Phantom-GRAPE
High Performance Library to Accelerate N-body Calculation with SIMD Instruction Set

Kohji Yoshikawa

Astrophysics Group
Center for Computational Sciences
University of Tsukuba

Collaborators: Keigo Nitadori (AICS, RIKEN)
Ataru Tanikawa (AICS, RIKEN)
Takashi Okamoto (Hokkaido Univ.)

筑波大学
University of Tsukuba

筑波大学
計算科学研究センター
Center for Computational Sciences

N-body Simulations in Astrophysics

- self-gravitating systems
  - stars, star clusters, galaxies, galaxy clusters, …
  - the matter (stars, gas and dark matter) is under the gravitational field exerted by themselves
  - important physics for formation and time evolution of these objects

NGC4414

- N-body simulations
  - indispensable ingredients for numerical simulations of most astrophysical objects
  - collisional system : star clusters, black holes need for high-precision calculations

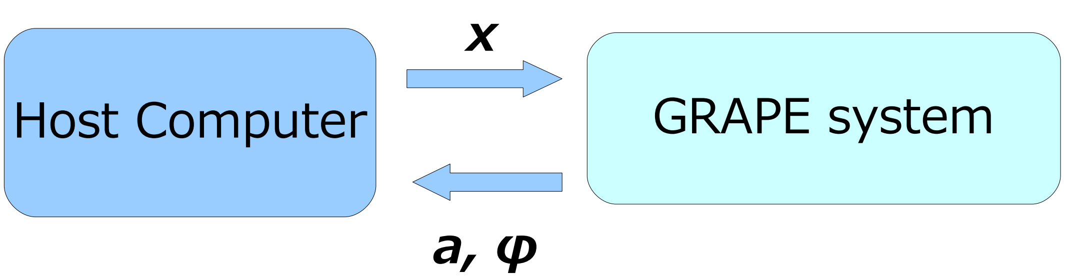
    → brute-force scheme
  - collisionless system : galaxies, galaxy clusters can be simulated with lower-precision calculations

    → Tree scheme

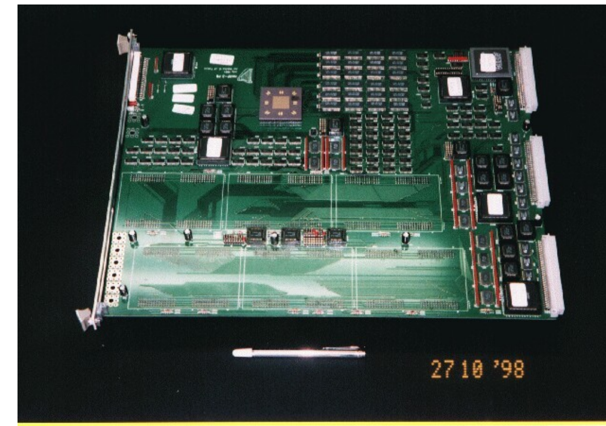
M80 / globular cluster

GRAPE Family

- GRAPE systems (GRAPE-1 to GRAPE-8)
  - special purposed hardware to accelerate N-body calculation
  - developed by the team led by Jun Makino
  - won the Gordon-Bell prize 7 times (1995, 1996, 1999, 2000, 2001, 2002, 2003)
  - odd-numbered version: for collisionless systems
    even-numbered version: for collisional systems
  - work as external accelerators connected with a host computer through I/O bus such as PCI-X and PCIe.



GRAPE-4 (1998)



GRAPE-6A (2003)

$$\frac{d\vec{v}_i}{dt} = \sum_{j=1}^{N} \frac{G m_j (\vec{x}_i - \vec{x}_j)}{(|\vec{x}_i - \vec{x}_j|^2 + \varepsilon^2)^{3/2}}$$

Reluctant with External Devices, When We Find ,,,

- We have to rewrite codes for each kind of external hardware
- Communication overhead between host computers and external devices
- Expensive...
- Too much electricity...
- Frequent hardware failure...
- External devices are quickly getting obsolete...

Another strategy to accelerate N-body calculations other than the use of external devices...

SIMD Instruction

- "Single Instruction Multiple Data" instructions

  perform the same operation on multiple points simultaneously

f0	f1	f2	f3

+

g0	g1	g2	g3

||

f0+g0	f1+g1	f2+g2	f3+g3

  - originally equipped for handling media data
  - operations are carried out on dedicated registers

- Most of the modern processors are equipped with SIMD instruction sets.
  - Intel x86 : Streaming SIMD Extension (SSE)
    Advanced Vector eXtension (AVX)
  - Fujitsu SPARC VIIIfx: HPC-ACE
  - IBM POWER: AltiVec

- Difficult to utilize SIMD instructions with high-level languages
  - Even commercial compiler are not able to utilize SIMD instruction effectively.
  - Intrinsic functions or assembly language are necessary for explicit use of SIMD instructions

SIMD Instructions on Intel-64

- Streaming SIMD Extension (SSE)
  - operates 2 double precision (DP) or 4 single precision (SP) calculations
  - 16 128-bit-wide registers (XMM registers)
- Advanced Vector eXtension (AVX)
  - operates 4 double precision (DP) or 8 single precision (SP) calculations
  - 16 256-bit-wide registers (YMM registers)
  - available for Sandy-Bridge architecture and later
- Fast Inverse Square Root
  - an instruction for approximate inverse square root
  - 12-bit accuracy
  - can be more accurate with Newton-Raphson iteration
  - very useful for *N*-body calculation

Example: AVX instructions

- CPP Macros for AVX instructions

```
#define VLOADPS(mem, reg) asm("vmovaps %0, %"reg::"m"(mem));
#define VSTORPS(reg, mem) asm("vmovaps %"reg ", %0" ::"m"(mem));
#define VADDPS(reg1, reg2, dst) asm("vaddps " reg1 "," reg2 "," dst);
#define VMULPS(reg1, reg2, dst) asm("vmulps " reg1 "," reg2 "," dst);
#define VSQRTPS(reg, dst) asm("vsqrtps " reg "," dst);
#define VDIVPS(reg1, reg2, dst) asm("vdivps " reg1 "," reg2 "," dst);
#define VSUBPS(reg1, reg2, dst) asm("vsubps " reg1 "," reg2 "," dst);
#define VRSQRTPS(reg, dst) asm("vrsqrtps " reg "," dst);
```

- Dedicated registers for AVX instructions

```
#define YMM00 "%ymm0"
#define YMM01 "%ymm1"
#define YMM02 "%ymm2"
#define YMM03 "%ymm3"
```

Example: AVX instructions

```
#define N (262144)

static float a[N], b[N], c[N];

for(int i=0;i<N;i++) {
      c[i] = a[i]+b[i];
      c[i] = c[i]*b[i];
      c[i] = sqrtf(c[i]);
      c[i] = b[i]/c[i];
}
```

```
#define N (262144)

static float a[N], b[N], c[N];

for(int i=0;i<N;i+=8) {
      VLOADPS(a[i], YMM00);
      VLOADPS(b[i], YMM01);
      VADDPS(YMM0, YMM01, YMM02);
      VMULPS(YMM02, YMM01, YMM02);
      VSQRTPS(YMM02, YMM02);
      VDIVPS(YMM01, YMM02, YMM02);
      VSTORPS(YMM02, c[i]);
}
```

The loop implemented with the AVX instruction set is about three times faster.

Phantom-GRAPE

- Software emulation of GRAPE-systems accelerated with SIMD instructions
- Both for collisional and collisionless N-body simulations
- First implemented with SSE instructions in 2005

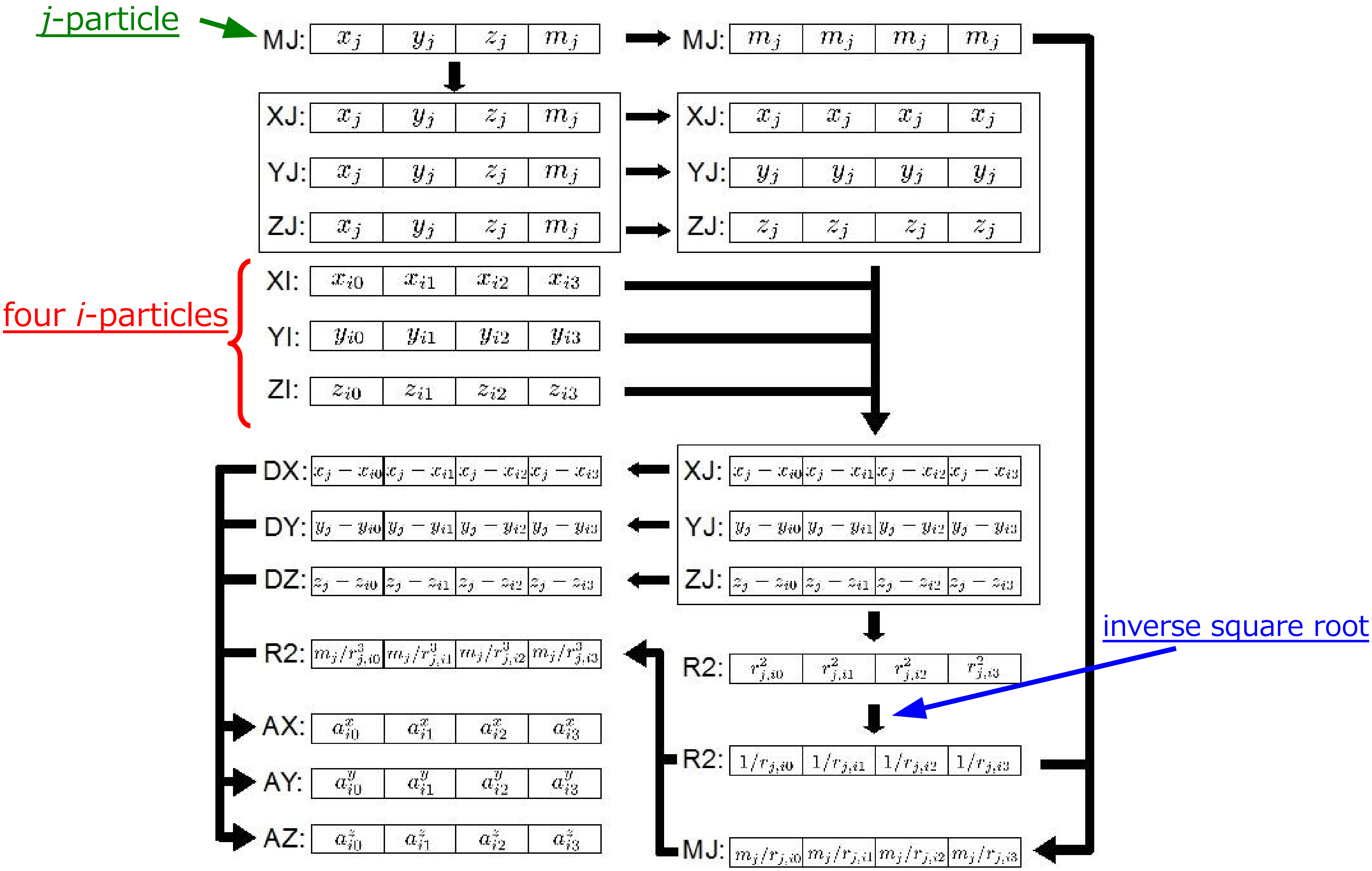
  Nitadori, K., Makino, J., Hut, P., 2006. New Astronomy 12, 169.
- Updated with AVX instructions in 2011

  Tanikawa, A., Yoshikawa, K., Okamoto, T., Nitadori, K., 2012. New Astronomy 17, 82

  Tanikawa, A., Yoshikawa, K., Nitadori, K., Okamoto, T., 2013. New Astronomy 19, 74
- Compatible API with GRAPE-5 and GRAPE-6
- Compute forces on 4 *i*-particles simultaneously with SSE instructions
- Compute forces exerted by 2 *j*-particles simultaneously with AVX instructions

$$\frac{d\vec{v}_i}{dt} = \sum_{j=1}^{N} \frac{Gm_j(\vec{x}_i - \vec{x}_j)}{(|\vec{x}_i - \vec{x}_j|^2 + \varepsilon^2)^{3/2}}$$

Implementation



Implementation

```
for(j=0;j<nj;j++) {
  dx = xj[j][0]-xi;
  dy = xj[j][1]-yi;
  dz = xj[j][2]-zi;

  rsq = dx*dx+dy*dy+dz*dz+eps2;
  rinv = 1.e0/sqrt(rsq);
  rinv *= rinv;
  rinv *= rinv;

  axi += mj[j]*rinv*dx;
  ayi += mj[j]*rinv*dy;
  azi += mj[j]*rinv*dz;
}
```

```
for(j=0;j<nj; j++) {
  LOADPS(*jp++, MJ);
  MOVAPS(MJ, X2);
  MOVAPS(MJ, Y2);
  MOVAPS(MJ, Z2);
  BCAST0(X2);
  BCAST1(Y2);
  BCAST2(Z2);
  BCAST3(MJ);
  SUBPS_M(*ipdata->x, X2);
  SUBPS_M(*ipdata->y, Y2);
  SUBPS_M(*ipdata->z, Z2);

  MOVAPS(X2, DX);
  MULPS(X2, X2);

  MOVAPS(Y2, DY);
  MULPS(Y2, Y2);

  MOVAPS(Z2, DZ);
  MULPS(Z2, Z2);

  ADDPS(X2, Z2);
  ADDPS(Y2, Z2);
  ADDPS(EPS2, Z2);
```

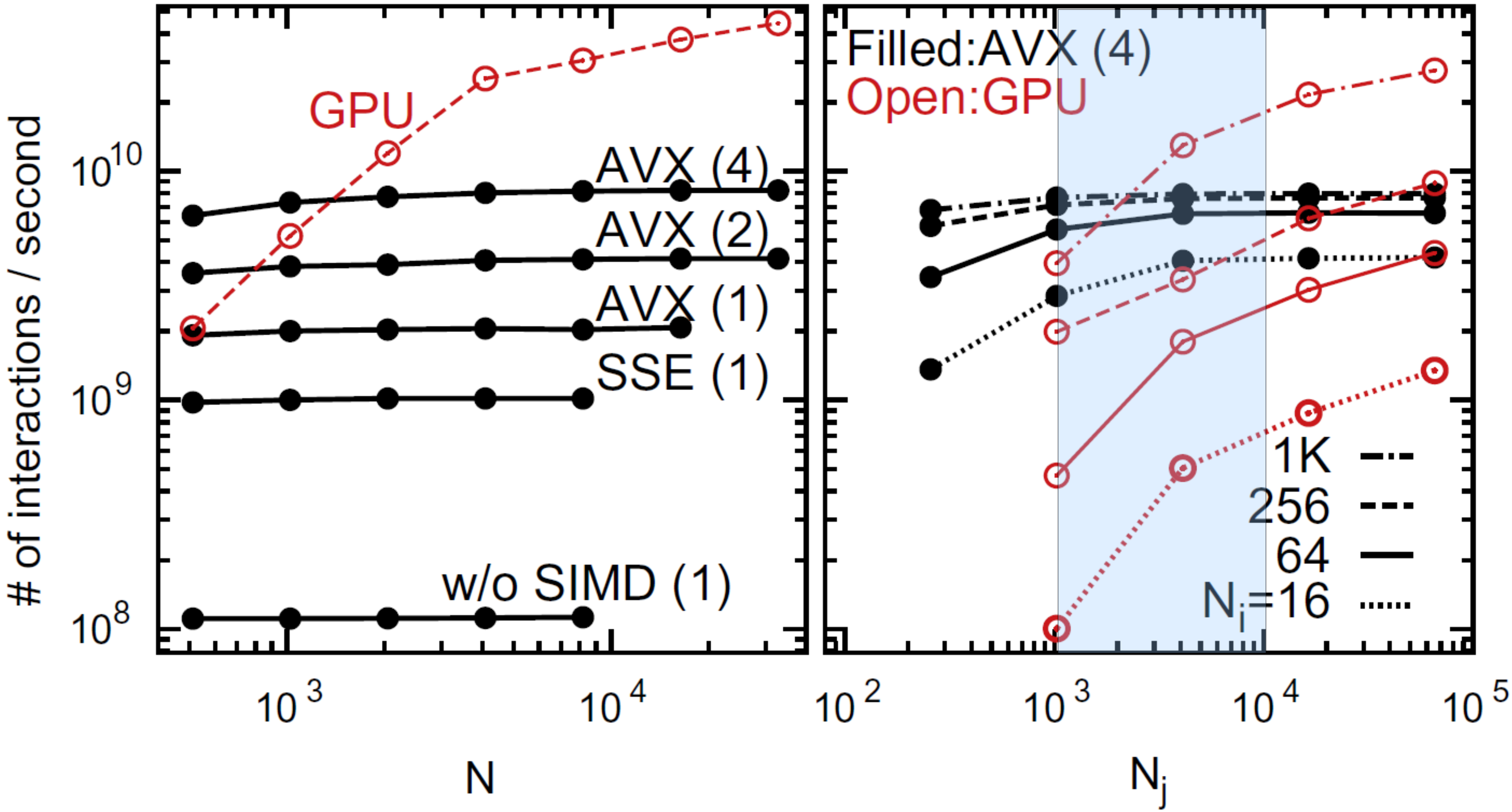
```
  RSQRTPS(Z2, Z2);
  MULPS(Z2, MJ); // MJ := m/r
  MULPS(Z2, Z2); // Z2 := 1/r**2
  MULPS(MJ, Z2); // Z2 := m/r**3

  MULPS(Z2, DX);
  ADDPS(DX, AX);

  MULPS(Z2, DY);
  ADDPS(DY, AY);

  MULPS(Z2, DZ);
  ADDPS(DZ, AZ);
}

STORPS(AX, *fodata->ax);
STORPS(AY, *fodata->ay);
STORPS(AZ, *fodata->az);
```

24 cycles / interaction with SSE instructions

400 cycles / interaction without SSE instructions

Performance

CPU: Core-i7 2600K GPU: NVIDIA GeForce GTX580



- Performance of Phantom-GRAPE is almost independet of # of particles
- GPU performance is hampered by communication overhead for small # of particles

Summary / Future Propects

- Phantom-GRAPE : software “GRAPE” accelerated with SSE and AVX instructions
- Performance of Phantom-GRAPE is comparable to that of GPU
- Free from communication overhead between host computers and external devices
- Several famous simulation codes adopt Phantom-GRAPE in their N-body module.

  ASURA: N-body + Hydrodynamics code for galaxy formation

  GreeM: N-body code for large-scale structure formation in the universe

  AMUSE: Compilation of astrophysics simulation code

- AVX-512 instruction set on Intel KNL.
- Publicly available at

  https://code.google.com/p/phantom-grape/