Compiling Software Code to FPGA-Based Application Accelerator Processors for the XD1

Doug Johnson

David Gardner

Celoxica, Inc.

doug.johnson@celoxica.com

+1-310-543-2468

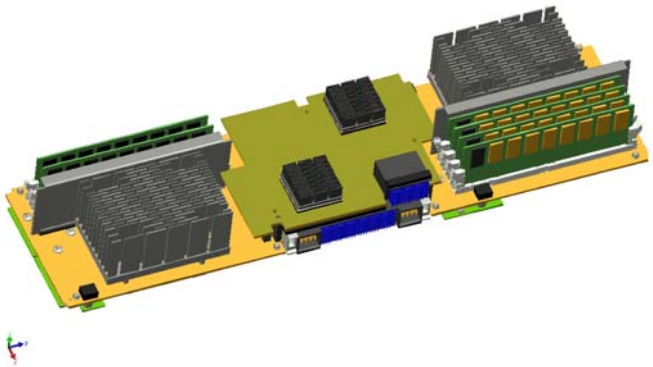
The Opportunity

HPC Leader

XD1 FPGA systems

Tools for Programming FPGA

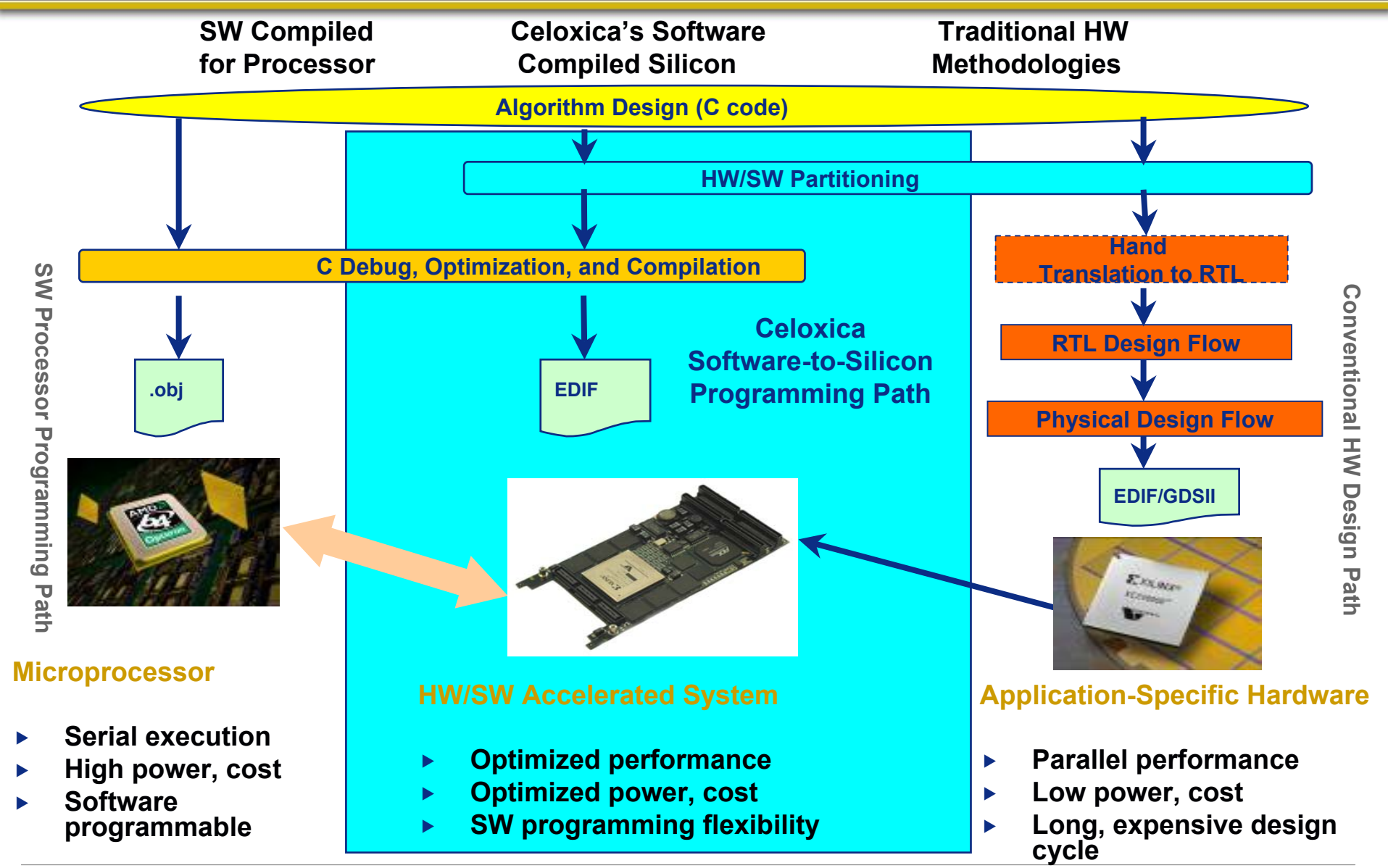
from Software



- **Make FPGA programming transparent for High-Performance Computing users**
- **Enable FPGA use to enhance compute system performance and flexibility**
- **Open new applications and new market opportunities for Cray HPC solutions**

Algorithm Implementation in FPGAs

Celoxica
Turning Software into Silicon



Programming the Cray XD1 FPGA with DK

Celoxica
Turning Software into Silicon

Algorithmic C-Code

Hardware-Software Partitioning

C-Debug, Optimization and Compilation

Software Compiler

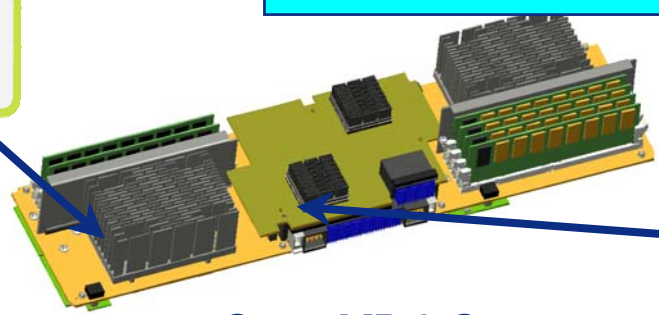
DK design suite

Celoxica's Software-to-Silicon Programming Path

Direct synthesis from algorithmic C-code to XD1 FPGA

Enables software engineers to program FPGAs without HDLs

Area and performance optimization by exploiting parallelism in FPGA architecture



Cray XD1 Compute Blade

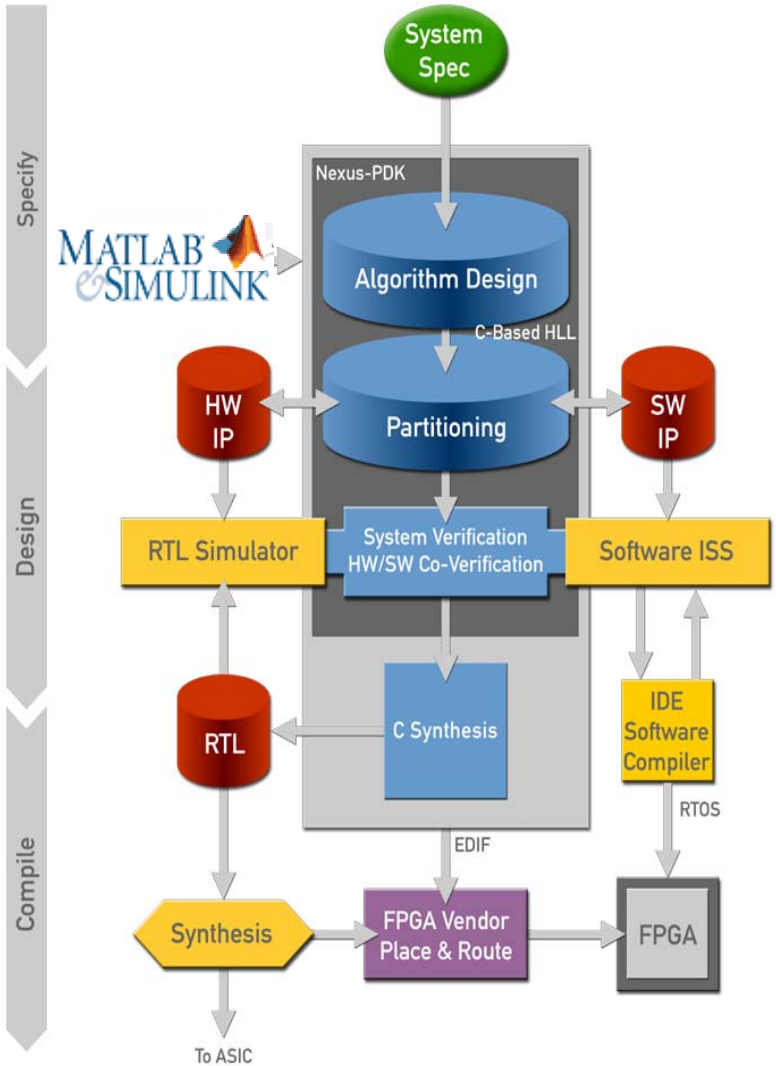
Acceleration FPGA

VIRTEX-II PRO

Designing FPGA Hardware from C using the DK Design Suite

Algorithm design

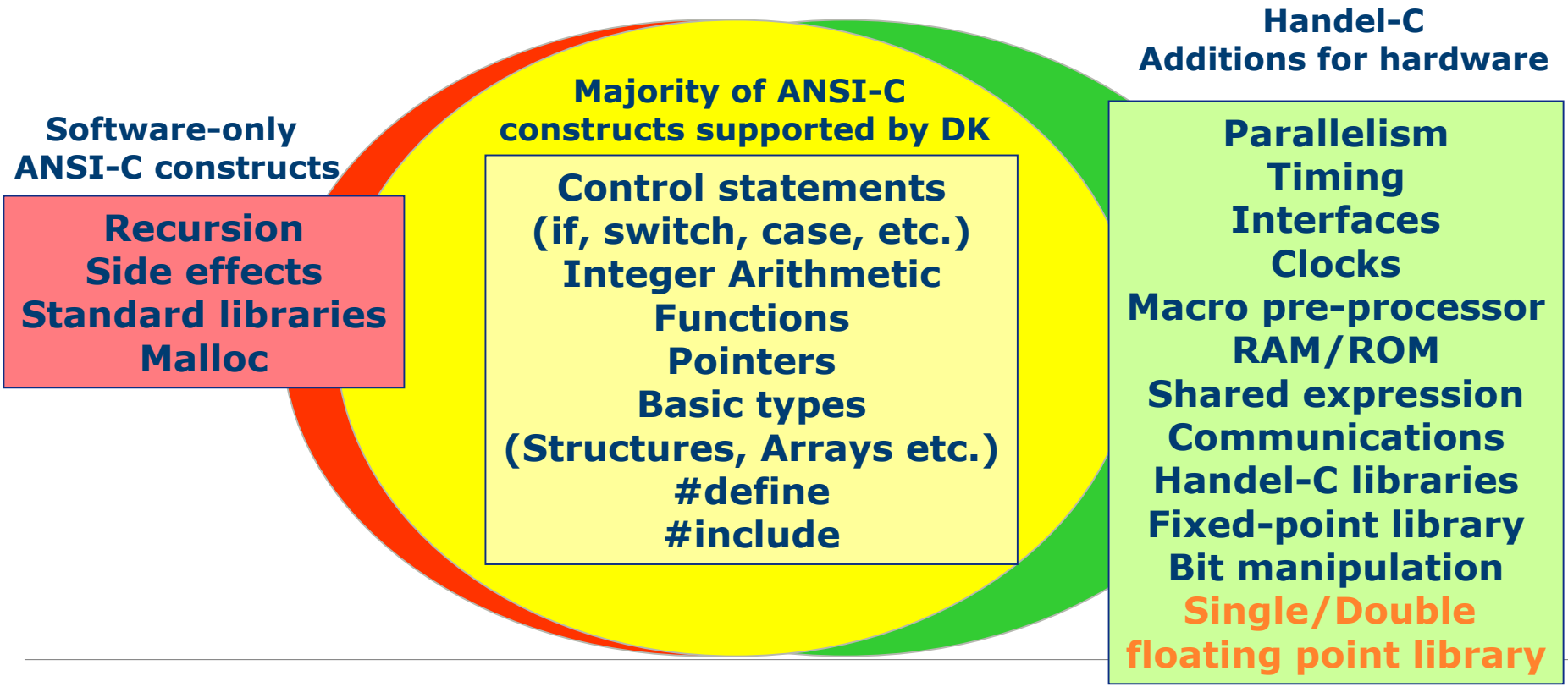
- **Co-design**
  Provide rapid iteration of partitioning decisions throughout flow
- **Co-verification**
  Drive continuous system verification from concept to hardware



C-Synthesis

- **C to RTL**
  Generate human-readable VHDL and Verilog for ASIC RTL hand-off
- **C to FPGA**
  Direct implementation to device optimized programmable logic

- **Handel-C adds constructs to ANSI-C to enable DK to directly implement hardware**
  - fully synthesizable and based upon standard ANSI-C
  - Implements C algorithm direct to optimized FPGA or outputs RTL from C



Fundamental Challenge for C-based Synthesis

Behavioral Synthesis **is:**

the *automated transformation* from a *higher level of abstraction*
(C/SystemC-based Algorithm or TLM implementation)
to a hardware implementation
(RTL description or gate netlist)

To do this ALL Behavioral Synthesis tools using ANY C/C++-based language must address the following:

- **Concurrency**
- **Timing**
- **Data Types**
- **Communication**
- **Resource Sharing and Implementation**

Celoxica Philosophy: Put control in the user's hands!

- Course and fine grain parallelism in Handel-C

```
void main(void)
{
      par{
            processA(…);
            processB(…);
            …
      }
}
```

```
// 1 Clock Cycle
  par{
            a=1;
            b=2;
            c=3;
            }
```

```
// 3 Clock Cycles

            a=1;
            b=2;
            c=3;
```

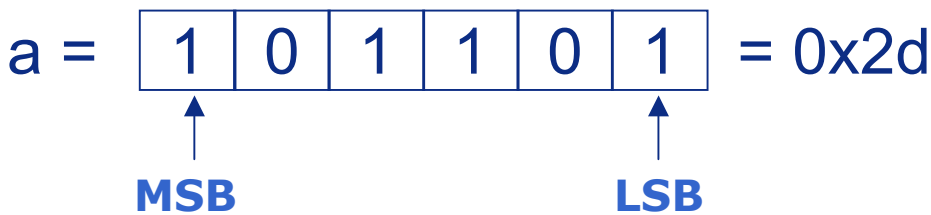
- Assignments and delay statements take 1 clock cycle
- Combinatorial expressions computed between clock edges
  - Most complex expression determines clock period
  - Example: takes 1+n cycles (n is number of iterations)

```
index = 0;                    // 1 Cycle
while (index < length){
        if(table[index] = key)
                found=index;     // 1 Cycle
        else
                index = index+1; // 1 Cycle
        }
}
```

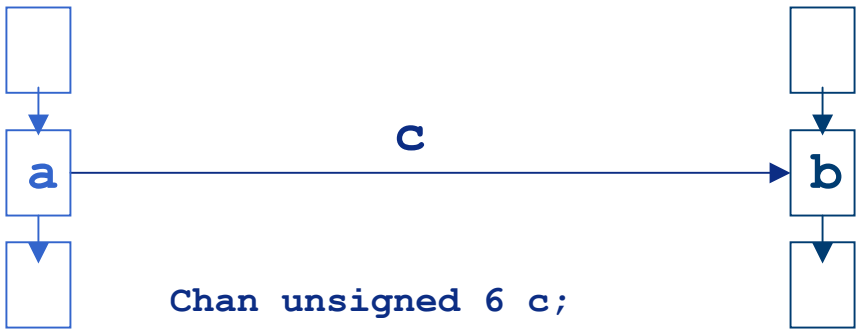
- Handel-C has one basic type - integer
- May be `signed` or `unsigned`
- Can be any width, not limited to 8, 16, 32 etc.
- Fixed-point data types also supported for fractional representation

Variables are mapped to **hardware registers**.

```
void main(void)
{
        unsigned 6 a;
        a=45;
}
```



- **Allow communication and synchronization between two parallel branches**
  - **Semantics based on CSP: unbuffered (synchronous) send and receive**
- **Declaration**
  - **Specifies data type to be communicated**




```
{
    …
    c!a+1;   //write a+1 to c
    …
}
```

```
{
    …
    c?b;   //read c to b
    …
}
```

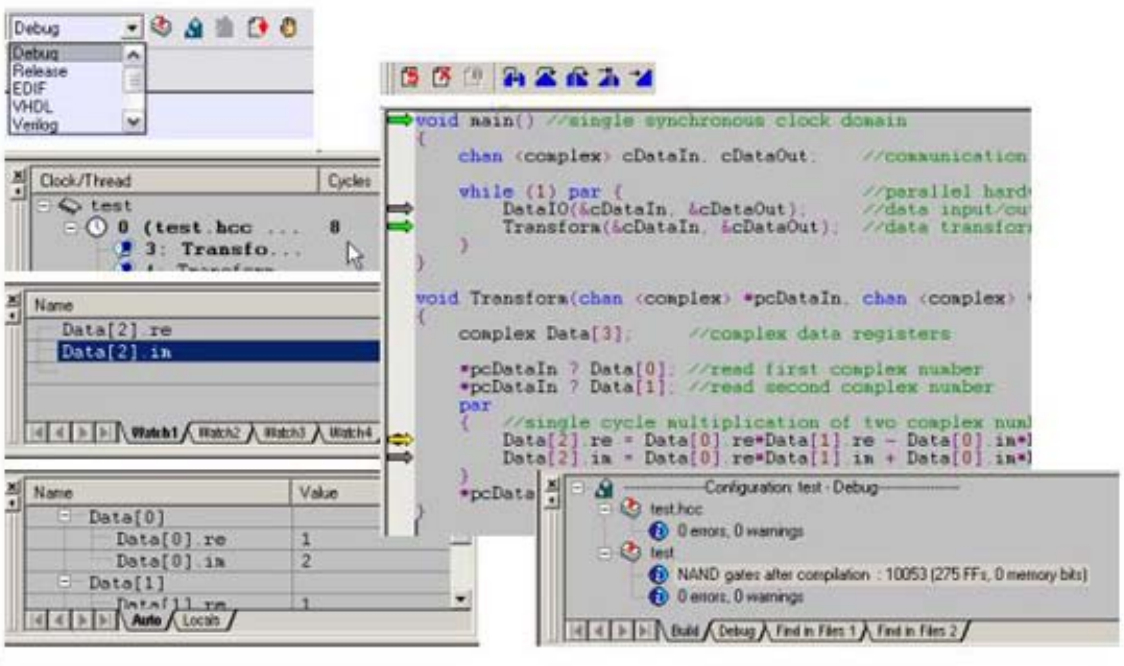
- Interfaces allow Handel-C designs to connect to external hardware and logic.
- Three types of interfaces
  - Buses – used for connecting to external pins
  - Ports – used for creating connection points for external logic.
    - e.g. Creating the ports for a VHDL entity
  - User Defined – used for including external logic blocks inside a Handel-C design.
    - e.g. Including an EDIF black box inside a design.

DK Design Suite IDE

Mix C/C++ & HC (for the description of parallel algorithms)

C/C++ code used in simulation, functional test benches & HW-SW co-design

HC simulated cycle accurately and implemented as EDIF & RTL

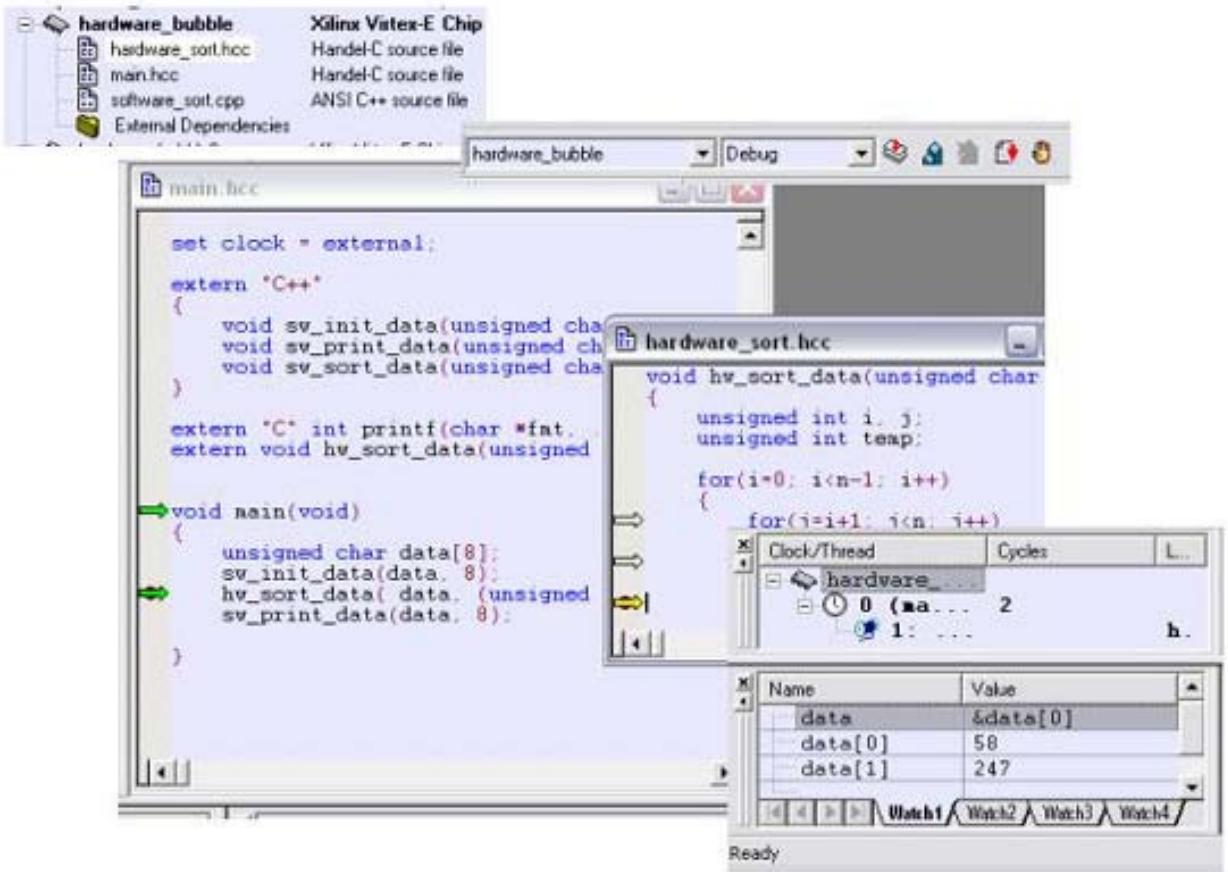


- **Project management**
- **Symbolic source browsers**
- **Syntax highlighting**
- **Cycle-accurate multithreaded symbolic debugger:**
  - **single step execution**
  - **break points;**
  - **variable watch windows**
  - **thread focus.**
- **Mix C/C++ & HC code**

Mixed C/ C++/ HC Simulation

Celoxica
Turning Software into Silicon

- Supports TLM, mixed abstraction modeling & simulation
- HC, C & C++ native support

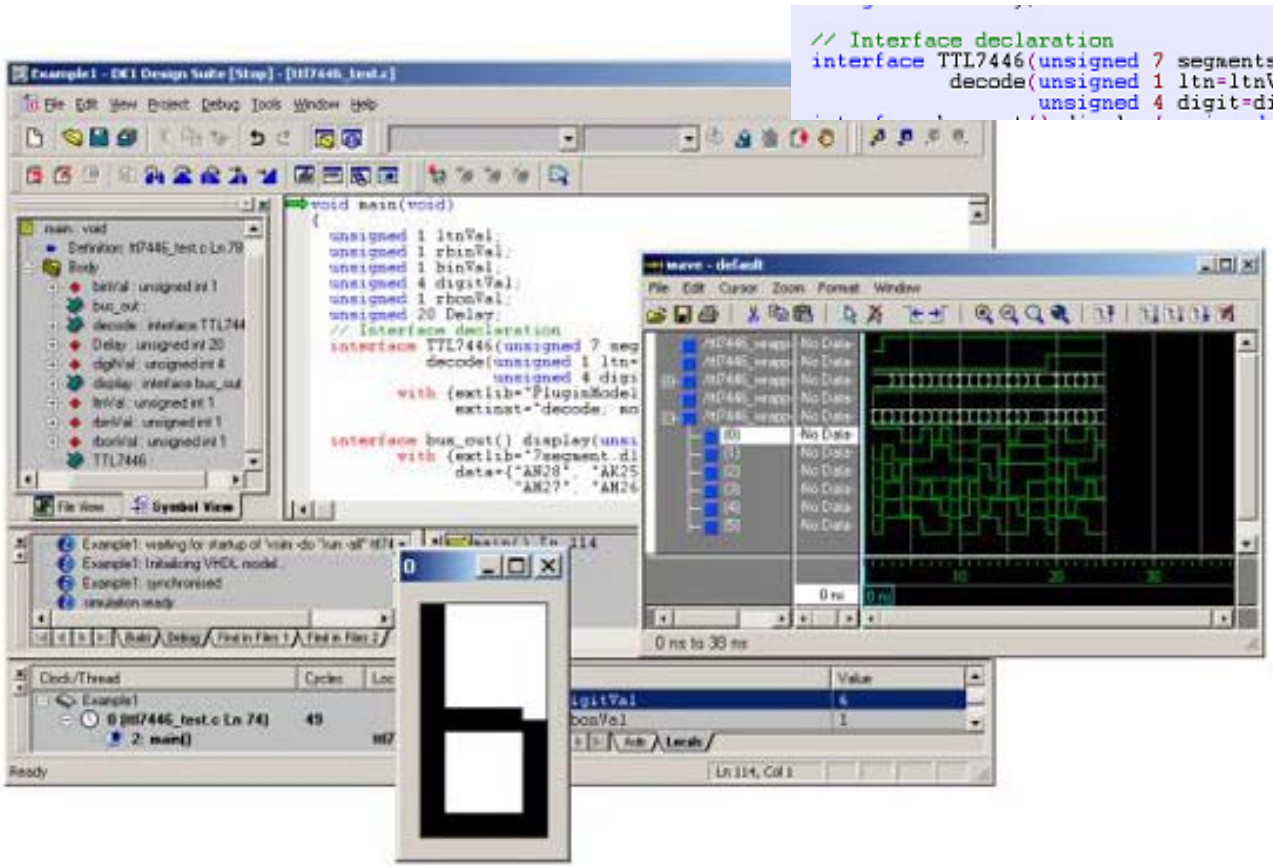


- **Source-level parallel debug**
- **Single stepping & break points**
- **HW simulated within the system model**
- **Fast simulation**
- **Functional verification with mixed languages**
- **Architectural exploration**
- **Refine & transform to HW**
- **C/C++ testbenches developed into models & used throughout code**

Mixed Language Simulation

Celoxica
Turning Software into Silicon

- Connects with 3rd party simulators
- VHDL/ Verilog/ SystemC/ Simulink/ Software ISS models

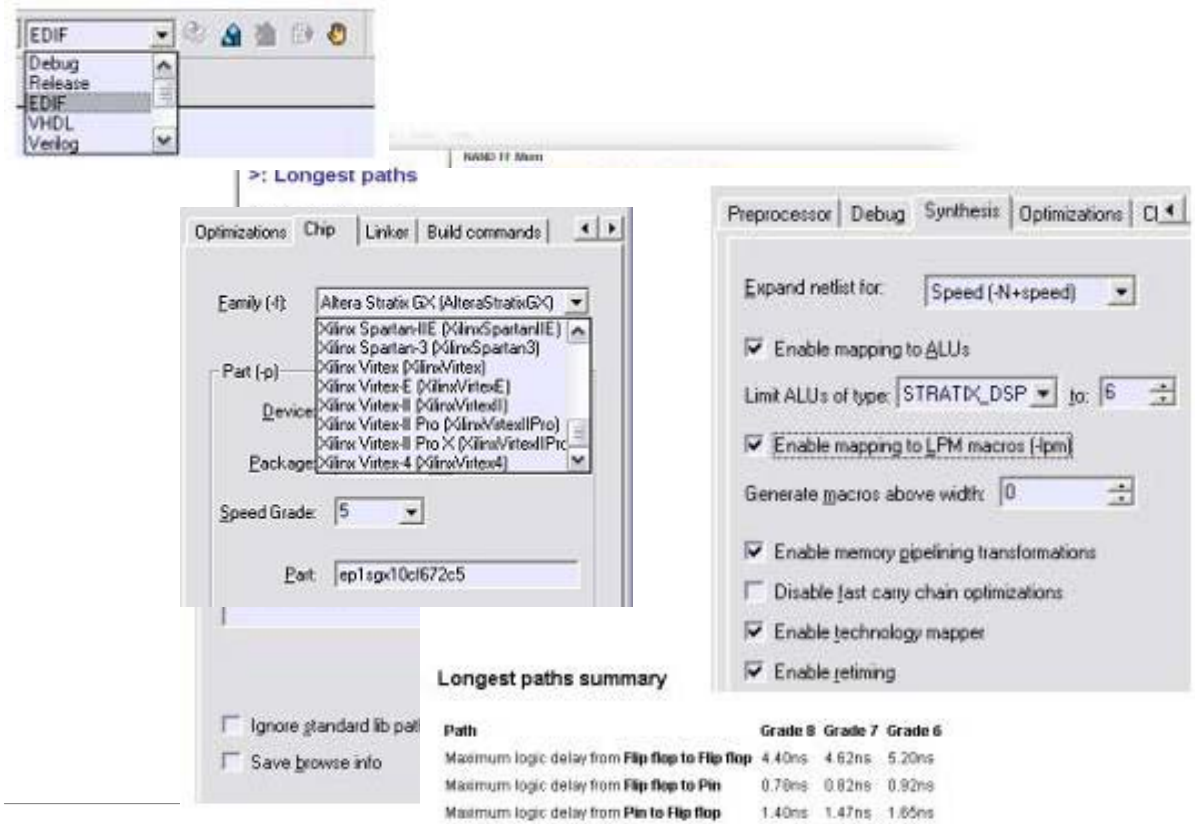


- IP reuse
- Static timing analysis
- Libraries for HW-SW simulation, system modeling
- Libraries for virtual peripherals & platform abstraction

C Synthesis to FPGA Gates & RTL

Celoxica
Turning Software into Silicon

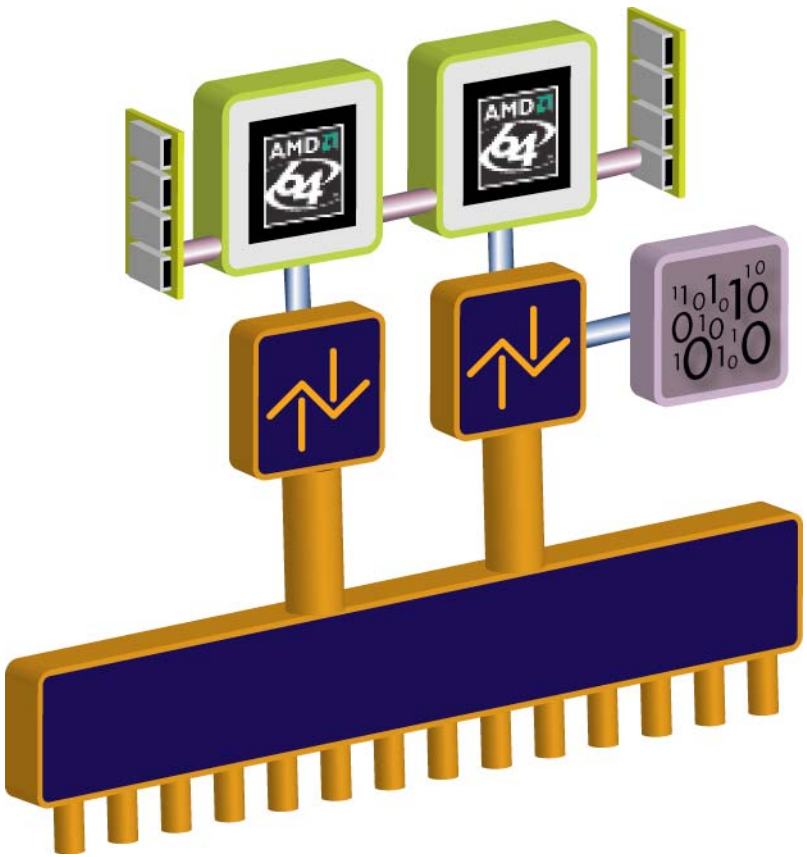
- Auto Generation of EDIF netlist and IEEE RTL
- Static timing related directly to areas of source code for optimization
- Detailed time/area estimations supports experimentation
- Output nets names relate directly to source code.



- Set timing constraints for I/O signals
- Technology mapped (to LUTs) EDIF output
- Use of embedded ALU primitives (e.g. for Stratix DSP blocks & Virtex-II/IV multipliers)
- Automatic pipelining of RAM accesses: (e.g. Actel BlockRAM, Altera EAB and Xilinx BlockRAM)
- Retiming Synthesis

Implementing a Search Kernel Algorithm on Cray's XD1 Using DK

XD1 Application Accelerator



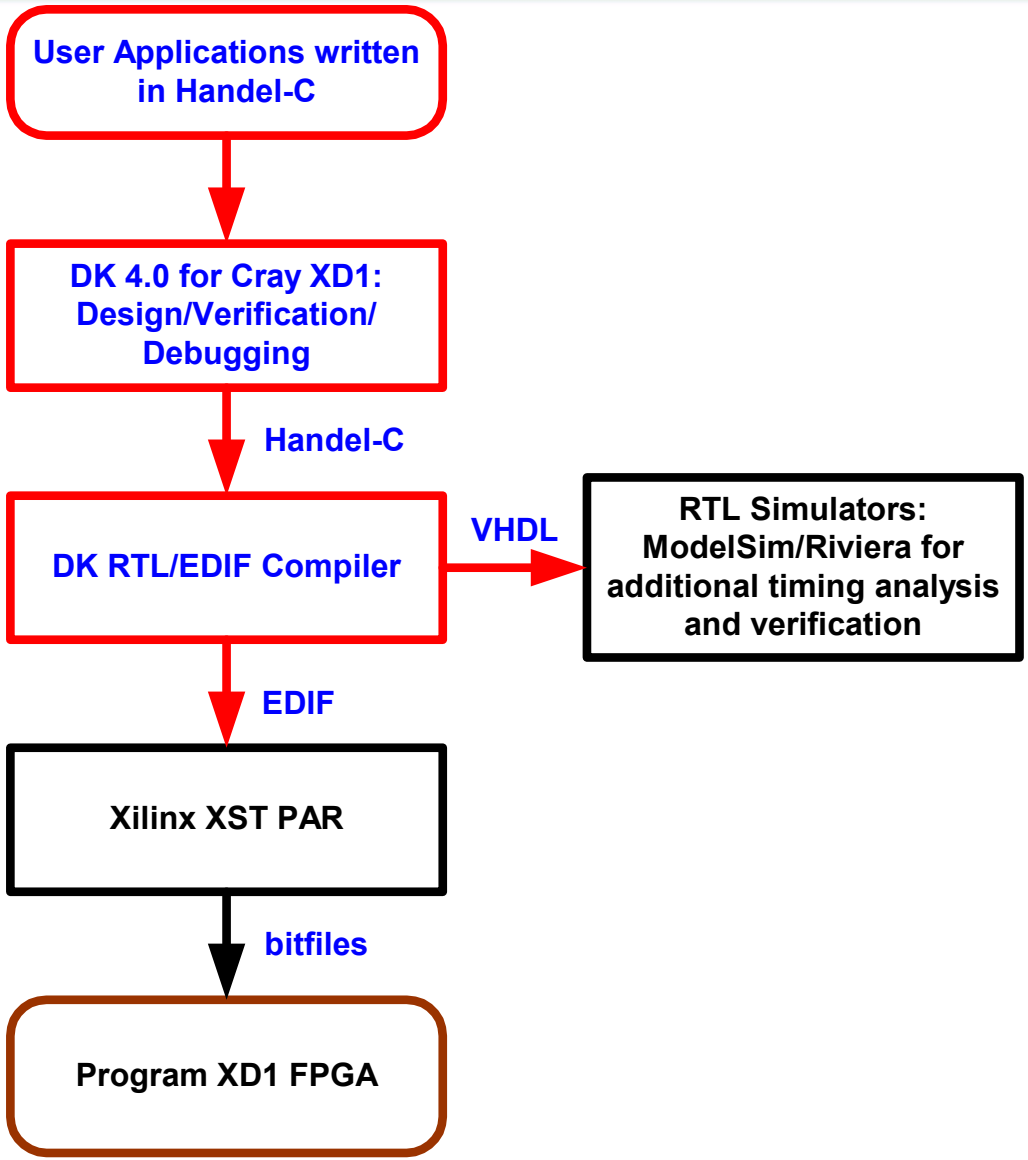
Application Acceleration

- *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.

Slide Courtesy of Cray Inc.

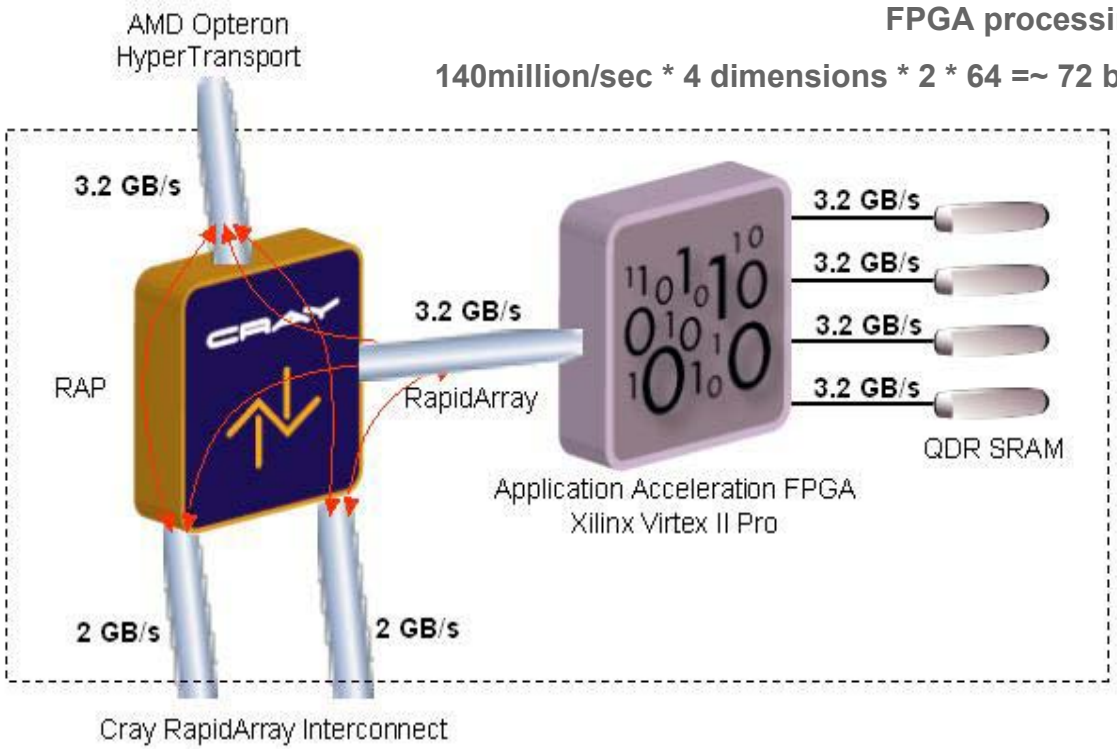
Search Kernel Algorithm Example

- **Hypercube Range Search Implementation**
  - Algorithm has a set of *p* points and a set of *h* hypercubes, both of which having *d* dimensions.
  - System identifies each point-hypercube combination where the point lies completely within the hypercube
  - All of the point and hypercube parameters are represented by 16-bit positive integers
  - Result of the search kernel should be represented as bit-vectors where each bit represents a hit/miss flag for every point-hypercube combination
  - Resulting bit-vectors must be written back into the processor's main memory
  - Parameters
    - Number of points will be limited to around 10^7
    - Number of hypercubes, h, was set at 64
    - Number of dimensions was set to 4


User Applications written in Handel-C
DK 4.0 for Cray XD1: Design/Verification/ Debugging
Handel-C
DK RTL/EDIF Compiler
VHDL
RTL Simulators: ModelSim/Riviera for additional timing analysis and verification
EDIF
Xilinx XST PAR
bitfiles
Program XD1 FPGA

XD1 Application Acceleration System

2.2 GHz Opteron



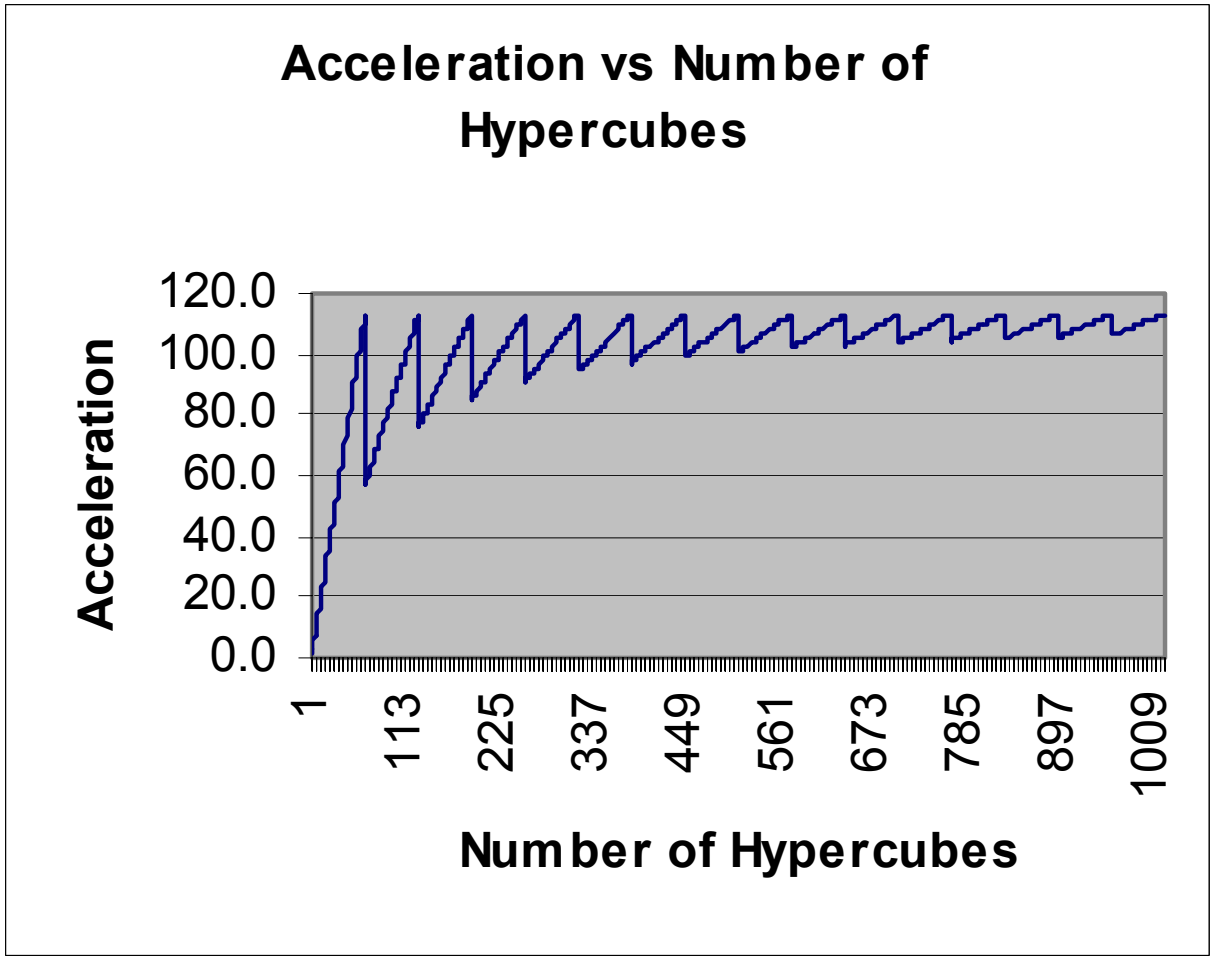
FPGA processing 64 hypercubes/cycle

140million/sec * 4 dimensions * 2 * 64 =~ 72 billion comparisons/sec

128,000 points processed 500 times

Calculation takes about 0.47 seconds

Slide Courtesy of Cray Inc.


Acceleration vs Number of Hypercubes
Acceleration
120.0
100.0
80.0
60.0
40.0
20.0
0.0
1
113
225
337
449
561
673
785
897
1009
Number of Hypercubes