Reconfigurable Computing

Trends and Exploration

In wisdom gathered over time I have found that every experience is a form of exploration.
- Ansel Adams

Philip Leong (philip.leong@sydney.edu.au)
School of Electrical and Information Engineering

http://phwl.org/talks

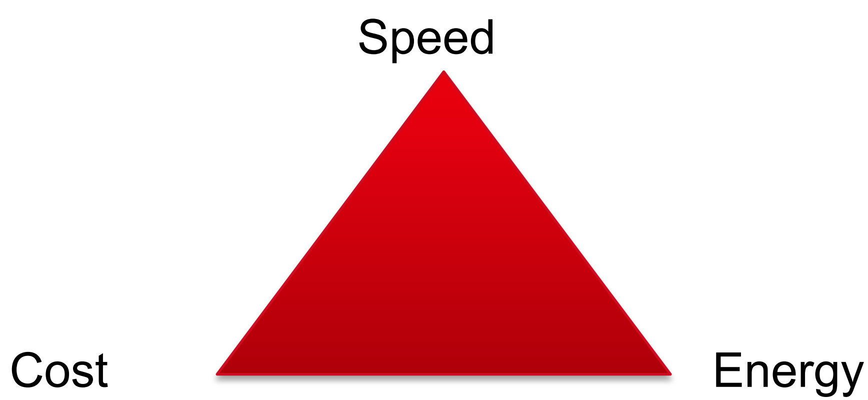
THE UNIVERSITY OF SYDNEY

Introduction

- How do we measure performance?
- What tools can we use to explore a design space?
- What is the impact of VLSI technology on FPGA design?
- Technology trends influence architecture. Can we understand how they change with time?
- Case study
  - Matrix multiplication

Performance

- Understand what needs to be optimised (and what doesn't)
- Tradeoff between speed, area, latency, throughput, energy, cost, accuracy …
  - Cannot optimise them all, e.g. usually can increase speed if cost unimportant
- Good design is a tradeoff

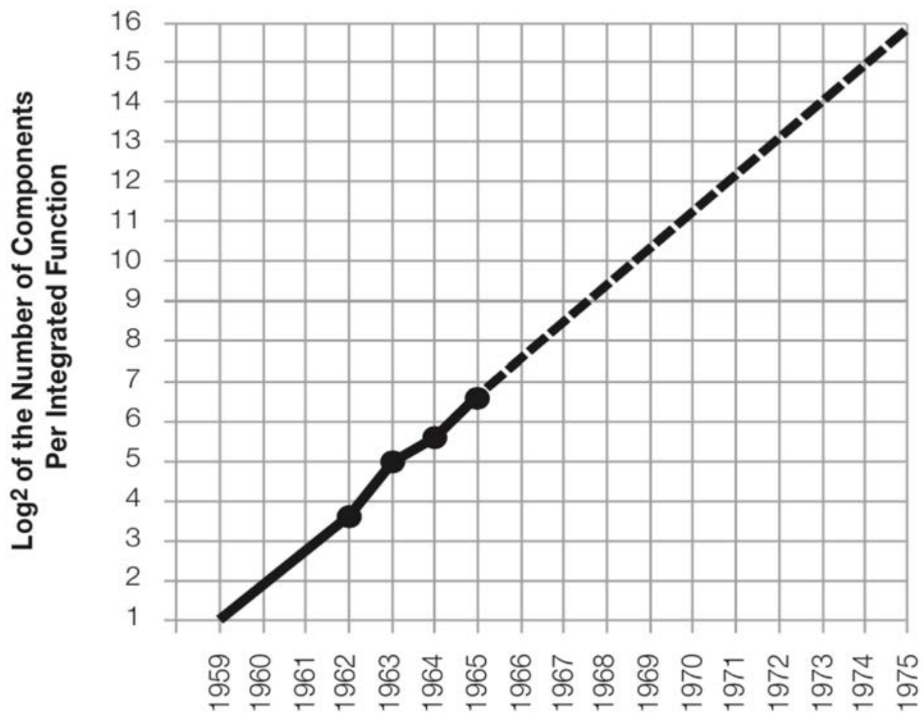


Area

THE UNIVERSITY OF SYDNEY

Moore's Law [1]

- Gordon Moore in 1965 predicted number of transistors in an IC will double ≈ two years
- This has driven the semiconductor industry for many decades
- Made FPGAs practical (first commercial FPGA XC2064 which had 64 CLBs with 2x 3-LUTs per CLB)



[1] G. E. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, April 1965

Moore's Law [1]

- He made the bold claim that 65,000 components could fit on an IC by 1975 (at the time they had 50)!
- Cartoon is from the same paper

[1] G. E. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, April 1965

Year	Feature Size	Xilinx FPGA family		Device	LUTs	DSP/Mult blocks	BRAM Kbits	LUTs /DSP	LUTs /BRAM	Altera FPGA family		Device	ALMs (LEs)	DSP /Mult blocks	BRAM Kbits	LEs/ DSP	LEs/ BRAM
2011	28 nm	Virtex 7	V	XC7V2000T	1,221,600	2,160	46,512	566	26								
			VX	XC7VX1140T	712,000	3,600	67,680	198	11								
			VH	XC7VH870T	547,600	2,520	50,760	217	11								
2010										Stratix V	GT	5SGTC7	622,000	512	50,000	1,215	12
											GX	5SGXBB	952,000	704	52,000	1,352	18
											GS	5SGSD8	695,000	3,926	50,000	177	14
											E	5SEEB	952,000	704	52,000	1,352	18
2009	40 nm	Virtex 6	LX	XC6VLX760	474,240	864	25,920	549	18								
			SX	XC6VSX475T	297,600	2,016	38,304	148	8								
			HX	XC6VHX565T	354,240	864	32,832	410	11								
2008										Stratix IV	GT	EP4S100G5	531,200	1,024	27,376	519	19
											GX	EP4SGX530	531,200	1,024	27,376	519	19
											E	EP4SE820	813,050	960	33,294	847	24
2006	65 nm	Virtex 5	LX	XC5VLX330	207,360	192	10,368	1,080	20								
			SX	XC5VSX240T	149,760	1,056	18,576	142	8								
			FX	XC5VFX200T	122,880	384	16,416	320	7								
										Stratix III	L	EP3SL340	337,500	576	16,272	586	21
											E	EP3SE260	255,000	768	14,688	332	17
2005	90 nm									Stratix II	GX	EP2SGX130/G	132,540	252	6,747	526	20
	130 nm											EP2S180	179,400	384	9,383	467	19
2004	90 nm	Virtex 4	LX	XC4VLX200	178,176	96	6,048	1,856	29								
			SX	XC4VSX55	49,152	512	5,760	96	9								
			FX	XC4VFX140	126,336	192	9,936	658	13								
2002	130 nm									Stratix	GX	EP1SGX40D	41,250	56	3,423	737	12
												EP1S80	79,040	88	7,428	898	11
2001	130 nm	Virtex II	Pro	XC2VP100	88,192	444	7,992	199	11								
			Pro X	XC2VPX70	66,176	308	5,544	215	12								
	0.15 um		V	XC2V8000	93,184	168	3,024	555	31	Mercury		EP1M350	14,400	0	115	-	125
2000	0.18 um									Excalibur		EPXA10	38,400	0	3,146	-	12
1999	0.18 um	Virtex E		XCV3200E	64,896	0	851	-	76								
1998	0.22 um									Flex 10KE		EPF10K200E	9,984	0	98	-	102
	0.25 um	Virtex		XCV1000	24,576	0	131	-	188								
1997	0.35 um	4000 E/XL		XC4085XL	12,544	0	0	-	-								
1996	0.3 um									Flex 10KA		EPF10K250A	12,160	0	41	-	297
1995	0.42 um									Flex 10K		EPF10K100	4,992	0	25	-	200
1992	0.6 um									Flex 8000		EPF81500A	1,296	0	0	-	-
1991	0.8um	4000 series		XC4025	2,048	0	0	-	-								
1985	2 um	2000 series		XC2018	400	0	0	-	-								

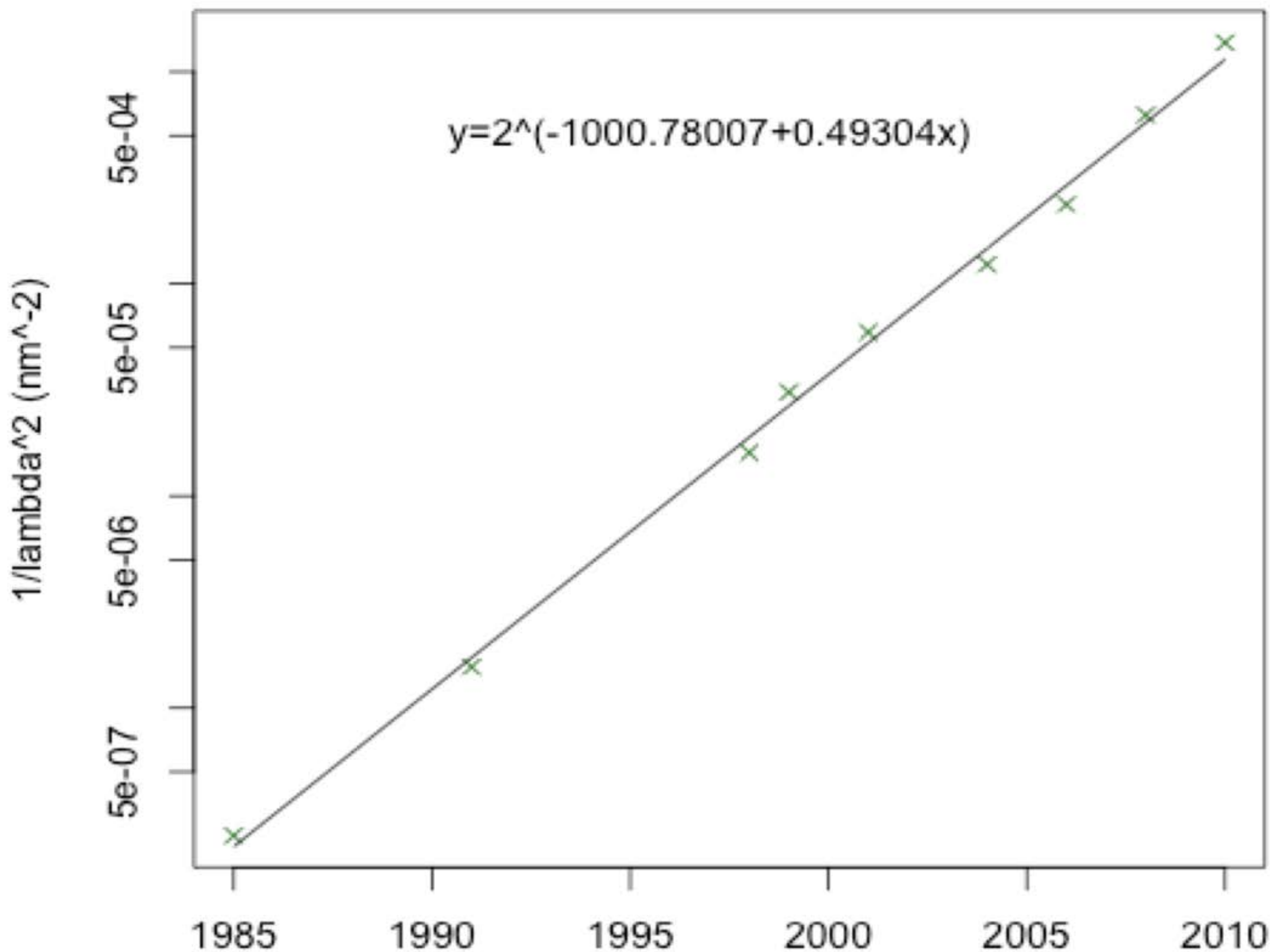
from [2]

Question

- If x is the year and y is the number of transistors on an integrated circuit, give an equation to model Moore's Law.

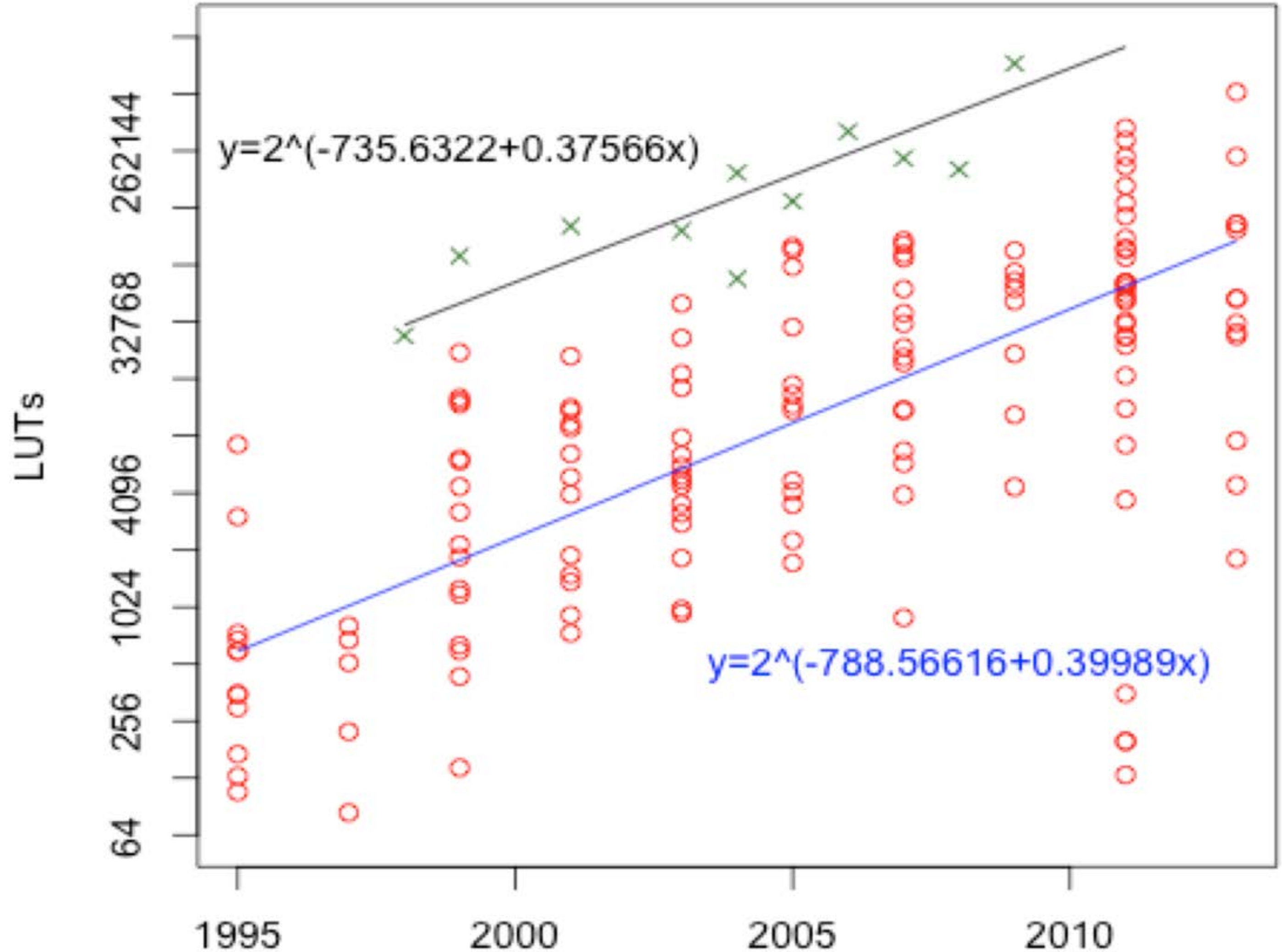
$1/\lambda^2$ vs year [2]

- FPGA lambda from previous table plotted vs year
- Transistor density doubling every two years, in agreement with Moore's Law
- Can use equation to estimate extrapolate



Design Size (number of LUTs) [2]

- x's are the number of LUTs in the largest FPGA of that year
- o's are FCCM designs
- Tech design size doubles every 2.5 years (slightly slower than Moore's Law)
- Inaccuracies because we don't count clock trees and hard blocks



Speed

Speedup

- $T_{execution}=T_{clk} * N$
  - Where N is the number of clock cycles to complete the task
- Speed $S = 1/T_{execution}$
- The **speedup** of machine A with execution time T_A over machine B with execution time T_B
  - $\text{Speedup} = \text{Speed}_A/\text{Speed}_B$

    $= T_B/T_A$
- Real-time measures often reflect performance per unit time
  - GOPS (billion operations per second)
  - GFLOPS (billions of floating point operations per second)

Amdahl's Law [3]

- Gene Amdahl in 1967 gave us a way to think about parallelism
- If B is the fraction of algorithm which is serial (e.g. I/O), and T_p is the execution time for $_p$ processors)
- Speedup = T_1 / T_p

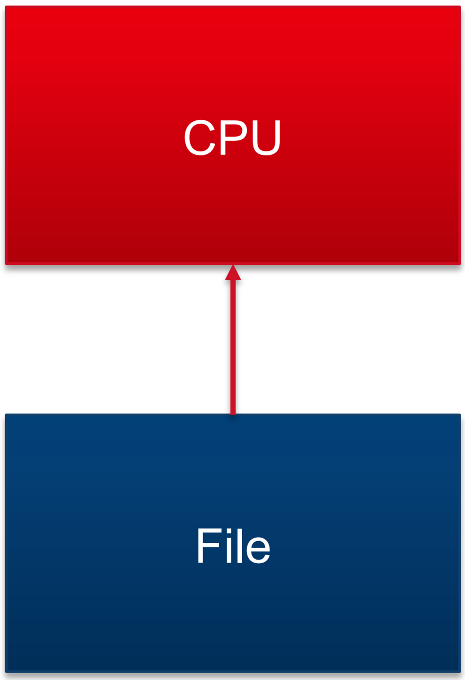
$$= \frac{p}{pB + (1 - B)}$$

- This equation gives us a way to estimate the speedup of a system

Amdahl's Law Example

Most important issue is I/O (and memory) overhead!

CPU takes 3600 s to process data



Recv 100GB of data (transfer takes 100 s)

- A program takes 3600 s to execute but must read 100GB of data from a file. If we replace the CPU with an FPGA accelerator which is 100x faster, what is the speedup?
- Speedup = $\dfrac{p}{pB+(1-B)}$
- B=100/(3600+100)=0.027
- p=100
- Speedup = 3700 s / 137 s

  = 100/(100*B+(1-B))

  =27.2059

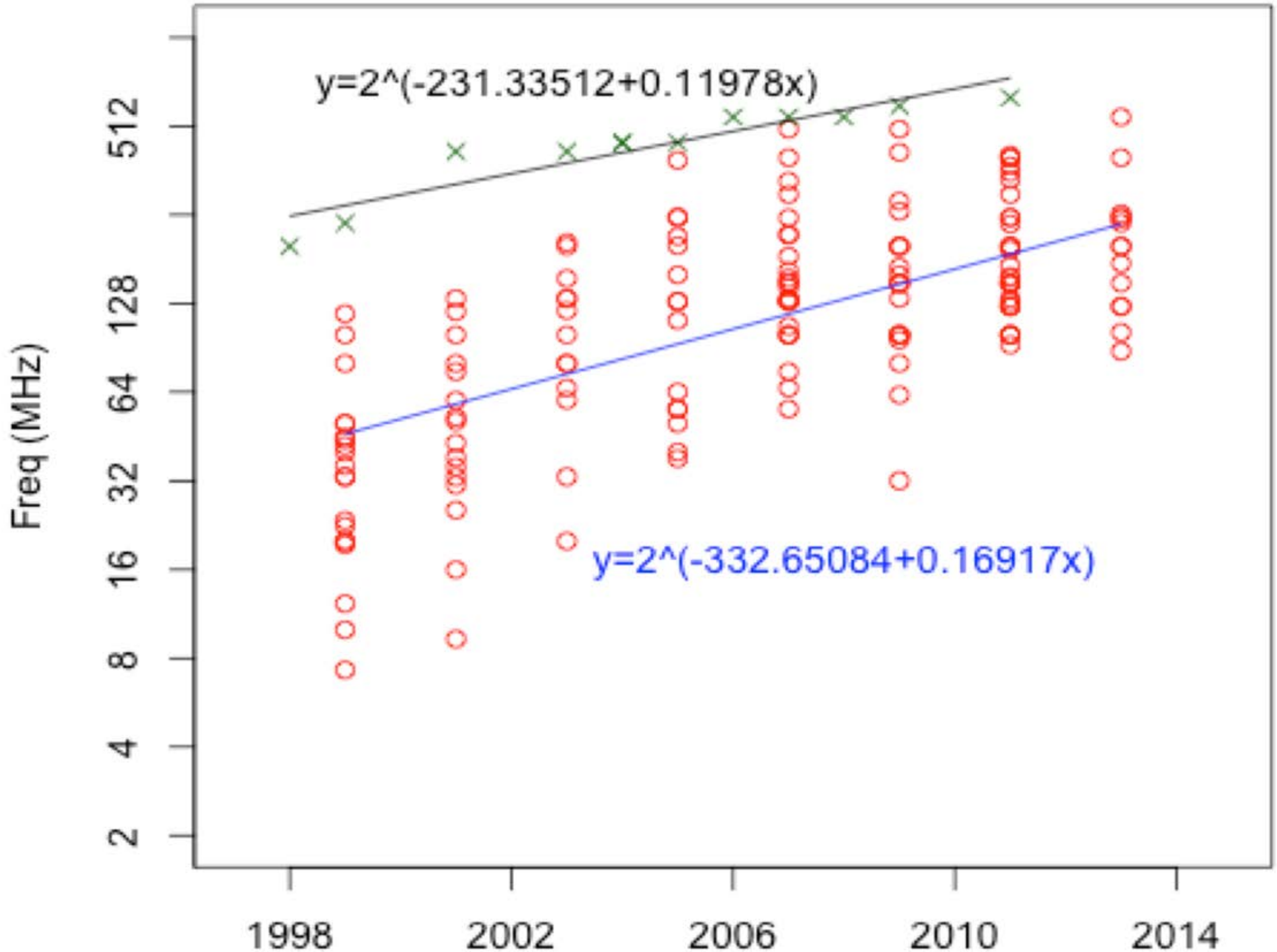
Dennard's Law [4]

› Dennard in 1974: as transistor feature size (κ or commonly λ) decreases, power stays proportional to area

Device or Circuit Parameter	Scaling Factor
Device dimension t_{ox}, L, W	$1/\kappa$
Doping concentration N_a	κ
Voltage V	$1/\kappa$
Current I	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit VC/I	$1/\kappa$
Power dissipation/circuit VI	$1/\kappa^2$
Power density VI/A	1

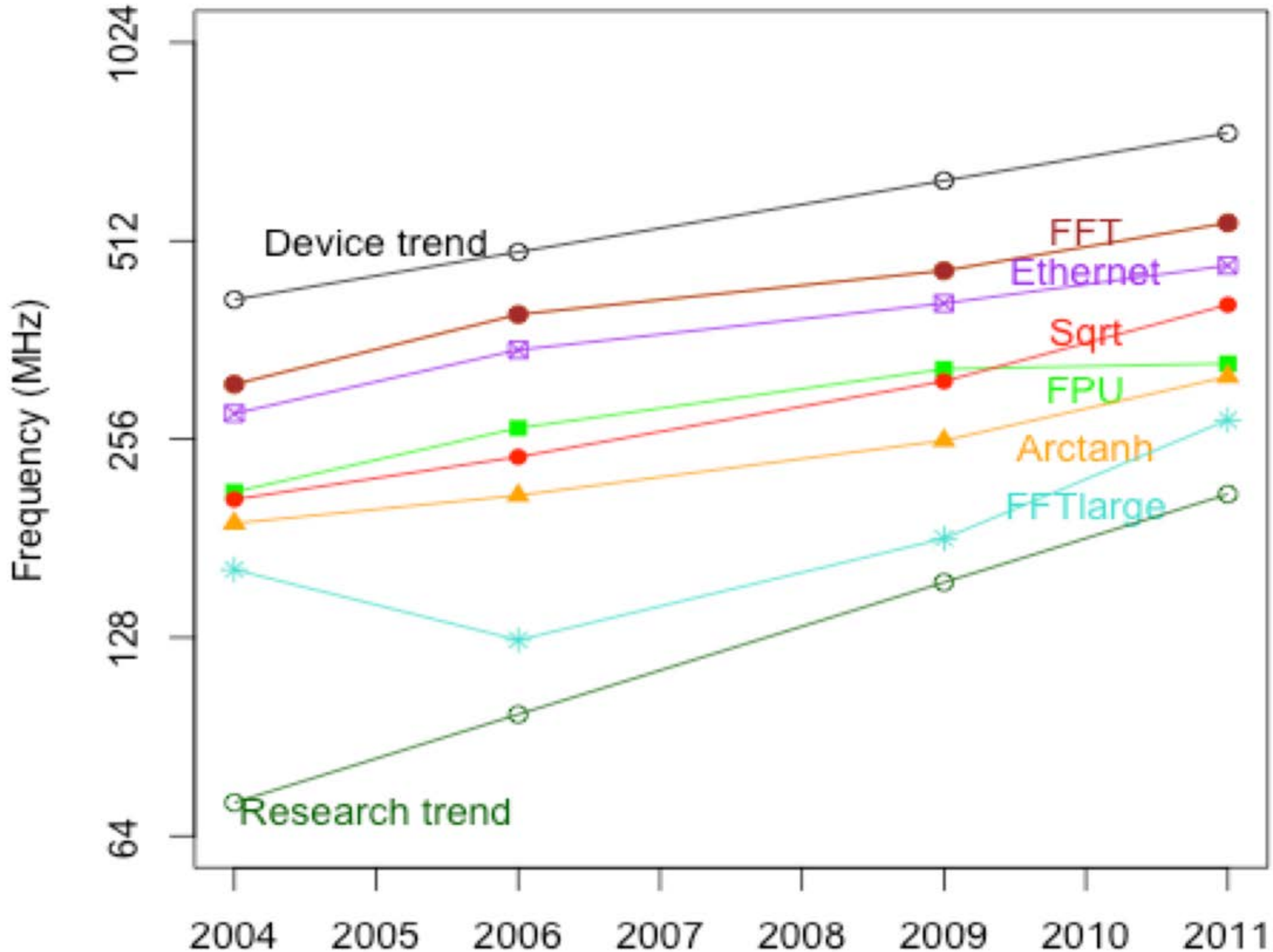
Clock Frequency (1999-2013) [2]

- **Tech freq doubles every 8 years**
- **Research freq doubles every *6* years**
- Tracking with what might be expected based on technology scaling



Frequency of IP Cores [2]

- Device technology trend is black line
- Designs have trend consistent with technology



THE UNIVERSITY OF SYDNEY

Breakdown of Dennard's Law

- Clock speeds are not rising according to Dennard's Law as transistors have stopped getting faster
- Voltage essentially stopped shrinking 10 years ago
  - Thermal noise (kT/q = 25 mV at room temperature)
  - Subthreshold leakage current
- Cannot reduce voltage and current so that power density is no longer constant
  - In fact rising sharply
  - Designs used to be speed constrained, now they are power constrained
- Cannot turn on all parts of the chip at the same time (% which must be off is called **Dark Silicon**)

Power

THE UNIVERSITY OF SYDNEY

Power Consumption

- P=CV²f and it fundamentally limits performance gains
- Three main components in an FPGA
  - Static, Dynamic, I/O
- Dennard scaling says halving lambda decreases P by 4 (broken down due to statis P)

Figure 7. Static and Dynamic Power Comparison for Same Architecture on Same Process at 0.85 V and 1.0 V

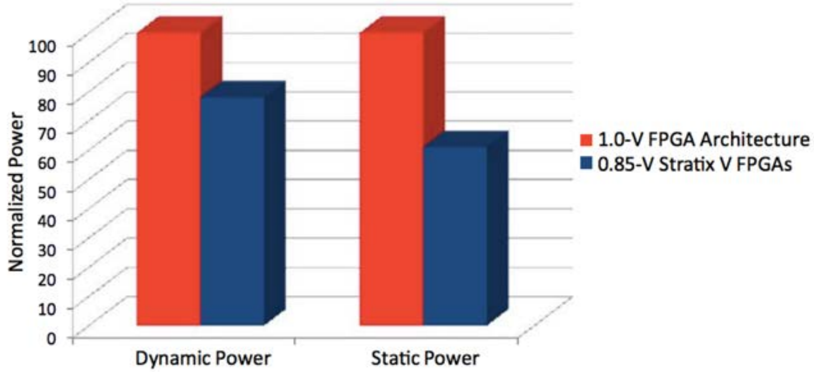
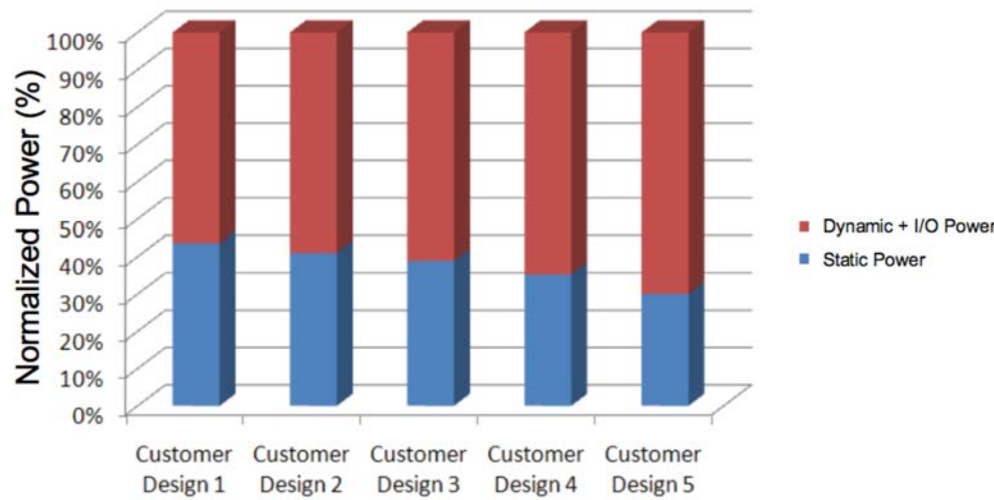


Figure 5. Total Power Breakdown Across Various High-End FPGA Customer Designs



Altera White Paper https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf

Static Power

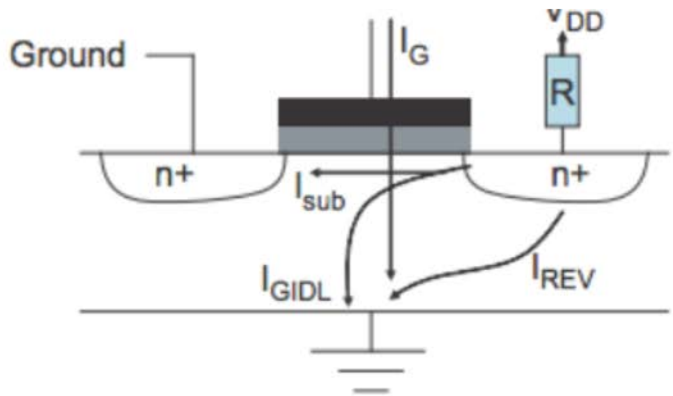


Table 1. Main Sources of Transistor Leakage

Main Sources of Leakage	Impact	Mitigation Techniques
Subthreshold leakage (I_{sub})	Dominant	■ Lower voltage ■ Higher voltage threshold ■ Longer gate length ■ Dopant profile optimization
Gate direct-tunneling leakage (I_G)	Dominant	High-k metal gate (HKMG)
Gate-induced gate leakage (I_{GIDL})	Small	Dopant profile optimization
Reverse-biased junction leakage current (I_{REV})	Negligible	Dopant profile optimization

Altera White Paper https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf

Dynamic Power

$$P_{dynamic} = \left[\frac{1}{2}CV^2 + Q_{ShortCircuit}V\right]f \cdot activity$$

Capacitance charging

Short circuit charge during switching

Percent of circuit that switches each cycle

I/O Power

Table 2. Main Factors Impacting General-Purpose I/O Power

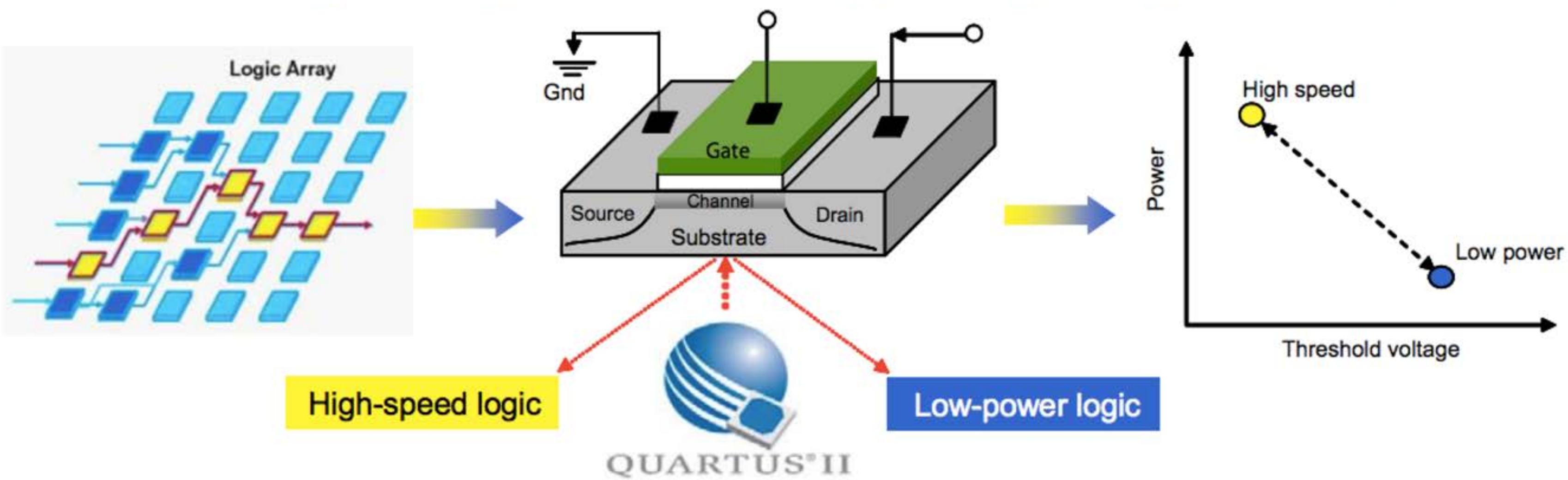
Main Factors Impacting I/O Power	Mitigation Techniques
Termination resistors (on-chip series termination (R_S OCT) and on-chip parallel termination (R_T OCT))	Dynamic on-chip termination (DOCT)
Output buffer drive strength	Programmable drive strength
Output buffer slew rate	Programmable slew rate
I/O standard (single ended, voltage referenced, or differential)	Support for multiple I/O standards
Voltage supply	Support for various voltage rails
Capacitive load (charging/discharging)	Interface dependent

Altera White Paper https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf

Programmable Power Technology

- High threshold voltage – low leakage but low speed
- Not all LUTs are on the critical path so some can be slower
- CAD tools plus configurable substrate bias allow reduced power without sacrificing speed

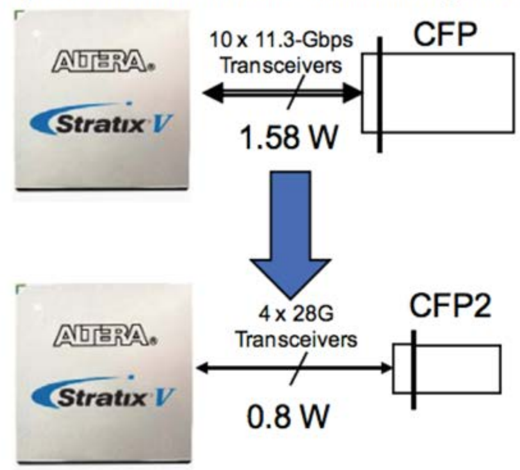
Figure 18. Programmable Power Technology Enabled by Adjusting Back-Bias Voltage



Reducing Power in FPGA Designs

- Use minimum possible voltage
- Reduce switching activity
- Use most advanced process technology with best hard blocks
- Use device with appropriate hard blocks
- Do not clock unused parts of circuit

Figure 14. Increase Bandwidth and Cut Power by Half Using 28-Gbps Transceivers



Altera White Paper https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf

ASICs vs FPGAs [6]

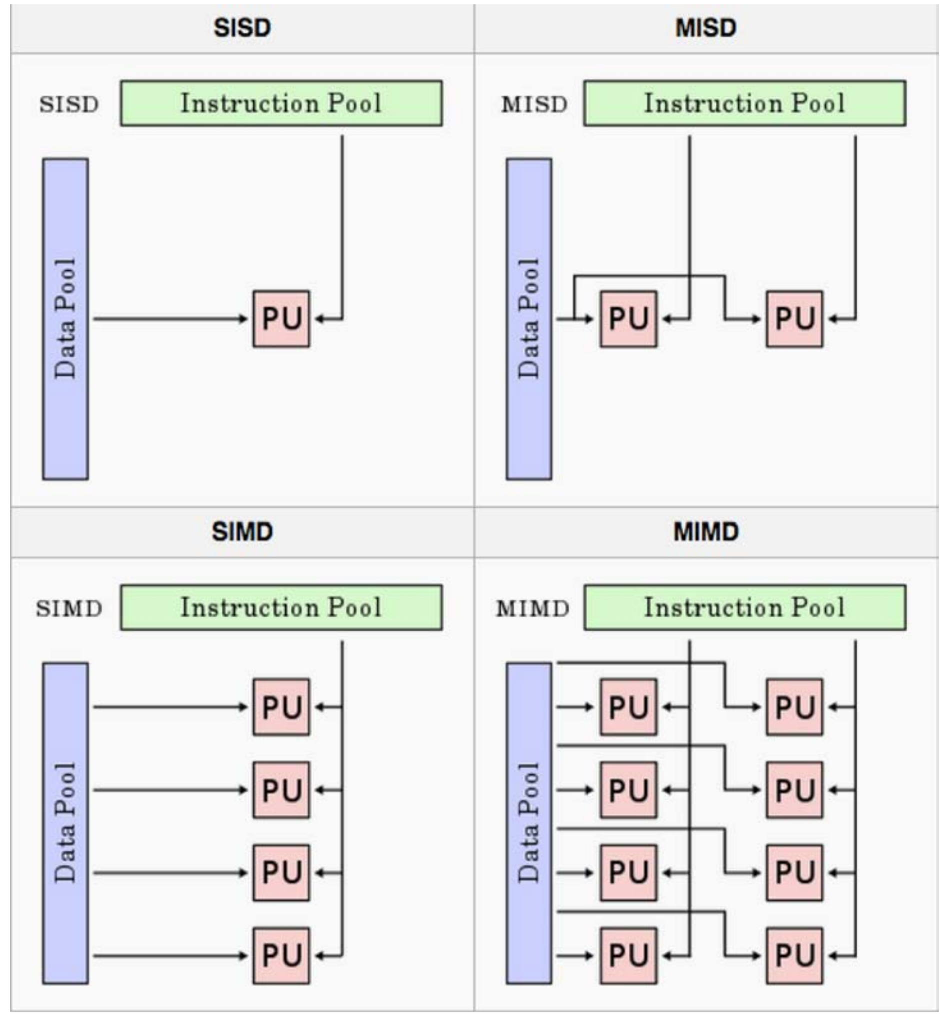
- Kuon and Rose compared FPGAs and ASICs on a number of benchmarks and found that FPGAs are
  - 20x larger area
  - 3-4x slower
  - 10x higher power
- Embedded blocks improve area and power significantly (if utilised)

Design Space Exploration

THE UNIVERSITY OF SYDNEY

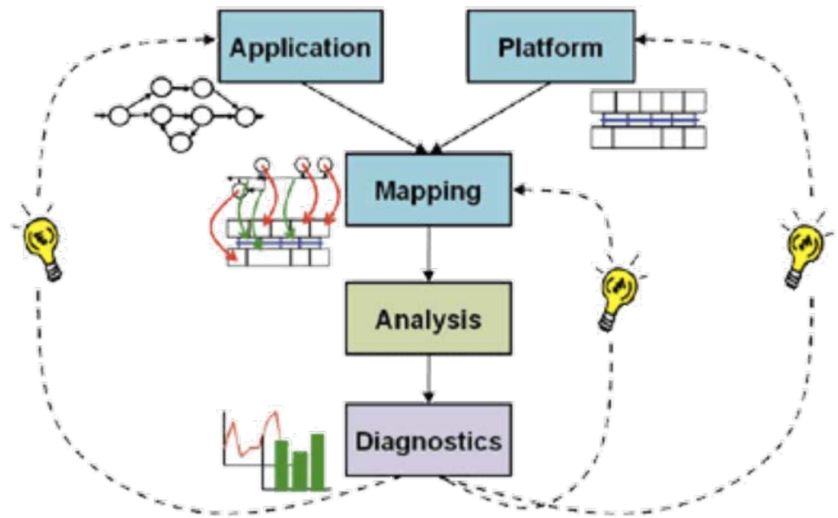
Flynn's Taxonomy

- Classification of computer architectures made in 1966 by Michael Flynn (IBM)
- Based on whether instruction and data streams are parallel
- SISD – serial processor
- SIMD – array or vector processor
- MISD – for fault tolerance, systolic array
- MIMD – multicore or distributed processor



Design Space Exploration

› Options include
  - Algorithm (most important)
  - Parallelism
  - Precision
  - Interface
  - Customisation

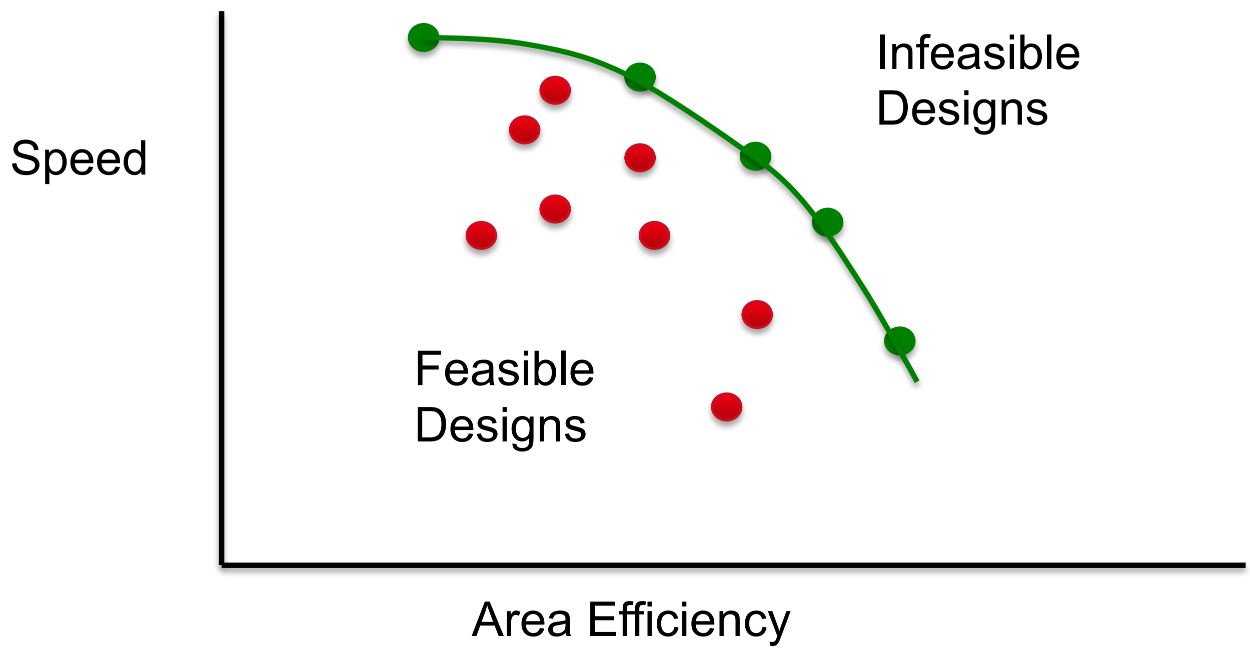


› Within each are other options and so the actual design space is extremely large

› Key to making good designs is to have good judgment regarding the tradeoffs
  - These may be different depending on what you need to optimise
  - Can be estimated using back-of-envelope techniques and reduced implementations
  - **Finding suitable input data to characterise your application is also a big issue**

Pareto Frontier

- For competing factors such as speed and area efficiency (1/area)
- Pareto Frontier separates infeasible from feasible designs
- We want to be as close to the optimal as possible



Summary

- Introduced some important principles
  - Moore's Law (tells us how IC area scales)
  - Dennard's Law (tells us how IC technology scales)
  - Amdahl's Law (tells us how to estimate speedup for parallel processing)
- FPGA designs have followed technology
- Design space is large (curse of dimensionality) so we need to be selective and tried to be close to Pareto Frontier
- Exploration must be done right to avoid having to redesign system

References

[1] G. E. Moore, “Cramming more components onto integrated circuits,” Electronics, vol. 38, no. 8, April 1965

[2] Lesley Shannon, Veronica Cojocaru, Cong Nguyen Dao, and Philip H.W. Leong. Trends in reconfigurable computing: Applications and architectures. In *Proc. FCCM*, pages 1–8, 2015

[3] Amdahl, Gene M. (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities”. AFIPS Conference Proceedings (30): 483–485. doi:10.1145/1465482.1465560

[4] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” JSSC, vol. 9, no. 5, pp. 256–268, Oct 1974.

[5] Altera White Paperhttps://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf

[6] Ian Kuon; Rose, J., "Measuring the Gap Between FPGAs and ASICs," TCAD, vol.26, no.2, pp.203,215, Feb. 2007

Review Exercises

- Explain in your own words:
  - Moore's Law (tells us how IC area scales)
  - Dennard's Law (tells us how IC technology scales)
  - Amdahl's Law
- A problem has a section of non-parallelisable code which takes 100 s to execute, and the rest of the code is parallelisable and takes 1 hour to process. If we are given the task of designing an FPGA accelerator to replace the CPU and wish to achieve a speedup of 100, what should the speedup of the FPGA accelerator core be? What if it takes a day to process?

Case Study – Matrix Multiplication

Introduction

- Serve as an example of design exploration of matrix multiplication
- While examples are for a processor with cache, they are equally valid for an FPGA with external memory

Outline

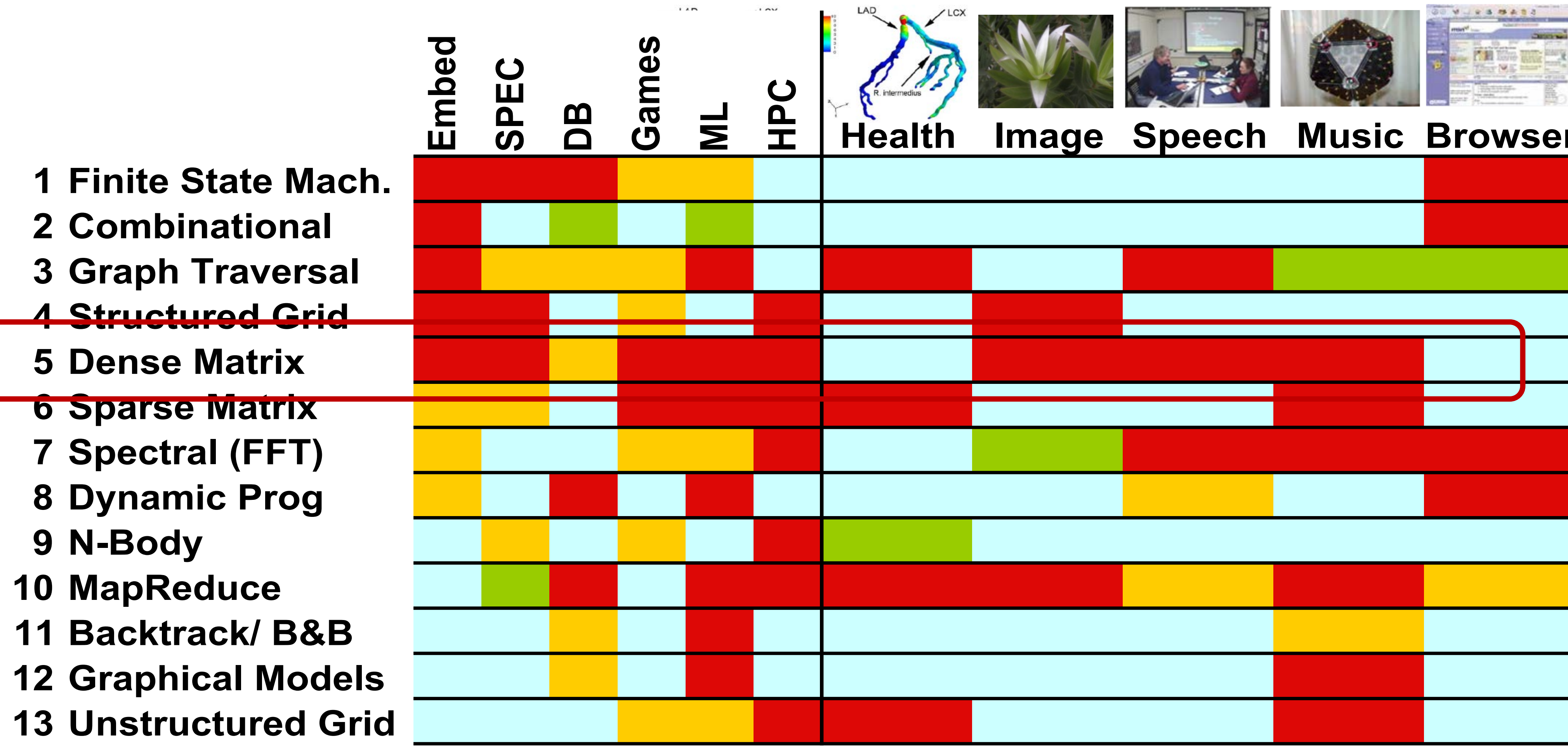
- Performance Modeling
- Matrix-Vector Multiply (Warmup)
- Matrix Multiply Cache Optimizations

Why Matrix Multiplication?

- An important kernel in many problems
  - Appears in many linear algebra algorithms
    - Bottleneck for dense linear algebra
  - One of the 7 dwarfs / 13 motifs of parallel computing
  - Closely related to other algorithms, e.g., transitive closure on a graph using Floyd-Warshall
- Optimization ideas can be used in other problems
- The best case for optimization payoffs
- The most-studied algorithm in high performance computing

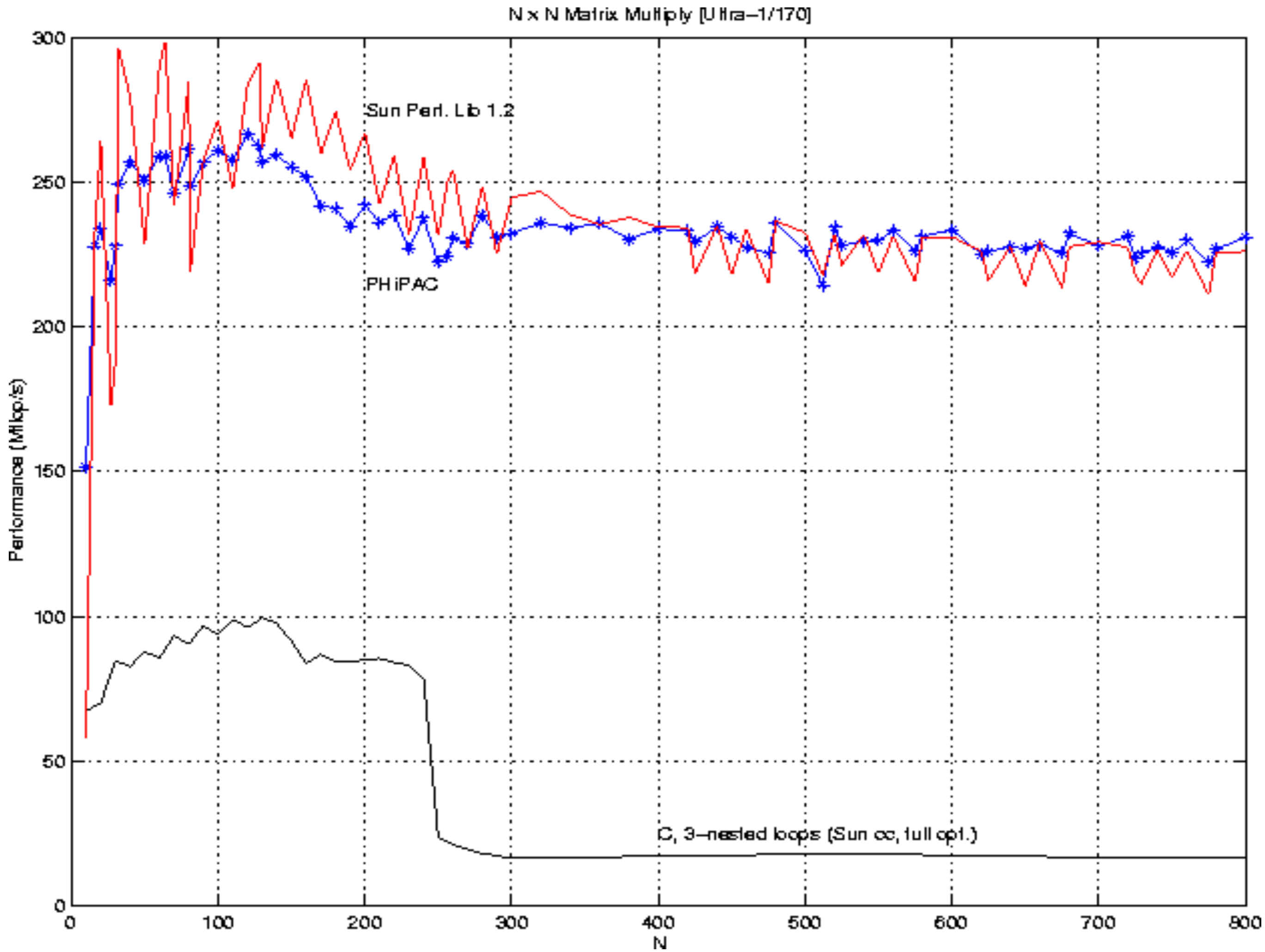
Slide: James Demmel UCB

Motif/Dwarf: Common Computational Methods (Red Hot → Blue Cool)



Slide: James Demmel UCB

Matrix-multiply, optimized several ways



Speed of n-by-n matrix multiply on Sun Ultra-1/170, peak = 330 MFlops

Note on Matrix Storage

- A matrix is a 2-D array of elements, but memory addresses are "1-D"
- Conventions for matrix layout
  - by column, or "column major" (Fortran default); A(i,j) at A+i+j*n
  - by row, or "row major" (C default) A(i,j) at A+i*n+j
  - recursive (later)

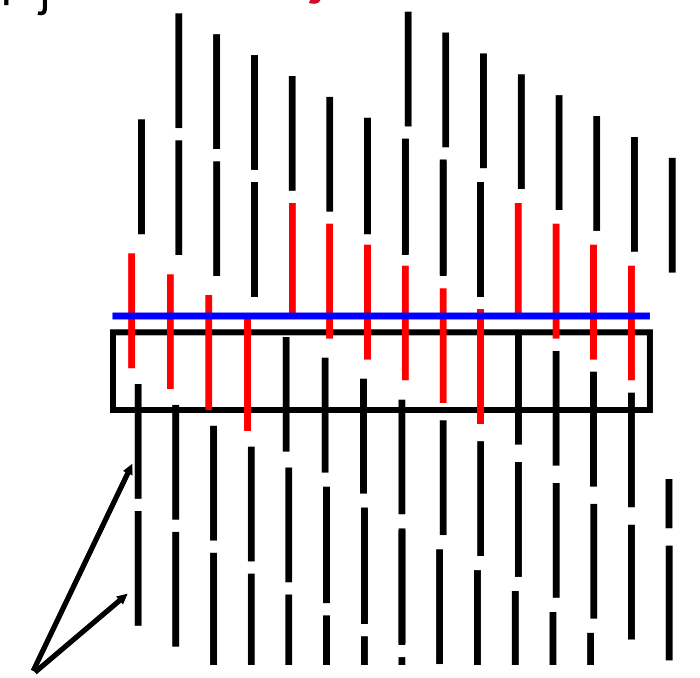
Column major

0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

Row major

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

Column major matrix in memory



- Column major (for now)

Using a Simple Model of Memory to Optimize

- Assume just 2 levels in the hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - t_m = time per slow memory operation
  - f = number of arithmetic operations
  - t_f = time per arithmetic operation << t_m
  - $\boxed{q = f / m \text{ average number of flops per slow memory access}}$

Computational Intensity: Key to algorithm efficiency

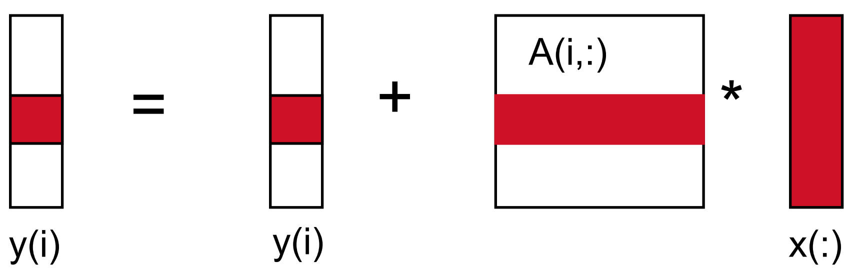
- Minimum possible time = $f^* t_f$ when all data in fast memory
- Actual time
  - $f * t_f + m * t_m = f * t_f * (1 + \boxed{t_m/t_f} * 1/q)$

Machine Balance: Key to machine efficiency

- Larger q means time closer to minimum $f * t_f$
  - $q \geq t_m/t_f$ needed to get at least half of peak speed

Warm up: Matrix-vector multiplication

```
{implements y = y + A*x}
for i = 1:n
        for j = 1:n
                y(i) = y(i) + A(i,j)*x(j)
```



Warm up: Matrix-vector multiplication

```
{read x(1:n) into fast memory}
{read y(1:n) into fast memory}
for i = 1:n
    {read row i of A into fast memory}
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j)
{write y(1:n) back to slow memory}
```

- m = number of slow memory refs = $3n + n^2$
- f = number of arithmetic operations = $2n^2$
- $q = f / m \approx 2$

- Matrix-vector multiplication limited by slow memory speed

Modeling Matrix-Vector Multiplication

- Compute time for nxn = 1000x1000 matrix
- Time
  - $f * t_f + m * t_m = f * t_f * (1 + t_m/t_f * 1/q)$
  - $= 2*n^2 * t_f * (1 + t_m/t_f * 1/2)$
- For t_f and t_m, using data from R. Vuduc's PhD (pp 351-3)
  - http://bebop.cs.berkeley.edu/pubs/vuduc2003-dissertation.pdf
  - For t_m use minimum-memory-latency / words-per-cache-line

	Clock	Peak	Mem Lat (Min,Max)		Linesize	t_m/t_f
	MHz	Mflop/s	cycles		Bytes	
Ultra 2i	333	667	38	66	16	24.8
Ultra 3	900	1800	28	200	32	14.0
Pentium 3	500	500	25	60	32	6.3
Pentium3M	800	800	40	60	32	10.0
Power3	375	1500	35	139	128	8.8
Power4	1300	5200	60	10000	128	15.0
Itanium1	800	3200	36	85	32	36.0
Itanium2	900	3600	11	60	64	5.5

machine balance (q must be at least this for ½ peak speed)

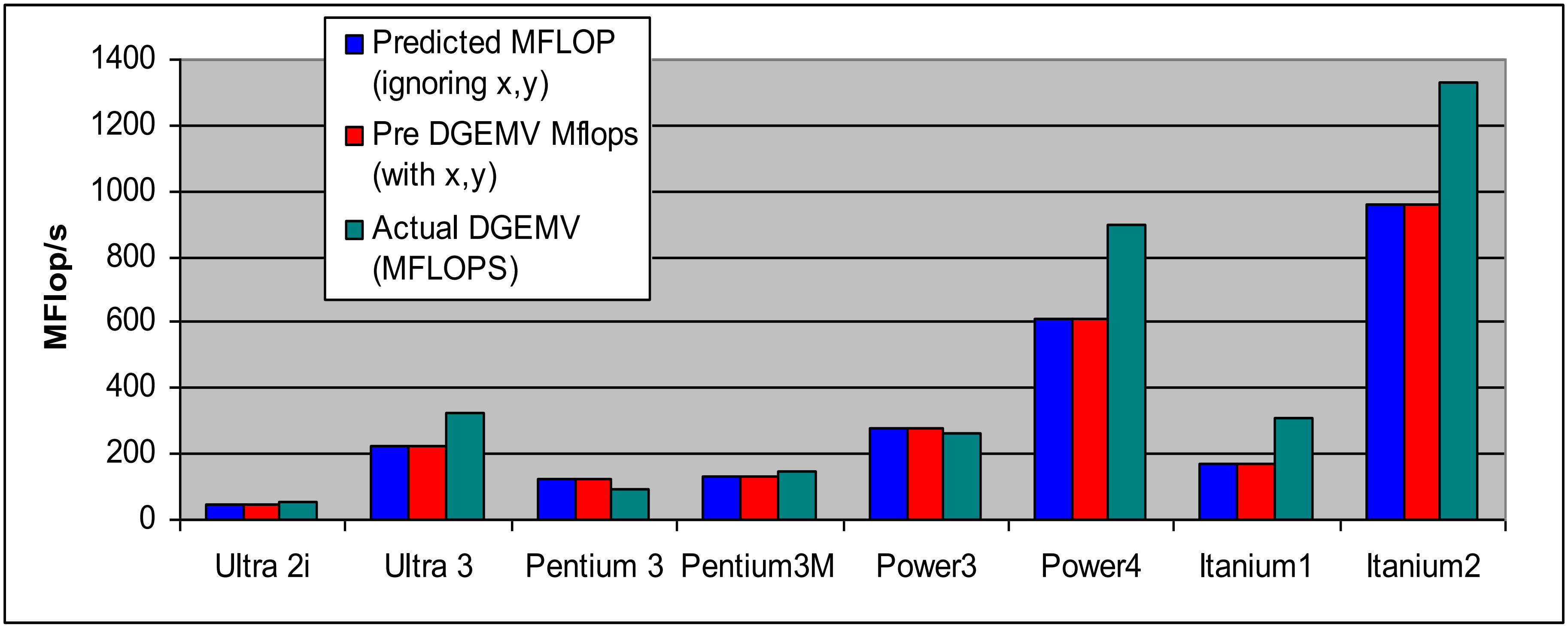
Simplifying Assumptions

› What simplifying assumptions did we make in this analysis?
  - Ignored parallelism in processor between memory and arithmetic within the processor
    - Sometimes drop arithmetic term in this type of analysis
  - Assumed fast memory was large enough to hold three vectors
    - Reasonable if we are talking about any level of cache
    - Not if we are talking about registers (~32 words)
  - Assumed the cost of a fast memory access is 0
    - Reasonable if we are talking about registers
    - Not necessarily if we are talking about cache (1-2 cycles for L1)
  - Memory latency is constant

› Could simplify even further by ignoring memory operations in X and Y vectors
  - Mflop rate/element = $2 / (2^* t_f + t_m)$

Validating the Model

- How well does the model predict actual performance?
  - Actual DGEMV: Most highly optimized code for the platform
- Model sufficient to compare across machines
- But under-predicting on most recent ones due to latency estimate



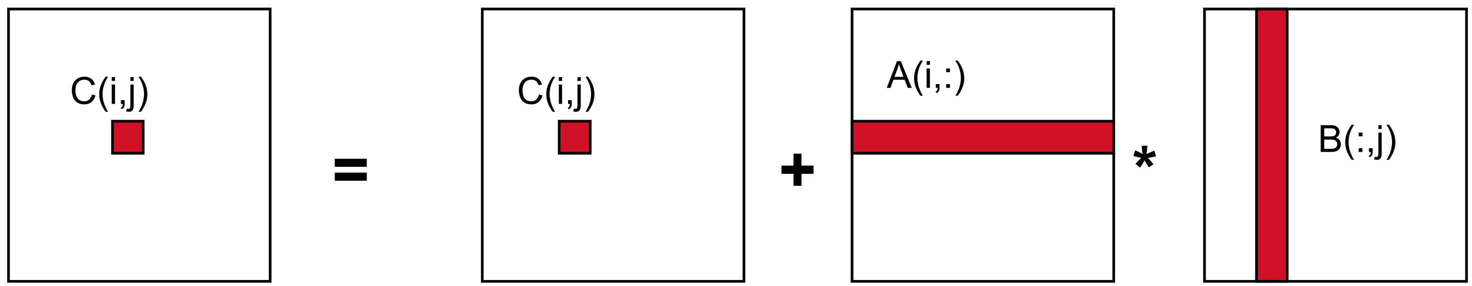
Naïve Matrix Multiply

{implements C = C + A*B}

```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

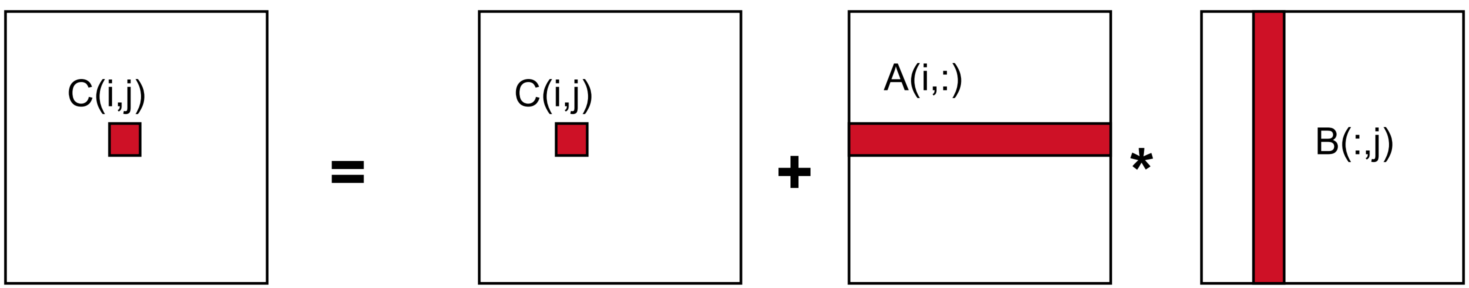
Algorithm has $2*n^3 = O(n^3)$ Flops and operates on $3*n^2$ words of memory

q potentially as large as $2*n^3 / 3*n^2 = O(n)$



Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}
   for j = 1 to n
      {read C(i,j) into fast memory}
      {read column j of B into fast memory}
      for k = 1 to n
         C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write C(i,j) back to slow memory}
```



Naïve Matrix Multiply

Number of slow memory references on unblocked matrix multiply

$m = n^3$ to read each column of B n times

$+ n^2$ to read each row of A once

$+ 2n^2$ to read and write each element of C once

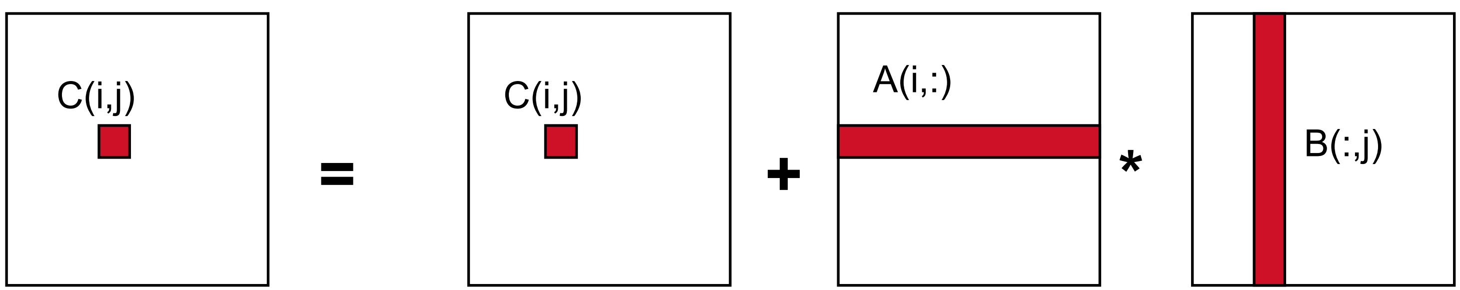
$= n^3 + 3n^2$

So $q = f / m = 2n^3 / (n^3 + 3n^2)$

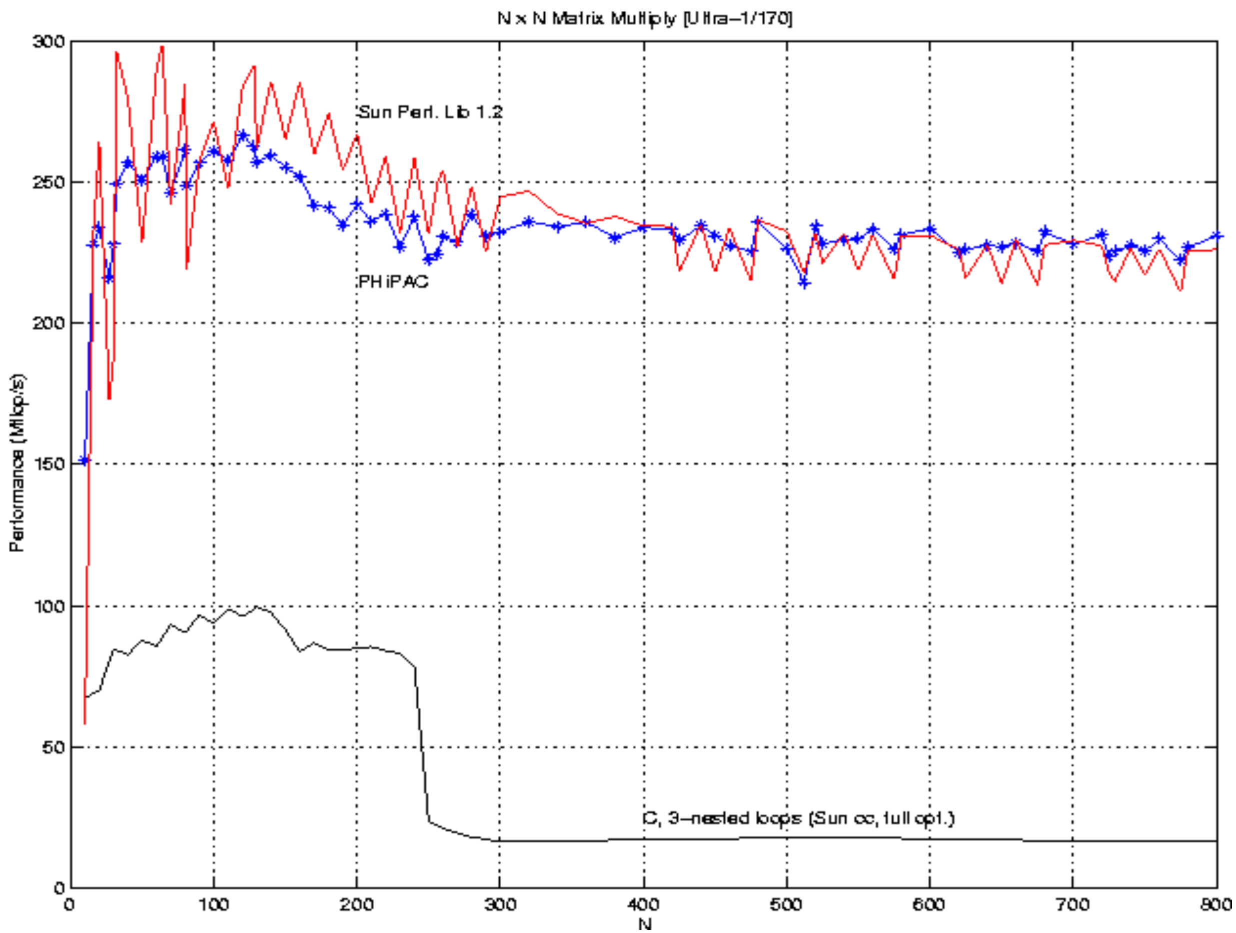
≈ 2 for large n, no improvement over matrix-vector multiply

Inner two loops are just matrix-vector multiply, of row i of A times B

Similar for any other order of 3 loops



Matrix-multiply, optimized several ways

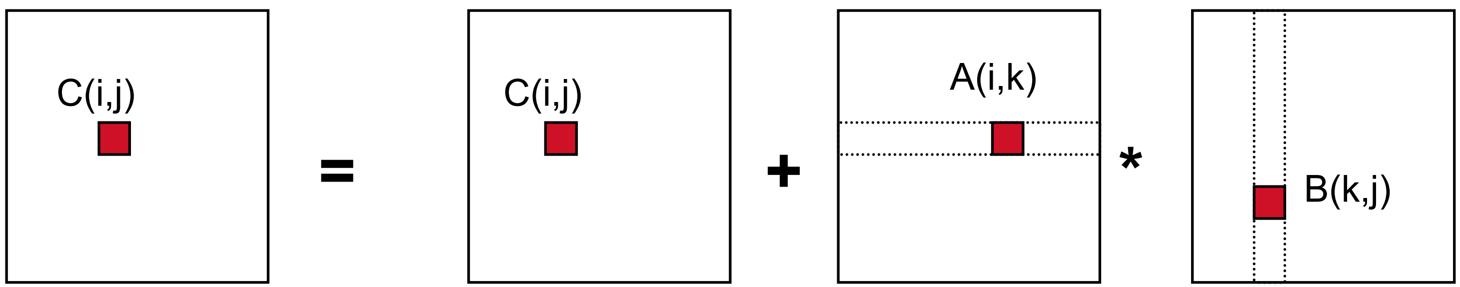


Speed of n-by-n matrix multiply on Sun Ultra-1/170, peak = 330 MFlops

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be N-by-N matrices of b-by-b subblocks where b=n / N is called the block size

```
for i = 1 to N
   for j = 1 to N
      {read block C(i,j) into fast memory}
      for k = 1 to N
           {read block A(i,k) into fast memory}
           {read block B(k,j) into fast memory}
            C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
      {write block C(i,j) back to slow memory}
```



Blocked (Tiled) Matrix Multiply

Recall:

m is amount memory traffic between slow and fast memory

matrix has nxn elements, and NxN blocks each of size bxb

f is number of floating point operations, $2n^3$ for this problem

q = f / m is our measure of algorithm efficiency in the memory system

So:

$m = N^*n^2$ read each block of B N^3 times ($N^3 * b^2 = N^3 * (n/N)^2 = N^*n^2$)

$+ N^*n^2$ read each block of A N^3 times

$+ 2n^2$ read and write each block of C once

$= (2N + 2) * n^2$

So computational intensity $q = f / m = 2n^3 / ((2N + 2) * n^2)$

$\approx n / N = b$ for large n

So we can improve performance by increasing the blocksize b

Can be much faster than matrix-vector multiply (q=2)

Using Analysis to Understand Machines

The blocked algorithm has computational intensity $q \approx b$

› The larger the block size, the more efficient our algorithm will be

› Limit: All three blocks from A,B,C must fit in fast memory (cache), so we cannot make these blocks arbitrarily large

› Assume your fast memory has size M_{fast}

$$3b^2 \leq M_{fast}, \quad \text{so} \quad q \approx b \leq (M_{fast}/3)^{1/2}$$

- To build a machine to run matrix multiply at 1/2 peak arithmetic speed of the machine, we need a fast memory of size

$$M_{fast} \geq 3b^2 \approx 3q^2 = 3(t_m/t_f)^2$$

- This size is reasonable for L1 cache, but not for register sets
- Note: analysis assumes it is possible to schedule the instructions perfectly

	t_m/t_f	required KB
Ultra 2i	24.8	14.8
Ultra 3	14	4.7
Pentium 3	6.25	0.9
Pentium3M	10	2.4
Power3	8.75	1.8
Power4	15	5.4
Itanium1	36	31.1
Itanium2	5.5	0.7

Limits to Optimizing Matrix Multiply

› The blocked algorithm changes the order in which values are accumulated into each C[i,j] by applying commutativity and associativity

- Get slightly different answers from naïve code, because of roundoff - OK

› The previous analysis showed that the blocked algorithm has computational intensity:

$$q \approx b \leq (M_{fast}/3)^{1/2}$$

› There is a lower bound result that says we cannot do any better than this (using only associativity)

› **Theorem (Hong & Kung, 1981): Any reorganization of this algorithm (that uses only associativity) is limited to $q = O((M_{fast})^{1/2})$**

- **#words moved between fast and slow memory = $\Omega (n^3 / (M_{fast})^{1/2})$**

What if there are more than 2 levels of memory?

- Need to minimize communication between all levels
  - Between L1 and L2 cache, cache and DRAM, DRAM and disk…
- The tiled algorithm requires finding a good block size
  - Machine dependent
  - Need to "block" b x b matrix multiply in inner most loop
    - 1 level of memory $\Rightarrow$ 3 nested loops (naïve algorithm)
    - 2 levels of memory $\Rightarrow$ 6 nested loops
    - 3 levels of memory $\Rightarrow$ 9 nested loops …
- Cache Oblivious Algorithms offer an alternative
  - Treat nxn matrix multiply as a set of smaller problems
  - Eventually, these will fit in cache
  - Will minimize # words moved between every level of memory hierarchy – at least asymptotically

Summary

- Described a way to think about computation and memory – computational intensity
- Introduced the concept of blocking to increase computational intensity

Review Exercises

- Explain in your own words:
  - Computational intensity
- Do a similar analysis computational intensity analysis for a different algorithm e.g. FFT

An FPGA Delay Model

THE UNIVERSITY OF SYDNEY

THE UNIVERSITY OF SYDNEY

A Detailed Delay Path Model for FPGAs

This work based on a paper at FPT09 [1]

Eddie Hung[1], Steven J. E. Wilton[1],

Haile Yu[2], Thomas C. P. Chau[2], Philip H. W. Leong[2*]

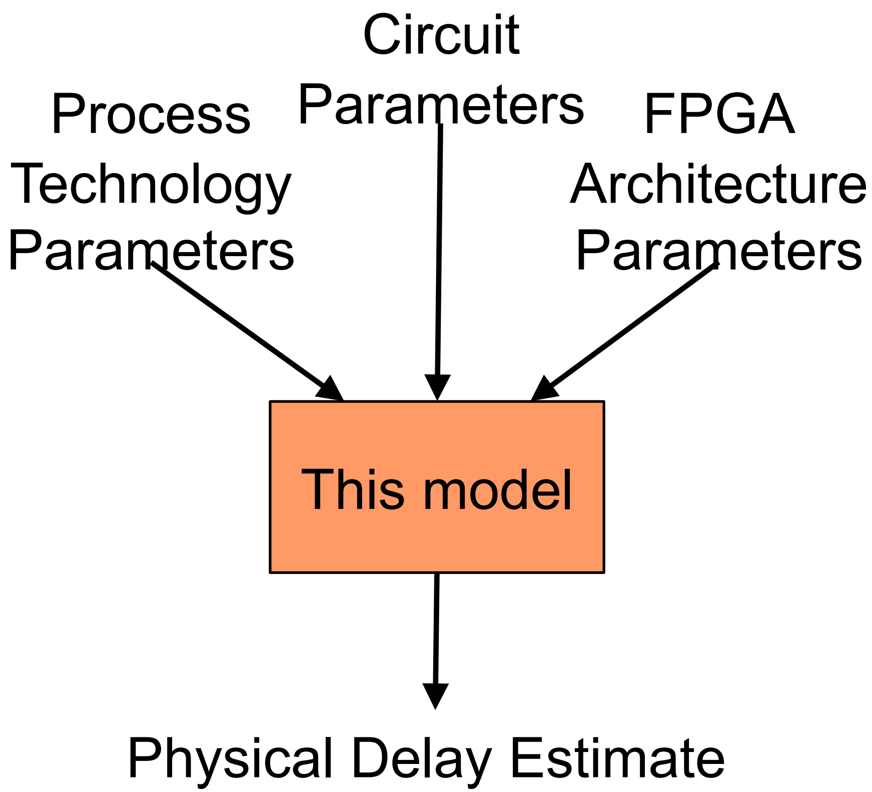
[1] Department of ECE, University of British Columbia

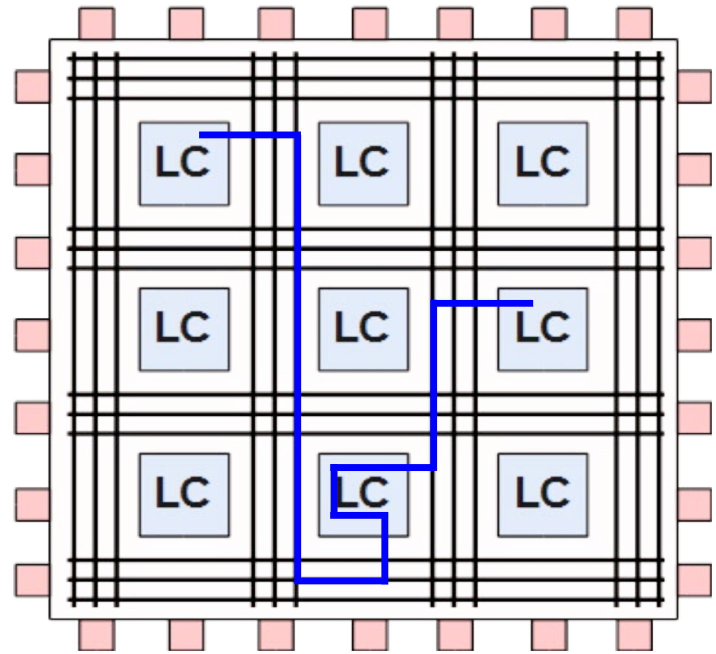
[2] Department of CSE, Chinese University of Hong Kong

[*] *Now with* School of Electrical and Information Engineering, University of Sydney

Funded by NSERC of Canada and RGC of HKSAR

Overview





Compared to previous models:

- Simpler, closed-form, equally accurate

Motivation: FPGA Design

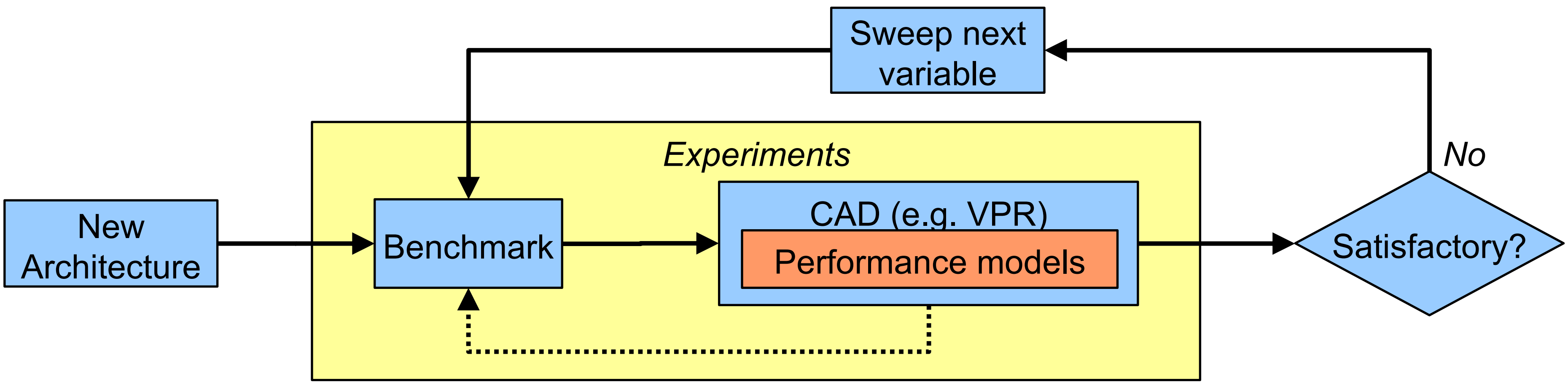
Important when *designing* FPGA architectures:

- Need methods to estimate their performance ahead of time
- Two different ways of investigating new architectures:
  - Analytical Models (our approach)
  - Experimental Techniques

Motivation: FPGA Design

Existing FPGA design approach:

- Iteratively change details and experimentally measure improvement using benchmarks



- Problems:
  - Slow and resource-hungry
  - Lack of intuition and insight into why

Motivation: Analytical Models

New paradigm emerging: Analytical Modelling

- Capturing the *essence* of programmable logic in a set of simple equations

THE UNIVERSITY OF SYDNEY

Motivation: Analytical Models

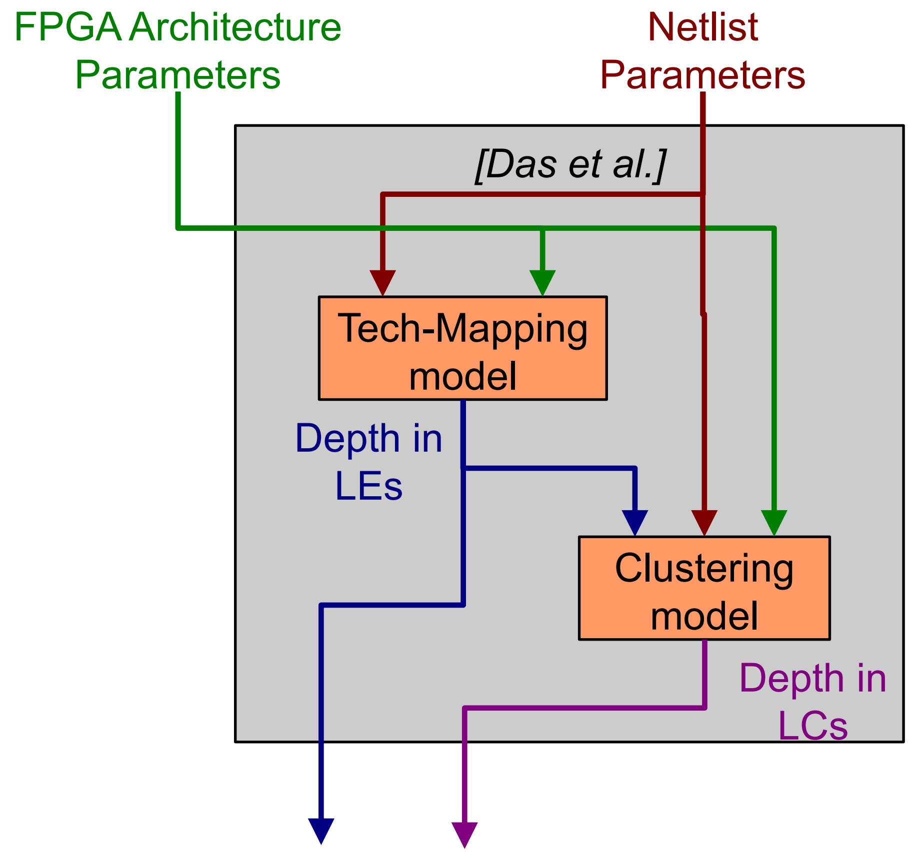
- Why analytical modelling?
  - Faster
  - Allow early exploration of radical architectures

- What makes a good model?
  - Analytical – not rely on curve fitting
  - Simple – more insight into architectural trade-offs
  - Circuit Independent – capturing average behaviour

Where this work fits in

Delay Model:

- Logical delay model presented by Das *et al.* at FPL 2009 [2]

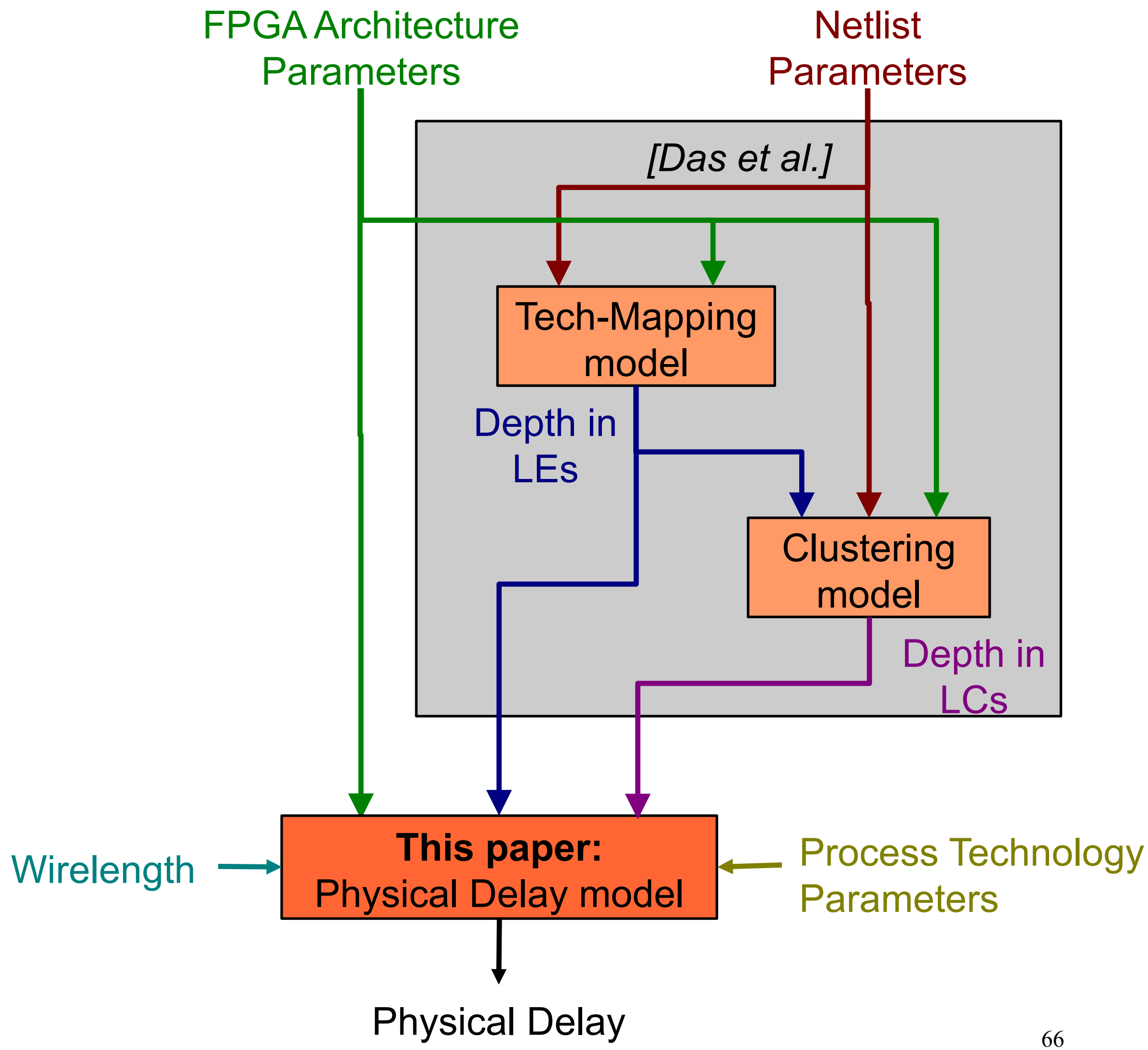


Where this work fits in

Delay Model:

- Logical delay model presented by Das *et al.* at FPL 2009 [2]
- **This paper:**

  A model which relates *logical* delay to *physical* delay



THE UNIVERSITY OF SYDNEY

Motivation: Analytical Models

- Can the models from the experimental flow be re-used for the analytical flow?
  - Requires routing/timing graphs
  - Lack of delay model for logic cluster

What makes deriving this model hard?

- Would like our model to be:
  - Flexible, coping with range of modern architectures
  - Accurate
  - Closed-form
  - Fast
- But complex interactions exist between FPGA architecture and circuit implementation:
  - e.g. Buffer sizes change depending on loading

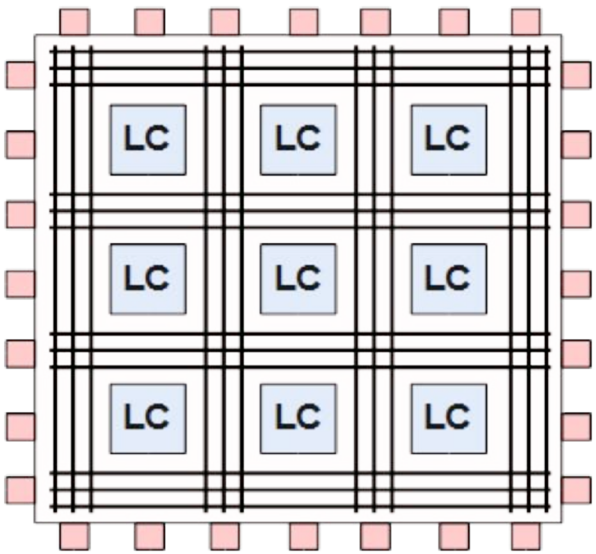
Circuit Assumptions and Delay Model

Circuit Assumptions

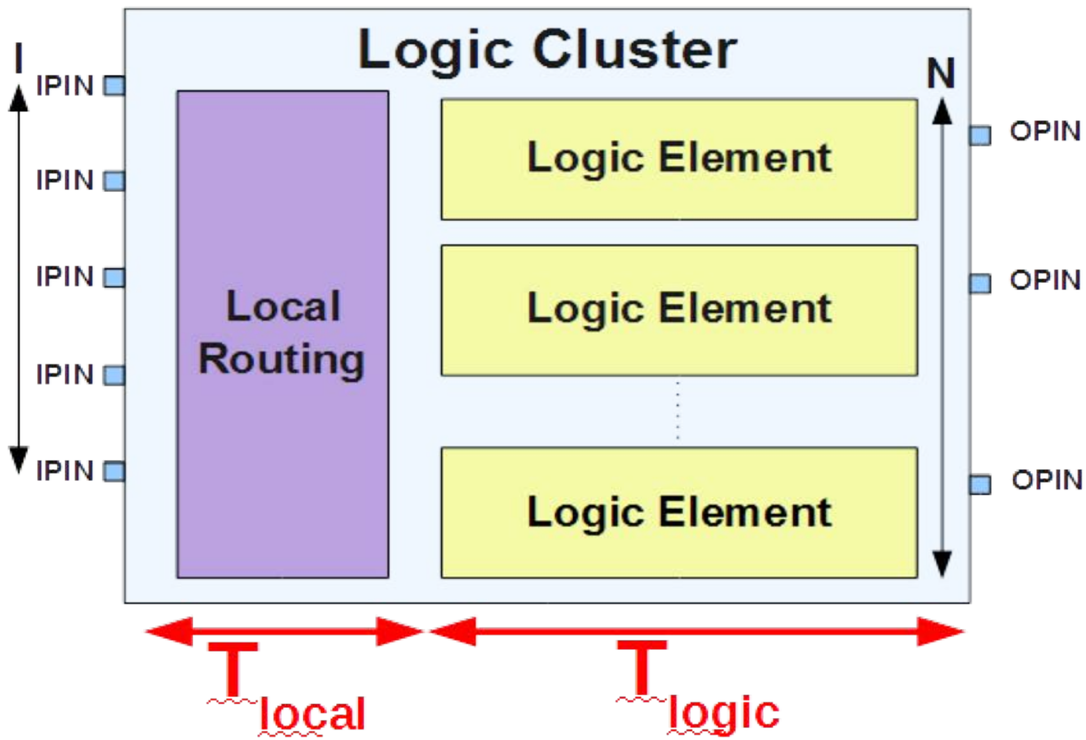
- Island-Style FPGA
  - 2-D array of *Logic Clusters* surrounded by a *Global Interconnect* of routing tracks
- Delay model broken down into:
  - Local Routing Delay:
  - Logic Element Delay:
  - Global Interconnect Delay:

Logic Cluster: T_{local} and T_{logic}

›Collection of logic elements accessed through a local routing network with a shared set of inputs



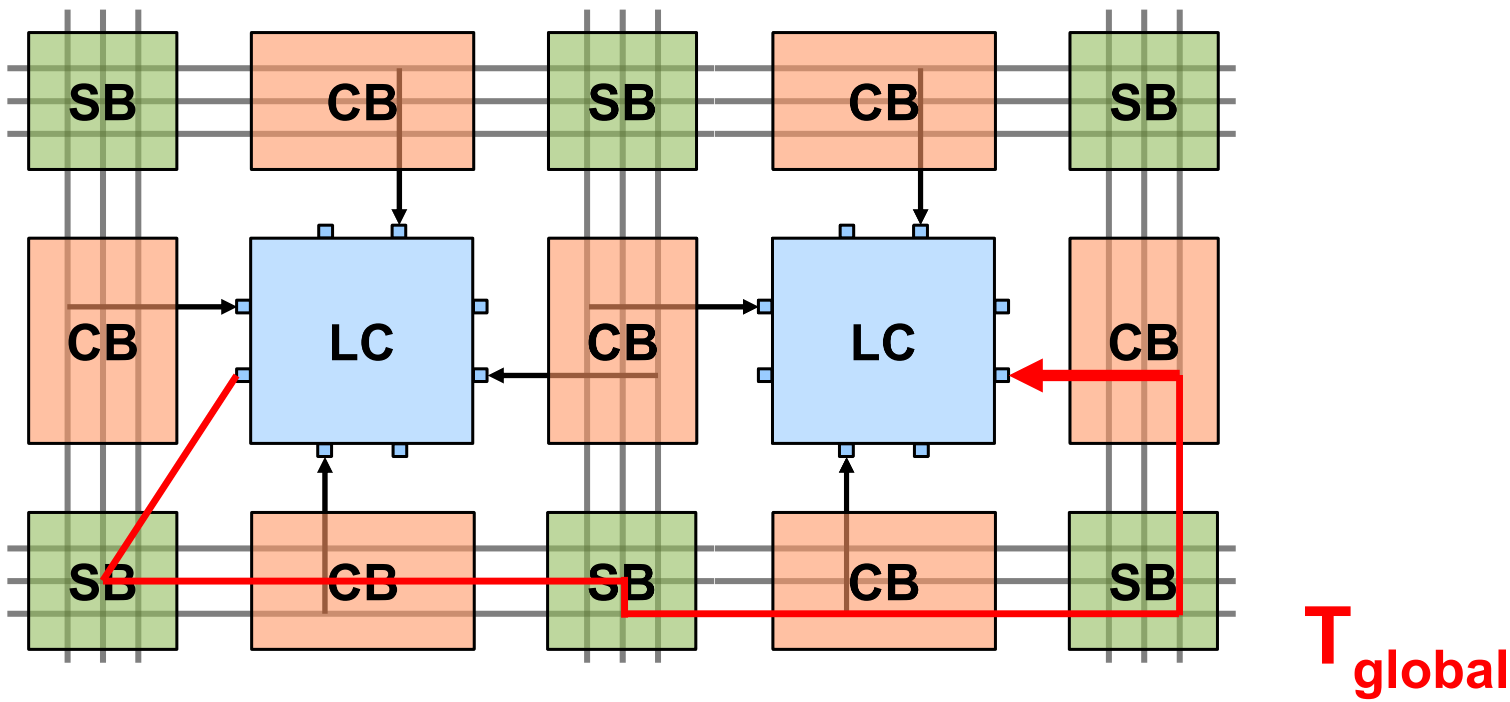
Island-Style FPGA



THE UNIVERSITY OF SYDNEY

Global Interconnect: T_{global}

› Composed of horizontal/vertical tracks and Connection/Switch Boxes



Delay Model: How it fits together

Expected number of LCs on critical path

Expected number of LEs on critical path

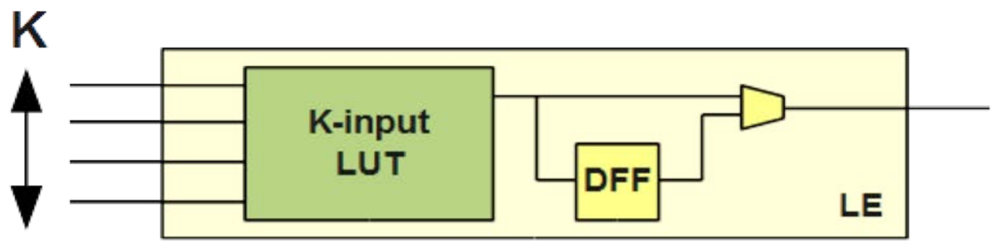
$$T_{crit} = d_c \cdot T_{global} + d_k \cdot \left(T_{local} + T_{logic}\right)$$

Critical Path Delay

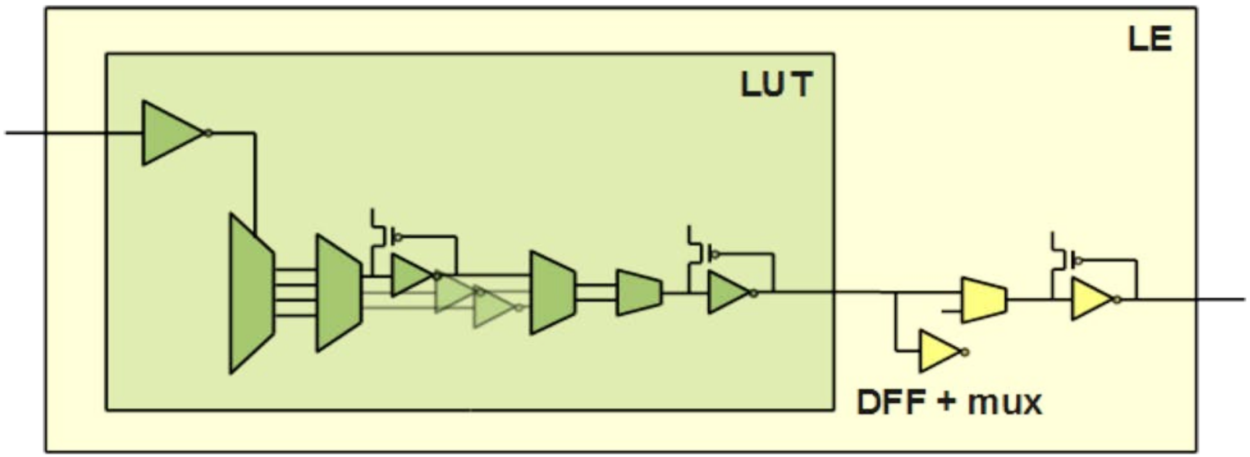
Function of the expected wirelength between LCs

Circuit Assumptions: Logic Element

- Lookup table with D flip-flop and bypass mux



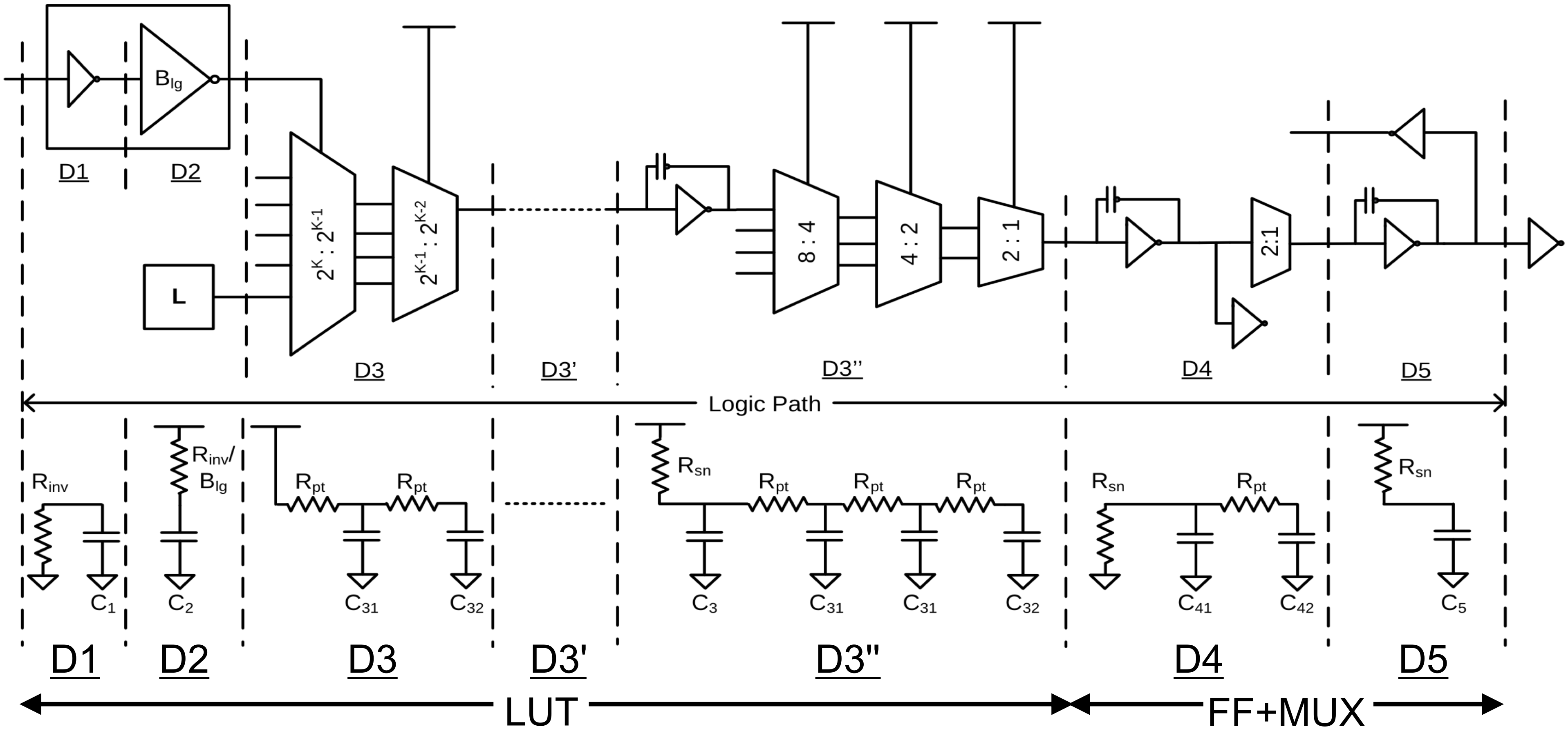
Architecture View



Circuit View

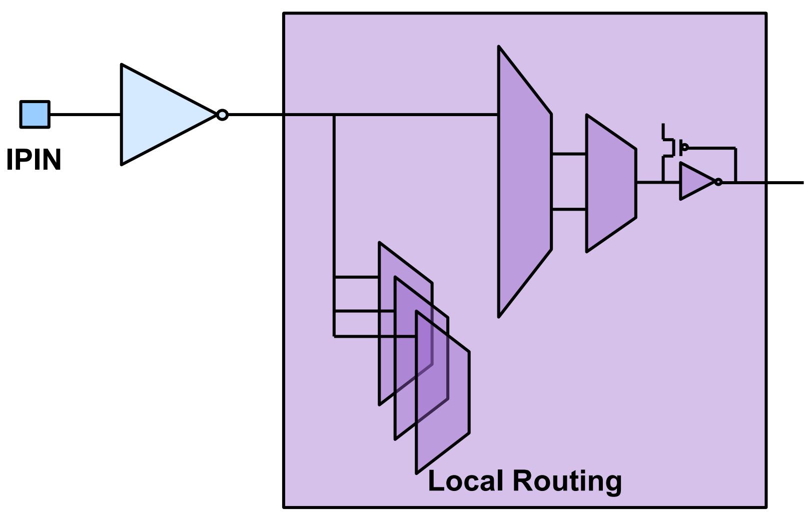
Circuit Assumptions: Logic Element

RC Network View

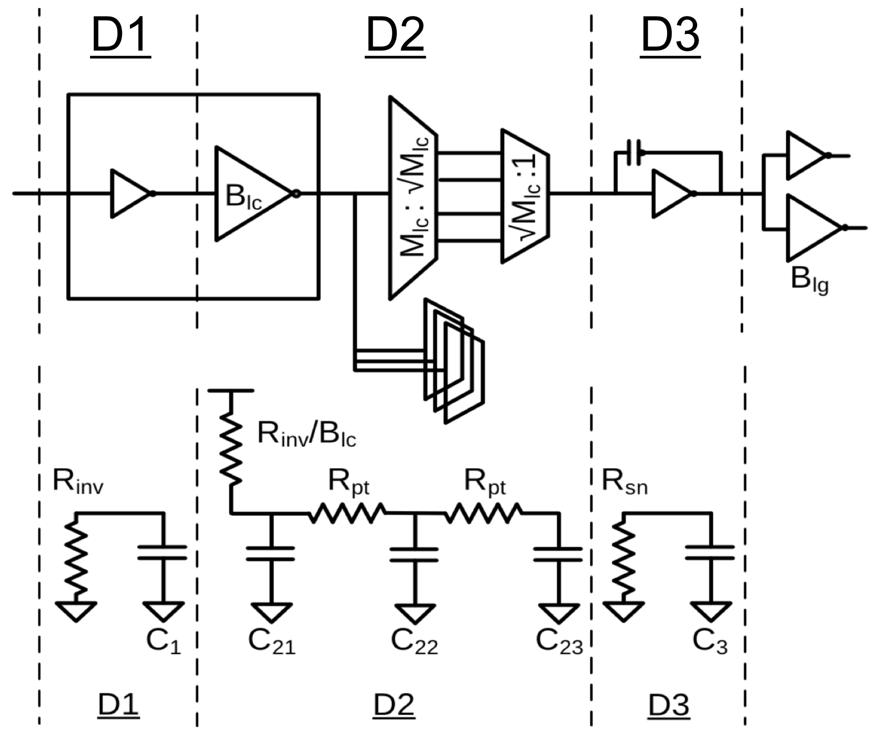


Circuit Assumptions: Local Routing

› Fully connected crossbar implemented using multiplexers



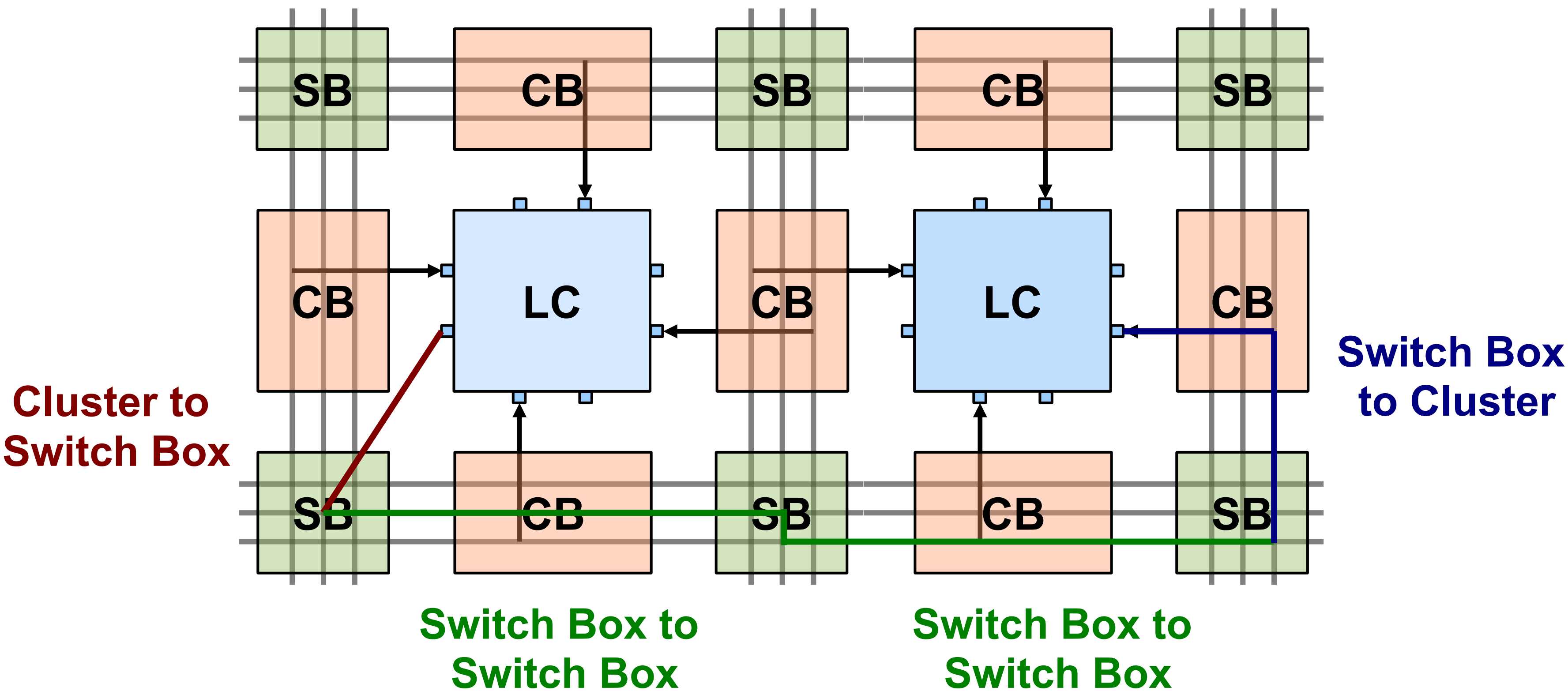
Circuit View



RC Network View

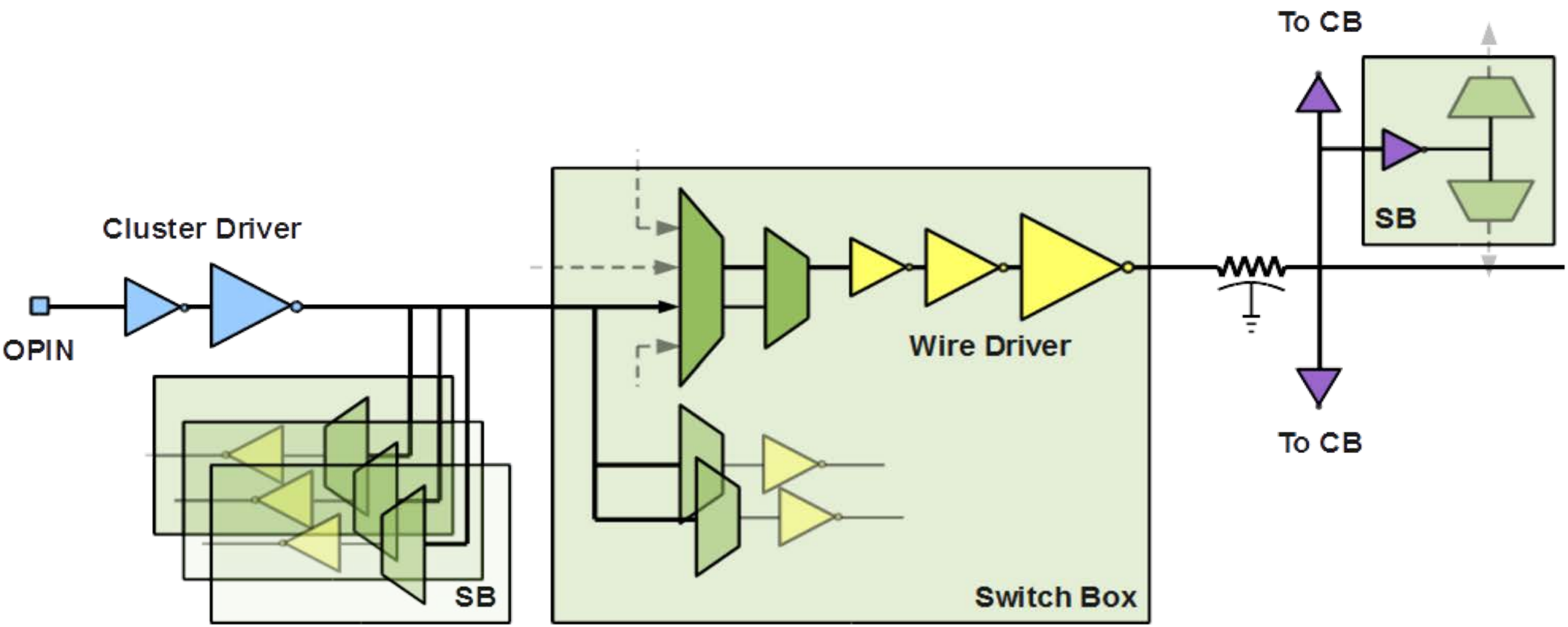
Global Interconnect Delay

› Further divided into:



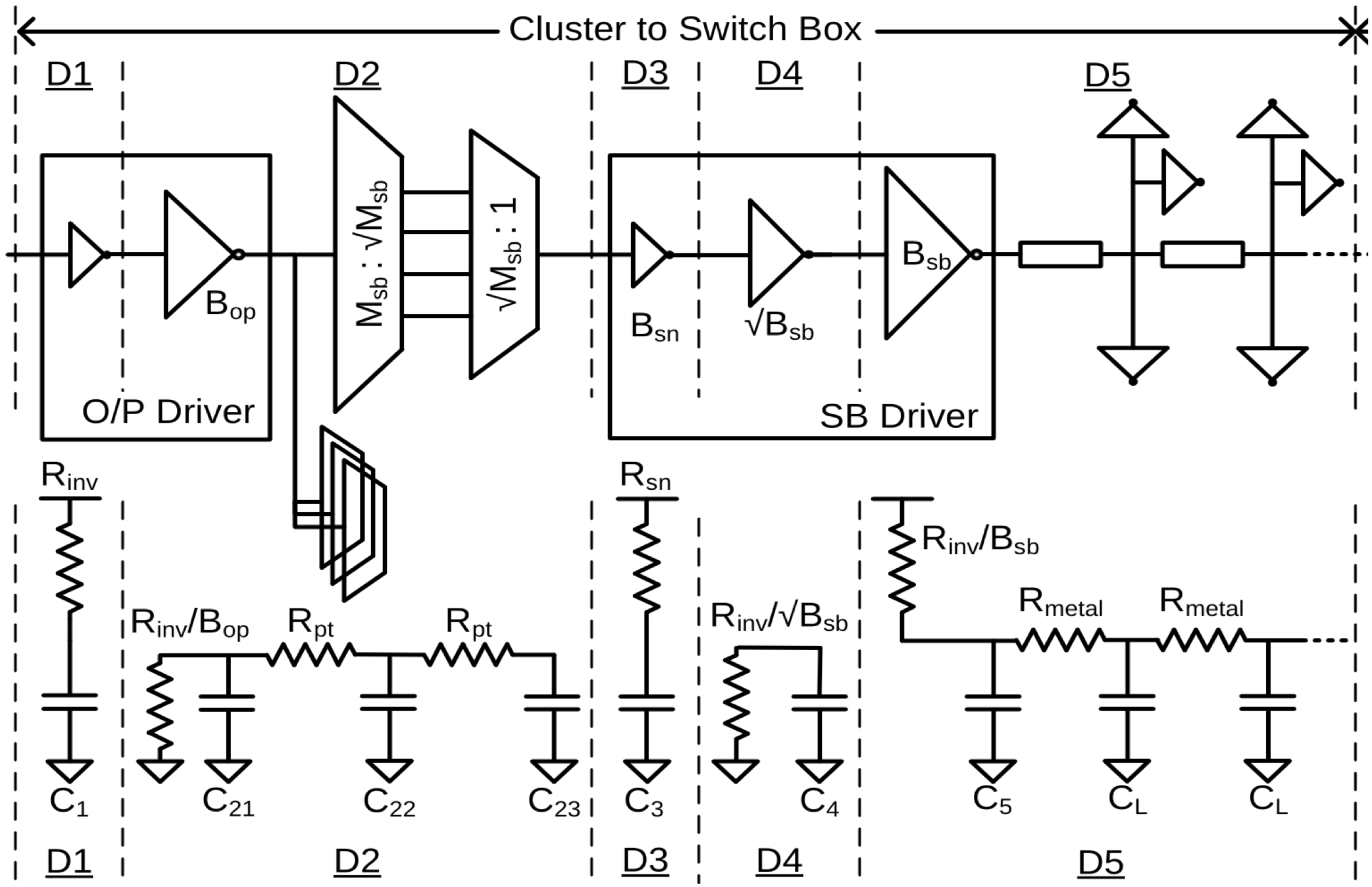
Circuit Assumptions: Cluster-Switch

- Single-Driver Routing



Circuit View

Circuit Assumptions: Cluster-Switch

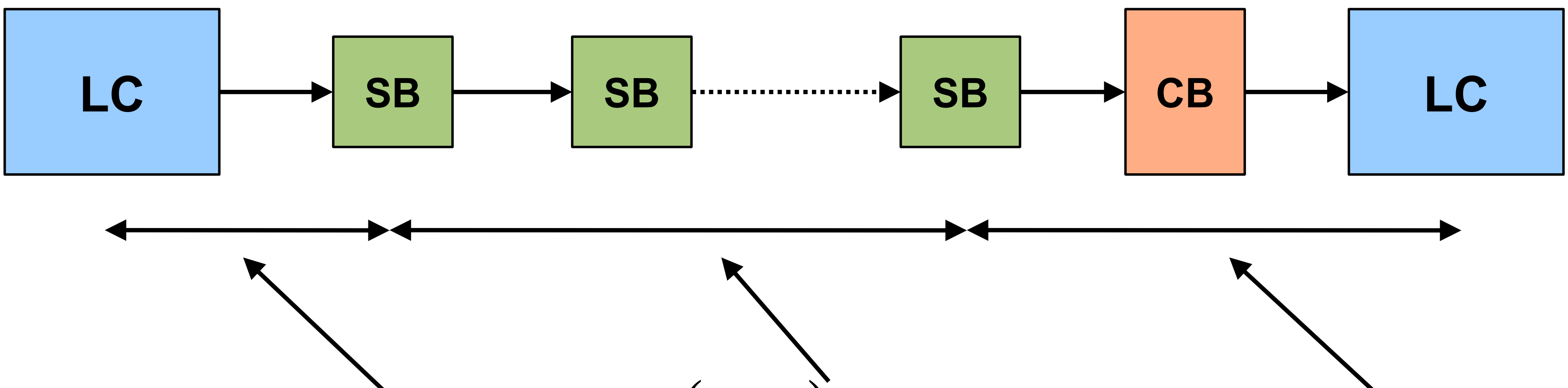


RC Network View

Global Interconnect Delay

- Similarly for and

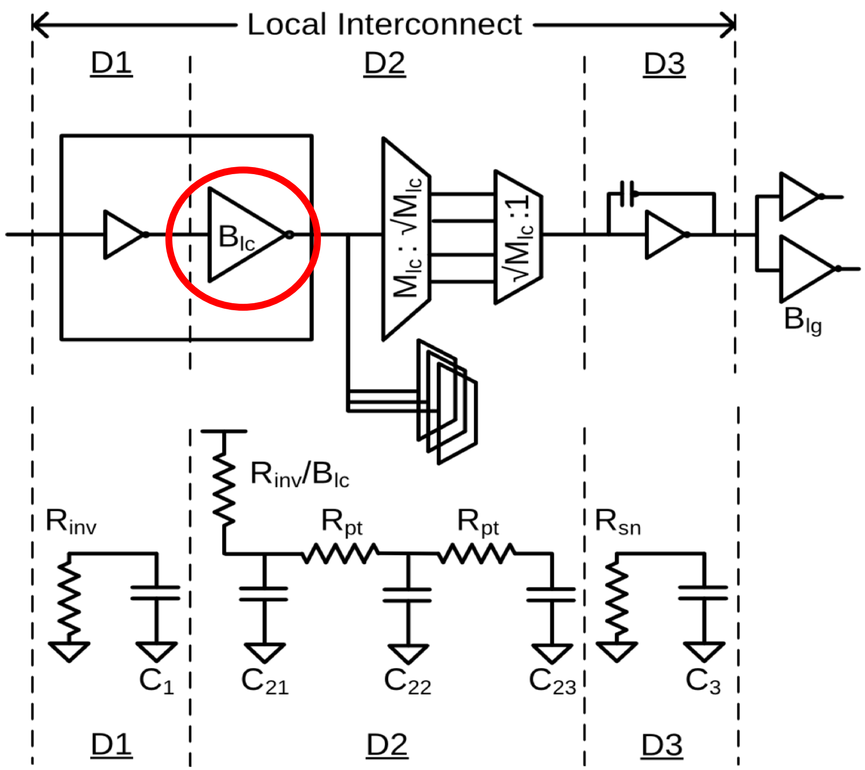
- Combining them:



$$T_{global} = T_{cluster-switch} + f(WL) \cdot T_{switch-switch} + T_{switch-cluster}$$

Optimal Buffer Sizing

- RC values depends on buffer sizing
  - Using equations: differentiate to find optimal size



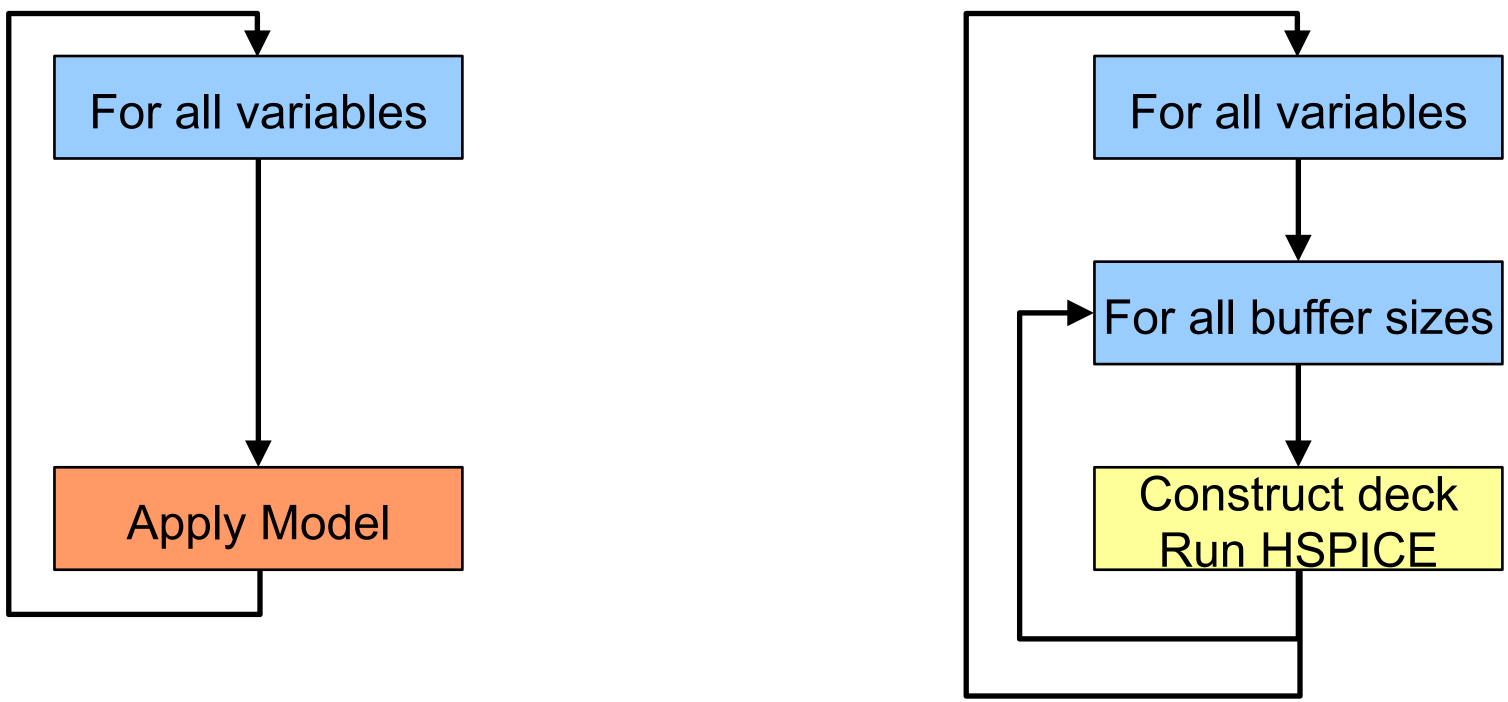
$$B_{lc} = \sqrt{\frac{C_{21}' + C_{22} + C_{23}}{0.69 C_{g,inv}}}$$

RC Network View

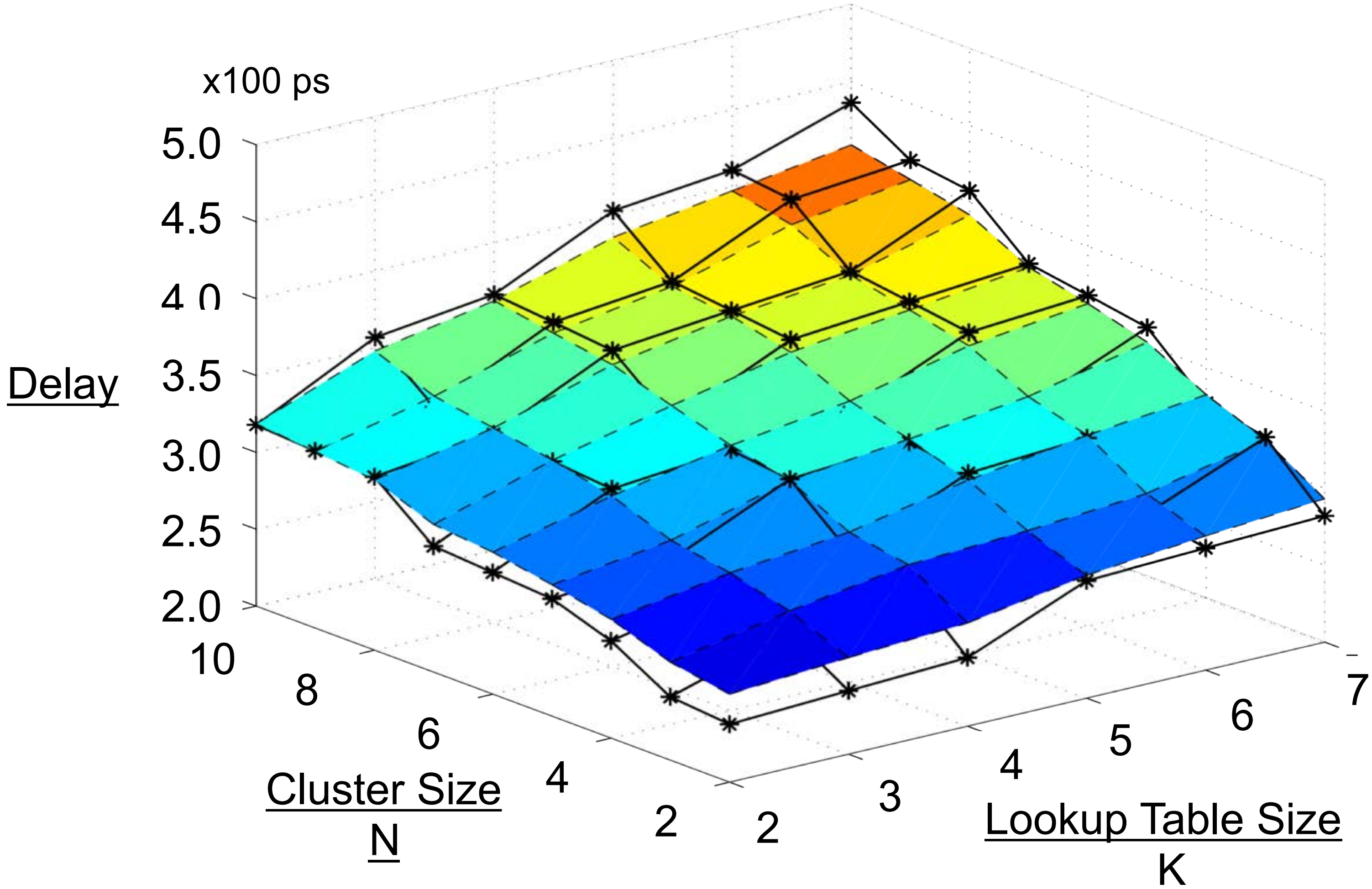
Model Validation

Model Validation: Methodology

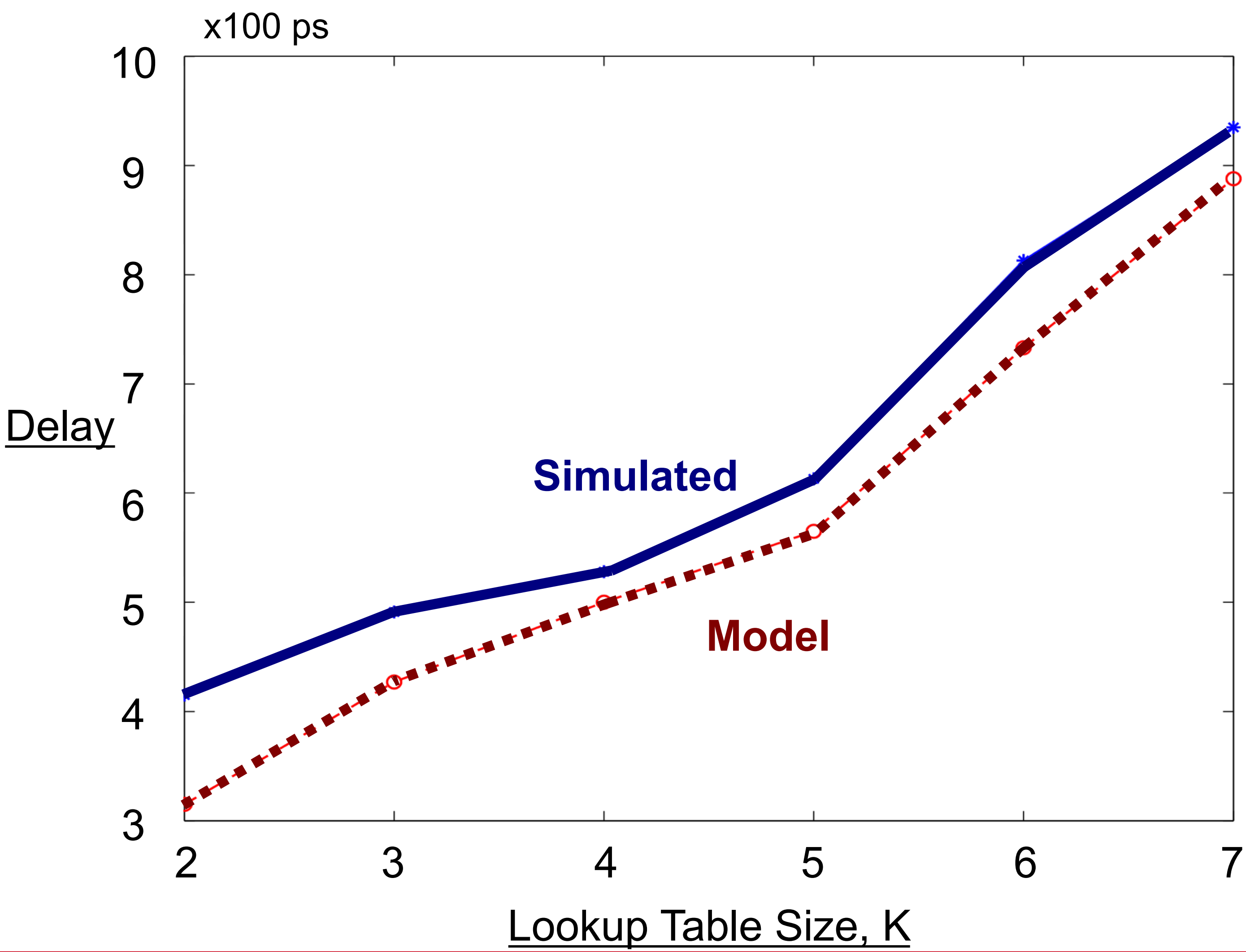
- Analytical Model



Model Validation: T_{local}

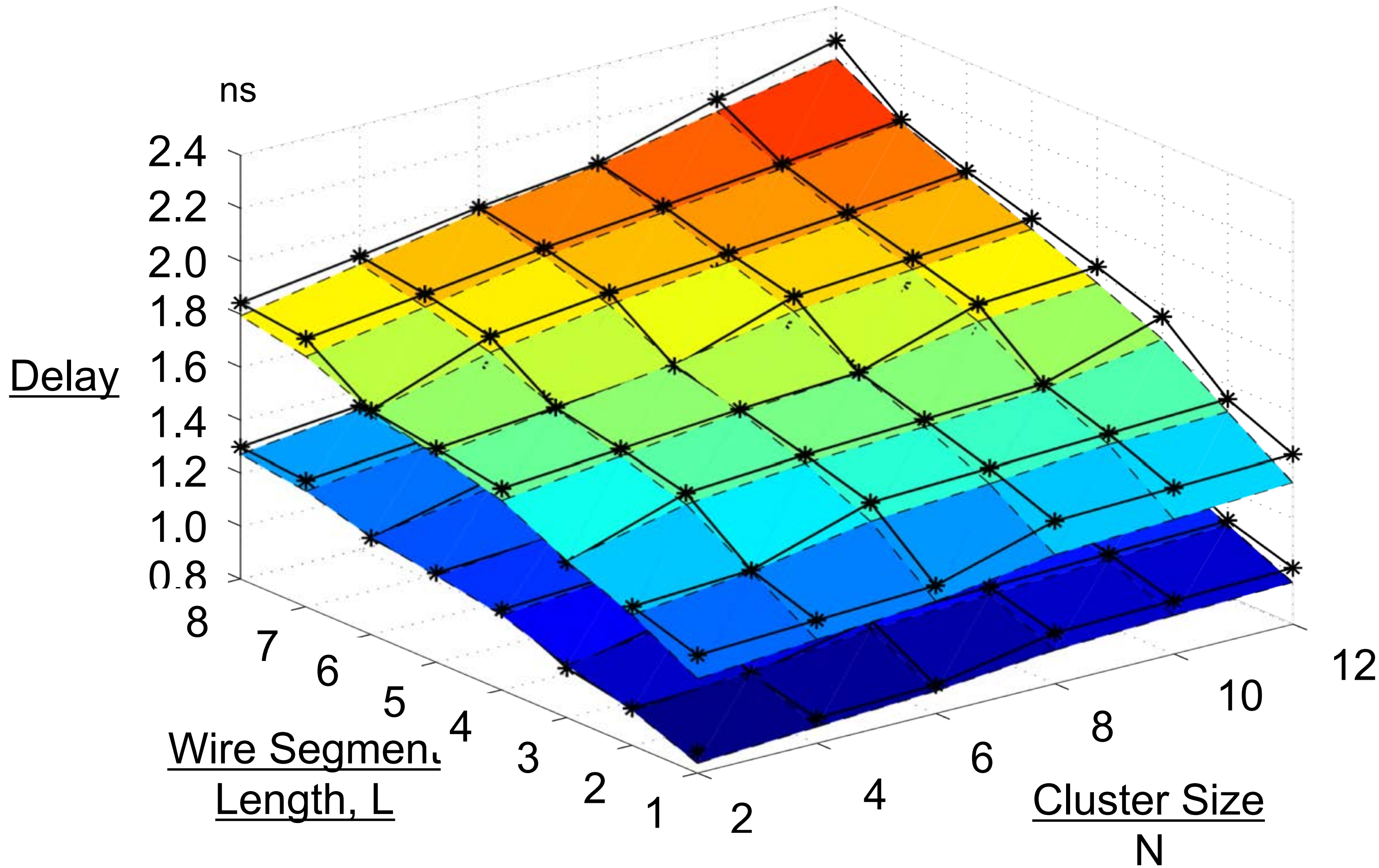


Model Validation: T_{logic}



THE UNIVERSITY OF SYDNEY

Model Validation: T_{global}



THE UNIVERSITY OF SYDNEY

Model Validation: Previous Work

- Consistent with previous methods
  - But orders of magnitude faster

		Experimentally Derived	
N	Our Model	Ahmed et al.	Our HSPICE
2	253	221	267
4	286	301	298
6	321	332	326
8	352	331	349
10	361	337	362

T_{local} (ps) for K=4 at 0.18um

Model Applications

(1) Speeding up FPGA architecture design

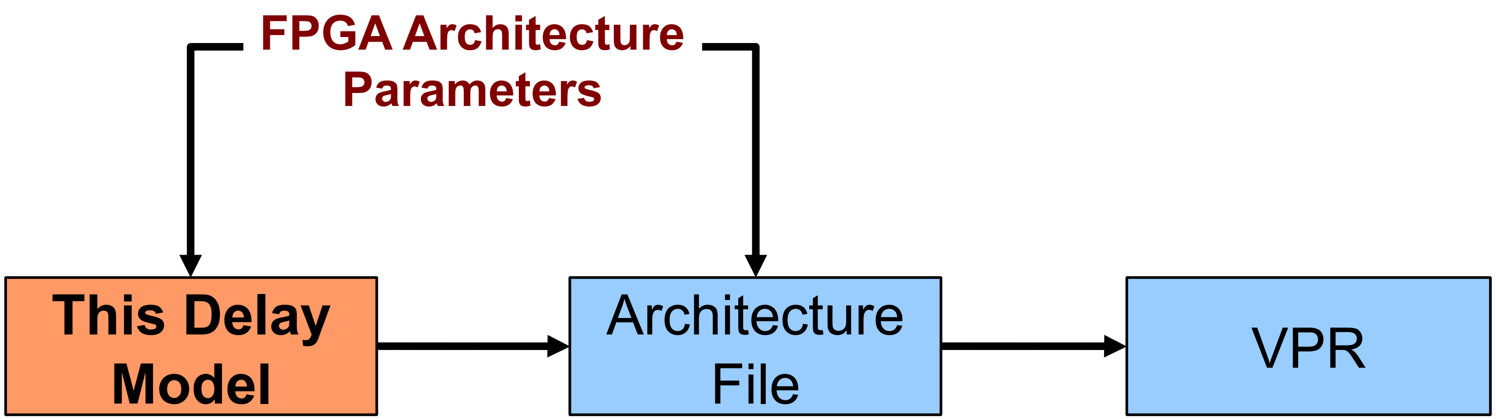
(2) In conjunction with experimental techniques

- Generate delays for use in VPR architecture files

(3) Gain additional insight into FPGAs

Application: Experimental Techniques

- VPR does not have a parameterised delay model for T_{local} and T_{logic}
  - Physical delays currently specified per-architecture
  - Can use our model to generate realistic delays



THE UNIVERSITY OF SYDNEY

Application: Gaining Insight

- Abstract away technology parameters
- Leaving behind a 'distilled' expression with architecture parameters only:

$$T_{local} \approx A_0 + A_1\sqrt{2N + K + NK} + A_2 NK$$

  - **Not possible** using experimental techniques

- Interesting insight:

N has about the same effect on delay as K

Conclusion

- Circuit-level description of FPGA presented
- Simple yet accurate delay model derived

- Future directions:
  - Incorporate more recent architectural developments
  - Investigate effects of process technology scaling
  - Develop associated area model to explore tradeoffs

THE UNIVERSITY OF SYDNEY

References

[1] Eddie Hung, Steven J. E. Wilton, Haile Yu, Thomas C. P. Chau, and Philip H.W. Leong. A detailed delay path model for FPGAs. In Proc. International Conference on Field Programmable Technology (FPT), pages 96–103, 2009.

[2] Joydip Das, Andrew Lam, Steven J.E. Wilton, Philip Leong, and Wayne Luk. An analytical model relating FPGA architecture to logic density and depth. IEEE Transactions on VLSI Systems, 9(12):2229–2242, 2011.