GRAPE-DR and Next-Generation GRAPE

Jun Makino
Center for Computational Astrophysics
and
Division Theoretical Astronomy
National Astronomical Observatory of Japan

Accelerated Computing Jan 28, 2010

Who am I?

Current position: Director, Center for Computational Astrophysics (CfCA), National Astronomical Observatory of Japan
CfCA computers: Cray XT4 (812 quad-core nodes), NEC SX-9, several GRAPE hardwares....

What I have been doing for the last 20 years:
Developing GRAPE and similar hardwares for astrophysical N-body simulations, using them for research.

Talk structure

- Short history of GRAPE
  - GRAPE machines
- GRAPE-DR
  - Architecture
  - Comparison with other architecture
  - Development status
- Next-Generation GRAPE
  - Future of accelerators

Claims by the organizer:

"Accelerated Computing" is an old concept that is recently redefined in High-Performance Computing. It was started by dedicated machines like GRAPEs, but a great revolution has been occurring fueled by recent advancement in GPU Computing, both in hardware and in software such as CUDA C and OpenCL.

"Accelerated Computing" という言葉は以前から使われていましたが、最近再定義されつつあります。かつての GRAPE のような専用演算器から始まり、ここ数年の GPU コンピューティングのめざましい性能の向上によって、ハードウェアのみならず CUDA C や OpenCL のようなソフトウェアにおいても革新がもたらされました。

Claims by the organizer:

"Accelerated Computing" is an old concept that is recently redefined in High-Performance Computing. It was started by dedicated machines like GRAPEs, but a great revolution has been occurring fueled by recent advancement in GPU Computing, both in hardware and in software such as CUDA C and OpenCL.

"Accelerated Computing" という言葉は以前から使われていましたが、最近再定義されつつあります。かつての GRAPE のような専用演算器から始まり、ここ数年の GPU コンピューティングのめざましい性能の向上によって、ハードウェアのみならず CUDA C や OpenCL のようなソフトウェアにおいても革新がもたらされました。

GRAPE is past, GPGPU is future!

Claims by the organizer:

"Accelerated Computing" is an old concept that is recently redefined in High-Performance Computing. It was started by dedicated machines like GRAPEs, but a great revolution has been occurring fueled by recent advancement in GPU Computing, both in hardware and in software such as CUDA C and OpenCL.

"Accelerated Computing" という言葉は以前から使われていましたが、最近再定義されつつあります。かつての GRAPE のような専用演算器から始まり、ここ数年の GPU コンピューティングのめざましい性能の向上によって、ハードウェアのみならず CUDA C や OpenCL のようなソフトウェアにおいても革新がもたらされました。

GRAPE is past, GPGPU is future!

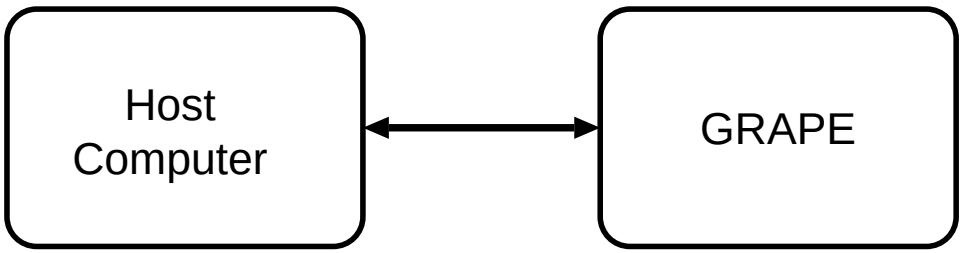
You can see I'd disagree.

Short history of GRAPE

- Basic concept
- GRAPE-1 through 6
- Software Perspective

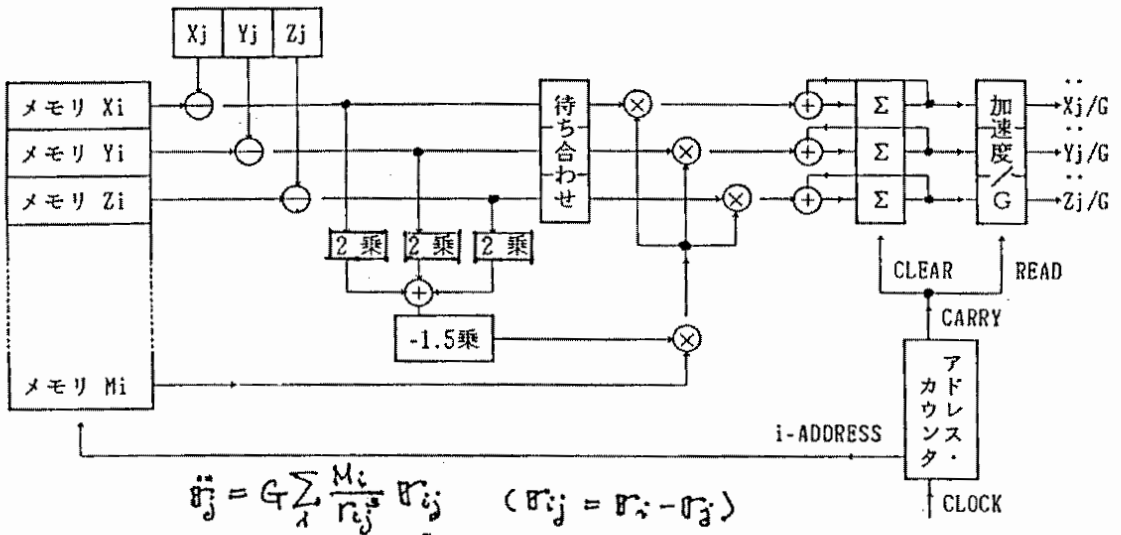
Basic concept (As of 1988)

- **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 calculates the particle-particle interaction can accelerate overall calculation.**
- **Original Idea: Chikada (1988)**



Accelerated Computing two decades ago

Chikada's idea (1988)



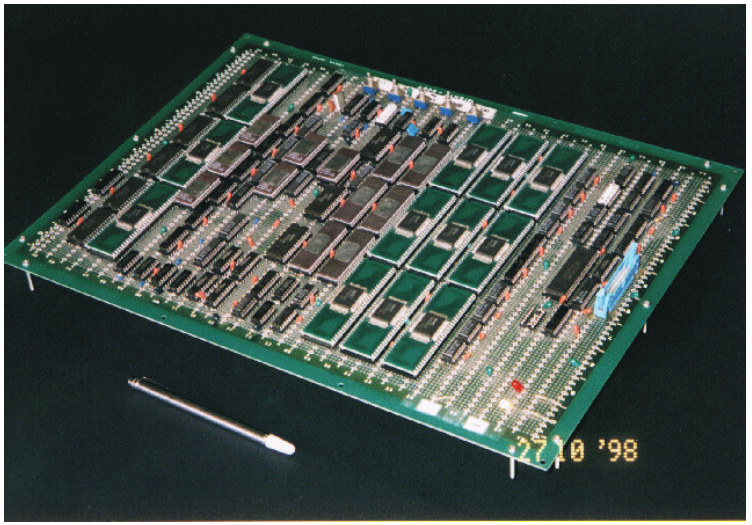
$$\ddot{\vec{r}}_j = G\sum_i \frac{M_i}{r_{ij}^3}\vec{r}_{ij} \qquad (\vec{r}_{ij} = \vec{r}_i - \vec{r}_j)$$

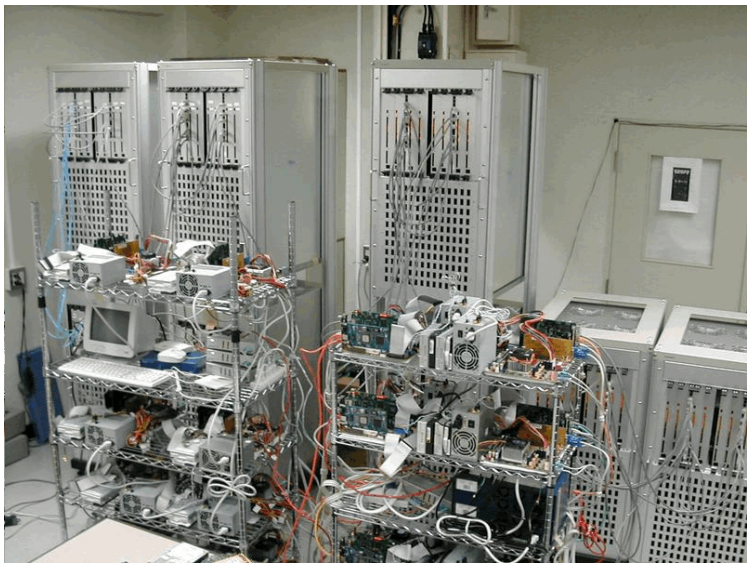
+, −, ×, 2乗は1 operation, -1.5乗は多項式近似でやるとして10operation 位に相当する.
総計24operation.
各operation の後にはレジスタがあって, 全体がpipelineになっているものとする.
「待ち合わせ」は2乗してMと掛け算する間の時間ズレを補正するためのFIFO(First-In First-Out memory).
「Σ」は足し込み用のレジスタ. N回足した後結果を右のレジスタに転送する.

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.

- Hardwired pipeline for force calculation (similar to Delft DMDP)
- Hybrid Architecture (things other than force calculation done elsewhere)

GRAPE-1 to GRAPE-6



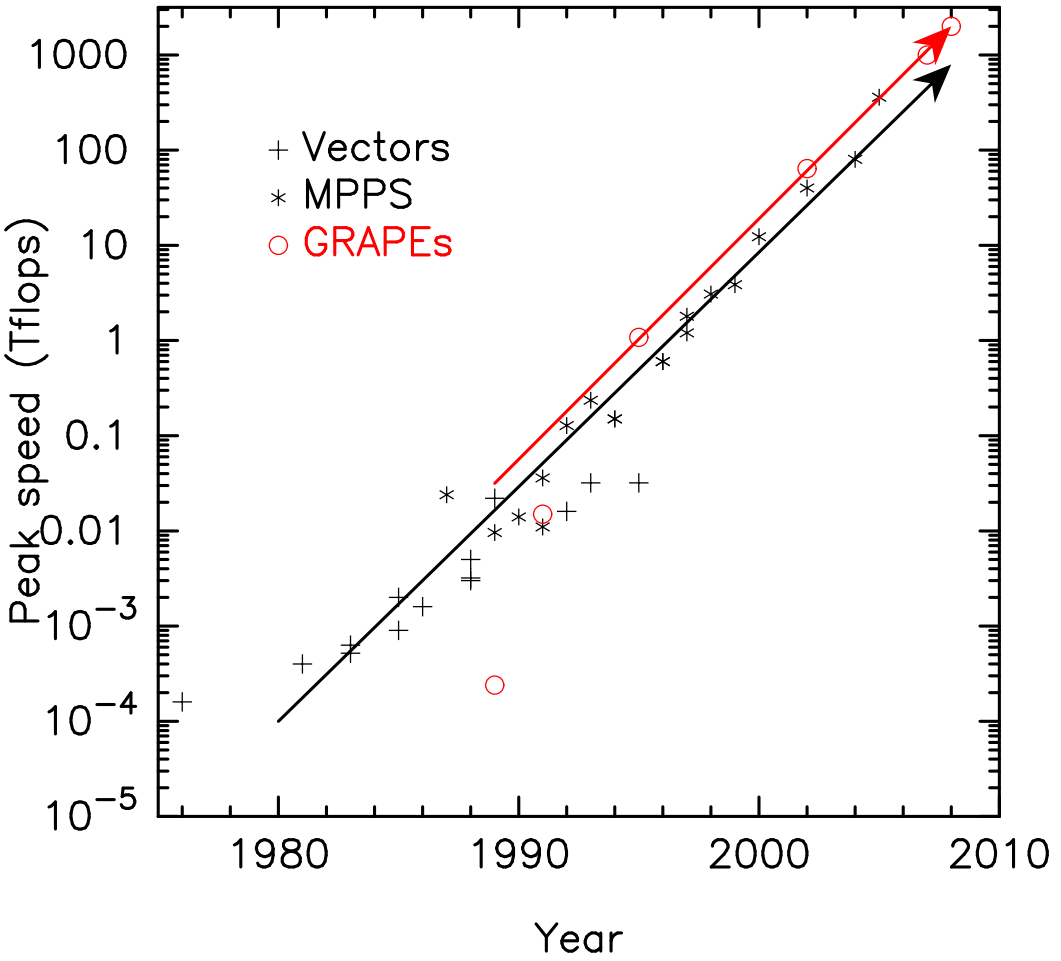


GRAPE-1: 1989, 308Mflops

GRAPE-4: 1995, 1.08Tflops

GRAPE-6: 2002, 64Tflops

Performance history



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

Development cost was around 1/100.

Software development for GRAPE

GRAPE software library provides several basic functions to use GRAPE hardware.

- **Sends particles to GRAPE board memory**
- **Sends positions to calculate the force and start calculation**
- **get the calculated force (asynchronous)**

User application programs use these functions. Algorithm modifications (on program) are necessary to reduce communication and increase the degree of parallelism

Analogy to BLAS

Level	BLAS	Calc:Comm	Gravity	
0	`c=c-a*s`	1:1	$f_{ij} = f(x_i, x_j)$	1:1
1	AXPY	$N : N$	$f_i = \Sigma_j f(x_i, x_j)$	$N : N$
2	GEMV	$N^2 : N^2$	$f_i = \Sigma_j f(x_i, x_j)$ for multiple i	$N^2 : N$
3	GEMM	$N^3 : N^2$	$f_{k,i} = \Sigma_j f(x_{k,i}, x_{k,j})$ "Multiwalk"	$N^2 : N$

- Calc $\gg$Comm essential for accelerator
- Level-3 (matrix-matrix) essential for BLAS
- Level-2 like (vector-vector) enough for gravity
- Treecode and/or short-range force might need Level-3 like API.

Porting issues

- Libraries for GRAPE-4 and 6 (for example) are not compatible
- Even so, porting was not so hard. The calls to GRAPE libraries are limited to a fairly small number of places in application codes.
- Backporting the GRAPE-oriented code to CPU-only code is easy, and allows very efficient use of SIMD features.
- In principle the same for GPGPU or other accelerators.

Real-World issues with “Porting”

— Mostly on GPGPU....

- Getting something run on GPU is not difficult
- Getting a good performance number compared with non-optimized, single-core x86 performance is not so hard. (20x!, 120x!)

Real-World issues with "Porting" continued

- Making it faster than 10-year-old GRAPE or highly-optimized code on x86 (using SSE/SSE2) is *VERY, VERY HARD* (you need Keigo or Evghenii...)
- These are *mostly* software issues
- Some of the most serious ones are limitations in the architecture (lack of good reduction operation over processors etc)

I'll return to this issue later.

Quotes

From: Twelve Ways to Fool the Masses When Giving Performance Results on ~~Accelerators~~ Parallel Computers (D. H. Bailey, 1991)

1. Quote only 32-bit performance results, not 64-bit results.
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
6. Compare your results against scalar, unoptimized code on ~~Xeons~~ Crays.
7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

History repeats itself — Karl Marx

"Problem" with GRAPE approach

- Chip development cost becomes too high.

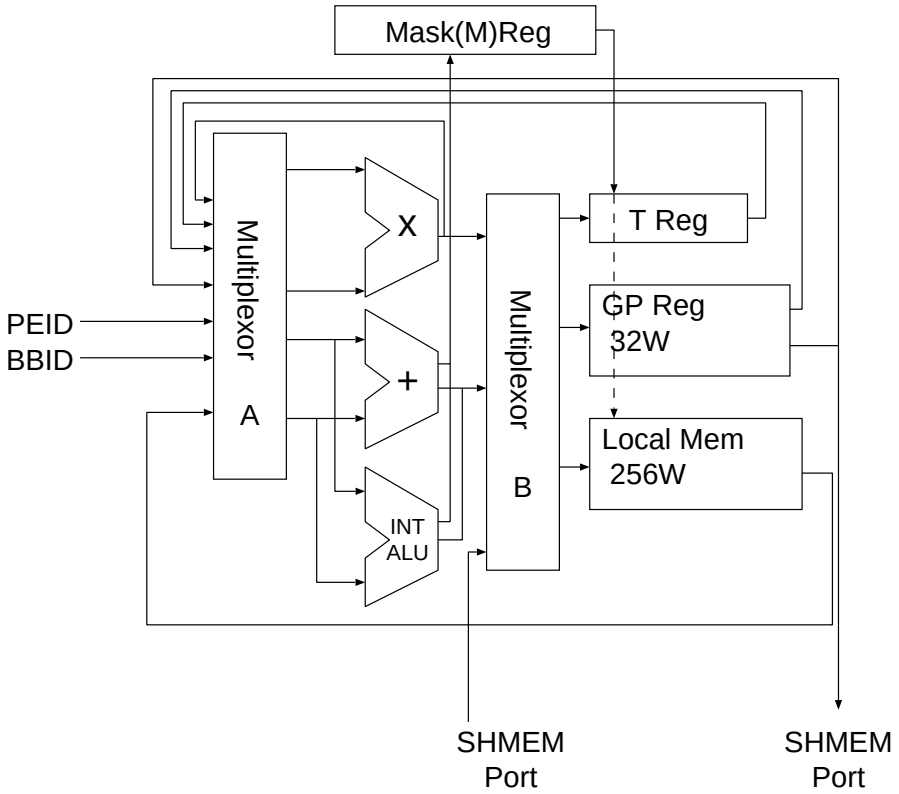
Year	Machine	Chip initial cost	process
1992	GRAPE-4	200K$	1μm
1997	GRAPE-6	1M$	250nm
2004	GRAPE-DR	4M$	90nm
2010?	GDR2?	$>$ 10M$	45nm?

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

Next-Generation GRAPE — GRAPE-DR

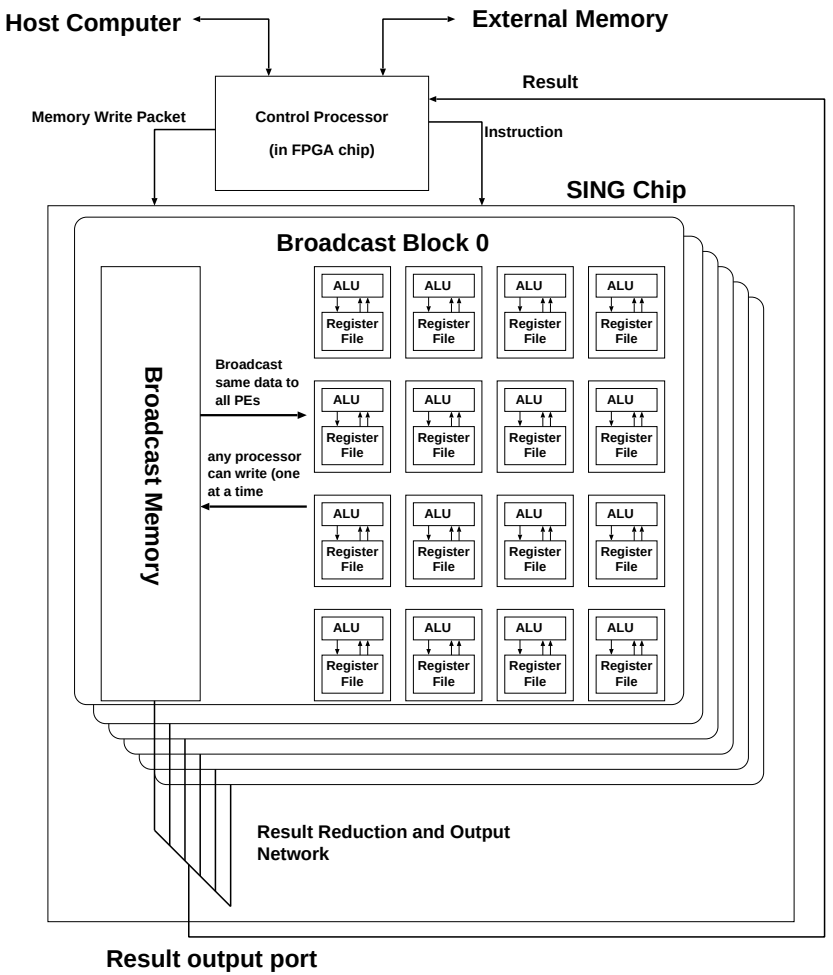
- Planned peak speed: peak 2 Pflops SP/1Pflops DP
- New architecture — wider application range than previous GRAPEs
- primarily to get funded
- No force pipeline. SIMD programmable processor
- Completion year: FY 2008-2009

Processor architecture



- Float Mult
- Float add/sub
- Integer ALU
- 32-word registers
- 256-word memory
- communication port

Chip architecture



- 32 PEs organized to "broadcast block" (BB)
- BB has shared memory. Various reduction operation can be applied to the output from BBs using reduction tree.
- Input data is broadcasted to all BBs.
- "Solved" data movement problem: Very small number of long wires and off-chip IO.

Computation Model

Parallel evaluation of

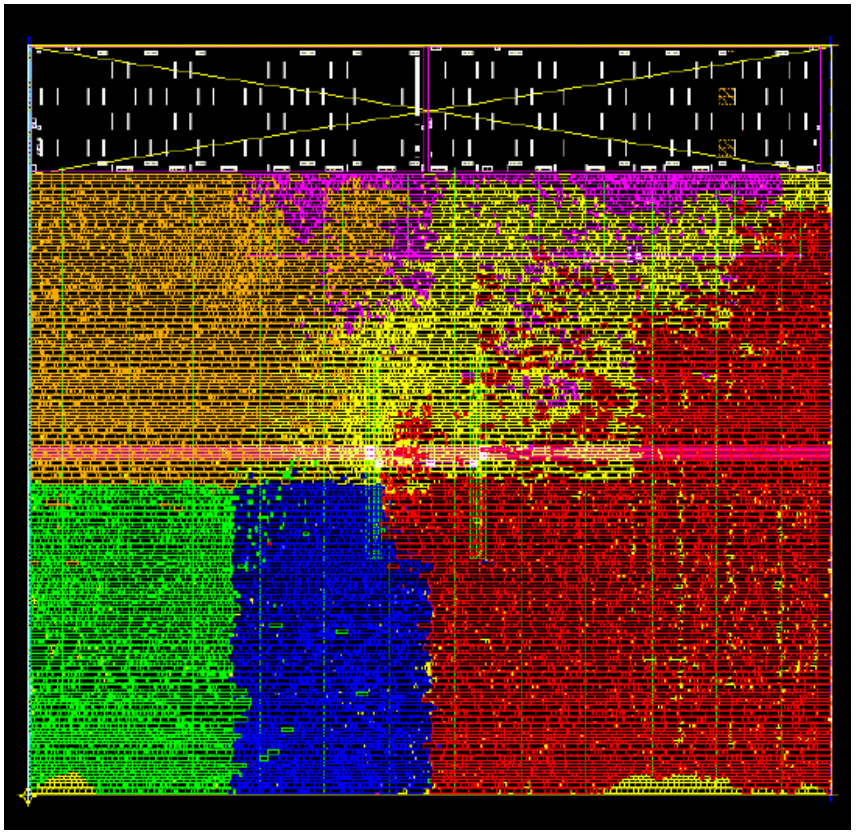
$$\boldsymbol{R}_i = \sum_j \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{y}_j)$$

- parallel over both i and j (Level-2 gravity)
- y_j may be omitted (trivial parallelism)
- $\boldsymbol{S}_{i,j} = \sum_k \boldsymbol{f}(\boldsymbol{x}_{i,k}, \boldsymbol{y}_{k,j})$ also possible (Level-3 BLAS)

The Chip

Sample chip delivered May 2006
90nm TSMC, Worst case 65W@500MHz

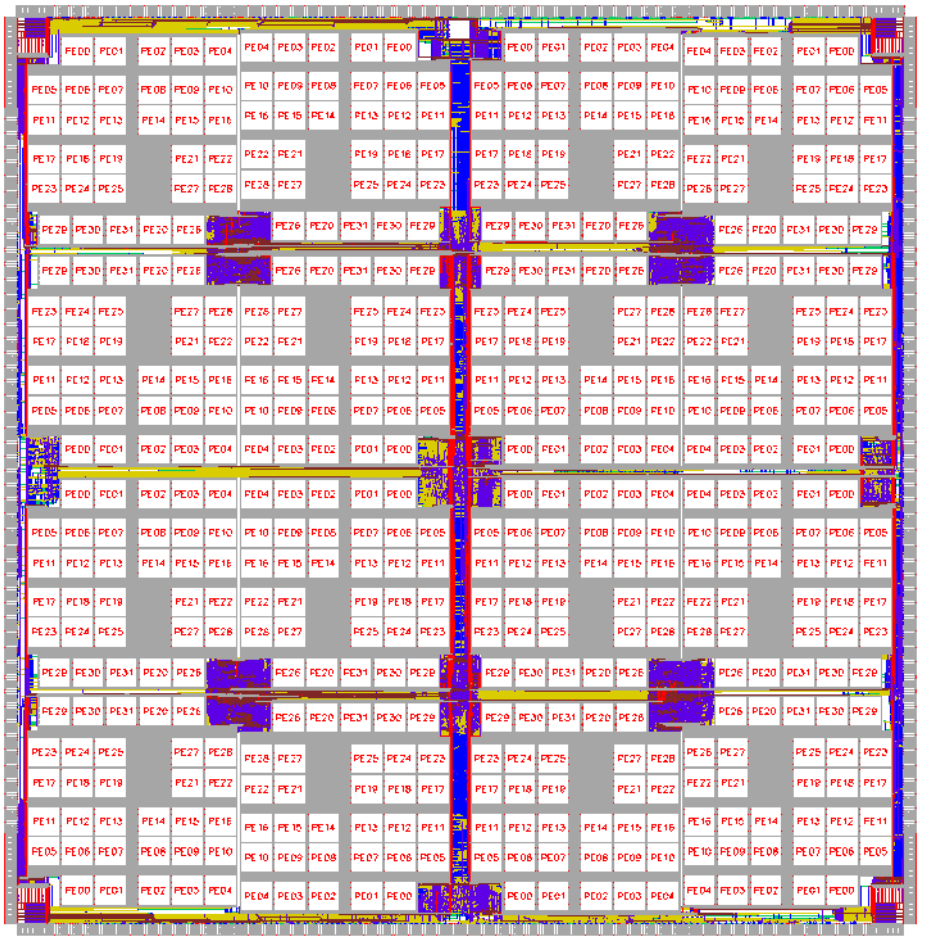
PE Layout



Black: Local Memory
Red: Reg. File
Orange: FMUL
Green: FADD
Blue: IALU

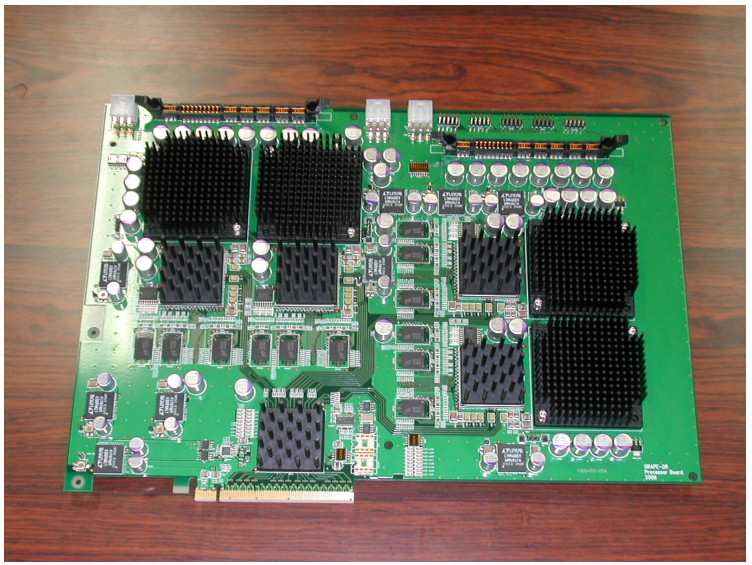
0.7mm by 0.7mm
800K transistors
0.13W@500MHz
1Gflops/512Mflops peak (SP/DP)

Chip layout



- 16 blocks with 32PEs each
- Shared memory within blocks
- 18mm by 18mm chip size

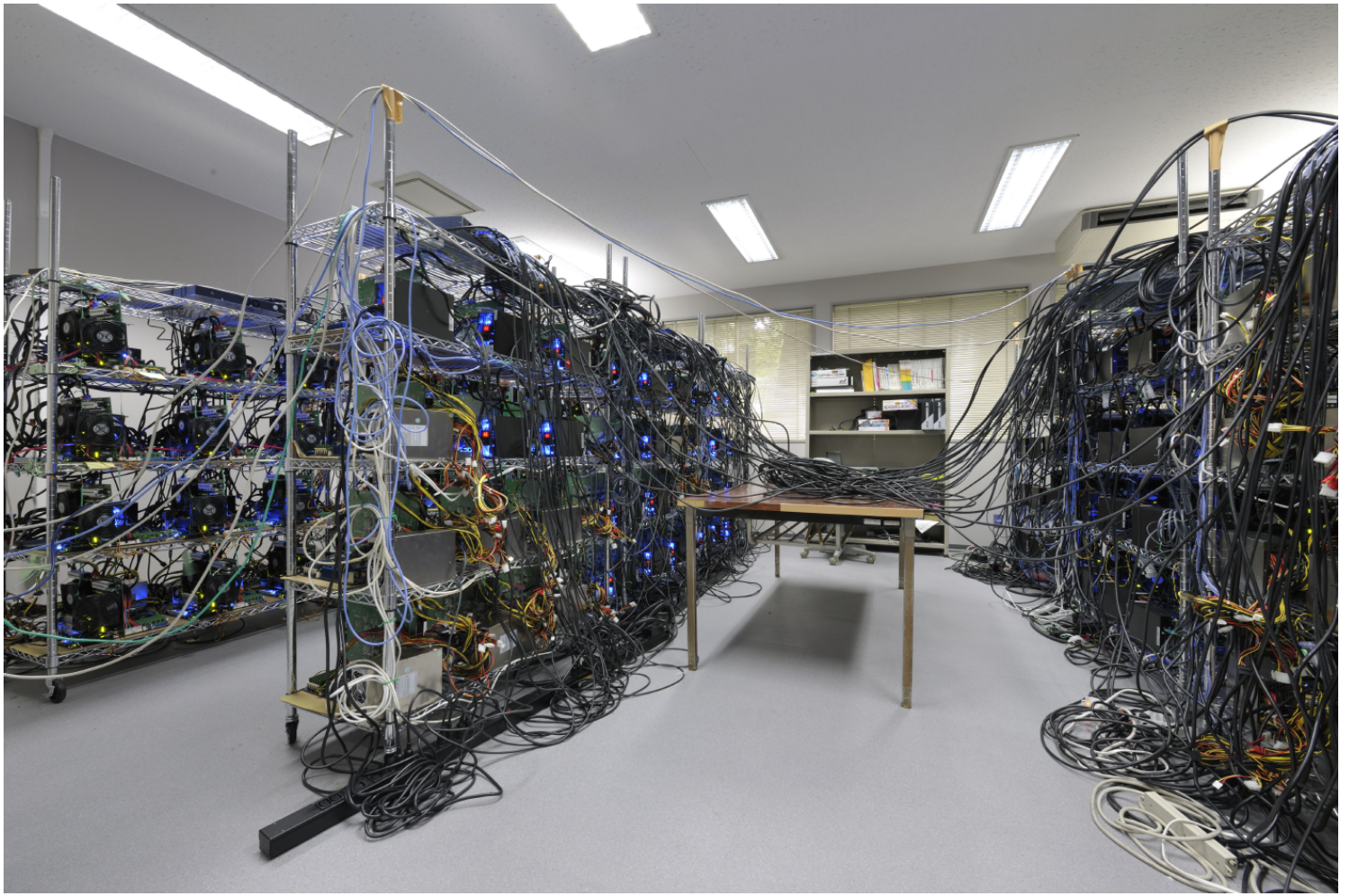
Processor board



PCIe x16 (Gen 1) interface
Altera Arria GX as DRAM controller/communication interface

- Around 200W power consumption
- 800Gflops DP peak (400MHz clock)
- Available from K&F Computing Research

GRAPE-DR cluster system



GRAPE-DR cluster system

- 128-node, 128-card system (105TF theoretical peak @ 400MHz)
- Linpack measured: 24 Tflops@400MHz (with HPL 1.04a. still lots of tunings necessary....)
- Gravity code: 340Gflops/chip, working
- Host computer: Intel Core i7+X58 chipset, 12GB memory
- network: x4 DDR Infiniband
- plan to expand to 384-board system RSN. (Cables and switches are arriving now.)

Software Environment

- Kernel libraries
  - DGEMM
    - BLAS, LAPACK
  - Particle-Particle interaction
- Assembly Language
- HLL, OpenMP-like interface

Idea based on PGDL (Hamada, Nakasato)
— pipeline generator for FPGA

HLL example

Nakasato (2008), based on LLVM.

```
VARI xi, yi, zi;
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;
rinv = rsqrt(r2);
mr3inv = rinv*rinv*rinv*mj;
fx+=  mr3inv*dx;
fy+=  mr3inv*dy;
fz+=  mr3inv*dz;
```

Driver functions

Generated from the description in the previous slide

```
int SING_send_j_particle(struct grape_j_particle_struct *jp,
                         int index_in_EM);
int SING_send_i_particle(struct grape_i_particle_struct *ip,
                         int n);
int SING_get_result(struct grape_result_struct *rp);
void SING_grape_init();
int SING_grape_run(int n);
```

DGEMM kernel in assembly language (part of)

```
## even loop
bm b10 $lr0v
bm b11 $lr8v
dmul0 $lr0 $lm0v    ;   bm $lr32v c0 0      ; rrn fadd c0 256 flt72t
dmul1 $lr0 $lm0v    ; upassa $fb $t $t      ; idp 0
dmul0 $lr0 $lm256v  ; faddAB $fb $ti $lr48v ; bm $lr40v c1 0
dmul1 $lr0 $lm256v  ; upassa $fb $t $t
dmul0 $lr2 $lm8v    ; faddAB $fb $ti $lr56v ; bm $lr32v c2 1
dmul1 $lr2 $lm8v    ; faddA $fb $lr48v $t
.....
dmul0 $lr14 $lm504v ; faddA $fb $ti $lr32v  ; bm $lr40v c63 31
dmul1 $lr14 $lm504v ; faddA $fb $lr56v $t
faddA $fb $ti $lr40v
nop
```

“VLIW”-style

OpenMP-like compiler

Goose compiler (Kawai 2009)

```
#pragma goose parallel for icnt(i) jcnt(j) res (a[i][0..2])
    for (i = 0; i < ni; i++) {
        for (j = 0; j < nj; j++) {
            double r2 = eps2[i];
            for (k = 0; k < 3; k++) dx[k] = x[j][k] - x[i][k];
            for (k = 0; k < 3; k++) r2 += dx[k]*dx[k];
            rinv = rsqrt(r2);
            mf = m[j]*rinv*rinv*rinv;
            for (k = 0; k < 3; k++)  a[i][k] += mf * dx[k];
        }
    }
```

Translated to assembly language and API calls.
Emit very efficient code also for GPGPUs or x86 SIMD extensions.

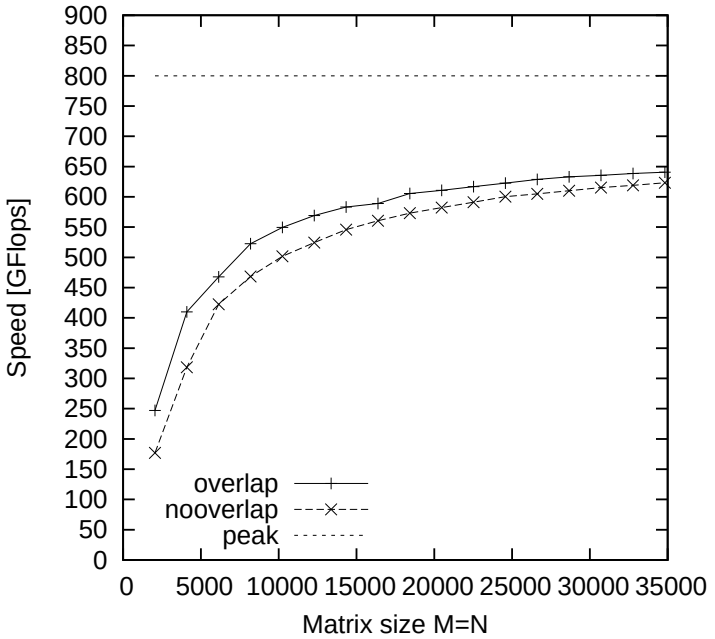
Performance and Tuning examples

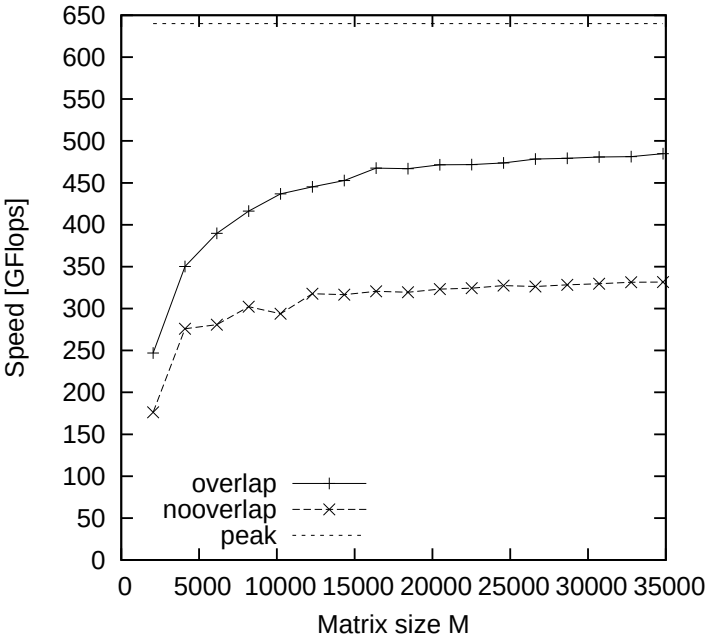
- **HPL (LU-decomposition)**
- **Gravity**

Based on the work by H. Koike (Thesis work)

LU-decomposition

DGEMM performance





M=N, K=2048, 640 Gflops N=K=2048, 450 Gflops

FASTEST single-chip and single-card performance on the planet. (HD5870/5970 might be faster...)

DGEMM tuning

Key to high performance: Overlapping communication and Calculation

- **PE kernel calculates C(8,2)= A(32,8) * B(8,2)**
- **512 PEs calculate C(256,2)= A(512,256)* B(512,2)**
- **Next B sent to chip while calculation**
- **Previous C sent to host while calculation**
- **Next A sent from host to GDR card while calculation**

Everything other than the transfer of B from host to GDR card is hidden.

What limits the HPL performance?

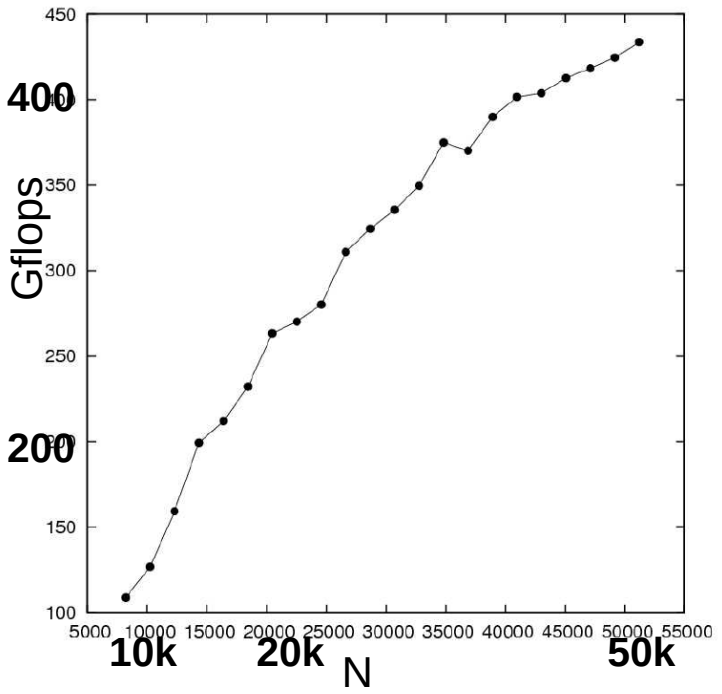
- CPU/Accelerator speed ratio
- CPU/Accelerator communication speed
- Node-node communication
- Size of the main memory

Large main memory can hide whatever performance problems for HPL benchmark.

Some numbers

Machine	speed (per node) Gflops	memory (per node) GB	ratio
Jaguar	125	32	4
Tianhe-1	200	32	6
FX-1	40	32	1.3
ES2	1600	1024	1.6
GDR	800	12	67

LU-decomposition performance



Speed in Gflops as function of Matrix size

430 Gflops (67% of raw DGEMM speed) for N=50K (24GB limit)

11x speedup over CPU (4 cores, Goto BLAS, highly tuned code faster than HPL).

Tianhe: 2.3x over CPU using HD4870

LU-decomposition tuning

- Almost every know techniques
  - except for the concurrent use of CPU and GDR (we use GDR for column factorization as well...)
  - right-looking form
  - TRSM converted to GEMM
  - use row-major order for fast $O(N^2)$ operations
- Several other "new" techniques
  - Transpose matrix during recursive column decomposition
  - Use recursive scheme for TRSM (calculation of L^{-1})

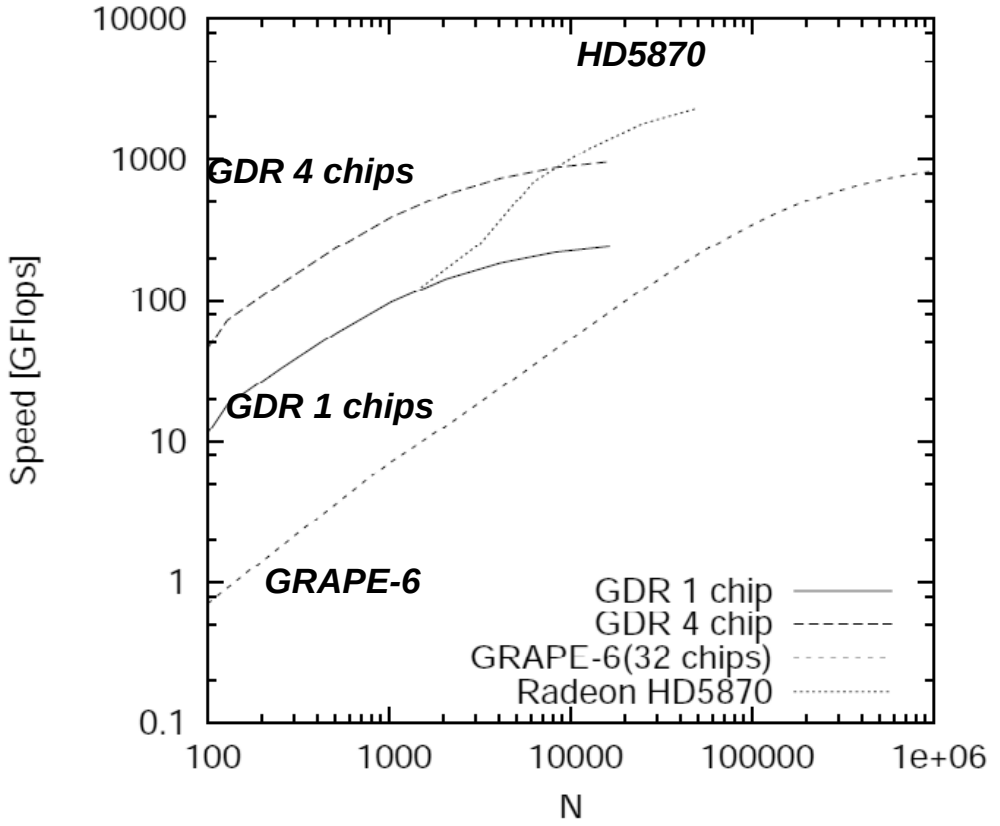
HPL (parallel LU) tuning

- **Everything done for single-node LU-decomposition**
- **Both column- and row-wise communication hidden**
- **TRSM further modified: calculate LT^{-1} instead of $T^{-1}U$**
- **More or less working, tuning still necessary**

Two months for coding and debugging so far.

N=30K, single node: 290Gflops
N=96K, 9 nodes: 2613 Gflops

Gravity kernel performance



Performance for small N much better than GPU

(for treecode, the multiwalk method greatly improves GPU performance, though)

Comparison with GPGPU

Pros:

- Significantly better silicon usage
  512PEs with 90nm
  40% of the peak DP speed of HD5870 with 1/2 clock and 1/5 transistors
- Better efficiency — Designed for scientific applications
  hardwired reduction, small communication overhead, etc

Cons:

- Higher cost per silicon area...
  (small production quantity)
- Longer product cycle... 5 years vs 1 year

Good implementations of N-body code on GPGPU are there (Hamada, Nitadori, ...)

GPGPU performance for N-body simulation

- Impressive for a trivial N^2 code with shared timestep (x100 performance!!!) — actually x10 compared to a good SSE code.
- $\sim$ x5 for production-level algorithms (tree or individual timestep), $\sim$ x2 or less for the same price, even when you buy GTX295 cards and not Tesla and after Keigo developed new algorithms (without him, who knows?).

GPGPU tuning difficulties

- huge overhead for DMA and starting threads (much longer than MPI latency with IB)
- lack of low-latency communication between threads

GRAPE and GRAPE-DR solution

- PIO for sending data and commands from host to GDR
- hardware support for broadcast and reduction
- a number of other small improvements

Near-peak performance with minimal bandwidth for both on-board memory and host.

Next-Generation GRAPE(-DR)

Question:
Any reason to continue hardware development?

- **GPUs are fast, and getting faster**
- **FPGAs are also growing in size and speed**
- **Custom ASICs practically impossible to make**

Next-Generation GRAPE

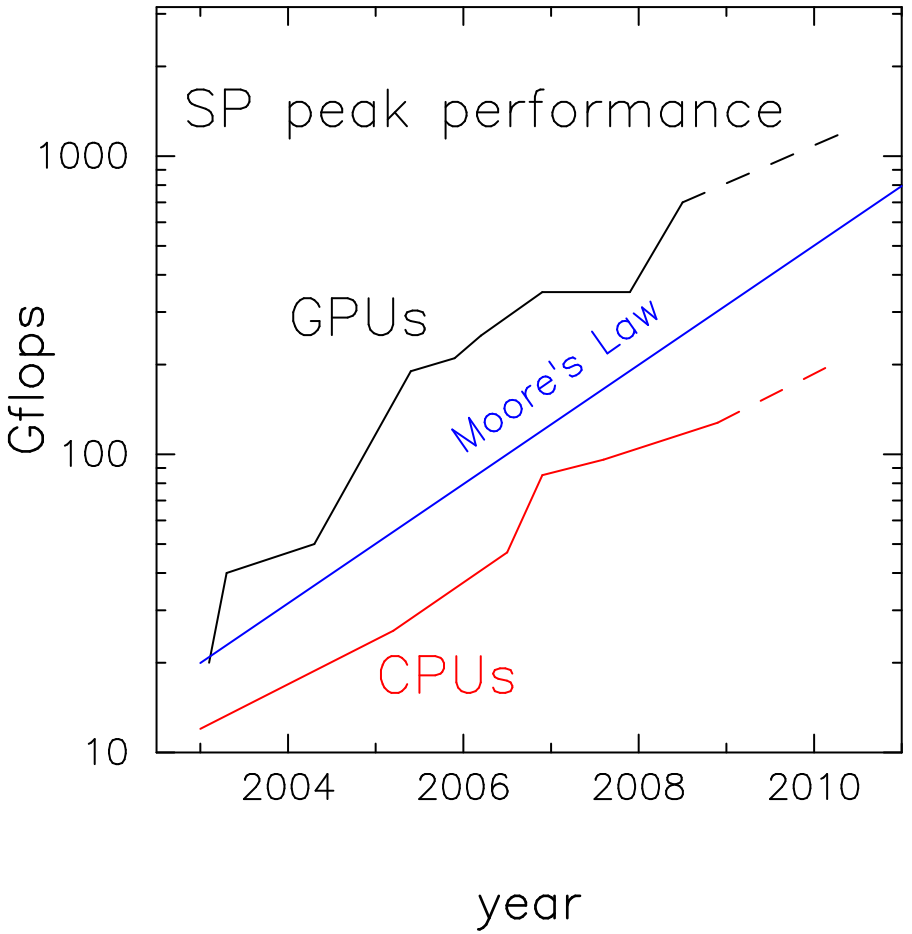
Question:
Any reason to continue hardware development?

- **GPUs are fast, and getting faster**
- **FPGAs are also growing in size and speed**
- **Custom ASICs practically impossible to make**

Answer?

- **GPU speed improvement might slow down**
- **FPGAs are becoming far too expensive**
- **Power consumption might become most critical**
- **Somewhat cheaper way to make custom chips**

GPU speed improvement slowing down?



Clear "slowing down" after 2006 (after G80)

Reason: shift to more general-purpose architecture

Discrete GPU market is eaten up by unified chipsets and unified CPU+GPU

Structured ASIC

- **Something between FPGA and ASIC**
- **From FPGA side: By using one or few masks for wiring, reduce the die size and power consumption by a factor of 3-4.**
- **eASIC: 90nm (Fujitsu) and 45nm (Chartered) products.**
- **45nm: up to 20M gates, 700MHz clock. 1/10 in size and 1/2 in the clock speed compared to ASIC. (1/3 in per-chip price)**
- **1/100 initial cost**

Will this be competitive?

Rule of thumb for a special-purpose computer project:

> **Price-performance should be more than 100 times better at the beginning of the project**

— x 10 for 5 year development time
— x 10 for 5 year lifetime

Compared to CPU: Okay
Compared to GPU: ???

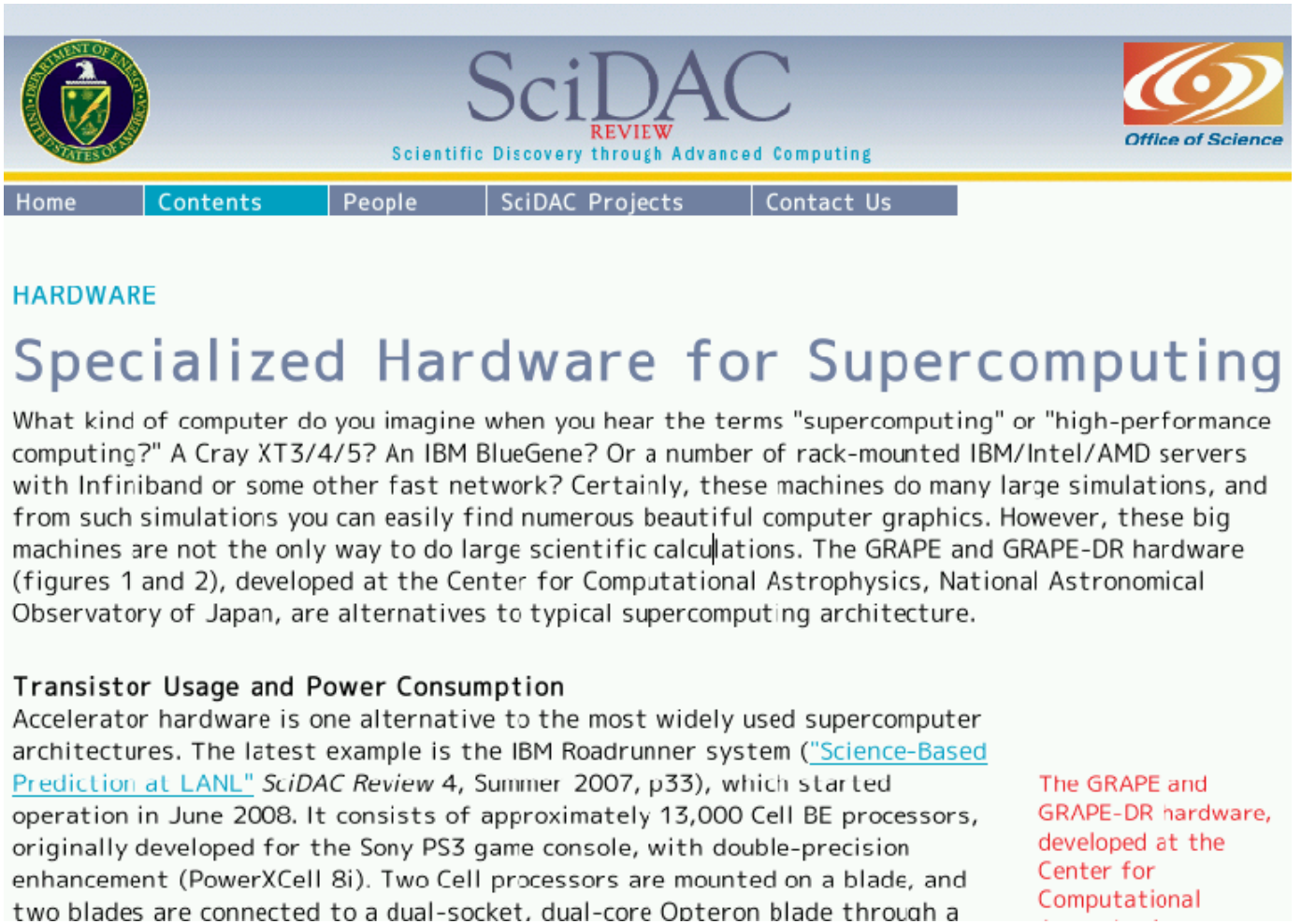
Will GPUs 10 years from now 100 times faster than today?

Summary

- **GRAPE-DR, with programmable processors, will have wider application range than traditional GRAPEs.**
- **A Small cluster of GDR is now up and running**
- **Peak speed of a card with 4 chips is 800 Gflops (DP).**
- **DGEMM performance 640 Gflops,**
  LU decomposition $>$ 400Gflops
- **Currently, 128-card, 512-chip system is up and running**
- **We might return to custom design with structured ASIC**

Further reading...

http://www.scidacreview.org/0902/html/hardware.html



SciDAC REVIEW
Scientific Discovery through Advanced Computing

Office of Science

Home | Contents | People | SciDAC Projects | Contact Us

HARDWARE

Specialized Hardware for Supercomputing

What kind of computer do you imagine when you hear the terms "supercomputing" or "high-performance computing?" A Cray XT3/4/5? An IBM BlueGene? Or a number of rack-mounted IBM/Intel/AMD servers with Infiniband or some other fast network? Certainly, these machines do many large simulations, and from such simulations you can easily find numerous beautiful computer graphics. However, these big machines are not the only way to do large scientific calculations. The GRAPE and GRAPE-DR hardware (figures 1 and 2), developed at the Center for Computational Astrophysics, National Astronomical Observatory of Japan, are alternatives to typical supercomputing architecture.

Transistor Usage and Power Consumption

Accelerator hardware is one alternative to the most widely used supercomputer architectures. The latest example is the IBM Roadrunner system ("Science-Based Prediction at LANL" *SciDAC Review* 4, Summer 2007, p33), which started operation in June 2008. It consists of approximately 13,000 Cell BE processors, originally developed for the Sony PS3 game console, with double-precision enhancement (PowerXCell 8i). Two Cell processors are mounted on a blade, and two blades are connected to a dual-socket, dual-core Opteron blade through a

The GRAPE and GRAPE-DR hardware, developed at the Center for Computational

Machine code

108-bit 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 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 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 w
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 l
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 1
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 1
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 1
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 1
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 1
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
```

Interface struct

```
struct grape_j_particle_struct{
  double xj;
  double yj;
  double zj;
  double mj;
};
struct grape_i_particle_struct{
  double xi;
  double yi;
  double zi;
};
struct grape_result_struct{
  double fx;
  double fy;
  double fz;
};
```

Unique feature as parallel language

- Only the inner kernel is specified
- Communication and data distribution are taken care of by hardware and library. User-written software does not need to care.

GRAPEs with eASIC

- Completed an experimental design of a programmable processor for quadruple-precision arithmetic. 6PEs in nominal 2.5Mgates.
- Started designing low-accuracy GRAPE hardware with 7.4Mgates chip.

Summary of planned specs:

- around 8-bit relative precision
- support for quadrupole moment in hardware
- 100-200 pipelines, 300MHz, 2-4Tflops/chip
- small power consumption: single PCIe card can house 4 chips (10 Tflops, 50W in total)

Will this be competitive?

Rule of thumb for a special-purpose computer project:

Price-performance should be more than 100 times better at the beginning of the project

— x 10 for 5 year development time
— x 10 for 5 year lifetime

Compared to CPU: Okay
Compared to GPU: ???