Optical Character Recognition with CUDA C

Jeremy Reed

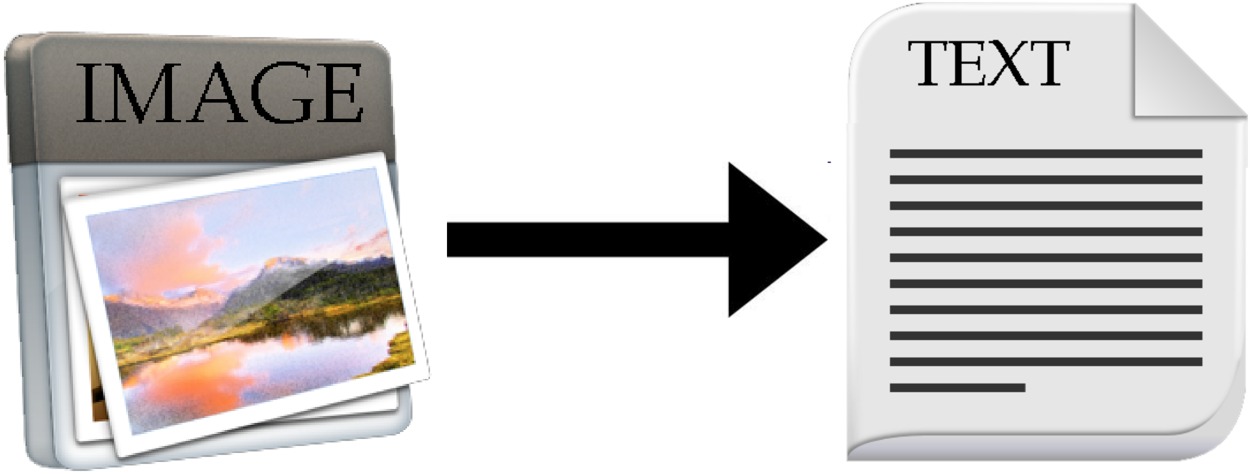
Definitions:

- GPU: Graphics Processing Unit
- CUDA: Compute Unified Device Architecture (NVIDIA standard)
- CUDA C: An extension of the C programming language used to interface with and program NVIDIA GPUs.
- Thread: A single path of execution.
- Block: A group of threads.

NVIDIA GeForce GTX 260

- 896MB RAM
- 192 multi-processors (MP)
- Each MP holds 8 “streaming processors” (SP) @ ~1.2GHz
- Each SP can execute 1 Block of up to 512 threads.
- 192 x 8 x 512 = 786,432 threads

Optical Character Recognition (OCR)



- The task of turning images into text

OCR Engine Overview

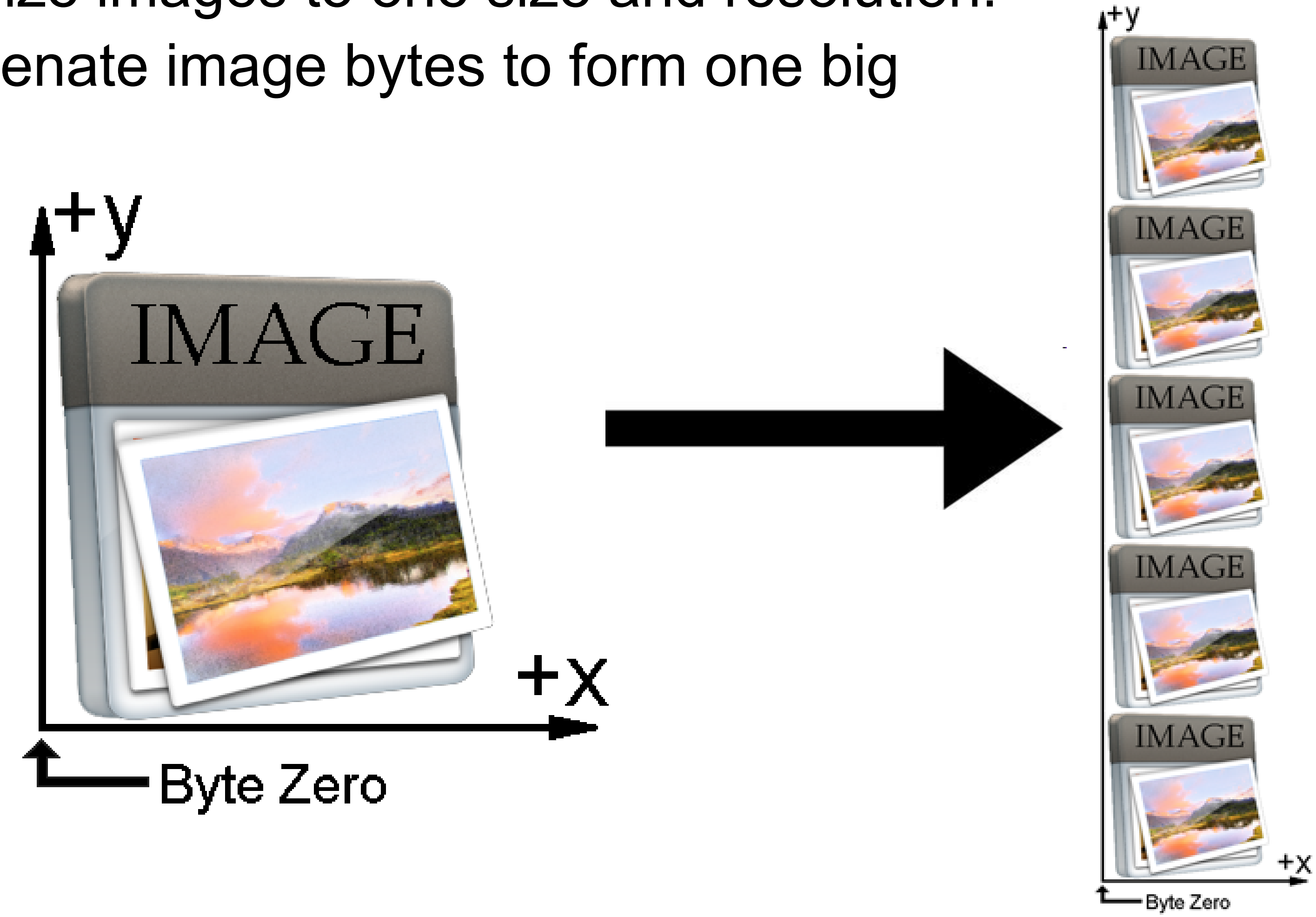
- Pre-processing: remove pixel noise, correct rotated images, determine font size/style, adjust threshold for monochrome conversion…
- Isolation: find blocks of text, lines and individual glyphs
- Identification: identify each glyph
- Post-processing: re-assemble document in text format, use spell checker or dictionary to enhance accuracy on a word level…

The Game

- How do we utilize the massively parallel architecture of the GPU to perform OCR?
  - How do we organize the data?
  - How do we split execution within the OCR engine?

Data Organization

- Normalize images to one size and resolution.
- Concatenate image bytes to form one big array.



Execution Organization

- The recognition process has 5 subtasks:
  1. Horizontal bit count to find lines
  2. Vertical bit count within line boundaries to find glyph boundaries
  3. Horizontal bit count within each glyph to trim space from edges
  4. 3x3 region bit counter for each glyph within glyph boundaries to produce the global density vector
  5. Brute-force nearest-neighbor search to identify each glyph

- For each subtask, assign a thread:
  1. x pixel rows
  2. y lines
  3. z glyphs
  4. z glyphs
  5. z glyphs

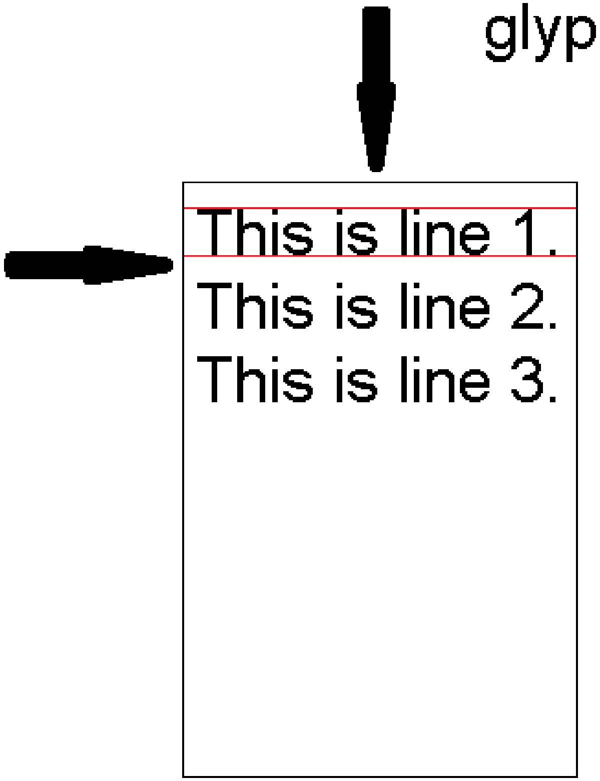
Execution Organization - 2

1.

0 black pixels indicate space between lines

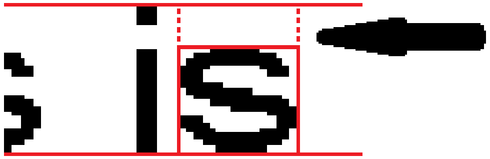
2.

0 black pixels within line boundaries indicate vertical glyph boundaries

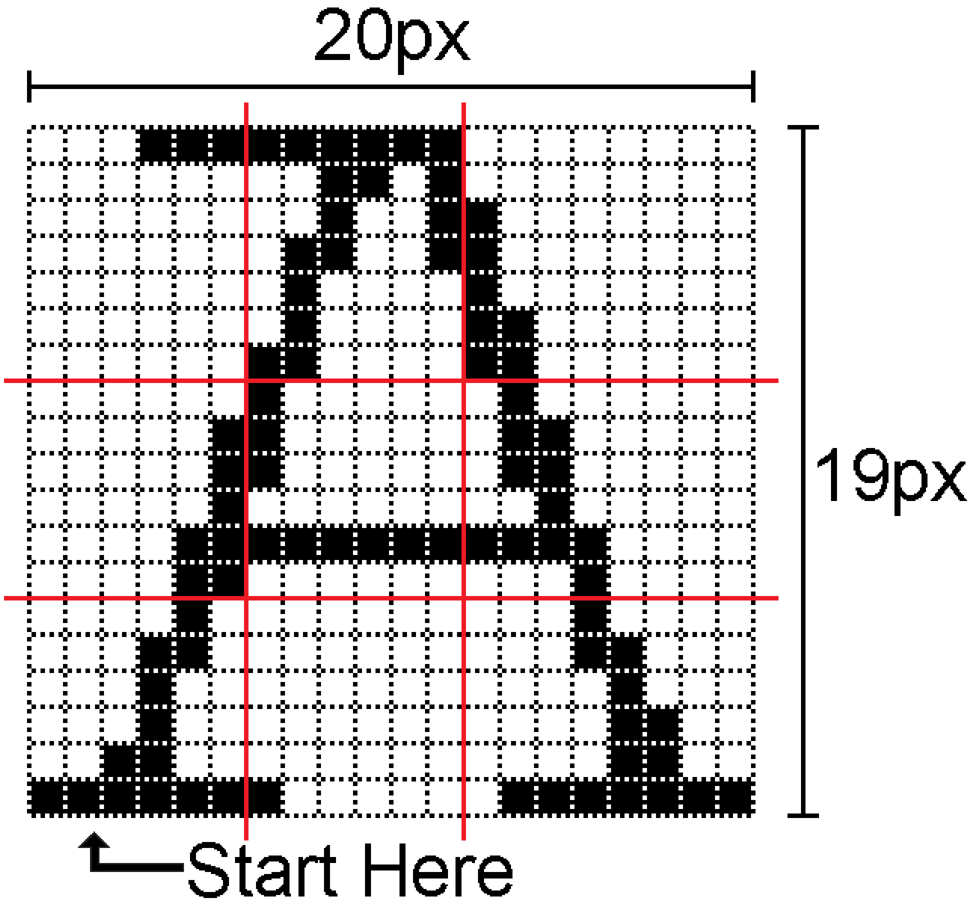


3.

0 black pixels within vertical glyph boundaries indicate glyph edges



A Recognition Algorithm:
3x3 Global Density



- Region Counts:

  (13, 1, 15, 7, 9, 11, 3, 18, 7)

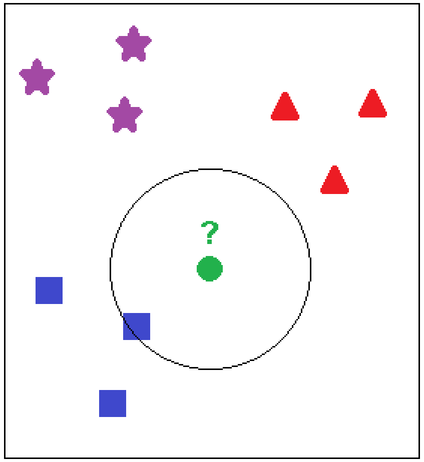
- Region Counts / Total Area = Global Density

- Global Density Vector:

  (.034, .002, .039, .018, .024, .029, .008, .047, .018)

3x3 Global Density - 2

- Brute-force nearest-neighbor search to identify the glyph.

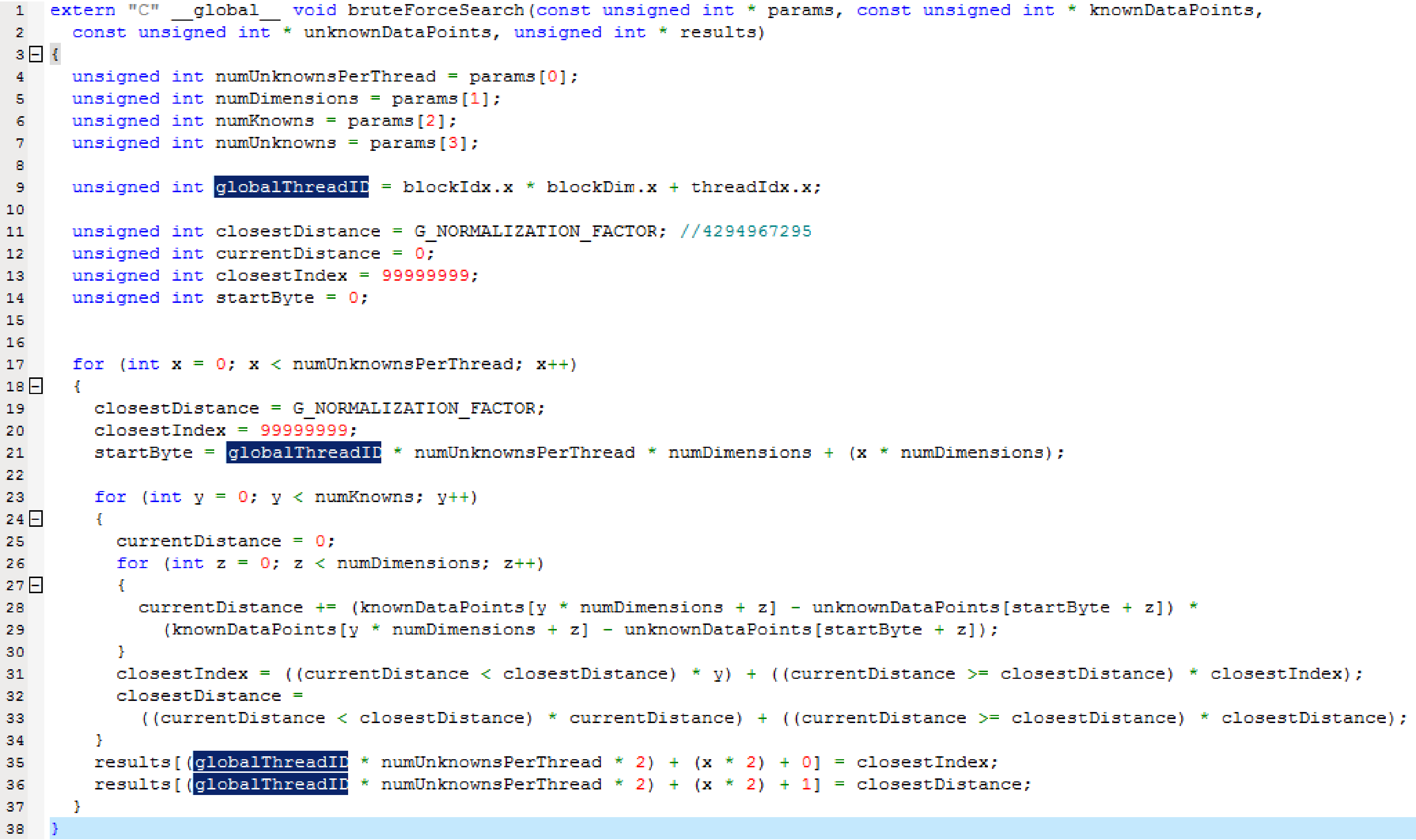


Unknown Vector:
(.034, .002, .039, .018, .024, .029, .008, .047, .018) ●

Known Vectors (or Training Set):
(.034, .002, .039, .018, .024, .029, .008, .047, .018) = A ■
(.026, .016, .032, .018, .016, .032, .029, .016, .029) = B ▲
(.014, .019, .028, .033, .000, .000, .031, .019, .031) = C ★
…

- Can be 5x5, 7x7, 5x7, etc.
- General purpose algorithm, theoretically works for all alphabets, fonts

CUDA C Example
Brute Force Search



```
extern "C" __global__ void bruteForceSearch(const unsigned int * params, const unsigned int * knownDataPoints,
  const unsigned int * unknownDataPoints, unsigned int * results)
{
  unsigned int numUnknownsPerThread = params[0];
  unsigned int numDimensions = params[1];
  unsigned int numKnowns = params[2];
  unsigned int numUnknowns = params[3];

  unsigned int globalThreadID = blockIdx.x * blockDim.x + threadIdx.x;

  unsigned int closestDistance = G_NORMALIZATION_FACTOR; //4294967295
  unsigned int currentDistance = 0;
  unsigned int closestIndex = 99999999;
  unsigned int startByte = 0;


  for (int x = 0; x < numUnknownsPerThread; x++)
  {
    closestDistance = G_NORMALIZATION_FACTOR;
    closestIndex = 99999999;
    startByte = globalThreadID * numUnknownsPerThread * numDimensions + (x * numDimensions);

    for (int y = 0; y < numKnowns; y++)
    {
      currentDistance = 0;
      for (int z = 0; z < numDimensions; z++)
      {
        currentDistance += (knownDataPoints[y * numDimensions + z] - unknownDataPoints[startByte + z]) *
          (knownDataPoints[y * numDimensions + z] - unknownDataPoints[startByte + z]);
      }
      closestIndex = ((currentDistance < closestDistance) * y) + ((currentDistance >= closestDistance) * closestIndex);
      closestDistance =
        ((currentDistance < closestDistance) * currentDistance) + ((currentDistance >= closestDistance) * closestDistance);
    }
    results[(globalThreadID * numUnknownsPerThread * 2) + (x * 2) + 0] = closestIndex;
    results[(globalThreadID * numUnknownsPerThread * 2) + (x * 2) + 1] = closestDistance;
  }
}
```

Theorycrafting for Fun!

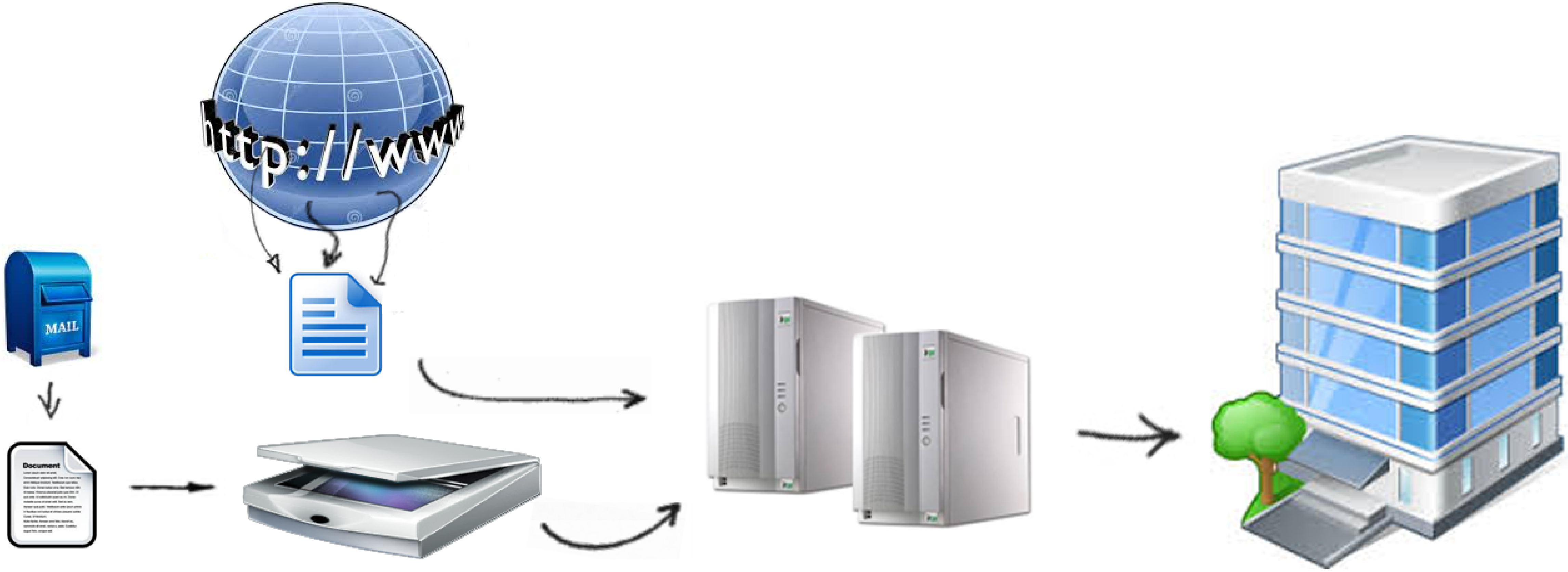
- The UK “NAK” GPU Cluster contains 64 nodes, each equipped with a NVIDIA GeForce 9500 GT GPU (512MB RAM, 32 SMs)
  - Could theoretically maintain a rate of roughly ~8,000 pages per second (~32,000 pages per run / ~4s per run)
  - Would only take ~2.5 years to OCR all print materials in the Library of Congress

Business Need

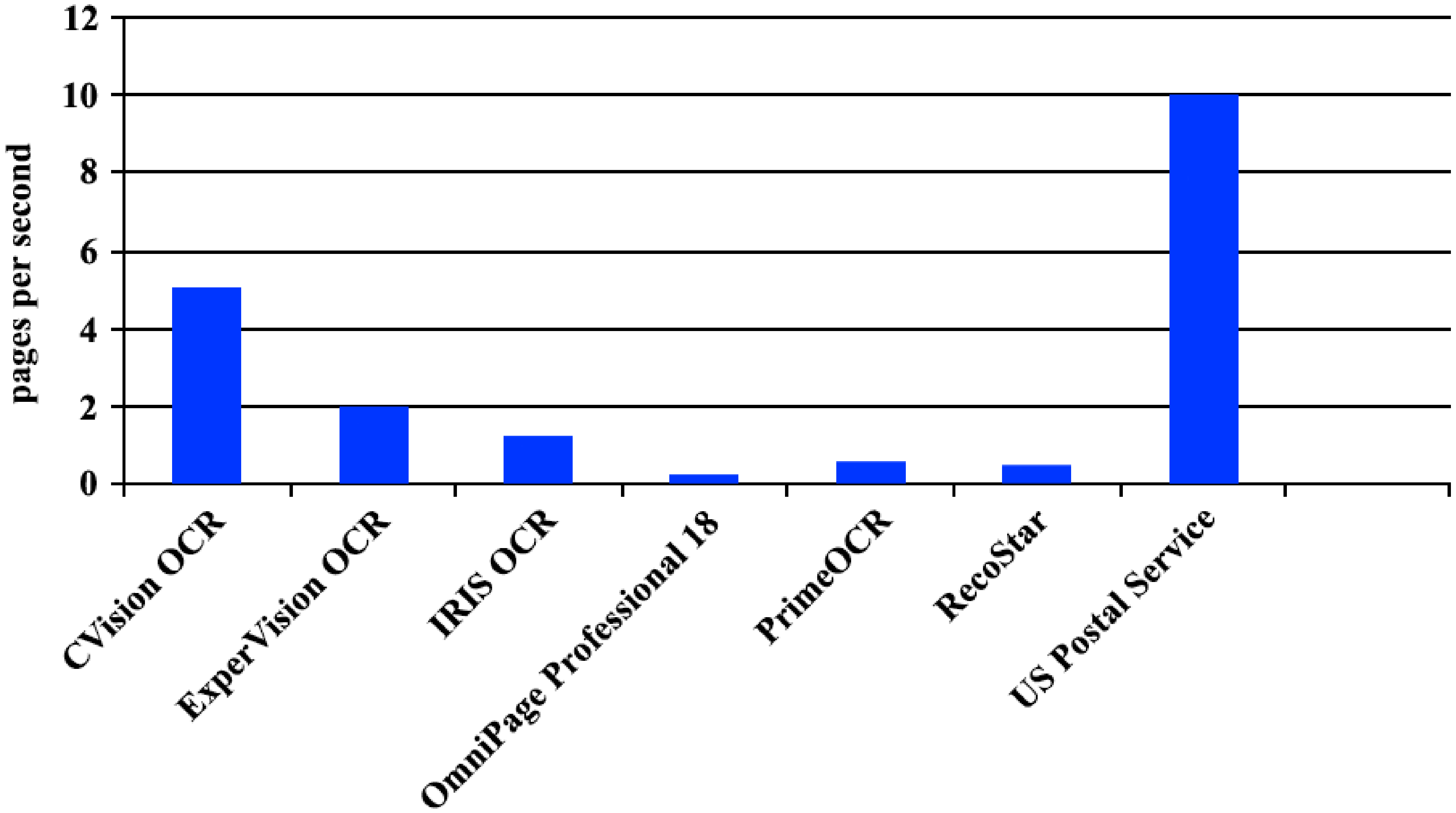
I have thousands of documents:

- Clinical report forms
- Insurance claim forms
- Benefit claim forms
- Other forms

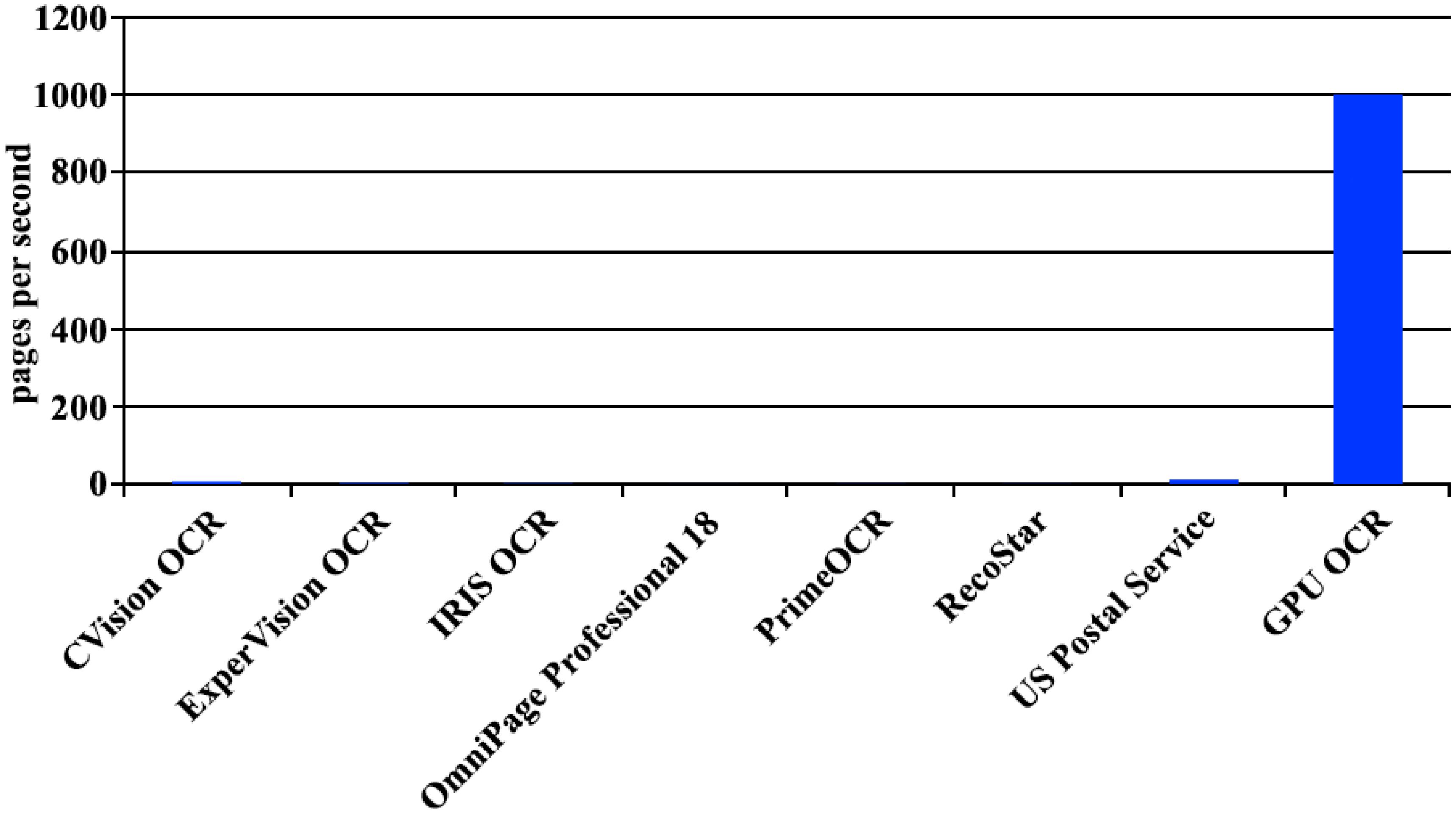
Typical Document Workflow



OCR is a Bottleneck!



Not anymore!



pages per second
1200
1000
800
600
400
200
0
CVision OCR
ExperVision OCR
IRIS OCR
OmniPage Professional 18
PrimeOCR
RecoStar
US Postal Service
GPU OCR

Why is it so fast?

- pages can be turned into increasingly denser grids
- lots of repetitive operations
- majority of operations are bit counts (shifts and ANDs)

Why is it so fast?

- pages can be turned into increasingly denser grids
- lots of repetitive operations
- majority of operations are bit counts (shifts and ANDs)

It can be faster!!

Scalable

- ~1000 pages per second per 1.5GB of RAM*
- ~$10k server will do ~10,000 pgs / sec
- As technology scales, so will the software

Limitations and Next Steps

- Prototype is currently limited to printed text with characters separated by at least a 1-pixel wide vertical strip of white space

- New hardware with DP will allow for a **substantial** decrease in memory overhead, increasing page throughput
- Support for ligatures, kerning, hand-written text, on-device pre-processing, document identification, and more

Limitations and Next Steps

- Prototype is currently limited to printed text with characters separated by at least a 1-pixel wide vertical strip of white space

- New hardware with DP will allow for a **substantial** decrease in memory overhead, increasing page throughput
- Support for ligatures, kerning, hand-written text, on-device pre-processing, document identification, and more

Where do you want it to go?

Questions

?

jpreed00@gmail.com