Application of Many-core Accelerators for Problems in Astronomy and Physics

N.Nakasato (University of Aizu, Japan)

in collaboration with

F.Yuasa, T.Ishikawa, J.Makino, H.Daisaka

Agenda

- Our Problems
- Recent Development of Many-core Accelerator Systems
- Our Approach to the problems
- Performance evaluation
- Summary

Particle Simulations

- Simulate evolution of the universe
  - As a collection of particles
  - Depending on scale, each particle represents
    - Galaxy
    - Star
    - Asteroid
    - Gas blob etc.
  - Particles are interacting
    - Mainly by gravity
      - Long-range force

Numerical Modeling

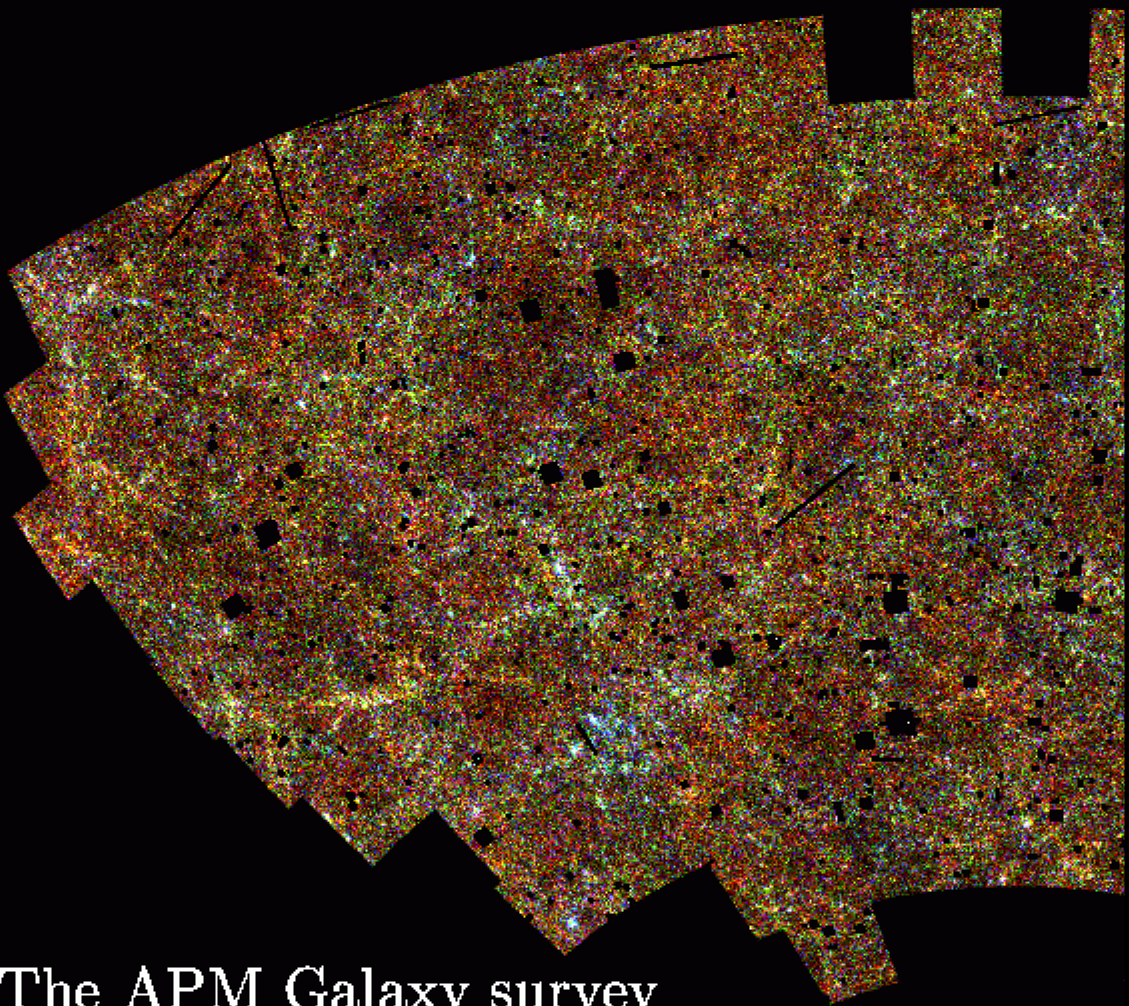
- Solve ODE for many particles

$$\frac{d\overrightarrow{v_i}}{dt} = \sum_{j=1}^{N} \vec{f}(\vec{r}_i - \vec{r}_j)$$

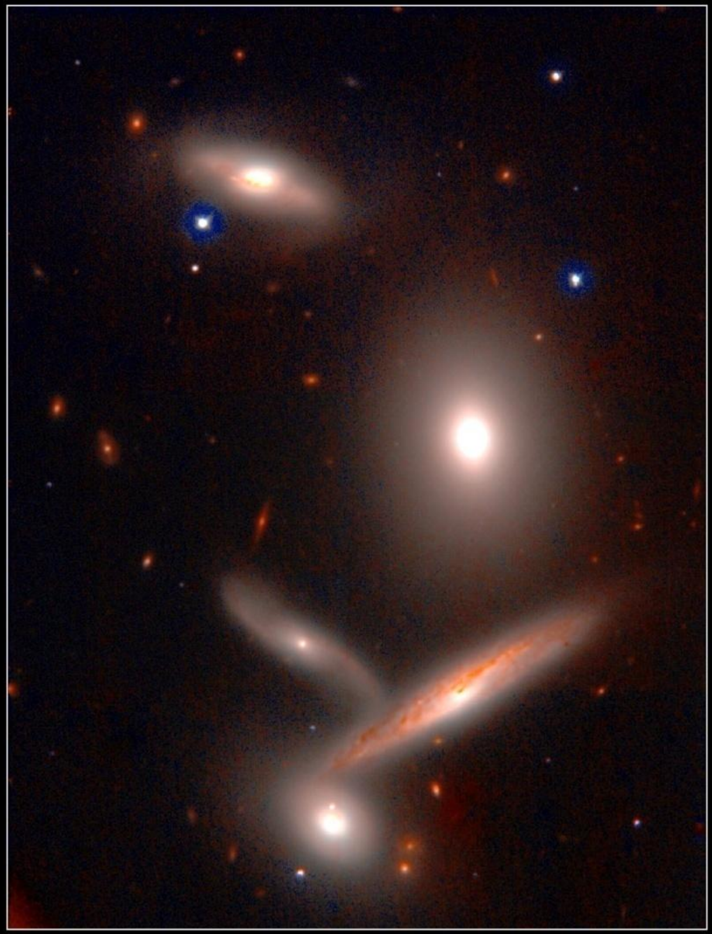
where f is gravity, hydro force etc…

- Two main problems
  - How to integrate the ODE?
  - How to compute RHS of ODE?
    - We will use accelerators for this part

Grand Challenge Problems



The APM Galaxy survey
Maddox Sutherland Efstathiou & Loveday



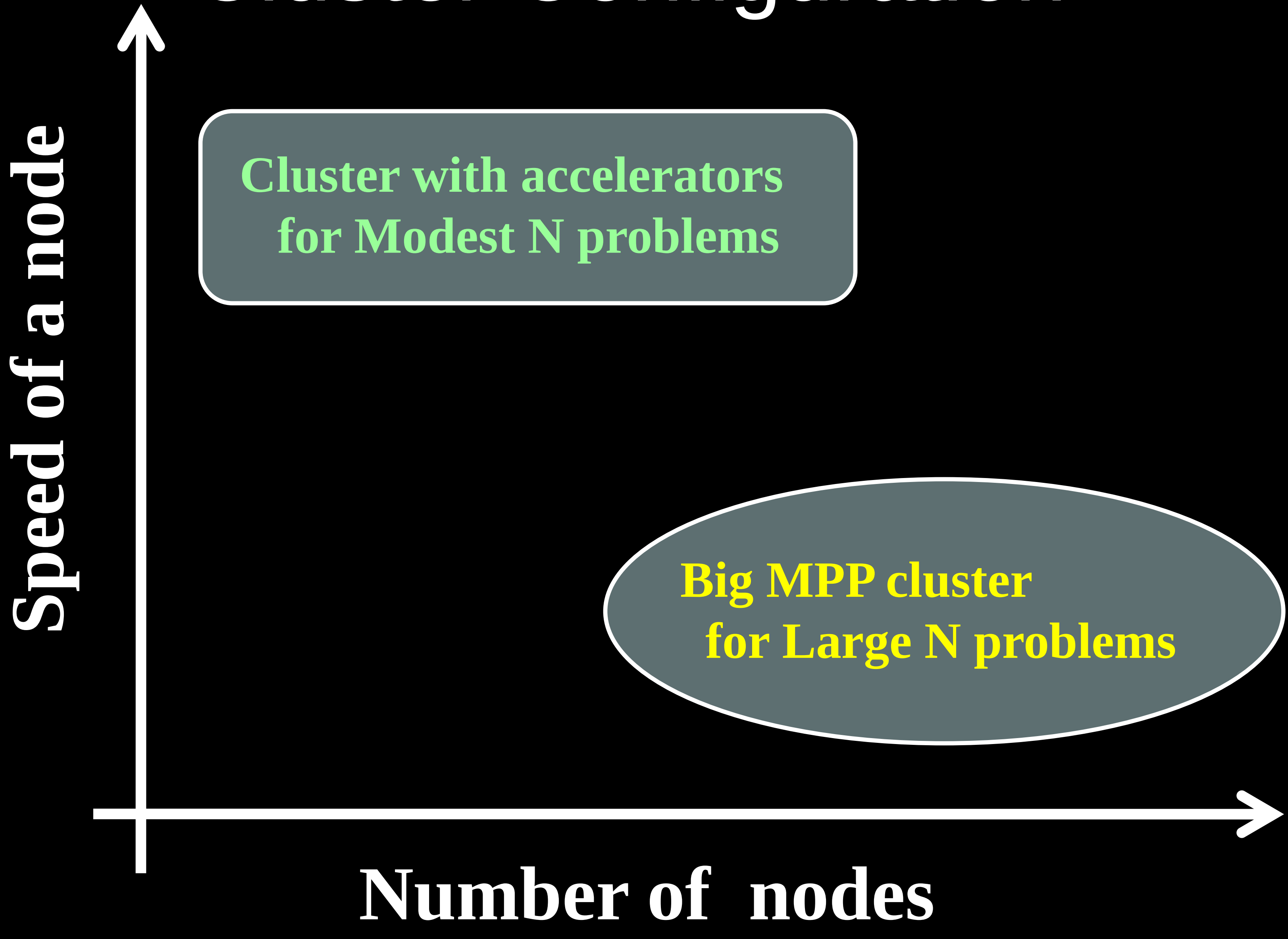
Hickson Compact Group 40
Subaru Telescope, National Astronomical Observatory of Japan
CISCO (J & K′)
January 28, 1999

Grand Challenge Problems

- Simulations with very huge N
  - How is mass distributed in the Universe?
    - One big run with N ~ 10^{9-12}
  - Scalable on a simple big MPP system
    - Limited by memory size

- Modest N but complex physics
  - Precise modeling of formation of astronomical objects like galaxy, star, solar system.
  - Need many runs with N ~ 10^{6-7}

Cluster Configuration



Accelerator?

- A device that assist a main computer
  - for speeding a specific calculation
    - Cell, ClearSpeed, GPU etc.

- Many-core accelerator is
  - Parallel computer on a chip
    - Difficulties raised in parallel computing applies
  - Very high performance on specific tasks
  - Developing so fast
    - changes in mice year?

Many-core Accelerators

- Cell, ClearSpeed, GPU etc.
  - have FP units as many as 32 – 1000 or more
  - Number of FP units is continuously rising…
    - Driven by demand for high performance gaming!
    - 2 x growth with every generation (~1.5 yr or so)

Latest Cypress GPU (ATi)
1600 FP units (single precision)
Running at 850 MHz
1 GB
16x PCI-E gen2
Consume ~ 200W

TOP500 List

Two systems use accelerators out of top 5 systems

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}
1	Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00
2	DOE/NNSA/LANL United States	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042.00	1375.78
3	National Institute for Computational Sciences/University of Tennessee United States	Kraken XT5 - Cray XT5-HE Opteron Six Core 2.6 GHz / 2009 Cray Inc.	98928	831.70	1028.85
4	Forschungszentrum Juelich (FZJ) Germany	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825.50	1002.70
5	National SuperComputer Center in Tianjin/NUDT China	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband / 2009 NUDT	71680	563.10	1206.19

PowerXCell 8i

Radeon HD4870

Green500 List

All top systems use accelerators

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)	TOP500 Rank*
1	722.98	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 Ghz, 3D-Torus	59.49	110
1	722.98	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 Ghz, 3D-Torus	59.49	111
1	722.98	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 Ghz, 3D-Torus	59.49	112
4	458.33	DOE/NNSA/LANL	Blade[illegible] Powe[illegible] G[illegible]z, Infiniband	276	29
4	458.33	IBM Poughkeepsie Benchmarking Center	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 Ghz, Infiniband	138	78
6	444.25	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 Ghz, Voltaire Infiniband	2345.5	2
7	428.91	National Astronomical Observatory of Japan	[illegible]	51.2	445
8	379.24	National SuperComputer Center in Tianjin/NUDT	[illegible]	1484.8	5
9	378.77	King Abdullah University of Science and Technology	Blue Gene/P Solution	504	18

PowerXCell 8i

GRAPE-DR

Radeon HD4870

Using GPU is easy if…

- Use the existing library
  - LINPACK relies on DGEMM
    - DGEMM performance of GPU > 100 Gflops
  - FFT on GPU ~ 50 Gflops (SP)
  - N-body on GPU ~ 100 Gflops (DP)

- For more general problems
  - Rewriting the existing code base
    - Rewriting itself is not so difficult
    - Optimizing it is the problem depending on a given architecture

Architecture of Accelerators (1)

- CPU controls GPU
  - Application running on CPU
  - kernel running on GPU

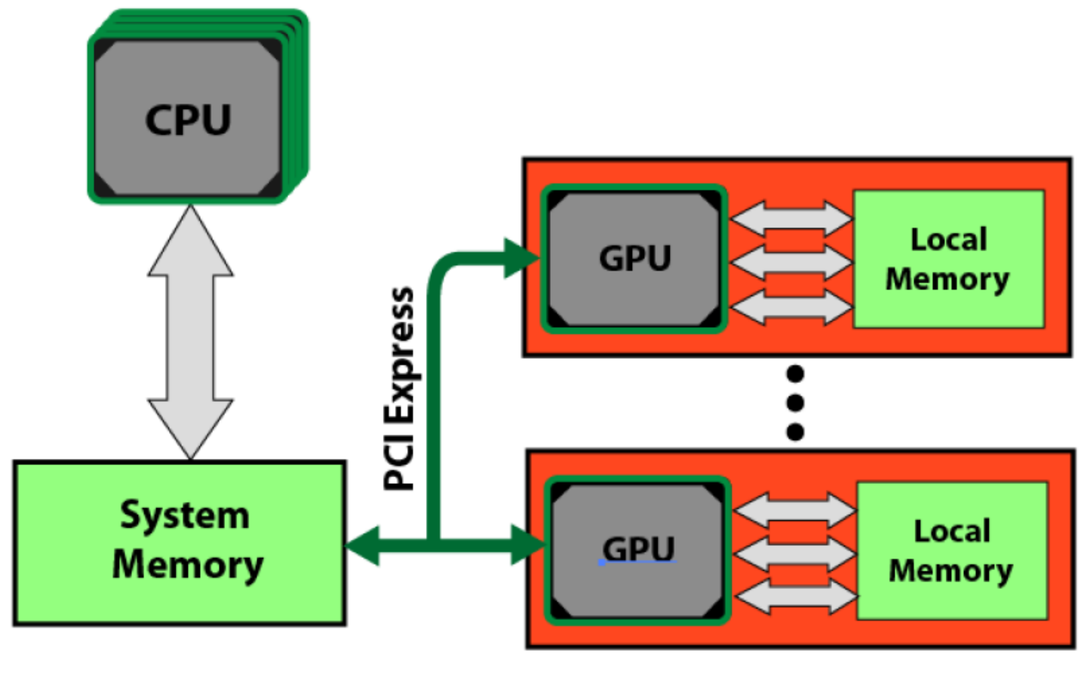
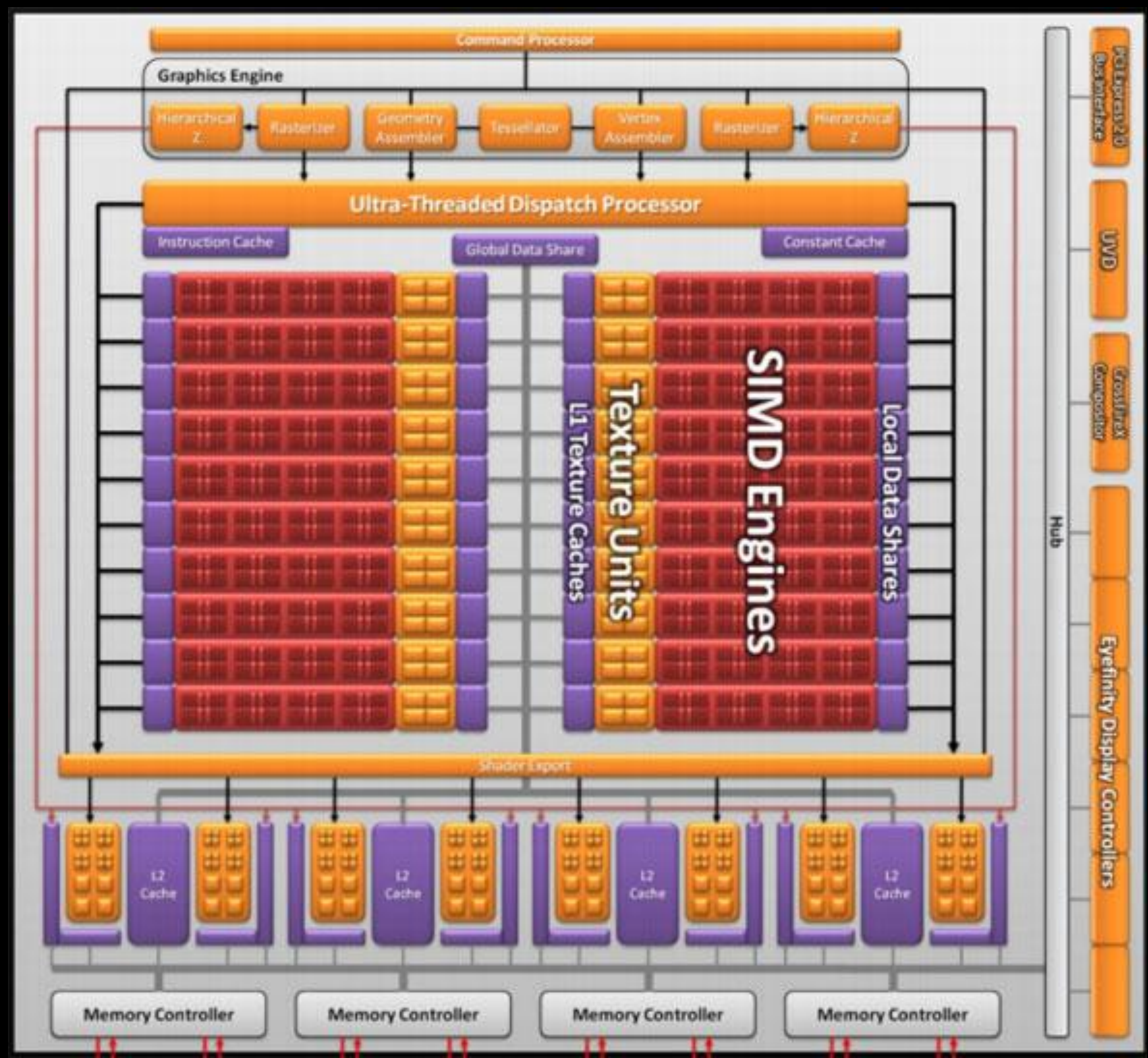


Figure 1.2 CAL System Architecture

Architecture of Accelerators (2)

GPU consists of many FP units



Challenges

- How to program many-core systems?
  - Like a vector-processor but not exactly same
  - Many programming models/APIs for rapidly changing architectures

- Memory wall
  - at the local memory
    - 2.7 Tflops vs. 153 GB s^{-1}
  - at I/O the accelerators
    - Only 16 GB s^{-1}
    - External I/O in cluster configuration is more severe

Programming Many-core Accelerators

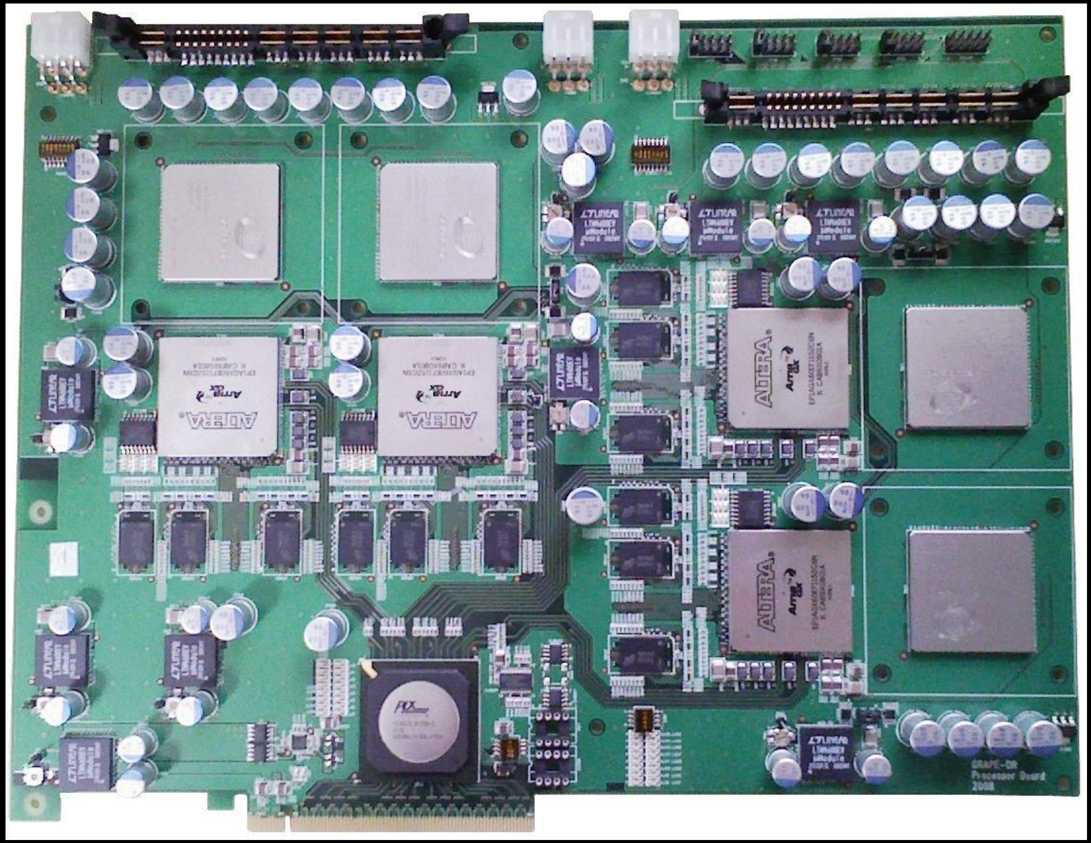
- To use accelerators, need two programs
  - A program running on host
  - A program running on accelerators
    - Compute kernel

- Example
  - C for CUDA / Brook+
    - Host program in C++
    - Compute kernel in extended C
      - Function with appropriate keyword
      - Separate source code

Programming efforts require

- on how we I/O to/from accelerators
  - Mainly programming for CPU
    - relatively easy
- on how we use FP units
- on how we use internal memories
  - Programming for GPU
    - strongly dependent on a given architecture
    - where we need to optimize
- on how we program a cluster of GPU
  - no definitive answer

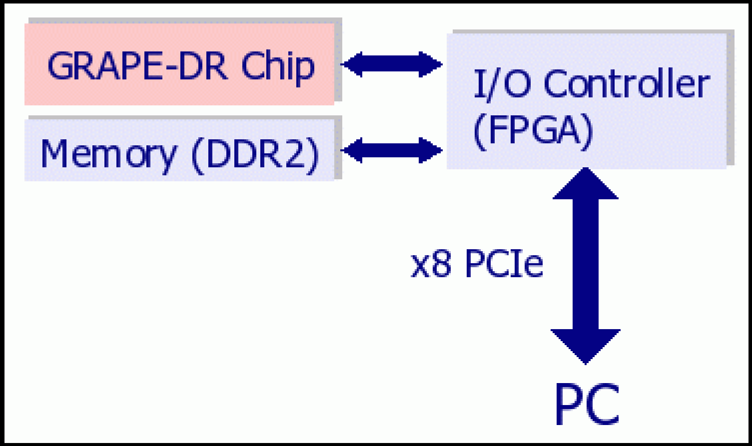
GRAPE-DR (1)

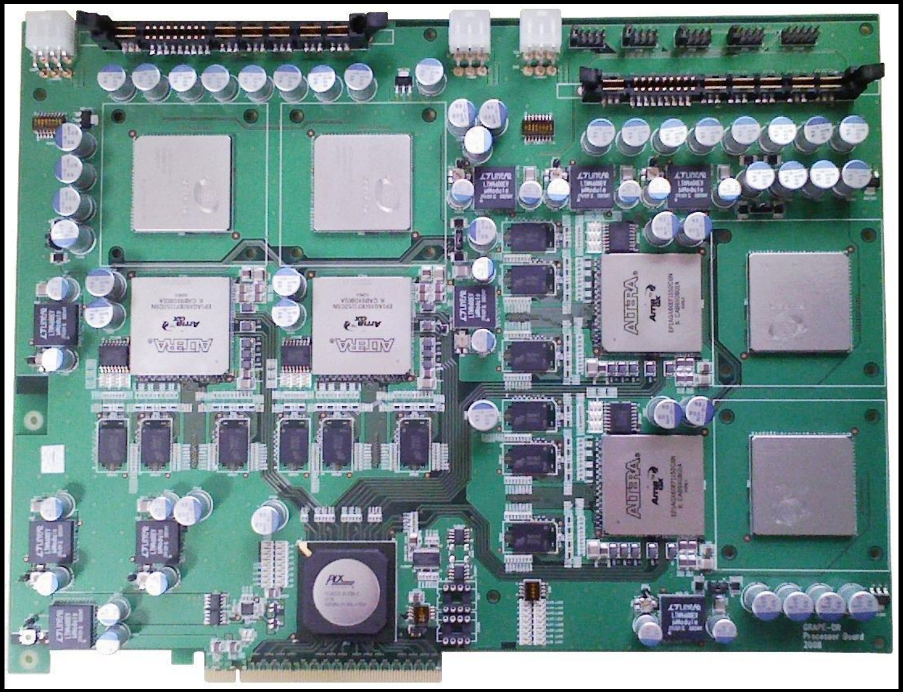


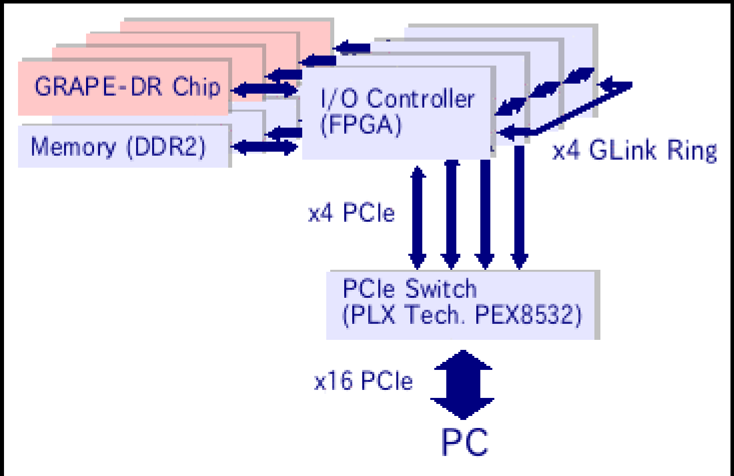
One Chip:
512 PEs
Running at 400 MHz
8x PCI-E gen1
288 MB
Consume ~ 50 W

Ranked at 445th on TOP500
Ranked at 7th on Green500

GRAPE-DR (2)



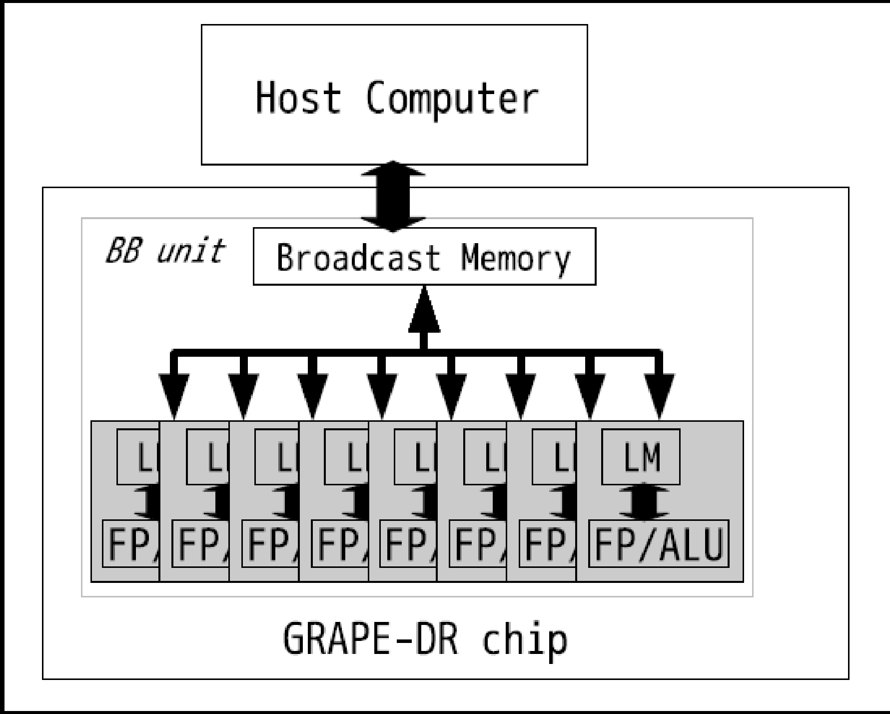




http://kfcr.jp/

Many-core Accelerators

- Both GRAPE-DR and R700 GPU
  - DP performance > 200 GFLOPS
  - Have many local registers : 72/256 words
  - Resource sharing in SP and DP units



But different in

- **R700 has more complex VLIW stream cores**
- **R700 has no BM**
- **R700 has faster memory I/O**
- **DR has reduction network for efficient summation**

Numerical Modeling

- Solve ODE for many particles

$$\frac{d\overrightarrow{v_i}}{dt} = \sum_{j=1}^{N} \vec{f}(\vec{r}_i - \vec{r}_j)$$

where f is gravity, hydro force etc…

- Two main problems
  - How to integrate the ODE?
  - How to compute RHS of ODE?
    - We will use accelerators for this part

A simple way to compute RHS

- Compute force summation as

```
for i = 0 to N-1
  s[i] = 0
  for j = 0 to N-1
    s[i] += f(x[i], x[j])
```

Fig. 1. A simple nested loop to computer a general force calculation.

  - Each s[i] can be computed independently
    - Massively parallel if N is large
    - Given i & j, each f(x[i],x[j]) can be computed independently if f() is complex

Unrolling (vectrization)

- Parallel nature enable us to unroll the outer-loop in n-ways

```
for i = 0 to N-1 each 4
  s[i] = s[i+1] = s[i+2] = s[i+3] = 0
  for j = 0 to N-1
    s[i]   += f(x[i],   x[j])
    s[i+1] += f(x[i+1], x[j])
    s[i+2] += f(x[i+2], x[j])
    s[i+3] += f(x[i+3], x[j])
```

  - Two types of variables
    - x[i] and s[i] are unchanged during j-loop
    - x[j] is shared at each iteration
  - Map computation for each x[i] to PE on accelerators

Optimization on GPU

```
for i = 0 to N-1
  acc[i] = 0
  for j = 0 to N-1
    acc[i] += f(x[i], x[j])
```

~ 300 Gflops

```
for i = 0 to N-1 each 4
  acc[i] = acc[i+1] = acc[i+2] = acc[i+3] = 0
  for j = 0 to N-1
    acc[i]   += f(x[i],   x[j])
    acc[i+1] += f(x[i+1], x[j])
    acc[i+2] += f(x[i+2], x[j])
    acc[i+3] += f(x[i+3], x[j])
```

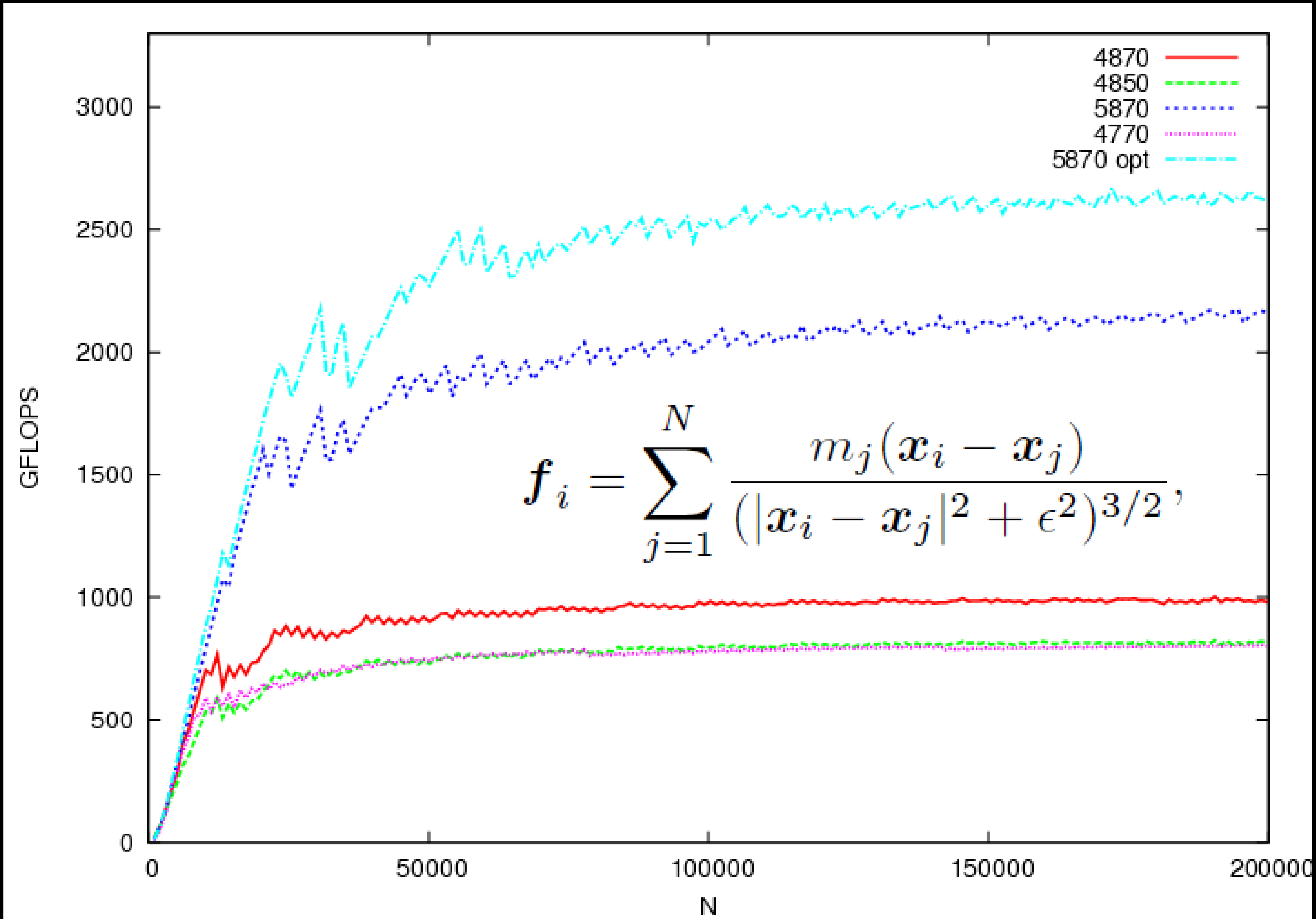
~ 500 Gflops

```
for i = 0 to N-1 each 4
  acc[i] = acc[i+1] = acc[i+2] = acc[i+3] = 0
  for j = 0 to N-1 each 4
    for k = 0 to 3
      acc[i  ] += f(x[i  ], x[j+k])
      acc[i+1] += f(x[i+1], x[j+k])
      acc[i+2] += f(x[i+2], x[j+k])
      acc[i+3] += f(x[i+3], x[j+k])
```

~ 700 Gflops

Performance of O(N^2) algorithm

On a recent GPU ~ 1.3 Tflops



Our Compiler

- Accelerates force summation loop
- Support two accelerators
  - R700/R800 architecture GPU
  - GRAPE-DR
    - Developed by J.Makino etal.
- Precision controllable
  - Single, Double, & Quadruple precision
    - QP through DD emulation techniques
  - Partially support mixed precision

Our programming model

- User write a source in DSL such as

```
LMEM xi, yi, zi, e2;
BMEM xj, yj, zj, mj;
RMEM ax, ay, az;

dx = xj - xi;
dy = yj - yi;
dz = zj - zi;

r1i = rsqrt(dx**2 + dy**2 + dz**2 + e2);
af = mj*r1i**3;

ax += af*dx;
ay += af*dy;
az += af*dz;
```

  – Our compiler generates optimized machine code for GPU / GRAPE-DR

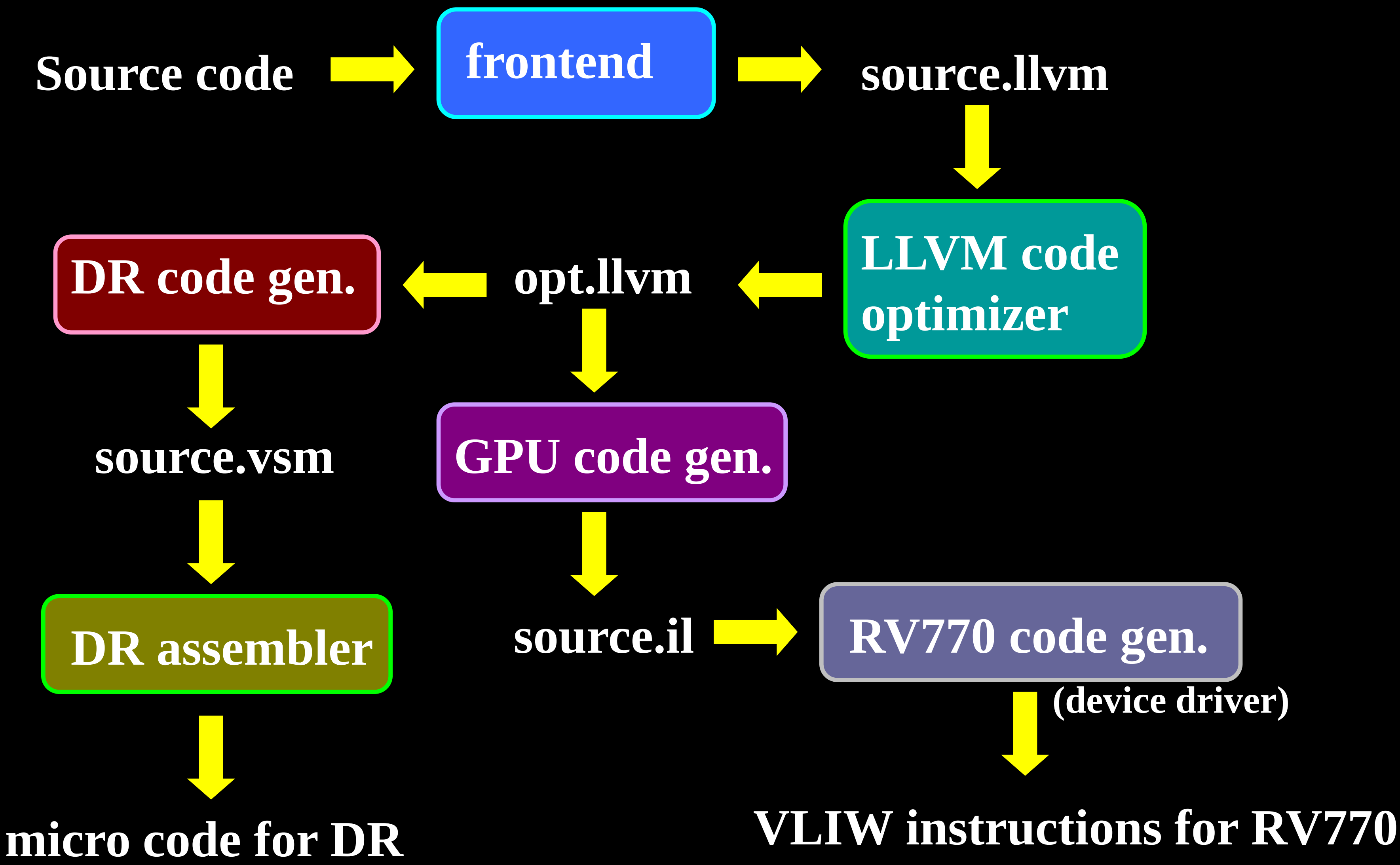
Comparison

- Our approach is in between two conventional approaches
  - Automatic parallel compiler
    - A user just feed an existing source code
    - But not effective in general
  - Let-users-do-everything-type compiler
    - C for CUDA, OpenCL, Brook+ etc.
    - A user have to specify every details of
      - Memory layout and its movement
      - SIMD operations
      - Threads management on GPU

Details of our compiler

- Written in C++
  - Prototype was developed in Ruby
- We use following software/library
  - Boost sprit for the parser
  - Low Level Virtual Machine for the optimizer
  - Google template library for the code generators

Compiler work flow



http://galaxy.u-aizu.ac.jp/trac/note/

Example 1 : N-body

- Simple softened gravity

$$\boldsymbol{f}_i = \sum_{j=1}^{N} \frac{m_j(\boldsymbol{x}_i - \boldsymbol{x}_j)}{(|\boldsymbol{x}_i - \boldsymbol{x}_j|^2 + \epsilon^2)^{3/2}},$$

↓

```
LMEM xi, yi, zi, e2;
BMEM xj, yj, zj, mj;
RMEM ax, ay, az;

dx = xj - xi;
dy = yj - yi;
dz = zj - zi;

r1i = rsqrt(dx**2 + dy**2 + dz**2 + e2);
af = mj*r1i**3;

ax += af*dx;
ay += af*dy;
az += af*dz;
```

Example 2: Feynman-loop integral

$$
\begin{aligned}
I &= \int_0^1 dx \int_0^{1-x} dy \int_0^{1-x-y} dz F(x,y,z), \\
F(x,y,z) &= D(x,y,z)^{-2} \\
D &= -xys - tz(1-x-y-z) + (x+y)\lambda^2 \\
&\quad +(1-x-y-z)(1-x-y)m_e^2 \\
&\quad +z(1-x-y)m_f^2. \qquad (2)
\end{aligned}
$$

```
LMEM  xx, yy, cnt4;
BMEM  x30_1, gw30;
RMEM  res;
CONST tt, ramda, fme, fmf, s, one;

zz = x30_1*cnt4;
d = -xx*yy*s-tt*zz*(one-xx-yy-zz)+(xx+yy)*ramda**2 +
      (one-xx-yy-zz)*(one-xx-yy)*fme**2+zz*(one-xx-yy)*fmf**2;
res += gw30/d**2;
```

QD operations on GPU

- We have implemented so-called DD emulation scheme on GPU&GRAPE-DR
  - QD variable is expressed as summation of two double precision variables
  - QD operations are emulated with DP operations
    - At least 20 times slower performance
    - Practical performance is more than 30 times slower on Core i7 CPU

Performance of QP operations

- Computation of Feynman-loop integral
  - elapsed time in QP operations

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	clock
GRAPE-DR	0.21	1.21	7.83	55.1	380
RV770	0.09	0.66	5.03	39.7	750
Core i7	7.39	59.0	472		2670

  - CPU ~ 80 Mflops
  - R700 GPU ~ 6.43 – 7.57 Gflops
  - GRAPE-DR ~ 2.67 – 5.46 Gflops
- Tow reasons why QP is so fast
  - High compute density
  - DR & R700 are register rich

Development of QP arithmetic units

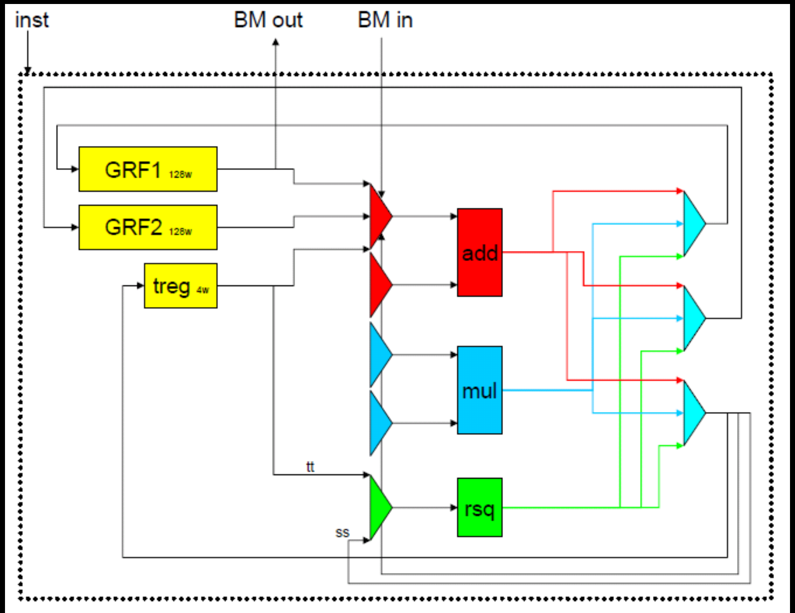
- QP emulation is not efficient
  - A factor of 20 performance penalty
  - Power consumption

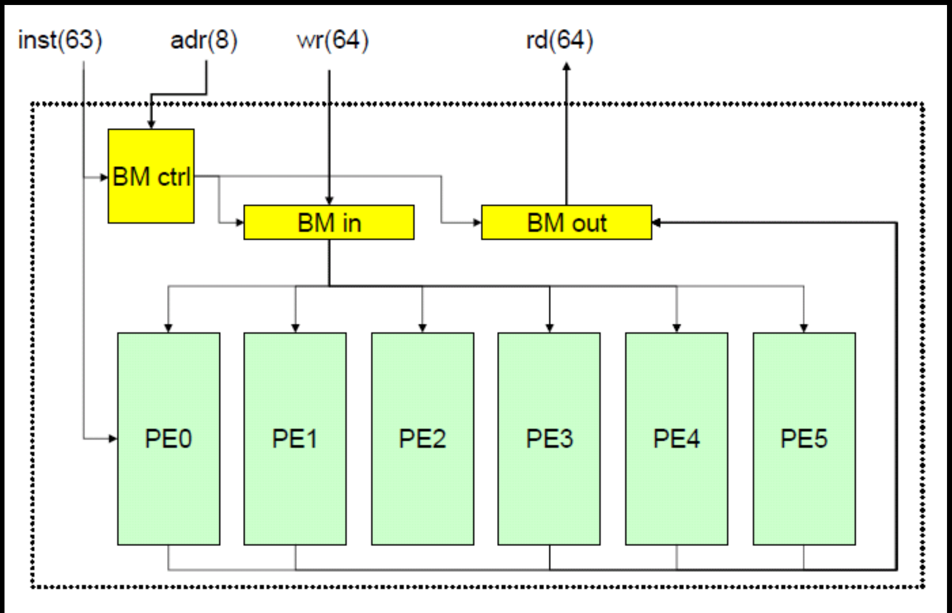
- If we have a dedicated QP unit
  - should be faster and energy efficient
  - but no commercial demand (yet)

We investigated a prototype of accelerator with QP arithmetic units

Status of Project

- We have implemented QP arithmetic units
  - Designed for Feynman integrals
  - 116 bit for mantissa, 11 bit for exponent
  - Add & Mul & inverse sqrt units
  - Implemented by VHDL





Summary

- Is a many-core accelerator is effective for
  - Massively parallel problems : YES
    - Monte-calro on million phase space points
  - $O(N^2)$ problems : YES
    - Gravity, Feynman integrals
  - $O(N^{1.5})$ problems : Yes
    - Matrix multiply (DGEMM)
  - O(N log N) & O(N) problems
    - Generally it is not easy to optimize…
  - High precision operations : Yes

- Key is data reuse = high compute density

Conclusion

- Many-core accelerators are effective in problems in astronomy and physics
  - But how to program it effectively?
- We have constructed a compiler for many-core accelerators
  - That accelerate force-calculation-loop
  - Features simplicity and controllable precision
- Planed Extension
  - Support O(N log N) method on GPU