Special Purpose Machines

What is the best way to build a VLSI chip to solve problem X?

Best = Minimum area, Minimum time, ...

X = Matrix multiplication, sorting, FFT, ...

Upper bounds:

- Solve the problem using Mesh/Butterfly/... and layout the network in VLSI.
- Devise new networks better suited for VLSI.

Lower bounds:

Prove: "No matter what network is used, area/time/... have to be at least ..."

Lower Bounds

Memory based bounds: $A \geq f$

"At some point while solving problem, it is necessary to remember f words."

"Chip must have area at least f to remember f words."

I/O based bounds: $AT \geq f$

"Input/Output consists of f words

"Every chip with area A can read at most A words per time step."

Bisection based bounds: $AT^2 \geq f^2$

"At most $\sqrt{A}$ words can flow from one half of chip to other half in one step."

"Problem requires at least f words to flow."

Model

Each processor can store 1 word. (w bits)

Each processor can read/write 1 word per step.

Each processor can send 1 word on each link at each step.

Each input word is read just once.

Input/Output is where/when oblivious.

INPUT: $X : x_1, x_2, \ldots, x_m$

OUTPUT: $Y : y_1, \ldots, y_n$

Where Oblivious: Which processor reads x_i (generates y_j) is fixed before beginning of execution.

When Oblivious: When x_i is read (y_j generated) is fixed before beginning of execution.

Lower bounds on chip area

Chip state = Values stored in the processors. P processor chip can have 2^{Pw} states.

Chip Behaviour = Values generated as ouptuts. n output chip can have $\leq 2^{nw}$ behaviours.

Chip Behaviour = f(initial chip state, chip inputs)

Basic Observation: Chip produces different behaviour in two executions given the same input $\Rightarrow$ Chip state was different before inputs were given.

Typical Argument: Prove that given chip produces different behaviours after time t given same input after time t in executions $E_1, E_2, \ldots, E_N$.

$\Rightarrow$ Chip state at time t distinct in all executions.

$\Rightarrow 2^{Pw} \geq N$

$\Rightarrow P \geq \frac{\log N}{w}$

$\Rightarrow$ Chip Area $\geq \frac{\log N}{w}$

Cyclic Shift Lower bound

INPUT 1: $x_{n-1}, x_{n-2}, \ldots, x_0$ "Data"

INPUT 2: s "shift amount"

Assume $w \geq \log n$

OUTPUT: $y_{n-1}, y_{n-2}, \ldots, y_0$

where $y_i = x_{i+s) \bmod n}$

Lemma: Cyclic Shift requires $A = \Omega(n)$

Claim: All data words must be read before any output word is generated.

Proof of Lemma: Let t = time instant when all data words read, no output word generated.

Consider executions with shift amount $s = 0$.

Number of behaviours for the chip : 2^{nw}

$\exists\ 2^{nw}$ distinct executions in which same input is read after time t, and in which distinct behaviour is produced after time t.

$\Rightarrow A = \Omega(\frac{\log 2^{nw}}{w}) = \Omega(n)$

Proof of Claim

Suppose y_j is generated before x_i is read in some execution.

Obliviousness $\Rightarrow$ above is true for all executions.

Set $s = i - j \bmod n$

$\Rightarrow$ Chip must output $y_j = x_{j+s \bmod n} = x_i$

We can feed different value for x_i after seeing what chip outputs as y_j.

$\Rightarrow$ Chip forced to make mistake. Contradiction.

Informal Interpretation: There exists a time instant t at which chip must have all n data words in memory. So n processors are necessary, and hence n area.

Dependence

INPUTS: $X : x_0, x_1, x_2, \ldots, x_{n-1}$

OUTPUTS: $Y : y_0, y_1, y_2, \ldots, y_{m-1}$

y_i depends upon x_j iff:

There exist two executions in which

1. Distinct values are assigned to input x_j.
2. Every other input has the same value in both executions
3. y_i takes on a different value in the two executions.

Example: In cyclic shift every output word depends upon every data word.

Implication of dependence: If y_i depends upon x_j then x_j must be read before y_i is generated.

Dependence between Sets

INPUTS: X, Subset X' : $x_0, x_1, x_2, \ldots, x_{n-1}$

OUTPUTS: Y, Subset Y' : $y_0, y_1, y_2, \ldots, y_{m-1}$

Y' f-depends up X' iff:

1. y_i depends upon x_j for all i, j.
2. There exist f executions in which:
   (a) The values given to the inputs in $X - X'$ are the same.
   (b) The outputs Y' take on distinct values in each execution.

Main Theorem: If Y' f-depends upon X' then Area $= \frac{\log f}{w}$.

Proof: All X' must be read before any Y' is generated.

After all X' is read but before any Y' is generated, chip will be in f possible states.

Dependence with Identity Map

INPUTS: X, Subset $X' : x_0, x_1, x_2, \ldots, x_{n-1}$

OUTPUTS: Y, Subset $Y' : y_0, y_1, y_2, \ldots, y_{n-1}$

Y' depends upon X' with identity map iff:

1. y_i depends upon x_j for all i, j.
2. There exist 2^{nw} executions in which:
   (a) The values given to the inputs in $X - X'$ are the same.
   (b) The input X' take distinct values.
   (c) The outputs Y' take on exactly the same values as the inputs X'.

Theorem: If Y' depends upon X' with identity map then Area $= \Omega(n)$.

Proof:

Y' depends upon X' with identity map

$\Rightarrow Y'$ 2^{nw}-depends upon X'.

Integer Multiplication

INPUTS: $a_{n-1} \ldots a_1 a_0 \quad b_{n-1} \ldots b_1 b_0$

OUTPUT: $c_{2n-1} \ldots c_1 c_0$

(a_i, b_j, c_k are all bits)

Theorem: A $= \Omega(n)$ (bit model)

Proof:

X' : $b_{n-1} \ldots b_{n/2}$

Y' : $c_{3n/2-1} \ldots c_n$

Every bit in X' depends upon every bit in Y'.

... with $a =$ power of 2.

Identity map with $a = 2^{n/2}$.

Integer Addition

Lemma: Addition modulo 2^n requires area $\theta(1)$.

"2s complement addition"

Proof:

Lemma: Addition modulo $2^n - 1$ requires area $\Omega(n)$.

"1s complement addition"

Proof: Homework

Summary of Area Bounds

Word model: $w = \theta(\log n)$

Cyclic shift	$\Omega(n)$
Sorting	$\Omega(n)$

Bit model:

Integer Multiplication	$\Omega(n)$
Addition mod 2^n	$\theta(1)$
Addition mod $2^n - 1$	$\Omega(n)$.
Convolution	$\Omega(n)$
Sorting n integers each $1 + \log n$ bits wide	$\Omega(n)$
Sorting n integers each $2 \log n$ bits wide	$\Omega(n \log n)$

Exercise: Find matching upper bounds.

AT^2 Bounds

Behaviour of L : Values output by L.

(Communication) Transcript of C : Concatenation of all the words transmitted across C over time.

Behaviour of $L =$ f(Values input inside L, Transcript(C))

Basic idea: L has different behaviour in two executions though same values input inside L

$\Rightarrow$ Transcript(C) is different in the two executions.

Typical Argument: Prove that L has different behaviours in executions $E_1, E_2, \ldots, E_f$ although same external values are input in L.

$\Rightarrow$ Transcript(C) is different in each execution.

Number of different transcripts possible:

$T =$ execution time

$|C|$: number of wires crossing C.

Number $\leq 2^{w|C|T}$

Main Lemma

Suppose $|C| \leq h + 1 \leq \sqrt{A} + 1$.

Suppose L has different behaviours in executions $E_1, E_2, \ldots, E_f$ although same values are input in L in all executions. Let

T : Time to finish computation.

Then $AT^2 = \Omega(\frac{\log^2 f}{w^2})$

Proof: $f \leq 2^{w|C|T} \leq 2^{w(\sqrt{A}+1)T}$
$\Rightarrow \frac{\log f}{w} \leq (\sqrt{A} + 1)T \leq 2\sqrt{A}T$

"C = small cut" : $|C| \leq \sqrt{A} + 1$.

Values input inside R will be different in $E_1, \ldots, E_f$.

Cyclic Shift

INPUT 1: $x_{n-1}, x_{n-2}, \ldots, x_0$ "Data"

INPUT 2: s "shift amount"

Assume $w \geq \log n$

OUTPUT: $y_{n-1}, y_{n-2}, \ldots, y_0$

where $y_i = x_{i+value(s) \bmod n}$

Lemma: Cyclic Shift requires $AT^2 = \Omega(n^2)$

Upper Bound Comparison:

$\sqrt(n) \times \sqrt(n)$ mesh : $T = O(\sqrt{n})$, $A = O(n)$

$\Rightarrow AT^2 = O(n^2)$: Optimal. Area cannot be reduced without increasing time, and vice versa.

n node hypercube: $T = O(\log n)$, $A = O(n^2)$

$\Rightarrow AT^2 = O(n^2 \log^2 n)$: Suboptimal. Improvement possible in time or area without increasing the other.

n node Butterfly: $T = O(\log n)$, $A = O(n^2/\log^2 n)$

$\Rightarrow AT^2 = O(n^2)$: Optimal.

Proof Outline

Proc P is output-heavy : P outputs $\geq n/3$ values.

Output-heavy proc $\Rightarrow T \geq n/3 \Rightarrow AT^2 = \Omega(n^2)$

Assume no processor is output-heavy:

Partition Lemma: There exists small cut C and constant α such that $X' = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{n/6}}\}$ are input on one side and $Y' = \{y_{i_1-\alpha}, x_{i_2-\alpha}, \ldots, x_{i_{n/6}-\alpha}\}$ are output on the other side.

WLOG X' is input in R, Y' is output in L.

Consider $2^{wn/6}$ executions in which

1. All $x_i \notin X'$ are given value 0. $s = \alpha$.
2. X' takes distinct values in each execution. X' has $wn/6$ bits $\Rightarrow$ possible.

$\Rightarrow y_{i_j-\alpha} = x_{i_j}$ in all executions.

$\Rightarrow Y'$ is different in all executions.

Transcript theorem applies.

$AT^2 = \Omega\left(\frac{\log^2 2^{wn/6}}{w^2}\right) = \Omega(n^2)$

Proof of Partition Lemma

Claim: There exists small cut C s.t. at least $n/3$ outputs are generated on each side.

Proof of claim: Modified sliding argument.

WLOG assume side L reads at least $n/2$ inputs.

Inputs on side L: $\quad x_{i_1}, \quad x_{i_2}, \quad \ldots, \quad x_{i_{n/2}}$

Outputs on side R: $\quad y_{j_1}, \quad x_{j_2}, \quad \ldots, \quad x_{j_{n/3}}$

Put complete bipartite graph on above inputs/outputs.

Assign colours to edges: $(x_p, y_q) = p - q \bmod n$

$0 \leq \text{Colour} < n$.

$n^2/6$ edges $\Rightarrow$ At least $n/6$ edges of same colour.

colour(x_p, y_q) = shift count which will force y_q to take value x_p.

X', Y' = endpoints of edges with most popular colour.

Summary

Idea 1: C = small cut, X' = subset of chip inputs, Y' = subset of chip outputs s.t.

1. $|X'| = |Y'|$.
2. X' are read on the left of C and Y' generated on right.
3. By holding the rest of the inputs fixed at some special value, it is possible to force $Y' = X'$, for all assignments to X'.

Then $AT^2 = \Omega(|X'|^2)$

Idea 2: Assume no processor is output-heavy. Then it is possible to find a small cut such that each side of the cut generates $n/3$ outputs, where n = number of outputs.

Idea 3: Averaging idea to show the existence of colour with at least $n/6$ edges.

Ideas 1, 2 are applicable in general. Idea 3 is also useful.

Boolean Matrix Multiplication

INPUTS: $n \times n$ matrices A, B.

OUTPUTS: $n \times n$ matrix C, $C = AB$.

Theorem: $AT^2 = \Omega(n^4)$.

Proof Outline:

1. Assume no processor generates $\geq n^2/3$ outputs.

2. $\exists$ cut π such that $\geq n^2/3$ bits of C are generated on both sides.

3. One of the following is true:

   (a) $n^2/6$ bits of A are generated on one side of π which can be forced to move across the cut by setting B to a shift matrix.

   (b) $n^2/6$ bits of B are generated on one side of π which can be forced to move across the cut by setting B to a shift matrix.

Graph G

VERTICES: one vertex for each element of A, B, C.

EDGES: $(a_{ij},\ c_{i,j+k \bmod n})$ of color k
$(b_{ij},\ c_{i+k \bmod n,j})$ of color $n+k$

Lemma: Any partition of G with at least $n^2/3$ vertices of C on each side has $\Omega(n^3)$ edges crossing the partition.

$\Rightarrow$ $\Omega(n^2)$ edges of a single color cross the partition.

$\Rightarrow$ If color $\leq n$, use shift matrix for B; else use shift matrix for A.

$\Rightarrow$ $\Omega(n^2)$ bits flow from either A or B across π.

$\Rightarrow$ Main theorem proved!

Proof of Lemma

Embed complete directed graph $K_{n^2} \to C$

(Edges of $K_{n^2} \to$ paths of G)

$\alpha =$ congestion of embedding.

$\beta =$ number of edges of G in the partition.

Key observation:

$\beta\alpha \geq$ paths crossing the partition $\geq \frac{n^2}{3}\frac{n^2}{3}$

Next we prove: $\alpha = n$

$\Rightarrow \beta \geq n^3/9$

Estimating congestion α

EMBEDDING:

Path $(c_{ij}, c_{kl}) \to (c_{ij}, a_{ik}), (a_{ik}, c_{il}), (c_{il}, b_{jl}), (b_{jl}, c_{kl})$

Congestion of (c_{ij}, a_{ik}) (directed):

paths carried $(c_{ij}, c_{k*}) \Rightarrow$ Congestion $= n$

Congestion of (a_{ik}, c_{il}) (directed):

paths carried $(c_{i*}, c_{kl}) \Rightarrow$ Congestion $= n$

Congestion of (c_{il}, b_{jl}) (directed):

paths carried $(c_{ij}, c_{*l}) \Rightarrow$ Congestion $= n$

Congestion of (b_{jl}, c_{kl}) (directed):

paths carried $(c_{*j}, c_{kl}) \Rightarrow$ Congestion $= n$

Upper bounds for matrix multiplication

$n \times n$ **Mesh:**

$A = O(n^2)$

$T = O(n)$

$AT^2 = O(n^4)$ Optimal!

n^3 **processor Hypercube:**

$A = O(n^6)$

$T = O(\log n)$

$AT^2 = O(n^6 \log^2 n)$ Very suboptimal.

Using specialized networks it is possible to get optimal AT^2 even for small T.