GRAPE-DR and its programming environment (if any)

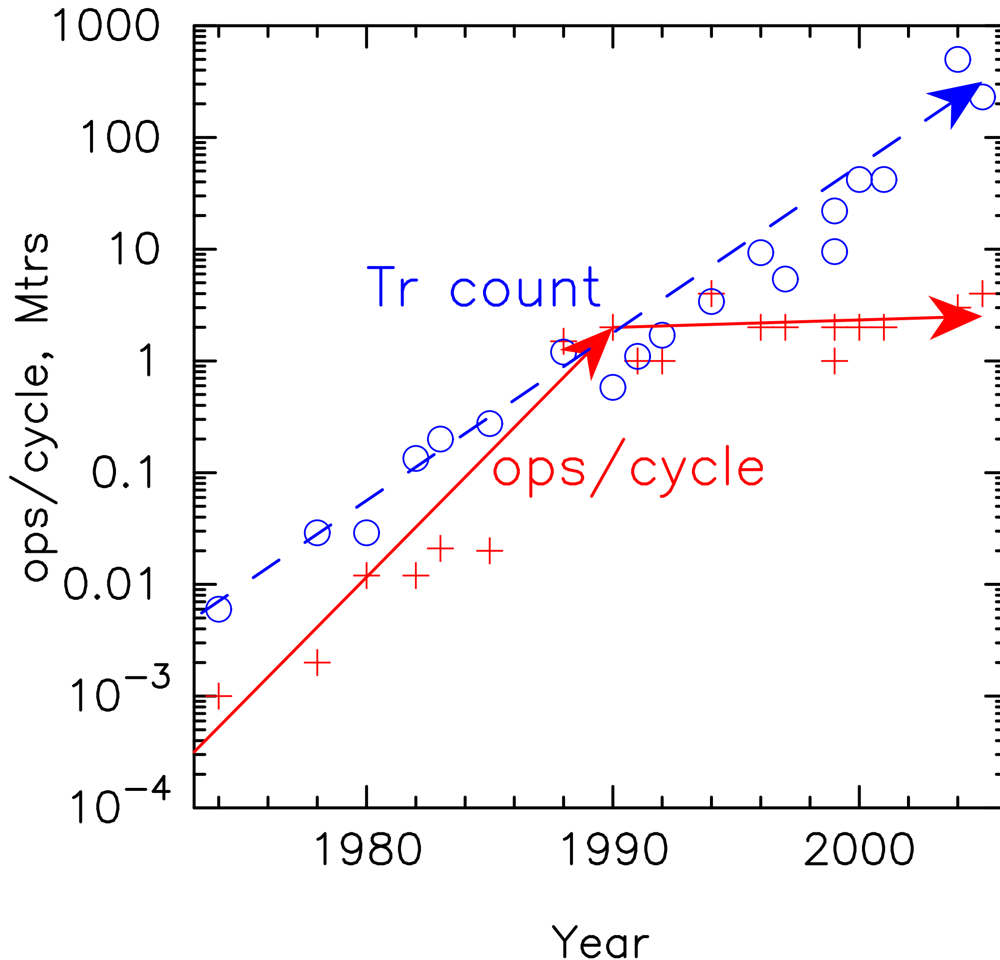
Jun Makino

University of Tokyo

Talk overview

- **Introduction: Why GRAPE-DR**
- **GRAPE-DR processor details**
- **Programming in Assembly language**
- **Compilers!**
- **Summary**

Evolution of microprocessors

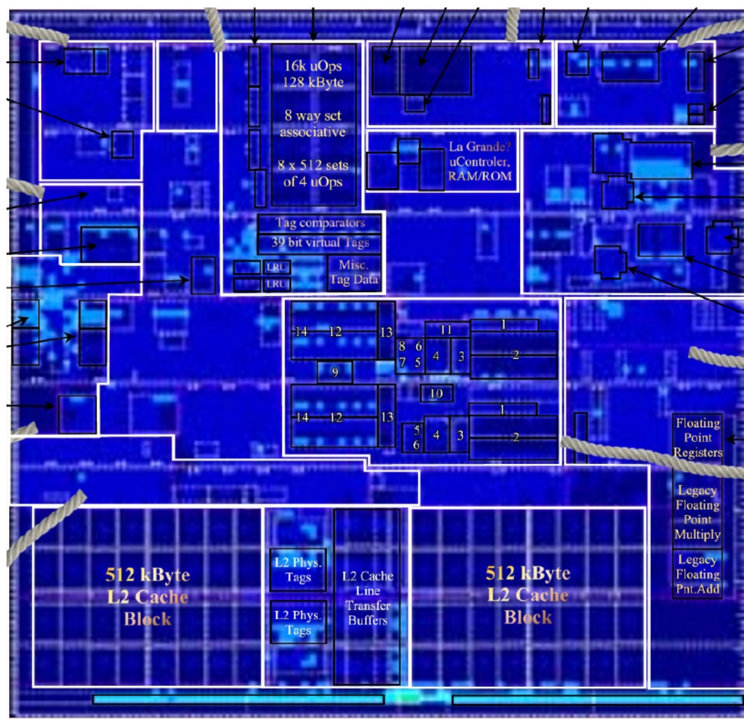


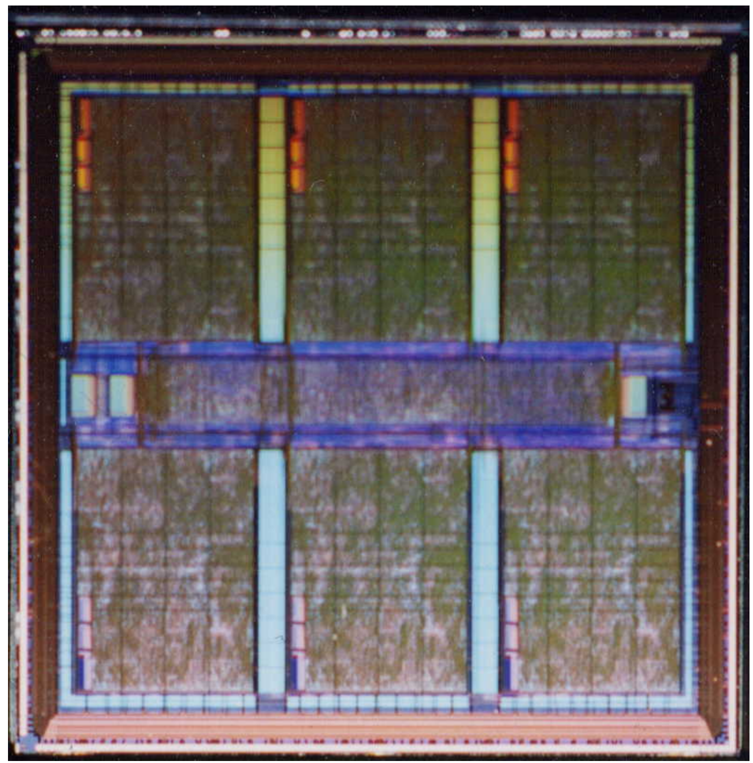
Number of transistors doubles every 18 months (*"Moore's Law"*)

Number of floating point units got stuck at $O(1)$. Never reached more than 4.

You can do much better than COTS microprocessor if you can use more than 10% of transistors for FP operations!

Intel P4 and GRAPE-6



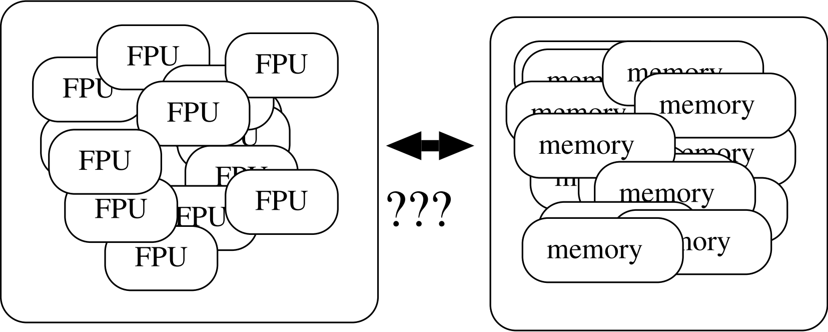


Intel Prescott (2004)
2 FP ops/clock
90nm, 7.6 GF, $>$ 100W?

GRAPE-6 chip (2000)
$\sim$ 400 FP ops/clock
0.25μm, 31 GF, 10W

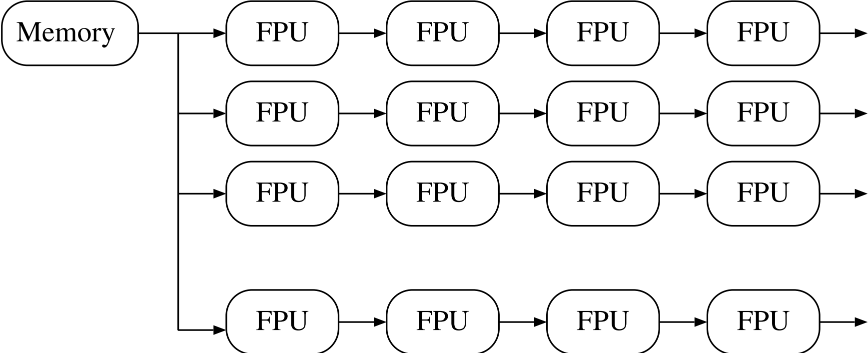
The GRAPE approach

General-purpose



How to connect processors and memories???

GRAPE



Hardwired pipeline
One memory serves many pipelines

Problem with GRAPE approach

- Chip development cost becomes too high.

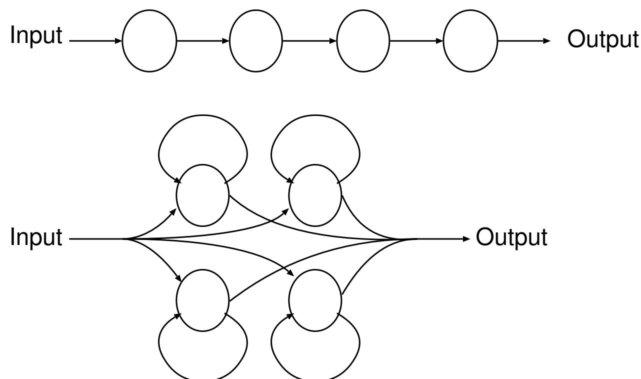
Year	Machine	Chip NRE
1992	GRAPE-4	200K$
1997	GRAPE-6	1M$
2004	GRAPE-DR	4M$

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
- Planned completion year: FY 2008 (early 2009)

Difference from previous GRAPE architecture



- **No hardwired pipeline, simple SIMD parallel processor.**

  Development codename: SING (*S*ing *i*s *n*ot *G*RAPE) (Eiichiro Kokubo)

- **Much like the Connection Machine**

- **Performance hit: factor 3-10? (We'll see)**

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.

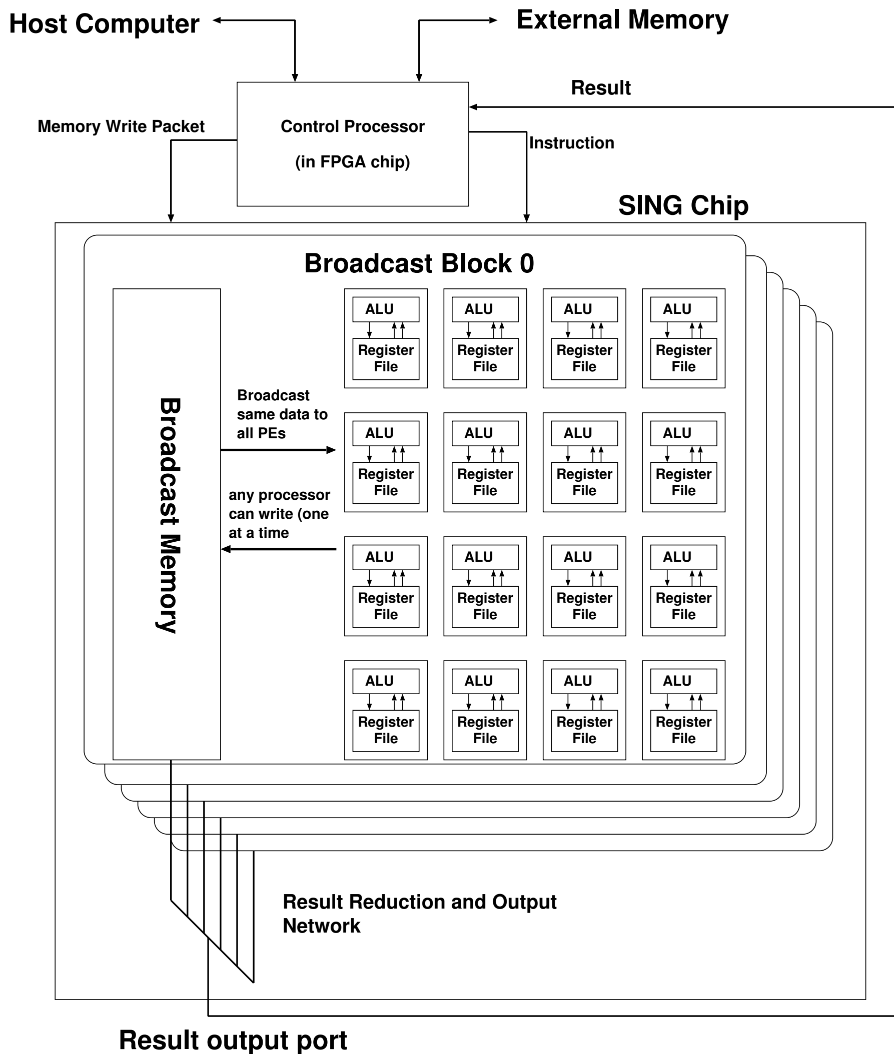
Why we changed the architecture?

- **To get budget (N-body problem is too narrow...)**
- **To allow wider range of applications**
  - **Molecular Dynamics**
  - **Boundary Element method**
  - **Dense matrix computation (Linpack, 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**

GRAPE-DR processor structure

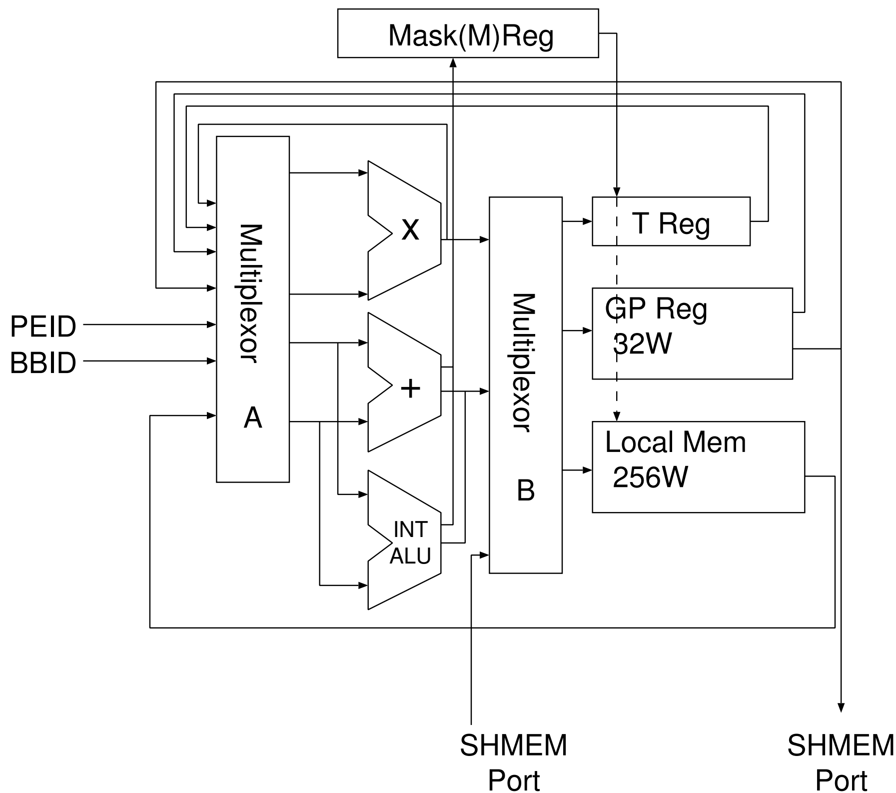


Collection of small processor, each with ALU, register file (local memory)

One chip will integrate 512 processors
Single processor will run at 500MHz clock (2 operations/cycle).

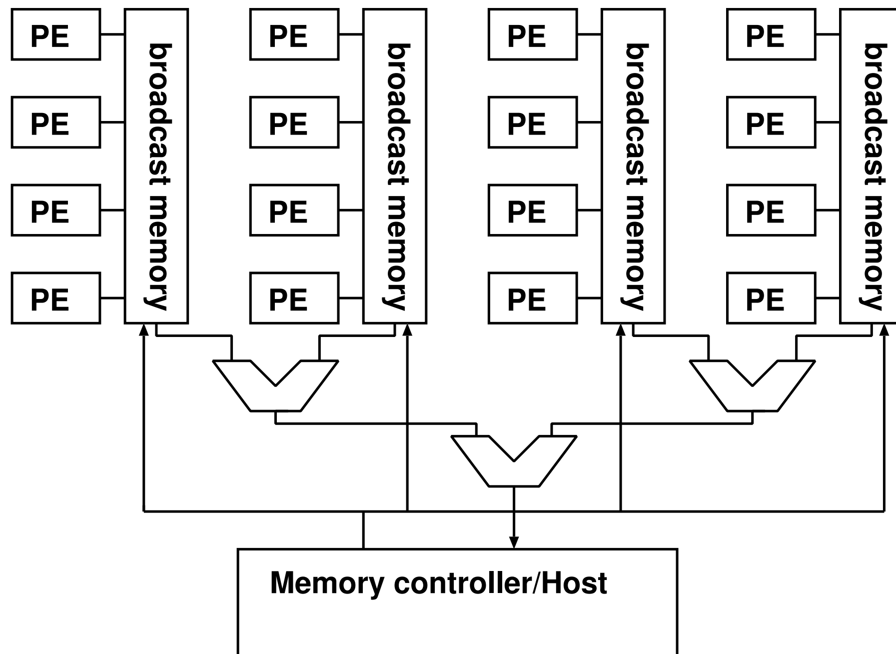
Peak speed of one chip: 0.5 Tflops (20 times faster than GRAPE-6).

PE 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)

Top-level architecture



- two-level hierarchy
- External memory $\rightarrow$ bloadcast memory
- bloadcast memorry $\rightarrow$ reduction tree $\rightarrow$ output port
- Essentially the same as the board-level structure of large GRAPE-6 board.

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**

How do you really use it?

Machine language: 108 bits horizontal microcode

```
DUM l m m m t t t t r r  r r r  r r r  r r r l l  l l l l f f f f f f f f f f f f f f f f f f  i i f b  b b
DUM l _ _ _ _ _ _ _ _ _  _ _ _  _ _ _  _ _ _ _ _  _ _ _ _ m m m m m m m m m m m m a a a a a a  a a s m  m m
DUM : i o i w l s i w i  w w w  r r r  r r r w i  a a t w u u u u u u u u u u u u d d d d d d  l l e _  _ _
DUM : m m f r m h s r s  a a w  a a w  a a w r s  d d r l l l l l l l l l l l l l d d d d d d  u u l w  a p
DUM : r r s i a o e i e  d d l  d d l  d d l i e  r r e : _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _  _ _ : r  d e
DUM : : : e t d r l t l  r r :  r r a  r r b t l  : i g : s s r n s s r n r n i i n n s r n i i  i u : i  r a
DUM : : : l e r t : e :  : i :  a i :  b i : e :  : : a : h h o o h h o o o o s s o o i o o s s  a n : t  : d
DUM : : : : : : s : : :  : : :  : a :  : b : : :  : : d : i i u r i i u r u r e e r r g u r e e  l s : e  : r
DUM : : : : : : t : : :  : : :  : : :  : : : : :  : : r : f f n m f f n m n m l l m m n n m l l  u i : :  : :
DUM : : : : : : o : : :  : : :  : : :  : : : : :  : : : : t t d a t t d a d a a b a a b d a a b  o g : :  : :
DUM : : : : : : p : : :  : : :  : : :  : : : : :  : : : : 2 5 a l 2 5 b l : l : : l l : : l : :  p n : :  : :
DUM : : : : : : : : : :  : : :  : : :  : : : : :  : : : : 5 0 : a 5 0 : b : o : : a b : : o : :  : e : :  : :
DUM : : : : : : : : : :  : : :  : : :  : : : : :  : : : : a a : : b b : : : : : : : : : : : : :  : d : :  : :
ISP 1 0 0 0 0 0 0 0 1 1  0 0 1  0 0 0  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 3  A 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  0 1 1  0 1 1  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  2 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  2 0 1  0 0 0  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 3  A 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  4 1 1  0 1 1  2 1 1 0 2  0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3  0 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 0 0  0 0 1  4 1 1  0 0 0 0 0  0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3  A 0 0 1  0 0
DUM
DUM IDP header format: IDP len addr bbn bbnmask, all in hex
DUM RRN format
DUM ADDR N BBADR REDUC WL FSEL NA NB SB RND NO OP UN ODP SREGEN
IDP 1 1000  0 0
RRN 0    1  0     0    1   0   0  0   0  0  0  0   0  1    1
IDP 1 1000  0 0
RRN 0    1  0     1    1   0   0  0   0  0  0  0   0  1    1
```

How do you really use it?

Machine language: 108 bits horizontal microcode

```
DUM l m m m t t t t r r  r r r  r r r  r r r l l  l l l l f f f f f f f f f f f f f f f f f f f  i i f b  b b
DUM l _ _ _ _ _ _ _ _ _  _ _ _  _ _ _  _ _ _ _ _  _ _ _ _ m m m m m m m m m m m m a a a a a a a  a a s m  m m
DUM : i o i w l s i w i  w w w  r r r  r r r w i  a a t w u u u u u u u u u u u u d d d d d d d  l l e _  _ _
DUM : m m f r m h s r s  a a w  a a w  a a w r s  d d r l l l l l l l l l l l l l d d d d d d d  u u l w  a p
DUM : r r s i a o e i e  d d l  d d l  d d l i e  r r e : _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _  _ _ : r  d e
DUM : : : e t d r l t l  r r :  r r a  r r b t l  : i g : s s r n s s r n r n i i n n s r n i i  i u : i  r a
DUM : : : l e r t : e :  : i :  a i :  b i : e :  : : a : h h o o h h o o o o s s o o i o o s s  a n : t  : d
DUM : : : : : : s : : :  : : :  : a :  : b : : :  : : d : i i u r i i u r u r e e r r g u r e e  l s : e  : r
DUM : : : : : : t : : :  : : :  : : :  : : : : :  : : r : f f n m f f n m n m l l m m n n m l l  u i : :  : :
DUM : : : : : : o : : :  : : :  : : :  : : : : :  : : : : t t d a t t d a d a a b a a b d a a b  o g : :  : :
DUM : : : : : : p : : :  : : :  : : :  : : : : :  : : : : 2 5 a l 2 5 b l : l : : l l : : l : :  p n : :  : :
DUM : : : : : : : : : :  : : :  : : :  : : : : :  : : : : 5 0 : a 5 0 : b : o : : a b : : o : :  : e : :  : :
DUM : : : : : : : : : :  : : :  : : :  : : : : :  : : : : a a : : b b : : : : : : : : : : : : :  : d : :  : :
ISP 1 0 0 0 0 0 0 0 1 1  0 0 1  0 0 0  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 3  A 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  0 1 1  0 1 1  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  2 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  2 0 1  0 0 0  0 0 0 0 2  2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 3  A 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 1 1  4 1 1  0 1 1  2 1 1 0 2  0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3  0 0 0 0  0 0
ISP 1 0 0 0 0 0 0 0 0 0  0 0 1  4 1 1  0 0 0 0 0  0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3  A 0 0 1  0 0
DUM
DUM IDP header format: IDP len addr bbn bbnmask, all in hex
DUM RRN format
DUM ADDR N BBADR REDUC WL FSEL NA NB SB RND NO OP UN ODP SREGEN
IDP 1 1000  0 0
RRN 0    1  0     0    1   0   0  0   0  0  0  0   0  1    1
IDP 1 1000  0 0
RRN 0    1  0     1    1   0   0  0   0  0  0  0   0  1    1
```

Probably not the way you want to program.

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 accumurate**
5. **retrieve the result**

This model, with some extention, 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 declerations
- Compiler on top of this environment

Example: acceleration and jerk

- **definition of "$\boldsymbol{i}$"s**
- **definition of "$\boldsymbol{j}$"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 conversin 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
alias dt $lr0
alias dxp  $r32v
alias ls18v  $r2v
alias lj6v $r24v
alias la2v $r20v
alias lv0v $r36v
alias lx0v $lr12v
bm tj $lr0v
vlen 3
bm jx6 lj6v
bm ax2 la2v
bm v0x lv0v
```

```
fsub ti $lr0 $t
upassa $ti $ti  dt  dtlm
nop
fmul $t ls18v $t
fmul $ti f"0.75" ; fadd $ti lj6v
fmul $ti f"1.5" $t ; fadd $fb lj
fmul $ti dt ;  bm x0 lx0v
fadd $fb la2v $t ; fmul dxp dtlm
fadd $fb la2v $t ; fmul $ti dt
fmul $ti dt       ; upassa $fb $t
fadd $fb lv0v $t ; fmul $ti f"2.
fmul $ti dt ;       fadd $fb lv0v
fadd $fb lx0v $t lx0v
```

Getting other j data etc

```
vlen 1
bm mj  lmj
bm eps2 leps2
bm idxj lidxj
nop
nop
upassa il"36" $t $t
ulsr lidxj $ti $t
upassa $ti $ti llidxj
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"000fffff" $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 declaretion (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.**