Composing Finite State Transducers on GPUs

Arturo Argueta & David Chiang
{aargueta,dchiang}@nd.edu,
Computer Science and Engineering, University of Notre Dame

Abstract

Weighted finite automata[1] and transducers are widely used in Natural Language Processing (NLP) to perform tasks such as morphological analysis, part-of-speech tagging, chunking, named entity recognition, speech recognition, and others.

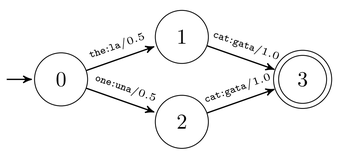
We introduce a Graphics Processing Unit (GPU) implementation of the composition algorithm, achieving speedups of up to 6x over serial implementations and 4.5x over OpenFST.

Merrill et al.[2] showed that GPUs can be used to accelerate breadth-first search to traverse up to 3.3 billion edges per second.

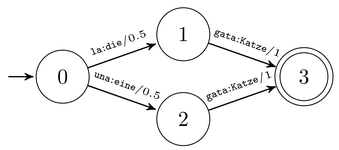
Composition Algorithm

The composition algorithm feeds the output symbols from transducer $M_1=(Q_1, \Sigma, \Gamma, s_1, F_1, \delta_1)$ to a transducer $M_2=(Q_2, \Gamma, \Delta, s_2, F_2, \delta_s)$ where:

- **Q** a set of states
- **Σ** finite input vocabulary
- **Γ** finite output/input shared vocabulary
- **Δ** finite output vocabulary
- **s** a start state
- **F** a set of final states
- **δ** transition function: if M is in state q and the next input symbol is a, then **δ**[a][q,q'] is the weight of going to state q'.

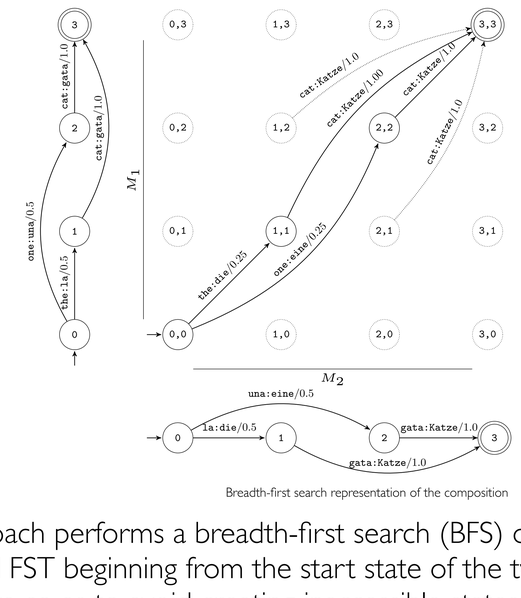


M_1



M_2

Method



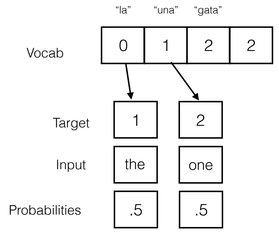
Breadth-first search representation of the composition

Our approach performs a breadth-first search (BFS) of the composed FST beginning from the start state of the two input transducers, so as to avoid creating inaccessible states. As it is standard, the BFS uses two data structures, a frontier queue (A) and a visited set (V), which is always a superset of A. For each state pair q_1,q_2 (q_1 representing a state from M_1 and q_2 from M_2) popped from A, the algorithm composes all transitions that share the same symbol from the shared vocabulary. The composed edges are added to the final transducer, and the corresponding target states are added to the frontier for future expansion.

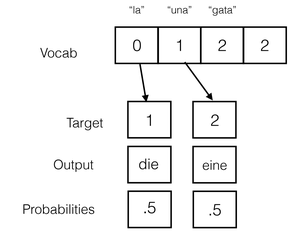
Results

Method		Time(s)	Speedup
Serial	OpenFST	374	4.52
	Our Serial	518	6.26
Parallel	Our CUDA	82.9	1.00

Representations



M_1 transitions leaving state 0



M_2 transitions leaving state 0

We store Finite State Transducers using a combination of compressed sparse row (CSR) matrix format and coordinate (COO)[3]. For each pair of states q_1,q_2 on the input transducers M_1,M_2, a thread is launched for each edge with output/input symbol w_t from the shared vocabulary **Γ**. Equivalently, one thread is launched for each k such that $Vocab[w_t] \leq k < Vocab[w_t+1]$, for a total of $Vocab[w_t+1] - Vocab[w_t]$ threads for each state in M.

Data

The probabilities $P(f \mid e)$ were obtained by using GIZA++ and used to train a first transducer. Word level alignments are used to generate a word-level transducer. To generate a transducer that reads an utterance of length n and output a translation, the automata mentioned earlier are intersected with the word level transducers. De-en and en-de transducers were trained from the Europarl corpus using 150k parallel lines.

References

1. Mohri, M et al. (2002) Weighted finite-state transducers in speech recognition. Computer Speech and Language. pages 69-88
2. Merrill, D. et al. (2012). Scalable GPU graph traversal. *Proc. 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pages 117-128.*
3. Argueta, A. et al. (2017). Decoding with Finite-State Transducers on GPUs. *Proc. of the 15th Conference of the European Chapter of the Association for Computational Linguistics. pages 1044-1052.*