apeNEXT: a Multi-TFlops Computer for Lattice QCD

INFN Ferrara, Rome DESY Zeuthen Université de Paris-Sud, Orsay

Outline:

- ❑ Application signature
- ❑ Hardware architecture characteristics
- ❑ Software
- ❑ Benchmarks
- ❑ Status

Application Signature: Linear Algebra

- BLAS1: `vnorm`, `zdotc`, `zaxpy`
- Products of arrays of matrices, e.g. SU(3)

$$W_{x,a,b} = \sum_c U^\dagger_{x,a,c} V_{x,c,b}$$

- Global reductions operations: global sum (may include broadcast)

I/O operations (complex words) vs. floating point operations:

operation	#read	#write	Flop	Flop/(#read+#write)
`vnorm`	1	-	4	4
`zdotc`	2	-	8	4
`zaxpy`	2	1	8	3.3
$U\ V$	18	9	202	7.5

Application Signature: Wilson-Dirac Operator

$$M_{xy}[U]\ \Phi_y =$$

$$\left\{\delta_{xy} - \kappa \sum_{\mu} \left[(1-\gamma^{\mu})\ U_{\mu,x}\ \delta_{x+\hat{\mu},y} + (1+\gamma^{\mu})\ U^{\dagger}_{\mu,x-\hat{\mu}}\ \delta_{x-\hat{\mu},y}\right]\right\} \Phi_y =$$

$$\left\{\delta_{xy} - \kappa D_{xy}[U]\right\}\ \Phi_y$$

Objects:	Φ_x	quark field	3×4 complex
	$U_{x,\mu}$	gluon field	3×3 complex
	γ_μ	γ matrix	4×4 complex

Parallelisation using space domain decomposition

➔ nearest neighbour communication

➔ homogeneous distribution

Application Signature: Wilson-Dirac Operator (cont.)

Consider following (worst) case:

- parallelization in 3 dimensions
- local lattice $V_{\mathrm{local}} = 2 \times 2 \times 2 \times L$
- local gauge fields

I/O operations vs. floating point operations:

operation	#read	#write	Flop	Flop/(#read+#write)
$D[U]\,\Phi$	168	12	1320	7

Local I/O operations vs. remote communication:

operation	#remote	(#read+#write)/#remote
$D[U]\,\Phi$	36	5

Hardware Characteristics: Arithmetic Unit

- floating point unit (FPU) performs one operation $a \times b + c$ per clock cycle, where a, b, c complex numbers

➔ 8 Flops / cycle = 1.6 GFlops/sec

- IEEE, 64-bit floating point numbers

- arithmetic unit also executes integer, logical and LUT operations on pairs of 64 bit operands

Hardware Characteristics: Memory Hierarchy

memory controller

- supports 256 MBytes upto 1 GBytes DDR-SDRAM (with ECC)
- maximum bandwidth is one word per clock cycle

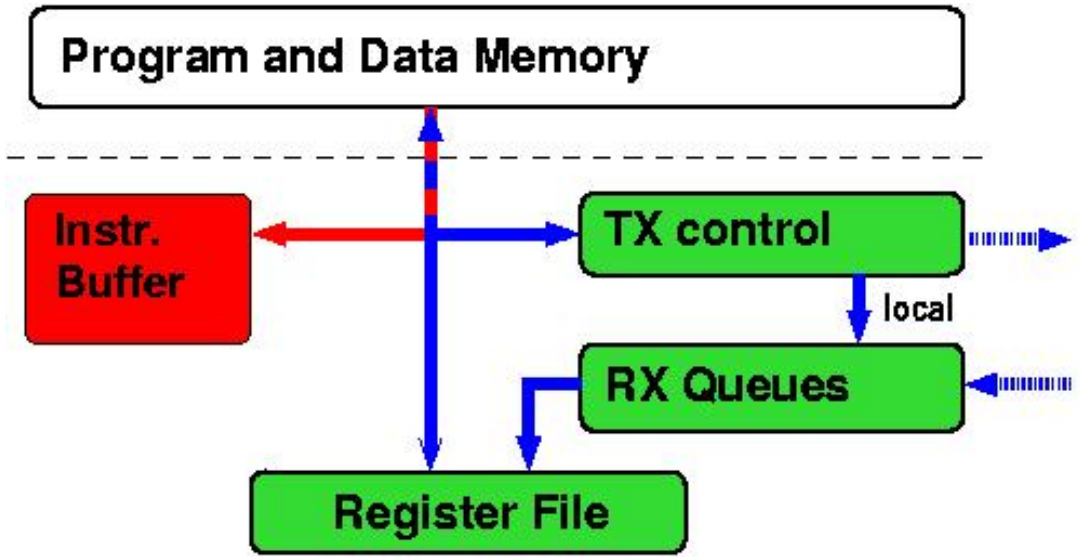
➔ 2×64 **bit word/cycle =** 3.2 **GBytes/sec**

- latency ≥ 16 cycles
- used for loading data and program instructions

instruction buffer

- allows storing 4k compressed, very long instructions words (VLIW)
- can be used as FIFO or dynamic/static cache

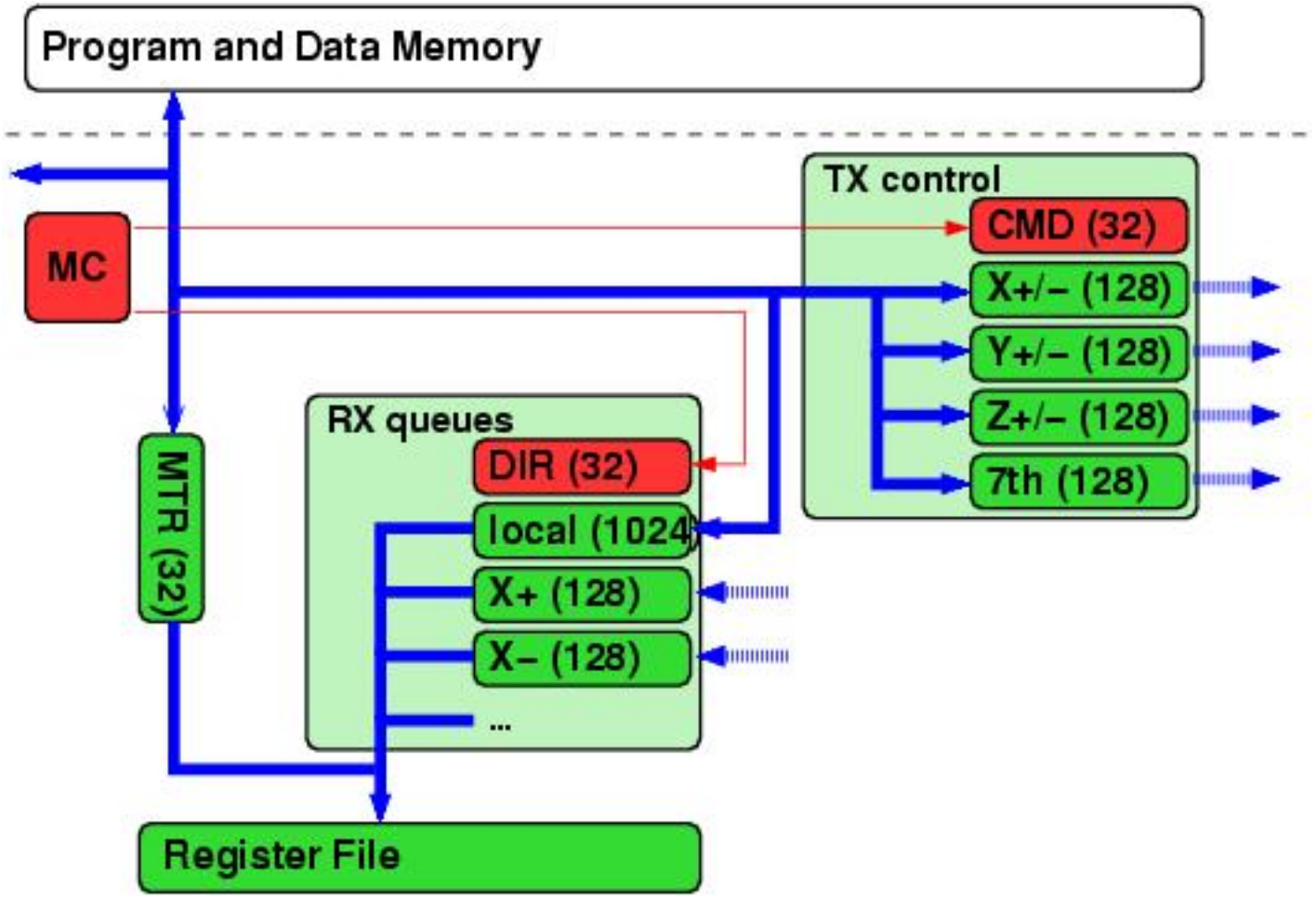
Hardware Characteristics: Memory Hierarchy (cont.)



register file

- 2×256 64-bit registers

Hardware Characteristics: Prefetch Queues



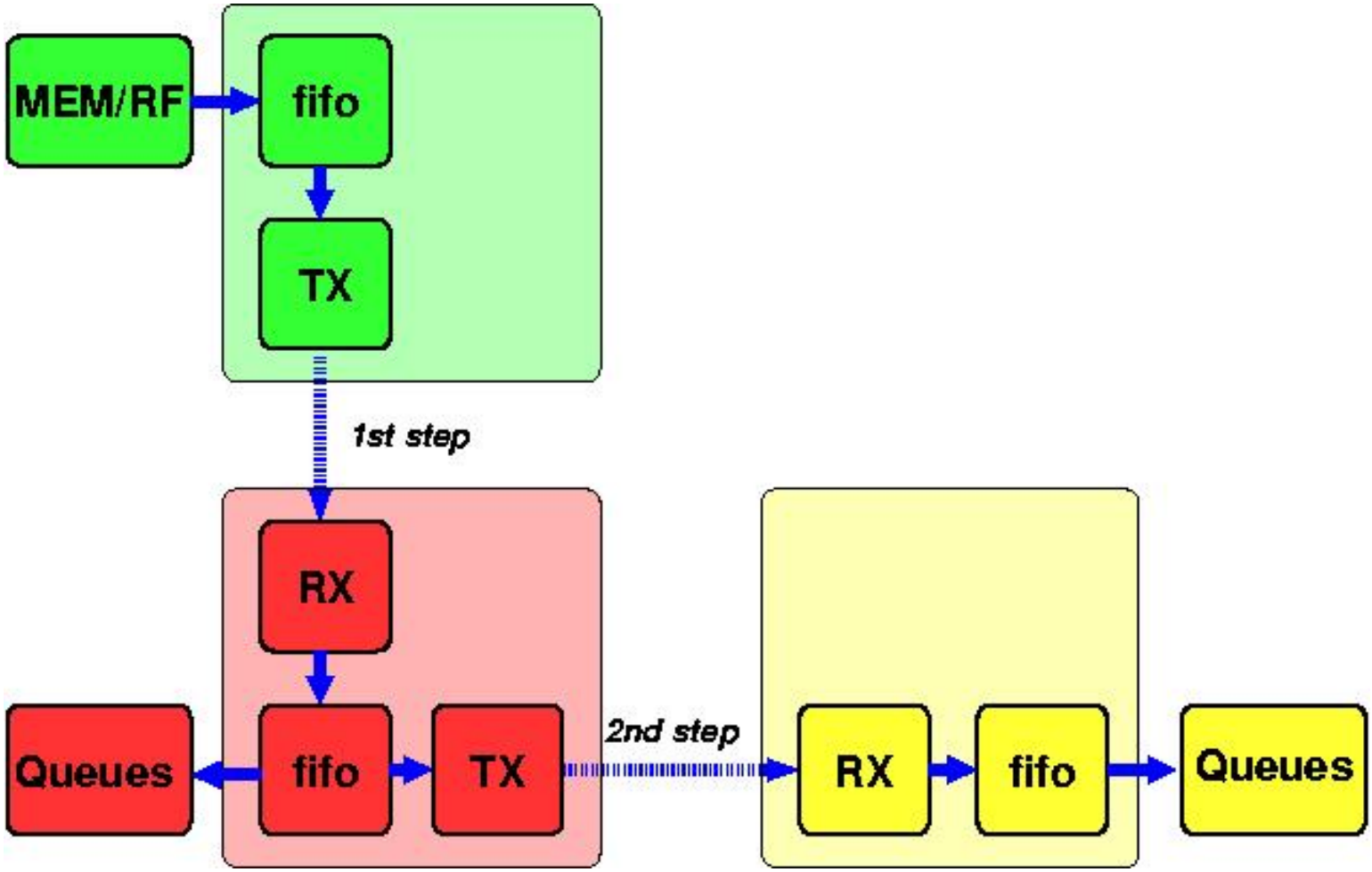
Hardware Characteristics: Network

- 7 bi-directional LVDS links: $\pm x$, $\pm y$, $\pm z$, 7th
- gross bandwidth per link is one byte per clock cycle

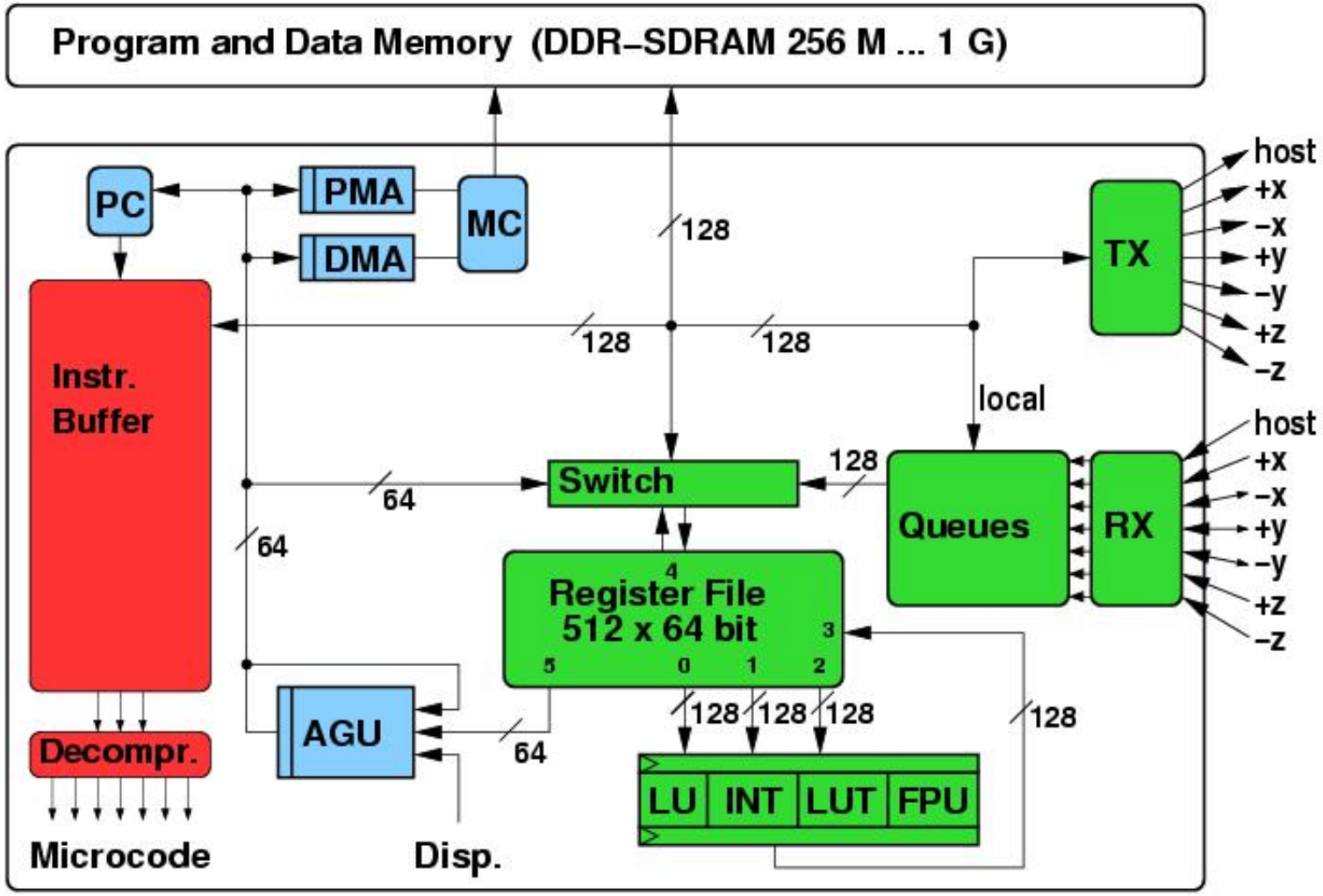
➔ 8 **bit/cycle =** 200 **MBytes/sec**

- transmission by frames of 128 bit data + 16 bit CRC
  ➔ effective bandwidth $\leq$ 180 MBytes/sec
- concurrent send and receive and concurrent transfer along orthogonal directions
- low latency: $\approx$ 25 cycles (125 ns)
- support for non-homogeneous communications
- configurable direction mapping

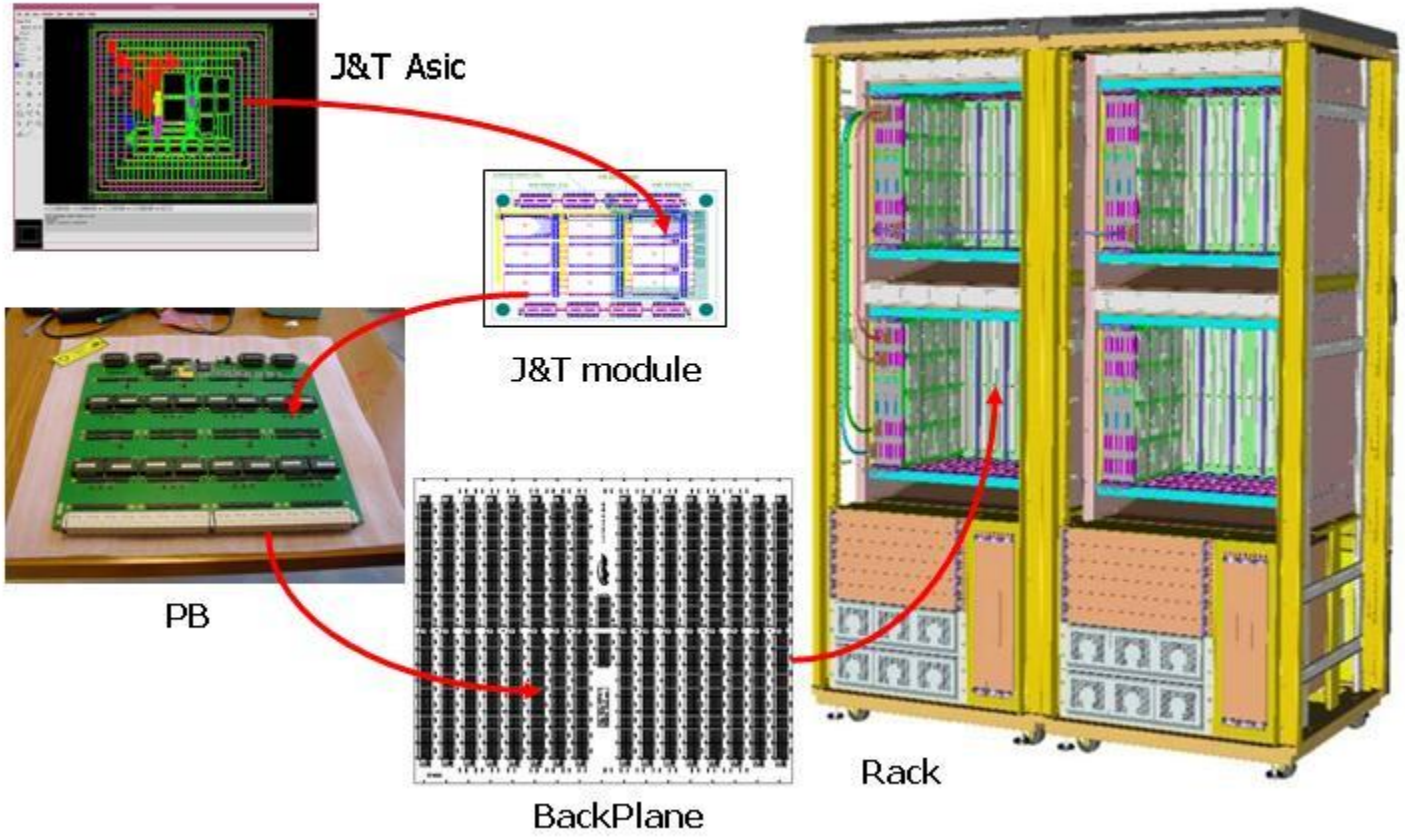
Hardware Characteristics: Network (cont.)



Processor Overview



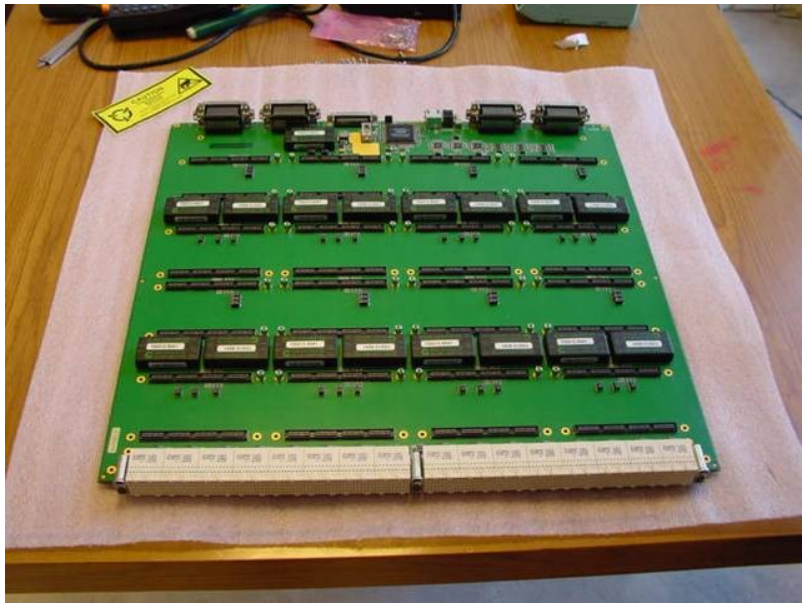
System Overview



Technical Details

Processor

- 0.18μ CMOS
- 600 pins



Processing Board

- slots for 16 daugther boards (1 processor per daugther board)
- FPGA for global signal and I2C handling
- 1728 differential signals for LVDS
- mechanical design relevant (BP-PB insertion force: 80-150 kg)

Technical Details (cont.)

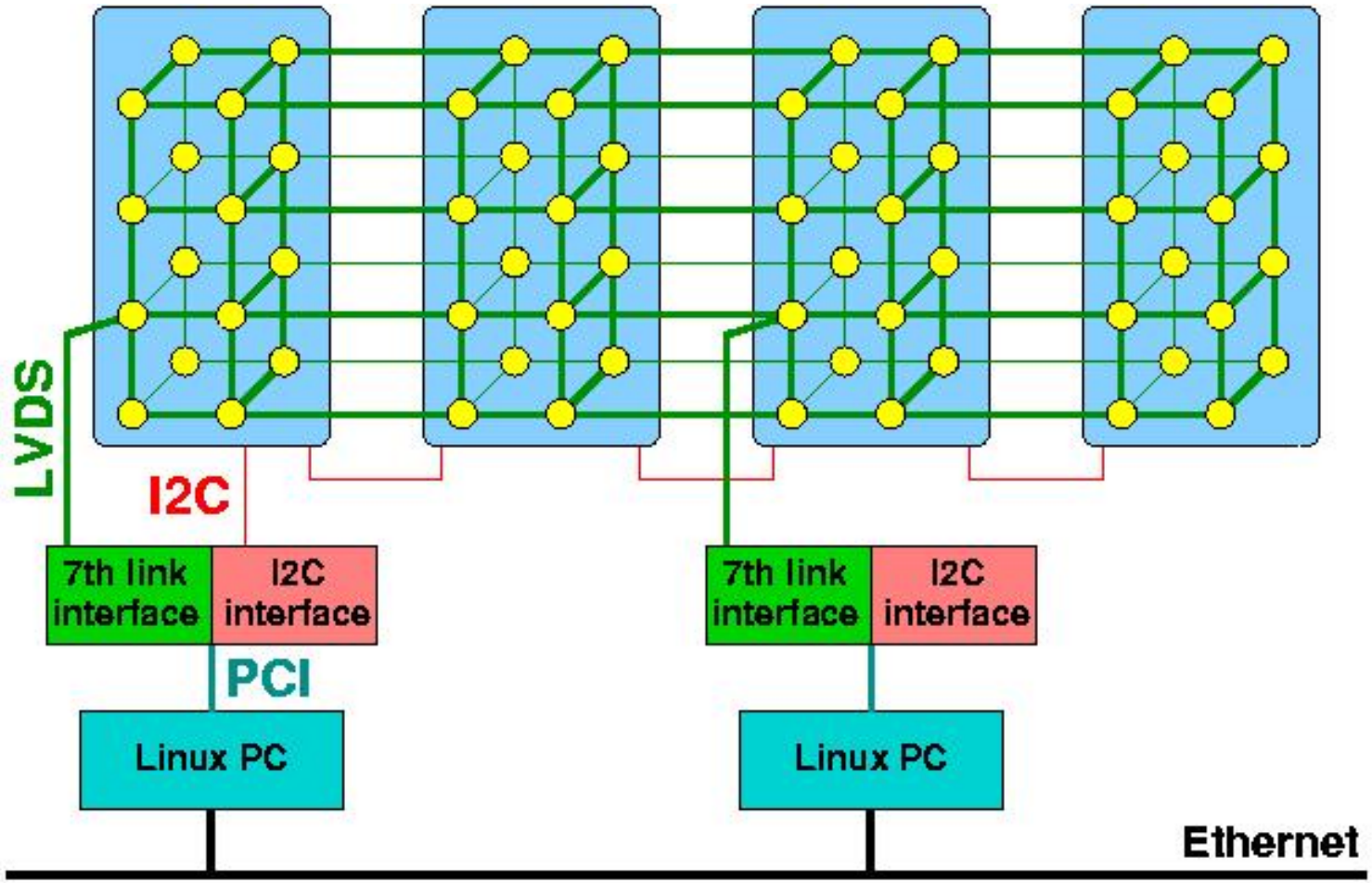
Backplane

- slots for 16 processing boards
- 4600 differential signals for LVDS, 16 layers

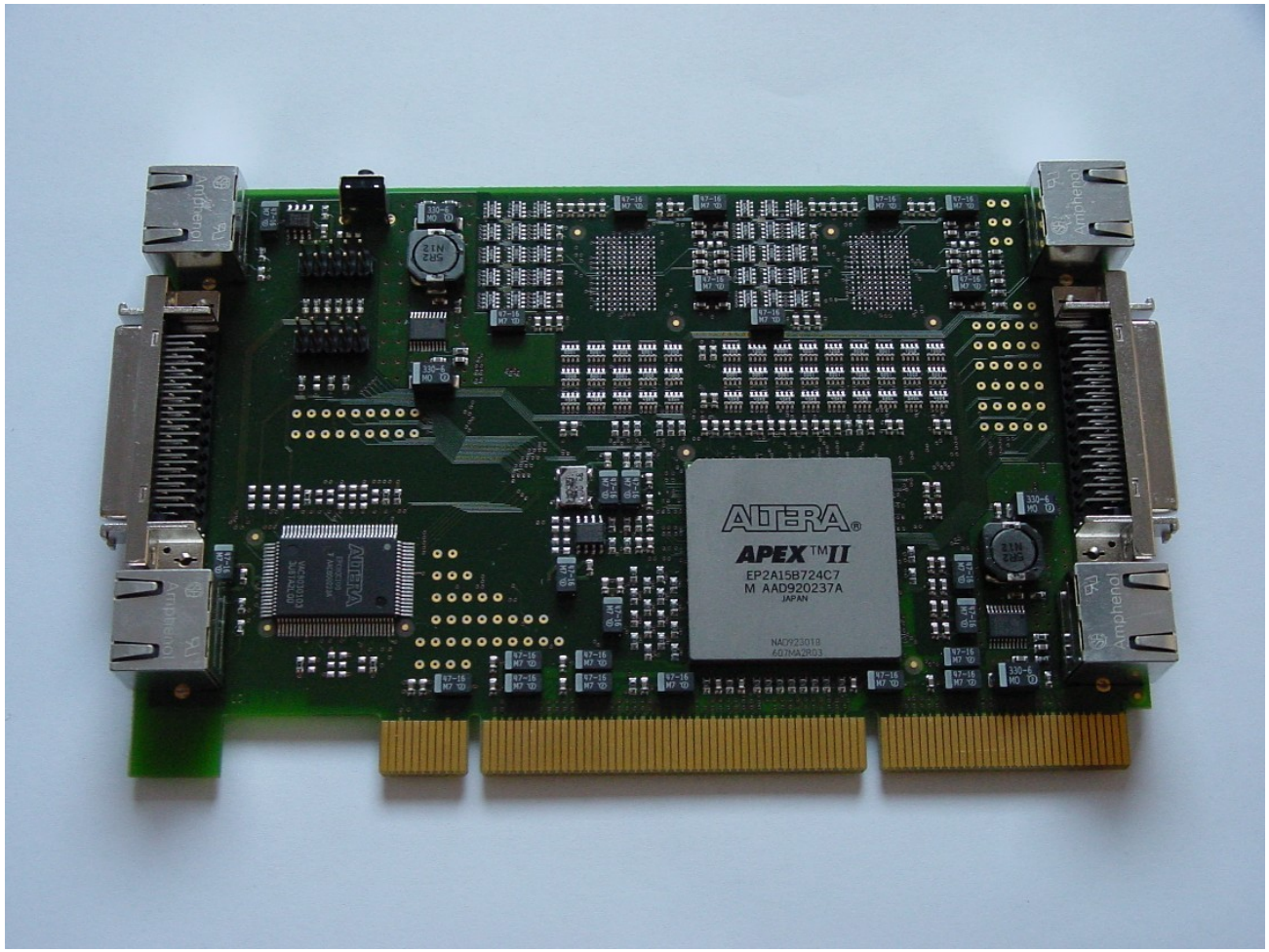
Rack

- slots for 2 backplanes
- footprint O(1 m^2)
- power consumption: 9 kW (estimated)
- air cooled
- hot-swap power supply

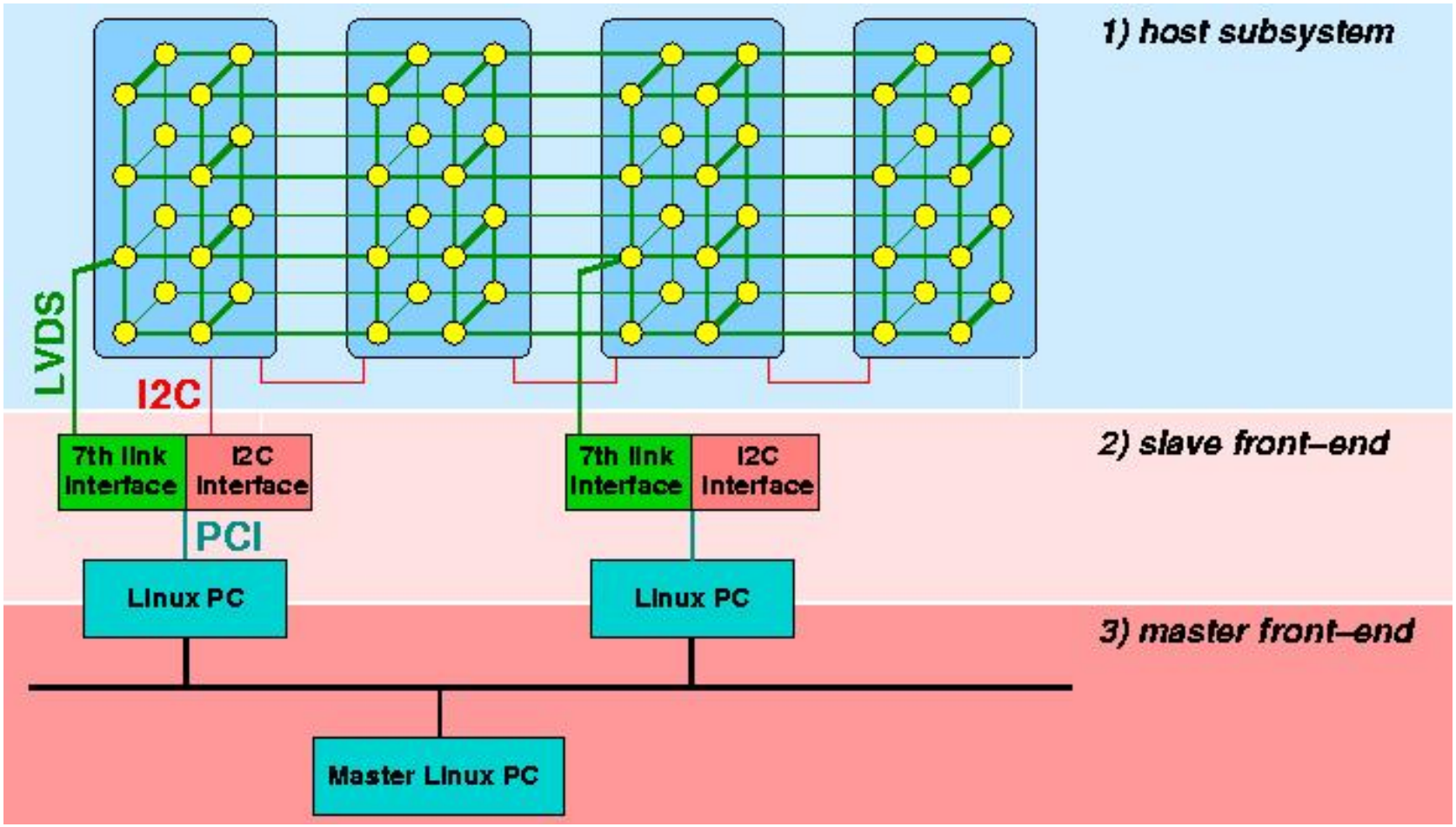
Global Architecture



Host Interface Board



Operating System



Programming Languages

TAO

- FORTRAN-like programming language
- Dynamical grammar allowed OO-style programming
- Needed for smooth transition from APEmille to apeNEXT

C

- Based on freely availabel lcc
- Most of ISO C99 standard supported
- Few language extensions

SASM

- High level assembly
- Aim: assembler programming not required

C-Compiler: Syntax Extensions

- New data types: `complex`, `vector`
- New operators: `~` (complex conjugation)
- New condition types: `where()`, `any()`, `all()`, `none()`

C-Compiler: Architecture Support

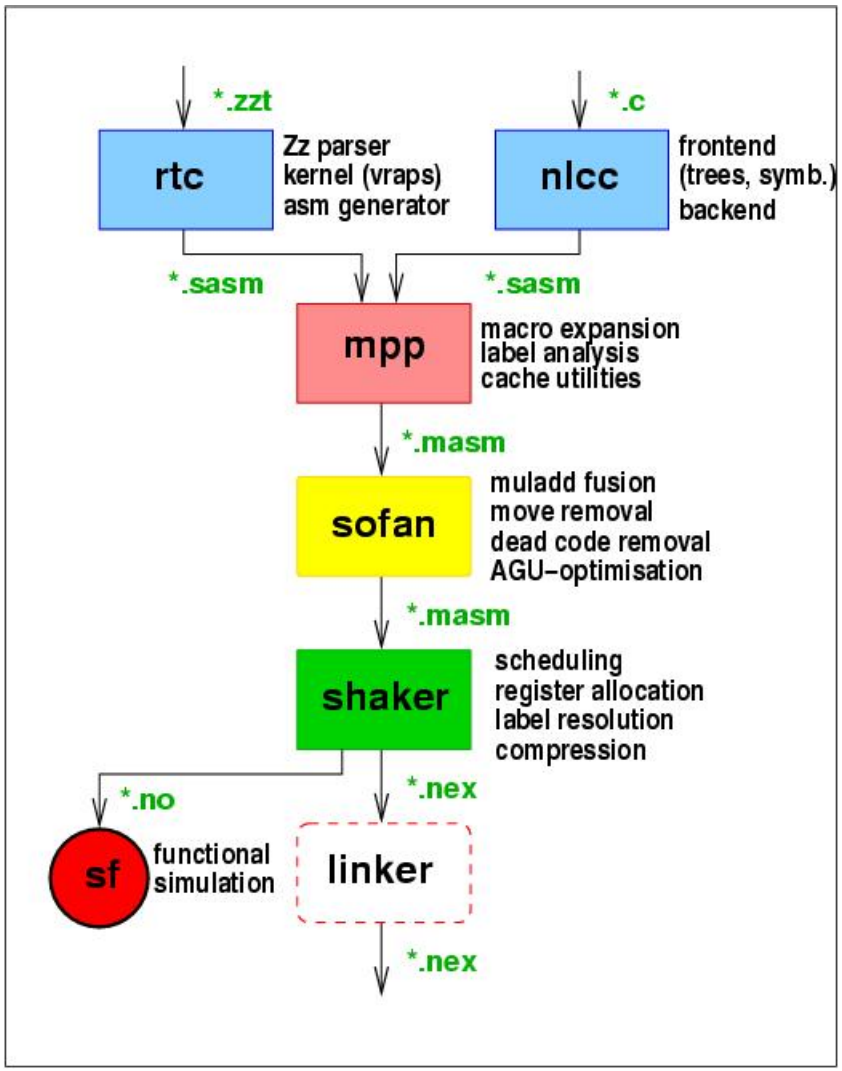
- `register struct` ➔ burst memory access
- Directives for controlling instruction buffer: `#pragma cache`
- Magic offsets for remote communication:

```
complex  a[1],  b;

b = a[0+X_PLUS];          // read data from node in X+ direction
```

- Macros for data prefetching:

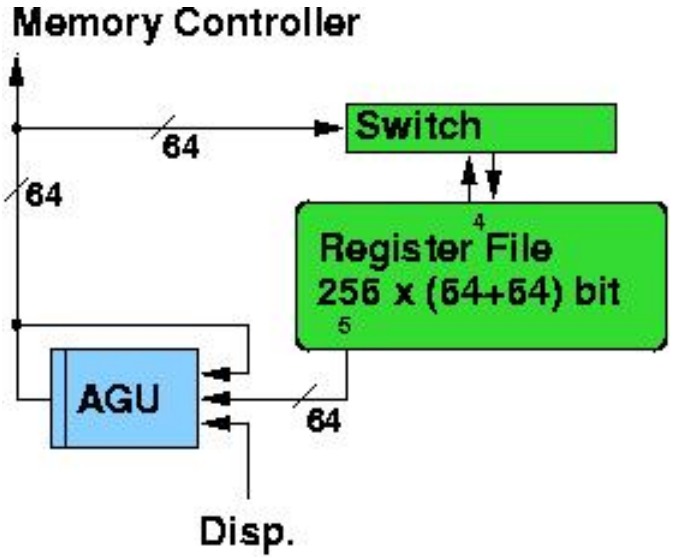
```
complex           a;
register complex  ra;

prefetch(a);              // memory → queue
fetch(ra);                // queue → register file
```

Software Overview



Assembler Optimizer: Sofan

- Optimization operating on low-level assembly
- Based on optimization toolkit SALTO (IRISA, Rennes)
- Optimization steps:
  - merging APE-normal operations
  - removing dead code
  - eliminating register moves
  - optimizing address generation:
  - instruction pre-scheduling
  - ...



Benchmarks: Linear Algebra

operation	IO-Op	Flop	sustained performance	
			"maximum"	measured
`vnorm`	1	4	50%	37%
`zdotc`	2	8	50%	41%
`zaxpy`	3	8	33%	29%
$U\ V$	27	202	92%	65%

"maximum" sustained performance ← ignoring latency of floating point pipeline and loop overhead

Optimization "tricks":

➔ loop unrolling

➔ burst memory access

➔ instructions kept in buffer

Performance limitations:

➔ start-up latency

➔ loop overhead

Example: Optimized `vnorm` in C

```
typedef struct {                                  // definition of structure
  complex c[16];
} bcomplex;

#pragma cache                                      // keep function in I-buffer
complex vnorm(bcomplex x[256]) {                   // input vector
  int                 i;
  complex             z;                           // result variable
  register bcomplex   rx, rz;                      // register structures

  /* initialization */
  ...
```

```
/* calculate product */
ix = 0;
while (i < 256) {
    #pragma localmem
    rx = x[i];                                  // use burst memory access

    rz.c[0]  += (~rx.c[0]) * rx.c[0];           // calculate partial sums
    rz.c[1]  += (~rx.c[1]) * rx.c[1];
    ...
    rz.c[15] += (~rx.c[15]) * rx.c[15];

    i++;

    ...                                         // possibly do loop unrolling
}

/* final operations */
...                                             // sum partial sums
z = rz.c[0];                                    // save final results
```

Benchmarks: Results from C

operation	assembler	C	C + Sofan
`vnorm`	37%	31%	34%
`zdotc`	41%	28%	40%

➔ Assembler programming not required

Benchmarks: Global Sum

Results form local sums need to be shifted (twice) along all nodes
➔ $2 \times (P_x + P_y + P_z - 3)$ communications

Measured time: ~ 0.2 μs / communication (neglecting electrical delays)

➔ ~ 7.5 μs on an $8 \times 8 \times 8$-system

For comparison: scalar product for $N = 32^3 \times 64 \times 12$ requires ~ 700 μs.

Benchmarks: Wilson-Dirac Operator

$$\Psi_x = D_{xy}[U]\,\Phi_y$$

Consider worst case: local lattice size 16×2^3

Measured sustained performance: 55%
Measured number of stretch cycles: 4%

Optimization "tricks":

➔ keep gluon fields local

➔ pre-fetching 2 sites ahead

➔ orthogonal communication directions

➔ some unrolling

Status

component	status
processor	prototype wafers ready
processing board	prototypes tested
backplane	prototype tested
rack	prototype available
host interface board	developing
TAO compiler	stable prototype
C compiler	prototype
assembler generator	release
microcode generator	release
assembler optimizer	developing
linker	planned
operating system	developing

Summary

- ✔ Few signatures of QCD applications performance relevant
- ✔ Good basis for architectural optimization
- ✔ apeNEXT hardware charateristics suitable for these applications
- ✘ Other applications not yet explored
- ✔ apeNEXT prototype hardware available or available soon
- ✔ Most apeNEXT software reached prototype level