Inclusion of large input corpora in Statistical Machine Translation

Bipin Suresh
Stanford University
bipins@stanford.edu

ABSTRACT
In recent years, the availability of large, parallel, bilingual corpora has gone untapped by the statistical machine learning community. The crux of the problem lies in the inherent linearity of the traditional machine-translation algorithms, which impedes easy inclusion of new, large input corpora. However, it has been speculated [1] that there exists a log-linear relationship between the training corpora size and performance in machine-translation tasks. In this paper, we cast the IBM Model-1 algorithm into three formats that allow it to use large input corpora, and then verify the performance-gain claims. The models we introduce provide a scalable architecture for incorporating further corpora easily and cheaply, which we then show to translate into higher performance in machine-translation tasks.

Keywords
Natural language processing, Statistical machine translation, IBM Model-1

1. INTRODUCTION
IBM's Model-1[2] is a word-alignment machine-translation model that makes use of parallel bilingual corpora. It assumes that a source sentence *S* of length *l* is translated into a target sentence *T* of size *m* by choosing, for every position in the target sentence *j* ($1 \leq j \leq m$), a word in the source sentence *eaj* (which includes a special NULL word), according to some alignment *a*, and then translating the word *eaj* to a word *fj* with the help of a translation model *t(f, e)*.

Model-1 makes a few simplifying assumptions: first, it assumes that every possible length of the target sentence (less than some arbitrary upper-bound) has uniform probability ε; next, that all possible choices of source sentence generating words are equally likely; and finally, that the translation probability *t(f, e)* depends only on the source language word:

$$p(T|S) = \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j|e_{a_j})$$

The above equation by Brown et al[2] provides an estimate of the probability of a source sentence *S* translating into a target sentence *T*. Since Model-1 makes a further simplification assumption that each target word can be generated by a single source word (including the NULL word) only, we can work out the best alignment vector *a* as follows:

The parameters of Model-1 are typically estimated using an Estimation Maximization (EM) algorithm[3].

$$a = argmax_a \prod_{j-1}^{m} t(f_j|e_{a_j})$$

A straightforward implementation of the algorithm has a serious problem though: since the algorithm has to iterate over every alignment of every singe sentence pair, it quickly becomes computationally infeasible. A common method to overcome this shortcoming is explained in detail in [3]. Figure 1 summarizes the algorithm currently in common usage.

Model-1 has many shortcomings: (1) each target sentence word can be generated by only one source word; (2) the position of any word in the target sentence is independent of the corresponding word in the source sentence and; (3) the translation of one target word does not take into account how many other words the corresponding source word has already been translated to. Several improvements have been proposed [4] to counter these issues. In their original paper[2], Brown et al also propose higher order models – IBM Model-2, Model-3, Model-4 and Model-5 – which attempt to attack these problems systematically.

However, there is also a practical consideration that goes unnoticed: Model-1 is essentially a linear algorithm, looping over every document sentence pair; and for each sentence pair, looping over every source-target word pair. This makes incorporating new large corpora extremely expensive, both in terms of memory as well as the time taken for execution. In this paper we attempt to remodel Model-1 to make it more parallelizable.

Our contributions are as follows: We first cast the model into a matrix form, which enables us to compute scores in parallel by using fast, parallelized matrix operations. We then model the matrix as sparse matrices to reduce the memory foot-print while maintaining the gains in speed. Finally, we cast the model into the Map-Reduce framework to explore parallelization across a distributed architecture of machines. The paper is organized roughly along these contributions.

```
Input  list of sentence pairs (f, e)
Output  translation probability table t(f|e)
1:  initialize t(f|e) uniformly
2:  do
3:      set count(f|e) = 0, for all f, e
4:      set total(e) = 0, for all f
5:      for each sentence pair (f, e) do
6:          for each word f in f do
7:              set s-total(f) = 0
8:              for each word e in e do
9:                  s-total(f) += t(f|e)
10:             end for
11:         end for
12:         for each word f in f do
13:             for each word e in e do
14:                 count(f|e) += t(f|e) / s-total(f)
15:                 total(e) += t(f|e) / s-total(f)
16:             end for
17:         end for
18:     end for
19:     for all words f do
20:         for all words e do
21:             t(f|e) = count(f|e) / total(e)
22:         end for
23:     end for
24:  until convergence
```

Figure 1: IBM Model-1pseudo-code

2. MATRIX MODEL

2.1 Full matrices on GPUs

Recent developments in graphics processing units (GPU) have enabled large-scale matrix manipulation operations to be computed efficiently in parallel. Multiple parallel computing architectures have been built to support application programmers. Two prominent approaches are Nvidia's CUDA, and OpenCL.

To harness these powers of the GPU, we cast the IBM Model-1 algorithm into a series of matrix manipulation operations. First, we composed the entire corpora as a 3-dimensional matrix, with the first dimension being the source-words *f*, the second dimension being the target-words *e*, and the last dimension being the documents. A cell *docs(f, e, d)* is the count of the number of times word *f* was translated to word *e* in document *d*. Our algorithm to compute the translation probabilities between word *f* and word *e* is as follows:

Figure-2: A matrix-operations only model of IBM's Model-1

```
Input: list of sentence pairs (f, e)
Output: translation probability table t(f, e)
0.   create a 3-d matrix docs(f, e, d)
1.   uniform = 1 / max(f)
2.   t = ones(max(f), max(e)) * uniform
3.   do

4.       t = repmat(t, [1 1 nDocs])
5.       counts = docs .* t

6.       sums = sum(counts, 2)
7.       divisors = 1 ./ sums, where sums ≠ 0
8.       normcounts = bsxfun(@times, counts, divisors)

9.       interDocCounts = sum(normcounts, 3)
10       s = sum(interDocCounts, 1)
11.      norms = 1 ./ s where s ≠ 0

12.      t = bsxfun(@times, interDocCounts,  norms)

13.  until convergence
```

In the first two steps, we initialize the translation table between all words *f* and *e* with a uniform probability. Then, until convergence, we perform the following matrix operations: in step 4, we make a n-dimensional copy of our translation table, so that in step 5, we can do a element wise matrix multiplication with the *docs* matrix. Note that the element-wise matrix multiplication is inherently parallelizable, which makes step 5 very fast to compute. Next, in steps 6-8, we normalize the counts for every f. The function *bsxfun* is a special MATLAB® command which manipulates each element in a matrix with its corresponding element in a vector. Again, since each of the operations can be done in parallel, very little time is spent on the actual execution of these commands. Finally, in steps 9-11, we sum across all documents, and then re-normalize the translation probability table.

The effects of using a parallel GPU-based architecture is immediately evident in our results. In Figure 4(a), we compare the performance in time required to complete execution of 5 iterations of the algorithm for different corpora sizes. In Figure 4(b), we plot the growth of the execution times. The GPU based version is not only is faster, but also grows more slowly than the non-GPU version. For the non-GPU version, a 10-fold increase in data-size increases execution-speed by 10 times, while for the GPU version, the increase is at 4.96 times.

2.2 Sparse matrices on multicoreprocessors

The algorithm presented in Figure-2, though speedy, has some serious shortcomings. It constructs two large matrices – *docs*, and *t*, each of which has a size of max(*f*) * max(*e*) * nDocs. As the size of the corpora increases, this poses a serious memory consumption problem.

Our insight into this problem was to notice that the *docs* matrix was usually very sparse – each sentence-pair used very few words from the either vocabulary. We transformed our entire algorithm to work off sparse matrices. There is a caveat to the process – the sparse matrix support on the GPUs, at the time of the writing of this paper, is tenuous at best, and the version of MATLAB®'s GPU toolkit – Jacket® – we had installed on our machines did not support sparse matrices.

```
Input: list of sentence pairs (f, e)
Output: translation probability table t(f, e)

create a list of docs with pairs mapping
every f to every e

uniform = 1 / max(f)
t = ones(max(f), max(e)) * uniform

repeat until convergence

    execute in parallel for every doc d

        nPairs = numberOfPairs(d)

        c = zeros(nPairs)
        for j = 1:nPairs(d)
            c(j, 3) = d(j, 3) * t(d(j, 1), d(j, 2));
        end

        spCounts = sparseMatrix(c)
        sums = sum(spCounts, 2);


        listc = zeros(nPairs, 3);
        listc(:, 1) = d(:, 1);
        listc(:, 2) = d(:, 2);

        listt = zeros(nPairs, 2);
        listt (:, 1) = doci(:, 2);

        normalize every entry in spCounts by
        sums and save in listc and listt

        append list  to countList
        append listt to totalList

    end


    counts = sparseMatrix(countList)
    norms = sparseMartix(totalList);

    normalize every entry in counts by norms
    and save in the results in t
```

Figure 3: Parallel sparse matrix version of IBM Model-1

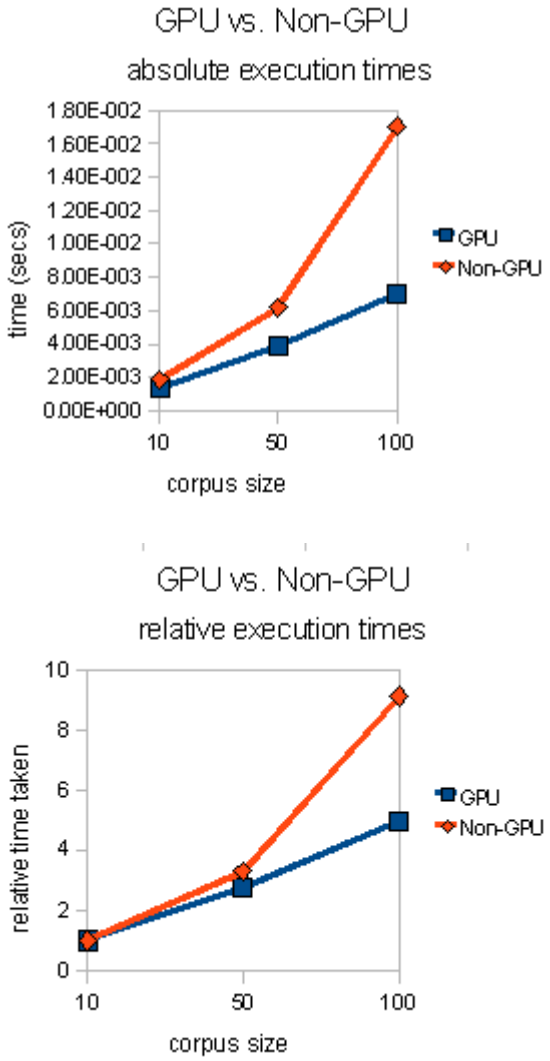

Figure 4: (a) is the absolute execution times for two models – the GPU and the non-GPU full matrix models. (b) is the same graph but on a relative scale to highlight scaling properties with respect to increases in corpus-size .

However, to make our point, we parallelized our code so that it could be run across multiple cores on the same machine. We present our algorithm in Figure 3, and our results in Figure 5.

In Figure 5 we measure the scaling properties across multiple cores. The lines represent performance for 1, 2, and 4 multicores respectively. Each data-point corresponds to the amount of time taken for execution relative to the amount of time taken to train a model of corpus size 100. As can be seen from the graph, using a 4 processing cores, an 100 fold increase of training size only increases training time by 41%.

It is also reassuring to know that the scaling properties increase with the increase of processing cores. So it is not only that the absolute processing time decreases with the number of cores, but also that the rate at which the execution time grows with increases in corpus size, decreases with the number of cores. This leads us to believe that with a substantial number of cores, we should be able to achieve a better and better scaling factors.

Finally, we also note that, since we store only the pairs that appear in documents, the memory foot-print of the sparse-matrix implementation is *much* lower than that of the full matrix implementation. This is evidenced in the corpus sizes we were able to perform experiments on – moving from 100 to 10000.

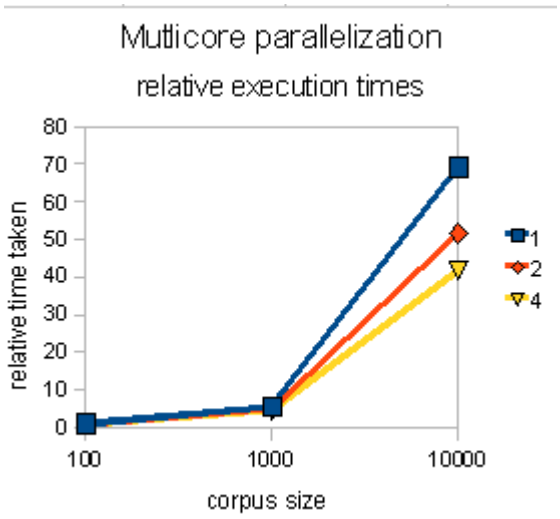

Figure 5: Relative execution times across multiple processor cores. Each data point is the time taken relative to time for the smallest corpus size(100)

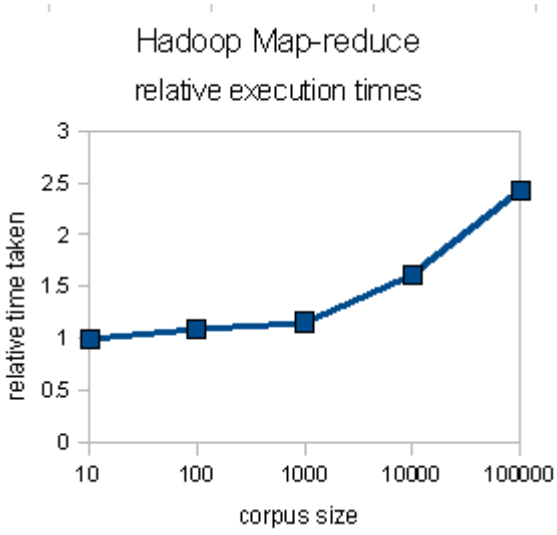

Figure 6: Relative execution times across various corpus sizes for the Hadoop Map-reduce infrastructure. A 10,000 fold increase in corpus-size increases execution time by 2.4 times only.

3. MAP-REDUCE MODEL

Recently, the Hadoop infrastructure, based on the Map-Reduce framework defined by Dean and Ghemawat[5], has appeared as an alternative for parallelizing tasks. The framework is especially useful since it shields the application programmer from the intricacies of fault-tolerance, data transfer and synchronization. The Hadoop framework allows for writing applications that process vasts amounts of data in parallel across a large cluster of nodes.

The Map-reduce framework is built on the insight that a large number of tasks have the same basic two-phase structure: a map-stage, in which a large number of records are processed to produce some local results; and a reduce stage, in which these intermediate results are aggregated to produce the final output.

Specifically, the application programmer provides two functions:

map: <k1, v1> → [<k2, v2>]

reduce: <k2, [v2]> → [<k3, v3>]

The Map-reduce infrastructure has previously been investigated[6] for the purposes of statistical machine translation. Our contributions include rewriting the entire pipe-line, including the normalization-step of IBM's Model-1 as Map-reduce tasks, instead of computing only the maximum likelihood estimates, and then computing the normalizations as a separate non-map-reduce step.

Specifically, we built two map-reduce modules, with the second one chained to the first. The first module corresponded to the counting section of the EM, while the second module corresponds to the normalization section:

Module-1: Count

map: <f, e> → [<fj, pair{ei, count(fj, ei)*t(fj, ei)}>];

reduce: <fj, [pair{ei, p}]> → [<fj+ei, sum(p_ei)>]

Module 2: Normalize

map: <f+e, p> → <e, pair{f, p}>;

reduce: <e, [pair{f, p}]> → [<fj+e, p / sum(p_fj)>]

An additional advantage is that, using this method, one can use a combiner module (which can be viewed as a local reduce module, working on the output of the local reducer), to speed things up a lot. Because of Zipf's law, this reduction is found to be significant[6].

Having constructed the model, we tested it on two different dimensions: (1) the effects of increased corpora size and; (2) the effects of increasing the number of machines on the execution times for a fixed size corpora.

Figure 6 presents the relative performance of Hadoop based on an initial corpus size of 10. The Map-reduce framework scales very well – a 10,000 fold increase in corpus size increases execution time by 2.4 times only.

Figure 7 presents a study of how Hadoop scales with number of machines available. For this experiment, we held the size of the corpus constant (100,000 documents), and measured the execution times as we added nodes. Our experiments were limited to a maximum of 15 nodes only since we were working within a school environment.

Moving from a single node system to a 15 node system decreases execution time by 44%. Gains however, seem to be already beginning to saturate.

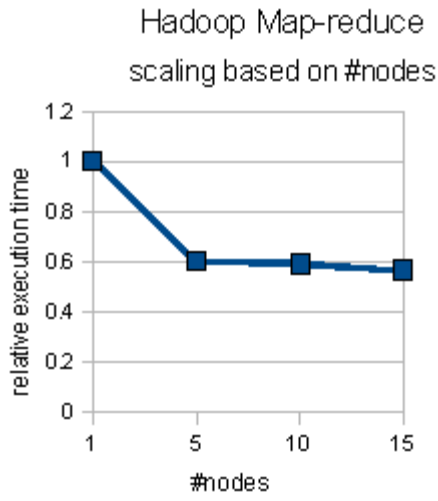

Figure 7: Scaling properties of Hadoop's Map-reduce with increased number of nodes.

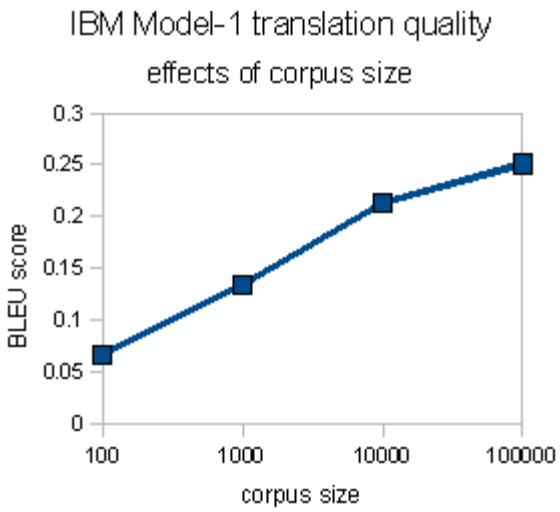

Figure 8: Effects of corpus size on translation quality.

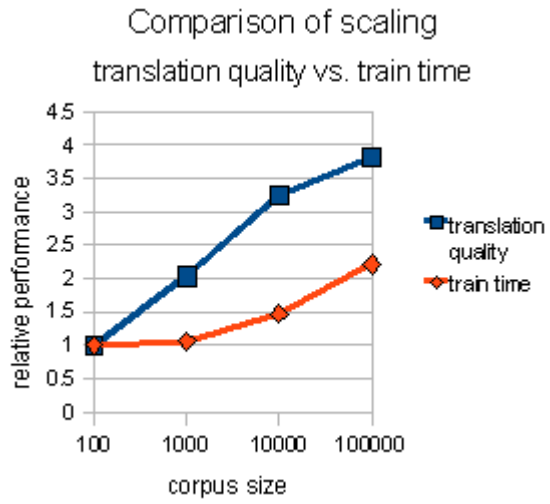

Figure 9: Relative scaling of translation quality and training time (relative to model trained on 100 documents)

4. EVALUATION

In [7] and [8], Koehn et al speculate a log-linear relationship between training corpus size and statistical machine translation quality. In [9], Brants et al. demonstrated that increasing the quantity of training data for language modeling significantly improved translation quality for Arabic-English translation systems. Armed with our infrastructure, our contribution consists of evaluating whether an increase in corpus size has an positive effect machine translation.

Our system was built on top of the Moses toolkit[10]. We used the Europarl corpus[11], extracted from the proceedings of the European parliament. For the purposes of our tests, we used the French to English bilingual corpus.

We calculate and report BLEU scores[12] as measures of translation quality. BLEU has frequently been reported as correlating well with human judgment, and remains a benchmark in the statistical machine translation community.

Figure 8 shows the effects of corpus size on BLEU scores for IBM's Model-1 algorithm. Figure 9 then overlays the scaling properties of the Hadoop architecture on the relative gains in translation quality.

The figures make it evident that an increased corpus size does indeed increase statistical machine translation quality. It is also gratifying to note that the slope of the second line (training time) in figure 9 is lesser than the slope of the first line (translation quality). This seems to suggest that the trade off of using more training data for better performance is worthwhile one.

4.1 Technical notes

For the purposes of evaluation, our experiments were conducted on the following machine architectures:

GPU: GeForce GTX 285, 1441 MHz, 1023 MB VRAM

Multicore processor: 8-core Opteron 2384 (SB X6240); 31.49 GB RAM, 10.00 GB swap

Hadoop infrastructure: 15 x Intel Core2 Duo CPU E6600 @ 2.40 GHz, 2 GB RAM, CentOS 5 x86_64. 1 Master. 15 slaves.

5. DISCUSSION AND FUTURE WORK

This paper was written on a hunch that, like in many other natural language processing tasks, the size of the training corpus played a dominant role in statistical machine translation. We built three models on state-of-the-art parallel infrastructure, and provided evidence that they scale reasonably well. We then used this infrastructure to confirm the hypothesis by training a machine translation system on a large corpus size, and determining its performance on a standard translation task.

As immediate follow-ups to this work, we see two avenues: (1) scaling the architectures by adding more hardware, and determining if the trend of increasing translation quality still holds; and (2) applying the parallelization approaches to more complex translation models

6. REFERENCES

[1] *Advances in Statistical Machine Translation: Phrases, Noun Phrases and Beyond*, Philipp Koehn

[2] *The mathematics of statistical machine translation: parameter estimation. Computational Linguistics, 19(2):263–311.* Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993a

[3] *A Statistical MT Tutorial Workbook*., Kevin Knight

[4] *Improving IBM Word-Alignment Model 1*. Robert C Moore.

[5] *MapReduce: Simplified data processing on large clusters*. In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), pages 137–150, SanFrancisco, California. Jeffrey Dean and Sanjay Ghemawat. 2004.

[6] Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce. Christopher Dyer, Aaron Cordova, Alex Mont, Jimmy Lin

[7] *Statistical Phrase-Based Translation*, Philipp Koehn, Franz Josef Och, and Daniel Marc

[8] *Advances in Statistical Machine Translation: Phrases, Noun Phrases and Beyond*, Philipp Koehn

[9] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 858–867, Prague, Czech Republic.

[10] http://www.statmt.org/moses/

[11] *Europarl: A Parallel Corpus for Statistical Machine Translation*, Philipp Koehn, MT Summit 2005

[12] *BLEU: a method for automatic evaluation of machine translation,* Papineni, K., Roukos, S., Ward, T., and Zhu, W. J. (2002).