Parallelizing Description Logics

Frank W. Bergmann (fraber@cs.tu-berlin.de),
J. Joachim Quantz (jjq@cs.tu-berlin.de),
Projekt KIT-VM11, FR 5-12, Technische Universität Berlin,

1 Introduction

In the last 15 years Description Logics (DL) have become one of the major paradigms in Knowledge Representation. Whereas in the beginning it was hoped that DL provide representation formalisms which allowed efficient computation, at least three trends in recent years caused efficiency problems for DL systems and applications:

- a trend towards expressive dialects;
- a trend towards complete inference algorithms;
- a trend towards large-scale applications.

With the current state of technology it seems not possible to build a DL system for large-scale applications which offers an expressive dialect with complete inference algorithms. The standard strategy to cope with this dilemma is to restrict either expressivity, or completeness, or application size.

In [Bergmann, Quantz 95] we investigate an alternative approach, namely a parallelization of Description Logics. We present two alternative Prolog implementations of parallelized propagation on a loosely coupled MIMD (Multiple Instruction Multiple Data) system. In the following we briefly resume the argumentation of this paper and present the experimental results of the implementations.

2 Inference Components in DL Systems

In this section we briefly sketch the three main inference components of *normalize-compare* based DL systems and point out their complexity and their potential for parallellization.

Subsumption Checking. To test subsumption between two terms, both terms are first *normalized* and then structurally compared. The format of normalization rules depends on the expressiveness of the DL dialect. For the purpose of this

paper it is sufficient to consider normalforms as sets of atoms and normalization rules as having the form

$$\alpha_1, \ldots, \alpha_n \quad \rightharpoonup \quad \beta$$

i.e. if the atoms $\alpha_1, \ldots, \alpha_n$ are contained in a normalform, then β is added to this normalform.

Classification. When processing the term introductions, each term name is classified, i.e. compared with all previously introduced names. As a result *subsumption hierarchies* for concepts and roles are obtained, which are directed acyclic graphs.

Propagation. The two reasoning components described so far are usually called *terminological reasoning*. We will now turn our attention towards *assertional reasoning*, i.e. reasoning on the object level. The main difference between terminological and assertional reasoning is that the former is inherently local, whereas the latter is inherently global. In principle we can distinguish between a local phase and a nonlocal phase in object-level reasoning.

In the local phase we determine for an object the *most specific concept* it instantiates. This can be done by using the standard normalize and compare predicates and the search for direct supers in the classification component. Thus we normalize the description of an object thereby obtaining a normal form and compare it with the normal forms of the concepts in the hierarchy. In addition to this standard classification we also have to apply DL rules when processing objects. This is achieved by applying all rules whose left-hand sides subsume the object's normal form. After this application the normal form is again normalized and classified until no new rules are applicable.

In the nonlocal phase we have to propagate information to other objects. For illustration consider the following propagation rules:

$$\begin{aligned}
o_1 :: \text{all(r,c)} \;\&\; \text{r:}o_2 &\quad \rightharpoonup \quad o_2 :: \text{c} \\
o_1 :: \text{r:}o_2 \;\&\; \text{atmost(1,r)},\; o_2 :: \text{c} &\quad \rightharpoonup \quad o_1 :: \text{all(r,c)} \\
o_1 :: \text{r}_1\text{:}o_2,\; o_2 :: \text{r}_2\text{:}o_3 &\quad \rightharpoonup \quad o_1 :: \text{r}_1\text{comp r}_2\text{:}o_3 \\
o_1 :: \text{r:}o_2 &\quad \rightharpoonup \quad o_2 :: \text{inv(r):}o_1
\end{aligned}$$

We call these rules nonlocal since information at an object o_1 can have impact on an object o_2. Depending on the "connectivity" of the objects, adding a new description at an object can thus cause a reclassification of arbritrarily many other objects.

Parallelization Strategies. In principle, all three inference components show some parallel potential. We claim, however, that parallelizing propagation is the most promising [Bergmann 95]. The reason for this is that the basic operations in normalization, comparison, and classification are rather fine-grain, compared with the message passing overhead of MIMD systems. Object-level propagation, on the other hand, is an ideal candidate for our parallelization strategy. Each propagation is rather time-consuming and causes additional propagations which

can be straightforwardly parallelized since they are both independent and monotonic. In the following section we will present two different strategies for parallelizing propagation.

We begin by noting several relevant properties of object-level propagation. As already indicated above propagation of information from one object to another can cause additional propagation to other objects. This kind of 'ping-pong' interaction terminates only when a fixed point is reached an no new information is produced.

During this process, the number of propagation under consideration rises exponentially with respect to the *connectivity* between objects. This 'chain reaction' will stop as soon as the new information (from the object tell) becomes more and more integrated into the network. This results in a smaller 'fan out' that leads to a decrease of pending propagations until the fixed point is reached.

Given this analysis we consider two parallel paradigms as potential candidates for an implementation:

- The *farm* paradigm consists of one central process 'master' that coordinates several distributed 'worker' processes. The initial task is split by the master into several subtasks that are evenly distributed among the workers that process their tasks and return the results to the master.

- In the *communicating-objects* paradigm the central management institution (master) of the farm parallelism is replaced by (local) knowledge of all objects about the addresses of their neighbours. Objects communicate directly with each other, in contrast to the centred communication scheme of the farm. This helps to avoid communication bottlenecks in a network.

3 Experimental Results

We chose the 'Parsytec Multicluster II' machine as base for the parallel implementation of FLEX [Quantz et al. 94]. It consists of 16 processing nodes that each contain an INMOS T800 Transputer with 4 MByte of RAM. Each Transputer consists of a RISC processing kernel, a memory interface and 4 DMA driven serial interfaces of 1.2 MBytes/s each.

The language used to implement Distributed FLEX is a Prolog dialect called Brain Aid Prolog (BAP). It represents a 'standard' Clocksin & Mellish [Clocksin, Mellish 84] Prolog with parallel library extensions, implementing a scheme similar to CSP [Hoare 85]. Parallelism and synchronization is expressed explicitly using the notion of 'processes' and 'messages'. A process in BAP is a single and independent Prolog instance with a private database. A message is any Prolog term that becomes exchanged between two processes. Messages are send and received using the send_msg(Dest, Msg) and rec_msg(Sender, Msg) predicates. Message sender and destination are identified by their 'process id' (PID). Messages are routed transparently through the network. The order of messages is maintained when several messages are send from the same sender to the same destination. When a message reaches its destination process, it is

	(seq)	1	2	3	5	8
c10_3_2	(58)	78	63	55	56	58
c20_3_2	(177)	253	185	159	160	162

Figure 1: FARM Execution times (seconds).

		1		2		3		5		8		11		15
c10_3_2	(1.0)	59	(1.9)	30	(1.8)	32	(3.3)	18	(3.3)	18	(2.8)	21	(3.3)	18
c10_3_3	(1.0)	43	(1.5)	28	(1.5)	28	(2.4)	18	(2.7)	16	(2.5)	17	(2.9)	15
c10_3_4	(1.0)	327	(2.0)	159	(2.3)	141	(3.1)	105	(3.8)	87	(4.4)	74	(4.4)	73
c20_3_2	(1.0)	179	(1.8)	97	(3.0)	59	(3.1)	58	(4.0)	45	(4.8)	37	(4.5)	40
c20_3_3				145		129		58		56		66		51
c20_3_4				240		173		164		70		74		68
c20_5_3				527		355		173		155		141		160
c20_5_4				344		453		411		185		126		189
c40_3_2				314		176		137		105		111		72
c40_3_3						258		231		141		111		87
c40_3_4						569		467		264		319		230
c80_3_2						1032		665		225		200		181
c80_3_3										443		336		266
c80_3_4										947		662		583

Figure 2: Distributed Objects Execution Times (seconds).

stored in a special database, called 'mail box'. Each process owns its private mail box in which the messages are stored FIFO.

The main area of FLEX applications within the KIT research group is Natural Language Processing (NLP). Unfortunately memory limitations kept us from using these applications as benchmarks. Instead we imitate the structure of our NL applications leading to benchmarks with similar behaviour but much lower memory requirements.

Figure 1 shows the execution times for the farm benchmarks. The first row contains the benchmark names that are composed by three numbers that indicate the number of objects, concepts and fanout respectively (for example 'c20_5_3' means that the benchmark consists of 20 objects, 5 concepts and a fan out of 3). The following rows give the execution times with respect to the number of processors. The '(seq)' fields gives the reference time of the (original) sequential version of FLEX.

The parallelization of FLEX using the farm paradigm showed rather poor results (efficiencies $< 50\%$). This can be explained by the rather high costs to distribute the system state out to the workers and to integrate the computation results back into the system state. Both activities have to be done sequentially, thus slowing down the parallel execution.

The distributed objects parallelization results turns out to be a lot more promising. The table in Figure 2 [1] shows high speedups (efficiencies $> 80\%$) for all benchmarks, if the number of objects exceeds the number of processors by a

[1] The missing figures in the lower left corner are due to memory restrictions

certain factor (between 5 and 10). This result can be interpreted by the perspective that the network efficiently is mainly dependent on the number of pending propagations in the network [Bergmann, Quantz 95]. If this number is too low, few processing nodes are busy, resulting in an uneven diestribution of workload.

4 Conclusion

The results of the parallel implementation of FLEX are in general very satisfying. We achieved high speedups with applications structurally similar to natural language processing. The efficiency of execution is mainly dependent on the propagation/processor ratio and thus with the application size. This is an important result because especially large applications are to be considered candidates for a parallel implementation. Theoretical considerations [Bergmann 95] show that there are only few technical limits to the scalability of the distributed objects implementation.

We assume that the communication structure of FLEX is similar to many other applications in Artificial Intelligence. In particular, applications involving forward-chaining inferencing and agenda mechanisms are potential candidates for a parallelization based on the our aproach, due to structural similarities of data and control flow.

References

[BAP 93] F.W. Bergmann, M. Ostermann, G. von Walter, "Brain Aid Prolog Language Reference", Brain Aid Systems, 1993

[Bergmann 94] F.W. Bergmann, "Data Parallelism in Description Logics", *Transputer Anwender Treffen' 94*, 12–14

[Bergmann 95] F.W. Bergmann, "Parallelisierung eines Wissensrepräsentationssystems", (in preparation), TU Berlin, 1995

[Bergmann, Quantz 95] F.W. Bergmann, J.J. Quantz "Parallelizing Description Logics" (submitted for publication, http://www.cs.tu-berlin.de/ fraber/papers/BergmannQuantz95.html)

[Bond, Gasser 88] A. Bond, L. Gasser, "Readings in Distributed Artificial Intelligence", Morgan Kaufmann, Los Angeles, CA, 1988

[Clocksin, Mellish 84] W.F. Clocksin, C.S. Mellish, "Programming in Prolog", Springer Verlag, Berlin, 1984

[Dossier 91] A.C. Dossier, "Intelligence Artificielle Distribuee", "Bulletin de l'AFIA", **6**, 1991

[Hoare 85] C. A. R. Hoare, "Communicating Sequential Processes", Prentice Hall, Englewood Cliffs, N.J., USA, 1985

[Quantz et al. 94] J.J. Quantz, G. Dunker, V. Royer, "Flexible Inference Strategies for DL Systems", in F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (eds), *International Workshop on Description Logics*, DFKI Report D-94-10, DFKI Saarbrücken, 27–30, 1994