ACM/SIGDA FPGA 2008

2nd Annual

Pre-Conference Workshop

Designing with Extreme Parallelism

Sunday, Feb 24, 2:00 pm – 5:30pm

Pinos-Alones Room (4th floor)

Workshop Goals

- Welcome ideas on CAD, architecture, and novel mainstream uses of FPGAs
- Plan the course for future FPGA-related research in general
- Create a vision and inspire young researchers with technical ideas, funding avenues, and relevance

Last Year's Workshop

- Grand Challenges on FPGA Research
- Key findings
  - Jason Cong: Optimality gap in CAD?
  - Kurt Keutzer: Applications-to-hardware bridge
  - Grant Martin: Stop focusing on FPGAs!
  - Vaughn Betz: Compile/design time, power, I/O
  - Steve Trimberger: Timelines, dog food, binary compatible
  - Jonathan Rose: ASIC/FPGA gap, more theory
  - André DeHon: Defects, variation, transients

Designing with Extreme Parallelism

- FPGAs hold millions of gates
  - Not just your grand-daddy's glue logic anymore…
- FPGAs build large, digital systems
  - Lots of inherent parallelism (it's hardware!)
- Time-to-system is key advantage of FPGAs
  - Compile time?
  - Design time?
  - Synthesis from high level programming tools?

Designing with Extreme Parallelism

- Custom compute engines
- Orders of magnitude
  - Speedup
  - Lower power

- Parallel language tools?
  - For ASICs
  - For FPGAs
  - For other parallel systems?

Designing with Extreme Parallelism

- **Designing Parallel Systems with High-level Languages**
  - **Overview of Parallel Landscape**
  - **Catapult C (Mentor Graphics)**
  - **Mitrion C (Mitrionics)**
  - **Reconfigurable Toolbox (DSPlogic)**
  - **CUDA (NVIDIA)**

Format and Time Table

Session 1 (2:00pm)

- Prof. Tarek El-Ghazawi, *GWU*
- Dr. Andres Takach, *Mentor Graphics*
- Dr. Michael Babst, *DSPlogic*

Break (3:30pm)

Session 2 (4:00pm)

- Mr. Stefan Möhl, *Mitrionics*
- Dr. David Kirk, *NVIDIA*
- Open Discussion

End (~5:30pm)

FPGA 2008 Registration (6:00pm)
FPGA 2008 Reception (7:00pm)

Prof. Tarek El-Ghazawi

- **Dept of ECE, The George Washington University**
- **Co-Director of NSF CHREC**
- **Director of Institute of Massively Parallel Applications and Computing Technologies (IMPACT), GWU**

- **Major developer of Unified Parallel C (UPC)**
  - **Author of UPC book**
- **Assoc. Editor IEEE Trans. on Computers**

Dr. Andres Takach

- **Chief Scientist, Mentor Graphics**
- **C-based hardware synthesis**
- **Faculty member at Illinois Institute of Technology 1993 – 1997**
- **PhD from Princeton in 1993**

Michael Babst

- **President and Founder of DSPlogic, Inc**
- **High-Performance Reconfigurable Computing and FPGA-based Digital Signal Processing**
- **Founding member and Vice President of Eka Systems, a wireless networking startup**
- **Over 16 years of product development and management experience at General Electric Aerospace, Lockheed Martin, Comsat Laboratories, and Viasat**
- **Ph.D. from the Pennsylvania State University**

Break

Format and Time Table

Session 1 (2:00pm)

- Prof. Tarek El-Ghazawi, *GWU*
- Dr. Andres Takach, *Mentor Graphics*
- Dr. Michael Babst, *DSPlogic*

Break (3:45pm)

Session 2 (4:15pm)

- Mr. Stefan Möhl, *Mitrionics*
- Dr. David Kirk, *NVIDIA*
- Open Discussion

End (~5:45pm)

FPGA 2008 Registration (6:00pm)
FPGA 2008 Reception (7:00pm)

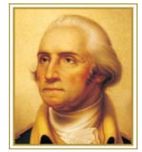
Stefan Möhl

- **Co-founder of Mitrionics AB in 2001**
  - **Vice President**
  - **Chief Scientific Officer**
- **Researches processor architecture and language design in the context of parallel processing**
- **Studied Computer Science, Philosophy, Linguistics and Psychology**
- **Founder of e-commerce startup Shopsense in 1999**

David Kirk

- Chief Scientist at NVIDIA since 1997
- Leads NVIDIA's graphics technology development
- Elected to the National Academy of Engineering (NAE) – 2006
- SIGGRAPH Computer Graphics Achievement Award – 2002
- Chief Scientist, Head of Technology for Crystal Dynamics, 1993 to 1996
- Engineer at Apollo Systems Division (HP), 1989 to 1991

Open Discussion



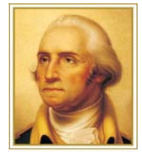
Designing with Extreme Parallelism: The Programming Issues

Tarek El-Ghazawi

The George Washington University

Co-Director, NSF Center for High-Performance Reconfigurable Computing
Director, Institute for Massively Parallel Applications and Computing Technologies (IMPACT)

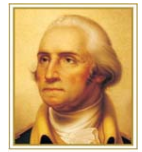
Outline



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

- Intuitional questions
- Classifications

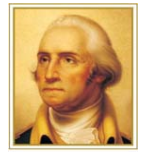
Designing with Parallelism? What options do you have?



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

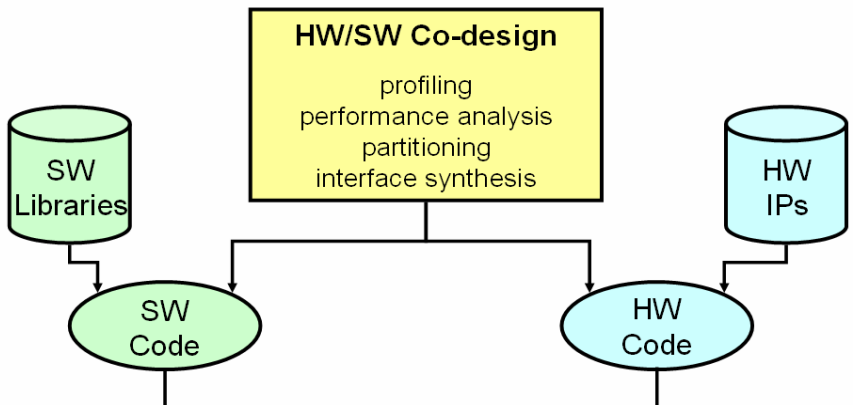
- Leave it all to a parallelizing compiler
  - Parallel computing community gave up!
- Take charge
  - Explicitly or Implicitly
    - Express Parallelism
    - Express Data exchanges

Designing with Accelerators? How integrative is the flow?

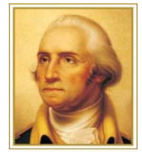


THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

- Tool facilitates design for accelerator side only. Programmer creates a separate microprocessor solution and integrates the pieces.
- Tool supports co-design, e.g. partitioning and co-scheduling. See example from FPGAs.

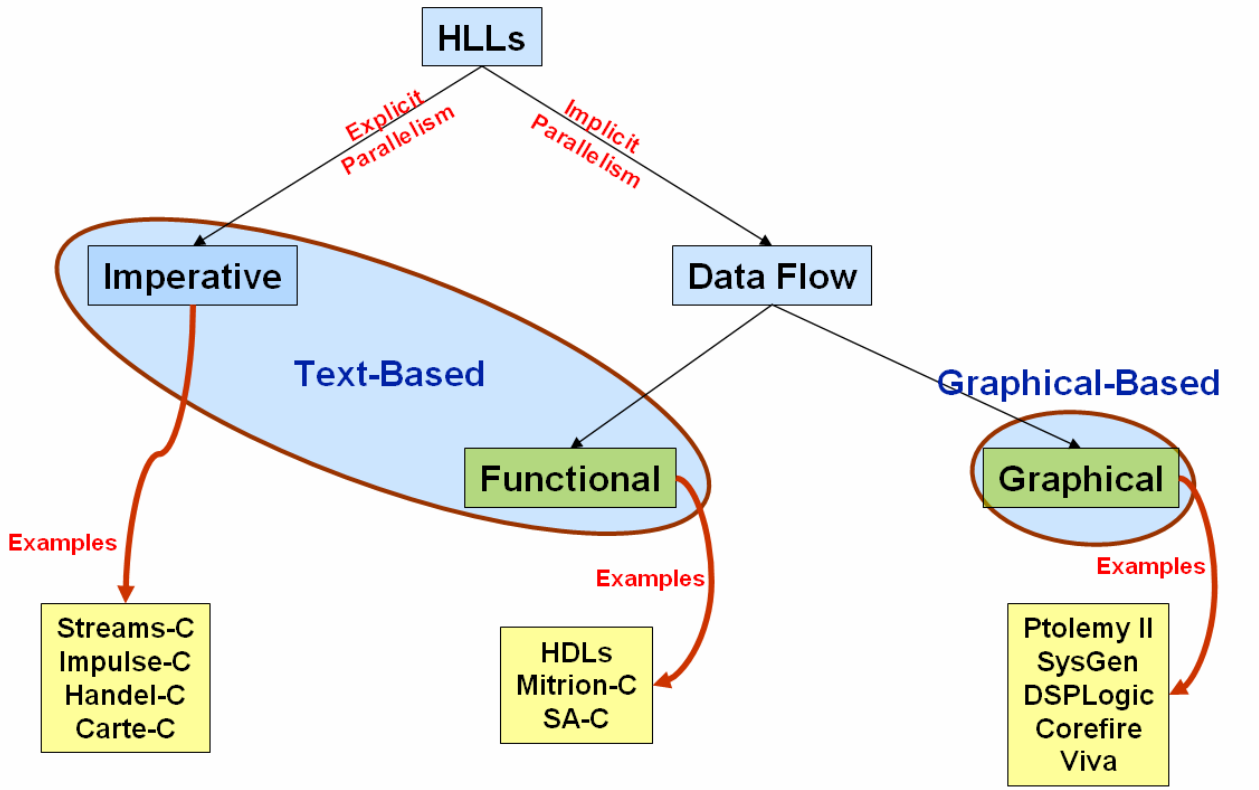


A Taxonomy for Methods of Expressing Parallelism

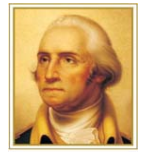


THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

HLL Classification



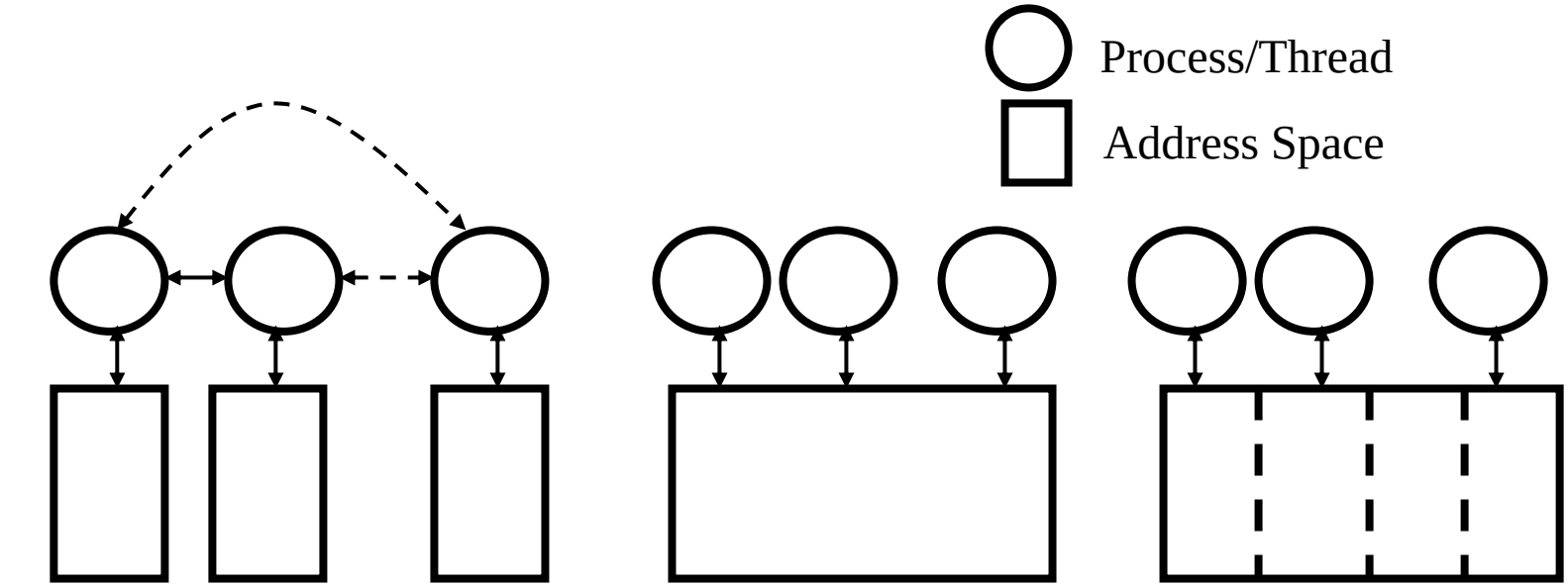
Basic methods for expressing data exchanges



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

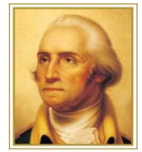
- Two basic models, derivatives exist
- Parallel activities do not see each others memory, data is exchanged via messages
- Parallel activities share the same memory view, data is exchanged via reads and writes into the shared space

Exchanging Data



MPI **OpenMP** **UPC**

Questionnaire



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

CHREC
NSF Center for High-Performance Reconfigurable Computing

Catapult C® Synthesis

Creating Parallel Hardware from C++

Andres Takach

Chief Scientist, C-Based Design

February 2008

Mentor Graphics®

Agenda

- Implementing bit-accurate data types in C++
- Implementing parallelism from sequential C++
  - A simple FIR filter
  - Saturation and rounding implications
- Creating pipelined hierarchical systems
  - Discrete Cosine Transform
    - 1 pixel per clock cycle throughput

Bit Accurate Data Types

- Hardware Designers need exact bit widths
  - Extra bits costs gates ($$) and performance ($$)
- C++ data types are insufficient for modeling and have ambiguities
- DSP designers use rounding and saturation
  - Hardware engineers generally do not, unless pressed
- SystemC data types have ambiguities and limitations for algorithm modeling
- Mentor Graphics Algorithmic C data types provide superior vehicle for bit accurate hardware design

Mentor Graphics "Algorithmic C" types

- Fixed-point and Integer types
- Faster simulation
  - Up to 200x faster than SystemC types
- Consistent, with no ambiguities
- Parameterized
  - Facilitate reusable algorithmic development
- Built in Rounding and Saturation modes
- Usable within a SystemC environment
- Freely available for anyone to download

http://www.mentor.com/products/c-based_design

Templatized AC Fixed Data Types

```
ac_fixed<W,I,S,Q,O> my_variable
```

- `W` = Overall Width
- `I` = Number of integer bits
- `S` = signed or unsigned (boolean)
- `Q` = Quantization mode
- `O` = Overflow mode

```
ac_fixed<8,1,true,AC_RND,AC_SAT> my_variable ;
    "0.0000000" 8-bit signed, round & saturate
ac_fixed<8,8,true,AC_TRN,AC_WRAP> my_variable ;
    "00000000" 8-bit signed, no fractional bits.
```

Using C++ For Parallel Hardware

- Function call with all I/O on the interface
  - Represents the I/O of the hardware to be built
- C++ allows object-oriented reusable hardware
  - Technology and Frequency independent
  - Multiple instantiations of functions with state
    - Just like RTL component instantiation
  - Instantiations with different implementations
    - Like VHDL architectures
  - Enables resource sharing across time and function
    - Unlike RTL

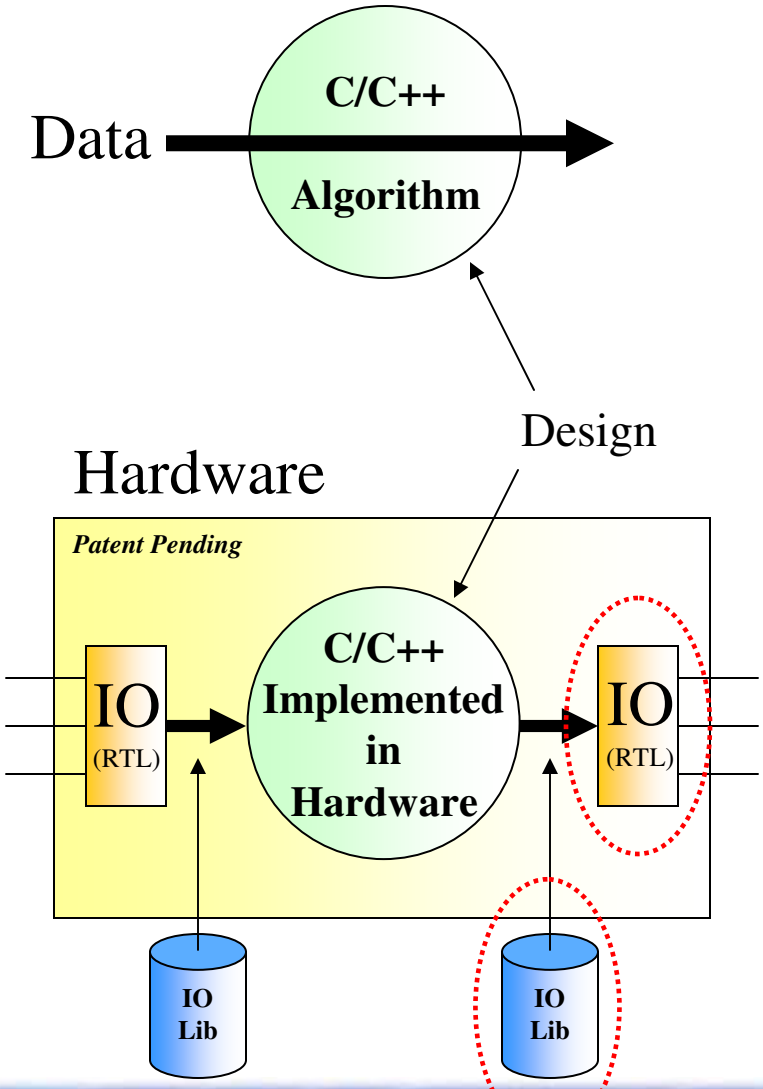
A Simple FIR Filter Model

```
void fir_filter (
     IN_TYPE       &input,
     COEFF_TYPE    coeffs[NUM_TAPS],
     OUT_TYPE      &output
) {

     static IN_TYPE regs[NUM_TAPS];
     MAC_TYPE temp = 0.0;

     SHIFT:for (int i=NUM_TAPS-1 ; i>=0; i--) {
          if (i==0) regs[i] = input ;
          else regs[i] = regs[i-1] ;
     }

     MAC:for (int j = 0 ; j < NUM_TAPS ; j++ )
          temp += regs[j] * coeffs[j] ;

     output = temp ;
}
```

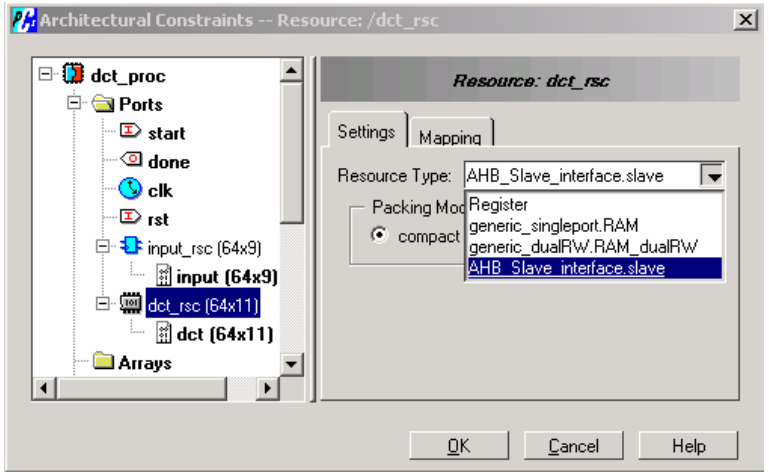
- **Input, coefficients and output**
- **Static taps**
- **MAC type for full precision**
- **Output type for optional rounding and saturation**

Defining The Hardware Interface

Patented Interface synthesis technology makes it possible



- Untimed C++ has no concept of time



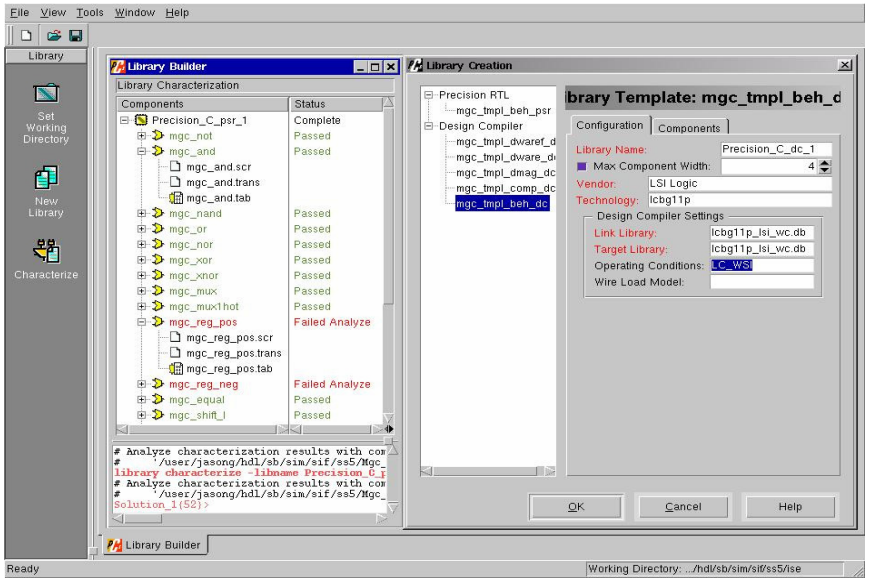
- How does this help?
  - ANY interface is possible
  - Design is built to the interface
  - C++ source remains independent of the interface

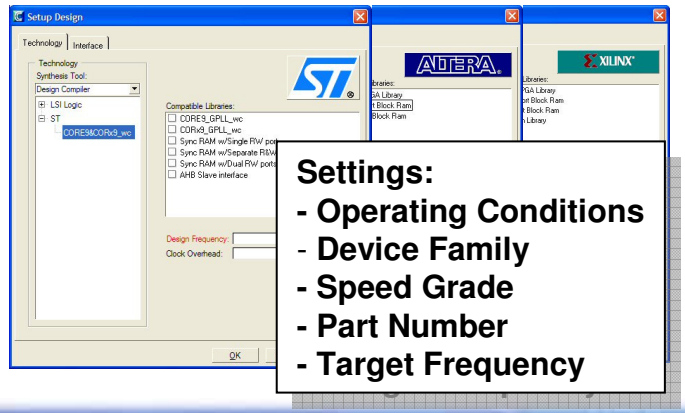
Optimizing Untimed C++

- Catapult maps user-selected physical resources for each variable in the C++ code
  - Wires, handshakes, registers, RAM's, custom interfaces, custom components
- Catapult builds efficient hardware optimized to the constraints of resource bandwidth
- Catapult enables you to quickly find architectural bottlenecks in an algorithm
- Datapath pipelines are created automatically in order to meet desired operating frequency

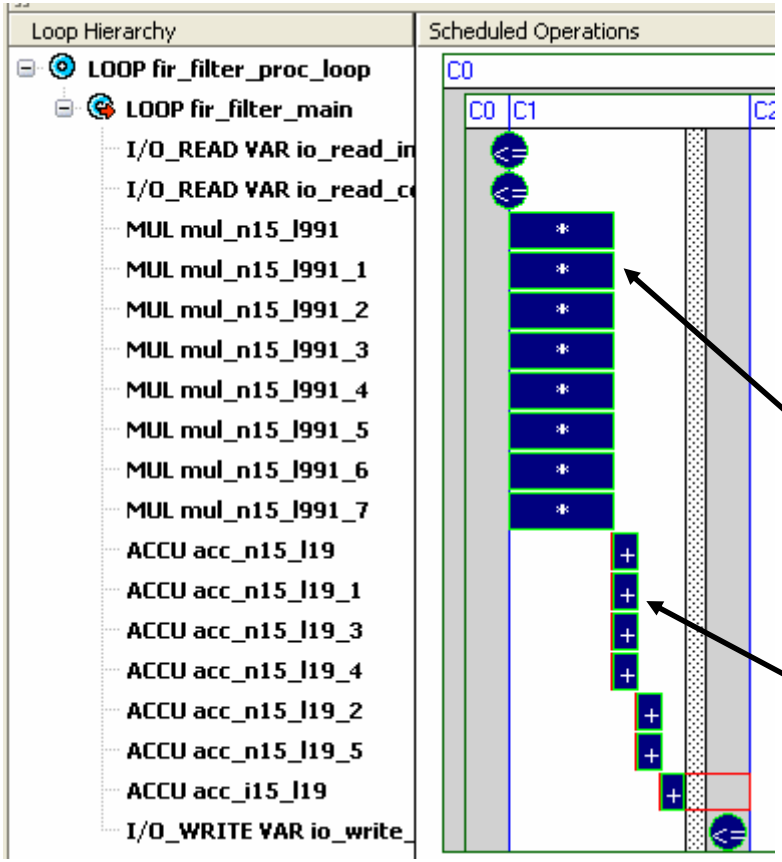
Technology Driven Synthesis

- The exact ASIC or FPGA technology is “characterized” for accurate timing and area metrics for operators of differing bit widths
- Algorithms are scheduled using these technology specific libraries
  - Like having a synthesis timing guru creating your RTL
- Key for high quality, technology-optimized designs with specific operating frequency requirements





FIR Filter Unrolled For Parallelism



- 8 multipliers and an adder tree
- Optimization for latency results in fast components at the chosen operating frequency (250 MHz, 180nm)
- ASIC synthesis typically offers 4 operator area/speed possibilities

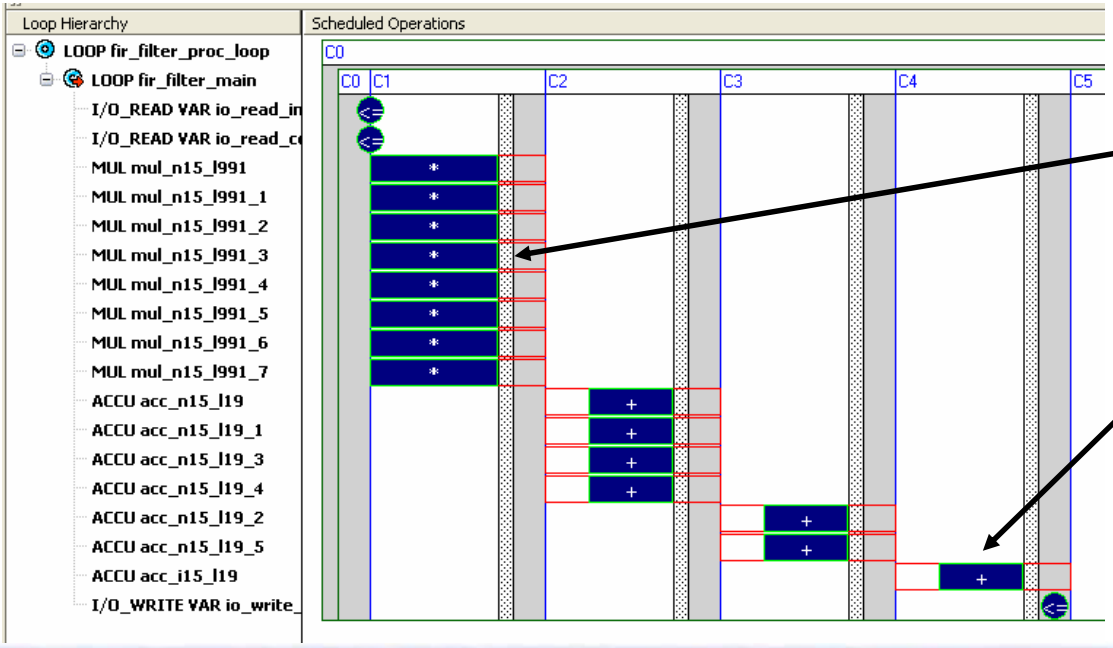
Fastest multiplier in technology

Fastest adders in technology

103,000 sq. microns

Optimizing For Throughput

- Same C++ code can be scheduled to use smaller multipliers
  - Smaller adders too
- Pipelining still keeps throughput data rate
- 40% saving in area after synthesis of completely different RTL
- Characterized target library leverages technology timing



Slower multiplier in technology

Slower adders in technology

63,000 sq. microns

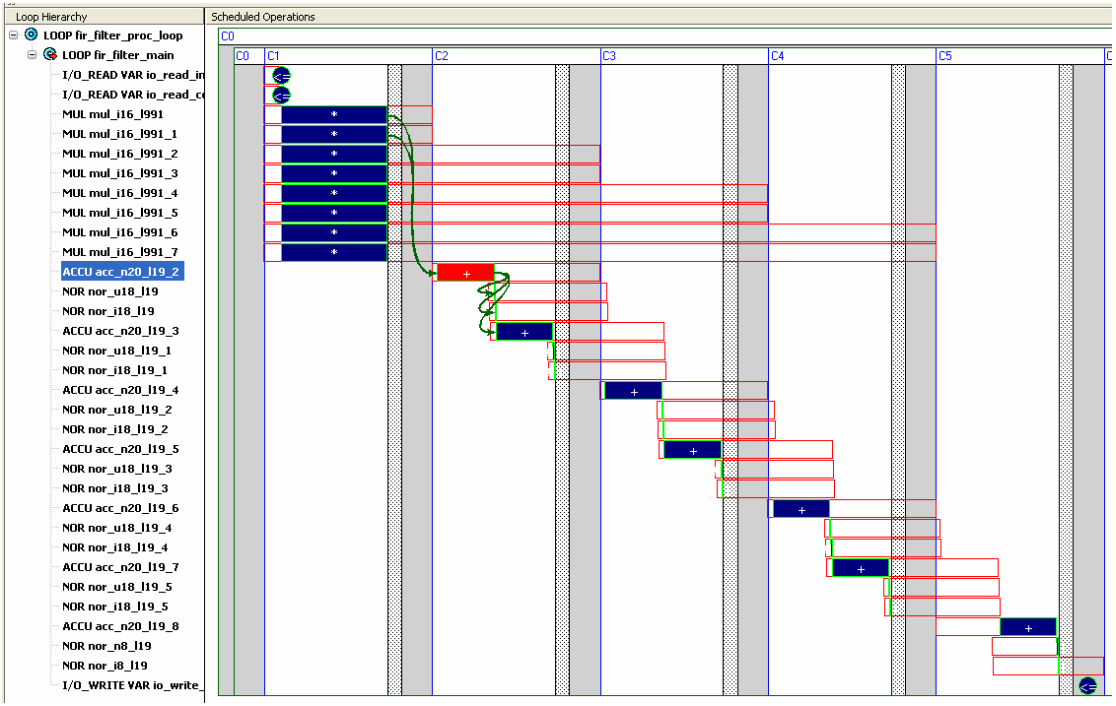
Partial Unrolling

- Allows “n” copies of the loop body to be created in parallel
- 1, 2, 3, 4 copies will give different throughputs
  - 1 => 8 cycles
  - 2 => 4 cycles
  - 3 => 3 cycles (3,3,2)
  - 4 => 2 cycles
- This assumes that all I/O has enough bandwidth
  - With FIR filters, the coefficients (if programmable) and tap storage (registers or RAM’s) are key

The Trouble With Saturation

- Saturation is order-dependent and undesirable when creating parallelism
  - e.g. with an 8-bit signed integer storage (-128 to 127)
    - (100 + 50 -50) is not the same as (100 -50 +50)
  - Creates dependency chains
- Rounding adds carry-in further downstream
- Preferable to do arithmetic at higher precision
  - And then round and truncate at the end

Rounding And Saturating Accumulator



- Creates long string of arithmetic as it must be done in the same sequential order as the C++ to guarantee algorithm match
- Larger area & lower performance than using full precision and rounding at the end

8x8 Discrete Cosine Transform

- **Simple two-dimensional "numerical recipe" implementation**
  - **Rows, then columns, with intermediate storage array**

```
#include "constants.h"
void dct_float(double input[8][8], double dct[8][8]) {

  double temp[8][8];
  double tmp;

  mult1:for (int y=0; y < 8; y++ )
    middle1:for (int x=0; x < 8; x++ ) {
      tmp = 0;
      inner1:for (int i=0; i < 8; i++ )
        tmp = tmp + input[y][i] * coeff[x][i];
      temp[y][x] = tmp;
    }

  mult2:for (int x=0 ; x < 8; x++ )
    middle2:for (int y=0; y < 8; y++ ) {
      tmp = 0;
      inner2:for (int i=0 ; i < 8 ; i++ )
        tmp = tmp + temp[i][x] * coeff[y][i];
      dct[y][x] = tmp/32;
    }
}
```

Multiply-accumulate 512 times each = 1024 Multiplies

One nested loop set for rows

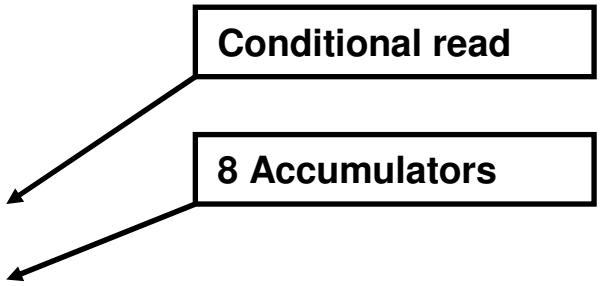
Second nested loop set for columns – outputs data by columns sequentially

Hardware Design with C++

- Algorithmic Synthesis maintains memory architecture
  - Shift register or…
  - Circular buffer
- Just unrolling sequential algorithms may not yield optimum parallel hardware architecture
- C++ code must reflect memory accesses required for desired hardware architecture

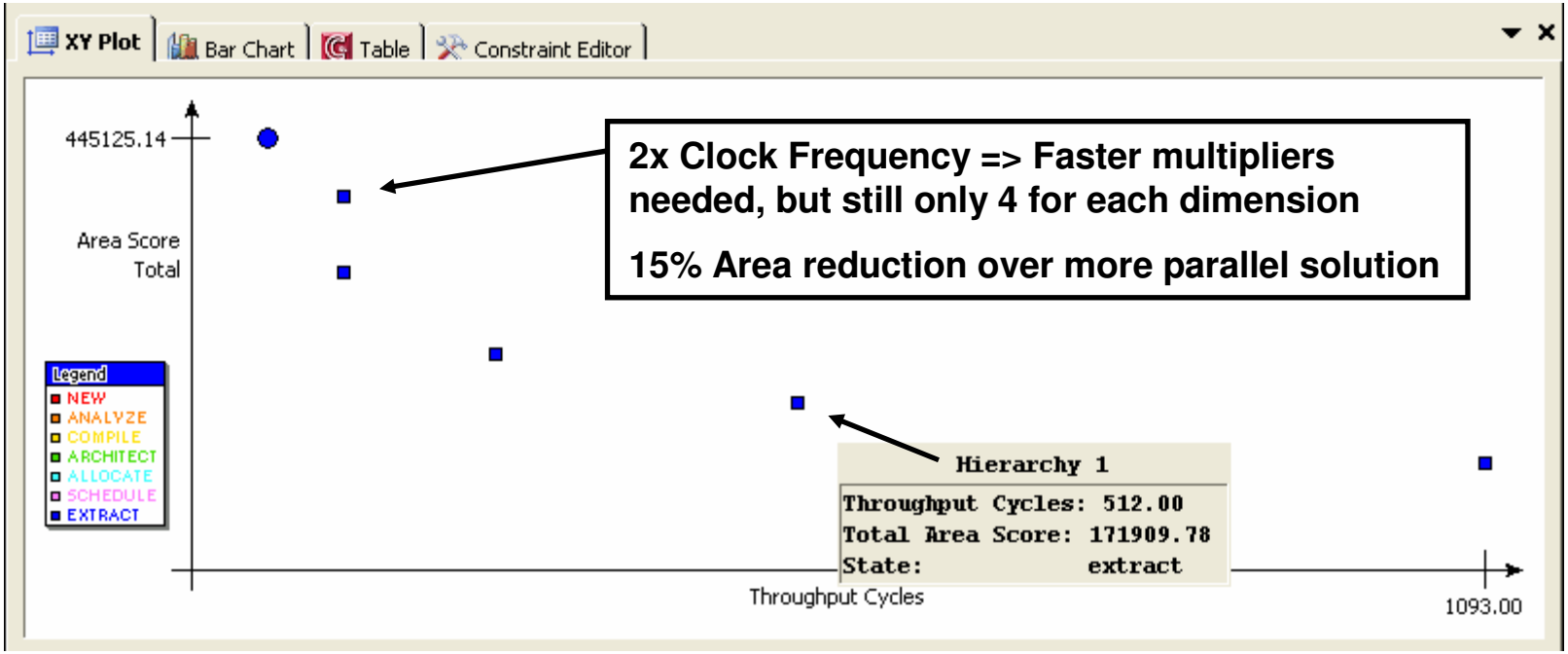
2D DCT for hardware

```
void dct_rows (
     pixel_in_t input[8][8],
     temp_t temp[8][8]
) {
     temp_t tmp[8] ;
     pixel_in_t pixel_read ;
     outer1:for (int y=0; y < 8; y++ ) {
          middle1:for (int x=0; x < 8; x++ ) {
               inner:for (int i=0; i < 8; i++ ) {
                    if (i==0) pixel_read = input[y][x] ;
                    if (x==0) tmp[i] = 0 ;
                    tmp[i] += pixel_read * coeff[i][x] ;
                    if (x==7) temp[y][i] = tmp[i] ;
               }
          }
     }
}
```



- Read inputs in linear order to allow streaming of data
  - Avoid reads of same memory location
- Parallel accumulators change the algorithm architecture

X/Y plot for 180nm at 90MHz



XY Plot | Bar Chart | **Table** | Constraint Editor

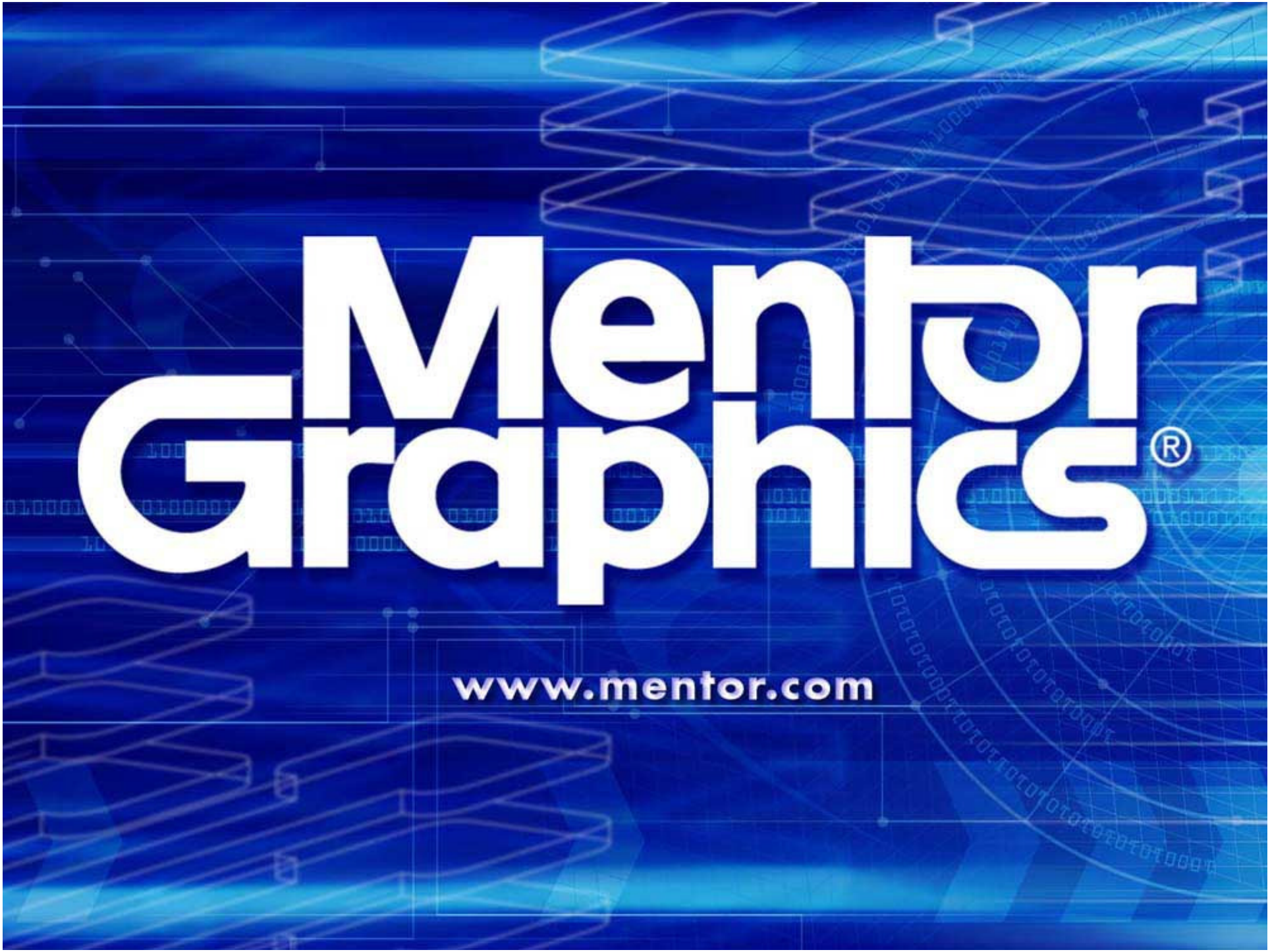
Solution	Status	Total Area Score	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time
One Multiplier	Passed extract	110221.65	1093	12144.44	1093	12144.44
Hierarchy 1	Passed extract	171909.78	1534	17044.44	512	5688.89
Hierarchy 2	Passed extract	222619.16	770	8555.56	256	2844.44
Hierarchy 4	Passed extract	386888.59	388	4311.11	128	1422.22
Hierarchy 8	Passed extract	445125.14	197	2188.89	64	711.11
Hierarchy 4 2x Clock	Passed extract	384655.19	390	2166.67	128	711.11

FPGA Targets

- Catapult's core usage centers around ASIC design
- FPGA's require unique optimization and mapping to achieve high performance results
- FPGA multipliers are fixed in performance capability
  - 9x9 or 18x18 may not fit bit widths exactly
    - 10x8 65nm pipelined multiplier: ~250 pS
    - Virtex5 DSP48E or Stratix III DSP ~2000 pS
- Greater parallelism at lower frequencies than ASIC target may be desirable
- Technology-aware pipelining can alleviate throughput at the expense of latency
- Fabric arithmetic (carry chains) often limit Fmax
- Catapult's new FPGA accelerated libraries produce results which approach the theoretical max for FPGA frequency

Summary

- Highly Parallel bit-accurate hardware implementations are being implemented with commercial ESL tools today
- Pure ANSI C++ is familiar and requires no proprietary tools for development
- Technology-aware High-Level-Synthesis allows algorithm retargeting at an abstraction much higher than RTL
- Implementation is based on target technology characteristics yielding more efficient hardware
  - More parallelism vs higher clock rate


Mentor Graphics®
www.mentor.com

Hybrid CPU/FPGA Computing and Applications

Achieving Performance, Productivity, Portability

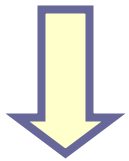
Michael S. Babst, Ph.D.
President
301-977-5970 x705
mike.babst@dsplogic.com

DSPlogic
www.dsplogic.com
1-877-DSP-LOGIC

Reconfigurable Computing Mission

Achieve Maximum Algorithm Acceleration

with Minimum Investment



Key Requirements of any RC Development Flow:

- ***Performance***
- ***Productivity***
- ***Portability***

Sweet RC Dreams…

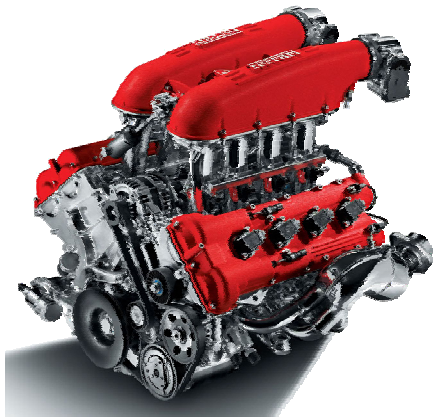
I am so happy, my C code runs 100x faster without changing one line of code. It runs on everyone's reconfigurable computer, so I can easily compare them and just buy the fastest one! It even runs on both Xilinx *and* Altera FPGAs!

RC Fears

- *What* kind of C do I have to learn?
- Can't you partition my algorithm for me?
- Fixed point? You must be kidding!
- Did you say – VHDL?

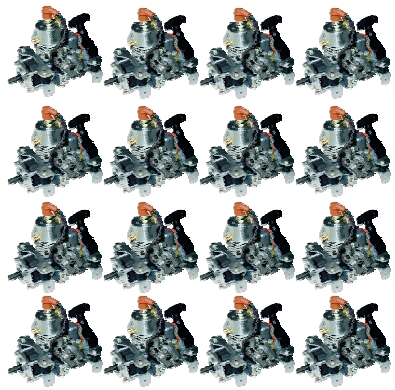
CPU vs. FPGA Computing

```
struct s_point pointprojection(struct s_plane plane, struct s_point p1)
{
//Get the projection of point p1 on the plane
v = p1 - plane.p; result = v - (v*plane.n)plane.n + plane.p
//
double temp;
struct s_point vec1, proj;
//
//
vec1.x = p1.x-plane.p.x;
vec1.y = p1.y-plane.p.y;
vec1.z = p1.z-plane.p.z;
//
//                        temp = vec1 dot plane.n
//
temp = plane.n.x*vec1.x + plane.n.y*vec1.y + plane.n.z*vec1.z;
//
//Get the global coordinates for p1's projection
//
proj.x = vec1.x - temp*plane.n.x + plane.p.x;
proj.y = vec1.y - temp*plane.n.y + plane.p.y;
proj.z = vec1.z - temp*plane.n.z + plane.p.z;

return proj;
}
```



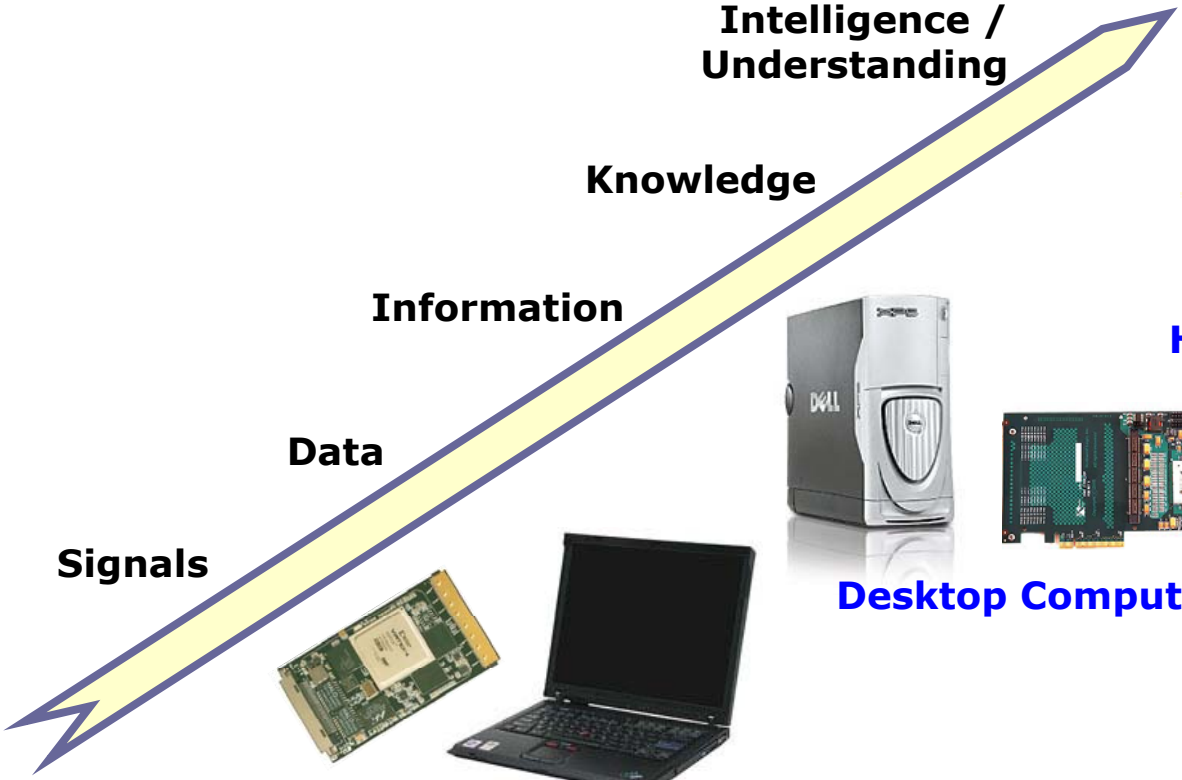
CPU:
One Big, Fast Engine

```
struct s_point pointprojection(struct s_plane plane, struct s_point p1)
{
//Get the projection of point p1 on the plane
v = p1 - plane.p; result = v - (v*plane.n)plane.n + plane.p
//
double temp;
struct s_point vec1, proj;
//
//
vec1.x = p1.x-plane.p.x;
vec1.y = p1.y-plane.p.y;
vec1.z = p1.z-plane.p.z;
//
//                        temp = vec1 dot plane.n
//
temp = plane.n.x*vec1.x + plane.n.y*vec1.y + plane.n.z*vec1.z;
//
//Get the global coordinates for p1's projection
//
proj.x = vec1.x - temp*plane.n.x + plane.p.x;
proj.y = vec1.y - temp*plane.n.y + plane.p.y;
proj.z = vec1.z - temp*plane.n.z + plane.p.z;

return proj;
}
```



FPGA:
Many Small, Slow,
Specialized Engines

Hybrid CPU/FPGA Platforms and Applications



High Performance Computing

- Informatics
- Cognitive computing
- Software acceleration
- Finance
- Data Fusion
- Cryptography
- Software Defined Radio
- Wireless/Satellite communications
- Signal Processing
- Sensor data acquisition

Desktop Computing

Portable Computing

Embedded Computing

Success Depends Upon…

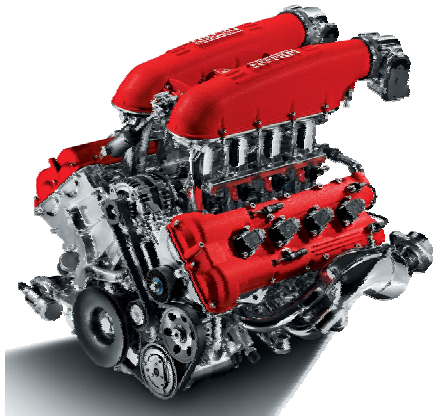
Algorithm Understanding

Interfaces

Programming and Debugging

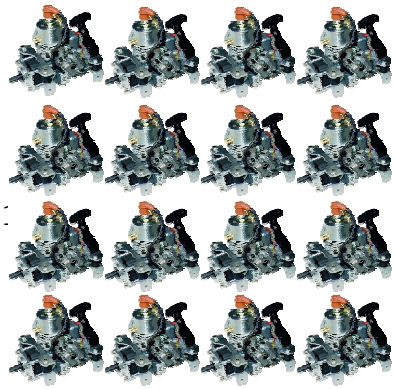
Compilation

Implementation



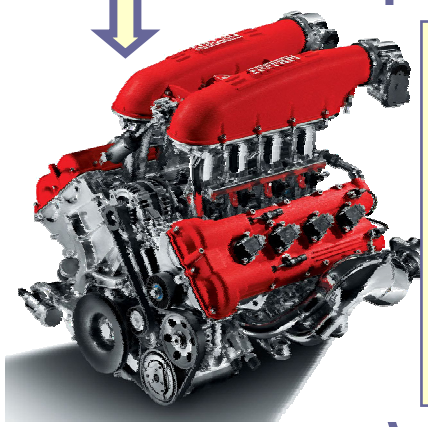
0 1 0 1 1 0 1 0 1 1 0 1

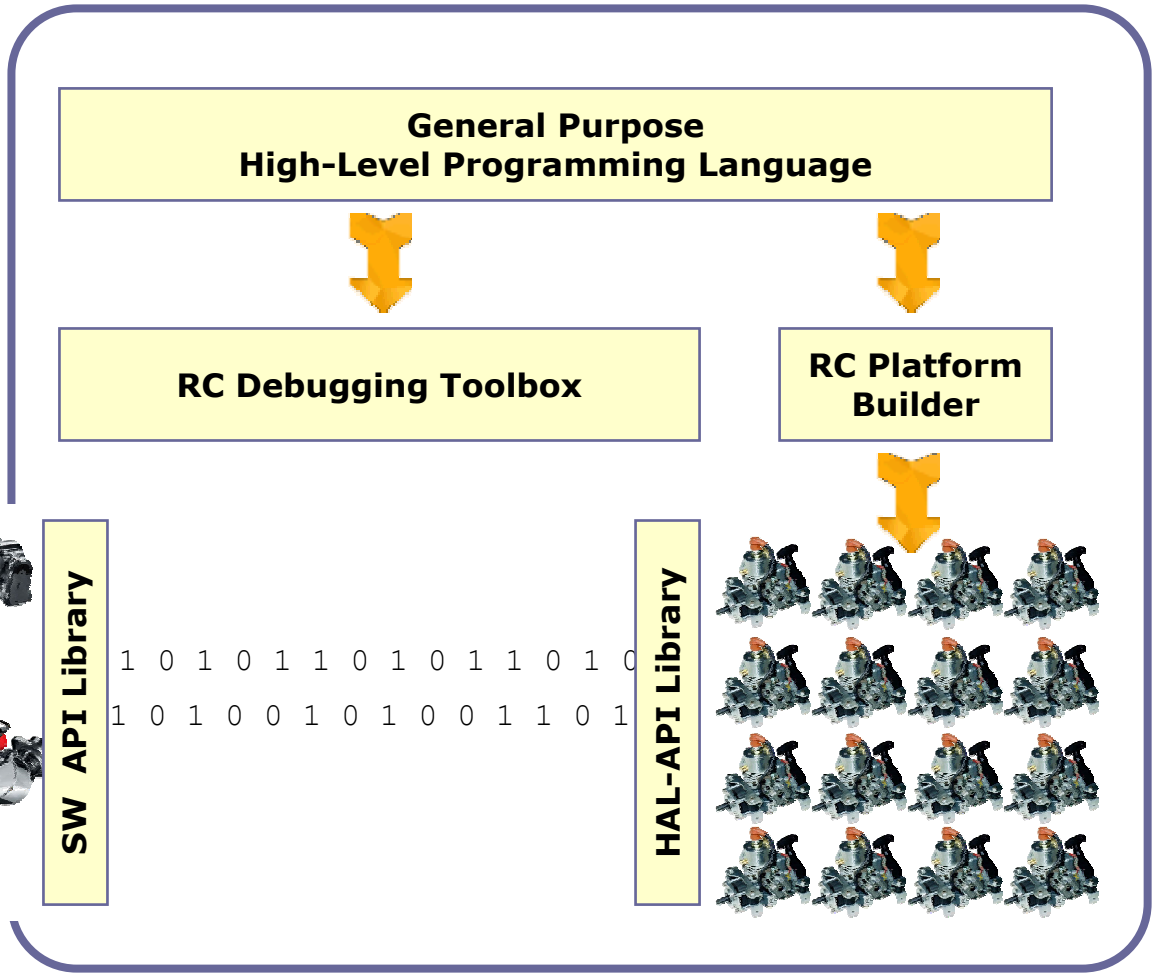
0 1 0 0 1 0 1 0 0 1 1 0 1



What is the RC Toolbox?

- Programming Environment
  - MATLAB/Simulink under Windows
- Target Runtime Environment
  - Linux, Windows
  - Matlab/Simulink Not Required
  - Platform Support Packages



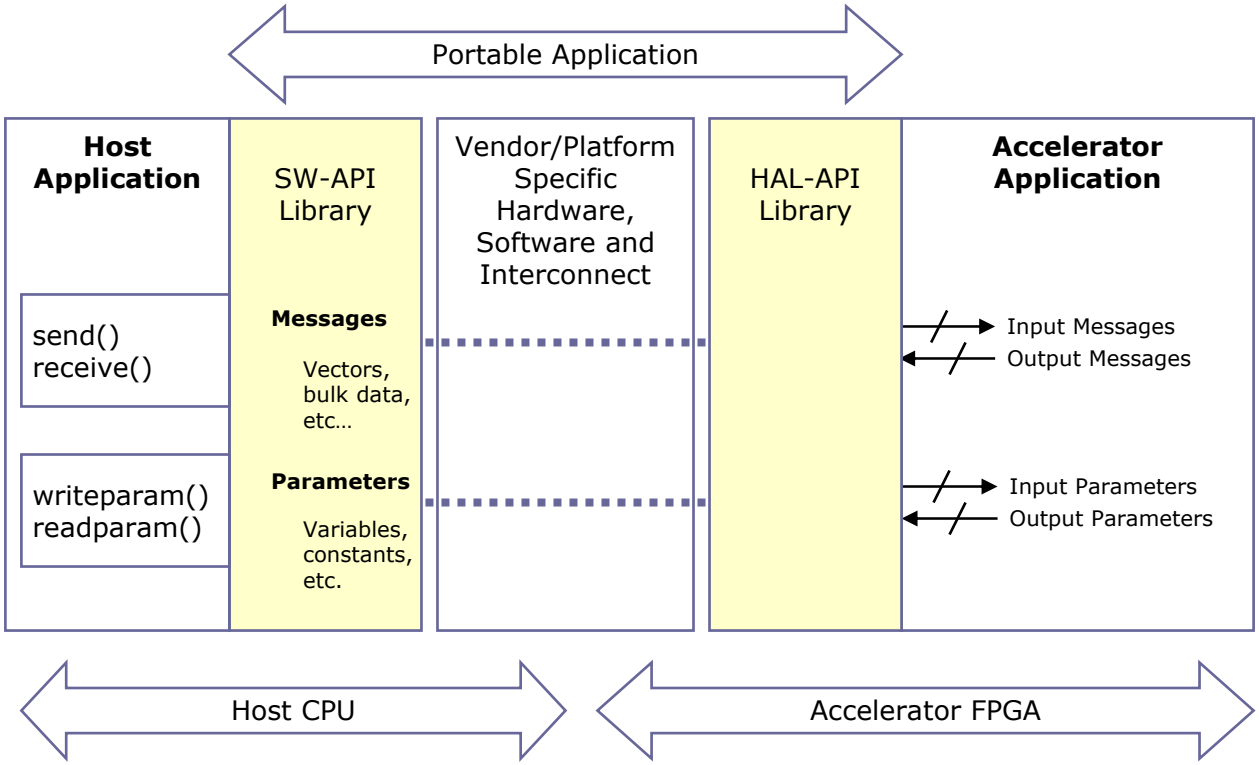


Interface Abstraction for Portability

- How will you call the Accelerated Function?

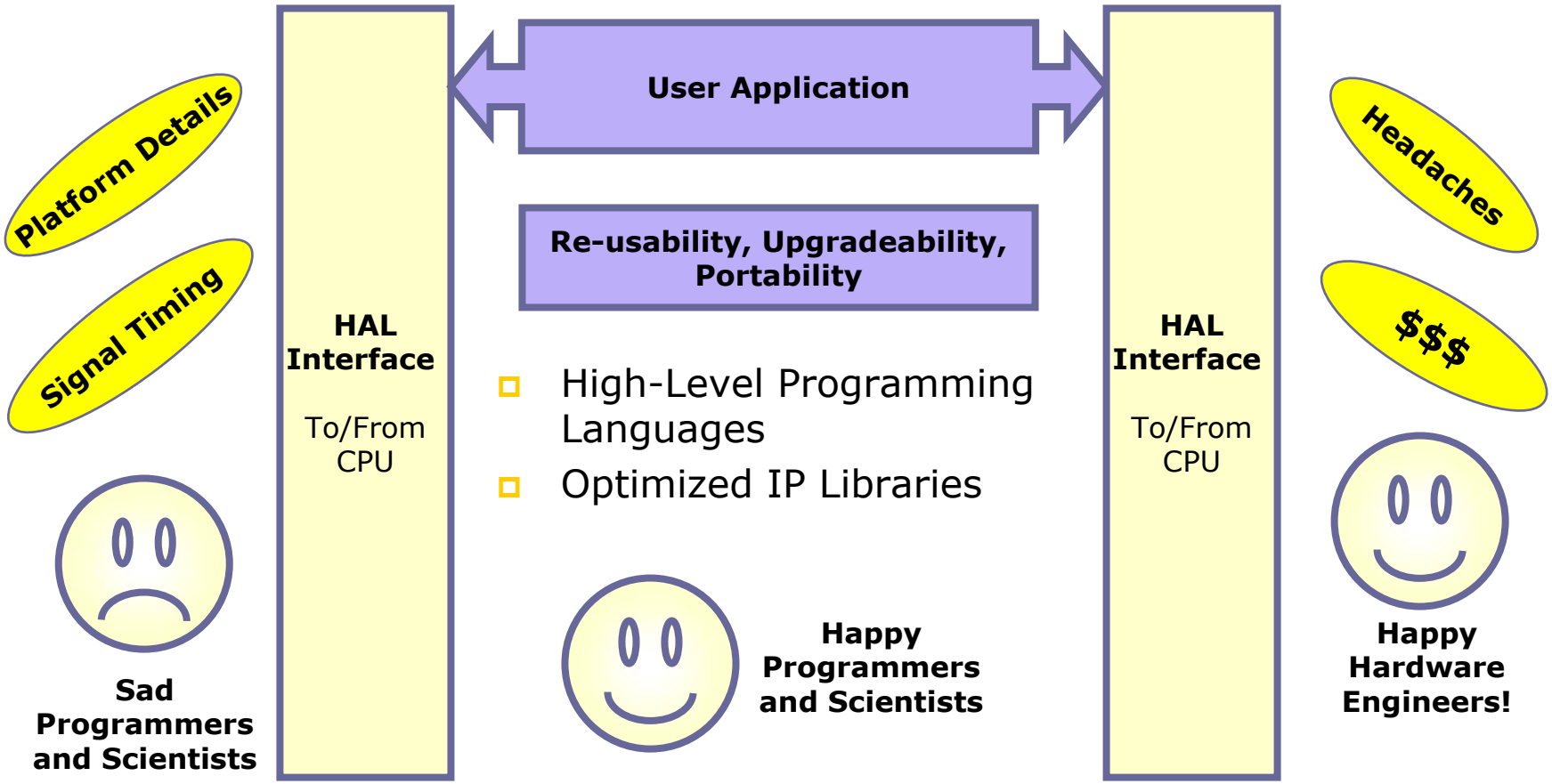
Function	OpenFPGA GenAPI	RCIO API
Setup Accelerator Function Call	ofpga_init ofpga_load_algorithm ofpga_status ofpga_close ofpga_run	rcio_init rcio_config rcio_status rcio_close
Send/Receive Data to Accelerated Function	ofpg_send ofpga_receive	rcio_send rcio_receive rcio_stream
Send/Receive Constants to Accelerated Function	ofpga_write_register ofpga_read_register	rcio_writeparam rcio_readparam

Interface Abstraction for Portability



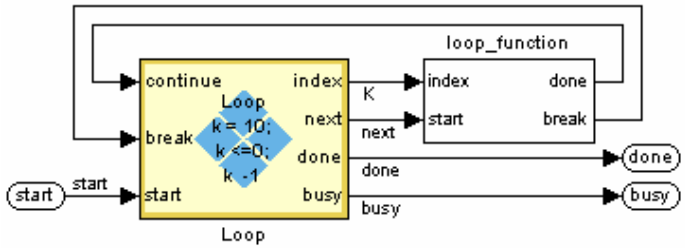
Interface Abstraction - HAL

- How will you program the Accelerated Function?



Reconfigurable Computing Toolbox

- High-Level, General Purpose Graphical Programming Language Providing:
- High Performance
  - 200 MHz Typical Clock Speeds on Virtex 2 and Virtex 4
  - Easily express of parallelism and pipelining
- High Productivity
  - High-Level Programming Abstraction
  - Low learning curve
  - Debugging within Matlab
  - Optimized Bitstream Generation within MATLAB™
- High Portability
  - Interface Abstraction
  - Multiple CPU(s) /FPGA(s)
  - Embedded, portable, desktop, HPC



High-performance, General Purpose Graphical Programming Language
RC Blockset

Program Flow sequential, parallel, and pipelined	**Math** (including floats)
Parallel Memory Access	**Portable RC Abstraction Layers**

Integration with other technologies

- **IP Cores**
- **HDL**
- **Other HLLs**

RC Debugging Toolbox
Program validation, data visualization

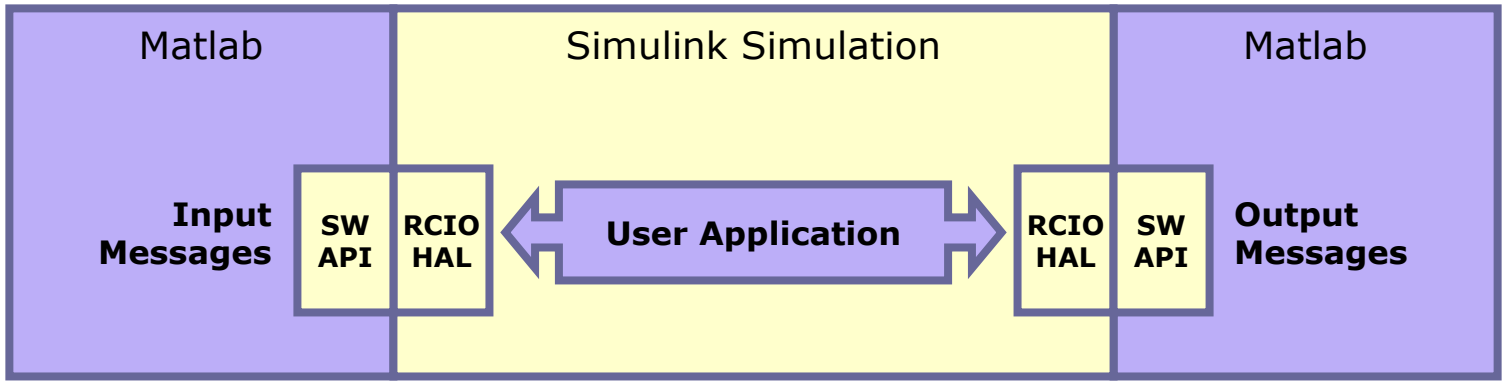
Host I/O

Platform Support Package

RC Platform Builder
Fully automated bitstream compilation

Verified, robust, high-performance bitstream

Programming and Debugging Environment



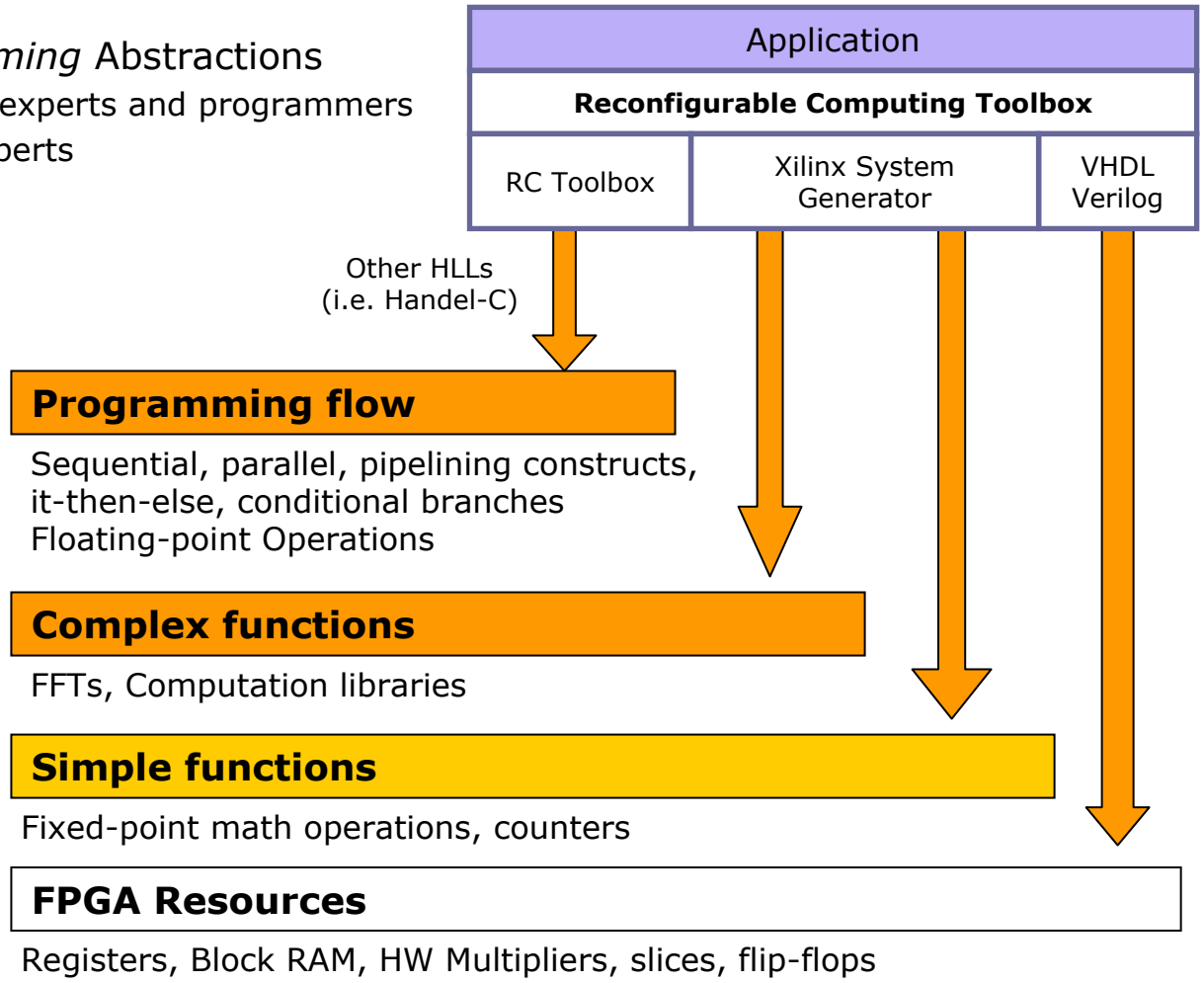
- Program, Debug, and Validate entire Accelerated Function within Matlab and Simulink.
- Simulate / Validate FPGA core
  - Only User Application Needs to be Simulated
- Generate Function inputs and Analyze Outputs in Matlab
  - Convenient use of standard Matlab vector/matrix handling, Plotting, etc.
- Debug for all target platforms simultaneously
  - Target Runtime Platform is Selectable in Platform Builder
  - Guaranteed operation up to SW API level
- Target/Runtime Environment
  - Windows or Linux Platform
  - Does not require Matlab, Simulink, Windows

High-Level Graphical Programming

For Performance, Productivity, and Portability

Programming Abstractions for Portability

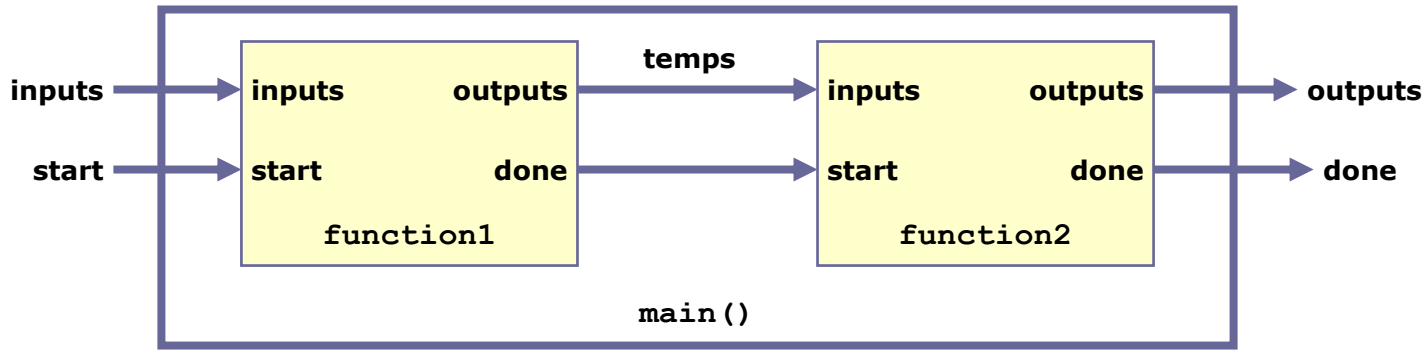
- Multiple FPGA *Programming* Abstractions
  - High level for domain experts and programmers
  - Low level for FPGA experts



Sequential Programming

- Example main() program

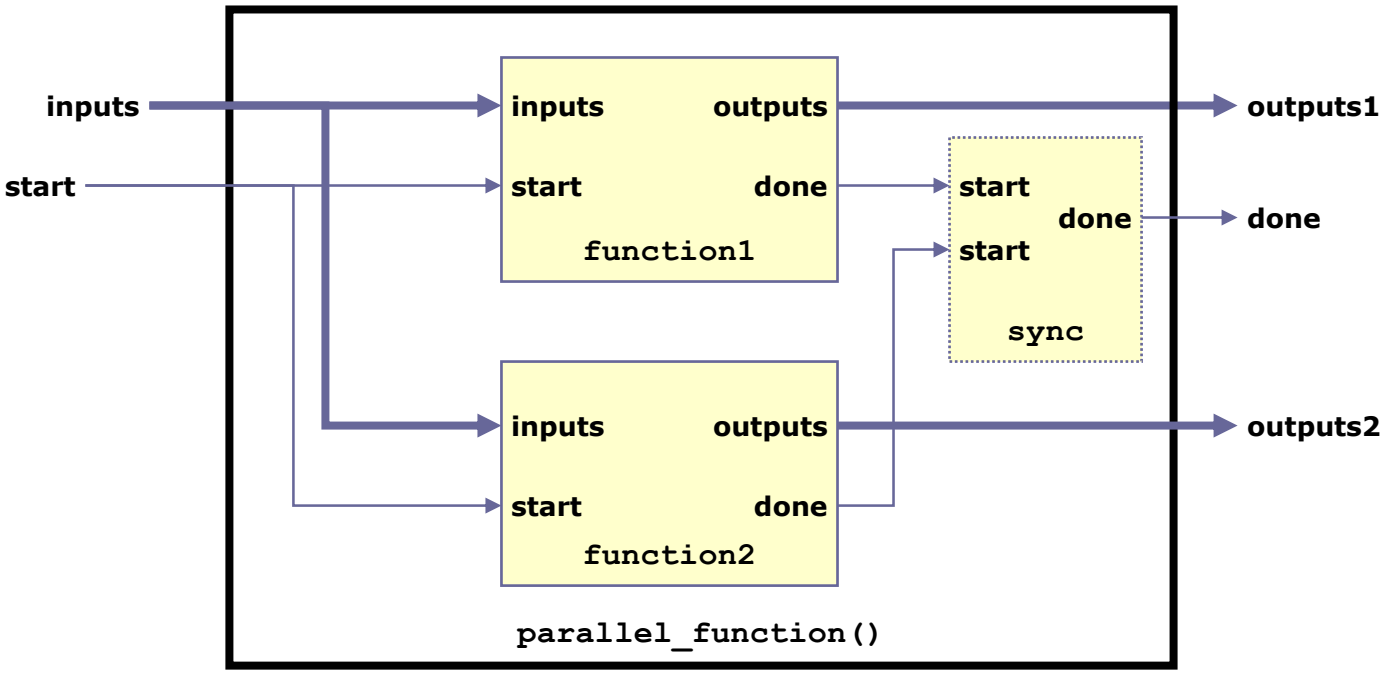
```
// Equivalent pseudo code

outputs main(inputs){
  temps   = function1(inputs);
  outputs = function2(temps);
}
```



- Hierarchical variable scope
- Data Types Automatically Propagated
- Variables with Global and Local Scope Minimize Clutter

Parallel Programming

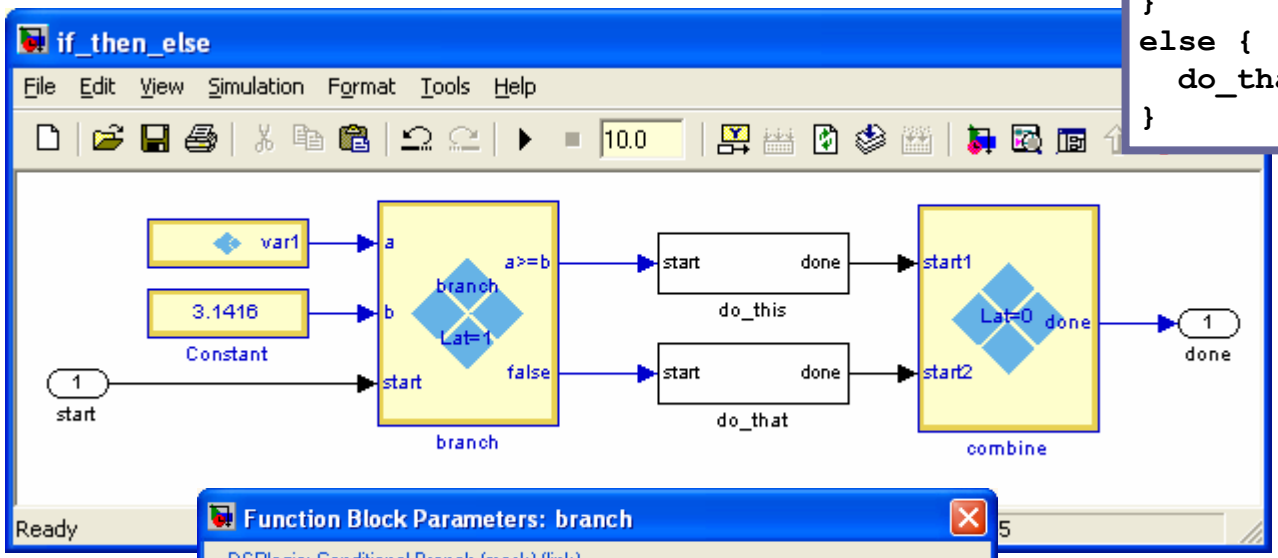
```
// Equivalent pseudo code

parallel_function(inputs, outputs1, outputs2){
  #pragma parallel
  outputs1 = function1(inputs);
  outputs2 = function2(inputs);
}
```

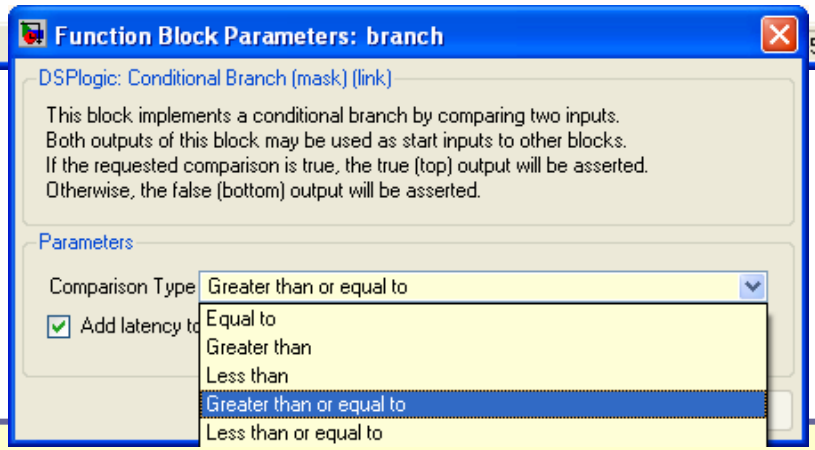


Conditional Branch (if-then-else)

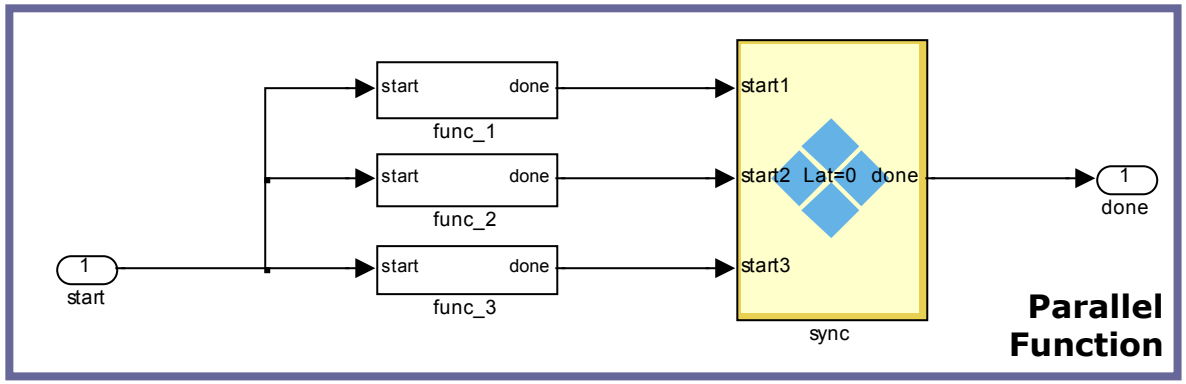
- If condition is true, initiate a task
  - otherwise, initiate an alternate task
- Requires Combine block, i.e. "}"

```
// Equivalent pseudo code

if (a xx b)   {
  do_this();
}
else {
  do_that():
}
```

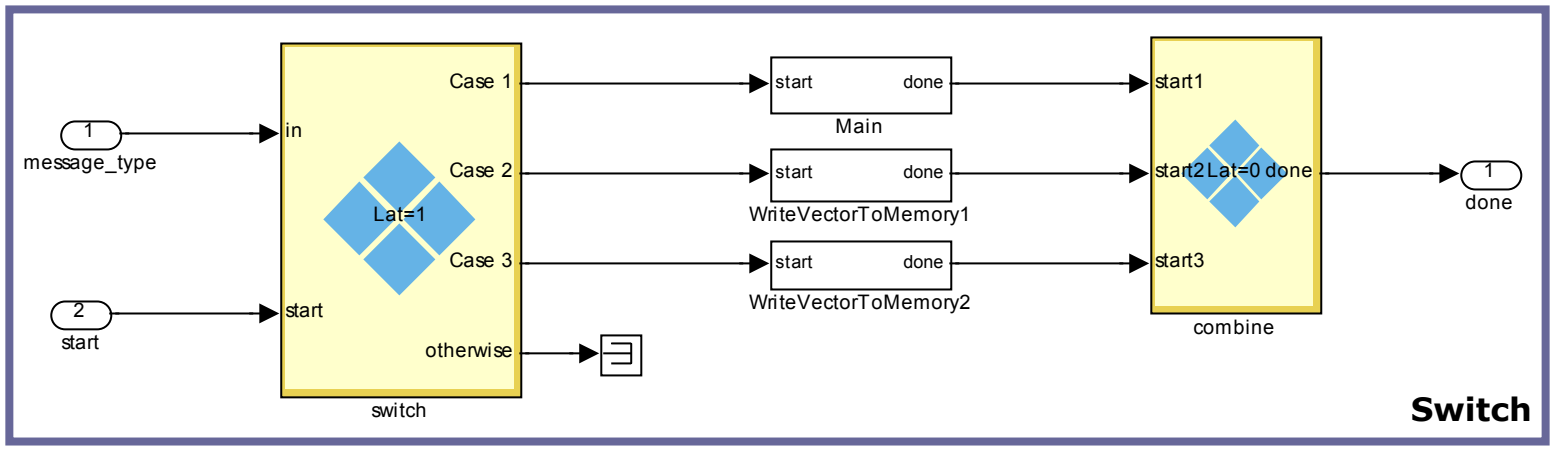




Parallel vs. Branch/Switch operations

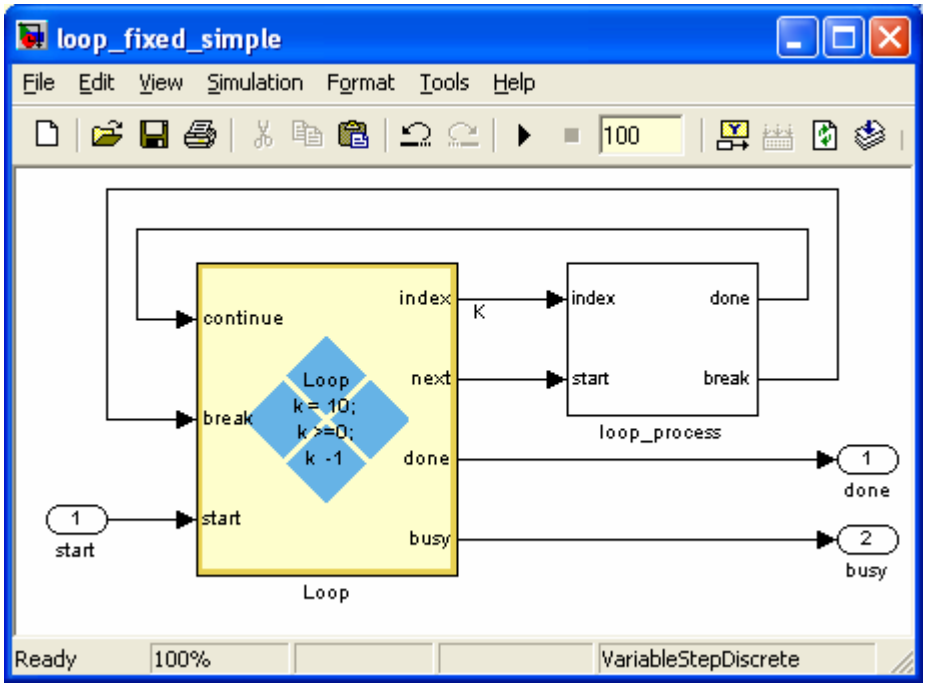


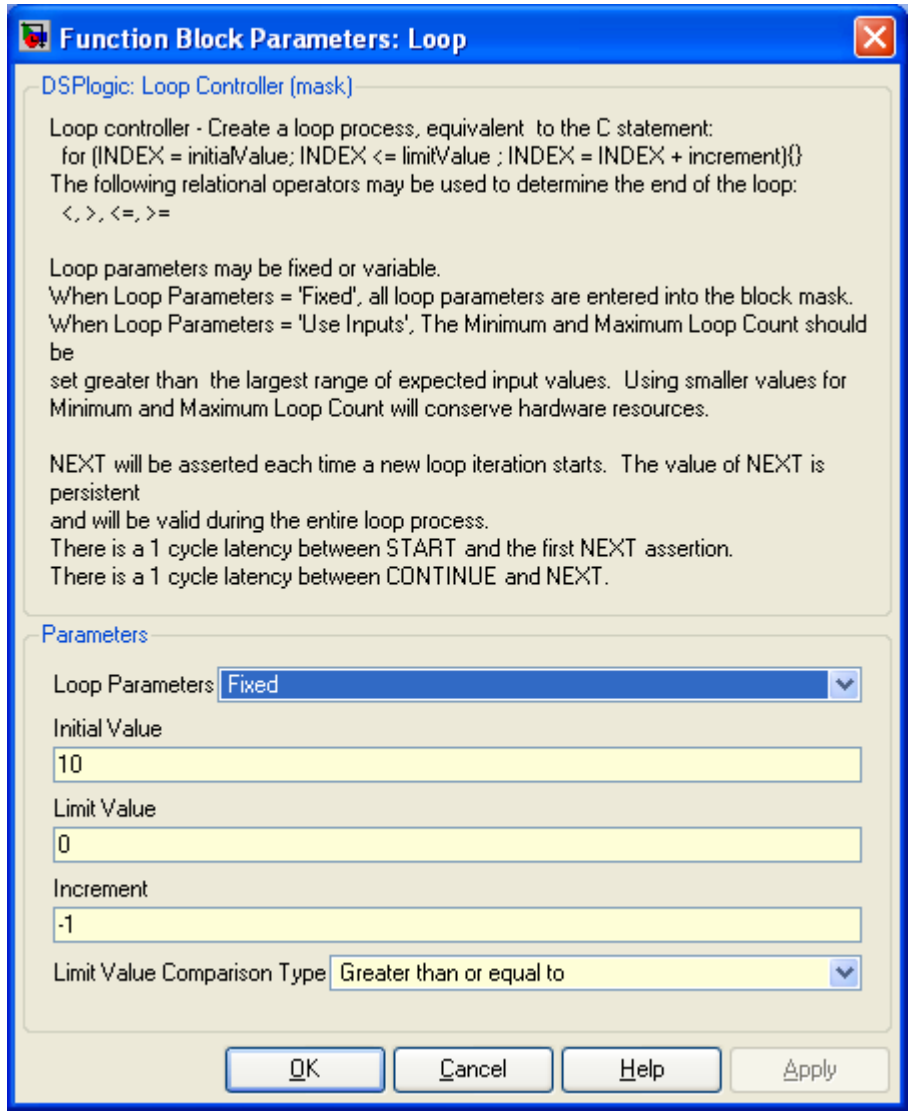
More Efficient Resource Usage



Iterative Constructs (Loops)

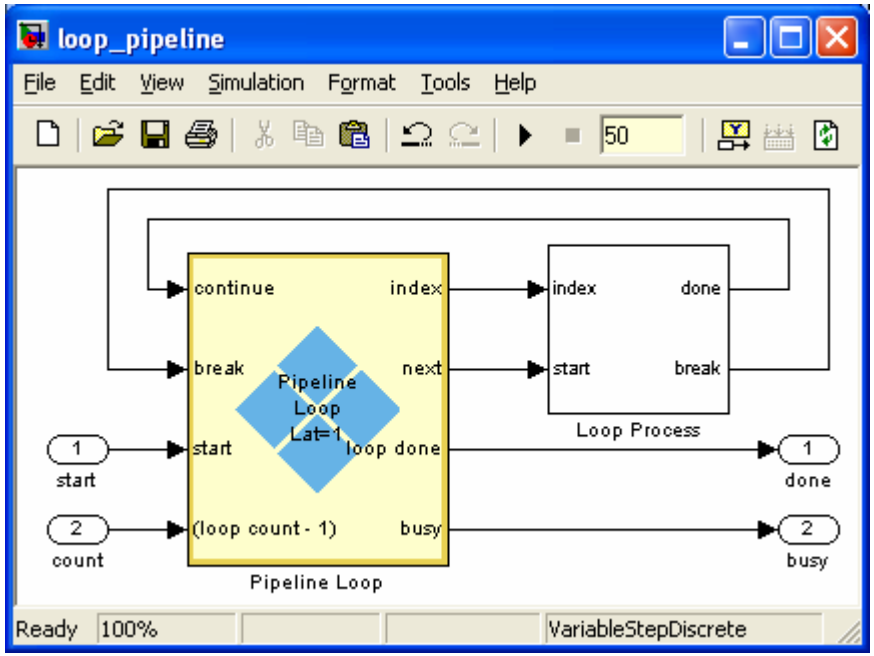
```
// Equivalent software operation

for (k=10;k>=0;k=k-1){
  loop_function(index);
}
```





Pipelined Loop

```
// Equivalent software operation

for (k=0;k<loop_count;k++){
  loop_function(data1);
}
```



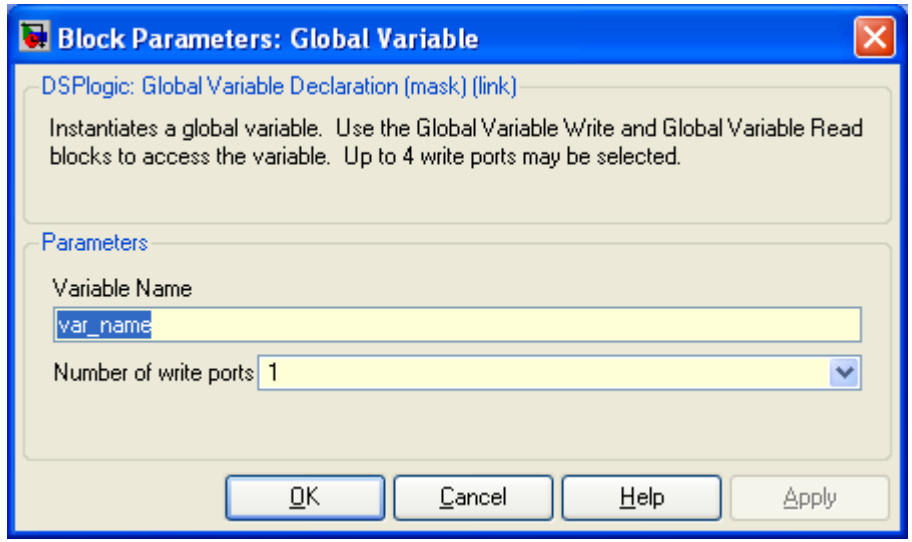
- Simple loop execution
  - 'Loop' initiates execution of 'loop_function' in accordance with loop equation
  - 'K' is updated and valid on each 'loop_function/start'
- Pipelined
  - 'Loop' will start 'loop_function' on consecutive clock cycles until loop is complete or a 'break' event occurs.
- Uses
  - When maximum loop execution speed is required
  - When loop_function() has no data dependencies, or well-known dependencies.
- Break
  - 'loop_function' can force a break from loop any time
- Automatic pipeline latency handling
  - 'Pipeline Loop/done' is not asserted until 'loop_function' pipeline is completed.

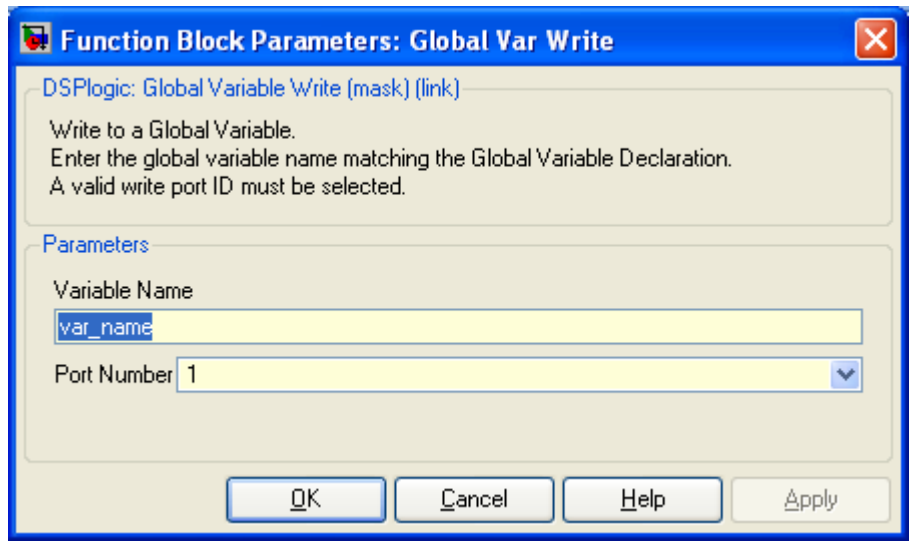
Variable read/write operations

- Declaration

- Write Access
  - Select Port
  - Variable type automatically determined
  - All ports must write same type

- Read Access

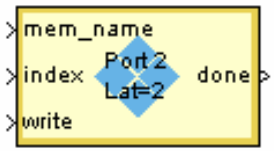




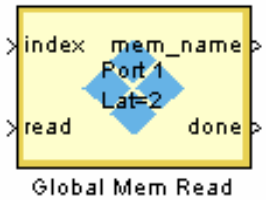
Memory / Arrays

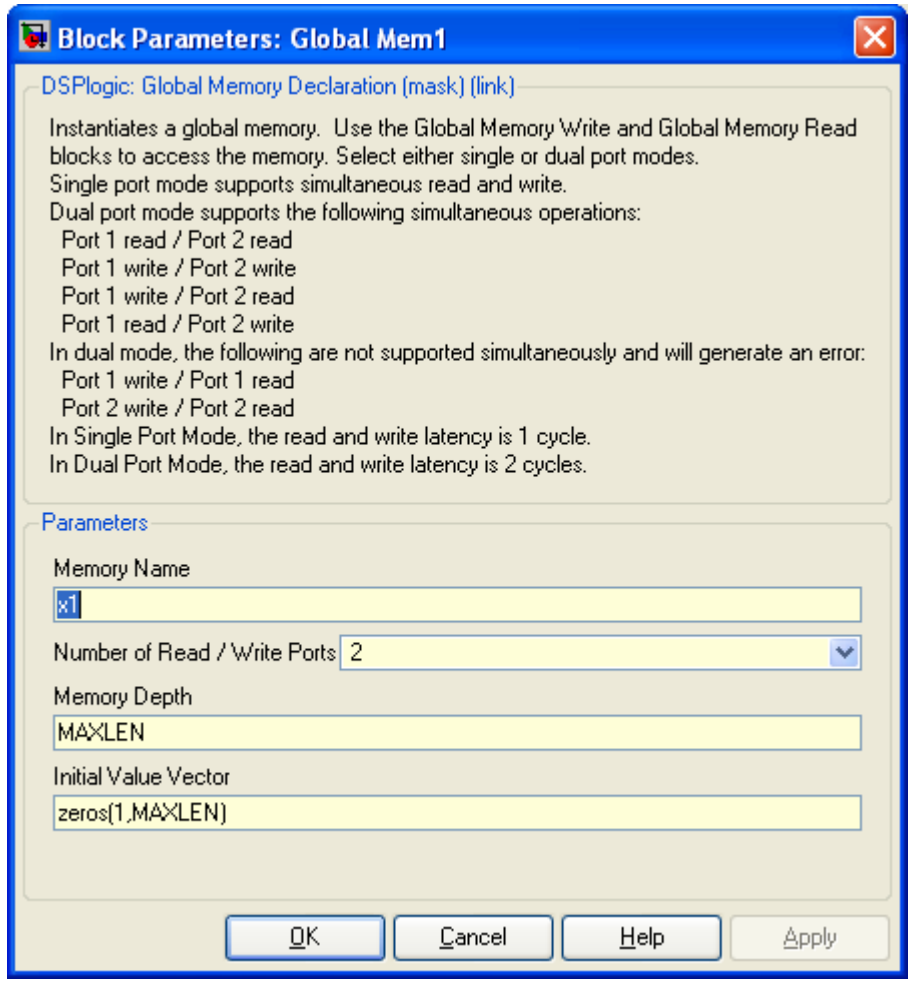
- Declaration

- Write Access
  - Select Port
  - Variable type automatically determined
  - All ports must write same type



- Read Access



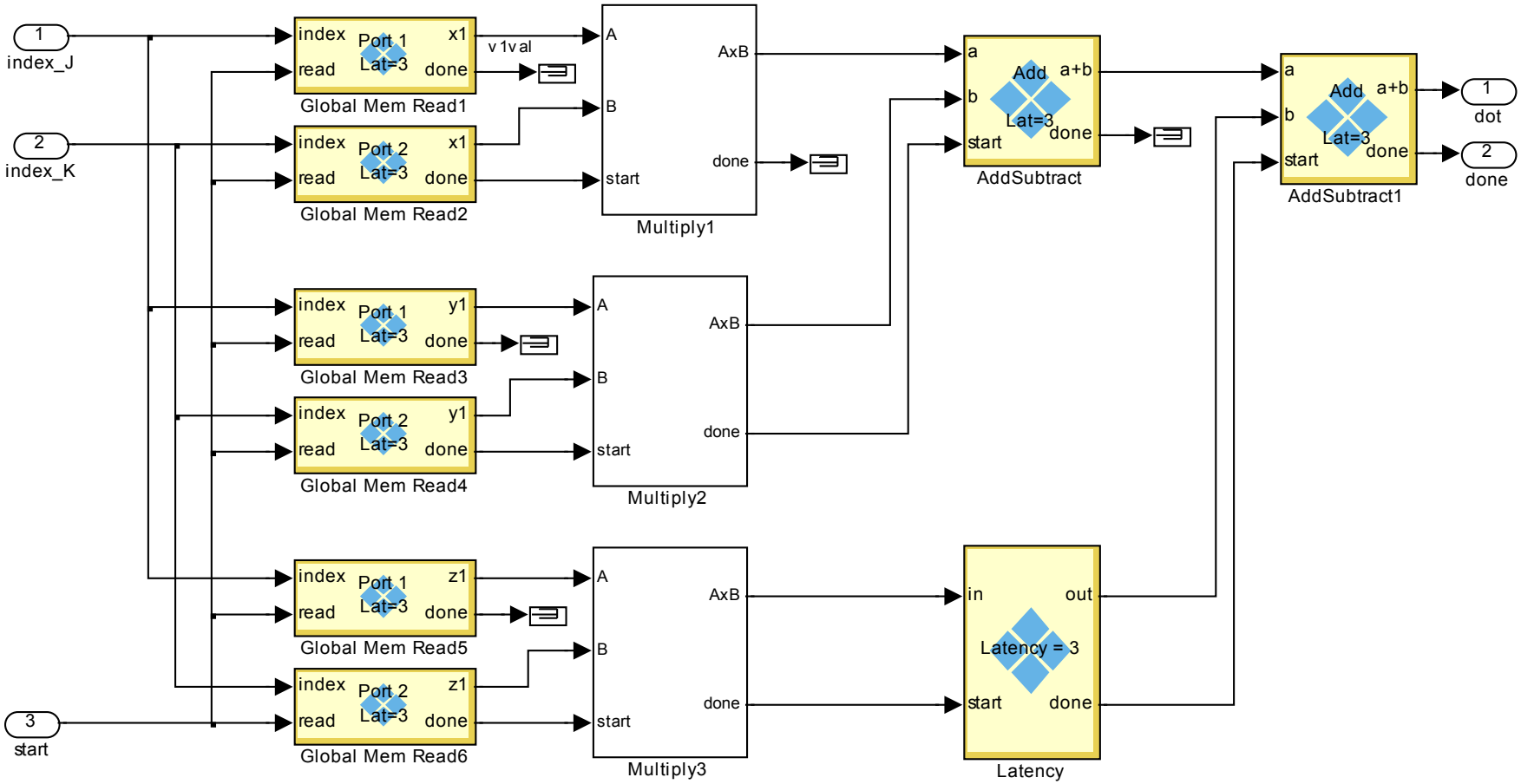


Example Function

Reference function

```
function indx = testfunction(x1,y1,z1);

lcount = length(x1);
for j = 1:lcount
  for k = 1:lcount
    indx(j,k) = x1(j)*x1(k) + y1(j)*y1(k) + z1(j)*z1(k);
  end
end
```

Example – 3-D dot product



This simple program performs
6 memory reads, **3** Floating-point Multiplies and **2** Floating Point Adds
in **Parallel** and **Fully Pipelined** (once every clock cycle)

You Can Do It!

- Bioinformatics
- Seismic Processing
- Molecular Dynamics
- Crash Simulation
- Cryptography
- Database searching
- Finance
- Astrophysics
- Remote Sensing
- Signal and Image Processing
- Traffic Simulation

Not just for DSP!

Contact Information

- For Additional Information, Please Contact:

Michael S. Babst, Ph.D.
President
301-977-5970 x705
mike.babst@dsplogic.com

DSPlogic

1-877-DSP-LOGIC
www.dsplogic.com

The Mitrion Virtual Processor

Using FPGAs in HPC

Stefan Möhl, Co-founder, Chief Strategy Officer,

Mitrionics

mitrionics™

Why FPGAs in HPC?

- Compared to fixed silicon at the same process technology (65nm):
  - ~10 times slower clock frequency
  - ~50-100 times larger area used per gate
- Compared to CPUs
  - 10-100 times faster
  - At 20-25 Watts!
  - Performance gap is increasing

mitrionics™

The HPC FPGA Eco-System

mitrionics™

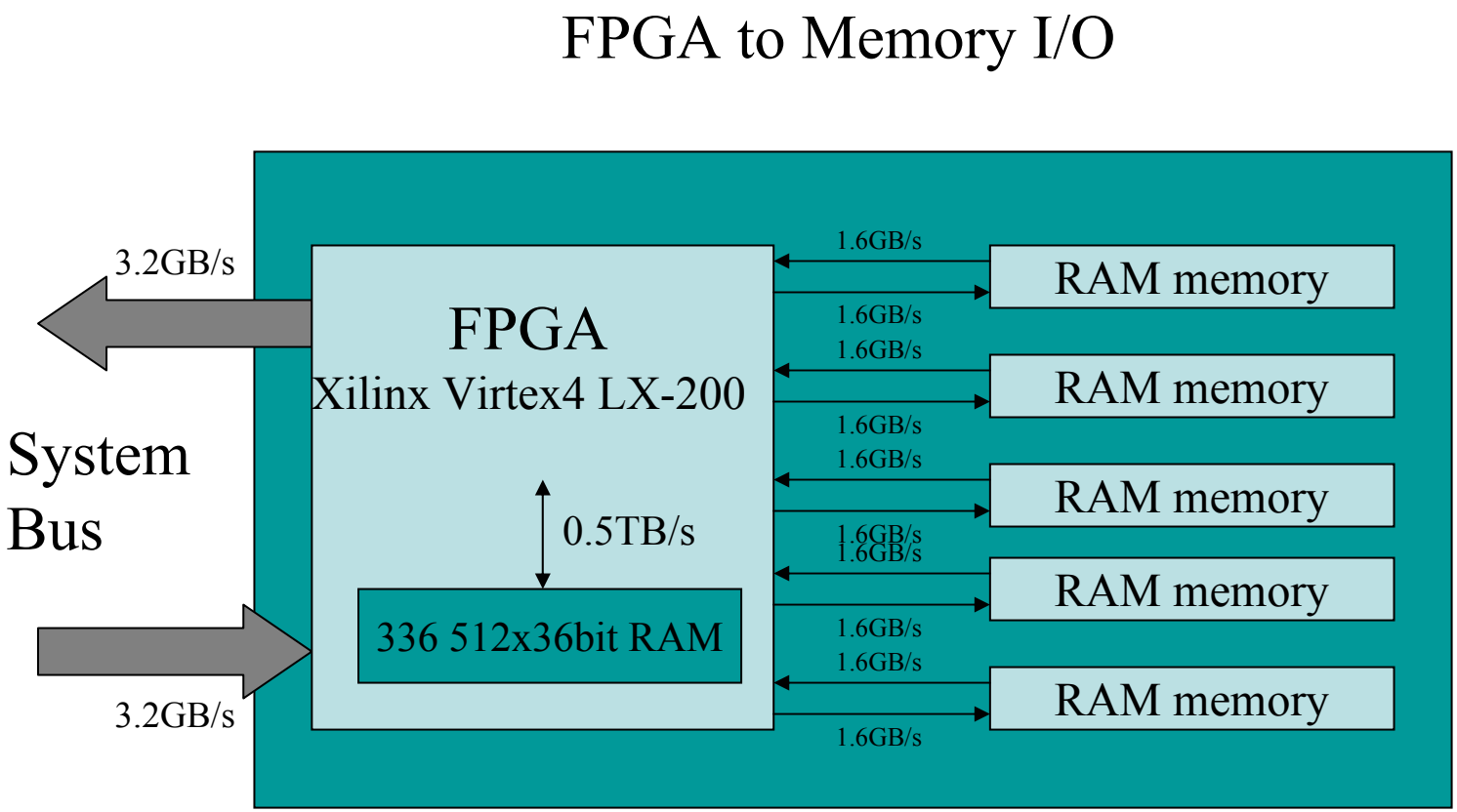
FPGAs in HPC vs Embedded

- Embedded is about building devices
  - Adapt the machine to the implementation
  - Hardware design

- HPC is about using the machine
  - Find the implementation that suits the machine
  - Software programming

mitrionics™

A Typical FPGA Module (SGI RC100)



mitrionics™

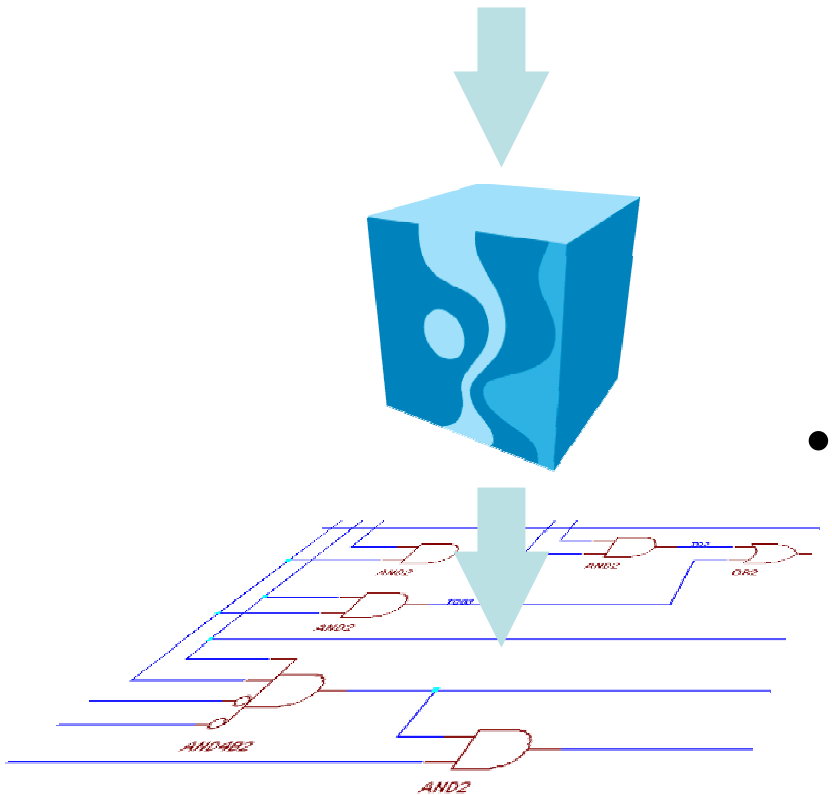
The Mitrion Platform

1) The Mitrion Virtual Processor
   - A configurable processor design for a fine-grain massively parallel, soft-core processor
   - 10-30 times faster than traditional CPUs at 20-25 Watts
   - Executes a program in an FPGA
2) The Mitrion-C programming language
   - An intrinsically parallel C-family language
3) The Mitrion Software Development Kit
   - Compiler
   - Debugger/Simulator
   - Processor configurator

mitrionics™

The Mitrion Virtual Processor

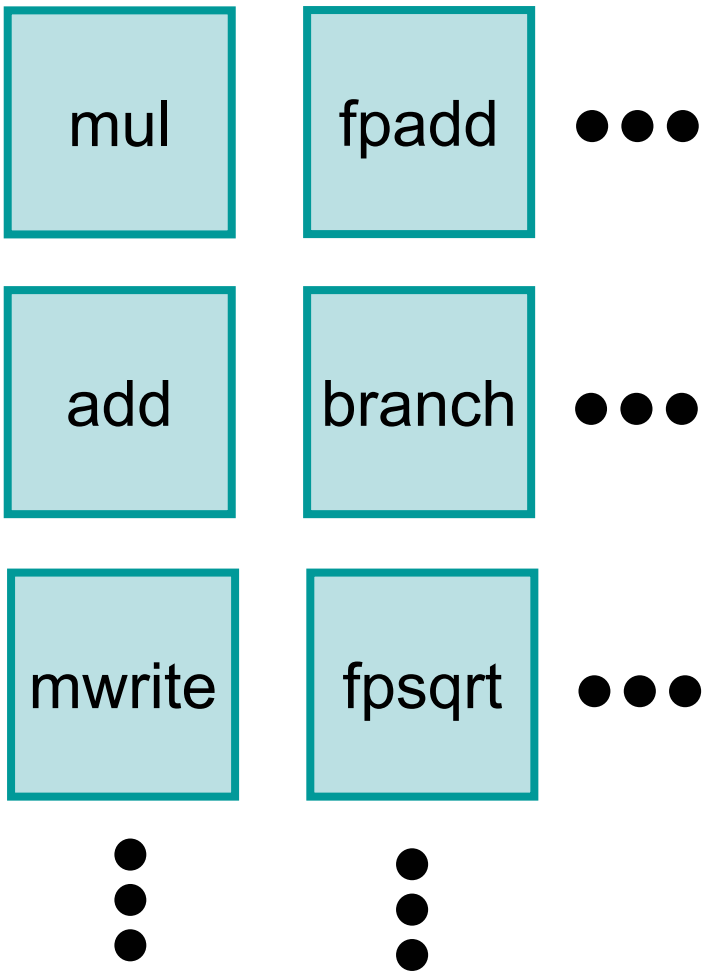
```
int:48<30> main()
{
  int:48 prev  = 1;
  int:48 fib   = 1;

  int:48<30> fibonnacci = for(i in <1..30>)
  {
    fib  = fib+prev;
    prev = fib;
  } <>fib;

} fibonnacci;
```



- Fine-grain parallel
  - Instruction level parallel
- A configurable processor design
  - The processing elements suitable for the application
  - Processing elements adapted to bit-width
- Allows software programming of FPGAs
  - You program the MVP, rather than design HW

mitrionics™

The Mitrion Virtual Processor



A set of pre-designed and verified hardware tiles

- Roughly 60 different kinds of tiles
- Arithmetic-logic and I/O tiles
- Control flow tiles

All tiles can connect to each other

- Tiles will function correctly regardless of configuration

Tile-set is Turing complete

- There is always a configuration of tiles that perform any program you write

Adaptable register and bus widths

- Tiles function correctly from 1-64 bits

mitrionics™

Mitrion Offers Portability And Scalability

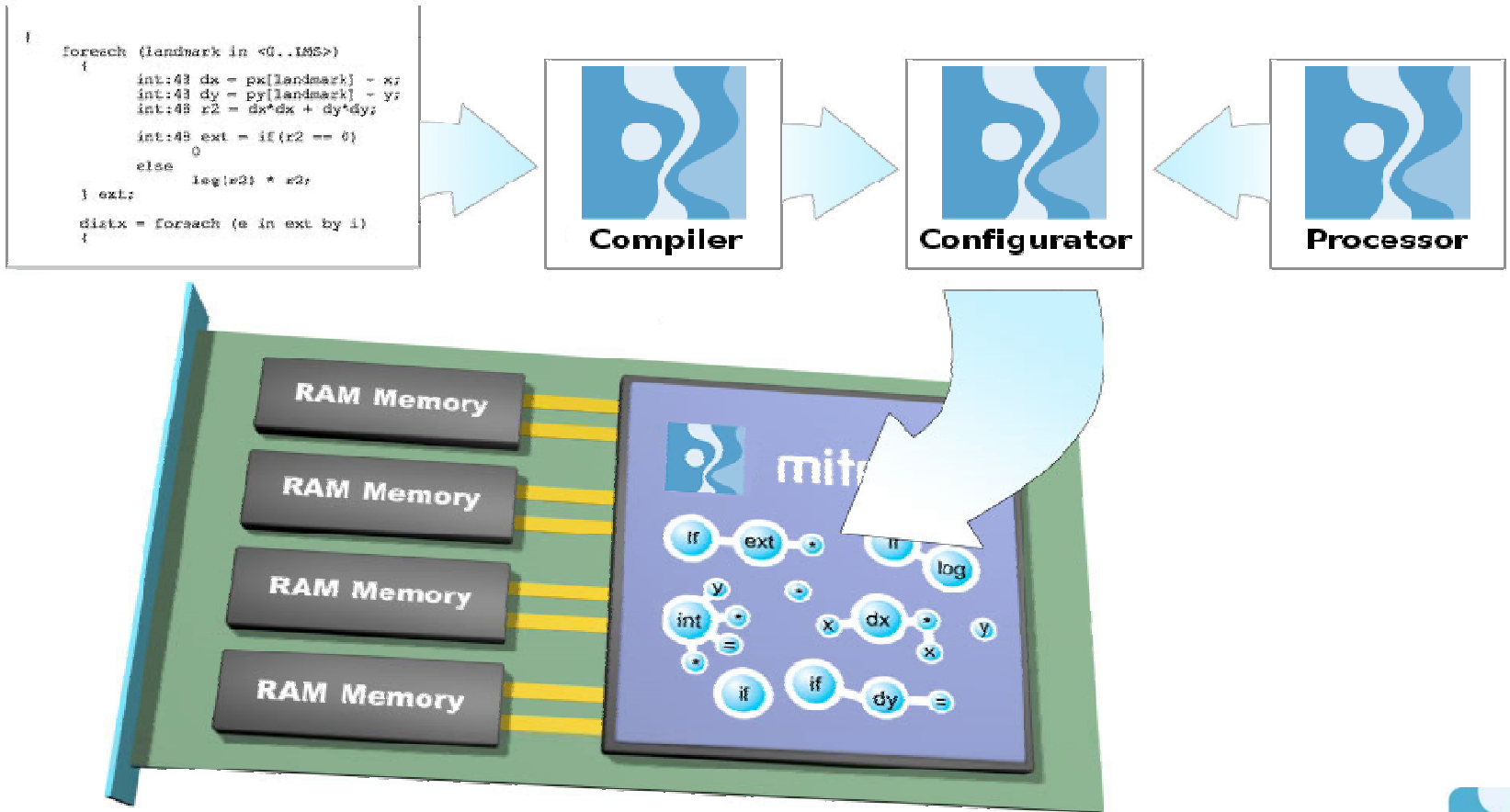
- You program the processor, you do not design a circuit for a specific FPGA platform

Just configure a processor for the new platform from your old source-code.

- Easy upgrades to the next generation of performance available.
- Exchange code with colleagues on other platforms

mitrionics™

The Mitrion Platform



mitrionics™

Programming the Mitrion Virtual Processor

Mitrion-C

- The MVP needs a fine-grain, fully parallel programming language
  - Parallel at the level of individual instructions
- A new C-family language - not ANSI-C!
  - Similar to C about as much as Java is similar to C
  - The code has to be re-written for parallelism anyway
- Allows a complete data-dependency graph to be created of the algorithm
  - Full representation of all parallelism in the algorithm

mitrionics™

Mitrion-C

- Implicitly parallel
  - Describes *data dependencies* rather than order-of-execution
  - Allows fine-grain parallelism to be described
- Syntactic support to make design-decisions regarding parallelism salient
  - Designed to *preserve* parallelism in the algorithm
  - Designed to *reveal* parallelism in the algorithm
- Defined behaviour
  - Parallelism is an integrated part of the language, not an optimization or a language extension

Preserving parallelism

- Data dependencies, not order of execution

```
x = a + b;
y = c + d;
Z = x * a;
```

Preserving Parallelism

- Non-strict execution

```
int:8 a; int:8 b; int:8 c;
x = f(a, b, c);

int:8<100> a; int:8<100> b;
(x, y) = f(a, b);
z = g(x, y);
```

mitrionics™

Revealing parallelism

- Loop syntax reveals loop dependencies

```
int:8 s = 0;
x = for(e in v)
{
      s = s + e;
} s;
```

```
x = foreach(e,f in u, v)
{
      s = e * f;
} s;
```

mitrionics™

Specifying parallelism

- Type system controls kind of parallelism

```
int:8<100> a;          // Pipelined execution

int:8[100] a;          // Unrolled execution

mem int:8[100] a;      // Random access
                          sequential
```

mitrionics™

Thank You!

Mitrionics AB
Ideon Science Park
SE-223 70 Lund
www.mitrionics.com

stefan@mitrionics.com

mitrionics™

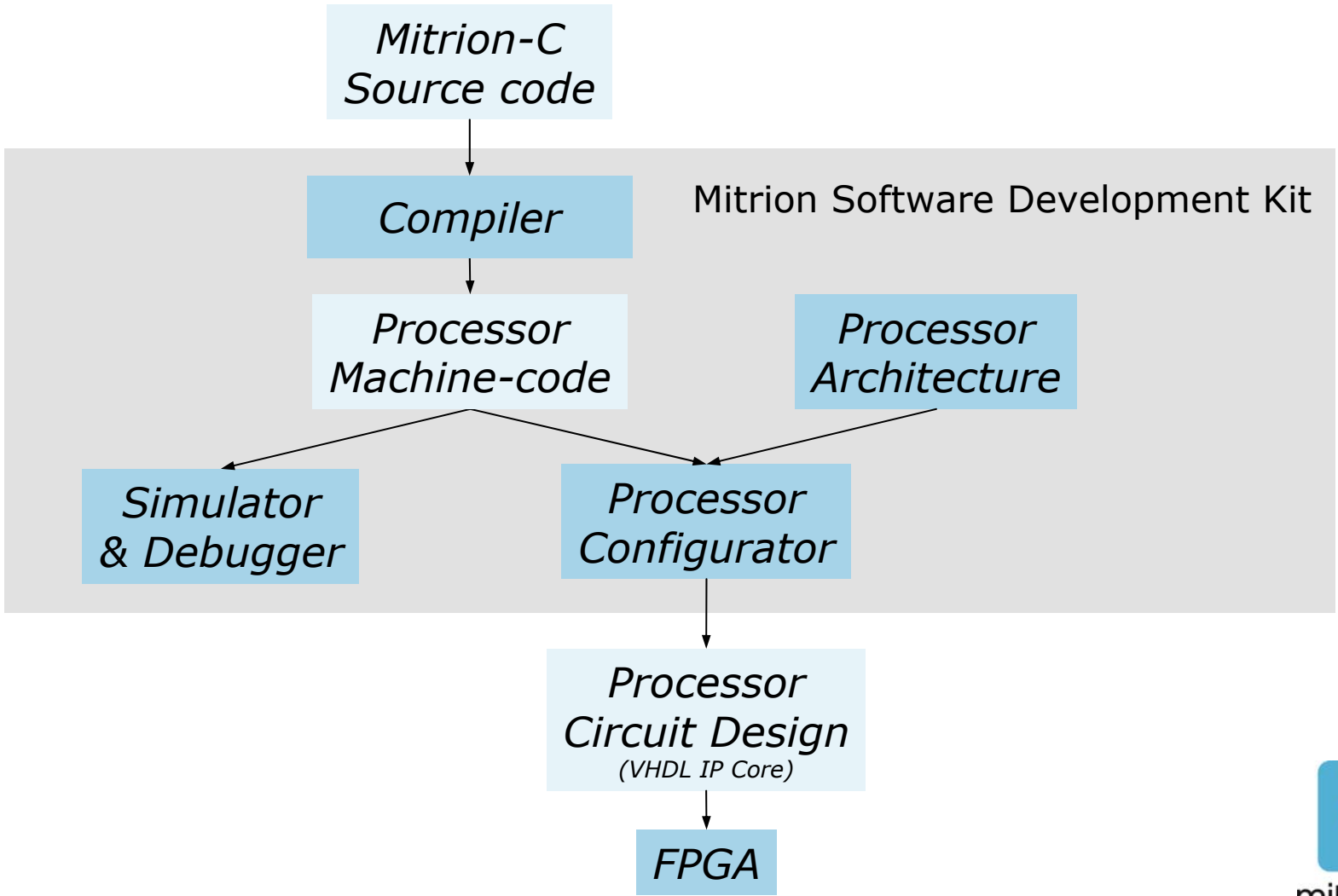
Applying the Mitrion Platform

1) Identify the time-critical portion of the program
2) Write it as code in Mitrion-C
3) Perform a function call to run your Mitrion-program

- Most of your code is unchanged

mitrionics™

Compiling A Mitrion-C Program



mitrionics™

NVIDIA®

NVIDIA CUDA Software and GPU Parallel Computing Architecture

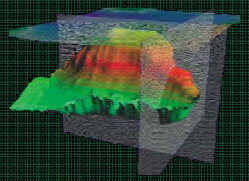
David B. Kirk, Chief Scientist

Outline

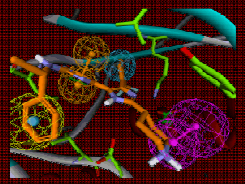
- Applications of GPU Computing
- CUDA Programming Model Overview
- Programming in CUDA – The Basics
- How to Get Started!

- Exercises / Examples Interleaved with Presentation Materials
  - Homework for later ☺

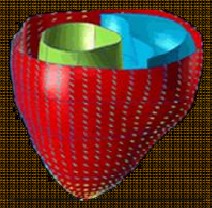
Future Science and Engineering Breakthroughs Hinge on Computing



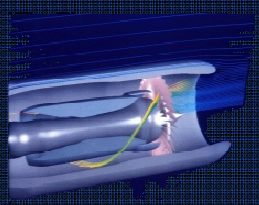
Computational Geoscience



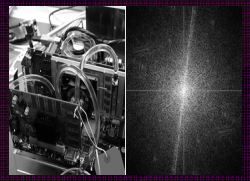
Computational Chemistry



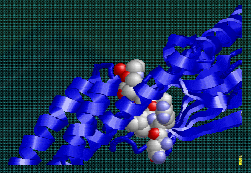
Computational Medicine



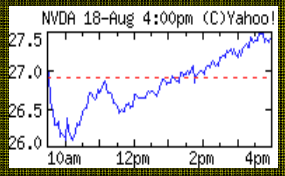
Computational Modeling



Computational Physics



Computational Biology



Computational Finance

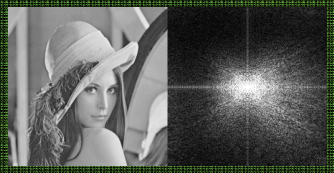
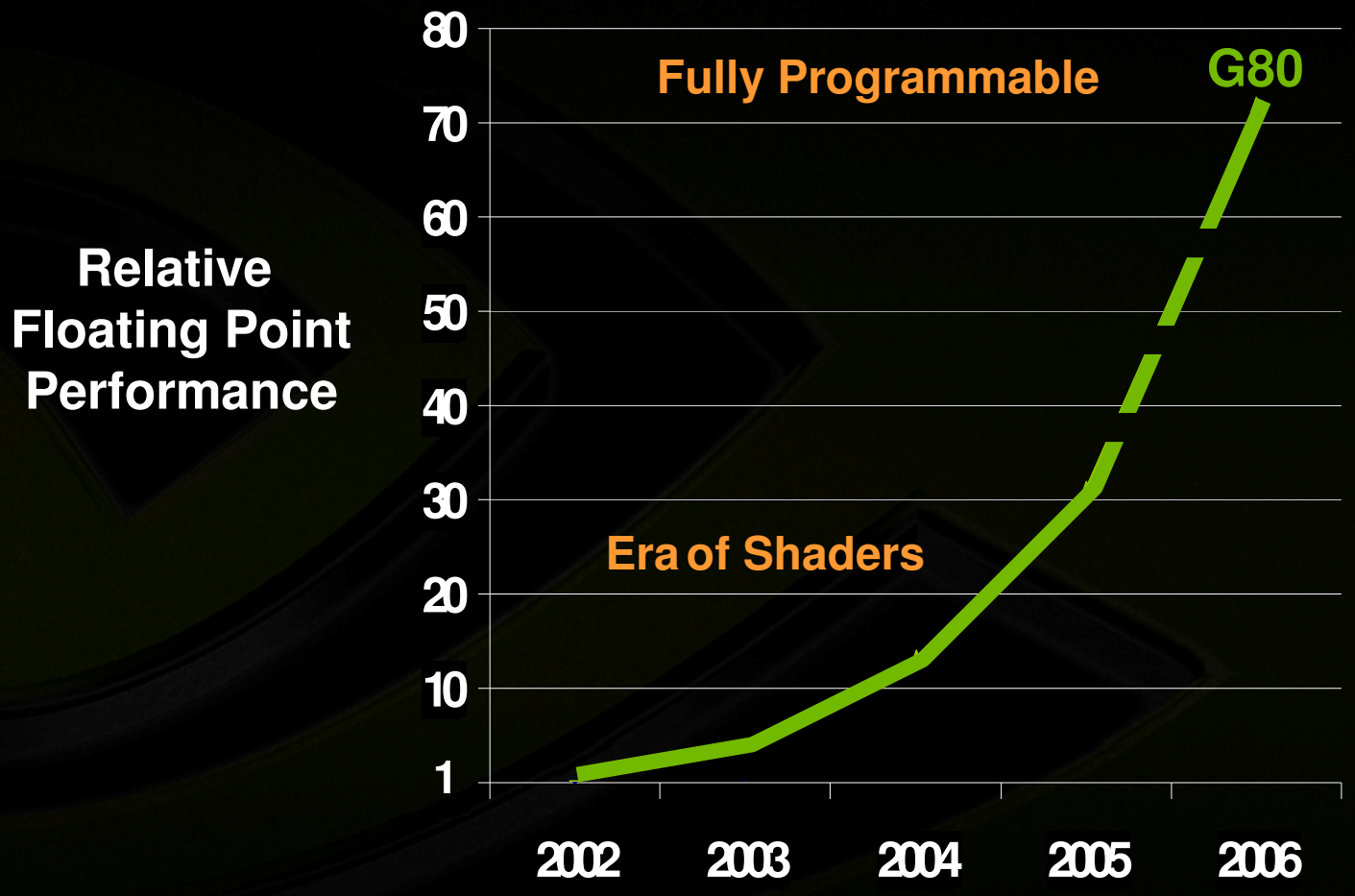


Image Processing

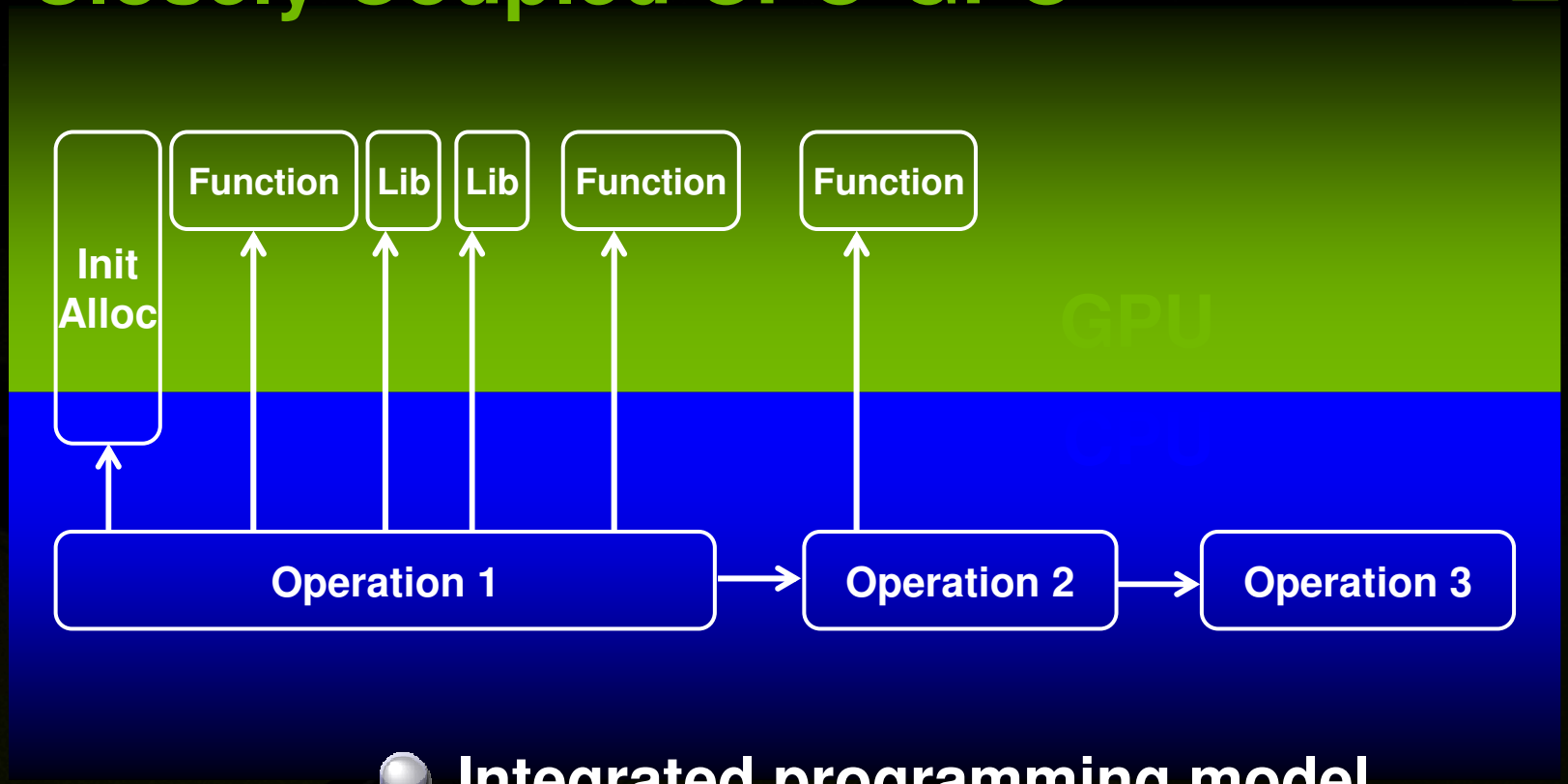
Faster is not “just Faster”

- 2-3X faster is “just faster”
  - Do a little more, wait a little less
  - Doesn’t change how you work
- 5-10x faster is “significant”
  - Worth upgrading
  - Worth re-writing (parts of) the application
- 100x+ faster is “fundamentally different”
  - Worth considering a new platform
  - Worth re-architecting the application
  - Makes new applications possible
  - Drives “time to discovery” and creates fundamental changes in Science

The GPU is a New Computation Engine



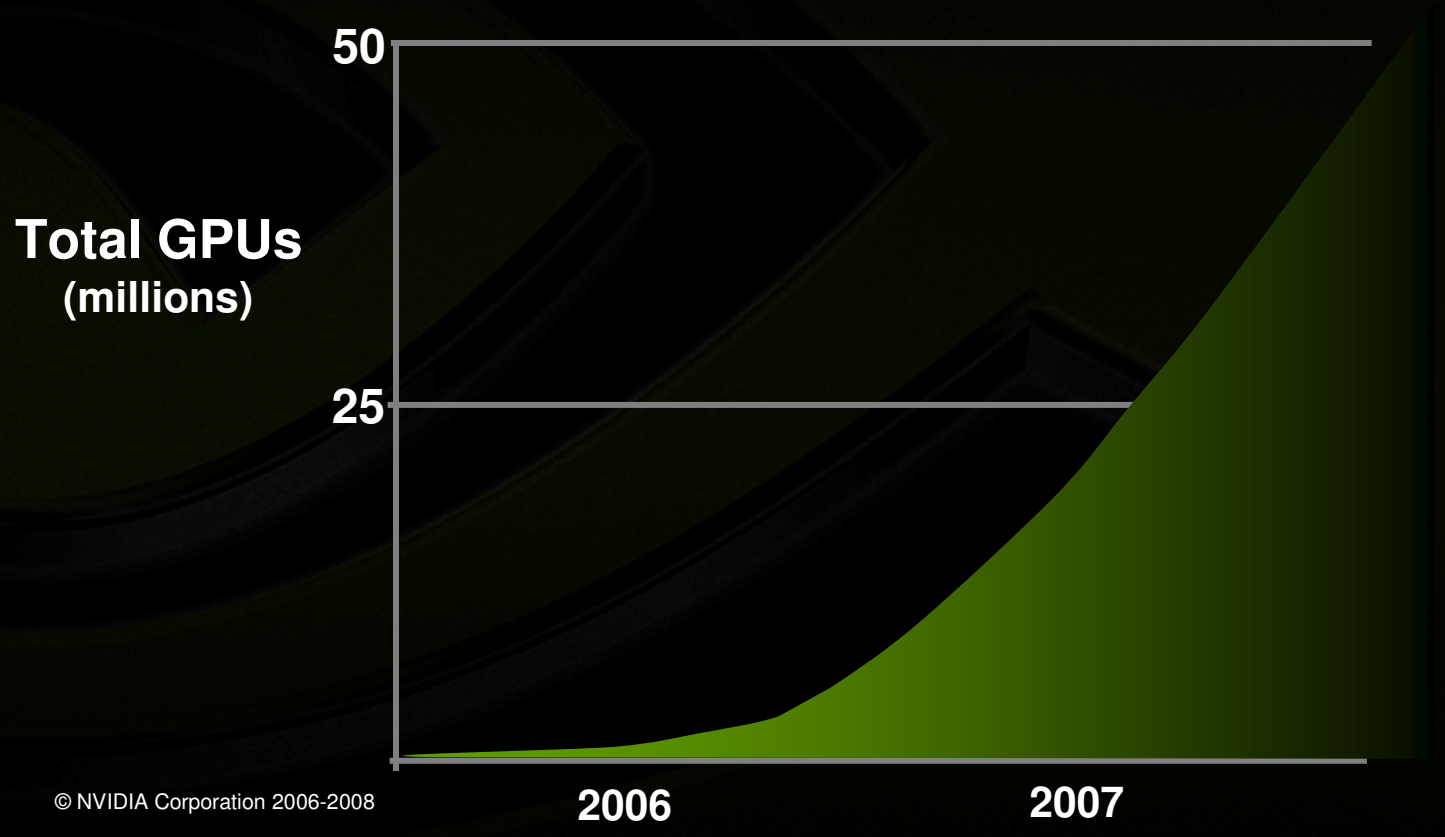
Closely Coupled CPU-GPU



- Integrated programming model
- High speed data transfer – up to 3.2 GB/s
- Asynchronous operation
- Large GPU memory systems

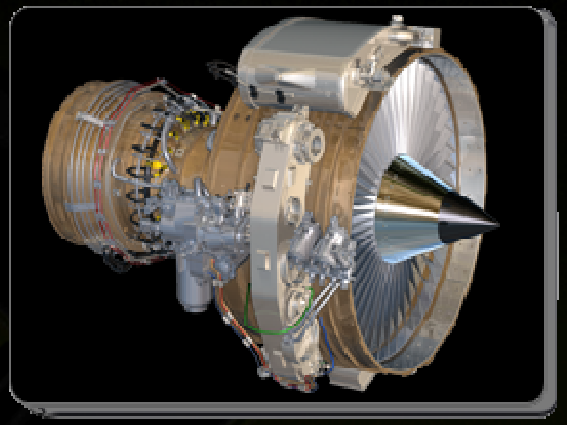
Millions of CUDA-enabled GPUs

- Dedicated computing
- C on the GPU
- Servers through Notebook PCs

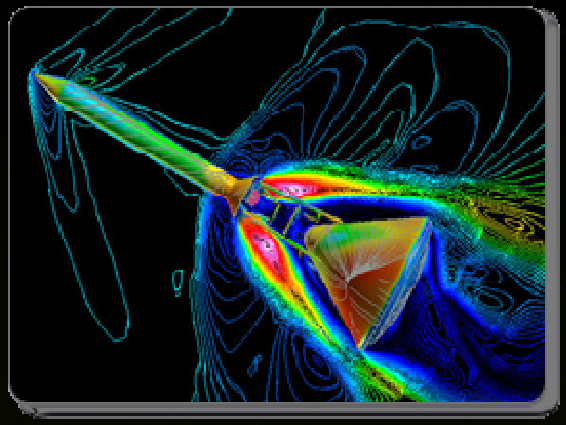


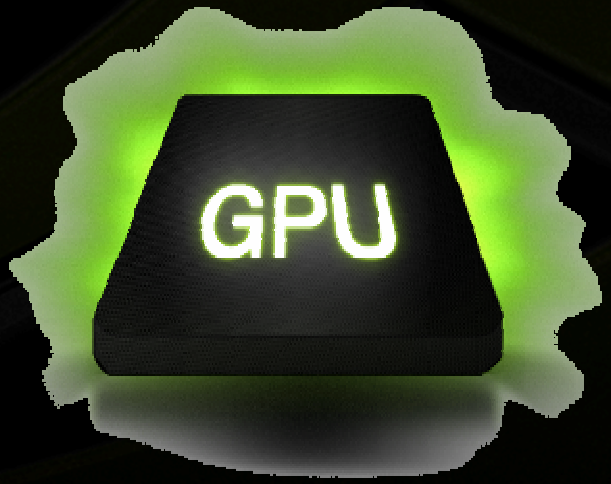
GeForce®
Entertainment

Quadro®
Design & Creation



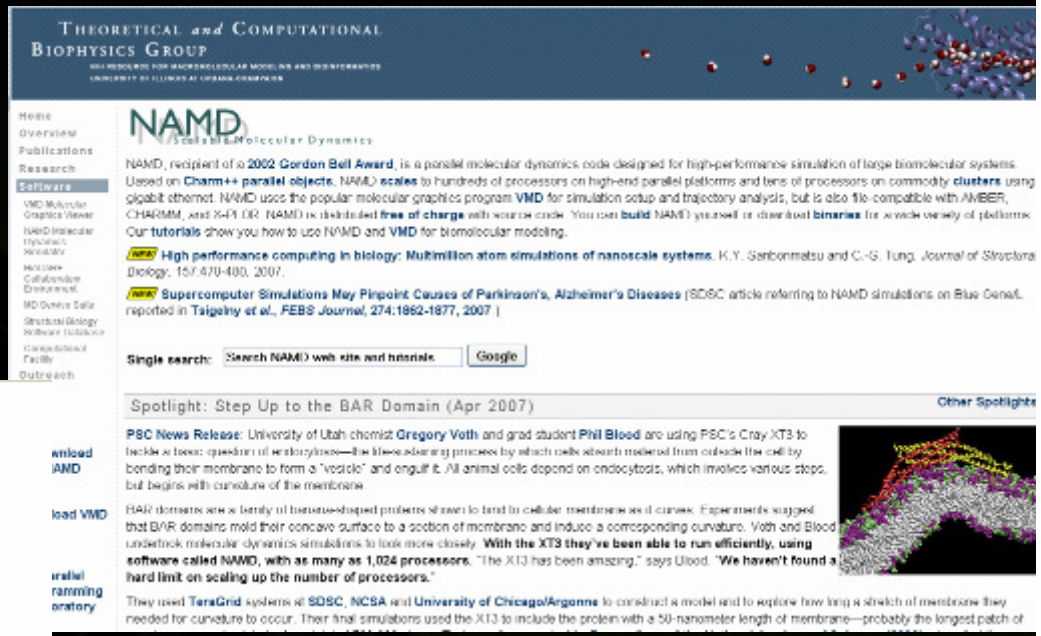
Tesla™
High Performance Computing





VMD/NAMD Molecular Dynamics

- 240X speedup
- Computational biology



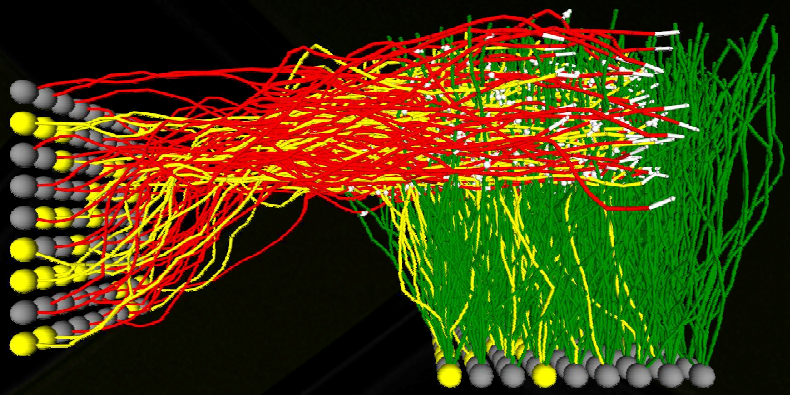
Parallel GPUs with Multithreading: 705 GFLOPS /w 3 GPUs

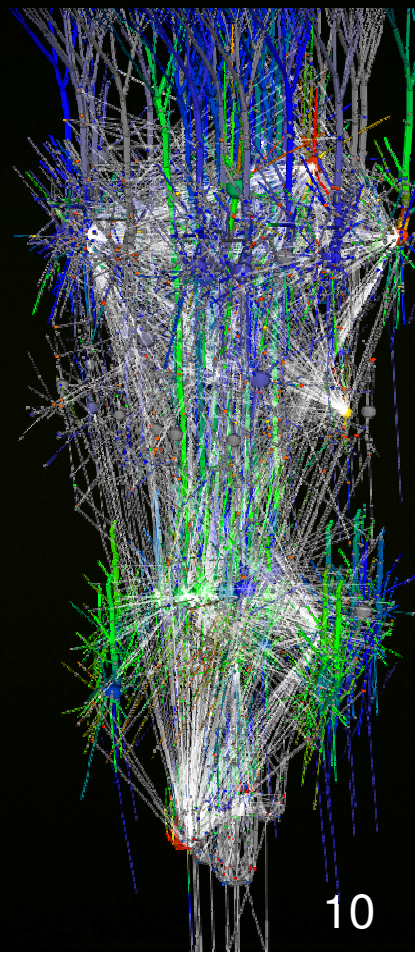
- One host thread is created for each CUDA GPU
- Threads are spawned and attach to their GPU based on their host thread ID
  - First CUDA call binds that thread's CUDA context to that GPU for life
  - Handling error conditions within child threads is dependent on the thread library and, makes dealing with any CUDA errors somewhat tricky, left as an exercise to the reader.... ☺
- Map slices are computed cyclically by the GPUs
- Want to avoid false sharing on the host memory system
  - map slices are usually much bigger than the host memory page size, so this is usually not a problem for this application
- Performance of 3 GPUs is stunning!
- Power: 3 GPU test box consumes 700 watts running flat out

http://www.ks.uiuc.edu/Research/vmd/projects/ece498/lecture/

EvolvedMachines

- Simulate the brain circuit
- Sensory computing: vision, olfactory
- 130X Speed up

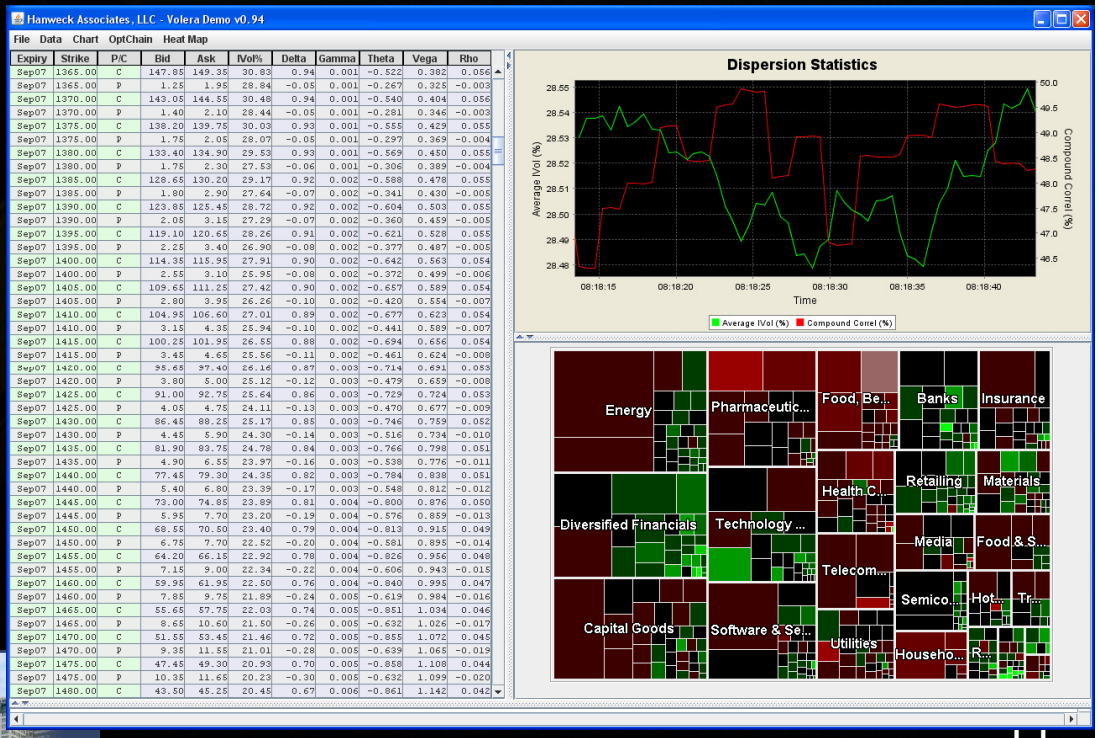




EvolvedMachines

Hanweck Associates

- VOLERA, real-time options implied volatility engine
- Accuracy results with SINGLE PRECISION
- Evaluate all U.S. listed equity options in <1 second



(www.hanweckassoc.com)

LIBOR APPLICATION:

Mike Giles and Su Xiaoke
Oxford University Computing Laboratory

- LIBOR Model with portfolio of swaptions
- 80 initial forward rates and 40 timesteps to maturity
- 80 Deltas computed with adjoint approach

	No Greeks		Greeks	
Intel Xeon	18.1s	-	26.9s	-
ClearSpeed Advance 2 CSX600	2.9s	6x	6.4s	4x
NVIDIA 8800 GTX	0.045s	400x	0.18s	149x

"The performance of the CUDA code on the 8800 GTX is exceptional"
-Mike Giles

Source codes and papers available at:
http://web.comlab.ox.ac.uk/oucl/work/mike.giles/hpc

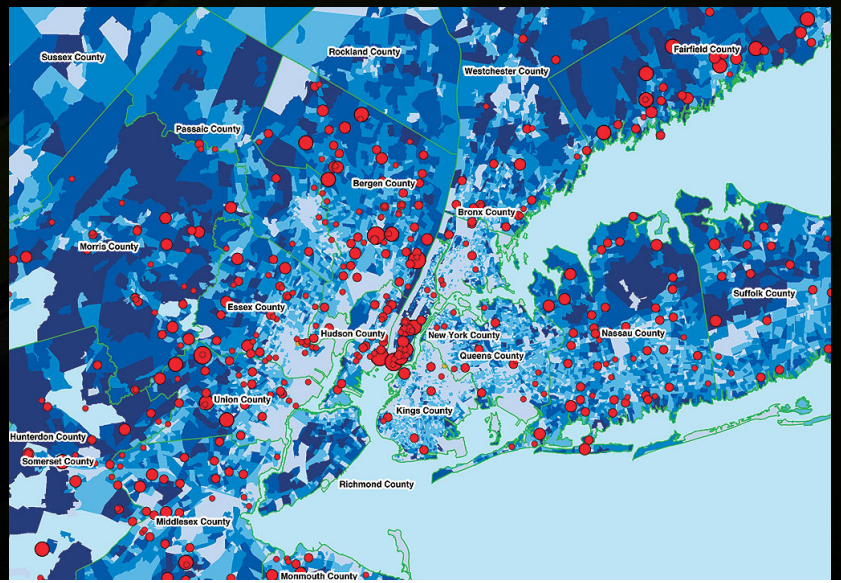
Manifold 8 GIS Application

From the Manifold 8 feature list:

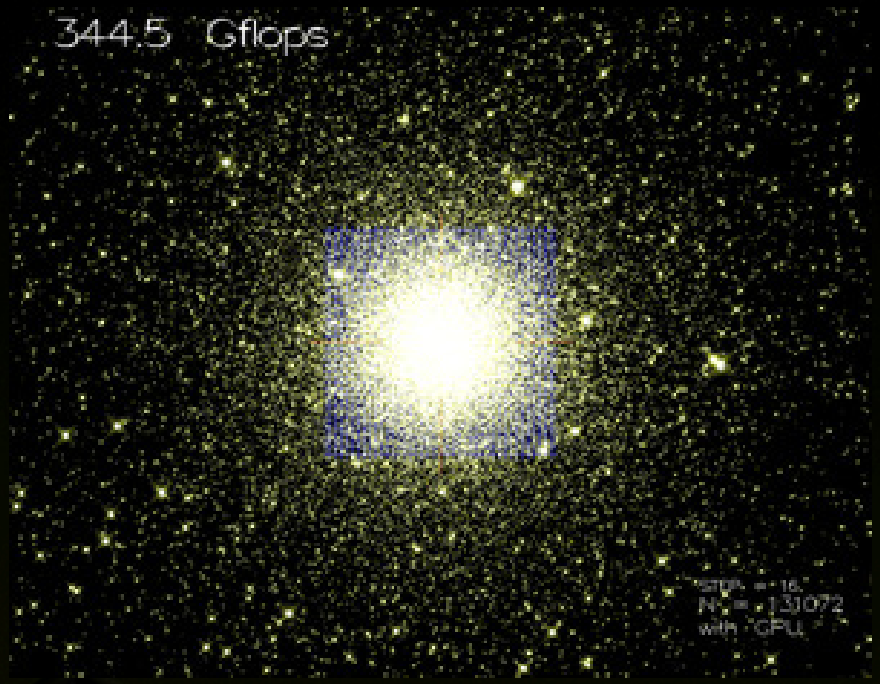
… applications fitting CUDA capabilities that might have taken **tens of seconds or even minutes** can be accomplished in **hundredths of seconds**. … **CUDA will clearly emerge to be the future of almost all GIS computing**

From the user manual:

"NVIDIA CUDA … could well be the most revolutionary thing to happen in computing since the invention of the microprocessor



nbody Astrophysics



Astrophysics research

1 GF on standard PC

300+ GF on GeForce 8800GTX

http://progrape.jp/cs/

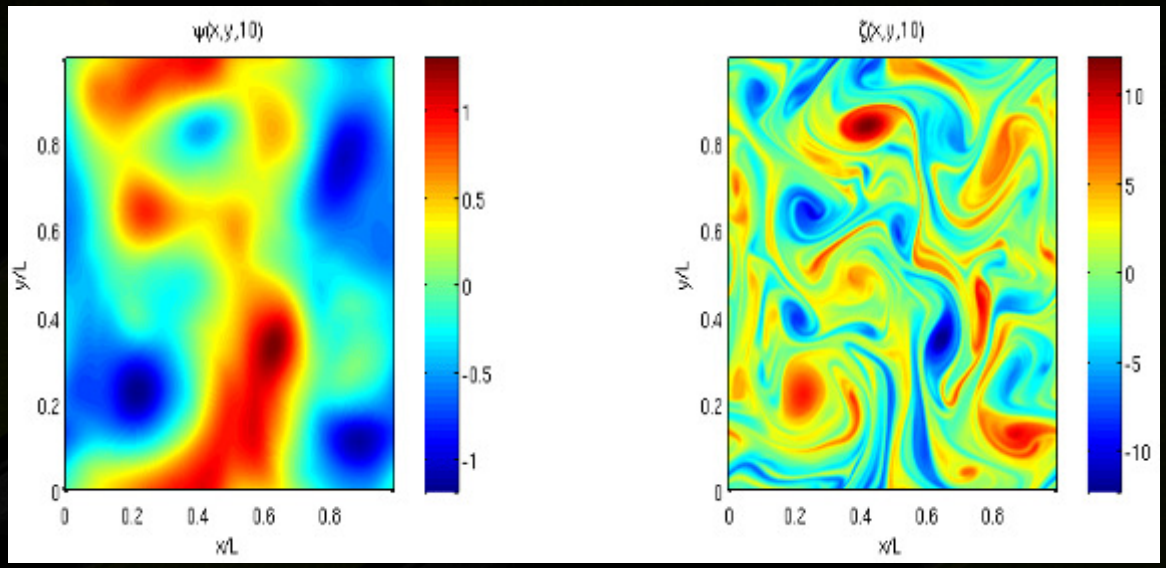
Faster than GRAPE-6Af custom simulation computer

Video demo

Matlab: Language of Science

17X with MATLAB CPU+GPU

http://developer.nvidia.com/object/matlab_cuda.html



Pseudo-spectral simulation of 2D Isotropic turbulence

http://www.amath.washington.edu/courses/571-winter-2006/matlab/FS_2Dturb.m

NVIDIA®

CUDA Programming Model Overview

GPU Computing

- GPU is a massively parallel processor
  - NVIDIA G80: 128 processors
  - Support thousands of active threads (12,288 on G80)
- GPU Computing requires a programming model that can efficiently express that kind of parallelism
  - Most importantly, data parallelism
- CUDA implements such a programming model

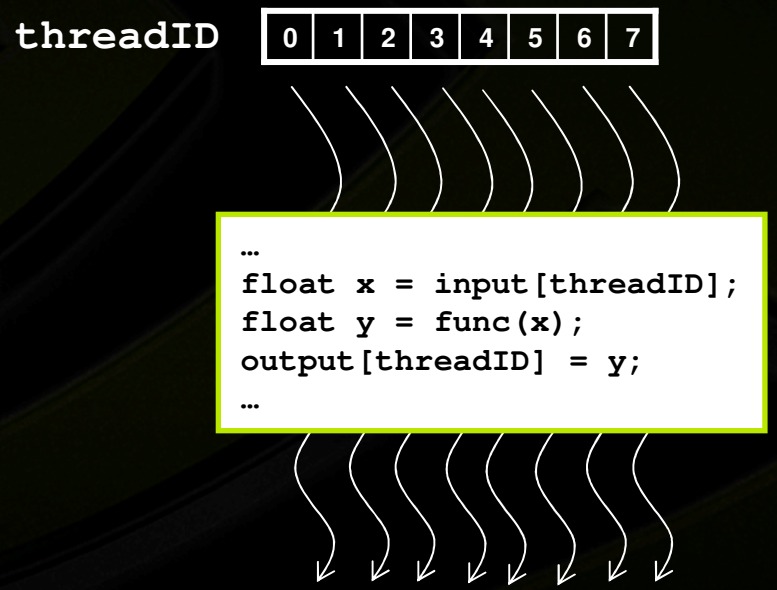
CUDA Kernels and Threads

- Parallel portions of an application are executed on the device as kernels
  - One kernel is executed at a time
  - Many threads execute each kernel
- Differences between CUDA and CPU threads
  - CUDA threads are extremely lightweight
    - Very little creation overhead
    - Instant switching
  - CUDA uses 1000s of threads to achieve efficiency
    - Multi-core CPUs can use only a few

Definitions:
Device = GPU; *Host* = CPU
Kernel = function that runs on the device

Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



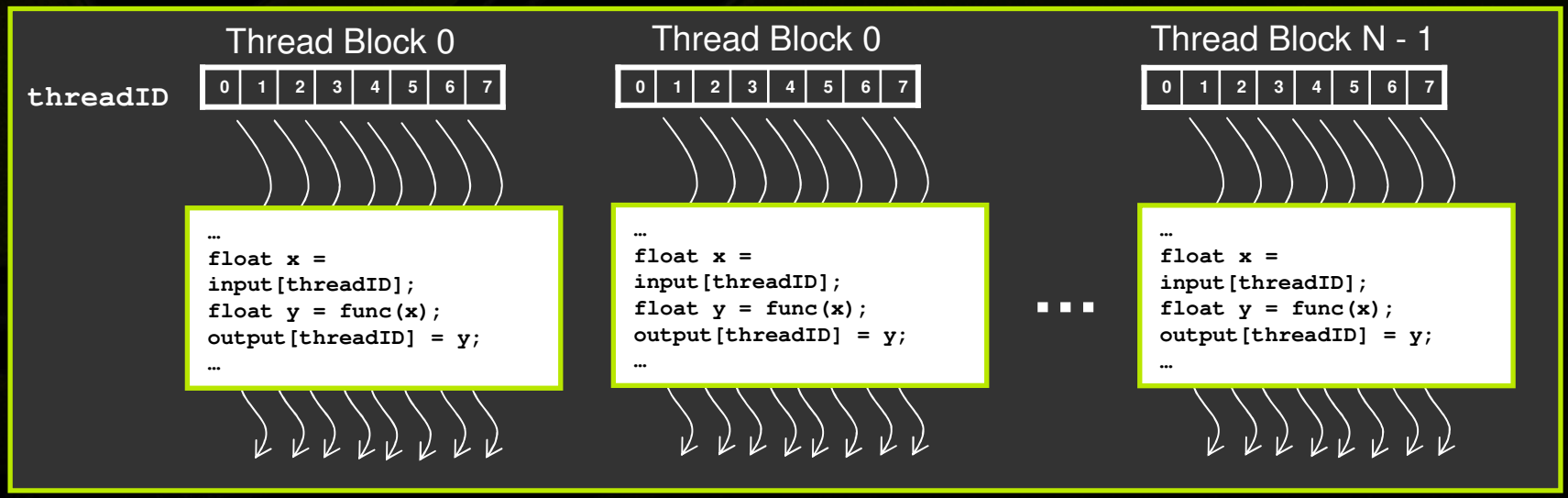
Thread Cooperation

- The Missing Piece: threads may need to cooperate
- Thread cooperation is valuable
  - Share results to save computation
  - Synchronization
  - Share memory accesses
    - Drastic bandwidth reduction
- Thread cooperation is a powerful feature of CUDA

Thread Blocks: Scalable Cooperation

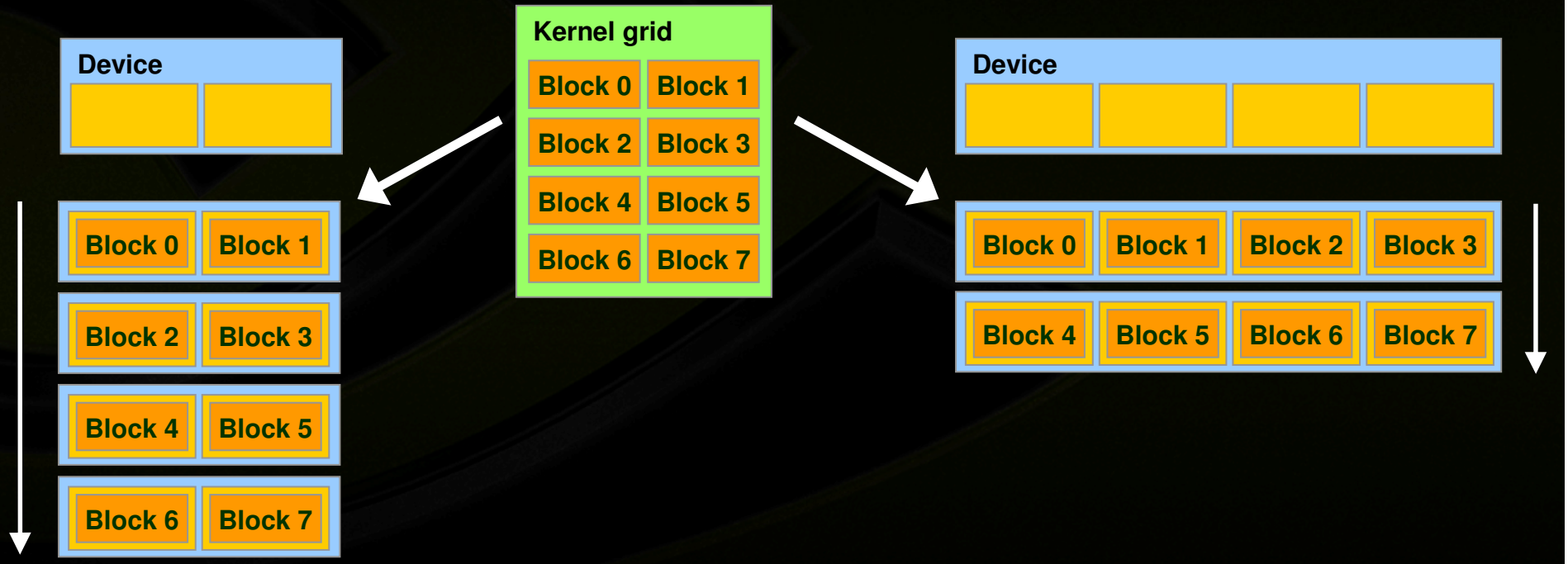
- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via **shared memory**
  - Threads in different blocks cannot cooperate

- Enables programs to **transparently scale** to any number of processors!



Transparent Scalability

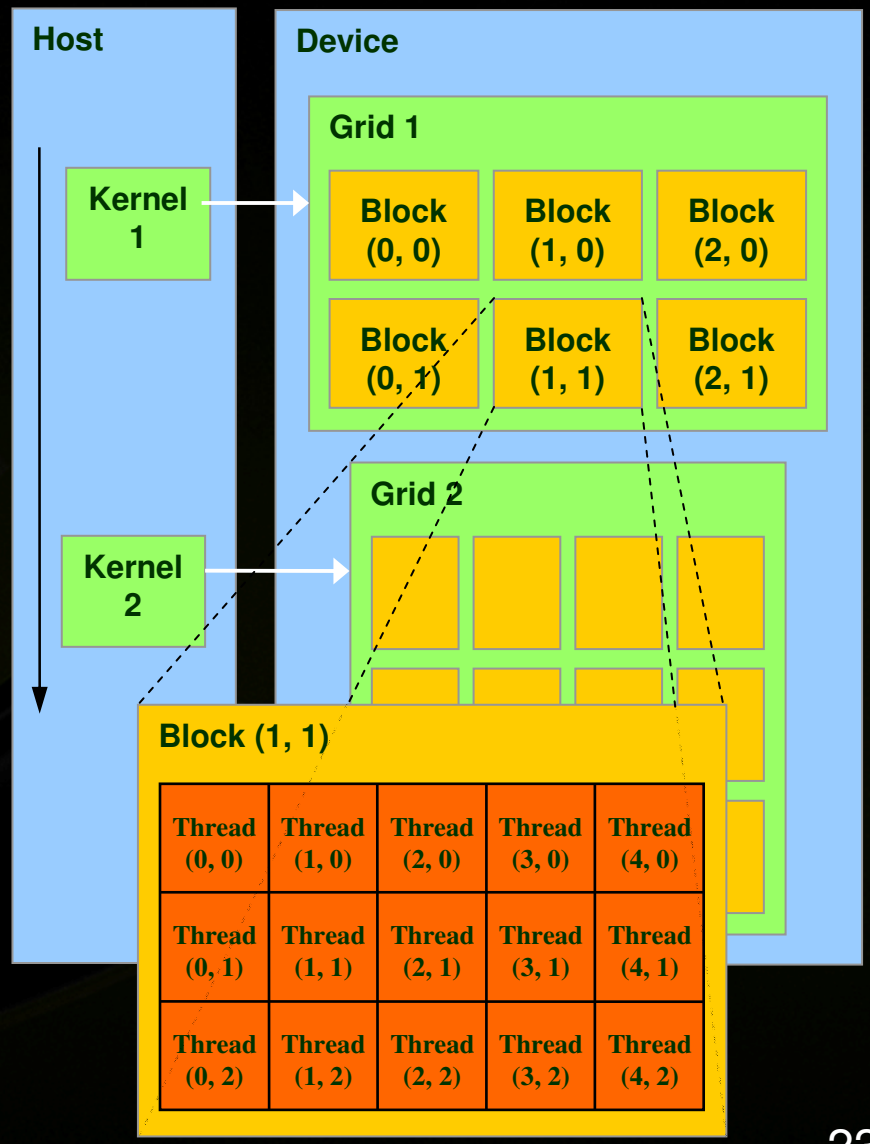
- Hardware is free to schedule thread blocks on any processor at any time
  - A kernel scales across any number of parallel multiprocessors



CUDA Programming Model

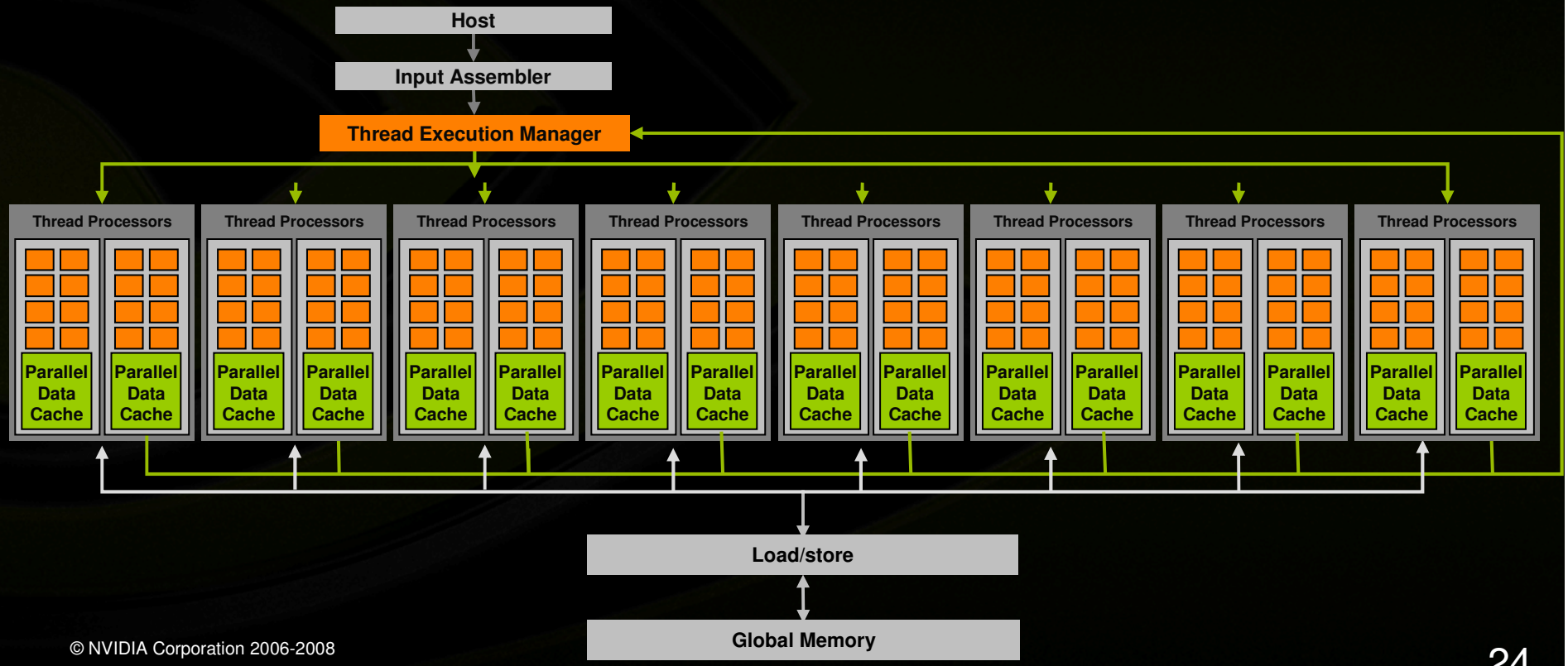
A kernel is executed by a grid of thread blocks

- A thread block is a batch of threads that can cooperate with each other by:
  - Sharing data through shared memory
  - Synchronizing their execution
- Threads from different blocks cannot cooperate



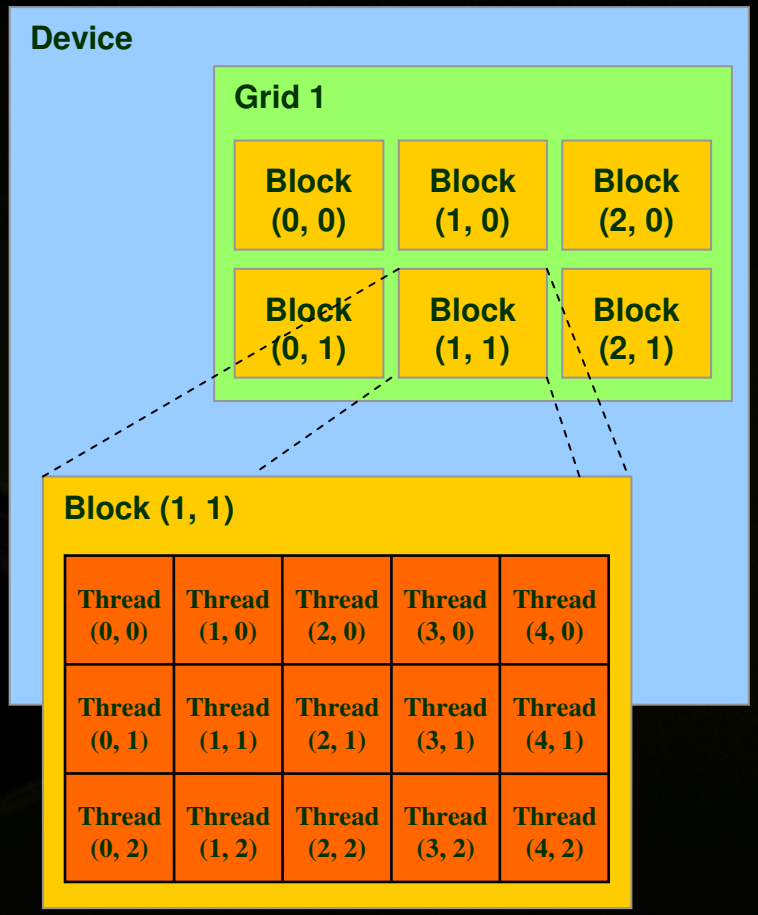
G80 Device

- Processors ■ execute computing threads
- Thread Execution Manager issues threads
- 128 Thread Processors grouped into 16 Multiprocessors (SMs)
- Parallel Data Cache (Shared Memory) enables thread cooperation



Thread and Block IDs

- Threads and blocks have IDs
  - Each thread can decide what data to work on
- Block ID: 1D or 2D
- Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multi-dimensional data
  - Image processing
  - Solving PDEs on volumes



Kernel Memory Access

- Registers
- Global Memory (external DRAM)
  - Kernel input and output data reside here
  - Off-chip, large
  - Uncached
- Shared Memory (Parallel Data Cache)
  - Shared among threads in a single block
  - On-chip, small
  - As fast as registers

- The host can read & write global memory but not shared memory

Execution Model

- Kernels are launched in grids
  - One kernel executes at a time
- A block executes on one Streaming Multiprocessor (SM)
  - Does not migrate
- Several blocks can reside concurrently on one SM
  - Control limitations (of G8X/G9X GPUs):
    - At most 8 concurrent blocks per SM
    - At most 768 concurrent threads per SM
  - Number is further limited by SM resources
    - Register file is partitioned among all resident threads
    - Shared memory is partitioned among all resident thread blocks

CUDA Advantages over Legacy GPGPU
(Legacy GPGPU is programming GPU through graphics APIs)

- Random access byte-addressable memory
  - Thread can access any memory location
- Unlimited access to memory
  - Thread can read/write as many locations as needed
- Shared memory (per block) and thread synchronization
  - Threads can cooperatively load data into shared memory
  - Any thread can then access any shared memory location
- Low learning curve
  - Just a few extensions to C
  - No knowledge of graphics is required
- No graphics API overhead

CUDA Model Summary

- Thousands of lightweight concurrent threads
  - No switching overhead
  - Hide instruction and memory latency
- Shared memory
  - User-managed L1 cache
  - Thread communication / cooperation within blocks
- Random access to global memory
  - Any thread can read/write any location(s)
- Current generation hardware:
  - Up to 128 streaming processors

Memory	Location	Cached	Access	Scope ("Who?")
Shared	On-chip	N/A	Read/write	All threads in a block
Global	Off-chip	No	Read/write	All threads + host

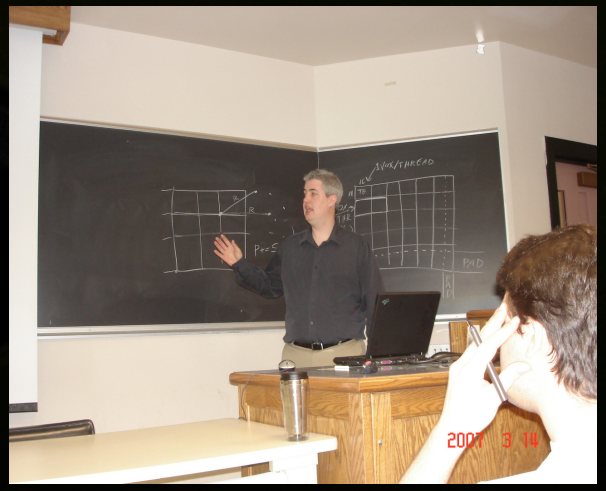
Getting Started

Get CUDA

- CUDA Zone: http://nvidia.com/cuda
  - Programming Guide and other Documentation
  - Toolkits and SDKs for:
    - Windows
    - Linux
    - MacOS
  - Libraries
  - Plugins
  - Forums
  - Code Samples

Come visit the class!

- UIUC ECE498AL – Programming Massively Parallel Processors (http://courses.ece.uiuc.edu/ece498/al/)
  - David Kirk (NVIDIA) and Wen-mei Hwu (UIUC) co-instructors
  - CUDA programming, GPU computing, lab exercises, and projects
  - Lecture slides and voice recordings



Questions?