A Parallel Multikey Quicksort Algorithm for Mining Multiword Units

Rui Pereira, Paul Crocker, Gaël Dias

Beira Interior University, Computer Science Department
Rua Marquês d'Ávila e Bolama, 6201-001, Covilhã, Portugal
{rpereira, crocker, ddg}@di.ubi.pt

Abstract

This paper describes a parallel algorithm to compute positional ngram statistics based on masks and suffix arrays. Positional ngrams are ordered sequences of words that represent continuous or discontinuous substrings of a corpus. In particular, the positional ngram model has shown successful results for the extraction of discontinuous collocations from large corpora. However, its computation is heavy. For instance, 4.299.742 positional ngrams (n=1..7) can be generated from a 100.000-word size corpus in a seven-word size window context. In comparison, only 700.000 ngrams would be computed for the classical ngram model. It is clear that huge efforts need to be made to process positional ngram statistics in reasonable time and space. For that purpose, we propose a parallel algorithm based on the concept of Parallel Sorting by Regular Sampling (PSRS) described in (Shi and Schaeffer, 1992).

Introduction

In the context of word associations, multiword units (sequences of words that co-occur more often than expected by chance) are frequently used in everyday language, usually to precisely express ideas and concepts that cannot be compressed into a single word. For instance, [Bill of Rights], [swimming pool], [as well as], [in order to], [to comply with] or [to put forward] are multiword units. As a consequence, their identification is a crucial issue for applications that require a certain degree of semantic processing (e.g. machine translation, information extraction, information retrieval or summarization). In order to identify and extract multiword units, (Dias, 2002) has proposed a statistically-based architecture called SENTA (Software for the Extraction of N-ary Textual Associations) that retrieves, from text corpora, relevant contiguous and non-contiguous sequences of words.

However, the computation of SENTA is heavy. As it is based on positional ngrams (ordered sequences of words that represent continuous or discontinuous substrings of a corpus computed in a (*2.F+1*)-word size window context), the number of generated substrings rapidly explodes and reaches astronomic figures. (Dias, 2002) shows that Δ positional ngrams can be computed in an *N*-size corpus for a (*2.F+1*)-size window context (See Equation 1).

$$\Delta = \left(N - 2.F\right) \times \left(1 + F + \sum_{k=3}^{2.F+1} \sum_{i=1}^{F} \sum_{j=1}^{F} C_{j-1}^{i-1} C_{j}^{k-i-1}\right)$$

Equation 1: Number of positional ngrams

So, for instance, 4.299.742 positional ngrams would be generated from a 100.000-word size corpus in a seven-word size window context. It is clear that huge efforts need to be made to process positional ngram statistics in reasonable time to tackle real world applications that deal with Gigabytes of data. For that purpose, (Gil and Dias, 2003) have proposed an implementation that computes positional ngrams statistics in *O(h(F) N log N)*[1] time complexity based on the *Virtual Corpus* approach introduced by (Kit and Wilks., 1998). In particular, they apply a suffix-array-like method, coupled to the multikey quicksort algorithm (Bentley and Sedgewick, 1997) to compute positional ngram frequencies. Although their C++ implementation, realized over the CETEMPúblico[2] corpus, has shown satisfactory results by taking 8.34 minutes to compute the positional ngram frequencies for a 1.092.723[3]-word corpus on an Intel Pentium III 900 MHz Personal Computer for a seven-word size window context, improvements still need to be made.

So, in this paper, we propose a parallel multikey quicksort algorithm that allows faster computation of positional ngrams frequencies taking into account the processing power of different central units spread over a network. In particular, we propose a parallel algorithm based on Parallel Sorting by Regular Sampling (PSRS) as described in (Shi and Schaeffer, 1992) that apply their method to randomly generated 32 bit integers and use the classical quicksort (Hoare, 1962) as the sequential sorting algorithm. For a variety of shared and distributed memory architectures, their results display better than half linear speedups. In the following sections, we will present our PSRS algorithm that sorts positional ngrams using the multikey quicksort as the sorting algorithm.

This article is divided into four sections: (1) we explain the basic principles of positional ngrams and the mask representation to build the *Virtual Corpus*; (2) we present the suffix-array-based data structure that allows counting occurrences of positional ngrams; (3) we explain our PSRS algorithm; (4) we present some results.

Positional Ngrams

Principles

The original idea of the positional ngram model comes from the lexicographic evidence that most lexical relations associate words separated by at most five other words (Sinclair, 1974). As a consequence, lexical relations such as collocations can be continuous or discontinuous

[1] N is the size of the corpus and F is the window size.

[2] The CETEMPúblico is a 180 million-word corpus of Portuguese. It can be obtained at http://www.ldc.upenn.edu/.

[3] This represents 46.986.831 positional ngrams.

sequences of words in a context of at most eleven words (i.e. *5* words to the left of a pivot word, *5* words to the right of the same pivot word and the pivot word itself). In general terms, a collocation can be defined as a specific[4] continuous or discontinuous sequence of words in a (*2.F+1*)-word size window context (i.e. *F* words to the left of a pivot word, *F* words to the right of the same pivot word and the pivot word itself). This situation is illustrated in Figure 1 for the collocation *Ngram Statistics* that fits in the window context.

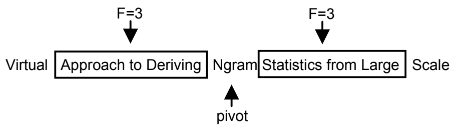


Figure 1: *2.F+1*-word size window context

Thus, as computation is involved, we need to process all possible substrings (continuous or discontinuous) that fit inside the window context and contain the pivot word. Any of these substrings is called a positional ngram. For instance, [Ngram Statistics] is a positional ngram as is the discontinuous sequence [Ngram ___ from] where the gap represented by the underline stands for any word occurring between Ngram and from (in this case, Statistics). More examples are given in Table 1.

Positional 2grams	Positional 3grams
[Ngram Statistics]	[Ngram Statistics from]
[Ngram ___ from]	[Ngram Statistics ___ Large]
[Ngram ___ ___ Large]	[Ngram ___ from Large]
[to ___ Ngram]	[to ___ Ngram ___ from]

Table 1: Possible positional ngrams

In order to compute all the positional ngrams of a corpus, we need to take into account all the words as possible pivot words. A simple way would be to shift the two-window context to the right so that each word would sequentially be processed. However, this would inevitably lead to duplications of positional ngrams. Instead, we propose a one-window context that shifts to the right along the corpus as illustrated in Figure 2. It is clear that the size of the new window should be *2.F+1*.

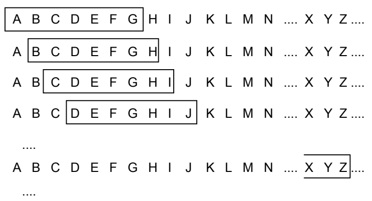

A B C D E F G H I J K L M N .... X Y Z ....
A B C D E F G H I J K L M N .... X Y Z ....
A B C D E F G H I J K L M N .... X Y Z ....
A B C D E F G H I J K L M N .... X Y Z ....
....
A B C D E F G H I J K L M N .... X Y Z ....
....

Figure 2: One-window context for *F=3*

This new representation implies new restrictions. While all combinations of words were valid positional ngrams in the two-window context, this is not true for a one-window context. Indeed, two restrictions must be observed.

Restriction 1: Any substring, in order to be valid, must contain the first word of the window context.

Restriction 2: For any continuous or discontinuous substring in the window context, by shifting the substring from left to right, excluding gaps and words on the right and inserting gaps on the left, so that there always exists a word in the central position *cpos* (Equation 2) of the window, there should be at least one shift that contains all the words of the substring in the context window.

$$cpos = \left[\frac{2.F+1}{2} \right] + 1$$

Equation 2: Central position of the window

For example, from the first case of Figure 2, the discontinuous sequence [A B _ _ E _ G] is not a positional ngram although it is a possible substring as it does not follow the second restriction. Indeed, whenever we try to align the sequence to the central position, at least one word is lost as shown in Table 2:

Possible shift	Central word	Disappearing words
[_ _ A B _ _ E]	B	G
[_ _ _ A B _ _]	A	E, G

Table 2: Shifting Substrings

In contrast, the sequence [A _ C _ E F _] is a positional ngram as the shift [_ A _ C _ E F], with C in the central position, includes all the words of the substring.

Basically, the first restriction aims at avoiding duplications and the second restriction simply guarantees that no substring that would not be computed in a two-window context is processed.

Virtual Representation

The representation of positional ngrams is an essential step towards efficient computation. For that purpose, we propose a reference representation rather than an explicit structure of each positional ngram. The idea is to adapt the *suffix* representation (Manber and Myers, 1990) to the positional ngram case.

Following the suffix representation, any continuous corpus substring is virtually represented by a single position of the corpus as illustrated in Figure 3. In fact, the substring is the sequence of words that goes from the word referred by the position till the end of the corpus.

Unfortunately, the suffix representation can not directly be extended to the specific case of positional ngrams. One main reason aims at this situation: a positional ngram may represent a discontinuous sequence of words. In order to overcome this situation, we propose a representation of positional ngrams based on **masks**.

[4] As specific, we intend a sequence that fits the definition of collocation given by (Dias, 2002): “A collocation is a recurrent sequence of words that co-occur together more than expected by chance in a given domain”.

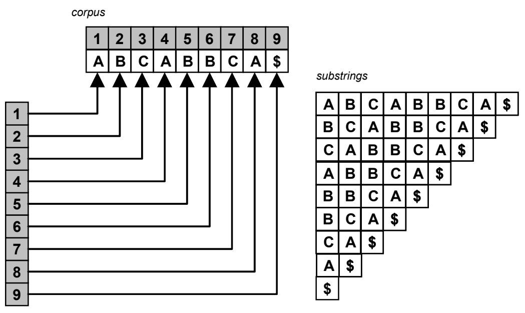


Figure 3: Suffix Representation[5]

As we saw in the previous section, the computation of all the positional ngrams is a repetitive process. For each word in the corpus, there exists an algorithmic pattern that identifies all the possible positional ngrams in a *2.F+1*-word size window context. So, what we need is a way to represent this pattern in an elegant and efficient way. One way is to use a set of masks that identify all the valid sequences of words in a given window context. Thus, each mask is nothing more than a sequence of 1 and 0 (where 1 stands for a word and 0 for a gap) that represents a specific positional ngram in the window context. An example is illustrated in Figure 4.

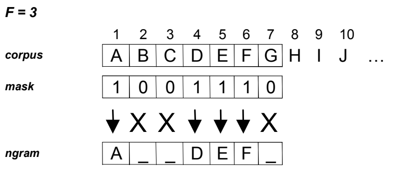


Figure 4: Masks

Computing all the masks is an easy and quick process. In our implementation, the generation of masks is done recursively and is negligible in terms of space and time. In Table 3, we give the number of masks *h(F)* for different values of *F*.

F	h(F)
1	4
2	11
3	43
4	171
5	683

Table 3: Number of masks

In order to identify each mask and to prepare the reference representation of positional ngrams, an array of masks is built as in Figure 5.

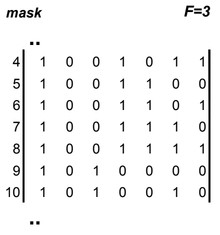


Figure 5: Masks Array

[5] The $ symbol stands for the end of the corpus.

From these structures, the virtual representation of any positional ngram is straightforward. Indeed, any positional ngram can be identified by a position in the corpus and a given mask. Taking into account that a corpus is a set of documents, any positional ngram can be represented by the tuple $\{\{id_{doc}, pos_{doc}\}, id_{mask}\}$ where id_{doc} stands for the document id of the corpus, pos_{doc} for a given position in the document and id_{mask} for a specific mask. An example is illustrated in Figure 6.

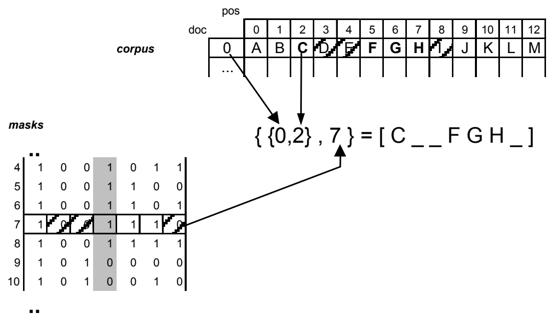


Figure 6: Virtual Representation

As we will see in the following section, this reference representation will allow us to follow the *Virtual Corpus* approach introduced by (Kit and Wilks, 1998) to compute ngram frequencies.

Computing Frequencies

With the *Virtual Corpus* approach, counting continuous substrings can easily and efficiently be achieved. After sorting the suffix-array data structure presented in Figure 3, the count of an ngram consisting of any n words in the corpus is simply the count of the number of adjacent indices that take the n words as prefix. We illustrate the *Virtual Corpus* approach in Figure 7.

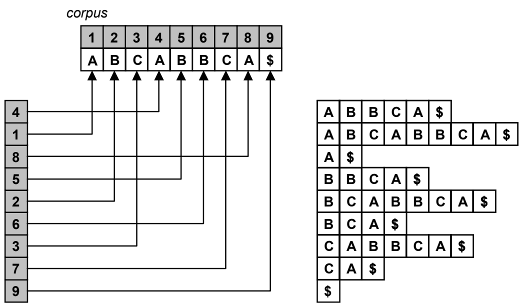


Figure 7: *Virtual Corpus* Approach

Counting positional ngrams can be computed exactly in the same way. The suffix-array structure is sorted using lexicographic ordering for each mask in the array of masks.

After sorting, the count of a positional ngram in the corpus is simply the count of adjacent indices that stand for the same sequence. We illustrate the *Virtual Corpus* approach for positional ngrams in Figure 8.

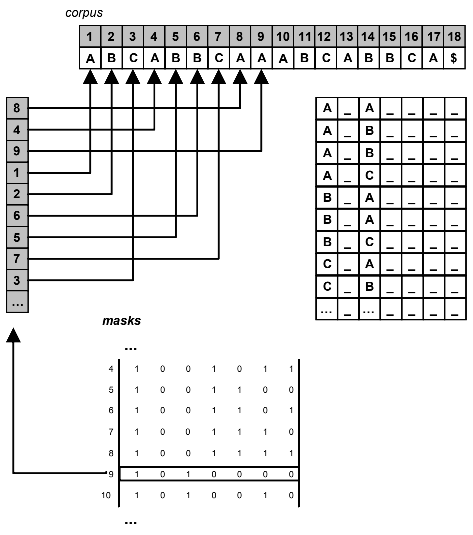


Figure 8: *Virtual Corpus* for positional ngrams

The efficiency of the counting mainly resides in the use of an adapted sort algorithm. For the specific case of positional ngrams, we have chosen to implement the multikey quicksort algorithm (Bentley and Sedgewick, 1997) that can be seen as a mixture of the Ternary-Split Quicksort (Bentley and McIlroy, 1993) and the MSD[6] radixsort (Anderson and Nilsson, 1998). Different reasons have lead to use the Multikey Quicksort algorithm. First, it performs independently from the vocabulary size. Second, it shows $O(N \log N)$ time complexity. Third, (Anderson and Nilsson, 1998) show that it performs better than the MSD radixsort and proves comparable results to their newly introduced forward radixsort.

The algorithm processes as follows: (1) the array of string is partitioned into three parts based on the first symbol of each string. In order to process the split, a pivot element is chosen just as in the classical quicksort giving rise to: one part with elements smaller than the pivot, one part with elements equal to the pivot and one part with elements larger than the pivot; (2) the smaller and the larger parts are recursively processed in exactly the same manner as the whole array; (3) the equal part is also sorted recursively but with partitioning starting from the second symbol of each string; (4) the process goes on recursively: each time an equal part is being processed, the considered position in each string is moved forward by one symbol.

As we already said, the efficiency of the counting mainly resides in the use of an adapted sort algorithm. Moreover, the sorting phase is the most time consuming of our global architecture that extracts collocations. So, we define a Parallel Sorting by Regular Sampling Multikey Quicksort algorithm in order to fasten this stage.

PSRS Multikey Quicksort Algorithm

The Parallel Algorithm we propose is based on Parallel Sorting by Regular Sampling (PSRS) as described in (Shi and Schaeffer, 1992). In particular, they apply their method to randomly generated 32 bit integers and use the classical quicksort (Hoare, 1962) as the sequential sorting algorithm. For a variety of shared and distributed memory architectures, their results display better than half linear speedups. Our PSRS algorithm sorts positional ngrams using the multikey quicksort as the sorting algorithm. Our algorithm can be divided into three distinct phases: a parallel multikey quicksort phase; reorganization by global pivots phase; a merge sort phase.

The parallel multikey quicksort phase consists in partitioning the original data (i.e. the corpus) into p contiguous lists, one per node[7], and uses the multikey quicksort algorithm to sort each local contiguous list. In fact, in step 1, each node reads the entire data file to one suffix array structure into his local memory as in figure 9[8].

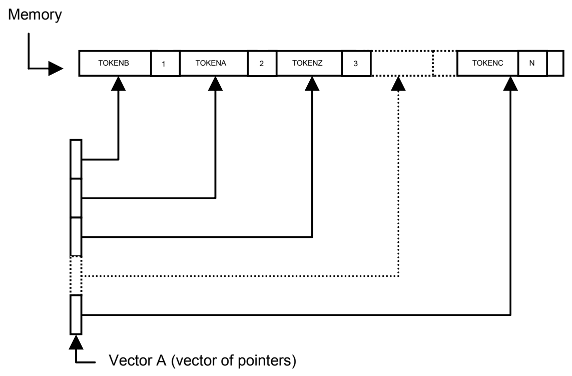
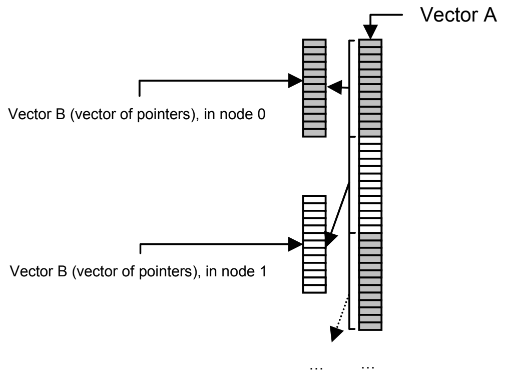


Figure 9: *Virtual Corpus* in memory

In step 2, each of the p nodes determines a contiguous list of size $w = N/p$ from the original data where N is the size of the corpus. In fact, each node makes a copy of size w of the suffix-array. We shall call this vector B which will be sorted. This situation can be seen in Figure 10.



Vector A
Vector B (vector of pointers), in node 0
Vector B (vector of pointers), in node 1

Figure 10: Suffix-array division

In step 3, each node sorts its local contiguous list using the multikey quicksort algorithm which implements the "*median of three modification method to improve the average performance of the algorithm while making the worst case unlikely to occur in practice*" (Lan and Mohamed, 1992). After this phase, all local contiguous

[6] MSD stands for Most Significant Digit.

[7] A node represents a processing unit.

[8] The reader will note that after each token we insert its position in the file.

lists are sorted following a given mask as shown in Figure 11.

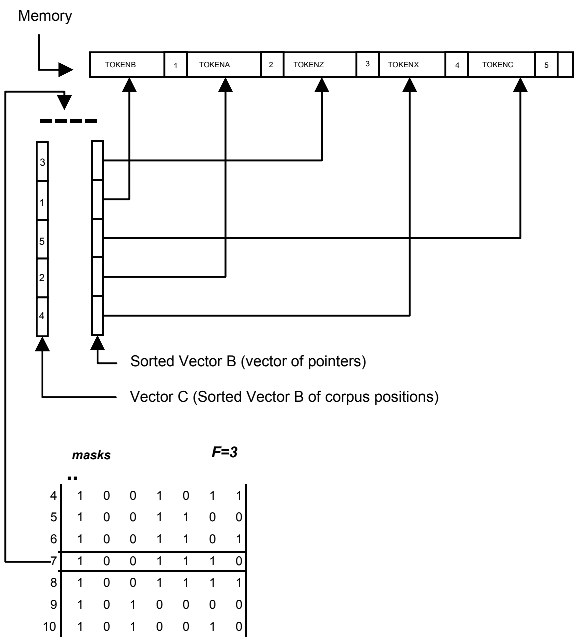


Figure 11: Sorted Suffix-array

The Vector C is constructed as, when we wish to communicate information with another node, we cannot exchange pointer information as they may not be the same on different nodes, but we can exchange information relative to the original positions of the tokens.

The reorganization by global pivots phase consists in (1) determining the *(p-1)* local pivots on each node, (2) determining the global pivots from the *p*(p-1)* local pivots and (3) reorganizing the local list in terms of the global pivots. The idea is to join and sort all local sorted contiguous lists in a parallel way with good load balancing. For that purpose, we use a Regular Sampling approach suggested by (Shi and Schaeffer, 1992) described as follows.

Each node determines *(p-1) local* pivots from its sorted list. The node 0 gathers the *p*(p-1)* local pivots (first inter-node communication). The node 0 calculates the global pivots from the list of all local pivots and broadcasts the global pivots to all nodes (second inter-node communication). This situation is shown in Figure 12.

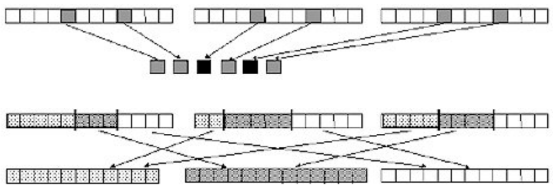


Figure 12: Reorganization for three nodes

Finally, the merge sort phase consists of creating on each node one final locally sorted list using a merge sort list. For that purpose, each node splits its sorted list into *p* sorted sub-lists based on the values of the global pivots.

Then, each node keeps one sorted sub-list and communicates the others to the appropriate nodes. Due to the fact that the sub-lists are of unequal size, we must first communicate the number of data items each node must send/receive from each other node before performing the actual data transfers (note that only vectors of integers will be passed i.e. integers representing the original token positions).

In order to reduce the communication costs and network traffic we use a customized collective communication (ALL-to-ALL) based on phases in which only pairs of processors communicate as shown in figure 13.

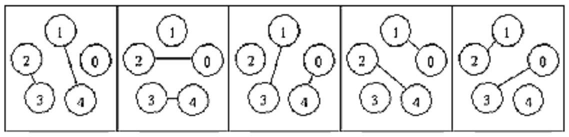


Figure 13: Collective communication

This step is then followed by a merge of the received vector[9]. In fact, each node merges the *p* sorted sub-lists into one local sorted list using the merge sort algorithm and then calculates the frequency of each ngram. Then, each node communicates the position of first instance of each n-grams plus its frequency to node zero where the matrix of all ngrams frequency is constructed.

The global architecture of our PSRS can be summarized as the following steps:

1. The original data file (size *N*) is copied to all *p* processor nodes of the cluster.
2. Each of the *p* nodes reads the data file to its local memory and builds the suffix-array.
3. Each of p nodes determines a contiguous list of size *w = N/p* from the original data.
4. Each node creates the valid masks.
5. For each valid mask:
    a. Each node sorts the contiguous list using the multikey quicksort algorithm following the current mask.
    b. Each node determines *(p-1)* local pivots from its sorted list.
    c. The node 0 gathers all local pivots (first inter-node communication).
    d. The node 0 calculates the global pivots from the list of all local pivots.
    e. Node 0 broadcasts the global pivots to all nodes (second inter-node communication).
    f. Each node splits its sorted contiguous list into *p* sorted sub-lists based on the values of the global pivots.

[9] In particular, we can overlap communication and merge routines using the non-blocking send/receive routines of the MPI standard.

g. Each node keeps one sorted sub-list and passes the others to the appropriate nodes (third inter-node communication and the most expensive).
h. Each node merges the *p* sorted sub-lists into one local sorted list using the merge sort algorithm.
i. Each node communicates the position of first instance of each ngram plus its frequency to node zero.

6. The node 0 constructs the matrix of all ngrams frequency.

Results

The algorithm has been implemented using the single program multiple data (SPMD) programming methodology using the ANSI C programming together with the freely available MPICH implementation of the Message Passing Interface (MPI) library. A network of up to ten identical workstations was used. Each workstation consists of a single Pentium IV 2.4 GHZ processor with 512 Mb of RAM running the Windows XP operating system and they are connected via a 100 Mb Ethernet network.

We have conducted a series of experiments for various different sized sub-corpora of the CETEMPúblico Portuguese corpus using a seven-word size window context, for which we present two examples. The details of the two chosen test cases are given in Table 3.

Corpus	Case A	Case B
Size in Mb	6,829	20,477
# of words	1.000.000	3.000.000
#of ngrams	42.999.742	128.999.742

Table 3: Sub-Corpora Test Cases

The experimental results based on the two test cases are then presented in Tables 4 and 5.

Processors	Time/Minutes	Speedup	Efficiency
1	3,18		
2	2,11	1,51	0,75
3	1,79	1,78	0,59
4	1,46	2,17	0,54
5	1,32	2,42	0,48
6	1,18	2,69	0,45
7	1,15	2,77	0,40
8	1,03	3,10	0,39
9	1,00	3,16	0,35
10	0,98	3,24	0,32

Table 4: Results for the Sub-Corpora, Case A

In (Gil and Dias, 2002) the sequential algorithm took 8.34 minutes to sort and calculate the ngram frequencies of a 1.092.723-word corpus on an Intel Pentium III 900 MHz. Our result for a single processor for a 1.000.000 word corpus is 3.18 minutes which is to be expected as the performance of our single processor is approximately 2-3 times faster that the one used in (Gil and Dias, 2002). For Case B, a corpora three times that of case A, the time of 11,71 minutes is obtained for a single processor, in other words bearing out the *O(h(F) N log N)* complexity of the sequential algorithm.

Processors	Time/Minutes	Speedup	Efficiency
1	11,71		
2	7,64	1,53	0,77
3	5,79	2,02	0,67
4	4,79	2,44	0,61
5	4,22	2,78	0,56
6	3,91	3,00	0,50
7	3,68	3,19	0,46
8	3,50	3,34	0,42
9	3,34	3,50	0,39
10	3,27	3,58	0,36

Table 3: Results for the Sub-Corpora, Case B

The performance of the parallel algorithm is similar in both cases, slightly better in the larger case B as would be expected. Initially with a small number of processors reasonable speedups and efficiency are obtained but this parallel performance deteriorates with the augmenting number of processors due to high levels of communications. However the overall time taken for the sort is still a monotonically decreasing function when using up to 10 processors (See Figure 14 and Figure 15).

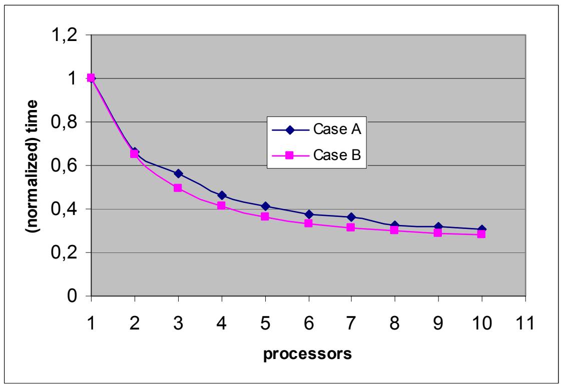


Figure 14: Running Time *vs* number of Processors

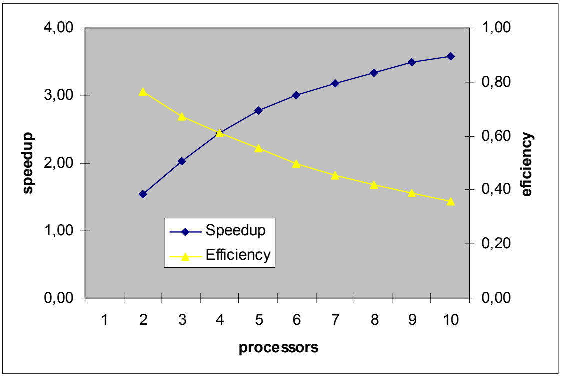


Figure 15: Speedup and Efficiency for Case B

Conclusions and Future Work

Mining multiword units in real life situation means the ability to deal with Gigabytes of data in a useful time frame necessitating the use of high performance computing architectures. Low cost network-based distributed and parallel architectures are a useful alternative for high cost proprietary machines (Kacsuk and Vajda, 1999) and offer a flexible, scaleable and readily available solution. In this paper, we have proposed a PSRS multikey quicksort algorithm to compute positional ngram frequencies which will be integrated to the SENTA system developed by (Dias, 2002). However, speedups can surely be achieved adapting to our current parallel architecture the work of (Yamamoto and Church, 2000) to positional ngrams that propose to compute categories of contiguous substrings instead of the substrings themselves.

References

Alexandre Gil and Gaël Dias. 2002. *Using Masks, Suffix Array-based Data Structures and Multidimensional Arrays to Compute Positional Ngram Statistics from Corpora*. Workshop on Multiword Expressions of the 41st ACL meeting. 7-12 July. Sapporo. Japan. http://www.di.ubi.pt/~ddg/publications/acl2003-1.pdf

Arne Anderson and Stefan Nilsson. 1998. *Implementing Radixsort*. ACM Journal of Experimental Algorithmics, Vol. 3. http://citeseer.nj.nec.com/79696.html

Chunyu Kit and Yorick Wilks. 1998. *The Virtual Approach to Deriving Ngram Statistics from Large Scale Corpora*. International Conference on Chinese Information Processing Conference, Beijing, China, 223-229. http://citeseer.nj.nec.com/kit98virtual.html.

Gaël Dias 2002. *Extraction Automatique d'Associations Lexicales à partir de Corpora*. PhD Thesis. New University of Lisbon (Portugal) and University of Orléans (France). http://www.di.ubi.pt/~ddg/publications/thesis.pdf.gz

Hoare, C. A. R. 1962. *Quicksort*. Computer Journal 5, 1(April 1962), 10-15.

John Sinclair. 1974. *English Lexical Collocations: A study in computational linguistics*. Singapore, reprinted as chapter 2 of Foley, J. A. (ed). 1996, John Sinclair on *Lexis and Lexicography*, Uni Press.

Jon Bentley and Robert Sedgewick. 1997. Fast Algorithms for Sorting and Searching Strings. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orléans. http://citeseer.nj.nec.com/bentley97fast.html.

Jon Bentley and Douglas McIlroy. 1993. Engineering a sort function. *Software - Practice and Experience*, 23(11):1249-1265.

Kacsuk, P. and Vajda, F. 1999. *Network-based Distributed Computing*, Prospective Reports on ICST Research in Europe. http://www.ercim.org/publication/prosp/

Lan,Y. and Mohamed, M.A. 1992. *Parallel Quicksort in Hypercubes* Symposium on Applied Computing, Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's, Kansas City, United States, 1992, pp740-746.

Mikio Yamamoto and Kenneth Church. 2000. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a corpus. *Association for Computational Linguistics*, 27(1):1-30. http://www.research.att.com/~kwc/CL_suffix_array.pdf

Message Passing Interface Forum. MPI: A message-Passing Interface Standard. Int. Journal of Supercomputer Applications, 8(3/4):165-414, 1994

MPICH MPI Software http://www-unix.mcs.anl.gov/mpi/mpich/

Shi, H. and Schaeffer, J. 1992. *Parallel Sorting by Regular Sampling,* Journal of Parallel and Distributed Computing, Vol 14, Nº 4, pp361-372.

Udi Manber and Gene Myers. 1990. *Suffix-arrays: A new method for on-line string searches*. First Annual ACM-SIAM Symposium on Discrete Algorithms. 319-327. http://www.cs.arizona.edu/people/udi/suffix.ps