VOL. 9, NO. 8, AUGUST 2014 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

DATA AND KNOWLEDGE PARALLEL PROCESSING BY MEANS OF N-TUPLE ALGEBRA

B. A. Kulik[1] and A. Ya. Fridman[2]
[1]Institute of Problems in Mechanical Engineering, Russian Academy of Sciences (RAS), 61 Bol'shoi pr., Petersburg, Russia
[2]Institute for Informatics and Mathematical Modelling, Kola Science Centre of RAS, Apatity, Russia
E-Mail: ba-kulik@yandex.ru

ABSTRACT

N-tuple algebra is a mathematical system to formalize *n*-ary relations. This algebra provides for modelling both data (graphs, *n*-ary relations) and knowledge (semantic networks, reasoning models, formulas of propositional and predicate calculi, production systems, ontologies, etc.) by the same structures. These structures look like matrices and can be easily processed by parallel algorithms.

Keywords: parallel computation, *N*-tuple algebra, data structure, knowledge base.

1. INTRODUCTION

To provide parallel computations in many problems and algorithms, it is necessary to preliminarily compose a software connection graph and then choose its independent branches suitable for parallel processing. In particular, modern intelligence systems mostly obtain parallelism by composing a connection graph (for instance, a Kowalsky connection graph for pairs of resolving disjuncts [1]) and choosing its connections allowing for parallel processing [2]. This technique is very complicated to use and low efficient due to the necessity of considering many conditions and restrictions.

Efficient paralleling of algorithms is achievable in cases when initial data are represented in a matrix form. Conversely, most knowledge processing and analyzing systems use structures not similar to matrices. This is why it looks reasonable to propose such a generalized structure form for data and knowledge representation that would allow for a comparatively easy transformation of a computational process into a large number of independent operations. In our opinion, *N*-tuple algebra (NTA) [3 - 5] can solve this kind of problems by expressing many formats of data and knowledge as NTA structures.

2. NTA BRIEF

NTA is mathematically designed as an algebraic system that needs a support, totalities of operations and relations as well as their properties to be defined. Sometimes, these properties can be uniquely fixed by proving an isomorphism between a given algebraic system and a known one. Particularly, we have proved that NTA is isomorphic to algebra of sets and belongs to the class of Boolean algebras [4].

"NTA support" is an arbitrary set of *n*-ary relations expressed by specific structures called *NTA objects*. We will introduce them some later. Every NTA object is immersed into a certain space of *attributes*. *Domain* is a set of values of an attribute. Names of NTA objects contain an identifier followed by a sequence of attributes names in square brackets; these attributes determine the *relation diagram* in which the NTA object is defined. For example, $R[XYZ]$ denotes an NTA object given within the space of attributes X, Y and Z.

NTA *basic operations* include the *algebra of sets operations*, namely intersection, union, and complement as well as *attributes operations* (renaming and transposition of attributes, elimination and addition of a dummy attribute). Combinations of the listed operations allows defining *auxiliary operations* upon relations, they are join, composition, transitive closure, etc. To compare NTA objects, we use two *basic relations*, namely *inclusion* and *equality*. By its analytical capabilities, we can compare NTA with predicate calculus, and NTA objects model truth areas of predicates and logic formulas.

NTA objects provide a condensed representation of *n*-ary relations. When necessary, by means of specific algorithms these objects can be transformed into ordinary *n*-ary relations containing sets of *n*-tuples called *elementary n-tuples* in NTA. Structures defined on the same relation diagram are called *homotypic* ones. In NTA, it is possible to implement operations of algebra of sets on NTA objects with different relation diagrams as well.

NTA objects, namely *C*-n-tuples, *C*-systems, *D*-n-tuples, and *D*-systems are formed as matrices of subsets of attributes domains called *components*. The components include two types of *dummy components*. One of them is called the *complete component*; it is used in *C*-n-tuples and denoted by "*". A dummy component "*" added in the *i*-th place in a *C*-n-tuple or in a *C*-system equals to the set corresponding to the whole domain of the *i*-th attribute in the relation diagram. Another dummy component ($\varnothing$) called an *empty set* is used in *D*-n-tuples.

Let us now proceed with description of NTA major structures; they are *C*-systems and *D*-systems.

We record a *C-system* as a matrix of component sets framed with square brackets.

For example, $R[XYZ] = \begin{bmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \end{bmatrix}$ is a *C*-system that can be transformed into an ordinary relation (i.e. a set of elementary *n*-tuples) calculated by formula $R[XYZ] = (A_1 \times A_2 \times A_3) \cup (B_1 \times B_2 \times B_3)$ where $A_1, B_1 \subseteq X$; $A_2, B_2 \subseteq Y$; $A_3, B_3 \subseteq Z$.

C-systems are convenient for representing disjunctive normal forms of unary predicates. A one-line *C*-system is called a *C-n-tuple*; it is similar to a row vector in matrix algebra. In logic, a *C*-n-tuple corresponds to a separate conjunct.

D-systems model conjunctive normal forms of unary predicates. We denote a *D*-system as a matrix of component sets framed with reversed square brackets. *D*-systems provide easy calculating of *C*-systems' complements. For instance, the *D*-system $\overline{T}\,[XYZ] = \left]\begin{matrix}\overline{A} & \varnothing & \overline{C} \\ \overline{D} & \overline{E} & \varnothing\end{matrix}\right[$ is the complement of the *C*-system $T\ [XYZ] = \begin{bmatrix}A & * & C \\ D & E & *\end{bmatrix}$. Alike a *C*-system, a one-line *D*-system is called a *D-n-tuple*. In logic, a *D*-n-tuple models a separate disjunct.

Calculations of unions and intersections for *C*- and *D*-structures are specific; you can find their detailed description in [3, 4].

Please note that NTA provides implementing all operations of algebra of sets and all checks of relations among NTA objects (e.g., equality and inclusion) in matrix form, without having to represent these objects as sets of elementary *n*-tuples.

To process NTA objects defined on different diagrams, we have developed some *attributes operations*, *addition of a dummy attribute* (+*Attr*) and *elimination of an attribute* (-*Attr*) in particular. The operation +*Attr* correspond to the rule of generalization in predicate calculus, so it does not change the semantics of any relations. This operation upon any NTA object simultaneously adds the name of a new attribute into the relation diagram and adds a new column with dummy components into the respective place of a matrix representation. Given the relation $R_k[XZ] = \begin{bmatrix}A_1 & A_3 \\ B_1 & B_3\end{bmatrix}$ models the predicate $R_k(x, z)$, adding a dummy attribute *Y* into R_k [*XZ*] results in the formula $\forall y\ (R_k(x, z))$. This operation is done as $+Y\,(R_k\,[XZ]) = \begin{bmatrix}A_1 & * & A_3 \\ B_1 & * & B_3\end{bmatrix}$.

The operation +*Attr* is often used to reduce some different-type NTA objects to the same relation diagram. Then we can perform all necessary operations and checks by means of standard NTA algorithms. Considering this, we have introduced *generalized operations* ($\cap_G$, $\cup_G$). They are possible after reducing NTA objects to the same relation diagram and semantically correspond to logical connectives: conjunction and disjunction. Our algebra of relations with these generalized operations is proved to be isomorphic to the ordinary algebra of sets. This way we have eliminated the restriction in the theory of relations stating that algebra-of-sets laws are only applicable to the relations defined upon the same Cartesian product.

For corresponding logic formulas, elimination of an attribute (let it be *X* for example) from *C*-structures results in quantification $\exists x$, and from *C*-structures - in quantification $\forall x$. In NTA, this operation leads to deleting an attribute from the relation diagram and the respective column from the matrix representation of an NTA object. For instance, calculating -*Y*(*R* [*XYZ*]) for the *D*-system $R[XYZ] = \left]\begin{matrix}A & \varnothing & B \\ C & D & \varnothing\end{matrix}\right[$ gives us $Q\,[XZ] = \left]\begin{matrix}A & B \\ C & \varnothing\end{matrix}\right[$.

An extended NTA version stipulates using some totalities of attributes as separate compound ones. In this case, NTA objects' components include *n*-ary relations rather than ordinary sets. Another NTA extension deals with NTA objects whose attribute domains are different from ordinary sets (e.g. fuzzy sets). If attributes are defined as a system of intervals over a number axis, NTA proposes the *interval quantization method* [5]. After applying this method, such attributes can be processed with all NTA techniques.

A considerable part of NTA algorithms (union, intersection, complement, check of inclusion, etc.) have polynomial complexity. Transformation of *C*-systems into *D*-systems and the opposite operation are exponentially complex. Many information processing procedures do not use such transformations. If they are necessary still, their complexity can be reduced to the polynomial one for some special structures of NTA objects [6].

Meanwhile, we have investigated and grounded NTA applicability for the following data and knowledge structures [4, 5]:

a) graphs and networks;
b) models of propositional calculus and predicate calculus;
c) artificial intelligence systems, namely semantic networks, expert systems, frames, ontologies;
d) models for deductive and abductive reasoning;
e) logic-probabilistic methods including probabilistic logic;
f) discrete automata.

We continue researches in using NTA for other structures, e.g. systems with uncertainties and dynamic systems.

3. PARALLELIZATION IN NTA STRUCTURES

NTA operations are based on theory-of-sets operations with components that is subsets of the attributes domains. For instance, if we want to find intersection of two homotypic *C*-n-tuples, we need to intersect pairs of components in corresponding attributes. This calculation results in an empty set, if any of the pairs intersections are empty. Intersection of a *C*-n-tuple *R* with a homotypic *C*-system *Q* is the result of *R* intersection with every *C*-n-tuple from *Q*. For two homotypic *C*-n-tuples *R* and R_1, inclusion $R \subseteq R_1$ is right, if every component of *R* is included into the respective component of R_1. Some NTA operations do not require for operations with components. As an example, union of two homotypic *C*-systems can be easy obtained by ascription all *C*-n-tuples of the first *C*-system to the second *C*-system.

VOL. 9, NO. 8, AUGUST 2014 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

The above-listed cases describe features of NTA operations completely enough. Now we can proceed with grounding of architecture of a computing system most suitable for parallelization of these operations. We exemplify our ideas with the intersection operation for two homotypic *C*-systems that is done very often. Let such systems be given as

$$R_1[XYZ] = \begin{bmatrix} \{a,b,d\} & \{f,h\} & \{b\} \\ \{b,c\} & * & \{a,c\} \end{bmatrix}, R_2[XYZ] = \begin{bmatrix} \{a,d\} & * & \{b,c\} \\ \{b,d\} & \{f,h\} & \{a,c\} \\ \{b,c\} & \{g\} & \{b\} \end{bmatrix}$$

Alike in *C*-n-tuples, we calculate their intersection as intersection of every *C*-n-tuple from the first *C*-system with every *C*-n-tuple from the second *C*-system:

$$[\{a,b,d\}\ \{f,h\}\ \{b\}] \cap [\{a,d\} * \{b,c\}] = [\{a,d\}\ \{f,h\}\ \{b\}];$$
$$[\{a,b,d\}\ \{f,h\}\ \{b\}] \cap [\{b,d\}\ \{f,h\}\ \{a,c\}] = \varnothing;$$
$$[\{a,b,d\}\ \{f,h\}\ \{b\}] \cap [\{b,c\}\ \{g\}\ \{b\}] = \varnothing;$$
$$[\{b,c\} * \{a,c\}] \cap [\{a,d\} * \{b,c\}] = \varnothing;$$
$$[\{b,c\} * \{a,c\}] \cap [\{b,d\}\ \{f,h\}\ \{a,c\}] = [\{b\}\ \{f,h\}\ \{a,c\}];$$
$$[\{b,c\} * \{a,c\}] \cap [\{b,c\}\ \{g\}\ \{b\}] = \varnothing.$$

Then we form a *C*-system of non-empty *C*-n-tuples:

$$R_1 \cap R_2 = \begin{bmatrix} \{a,\ d\} & \{f,\ h\} & \{b\} \\ \{b\} & \{f,\ h\} & \{a,\ c\} \end{bmatrix}.$$

This typical example can assist in formulating main laws to parallelize NTA operations. First, it is possible to overlap operations with components during intersecting pairs of them. Second, we can parallel intersections of a *C*-n-tuple from the first *C*-system with all *C*-n-tuples from the second *C*-system. And third, there exists a certain computing structure providing parallel calculations with all components necessary to obtain the final result.

The saving of time gained due to parallel computations in NTA can be fair. Suppose, we want to calculate intersection of two *C*-systems with *K* attributes according to the third of the proposed parallelization variants. If the first *C*-system has *M* lines and the second one has *N* lines, acceleration in time will be $K \cdot M \cdot N$.

To process components of NTA objects, it is enough to have processing cells capable to deal with Boolean vectors. Working with every attribute takes a set of uniform processing cells with common memory that stores source and calculated components of the given attribute. Such a structure corresponds to the symmetric multiprocessing (SMP) architecture [7].

To accomplish calculations with sets of attributes, it is reasonable to use SMP modules as cells of computing systems with distributed memory. i.e. for massive parallel processing (MPP) [8]. Since attributes can differ in size significantly, the total computing structure shall include different SMP-clusters diverging in capacity of processing cells and the respective memory modules. Thus, the general task of parallelization for NTA operations can be solved by a hybrid architecture combining features of systems with MPP and SMP architecture (see Figure-1).

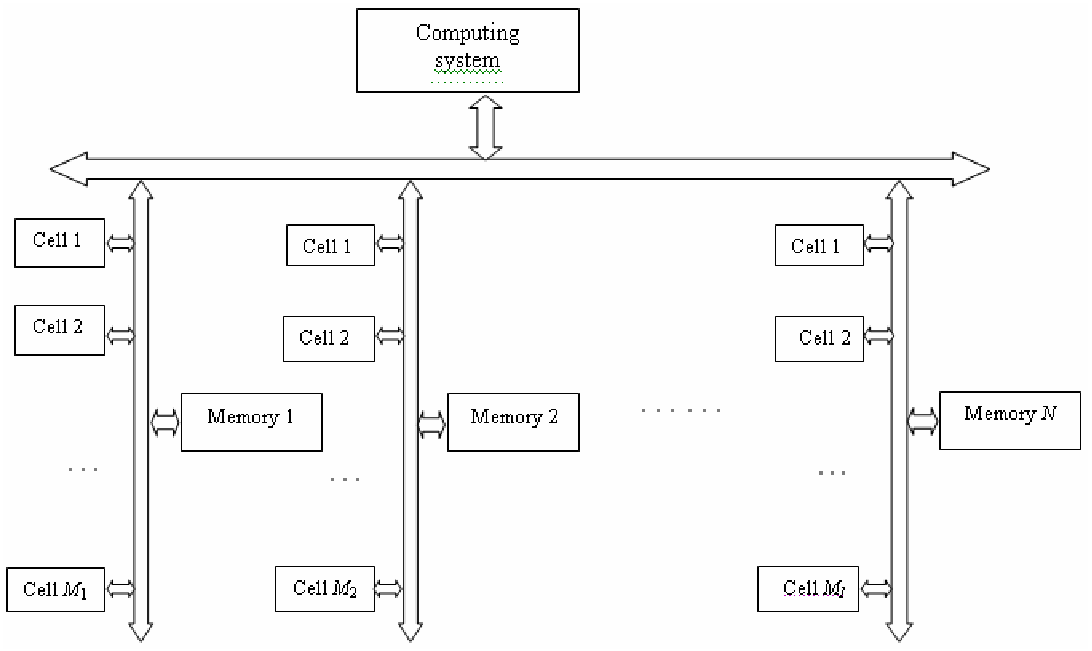

Figure-1. Chart of a hybrid MPP-SMP architecture.

4. CONCLUSIONS

NTA supports representation and processing of various types of data and knowledge formats. In this paper, we propose a computing system with hybrid architecture providing for effective parallelization of operations with NTA structures.

ACKNOWLEDGEMENT

The authors would like to thank the Russian Foundation for Basic Researches (grants 12-07-00302, 12-07-00550, 12-07-00689, 13-07-00318, 14-07-00256, 14-07-00257, 14-07-00205) and the Chair of the Russian Academy of Sciences (project 4.3 of the Programme # 16) for their aid in partial funding of this research.

REFERENCES

[1] R. Kowalski. 1975. A Proof Procedure Using Connection Graphs. J. of the ACM. 22D: 572-599.

[2] R. Loganantharaj and R.A. Mueller. 1986. Parallel Theorem Proving with Connection Graph. 8th Int. Conf. on Autom. Deduc. LNCS 230: 337-352.

[3] B.A. Kulik. N-Tuple Algebra-Based Probabilistic Logic. 2007. J. Computer and Systems Sciences International. 46(1): 111-120.

[4] B. Kulik, A. Fridman and A. Zuenko. 2013. Logical Inference and Defeasible Reasoning in N-tuple Algebra. In: Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems, IGI Global.

[5] B. Kulik, A. Fridman and A. Zuenko. 2012. Algebraic Approach to Logical Inference Implementation. J. Computing and Informatics (CAI), Slovakia. 31(6): 1295-1328.

[6] B.A. Kulik. 1995. New Classes of Conjunctive Normal Forms with a Polynomially Recognizable Property of Satisfiability. J. Automation and Remote Control. 56(2): 245-255.

[7] Ch. Severance and K. Dowd. 1998. High Performance Computing (RISK Architectures, Optimization and Benchmarks), 2nd Edition, O'Reily Media.

[8] K.E. Batcher. 1980. Design of a Massively Parallel Processor. IEEE Trans. on Computers. 29(9): 836-840.