

Special-Purpose Computing for Dense Stellar Systems

Jun Makino

Center for Computational
Astrophysics

National Astronomical Observatory
of Japan

Talk overview

- Introduction: Short history of GRAPE
- GRAPE-DR project
- GRAPE-DR processor details
- Programming environment
- Project status
- Summary

Short history of GRAPE

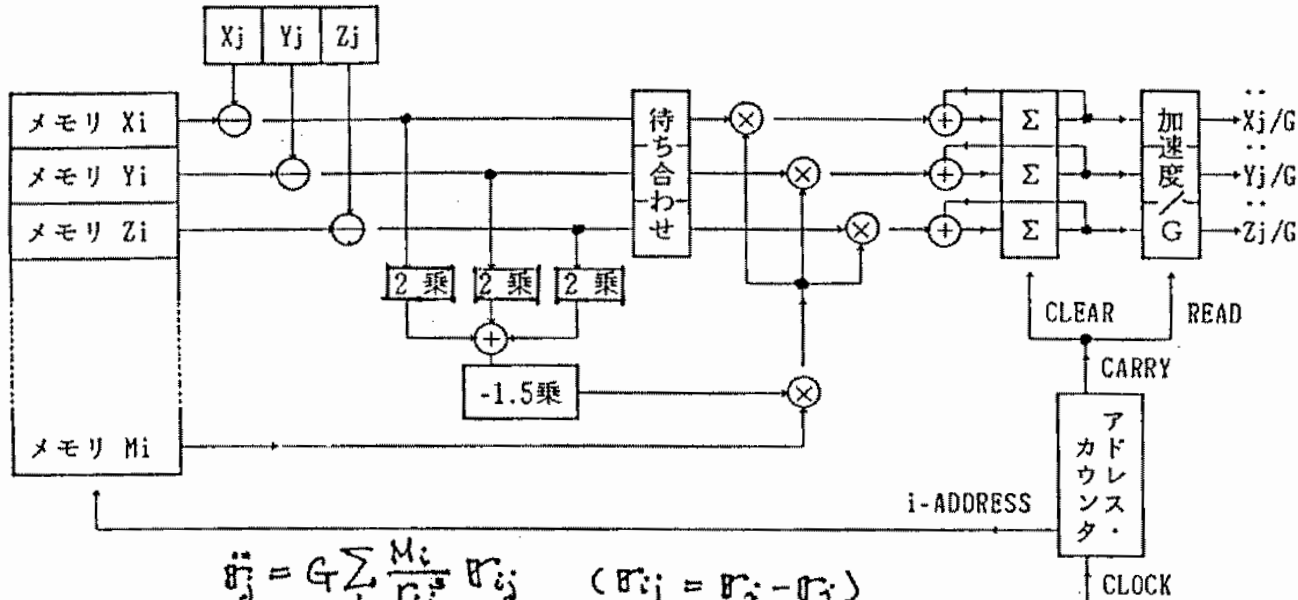
- Basic concept
- Chikada's proposal
- GRAPE-1 through 6
- Other GRAPEs

Basic concept

- With N -body simulation, almost all calculation goes to the calculation of particle-particle interaction.
- This is true even for schemes like Barnes-Hut treecode or FMM.
- A simple hardware which just calculates the particle-particle interaction can greatly accelerate overall calculation.

Proposal by Chikada

1988, Summer school for astronomy and astrophysics



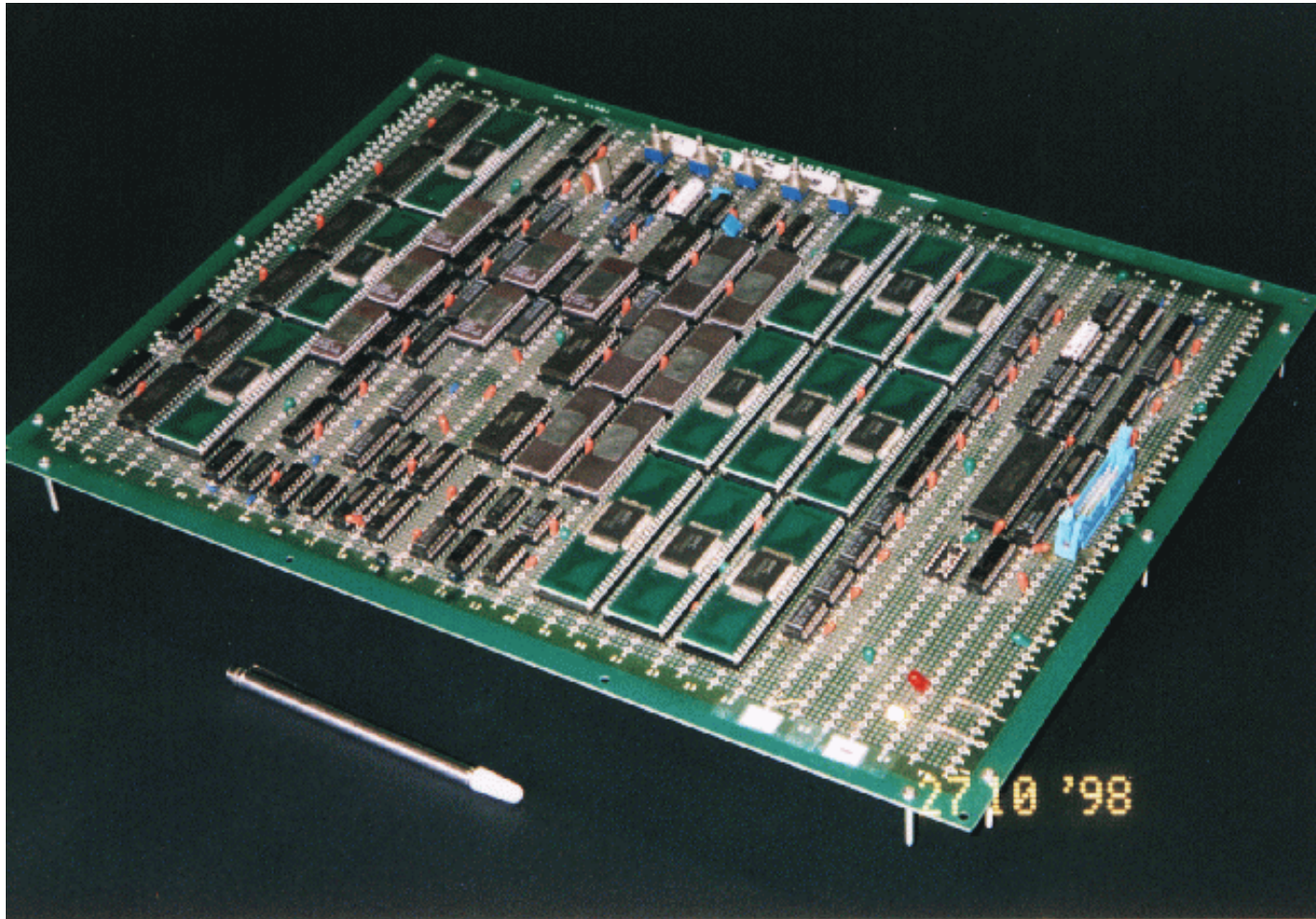
$$\ddot{\vec{r}}_j = G \sum_i \frac{M_i}{r_{ij}^3} \vec{r}_{ij} \quad (\vec{r}_{ij} = \vec{r}_i - \vec{r}_j)$$

+, -, ×, 2乗は1 operation, -1.5乗は多項式近似でやるとして10operation 位に相当する。
 総計24operation.
 各operationの後にはレジスタがあって、全体がpipelineになっているものとする。
 「待ち合わせ」は2乗してMと掛け算する間の時間ズレを補正するためのFIFO(First-In First-Out memory).
 「Σ」は足し込み用のレジスタ。N回足した後結果を右のレジスタに転送する。

図2. N体問題のj-体に働く重力加速度を計算する回路の概念図。

Sugimoto “jumped in”, with a few graduate students.

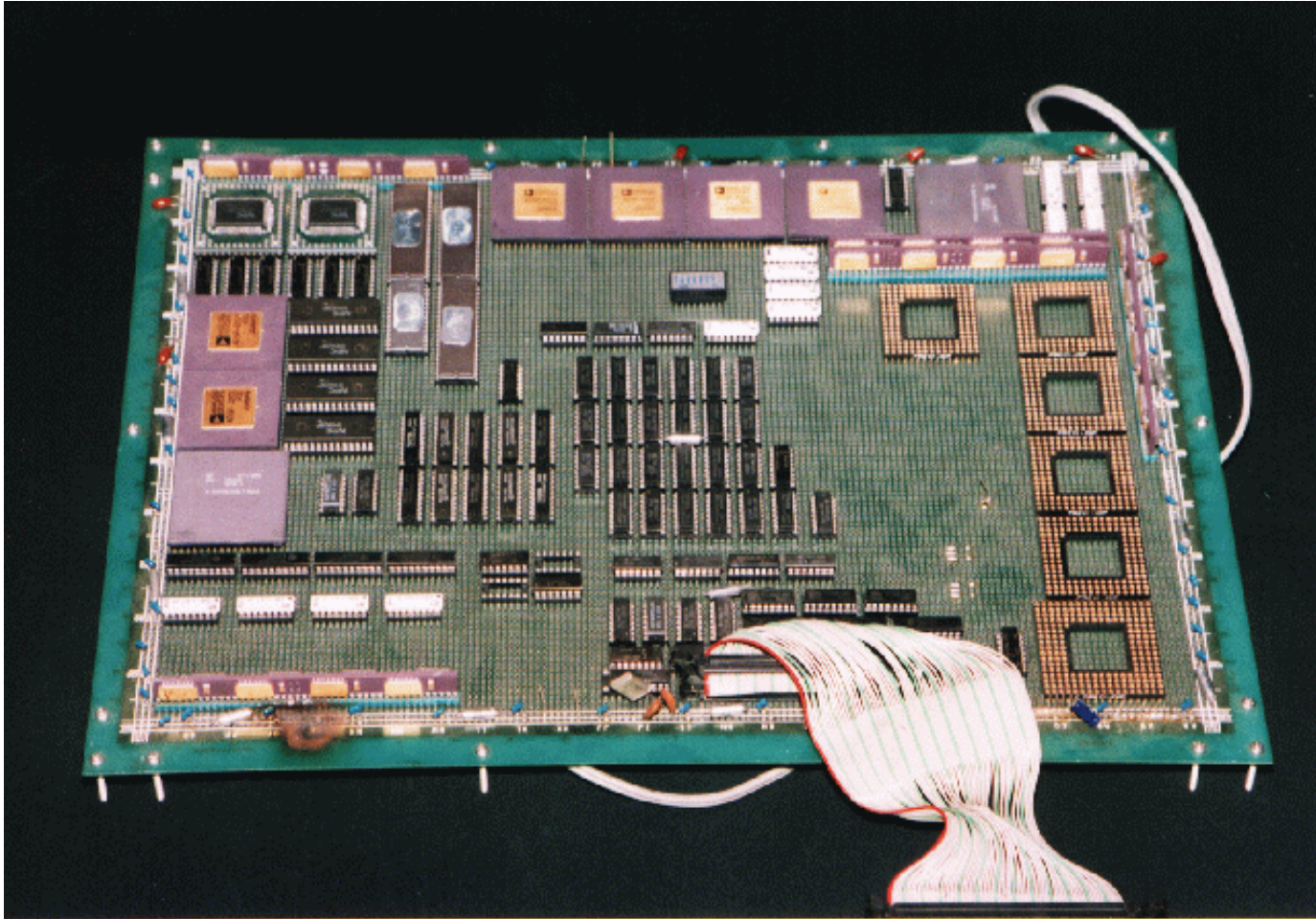
GRAPE-1 (1989)



GRAPE-1

- “Mixed precision”. Fixed point–logarithmic–fixed point.
- around 100 IC chips
- around USD 2,000
- 240Mflops
- IEEE-488 interface, $\sim 100\text{KB/s}$

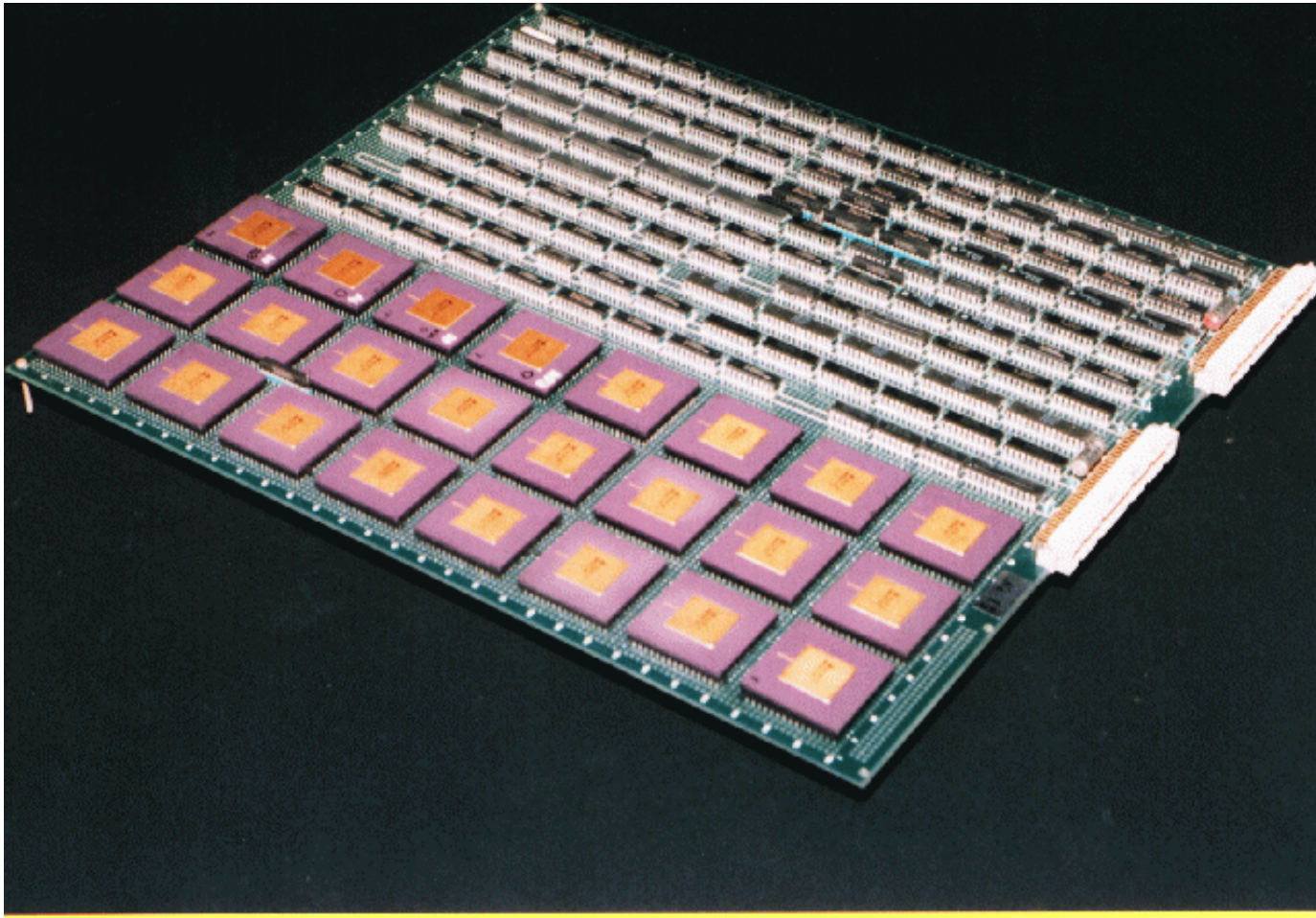
GRAPE-2(1990)



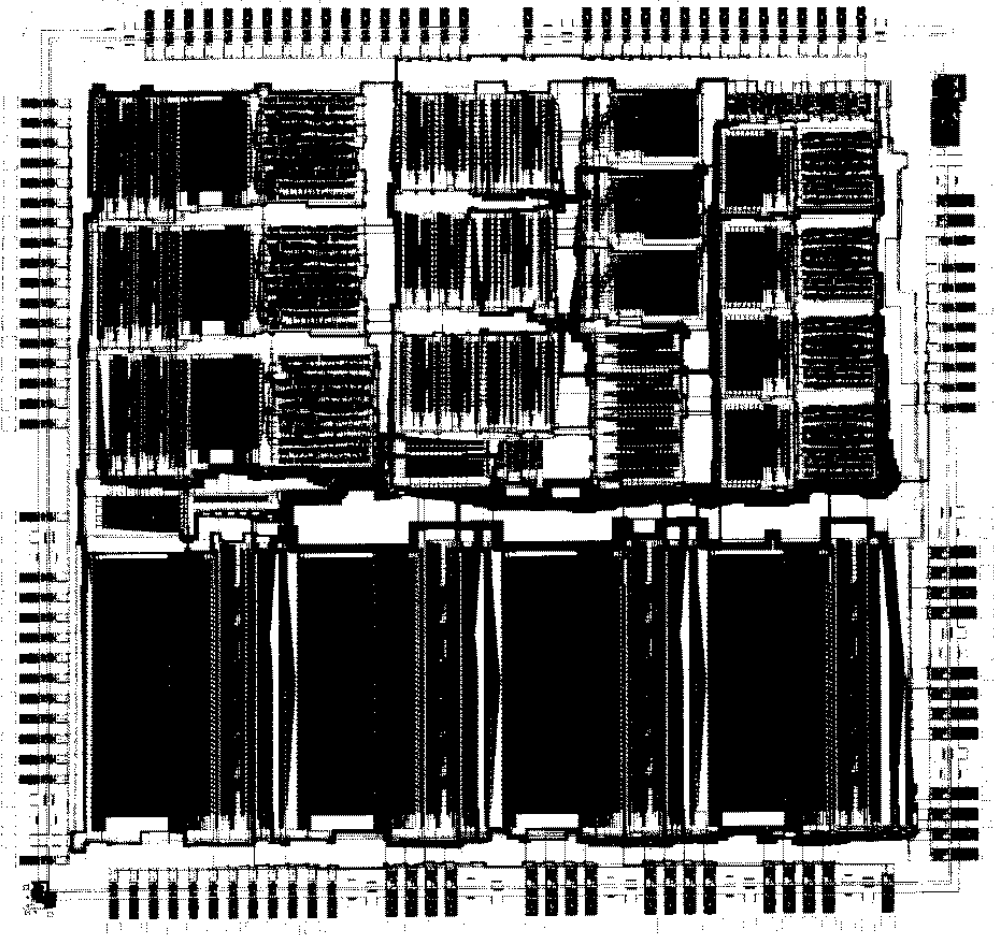
GRAPE-2

- Mixed double precision/single precision.
- 40Mflops
- VME interface, 2-3MB/s

GRAPE-3(1991)



GRAPE-3 LSI



2 mm

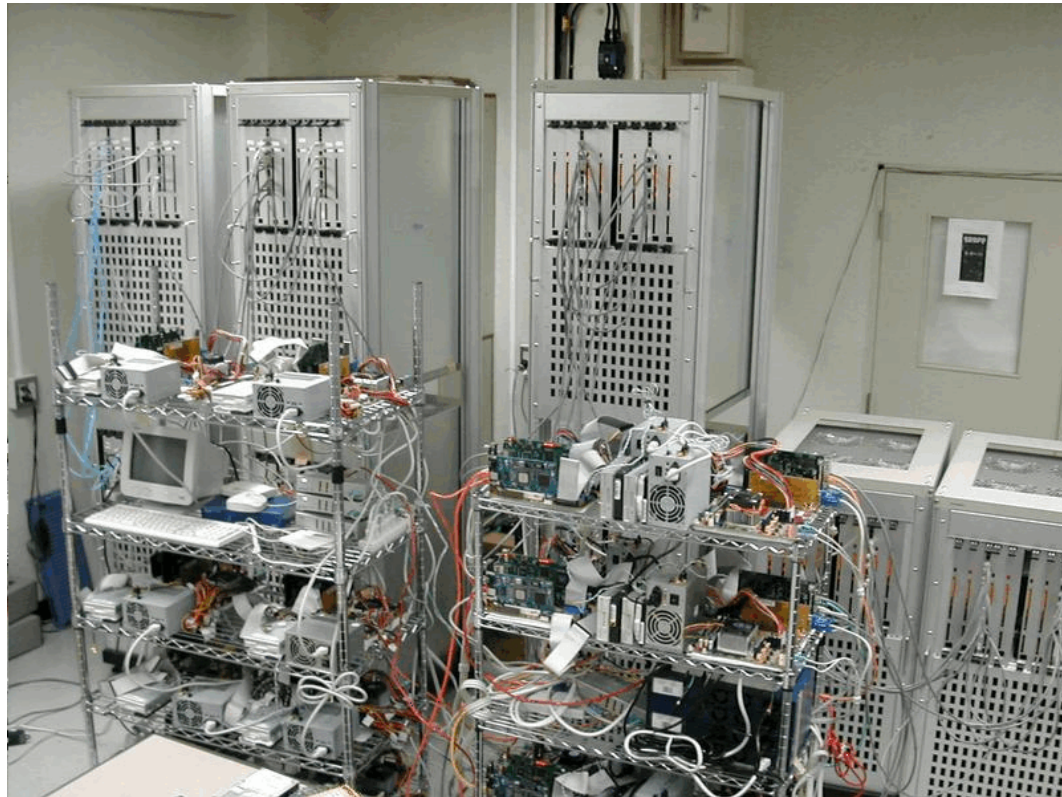
GRAPE-4(1995)



GRAPE-4 Overview

- 36 boards each with 48 chips
- 640 Mflops per chip (20 operations, 32MHz clock)
- LSI logic $1\mu\text{m}$

GRAPE-6(2001)



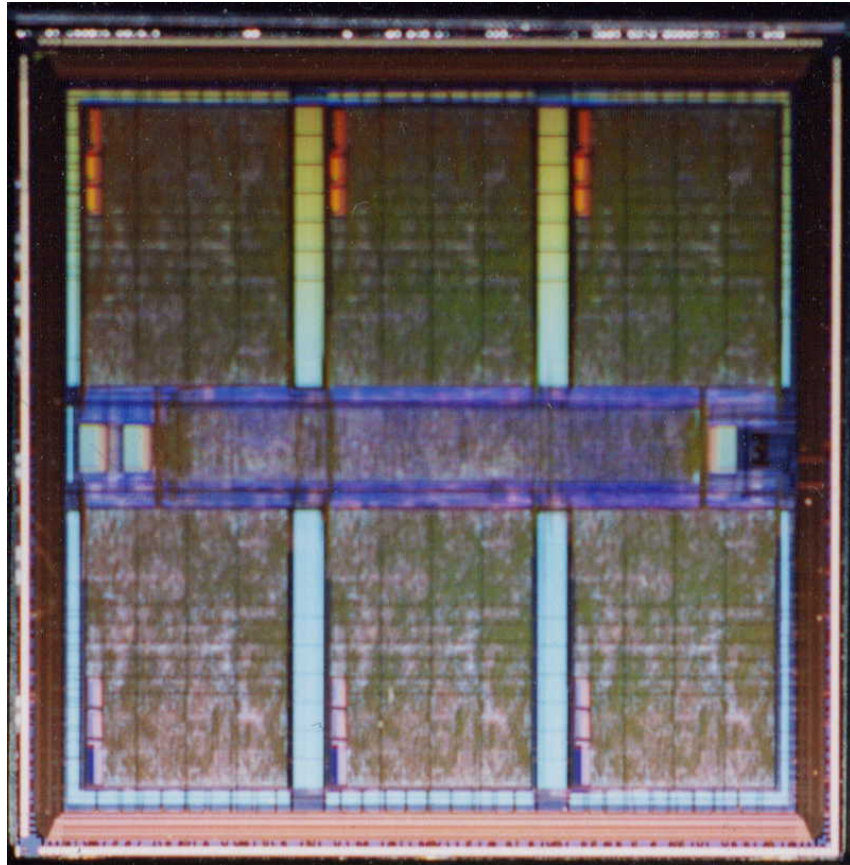
64 Tflops peak

**2048 processor
chips**

64 processor boards

16 hosts

Processor LSI



- 0.25 μm design rule
(Toshiba TC-240,
1.8M gates)
- 90 MHz Clock
- 6 pipeline processors
- 31 Gflops / chip

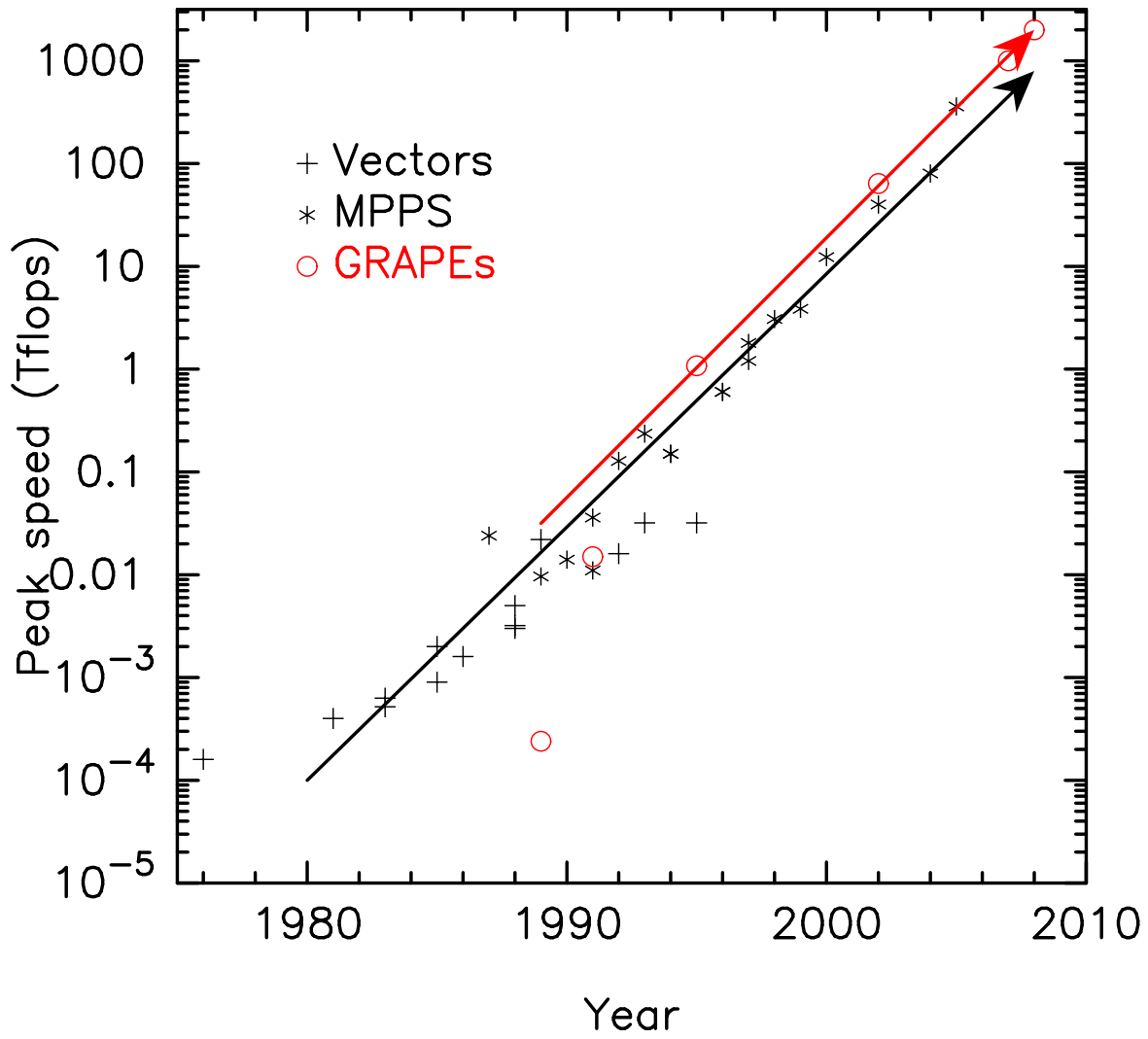
Comparison with a recent Intel processor

	GRAPE-6	Intel Woodcrest (Xeon 5160)
Design rule	250nm	65nm
Clock	90MHz	3GHz
Peak speed	32.4Gflops	24Gflops
Power	10W	80 W
Perf/W	3.24Gflops	0.3 Gflops

Other GRAPEs

- 1991 GRAPE-1A : re-designed GRAPE-1
- 1992 GRAPE-2A : MD simulations
- 1992 HARP-1: Hermite scheme
- 1993 GRAPE-3A: Commercial version of 3
- 1996 MD-GRAPE: First custom chip for MD
- 2001 MDM : RIKEN MDGRAPE-2, 75 Tflops
- 2006 PE : RIKEN MDGRAPE-3, 1Pflops
- 1992- MD-Engine: Another commercial MD-GRAPE(NEC)

Performance history



Since 1995 (GRAPE-4), GRAPE has been faster than general-purpose computers.

Development cost was around 1/100.

Should we just continue?

Problem with GRAPE approach

- Chip development cost becomes too high.

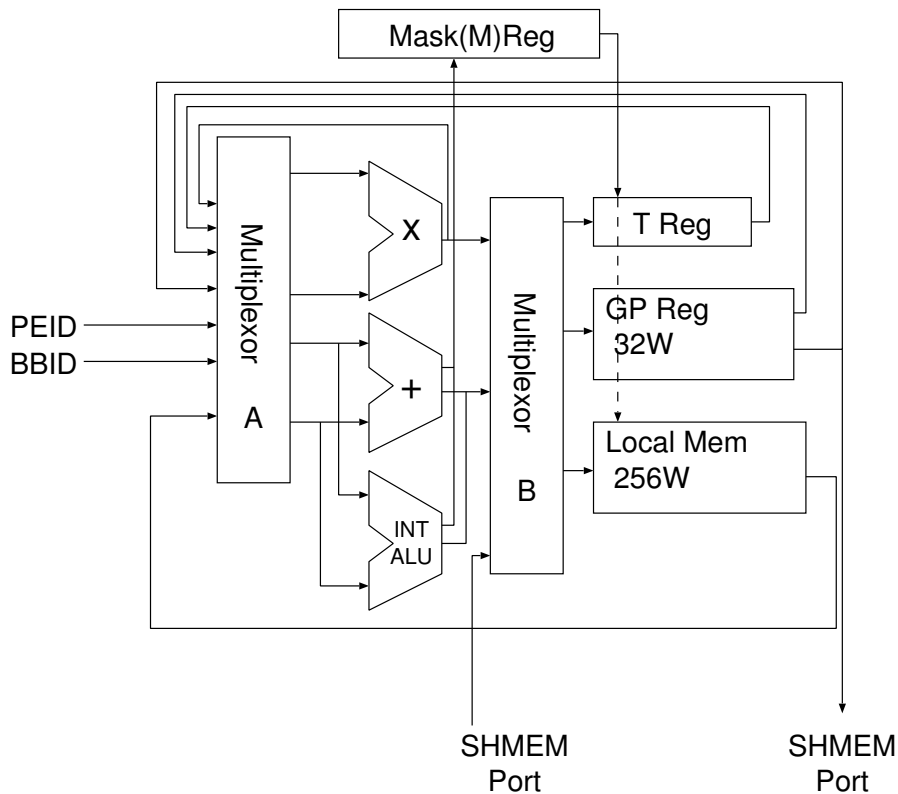
Year	Machine	Chip Initial Cost
1992	GRAPE-4	200K\$
1997	GRAPE-6	1M\$
2004	GRAPE-DR	4M\$

Initial cost should be 1/4 or less of the total budget.
How we can continue?

Next-Generation GRAPE — GRAPE-DR

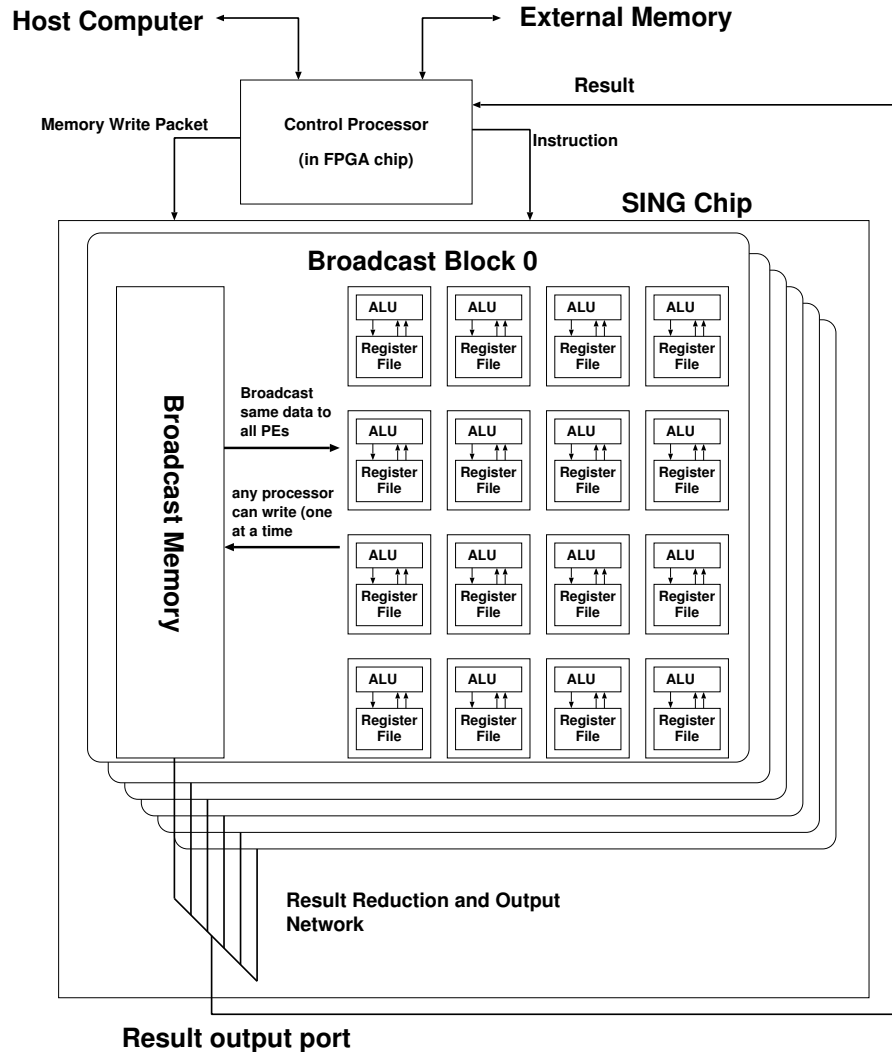
- Planned peak speed: 2 Pflops
- **New architecture — wider application range than previous GRAPEs**
- primarily to get funded
- No force pipeline. SIMD programmable processor
- Planned completion year: FY 2008 (early 2009)

Processor architecture



- Float Mult (24 bit mantissa, with full 49 bit output)
- Float add/sub (60 bit mantissa)
- Integer ALU (72 bit)
- 32-word (72 bit) general-purpose register file
- 256-word (72 bit) memory
- ports to shared memory (shared by 32 processors)

Chip structure



Collection of small processors.

One chip integrates 512 processors.

Single processor runs at 500MHz clock (2 operations/cycle).

Peak speed of one chip: 0.5 Tflops (20 times faster than GRAPE-6).

Why we changed the architecture?

- To get budget (N -body problem is too narrow...)
- To allow wider range of applications
 - Molecular Dynamics
 - Boundary Element method
 - Dense matrix computation
 - SPH
- To allow wider range of algorithm
 - FMM
 - Ahmad-Cohen
- To try something new.

Why we changed the architecture?

- To get budget (N -body problem is too narrow...)
- To allow wider range of applications
 - Molecular Dynamics
 - Boundary Element method
 - Dense matrix computation (Linpac, **TOP500!**)
 - SPH
- To allow wider range of algorithm
 - FMM
 - Ahmad-Cohen
- To try something new.

Comparison with FPGA

- much better silicon usage (ALUs in custom circuit, no programmable switching network)
- (possibly) higher clock speed (no programmable switching network on chip)
- easier to program (no VHDL necessary; assembly language and compiler instead)
- major drawback: somebody (*which means me...*) need to develop the chip

How do you use it?

- **GRAPE:** We'll write the necessary software. Move from GRAPE-6 will be less painful than move from GRAPE-4 to GRAPE-6.
- Matrix etc ... RIKEN/NAOJ will do something
- New applications:
 - Primitive Compiler available
 - For high performance, you need to write the kernel code in assembly language

Primitive compiler

(Nakasato 2006)

```
/VARI  xi, yi, zi, e2;
```

```
/VARJ  xj, yj, zj, mj;
```

```
/VARF  fx, fy, fz;
```

```
dx = xi - xj;
```

```
dy = yi - yj;
```

```
dz = zi - zj;
```

```
r2 = dx*dx + dy*dy + dz*dz + e2;
```

```
r3i= powm32(r2);
```

```
ff = mj*r3i;
```

```
fx += ff*dx;
```

```
fy += ff*dy;
```

```
fz += ff*dz;
```

- Assembly code
- Interface/driver functions

are generated from this "high-level description".

Development status

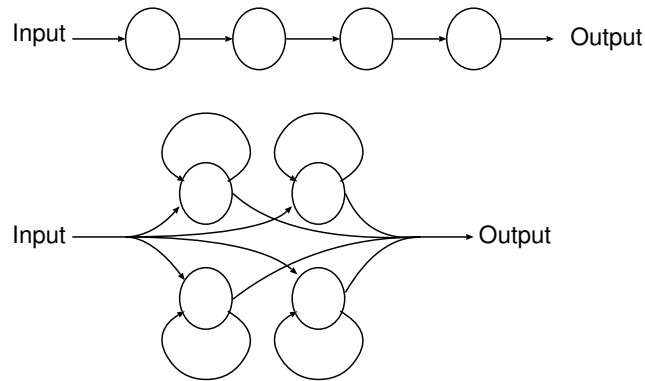


1st prototype board. (Designed by Toshi Fukushige)
Confirmed successful operation at 500MHz clock
Currently working on softwares and FPGA design to
run real applications

Summary

- GRAPE-DR, with programmable processors, will have wider application range than traditional GRAPE.
- Assembly language defined.
- Primitive compiler is ready.
- Processor chip is completed and no problem found (so far).

Difference from previous GRAPE architecture

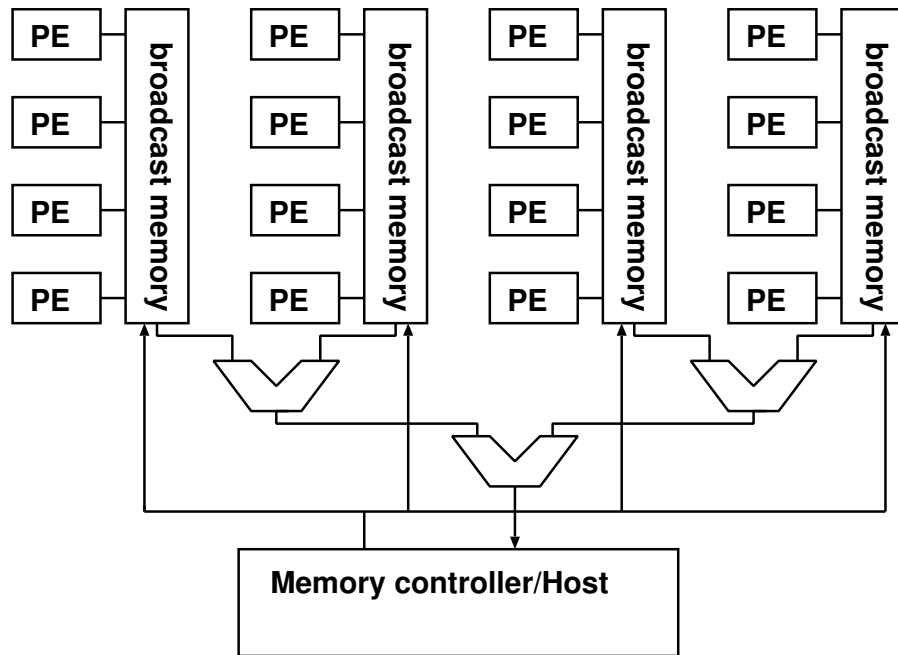


- No hardwired pipeline, simple SIMD parallel processor.

Development codename: **SING** (*Sing is not GRAPE*)
(Eiichiro Kokubo)

- Much like the Connection Machine
- Performance hit: factor 3-10? (We'll see)

Top-level architecture



- two-level hierarchy
- External memory → broadcast memory
- broadcast memory → reduction tree → output port
- Essentially the same as the board-level structure of large GRAPE-6 board.

What do we need to specify?

Consider a GRAPE-like model:

We calculate $\sum_j f(x_i, x_j)$, for many “ i ”s and many “ j ”s

1. First send all “ j ”s
2. Repeat the following
3. send “ j ”s to the local memory of PEs
4. Let the chip calculate the interaction $f(x_i, x_j)$ and accumulate
5. retrieve the result

This model, with some extension, seems to be able to express most of the things we want to do on GRAPE-DR.

Interface definitions

We need ways to specify

- actual PE operations to calculate f
- procedure and data type for
 - “ j ”s
 - “ i ”s
 - results

Our current solution

- actual PE operations specified in assembly language
- interface functions generated from variable declarations
- Compiler on top of this environment

Example: acceleration and jerk

- definition of “*i*”s
- definition of “*j*”s
- definition of results
- actual code

definition of “*i*”s

```
var vector long xi      hlt  flt64to72
var vector long yi      hlt  flt64to72
var vector long zi      hlt  flt64to72
var vector short vxi    hlt  flt64to36
var vector short vyi    hlt  flt64to36
var vector short vzi    hlt  flt64to36
var vector short eps2i  hlt  flt64to36
var vector short idxi   hlt  fix32to36ru
var vector long ti      hlt  flt64to72
```

“hlt” means “*i*” data

last magic words specify data conversion

Interface function

```
struct SING_hlt_struct0{
    double xi;
    double yi;
    double zi;
    double vxi;
    double vyi;
    double vzi;
    double eps2i;
    UINT32 idxi;
    double ti;
};

int SING_send_i_particle(struct SING_hlt_struct0 *ip,
                        int n);
```

definition of “*j*”s

bvar long x0	elt	flt64to72
bvar long tj	elt	flt64to72
bvar short sx18	elt	flt64to36
bvar short jx6	elt	flt64to36
bvar short ax2	elt	flt64to36
bvar short v0x	elt	flt64to36
bvar short mj	elt	flt64to36
bvar short eps2	elt	flt64to36
bvar short idxj	elt	fix32to36ru

(shown x-components only)

“bvar”: stored in the broadcast memory

“elt”: “*j*” data

Interface function

```
struct SING_elt_struct0{
    double x0;
    double tj;
    double sx18;
    double jx6;
    double ax2;
    double v0x;
    double mj;
    double eps2;
    UINT32 idxj;
};

int SING_send_elt_data0(struct SING_elt_struct0 *ip,
                       int index_in_EM);
```

definition of result

```
var vector long accx  rrn  flt72to64 fadd
var vector long accy  rrn  flt72to64 fadd
var vector long accz  rrn  flt72to64 fadd
var vector long jerkx rrn  flt72to64 fadd
var vector long jerky rrn  flt72to64 fadd
var vector long jerkz rrn  flt72to64 fadd
var vector long pot   rrn  flt72to64 fadd
var vector long nnb   rrn  fix72to64w umin
```

“rrn” means that it is result

last two magic words give data conversion and reduction

Interface function

```
struct SING_result_struct{
    double accx;
    double accy;
    double accz;
    double jerkx;
    double jerky;
    double jerkz;
    double pot;
    UINT64 nnb[2];
};
int SING_get_result(struct SING_result_struct *rp);
```

Other variables

```
var short   lvxj
var short   lvyj
var short   lvzj
var vector  short lvxjv lvxj
var short   lmj
var short   leps2
var short   lidxj
var long    llidxj
var vector  long  dtlm
```


Code (1): initialization

```
loop initialization
vlen 4
nop
nop
upassa hl"7ff000000ffffffff" $t $t
upassa $ti $ti nnb
uxor $t $t $t
upassa $ti $ti accx
upassa $t $t accy
upassa $t $t accz
upassa $t $t jerkx
upassa $t $t jerky
upassa $t $t jerkz
upassa $t $t pot
```

Code (2): predictor

loop body

vlen 4

```
fsub ti $lr0 $t
```

alias dt \$lr0

```
upassa $ti $ti dt dtlm
```

alias dxp \$r32v

```
nop
```

alias ls18v \$r2v

```
fmul $t ls18v $t
```

alias lj6v \$r24v

```
fmul $ti f"0.75" ; fadd $ti lj6v
```

alias la2v \$r20v

```
fmul $ti f"1.5" $t ; fadd $fb lj
```

alias lv0v \$r36v

```
fmul $ti dt ; bm x0 lx0v
```

alias lx0v \$lr12v

```
fadd $fb la2v $t ; fmul dxp dtlm
```

bm tj \$lr0v

```
fadd $fb la2v $t ; fmul $ti dt
```

vlen 3

```
fmul $ti dt ; upassa $fb $t
```

bm jx6 lj6v

```
fadd $fb lv0v $t ; fmul $ti f"2.
```

bm ax2 la2v

```
fmul $ti dt ; fadd $fb lv0v
```

bm v0x lv0v

```
fadd $fb lx0v $t lx0v
```

Getting other j data etc

```
vlen 1
```

```
bm mj lmj
```

```
bm eps2 leps2
```

```
bm idxj lidj
```

```
nop
```

```
nop
```

```
upassa il"36" $t $t
```

```
ulsr lidj $ti $t
```

```
upassa $ti $ti llidj
```

```
vlen 4
```

```
nop
```

```
nop
```

Set mask bits for $i \neq j$

```
upassa idxi   idxi   $t
uxor  $ti lidxj $t
moi 2
ulnot $ti $ti $t  # mreg 1 indicates i != j
moi 0
```

Aliases for registers

alias dx \$r0v

alias dy \$r4v

alias dz \$r8v

alias lxj \$lr12

alias lyj \$lr14

alias lzj \$lr16

alias dr2 \$r12v

alias sqtmp \$r16v

alias resexp \$r16v

alias rinv \$r20v

alias ahalf \$r16v

alias ptmp \$lr24v

alias axtmp \$lr32v

alias aytmp \$lr40v
alias aztmp \$lr48v
alias vjvr \$r56v
alias vxjr \$r56
alias vyjr \$r57
alias vzjr \$r58
alias dvx \$r24v
alias dvy \$r28v
alias dvz \$r32v
alias rtmp \$r16v
alias mrinv \$r12v
alias mr3inv \$r12v
alias r2inv \$r16v
alias rvtmp \$r36v
alias jxtmp \$r52v

alias jytmp \$r56v

alias jztmp \$r60v

alias jxacc \$lr44v

alias jyacc \$lr16v

alias jzacc \$lr36v

Calculate r^2

```
fsub lxj xi dx $t
fsub lyj yi dy ; fmul $ti $ti $t
fsub lzj zi dz
fmul dy dy ; fadd $t eps2i $t
fmul dz dz ; fadd $fb $ti $t
fadd $fb $ti dr2 $t
```


Calculate r_{min}

```
imm ffffffff $t
uand $ti dr2 $t
uor $ti llidxj $t
mi 2
umin $ti nnb $t
upassa $ti $ti nnb
mi 0
nop
nop
```

Calculate r^{-3}

```
ulsr dr2      il"61"  $t
usub hl"5fe" $ti     $t      # $t is half of the exp
ulsl $ti      il"60"  resexp
moi 1
    uand h"001000000" dr2
moi 0
    uand dr2 h"000ffffff" $t
    uor  $ti h"3ff000000" $t rinv
    fmul $ti f"0.14823" $t
    fadd $ti f"-0.73758" $t
    fmul  $ti rinv $t ; upassa lvxjv lvxjv vjvr
    fadd f"1.58935" $ti $t
mi 1
```

```
fmul f"1.41421356" $ti $t
```

```
mi 0
```

```
nop
```

```
fmul $t resexp $t rinv # Here the result is the initi
```

N-R iteration

```
fmul dr2 f"0.5" ahalf # r8v is 0.5a
fmul rinv rinv $t      ; upassa pot pot ptmp
fmul $ti ahalf $t      ; upassa accx accx  axtmp
fsub f"1.5" $ti $t
fmul rinv $ti $t rinv
fmul $ti $ti $t        ; upassa accz accz  aztmp
fmul $ti ahalf $t      ; upassa accx accx  axtmp
fsub f"1.5" $ti $t
fmul rinv $ti $t rinv
....
```

Add accel

```
fmul $ti $ti $t          ; upassa accz accz  aztmp
fmul $ti ahalf $t        ; upassa accy accy  aytmp
fsub f"0.5" $ti $t
fmul rinv $ti $t
fadd rinv  $ti $t rinv
mi 2
fmul lmj $ti $t mrinv
fmul rinv rinv $t r2inv; fsub ptmp $ti pot
fmul $ti mrinv $t mr3inv
fmul $ti dx
fmul $t  dy ; fadd $fb axtmp accx
fmul $t  dz ; fadd $fb aytmp accy
fadd $fb aztmp accz
```

Jerk

```
nop
nop
fsub vxjr vxi dvx $t
fsub vyjr vyi dvy ; fmul $ti dx $t
fsub vzjr vzi dvz
fmul dvy dy ; upassa jerkx jerkx jxacc
fmul dvz dz ; fadd $fb $t $t
fadd $fb $ti rvtmp $t
fmul f"-3" $ti $t
fmul $ti r2inv $t ; upassa jerkz jerkz jzacc
fmul $ti dx ; upassa jerky jerky jyacc
fmul $t dy ; fadd $fb dvx jxtmp
fmul $t dz ; fadd $fb dvy jytmp
fadd $fb dvz jztmp
```

```
fmul mr3inv jxtmp ; upassa mr3inv mr3inv $t
fmul $ti jytmp ; fadd $fb jxacc jerkx
fmul $t jztmp ; fadd $fb jyacc jerky
fadd $fb jzacc jerkz
```

Function to run

```
int SING_grape_run(int n);
```


Primitive compiler

(Nakasato 2006)

```
/VARI  xi, yi, zi, e2;  
/VARJ  xj, yj, zj, mj;  
/VARF  fx, fy, fz;  
  
dx = xi - xj;  
dy = yi - yj;  
dz = zi - zj;  
r2 = dx*dx + dy*dy + dz*dz + e2;  
r3i= powm32(r2);  
ff = mj*r3i;  
fx += ff*dx;  
fy += ff*dy;  
fz += ff*dz;
```

Compiler language

- Interface specification essentially the same as assembly language.
- By far more compact and easier to write.

Current Limitations:

- no data type declaration (everything is float)
- no optimization

High-level architecture

- Single card: 4 chips, PCI-X/PCI-E/Hypertransport(?) interface, 2 Tflops.
- Two cards per single node, host: x86 PCs.
- Host network: 512 node, fast 10GbE switches?

Difference from GRAPE-6:

- No custom network
- No large card

Development schedule

(tentative)

2006 Fall	First sample chip
2007 Spring	Prototype board
2008 Spring	Large parallel system
2009 Spring	Final system

Summary

- GRAPE-DR, will have wider application range than traditional GRAPEs, with programmable processors.
- Assembly language defined.
- Primitive compiler is ready.