Parallelizing a Statistical Machine Translator

Berkeley ParLab

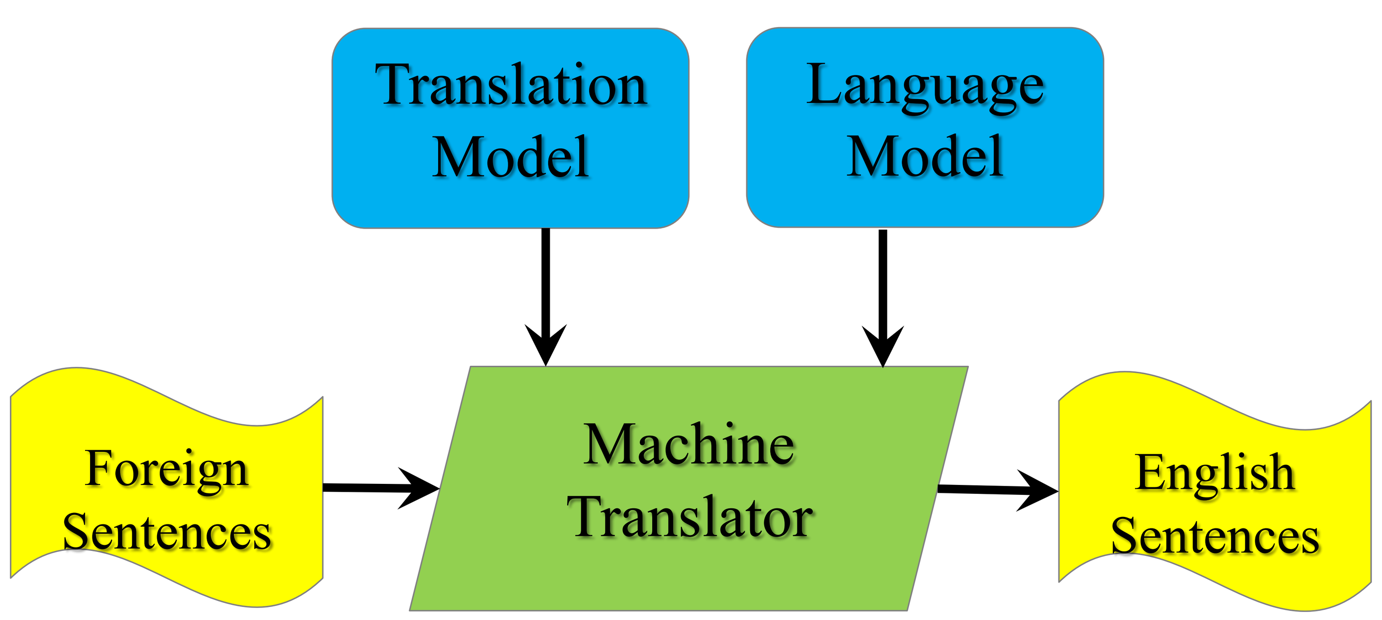
Chao-Yue Lai, Katya Gonina, Kurt Keutzer

Machine Translation

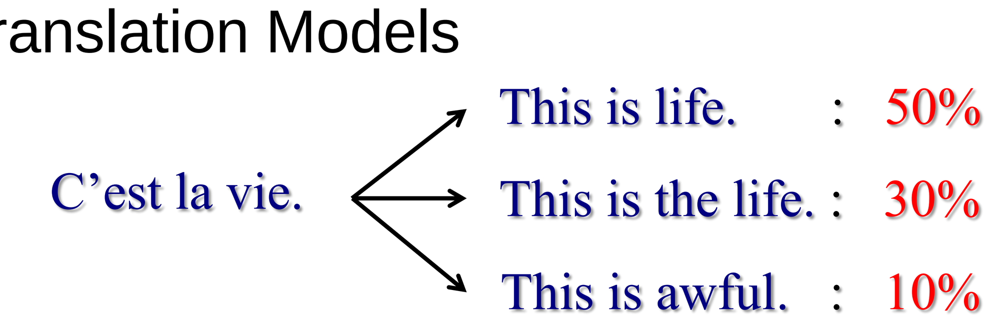
- High-quality translation is important.

- But high-quality translation: minutes for a 50-word sentence.
  - Huge language models
  - Complicated algorithm
- Need for speedups
  - Tons of texts on the Internet
  - Enable read-time translators

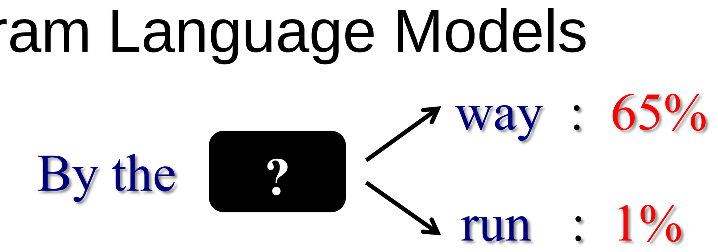
Overview of a Statistical Machine Translator



- Models: statistical information gathered from a large set of written documents
- Translation Models



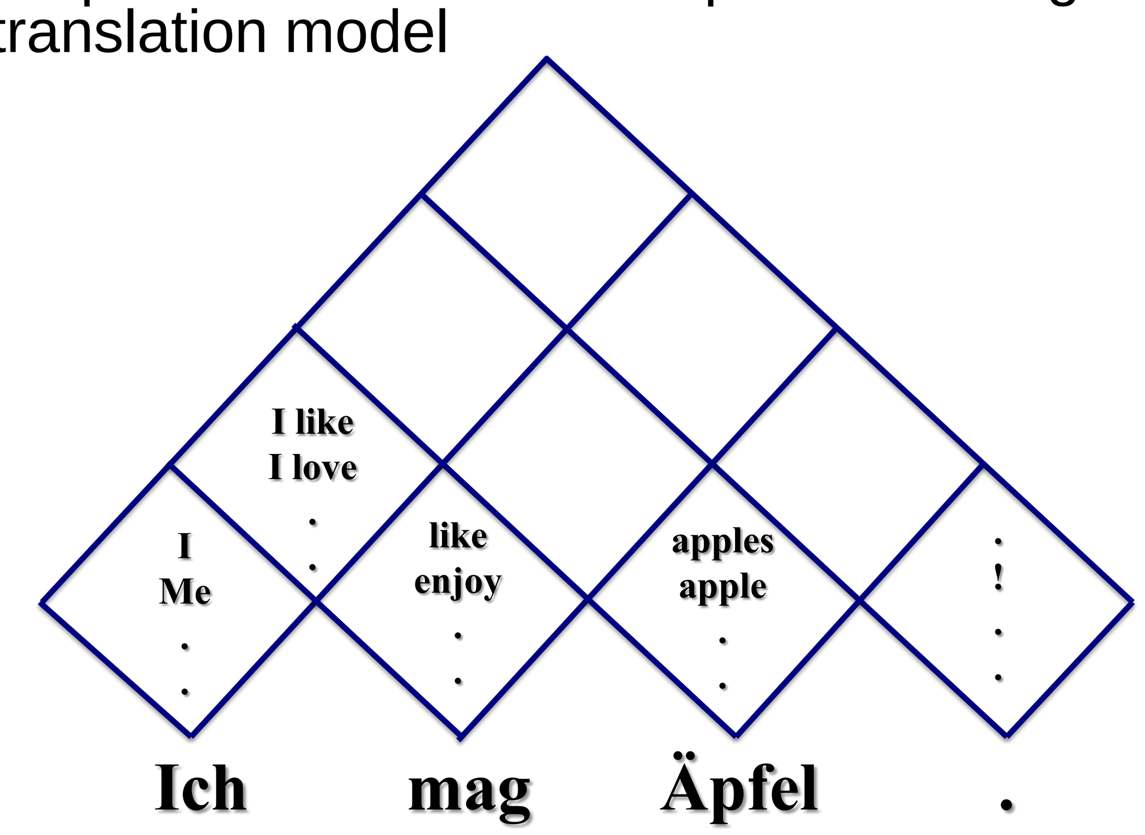
- N-gram Language Models



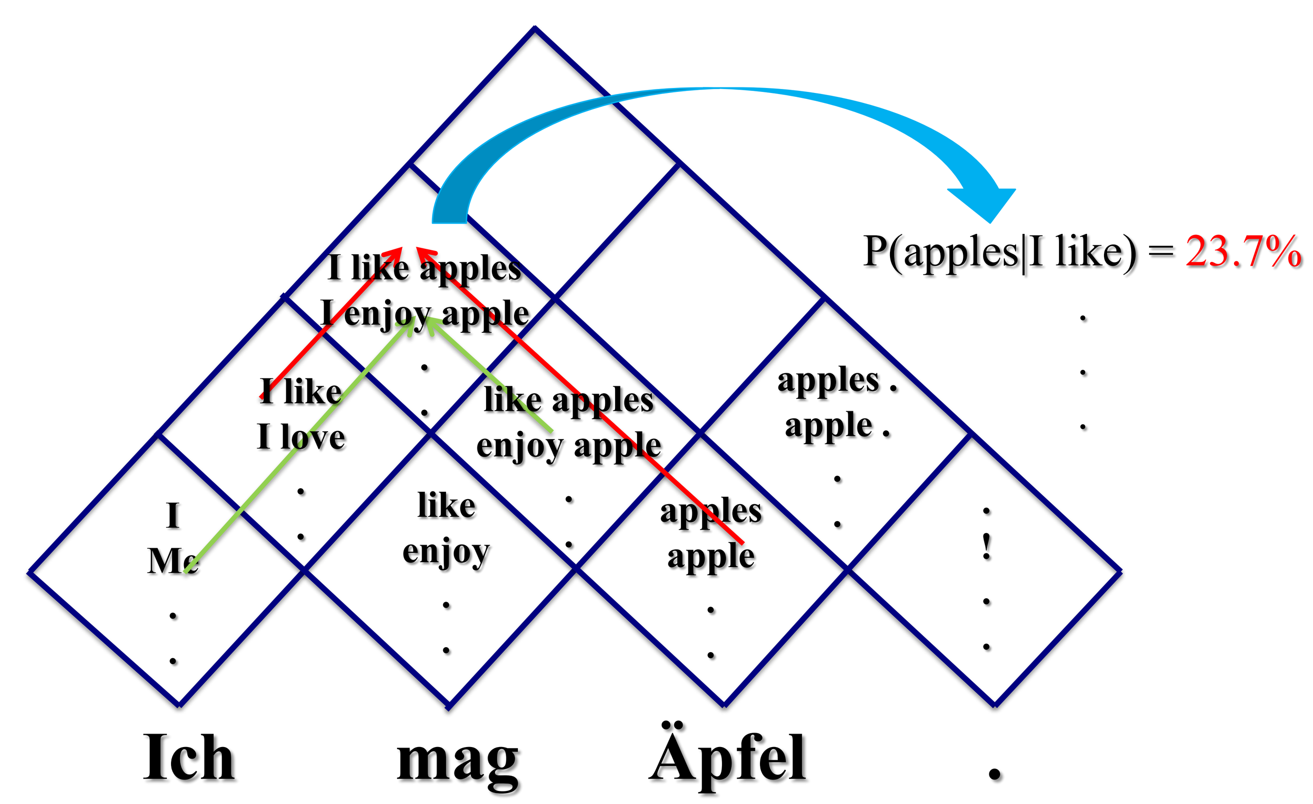
- P(N-th word | preceding (N-1) words)
- N can be up to 10 for quality

How it Works

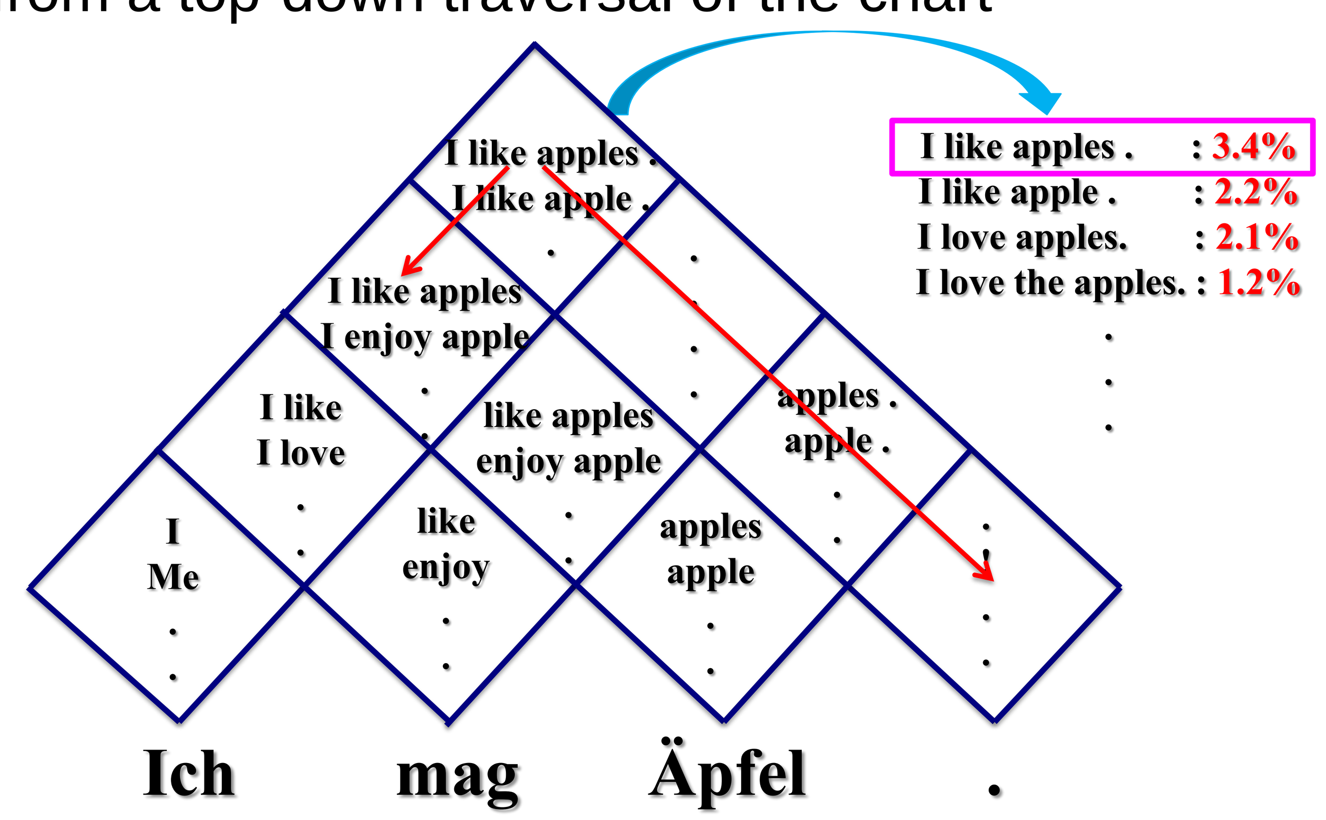
- Step 1: find translations of phrases using the translation model



- Step 2: combine translations and fill the chart bottom-up-wise using the language model



- Step 3: extract the most probable translation from a top-down traversal of the chart

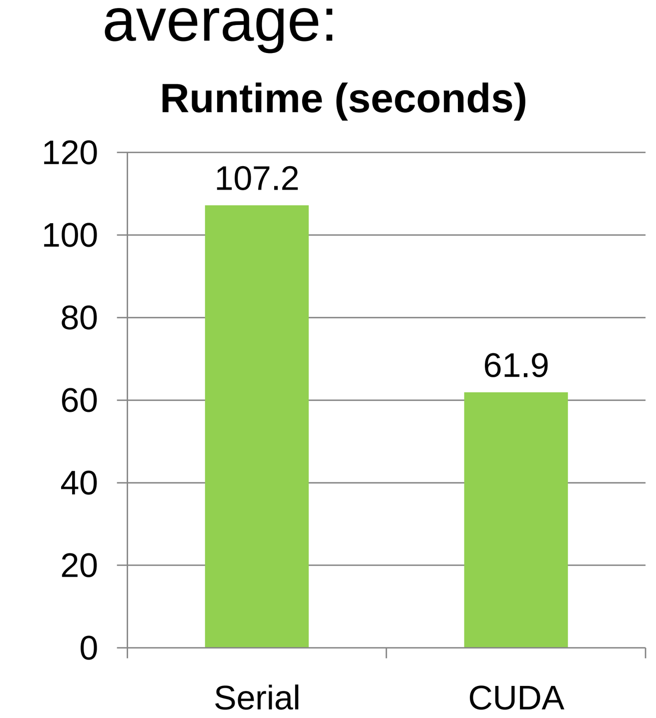


Parallelization Techniques

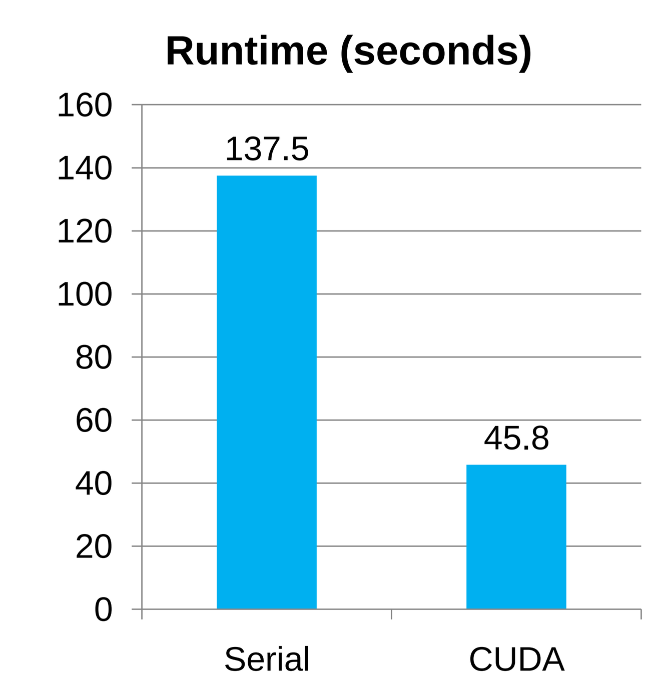
- Flattening Language Model from a map to an N-level indexed array
  - Less overhead
  - Still suffering from uncoalesced data accesses
- Several levels of parallelization
  - Up to 50 independent grids per level
  - Up to 50 split points per grid
  - Up to 1000 x 1000 combinations of translations per split point
- Using thrust CUDA libraries
  - Efficient frequently used routines like sorting and reducing by key
  - Simpler codes

Results

- 200 sentences (Spanish → English) with 28 words in average:



- GPU performs better in longer sentences (#words > 40)



Future Work

- Optimizing Language Model accesses
  - Flattening N-gram to bigram
  - Using CUDA hash map implementations
- Experiment with larger, more sophisticated machine translators (Berkeley Translator)
- Combining this work with speech recognition framework to build an oral translation framework