

ECE 497NC: Unconventional Computer Architecture

Lecture 13: Biological Computing

What is Biological Computing?

- Using biological processes to perform computation
 - Most efforts focus on DNA-based reactions
- Tremendous parallelism
- Slow cycle times
 - Biological gates may operate at millihertz frequencies
- Leverage ability of biological processes to manipulate matter at the molecular level
- Interface with an organism to be studied/manipulated

Approach 1: Construct Logic Gates Within Cells

- Basic question: Can we use biological processes within cells to implement logic gate-like functions?
- Things to consider:
 - What to use to carry signals
 - Need to be able to generate
 - Need to be able to use as input
 - Need underlying process with high gain for 0-1 mapping
 - How to input signals to the system
 - How to read outputs

A Little Biology

- DNA contains the basic information used to replicate cells and proteins
- Protein manufacture is a two-step process:
 1. Information in DNA is copied into RNA
 2. RNA copy then used to make proteins
- The most common control point is the DNA->RNA copying
 - Protein binding to particular sites within DNA interferes with copying of adjacent information into RNA
- Proteins are continually degraded within cells
 - To maintain a given protein concentration, cell has to keep producing it

Approach

- Use levels of specific proteins as signals
 - Have minimum and maximum concentrations
 - Can “program” cells to produce them
 - Relatively independent – multiple signals within a cell
 - Can get good “inverting” behavior
 - Use different dimers to control “gain”

Results

- Chemical reactions give good inverter-like response
 - High gain
 - Saturation at 0 and 1
 - Relative insensitivity to operating conditions
- Issues
 - Slow (millihertz)
 - No demonstration in paper of actual implementation
 - Based on more recent talk, I believe that individual gates have been built
- Application
 - Use of cells to fabricate nanotech-scale devices

Approach 2: Use DNA to Perform Computations

- This is what people generally talk about when they talk about biological computing
- Basic Idea: Encode all possible inputs to an operation as strands of DNA, then sort out input strands that match a particular criterion
 - Useful for “search” problems, which tend to be NP-complete
 - Vast parallelism – many strands of DNA/tube
 - Relatively slow -- ~15 minutes/step

Theory: What Sorts of Problems Can Be Solved?

- Basic operations
 - Separate: Extract all DNA with a particular bit set from a sample
 - Merge: Mix two samples
 - Detect: Determine if at least one molecule of DNA left in a tube
 - Amplify: Replicate a tube of DNA exactly
- Implementable models
 - Branching Programs (Nondeterministic as well if we have amplify)
 - Turing machines are equivalent to branching programs, modulo size
 - Poly-size BP = log-space non-uniform TM
 - Big concern: constants and orders of growth
 - For conventional computer, “polynomial” is often good enough
 - Need much tighter bound when each step takes a long time

Practice: Sticker Model

- What needs to be defined?
 - How is data represented on DNA?
 - How do we set/clear bits?
 - How do we perform separation?
- Data Representation
 - Single DNA strand with N bases (ATCG)
 - M bases represent a bit, storage becomes N/M bits
 - Trade off space for reliability by changing M
 - Create “sticker” strands that bind to a particular region of M bases
 - Memory strand needs to be configured so that each M-base region sufficiently unique
 - Presence of sticker = 1, absence = 0

Sticker Model

- **Setting Bits**
 - Just add solution of the appropriate sticker to the tube
- **Clearing Bits**
 - Use chemical process to remove the appropriate sticker (still somewhat undefined)
- **Separation Operation**
 - Construct “probes” that bind to a region if no sticker present, one end of probe is fixed
 - Mix DNA solution with probes, binds all strands with that bit 0 to a probe
 - Remove probes from solution, separate bound strands from probes.
 - Now have two samples, one with bit set and one with bit clear

Initialization

- Want to create input set with every possible combination
- Start with memory strands with no bits set
- For each of the input bits:
 1. Separate DNA into equal portions
 2. Mix excess sticker solution with one portion
 3. Combine separated portions
 4. Cause stickers to release from memory strands
 5. Allow to recombine randomly
- If done perfectly with 2^L ($L=N/M$) strands, creates any given combination with probability 63%
 - Probability decreases if operations not perfect
 - Can use more DNA to increase probabilities

Sample Problem

- Minimal Set Cover

- Given B bags, containing some objects of A types, what is the smallest set of bags that contains all types

- Approach

1. Create memory strands with $K = B + A$ regions, initialize B regions randomly
 - First B regions represent bags, later A bits will represent objects.
2. For $I = 1$ to B
 1. Separate solution based on bit I
 2. Set bits in the A region that of strands with bit I set, corresponding to the objects in bag B
 3. Recombine
3. For $J = B + 1$ to $B + A$
 1. Separate out and discard strands that don't have bit J set
4. Perform counting loop to sort remaining strands by how many bits in B region set

Practical Issues

- Mostly relate to the fact that DNA processing is unreliable
 - Authors suggest techniques to improve reliability
 - Can use redundancy in the initial sample to reduce the consequences of errors
 - One approach is to view the biological computing part as a “filter” that reduces the number of possibilities
 - For NP-complete problems, can then use electronic techniques to check results of the biological step
- Authors propose prototype DNA computing workstation
 - Mainly intended to explore concepts, not to be built

Specific Example: DES

- **Data Encryption Standard**
 - Produces 64-bit ciphertext from 64-bit plaintext using 56-bit key
 - For a while, government was backing for cryptography
 - I believe they've recently backed off from this
 - No known attack much more efficient than brute-force trying of all possible keys
- **Basic Approach**
 - Assumes that we have one plaintext-ciphertext pair, want to determine key
 - Use vast parallelism to try all 2^{56} keys in parallel
 - Initialize DNA strands to represent all keys
 - In parallel, encrypt the plaintext using all possible keys
 - Find and output the key whose encrypted ciphertext matches the ciphertext from the pair

Implementation

- Memory strands
 - 11580 nucleotides/strand
 - 20 nucleotides/block = 579 blocks = 579 bits of data
 - 56 blocks hold key, 64 hold ciphertext, rest are temporary storage
- Initialization
 - Random initialization of key region, rest set to 0
- DES Encryption
 - Series of stages
 - Each stage made up of logic operations on fixed sets of bits
 - Implement logic operations by separating out all combinations of the bits, setting desired bit to one for those with output = 1
 - Stage generates 2 bits of output data, many bits of scratch
 - Clear memory after each stage

Practical Issues

- Execution time?
 - 6719 steps
 - At 1 day/step (grad student), takes 18 years
 - At 1 hour/step (simple machine), takes 9 months
 - At 1 minute/step (really snazzy machine), takes 5 days
- Error issues
 - Consider 63% as a “reasonable” chance of getting a correct result
 - With error rates of 10^{-4} per operation, need 1.4 grams DNA, expect .008 distractors/run
 - With error rates of 10^{-3} per operation, need 580 grams DNA, expect 3.2 distractors
 - With error rates of 10^{-2} /operation, need 1.5×10^{29} grams DNA, 8.3×10^{26} distractors
 - Something like 23x the mass of the Earth

Is Biological Computing Practical?

- Still really early
 - Couple real experiments have been run
- Issues
 - Reliability – current processing technologies are nowhere near as reliable as required
 - Automation – need to design machines that can perform these computations
 - Basic operation time – makes a big difference, current systems are too slow for practicality
- Applications
 - Best suited for NP-complete problems that are amenable to brute force
 - Issue in that the size of the problem is limited by physical issues
 - For example, can probably defeat DNA-based code breaking by using longer keys