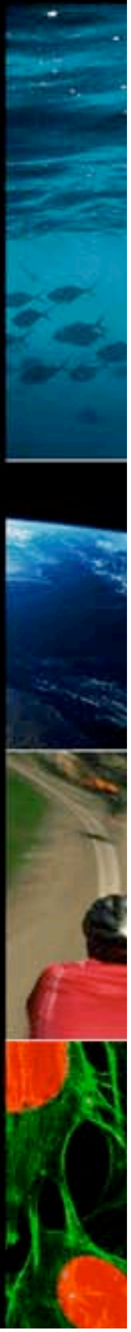
CRAY The Supercomputer Company

Applications Development with FPGAs
Simulate but Verify



G. Wenes, J. Maltby, D. Strenski

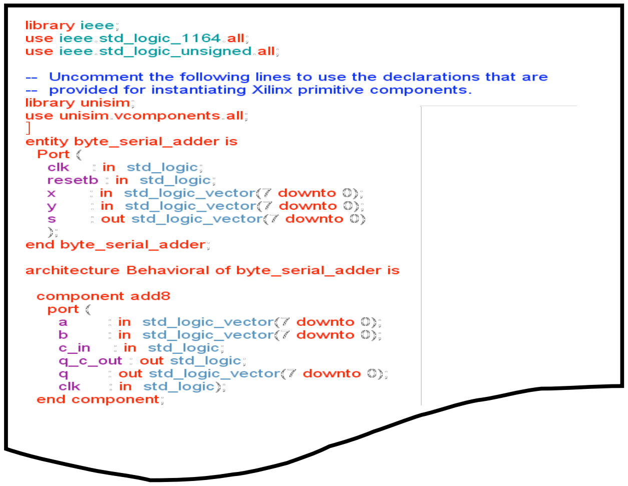
CUG 2005

Albuquerque, NM

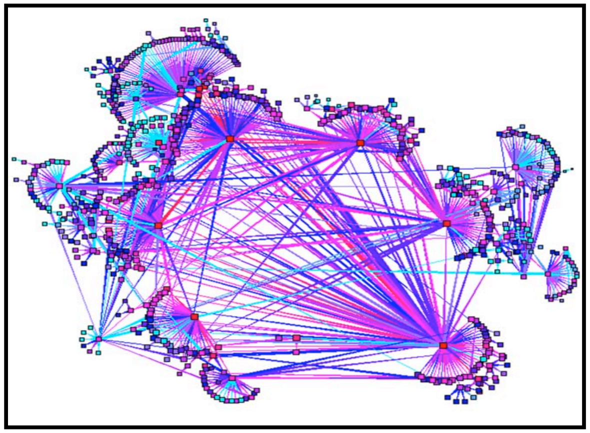
Simulate but Verify

- Applications Development with FPGAs
  - Reconfigurable Computing (RC)
    - Applications acceleration with dedicated or special-purpose HW
    - Programmable environment
  - Not a familiar or common programming paradigm in HPC

- Emerging Class of Applications in Complex Network Analysis (CNA)
  - Verification, rather than simulation, at their core
  - Roots in circuit design but …
    - Much indebted to theoretical computer science
  - HPC class type of applications
    - EDA
    - KDD

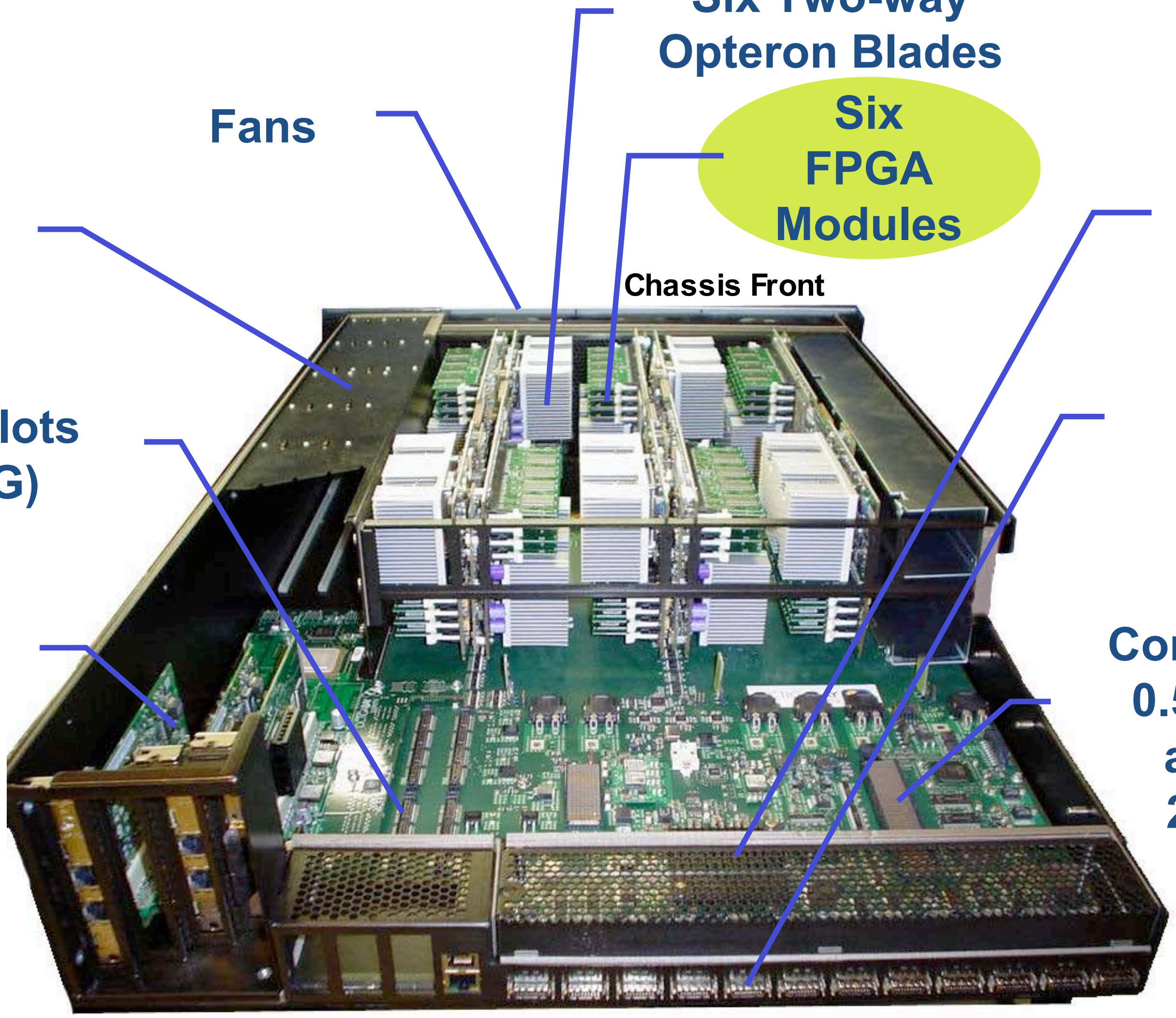


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library unisim;
use unisim.vcomponents.all;
]
entity byte_serial_adder is
  Port (
    clk    : in  std_logic;
    resetb : in  std_logic;
    x      : in  std_logic_vector(7 downto 0);
    y      : in  std_logic_vector(7 downto 0);
    s      : out std_logic_vector(7 downto 0)
    );
end byte_serial_adder;

architecture Behavioral of byte_serial_adder is

  component add8
    port (
      a       : in  std_logic_vector(7 downto 0);
      b       : in  std_logic_vector(7 downto 0);
      c_in    : in  std_logic;
      q_c_out : out std_logic;
      q       : out std_logic_vector(7 downto 0);
      clk     : in  std_logic);
  end component;
```



XD1 Architecture and FPGAs

Cray XD1



Six SATA Hard Drives

Three I/O Slots (e.g. JTAG)

Four 133 MHz PCI-X Slots

Fans

Six Two-way Opteron Blades

Six FPGA Modules

Chassis Front

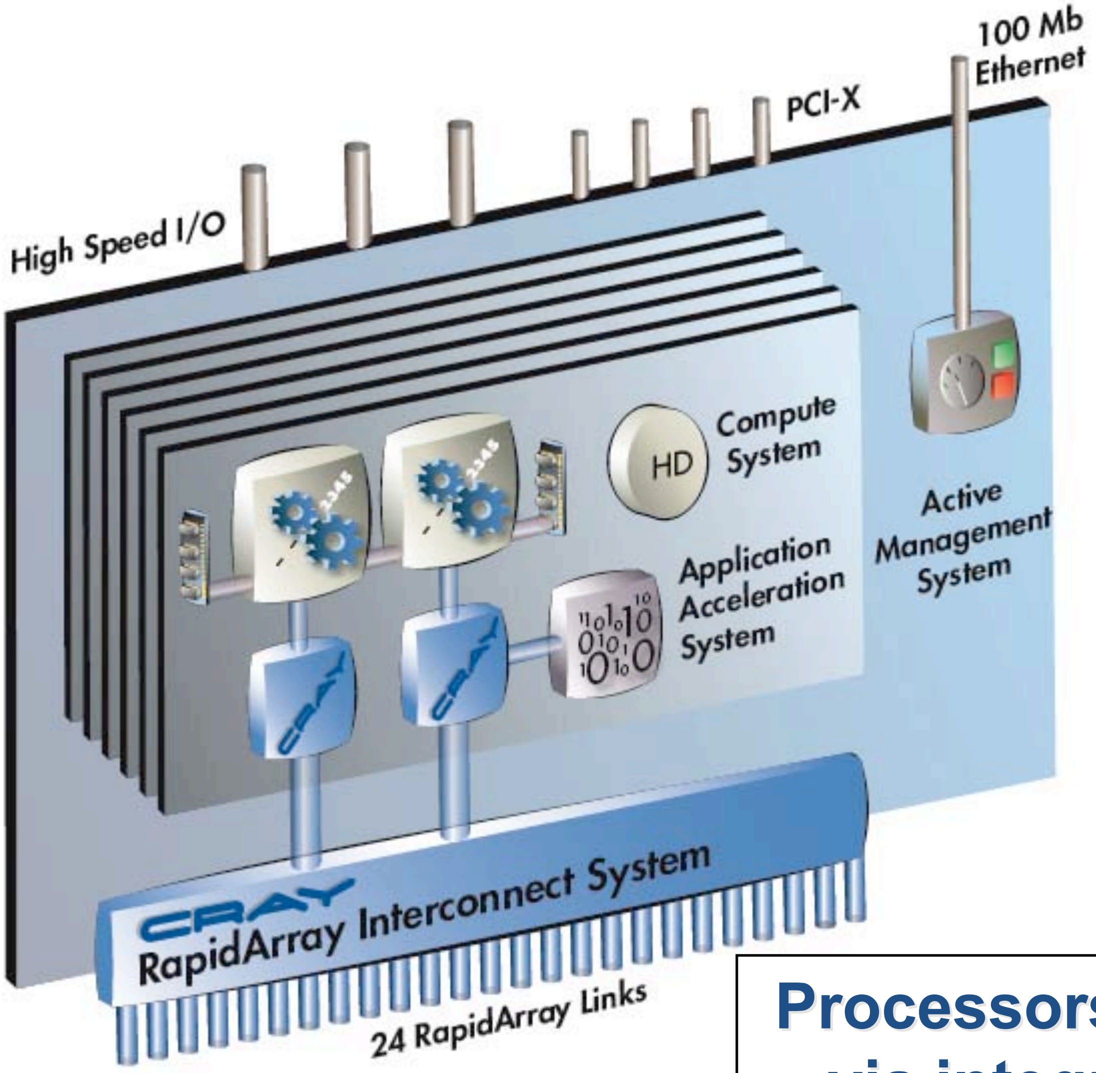
0.5 Tb/s Switch

12 x 2 GB/s Ports to Fabric

Connector for 2nd 0.5 Tb/s Switch and 12 More 2 GB/s Ports to Fabric

Chassis Rear

Cray XD1 System Architecture



Compute

- **12 AMD Opteron 32/64 bit, x86 processors**
- **High Performance Linux**

RapidArray Interconnect

- **12 communications processors**
- **1 Tb/s switch fabric**

Active Management

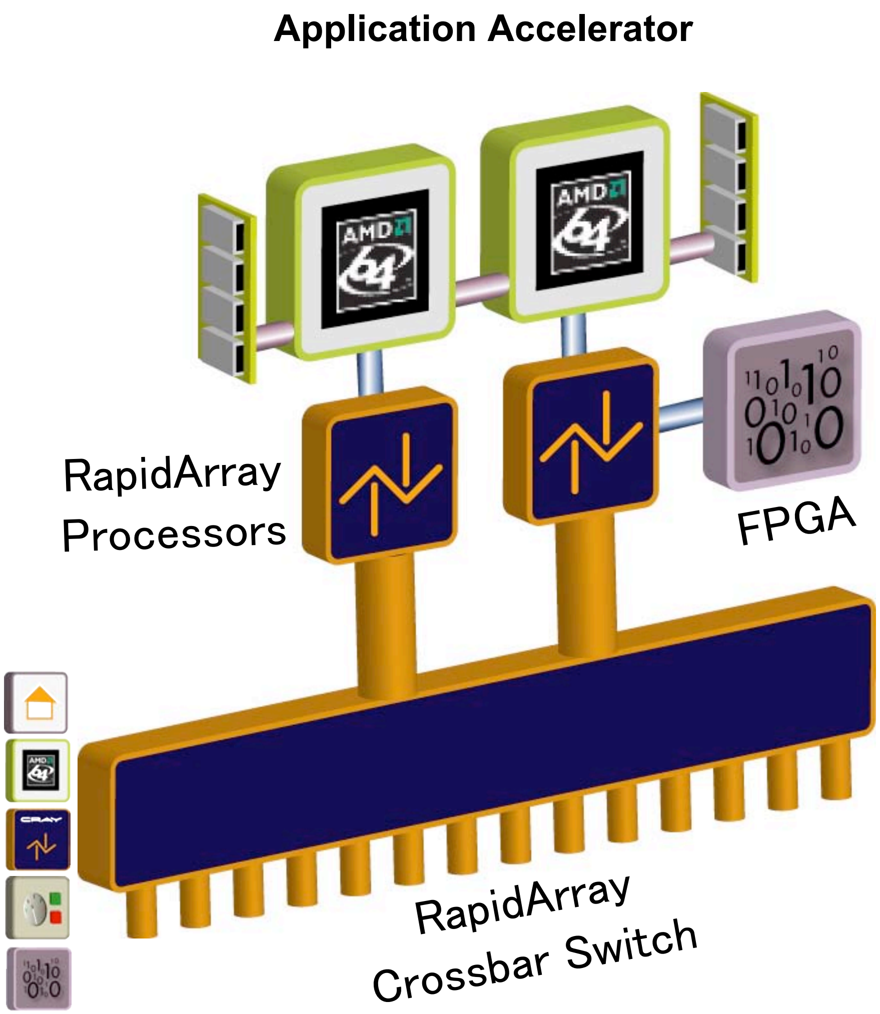
- **Dedicated processor**

Application Acceleration

- **6 co-processors (FPGAs)**

Processors directly connected via integrated switch fabric

Application Acceleration FPGA



Application Acceleration

- ***Reconfigurable Computing***
- **Tightly coupled to Opteron**
- **FPGA acts like a programmable co-processor**
- **Performs vector operations**
- **Well-suited for:**
  - Searching, sorting, signal processing, audio/video/image manipulation, encryption, error correction, coding/decoding, packet processing, random number generation.

SuperLinear speedup for key algorithms

Applications Acceleration

Application Acceleration FPGA

Well suited for:

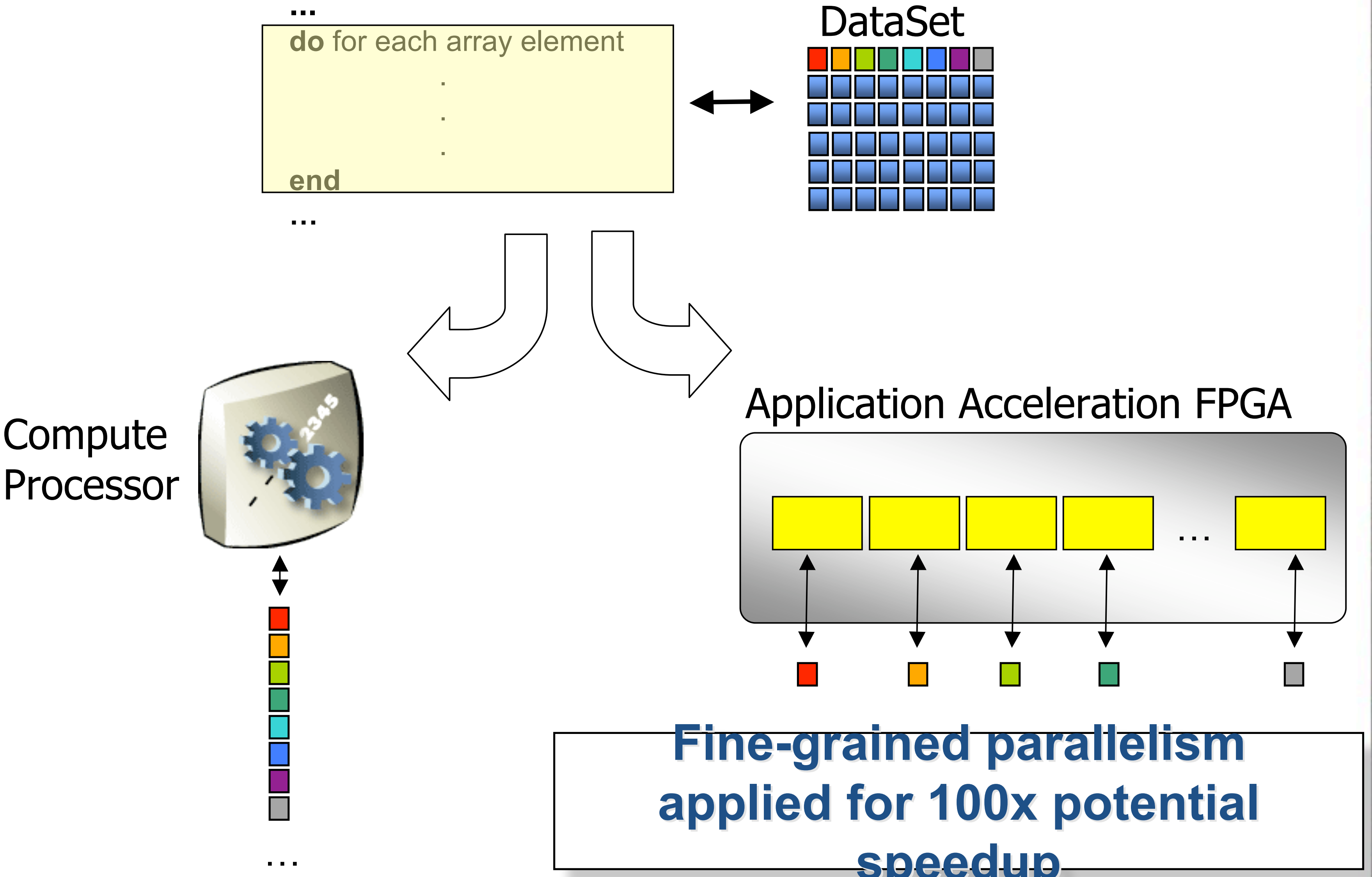
Searching, sorting, signal processing,
audio/video/image manipulation,
encryption, error correction, coding/decoding,
packet processing, random number generation …

But also:

Seismic imaging, molecular dynamics,
bioinformatics, …

Fine-grained parallelism applied for 100x potential speedup

Application Acceleration FPGA



```
...
do for each array element
                .
                .
                .
end
...
```

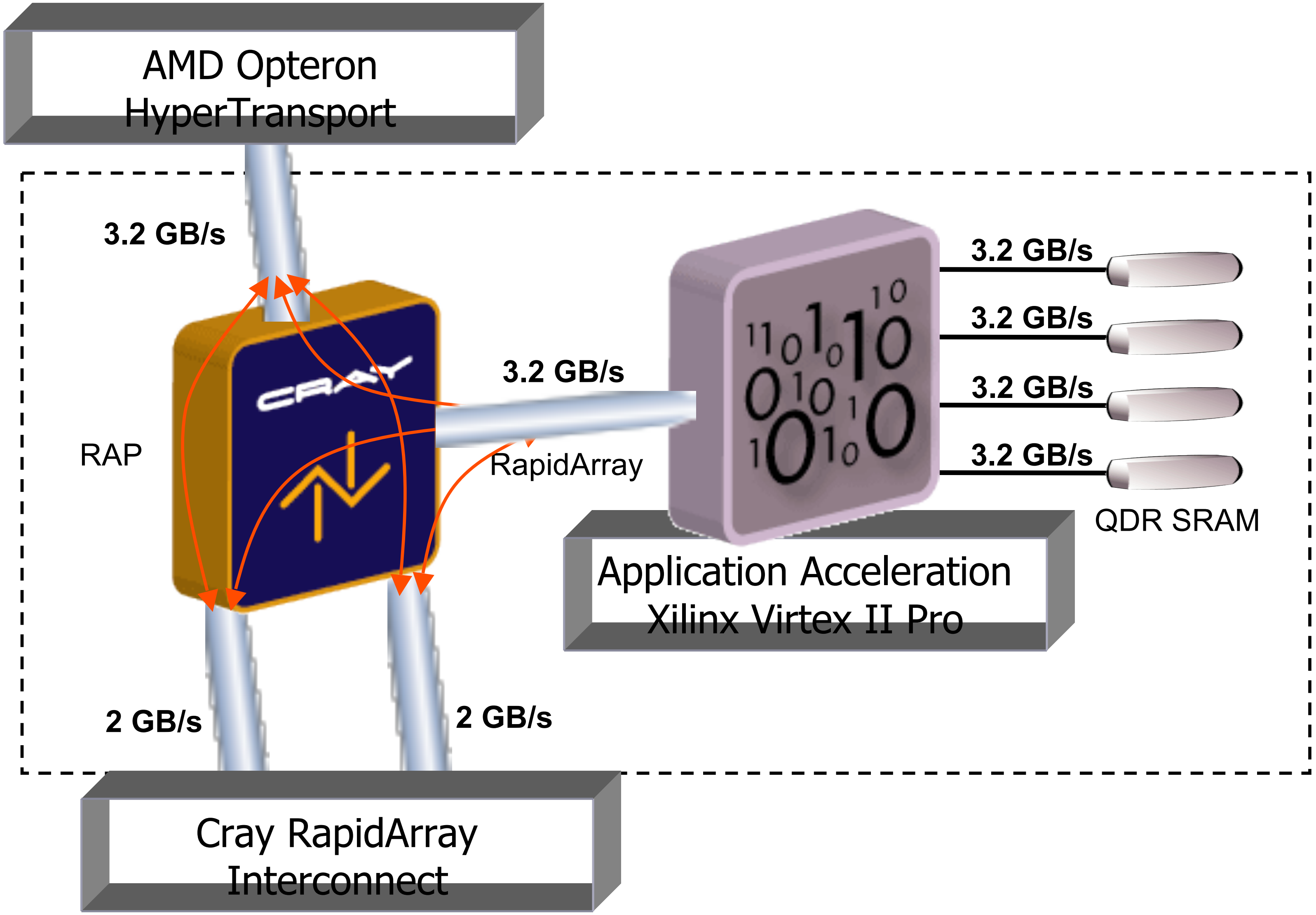
Fine-grained parallelism applied for 100x potential speedup

Reconfigurable Computing

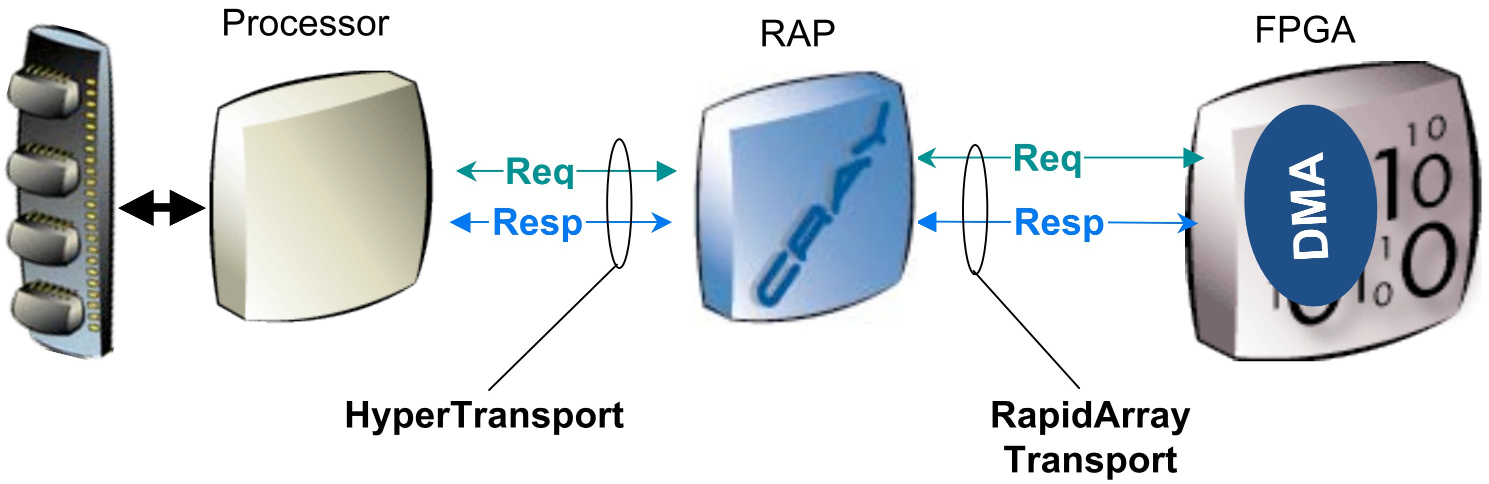
The Barriers to Reconfigurable Computing

- Starving the FPGA
  - Bandwidth and latency to the FPGA limited by PCI bus
- FPGA, Processor Interaction
  - Job scheduling, Linux integration, memory mapping
- Programming Tools
  - Programming hardware requires special tools, special expertise

Application Acceleration Co-Processor

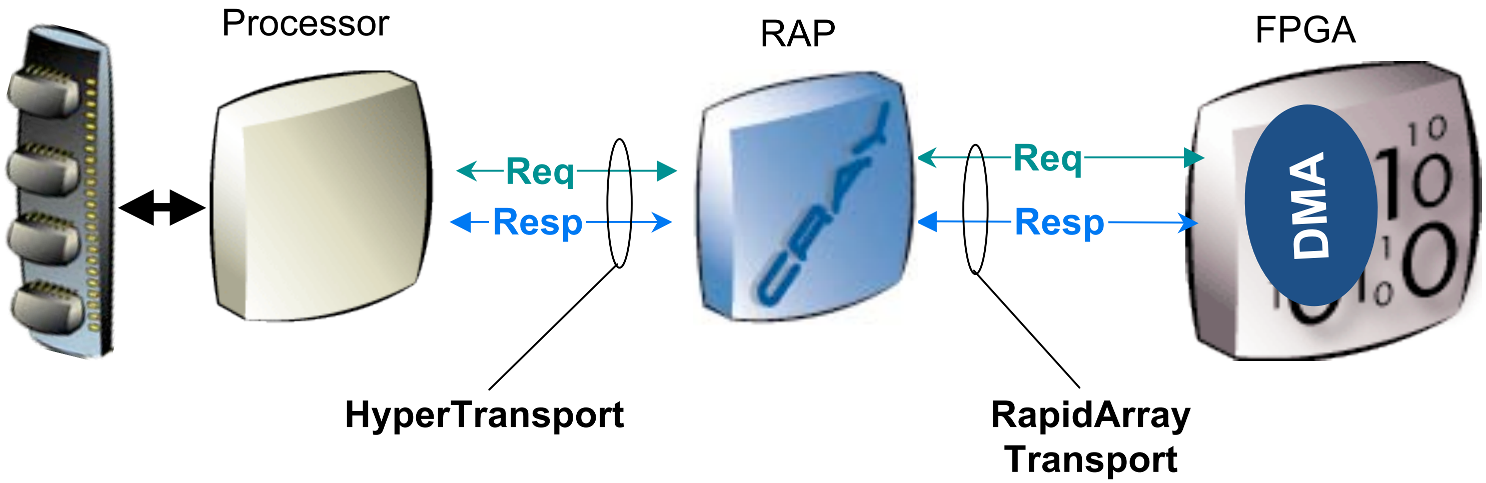


Processor to FPGA and *vice versa*



- Since the Acceleration FPGA is connected to the local processing node through its HyperTransport I/O bus, the FPGA can be accessed directly using reads and writes.
- Additionally, a node can also transfer large blocks of data to and from the Acceleration FPGA using a simple DMA engine in the FPGA's RapidArray Transport Core.
- The Acceleration FPGA can also directly access the memory of a processor. Read and write requests can be performed in bursts of up to 64 bytes.
- The Acceleration FPGA can access processor memory without interrupting the processor.
- Memory coherency is maintained by the processor.

Processor to FPGA and *vice versa*



	array	pointer	memcpy
Write (MB/s)	1260	1320	1320
Read(MB/s)	5.94	5.95	6.01

J. Tripp et al, FCCM'05

Reconfigurable Computing

The Barriers to Reconfigurable Computing

- Starving the FPGA
  - Bandwidth and latency to the FPGA limited by PCI bus
- FPGA, Processor Interaction
  - Job scheduling, Linux integration, memory mapping
- Programming Tools
  - Programming hardware requires special tools, special expertise

FPGA Linux API

- **Administration Commands**
  - fpga_open – allocate and open fpga
  - fpga_close – close allocated fpga
  - fpga_load – load binary into fpga
- **Control Commands**
  - fpga_start – start fpga (release from reset)
  - fpga_stop – stop fpga
- **Status Commands**
  - fpga_status – get status of fpga
- **Data Commands**
  - fpga_put – put data to fpga ram
  - fpga_get – get data from fpga ram
- **Interrupt/Blocking Commands**
  - fpga_intwait – blocks process waits for fpga interrupt

Programmer sees get/put and message passing programming model

Reconfigurable Computing

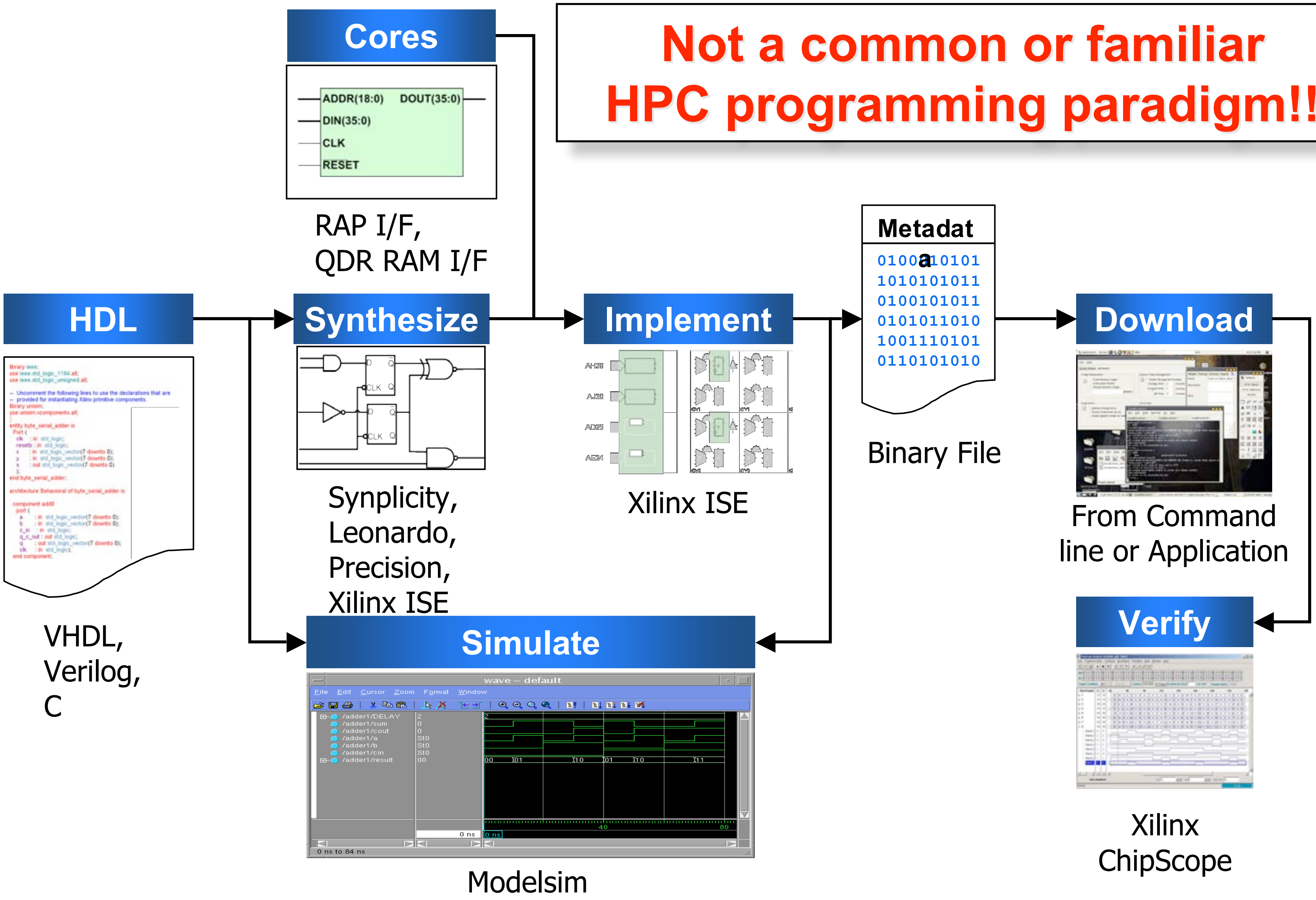
The Barriers to Reconfigurable Computing

- Starving the FPGA
  - Bandwidth and latency to the FPGA limited by PCI bus
- FPGA, Processor Interaction
  - Job scheduling, Linux integration, memory mapping
- Programming Tools
  - Programming hardware requires tools, expertise

Applications Development Framework

FPGA Development Flow

Not a common or familiar HPC programming paradigm!!



HDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
library unisim;
use unisim.vcomponents.all;

entity byte_serial_adder is
 Port (
  clk    : in  std_logic;
  resetb : in  std_logic;
  x      : in  std_logic_vector(7 downto 0);
  y      : in  std_logic_vector(7 downto 0);
  s      : out std_logic_vector(7 downto 0)
  );
end byte_serial_adder;

architecture Behavioral of byte_serial_adder is

 component add8
  port (
   a       : in  std_logic_vector(7 downto 0);
   b       : in  std_logic_vector(7 downto 0);
   c_in    : in  std_logic;
   q_c_out : out std_logic;
   q       : out std_logic_vector(7 downto 0);
   clk     : in  std_logic);
 end component;
```

VHDL,
Verilog,
C

Cores

RAP I/F,
QDR RAM I/F

Synthesize

Synplicity,
Leonardo,
Precision,
Xilinx ISE

Implement

Xilinx ISE

Metadat
a

```
0100a10101
1010101011
0100101011
0101011010
1001110101
0110101010
```

Binary File

Download

From Command
line or Application

Verify

Xilinx
ChipScope

Simulate

```
/adder1/DELAY   2
/adder1/sum     0
/adder1/cout    0
/adder1/a       St0
/adder1/b       St0
/adder1/cin     St0
/adder1/result  00
                0 ns
0 ns to 84 ns
```

Modelsim

Three-Phase Implementation

- Traditional Programming Model
  - VHDL, Verilog
- Off-The-Shelf Libraries
  - Cray and third party *acceleration libraries*
  - Prepackaged, turnkey *applications*
- High-Level Compilers
  - C, Graphical, Matlab, …

- *Leverage of existing IP Cores (Xilinx, OpenCore.org) or industry/academic initiatives(OpenFPGA)*
- *Academic collaborators*

First Level Abstraction: VHDL

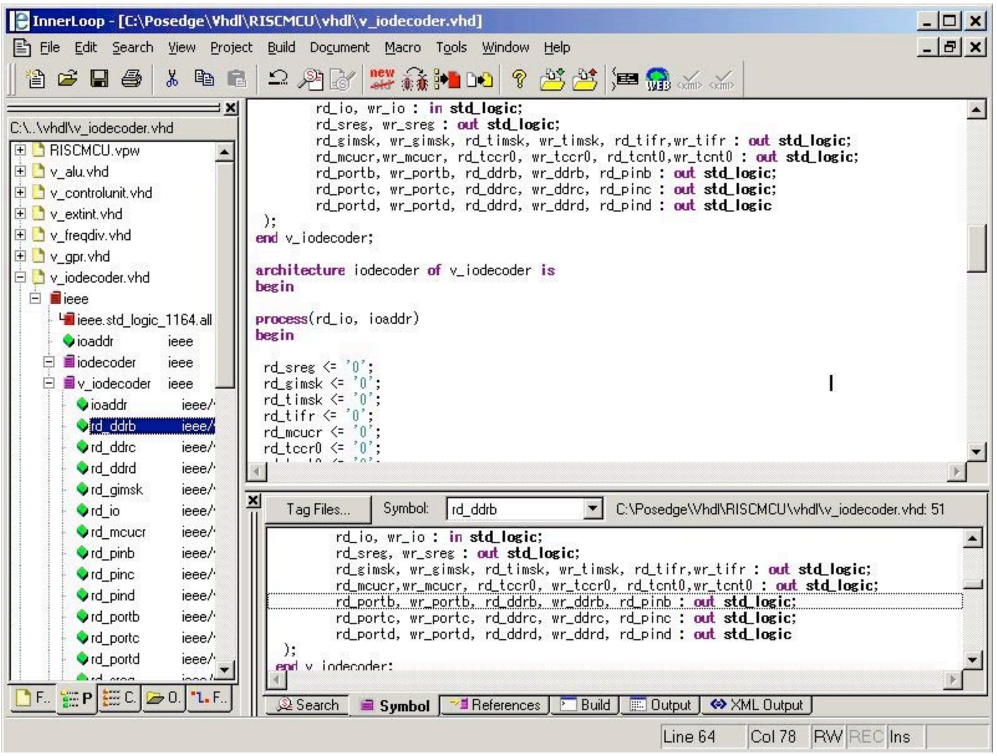
http://www.eda.org/fphdl/

Floating-Point HDL Packages Home Page

Working on a floating point synthesis package for VHDL and Verilog based on IEEE 754. We are a task force assigned to the 1076.3 working group and will be releasing our code as part of that IEEE PAR.

Group Objectives:

Create a parameterized package for variable width floating point in both VHDL and Verilog.

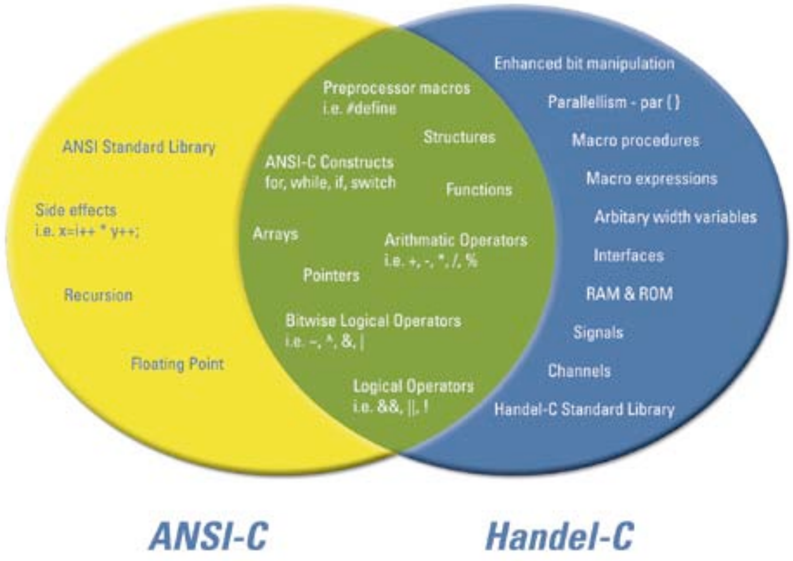


```
        rd_io, wr_io : in std_logic;
        rd_sreg, wr_sreg : out std_logic;
        rd_gimsk, wr_gimsk, rd_timsk, wr_timsk, rd_tifr,wr_tifr : out std_logic;
        rd_mcucr,wr_mcucr, rd_tccr0, wr_tccr0, rd_tcnt0,wr_tcnt0 : out std_logic;
        rd_portb, wr_portb, rd_ddrb, wr_ddrb, rd_pinb : out std_logic;
        rd_portc, wr_portc, rd_ddrc, wr_ddrc, rd_pinc : out std_logic;
        rd_portd, wr_portd, rd_ddrd, wr_ddrd, rd_pind : out std_logic
 );
end v_iodecoder;

architecture iodecoder of v_iodecoder is
begin

process(rd_io, ioaddr)
begin

 rd_sreg <= '0';
 rd_gimsk <= '0';
 rd_timsk <= '0';
 rd_tifr <= '0';
 rd_mcucr <= '0';
 rd_tccr0 <= '0';
```

```
        rd_io, wr_io : in std_logic;
        rd_sreg, wr_sreg : out std_logic;
        rd_gimsk, wr_gimsk, rd_timsk, wr_timsk, rd_tifr,wr_tifr : out std_logic;
        rd_mcucr,wr_mcucr, rd_tccr0, wr_tccr0, rd_tcnt0,wr_tcnt0 : out std_logic;
        rd_portb, wr_portb, rd_ddrb, wr_ddrb, rd_pinb : out std_logic;
        rd_portc, wr_portc, rd_ddrc, wr_ddrc, rd_pinc : out std_logic;
        rd_portd, wr_portd, rd_ddrd, wr_ddrd, rd_pind : out std_logic
 );
end v_iodecoder;
```

Higher Level Abstractions: C Languages

- Requirements
  - At a minimum:
    - Multi-threaded
    - Communication and synchronization channels
    - Bit level manipulations
  - Others:
    - Good-to-excellent QoR
    - Target architectures
    - Standards?



ANSI Standard Library

Side effects
i.e. x=i++ * y++;

Recursion

Floating Point

Preprocessor macros
i.e. #define

Structures

ANSI-C Constructs
for, while, if, switch

Functions

Arrays

Arithmatic Operators
i.e. +, -, *, /, %

Pointers

Bitwise Logical Operators
i.e. ~, ^, &, |

Logical Operators
i.e. &&, ||, !

Enhanced bit manipulation

Parallelism - par ( )

Macro procedures

Macro expressions

Arbitary width variables

Interfaces

RAM & ROM

Signals

Channels

Handel-C Standard Library

ANSI-C

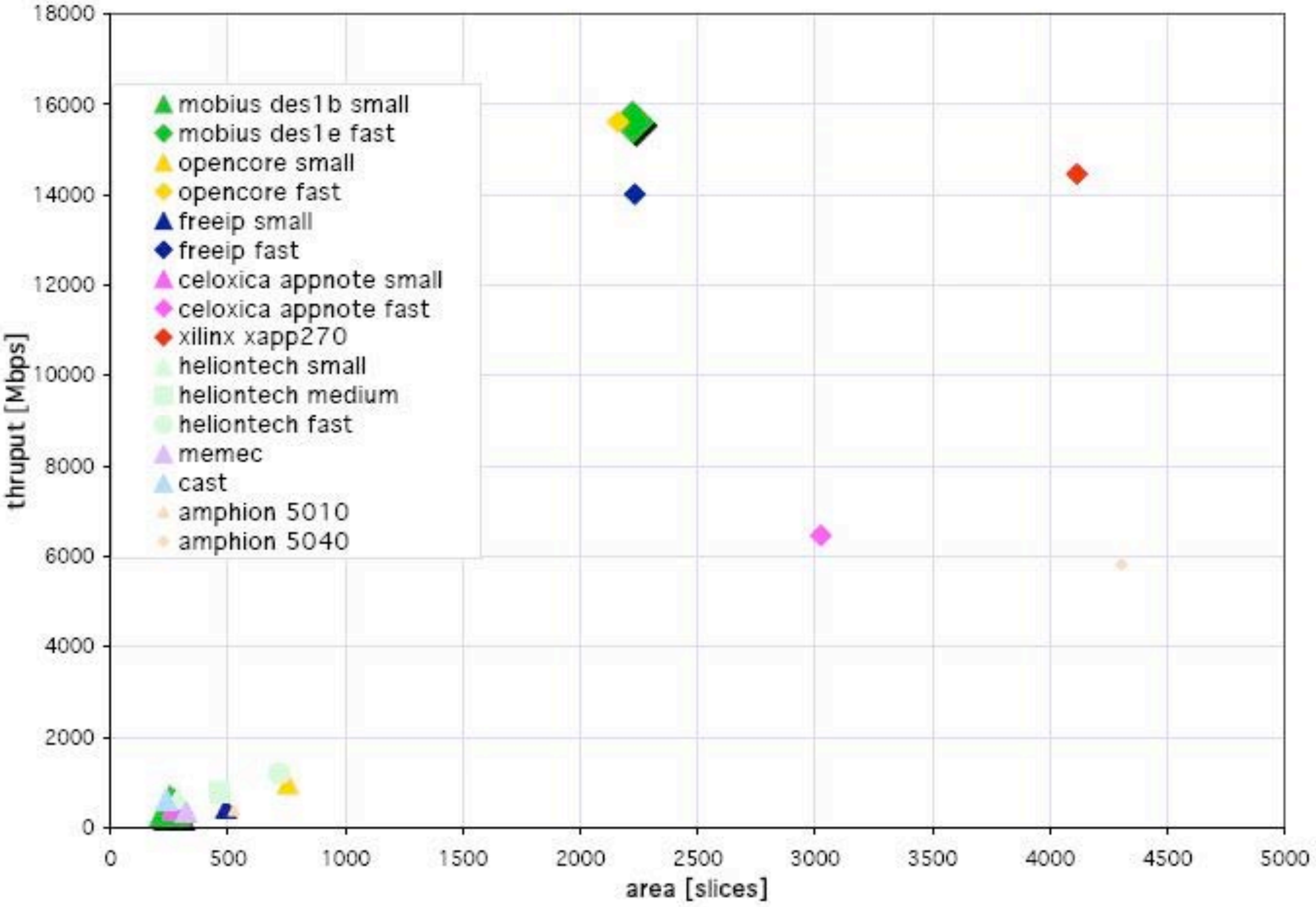
Handel-C

Higher Level Abstractions: Mobius

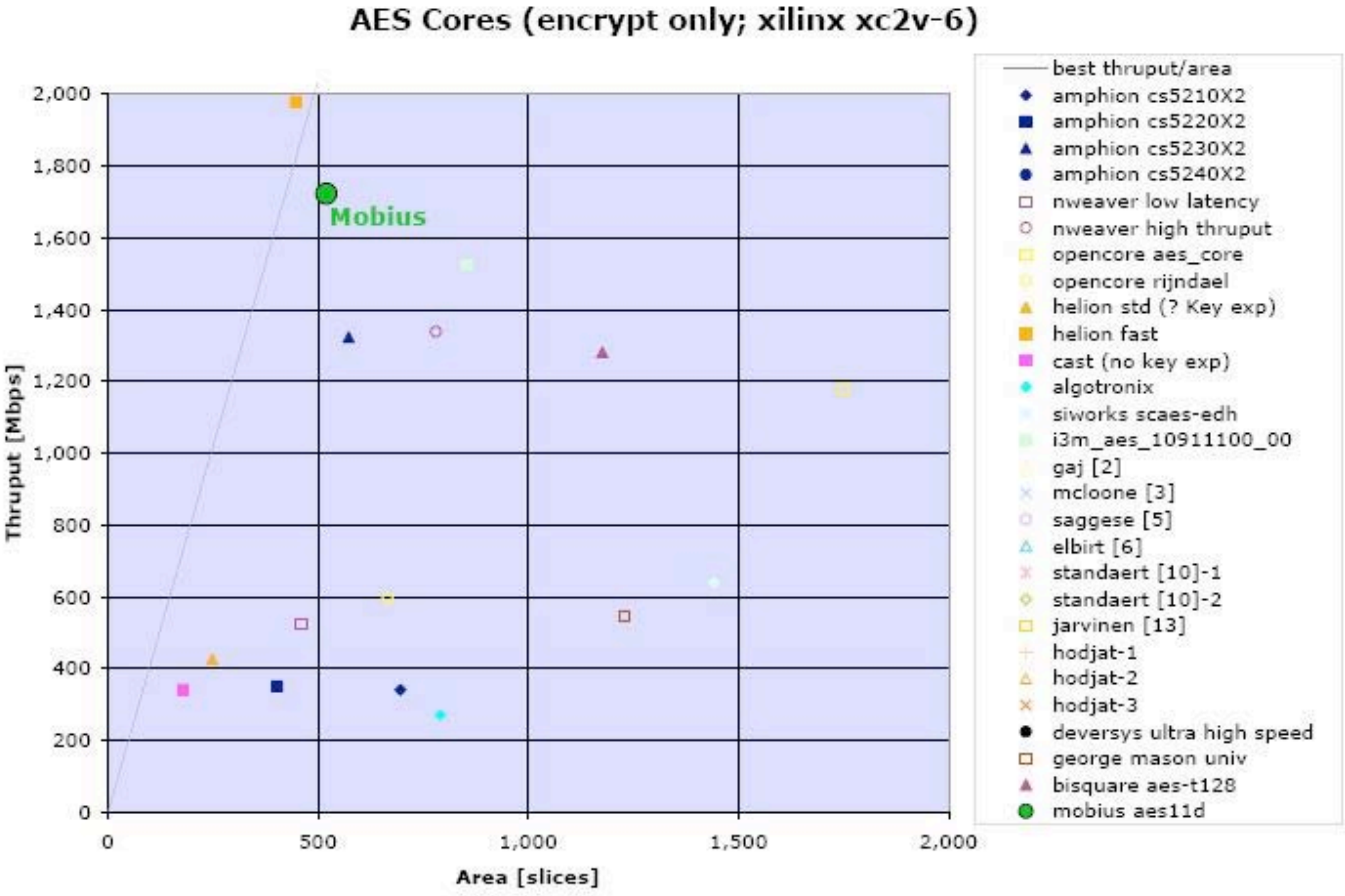
(www.codetronix.com)

- Pascal-like CSP based language
  - Types ,records, arrays, fp arithmetic
- Synchronization and communication by handshaking over channels
- Generate HW, SW or HW/SW code
- General purpose & dataflow algorithms

FPGA Applications: DES

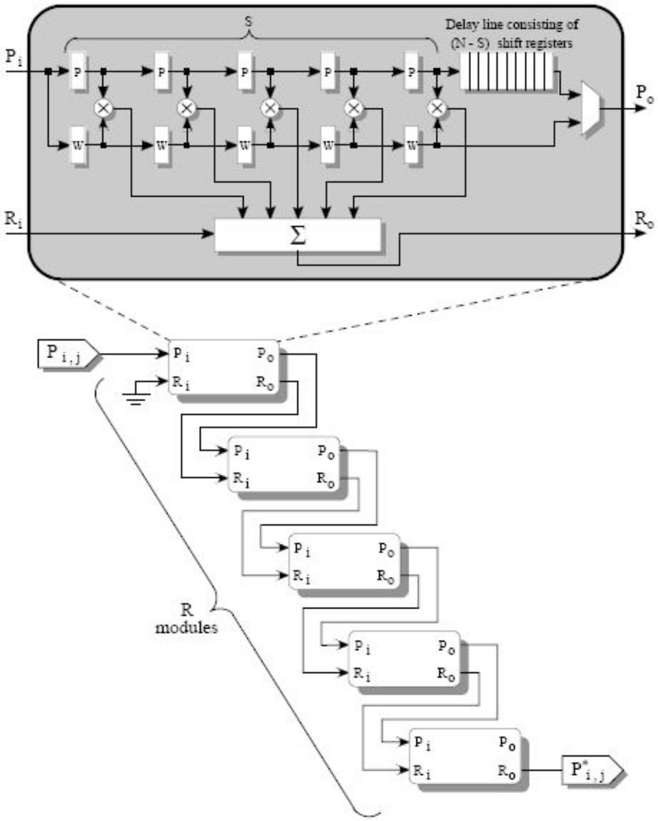


FPGA Applications: AES



FPGA Applications

FPGA Applications: 1D Convolution

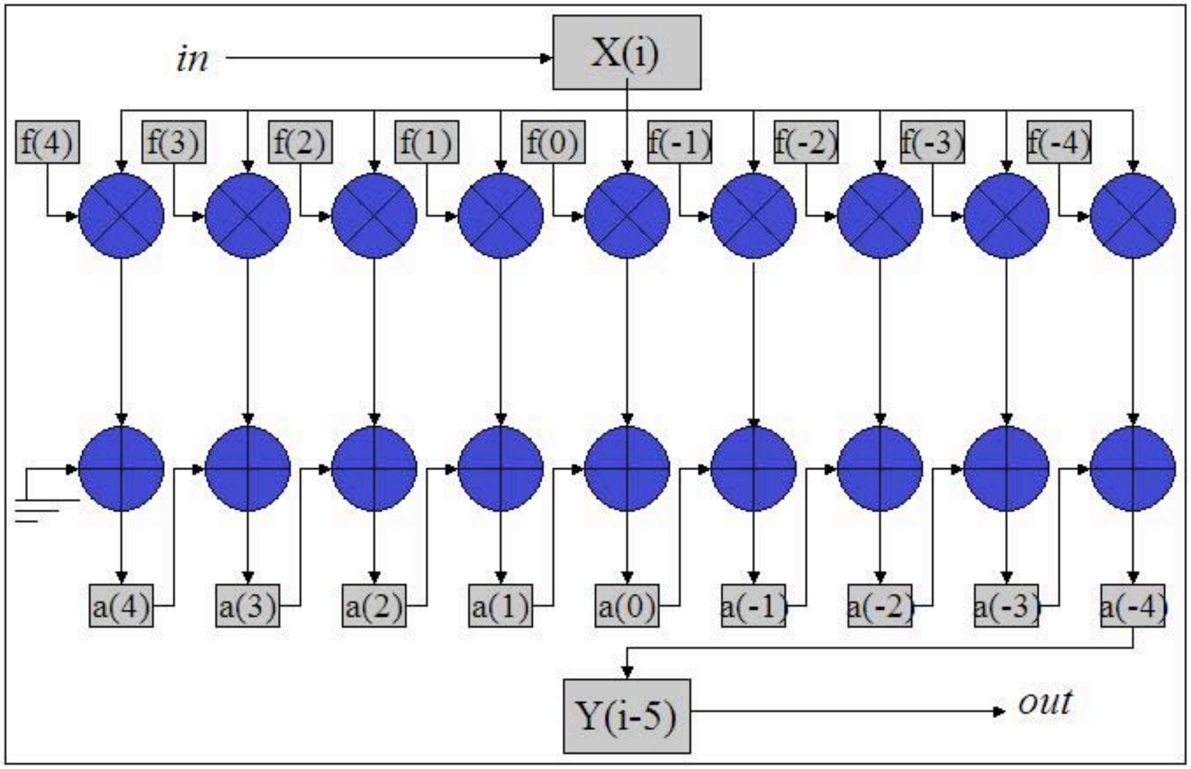


$$P^*(i) = \sum_{n=-N}^{N} w(n)P(i-n)$$

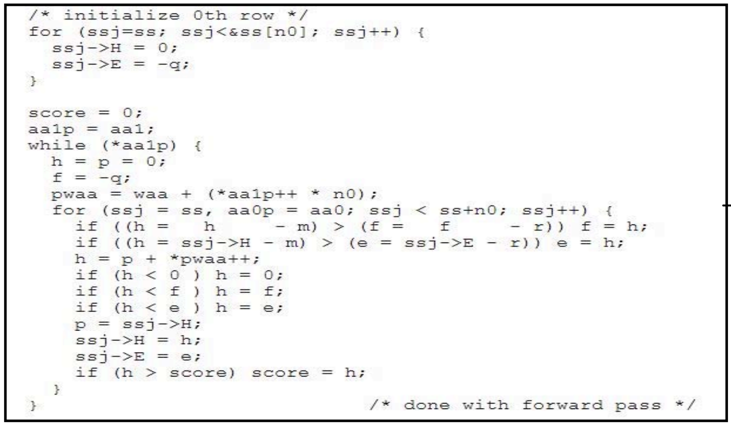
FPGA Applications: 1D Convolution

$$Y(i)=\sum_{n=-N}^{N} f(n)X(i-n)$$

Convolution Unit



Example: Smith-Waterman



```
/* initialize 0th row */
for (ssj=ss; ssj<&ss[n0]; ssj++) {
  ssj->H = 0;
  ssj->E = -q;
}

score = 0;
aa1p = aa1;
while (*aa1p) {
  h = p = 0;
  f = -q;
  pwaa = waa + (*aa1p++ * n0);
  for (ssj = ss, aa0p = aa0; ssj < ss+n0; ssj++) {
    if ((h =   h      - m) > (f =   f      - r)) f = h;
    if ((h = ssj->H - m) > (e = ssj->E - r)) e = h;
    h = p + *pwaa++;
    if (h < 0 ) h = 0;
    if (h < f ) h = f;
    if (h < e ) h = e;
    p = ssj->H;
    ssj->H = h;
    ssj->E = e;
    if (h > score) score = h;
  }
}                              /* done with forward pass */
```

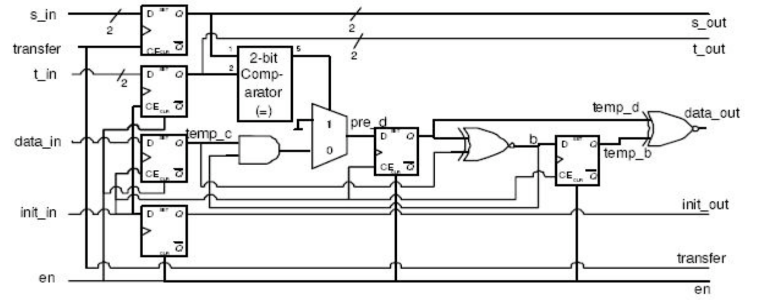


Fig. 3. The Smith-Waterman processing element (PE). Boxes represent D-type flip-flops.

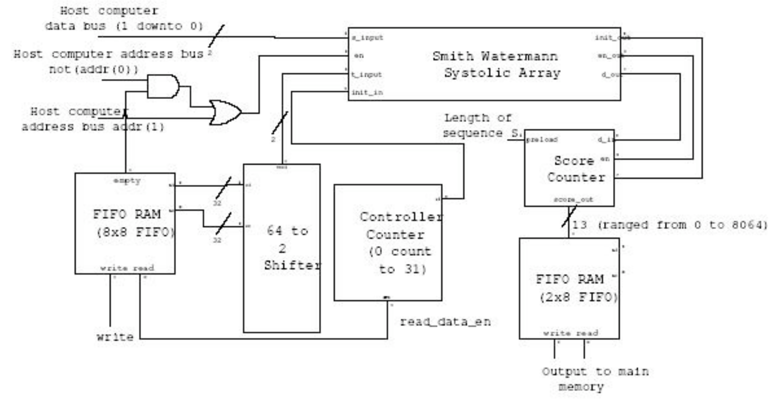
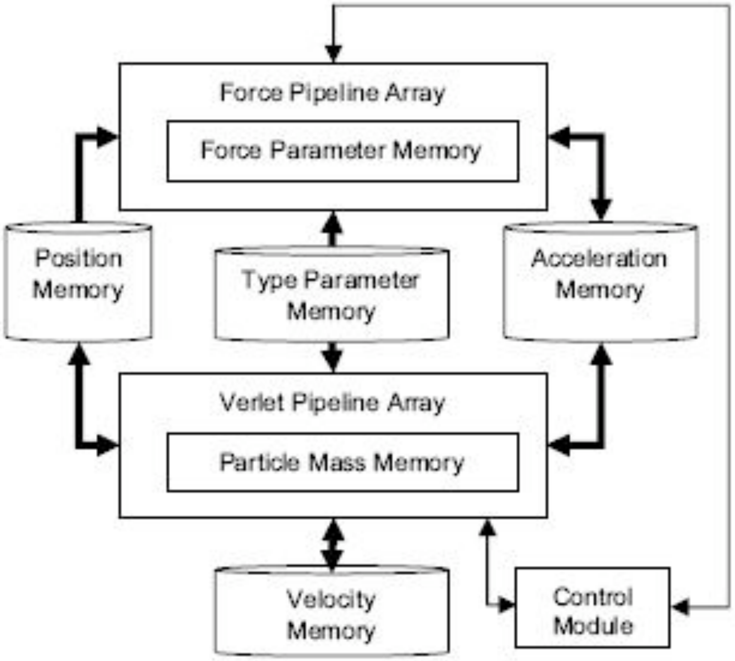


Fig. 5. System data path.

Biosciences: Molecular Design

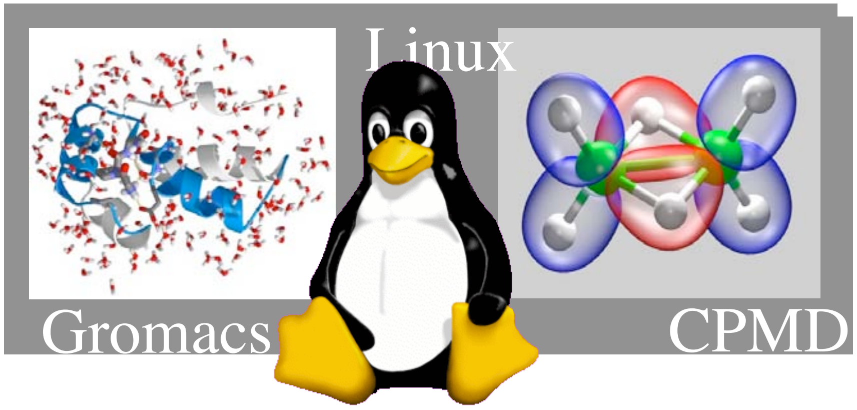
- RC for Molecular Modeling & Docking
  - Reduced Precision
    - Fidelity of observed macro phenomena
    - High frequency/low frequency
  - Critical resource: hard multipliers
  - O(N logN) methods



$$\mathbf{F}_i^C = q_i \sum_{j\neq i} \left(\frac{q_j}{|\mathbf{r}_{ji}|^3}\right)\mathbf{r}_{ji}$$

$$\mathbf{F}_i^{LJ} = \sum_{j\neq i}\frac{\epsilon_{ab}}{\sigma_{ab}^2}\left\{12\left(\frac{\sigma_{ab}}{|r_{ji}|}\right)^{14} - 6\left(\frac{\sigma_{ab}}{|r_{ji}|}\right)^{8}\right\}\mathbf{r}_{ji}$$

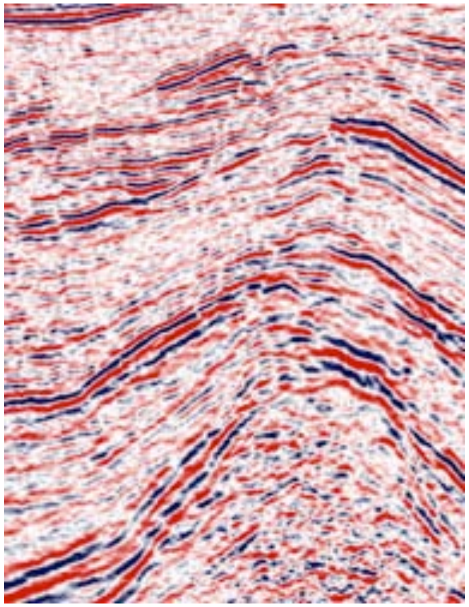
Y. Gu et al, FCCM'05



Sample Applications for Acceleration

Kirchhoff Pre-stack Time Migration

- Estimated or preliminary speed-ups of 10-50 versus 2.4 GHz Pentium 4
  - 50M Kirchhoff summations per second
- Power consumption < x2
- Footprint ~ x1.0



(TAMU)

Cost Analysis

- Commodity industry cost structure is driven by
  - Cost of infrastructure
  - Cost of operation (power, cooling, …)
- Move processing infrastructure closer to acquisition infrastructure

Verification Languages

Verification Languages and LTL

- Linear Temporal Logic (LTL)
    - The usual Boolean propositional logic (AND, OR, NOT)
    - Temporal operators (NEXT, UNTIL)
    - Quantifiers (FORALL, EXIST-ONE)
- In EDA:
    - Mathematical basis of verification languages
        - Mathematical syntax akward
- In CS:
    - Concurrency and computer-aided SW verification
- In CNA (complex network analysis):
    - Control (output regulation, stabilization, …) of complex systems

What is Network Analysis…?

Goal:

Develop comprehensive, mathematically rigorous, and empirically grounded framework within which to understand complex networks and apply understanding to real world problems.

Applications of interest:

- Social systems (e.g., terrorist networks, WMD programs, socioeconomic systems);
- Biological networks (e.g., gene regulatory networks, metabolic, protein interaction);
- Technological systems (e.g., EP grids
- Information systems (e.g., www);

Analysis:

- Information extraction
- Network control

What are Networks …?

Nodes:

- Extremely simple dynamic systems
  - (low cognition agents, Boolean), …

Edges:

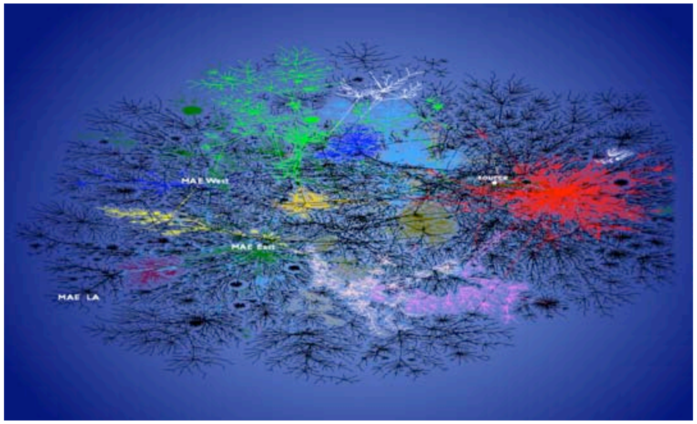
- Uni/bi-directional

Interaction (Control, Communication, and Synchronization):

- Linear Temporal Logic (LTL)

Questions asked:

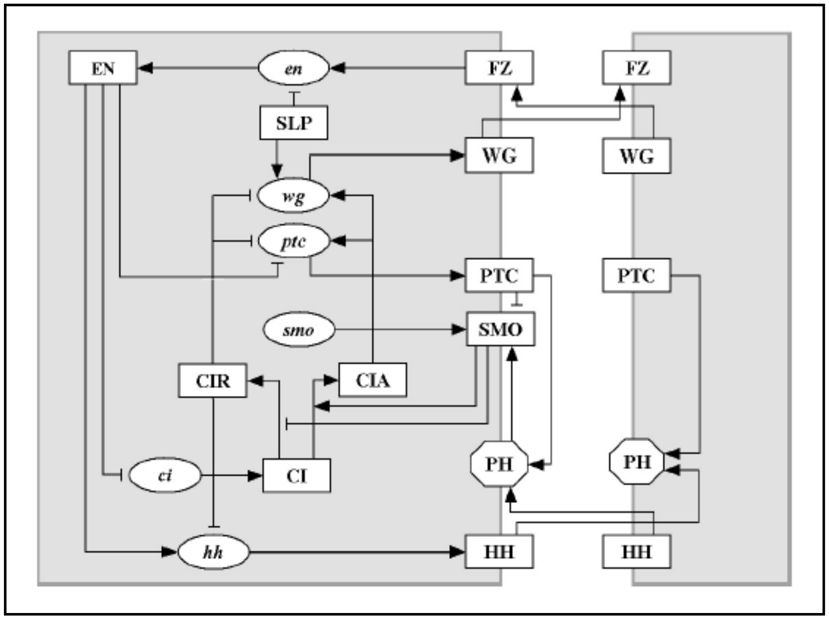
- Is there a solution?
- Finite number of transitions?
- Is there a path?
- What is the shortest path?



Gene Regulatory Networks in LTL

Drosophila gene regulatory network

Standard model for segment polarity gene network admits finite bisimulation which preserves state space equilibrium structure and, therefore, gene expression patterns.



gene interaction network

$$hh_i^{t+1} = EN_i^t \text{ and not } CIR_i^t$$

$$en_i^{t+1} = (WG_{i-1}^t \text{ or } WG_{i+1}^t) \text{ and not } SLP_i^t$$

$$ptc_i^{t+1} = CIA_i^t \text{ and not } EN_i^t \text{ and not } CIR_i^t$$

$$ci_i^{t+1} = \text{not } EN_i^t$$

$$EN_i^{t+1} = en_i^t$$

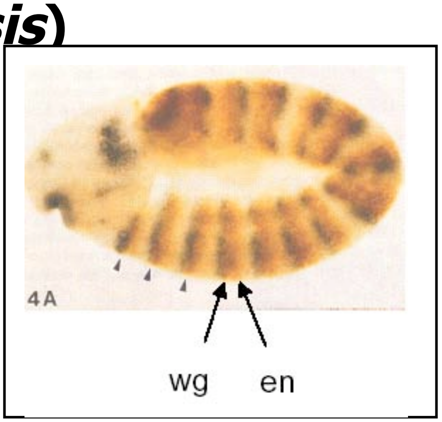
$$WG_i^{t+1} = wg_i^t$$

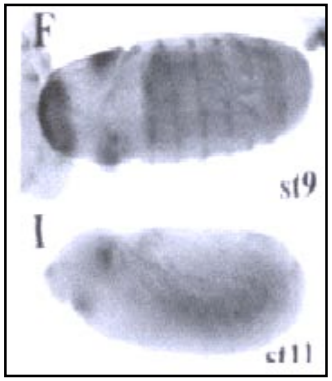
$$CI_i^{t+1} = ci_i^t$$

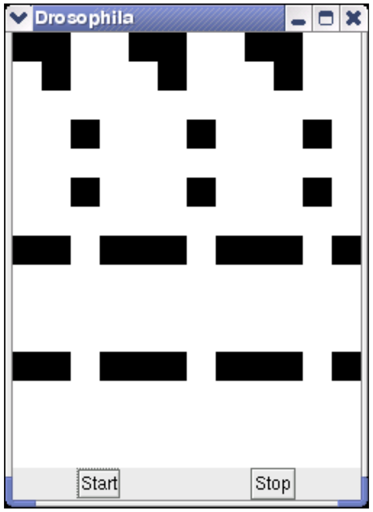
$$HH_i^{t+1} = hh_i^t$$

sample vertex update rules

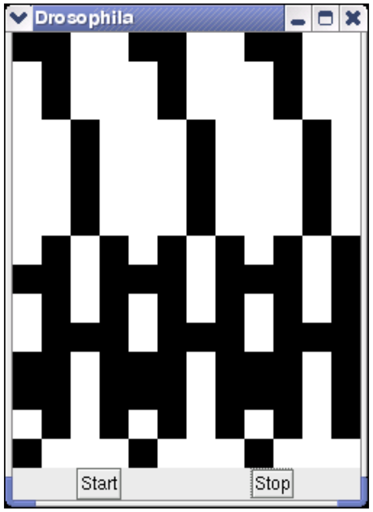
Drosophila gene regulatory network (*Colbaugh, Glass analysis*)







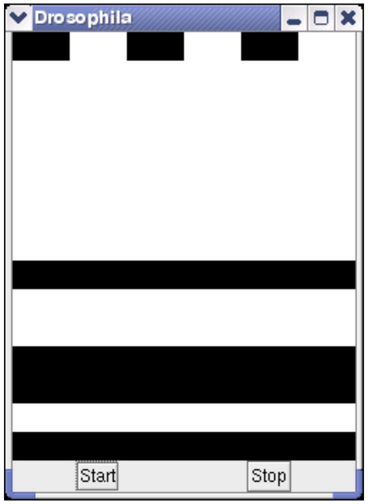
initial state



wg

en

normal equilibrium



mutant equilibrium