EVALUATION OF A PARALLEL CHART PARSER

Ralph Grishman and Mahesh Chitrao
Computer Science Department
New York University
251 Mercer Street
New York, New York 10012

Abstract

We describe a parallel implementation of a chart parser for a shared-memory multiprocessor. The speed-ups obtained with this parser have been measured for a number of small natural-language grammars. For the largest of these, part of an operational question-answering system, the parser ran 5 to 7 times faster than the serial version.

1. Introduction

We report here on a series of experiments to determine whether the parsing component of a natural language analyzer can be easily converted to a parallel program which provides significant speed-up over the serial program.

These experiments were prompted in part by the rapidly growing availability of parallel processor systems. Parsing remains a relatively time-consuming component of language analysis systems. This is particularly so if constraints are being systematically relaxed in order to handle ill-formed input (as suggested, for example, in (Weischedel and Sondheimer, 1983)) or if there is uncertainty regarding the input (as is the case for speech input, for example). This time could be reduced if we can take advantage of the new parallel architectures. Such a parallel parser could be combined with parallel implementations of other components (the acoustic component of a speech system, for example) to improve overall system performance.

2. Background

There have been a number of theoretical and algorithmic studies of parallel parsing, beginning well before the current availability of suitable experimental facilities.

For general context-free grammars, it is possible to adapt the Cocke-Younger-Kasami algorithm (Aho and Ullman 1972, p. 314 ff) for parallel use. This algorithm, which takes time proportional to n^3 (n = length of input string) on a single processor, can operate in time n using n^2 processors (with shared memory and allowing concurrent writes). This algorithm, and its extension to unification grammars, has been described by Haas (1987b). The matrix form of this algorithm is well suited to large arrays of synchronous processors. The algorithm we describe below is basically a CYK parser with top-down filtering[1], but the main control structure is an event queue rather than iteration over a matrix. Because the CYK matrix is large and typically sparse [2], we felt that the event-driven algorithm would be more efficient in our environment of a small number of asynchronous processors ($\ll n^2$ for our longest sentences) and grammars augmented by conditions which must be checked on each rule application and which vary widely in compute time.

Cohen et al. (1982) present a general upper bound for speed-up in parallel parsing, based on the number of processors and properties of the grammar. Their more detailed analysis, and the subsequent work of Sarkar and Deo (1985) focus on algorithms and speed-ups for parallel parsing of deterministic context-free grammars. Most programming language grammars are deterministic, but most natural language grammars are not, so this work (based on shift-reduce parsers) does not seem directly applicable.

Experimental data involving actual implementations is more limited. Extensive measurements were made on a parallel version of the

[1] We also differ from CYK in that we do not merge different analyses of the same string as the same symbol. As a result, our procedure would not operate in linear time for general (ambiguous) grammars.

[2] For grammar #4 given below and a 15-word sentence, the matrix would have roughly 15,000 entries (one entry for each substring and each symbol in the equivalent Chomsky normal form grammar), of which only about 1000 entries are filled.

Hearsay-II speech understanding system (R. Fennel and V. Lesser, 1977). However, the syntactic analysis was only one of many knowledge sources, so it is difficult to make any direct comparison between their results and those presented here. Bolt Beranek and Newman is currently conducting experiments with a parallel parser quite similar to those described below (Haas, 1987a). BBN uses a unification grammar in place of the procedural restrictions of our system. At the time of this writing, we do not yet have detailed results from BBN to compare to our own.

3. Environment

Our programs were developed for the NYU Ultracomputer (Gottlieb et al., 1983), a shared-memory MIMD parallel processor with a special instruction, *fetch-and-add*, for processor synchronization. The programs should be easily adaptable to any similar shared memory architecture.

The programs have been written in ZLISP, a version of LISP for the Ultracomputer which has been developed by Isaac Dimitrovsky (1987). Both an interpreter and a compiler are available. ZLISP supports several independent processes, and provides both global variables (shared by all processes) and variables which are local to each process. Our programs have used low-level synchronization operations, which directly access the fetch-and-add primitive. More recent versions of ZLISP also support higher level synchronization primitives and data structures such as parallel queues and parallel stacks.

4. Algorithms

Our parser is intended as part of the PROTEUS system (Ksiezyk et al. 1987). PROTEUS uses augmented context-free grammars – context-free grammars augmented by procedural *restrictions* which enforce syntactic and semantic constraints.

The basic parsing algorithm we use is a *chart parser* (Thompson 1981, Thompson and Ritchie, 1984). Its basic data structure, the *chart*, consists of nodes and edges. For an n word sentence, there are $n + 1$ nodes, numbered 0 to n. These nodes are connected by *active* and *inactive edges* which record the state of the parsing process. If $A \rightarrow W\ X\ Y\ Z$ is a production, an active edge from node n_1 to n_2 labeled by $A \rightarrow W\ X\ .\ Y\ Z$ indicates that the symbols $W\ X$ of this production have been matched to words $n_1 + 1$ through n_2 of the sentence. An *inactive* edge from n_1 to n_2 labeled by a category Y indicates that words n_1+1 through n_2 have been analyzed as a constituent of type Y. The "fundamental rule" for *extending* an active edge states that if we have an active edge $A \rightarrow W\ X\ .\ Y\ Z$ from n_1 to n_2 and an inactive edge of category Y from n_2 to n_3, we can build a new active edge $A \rightarrow W\ X\ Y\ .\ Z$ from n_1 to n_3. If we also have an inactive edge of type Z from n_3 to n_4, we can then extend once more, creating this time an inactive edge of type A (corresponding to a completed production) from n_1 to n_4.

If we have an active edge $A \rightarrow W\ X\ .\ Y\ Z$ from n_1 to n_2, and this is the first time we have tried to match symbol Y starting at n_2 (there are no edges labeled Y originating at n_2), we perform a *seek* on symbol Y at n_2: we create an active edge for each production which expands Y, and try to extend these edges. In this way we generate any and all analyses for Y starting at n_2. This process of seeks and extends forms the core of the parser. We begin by doing a seek for the sentence symbol S starting a node 0. Each inactive edge which we finally create for S from node 0 to node n corresponds to a parse of the sentence.

The serial (uniprocessor) procedure [3] uses a task queue called an *agenda* . Whenever a *seek* is required during the process of extending an edge, an entry is made on the agenda. When we can extend the edge no further, we go to the agenda, pick up a seek task, create the corresponding active edge and then try to extend it (possibly giving rise to more seeks). This process continues until the agenda is empty.

Our initial parallel implementation was straightforward: a set of processors all execute the main loop of the serial program (get task from agenda / create edge / extend edge), all operating from a single shared agenda. Thus the basic unit of computation being scheduled is a seek, along with all the associated edge extensions. If there are many different ways of extending an edge (using the edges currently in the chart) this may involve substantial computation. We therefore developed a second version of the parser with more-fine-grained parallelism, in which each step of extending an active edge is treated as a separate task which is placed on the agenda. We present some comparisons of these two algorithms below.

There was one complication which arose in the parallel implementations: a race condition in the application of the "fundamental rule". Suppose processor P_1 is adding an active edge to the

[3] written by Jean Mark Gawron.

chart from node n_1 to n_2 with the label $A \rightarrow W\ X\ .\ Y\ Z$ and, at the same time, processor P_2 is adding an inactive edge from node n_2 to n_3 with the label Y. Each processor, when it is finished adding its edge, will check the chart for possible application of the fundamental rule involving that edge. P_1 finds the inactive edge needed to further extend the active edge it just created; similarly, P_2 finds the active edge which can be extended using the inactive edge it just created. Both processors therefore end up trying to extend the edge $A \rightarrow W\ X\ .\ Y\ Z$ and we create duplicate copies of the extended edge $A \rightarrow W\ X\ Y\ .\ Z$. This race condition can be avoided by assigning a unique (monotonically increasing) number to each edge and by applying the fundamental rule only if the edge in the chart is older (has a smaller number) than the edge just added by the processor.

As we noted above, the context-free grammars are augmented by procedural restrictions. These restrictions are coded in PROTEUS Restriction Language and then compiled into LISP. A restriction either succeeds or fails, and in addition may assign features to the edge currently being built. Restrictions may examine the substructure through which an edge was built up from other edges, and can test for features on these constituent edges. There is no dependence on implicit context (e.g., variables set by another restriction). As a result, the restrictions impose no complications on the parallel scheduling; they are simply invoked as part of the process of extending an edge.

5. Grammars

These algorithms were tested on four grammars:

1. A "benchmark" grammar:

$$S \rightarrow XXXXXXXXXXXX$$
$$X \rightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j$$

2. A very small English grammar, taken from (Grishman, 1986) and used for teaching purposes. It has 23 nonterminal symbols and 38 productions.
3. Grammar #2, with four restrictions added.
4. The grammar for the PROTEUS question-answering system, which includes yes-no and wh- questions, relative and reduced relative clauses. It has 35 non-terminal symbols and 77 productions.

6. Method

The programs were run in two ways: on a prototype parallel processor, and in simulated parallel mode on a standard uniprocessor (the uniprocecessor version of ZLISP provides for relatively efficient simulation of multiple concurrent processes). The runs on our prototype multiprocessor, the NYU Ultracomputer, were limited by the size of the machine to 8 processors. Since we found that we could sometimes make effective use of larger numbers of processors, most of our data was collected on the simulated parallel system. For small numbers of processors (1-4) we had good agreement (within 10%, usually within 2%) between the speed-ups obtained on the Ultracomputer and under simulation [4].

7. Results

We consider first the results for the test grammar, #1, analyzing the sentence

$$jjjjjjjjjjjj$$

This grammar is so simple that we can readily visualize the operation of the parser and predict the general shape of the speed-up curve. At each token of the sentence, there are 10 productions which can expand X, so 10 seek tasks are added to the agenda. If 10 processors are available, all 10 tasks can be executed in parallel. Additional processors produce no further speed-up; having fewer processors requires some processors to perform several tasks, reducing the speed-up. This general behavior is borne out by the curve shown in Figure 1. Note that because the successful seek (for the production $X \rightarrow j$) leads to the creation of an inactive edge for X and extension of the active edge for S, and these operations must be performed serially, the maximal parallelism is much less than 10.

The next two figures compare the effectiveness of the two algorithms – the one with coarse-grained parallelism (only seeks as separate tasks) and the other with finer-grain parallelism (each seek and extend as a separate task). The finer-grain algorithm is able to make use of more parallelism in situations where an edge can be extended in several different ways. On the other

[4]For larger numbers of processors (5-8) the speed-up with the Ultracomputer was consistently below that with the simulator. This was due, we believe, to memory contention in the Ultracomputer. This contention is a property of the current bus-based prototype and would be greatly reduced in a machine using the target, network-based architecture.

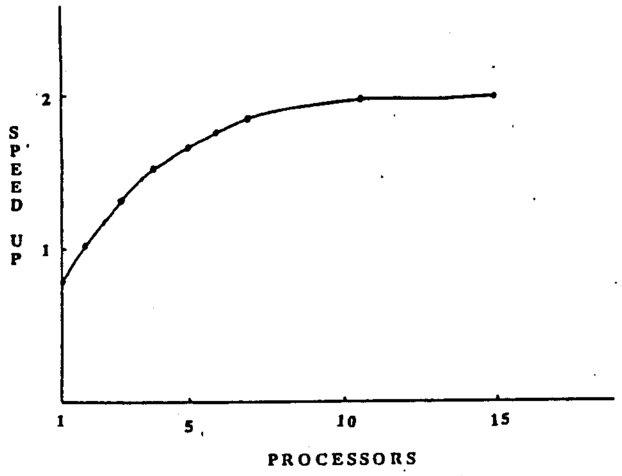

Figure 1: Speed-up (relative to serial parser) for grammar #1 and sentence jjjjjjjjjjj.

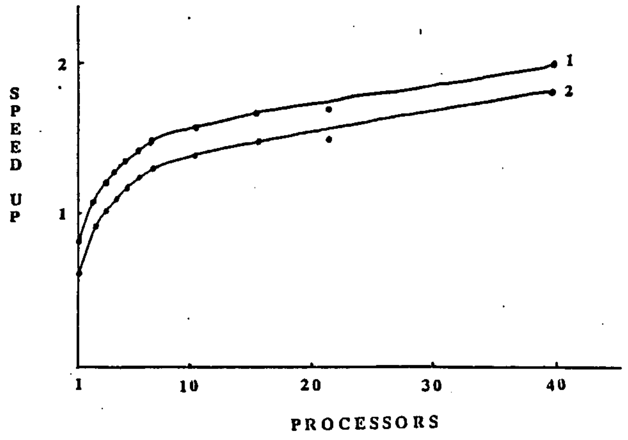

Figure 2: Speed-up (relative to serial parser) for grammar #2 (small grammar without restrictions) on a 3-word sentence for the coarse-grained algorithm (1) and the fine-grained algorithm(2).

hand, it will have more scheduling overhead, since each extend operation has to be entered on and removed from the agenda. We therefore can expect the finer-grained algorithm to do better on more complex sentences, for which many different extensions of an active edge will be possible. We also expect the finer-grained algorithm to do better on grammars with restrictions, since the evaluation of the restriction substantially increases the time required to extend an edge, and so reduces in proportion the fraction of time devoted to the scheduling overhead. The expectations are confirmed by the results shown in Figures 2 and 3. Figure 2, which shows the results using a short sentence and grammar #2 (without restrictions), shows that neither algorithm obtains substantial speed-up and that the fine-grained algorithm is

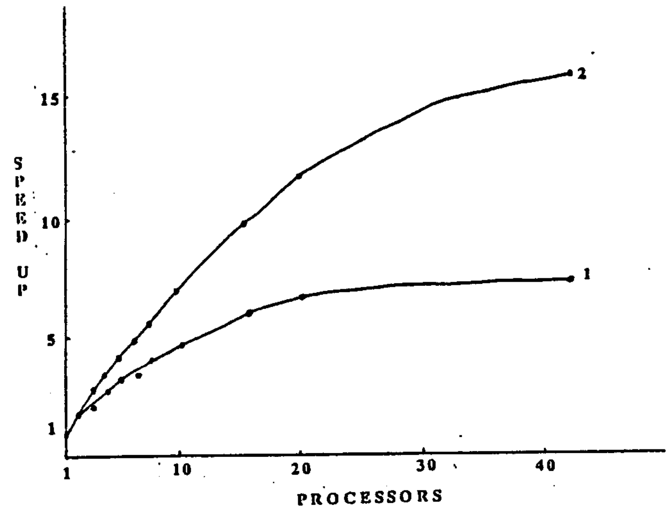

Figure 3: Speed-up (relative to serial parser) for grammar #3 (small grammar with restrictions) on a 14-word sentence for the coarse-grained algorithm (1) and the fine-grained algorithm(2).

in fact slightly worse. Figure 3, which shows the results using a long sentence and grammar #3 (with restrictions), shows that the fine-grained algorithm is performing much better.

The remaining three figures show speed-up results for the fine-grained algorithm for grammars 2, 3, and 4. For each figure we show the speed-up for three sentences: a very short sentence (2-3 words), an intermediate one, and a long sentence (14-15 words). In all cases the graphs plot the number of processors vs. the true speed-up – the speed-up relative to the serial version of the parser. The value for 1 processor is therefore below 1, reflecting the overhead in the parallel version for enforcing mutual exclusion in access to shared data and for scheduling extend tasks.

Grammars 2 and 3 are relatively small (38 productions) and have few constraints, in particular on adjunct placement. For short sentences these grammars therefore yield a chart with few edges and little opportunity for parallelism. For longer sentences with several adjuncts, on the other hand, these grammars produce lots of parses and hence offer much greater opportunity for parallelism. Grammar 4 is larger (77 productions) and provides for a wide variety of sentence types (declarative, imperative, wh-question, yes-no-question), but also has tighter constraints, including constraints on adjunct placement. The number of edges in the chart and the opportunity for parallelism are therefore fairly large for short sentences, but grow more slowly for longer sentences than with grammars 2 and 3.

These differences in grammars are reflected

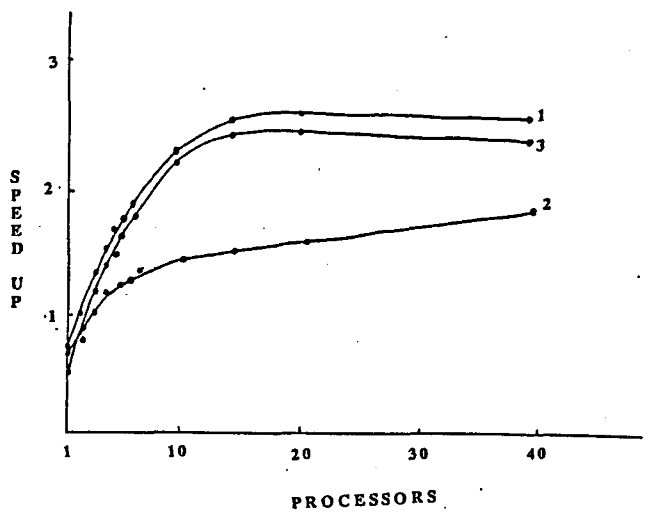

Figure 4: Speed-up (relative to serial parser) for grammar #2 (small grammar without restrictions) using the fine-grained algorithm for three sentences: a 10 word sentence (curve 1), a 3-word sentence (curve 2) and a 14-word sentence (curve 3).

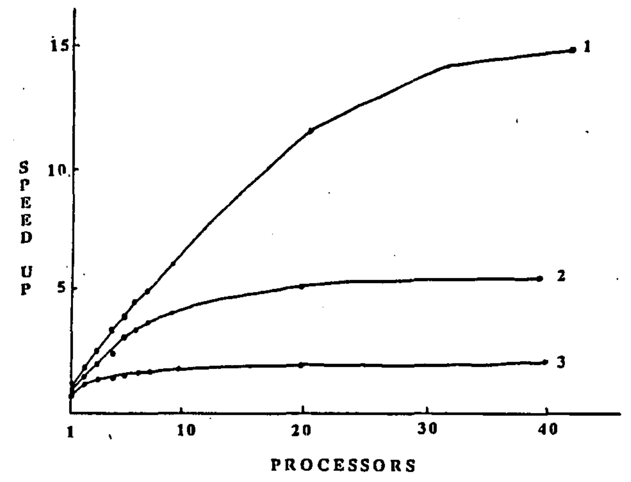

Figure 5: Speed-up (relative to serial parser) for grammar #3 (small grammar with restrictions) using the fine-grained algorithm for three sentences: a 14-word sentence (curve 1), a 5-word sentence (curve 2), and a 3-word sentence (curve 3).

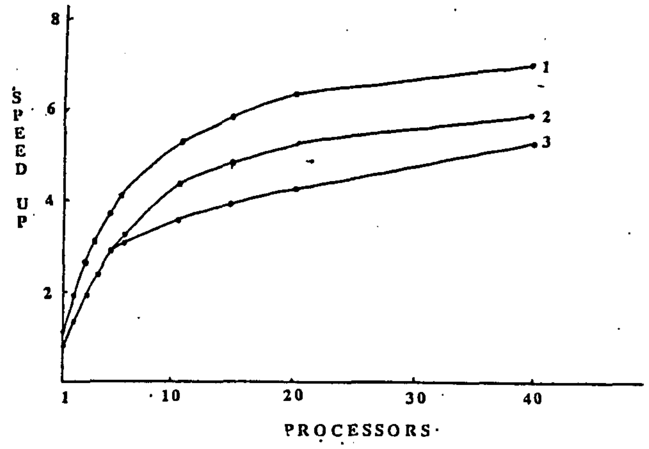

Figure 6: Speed-up (relative to serial parser) for grammar #4 (question-answering grammar) using the fine-grained algorithm for three sentences: a 15-word sentence (curve 1), a 2-word sentence (curve 2), and a 8-word sentence (curve 3).

in the results shown in Figures 4-6. For the small grammar without restrictions (grammar #2), the scheduling overhead for fine-grain parallelism largely defeats the benefits of parallelism, and the overall speed-up is small (Figure 4). For the same grammar with restrictions (grammar #3), the effect of the scheduling overhead is reduced, as we explained above. The speed-up is modest for the short sentences, but high (15) for the long sentence with 15 parses (Figure 5). For the question-answering grammar (grammar #4), the speed-up is fairly consistent for short and long sentences (Figure 6).

8. Discussion

Through relatively small changes to an existing serial chart parser, we have been able to construct an effective parallel parsing procedure for natural language grammars. For our largest grammar (#4), we obtained consistent speed-ups in the range of 5-7. Grammars for more complex applications, and those allowing for ill-formed input, will be considerably larger and we can expect higher speed-ups.

One issue which should be re-examined in the parallel environment is the effectiveness of top-down filtering. This filtering, which is relatively inexpensive, blocks the construction of a substantial number of edges and so is generally beneficial in a serial implementation. In a parallel environment, however, the filtering enforces a left-to-right sequencing and so reduces the opportunities for parallelism. We intend in the near future to try a version of our algorithm without top-down filtering in order to determine the balance between these two effects.

9. Acknowledgements

This report is based upon work supported by the Defense Advanced Research Projects Agency under Contract N00014-85-K-0163 from the Office of Naval Research, by the National Science Foundation under Grant No. DCR-8501843 and by the International Business Machines Corp. under contract 534816.

References

[1] Alfred Aho and Jeffrey Ullman, 1972, *The Theory of Parsing, Translation, and Compiling - Volume I: Parsing,* Prentice-Hall, Englewood Cliffs, NJ.

[2] Jacques Cohen, Timothy Hickey, and Joel Katcoff, 1982 Upper bounds for speedup in parallel parsing, *J. Assn. Comp. Mach. 29* (2), pp. 408-428.

[3] Isaac Dimitrovsky, 1987 *ZLISP 0.7 Reference Manual,* Department of Computer Science, New York University, New York.

[4] R. Fennel and V. Lesser, 1977, Parallelism in AI problem solving: a case study of Hearsay II, *IEEE Trans. Comp. C-26,* pp. 98-111.

[5] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Lawrence Rudolph, and Marc Snir, 1983, The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer, *IEEE Trans. Comp.,* pp. 175-189.

[6] Andrew Haas, 1987a, Parallel parsing, Talk at *Workshop on JANUS and Parallel Parsing,* Feb. 24-25, Bolt Beranek and Newman, Cambridge, MA.

[7] Andrew Haas, 1987b, Parallel Parsing for Unification Grammars. *Proc. IJCAI-87,* pp. 615-618.

[8] Tomasz Ksiezyk, Ralph Grishman, and John Sterling, 1987, An equipment model and its role in the interpretation of noun phrases. *Proc. IJCAI-87,* pp. 692-695.

[9] Dilip Sarkar and Narsingh Deo, 1985, Estimating the speedup in parsing, Report CS-85-135, Computer Science Dept., Washington State University.

[10] Henry Thompson, 1981, Chart parsing and rule schemata in phrase structure grammar, *Proc. 19th Annl. Meeting Assn. Computational Linguistics*, Stanford, CA, 167-72.

[11] Henry Thompson and Graeme Ritchie, 1984, Implementing natural language parsers. In *Artificial Intelligence Tools, Techniques and Applications* , T. O'Shea and M. Eisenstadt, eds., Harper and Row, New York.

[12] Ralph M. Weischedel and Norman K. Sondheimer, 1983, Meta-rules as a Basis for Processing Ill-Formed Input, *Am. J. Computational Linguistics*, 9(3-4), pp. 161-177.