Dancing with NBODY6: A Development of a GPU module for NBODY6

Keigo Nitadori

RIKEN AICS

keigo@riken.jp

What's NBODY6/GPU

- NBODY6 is
  - an *N*-body code to simulate star clusters (from NBODY0 to NBODY7)
  - developed by Sverre J. Aarseth of IoA Cambridge
  - Used by several groups world-wide
- NBODY6/GPU is
  - a GPU acceleration module for NBODY6
  - developed by Sverre Aarseth & Keigo Nitadori since 2008

Today I talk on…

- The history, design and implementation of NBODY6/GPU
  1. About NBODYx
  2. On Ahmad-Cohen neighbor scheme
  3. Development of the GPU module
  4. API and functionality of the module

Honestly…

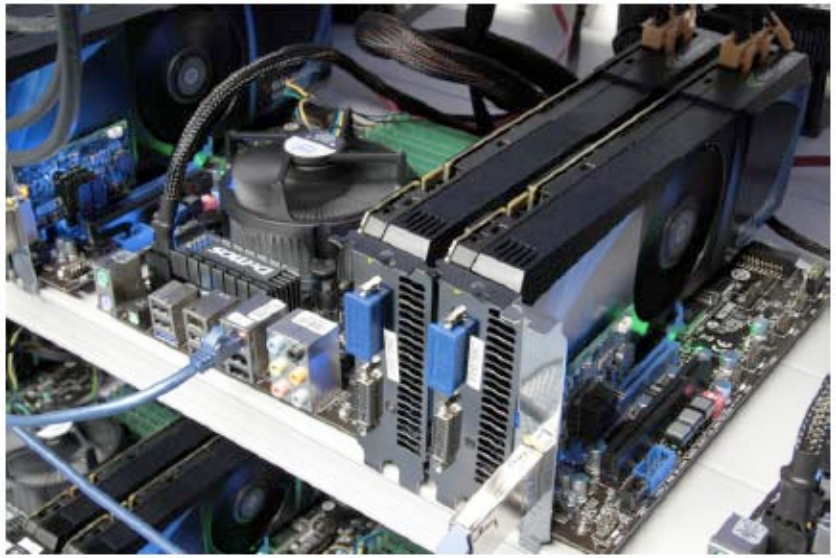
- NBODY6 is a SPPAGETY code
  - I can’t read nor modify!
    - FORTRAN 77 or earlier
    - Everything is ‘procedure’
    - Huge common block and many goto
  - How we could do that?
    - It strongly owes to the patience and high-motivation of Sverre Aarseth

Example

```
      IF (J.GT.N) THEN
           TPRED(J) = -1.0
           CALL JPRED(J)
           J1 = 2*(J - N) - 1
           IF (LIST(1,J1).GT.0) THEN
               NP = LIST(1,J1)
               DO 44 LL = 2,NP+1
                    JJ = LIST(LL,J1)
                    TPRED(JJ) = -1.0
                    CALL JPRED(JJ)
44             CONTINUE
           END IF
       END IF
```

What' a GPU?

- Graphical Processing Unit
  - Originally, for the acceleration of 3D games on PC
  - Recently, used for the acceleration of more general computations
  - Can be used as a *generic* of GRAPE series (GRAvity PipE, accelerators for gravitational *N*-body problems)



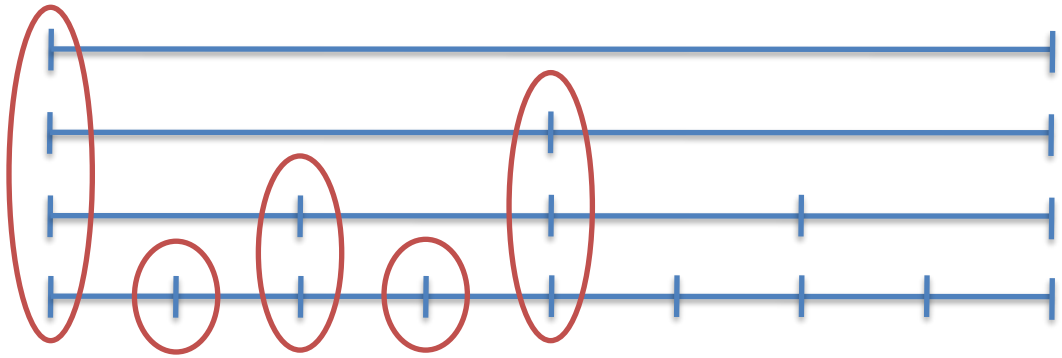
NBODYx ($0 \leq x \leq 7$)

	Integrator	Singularity	Neighbour scheme	GRAPE
NBODY1	Adams	Softening	No	(No)
NBODY2	Adams	Softening	Yes	No
NBODY3	Adams	MREG	No	No
NBODY4	Hermite	MREG	No	Yes
NBODY5	Adams	MREG	Yes	No
NBODY6	Hermite	MREG	Yes	No

- NBODY0 (educational), NBODY1h, NBODY7 (with PN)
- NBODY4 + GRAPE-6 has been mostly used recently

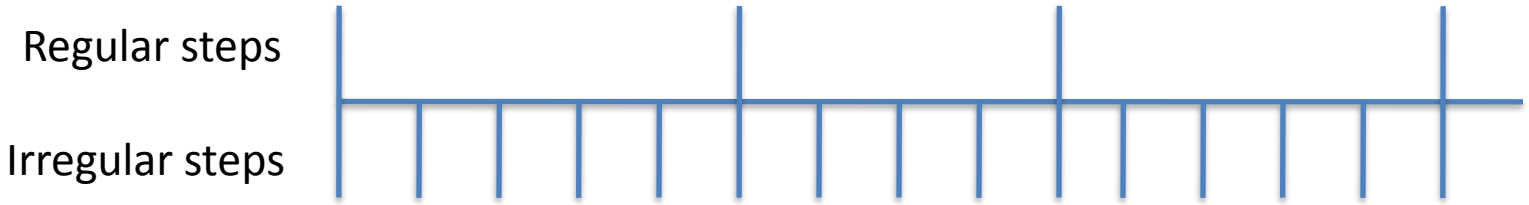
Individual timestep scheme

- Not just a variable timestep for the ODE (ordinary differential equation) integrator
- Allows EACH particle to have ITS OWN time-step
- For the positions of the other, use predictor
- Timestep restriction to 2^n was introduced later



Ahmad-Cohen neighbor scheme

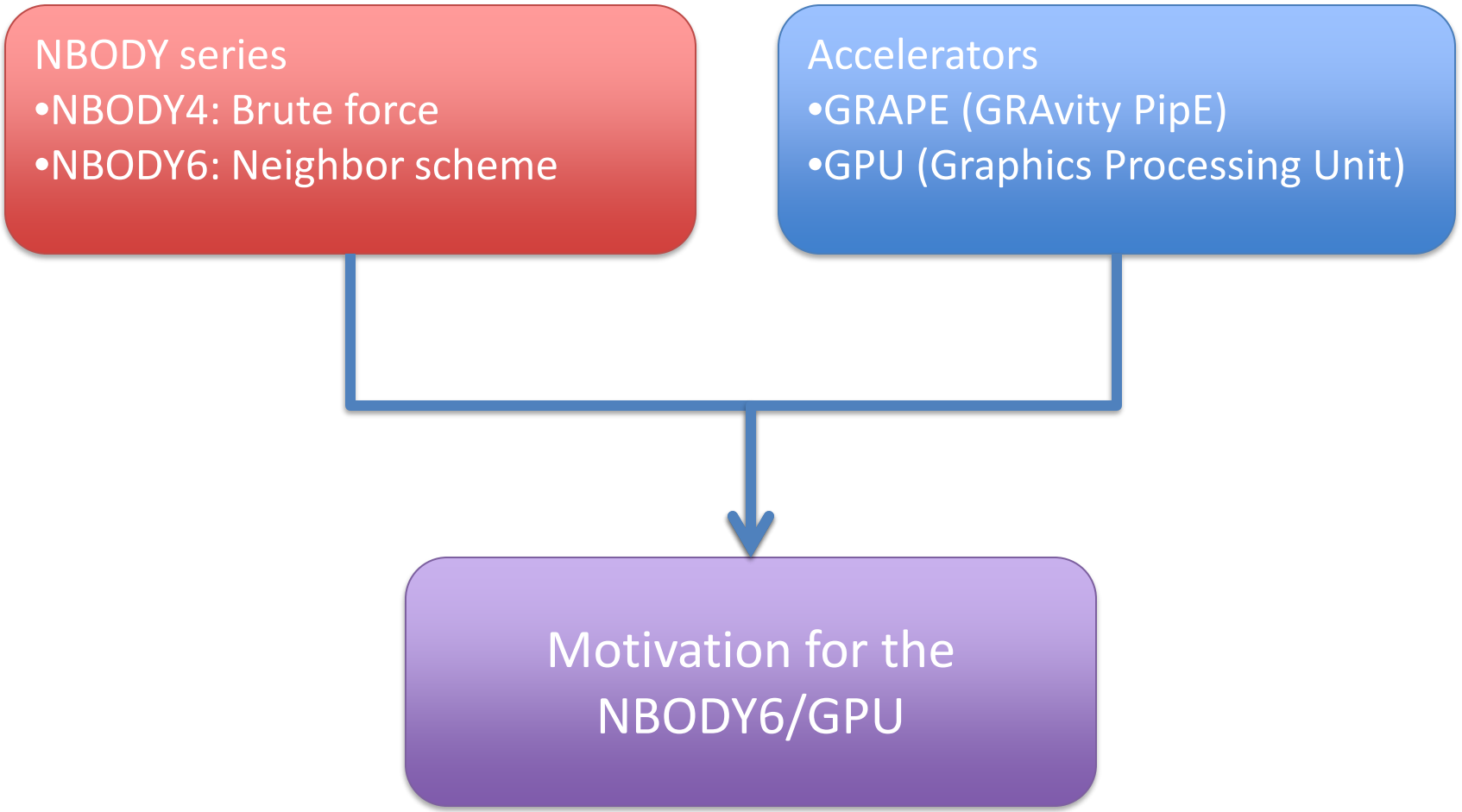
- Only the force from the neighbor stars are evaluated frequently (Irregular-step)
- At the large Regular-step, the total force from all the stars are evaluated and the neighbor-list is updated
- Mention as the most efficient method for serial implementations for CPUs



So what?
(Dakara nani-ga iitai-no, kimi-wa?)

- Only the brute-force method (however, with the individual timestep scheme) has been accelerated with GRAPE (NBODY4)
- Accelerating NBODY6 with the AC neighbor scheme using GPU is a CHALLENGE
  - Only the serial CPU implementation of NBODY6 had existed
  - There had been a trial to accelerate NBODY6 with GRAPE-6 (failed)
  - GPU is programmable!
  - Sverre loves AC scheme

Review


NBODY series
•NBODY4: Brute force
•NBODY6: Neighbor scheme
Accelerators
•GRAPE (GRAvity PipE)
•GPU (Graphics Processing Unit)
Motivation for the NBODY6/GPU

Mathematical view of the AC neighbor scheme

- Integrating a particle is integrating the polynomial

$$\Delta v_i = \int_t^{t+\Delta t} f_i(t)dt$$

- The polynomial is a summation

$$f_i(t) = \sum_{j \neq i}^{N} f_{ij}(t)$$

- that can be split into two terms

$$f_i = f_{i,reg} + f_{i,irr} = \sum_{\{\, j | \Delta t_{ij} \leq \Delta t_{i,reg} \}} f_{ij} + \sum_{\{\, j | \Delta t_{ij} > \Delta t_{i,reg} \}} f_{ij}$$

cont.

- Time scale of each interaction might be evaluated from the hardness of the polynomial

$$\Delta t_{ij} \propto T\left(f_{ij}(t)\right)$$

Paradox?（笑えない冗談）

- It’s safe to define

$$\Delta t_i = \min_{j \neq i} \Delta t_{ij}$$

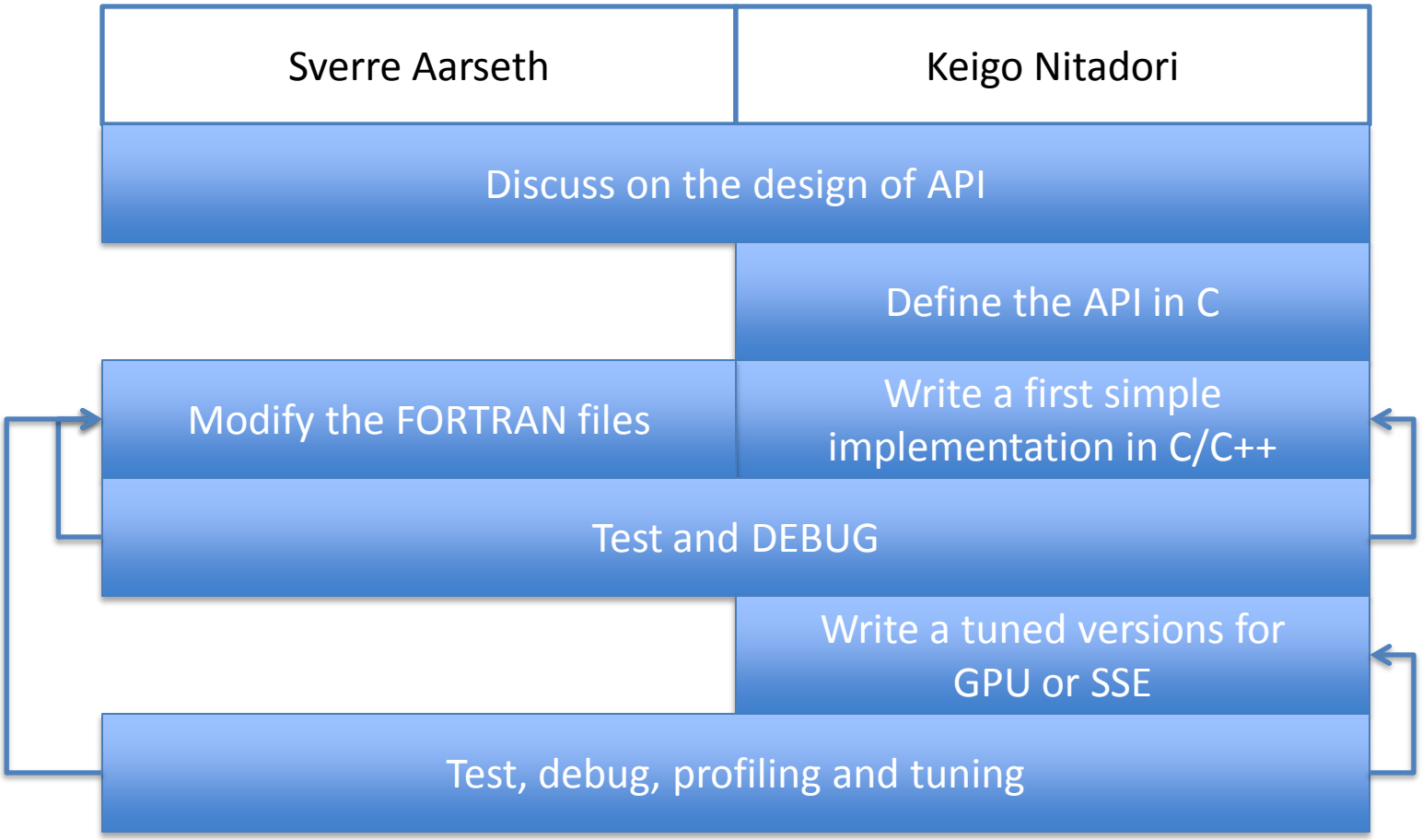
$$\Delta t_{ij} = \min(\Delta t_{i,} \Delta t_j)$$

  if we can define the timescale both on a particle and an interaction
- It goes back to the shared timestep scheme

Making a plug-in for NBODY6

- ~~NBODY6 is fully modularized~~
- NBODY6 consists of
  - params.h, common6.h
  - Many "xxxxxx.f" subroutines
  - Makefile
- To add a plug-in
  - Make a new directory (ex. GPU)
  - Copy and modify the file for subroutines to override
  - Add files for the new routines
  - Prepare a new Makefile
    - Copy back *.o files and re-link

The development process



Starting point

- GPROF told us that 95% of CPU time of the original version was consumed for the *regular step* on a 16k stars run
  - So called *hotspot*
- If we make the part 20x faster, the run becomes 10x faster
  - So called Amdahl's low
  - Even if this part is accelerated infinitely, 20x is the upper limit
- Soon, the *irregular step* part became the bottleneck

API for the regular-force acceleration

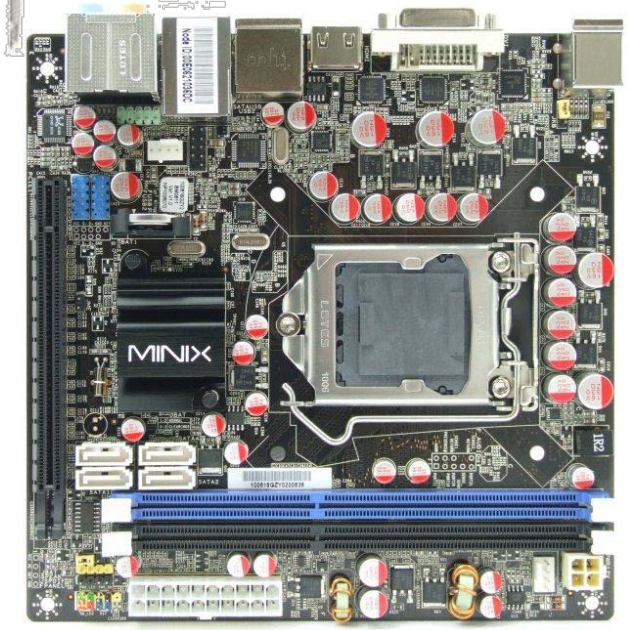
- gpunb_open(), gpunb_close(), gpunb_send(), gpunb_regf()
- Calculate forces on Ni stars from Nj stars
  - if $|\boldsymbol{r}_j - \boldsymbol{r}_i| < h_i$
    - skip the force from being accumulated
    - The index *j* is added to the neighbor list of particle *i*
  - Single-precision was found to be accurate enough
    - Close interactions are evaluated on the host side

And for the irregular-force…

- The next bottleneck
- The API became more complicated
  - open, close, set_jp, set_list, pred_all, pred_part, firr, firr_vec
- GPU implementation was not so fast
  - CPU version with SSE/OpenMP is adopted now
  - 10 μs / block-step
- Single Interface Multiple Implementations
  - Simple C/C++, SSE + OpenMP, CUDA for GPU

Special purpose machine for NBODY6

- 100k JPY (cf. 5M JPY for 1 Tflops GRAPE-6)



Summary

- NBODY6/GPU made it very inexpensive to simulate star clusters with 100k-200k stars
- Still we need a nice new point-mass *N*-body code
  - From scratch, form algorithms and mathematics
  - With modern language (C++, D??)
  - Accelerators and parallelization