PARSING IN PARALLEL

ELIEZER L. LOZINSKII1 and SERGEI NIRENBERG2

1Institute of Mathematics and Computer Science, The Hebrew University, Jerusalem 91904, Israel
2Department of Computer Science, Colgate University, Hamilton, NY 13346, U.S.A.

(Received 8 March 1985; revision received 10 October 1985)

Abstract—Application of parallel methods have become recently a general trend in information processing. We present a technique of parallel parsing of natural language which is based on distribution of elements of a sentence among a number of independent processors so that processing is carried out in parallel in multiple locations within a sentence. All the processors are identical, interchangeable and run the same software. An estimate of the time efficiency of this algorithm is given. The algorithm is suitable for parsing of ungrammatical and illformed sentences.

Parsing natural language  Parallel processing  Computational linguistics

1. INTRODUCTION

Recent developments in computer architecture and, particularly, breakthroughs in VLSI technology gave a strong impetus to investigations of parallel methods in data processing. This evolution has involved the field of natural language processing. A number of studies of this problem [1–4] employ a kind of parallelism. This parallelism is task-oriented in that different smart processors working in parallel perform different specialized functions, and the input of a processor is usually the output its predecessor produced.

We suggest a different approach which makes possible a higher degree of parallelism and reliability. We suggest that the input string be split into word-length substrings and that a relatively simple processor be assigned to each input string element.

With this approach the number of processors involved grows, which leads to an increase in parallelism. All processors are identical, operate jointly but independently and asynchronously, and run the same software. Should a processor "die", a neighbour will complete the work it abandoned. In the extreme case when all but one processor "die", the remaining processor will be able to accomplish the entire job. This is the degenerate, sequential, case. Thus, a high degree of reliability is inherent in the system.

The method we suggest is especially advantageous in processing languages that possess a certain property which we call the property of locality. Its essence can be described informally as follows (see Fig. 1). Assume a certain natural language (NL) processor that obtains an input string and produces an output structure. If some output elements, \( v_i, v_j \), depend on input substrings, \( w_i, w_j \), respectively, then both \( v_i \) and \( v_j \) can be produced in parallel by two processors, \( P_1 \) and \( P_2 \). Thus, we say that a language has the property of locality with respect to an NL processor if the latter produces output elements on the basis of relatively small portions of its input. It will be shown that a nontrivial subset of English has the property of locality with respect to syntactic parsing.

![Fig. 1. A scheme of parallel natural language processing based on locality.](image-url)
We will illustrate our approach to parallel processing of natural language by describing PARPAR, a system of parallel syntactic parsing of a subset of English.

2. THE PARPAR SYSTEM

PARPAR consists of three stages carried out by three different routines. The common ground for these routines is the multiprocessor environment they work in. At each stage the elements of an input string are distributed among the corresponding number of processors arranged in a linear structure connected by a simple network (cf. Fig. 2) that allows local communication between neighbouring processors as well as broadcasting global signals to all processors in the network. The processors are identical and run identical software. A specialized grammar is used by each of the three stages. A syntactic parse tree for an input English sentence is produced through a joint effort of all the system processors.

We shall illustrate PARPAR operation with an example parse (Fig. 3). The first stage of PARPAR obtains an input sentence and produces a set of grammatical strings of syntactic category names to which the words of the input sentence belong (together with the values of applicable feature dimensions, such as number, tense, etc.). The second stage obtains the output of the first stage and generates a string in which the left boundaries of sentence constituents (such as noun phrase (NP), verb phrase (VP), etc.) have been determined. At the third stage the corresponding right boundaries are determined, and a parse tree (Fig. 4) is completed. Note that during the second and the third stages some of the candidate strings from the previous stages are ruled out due to ungrammaticality.

2.1 Friendly-Neighbours

The first stage of PARPAR has been dubbed "Friendly Neighbours", or FN. Let the input to the first stage be a sentence \( n \) words long (counting an "end-of-sentence" delimiter). Then \( n \) processors are assigned to the input string, one for each word. The processors are numbered from left to right. Each processor is supplied with a special table grammar: the "friendly-neighbours" (or FN-) table (a fragment of the FN-table used in this implementation is shown in Fig. 5).

The FN-table specifies, for a processor in position \( i \), the list of syntactic categories which are compatible (in the given language) with the pair of categories of the words assigned to the processors which are located at positions \( i - 1 \) and \( i + 1 \), i.e. the neighbours of the current processor.

The table in Fig. 5 reflects a subset of English. It stipulates, for example, that the slot between a determiner on the left and a noun on the right may be occupied by an adjective or a noun; that only a noun may appear between an adjective and a verb; that if the left neighbour of a processor \( P \) contains an adverb and the contents of the right neighbour are unknown, then \( P \) can contain a verb, an adjective or an adverb, and so on. The right column of the FN-table contains the tests which are to be performed and the conditions that must be satisfied by each triad of successive words in a grammatically correct sentence. These tests are similar in design to those used on the arcs of an augmented transition network grammar.

Each processor at this stage runs the routine FN. On obtaining an input word assigned to the processor, FN fetches from a dictionary the set \( W \) of all the syntactic categories (roughly, "parts of speech") that this word can potentially belong to. The morphological stage of analysis has not been a special concern of ours, since there are several well-known programs which carry out

![Local bus](image)

![Global bus](image)

Fig. 2. A network of processors used in PARPAR.
The little boy whom John met went to the river

![Diagram of parsing stages]

Fig. 3. Stages of PARPAR parsing.

![Output tree diagram]

Fig. 4. A PARPAR output tree.
The table is used as follows:
given the syntactic category of the word to the left in the string (rows) and the word
to the right (columns), the word is current processor can, according to the grammar
used (cf. Fig. 7) belong to the specified categories. The entries in the table consist of
category names and tests that must evaluate to “t” for the category to be valid. # is
the beginning-of-sentence sign.
Example:
If to the left of current word is beginning of sentence and to the right, a noun, the words
category can be an adjective, a preposition, a determiner or a noun. A test must be
performed to exclude strings like “a tables” (nu-agree Det N) or “masses consumption”
(sing N).

Fig. 5. A fragment of the table grammar for Friendly Neighbours.

this kind of processing quite successfully [5]. In this project we made use of the SMORPHPARSE
program [6].

Next, FN examines its neighbours in order to choose from the set W such a syntactic reading
that is compatible (according to the FN-table) with those contained in the neighbouring processors
(thus, showing its “friendliness” to them). For this purpose FN intersects W with the appropriate
list from the FN-table and counts the cardinality of the resulting set, RES. If RES contains a single
reading (this is the best case), then the latter is chosen as the value of the syntactic category for
the input word. If RES contains more than one reading then one of them is chosen according to
a special priority scheme based on probability calculations. If the intersection is empty then an
alarm signal is issued to the neighbours requiring changes in their records.

Suppose that when FN checks the contents of the neighbours of a processor P, neither of them
has yet declared its reading (remember that the processors operate asynchronously). In this case,
if the set W of P contains a single reading, then it is chosen; otherwise, should the program now
choose its own candidate from W this may lead to an eventual incompatibility with its neighbours
(an “unfriendly” behaviour). So, FN might postpone its decision. To avoid the deadlock of infinite
postponement the program “tosses a coin” (the relative probability of outcomes can be controlled).
If the outcome is “heads” a processor’s program chooses a reading on its own, otherwise it waits
to check the contents of its neighbours again.

The flow chart of the Friendly Neighbours routine is presented in Fig. 6.

Each string of compatible syntactic readings produced by FN is transferred to the second stage
of parsing, carried out by a different set of processors, while the first stage resumes its operation
until no more well-formed strings can be found.

After the first stage the two lowest levels of the final parse tree (cf. Fig. 3) are already at our
disposal: the leaves (the input words) and the preterminal level symbols (syntactic category names
for these words). Such by-products of the first stage as the information on modality (mood, tense,
etc.) are detected and retained to be incorporated into the result at a later stage.

2.2 Detection of sentence constituents

The subset of English which we consider in this paper is defined by the context-free grammar
presented in Fig. 7. It deals with the following sentence constituents: noun phrases (NP), verb
phrases (VP), prepositional phrases (PP), virtual noun phrases (NP): those represented by relative
pronouns, principal clause (S), and relative clause (S’). The syntactic word categories are restricted
in this sublanguage to determiners (DET), nouns (N), verbs (V), adjectives (ADJ), adverbs (ADV),
and prepositions (PREP).

This grammar determines the contents of the special grammatical table (CB-table) for Stage 2
of PARPAR. This tabular grammar contains all possible triads of syntactic categories (located, say,
in positions \( i - 1, \ i, \ i + 1 \) in the string), and for each triad specifies (in column "CB") what constituent (if any) starts in the middle position \( i \). A fragment of this CB-table (for our sub-language) is shown in Fig. 6. There are four types of triads in the CB-table:

1. An illegal triad: “DET, DET, V”.
2. A legal triad such that no constituent starts (opens) in the middle position: “ADJ, ADJ, ADJ”.
3. A triad that unambiguously determines the constituent(s) starting in position \( i \). For example, a determiner at the beginning of a sentence followed by an adjective starts both the sentence and a noun phrase (see row 3, Fig. 8), a preposition that comes after a verb and is followed by a determiner starts a prepositional phrase (row 7).
4. A triad which does not supply sufficient information for an unambiguous decision about constituent boundaries. This means that different constituents may start in the middle position and that the disambiguation depends on the knowledge of context beyond the triad.

The position at which a constituent starts (“opens”), called a left constituent boundary (LCB), has a special role in the processing. A language is called an \textbf{LCB-local} language if its CB-table contains no rows with ambiguous left boundaries. This implies the following

\textit{Proposition 1}. If a sentence belongs to an LCB-local language then the left boundaries of all the sentence constituents can be detected by observing only the triads for successive input string elements.

\[
\begin{align*}
S & \rightarrow PP^* \ NP \ VP | VP \\
S' & \rightarrow NP^* [NP] \ VP \\
NP & \rightarrow [DET] \ ADJ^* \ N^* \ N \ [S'] | S' \\
VP & \rightarrow ADV^* \ V \ [NP] \ [NP] \ PP^* \\
PP & \rightarrow PREP \ [PREP] \ (NP) \ | S' \\
NP' & \rightarrow \text{who} | \text{which} \ldots \\
DET & \rightarrow \text{a} | \text{the} \ldots \\
N & \rightarrow \text{book} | \text{John} \ldots \\
V & \rightarrow \text{give} | \text{meet} \ldots \\
ADJ & \rightarrow \text{big} | \text{red} \ldots \\
ADV & \rightarrow \text{quickly} | \text{seldom} \ldots \\
PREP & \rightarrow \text{to} | \text{out} | \text{of} \ldots 
\end{align*}
\]

Fig. 7. The subset of English used in current implementation of PARPAR.
<table>
<thead>
<tr>
<th>Row No.</th>
<th>Position</th>
<th>CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DET</td>
<td>DET V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADJ</td>
<td>ADJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td>DET</td>
</tr>
<tr>
<td></td>
<td>NP'</td>
<td>ADJ</td>
</tr>
<tr>
<td>4</td>
<td>NP</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>NP</td>
<td>V</td>
</tr>
<tr>
<td>6</td>
<td>V</td>
<td>PREP</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8. A fragment of the Constituent Boundary table grammar used at stage 2 of PARPAR.

Hence, using n processors for a sentence of n words, the task of detecting left boundaries of all its constituents can be accomplished in a constant time, because to do this, it is sufficient for every processor to consult only its two immediate neighbours.

Let m be the number of word syntactic categories recognized by the grammar. Then the number of all different triads in CB-table, M, is \( m^3 \). Since m is usually small (it hardly ever exceeds 20 if one does not use syntactic features like, for instance, verb transitivity as category-defining property; it follows that \( M < 10^6 \)), LCB-locality of a language can be proven by an exhaustive scan of all legal triads in a grammar. This is exactly the way in which we prove the following

**Proposition 2.** The subset of English defined by the grammar of Fig. 7 is LCB-local.

If a certain extension of a language violates the latter’s LCB-locality then the parallel detection of LCB can be provided either by extracting the “violators” into a special CB-subtable to be treated in an ad hoc manner, or (if the quantity of these violators is considerable) by increasing the range of neighbours to be accessed by a processor, say, from one on every side to two, thus exchanging triads for 5-tuples.

At this, second, stage of PARPAR each processor is assigned an element of the string produced by FN, and, after accessing the contents of its neighbours, obtains a triad of syntactic readings. Next, the processor looks into the CB-table, and, for each sentence constituent that can start in its position produces a quadruple of the form

\[ \langle C, x, y, f \rangle, \]

where C stands for the constituent name; x and y stand for the string positions of its left and right boundary, respectively, such that \( x = i \), and y is as yet unknown; f is a status flag.

If several constituents start in the same position (e.g. in Fig. 3, S and NP start in position 1) then the corresponding processor produces a string of quadruples which we call a chunk. The last quadruple of a chunk has \( f = 1 \) (the “end-of-chunk” flag), while the rest have \( f = 0 \). The last quadruple of a sentence has \( f = 2 \) (the “end-of-sentence” flag). For instance, the first processor in Fig. 3 produces the chunk \( \langle \langle S, 1, y, 0 \rangle \langle NP, 1, y, 1 \rangle \rangle \); the processor #5 detects a noun phrase (“John”) starting and ending in the same position, and therefore produces \( \langle NP, 5, 5, 1 \rangle \).

The output of PARPAR’s Stage 2 is, therefore, a string of chunks (of quadruples).

### 2.3 The pyramid

A sentence constituent is said to be open if its beginning position (the left boundary) is known, but its end position (the right boundary) is not detected yet. If both boundaries of a constituent are known, it is said to be closed.

A string of quadruples produced at Stage 2 represents all the sentence constituents, but only a few of them are closed (exactly those that consist of one word). In order to complete the parsing of a sentence and build the parse tree it is necessary to close all the open constituents and determine the way they nest. This task is carried out during the third stage PARPAR by a common effort of
a set of processors operating in parallel and performing an algorithm called PYRAMID (Fig. 9 suggests the name). At any given time every processor is in a state which can be one of the following:

1. **active**—the processor is involved in an activity;
2. **passive**—the processor stores its last results, provides its left neighbour with an exclusive access to its memory, and resigns ceasing its processing of the given sentence;
3. **neutral**—the processor executes a "coupling" protocol (see below);
4. **final**—the processor completes the parsing and outputs the results.

When a processor P is active it obtains a chunk (a substring of quadruples) and attempts to close open constituents in it. When P completes all it can achieve with a particular input chunk (in all probability, failing to close some open constituents), then it begins a coupling protocol trying to cooperate with its neighbours in advancing the parsing process. For this purpose P invokes procedure STATE-CHOICE (see Fig. 10) which allows it to couple with one of its neighbours provided that the latter reciprocates.

If this neighbour happens to be the left one, P next assumes the passive state, and activates procedure RESIGN (see below) which stores its chunk (partially processed), and yields to its left neighbour the access to its memory. If P couples with its right neighbour, it assumes the active state, concatenates the neighbour's chunk to its own by getting access to the memory of this neighbour, and resumes parsing of this new input from the point it stopped (and, most importantly, with the memory of processing the previous chunk!). This is carried out by the procedure CORE.

Thus, in the course of the execution of PYRAMID the chunks become ever longer, and the number of processors involved in parsing steadily decreases. Eventually a single processor remains (at the top of the pyramid) whose chunk covers the entire input string. Therefore, when this processor reaches the end of its chunk (end of the sentence), all the sentence constituents are already closed and properly nested (if the sentence has a reading at all). At this moment this processor assumes the state **final** and activates the procedure WRAP-UP which builds the parse tree for the input sentence using the results of PYRAMID and FRIENDLY-NEIGHBOURS.

The **PYRAMID** algorithm is presented in Fig. 10.

Procedure **CORE** requires an explanation. The grammar of Fig. 7 is context-free. Therefore, starting at a left constituent boundary one can use any context-free parsing algorithm for detecting the right boundary of the constituent, which is precisely the goal of **PYRAMID**.

We use interruptible transition networks (ITNs) for this stage of parsing (see Ref. [7]). The processors in **PYRAMID** operate asynchronously. This means that in a general case a processor will not learn about the status it assumes next until at least one of its neighbours finishes processing. Hence, the transition network used by **CORE** must be able to function in an interrupt mode. This
Program PYRAMID (input_string, result_tree);
(*LF,RF: left,right flag;LEFT,RIGHT: left,right neighbour *)
Procedure LOOK_LEFT;
begin
  LF:=1;
  if RF(LEFT)=1 then STATE:=’passive’ else LF:=0
end; (* LOOK_LEFT *)
Procedure LOOK_RIGHT;
begin
  RF:=1;
  if LF(RIGHT)=1 then STATE:=’active’ else RF:=0
end; (* LOOK_RIGHT *)
Procedure STATE_CHOICE;
begin
  STATE:=’neutral’;
  repeat
    if LEFT=nil then
      if RIGHT=nil then STATE:=’final’ else LOOK_RIGHT
      else if RIGHT=nil then LOOK_LEFT
      else begin
        TOSS_A_COIN ("a random number ")
        case COIN of 'heads': LOOK_LEFT:
          'tails': LOOK_RIGHT
        end
      end
    until STATEჯ’neutral’
  end; (* STATE_CHOICE *)
Procedure CORE;
(* activates ITN *)
Procedure RESIGN;
(* provides the left neighbour of a passive processor with an exclusive access to
the memory of the latter, and stops the passive one *)
Procedure WRAP_UP;
(* completes the parse tree *)
begin (* the main program PYRAMID is executed in parallel
by all the pyramid processors *)
  INIT; (* initialization *)
  while STATE=’active’ do
  begin
    STATE_CHOICE;
    case STATE of
      ‘active’: CORE;
      ‘passive’: RESIGN;
      ‘final’: WRAP_UP
    end
  end
end. (* PYRAMID *)

Fig. 10. The PYRAMID algorithm.

network was designed to deviate as little as possible for a conventional ATN [8]. The most
significant augmentation of the ATN facilities is the presence of a new type of arc: the interrupt
arc. It has the effect of stopping the parsing of a chunk and storing the current states of the
Corresponding processor in such a way that traversal of the transition network can be resumed later
from the same network state.

Figure 11 illustrates the PYRAMID stage of parsing of the sample sentence of Fig. 3. PYRAMID
gets the initial string of quadruples produced at the previous stage (see row 0 at the bottom of
Fig. 11). Suppose, for example, that at the first level of PYRAMID (row 1 in Fig. 11) processors
#4, #6 and #9 become active; #5, #7 and #11, passive; and #1 and #8 remain neutral (in
Fig. 11 the processor states are denoted by “a”, “p” and “n”, respectively). As a result of the activity of
#4, 6 and 9 the delimiters in <NP, 4, 4, 1> and <VP, 6, 6, 1> are changed to “0”
(because the corresponding chunks grow and these quadruples find themselves inside those chunks),
the noun phrase <NP, 9, y, 1> is closed becoming <NP, 9, 10, 2>, and the quadruple <, , 11, 11, 2>
that denotes the end of the sentence turns passive (all immediate changes are underlined in Fig. 11).
There are two kinds of quadruples in the strings processed by PYRAMID: active quadruples (enclosed in angular brackets ⟨…⟩) which are to be scanned during the subsequent processing, and passive ones (enclosed in square brackets […]) which are to be omitted. Indeed, if at a certain level (see, for instance, the next paragraph) a constituent, C, with boundaries i,j is closed (e.g. ⟨S, 4, 6, 0⟩ in row 2, Fig. 11), then all the quadruples with the boundaries x,y such that \(i \leq x < y \leq j\), which are located between the boundaries of C, become passive because they must have already been closed, and hence need not be processed any more. So, the length of a string to be processed is actually determined by the number of its active quadruples.

On the second level 5 processors, namely #1, 4, 6, 8 and 9, remain in PYRAMID following the resignation of #5, 7, 11 (see row 2 in Fig. 11). Suppose that processors #4 and 8 assume the active status, #6 and 9, the passive, and #1, the neutral. Then #4 succeeds in closing ⟨S, 4, y, 0⟩ (result: ⟨S, 4, 6, 0⟩) and turns ⟨NP′, 4, 4, 0⟩, ⟨NP, 5, 5, 1⟩ and ⟨VP, 6, 6, 0⟩ into passive quadruples. #8 closes ⟨PP, 8, y, 1⟩ (result: ⟨PP, 8, 10, 0⟩), and makes ⟨NP, 9, 10, 2⟩ passive. Thus, on this level two sentence constituents are closed simultaneously by parallel processors, the number of active quadruples is reduced from 9 to 5, and only 3 processors remain in PYRAMID.

At his point 3 chunks with 3 unclosed sentence constituents remain: ⟨S, 1, y, 0⟩, ⟨NP, 1, y, 0⟩, and ⟨VP, 7, y, 1⟩. These constituents are closed at the next two levels (rows 3, 4). This leaves only one active quadruple, ⟨S, 1, 10, 0⟩, that represents the entire initial sentence. This single processor #1 left in PYRAMID gets into the final state and completes the parse tree (see Fig. 3).

3. PERFORMANCE

The PYRAMID algorithm is the central part of PARPAR. What are its time consumption characteristics? PYRAMID has a speed-up (in comparison with sequential ones) because different sections of an input string (e.g. different sentence constituents) are processed simultaneously by parallel processors, each contributing to the final result.
Let \( m \) denote the total number of processors used at some level of \textsc{pyramid} (except those ones that resigned at previous levels), and \( u \) stand for the number of uncoupled processors at this level. All passive processors resign. Therefore, the number, \( m' \), of processors passed to the next level is
\[
m' = m - (m - u)/2 = (m + u)/2.
\]

Let us define the \textsc{shrinkage}, \( s \), of the pyramid as
\[
s = m/m' = 2/(1 + u/m).
\]

In the best case (performance-wise) all the processors at a certain level are coupled, but in the worst case (see Fig. 10) the number of uncoupled processors can attain its maximum of \((m + 2)/3\), that is \(0 \leq u \leq (m + 2)/3\), and consequently (for large \( m\)):
\[
1.5 \leq s \leq 2. \tag{1}
\]

Let \( H \) denote the \textsc{height} of the pyramid, defined as the number of levels in \textsc{pyramid} with \( n \) processors at its bottom, so that
\[
H = \log n. \tag{2}
\]

Then (1) implies
\[
\log_2 n \leq H \leq \log_{1.5} n = 1.7 \log_2 n. \tag{3}
\]

An active processor gets access to the chunk that was previously processed by its passive partner, appends this chunk to its own, resumes parsing at the beginning of the newly added chunk and finishes at its end. Each processor employs a sequential parsing algorithm with running time depending on the length of the chunk. The latter is determined by the number of its active quadruples.

In order to estimate the length of a chunk at an arbitrary level of \textsc{pyramid} consider two sentence constituents, \( C_1 \) and \( C_2 \), represented by quadruples \( Q_1 = \langle C_1, x_1, y_1, d_1 \rangle \) and \( Q_2 = \langle C_2, x_2, y_2, d_2 \rangle \), respectively. We say that \( C_1 \) \textbf{contains} \( C_2 \) (\( C_1 \Rightarrow C_2 \)), and \( Q_1 \) \textbf{contains} \( Q_2 \) (\( Q_1 \Rightarrow Q_2 \)), if \( x_1 < x_2 \) and \( y_1 > y_2 \). If \( C_1 \Rightarrow C_2 \) and there is no constituent \( C_3 \) such that \( C_1 \Rightarrow C_3 \Rightarrow C_2 \) then \( C_2 \) \textbf{embedded} in \( C_1 \) (\( C_1 \Rightarrow C_2 \)), and \( Q_2 \) is embedded in \( Q_1 \) (\( Q_1 \Rightarrow Q_2 \)). In a parse tree (cf. Fig. 4) each sentence constituent is represented by a node in such a way that if \( C_1 \Rightarrow C_2 \) then \( C_2 \) belongs to the sub-tree of \( C_1 \), and if \( C_1 \Rightarrow C_2 \) then \( C_2 \) is an immediate successor of \( C_1 \) in the tree.

Let \( C \) embed several constituents:
\[
C_0 \Rightarrow (C_1, C_2, \ldots, C_k), \quad Q_0 \Rightarrow (Q_1, Q_2, \ldots, Q_k).
\]

The \textsc{pyramid} algorithm specifies that \( Q_0 \) cannot be closed until all the embedded quadruples \( Q_1, Q_2, \ldots, Q_k \) are closed, and all these quadruples remain active until then; when \( Q_0 \) is closed all the embedded quadruples become passive. Consider a chunk which is going to be processed on level \( i \) of \textsc{pyramid}, that is, its previous owner has completed in task, and resigned on level \( i - 1 \), and now the chunk is going to be accessed by its left neighbour for further processing (see Fig. 13). The chunk had been parsed by a processor on level \( i - 1 \), and that processor closed all the constituents that could be closed, and imposed the passive status onto the corresponding embedded quadruples.

Shaded boxes in Fig. 13 correspond to closed active constituents (passive ones are now shown), broken lines depict open (and therefore, active) constituents. If a constituent has been closed on level \( j \) of \textsc{pyramid} or embeds a constituent that has been closed on level \( j - 1 \) then its name has the subscript \( j \). In the example of Fig. 13 the constituent \( a \) is beyond the boundaries of the chunk, and therefore constituent \( d \) (which embeds a together with \( b \) and \( c \)) cannot be closed within the scope of this chunk. So, the constituents \( b \), \( c \), and \( d \) remain active.

![Fig. 12. A hypothetical worst-case distribution of couples and uncoupled processors (the latter are depicted by circles).](image-url)
Constituents d, e and f are embedded in a third level constituent, g. The fact that d is open keeps e, f and g active. By the same token h, v and w are active, and so on. A similar situation may be found at the right end of the chunk, namely, “layers” of active constituents from level 1 up to the level of the chunk. Let $r$ denote the average number of constituents embedded in one constituent of the next higher level. If a constituent of level $j$ is open or located beyond the chunk under consideration, it may keep on average $r - 1$ constituents of the same level $j$ in active state (and prevent one constituent of level $j + 1$ from closing). If a chunk is to be parsed on level $i$ (i.e. for its parsing was last attempted on level $i - 1$) then the average number of its active constituents, $x_i$, is

$$x_i = (r - 1)(i - 1).$$  \(4\)

Let $t(n)$ and $r(n)$ denote the average time of sequential parsing of a string of length $n$ and of pyramid parsing of the same string, respectively. According to Earley [9], in a general case of a context-free grammar the time to parse a sentence of length $n$ sequentially is proportional to $n^4$. Hence,

$$r(n) = \sum_{i=0}^{H} t(x_i) = \sum_{i=0}^{H} \beta (r - 1)^3(i - 1)^3 = \frac{\beta}{4} (r - 1)^3(H - 1)^3H^2 < \gamma H^4$$

where $\beta$ and $\gamma$ are constants.

This yields the following

**Proposition 3.** If $L$ is a LCB-local language specified by a context-free grammar of sentence constituents, then the time complexity of pyramid parsing of a sentence of $L$ having length $n$ is $O((\log n)^4)$ using $n$ processors.

The approach presented in this paper is based on the fact that the language under the consideration possesses certain property of locality, and just this property allows an efficient parallel multiprocessor implementation of the system. In particular, the communication network can be made simple from the point of view both of its configuration (see Fig. 2), and of the required bandwidth of its channels.

Indeed, all messages in the course of performing the parallel parsing algorithm are transmitted by the local bus. The latter consists of $n$ independent sections (in an $n$-processor system), such that each section connecting a pair of processors serves only these two processors, because only strictly

Fig. 14. The speed-up of the pyramid algorithm compared to sequential parsing.
local interactions between neighbouring processors take place in the system, and no processor is involved in transmitting data between remote nodes of the network. During stages 1 and 2 (see Fig. 3) neighbours exchange very little data. During stage 3 sections of the local bus support the coupling protocol (based on checking the state of a few binary flags, see Fig. 10), and provide an access from an active processor to the memory of its right neighbour, which has resigned, and therefore the active processor is a single user of the memory and the communication line. Thus, different sections of the local bus work in parallel, but no single section has to transmit any parallel messages simultaneously.

The global bus is used for broadcasting messages, which are intended for detecting a distributed termination of processing at stages 1 and 2 (stage 3 is completed by a single processor which detects the termination of parsing of a sentence without any need for communication). These control messages are rare compared to the working messages of the local bus, and short (they actually consist only of identifiers of the sending processors, so they can be transmitted through the global bus within a tolerable constant delay).

The required bandwidth of the global bus can be further reduced to one control message at a time if the bus has a hierarchical structure (see Refs [10–12]), while the corresponding delay of $O(\log n)$ will still not affect the performance estimate of Proposition 3.

Now one can express the speed-up, $g$, of the pyramid algorithm in comparison with sequential parsing:

$$g = \frac{t(n)}{r(n)} = \frac{4n^3}{(r-1)^2 (H-1)^2 H^2} \geq \frac{4n^3}{(r-1)^2 \log_1^2 n - 1^2 \log_{\frac{1}{2}}^2 n}.$$

The lower bound of $g$ (for $r = 2$) is plotted in Fig. 14 which shows that using the parallel technique described in this paper the performance of syntactic parsing of natural language can be enhanced by a factor of tens.

4. CONCLUSION

In search for efficient techniques of natural language processing, we show that parallel algorithms can be successfully applied to a language possessing certain properties of locality. The approach is illustrated by an example of a nontrivial subset of English. We describe an algorithm of parallel parsing without specialized processors and a multiprocessor system for its implementation. Its performance is estimated, and it is shown to be faster by a factor of tens than the well known sequential parsing algorithms (cf. Refs [8, 9]). Additional development of both the algorithm and the concept of LCB-locality is needed.

An additional advantage of this technique is that, since it possesses a strong bottom-up flavor, it is suitable for processing ungrammatical and ill-formed sentences. Indeed, since processing starts independently and simultaneously in many locations within a sentence, maximum length well-formed subtrings can be salvaged and certain syntactic errors can be detected. In its present state the system does not, however, explicitly address robustness considerations.

Acknowledgment—The authors are grateful to the anonymous referee whose stimulating comments were of a great help in clarifying the description and analysis of the system.

REFERENCES

3. Small S. L. and Rieger C., Parsing and comprehending with word experts. TR 1039, Department of Computer Science, University of Maryland (1981).

**About the Author—Eliezer Lozinskii** was born in the U.S.S.R. Ph.D. in Computer Science, 1968, from the Institute of Cybernetics, Kiev. Repatriated to Israel in 1973. Affiliation: Department of Computer Science, The Hebrew University of Jerusalem, Israel. Presently on a sabbatical leave at the State University of New York at Stony Brook.

**About the Author—Sergei Nirenburg** was born in 1952 in Kharkov, U.S.S.R. He obtained his master’s degree in computational linguistics from Kharkov State University and Ph.D. in linguistics from The Hebrew University of Jerusalem, Israel.