Alternate Forms of Computation

Robert Senser, PhD

CSC 5446 Presentation

Spring 2015

Computation "Beyond" Von Neumann

- Von Neumann's 1945 architecture for "digital computers:"
  - Arithmetic logic unit
  - Processor registers
  - Control unit with instruction register and program counter("PC")
  - Memory to store data and instructions
  - The 1952 "ILLIAC:" example of an early Von Neumann machine
    - MTBF: 10 to 15 minutes; 2,800 vacuum tubes
- Also in 1940s-50s: analog computation was common
- Some other forms of computation in 2015:
  - Quantum Computing: advanced analog computation
  - DNA-based computation
  - FPGA: Field-programmable gate array
    - Including ASIC (Application-Specific Integrated Circuit)
  - GPGPU: General-Purpose Computation on Graphics Hardware
    - GPGPU does use a Von Neumann-like architecture.

Beyond Von Neumann

- We just looked at Quantum Computing
  - Is a type of analog computation
  - Clearly not digital computation
- Let's look at some other forms of computation
  - DNA-based Computation
    - Notion is using DNA behaviors to do computation
  - FPGA: Field-programmable Gate Array
    - A field-programmable gate array is an integrated circuit configured after manufacture using an hardware description language such as Verilog or VHDL
    - Can be used to do massively parallel processing
  - GPGPU: General-Purpose Computation on Graphics Hardware
    - Sometimes refereed to as "Ridiculously Parallel"
    - Example: 240 parallel cores, 12,000+ threads / core

DNA-based Computation

- Employs (bio)molecular manipulation to solve computational problems.
- In 2000, Kari et al. showed that is possible to solve the *Post Correspondence Problem ("PCP")* using DNA-based techniques.
  - The PCP is an undecidable decision problem that is simpler than the *halting problem* and is often used in decidability proofs.
  - The problem is to decide if sequence of indices exist such that :

$$\alpha_{i_1} \dots \alpha_{i_K} = \beta_{i_1} \dots \beta_{i_K}.$$

    where alpha and beta are words in some alphabet.
  - The problem is to decide if the sequence exists.

DNA-based Computation

Here is an example taken directly from the DNA2 reference:

Consider the following two lists:

α_1	α_2	α_3
a	*ab*	*bba*

β_1	β_2	β_3
baa	*aa*	*bb*

A solution to this problem would be the sequence (3, 2, 3, 1), because

$\alpha_3\alpha_2\alpha_3\alpha_1 = bba + ab + bba + a = bbaabbbaa = bb + aa + bb + baa = \beta_3\beta_2\beta_3\beta_1$.

Furthermore, since (3, 2, 3, 1) is a solution, so are all of its "repetitions", such as (3, 2, 3, 1, 3, 2, 3, 1), etc.; that is, when a solution exists, there are infinitely many solutions of this repetitive kind.

However, if the two lists had consisted of only α_2, α_3 and β_2, β_3, then there would have been no solution (because then no matching pair would have the same last letter, as must occur at the end of a solution).

DNA-based Computation

- A quote from DNA2: “DNA computing is fundamentally similar to parallel computing in that it takes advantage of the many different molecules of DNA to try many different possibilities at once.”
- The DNA algorithm from DNA1 is partially shown here:

DNA algorithm for the Bounded Post Correspondence Problem

Step 1. *Encoding the words and numbers.*

(a) Encode each word u_i, w_i, $1 \leqslant i \leqslant n$, in a DNA sequence.

(b) Encode each number i, $1 \leqslant i \leqslant n$, in a DNA sequence.

(c) Synthesize the encoding of a chosen “bridge” string β into a DNA sequence and put the resulting population of sequences β into two tubes A and B.

(d) Set $k = 1$.

DNA-based Computation

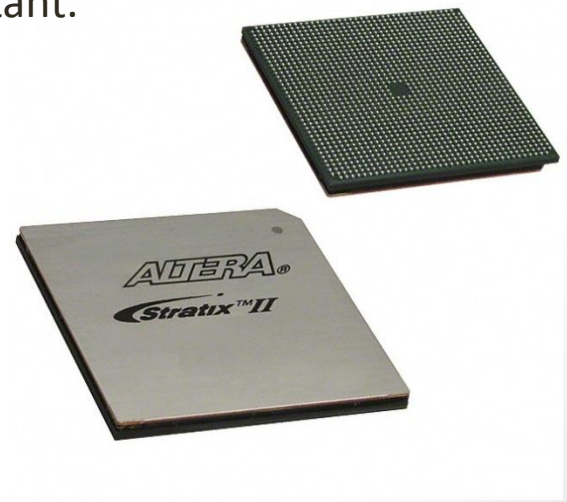
- The just-described research shows that DNA-based computation is possible.
- The Kari et al. research was published in 2000. So, this approach has not made great strides.
- Why Slow? (my personal views)
  - The calculation is a biochemical reaction requiring special equipment (PCR: sequencing, etc.) and special chemical *primers* to be generated.
  - This is like having to order/manufacture new chips for your computer when you change your program!
  - The speed of the reactions is not fast – takes minutes then time to sequence the resulting samples.

FPGA
Field-programmable Gate Array

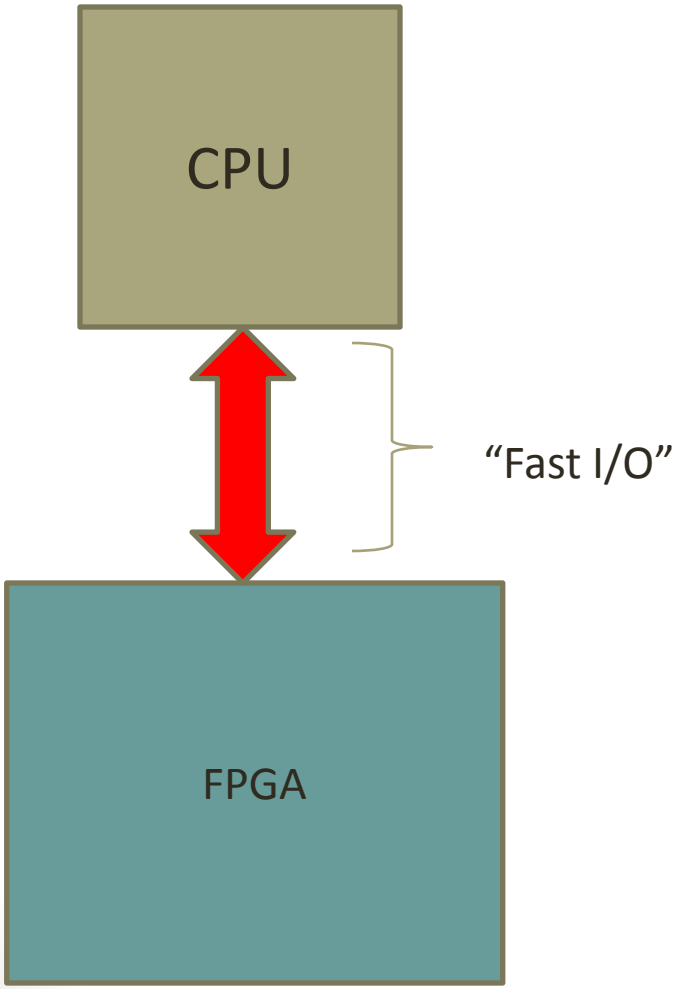
- FPGAs are essentially highly dense arrays of programmable logic.
- Developers (Programmers) can directly code the solution logic in hardware using a language such as Verilog or VHDL.
- This eliminates the “inefficiencies” cause by the traditional Von Neumann model.
  - There is no time required to find and decode instructions.
  - The hardware gates used to implement CPU/GPU optimizations are applied directly to the problem solution.
  - Works well when simple operations are done to each element in an array. Example: XOR array with a constant.
  - Downside: Not easy to program.

Verilog

```
1 module encoder_using_case(
2 binary_out , //  4 bit binary Output
3 encoder_in , //  16-bit Input
4 enable      //  Enable for the encoder
5 );
6 output [3:0] binary_out  ;
7 input  enable ;
8 input [15:0] encoder_in ;
.....
```



FPGA
Field-programmable Gate Array



- CPU handles control and I/O
- FPGA does || operations, Ex: XOR constant with 1M cells
- The need for the external I/O slows things down.

FPGA
Field-programmable Gate Array

- This approach is in use today; see [background].
- Can you see what is attractive about this?

C++: code to XOR when values change:

```
update(int  x, int  &y, int mask) {
while (1) {
  y = x ^ mask;
/* computer CPU's PC is constantly changing. */
}
```

Verilog: code to X)R when values change:

```
task update;
input [7:0] mask;
input[7:0] x;
output[7:0] y;
begin
y = x ^ mask;
end
end task;
```

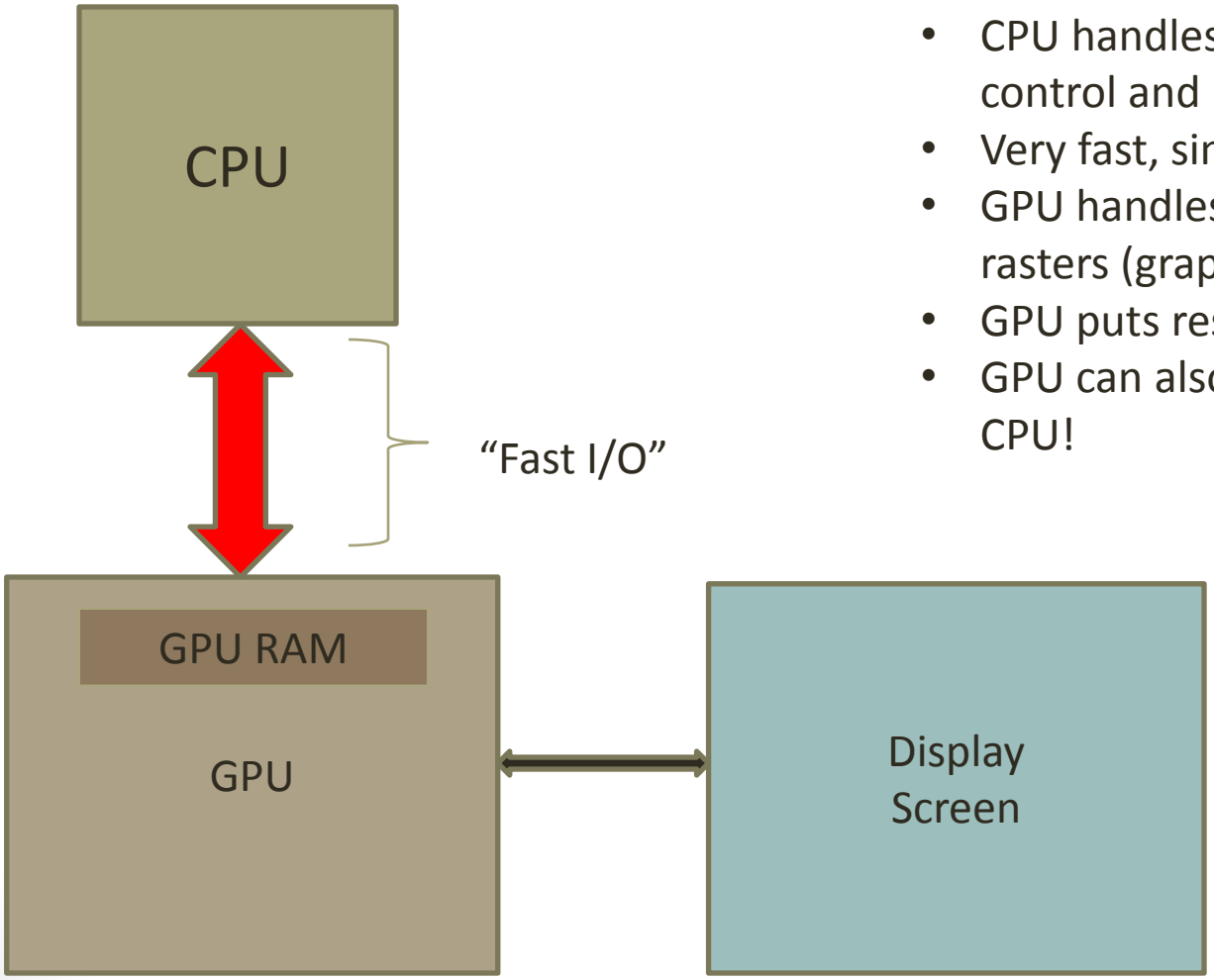
These become "gates".

GPGPU

General-Purpose Computation on Graphics Hardware

- A GPU is a specialized electronic circuit (processor) designed to rapidly manipulate memory in such a way to optimize images in a frame buffer.
- These processors are more effective than CPUs for graphical processing (especially in cost terms….).
- GPGPU generalizes the use of these GPUs for non-graphical use.
- Programming models for GPGPU
  - CUDA: Nvidia proprietary C/C++-like model.
  - OpenCL: Non-proprietary model.
- Products like MATLAB include support
  - Automatic use of GPGPU; no specialized coding.
  - Used with array-intense processing like matrix multiply or FFT.

GPU
Graphics Processing Unit



- CPU handles program, control and non-Display I/O.
- Very fast, simple processors.
- GPU handles pixels and rasters (graphics processing).
- GPU puts results on screen.
- GPU can also return results to CPU!

GPGPU

General-Purpose Computation on Graphics Hardware

- GPUs are a version of “SIMD” processing, called “SIMT”.
  - SIMD: Single Instruction, Multiple Data.
  - SIMT: Single Instruction, Multiple Thread.
  - With SIMT can have path divergence – that is some threads can take different paths:
    Sample Nvidia CUDA code:

```
__global__ void MVKernel_gm(Matrix A, Matrix X, Matrix Y)
{
 float Pvalue = 0;
  ....
  for (unsigned int k = 0; k < A.width; k++)
    {
      if(Row < A.height)
      Pvalue += A.elements[Row*A.width+k] * X.elements[k];
    }
  ....
}
```

- SIMD, and SIMT, are likely just faster classical (Van Neumann) digital computers.

Alternate Forms of Computation and Theory of Computability

- Do any of these alternate forms of computation affect how we look at Computability?
  - We have already "discounted" Quantum Computing.
    - Duetch's Algorithm did factor numbers using QC.
    - QC is still evolving.
  - FPGAs and boundaries of computing.
    - Can the logic on a FPGA be used make something beyond a "simple" Van Neumann computer?
  - GPGPUs and boundaries of computing.
    - Gives "times N" added throughput.
    - Could be hosted by a FPGA.
  - DNA-based (Warning Bob Senser's views here!)
    - Notice it is the chemicals that get manufactured, not gates and circuits. Then the DNA attaches itself to matching patterns.
    - Can this do better than just "times N" increased capacity?
    - My personal view is that this is an interesting area….

Conclusions

- What are your thoughts?
- What are the bounds of computing?
- Ever wonder why people can solve abstract problems that computers don’t appear to be able to solve?

References

- Here are some references/URLs used to produce this presentation:
  - Background: Shuai Che, Jie Li, Jeremy W. Sheaffer, Kevin Skadron, and John Lach. 2008. Accelerating Compute-Intensive Applications with GPUs and FPGAs. In Proceedings of the 2008 Symposium on Application Specific Processors (SASP '08). IEEE Computer Society, Washington, DC, USA, 101-107. DOI=10.1109/SASP.2008.4570793 http://dx.doi.org/10.1109/SASP.2008.4570793
  - DNA1: Lila Kari, Greg Gloor, Sheng Yu (January 2000). "Using DNA to solve the Bounded Post Correspondence Problem". Theoretical Computer Science 231 (2): 192–203. — Describes a solution for the bounded Post correspondence problem, a hard-on-average NP-complete problem. Also aviable here: http://www.csd.uwo.ca/~lila/pdfs/Using%20DNA%20to%20solve%20the%20Bounded%20Post%20Correspondence%20Problem.pdf
  - DNA2: http://en.wikipedia.org/wiki/DNA_computing
  - FPGA: http://en.wikipedia.org/wiki/Field-programmable_gate_array
  - GPGPU: http://www.ogf.org/OGF25/materials/1605/CUDA_Programming.pdf
  - ILLIAC: http://en.wikipedia.org/wiki/ILLIAC
  - Post Correspondence Problem: http://en.wikipedia.org/wiki/Post_correspondence_problem
  - Von Neumann: http://en.wikipedia.org/wiki/Von_Neumann_architecture