DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9514

Performance Considerations in VOTE for PEACE

Marco Dimas Gubitoso
Jörge Cordsen

Novembro 95

Performance Considerations in VOTE for PEACE

Marco Dimas Gubitoso
Instituto de Matemática e Estatística
University of São Paulo
Jörg Cordsen
GMD-FIRST Berlin

November 6, 1995

1 Introduction

This report is an initial description of ongoing research work between the university of São Paulo and GMD-FIRST's operating system group. The goal of the joined project is to develop performance prediction models for parallel programs, which are based on a virtual shared memory system and runs on the parallel computing system MANNA. As the report shows, comparison of prediction times and measurements allows an easy identification of an application's performance bottlenecks and, moreover, scalability problems in the system software.

2 VOTE

The VOTE system [2] is a member of the program family based PEACE operating system [1]. VOTE extends PEACE by a virtual shared memory (VSM) system to close the gap between usual shared memory programming methods and the architectural constraints of distributed memory machines. Therefore, the design philosophy is strongly related to a symbiosis of architectural transparency and performance aspects. This is managed by a

program family of memory consistency models, which is briefly described in the following subsection. Beside of trying to prevent memory access fault through specialized memory consistency models, it is also promising to tune the access fault handling mechanism. Section 2.2 describes the approach chosen in VOTE.

2.1 Consistency Models

Trying to support both aspects of transparency and performance require to design a VSM system that is based on the semantics of sequential consistency [3] ("a read operation will return the value of the last write operation to the same memory location") and, thus, supports a shared memory programming view even in a distributed memory machine. On top of the sequential consistency model, less strict consistency models are supported to allow further runtime improvements. The use of these extensions is tied to changes in the semantic of memory accesses, which the application must obey. In return, the extended memory consistency models require a reduced consistency maintenance with improvements in the runtime behavior.

At any point in time the programmer can switch between a variety of consistency models, which are tuned to support particular access patterns. There are simple models just providing an once–only handling of faulting read–write sequences (called *pending* model) and more advanced models which automatically performs claims or releases of access to shared data items for a later point in time (*continuous* model).

The most versatile model is the update consistency model which realizes a multiple reader/multiple writer operation mode. Consider, for example, the lock–step operation of the SPMD (single program–multiple data) computation model. During a computation phase (step), there is often no need for data consistency, since the parallel threads operate on different data items. Since the VSM system's granularity is a page, a very fine–grained data parallelism may cause serious problems (false sharing). Allowing multiple writers access to a local copy prevents the thrashing effects. Before entering the ensuing communication phase (lock), sequential consistency must be re–established by an operation scheme which merges the diverging copies into a consistent one.

The VOTE approach to re–establish sequential consistency is a two–stage restoration mechanism. When a process activates the update model, it re-

ceives two local copies (reference and local) of the VSM object. The reference version is solely for the reason to store the VSM object's state at the time the process left sequential consistency, whereas the local version is a private writeable version allowing an independent processing in the activation time of the update model.

$$\texttt{page} := \underbrace{\texttt{page} \oplus \overbrace{(\texttt{local} \oplus \texttt{reference})}^{\textit{library operation (i)}}}_{\textit{sequential consistent operation (ii)}}$$

Figure 1: Restoring update to sequential consistency

At deactivation, a difference set of the reference and local versions is created (see figure 1) by a bitwise exclusive–or operation. This can be done in a library implementation and, thus, operate in a decentralized fashion at high speed. In a second step, the difference set will be merged into the sequential consistent VSM object by virtue of the global sequential consistency model.

2.2 Access fault handling

The benefit of consistency models is that they prevent faulting accesses and, thus, maximize the time in between. A supplementary key to VSM performance is given through an optimization of the access fault handling with the goal to minimize the time required to handle a faulting access.

The VOTE system as a pure software solution performs access fault handling without any kind of hardware supported multi– or broadcast functionality. The software architecture to handle access faults is a straightforward scheme guaranteeing consistency maintenance on a per object basis. To do so, there is a local VSM management thread on each site that use VSM. These threads are called actors and they are controlled by a central knowledge site. This knowledge site is called *adviser* and runs as a preemptive thread on any processor. Although this thread can manage the complete set of VSM objects, it's mostly favorable to use several advisers on different

processors. This prevents a central resource accumulation and helps to avoid a bottleneck for access fault handling.

An access fault situation is initiated by a requesting site either causing a read or a write access fault. The local access fault handling looks up the responsible adviser and performs a call to that thread. To ensure consistency maintenance, the adviser thread manages all global state information about page locations (holding sites), access rights and the memory consistency models in use. When the adviser receives a page request it simply initiates a page transfer request, which will copy the requested page from a holding to the requesting site. The request also cover information about access rights. Thus, the involved actor threads are able to program the memory management unit in respect to the current memory consistency model.

Runtime of access fault handling			Virtual shared memory access fault (time in μsec.)	
Page copy at site		# other		
requesting	*knowledge*	copies	*read mode*	*write mode*
no	yes	0	832_1	930_4
no	yes	1	832_2	1270_8
yes	yes	0	—	650_6
yes	yes	1	—	990_7
no	no	1	1920_3	1620_5

Table 1: Costs of Access Fault Handling

The runtime numbers of the measurements can be found in table 1. The numbers indicate that VOTE is optimized to handle read access faults. Mainly this is done by caching VSM pages at the knowledge site. An access fault then can be handled directly by the adviser ($_{1,2,4}$). In turn ($_3$), it requires to send to the adviser a copy of a VSM page if a former exclusive write access holder must give away a copy to a thread that faulted on a read access. The result of handling such a read access fault are three readable VSM copies at the requesting, knowledge and holding site with an extra runtime of about 300 μsecs ($_{3-8}$). Normally, this is compensated since the approach saves about 780 μsecs ($_{8-1}$) for every succeeding read access fault

to the same VSM page.

Write access fault handling may due to the MRSW (multiple reader/single writer) sequential consistency maintenance be of high cost. The current VOTE system still use a sequential algorithm to send invalidation messages to holders of a read copy. This causes an increase in runtime of about 340 μsecs for each additional holder of a VSM page copy.

3 Description of Algorithm

The algorithm used for tests was Gaussian elimination, since it is highly parallelizable but potentially has a high communication cost.

One can parallelize Gaussian Elimination row-wise or column-wise. In a Fortran code, it is more natural to compute the columns (or blocks of columns) in parallel. In our case it is better to compute the rows in parallel, since in C++ the matrix is distributed in the memory row by row and making the matrix shared puts a contiguous part of a row in each VSM page.

Parallelization is done assigning a processor to each row, in a round robin fashion. The main problem is that at each iteration, one row must be sent to all other processors, generating a high communication cost and contention, as will be seen.

In the beginning of the program, the matrix is completely located in the first processor.

Initially a simplified version was implemented, with no pivoting. To avoid errors, the test for the diagonal element was replaced for a fake one, which made the reference, but replaced the value locally for a constant. This provides the same number of operations and remote references as the original code and makes the model simpler. Of course the result is not correct, but we wanted a testbed, not the actual routine, at this stage.

The code can be seen in figure 2.

4 Performance Model

The total execution time can be decomposed in several terms, basically grouped in two classes: the application time and overheads.

```
 1: // loop on columns
 2: for (j = 0; j < n-1; j++) {
 3:     sync->pass ();
 4:     if (la.v[j][j] == 0.0) faults++;          // RF
 5:     // row elimination
 6:     for (i = j+1; i < n; i++) {
 7:         if ((i % maxprocs) == id) {
 8:             la.v[i][j] /= -1.01;              // R+W F
 9:             for (k = j+1; k < n; k++) {
10:                 t = la.v[j][k];               // RF
11:                 la.v[i][k] += t*la.v[i][j];
12:             }
13:         }
14:     }
15: }
```

Figure 2: Modified Gaussian elimination

The application time (T_{ap}) is the time spent in actual computations according to the algorithm used. It can be computed counting the number of operations and considering all dependencies. It follows the theoretical complexity of the algorithm.

In general T_{ap} consists of a sequential part T_{seq} and a parallel part T_{par}. The application time for P processors is then:

$$T_{ap}(P) = T_{seq} + \frac{T_{par}}{P} \tag{1}$$

The model depends on the algorithm scalability and on system overheads. The actual computational part is highly parallel as can be seen in figure 2.

In fact the inside loop (on `i`) is a `DOALL` and the only sequential operation is the test on the diagonal element. The execution time for P processors can then be written as

$$T_P = \frac{T_1}{P} + \text{overheads}(P) \tag{2}$$

To complete the model, it is necessary to identify all kinds of overheads which may affect performance. The most important are discussed in the following sections:

4.1 Communication

Communication cost is, together with load unballancing, usually the most important cause for performance loss. This is particularly true for a virtual shared memory system where full pages are sent from one processor to another. The software overhead associated with management is considered as part of communication cost, in this model.

In our program, the communication is in page exchanges and in barriers. Since a barrier call is much faster than a page exchange, we will consider the later first.

To get a first approximation for communication cost, we need to count the number of page faults and its types. To compute this number, these considerations take place:

- At the very beginning the whole matrix is in the first processor.
- On the first access to a row there is a read fault followed be a write fault (line 8 in the code), except for the first processor. The read slow is because the page has to move from processor 1 to the adviser and then to the requesting processor. The write is faster because the page is already in the adviser.
  Since $\frac{1}{P} \times$ pages remain in the first processor and there are $n-1$ rows used in the computation, we get

  $$\frac{P-1}{P} \times (n-1) \times K \tag{3}$$

  read and write page faults, where K is the number of pages per row
- At each iteration, there are an additional $P-1$ read faults, since the row containing the diagonal element must be sent to the other processors.

The number of actual page faults depends on row's size.
There are $n-1$ iterations and the number of read faults is given by:

$$\text{TotPages} = \sum_{i=0}^{n-2} \left\lceil \frac{(n-i) * sizeof(double)}{sizeof(page)} \right\rceil * (P-1) \qquad (4)$$

For a matrix 500x500 (1 page per row), the number of read and write page faults are, respectively:

$$WF = 500 * \frac{(P-1)}{P} \qquad (5)$$

$$RF = WF + 499 * (P-1) \qquad (6)$$

and for a matrix 1000x1000, these values are:

$$WF = 2000 * \frac{(P-1)}{P} \qquad (7)$$

$$RF = WF + (\tfrac{3}{2}) * 999 * (P-1) \qquad (8)$$

The final communication costs are described in section 4.3.

4.2 Unballancing

In this case, unballancing is not important, except for the last few iterations of the inner loop, when some processors are idle. It is a second order effect and only important for a large ratio procs/rows, which is not the our case (up to 16 nodes).

Some processors will become idle when waiting for a barrier or the adviser, but this will be considered a contention overhead instead.

4.3 Contention

Contention happens in barriers and in the adviser. In both cases requests are handled sequentially.

Barrier contention can be easily modeled by a linear function starting with 2 processors, since for one processor there is no need for synchronization. This overhead is additive, that is, one has just to add the barrier time to the total time. However this cost is small when compared to communication.

The adviser contention, on the other hand, can not be modeled as an additional term. It is strongly related to communication cost and its description must be embedded inside the communication time expression.

The time to send a page to several processors can be decomposed in two parts: the time for the adviser to get the page and the time to send to the remaining processors.

Sending the page to the first requesting processor takes the time of a slow read fault (page to adviser and then to processor), and sending it to remaining ones takes $P-2$ fast read faults, since they are attended sequentially.

If we name the 'basic times' as

t_{frf} time for a fast read fault
t_{srf} time for a slow read fault
t_{fwf} time for a fast write fault

we can then write the total communication time, remembering that the first read is not on adviser, and for $P=1$ there are no faults,

$$\begin{aligned} T(reads) = & \; t_{srf} * WF + \text{TotPages}(t_{srf} + (P-2) * t_{frf}) \\ T(writes) = & \; t_{fwf} * WF \end{aligned}$$

where TotPages is the number of pages moved during the computation and not considering the initial distribution of the matrix.

The numbers for matrices 500×500 and 1000×1000 are:

$$\begin{aligned} T^{[500]}(reads) = & \; t_{srf} * WF + 499 * (t_{srf} + (P-2) * t_{frf}) \\ T^{[500]}(writes) = & \; t_{fwf} * WF \\ T^{[1000]}(reads) = & \; t_{srf} * WF + (3/2) * 999 * (t_{srf} + (P-2) * t_{frf}) \\ T^{[1000]}(writes) = & \; t_{fwf} * WF \end{aligned}$$

4.4 The model

We can now write a detailed model, based on the discussion in the previous sections.

The total running time for P processors is given by:

$$\begin{aligned} T_P = & \; \frac{T_1}{P} + T(reads) + T(writes) \\ = & \; \frac{T_1}{P} + (t_{srf} + t_{fwf}) * WF + \text{TotPages}(t_{srf} + Dist(P-2)) \end{aligned}$$

where $Dist(x)$ is a function which gives the broadcast cost of a page for x processors and TotPages can be computed by equation 4.

In the initial version, $Dist(x)$ has the form:

$$Dist(x) = x * t_{frf} \tag{9}$$

5 Measurements

In this section we compare the predicted and measured times for matrices 500×500 and 1000×1000.The time parameters for the used implementation are the following, in seconds:

$$t_{srf} = .002300 \tag{10}$$

$$t_{frf} = .000740 \tag{11}$$

$$t_{fwf} = .001040 \tag{12}$$

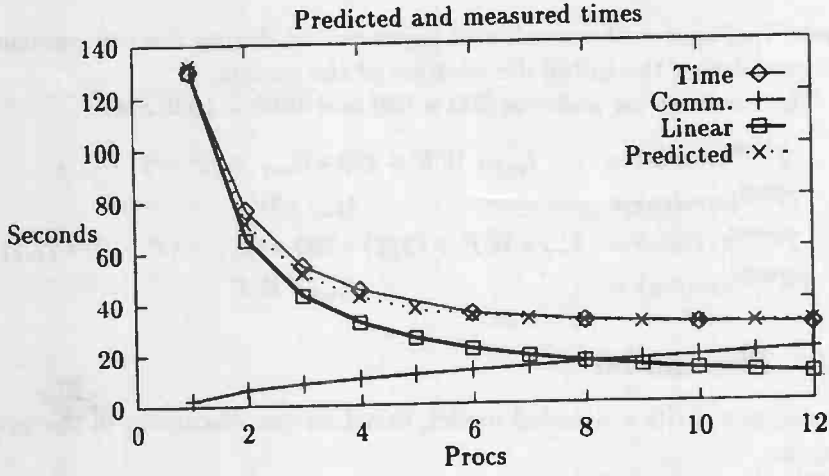

Figure 3: Predicted times for matrix 1000×1000

Figure 3 shows the predicted and measured times for the 1000×1000 matrix. The other curves presents the 'ideal time', which would provide linear speedup and the communication cost. The time spent on a single processor session is 129.94.

It is clear from the picture that the model describes the processing time within a reasonable accuracy. Figure 4 presents the predicted time for a larger number of processors.

From that picture we can verify that the optimal number of processors were reached in the experiments (12). Unless a more efficient communication method is used, one cannot expect a better speedup.

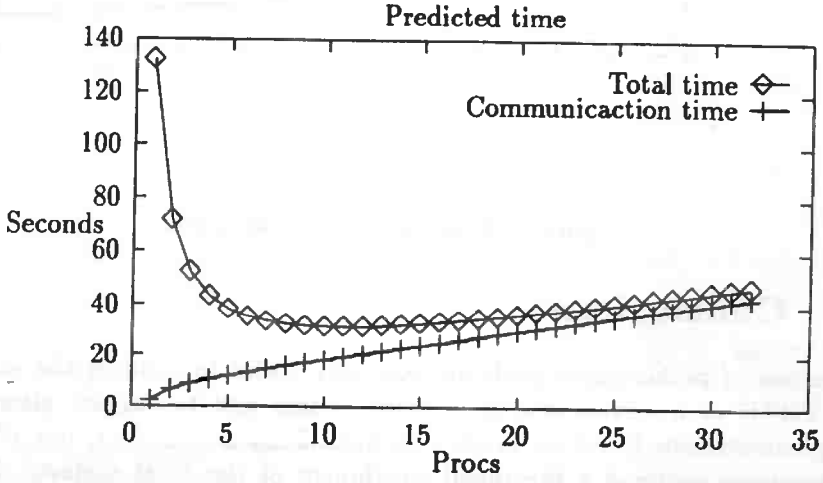

Figure 4: Predicted times

For the 500×500 matrix, the single processor time was 15.97s, and the results are present in figure 5. The speedup is not as good as the previous case because the ratio computation/overhead is not as high.

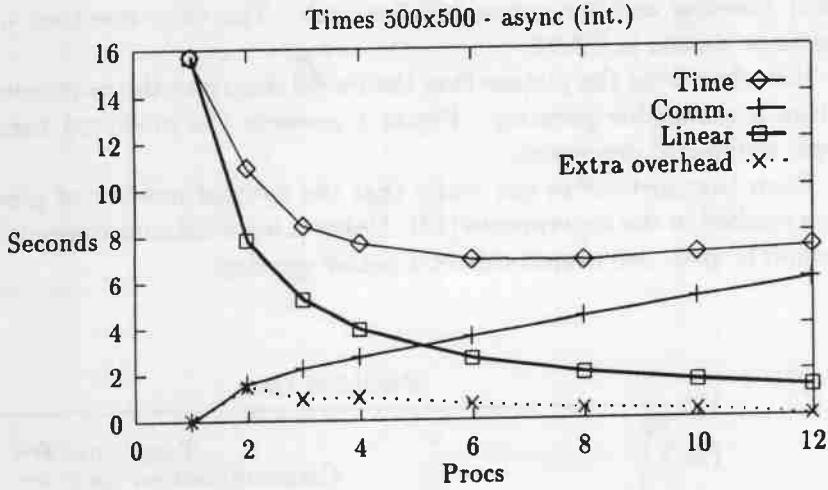

Figure 5: Times for matrix 500 × 500

6 Conclusions

The use of performance modeling was very useful to confirm the viability of VOTE as a virtual shared memory system and to suggest alternative implementations based on bottleneck identifications. In fact, one of these suggestions required a functional enrichment of the VSM system. A new version of the VOTE system includes a propagation model to multicast data areas. This considerably reduces the required communication and its results will be the subject of a future paper.

The model developed here, although simple, is powerful enough to predict accurately the values measured and can be extended to make a performance analysis a priori of specific problems, and to decide if is it worth to implement them under this platform.

References

[1] J. Cordsen, W. Schröder-Preikschat, "Towards a Scalable Kernel Architecture", In *Proceedings of the Autumn 1992 Openforum Technical Conference*, pp. 15–33, Utrecht, The Netherlands, November 23–27, 1992.

[2] J. Cordsen, "Basing Virtually Shared Memory on a Family of Consistency Models", In *Proceedings of the International Workshop on Support for Large Scale Shared Memory Architectures*, pp. 58–72, Cancun, Mexico, April, 1994.

[3] L. Lamport, "How to make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs", *IEEE Transactions on Computers*, Vol. C–28, No. 9, pp. 241–248, September, 1979.

[4] Flatt, H. P. Further results using the overhead model for parallel systems. *IBM Journal Res. Development* **35**, No. 5/6, 721-726 (1991).

[5] Flatt, H. P. and Kennedy, K. Performance of parallel processors. *Parallel Computing* **12**, No. 1, 1-20 (1989).

[6] Polychronopoulos, C. D. *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston, 1988.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1992 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

J.Z. Gonçalves, Arnaldo Mandel
COMMUTATIVITY THEOREMS FOR DIVISION RINGS AND DOMAINS
RT-MAC-9201, Janeiro 1992, 12 pp.

J. Sakarovitch
THE "LAST" DECISION PROBLEM FOR RATIONAL TRACE LANGUAGES
RT-MAC 9202, Abril 1992, 20 pp.

Valdemar W. Setzer, Fábio Henrique Carvalheiro
ALGOTRITMOS E SUA ANÁLISE (UMA INTRODUÇÃO DIDÁTICA)
RT-MAC 9203, Agosto 1992, 19 pp.

Claudio Santos Pinhanez
UM SIMULADOR DE SUBSUMPTION ARCHITECTURES
RT-MAC-9204, Outubro 1992, 18 pp.

Julio M. Stern
REGIONALIZAÇÃO DA MATRIZ PARA O ESTADO DE SÃO PAULO
RT-MAC-9205, Julho 1992, 14 pp.

Imre Simon
THE PRODUCT OF RATIONAL LANGUAGES
RT-MAC-9301, Maio 1993, 18 pp.

Flávio Soares C. da Silva
AUTOMATED REASONING WITH UNCERTAINTIES
RT-MAC-9302, Maio 1993, 25 pp.

Flávio Soares C. da Silva
ON PROOF-AND MODEL-BASED TECHNIQUES FOR REASONING WITH UNCERTAINTY
RT-MAC-9303, Maio 1993, 11 pp.

Carlos Humes Jr., Leônidas de O.Brandão, Manuel Pera Garcia
A MIXED DYNAMICS APPROACH FOR LINEAR CORRIDOR POLICIES
(A REVISITATION OF DYNAMIC SETUP SCHEDULING AND FLOW CONTROL IN MANUFACTURING SYSTEMS)
RT-MAC-9304, Junho 1993, 25 pp.

Ana Flora P.C.Humes e Carlos Humes Jr.
STABILITY OF CLEARING OPEN LOOP POLICIES IN MANUFACTURING SYSTEMS (Revised Version)
RT-MAC-9305, Julho 1993, 31 pp.

Maria Angela M.C. Gurgel e Yoshiko Wakabayashi
THE COMPLETE PRE-ORDER POLYTOPE: FACETS AND SEPARATION PROBLEM
RT-MAC-9306, Julho 1993, 29 pp.

Tito Homem de Mello e Carlos Humes Jr.
SOME STABILITY CONDITIONS FOR FLEXIBLE MANUFACTURING SYSTEMS WITH NO SET-UP TIMES
RT-MAC-9307, Julho de 1993, 26 pp.

Carlos Humes Jr. e Tito Homem de Mello
A NECESSARY AND SUFFICIENT CONDITION FOR THE EXISTENCE OF ANALYTIC CENTERS IN PATH FOLLOWING METHODS FOR LINEAR PROGRAMMING
RT-MAC-9308, Agosto de 1993

Flavio S. Corrêa da Silva
AN ALGEBRAIC VIEW OF COMBINATION RULES
RT-MAC-9401, Janeiro de 1994, 10 pp.

Flavio S. Corrêa da Silva e Junior Barrera
AUTOMATING THE GENERATION OF PROCEDURES TO ANALYSE BINARY IMAGES
RT-MAC-9402, Janeiro de 1994, 13 pp.

Junior Barrera, Gerald Jean Francis Banon e Roberto de Alencar Lotufo
A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9403, Janeiro de 1994, 28 pp.

Flavio S. Corrêa da Silva
ON THE RELATIONS BETWEEN INCIDENCE CALCULUS AND FAGIN-HALPERN STRUCTURES
RT-MAC-9404, abril de 1994, 11 pp.

Junior Barrera; Flávio Soares Corrêa da Silva e Gerald Jean Francis Banon
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES
RT-MAC-9405, abril de 1994, 15 pp.

Valdemar W. Setzer; Cristina G. Fernandes; Wania Gomes Pedrosa e Flavio Hirata
UM GERADOR DE ANALISADORES SINTÁTICOS PARA GRAFOS SINTÁTICOS SIMPLES
RT-MAC-9406, abril de 1994, 16 pp.

Siang W. Song
TOWARDS A SIMPLE CONSTRUCTION METHOD FOR HAMILTONIAN DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9407, maio de 1994, 13 pp.

Julio M. Stern
MODELOS MATEMATICOS PARA FORMAÇÃO DE PORTFÓLIOS
RT-MAC-9408, maio de 1994, 50 pp.

Imre Simon
STRING MATCHING ALGORITHMS AND AUTOMATA
RT-MAC-9409, maio de 1994, 14 pp.

Valdemar W. Setzer e Andrea Zisman
*CONCURRENCY CONTROL FOR ACCESSING AND COMPACTING B-TREES**
RT-MAC-9410, junho de 1994, 21 pp.

Renata Wassermann e Flávio S. Corrêa da Silva
TOWARDS EFFICIENT MODELLING OF DISTRIBUTED KNOWLEDGE USING EQUATIONAL AND ORDER-SORTED LOGIC
RT-MAC-9411, junho de 1994, 15 pp.

Jair M. Abe, Flávio S. Corrêa da Silva e Marcio Rillo
PARACONSISTENT LOGICS IN ARTIFICIAL INTELLIGENCE AND ROBOTICS.
RT-MAC-9412, junho de 1994, 14 pp.

Flávio S. Corrêa da Silva, Daniela V. Carbogim
A SYSTEM FOR REASONING WITH FUZZY PREDICATES
RT-MAC-9413, junho de 1994, 22 pp.

Flávio S. Corrêa da Silva, Jair M. Abe, Marcio Rillo
MODELING PARACONSISTENT KNOWLEDGE IN DISTRIBUTED SYSTEMS
RT-MAC-9414, julho de 1994, 12 pp.

Nami Kobayashi
THE CLOSURE UNDER DIVISION AND A CHARACTERIZATION OF THE RECOGNIZABLE Z-SUBSETS
RT-MAC-9415, julho de 1994, 29pp.

Flávio K. Miyazawa e Yoshiko Wakabayashi
AN ALGORITHM FOR THE THREE-DIMENSIONAL PACKING PROBLEM WITH ASYMPTOTIC PERFORMANCE ANALYSIS
RT-MAC-9416, novembro de 1994, 30 pp.

Thomaz I. Seidman e Carlos Humes Jr.
SOME KANBAN-CONTROLLED MANUFACTURING SYSTEMS: A FIRST STABILITY ANALYSIS
RT-MAC-9501, janeiro de 1995, 19 pp.

C.Humes Jr. and A.F.P.C. Humes
STABILIZATION IN FMS BY QUASI- PERIODIC POLICIES
RT-MAC-9502, março de 1995, 31 pp.

Fabio Kon e Arnaldo Mandel
SODA: A LEASE-BASED CONSISTENT DISTRIBUTED FILE SYSTEM
RT-MAC-9503, março de 1995, 18 pp.

Junior Barrera, Nina Sumiko Tomita, Flávio Soares C. Silva, Routo Terada
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY PAC LEARNING
RT-MAC-9504, abril de 1995,16 pp.

Flávio S. Corrêa da Silva e Fabio Kon
CATEGORIAL GRAMMAR AND HARMONIC ANALYSIS
RT-MAC-9505, junho de 1995,17 pp.

Henrique Mongelli e Routo Terada
ALGORITMOS PARALELOS PARA SOLUÇÃO DE SISTEMAS LINEARES
RT-MAC-9506, junho de 1995, 158 pp.

Kunio Okuda
PARALELISAÇÃO DE LAÇOS UNIFORMES POR REDUCÃO DE DEPENDÊNCIA
RT-MAC-9507, julho de 1995, 27 pp.

Valdemar W. Setzer e Lowell Monke
COMPUTERS IN EDUCATION: WHY, WHEN, HOW
RT-MAC-9508, julho de 1995, 21 pp.

Flávio S. Corrêa da Silva
REASONING WITH LOCAL AND GLOBAL INCONSISTENCIES
RT-MAC-9509, julho de 1995, 16 pp.

Julio M. Stern
MODELOS MATEMÁTICOS PARA FORMAÇÃO DE PORTFÓLIOS
RT-MAC-9510, julho de 1995, 43 pp.

Fernando Iazzetta e Fabio Kon
A DETAILED DESCRIPTION OF MAXANNEALING
RT-MAC-9511, agosto de 1995, 22 pp.

Flávio Keidi Miyazawa e Yoshiko Wakabayashi
POLYNOMIAL APPROXIMATION ALGORITHMS FOR THE ORTHOGONAL Z-ORIENTED 3-D PACKING PROBLEM
RT-MAC-9512, agosto de 1995, pp.

Junior Barrera e Guillermo Pablo Salas
SET OPERATIONS ON COLLECTIONS OF CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMINIG OF MORPHOLOGICAL MACHINES
RT-MAC-9513, agosto de 1995, 84 pp.

Marco Dimas Gubitoso e Jörg Cordsen
PERFORMANCE CONSIDERATIONS IN VOTE FOR PEACE
RT-MAC-9514, novembro de 1995, 18pp.