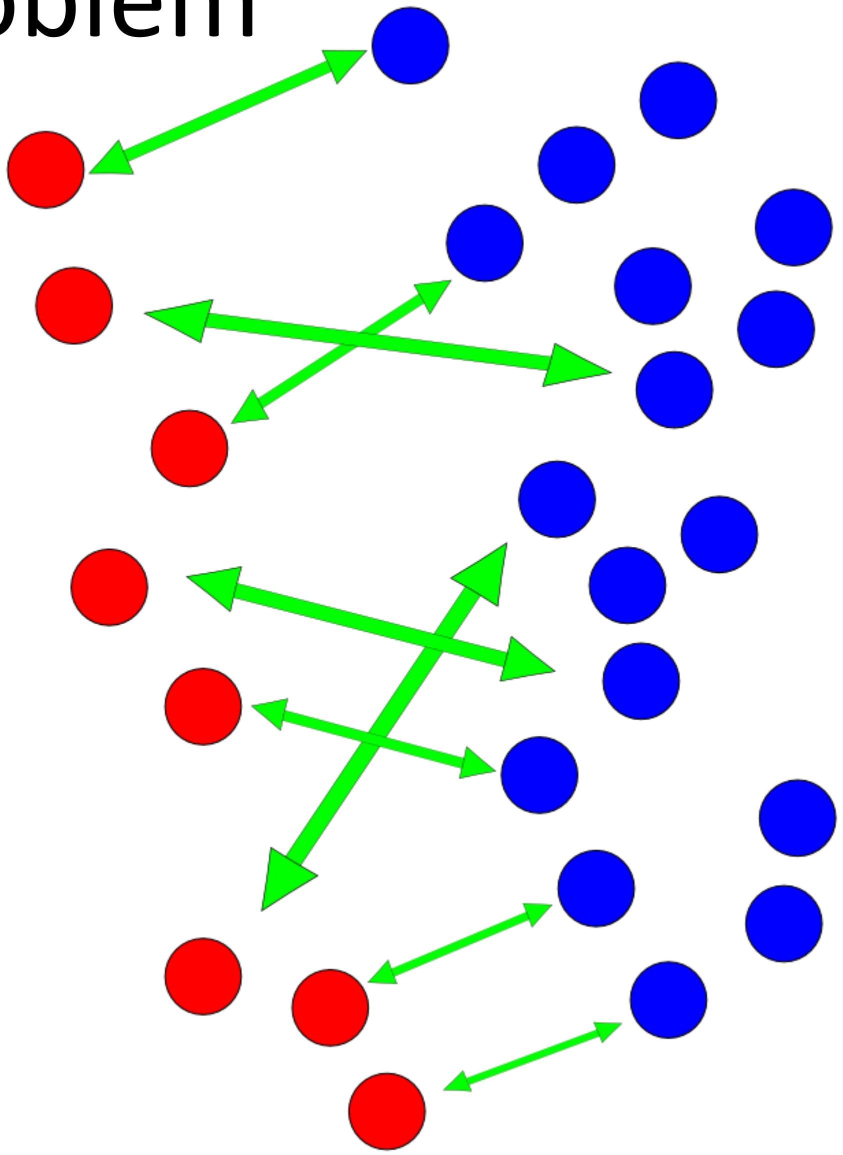
Parallelizing a T-Cell Cross-Regulation Machine Classifier

B649 12/5/12

Ian Wood and Yufan Chai

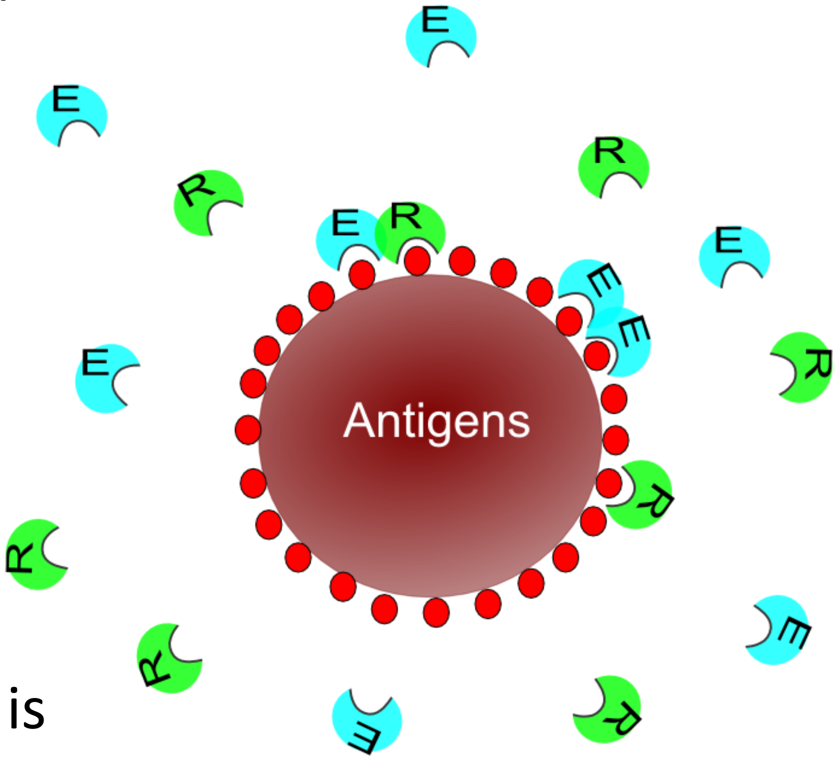
The Problem

- Two Populations of different sizes
- Each member of each population can form one, and only one, link to a member of the other population
- All member of one population should form a link if possible



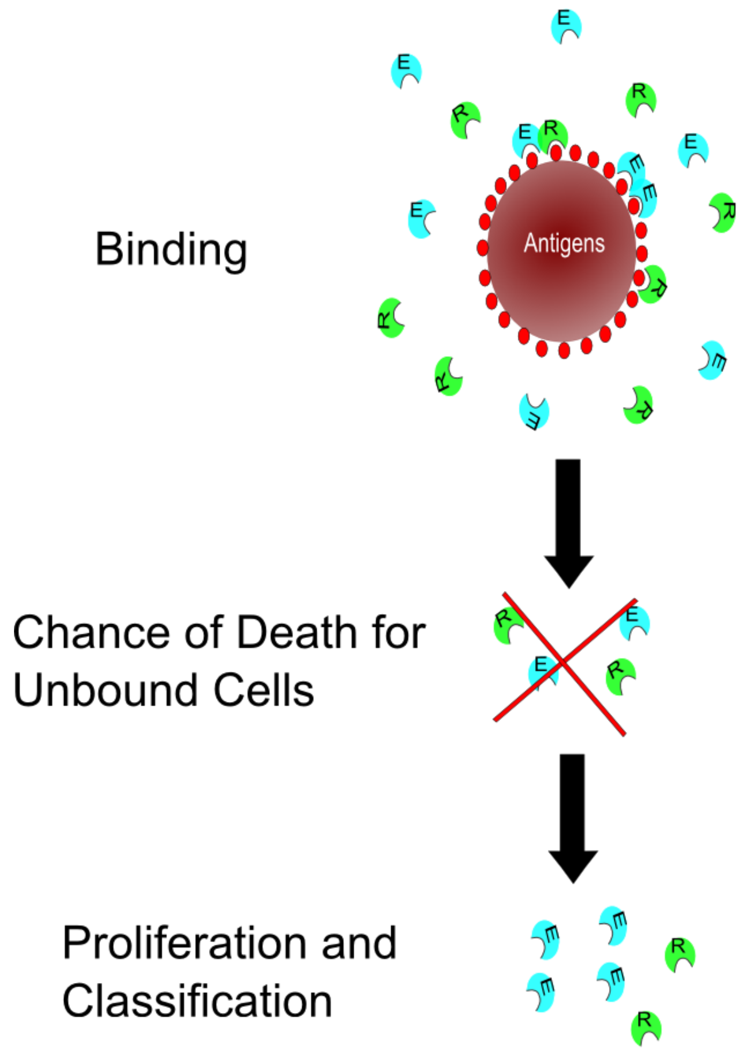
Specifics

- A machine classifier inspired by T-Cell Cross Regulation
- Documents are split into features called “Antigens”
- Populations of “T-Cells” are trained based on binding to Antigens
  - Two varieties, Effectors and Regulators
  - Initial ratio produced upon encountering a new antigen is set according to document label, constitutes training
  - Based on types of neighbors, T-cells replicate and die
  - Final ratios classify documents



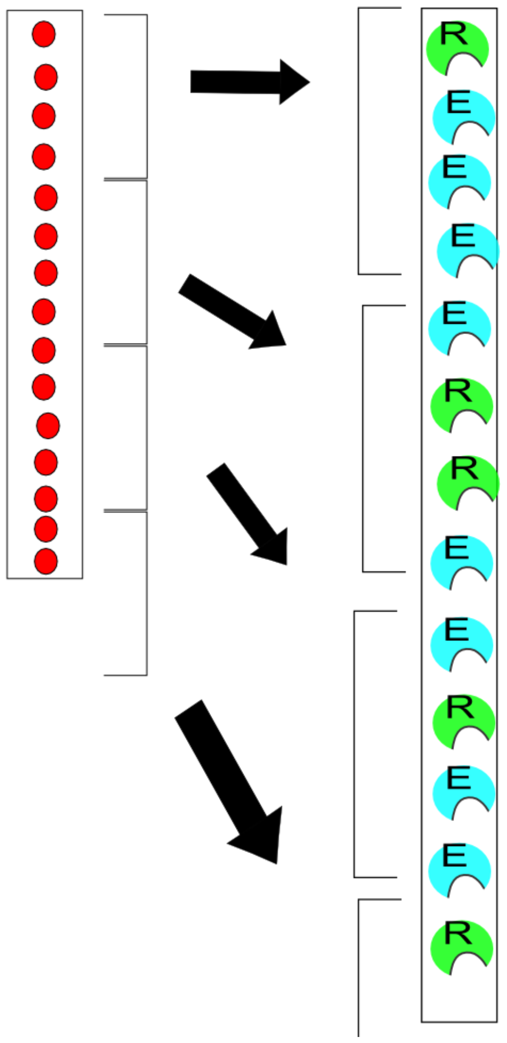
More Specifics

- Currently, T-Cells and Antigens both have strings and can bind if the strings match exactly
- Why parallelize?
  - Future Extensions will involve tracking individual tcells' age, and investigating other binding functions (substring matching, possibly regex)
- Targets for parallelization:
  - Binding Step
  - Classification Step


Binding
Antigens
Chance of Death for Unbound Cells
Proliferation and Classification
E
R

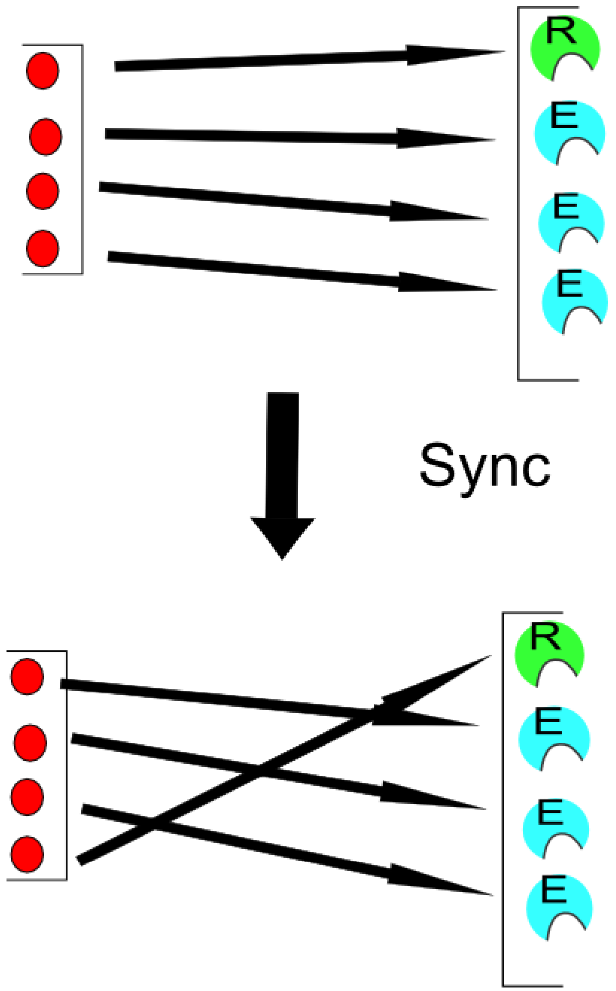
Approach: CUDA for Binding

- Binding occurs between 100,000s of Tcells and 10,000s of Antigens
- Divide Antigens into blocks for CUDA, one thread to execute for one Antigen
- Divide Tcells into same-sized blocks for CUDA
- Convert and send all Antigens and Tcells to GPU:
  - typedef struct {int matched; char str[STR_LEN];} Tcell;
  - typedef struct {int tcellid; char str[STR_LEN];} Antigen;



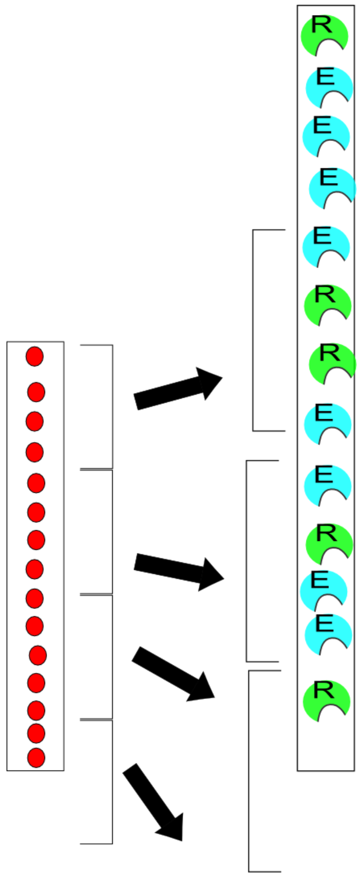
Within-Block Algorithm

- Initially, Antigen blocks and Tcell blocks are aligned by threadId
- While every antigen in the block has not been matched and has not checked every tcell in the block
  - Check whether antigen string and tcell string match
  - Update antigen and tcell pointer
  - sync_threads



Between-Block Algorithm

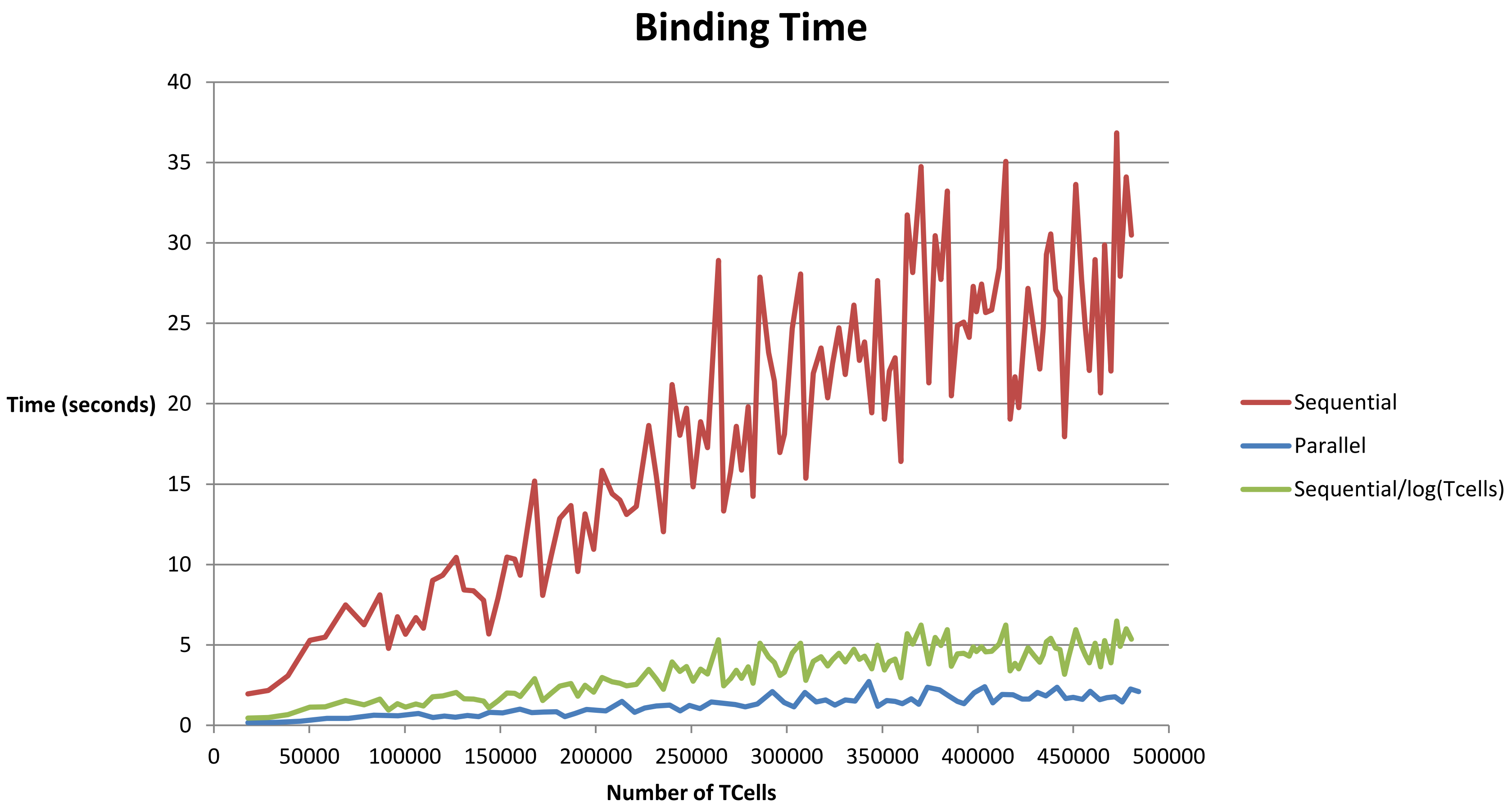
- Similar, but on a higher scale
- Initially Antigen blocks and Tcell blocks are aligned by blockId
- While any block has unmatched antigens
  - Run within-block algorithm
  - Add +1 to tcell block offset



Binding Performance

- Average binding time per document for two runs of the sequential version: 18.5743 seconds
- Average binding time per document for two runs of the CUDA parallel version: 1.650175 seconds
- Speedup = sequential/parallel = 11.256
- Specs: NVIDIA GeForce GTX 560 Ti running on Windows 7
  - 384 CUDA cores
  - 1 GB dedicated memory, 4 GB available

Binding Performance by Number of TCells



Another Algorithm

- Another AlgorithmUsing a list of Locks(mutex) instead of _syncthreads()
- Divide Antigen by Blocks, but not the Tcell
- Each Antigen locks a Tcell & does matching, doesn’t need to wait on other Antigens.
- Incomplete

Approach: TBB for Classification

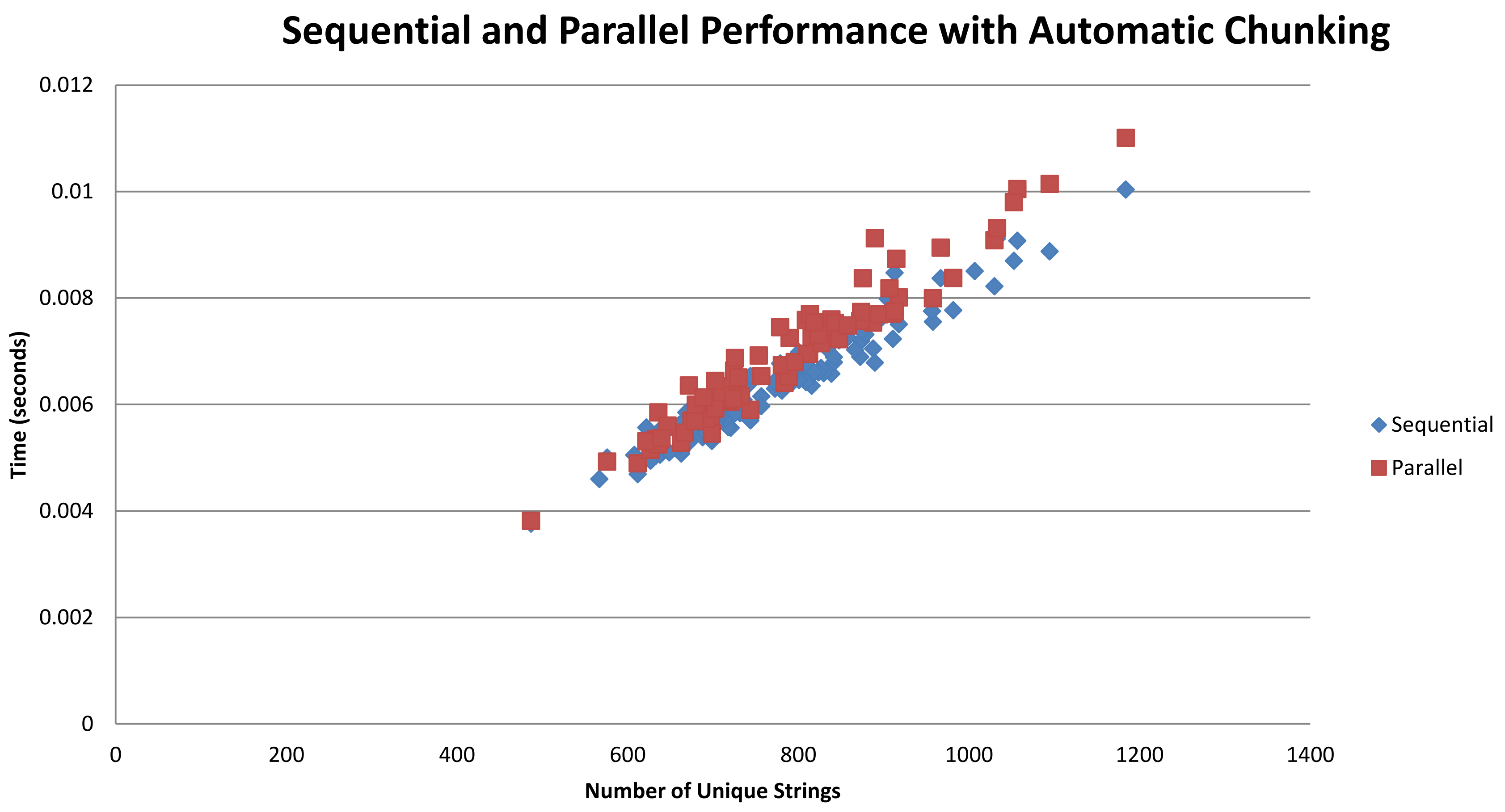
- Classification involves counting Tcells after proliferation and normalizing.
  - For every string f matching the document:
    - $Esum = \frac{ENum_f}{\sqrt{{ENum_f}^2 + {RNum_f}^2}}, Rsum = \frac{RNum_f}{\sqrt{{RNum_f}^2 + {ENum_f}^2}}$
  - Rsum > Esum ? document is relevant : irrelevant
- Approach: Simple Parallel Reduce with TBB

Classification Performance

- Average Classification time per document for two runs of the sequential version: 0.0063671
- Average Classification time per document for two runs of the parallel version: 0.0068383
- Speedup (Slowdown)= Sequential/Parallel = 0.9311

- Notes: Initially the performance greatly improved because the count was calculated during this step, however, it was much more efficient to keep a running count, which meant that TBB mostly just added overhead
  - With the change only from counting to only computing the cosine of the ratio of features, this might be another job for CUDA

Classification Performance By Number of Unique Strings



Overall Performance

- Average time for two runs of the whole program on 120 scientific articles:
  - Sequential Version: 2621.84 seconds = 43 minutes, 41.84 seconds
  - Parallel Version: 776.3875 seconds = 12 minutes, 56.3875 seconds
- Speedup= sequential/parallel = 3.377