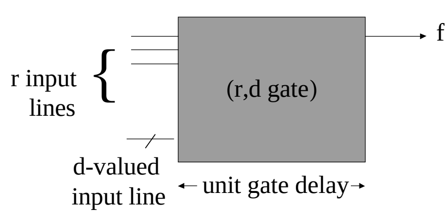
EE 486 : lecture 5, bounds on Arithmetic (Winograd's bounds)

M. J. Flynn

Bound on add

- Based on (r,d) gate model.
- The (r,d) gate can compute any r input function in a d valued truth system in a single gate delay.
- Model bounds number of fan-in limited gate delays, ignores: fan-out, wires or any other constraint.

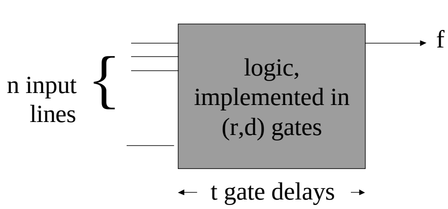
The (r,d) gate



Spira's lemma

- If a d-valued output f is a function of all n d-valued input arguments, then t, the number of (r,d) gate delays, is determined
- t is greater than or equal to ceiling [$\log_r$ n] in units of gate delays.

Spira's lemma



Winograd's bound on add

- The add bound is the application of Spira's lemma to the optimal *rns* representation. So, the time for addition is at least, t which is greater than or equal to
- [$\log_r$ 2[$\log_d$ α(N)]],where [ ] imply ceiling functions.
- For binary numbers the reduces to [$\log_r$ 2n] for the addition of 2 n bit numbers.

Winograd's add bound

- For the bound, the $\alpha(N)$ is the largest modulus while $[\log_d \alpha(N)]$ is simply the number of input lines needed to represent $[\alpha(N)]$
- For a prime base (eg 2), $\alpha(2^{12}) = 2^{12}$ and for a composite base, select the largest base factor (eg 10, or bi-quinary), $\alpha(10^{12}) = 5^{12}$

Winograd's add bound

- Bound can be for exactly N (or M in the *rns*) or for a representation that has equal or greater capacity. The latter usually gives a better bound. We call this $\alpha(>M)$. To find this we use the optimal *rns* algorithm and continue until the prime or prime power product is equal to or exceeds M.

Winograd's add bound

- Despite its limitations the bound can be closely approached or even bettered (using some sort of DOT function) by avoiding the requirements of the (r,d) gate.
- Indeed the bound doesn't apply at all for a redundant number representation (*rnr*); we'll see more of this later.

Bound on Multiply

- Represent numbers as composite of prime factors or powers of prime factors.
- This is the best representation for numbers that are to be multiplied or divided.
- Just add/subtract the corresponding prime factor exponent.

The log number sys., a practical realization of the multiply bound

- X can be represented by a sign + log (X).
- In *lns* $L_x = S_x$ (interger L_x).(fract L_x) this requires n = 1 + k + f bits. S_x is 0 if X is + and 1 if X is negative.
- $X = (-1)^{Sx} \times 2^{Lx}$; of course the base need not be 2.
- To represent numbers smaller than 1 use a bias (as in *fpns*).

More on the log number system

To represent numbers less than 1.0 use a bias, so $L_x = S_x$ $(iL_x).(fL_x)$ – bias where the bias would be 2^{k-1} or $2^{k-1} - 1$ (same as *fpns*)

Now multiply and divide become easy

$L_{XY} = L_X + L_Y$ and $L_{X/Y} = L_X - L_Y$; of course add/subtract now are much more complicated

X + Y = X (1+ Y/X) which requires a table lookup for the (1+ Y/X)

More on the log number system

Net… the *lns* is interesting only in special applications … signal processing, etc. and usually is restricted to numbers with limited precision.

Back to Winograd's bound on Multiply

- Similar reasoning to the add bound but uses a "log" or exponential representation for arguments. Result is surprising since it shows that the multiply delay bound is always as good as or better than the add delay.
- $[\log_r 2[\log_d \beta(N)]]$,where [ ] imply ceiling functions

Bound on Multiply

- Note that $\alpha(N) > \beta(N)$ for all N
- In particular:
  - Binary numbers $\beta(2^n)$ is 2^{n-2}
  - Prime base $\beta(p^n) = \max(p^{n-2}, \alpha(p^{n-1}))$
  - Composite base (base is $p_1 \times p_2$)= $\max(\beta(p_i^n))$

Bounds

- Bounds on add, multiply use different representations (non compatible)
- Bound on add can be used as bound on multiply on the optimal *rns*.
- All in all, it's hard to beat binary!

What about redundant numbers?

- The bounds don't apply.
- In *rnr* the carry is usually limited to one digit.
- So its always fixed, something like $[\log_r 2\beta]$ but the actual radix and the redundant radix may differ so it's a bit more complicated.

What about table look up?

- We can develop a fan in limited gate model for tables. This is NOT a bound, but serves as an indicator for comparisons.
- Assume a 2-D storage array, with a unit delay for storage itself. This array is addressed by n address bits; n/2 address the X decoder and n/2 address the Y decoder.

Table look up

- The X lines select a row, all $2^{n/2}$ elements in the row are accessed. The Y decoder selects the correct output line. All $2^{n/2}$ Y lines are ORed to an output.
- So X decode = $[\log_r n/2]$, Y decode = $[\log_r (n/2)+1]$, ORing the $2^{n/2}$ Y output lines = $[\log_r 2^{n/2}]$
- We could sum these terms, but a better model is available

Table look up

- We can overlap the X and Y decode, then the Y line selects a single gate which is then ORed as before.
- Now, the delay is = $2[\log_r n/2] +1+1+ [\log_r 2^{n/2}]$
- The first term is the X and Y decode, next the store gate, then the Y line select, then the OR logic.

So, which is better, tables or logic?

- For our table model, it's pretty clear that specific logic implementations will give better (smaller) gate delays. There are exceptions when the operand sizes are small
- We'll use tables in divide, square root and the higher level functions… these will all tend to be small "starter" tables.
- Note that a better table model is possible using n dimensional $(n > 2)$ tables.

So, which is better?

- Again it's hard to beat binary, but there are special application where either the residue or the log number systems work well.
- As we'll see tables and the redundant number representation. will play a role in getting the best in arithmetic design.