FPGA Random Forest Classifier

Exploring hardware based implementations of machine learning algorithms

Vivek Krishnan

15-418: Parallel Computer Architecture and Programming

Overview of Presentation

Problem and Scope

- CV/ML Algorithms in Embedded Devices
- Random Forest Classifier

Proposed Solution

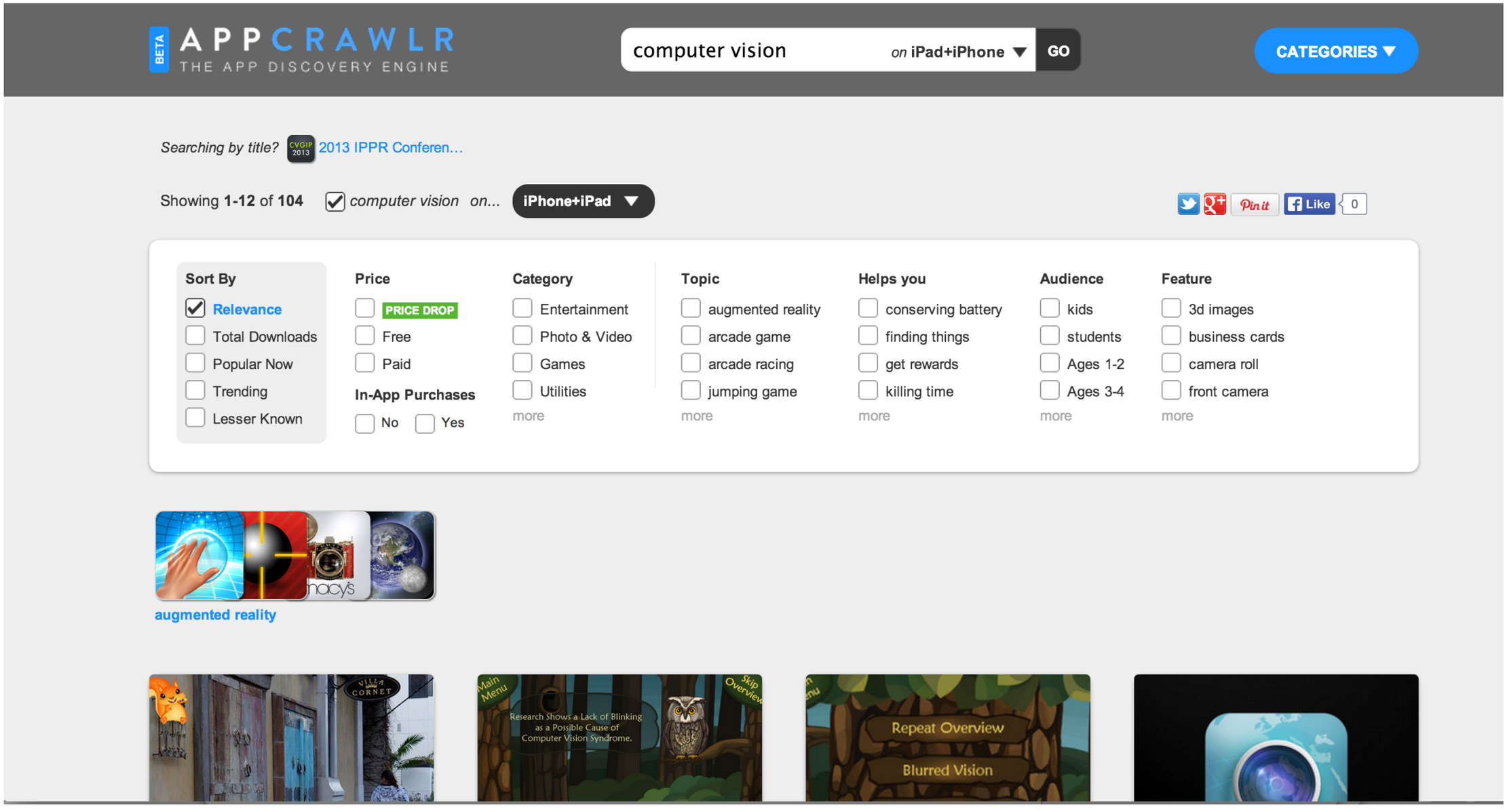
- Platform and Languages
- Designing a Decision Tree to Minimize Routing
- Implementing two Perpendicular Pipelines

Summary of Findings

- Recommendations to Chip Designers

Problem And Scope

AppCrawlr compiled a list of the top 100+ IOS app that integrate computer vision



Constant Processing and Analysis is Expensive

On mobile platforms energy is also an important factor along with time

Solution: Optimize the tasks that are repeated many times

Object Detection Algorithm on sample data

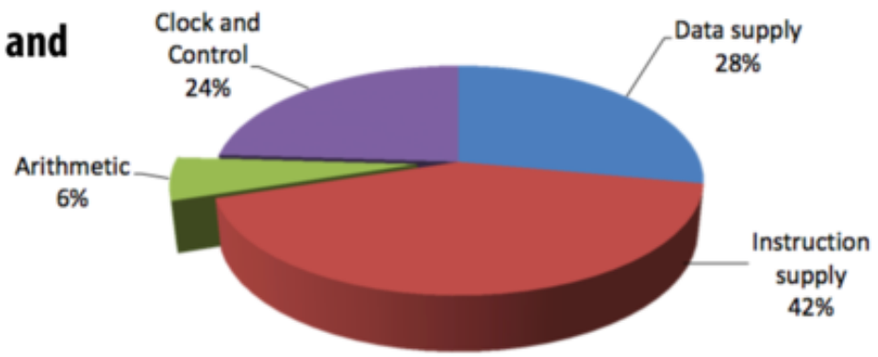
We have already discussed how using fixed function units implemented in hardware can improve efficiency by orders of magnitude.

Efficiency benefits of compute specialization

- Rules of thumb: compared to good-quality C code on CPU...

- Throughput-maximized processor architectures: e.g., GPU cores
  - ~ 10x improvement in perf / watt
  - Assuming code maps well to wide data-parallel execution and is compute bound

- Fixed-function ASIC ("application-specific integrated circuit")
  - ~ 100x or greater improvement in perf/watt
  - Assuming code is compute bound and and is not floating-point math



Efficient Embedded Computing [Dally et al. 08]

[Source: Chung et al. 2010 , Dally 08]

[Figure credit Eric Chung]

Decision Trees

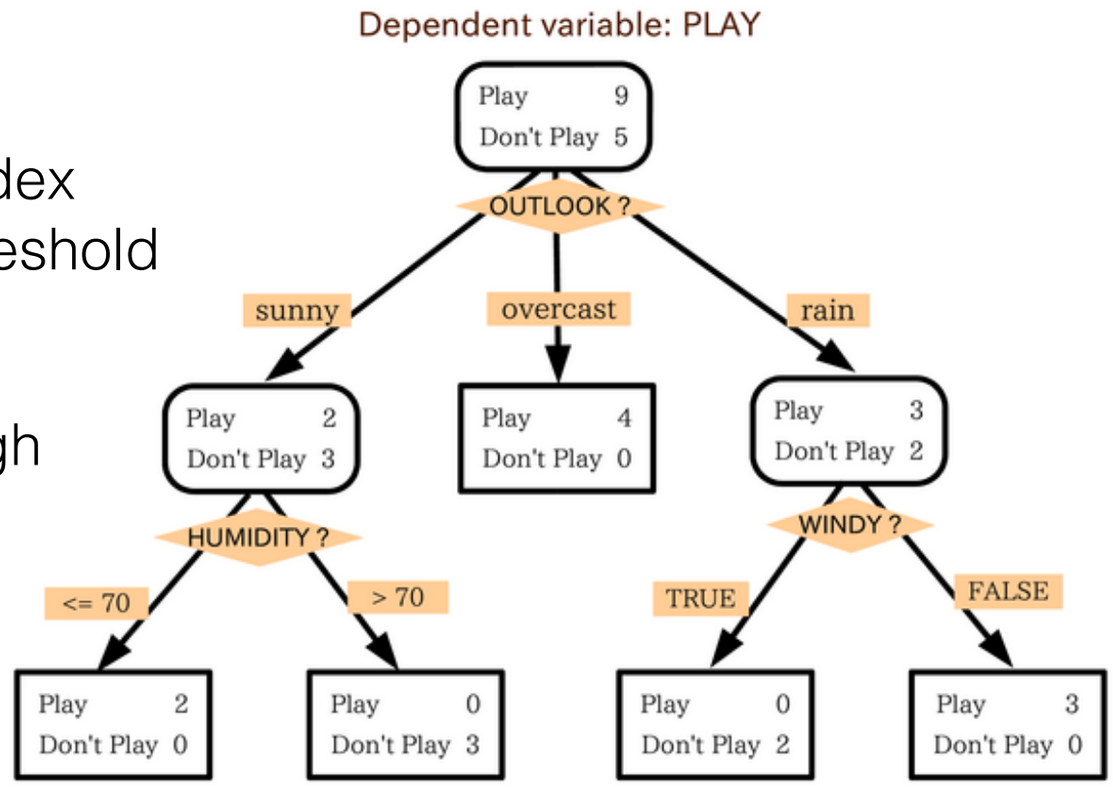
Given an input to the system, make a decision at each node whether to branch left or branch right

Each Node contains information about the index of the sample it will analyze as well as the threshold which determines the direction of branching

Output is dependent on the path taken through the decision tree

Processing on each **level** is independent

Going from **level-to-level** must be done sequentially

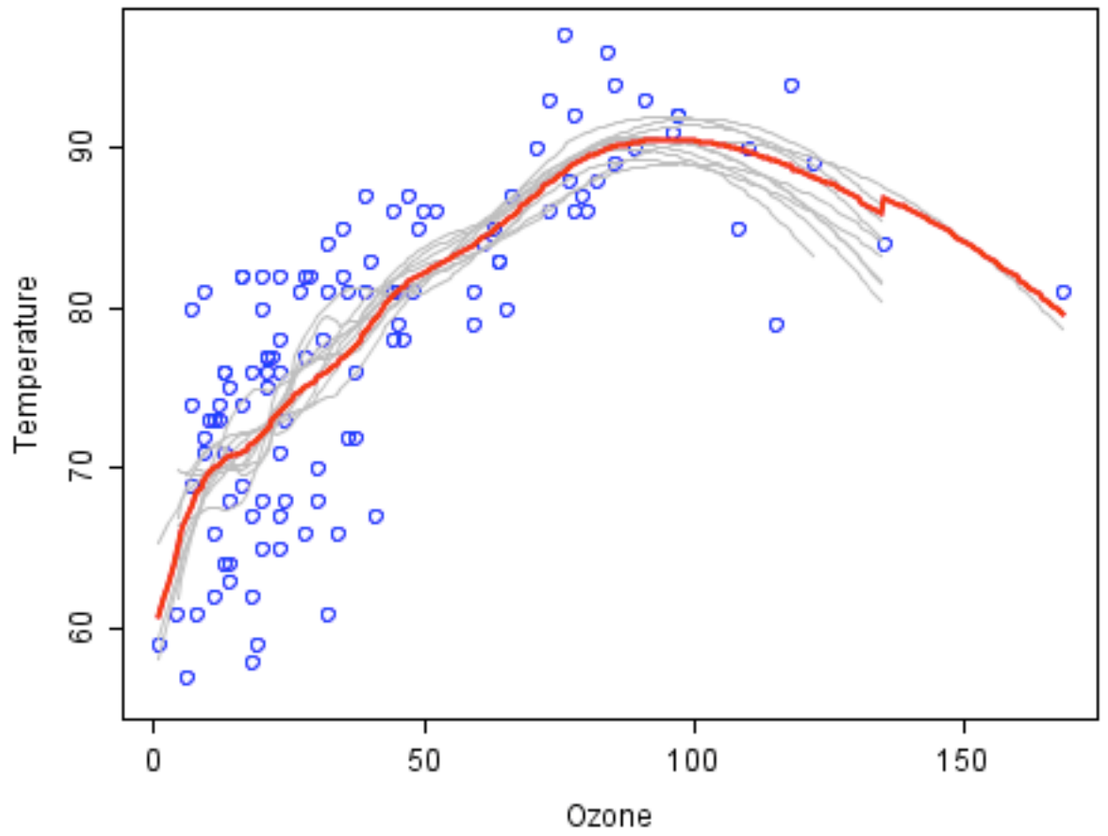


Random Forest

Fitting sample data to a specific decision tree may be inaccurate

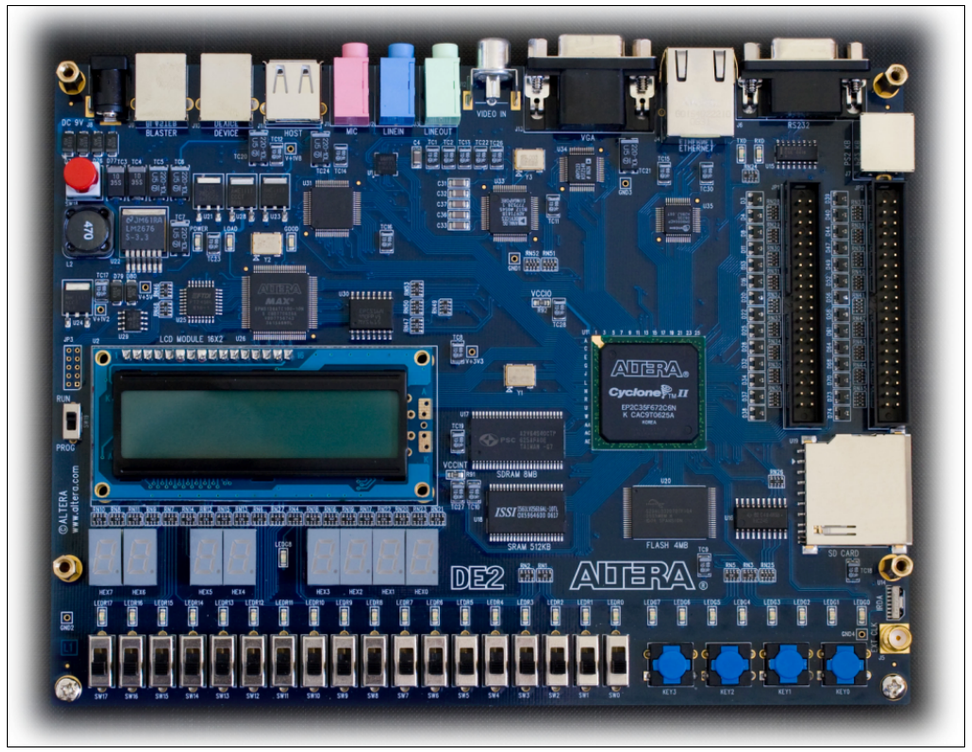
Instead, generate n trees with random fluctuations in construction. Average the output of all the trees to get better solution

Processing across **trees** is independent



Sample Data Points (blue points) are fitted by random trees (grey line) and the output of the forest (red line) is the best fit

Proposed Solution



ALTERA DE2 Board

Altera Cyclone® IV E FPGA

50 MHz Clock

2MB SRAM

128 MB SDRAM

Describe hardware using Verilog

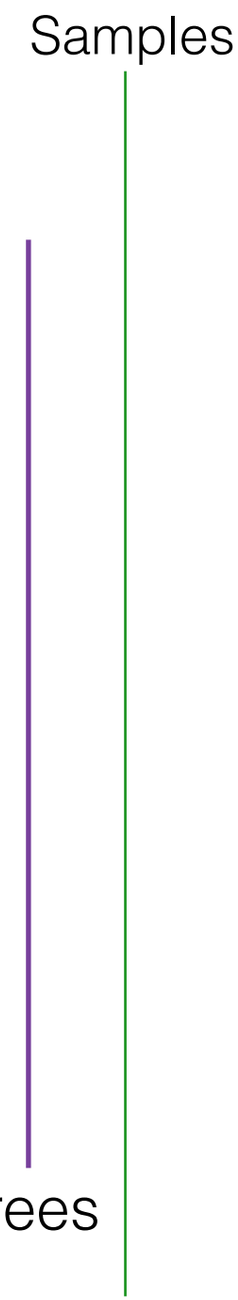
Abstraction of gate-level design

Used by a synthesizer to create the board,
is not actually executed

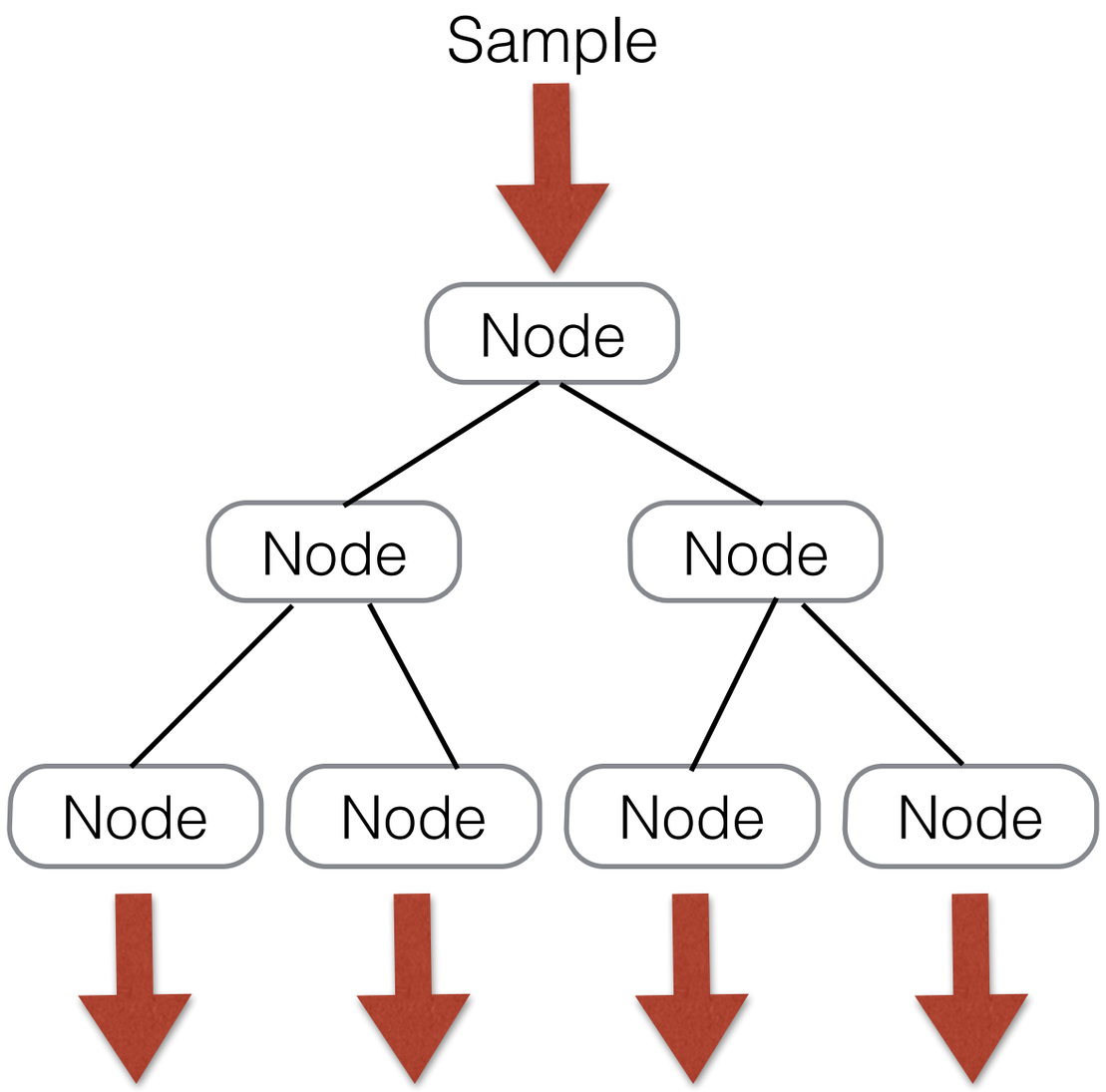
Specifying the data paths and connections

Basic Algorithm exploits data parallelism

```
for (sample in sample_data) {

        sample_result = 0;

        for (tree in forest) {
            local_result = 0;
            current_node = tree.base_node; //possible 0

            for (level < max_tree_depth) {

                node_index = NUM_NODES_PER_TREE * treeIdx + current_node
                node_threshold = get_threshold (node_index);
                sample_var_idx = get_corresponding_var (node_index);
                sample_var_data = sample.getVar(sample_var_idx);

                //If at a branching node in the tree
                if (level < max_tree_depth) {
                    branchLeft = (sample_var_data < node_threshold);
                    current_node = next_node (branchLeft);
                //If at a leaf node in the tree
                } else {
                    local_result = get_result (node_index);
                }

            }
            sample_result += local_result;
        }

        sample_responses[sample] = process_result (sample_result);
}
```



Simple Implementation of a Tree

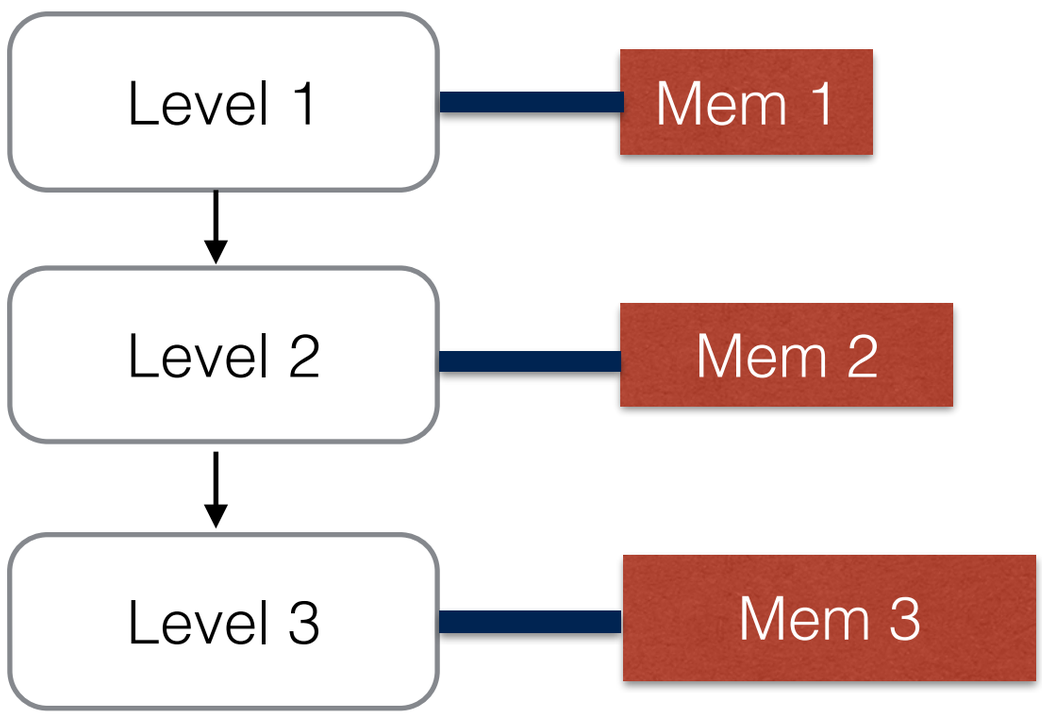


We want to maximize the amount of work we do on a board but the amount of resources is limited

Routing on the naive solution grows at a rate of 2^(depth of tree)

Only one level of a tree can be done in parallel

Better Way to Represent a Tree

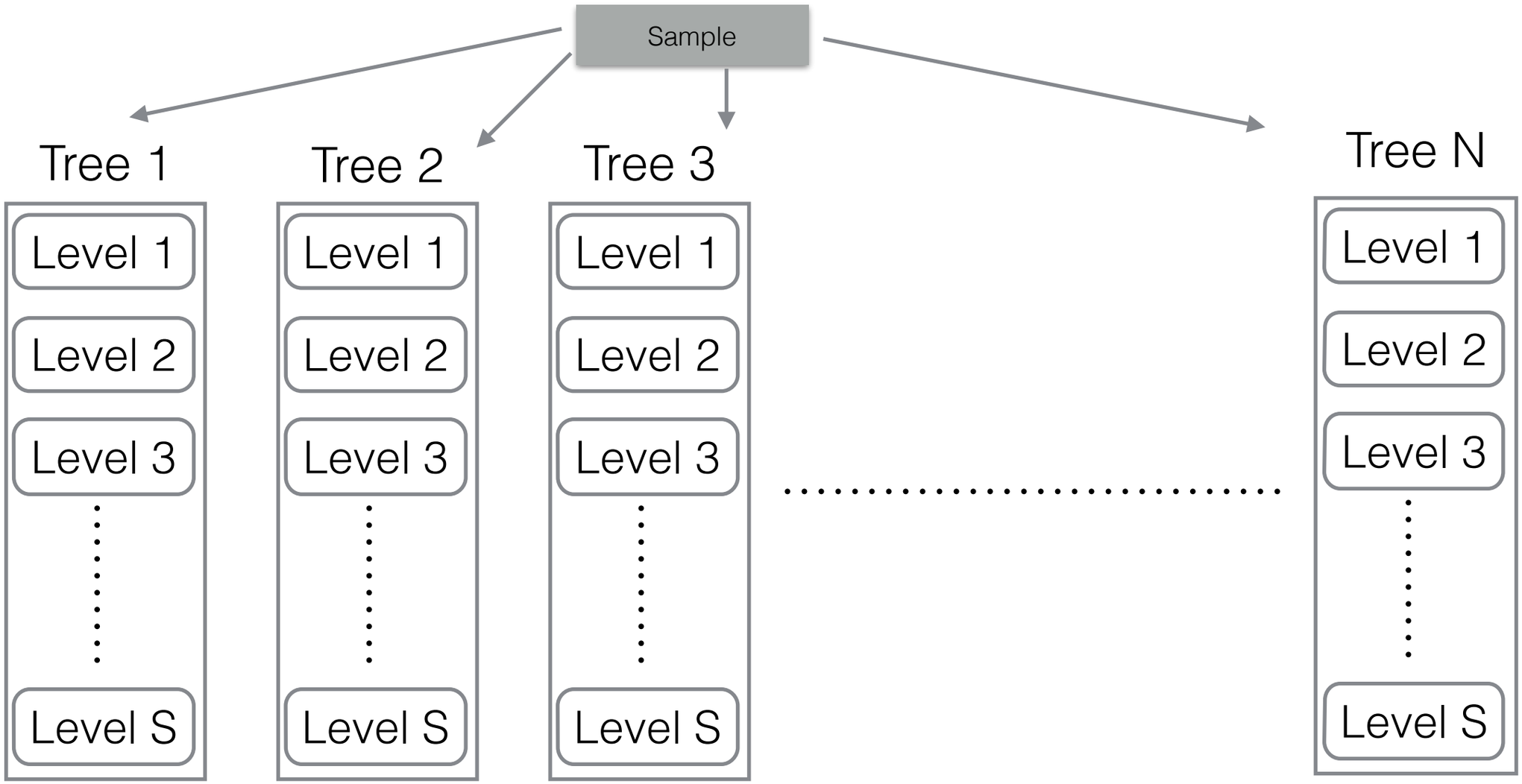


Output of each level includes the nodeIdx of the next node it will visit

Each level can access appropriate memory from fast SRAM

Now routing for the tree grows linearly with depth for the tree

Creating an Ensemble that can run in parallel



Output Bandwidth: N Data Values per Cycle

Reality

I/O sucks on FPGAs and is the bottleneck for the system

The USB blaster on the system I was using had a data rate of 480 Mbps

With a 50MHz Clock, this equates to about 10 bits per cycle

That is less than one floating point per cycle hooked up to a system that can output n different floating points per cycle

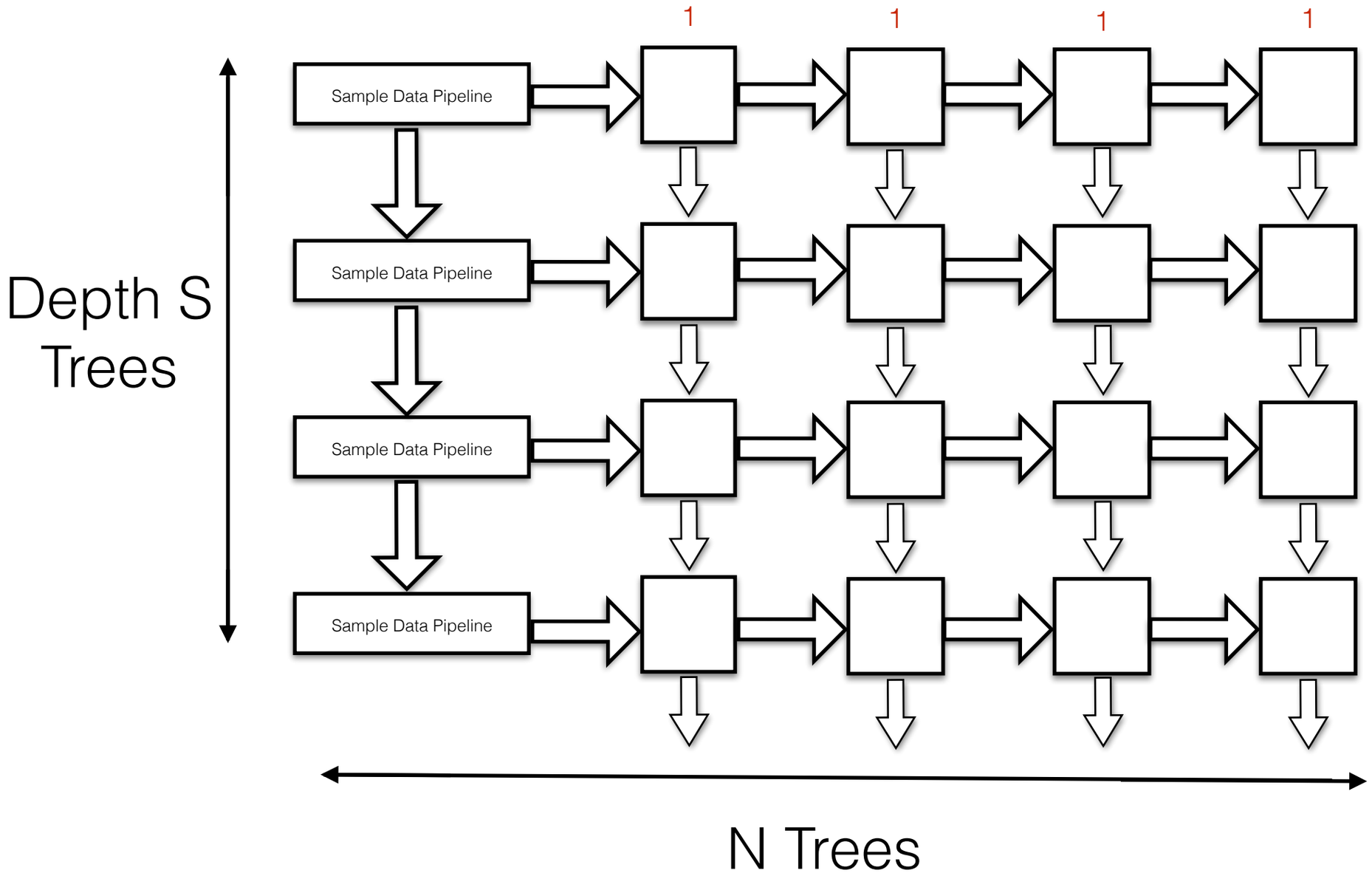
The data transfer rate of the outputs of the algorithm will stall near the output and back up the entire system. I/O cannot scale linearly with the number of trees that are needed to classify larger and larger datasets

Much rather have a system that will output a consistent one data value per cycle

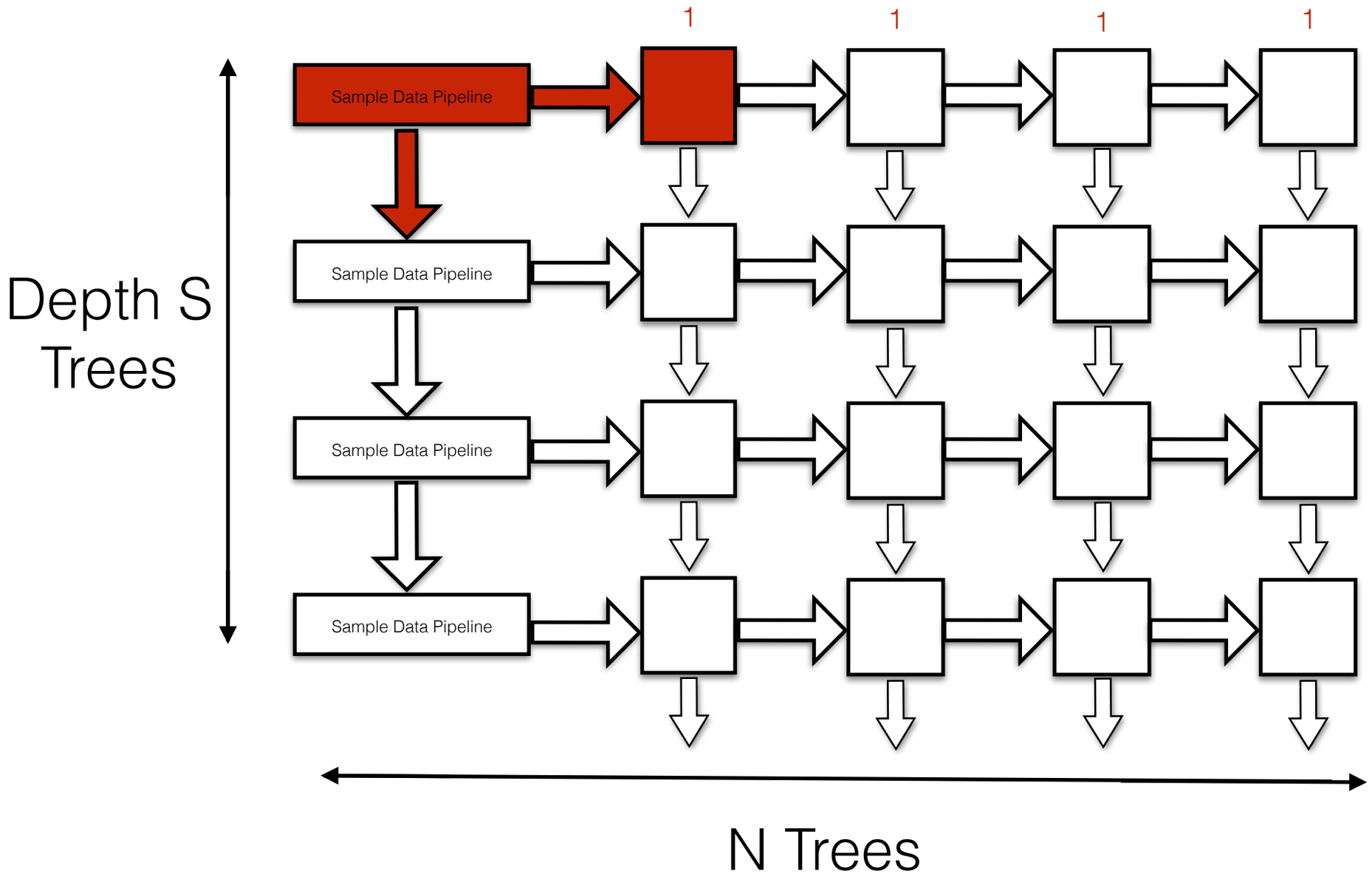
Can remove the logic needed to ensure that the next unit is ready for the sample data because data movement patterns are consistent and manageable

Perpendicular Data Pipelines

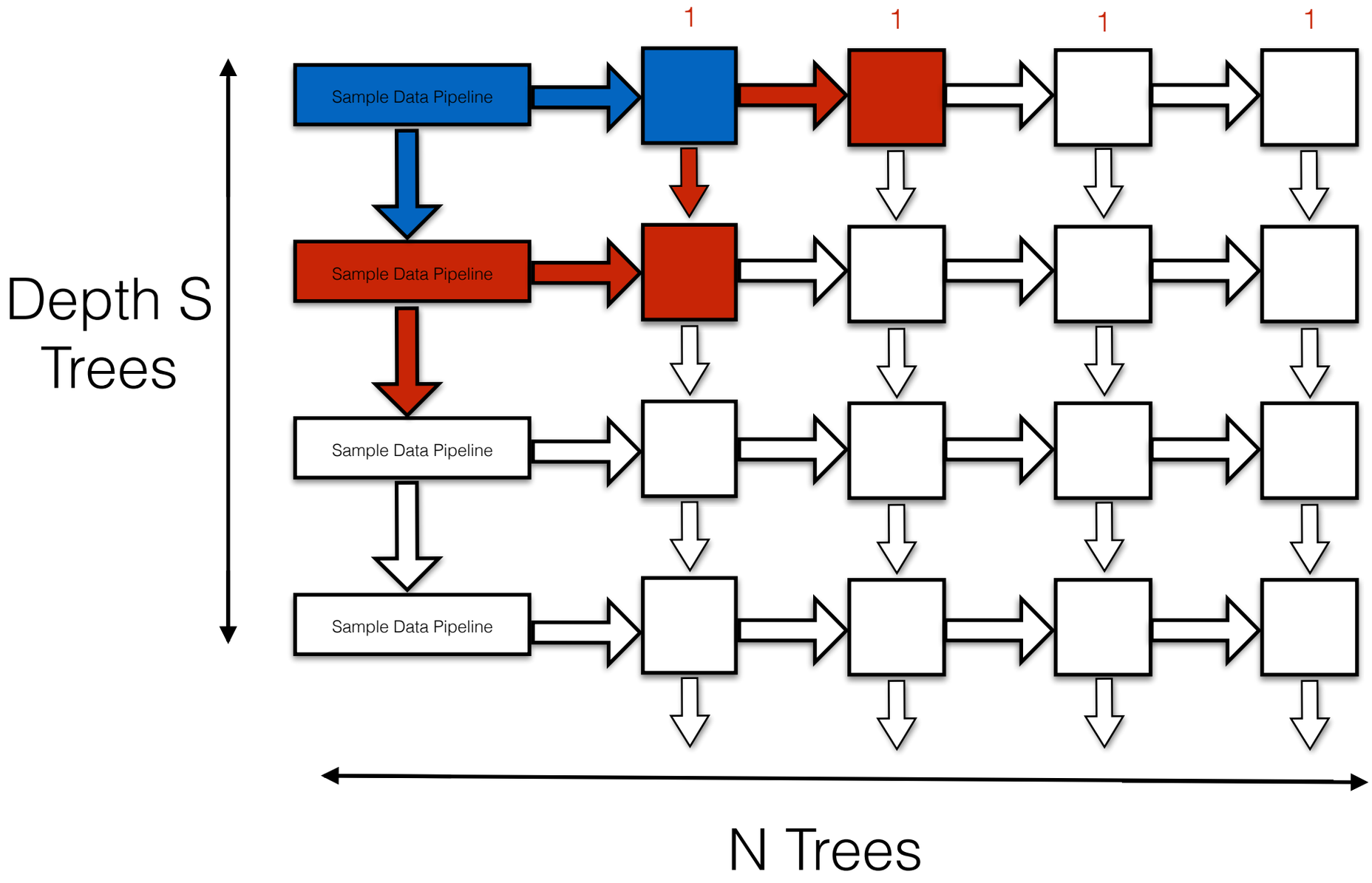
Output Bandwidth : 1 Sample per Cycle



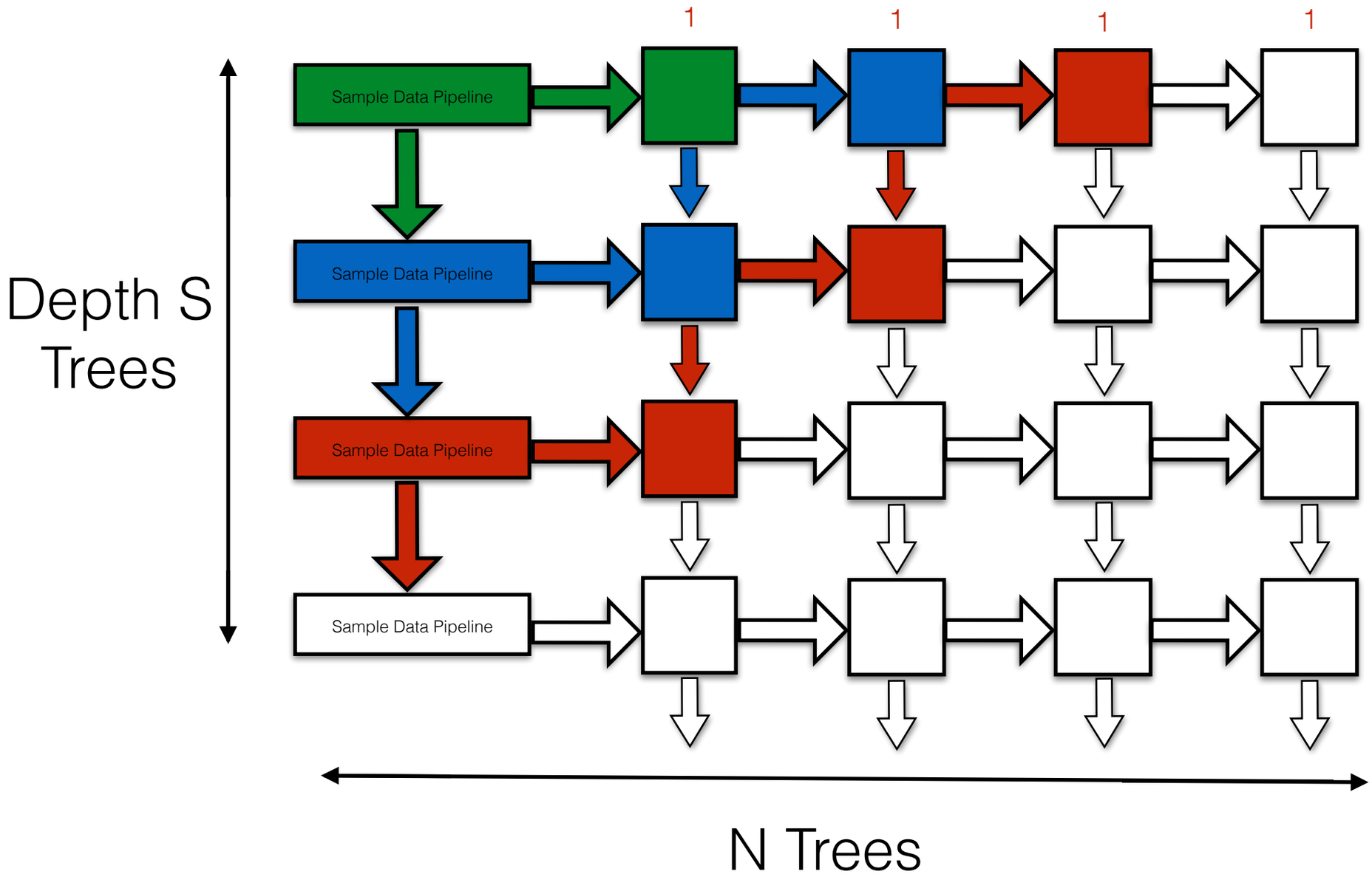
Data floods into the system from the sample Pipeline. It takes N+S clock cycles for the system to reach peak output.



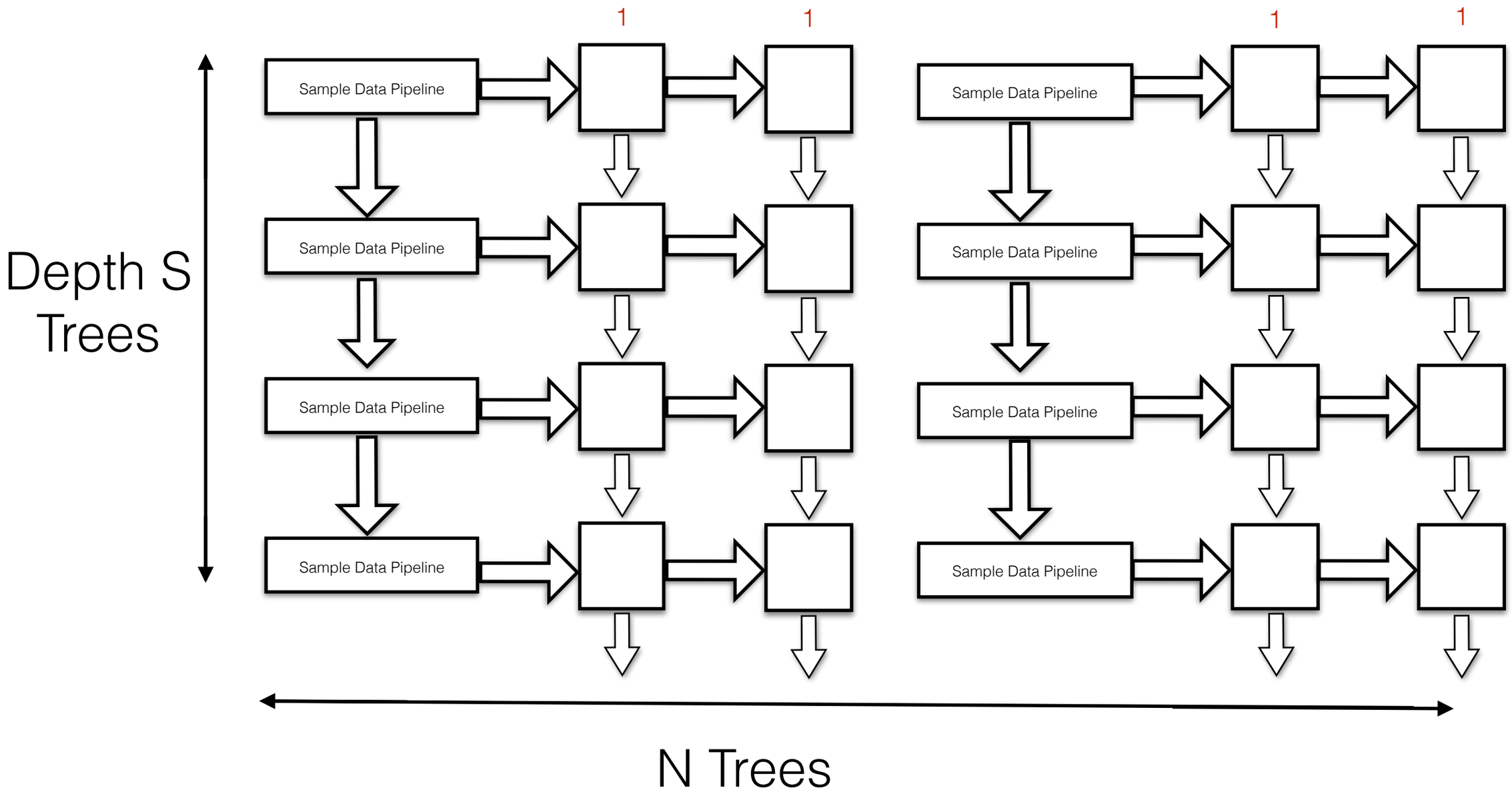
Once filled, system can output one processed sample per clock cycle



Cross dependencies used to harness the bandwidth



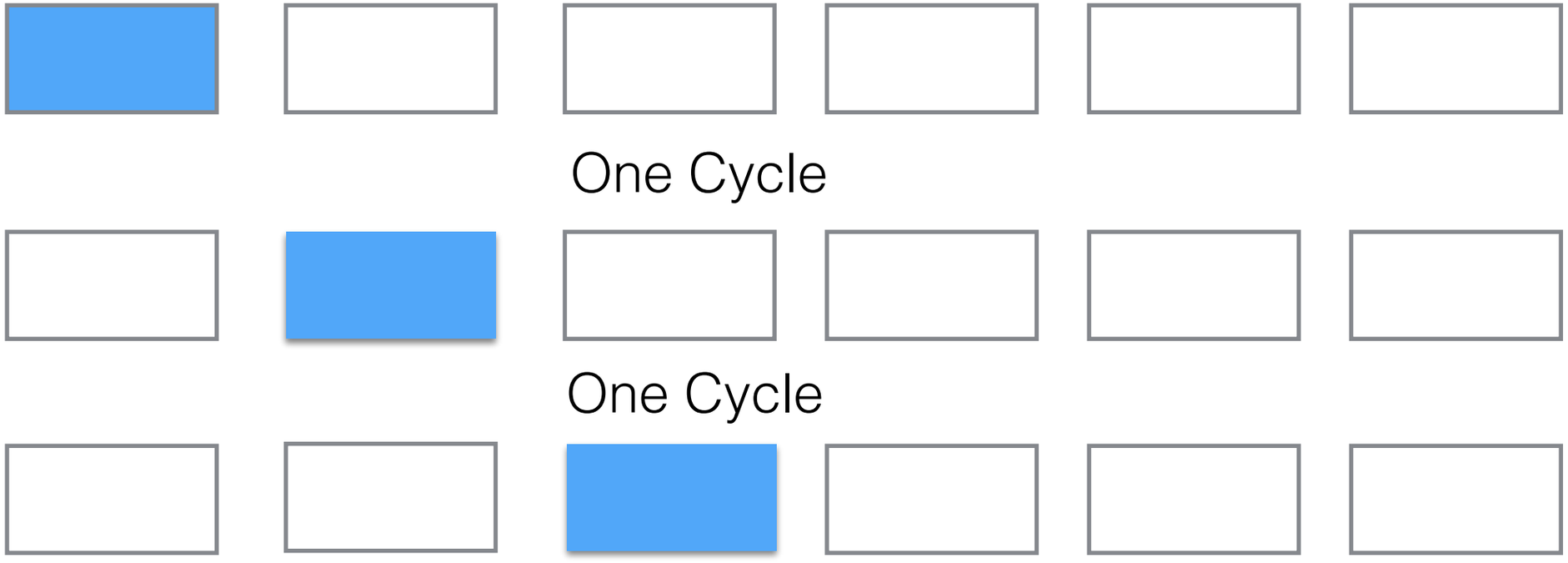
As board designed get better I/O rates, we can increase outputs per cycle by adding S new components for each new speed. Since the depth of the tree is not that large, this is not bad



Summing Results

Summing n results with a reduce structure takes log(n) levels and we would need to build a summing reduction tree from the outputs of all our decision trees.

Because our system has a predictable output pattern, we can make the summing logic simpler and lock-free (because we design hardware to avoid race conditions)



Summing is bad for the parallel C++ implementation

- Unpredictable access and completion pattern for threads based the memory accesses
- Have to use locks and mutexes (built in OpenMP reduction command) in order to get correct sums

But easy on the FPGA implementation

- Each tree outputs its completed data in a predictable pattern
- Every N consecutive outputs will belong to the same sample so the tree can also output the specific sums for each sample if necessary

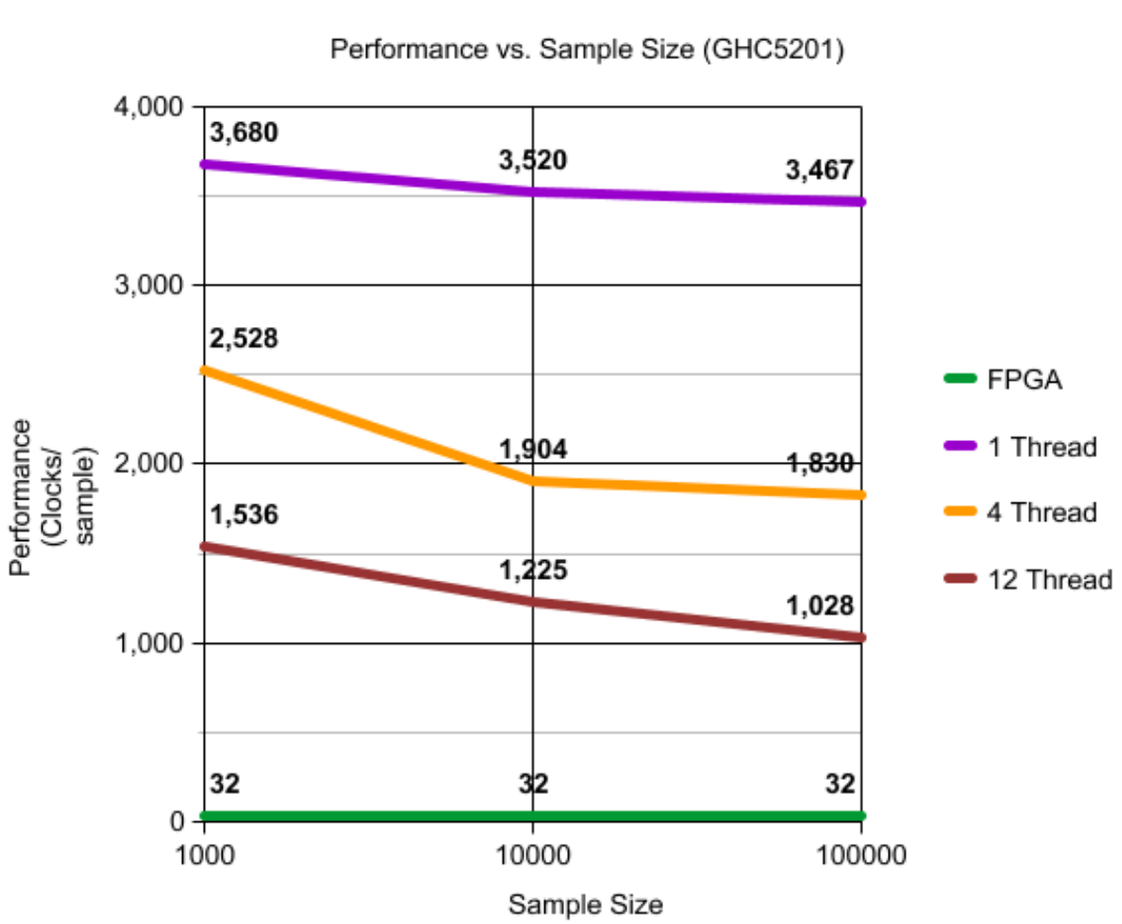
Summary of Findings

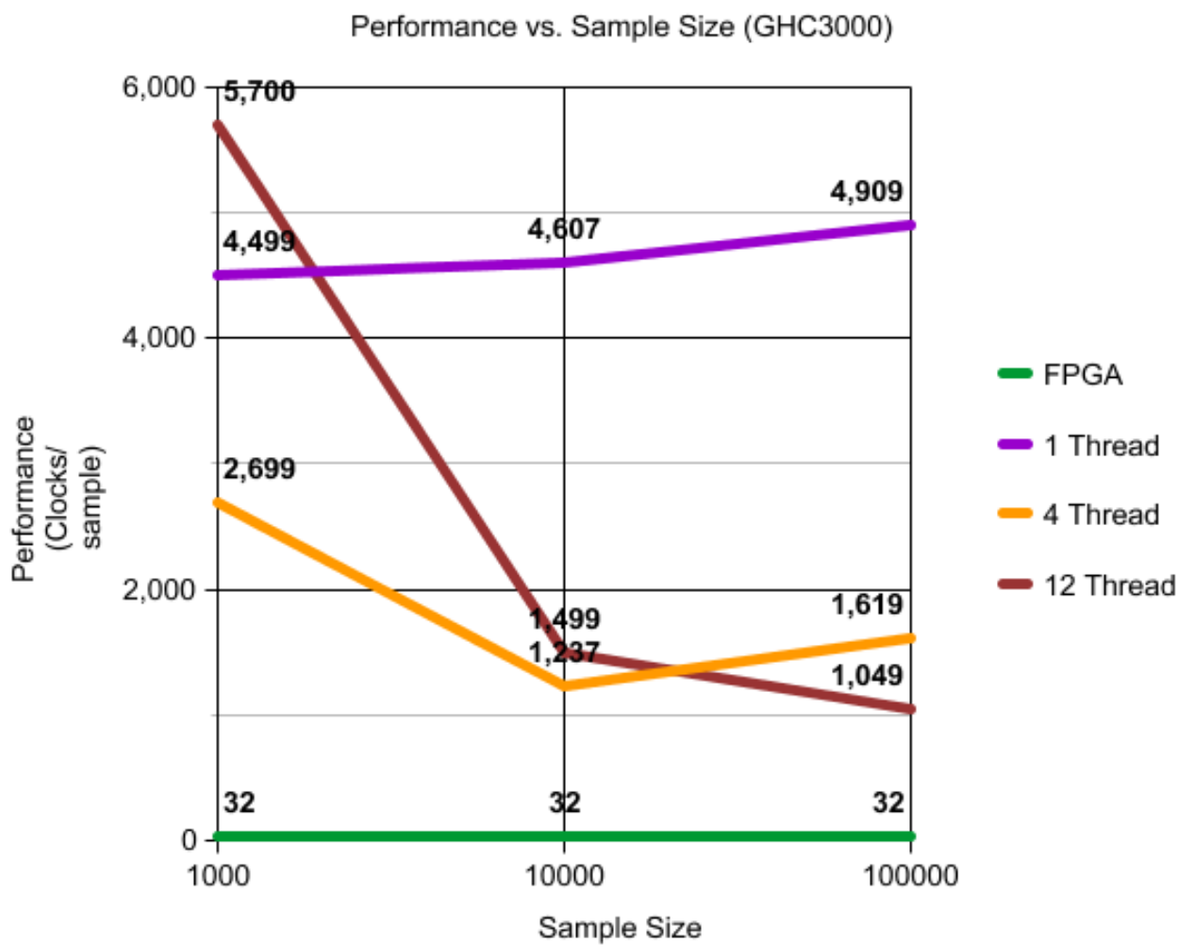
Total Execution Time (Cycles) / NumSamples

GHC 3000 : Two Core Intel Core 2 Duo, 2 physical cores, 12 OpenMP Threads

GHC5205: Four Core Intel Xeon 5600, 4 OpenMP Threads

Graph for 32 Bit Data Size, 32 Trees, Depth 6
Similar relative performance at varying forest sizes





Graph for 32 Bit Data Size, 32 Trees, Depth 6
Similar relative performance at varying forest sizes

FPGA operating at 1.2 V and I measured the current to get the effective watts of the system. Compared this to the average power consumption rating of the chips I compared the FPGA performance to

Cyclone IV FPGA	10.9 W
XeonFour Core	79 W (TDP) ~ 55 W (ACP)
Core 2 Duo	130 W (TDP) ~ 105 W (ACP)

Performance on a FPGA is roughly linear to the number of trees as long as sample size is large compared to forest size

However there is a limit on the amount of trees we can pack onto an FPGA board. Memory required by the random forest is roughly:

num_trees * [2 * 2^depth + 3 * 2^(depth-1)] * data_size

On the Cyclone IV, this implies that the maximum number of depth 6 trees for floating points that can be fit on a board is around 2000 for the memory. Because the logic of our design is minimized, the routing and logic on the board is not a resource constraint

Memory on Random Forest Chips

This algorithm is definitely memory-bound so significant resources for the chip should be put into providing a lot of fast access SRAM on the chip

Because of the stage independence in memory, it would be better to have a distributed memory system that focuses on maximizing input/output data pins. This is opposed to a larger bank of memory that sacrifices connections for the ability to store more data

I/O on a Random Forest Chip

Much of the efficiency from this implementation of the algorithm comes from the predictable data flow patterns of samples through the tree

Faster I/O on a device will directly affect the speed at which the algorithm can output data up to N data samples per cycle

If faster I/O cannot be achieved but the overall number of samples is low, it would be optimal to have a large output buffer for the I/O. Even if inflow to the buffer is faster than outflow while computation occurs, the samples could finish processing before the buffer is filled.