SIMULATION RESULTS OF A MULTIPROCESSOR PROLOG ARCHITECTURE BASED ON A DISTRIBUTED AND/OR GRAPH

C. Percebois, I. Futó, I. Durand, C. Simon, B. Bonhoure
Laboratoire "Langages et Systèmes Informatiques"
Université Paul Sabatier
118, route de Narbonne - 31062 Toulouse Cedex - France

ABSTRACT

This paper summarizes the principal results of the simulation of the COALA (Calculateur Orienté Acteurs pour la Logique et ses Applications) machine : an Actor-Oriented Computer for Logic and its Applications.

The simulation was supported by two basic software systems : a functional simulator simulating the functional aspects of the distributed interpreter written in PASCAL and a system simulator written in T-PROLOG, a simulation language based on PROLOG.

The system simulator enabled us to obtain the duration of execution of PROLOG programs executed on the COALA machine and to compare the effects of various numbers of processing elements and various network topologies on the overall performance of COALA.

1 - THE AND/OR CONNECTION GRAPH MODEL

Several execution models for parallel inference processing have been proposed such as data-flow models [AMA 84], AND-OR process models [CON 85] and reduction models [ONA 85]. In models which handle AND-parallelism, the consistency of solutions poses certain problems : most of these models rely upon the sequential AND/OR tree and handle the tree in a parallel fashion. A constraining and difficult process management technique is required to synchronize processes associated with the nodes of the AND/OR tree. As a result of this process management, a hierarchical tree structure, where process communication is based on data sharing, is necessary.

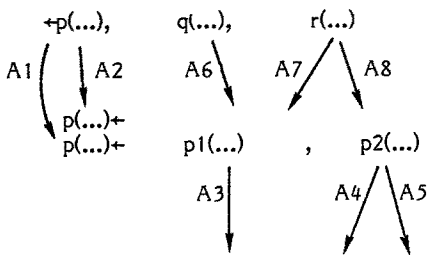
Inspired by R. Kowalski's connection graph [KOW 79], our model represents a new perspective on the parallel interpretation of PROLOG. This connection graph includes one arc for each pair of matching atoms on opposite sides of the implication symbol in different clauses. Associated with each arc is the resolvent obtained by resolving the atom connected by the arc.

In the sequential model [KOW 79], the selection of an arc connected to a goal and

the generation of the associated resolvent form together what is called a resolution step. In such a step, the selected arc is deleted and new arcs are added connecting atoms of the resolvent to the rest of the graph.

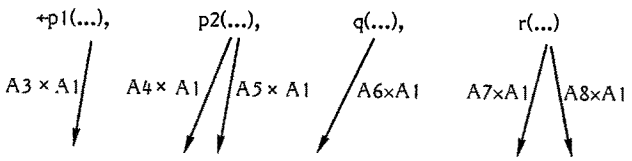
Clearly, in a parallel model [PER 86], both AND and OR parallelisms are related to the simultaneous treatment of several arcs : OR-parallelism is obtained by the parallel selection and resolution of the arcs descending from the same literal ; AND-parallelism arises from the simultaneous unification on the arcs descending from all the literals of the clauses connected by the arc chosen for the resolution.

Example : Let us consider the following graph :

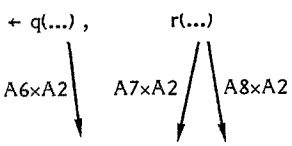


If the leftmost literal p is chosen for the resolution, arcs A1 and A2 are activated and each one produces a resolvent. For the resolvent of arc A1, the new arcs are built from arcs A3, A4, A5, A6, A7 and A8 ; for arc A2, from arcs A6, A7 and A8.

<u>Resolvent of arc A1</u> :



<u>Resolvent of arc A2</u> :

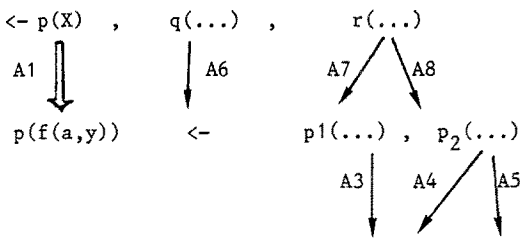


Two levels of parallelism can be observed : the first one appears when building different resolvents and the second one is associated with the creation of different arcs for each resolvent. These two levels correspond respectively to OR and AND parallelisms ; however, in our model, they are processed in the same way.

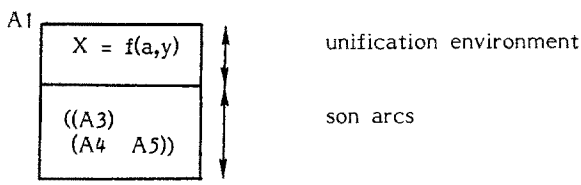
A precompilation of the source PROLOG program generates the initial connection

graph. To represent the graph, we have simply to represent each of its arcs. An arc connects two parent clauses, the "origin clause" and the "extremity clause", and knows its unification environment and the list of the arcs descending from the extremity clause called "son arcs". Arcs descending from other literals in the origin clause, called "brother arcs", will be communicated to the literal chosen for the resolution, when latter is selected.

Thus, in the graph :



<- p(X), q(...), r(...) is the origin clause, p(f(a,y)) <- p1(...), p2(...) is the extremity one ; A3, A4 and A5 are the son arcs and A6, A7 and A8 are the brother arcs. Arc A1 is implemented by the following structure :



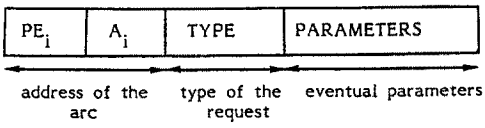
In this very simple data structure, it is no longer necessary to store the literal names.

2 - THE COALA MULTIPROCESSOR ORGANIZATION

Each processor possesses one part of the graph in a private local memory called "Graph Memory". Globally, the memory can be viewed as a set of local memories [KEL 79, MAG 80]. An arc of the graph can reference other distant arcs. This reference must be a tuple of the form (PE, address) where "PE" is the processing element whose memory contains the referenced arc and "address" is the address of the arc in that local memory. This solution suppresses the bottleneck due to a shared memory [HEW 84] where contention effectively swamps the benefits obtained from parallelism.

In the same way, the task queue from which processors seek eventual processing is distributed among the processing elements, like the graph. This avoids the bottleneck encountered when using a centralized task queue. If a processor needs to access an arc residing in another memory, it forms a request packet containing the address of the arc,

the type of operation sought, and various parameters if required [KEL 79]. The general structure of a request packet is given by the following figure :



An incoming message is interpreted by the processing element specified in the address of the arc. The dialogue between processing elements is taken into account by switches which deal with the routing of packets.

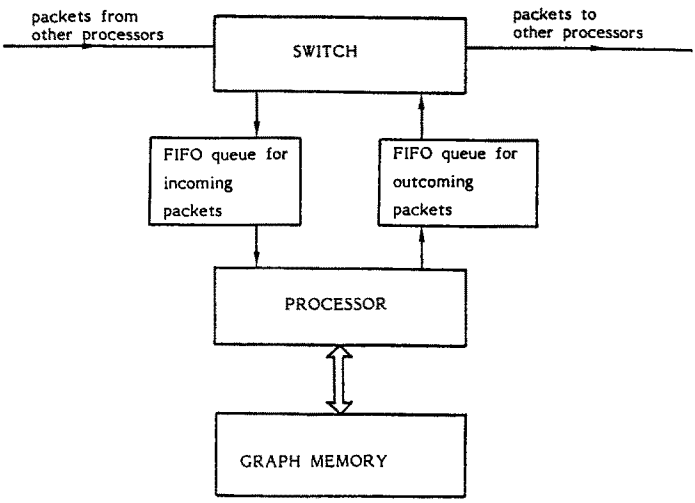


Figure 1 - General organization of the COALA machine

Conceptually, the topology of the communication network can be arbitrary, but must obviously ensure the reliable transmission of messages from one processor to another.

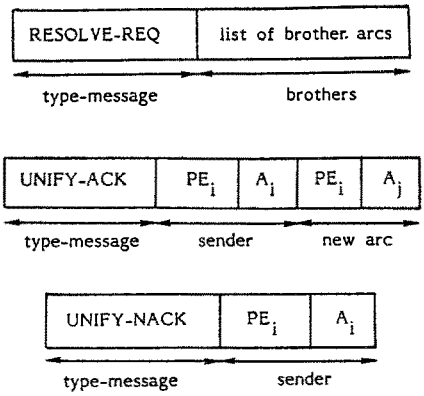
3 - COMMUNICATION BETWEEN PROCESSING ELEMENTS

The messages of the model are directly related to the basic operations of a PROLOG interpreter : resolution and unification.

3.1 - Resolution

At every step of the resolution, a literal is selected for resolution. A RESOLVE-REQ request is sent to each arc descending from this literal. This request contains the identification of all brother arcs of the selected literal, so as the ensure the upkeep of the graph. The arc receiving this request sends its unification environment to each of its sons and brothers through UNIFY-REQ requests. The replies corresponding to the UNIFY-REQ

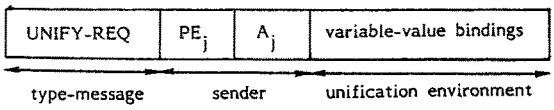
requests are either UNIFY-ACK in case of success, or UNIFY-NACK in case of failure.



When a resolvent is successfully determined, the resolution proceeds with the selection of a new literal to be resolved. The resolution of an arc having neither brother nor son corresponds to the empty clause. Thus an arc's environment is a solution to the initial problem. The arcs created for an unsuccessful resolvent are garbage collected by use of a third request type called DESTROY-REQ request.

3.2 - Unification

The UNIFY-REQ request triggers the unification between two environments : the environment in the request and the one of the arc to which the request is addressed.



By use of a reference counter and the BROTHERS-REQ message, an arc is garbage collected once it has performed all the unifications requested by other arcs. Thus, all the arcs of the graph are dynamically destroyed.

4 - A STRONGLY COUPLED SIMULATION METHOD

The first step of the simulation process was the realization of a functional simulator which simulates the functional aspects of the distributed interpreter. During this step, neither the impact of inter-processor communication nor network topology were taken into account. Written in PASCAL, the functional simulator executes program statements sequentially. This simulator enables one to obtain the duration of execution of messages on a MC68000 (10 MHz) microprocessor. The duration of execution of a message is the number of MC68000 cycles required to handle the message, where the base cycle time is 10^{-7} s.

In order to validate the COALA machine, it was necessary to compare the impact of

different numbers of processors (supposed here to be MC68000) and different network topologies on the machine's overall performance. Thus, the second step of the simulation process was the production of a system simulator. Written in T-PROLOG [FUT 84], a simulation language based on PROLOG, the system simulator assigns a process to each of the processors and executes the PROLOG program taking into account the duration of execution of messages, the state of the communication channels (busy/free) and the state of the processors (busy/free).

The T-PROLOG primitives such as new (create a process), send (send a message), wait-for (wait for a message) and hold (suspend a process) are used for communicating messages between processors and to simulate the elapsed time.

Finally, the general scheme of the simulation of COALA is given in Figure 2. Both simulators are running on a VAX 11/780. The functional simulator, in addition, runs on an SM-90 computer for MC68000 measurements.

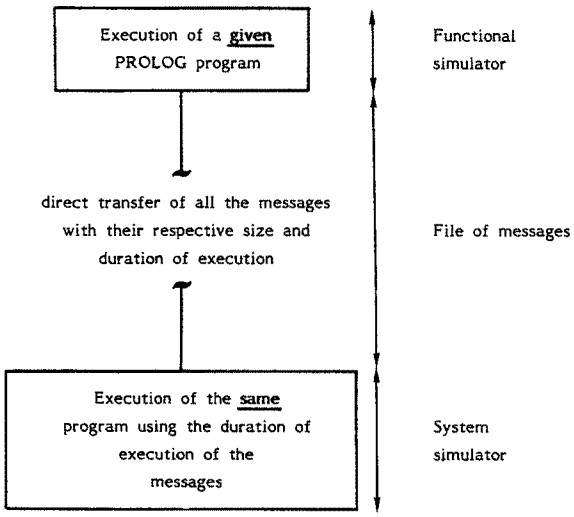


Figure 2 - The COALA system simulation

Contrary to most existing simulations which use a loosely coupled method [ONA 85], our simulation used a strongly coupled simulation method [FUT 86] because it was noticed that the transfer of statistical results from the functional simulator to the system simulator introduced a certain bias in our measurements. To illustrate this particularity, Table 1 gives a few execution time values, in µs, corresponding to the RESOLVE-REQ and UNIFY-REQ messages.

Measurements show that the average processing time for a message is about 1000 µs. The longest message is 170 bytes. Supposing a 10 Mbps communication channel, the transfer time of the longest message is 136 µs. This means that the transfer time cannot be neglected.

message / program	RESOLVE-REQ			max. / min.	UNIFY-REQ			max. / min.
	min.	max.	ave.		min.	max.	ave.	
4-QUEENS	239	7761	2209	32	1198	7330	3006	6
COLOR	2661	3288	1236	1.2	1037	3288	1977	3
REVERSE	769	3989	2535	5	1239	5517	3337	4

Table 1 - Variation of processing times

Finally, the simulation process considers that the network is not ideal (i.e. conflicts may occur), that the communication channel has a 10 Mbps transfer rate with full-duplex links and that each processor has a buffer of sufficient size. The results of this strongly coupled simulation method describe quite exactly the behaviour of the COALA machine using MC68000 and using the same PASCAL procedures as the distributed interpreter. As for the network topology, we have envisaged the completely connected, near-neighbour mesh, cube and hypercube networks with 4, 9, 16 and 25 processors.

For purposes of comparison, we introduced a program operating on large data structures such as lists, n-uplets and compound terms (4-queens) and a program operating on simple data structures such as constants and variables (pc database). Due to the difference in data structure handling, the duration of execution of these programs is largely dependent on these data structures. For example, in the 4-queens program (used in general to measure multiprocessor PROLOG architectures), the clause partitions are small but the search space becomes large because of the recursive use of the clauses. The corresponding PROLOG programs are given in detail in the appendix.

5 - SIMULATION RESULTS

Firstly, we chose a completely connected network in order to define the number of processors of the COALA machine. This network is ideal for our distributed interpreter but the number of switching elements required for a large number of processors renders the topology unrealizable. Figure 3 shows the overall performance of COALA for the 4-queens problem when increasing the number of processors.

As additional processors are used (up to 18), duration of execution decreases. Then it levels off : between 9 and 18 processors, execution time is nearly the same. However, this result shows that the COALA machine is able to support the parallelism in PROLOG programs.

As for the pc database program, the future behaviour of our machine has been simulated by increasing the number of assertions. Thus, pc database (n) indicates the pc database program with n assertions for two partitions. For each program, measurements show that duration of execution decreases and that saturation occurs. However, this saturation is

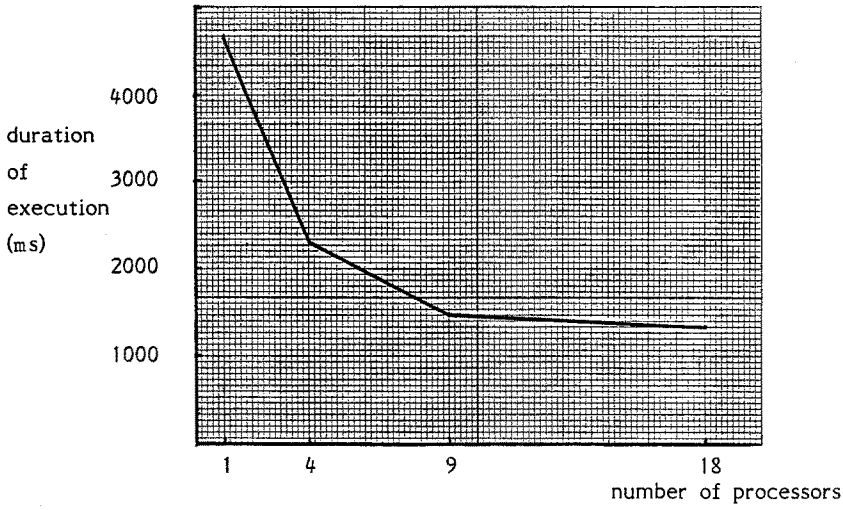


Figure 3 - Effect of the number of processors for the 4-queens problem

only bound to the application size and we assume that the number of processors does not entail a bottleneck situation in the performance of COALA. Figure 4 resumes this property.

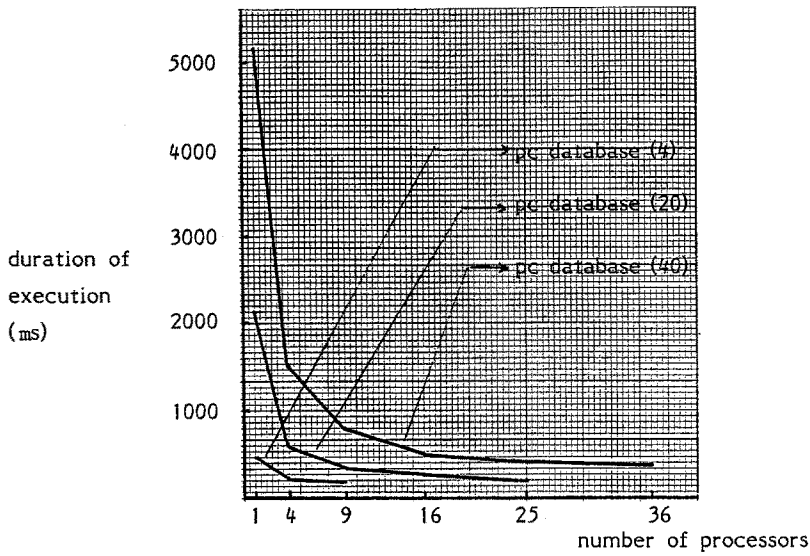


Figure 4 - Effect of the number of processors for the pc database problem

Some topologies have been simulated taking communication time into account. In addition to duration of execution, many other parameters have been used to define the topology of COALA. In particular, among these parameters is the diameter i.e. the longest path between two nodes in the network. Figure 5 shows the simulation results for the pc

database (20) problem.

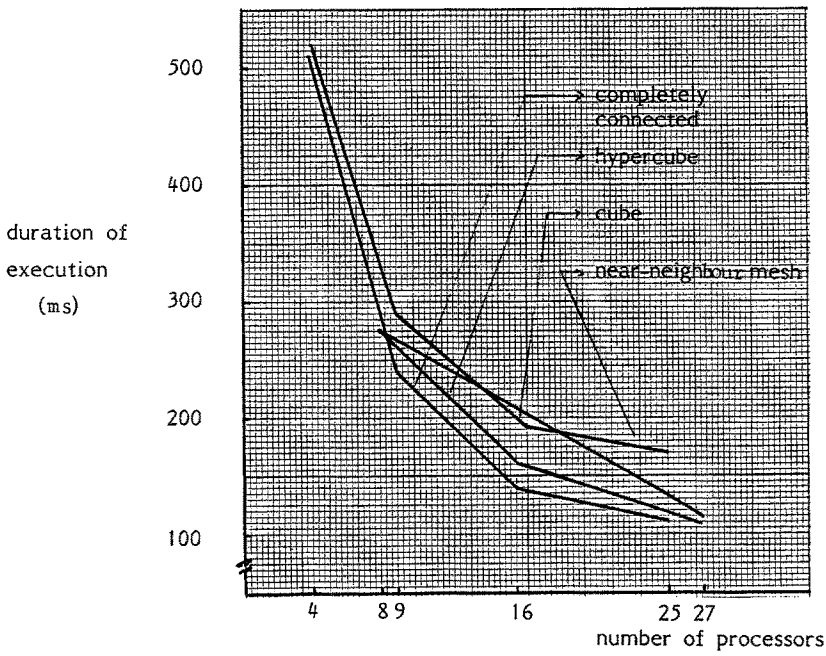


Figure 5 - Effect of the topology for the pc database (20) problem

With respect to these results illustrated in Figure 5, we propose the hypercube as the basic topology of the COALA machine. With the hypercube, messages traverse several numbers of channels and so exhibit different latencies but results are nearly the same as a completely connected topology. The degree i.e. the number of switches associated with a node permits an implementation with hundreds of processors.

When executing the pc database (20) program on the hypercube with 16 processors, only 2.2 channels are traversed on average. In the same conditions, the load average of each communication channel with respect to duration of execution is about 2 per cent and the maximum input FIFO queue is 70 messages. Table 2 relates in detail these results.

processor	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
total	188	188	188	188	188	186	186	186	186	170	186	180	120	147	147	147
max.	63	70	68	65	63	55	58	56	55	67	61	50	38	42	46	37
ave.	18	20.3	19.4	18.9	17.7	16.1	17.7	15.7	15.2	17.2	16.5	14	7.4	7.3	7.9	6.4

Table 2 - Distribution of messages in FIFO queues

These results correlate with an acceptable FIFO queue load balancing. In the same way, processor load balancing does not need dynamic process allocation as in other approachs [KEL 84]. In our model, the distribution of the arcs is equivalent to the distribution of the tasks among the processors. Figure 6 shows processor load balancing for the pc database (20) problem using 9 processors.

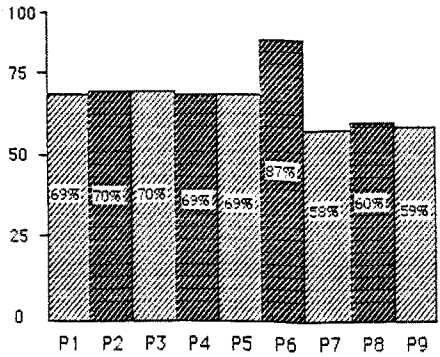


Figure 6 - Processor load balancing for the pc database (20) problem

Unfortunately, for programs operating on large data structures, results are not as good as for programs operating on simple data structures. This behaviour is related to the small partitions of clauses and to the use of recursivity. Figure 7 shows processor load balancing for the 4-queens problem using 9 processors. Results are similar with programs such as quicksort, reverse ...

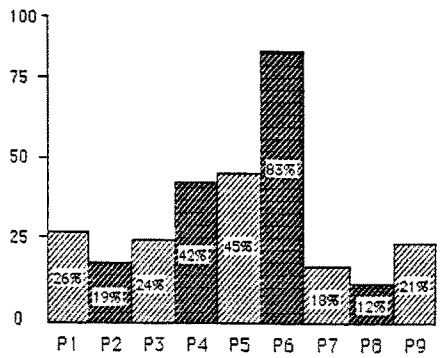


Figure 7 - Processor load balancing for the 4-queens problem

We are currently investigating various mechanisms intented to improve the model and to define a knowledge base machine [MUR 84]. In particular we seek to take advantage of dynamic results of programs operating on simple data structures.

When a PROLOG program uses a large number of assertions, the basic model builds

lists of sons and therefore lists of brothers which are for too lengthy Consequently, the processing time required to define the next resolvent is proportional to the number of UNIFY-ACK and UNIFY-NACK messages waited for by this resolvent. It is quite easy to modify the original representation by representing all the arcs of the same partition stored on the same processor by an extra-arc containing the list of real addresses of the original arcs.

When such an arc receives a UNIFY-REQ request, it distributes this request among its arcs. The algorithm used in treating the message is the same as before, except that no message is sent directly. Instead, the addresses of the newly created arcs update the old ones. If the new list of addresses contains at least one element, a UNIFY-ACK message is sent to the resolvent ; if the list is empty, a UNIFY-NACK message is sent.

An advantage of this representation is that the list of sons of an arc contains only one arc per processing element instead of all the arcs defining the literal. This property greatly reduces the number of messages as shown in Figure 8.

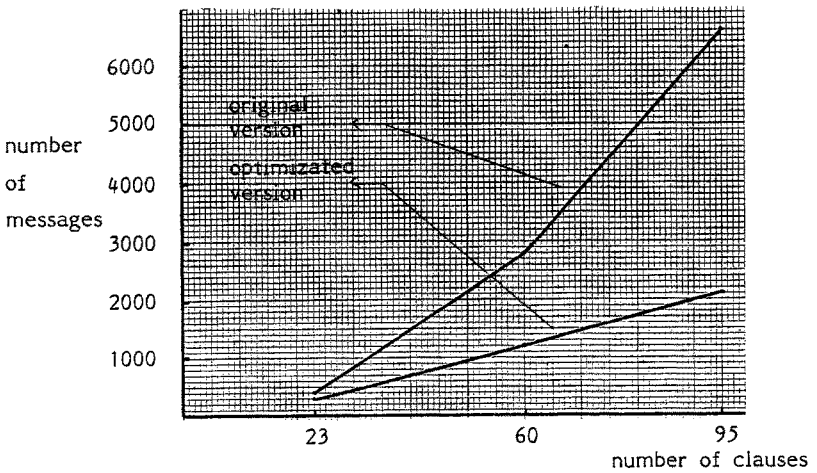


Figure 8 - Effect of extra-arcs for the pc database problem

The number of messages decreases substantially as the number of clauses increases. Consequently, processing time required to define the next resolvent decreases and is about 12 times smaller than in the original version when 95 clauses.

CONCLUSION

The AND/OR connection graph model represents a new perspective on the parallel interpretation of PROLOG. The graph is distributed, without duplication, among the processing elements ; consequently, accessing the graph entails no bottleneck situation . In particular, control is not based on a hierarchical structure.

The simulation system allows us to obtain dynamic measurements. Simulation conditions closely reflect the behaviour of the distributed interpreter running on the future COALA machine with MC68000 microprocessors as processing elements. A strongly coupled simulation method was used in order to simulate large PROLOG programs and large network topologies. This method does not introduce any approximation for measurements.

It appears that an acceptable performance may be achieved using somewhere near 100 processors. The hypercube topology seems to be the best compromise for the COALA network. Each processing element is connected through bidirectional full-duplex, point-to-point communication channels to its near-neighbours forming a small diameter network.

An important result is that programs operating on simple data structures do not require dynamic process allocation. The architecture is well-suited for knowledge base applications. For this purpose, we have defined mechanisms to improve efficiency of the AND/OR connection graph model and, in the future, we will investigate the adequacy of a knowledge base architecture to the COALA machine. For programs operating on large data structures, a better distribution of tasks among the processing elements must be define.

ACKNOWLEDGEMENTS

This work has been realized within Professor R. Beaufils's team and is supported by the "Gréco de Programmation" of the "Centre National de la Recherche Scientifique".

REFERENCES

[AMA 84] M. Amamiya, R. Hasegawa - A Logic Program Execution based on Data-Flow Control - Proc. Logic Programming Conference 84, November 1984, pp. 507-516.

[CON 85] J.S. Conery, D.F. Kibler - AND Parallelism and Nondeterminism in Logic Programs - New Generation Computing - 1985, Vol. 3, pp. 43-70.

[FUT 84] I. Futó, J. Szeredi - System Simulation and Co-operative Problem-Solving on a PROLOG basis - Implementations of PROLOG - ed. J. Campbell, Ellis Hortwood Ltd., England - 1984, pp. 164-174.

[FUT 86] I.Futó, C. Percebois, I. Durand, C. Simon, B. Bonhoure - Simulation Study of a Multiprocessor PROLOG Architecture - First Italian Conference on Logic Programming, Genova, Italy, March 1986 (invited paper).

[HEW 84] C. Hewitt, H. Lieberman - Design Issues in Parallel Architectures for Artificial Intelligence - COMPCON Spring 1984, February 1984, pp. 418-423.

[KEL 79] R.M. Keller, G. Lindstrom, S. Patil - A Loosely Coupled Applicative Multi-Processing System - AFIPS Press, 1979, pp. 613-622.

[KEL 84] R.M. Keller, F.C.H. Lin, J. Tanaka - Rediflow Multiprocessing - COMPCON Spring 1984, February 1984, pp. 410-417.

[KOW 79] R. Kowalski - Logic for Problem Solving - The Computer Science Library, Elsevier, 1979.

[MUR 84] K. Murakami, T. Kakuta, R. Onai - Architecture and Hardware Systems : Parallel Inference Machine and Knowledge Base Machine - Proc. International Conference on Fifth Generation Computer Systems 1984, November 1984, pp. 18-36.

[ONA 85] R. Onai, M. Aso, M. Shimizu, K. Masuda - Architecture of a Reduction-Based Parallel Machine : PIM-R - New Generation Computing - 1985, Vol. 3, pp. 197-228.

[PER 86] C. Percebois, I. Futó, I. Durand, C. Simon, B. Bonhoure - An Actor-Oriented Multiprocessor Architecture for PROLOG : COALA - First Italian Conference on Logic Programming, Genova, Italy, March 1986 (invited paper).

APPENDIX

4-queens program

```
queens(nil,Y,Y).
queens(c(X1,X2),Y,Z) :- select(U,c(X1,X2),V),
                        safe(U,Y,1),queens(V,c(U,Y),Z).
select(X,c(X,Y),Y).
select(U,c(X,Y),c(X,V)) :- select(U,Y,V).
safe(U,nil,W).
safe(U,c(P,Q),N) :- nodiag(U,P,N),plus(N,1,M),safe(U,Q,M).
nodiag(U,P,N) :- plus(P,N,T1),moins(P,N,T2),diff(U,T1),diff(U,T2).

:- queens(c(1,c(2,c(3,c(4,nil)))),nil,Q).
```

pc database (40) program

```
ibm_comp(PNAME,CPU,OS):-pnps(PN,PNAME,X,Y),pco(PN,CPU,OS),
                        bit_16(CPU),os_simil(OS).

bit_16(8086).
bit_16(8088).
bit_16(m8088).
bit_16(i8088_2).
bit_16(82286).
bit_16(82086).

os_simil(ms_dos).
os_simil(dos_10).
os_simil(z_dos).
os_simil(dos_211).
os_simil(dos_20).
os_simil(msdos_21).
os_simil(dos_30).
os_simil(prop_30).
os_simil(prop_20).
os_simil(dos_11).
```

```
pnps(p1,agil,3000,s1).
pnps(p2,corona_p,2700,s2).
pnps(p3,corona_p,2500,s3).
pnps(p4,mbc_555,1100,s4).
pnps(p5,zenith_1,3000,s6).
pnps(p6,genie_16,2000,s6).
pnps(p7,rainbow,2500,s7).
pnps(p8,osborne,2000,s6).
pnps(p9,dasher_1,3400,s8).
pnps(p10,z_158,5100,s5).
pnps(p11,turbo_pc,4000,s11).
pnps(p12,kay_16,2500,s10).
pnps(p13,copam_pc,2000,s11).
pnps(p14,don_pc,2000,s12).
pnps(p15,dyn_xt,2000,s13).
pnps(p16,laser_pc,2000,s14).
pnps(p17,ap_pc,2600,s15).
pnps(p18,m24_sp,3000,s16).
pnps(p19,micral,6000,s17).
pnps(p20,prop_16,4000,s18).
pnps(p21,com_64,500,s19).
pnps(p22,proper_8,2000,s18).
pnps(p23,vt_16,4000,s20).
pnps(p24,ncr_pc,3000,s21).
pnps(p25,oplite,3000,s22).
pnps(p26,pap_c,2500,s23).
pnps(p27,p_1600,2700,s24).
pnps(p28,ph_p3100,3000,s25).
pnps(p29,s_9001,2500,s4).
pnps(p30,sil_216,4000,s26).
pnps(p31,s_pc50,6500,s27).
pnps(p32,tandy,1500,s28).
pnps(p33,tele_pc,4000,s29).
pnps(p34,ti_pc,2500,s30).
pnps(p35,wang_pc,3000,s31).
pnps(p36,axel_20,2000,s32).
pnps(p37,caf_bip,2200,s32).
pnps(p38,ericsson,2800,s34).
pnps(p39,logical,6000,s35).

:- ibm_comp(PNAME,CPU,OS).

pco(p1,8088,ms_dos).
pco(p2,8088,ms_dos).
pco(p3,8088,ms_dos).
pco(p4,8086,dos_11).
pco(p5,z80,cp_m).
pco(p5,8086,z_dos).
pco(p6,8088,ms_dos).
pco(p7,8088,cp_m_86).
pco(p7,z80,cp_m).
pco(p8,z80,cp_m).
pco(p9,8088,dos_211).
pco(p10,8088,ms_dos).
pco(p11,i8088_2,ms_dos).
pco(p12,82286,dos_20).
pco(p13,8086,dos_20).
pco(p14,8088,ms_dos).
pco(p15,8086,dos_20).
pco(p16,8088,dos_20).
pco(p17,8086,msdos_21).
pco(p18,8086,ms_dos).
pco(p19,82086,dos_30).
pco(p20,8088,prop_30).
pco(p21,z80,cp_m).
pco(p22,z80,cp_m).
pco(p23,m8088,prop_20).
pco(p24,8088,ms_dos).
pco(p25,8088,ms_dos).
pco(p26,8088,ms_dos).
pco(p27,8088,ms_dos).
pco(p28,8088,ms_dos).
pco(p29,8088,ms_dos).
pco(p30,8088,ms_dos).
pco(p31,8088,ms_dos).
pco(p32,8088,ms_dos).
pco(p33,8088,ms_dos).
pco(p34,8088,ms_dos).
pco(p35,8088,ms_dos).
pco(p38,8088,ms_dos).
pco(p39,8088,ms_dos).
```