In Proceedings of Natural Language Understanding and Logic Programming (NLULP) 1999, pp. 45–57

Parallel Principle-Based Parsing

Sandiway Fong
NEC Research Institute
4 Independence Way
Princeton NJ, USA
sandiway@research.nj.nec.com

This paper describes a small-scale, coarse-grained parallel Prolog-based Principles-and-Parameters parser that is derived essentially "for free", modulo minor control flow changes, from an existing serial implementation. The grammar remains unchanged. The system is designed to operate as efficiently as possible under the severe constraints imposed by loosely-coupled processors. We demonstrate speed-up results on such a network using an adaptive task distribution strategy.

1 Introduction

This paper investigates whether significant speed-up can be achieved on a loosely-coupled, or distributed memory, parallel system for a Prolog principle-based parser. Due to the very high cost of processor communication, loosely-coupled systems constitute a particularly severe test. In fact, it is generally difficult to achieve any speed-up at all with non-trivial applications. [Tho89], for example, reports no speed-ups only slow-downs with parallel chart parsing. However, since any cluster of workstations constitutes a loosely-coupled system, such systems are by far the most commonly available. Using a variety of scheduling algorithms, we report a very usable 90% and 60% parallel efficiency for two and four processors on such a network. We stress these results are achieved without "parallelizing the code"; in fact, without any modifications to the grammar. The key here is that OR-parallelism requires no processor communication. In this case, the problem reduces to (1): finding enough parallelism to exploit, and (2): maintaining load balancing across the network. We analyze the test data and show that the distribution is highly non-uniform and data- or sentence-dependent. We describe and compare an adaptive load balancing strategy designed to satisfy (1) and (2) with standard distribution algorithms.

1.1 Principle-Based Parsing

We begin with a modular Prolog-based parser [Fon91] under Government-Binding (GB) theory, an influential linguistic theory of the 1980s within the Principles-

[0]The author gratefully acknowledges the invaluable assistance provided by the following persons: Robert C. Berwick for his encouragement and for providing testing facilities for the multiple-processor experiments. Zheng Lin for his expert advice on all aspects of parallel systems.

and-Parameters framework [Cho81].[1] In GB theory, the working hypothesis is that there exists a universal core grammar contains a small set of non-language-specific rules or constraints, known as *principles*, that spell out the possible phrase structures (PSs). Under this theory, each principle is partially responsible for constraining the space of legal parses for a given sentence. One distinguishing characteristic of this system (compared with traditional linguistic descriptions) is that it attempts to account for the apparent surface complexity of the language phenomenon in terms of a deeper but smaller set of basic rules. Moreover, principles, in general, are accorded equal status and are not prioritized to override or supercede each other.[2] With few, if any, pre-conditions on their application, principles are free to interact with one another, and complexity of description is traded for complexity of interaction.

Parametric variation among languages is encoded by a set of *language parameters*, or simple boolean values. For example, well-known differences such as basic subject-object-verb word order, ellipsis of subjects and *wh*-word fronting are captured here. Linguistic principles form sub-systems, called *modules* describing different aspects of grammar. For example, $\overline{\text{X}}$-theory describes the internal structure of phrases and how they are combined into sentences. Trace theory describes how fronted or scrambled phrases are related to their "original" positions. θ-theory describes how thematic-roles from heads like verbs are distributed to their arguments, and Binding theory constrains the distribution of appropriate antecedents for anaphors and pronouns. (See [Cho86], [LU88] and [vRW86] for further details.) The PAPPI system uses a single set of approximately thirty principles to handle a range of examples drawn from English, Japanese, Korean, Dutch, German, French, Spanish, Turkish and Hungarian. Parsing is carried out by constructing PSs that simultaneously satisfy all the principles. If there are two or more such PSs, then the input sentence has been determined to be ambiguous. If the input is ungrammatical, one or more principles will block. Following the theory, the PAPPI system will look for all possible derivations for all possible parses. We note here that the problem of finding all solutions is harder to speed-up than the corresponding problem of finding just one solution.

If each principle is represented as a separate predicate, a parser is simply some conjunction of these predicates, see figure 1. Each predicate is classified as a *filter*, F_i, an operation that rejects incorrect structures, or *generator*, G_i, an operation that produces one or more structures. Here, G_1 and G_2 will produce the initial, or underspecified, parse trees and identify the possible movement configurations, or trace/antecedent pairs, respectively. We assume these two operations are non-controversially generators, see figure 2. Our experiments will take place on this *coarse-grained* OR-parallel model, where each alternative structure can be farmed out to a different processor.[3]

[1]See [New91] for details of the impact of GB theory on the field of theoretical linguistics.

[2]In the sense that there is no strict order of application. Of course, this does not preclude principles from affecting grammaticality in different ways. For example, the mildly-unintelligible question *Who does Mary wonder why John hit* is ruled out by *Subjacency*, a principle that governs how far the object of the verb *hit*, namely *who*, can be preposed. Contrast this with the strongly-ungrammatical sentence *John is crucial to see this*, a violation of the *Empty Category Principle*.

[3]We note here that fine-grained OR-parallel *full* Prologs exist for shared-memory MIMD machines. See the references in the conclusions.

```
parse(S) :-
     G1(S,PS),
     F1(PS),
     ...
     Fi(PS),
     G2(PS,SS),
     Fi+1(SS),
     ...
     reportParse(SS),
     fail.
parse(_).
```

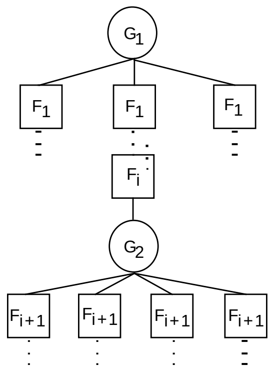


Figure 1: The Serial Parser

(1) G_1: Generating basic phrase structure

(i) John chose the book in the car

(ii) John [$_{\text{VP}}$ chose [$_{\text{NP}}$ the book [$_{\text{PP}}$ in the car]]]

(iii) John [$_{\text{VP}}$ [$_{\text{VP}}$ chose [$_{\text{NP}}$ the book]] in the car]

(2) G_2: Analyzing movement

(i) John wanted to be arrested

(ii) John wanted [$_{\text{CP/IP}}$ PRO$_i$ to be arrested t_i]

(iii) John$_i$ wanted [$_{\text{CP/IP}}$ PRO to be arrested t_i]

(iv) John$_i$ wanted [$_{\text{CP/IP}}$ t_i to be arrested NP-e]

(v) John$_i$ wanted [$_{\text{CP/IP}}$ t_i to be arrested t_i], etc.

Figure 2: Generators G_1 and G_2

Test Data	n	Σx (s)	$\bar{x}$ (s)
Complete	333	2418	6.26
Tail 10%	33	1874	56.8
Head 90%	300	545	1.8

Figure 3: Parse Times for the Lasnik & Uriagereka Corpus[4]

1.2 Loosely-Coupled Parallel Systems

As [Tho89] remarks, *loosely-coupled* systems, namely, systems consisting of processors that do not share memory or provide relatively slow access to shared memory, are more prevalent than the other forms, and therefore worth considering despite the severe penalty incurred for all non-local memory accesses. In its most common form, any typical collection of workstations such as a cluster of Sun Sparcstations on the Ethernet constitutes a loosely-coupled parallel system. We will assume precisely such a configuration and report on real parsing times gathered on 1–8 processor configurations. The recipe for good speed-up is simple: (1) (virtually) zero communication between processors, and (2) good load-balancing behaviour. We note here that some parallel Prolog programs, e.g. N-Queens [Lin92], have a very uniform task distribution. For others, like the one described here, task distribution strategy will play an important role.

Consider the test data summarized in figure 3. These are grammatical and ungrammatical examples taken “A Course in GB Syntax” [LU88].[5] Note that the (serial) parsing time for the worst 10% (about 1 minute) dominates that of the remaining 90% (under 2 seconds). Here, we will be primarily concerned with getting good speed-up over the former (our primary test set).

1.3 Previous NLP Work

An extensive literature on context-free grammars and its various subsets exists. [Nij89] surveys parallelism for the LR, CKY (see also [Haa87] for Unification Grammars) and Earley algorithms, and [CHK82] gives theoretical upper bounds for synchronous, bottom-up parsers. Chart parsing has been discussed in [GC88] and [Tho89]: the two papers reporting widely disparate results attributable to differences in architecture, namely shared-memory versus loosely-coupled. Finally, we mention that there are also connectionist approaches. For example, [S.93] describes a connectionist Principle-based parser.

2 Four Task Distribution Strategies

Good load balancing without communication is possible given uniform task size Otherwise, explicit load balancing may be required. Hence, it is important to pay careful attention to the test data. This section begins by examining the task distribution for the test data introduced above. Next, we introduce four different strategies and compare their performance on up to eight processors.

[4]In figure 3, the number of sentences is given by n. Σx (s) and $\bar{x}$ (s) are the total and mean serial parsing time, respectively. All results are for Quintus Prolog 3.1.4/3.2 on a Sun Sparcstation 10/51.

[5]In fact, approximately half of the 333 test examples are ill-formed.

2.1 The Test Data

The test set consists of the worst-performing 10% of the aforementioned [LU88] corpus in terms of processing time. Both grammatical and ill-formed sentences are represented. For brevity, the examples in the table in figure 4 are listed by chapter and sentence number from [LU88] only. (For the actual sentences, see Appendix A.) Figure 4 summarizes the salient features of each example as follows:

- Serial parsing time (the basis for comparison).
- Amount of communication required if parse trees are passed between processors.
- The branching factors, $b(G_1)$ and $b(G_2)$, which limit the degree of parallelism available.

Some general remarks are appropriate here. The opportunity for parallelism is much higher at G_2. However, the advantage of G_2 over G_1 is by means clear.[6] However, the actual task size is also important. Consider the graph in figure 5. It quantifies how much of the total time can be parallelized as a fraction of the total (serial) parse time.[7] Here, the balance is clearly in favour of G_1.

2.2 The Naïve Master/Slave Model

The idea behind the *naïve master/slave* model shown in figure 6 is quite simple: have one master processor dedicated to churning out trees which are passed to slaves for further processing.[8] Note that the amount of data passed by the master can be of the order of several megabytes (average being 383Kb), as shown above in figure 4. To help keep this cost down, our actual implementation also allows the master to act as a slave when all other slaves are busy, thus avoiding having to send the tree in this case.

2.3 The Round-Robin and Token-Based Models

The common characteristic of these two models is that they both trade redundant computation for avoiding communication. In a loosely-coupled environment this has been shown to be an effective strategy [Lin92]. The key idea being that all processors compute G_1 (in parallel), thereby having all parse trees locally available, but then agree upon *a plan* to divide up the remaining work, hopefully in a fair manner. This is the essential difference between the two methods:

[6] Also, Size $= b(G_1) = b(G_2) = 0$ for some entries. This means that there are no parse trees, and hence no tasks to distribute. We exclude these entries from our comparison.

[7] For our purposes, this is $1 - (t(G)/T_s)$, $t(G)$ being the total time taken before task distribution.

[8] The code should be completely obvious. The master executes `parse/1` and the slaves `parse2/1`. The master maintains a free/busy-slave table. `freeProc/1` picks a free processor from this table and `send/2` delivers it to `parse2/1` on the slave. We assume that `reportParse/1` delivers any parses found back to the master and that `free` notifies the master that the slave has finished with the tree.

Example	Time (s)	Size (Kb)	$b(G_1)$	$b(G_2)$	Example	Time (s)	Size (Kb)	$b(G_1)$	$b(G_2)$
[1:22]	7.74	75	18	96	[6:6d]	16.43	47	8	14
[2:79]	7.81	49	12	146	[6:34]	17.35	65	12	223
[4:50b]	8.13	81	16	64	[2:107]	22.34	571	16	42
[1:57b]	8.45	0	0	0	[1:30d]	22.46	93	20	402
[3:70b]	8.94	66	9	25	[3:17']	23.85	19	3	188
[3:71a]	9.01	66	9	25	[2:88]	28.42	152	32	176
[6:6c]	9.31	66	12	48	[6:43]	35.35	56	8	12
[6:5a]	9.33	24	4	6	[1:85b]	54.78	141	24	694
[6:41d]	9.34	57	12	28	[6:25]	64.81	64	8	47
[1:23]	9.55	50	12	146	[3:46]	70.67	46	6	452
[3:26a]	10.35	11	2	141	[6:47]	84.23	3726	32	41
[1:39d]	10.37	53	12	146	[2:110]	93.95	1258	40	571
[1:57c]	10.42	0	0	0	[3:36a]	119.55	249	15	293
[1:39b]	10.49	54	12	146	[3:36b]	120.58	249	15	293
[3:25]	12.56	11	2	175	[6:23]	126.57	175	10	110
[1:57a]	12.91	0	0	0	[1:57d]	328.38	4579	48	410
					[3:37]	489.17	488	8	7198

Figure 4: Characteristics of the Test Set

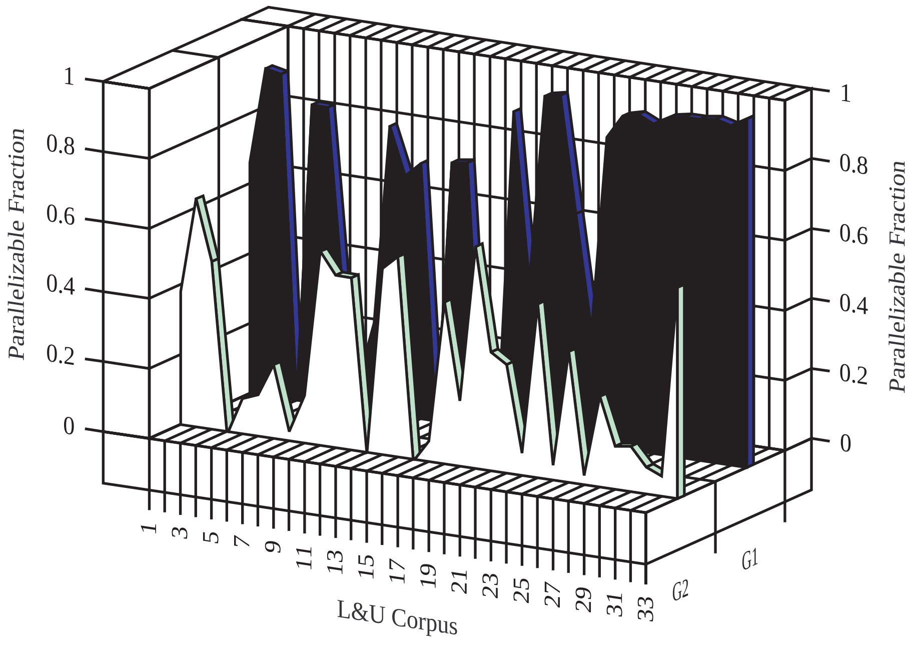


Figure 5: Parallelization Fractions for G_1 and G_2

```
parse($S$) :-              parse2($PS$) :-
     $G_1$($S$,$PS$),            $F_1$($PS$),
     freeProc($P$),             ...
     send($P$,$PS$),            $F_i$($PS$),
     fail.                      $G_2$($PS$,$SS$),
parse(_).                       $F_{i+1}$($SS$),
                                ...
                                reportParse($SS$),
                                fail.
                           parse2(_) :- free.
```

Figure 6: Naïve Master/Slave model

```
parse($S$) :-
     $G_1$($S$,$PS$),
     mod(I,N),
     $F_1$($PS$),
     ...
     $F_i$($PS$),
     $G_2$($PS$,$SS$),
     $F_{i+1}$($SS$),
     ...
     reportParse($SS$),
     fail.
parse(_) :- free.
```

Figure 7: Round-Robin model

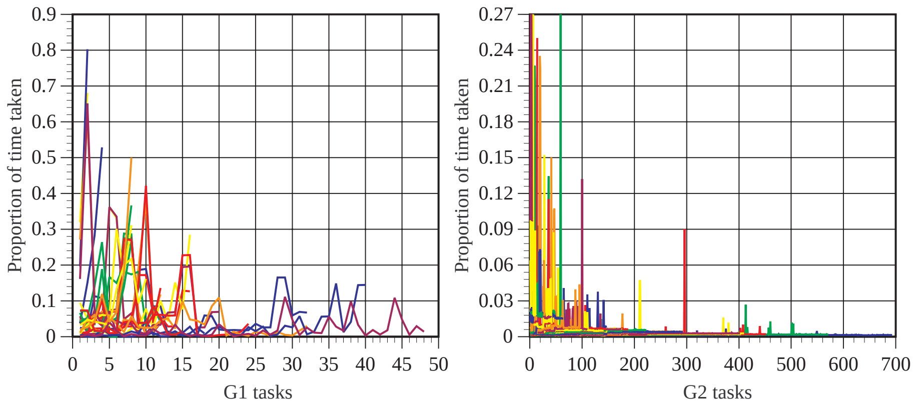


Figure 8: Task Distribution on G_1 and G_2

- **Round-Robin:** Let $c = 1, 2, 3, \ldots$ index the parses output by G_1. Each processor ist.$1 \leq i \leq n$ computes all c satisfying $i = \ c \bmod n$: namely `mod/2` in figure 7 — otherwise identical to figure 1.[9] No communication is necessary. However, depending on the input, the static partitioning may be unfair.
- **Token-Based:** Substitute `beg/0` for `mod/2`. (See figure 9a.) The idea is to have each processor "beg" for permission to handle a parse by sending c, the index, to a central token server. This server hands out permissions on an ordered, first-come-first-served basis. We gain run-time load balancing for the small communication overhead necessary to send indices only (crucially, not parses).

Is load balancing necessary? The task distribution data in figure 8 suggests that it really is. Note that the distribution is not only highly non-uniform at G_1, but also at G_2 where there is more parallelism.

2.4 The Two-Level Token-Based Model

We have the option of distributing tasks at G_1 or G_2. The tradeoff is going to be between the degree of parallelism and the fraction (of total time) available for speed-up. As the tests will show, this tradeoff is not justified overall. That is, we cannot exploit the additional parallelism that G_2 has to offer. The 2-Level Token model attempts to get the best-of-both-worlds by shifting to G_2 only when there is opportunity for speed-up. For example, the parasitic-gap sentence: "Which report did you file without reading?" has the following peculiar property, namely $b(G_1) = 1$ and $b(G_2) = 47$. In other words, there is only one initial parse tree but a high degree of ambiguity in terms of possible trace/antecedent pairs. No G_1-based scheduler can parallelize this example.

The idea is this: add an (initially) inactive beggar to G_2, see figure 9b. When there are idle processors, but the overall computation has not yet finished, `beg(2)` will be switched on. On our parasitic-gap sentence, initially only one processor will get permission for the single branch. The unsuccessful beggars

[9]For $n = 2$, processor 1 and 2 compute the odd and even parses, respectively.

(a) Single Level Token model

```
parse(S) :-
     G_1(S,PS),
     beg,
     F_1(PS),
     ...
     F_i(PS),
     G_2(PS,SS),
     F_{i+1}(SS),
     ...
     reportParse(SS),
     fail.
parse(_) :- free.
```

(b) Two Level Token model

```
parse(S) :-
     G_1(S,PS),
     beg(1),
     F_1(PS),
     ...
     F_i(PS),
     G_2(PS,SS),
     beg(2),
     F_{i+1}(SS),
     ...
     reportParse(SS),
     fail.
parse(S) :-
     restart -> parse(S) ; free.
```

Figure 9: Two Token Passing Models

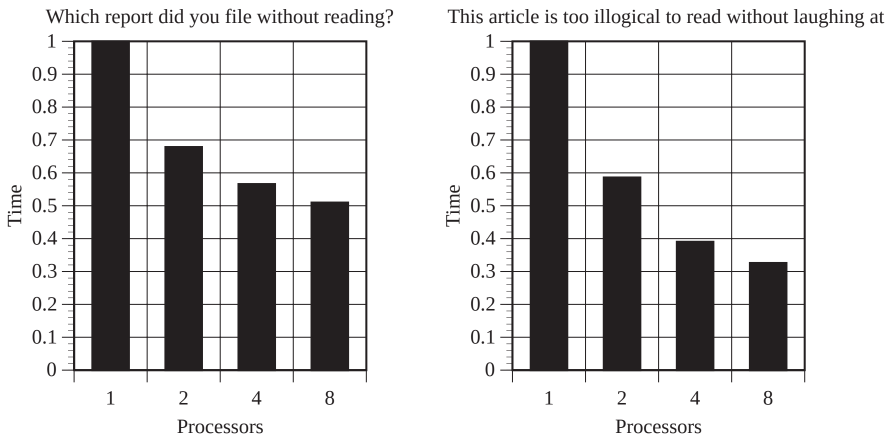


Figure 10: Two Test Cases for the Two-Level Model

will be restarted (via `restart`) with `beg(2)` turned on for all processors.[10] A further optimization, is to detect idle processors *before* they have completed G_1. For example, if there are eight branches, and some processor P begs for branch 3 when branches 3–8 have been allocated (but not necessarily completed), the token server can suggest that there is no further point in computing any more branches, but instead to go and beg for one of the as-yet-uncompleted branches. An excellent example is the sentence: "This article is too illogical to read without laughing at." (Incidentally, also the slowest test sentence.) Here, $b(G_1) = 8$, but $b(G_2) = 3778$ (just) for branch 8. Speed-up results for these two examples are shown in figure 10.

2.5 General Comparisons

The graphs in figure 11 summarize the results. The first graph provides a comparison of parallel efficiency. The second graph documents the aggregate run times for the entire test set.

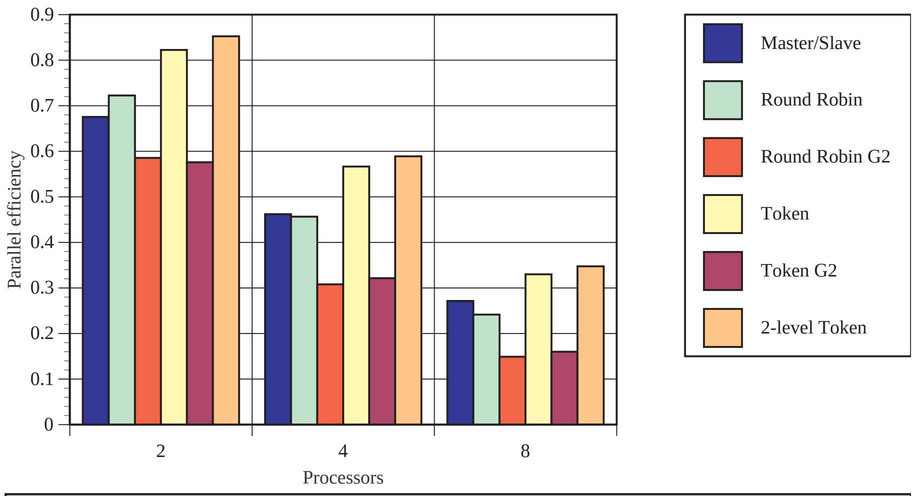
Here, as is conventional, efficiency E_n is defined as $T_s/(n \times T_p)$, the weighted ratio between single (T_s) and multi-processor (T_n) parse times. As expected, the 2-level Token model beats out the others, but the gain over the regular Token model is quite small. The efficiency close to 90% and 60% for the two and four processors, respectively, is quite respectable. For eight processors, there is a marked decline, suggesting that we are nearing the maximum amount of parallelism possible without further partitioning. However, even for this case, as the second graph shows, there is still a net decrease in the total run time. Finally, note that the G_2-only versions of the Round-Robin and Token algorithms are considerably slower than their G_1-only counterparts.

3 Conclusions

We began with the premise that it was easy to exploit multiple processors in the PAPPI framework. That is, "it comes for free" in the sense that no changes to the underlying grammar was necessary to exploit a coarse grain OR-parallel model. On a loosely-coupled system, experiments verified that the major issues were load balancing and avoiding communication at all costs. We saw that the underlying test data was highly non-uniform, (perhaps a property present in other non-trivial AI search applications). Despite this, we were able to achieve real, but limited speed-ups under small-scale parallelism. For future work, we note here that any improvements to the LR(1)-based algorithm that implements G_1 would automatically result in an increase in parallel efficiency.

However, we also draw the conclusion that further improvements are unlikely under the current assumptions given that there are only a very limited number of realistic low-overhead distribution strategies. An interesting comparison would be against a fine-grained, OR-parallel Prolog operating on shared-memory systems as described by [Car90] and [Hau90]. Since support for parallelism is provided at the Prolog implementation level, referential transparency would be maintained.

[10]This is a simplified description. The actual implementation has also to handle the synchronisation problem.



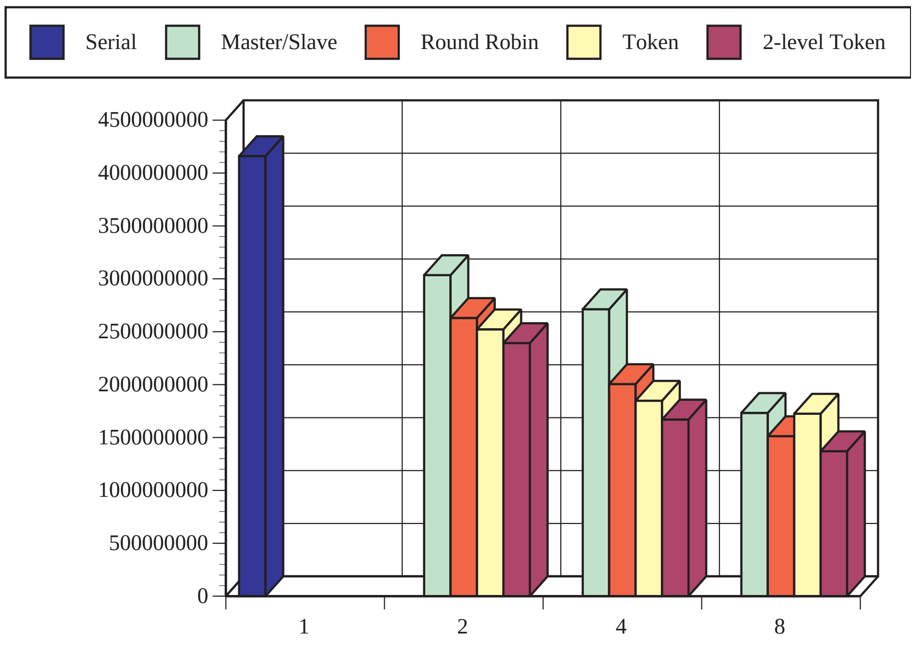


Figure 11: Comparison of Task Distribution Strategies

On the one hand, since linear speed-up is generally hard to obtain, it should be clear that "parallelism" in itself cannot substitute for genuine algorithmic improvements. However at the same time, linguistic theory continually evolves at a high rate. So, on the other hand, parallelism is a perfectly valid and dependable tool for getting real speed-ups without reverting to theory-specific optimizations that might not survive the next update to linguistic theory. In other words, parallelism is a practical way to maintain abstraction and modularity.

References

[Car90] M. Carlsson. *Design and Implementation of an OR-Parallel Prolog Engine.* PhD thesis, SICS, Sweden, 1990.

[CHK82] J. Cohen, T. Hickey, and J. Katcoff. Upper bounds for speedup in parallel parsing. *JACM*, 29(2), 1982.

[Cho81] N.A. Chomsky. *Lectures on Government and Binding.* Foris Publications, 1981.

[Cho86] N.A. Chomsky. *Knowledge of Language: Its Nature, Origin, and Use.* Prager, 1986.

[Fon91] S. Fong. *Computational Properties of Principle-Based Grammarical Theories.* PhD thesis, Artificial Intelligence Laboratory, MIT, 1991.

[GC88] R. Grishman and M. Chitrao. Evaluation of a parallel chart parser. In *Second Conference on Applied Natural Language Processing*, 1988.

[Haa87] A. Haas. Parallel parsing for unification grammars. In *IJCAI-87*, 1987.

[Hau90] B. Hausman. *Pruning and Speculative Work in OR-Parallel PROLOG.* PhD thesis, SICS, Sweden, 1990.

[Lin92] Z. Lin. Self-organizing task scheduling for parallel execution of logic programs. In *Proceeding of the 1992 Conference on Fifth Generation Computer Systems*, Japan, 1992.

[LU88] H. Lasnik and J. Uriagereka. *A Course in GB Syntax: Lectures on Binding and Empty Categories.* MIT Press, 1988.

[New91] F. J. Newmeyer. Rules and principles in the historical development of generative syntax, 1991.

[Nij89] A. Nijholt. Parallel parsing strategies in natural language processing. In *International Workshop on Parsing Technologies*, Carnegie Mellon University, 1989.

[S.93] Stevenson S. A competition-based explanation of syntactic attachment preferences and garden path phenomena. In *Proceedings of the ACL*, Ohio, 1993.

[Tho89] H. S. Thompson. Chart parsing for loosely coupled parallel systems. In *International Workshop on Parsing Technologies*, Carnegie Mellon University, 1989.

[vRW86] H.C. van Riemsdijk and E. Williams. *Introduction to the Theory of Grammar*. MIT Press, 1986.

A The Test Corpus

Sentences from [LU88] as summarized previously in the table in figure 4:

[1:22]	*I am eager John to be here
[2:79]	*I believe to be here
[4:50b]	*Who believes the claim that John left why
[1:57b]	*I wonder you think John said who you will see
[3:70b]	?The man who I wonder whether he will win the race disappeared
[3:71a]	?The man who you wonder whether he will win the race disappeared
[6:6c]	*I tried it to seem that Bill was intelligent
[6:5a]	*The man (that) it is likely to be clever disappeared
[6:41d]	*The belief of John to be clever is illogical
[1:23]	I am eager to be here
[3:26a]	The report was filed without being read
[1:39d]	*John is wanted to be here
[1:57c]	*I wonder you think who John said you will see
[1:39b]	John is believed to be here
[3:25]	What was filed without being read
[1:57a]	*I wonder you think John said you will see who
[6:6d]	*It is likely it to seem that Bill is intelligent
[6:34]	*John tried Bill to seem that he likes
[2:107]	They think that it is likely that pictures of each other are on sale
[1:30d]	I persuaded the bus to arrive on time
[3:17']	The report which I filed without reading disappeared
[2:88]	*I am proud of my belief to be intelligent
[6:43]	*John$_i$ seems that his$_i$ belief to be clever is ill founded
[1:85b]	*I tried to be likely that John is here
[6:25]	Someone$_i$ gave every actress$_j$ that he$_i$ met a book that she$_j$ appreciated
[3:46]	*The article which I filed it yesterday without reading is (over) here
[6:47]	John$_i$, Mary thinks that John$_i$ said that Susan believes I like
[2:110]	*They think that it surprised each other that Bill won
[3:36a]	?The man who I hired because Mary said would work hard disappeared
[3:36b]	*The man who I hired because he said would work hard disappeared
[6:23]	Every man$_i$ asked some actress$_j$ that he$_i$ met about some play that she$_j$ appeared in
[1:57d]	I wonder who you think John said you will see
[3:37]	This article is too illogical to read without laughing at