computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

INTELLIGENT SOLVING MACHINES ON MODERN PARALLEL ARCHITECTURES

Valeriy N. Koval, Volodymyr V. Savyak

V.M.Glushkov Institute of Cybernetics. National Academy of Sciences of Ukraine. 40, Prospect Academika Glushkova, 03680, Kiev, Ukraine. e-mail: icdepval@ln.ua, fax: (044) 2664549

Abstract: *The paper considers the creation of intelligent solving machines and the arrangement of parallel programming in intelligent distributed multiprocessor systems based on them. There are proposed some main concepts. A system is designed for programming in the C+Graph high-level language. The ideology proposed can be considered as an efficient development of structural high-level language interpretation when applied to multi-microprocessor systems. Equipment structure of the basic version of intelligent solving machines is considered and some characteristics are discussed.*

Keywords: *Intelligent solving machines, parallel architectures*

1. INTODUCTION

Performance and intelligence are the most important factors promoting the development of modern universal computers. The first factor resulted in the creation of parallel computer architectures for which universal microprocessors (MP) form the most expedient grounds. In such multi-microprocessor systems computation is arranged on the basis of *distributed information processing*, and MPs simultaneously fulfill certain integrated tasks, i.e. independent user program branches.

The second factor becomes clear when the notion of machine intellect (MI) is used. MI defines "internal computer intelligence", the colloquial expression [1]. In this case, computer intelligence is growing with the MI level enhancement.

For the last 5-6 years, V.M. Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine has been carrying out the research aimed at the development of new class of computers. These computers have the broad range of configurations, viz. from powerful workstations up to supercomputers. They are characterized by high and super-high performance as well as high level of MI. Trends of computers of the new class being developed can be seen only in the literature and some conference papers. Therefore, in the world of computer engineering industry the certain niche is observed today, i.e. there are *high-performance intelligent* computers. This direction is rather promising.

Within the framework of the above-mentioned global direction, the paper is focused on the design of the *distributed parallel knowledge-oriented* architectures, i.e. *intelligent solving machines* (ISM) implementing high- and super-high-level languages (HLL and SHLL) and effective operation with large-size data- and knowledge bases, while the problems of both conventional computation and artificial intelligence (AI) are solved [1].

The great deal of work has been done at V.M. Glushkov Institute of Cybernetics and the following outcomes resulted:

1. Microprogram computer and multiprocessor computer with developed internal languages and flexible architectures oriented towards specified problem classes (MIR, Ukraine, the shared intelligent terminal for the Elbrus supercomputer, Macroconveyor and others [9-13].
2. The intelligent systems solving problems in weakly formalised AI domains: classifying and generating; operation scheduling; searching for regularities; likelihood reasonings based on inductive and deductive inference computation, and so on [14-17].

The authors' views were also affected by modern parallel computation architectures, like SMP (Symmetric Multiprocessor System) and MPP (Mass Parallel Processing) [4].

2. INFOMATION TECHNOLOGIES IN A PROBLEM SOLVING FIELDS

Since the ISM-class computers solve AI problems, the paper briefly considers the information technology features dealing with AI solutions. Table 1 compares conventional problem-solving technologies with those of AI. The new information technology (NIT) is the technology based on the AI methods The main differences to conventional technologies are in an interractive and trial-and-error nature of an applications used for AI solutions.

Table 1. Comparing the Technologies

Characteristics	Conventional	NIT
Application domains	Strongly formalised	Strongly and weakly formalised
Solving method	Single run	Cyclic application of trial-and-error method
Problem solving process computation stage	Program execution	Design of problem models. Program synthesis and execution
Program design and execution order	Stages are strictly divided	Stages may alternate during problem execution
Human-computer operation style	Design of a complete problem model and of a program on this basis; interactive interrelation is for program debugging only	Strong interactive interrelation; models are corrected during problem design and solution

The cyclically applied trial-and-error method is the main method that solves AI problems because different solving design strategies are used. Various not fully certain situations (models) frequently emerge in this case. During clarification, they also frequently undergo different modifications. Therefore, the program design and execution stages alternate from time to time. Such stages are strictly divided under the traditional solving technology, but they are not divided under the technology used.

To support this *new information technology* (NIT) the new architectural solutions were needed. Thus, the authors attempted to implement them within the ISMs, the universal computers with the integrated architecture that combines the main features of the conventional and special-purpose AI problem-solving architectures.

3. ISMs: MAIN DESIGN PRINCIPLES

The ISM architecture integrates four basic properties:

- The hardware-software support for procedural and declarative HLLs and SHLLs, applied HLL is the basis in this case.
- The hardware-software support for operations on distributed graph-type data- and knowledge bases and on some other complicated data structures (CDS);
- Combination of the centralized and decentralized controls based on the two-stage interpretation;
- Distributed information processing, based on the application of multiprocessor and multicluster shared-memory architectures.

C+Graph, the basic input language in the ISM-class computers, inherits the features of the C++ and Java languages. It is also extended by the library means used to perform parallel operations on graphs, arrays and the centralized-decentralized control of computation process.

The internal ISM language is the Java-like HLL used to resolve one of the most critical language problems: narrowing a semantic gap between an initial user of HLL programs and their internal representation in the machine; it is possible to operate with knowledge like with CDSs, a user is able not to take part in computation process arrangement.

Knowledge and CDSs are represented in the ISMs as oriented graphs. For each graph, as a linking list there is the distributed representation (an adjacency matrix, etc.). In addition, the graph may be represented as a generalized object, i.e. without an internal structure. A graph-object set may be a semantic network, and it is possible to work with it while there is no distributed representation.

The graphs play one key role. On the one hand, a graph is a CDS, i.e. a data type characterised by its objects and operations on them. On the other hand, it may be a program being executed, i.e. type of control.

The graphs can help to represent quite easily such CDSs and knowledge structures as trees, including binary trees or semantic networks which can vary dynamically, to grow downward or widen and the like.

To process graph structures, C+Graph is extended by the object-oriented means, i.e. the graph-type class and its methods. These methods form the set of operations and interaction mechanisms used when graph-type variables are operated on. As examples of such functions, the can begraph creation and destruction, search along a graph, inserting subgraphs into a graph, generation of a new node or an arc and many others. C+Graph contains a number of set-theoretic operations, algebraic operations of ratios on graphs as on finite automata, operations on relations on graph arcs, control operations by connectors and some other

operations.

The parallel model for programming in C+Graph is based on the model of multiple-flow monoprocessor programming of the basic Java language. To transform this model in order to use it in multiprocessor systems, the notion of a *virtual distributed computation space* is introduced into C+Graph. The mentioned space includes a set of typical and interrelated *virtual processors* that have a hierarchic memory, i.e. a *processing* processor and a *manager* processor (MGP) (Figure 1).

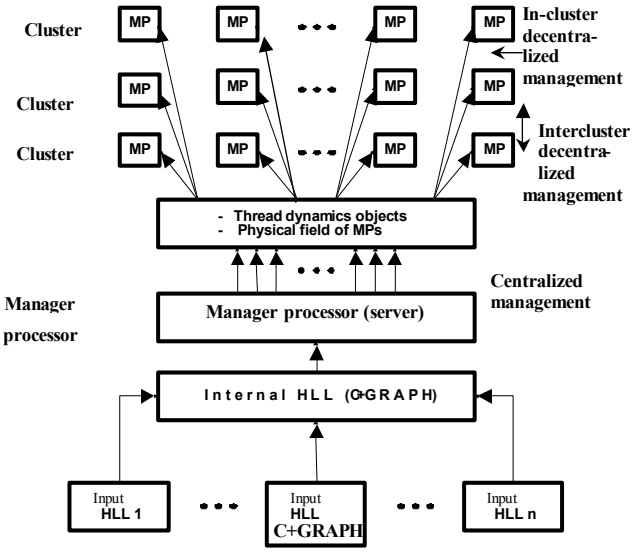

Fig. 1 - Program Objects and Their Parallel Execution.

The processing processors form a cluster computation module. The clusters have their two-way cluster-memory-type intra- and inter-cluster communication channels. Any virtual processing processor can communicate in parallel from one cluster with any virtual processing processor in another cluster in order to transfer data or control information.

The MGP is chosen depending on accepted computing network. It is connected with every virtual processing processor as regards to control and transferred data. The MGP specifies the distributed control in a virtual computing network.

For the parallel distributed programming, the following items are introduced into the library: 1) Vprocessors (virtual processors), the new abstract class; 2) the methods used to map virtual processors onto physical processors; 3) the classes and methods used to cover a virtual space by CDSs: DistrArray (an array), DistrMatrix (a matrix), DistrVector (a vector), DistRing (a ring), DistrStar (a star), DistrGrid (a grid), DistrTree (a tree), DistrLinkedList (a linked list), DistrGraph (a graph); 4) Thread and ThreadGroups, i.e. the in-built flow generation and control classes with their *start()* (start-up), *run()* (running), *stop()* (stopping) and *suspend()*, *resume()*, *join()*, *wite()* (synchronization) methods; 5) the *synchronized* descriptor, etc. In addition, different in-built and the new designed methods are also used for operations on parallel objects (CDSs).

The extended library makes it possible to solve broad range of problems without paying attention to computation process arrangement details. Preliminary CDS and program branch paralleling and distribution across a virtual computation space are performed by the extended library methods together with the input C+Graph means and together with the standardised multiprocessor operation system. The extended library is compatible with this system.

The centralized-decentralized control in the ISMs. The computer-aided programming functions are performed by the compiler and by the interpreter. They make up the virtual C+Graph machine. The compiler compiles binary codes, and the interpreter executes them by means of the MP commands, specifically by means of low-level language.

To maintain the multiprocessor-based execution, each physical MP is connected with one virtual C+Graph machine or with several virtual C+Graph machines. Therefore, it is necessary to create the two-level internal language where each level is implemented with the relevant hardware and software components. The *upper level* is the input algorithmic C+Graph language representation inside the machine after compilation in the Class-file form. This level fixes a machine purpose as a whole (Java machine, for instance) and the centralized control corresponding to the same level by means of the MGP.

The internal language interpreter performs the centralised control in accordance with a program execution graph.

This graph is a dynamic object. Compiler initially creates it when a program is transformed from an external language into an internal one, and its parallel branches are labelled. The same graph is continuously transformed further depending on a problem solving process. The program execution graph is a CDS for the interpreter. Each node in this graph is brought in correspondence with some internal language program section, and some separate sequence of nodes is brought in correspondence with a parallel branch.

The *lower level* is usual MP command level. It fixes direct information processing in an MP field, i.e. it is the decentralized control. Specifically, it captures the interpretation of the commands in C+Graph by the MP commands, and the execution of these commands is also fixed by it. Since there are many operating MPs, then it is possible for internal MP language objects to be *interpreted* in *parallel*. The parallel interpreter of the internal C+Graph language is as if distributed through the whole cluster space.

Thus, the two-stage interpretation is implemented in the ISMs at the internal C+Graph interpreter level in the first case, and, in the second case, the same takes place at the virtual MP level where the virtual MPs are located in the executing (physical) MP memory. The notion of two-stage interpretation when the parallel architecture with the internal two-level language is controlled resulted accordingly. Hence, it is possible to implement an HLL on a parallel architecture.

4. ISM EQUIPMENT STRUCTURE

The ISM equipment was designed in order to provide the hardware support for an internal language and graph operation mechanisms and, in addition, to exercise efficiently the centralized-decentralized control. Like some other high-performance systems, the ISM equipment uses the *cluster principle of system assembly*. However, one specifically provided (or one connected) MGP and the intercluster switching facilities are already present in this case (Fig. 2).

The cluster is the parallel and symmetric SMP-type architecture into which several buses are included for traffic improvement. Taken together, the clusters have the MPP-architecture features [3] and uses high-performance internode communication [2]. Each cluster can contain 2-8 MP modules built on different-type MPs, i.e. on RISC-, SISC-, digital, signal and other processors. The clusters communicate with each other by means of the switches of two types: a short-message switch (SMS) and an intercluster channel switch (ICS). Single switch model also can be used. In that case intercluster channel switch combines short-message and large volume data exchange functionality.

Anticipated basic characteristics of ISM-class computer version are given in Table 2. As it is seen, they fit present-day views about high-performance distributed architectures that meet, in their turn, high-performance computing requirements. Although, additional features were provided in the ISM. They promote the ISM universality, and human-computer interaction efficiency is also enhanced. Everything becomes clear when both traditional and AI problems are solved.

Table 2. Basic ISM-Class Computer Characteristics

Parameter Name	Parameter Values
MP type	AMD
Number of MPs in a cluster	2
Number of clusters in ISM	4
ISM memory capacity (Gbytes)	4
ISM disk memory capacity (Gbytes)	160
Operating system	Linux
Message Passing Interface	MPI
Basic language and extension means	Java, C+Graph

All the main technical solutions have been approved analytically as well as by the simulation means. In addition, they have been proved experimentally on the manufactured operating machine model with the minimal configuration (4 clusters, 8 microprocessors).

5. CONCLUSION

The paper discusses basic ISM-class computer design principles. User programming methods and computer-aided (internal) programming tools are considered. In general, the ISM-class computers are fit for parallel programming in multiprocessor distributed computation environments. C+Graph, the input language, is proposed

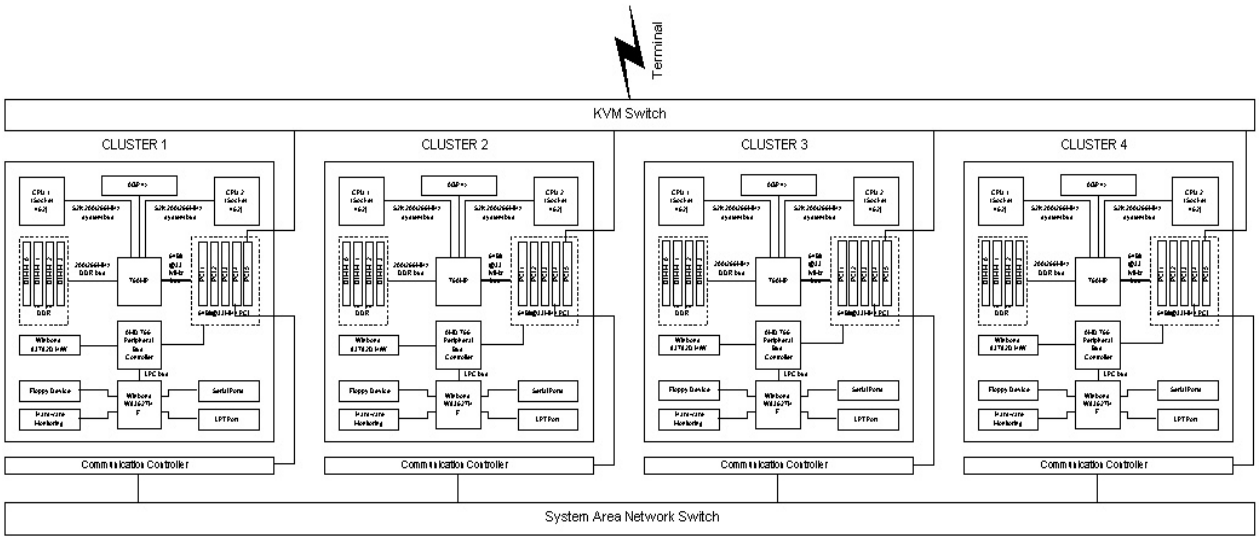

Fig. 2 - ISM Hardware Sample.

for user of parallel programming. User communicates with the multiprocessor system exclusively in HLLs while the multiprocessor system components operate in low-level languages. Moreover, a user does not take part in physical computation process arrangement at all. The present approach may be considered as successful extension of the known structural HLL interpretation principle. Therefore, it may now be called the *structural-program distributed HLL interpretation* principle.

6. REFERENCES

[1] Koval V.N. Bulavenko O.N., Rabinovich Z.L. Parallel Architectures and Their Development on the Basis of Intelligent Solving Machines, Warsaw, 2002.

[2] Savyak V.V. Internode communication interfaces,. Computer Review, #18-19, May 15, 2002.

[3] Savyak V.V. Effective clustering In-Depth, Networks & Telecommunication, #2, March 8, 2002.

[4] Voevodin V.V., Voevodin Vl.V. Parallel Computations, BHV, pp.120-180, 2002.

Volodymyr Savyak *was born in June 25,1975.*

He has graduated National Technical University of Ukraine "Kiev Polytechnical Institute" and obtained Master's degree on Computer Systems and Networks (system analytics, computer systems and network architecture, artificial intelligence).

*Now he as a graduate student of Glushkov Institute of Cybernetics NAS Ukraine and work as technical director of USTAR (*http://www.ustar.ua*). The company is engaged in a computer systems integration, high performance and high availability systems design and manufacturing.*

Areas of scientific interests: high availability and high performance computer systems design.

Hobbi: sports activities (wide range of extreme sports), traveling, music.