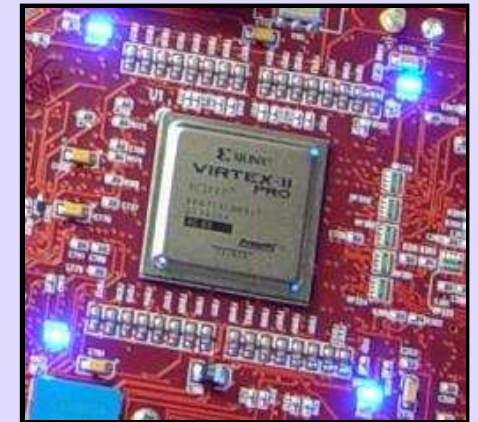


# Reconfigurable Computing

Adapting Computational Kernels to  
Field-Programmable Gate Arrays

Craig Ulmer

[cdulmer@sandia.gov](mailto:cdulmer@sandia.gov)



December 7, 2005

Adrian Javelo

Craig Ulmer and David Thompson

Keith Underwood and K. Scott Hemmert

UCLA

SNL/CA

SNL/NM



# Outline

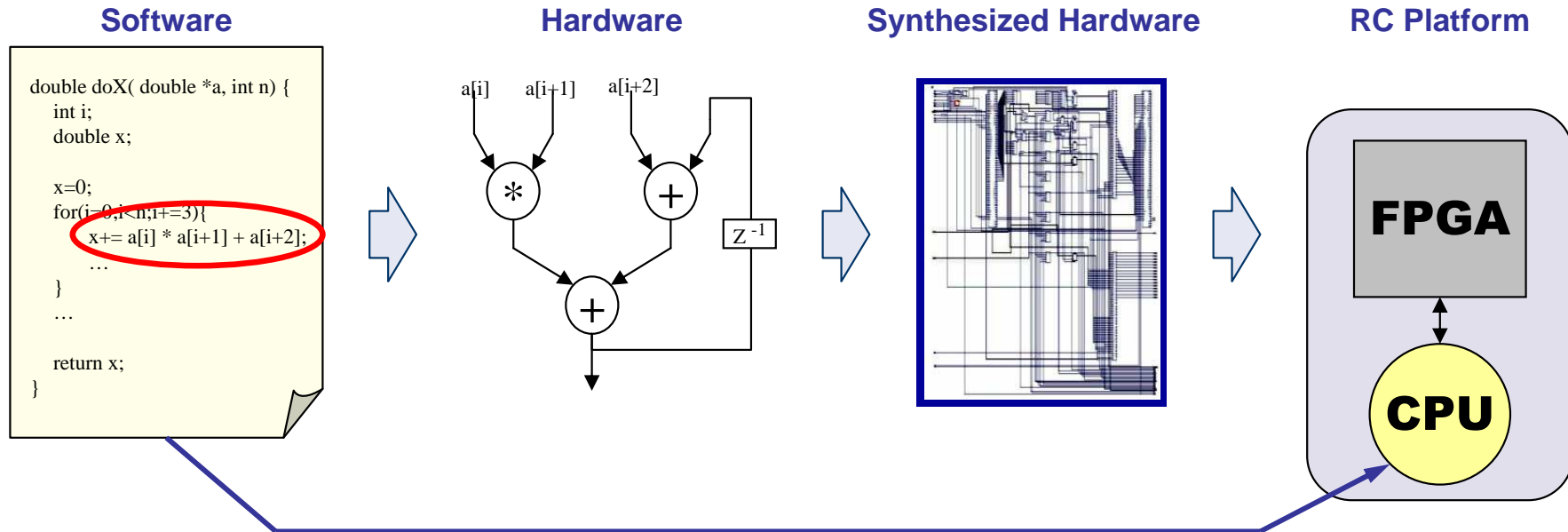
- Reconfigurable Computing
  - *LDRD progress update*
- Ray-Triangle Intersection Algorithm
  - *Porting computational kernel to hardware*
- Automation
  - *Replacing your intern with a Perl script*
- Summary



# LDRD Progress Update

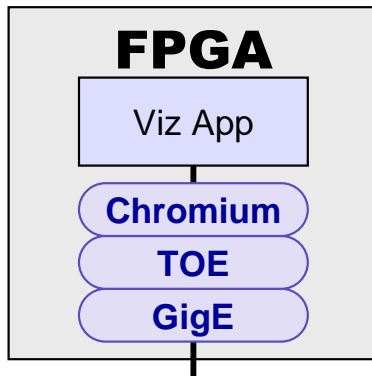
# In a nutshell..

- *Reconfigurable Computing*
  - Use FPGAs as a means of accelerating simulations
  - Implement key computations in hardware: soft-hardware

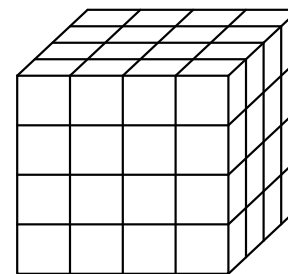


# LDRD Sub-projects

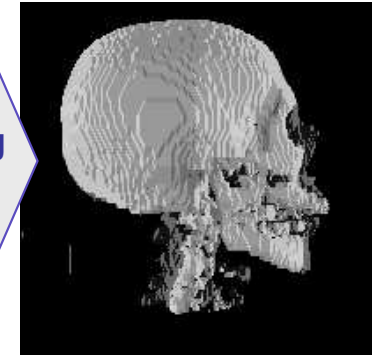
## Graphics Transport



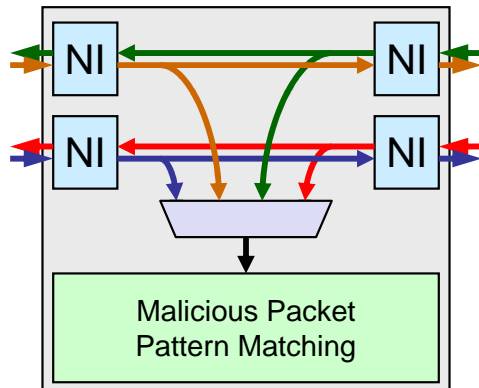
## Isosurfacing



Marching  
Cubes



## Network Intrusion Detection



## Cray XD1

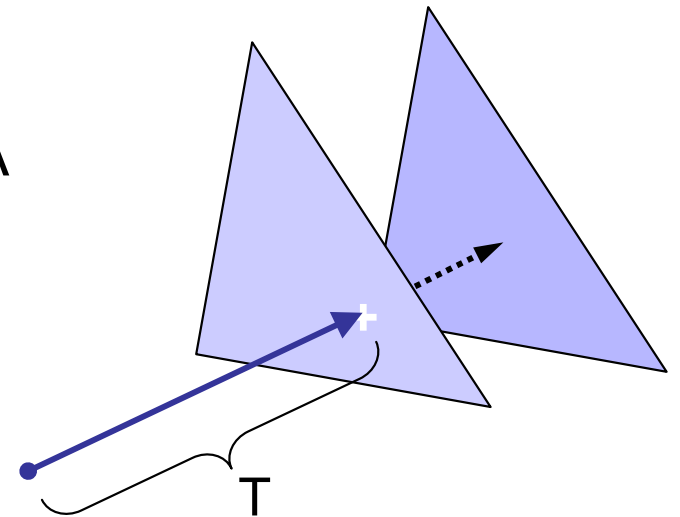




# Ray-Triangle Intersection Algorithm

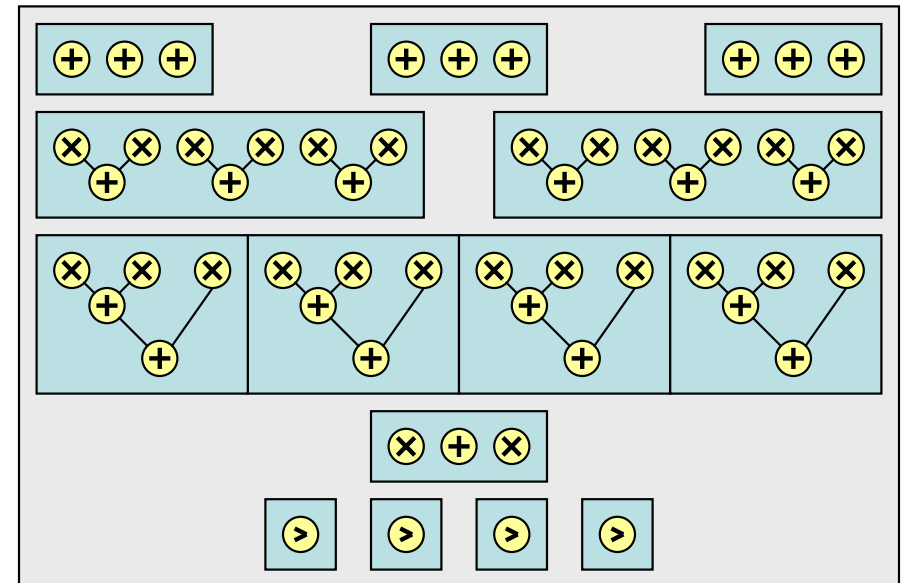
# Overview

- Interested in observing hardware design tradeoffs w/ FPGAs
  - Only recently has floating-point become practical
- Graphics algorithms often need to find Ray-Triangle intersection
  - Example: Use to find closest triangle to a point
  - Photon Mapping
- Adrian Javelo: Summer intern from UCLA
  - Implement algorithm in hardware
  - ..under realistic assumptions



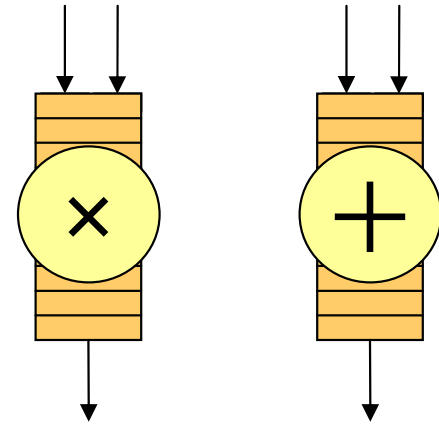
# Ray-Triangle Intersection Algorithm

- Moller and Trumbore algorithm (1997)
  - TUV intersection point
- Modified to remove division
  - 24 Adds
  - 26 Multiplies
  - 4 Compares
  - 15 Delays
  - 17 Inputs
  - 2 Outputs (+4 bits)



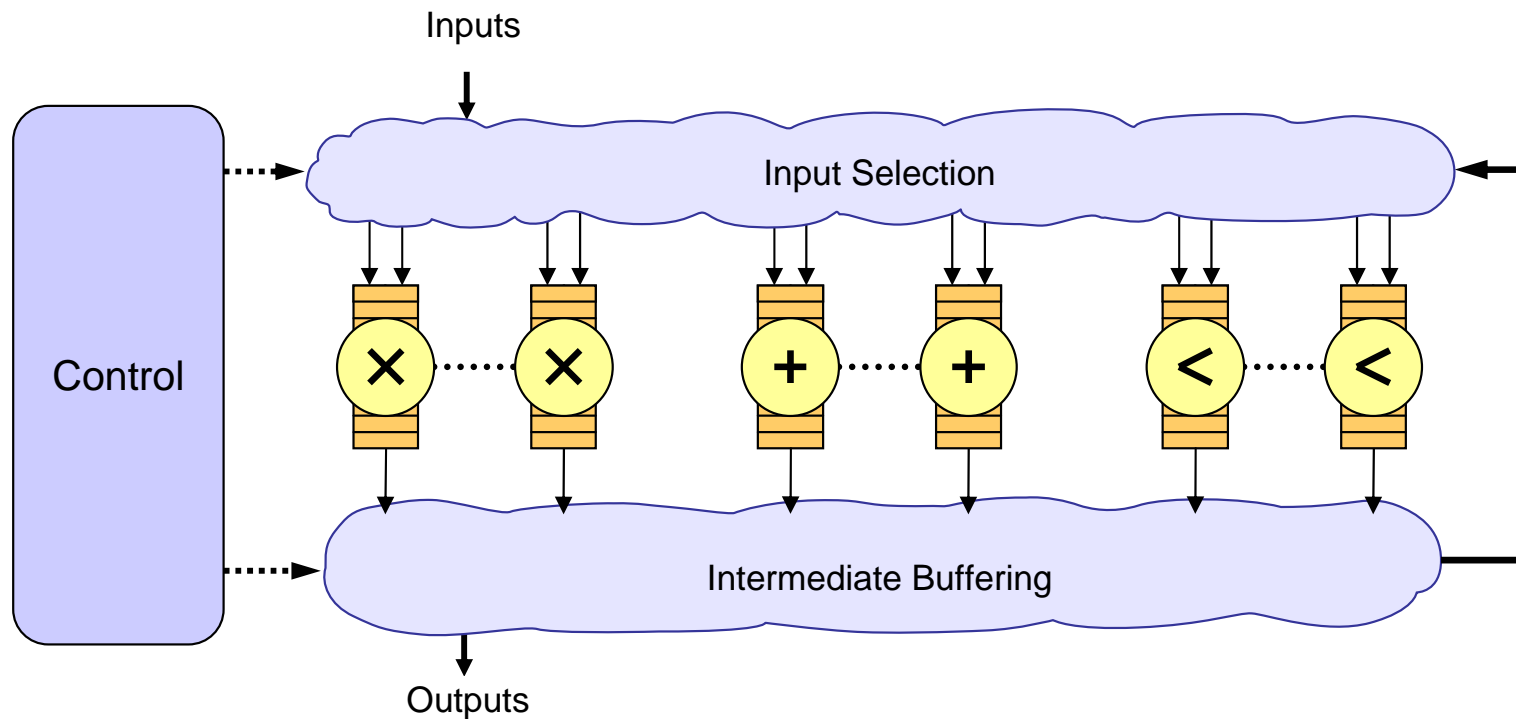
# Floating-Point Affects Design

- Ideal Case: Continuous pipeline
  - Possible with larger FPGAs
- SNL/NM: FP units for FPGAs
  - Big (~20 DP or ~70 SP in V2P50)
  - 10-17 stage pipelines
  - 130-200MHz
- Practical Case: Limited hardware
  - Assume 5 adders, 6 multipliers, 4 comparators
  - Reuse units, schedule data flow
  - Use strip mining to remove latency issue



# Hardware Data-Path

- Build wrapper around an array of FP units



# Scheduling: Single Iteration

	Adders					Multipliers						Comparators			
Stage	A0	A1	A2	A3	A4	M0	M1	M2	M3	M4	M5	C0	C1	C2	c3
0	T1-T0	T1-T0	T1-T0	T2-T0	T2-T0										
1	T2-T0	PO-T0	PO-T0	PO-T0											
2						PDxS2	PDxS2	PDxS2	PDxS2	PDxS2	PDxS2				
3	PDxS2	PDxS2	PDxS2			S2xS0	S2xS0	S2xS0	S2xS0	S2xS0	S2xS0				
4	S2xS0	S2xS0	S2xS0												
5						S2*C0	S2*C0	S2*C0	PD*C1	PD*C1	PD*C1				
6	S2*C0	PD*C1				S0*C0	S0*C0	S0*C0	S1*C1	S1*C1	S1*C1				
7	S2*C0	PD*C1	S0*C0	S1*C1											
8	S0*C0	S1*C1	U+V												
9						T*Dold	T*Dold								
10												U < O	V < O	UV > D	M0 < M1
11												OUT	OUT	OUT	OUT

40%

36%

# Scheduling: Two Iterations

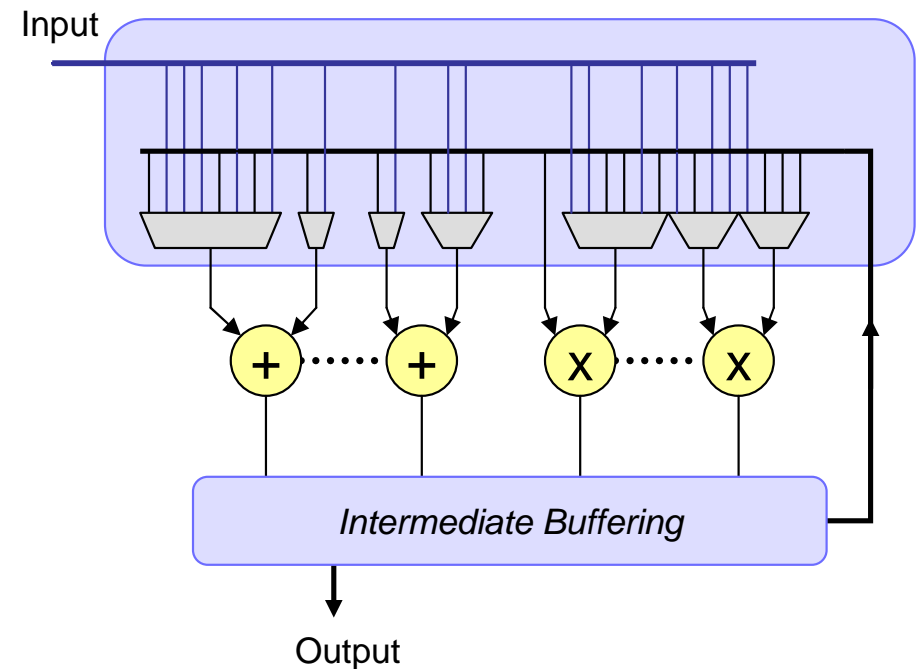
	Adders					Multipliers						Comparators			
Stage	A0	A1	A2	A3	A4	M0	M1	M2	M3	M4	M5	C0	C1	C2	c3
0	T1-T0	T1-T0	T1-T0	T2-T0	T2-T0	T*Dold	T*Dold					OUT	OUT	OUT	OUT
1	T2-T0	PO-T0	PO-T0	PO-T0								U < O	V < O	UV > D	M0 < M1
2	T1-T0	T1-T0	T1-T0	T2-T0	T2-T0	PDxS2	PDxS2	PDxS2	PDxS2	PDxS2	PDxS2	OUT	OUT	OUT	OUT
3	T2-T0	PO-T0	PO-T0	PO-T0		S2xS0	S2xS0	S2xS0	S2xS0	S2xS0	S2xS0				
4	PDxS2	PDxS2	PDxS2			PDxS2	PDxS2	PDxS2	PDxS2	PDxS2	PDxS2				
5	S2xS0	S2xS0	S2xS0			S2xS0	S2xS0	S2xS0	S2xS0	S2xS0	S2xS0				
6			PDxS2	PDxS2	PDxS2	S2*C0	S2*C0	S2*C0	PD*C1	PD*C1	PD*C1				
7	S2*C0	PD*C1	S2xS0	S2xS0	S2xS0	S0*C0	S0*C0	S0*C0	S1*C1	S1*C1	S1*C1				
8	S2*C0	PD*C1	S0*C0	S1*C1		S2*C0	S2*C0	S2*C0	PD*C1	PD*C1	PD*C1				
9	S0*C0	S1*C1	U+V	S2*C0	PD*C1	S0*C0	S0*C0	S0*C0	S1*C1	S1*C1	S1*C1				
10	S2*C0	PD*C1	S0*C0	S1*C1		T*Dold	T*Dold								
11	S0*C0	S1*C1	U+V									U < O	V < O	UV > D	M0 < M1

80%

72%

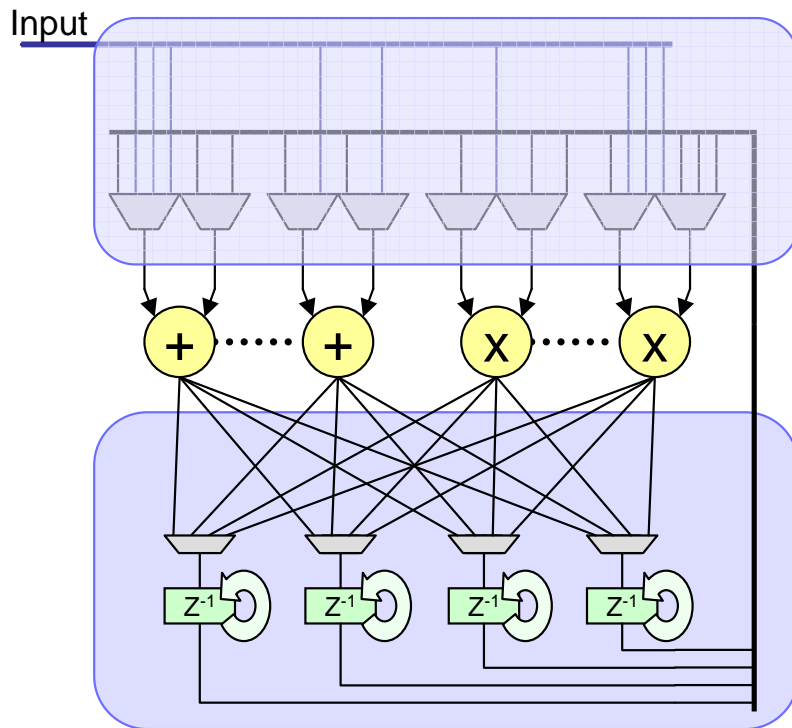
# Input Selection Hardware

- Assign operations to units
  - Each stage: M choose N
  - Affects input selection hardware
- First-Come-First-Serve
  - Input ports: 1-7 sources
  - More sources = More HW
  - Performance based on worst
- Balancing the Schedule
  - Ops assigned based on inputs
  - Input ports: 3-6 sources



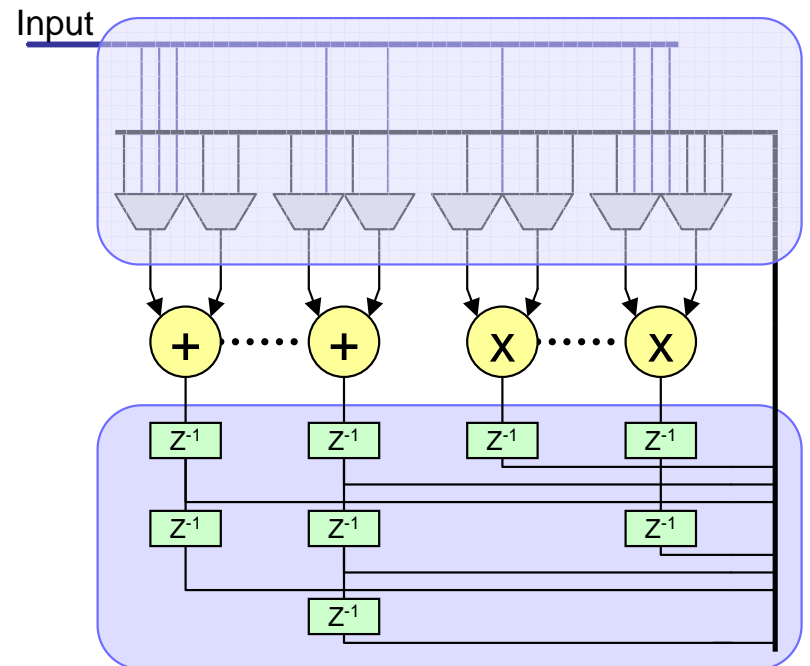
# Buffering Intermediate Values

## Buffer Reuse



*Minimize number of buffers*  
40 Memories, 40 MUXs

## Chaining



*Minimize control logic*  
81 Memories, 0 MUXs



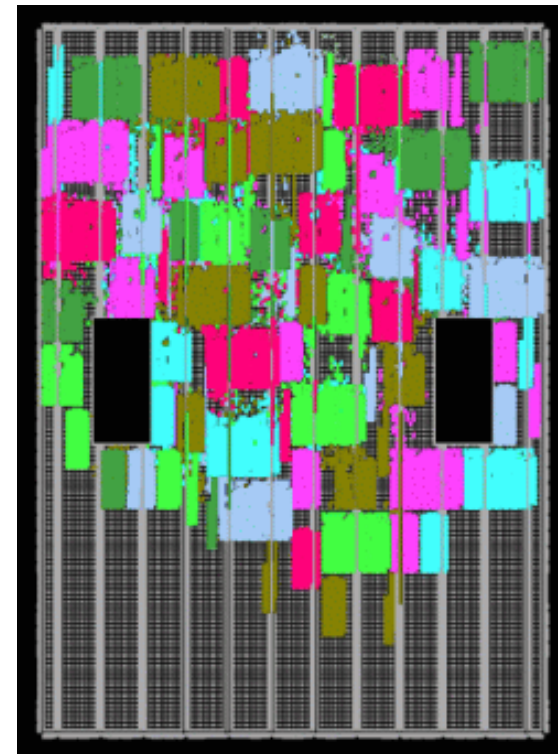
# Synthesis Results

- Synthesized three designs for V2P50 FPGA
  - Memory blocks implemented more compactly than MUXs
  - Simplified approach works best

Design	Build Time	V2P50 Area	Clock Rate	Performance
Buffer Reuse	2 hours	35%	125 MHz	1.04 GFLOPS
Chaining	30 minutes	29%	133 MHz	1.16 GFLOPS
Hybrid	30 minutes	28%	126 MHz	1.05 GFLOPS

# Alternate: Full Pipeline

- Implemented complete pipeline
  - Operations: 37
  - Pipeline Stages: 62
  - In-flight values: 564
  - FP Utilization: 100%
- Virtex II/Pro 50 (V2P50)
  - 57 % Area, 132 MHz
  - One day to design
  - 4 hours to compile
  - 4.8 GFLOPS
- Opteron: ~1 GFLOPS





# Current Direction: Automation

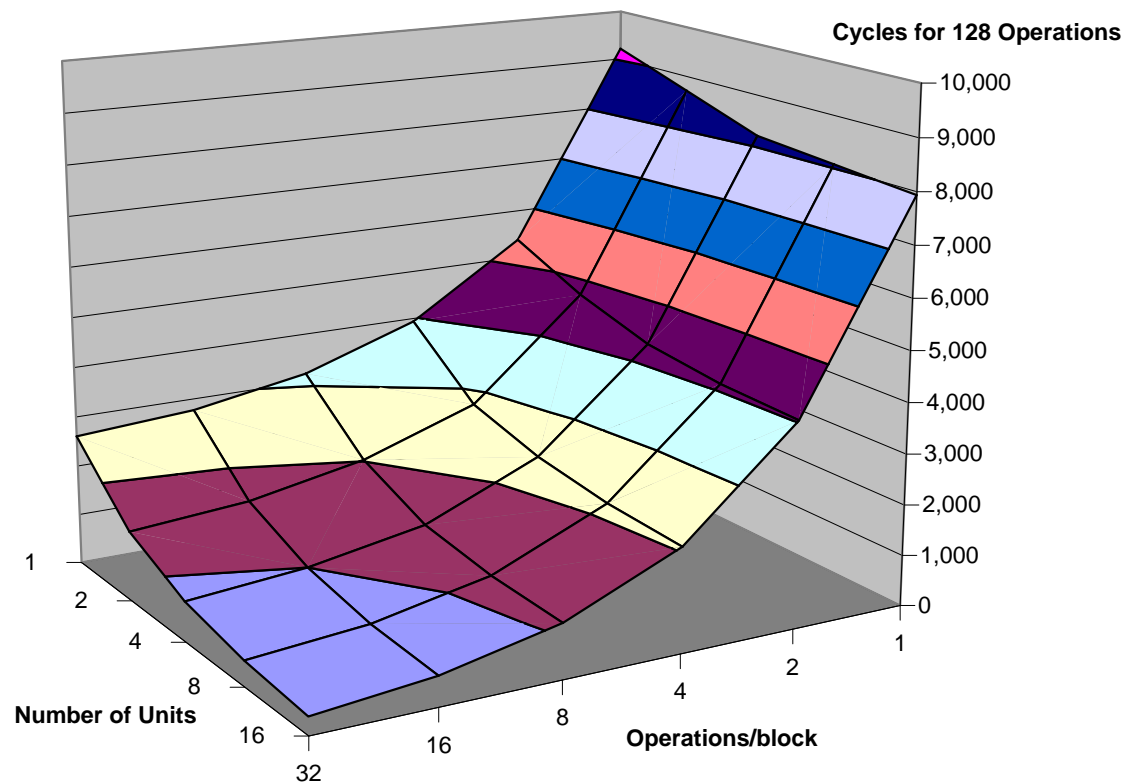


# Automating the Process

- Considerable amount of design/debug time
  - Schedule adjustments = redesign hardware
- Perl script
  - Translate schedule into hardware state machines
- Compiler tools
  - DFG: Build a dataflow graph from operations
  - Schedule: Find schedule that minimizes delay and buffers
  - Map: Assign ops to units, balancing inputs
  - VHDL: Generate synthesizable hardware

# Example: Resources vs. Performance

- Ray-Triangle Intersect
  - w/o strip mining
- Vary
  - Number of FP units
  - Operations/block
- Tradeoff
  - Depends on parallelism



# Concluding Remarks

- Reusing FP units enables FPGAs to process larger kernels
  - Apply traditional scheduling tricks to increase utilization
  - Algorithm shape affects performance
  - Simplicity wins
- Simple DFG tools go a long ways
  - Easy to adjust parameters, generate hardware
  - Interested in optimization advice or suggestions

