Global tree	9.4/9.5
Calc force	3.2
LET const/comm	6.5/15

Now everything except `calc_force` is slow...

Details of problems and solutions

- **Exchange_Particle**
- **LET construction/communication**
- **Tree construction**
- **Domain Decomposition**

Exchange_Particle

- **This is a ring code with cylindrical coordinates**
- **So an MPI process need to communicate only with its neighbor processes**
- **However, it actually communicated with all nodes in θ direction**
- **Turned out that the exchange particle function was not changed appropriately to handle periodic boundary. (Actually, a single algorithm could handle both cases, but current one is not that one)**

After the modification of a single line of code, time for exchange particle changed:

$$22\text{ms} \rightarrow 2.2\text{ms}$$

LET construction/communication

- `P2P_EXACT` **algorithm is not good for ring.**
- **Switched to** `P2P_FAST` **algorithm. Error is slightly larger but hopefully acceptable.**

LET construction time: $6.5\mathrm{ms} \rightarrow 3.5\mathrm{ms}$
LET communication time: $15\mathrm{ms} \rightarrow 0.3\mathrm{ms}$

Tree construction

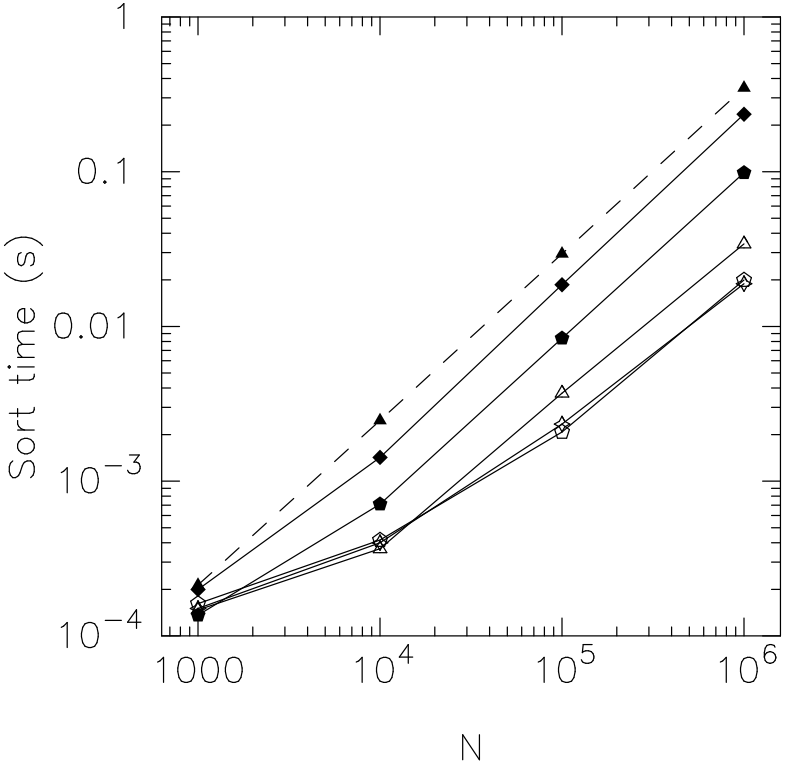
- **The parallel merge sort algorithm used in FDPS turned out to be very slow on Fugaku.**
- **First, I just activated the flag to use single-core** `std::sort`**.**

  Local tree construction time: $9.4\mathrm{ms} \rightarrow 0.56\mathrm{ms}$
- **I also implemented parallel sample sort**

  Local tree construction time: $0.56\mathrm{ms} \rightarrow 0.44\mathrm{ms}$

  (Much larger speedup for large N**)**

Performance of parallel sample sort



Wallclock time to sort structs of size 160bytes using 64-bit keys.

The dashed curve with filled triangles is the time for std::sort. Filled squares, pentagons, open triangles, squares and pentagons are the results of samplesort_bodies called with 2, 4, 12, 24, 48 threads.

code available at: https://github.com/jmakino/sortlib

Domain Decomposition

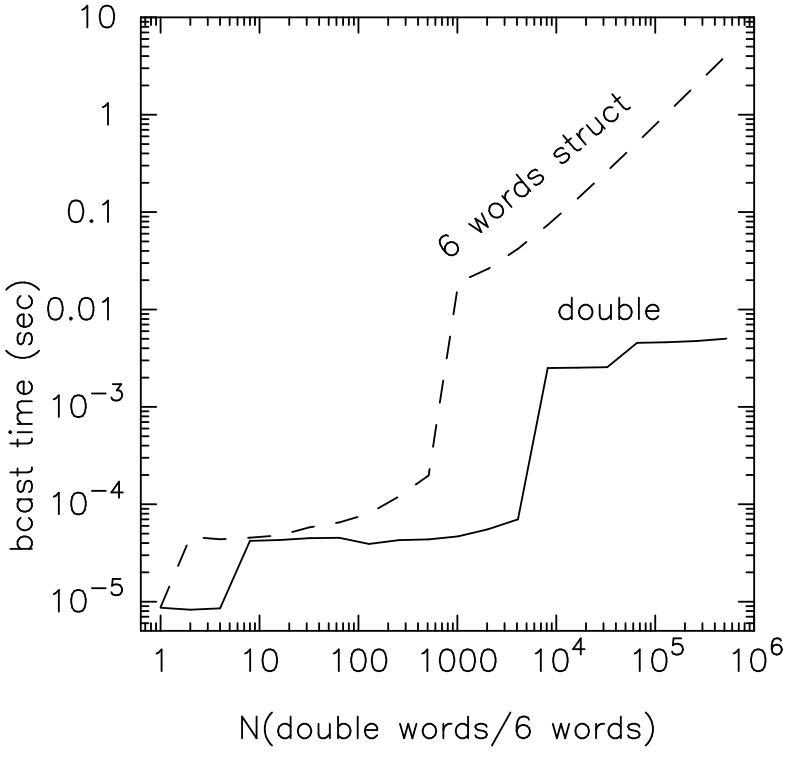
After inserting lots of barriers and measurement functions, it turned out

- **Collection of sample particles (implemented using MPI_AllgatherV) is very fast**
- **Actual calculation (done in rank 0) is not very fast**
- **Broadcast of calculated domains (using MPI_Bcast) is incredibly slow, something like 40ms/call. Seemingly a performance bug of MPI itself.**

Current hack: Avoid the use of MPI_Bcast and do redundant calculations on all nodes. Also, use parallel sort and OpenMP parallelization whenever appropriate

Domain Decomposition time: $8\text{ms} \rightarrow 0.4\text{ms}$

Fugaku MPI_Bcast performance



1024 nodes, 48 threads/process

6 words= sending structs consisting of 6 double-precision words

With a reasonalbe MPI implementation, sending N structs should take the time same as that for sending $6N$ double-precision words.

It takes actually much, much more time....

Current Status

Item	time(ms)
Total	7.2
Domain Decomposition	0.41
Exchange Particle	0.78
Force calculation	5.8
make tree	1
make LETs	1.6
exchange LETs	0.4
treewalk	0.44
force kernel	1.57
Others	0.1

Current performance for larger N

N	time per step(ms)
10^6	7.2
10^7	16.6
10^8	133.2

(Un)expected Findings

- **When carefully used, communication on Fugaku is pretty fast.**
- **”Carefully” means: measure everything and work around bugs, and also avoid communicating with many processes (large overhead)**
- **Intra-node parallel calculation can sometimes be surprizingly slow.**
- **Even so, single-core parts can easily form bottlenecks.**

Things to do

- **Domain decomposition: Should switch to $O(p^{1/3})$ algorithm from current $O(p)$ algorithm (p: number of processes)**
- **Exchange Particles: "find particle" part seems to be still slow**
- **Tree construction: time for parts other than sort can be improved**
- **LET construction: tree walk is still very slow.**
- **force calculation: should use PIKG or hand-optimized kernel**

Summary

- **Performance of FDPS on modestly large number of nodes (1024) of Fugaku was investigated for a “small” number of particles (2M).**
- **There turned out to be many inefficient parts not found previously, mainly because large-number-of-nodes, small-number-of-particles calculation has not been tried (or investigated deeply)**
- **So far, wall clock time per timestep has been reduced from 59ms (after the force kernel is SIMDized) to 7.2ms.**
- **I hope to make this version available as FDPS 7.1 soon.**
- **I believe further improvement, down to 3-4ms, should be possible.**