Appeared in Australian Computer Science Communications 6 (1984), 7.1-7.10.

A Parallel Algorithm for Context-Free Parsing

Richard P. Brent
Centre for Mathematical Analysis
Australian National University
Canberra, ACT 2601

Leslie M. Goldschlager
Dept. of Computer Science
University of Sydney
Sydney, NSW 2006

Abstract

We present an algorithm which solves the parsing problem for any context-free grammar, and is suitable for execution on a synchronous computer with unbounded parallelism. The algorithm parses arbitrary input strings of length n in time O(log n) on a unit-cost SIMDAG, or in time $O(\log^2 n)$ on a log-cost SIMDAG, using $O(n^6)$ processors in each case.

1. Introduction

For a number of years, main memory on existing sequential computers has been so inexpensive and therefore plentiful that it has been reasonable to write programs which assume that memory is unbounded, rather than of a small fixed size. Thus programs are written which, for inputs of length n, declare arrays of size n (and sometimes n^2 or faster growing functions of n). These programs work because there is adequate main memory (occasionally supplemented by virtual memory) to handle their data storage requirements for the practical input sizes which the programs are expected to encounter. As a rough rule of thumb, the unbounded main memory assumption is usually reasonable if the program's main memory usage grows by no more than a (small) polynomial function of n, the input size.

Current hardware trends suggest that a similar assumption of unboundedness regarding the number of simple processors available may soon be reasonable. That is, it will be economically and technically feasible to build machines with hundreds of thousands, or even millions, of parallel processing elements interconnected in a sufficiently general way to permit general-purpose programming [18]. A number of theoretical computer models with these characteristics are surveyed in [9] (see also [7, 10, 12, 14, 23]). Of course, any real machine will be built with a finite number of processors, and "virtual processors" may be used in the same way that virtual memory is now.

All the reasonable parallel computer models which have been proposed turn out to be equivalent (up to a low-degree polynomial) in their time and hardware resource usage [15]. Thus the class of problems called NC, i.e. those problems computable in $O(\log^k n)$ time and $O(n^c)$ processors for some constants k and c, is invariant with respect to the details of the parallel computer model. However, for the sake of definiteness, we shall use the SIMDAG model [15]. (SIMDAG stands for "Single Instruction stream, Multiple Data stream, Global data".) Essentially, a SIMDAG consists of a large number of processors which all synchronously execute the same instruction (but perhaps on different data) at any unit of time, and which communicate via a global memory. To facilitate operating on different data, each processor has access to its own (unique) processor number. The global memory allows simultaneous reads and writes, with the proviso that if two or more processors attempt to write to the same memory location at the same time unit, then the lower numbered processor has precedence. For a unit-cost SIMDAG each instruction is considered to take unit time, whereas for a log-cost SIMDAG the time required for an instruction may be proportional to the number of bits required to represent the data and addresses relevant to the instruction. For ideas regarding the practical implementation of machines which approximate SIMDAGs, see [3, 23].

Many interesting and natural problems are known to be in the class NC [10], e.g. arithmetic operations, common vector and matrix operations, sorting and searching, and many graph-theoretic problems [1, 2, 8, 13, 17, 19, 20, 22, 25, 26]. Ruzzo [21] was the first to show that context-free parsing is in NC. In fact, Ruzzo showed that context-free languages could be recognized in time $O(\log^2 n)$ with $O(n^{15})$ processors. Ruzzo also stated (without proof) that the problem could be solved with "about n^6" processors. In Section 4 we present a parallel algorithm which requires $O(n^6)$ processors and $O(\log n)$ or $O(\log^2 n)$ time (depending on whether we make the unit-cost or log-cost assumption). The proof is given in Section 5.

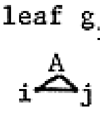
Our exponent 6 is better than Ruzzo's 15, but still uncomfortably high. It is conceivable that it might be reduced to 3 (or even 2.49555) as these numbers are the best exponents known for the time required for context-free language recognition on a sequential machine [11, 16, 24]. For further comments on the number of processors required, see Section 6. Lower bounds (for a more restrictive model) are discussed in [6].

2. Notation

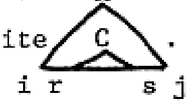
Let L be a (fixed) context-free language, and G a grammar for L in Chomsky normal form, i.e. each production of G has the form $A \rightarrow BC$ or $A \rightarrow d$, where A, B and C are non-terminals and d is a terminal symbol of G. The start symbol of G is denoted by S, and the number of non-terminals of G is denoted by N. $A \Rightarrow g_i \ldots g_j$ has the usual meaning (i.e. that there is a parse tree with root A and leaves $g_i \ldots g_j$). "$A \rightarrow BC$ in G" means that $A \rightarrow BC$ is a production of G.

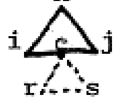
The input string to be parsed is $g_1 \ldots g_n$. We regard n as a variable but N as a constant. Thus, for example, the function $N^3 n^6$ is $O(n^6)$.

A tree with root A, leftmost leaf g_i and rightmost leaf g_j is denoted by



.

The degenerate cases $i = j$ (and $i = j = A$) are allowed if the tree has 2 (or 1) nodes. We shall identify trees with their roots, i.e. we may speak of "the tree A". The number of nodes in a tree A is denoted by $|A|$. If A has a subtree C with leftmost leaf g_r and rightmost leaf g_s, we assume that $i \leq r \leq s \leq j$ and write


. If C is replaced by a leaf c we write


.

W denotes the set $\{(A, i, j) \mid A$ is a nonterminal of G, $1 \leq i \leq n$, $1 \leq j \leq n\}$. It will be convenient to write A_{ij} for an element (A, i, j) of W.

When describing parallel algorithms, square brackets after a statement indicate that the statement should be executed in parallel by as many processors as necessary. For example, the statement

$$Q(A_{ij}B_{xy}) \; := \; A_{ij} = B_{xy} \qquad [A_{ij}, B_{xy} \in W]$$

initializes an array Q of dimension N^2n^4 and should be executed using N^2n^4 processors.

Finally, "$\log^k n$" is used as an abbreviation for $\lceil \log_2 n \rceil^k$.

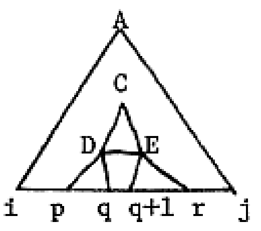
3. Preliminary Results

The algorithm described in Section 4 depends on the following trivial but useful properties of trees [4, 5]:

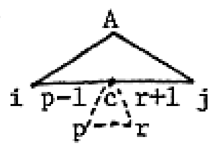
Lemma 1

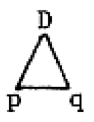
If A is a tree with $|A| = n$, there is a node C (called the "critical node") of A such that $|C| > n/2$ but $|D| \leq n/2$ for all children D of C.

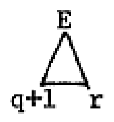
Intuitively, Lemma 1 says that the "large" tree



can be split into three "small" trees



,


and


,

where c is a leaf replacing the subtree C.

Lemma 2

Given a leaf b in a tree A with $|A| = n$, there is a node C of A such that $b \in C$, $|C| > n/2$, and $|D| \leq n/2$, where D is the subtree of C containing b.

Intuitively, the interpretation of Lemma 2 is similar to that of Lemma 1, except that we insist that a certain "distinguished" leaf b must be in the subtree C, and we can not conclude anything about the size of the subtree(s) of C which do not contain b. (In Lemma 2 we assume $n > 1$, for otherwise D does not exist.)

4. The Main Result and Parallel Algorithm

The main result of this paper is:

Theorem 1

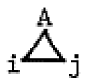
Let L be any context-free language. Then L can be recognized by a unit-cost SIMDAG in time $O(\log n)$ using $O(n^6)$ processors (i.e. for any input string $g_1 \ldots g_n$ we can decide if $g_1 \ldots g_n \in L$ in time $O(\log n)$ using $O(n^6)$ processors). On a log-cost SIMDAG L can be recognized in time $O(\log^2 n)$ with $O(n^6)$ processors.

The proof of Theorem 1 is constructive. First we give a parallel algorithm for context-free language recognition. Then, in Section 5, we show that the time and processor requirements of the algorithm satisfy the statement of Theorem 1. As above, let G be a grammar for L in Chomsky normal form.

The Parallel Algorithm

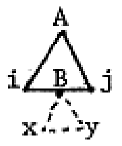
```
P($A_{ij}$) := ($i = j$) and ($A \rightarrow g_i$ in G)          [$A_{ij} \in W$];
Q($A_{ij}B_{xy}$) := $A_{ij} = B_{xy}$                           [$A_{ij}$, $B_{xy} \in W$];
for t := 1 to log n do
    begin
    U($C_{pr}B_{xy}$) := Q($C_{pr}B_{xy}$)                       [$C_{pr}$, $B_{xy} \in W$];
    if ($C \rightarrow DE$ in G) and
          ((Q($D_{pq}B_{xy}$) and P($E_{q+1,r}$)) or (P($D_{pq}$) and Q($E_{q+1,r}B_{xy}$)))
          then  U($C_{pr}B_{xy}$) := true                        [$1 \leq p, r \leq n$;  $1 \leq q < n$;  $B_{xy} \in W$;
                                                                  C, D, E nonterminals of G];

    if U($A_{ij}C_{pr}$) and U($C_{pr}B_{xy}$)
          then  Q($A_{ij}B_{xy}$) := true                        [$A_{ij}$, $B_{xy}$, $C_{pr} \in W$];
    V($C_{pr}$) := P($C_{pr}$)                                   [$C_{pr} \in W$];
    if ($C \rightarrow DE$ in G) and P($D_{pq}$) and P($E_{q+1,r}$)
          then  V($C_{pr}$) := true                              [$1 \leq p, r \leq n$;  $1 \leq q < n$;
                                                                  C, D, E nonterminals of G];

    if Q($A_{ij}C_{pr}$) and V($C_{pr}$)
          then  P($A_{ij}$) := true                              [$A_{ij}$, $C_{pr} \in W$]
    end;
if P($S_{1n}$) then accept else reject.
```

Intuitively, the arrays P and Q stored in global memory represent possible parse trees, and are updated on each iteration of the algorithm. We show in Section 5 that P and Q converge to their final values after at most log n iterations. $P(A_{ij})$ will eventually be set <u>true</u> iff there is a parse tree of the form



i.e. $A \Rightarrow g_i \ldots g_j$ in G.

$Q(A_{ij}B_{xy})$ will eventually be set <u>true</u> iff there is a parse tree of the form



i.e. $A \Rightarrow g_i \ldots g_{x-1} B g_{y+1} \ldots g_j$ in G.

The algorithm evaluates $P(B_{xy})$ and $Q(A_{ij}B_{xy})$ in parallel; if both are <u>true</u> (for one or more of the many possible B_{xy}) then $P(A_{ij})$ is <u>true</u>. (This is the key idea, but to get convergence in log n iterations we have to refine it and make use of Lemmas 1 and 2.) Finally, $g_1 \ldots g_n$ is accepted iff $P(S_{1n})$ is <u>true</u>, i.e. $S \Rightarrow g_1 \ldots g_n$ in G.

<u>5. Proof of Theorem 1</u>

Theorem 1 follows easily from the following two Lemmas.

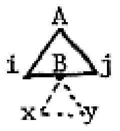
<u>Lemma 3</u>

If $g_1 \ldots g_n \in L$ then the algorithm of Section 4 accepts.

<u>Proof</u>

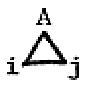
If $g_1 \ldots g_n \in L$ then $S \Rightarrow g_1 \ldots g_n$ in G, so there is a parse tree T with root S and n leaves, so $|T| = 2n-1 < 2^{\log n + 1}$. Hence, it is sufficient to show the following properties H1 and H2 hold after the t-th iteration of the algorithm:

H1: If there is a parse tree T of the form



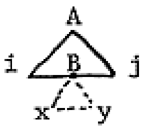
with $|T| \leq 2^t$, then $Q(A_{ij}B_{xy})$ = <u>true</u>;

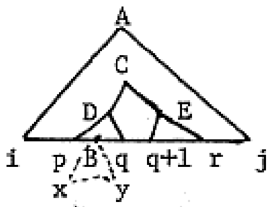
H2: If there is a parse tree T of the form

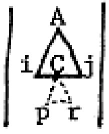
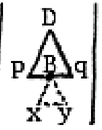
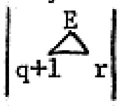


with $|T| \leq 2^{t+1}$ then $P(A_{ij})$ = <u>true</u>.

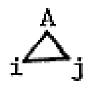
We shall prove H1 and H2 by induction on t. They are certainly true for t = 0 by the initialization phase of the algorithm (i.e. the first two parallel statements which initialize P and Q). Hence, we shall assume by induction that H1 and H2 hold (with t replaced by t-1) after t-1 iterations, and prove that H1 and H2 also hold after the t-th iteration.

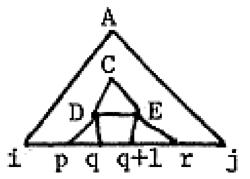
To prove H1, assume that there is a parse tree T of the form 
with $|T| \leqslant 2^t$. By Lemma 2,

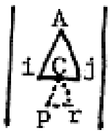
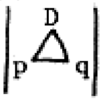
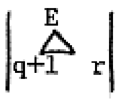
T = 

where $\Bigg|$  $\Bigg| \leqslant 2^{t-1}$, $\Bigg|$  $\Bigg| \leqslant 2^{t-1}$, $\Bigg|$  $\Bigg| \leqslant 2^t$, and $C \rightarrow DE$ is in G

(or the analogous case in which B is in the subtree with root E rather than D). Thus, by the inductive hypothesis, $Q(A_{ij}C_{pr}) = Q(D_{pq}B_{xy}) = P(E_{q+1,r}) =$ true. It is easy to check that the algorithm will set $U(A_{ij}C_{pr})$, $U(C_{pr}B_{xy})$, and finally $Q(A_{ij}B_{xy})$ to true, as required to prove H1.

To prove H2, assume that there is a parse tree T of the form 
with $|T| \leqslant 2^{t+1}$. By Lemma 1,

T = 

where $\Bigg|$  $\Bigg| \leqslant 2^t$, $\Bigg|$  $\Bigg| \leqslant 2^t$, $\Bigg|$  $\Bigg| \leqslant 2^t$, and $C \rightarrow DE$ is in G.

Thus, by the inductive hypothesis, $Q(A_{ij}C_{pr}) = P(D_{pq}) = P(E_{q+1,r}) =$ true. It is easy to check that the algorithm will set $V(C_{pr})$ and then $P(A_{ij})$ to true, as required to prove H2.

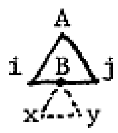
We have proved that H1 and H2 hold for all $t \geqslant 0$. The lemma now follows from H2 (with A = S and t = log n).

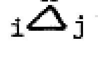
Lemma 4

If the algorithm of Section 4 accepts, then $g_1 \ldots g_n \in L$.

Proof

We shall prove a stronger result: after the t-th iteration,

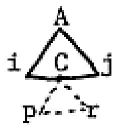
H1: if $Q(A_{ij}B_{xy}) =$ true then there is a parse tree of the form  ; and

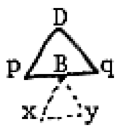
H2: if $P(A_{ij}) =$ true then there is a parse tree of the form  .

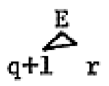
The proof is by induction on t. As inductive hypothesis we assume that H1 and H2 hold after the (t-1)-th iteration. Suppose that $Q(A_{ij}B_{xy})$ = true after the t-th iteration. Either $Q(A_{ij}B_{xy})$ was set to true before the t-th iteration, in which case the inductive hypothesis shows that the desired parse tree exists, or there is some $C_{pr} \in W$ such that $U(A_{ij}C_{pr})$ is set to true during the t-th iteration, as is $U(C_{pr}B_{xy})$. There are several cases to consider. For example, suppose that after the (t-1)-th iteration $Q(A_{ij}C_{pr})$ is true but $Q(C_{pr}B_{xy})$ is false. Then there exist nonterminals D, E and an index q such that $C \rightarrow DE$ in G and

$(Q(D_{pq}B_{xy})$ and $P(E_{q+1,r}))$ or $(P(D_{pq})$ and $Q(E_{q+1,r}B_{xy}))$

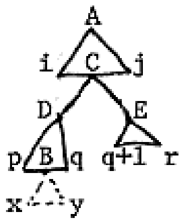
is true after the (t-1)-th iteration. Suppose that $Q(D_{pq}B_{xy})$ and $P(E_{q+1,r})$ are true after the (t-1)-th iteration (the case that $P(D_{pq})$ and $Q(E_{q+1,r}B_{xy})$ are true is similar). By the inductive hypothesis, there exist parse trees

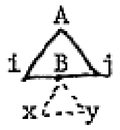


,


and


which can be combined to give a parse tree



of the desired form


.

From this and similar cases, we conclude that H1 holds after the t-th iteration.

To establish H2 after the t-th iteration, we argue in a similar manner. If $P(A_{ij})$ is true after the t-th iteration, either $P(A_{ij})$ was true after the (t-1)-th iteration (in which case we use the inductive hypothesis), or there exists $C_{pr} \in W$ such that $Q(A_{ij}C_{pr})$ and $V(C_{pr})$ were set to true during (or before) the t-th iteration. H2 now follows from a case analysis similar to that above.

Proof of Theorem 1

Inspection of the algorithm of Section 4 shows that it uses at most $N^3n^6 = O(n^6)$ processors. The execution time is dominated by the loop, which is executed log n times. Subscript calculations associated with the arrays can be precomputed before entering the loop, in time O(log n) on the unit-cost model, or $O(\log^2 n)$ on the log-cost model. Each iteration of the loop then takes constant time (on the unit-cost model) or O(log n) time (on the log-cost model), so the total running time is as claimed in the statement of the theorem.

6. Comments

The algorithm was presented in terms of accepting or rejecting its input string. However it is clear that, if each time an element of P or Q is set to true, information is retained about why that element was set to true, then a parse tree for the input string can be reconstructed.

There are connections between algorithms for context-free language recognition and algorithms for matrix multiplication. For serial computation, context-free language recognition can be reduced to multiplication of n by n matrices, so the time required is $O(n^{2.49555})$ [11, 16, 24]. Since time bounds for serial algorithms are often the same as processor bounds for fast parallel algorithms, we might hope to recognize context-free languages with $O(n^{2.49555})$ processors in time $O(\log^k n)$. However, it is not known if this is possible.

Considering the number of processors used by our parallel algorithm, the most expensive statement is:

$$\underline{\text{if}}\ U(A_{ij}C_{pr})\ \underline{\text{and}}\ U(C_{pr}B_{xy})\ \underline{\text{then}}\ Q(A_{ij}B_{xy}) := \underline{\text{true}}\quad [A_{ij}, B_{xy}, C_{pr} \in W]$$

which is essentially performing the multiplication of Boolean matrices of size $Nn^2 = O(n^2)$. Thus, it should be possible to reduce the number of processors required by our parallel algorithm to $O(n^{4.9911})$, where the exponent is twice the best known for matrix multiplication [11], and still retain the time bound $O(\log^k n)$, although k may be greater than in Theorem 1.

7. Conclusion

We have given an algorithm for parsing an arbitrary context-free grammar in $O(\log n)$ time on a unit-cost SIMDAG ($O(\log^2 n)$ time on a log-cost SIMDAG) with $O(n^6)$ processors. It is unlikely that the time bound could be improved much. In Section 6 we suggested that the processor bound might be reduced somewhat. Certainly the constant factor can be reduced considerably by taking advantage of the ordering constraints on the subscripts i, j, ... in the algorithm. On a real parallel machine with a bounded number of processors we would probably combine the ideas of the algorithm presented here with those of more traditional serial parsing algorithms, in order to make the best use of whatever degree of parallelism was available, as in [4, 5].

8. Acknowledgements

We thank the referees for their comments, which have helped to improve the presentation of the paper.

9. References

1. M. Ajtai, J. Kolmos and E. Szemeredi, "An O(n log n) sorting network", Proc. Fifteenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1983, 133-139.

2. A. Borodin, J. von zur Gathen and J. E. Hopcroft, "Fast parallel matrix and GCD computations", Proc. Twenty-Third Annual IEEE Symposium on Foundations of Computer Science, IEEE, New York, 1982, 65-71.

3. A. Borodin and J. E. Hopcroft, "Routing, merging, and sorting on parallel models of computation", Proc. Fourteenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1982, 338-344.

4. R. P. Brent, "The parallel evaluation of arithmetic expressions in logarithmic time", in Complexity of Sequential and Parallel Numerical Algorithms (edited by J. F. Traub), Academic Press, New York, 1973, 83-102.

5. R. P. Brent, "The parallel evaluation of general arithmetic expressions", J. ACM 21 (1974), 201-206.

6. R. P. Brent and L. M. Goldschlager, "Some area-time tradeoffs for VLSI", SIAM J. on Computing 11 (1982), 737-747.

7. A. K. Chandra, L. J. Stockmeyer and U. Vishkin, "A complexity theory for unbounded fan-in parallelism", Proc. Twenty-Third Annual IEEE Symposium on Foundations of Computer Science, IEEE, New York, 1982, 1-13.

8. F. Y. Chin, J. Lam and I.-N. Chen, "Efficient parallel algorithms for some graph problems", Comm. ACM 25 (1982), 659-665.

9. S. A. Cook, "Towards a complexity theory of synchronous parallel computation", L'Enseignement Mathematique 27 (1981), 99-124.

10. S. A. Cook, "An overview of computational complexity", Comm. ACM 26 (1983), 401-408.

11. D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication", SIAM J. on Computing 11 (1982), 472-492.

12. S. Fortune and J. Wyllie, "Parallelism in random access machines", Proc. Tenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1978, 114-118.

13. Z. Galil and W. J. Paul, "An efficient general-purpose parallel computer", Proc. Thirteenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1981, 247-262.

14. L. M. Goldschlager, "A unified approach to models of synchronous parallel machines", Proc. Tenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1978, 89-94.

15. L. M. Goldschlager, "A universal interconnection pattern for parallel computers", J. ACM 29 (1982), 1073-1086.

16. J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Massachusetts, 1979.

17. G. Lev, N. Pippenger and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks", IEEE Trans. on Computers C-30 (1981), 93-100.

18. C. A. Mead and L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Massachusetts, 1980.

19. F. P. Preparata, "New parallel-sorting schemes", IEEE Trans. on Computers C-27 (1978), 669-673.

20. J. H. Reif and L. G. Valiant, "A logarithmic time sort for linear size networks", Proc. Fifteenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1983, 10-16.

21. W. L. Ruzzo, "On uniform circuit complexity", J. Computer and System Sciences 22 (1981), 365-383.

22. Y. Shiloach and U. Vishkin, "Finding the maximum, merging and sorting in a parallel computation model", J. of Algorithms 2 (1981), 88-102.

23. J. D. Ullman, Computational Aspects of VLSI, Computer Science Press, Rockville, Maryland, 1984, Chapter 6.

24. L. G. Valiant, "General context free recognition in less than cubic time", J. Computer and System Sciences 10 (1975), 308-315.

25. L. G. Valiant, "A scheme for fast parallel communication", SIAM J. on Computing 11 (1982), 350-361.

26. L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication", Proc. Thirteenth Annual ACM Symposium on the Theory of Computing, ACM, New York, 1981, 263-277.