Writing the Viterbi Algorithm in PyCUDA for CPM decoding

Project Presentation

Jon Klein

University of Alaska, Fairbanks

December 7, 2011

PyCUDA is a python wrapper for the CUDA API

PyCUDA is a python wrapper for the CUDA API written in C++ and python. It provides access to the CUDA API from python with a few nice features:

- convenient memory allocation and matrix operations
- automatic error checking
- automatic object cleanup

This makes PyCUDA suitable for rapid prototyping of CUDA applications.

I tried several ways of optimizing the CUDA algorithm..

- Moving constant matricies from global to constant memory pause slows down execution time by 50%
- Moving less than 8kB of data to constant memory speeds up execution time by 5%
- Moving remaining matricies to shared memory speeds up execution time by another 5%
- Reducing communication between the host and devices provides a 120% speedup.
- Discarding intermediate values of probability matrix reduces shared memory consumption, permitting more states or longer observation periods.

I tried several ways of optimizing the CUDA algorithm..

- Moving constant matricies from global to constant memory pause slows down execution time by 50%
- Moving less than 8kB of data to constant memory speeds up execution time by 5%
- Moving remaining matricies to shared memory speeds up execution time by another 5%
- Reducing communication between the host and devices provides a 120% speedup.
- Discarding intermediate values of probability matrix reduces shared memory consumption, permitting more states or longer observation periods.

The Viterbi Algorithm

The Viterbi algorithm finds the most likely sequence of states given the outputs. It is used for pattern recognition, error-correcting codes, and detecting signals with memory [1].

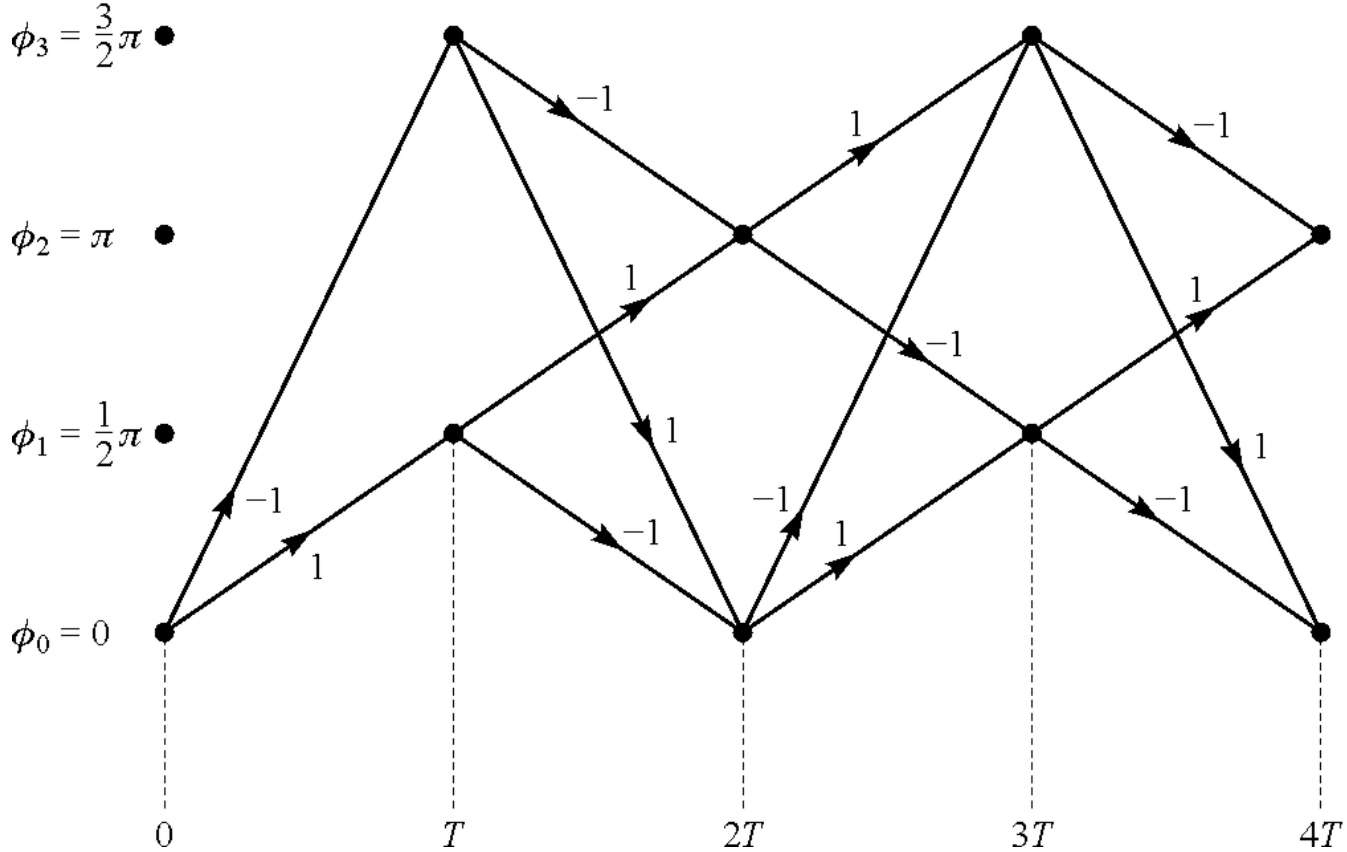


Figure: Trellis diagram of $\frac{\pi}{2}$ full response CPM [2]

The Viterbi algorithm is parallelizable

- The path leading to a state depends on calculating the path to previous states. The probability of each state at a moment in time can be calculated in parallel.
- The Viterbi algorithm can be run on multiple sequences in parallel.
- Many of the inputs are constant between between iterations, and can be left on device memory.

The probability of each state depends only on path states

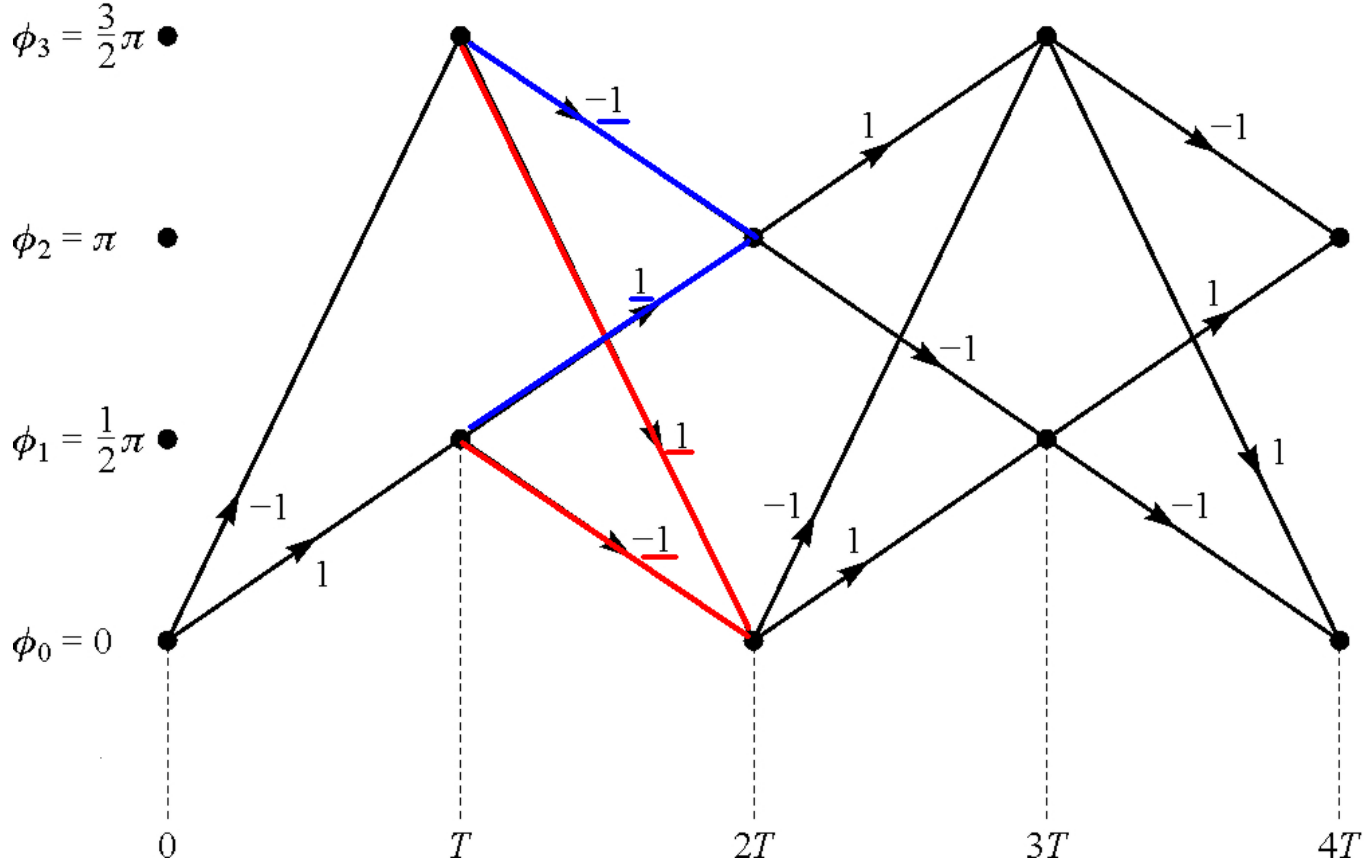


Figure: Decoding trellis diagram at time T=2Ts

CUDA parallelized pyterbi is at least an order of magnitude faster than the host

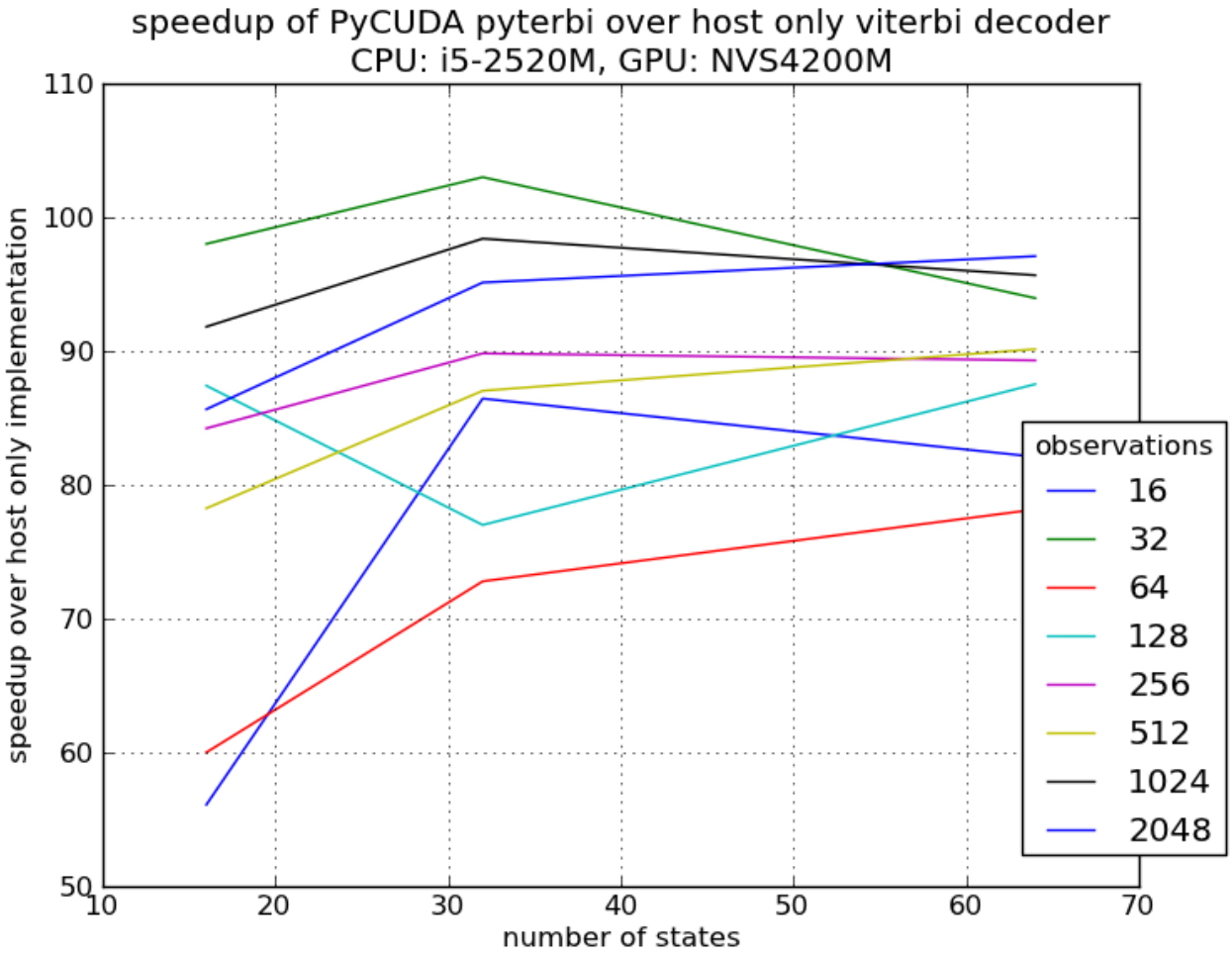
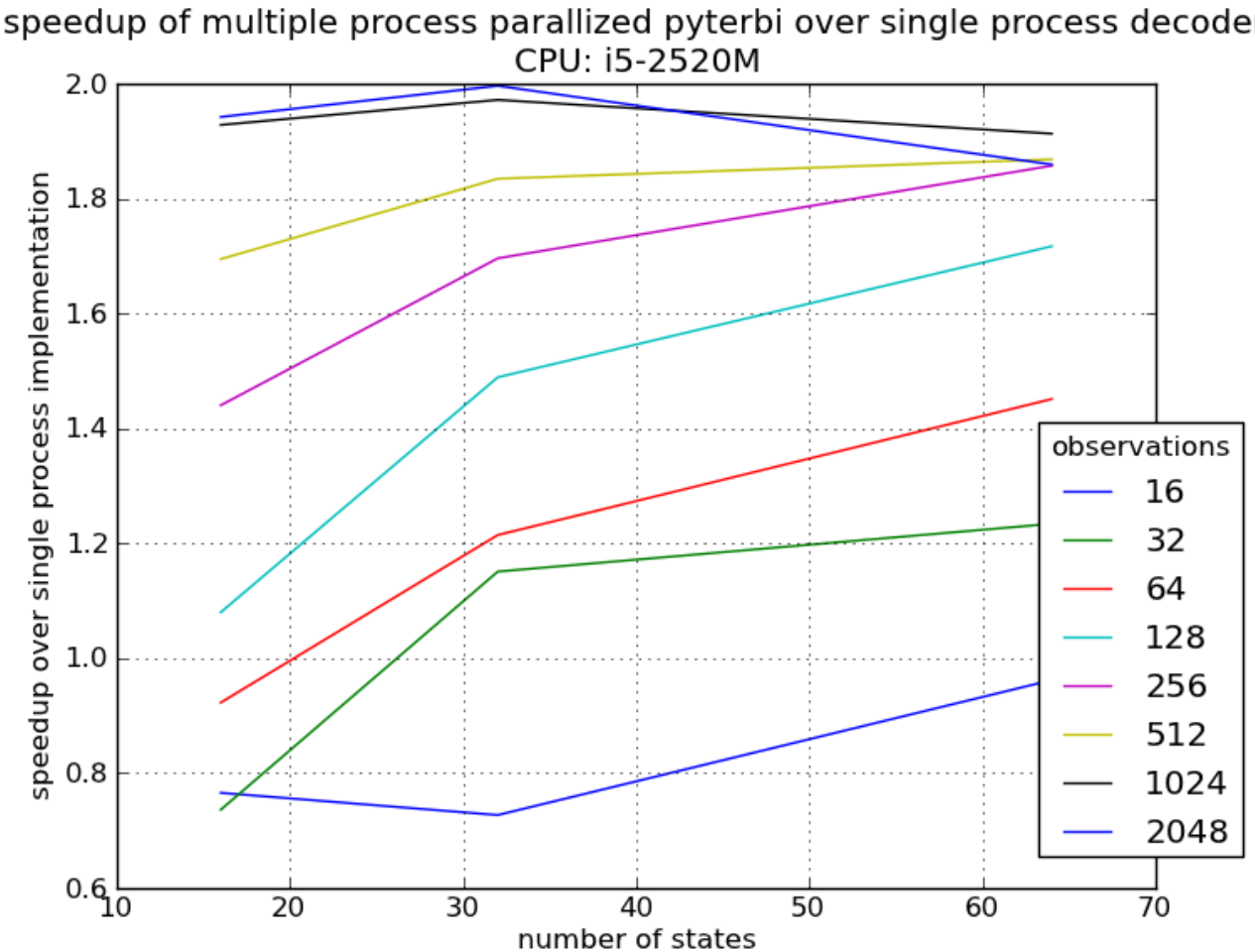


Figure: Speedup from parallelizing viterbi algorithm using CUDA

SMP parallelized pyterbi will speedup sufficently complex workloads

Pyterbi uses the pp python library to parallize pyterbi by splitting trellises across multiple cores.



Cluster parallelized Viterbi works, but currently doesn't make sense..

The pp parallel python library which pyterbi uses for SMP also works for computer clusters. In this mode, the host computer dispatches trellises to an arbitrary number of pp server nodes.

Node 1

- Intel i5-2520M, 2.50GHz
- 4096MB RAM
- Tethered on AT&T Cell Phone
- Ubuntu 11.04
- Fairbanks, Alaska

Node 2

- Shared 2.00 GHz AMD Opteron
- 192MB RAM
- 5 MBps Uplink
- Ubuntu 11.04
- VPS in New Jersey

Project Results

I have written, tested, and benchmarked:

- a reference host-only implementation of the Viterbi algorithm in Python.
- a CUDA parallelized Viterbi algorithm.
- a multi-process and cluster parallelized Viterbi implementation

There are some possible improvements remaining:

- Rewrite code in a faster language (C)
- Test kernel concurrency on more powerful graphics cards.
- Test realistic cases of cluster computing.
- Get pycuda working with pp for CUDA accelerated cluster computations.

The pyterbi source code is available at
`http://github.com/loxodes/pyterbi`

References

1. Lou, H.-L.; , "Implementing the Viterbi algorithm," Signal Processing Magazine, IEEE , vol.12, no.5, pp.42-52, Sep 1995
2. John Proakis. Digital Communications. McGraw-Hill Science/Engineering/Math, 5 edition, 2007.
3. C. Liu, "CuHMM: a CUDA Implementation of Hidden Markov Model Training and Classification," 2009