PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

A New Parallelization Method for Production Systems

E. Bahr, F. Barachini, H. Mistelberger

Alcatel Austria–ELIN Research Center
Ruthnergasse 1–7, A–1210 Vienna, Austria

Abstract

The growing importance of expert systems in real–time applications reveals the necessity of reducing response times. Since uniprocessor optimizations of production systems have widely been explored, only multiple processor architectures appear to provide further performance gain. Efficient exploitation of the inherent parallelism of production systems, however, requires suitable algorithms for load balancing without simultaneously increasing organization or communication overhead. We present a novel parallel algorithm for PAMELA expert systems, based on dynamic distribution of data processing. The concept is supported by a transputer based architecture with an advanced interconnection structure.[1]

1 Introduction

PAMELA (PAttern Matching Expert system LAnguage) [Barachini and Theuretzbacher 1988, Barachini 1988] was originally designed as a high performance rule–based expert system language especially suited to treat real–time problems. PAMELA's inference engine is highly optimized and makes the language one of the most efficient platforms for rule–based systems on uniprocessors. Nevertheless, the computational complexity of rule–based programs leads to considerable response times. Significant additional speed–ups are expected from parallel execution of the inference engine.

Parallel PAMELA (P^2AMELA) uses a parallel matching scheme not restricted to a specific matching algorithm. The matches are performed concurrently on a number of identical processing elements, requiring only little communication. This is achieved by means of a special scheduling algorithm. The parallelization algorithm is able to incorporate all optimization techniques of the serial PAMELA version.

A transputer based architecture, the "Parallel PAMELA Research Engine" (PRE), has been developed to support the needs of the parallel version of PAMELA. PRE uses a personal computer as master processor, with a multicast interface from the PC to the processing elements [Kasparec *et al.* 1989]. PRE is a research architecture and scalable to 32 transputers. This limitation is not due to the parallelization algorithm but arises from intended cost and complexity restrictions for the hardware architecture. Moreover, it is well known from the literature that the inherent parallelism in typical present–day production systems does not allow speed–up factors of more than 20. Hence, the number of 32 transputers is no obstacle for performing significant run–time experiments.

We discuss in detail the mapping of the fine–grain algorithm onto the coarse–grain PRE architecture. Preliminary performance data of a few hand–coded examples show the efficiency of our algorithm in exploiting inherent parallelism. These experiences serve as a motivation for a full implementation of a parallelizing production system compiler, which is in the final stage of development.

2 Production Systems

A (forward chaining) production system (PS) consists of a production memory containing rules, and a working memory (WM) containing data (working memory elements, WMEs) representing the system state. Real–time production systems are able to communicate with the outside world, e.g. for sampling data or for sending messages to another system.

A rule resembles the well–known IF...THEN... statement. It consists of a left hand side (LHS, corresponding to the IF–part) and a right hand side (RHS, corresponding to the THEN–part). The PS execution breaks into a sequence of "recognize–act cycles" (RACs). A single RAC consists of the following steps:

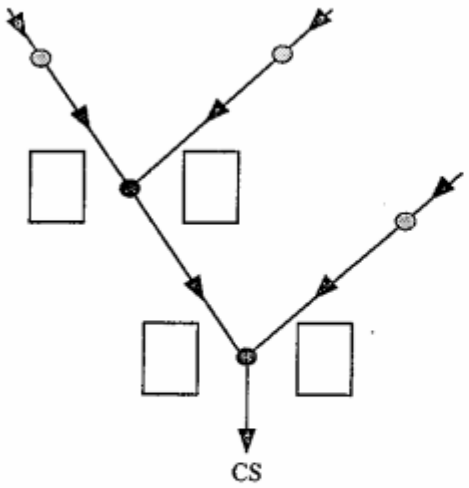
- During the "match phase", the LHSs satisfied by the WMEs are determined. For each valid rule a corresponding instantiation enters the "conflict set" (CS).
- During "conflict set resolution" (CSR) one of the rule instantiations in the CS is selected.
- During the "act phase" the RHS statements of the selected rule are executed. These statements usually change WM or initiate communication with the outside world.

1 This research is sponsored by the Austrian Innovations– und Technologiefonds as part of the InFACT project.

3 The Match Algorithm of Sequential Pamela

The RETE [Forgy 1979, Forgy 1982] and the TREAT [Miranker 1987] algorithm are the best known state saving algorithms, which avoid recomputations of comparisons done in previous RACs. Both algorithms map the patterns of the LHSs of the rules to nodes of a network. The inference engine of PAMELA uses a modified version [Barachini and Theuretzbacher 1988] of the RETE algorithm. Since we have chosen the RETE also for the implementation of our parallelization method, we sketch the basic mechanisms within a RETE network.

When a WME is added to or removed from the WM, a plus–token resp. a minus–token representing this action is passed to the RETE network. In one–input nodes (1INs) attributes of the incoming token are compared against constant values. Two–input nodes (2INs) have a token memory for each input. An incoming plus–token is stored in the token memory, a minus–token removes the corresponding 2IN token from the memory. In 2INs attributes of each incoming token will be compared against attributes of all tokens in the opposite token memory, according to the conditions in the LHS. On each (successful) match, a new token is generated and is sent to the successor node. If a token leaves the RETE network a rule instantiation enters the CS. Figure–1 shows a RETE network with three 1INs and two 2INs.



Figure–1: RETE network of a rule with three patterns

4 Parallelization of Production Systems

Before discussing parallelization, it seems appropriate first to distinguish several classes of parallel architectures. In the familiar Flynn taxonomy [Flynn 1972], SIMD (single instruction, multiple data), MISD (multiple instruction, single data), and MIMD (multiple instruction, multiple data) architectures constitute the variety of parallel architectures. Although there have been attempts to implement production systems on SIMD machines [Forgy 1980], MIMD architectures obviously better match the needs of production system algorithms. Parallel (distributed) systems of the MIMD class fall into two categories [Bhuyan 1987], multiprocessors (all processors share main memory) and multicomputers (each processor has its own local memory with its local address space, a processor cannot directly access another processor's local memory. Communication is accomplished via message–passing).

Two performance measures are of particular interest in evaluating parallel systems, speed–up (defined as the ratio of the execution times for one and for n processors) and efficiency (defined as speed–up divided by the number of processors) [Eager *et al.* 1989]. Efficiency depends on the ratio of communication and computation. Limiting factors are memory contention (with multiprocessors) and the communication overhead (with multicomputers), respectively.

Soon after the invention of the state saving algorithms, various investigations have been started on parallel architectures for production systems. There are several levels of parallelism inherent to production system algorithms like the one used in PAMELA. Apart from application parallelism (concurrent execution of loosely coupled production system tasks), there exists match parallelism on rule, inter–node, and intra–node level, act parallelism, and CSR parallelism [Gupta 1986]. The usefulness of exploiting a particular type of parallelism depends on the time spent for each phase. Typical numbers for RETE production systems are: match (up to) 90%, act 5%, and CSR 5% [Forgy 1979, Gupta 1986]. Most investigations therefore have concentrated on concurrent execution of the match phase. However, newer studies have shown[2] that some production systems spend considerably less than 90% in the match phase. With rule level parallelization, for the time of one RAC, each rule is assigned to a different processing element (PE). With inter–node level parallelization, each node of the RETE–network is assigned to a particular PE, whereas with intra–node level parallelization comparisons within a node are assigned to different PEs.

So far, none of the implementations of these ideas [Butler *et al.* 1988, Gupta 1986, Gupta and Tambe 1988, Kelly and Sevoria 1987, Miranker 1984, Oshisanwo and Dasiewicz 1985, Schreiner and Zimmermann 1987, Shaw 1987, Stolfo 1984, Tenorio 1984, Tien and Raghavendra 1987] has been

2 Private communication with Daniel Miranker

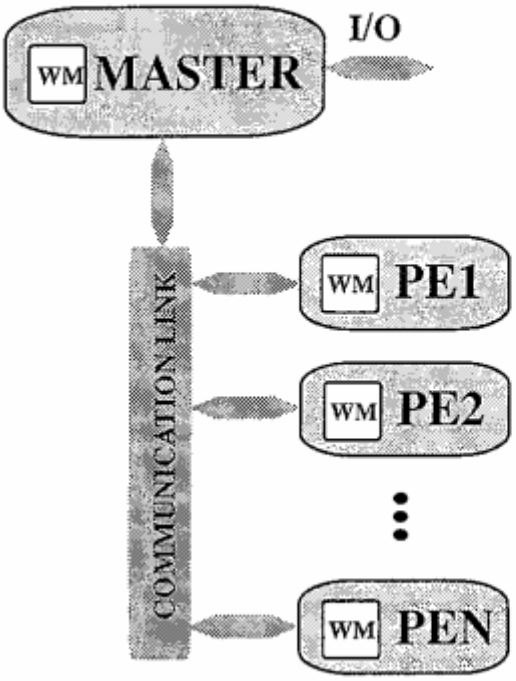
able to simultaneously cope with bottle–necks due to communication overhead or due to shared resources, and load balancing problems. The approach presented in this paper is placed among the intra node parallelizations, but avoids the above–mentioned problems. The algorithm also exploits parallelization of the CSR and is not restricted to RETE but can be applied to TREAT as well.

5 The Basic Idea of Independent Match Parallelization

Anticipating the very simple overall structure of the architecture (figure–2) we can sketch the steps of a RAC in Parallel PAMELA:

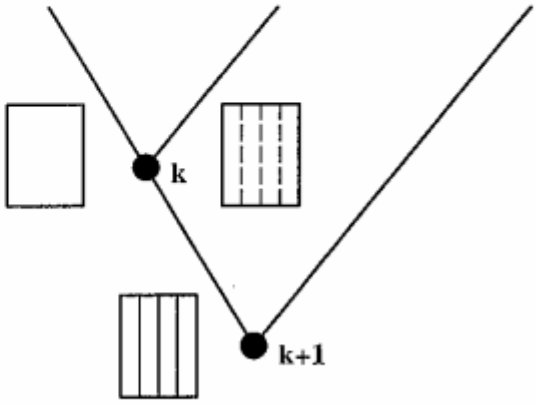
- During the match phase the comparisons are assigned to the PEs by a scheduling algorithm (without inter–PE communication).
- Each PE performs its local CSR (which means also a parallelization of the CSR) and sends its candidate rule instantiation to the "master" processor.
- The master selects one of these candidates (global CSR), executes the RHS of the corresponding rule, and sends the WME changes back to the PEs.

At the beginning of an RAC, each PE therefore must be able to decide independently which partition of the expected comparisons it intends to perform. This decision is made *dynamically during run–time* by a special scheduler running on each PE.



Figure–2: Hardware architecture

In order to illustrate the idea of the independent match parallelism, we consider a 2IN of a RETE–network (figure–3). It is assumed that both token memories of node k are known to all PEs. Each PE has physical copies of these memories and in this sense they are global. These memories have been independently generated from the WM, which is also global – ie. there is a copy of the WM on each PE. The task to be carried out is to compare each token of the left to-

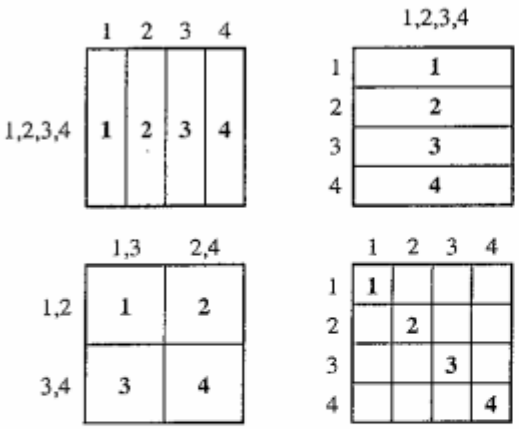


Figure–3: Partitioning of comparisons

ken memory with each token of the right memory, according to the comparison prescription of the 2IN.

In order to partition the comparison among the PEs, either the left or the right memory is divided into a number of blocks, equal to the number of PEs (in our example we assume 4 PEs). This partitioning is only "virtual" in the sense that both memories are still global. This is indicated by the dashed lines in figure–3. The partition just means that during the match phase in node k, the m–th PE takes the tokens in the m–th block and compares them against all tokens in the opposite memory. In this way, all comparisons in node k are performed by the PEs $m = 1,...,N$. But the run–time is reduced by a factor $1/N$, provided that the partitioned memory contains enough tokens. Each PE generates its own tokens corresponding to its successful matches. This leads to disjoint parts of the left memory at node $k+1$. These parts are local to each PE, ie. part m is only known to PE m (indicated by the solid lines in figure–3). The matches in the subsequent nodes performed by PE m can only be done with its local data. One can easily see that the conjunction of all comparisons gives the whole set of comparisons of the uniprocessor version. This is a necessary condition for consistency.

We demonstrate the consistency of the partitioning algorithm by an example with four PEs, which should perform all matches in a two–input node. The input memories of this node are assumed to be global. The left memory is represented by the vertical axes of the squares in figure–4 and the right memory by the horizontal axes. Then we have three possibilities to distribute the matches between the PEs: (virtual) partitioning of either the left token memory (first square



Figure–4: Consistency considerations

in figure–4) or the right token memory into four parts (second square), or partitioning of both memories into two parts with a suitable assignment of tokens to PEs (third square in figure–4). The last square in figure–4 represents an assignment of the tokens violating the consistency since not the whole cross product of matches is performed. This restriction can be easily taken care of by the following procedure. A partitioning number p_k is assigned to the partitioning node k. p_k is the dyadic logarithm of the number of portions into which the matches in node k has been partitioned. Formally we can define $p_k = 0$ for unpartitioned matching in node k. Furthermore, a maximum partitioning number p_{max} is introduced. This number can be calculated by the dyadic logarithm of the number of PEs, which is always a power of two. Then the restriction takes the form $p_k \leq p_{max}$ In general, the left memory in a RETE–node need not be global due to previous partitionings in predecessor nodes. In this case the right memory must not be fully partitioned according to the consistency requirements mentioned above.

Since each PE holds the whole RETE–network the PEs can process the data, assigned by the partitioning algorithm, without communication with other PEs. The matches are performed by using data which is local or global in the logical sense but strictly local with respect to the physical PE. During the match phase all required global data items are *not accessed by communication* with other PEs but are generated from the global WM, which is located on each PE.

This somewhat simplified picture can be applied to all active nodes in the RETE–network and shows three major points of our approach:

- the approach relies on data parallelism in token memory rather than on program parallelism,
- a very fine grained *dynamic* distribution of matches among the PEs leads to good load balancing,
- no communication is necessary during the match phase, since no PE requires data from another PE.

Compared to a *static* assignment of partitioning nodes our method is much more flexible, which is crucial if data load varies over time. This is especially the case for real–time production systems communicating with the outside world.

In order to clarify some open questions, a few remarks should be made. So far, we have only considered comparisons in 2INs and have not included those in 1INs. In principle it is possible to split the token flow already in an 1IN. Since the 1IN matches consume less than 5% of the total match time of a typical production system, we decided to discard this possibility in favour of more flexibility in partitioning later RACs.

For simplicity it has been assumed that the token memories contain a sufficient number of tokens, so that partitioning leads to parts with nearly equal numbers of tokens. In real life this may be a bit too optimistic. Assuming we have 4 PEs and only 3 tokens in the memory of a node, a division into four parts excludes one PE from processing this node and all successor nodes for current path of token flow. Therefore, three is the maximum speed–up factor in this node. Since we need no synchronization point after the execution of a node, the free PE can process another node in the meantime. Nevertheless, it is advisable to partition large memories because this reduces the chance of idle PEs. A special scheduling algorithm is called on each PE at the beginning of a match phase, which estimates in advance the optimum nodes for partitioning.

In reality the matching procedure is more complex since several token packages may enter the RETE–network at different nodes each RAC. The interference between token packages can be easily handled by processing package by package. All matches in the course of the token package's flow through the network are performed before the next package is processed. Furthermore, a token package can be partitioned at several nodes. This is allowed as long as the generalized consistency requirement

$$\sum_{k=first}^{last} \pi_k \leq \pi_{max} \qquad (1)$$

is obeyed. Due to the fact that we can have nodes with partially partitioning, ie. $p_k < p_{max}$, left memories can hold data of different "degree of locality", generated during previous RACs. Such data is known by fixed subsets of PEs. Therefore, if an incoming token package has to be matched against a left memory, the package might branch into token packages of different locality degrees. In all subsequent nodes these token packages have to be processed separately.

When rule instances finally enter the CS it must be guaranteed that the local CSs are disjoint in order not to contain dublicated instances. This is achieved by enforcing condition (1) but with "=" instead of "≤". Since the PEs' conflict sets form disjoint sets most of the CSR is automatically parallelized. Only the CSR for the best candidates received from the PEs is done by the master. CSR parallelism is only

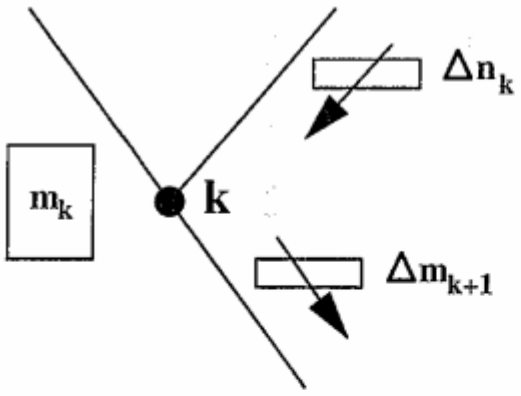
possible when the priority of rule instances do not depend on the existence or priority of other rule instances so that a global view of the CS is necessary. However, for LEX, MEA [Cooper and Wogrin 1988] and many other CSR strategies our algorithm exploits parallelism.

In contrast to rule–level parallelization our algorithm can take full advantage of RETE–network sharing. This is due to the fact that our algorithm relies on a kind of data flow splitting which is implicitely controlled by above mentioned consistency precautions.

6 The Scheduling Algorithm

At the beginning of each RAC, new tokens enter the RETE–network. They are counted and buffered into token packages. These incoming tokens have to be matched against the opposite memories in 2INs and emerging tokens are passed to the successor nodes. The task of the scheduling algorithm is to predict the match activity within the RETE–network for each token package. For this reason, the scheduler needs actual information about the number of entering tokens, size of token memories, and statistical data.

Since the scheduler works independently on each PE, it is only allowed to use globally known data. Otherwise, there would be no guarantee that the schedulers on the PEs arrive at the same results (e.g. decision on the partitioning nodes).



Figure–5: A typical two–input node

Figure–5 shows a typical situation in a 2IN. The number of matches in node k is just the product of the number of incoming tokens and the number of tokens in the opposite memory:

$$V_k = n_k \delta m_k$$

The number of emerging tokens (successful comparisons) can be estimated by

$$\delta m_{k+1} = p_k V_k$$

p_k is the probability that a particular match is successful. p_k itself is estimated by the ratio *(successful matches)/matches* of previously performed comparisons in node k and will be continuously updated.

Having calculated the number of expected matches for a certain entering token package in all relevant nodes of the RETE–network, the scheduler decides upon the optimum partitioning nodes for the considered token package. For this reason, the scheduler searches for the minimum of a special function μ representing a measure for the load balance among the PEs. The argument of this function is the number of the partitioning node *part*. For example, such a function could have the following form:

$$\mu(part) = \sum_{i=first}^{part-1} V_i + \sum_{i=part}^{i=last} \frac{V_i}{F_i}$$

From the entrance node *first* of the considered token package to the node *part–1*, no parallelization takes place. This contribution is represented by the first sum. The partitioning node *part* and all its successor nodes are parallelized. Hence, the number of comparisons per PE is only a fraction of the total number of comparisons in each node, leading to speed–up factors F_i in the second sum. These factors range between *1* and the number N of PEs. $F_i = 1$ means that one PE performs all matches in node i, $F_i = N$ refers to the most balanced parallelization. It can be easily shown that the speed–up factors cannot increase from one node to its successor, ie. $F_i \leq F_j$ for $i > j$. If, for instance, only one comparison has to be performed in node *first* then, obviously, $F_{first} = 1$ and $F_i = 1$ for all subsequent nodes. This kind of unbalanced distribution of comparisons has already been mentioned in the previous section. For the minimum function μ to work sufficiently well, the V_i's must represent the major portion of work to be performed during the matching. If it turns out that insertions into token memories take a significant amount of time, appropriate terms have to be added.

Of course, the scheduling scheme can be generalized to several partitioning nodes for each token package. This is achieved by iterative application of the minimum search, with updated V_ks for $k \geq part$ after each step.

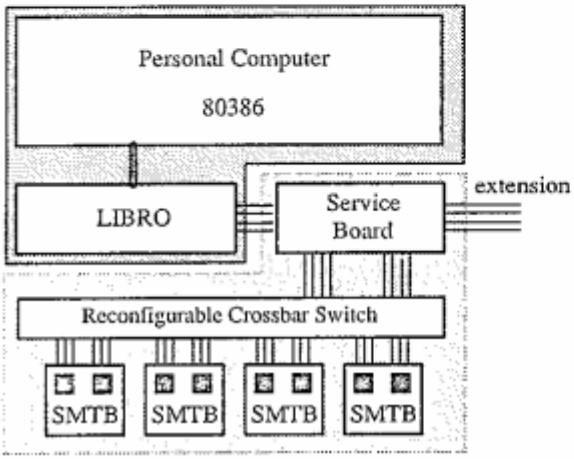
After having determined the partitioning node part, the token package actually enters the RETE–network.

7 The Parallel PAMELA Research Engine

It is a well–known experience that performance of (very expensive) shared–memory multiprocessors degrades at higher n (> 4) due to memory contention. The decision therefore has been made to construct a parallel architecture for PAMELA, offering the scalability of message passing machines as well as tightly coupled pairs of processors (figure–6). This tight coupling will facilitate future experiments using small

shared–memory subclusters constituting the otherwise loosely coupled architecture. The P²AMELA Research Engine (PRE) is a prototype serving for the evaluation of parallelized PAMELA (P²AMELA) expert systems.

In order to allow the PRE fit into a standard environment, an industry–standard 386–based PC was selected for the master processor. This allows the use of standard operating systems and tools (Unix, XWindows) as well as to interface to a variety of networks. The full PAMELA–C expert system shell therefore can be ported to the PRE.

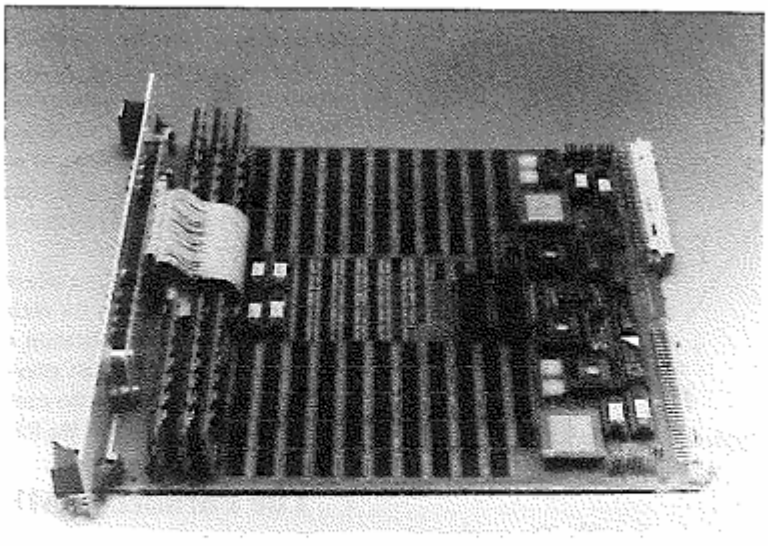


Figure–6: Basic architecture of the PRE

In implementing the PEs, the design is based on the Inmos transputer. In the course of the design, two particular problems were to be solved, namely (i) to effectively update working memory on the PEs, and, (ii) to exchange data between a transputer pair on one PE. Therefore, up to 16 double transputer boards can be served by the PC.

To solve the first problem indicated above, a 'Link Broadcast Interface' (LIBRO) board was developed. The first version of LIBRO was implemented on a PC add–on card. This PC LIBRO is a quadruple transputer link interface for personal computers; up to four boards per PC can be stacked and treated as a single device. The LIBRO solution allows WM contents and other global information to be broadcast through four to sixteen links under control of a master processor (PC). Each link channel is buffered in both directions, so fast access with string primitive instructions is possible.

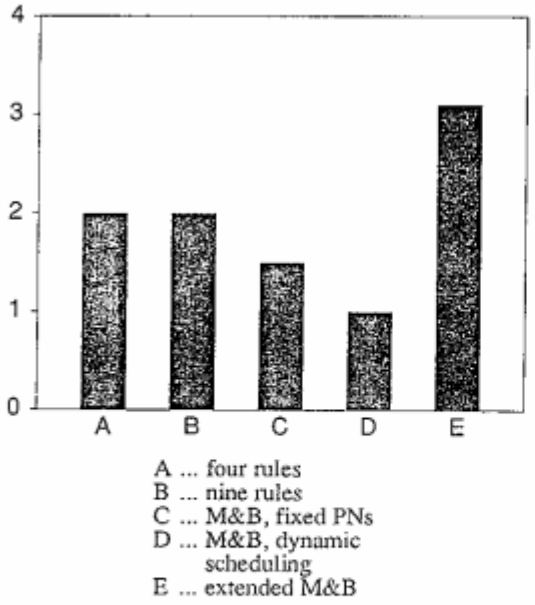
The core of the P²AMELA Research Engine consists of Swapable–Memory Transputer Board (SMTB) modules (figure–7) for the PEs. An SMTB incorporates two IMS–805 transputers at 25MHz with three free links each. The fourth link of each transputer is used to access a common memory swapping controller. The latter controls the access to four 1MByte blocks of memory. Each memory block is allocated to one of two transputers at a time. Control information supplied by the two transputers through a dedicated link is used to change the allocation status. Therefore, we have a kind of shared memory between the two transputers on an SMTB.



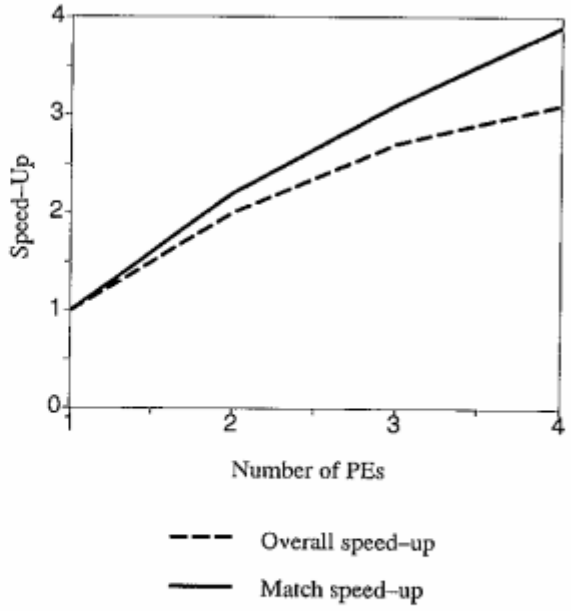
Figure–7: Swappable Memory Transputer Board

8 Preliminary run time measurements

In the absence of a production system compiler for the parallelization method described, we could only encode a few examples on a simple commercially available transputer based architecture. The results have been encouraging, although most of the examples exhibit rather low inherent parallelism. In addition, the scheduling algorithm has not yet been optimized and it therefore causes some overhead.



Figure–8: Example run–time measurements



Figure–9: Speed–up dependence for extended M&B

The examples A and B in figure–8 are simple production systems, characterized by four and nine rules, respectively. Examples C and D use implementations of "Monkeys and Bananas" with static and dynamic partitioning nodes, respectively. Unfortunately, the activity in the RETE–network for these examples is very low so that the low speed–up is not very surprising. Example E is an extended version of "Monkeys and Bananas" using more WMEs. It reveals the full power of the algorithm, yielding a speed–up factor of about 3.9 for the match phase (four PEs). Taking into account all overheads, the factor of 3.1 is still remarkable. Figure–8 shows the speed–up dependence on the number of PEs. Although these examples do not provide a representative set of production systems , they show the existence of expert systems with speed–ups ranging from minimal (one) to maximal (number of PEs) values. These results do not prove the efficiency of our parallelization algorithm but they serve as a motivation for further investigations.

9 Summary and Future Directions

We presented a new approach to parallel execution of production systems, exploiting data parallelism in token memory. The approach has the following advantages compared to other published parallelization methods that rely on program parallelism:

- high utilization of processing power,
- no need for locking mechanisms for the consistency of the RETE–network,
- small communication overhead, no bottle–neck on shared resources
- a scalable architecture

Possible disadvantages of the method presented may be:

- the memory per PE will be about the size of the mono–processor version,
- the scheduling algorithm causes additional computation overhead.

After the Parallel PAMELA–C system is fully implemented, measurements on a representative set of production systems will be performed in order to assess the quality of the parallelization method. Various strategies for scheduling on PRE and alternative parallel architectures will be investigated. In this respect, the adaption of the presented algorithm to a shared memory architecture is of particular interest. The usage of global data both simplifies the scheduling algorithm and increases its accuracy and flexibility. But in order to avoid memory and bus contention, the access to the global memory must either be infrequent or decoupled between the processing elements. Since the data of the RETE network are frequently accessed the contention problem does not allow a straightforward solution. Further investigations of this matter will be the subject of future research.

Acknowledgements

We owe thanks to J. Doppelbauer, H. Gräbner, F. Kasparec, and T. Mandl who constructed the multicomputer architecture we are going to use for the execution of production

systems. We are especially grateful to J. Doppelbauer for providing us with a photo of the Swapable Memory Transputer Board and with technical information about the hardware.

References

[Barachini and Theuretzbacher 1988] Barachini F., Theuretzbacher N.: "The Challenge of Real–Time Process Control for Production Systems", The Seventh National Conference on Artificial Intelligence (AAAI–88), St. Paul, Minnesota, Vol II, 1988.

[Barachini 1988] Barachini F. : "PAMELA: A Rule–Based AI Language for Process–Control Applications", Proceedings on the first International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Vol 2, pp 860–867, Tennessee, 1988.

[Bhuyan 1987] Bhuyan L.N.: "Interconnection Networks for Parallel and Distributed Processing"; IEEE Computer, June 1987, pp. 9 ff.

[Bhuyan 1989] Bhuyan L.N., Yang. Q., Agrawal D.P.: "Performance of Multiprocessor Interconnection Networks", IEEE Computer, February 1989, pp. 25 – 37

[Butler *et al.* 1988] Butler P.L., Allen J.P., Bouldin D.W. : "Parallel Architecture for OPS5", The 15th Annual International Symposium on Computer Architecture, Honolulu, Proceedings pp 452–457, 1988.

[Cooper and Wogrin 1988] Cooper T.A., Wogrin N. : "Rule–based Programming with OPS5", Morgan Kaufmann Publishers, Inc., Palo Alto, USA, 1988.

[Eager *et al.* 1989] Eager D.L., Zahorian J., Lazowska E.: "Speedup Versus Efficiency in Parallel Systems", IEEE Transactions on Computers, Vol. 38, No. 3, March 1989, pp. 408 ff.

[Flynn 1972] Flynn M.J., Some Computer Organizations and Their Effectiveness, IEEE Trans. Computers Vol. 21, No. 9, Sept. 1972, pp. 948 – 960

[Forgy 1979] Forgy C.L. : "On the Efficient Implementation of Production Systems", Ph.D. Thesis, Carnegie–Mellon University, 1979.

[Forgy 1980] Forgy C.L.: "Note on Production Systems and ILLIAC IV", Technical Report CMU–CS–80–130, CMU, Pittsburgh, 1980

[Forgy 1982] Forgy C.L. : "RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem", Artificial Intelligence, Vol. 19, pp. 17–37, 1982.

[Gupta 1986] Gupta A.: "Parallelism in Production Systems"; CMU–CS–86–122, Ph.D. Thesis, Carnegie–Mellon University, March 1986

[Gupta 1987] Gupta A. et al.: "Results of Parallel Implementation of OPS5 on the Encore Multiprocessor"; CMU–CS–87–146, August 1987

[Gupta and Tambe 1988] Gupta A., Tambe M. : "Suitability of Message Passing Computers for Implementing Production Systems", Proceedings of AAAI–88, Vol 2, pp. 687–692, St.Paul, Minnesota, 1988.

[Kasparec *et al.* 1989] Kasparec F., Doppelbauer J., Gräbner H., Mandl T.: "Advanced Transputer Interconnection Techniques"; 1st International Conference on the Application of Transputers (SERC/DTI), Univ. of Liverpool, Aug. 1989

[Kelly and Sevoria 1987] Kelly M.A ., Seviora R.E.: "A Multiprocessor Architecture for Production System Matching"; Proceedings of the AAAI–87, Vol.1,pp. 36–41, 1987 1987

[Miranker 1984] Miranker, D.P.: "The performance Analysis of TREAT : A DADO Production System Algorithm", International Conference on Fifth Generation Computing, Tokyo 1984, revised article 1986

[Miranker 1987] Miranker D.P.: "TREAT: A New and Efficient Match Algorithm for AI Production Systems"; Ph.D.Thesis, Columbia Unversity 1987

[Oshisanwo and Dasiewicz 1985] Oshisanwo A.O., Dasiewicz P.P.: "A Parallel Model and Architecture for Production] Systems"; Proceedings of the 1987 International Conference on Parallel Processing, pp.147–153 May 1985

[Schreiner and Zimmermann 1987] Schreiner F., Zimmermann G.: "PESA 1 – A Parallel Architecture for Production Systems"; Proceedings of the 1987 International Conference on Parallel Processing, pp. 166–169

[Shaw 1987] Shaw D.E.: "NON–VON's Applicability to Three AI Task Areas", IJCAI 1987

[Stolfo 1984] Stolfo S.J.: "Five Parallel Algorithms for Production System Execution on the DADO Machine"; National Conference on Artificial Intelligence, AAAI–1984

[Tenorio 1984] Tenorio M.F.M., "Parallelism in Production Systems", Ph.D. Thesis, University of California, 1984.

[Tien and Raghavendra 1987] Tien S–B.R., Raghavendra C.S.: "A Parallel Algorithm for Execution of Production Systems on HMESH Architecture"; Fall Joint Computer Conference, 1987, pp.349–356