A Programming Environment for DNA Computing

Steve Carroll (IANAB)
University of Illinois at
Urbana-Champaign

Motivation and Goals

- DNA based computing is one possible successor for conventional silicon technology
- Sticker DNA model was proposed, but tools are necessary to evaluate the efficiency of DNA algorithms
- Provide a complete programming environment for evaluating DNA algorithms
- Attempt to define a high level language that hides tedious details from the programmers so they can focus on algorithmic issues
- Evaluate the utility of compilers and simulators in DNA computation

Outline

- Background
- Simulator design
- DNA-C language design
- Compiler design issues
- Discussion and Conclusions

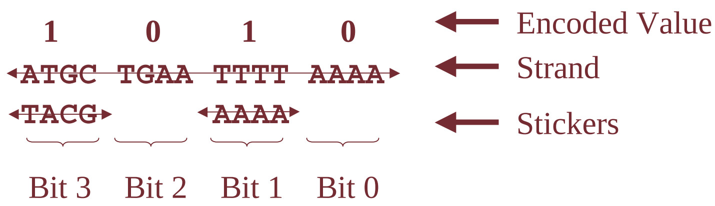
What is DNA Computation?

- Single-stranded DNA is a chain of amino acid bases (A,C,T,G)
  - each base has a complement: A ⇔ T C ⇔ G
- Each strand will bond to a complementary strand

```
←ACTGTTGCC→
 | | | | | | | | |
←TGACAACGG→
```

DNA Sticker Model

- Use bonded and non-bonded state to encode information
- Each DNA strand is broken up into bits
  - unique subsequences (comma-free codes)
- A sticker represents one bit segment and the sticker can only attach at that position
- If the sticker is attached, the bit is “on”
- A tube contains many strands encoding different values



Sticker Model II

- Algorithms exist for Minimal Set cover (NP complete) and DES decryption in polynomial time
- General computation model operates on all possible solutions at once, effectively trading time for space
- Initial condition is usually a single tube with one strand for each of the possible solutions called the initial set.
- Ex: Init(3,6) is the set of length-6 strands with all possible binary combinations of the first 3 bits

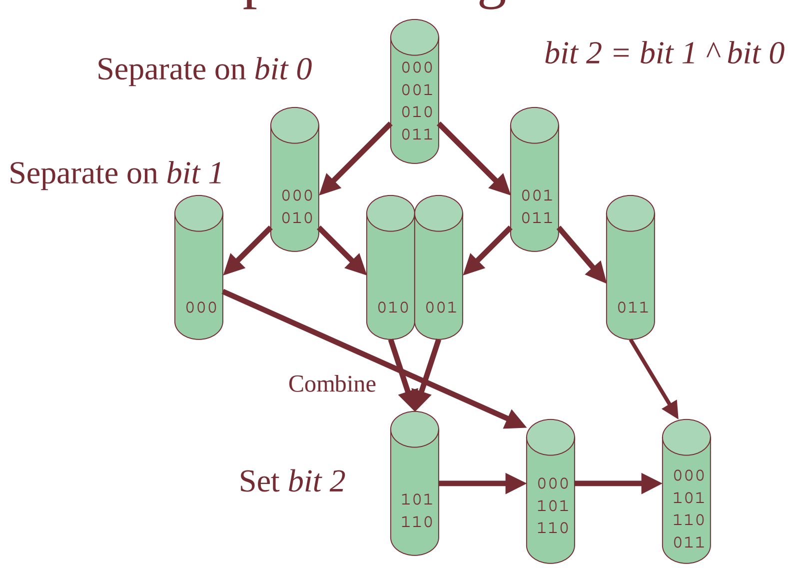
uninitialized bits are used for storing intermediate results →

```
000000
000001
000010
000011
000100
000101
000110
000111
```

Basic Operations

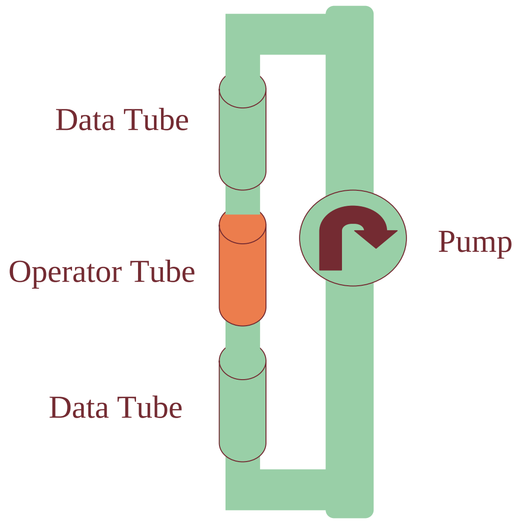
- SET
  - Sets a given bit in each strand in the tube
  - Add sticker for that bit to the tube.
- RESET
  - Clears a given bit in each strand in the tube
  - Add anti-stickers to remove the stickers for that bit
- SEPARATE
  - Separates one tube into two tubes based on whether or not a given bit position is on or off
  - A probe bonds to all unset strands
- COMBINE
  - Combines two tube contents into a single tube.

Implementing XORs



Proposed Machine Architecture

- Components
  - mixing apparatus (right)
  - Robot arm to bring the appropriate tubes to the mixer
  - MIPS-like controller processor
  - DataTube rack
  - StickerTube rack
  - AntiStickerTube rack



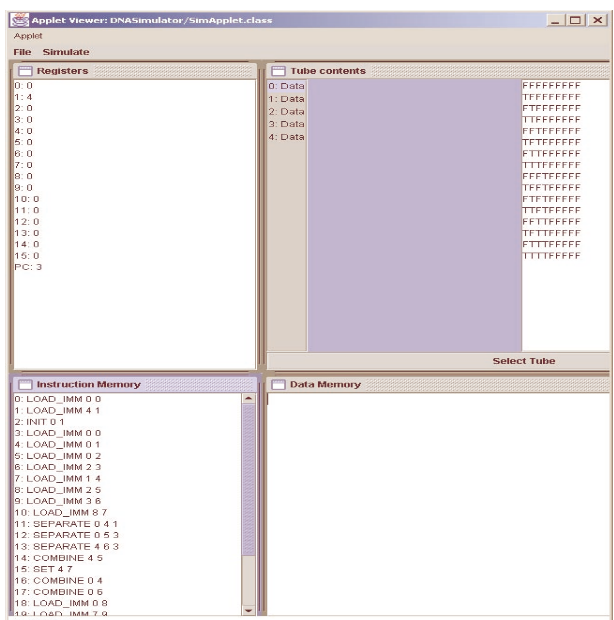
Operator Tubes

- Blank
  - no operation. Used for move and combine.
- Filter
  - Allows only stickers and anti-stickers through. Used for set and reset.
- Probe
  - Catches all the strands not stickered at a given bit. Used for separate.

Other Tube Types

- Data Tubes
  - Contains the data strands. Some start empty, others start initialized with the potential solution set.
- Sticker Tubes
  - Each sticker tube contains a large number of stickers for a given bit. Assumed that the sticker tube can be refilled after each use. Used for set operations.
- AntiSticker Tubes
  - Contains a large number of anti-stickers for a given bit. Used for reset operations.

Simulator GUI



DNA ISA

- Standard MIPS-like instructions for integer calculation and branching
- INIT tube_num, r
  - r is the register containing the number of bits
- SET tube1, sticker_tube
- RESET tube1, anti_sticker_tube
- COMBINE tube1, tube2
  - tube1 gets contents of tube1 and tube2
- SEPARATE tube1, tube2, probe_tube
  - tube1 is split into tube1 and tube2 based on the bit that is operated on by the probe tube.

Simulator Error Models

- Error percentages can be changed in the simulator
- Stickers that fall off
  - bit errors
- Strands stuck to the side of the tube
  - potential loss of the solution
- Stickers that adhere to the wrong place
  - Could cause two bits to be covered.
  - Stickers that adhere to close matches

DNA C

- An ANSI C dialect that can be used for DNA programming.
- Features chosen based on the needs of proposed algorithms.
- Two base types: “int” and “tube” and statically allocated arrays of these types
- Each tube has an array-like syntax
  - elements are the individual bits of the DNA strand
- No pointers, no structs and unions.
  - A simplification, could exist if type safety was enforced

DNA C declarations

```
int i,j;
int k[100];
tube l<|64|>;
tube m[64]<|10|>;
tube n; // invalid
```

Sample program

```
void main() {
  int I;
  tube t1<|64|>;
  tube t2<|64|>;
  tube t3<|64|>;
  t1 init 32;            // Init(32,64)
  t1<|8|> -> t2 : t3;    // separate the strands
                         // in t1 based on bit 8
                         // into t2 and t3
  for (I = 32; I < 64; I++) {
      t2<|I|> = 1;       // set the bit true
  }
  t1 = t2 + t3;          // combine
}
```

Tube Syntax

- Arrays of tubes:
  - `tube t[8]<|5|>;`
  - array of eight tubes containing 5 bit strands named **`t`**
  - Multidimensional arrays are also allowed.
- Combine tubes:
  - `t = t + t2;`
  - `t <- t2; t += t2;`
  - Contents of t2 added to t
- Logical operations (^, |, &,!):
  - `t<|5|> = t<|3|> ^ t<|4|>;`
  - Tube must be the same for each operand
- Transfer contents of the tube to another tube:
  - `-t1 = t2`

Tube Syntax (cont.)

- Assign a bit:
  - `t<|2|> = (I > 45);`
  - Bits are boolean variables.
- Copy a bit:
  - `t<|3|> = t<|4|>;`
- Separate tube based on bit I:
  - `t<|I|> -> t_on : t_off;`
- Initialize the tube:
  - `tube init int_val;`

Results

- The minimal set covering algorithm as described in the original paper was implemented in DNA C and executed for various problem sizes
- The runtime scales exponentially in the simulator, so it is only suitable for evaluating the algorithms with reasonably small problem size.
  - For minimal set cover and DES encryption, keep set sizes and key sizes small for debugging, testing, and refinement, then scale up to full size for actual computation.

Results (cont)

•Minimal Set Covering Algorithm

- •# of bits is the number of subsets
- •30 lines of source vs. 200 lines of assembly

# of bits	Runtime (secs)
4	27
8	126
16	1306
32	15348

Suggested Compiler Optimizations

- Improving error performance
  - compiler added redundancy
- Tube level parallelism
  - each biological step is very slow
  - optimization for performance should focus on minimizing the number of separates, sets, and resets.
  - even in a simple XOR conversion, if multiple mixers are available, the number of steps can be reduced from 4 to 3.
- Tube allocation will be necessary to reuse the tubes created in intermediate steps.

Discussion

- We can manipulate the tube data from the C program, but there is no way to read the strand data into the C code.
  - can implement the NAND gate, so all Boolean functions are realizable, but is this sufficient for all desired applications?
  - only feedback available is at the end of the execution
    - model might be more robust if an "is tube empty?" operation could be physically realized
- Optimizing for the least possible strand space, extra spaces are effectively scratch pads and can be reused.
- Most of the simulation work is embarrassingly data parallel

Conclusions

- Simulator is a good tool for demonstrating the concepts of DNA computing.
- Tool should be useful in evaluating whether the DNA computation model is expressive enough for our needs.
- DNA-C allows the programmer to focus on algorithm instead of bit level issues.
- A high level language makes it easy to develop a library of common routines like sorts.
- Need to evaluate new computation technologies from a programming perspective.

References

- **“DNA Sequences Useful for Computation”** by Eric Baum
- On Applying Molecular Computation To The Data Encryption Standard by Leonard M. Aldeman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. In DNA Based Computers: DMACS Workshop, 1996.
- A Sticker Based Model for DNA Computation by Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothemund, and Leonard M. Adleman. In DNA Based Computers: DMACS Workshop, 1996.