CS294-7: Reconfigurable Computing

Day 27: Tuesday, April 29

Everything Else...

- Admin
- Binding Time
- Specialization

Admin

- Xilinx/Multicontext, April 30, W5pm, Hogan Room
- Projects Due in 1 week
  - URLs by class time Tuesday
  - All presentations in 505 (even T afternoon)
  - 10 Minute talk slots (2-3 minutes questions)
- HKN at end of class

Specialization

Instantaneous computational needs are less than general case

→ especially when can predicated/select computation based on partial inputs.

Dual of *generalization* which we saw last time.

Arises:

- unfolding general structure
- folding in part of problem (partial evaluation)
- localized data values (common case optimization)

Today's Lecture

- Low Data Entropy
  - Binding Time
  - Probabilistic
- Specialization Benefits
- Issues
  - Instruction Load Time
  - Instruction Storage
  - Instance Generation
  - Variable/data-dependent space
  - Dynamic resource management

Low Data Entropy

"Opportunity" arises because data values are *not* completely random.

High amount of predictability (Low entropy)

Why?

- Values may be highly localized by nature of input data ("common cases")
- Data Processing Inequality (computed values have structure, not random)
- Binding Time – values often bound early, then reused for some operational epoch

Common Case

Data may take on one of only a few values:

- $P(x < 10) = 0.99$
- $P(x = 0) = 0.4$, $P(x = 1) = 0.3$, $P(x = 2) = 0.25$
- $P(x_i = x_{i-1}) = 0.8$
- $P(x_{i-1} - 2 \leq x_i \leq x_{i-1} + 2) = 0.98$

Data Processing Inequallity

Remember:

$$H(F(x)) \leq H(x)$$

Computation adds structure –
even if inputs are random ($H(x) = |x|$**),**
during computation, entropy reduced ($H(x) \leq |x|$**)**

value	$P(x_i)$	$P(x_i \vee x_{i+1})$	$P(x_i \vee x_{i+1} \vee x_{i+2})$	$P(x_i + x_{i+1})$	$P(x_i + x_{i+1} + x_{i+2})$	$P(x_i + x_{i+1} + x_{i+2} + x_{i+3})$
0	0.5	0.25	0.125	0.25	0.125	0.0625
1	0.5	0.75	0.875	0.50	0.375	0.25
2				0.25	0.375	0.375
3					0.125	0.25
4						0.0625
$H(x)$	1	0.71	0.54	1.50	1.81	2.03
$H(x)/\|x\|$	1	0.71	0.54	0.75	0.61	0.51

Binding Time

A value in a computation may be determined (bound), then used many times before it changes (if at all).

$$P(x = x_{bound}) = 1$$

Binding Time

There are a variety of binding time epochs:

- pre-fabrication (pixel widths in MPEG-2 decoder ASIC)
- application/algorithm selection (constant coefficients, structure)
- “compile” time (implementation values, data layout?)
- program startup (machine parameters, environment preferences)
- instantiation (instance variable)
- epochs w/in a program (between recalibration periods, changes in user preferences)
- loops (loop invariants)
- procedure/routine

Binding Time

Conceivably:

- specialize at binding time
- use specialized computation throughout bound domain

Binding Time

- static $\rightarrow P(x = x_{bound}) = 1$
  depends on identifying program structure
  (dismbiguating def/use)
- true, but not provable $\rightarrow P(x = x_{prev}) \approx 1$
  is dependent, but state-of-the-art in static analysis
  will not identify
- highly probably, but not guaranteed
  $\rightarrow P(x = x_{prev}) = 0.999$
  could be different, but highly unlikely
  (*e.g.* low, but not zero probability of aliasing
  (changed via side-effect))

Many computations may reduced to simply checking that input values unchanged.

Binding Time

Note difference between first (static/provable) case and last two

- $\rightarrow$ in first don't have to check
- $\rightarrow$ in last two, always have to check if valid
  - $\rightarrow$ checking O(input bits)

Data Entropy – Challenge

Can we capture/characterize the predictability of a task/application? (in a way which is useful to understand real computational requirements)

e.g. **In communications**
$\rightarrow$ **entropy tells us compressibility limits**
(even there, must start w/ model of data correlations)

- expected computational work reduction when values localized
- computational work as f(bound interval) (change frequency)
- computational work reduction when values highly predictable
- metric value static/dynamic-ness
  $\rightarrow$ impact on computational work

Specialization

Key idea: The more we know about the inputs, the less work we have to do to compute a function.

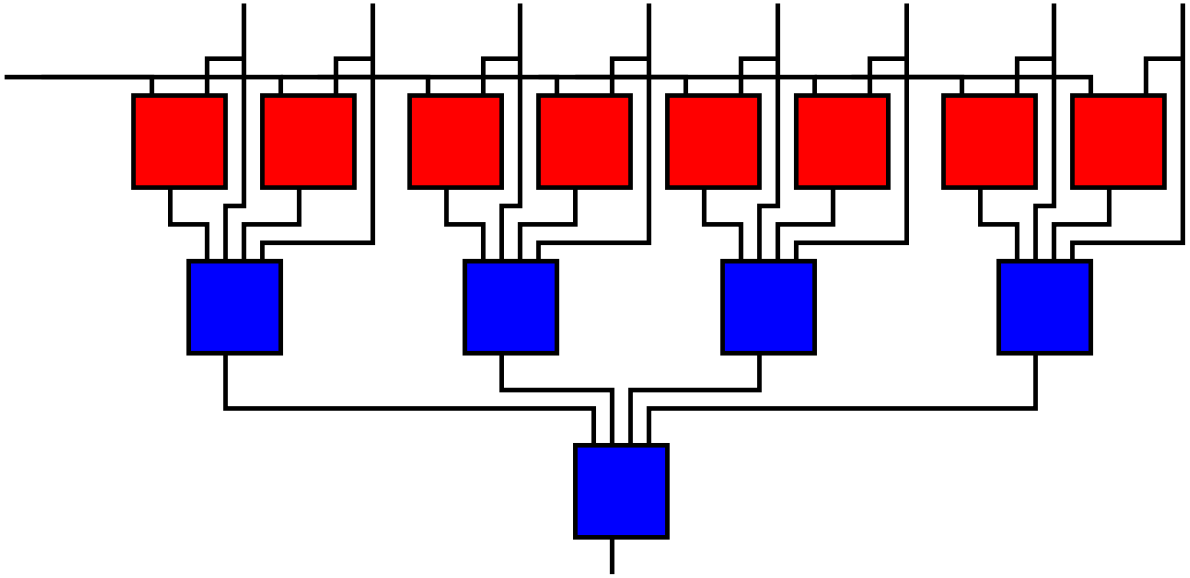
$$y = a * b + c * d | b = 0 \rightarrow y = c * d$$
$$y = a * b + c * d | b = 1 \rightarrow y = a + c * d$$
$$x + 10 > y | y > 2000, x < 100 \rightarrow 0$$
$$x + 10 > y | y = 2000 \rightarrow x > 1990$$

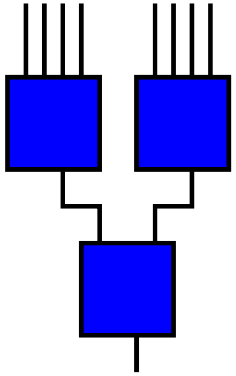
Specialization

May get savings from:

- removing state elements
- control paths to load
- logic simplification

e.g. Pattern Match





Specialization – fixed benefit (match)

(size)	CLBs	path	CLBs	path	AT Ratio	
$a = b$	b variable		b constant			w/state
(8)	2.5 (+4)	2	1.5	2	0.60	0.23
(16)	5.5 (+8)	3	2.5	2	0.30	0.12
(32)	10.5 (+16)	3	5.5	3	0.52	0.21
(64)	21.5 (+32)	4	10.5	3	0.37	0.15

Specialization – little benefit (DES F)

DES F Function:

Function (size)	Speed Mapped CLBs	Speed Mapped path	Area Mapped CLBs	Area Mapped path	Speed Mapped CLBs	Speed Mapped path	Area Mapped CLBs	Area Mapped path
F	k variable				k constant			
	290.5	7	282	8	$<$ 280	6	$\approx$ 250	7

Specialization – Variable (less than)

Area may depend on value:

Function (size)	Speed Mapped CLBs	Speed Mapped path	Area Mapped CLBs	Area Mapped path	Speed Mapped CLBs	Speed Mapped path	Area Mapped CLBs	Area Mapped path
$a \leq b$	b variable				b constant			
(8)	4	8	4	8	≤ 2	≤ 2	≤ 1.5	≤ 3
(16)	18.5	14	16.5	16	≤ 6.5	≤ 3	≤ 3	≤ 5
(32)	35	19	36	24	≤ 13.5	≤ 4	≤ 6	≤ 11
(64)	77.5	20	74.5	28	≤ 30	≤ 5	≤ 14	≤ 16

Specialization – UART FSMs

I8251 – Intel (PC) Standard UART

FSM	Fully Generic: Speed Mapped CLBs	Fully Generic: Speed Mapped path	Fully Generic: Area Mapped CLBs	Fully Generic: Area Mapped path	Specialized CLBs	Specialized path
I8251 processor i/o	11	3.5	11	3.5		
	fast (any configuration)				6.5	2
	small (any configuration)				5.5	2.5
I8251 transmitter	57.5	4.5	57.5	4.5		
	Asynchronous, parity				24	4
	Asynchronous, no parity				27	4.5
	2 Sync chars, parity				31	4.5
	1 Synch char, noparity				31	4
I8251 receiver	52.5	5.5	52.5	5.5		
	Asynchronous, parity				32.5	4.5
	Asynchronous, no parity				36	4.5
	External Sync, parity				29.5	4.5
	Internal, 2 Sync chars, parity				27	3.5
	Internal, 1 Sync chars, parity				28	4.5
	Internal, 1 Sync chars, no parity				31.5	4.5

Specialization – UART

design	Fully Generic				Specialized	
	Speed Mapped		Area Mapped			
	CLBs	**path**	CLBs	**path**	CLBs	**path**
I8251 core	358.5	8.5	348.5	10.5		
	Async, 64 clks/bit, 8e2				216.5	7
	Async, 16 clks/bit, 8n1				201	6
	Async, 1 clks/bit, 5n1				141.5	4.5
	Sync, internal, 2 sync, 8o				165	4.5
	Sync, external, 5n				136	5.5

Specialization – Multiply

Generic $n \times n$ Multiply:

- n^2 ands
- n n-bit adds

Specialized:

- naive: one add per on bit in constant

Specialization – Multiply

Never more than $\lfloor \frac{n}{2} \rfloor$ one bits in constant

- invert c ($2^{n+1} - 1 - c$)
- add 1 ($2^{n+1} - c$)
- multiply by x ($\left(2^{n+1} - c\right) \cdot x$)
- subtract from $2^{n+1} \cdot x$

At most: $\lfloor \frac{n}{2} \rfloor + 2$ adds for any constant
(most cases much smaller; ? tighter worst-case bound?)

N.B. requires more than constant propogation.

Specialization – Bounding

Knowing that a value will be constant — can get a bound on area (time).

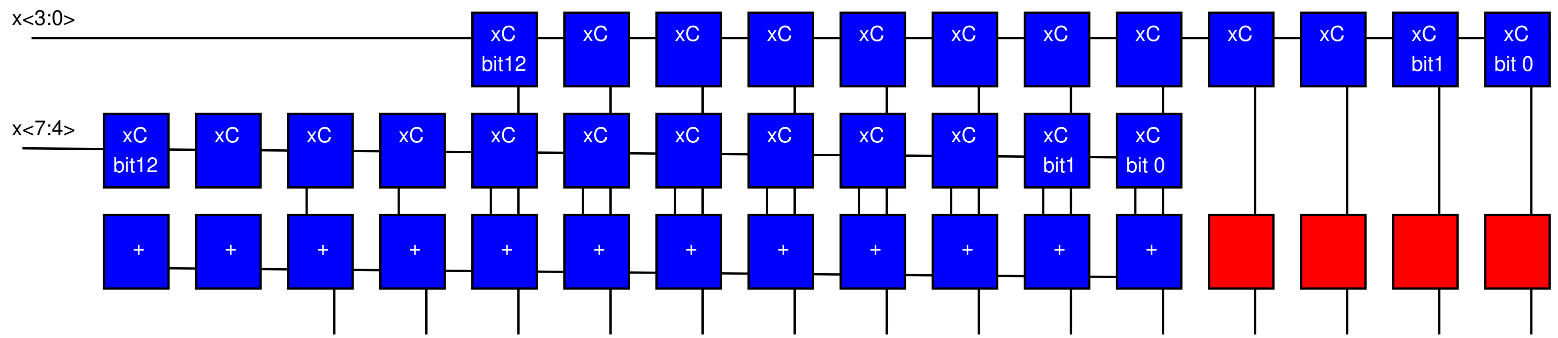
i.e. there is often a bound on (A op Constant) which is tighter than (A op B) and independent of Constant.

Examples:

- Pattern Match
- Less than
- Multiply

Specialization – Bounding Multiply

Area	Time	16×16 mpy	16×16 scale	8×8 mpy	8×8 scale
316 CLBs	26 ns	8216 (1.00)			
84 CLBs	40 ns		3360 (0.41)		
220 CLBs	12.1 ns			2662 (0.32)	
22 CLBs	25 ns				550 (0.07)



Specialization – Savings

Specialization savings depends on:

- which data known
- their values
- computational structure

Useful metrics for specializability?

- given PDF (on early bound values) $\rightarrow E(A \cdot T)$
- impact of specializing particular inputs to a function
- variability associated with specializing particular inputs
- ???

Tie in with data entropy/predictability
$\rightarrow$ predict/bound real computational work in a problem (like real information content in a datastream)

Instruction Load

Loading specialized instances into device takes time:
(assume cannot compute during load)

$$
\begin{aligned}
T_{task} &= T_{load} + Tsc \\
AT_s &= A_s \cdot (T_{load} + Tsc) \\
AT_g &= A_g \cdot T_g \\
A_s T_{load} + A_s T_{sc} &\leq A_g T_g \\
\frac{A_s T_{load}}{A_g T_g} + \frac{A_s T_{sc}}{A_g T_g} &\leq 1
\end{aligned}
$$

- Compute long enough with special value
  $\rightarrow$ ammortize out load time
- Short runs, benefit diminished by load time
- Still profitable if benefit large
  even when $T_{load} > Tsc$

Instruction Load

With background load (and can predict next case in advance):

$$AT_s = A_s \cdot \mathrm{max}(T_{load}, T_{sc})$$

$T_{load} < T_{sc} \rightarrow$ **no load penalty**

Instruction Space

Remember, instructions take up space, too:

$$AT_s = A_s \cdot (T_{load} + T_{sc})$$
$$A_s = A_{active} + A_{instruction}$$

	4-LUT Area	bits	SRAM/inst	DRAM/inst
Xilinx	640Kλ^2	$\approx$ 200	240Kλ^2	20Kλ^2
Model	880Kλ^2	64	80Kλ^2	6.4Kλ^2

Nick's sequence match – 8 4-LUTs/char

- $8 \cdot 200 = 1600$ **bits (versus 2 bits in problem)**
- $\rightarrow 800\times$ **more memory**
- **If storing large database** $\rightarrow A_{memory} >> A_{sc}$

Instruction Space

Nick's sequence match – 8 4-LUTs/char

– with relocatable macros:

- only need to save 16 different configurations
- compose using original 2 bit/char string

Runtime Codegen

One way to avoid space requirement is to compute specializations on the fly
(especially since potential cases exponential in number of bound bits)

(Load equations $T_{load} \rightarrow T_{gen}$)

$$\begin{aligned}
T_{task} &= T_{gen} + Tsc \\
AT_s &= A_s \cdot (T_{gen} + Tsc) \\
AT_g &= A_g \cdot T_g \\
A_s T_{gen} + A_s T_{sc} &\leq A_g T_g \\
\frac{A_s T_{gen}}{A_g T_g} + \frac{A_s T_{sc}}{A_g T_g} &\leq 1
\end{aligned}$$

Variable Space

Fully exploiting reduced area requirements of specialized instances $\rightarrow$ variable space requirements

Amount of simultaneous computation depends on data.

Many problems can exploit this
- particularly when have to fold problem to fit on device
- (get more simultaneous instances with smaller specialization)

Online composition and packing becomes a must.

In more sophisticated cases: online resource management, allocation, scheduling.

Review

- Low data entropy arises from
  - bound values
  - derived/correlated values
  - probabilistic value localization
- Specialized instances
  - smaller and faster (less $A \cdot T$)
  - size bound (tighter than general case)
  - savings depends on function structure
  - data dependent $A \cdot T$ in general
- Pay attention to
  - instruction load (generation) time
  - instruction area