Cray ARSC Presentation
XD1 FPGA
Cray XD1

- Product Overview
- Interconnect
- FPGA Application Acceleration Module
- Linux C API
- FPGA development flow

Questions?
The Cray XD1

Purpose-built and optimized for high performance workloads

- Built for price performance
- 30 times interconnect performance
- 2 times the density
- High availability
- Single system command & control
Cray XD1 System Architecture

**Compute**
- 12/24 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

**Processors directly connected via integrated switch fabric**
Interconnect
Cray XD1 Interconnect System

RapidArray
- Interconnect processors
- Switch fabric
- Communications software
HPC applications exhibit intense compute/communicate cycles

- 20% - 60% of time, CPUs sit idle, stalled by communications
- Application performance is very sensitive to latency and bandwidth

Interconnect Drives System Performance
Cray Communications Libraries
- MPI 1.2 library
- TCP/IP
- PVM
- Shmem
- Global Arrays
- System-wide process & time synchronization

RapidArray Communications Processor
- HT/RA tunnelling with bonding
- Routing with route redundancy
- Reliable transport
- Short message latency optimization
- DMA operations
- System-wide clock synchronization

Direct Connected Processor Architecture
Cray XD1 latency is 4 times lower than Infiniband. Cray XD1 can send 2 KB before Infiniband sends its first byte.
Direct Connect Topology

1 Cray XD1 Chassis
12 AMD Opteron Processors
53 GFLOPS
8 GB/s between SMPs
1.6 μsec interconnect
Integrated switching

3 Cray XD1 Chassis
36 AMD Opteron Processors
158 GFLOPS
8 GB/s between SMPs
1.8 μsec interconnect
Integrated switching

25 Cray XD1 Chassis, two racks
300 AMD Opteron Processors
1.3 TFLOPS
2 - 8 GB/s between SMPs
1.8 μsec interconnect
Integrated switching
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
QDR II Core

64 bit Read every clock

64 bit Write every clock

QDR II Core

D(35:0)
Q(35:0)
ADDR(19:0)
R
W
BW(3:0)
K_CLK
C_CLK

3.2 GB/s per RAM x 4 RAMs

Separate DDR Read and Write Busses

QDR I/F

rd_addr(19:0)
rd_data(71:0)
read
wr_addr(19:0)
wr_data(71:0)
write
byte_en(7:0)

200 MHz x 2 transfers/clock x 4 bytes/transfer x 2 ports/RAM x 4 RAMs = 12.8 GB/s

Cray Inc.
• 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.
FPGA Linux API

- **Administration Commands**
  - `fpga_open/close` – allocate or deallocate the fpga
  - `fpga_load/unload` – program or clear the fpga

- **Operation Commands**
  - `fpga_start/reset` – release the user logic from or place it into reset

- **Communication Commands**
  - `fpga_register/dreg_ftrmem` – map application memory to allow access by fpga
  - `fpga_memmap` – map fpga ram into application virtual space
  - `fpga_rd/wrt_appif_val` – read/write data from/to the FPGA
  - `fpga_intwait` – blocks process waits for fpga interrupt

- **Status Commands**
  - `fpga_is_loaded` – check if the fpga has been programmed
  - `fpga_status` – get status of fpga
FPGA Development Flow

1. HDL
   - VHDL, Verilog, C

2. Synthesize
   - Synplicity, Leonardo, Precision, Xilinx ISE

3. Implement
   - Xilinx ISE

4. Simulate
   - Modelsim

5. Download
   - Binary File
     - From Command line or Application

6. Verify
   - Xilinx ChipScope

Cores
- ADDR(18:0), DOUT(35:0)
- DIN(35:0)
- CLK
- RESET

RAP I/F, QDR RAM I/F

Metadat
- 01010101
- 1010101011
- 0100101011
- 0101011010
- 1001110101
- 0110101010
- 0110101010

0100101011
Additional High Level Tools

Adelante
Celoxica
Forte Design Systems
Mentor Graphics
ProsiLog
Synopsis

C Synthesis

process (a, m) is begin
z <= a and m;
end process;

int mask(a, m) {
  return (a & m);
}

SystemC,
ANSI C/C++

MATLAB/
Simulink

The MathWorks

System for
DSP

VHDL/Verilog

Xilinx

VHDL/Verilog Synthesis

Gate Level
EDIF File

Place and
Route

Binary File
for FPGA

Xilinx

Mentor Graphics
Synopsis
Synplicity
Xilinx

High Level
Flow

Standard
Flow
Questions?
• FPGA implements “Mersenne Twister” RNG algorithm often used for Monte Carlo analysis. The algorithm generates integers with a uniform distribution and won’t repeat for $2^{19937}-1$ values.

• FPGA automatically transfers generated numbers into two buffers located in the processor’s local memory.

• Processor application alternately reads the pseudo-random numbers from two buffers. As processor marks the buffers as ‘empty’, the FPGA refills them with new numbers.
## Random Number Results

<table>
<thead>
<tr>
<th>Source</th>
<th>Original C Code</th>
<th>VHDL Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>2.2 GHz Opteron</td>
<td>FPGA (XC2VP30-6) @ 200 MHz</td>
</tr>
<tr>
<td>Speed (32 bit integers/second)</td>
<td>~101 Million</td>
<td>~319 Million</td>
</tr>
<tr>
<td>Size</td>
<td>N/A</td>
<td>~25% of chip (includes RapidArray Core)</td>
</tr>
</tbody>
</table>

- FPGA provides 3X performance of fastest available Opteron.
- Algorithm takes up a small portion of the smallest FPGA.
- Performance is limited by speed at which numbers can be written into processor memory, not by FPGA logic. The logic could easily produce 1.6 billion integers/second by increasing parallelism.
• The current design leaves the FPGA 75% empty. This allows plenty of space to implement additional processing functions.

• One such function would be to normalize the distribution. On an Opteron, this drops the performance down to 3-4 Million 32 bit FP numbers per second (Box-Meuller algorithm).

• On the FPGA, a normal distribution could be produced by adding several uniformly distributed numbers (Central Limit Theorem). Since the algorithm can be further parallelized to produce integers at even higher rates internally, the FPGA could still produce normally distributed numbers at over 100 million per second.
Virtex-II Family Logic Blocks

1 LE = LUT + Register
1 Slice = 2 LEs
1 CLB = 4 Slices

### XC2VP30-6 Examples

<table>
<thead>
<tr>
<th>Function</th>
<th>f (MHz)</th>
<th>LE’s</th>
<th>BRAM</th>
<th>Mult.</th>
<th>Number Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bit Adder</td>
<td>194</td>
<td>66</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>64 bit Accumulator</td>
<td>198</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>18 x 18 Multiplier</td>
<td>259</td>
<td>88</td>
<td>0</td>
<td>1</td>
<td>136</td>
</tr>
<tr>
<td>SP FP Multiplier</td>
<td>188</td>
<td>252</td>
<td>0</td>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>1024 FFT (16 bit complex)</td>
<td>140</td>
<td>5526</td>
<td>22</td>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>
Switchless Torus Topology

27 chassis, 324 Processors
1.4 TFLOP
3x3x3 Chassis Torus

Nearest Neighbor
4-8 GB/s bandwidth per SMP
1.8 μsec latency

Worst case: 6 Hops, 2.8 μsec

Well-Suited for Nearest Neighbor Problems