CONCEPTION AND DESIGN OF PARALLEL AND DISTRIBUTED APPLICATIONS

Valentin CRISTEA

University Politehnica of Bucharest
E-mail: valentin@cs.pub.ro

The paper presents the state of the art and the main trends in the conception and design of parallel and distributed software. The conceptual classes, design and evaluation models, programming methods, and development tools are analyzed in relation with the evolution of application domains and of the technologies for parallel and distributed systems. Examples include author contributions and results obtained in the National Center for Information Technology – CoLaborator of the University Politehnica of Bucharest.

Key words: Parallel and distributed algorithms, high performance computing, parallel genetic algorithms, replicated workers, web application services

1. INTRODUCTION

Since different meanings have been associated with the terms parallel computing and distributed computing, it is useful to start with the definitions adopted in this paper. **Parallel computing** is solving an application by dividing it in processes that run simultaneously (on multiple processors). **Distributed computing** is solving an application by dividing it in processes that run (possibly together with other applications) on different autonomous computers that are interconnected with each other and cooperate, thereby sharing resources [2]. The main difference lies in the criteria used to decide if an application is parallel or distributed. Parallel computing uses the criterion of time (several processes are executed in the same time), while distributed computing takes the criterion of space (different processes are executed on different resources).

Traditionally, the parallel computing was mainly used in scientific and engineering applications, while distributed computing was the solution for commercial and data processing applications. But, the recent evolution of the domains blurs the differences between the parallel systems and the distributed ones. Both are using the same architectures: on one hand, the invention of fast network technologies enables the use of **clusters** in parallel computing; on the other hand, **parallel machines** are used as servers in distributed applications. Then, the application areas of parallel computing and distributed computing significantly overlap. Finally, the issues of parallelism and distribution intertwine and are researched together. To adapt to these trends, the term High Performance Computing (HPC) is commonly used for both concepts.

On the other hand, the performance of distributed systems is close to that of parallel systems, for machines in the latest generation. We give here two examples to sustain this idea. The first one refers to the category of the most powerful systems in the world. They are registered in the TOP500 list, which is published every six months by Dongarra, Meuer, and Strohmaier [9] based on the performance measured with the Linpack standard benchmark. Among other things, the list highlights the trend of continuous increase of the number of HPC systems that are based on constellation and cluster architectures. In the June 2003 edition of the list, there are 149 clusters and 140 constellations, out of 500 systems. In addition, the MCR Linux Cluster at Lawrence Livermore National Laboratory is ranked the third in the list, which demonstrates that systems with distributed architecture have comparable performance with that of „traditional" parallel systems.

The second example refers to the behaviour of „medium size" systems in large, realistic applications. It sustains the idea that the performance of distributed systems is close to that of parallel systems not only for specially designed benchmarks (such as Linpack), but also for some representative, complex, practical applications. This result has been obtained in a study of Parallel Genetic Algorithms (PGAs), conducted by the author [3].

Genetic algorithms represent an important class of large, realistic applications which address special claims to the programming language used for implementation and to its associated runtime system: a big amount of calculus on large data structures, provision for a variety of genetic operators, code reusability for solving classes of similar problems. The experiments have been developed on two platforms of the National Center for Information Technology – CoLaborator, in the following conditions:

- a cluster of SunBlade 100 workstations, each one with a 450 MHz processor,128 MB of RAM and a 100 Mbps Ethernet interconnection network (referred as "*Blade"* in this paper);
- a Sun E10000 supercomputer, having 32 * 400 MHz processors and 16 GB of RAM (referred as "*HPC"* in this paper); it is worth mentioning that this system was in the TOP 500 list of November '98;
- the same Sun implementation of the MPI communication library used in the application;
- the same PGA program.

Some results are presented in Figure 1, for the distributed population approach (named asynchronous island PGA). The overall population is divided in "islands" of the same size that are allocated to different processors. For the same size of the overall population, the increase of the number of processors determines the reduction of the number of individuals in each island. Consequently, the algorithm finishes faster, while assuring a similar exploration of the solution space. As can be seen, the two different platforms used for experiments register very close execution times, so that analyzed configurations - distributed (Blade) and parallel (HPC) – prove their suitability for this class of realistic, complex applications
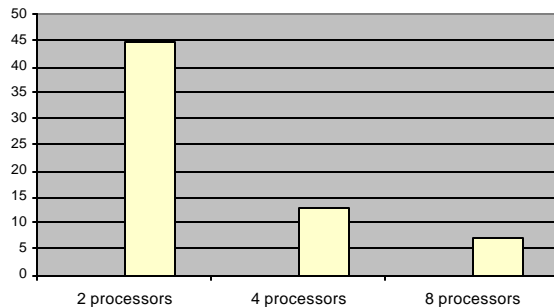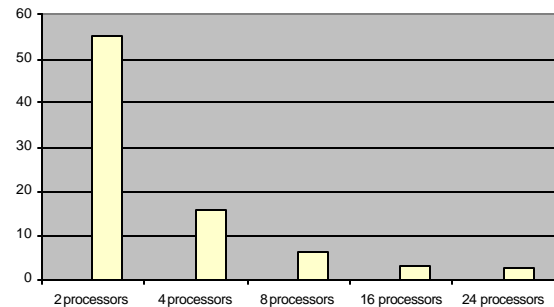

Fig. 1a. Blade.   Fig. 1b. HPC.

Fig. 1. Execution time vs. number of processors.

Of course, the use of similar parallel and distributed systems and of the same application development support in different application areas (research, industry, commerce, and services) influences the way we approach the **conception and design** of parallel and distributed (P&D) software. The requirements for this software extend beyond the classical attributes that dominated the production of parallel programs (efficiency, speedup, and reliability) and include new items such as: availability, ease of use, portability, re-usability, maintainability, scalability, and interoperability. Producing quality P&D software is no longer a question of creativity and cannot rely exclusively on the developers' abilities. Instead, a methodical approach must be used to maximize the range of options considered, provide mechanisms for evaluating alternatives and reduce the risk of obtaining bad solutions.

The paper presents actual trends in P&D software development and highlights some of the results obtained by the author in the improvement of the methodology for P&S software. The focus is on the following issues: conceptual and design models, performance analysis models, programming models and mixed programming, and large access to HPC applications viewed as Web application services.

2.   CONCEPTUAL AND DESIGN MODELS

The following presentation is based on a largely accepted model for P&D programs. According to this model [1], a P&D program is a collection of communicating sequential processes. The communication can be performed by *message passing* and/or by *data sharing*, depending on the programming model adopted for program development. The model benefits from a good and well studied theoretical background, of the methods for correctness proof and performance evaluation are based.

The use of a **conceptual model** facilitates the resolution of problems in very early phases. Also, it influences the design and the implementation phases, as well as the final result. The main conceptual classes used in P&D software development [5] are based on **data decomposition** (with the famous *result parallelism* model) and on **functional decomposition** (where we find the agenda of activities – very often implemented as *replicated workers* - and *specialist ensemble* – with two instances known as the *pipeline* model and the *network of processes*).

In the case of Genetic Algorithms (GAs), the parallelization can use any of the two approaches. A simple way to follow the first approach is to parallelize the loop that produces the new generation of individuals from the previous one. This is the **functional decomposition** of the problem. In the **domain decomposition** the population is divided into several quasi-independent sub-populations that occasionally interact. According to the size of sub-populations, we find fine grain PGAs and coarse grain PGAs. Individuals can periodically **migrate** from one population to another. The migration among sub-populations can be synchronized (Synchronous Island PGAs) or not (Asynchronous Island PGAs).

In [3] very good results have been reported due to the use of a **combined data / functional decomposition**. The **Asynchronous Island PGAs** has been selected since this solution is closer to the kind of migration found in nature. On the other hand, the **replicated workers** paradigm (that corresponds to a functional decomposition) has been selected to ease the algorithm implementation. The PGA is composed of one **master** process and several **worker** processes (see Fig. 2). The development of the PGA uses the shared objects concept, as found in Orca [10]. Two synchronization schemes have been identified in the program structure. One scheme is devoted to execution coordination of master and worker processes. It uses two shared objects, **go** and **WorkersActive** in order to implement barrier synchronizations. The second scheme is devoted to information sharing among processes. It uses the **Com** shared objects, for exchanging the best individuals (migrators) among workers.
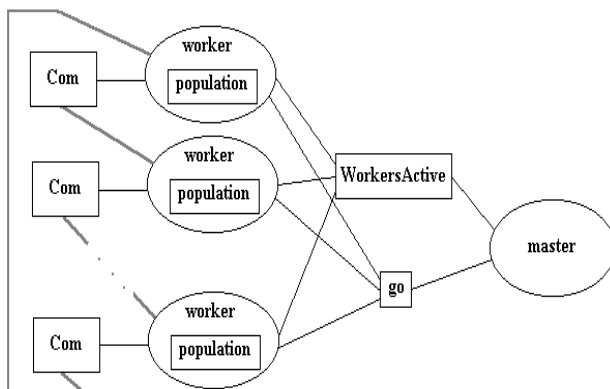

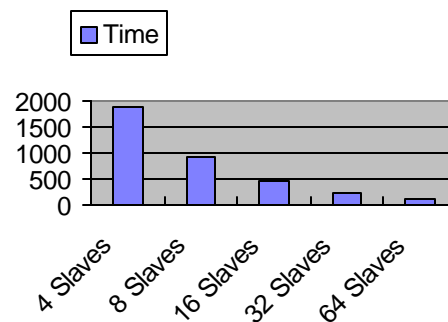Fig. 2. PGA combined domain / functional decomposition

Figure 3. Execution time (sec) versus number of processors for an eight 3-CCC graph with a total of 192 nodes

The master process starts the execution of the slaves, waits for their termination, and chooses the best solution. Each worker process executes the genetic algorithm; it initializes its own sub-population and works on it. Periodically, the worker puts in the **Com** buffer some individuals that other workers in the structure can get and use for the next iterations. Depending on how processes read the **Com** buffers, several topologies can be constructed: star, ring (figured in the picture), mesh, n-cube etc. Static and dynamic connection schemes can be easily implemented.

The PGA has been used for solving the graph partition problem. Several graphs have been considered for experiments: two separate 3-Cube-Connected Cycles (3-CCC) with a total of 48 nodes; four 3-CCC with

96 nodes and two extra links; eight 3-CCC with 192 nodes and four extra links. In all these cases, the minimum cut size is zero. Each run has been repeated ten times and the average running time has been calculated. The experimental results have been obtained by running the genetic program on a collection of 64 computers connected by an Ethernet sub-network. For the eight 3-CCC problem, the minimum cut size was obtained in ten percent of the experiments. These results are superior to the best ones found in the literature [8].

3. PERFORMANCE ANALYSIS MODELS

Any software process aims to produce correct and good performance programs. For P&D programs, performance cannot be characterized by a single figure. It is related with the application category and may include: execution time, requested memory, number of processors, scalability, reliability, fault tolerance, etc. We refer here to program execution time.

The *total execution time* T of a P&D program is the time that elapses from when the first processor starts executing on the problem to when the last processor completes execution. It is expressed as the sum of the computation time, T_{comp}, communication time, T_{commun}, and idle time, T_{idle}, over the number of processors, P:

$$T = (1/P) (T_{comp} + T_{commun} + T_{idle}) = (1/P) (\Sigma_{i=0,P-1} T^i_{comp} + \Sigma_{i=0,P-1} T^i_{commun} + \Sigma_{i=0,P-1} T^i_{idle})$$

The *computation time* depends on problem size N, on the number of tasks or processors, and on the characteristics of processors and memory. The methods for estimating the computation time in the sequential case can be easily extended for P&D programs.
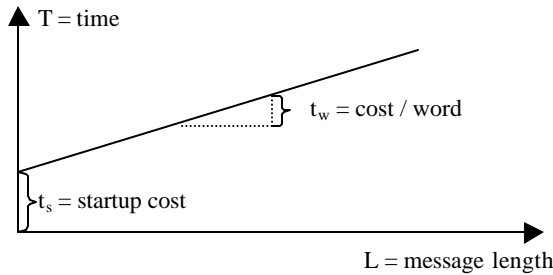
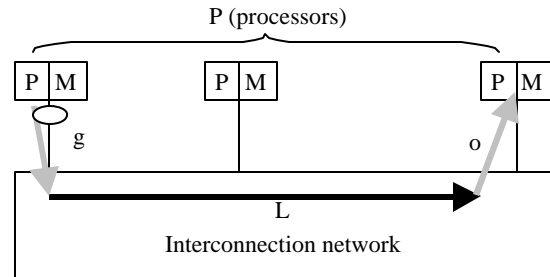

Fig. 4. Communication cost

Figure 5. The LogP model

The *communication time* is calculated by using simpler or more complicated models. In the simplest model [6], the following formula is used to compute the communication time per message, Tmsg:

$$Tmsg = t_s + t_w L$$

where t_s is a startup time for communication, and t_w is the cost for communicating a single word of a message of length L (see Fig. 4). More complicated models include several parameters. For example, the **LogP** model (Fig. 5) describes a distributed-memory multiprocessor in which processors (P) and memory (M) modules communicate through point-to-point messages and whose performance is characterized by the following parameters [18]:

- L - latency - an upper bound on the latency, or delay, incurred in communicating a message containing a small, fixed number of words from its source processor/memory module to its target.
- O - overhead – is the time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations.
- g - gap – is the minimum time interval between consecutive message transmissions / receptions at a processor. The reciprocal of g is the available per- processor communication bandwidth.
- P - number of processor / memory modules.

With this model, the transmission of a short message of length **L** between two machines takes **L+2o** time units, while the transmission of **n** messages consumes **L+2o + (n-1)g** time units. The use of the model highlights the weakness of simpler models and produces surprising results even for "standard", well-known problems. For example, the redesign of the well known broadcast algorithm of logarithmic complexity using the LogP model can save more than 20% of the execution time. In addition, a program that is designed with

the aim to minimize the total time based on the LogP model behaves well on a large spectrum of distributed configurations.

The example mentioned here refers to a parallel implementation of the *Radix Sort* algorithm [16] that exploits the LogP model. Parallel *radix sort* is a straightforward extension of a sequential radix-sort, which is based on the representation of keys as **b**-bits numbers: each digit in the key has **r** bits (r is the radix). The local radix sort performs b/r steps over the key, and each time determines the rank for each digit and permutes keys. In the parallel version, the keys are in a blocked layout across processors. The destination address of each key is obtained by computing a global histogram and the permutation involves all to all communication. Every step in the parallel radix sort has three phases for which the execution times have been calculated: multi-scan, multi-broadcast, and distribution.

This algorithm has been executed on a cluster with 8 computers in which LogP parameters were estimated as follows: $o = 5\,ms$, $L = 15\,ms$ and $g = 10\,ms$. The following execution times (expressed in the terms of the LogP model) have been obtained for the three phases of the algorithm:

multi-scan: $\quad\quad\quad\quad\quad\quad\quad$ Tms = Tms0 + 2^r * max(g, 2o + t_{add}) + (P-1) * (L + 2o + t_{add})

multi-broadcast: $\quad\quad\quad\quad\quad$ Tmb = Tmb0 + 2^r * max(g, 2o) + (P-1) * (L + 2o)

distribution: $\quad\quad\quad\quad\quad\quad$ Tdist = n * max(g, 2o + t_{addr})

which lead to the total execution time of $\quad$ T_{radix} = ceil(b/r) * (Tms + Tmb + Tdist)
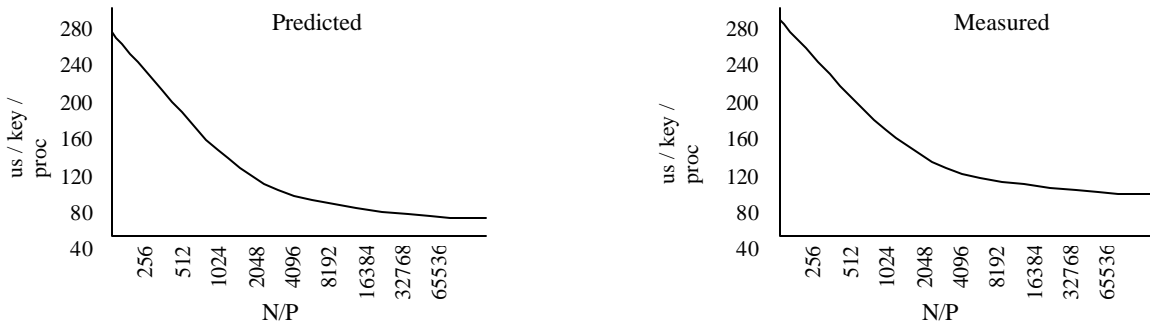

Fig. 6. Predicted and measured time/key/processor

In Fig. 6 the predicted and measured time/key/processor are presented. The experiments have been made using an MPI implementation of the algorithm, and they found that the model for evaluating communication complexity is close to the measured performance in a range of no more than 30% of the predicted values.

4.　PROGRAMMING MODEL

Usually, the developer has several options for the programming model and the corresponding programming tool: message passing in MPI, shared variables in OpenMP, threads in Java, etc. The selection criteria are: the ease of programming, ease of maintenance, portability, and, most important, the performance. MPI is designed for distributed memory and explicit message communication between processes. It has several advantages regarding the control of data placement, synchronization with procedure calls, and the very large use on machines with distributed memory and also on shared memory machines. Among disadvantages we could mention the difficulties in developing and debugging programs, additional time needed for communication (even inside the same node), and the high cost of the global operations. The other important programming tool, OpenMP, is tailored for the shared memory architecture and implicit communication. Programs are easier to implement and they make better use of the shared memory. But, the model is bound to shared memory machines and, in addition, the placement of the data on NUMA machines may create problems. In a study conducted by the author [13], a comparison has been made between MPI and OpenMP implementations, using the performance of a program for the Conjugate Gradient method. Three versions of the program have been compared: an OpenMP version, an MPI version and an optimized MPI version. In the second MPI implementation, the communication time was reduced based on a

mechanism that filters the segments of data values that must be transmitted between processes, which reduced the inter-process communication. Some results are highlighted in Fig. 7.
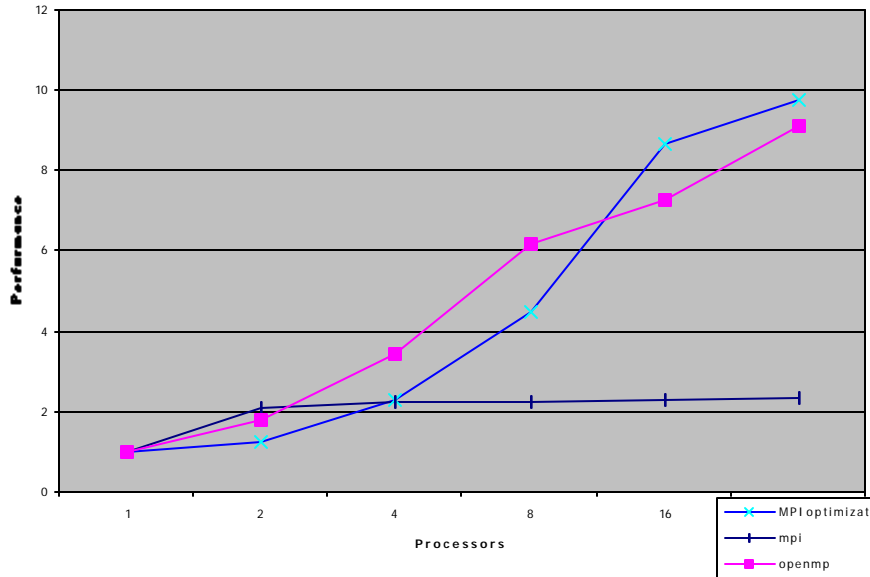

Fig. 7. Comparison of the speedup for three Conjugate Gradient program versions

As can be seen in this figure, the performance of the OpenMP version lies between the two MPI versions and it is closer to the optimized MPI version. This is due primarily to the reduction of the inter-process communication in the MPI optimized version, but also to the use of an MPI library that is tailored for the particular shared memory architecture of the HPC system used for execution.

This result has a side implication on another concept approached recently, namely the **mixed-mode** MPI-OpenMP **programming**. Some authors [19] considered that the mixed mode programming could bring important performance advantages if MPI is used for inter-node communication and OpenMP for intra-node communication. In addition, this mode could benefit from the advantages of OpenMP over MPI in some particular situations: code which scales poorly with MPI, replicated data, easier implementation, non-optimized intra-node MPI, etc. The results mentioned above (and other experiments developed in CoLaborator for the evaluation of the mixed mode programming) strengthen the opinion that mixed mode programming makes sense when the MPI implementation is poor and non-optimized for the configuration it runs on. In addition, the programs resulting from the mixed mode programming have a restricted portability.

5.   WEB HPC APPLICATION SERVICES

Due to the important increase of the commercial use of HPC systems, new requirements are addressed to high performance P&D software, such as the large availability and ease of use. In the research and academic sectors, the developers were traditionally the main users of the programs that were produced quite often for a specific application. Local terminals, directly connected to the HPC systems, provided the access to such specialized applications. The extension of collaborative research in which people from distant sites are working together in the same team, or just share the results of scientific experiments induced the necessity of remote access to HPC applications for authorized users. The main problems that had to be solved, for putting in practice this idea, were: resource protection, user authorization, and communication security. An extension of the concept is the use of a general Web browser for accessing the applications. An example is presented in Fig. 8 for a program devoted to image enhancement and segmentation, developed under the author's supervision in the CoLaborator Center. [12] The user has to introduce the identification of the input image file and some execution parameters (e.g. the number of processors used for program execution). The output window shows the results and some statistics on program execution (e.g. execution
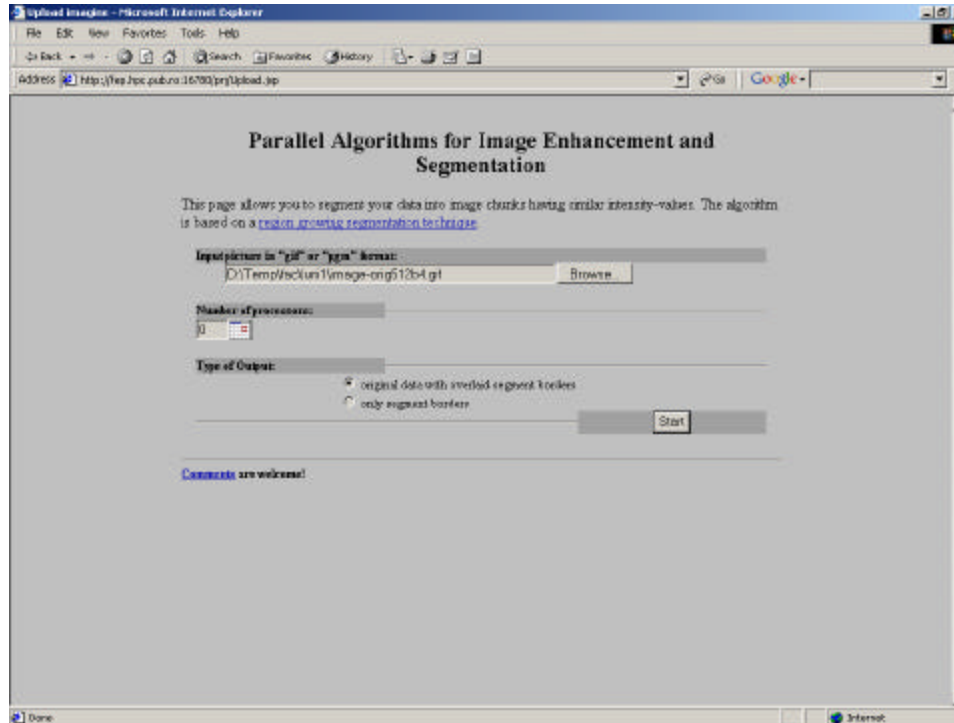
time, resource load, etc.).

Fig. 8 The access page of an HPC application

Fig. 9. Start page of the Channel Builder wizard

Web portals represent another advantageous solution for many reasons such as: a single sign-in point for users, a single access point to a collection of services, personalized presentation of the information, several channel types, etc. The portal developed at the National Center for Information Technology – CoLaborator allows the access to HPC applications, databases, collaborative services, news, consultancy, and documentation for remote and local users, through a Web interface. It incorporates several tools to support publishing and executing HPC applications [15], or building customized channels [14]. The first tool assists both the HPC application service providers and the users to publish and, respectively access HPC services. It supports the description of the application interface, automatic generation of the interfaces for the service provider and for the service user, the description of the service execution context, and the collection of statistics regarding the service utilization.

The Channel Builder offers a complete set of services for creating portal channels, channel management, and managing the resources used by different channels. This tool is portable: it is written in Java and uses the technologies J2EE, JavaServer Pages, SQL / JDBC, and XML / XSLT. The Channel

Builder is dedicated to users who create custom channels that contain static and dynamic information. The Channel Builder is ease to use, similarly to a graphic editor for Web sites.

6.   CONCLUDING REMARKS

The evolution of the HPC technology claims for new methods of software development. Advances register at the level of conceptual and design models, development tools and environments, and run time support platforms. The challenge is to increase the accessibility, reliability, scalability, security, and efficiency of complex scientific and commercial applications. This is the drive force of the research in many laboratories and centers, including the National Center for Information Technology – CoLaborator of the University Politehnica of Bucharest.

REFERENCES

1. ANDREWS, G.R. *Foundations of Multithreaded, Parallel and Distributed Programming*, Addison Wesley 2000
2. LEOPOLD, Claudia *Parralel and distributed computing*, John Wiley & Sons, 2001
3. CRISTEA, V., GODZA, G. *Genetic Algorithms and Intrinsic Parallel Characteristics*, In Proc. of Congress on Evolutionary Computation (CEC2000), La Jolla, San Diego, CA, 16-19 July, 2000, p.431-436.
4. GODZA, G., CRISTEA, V. *Comparative Study of COW and SMP Computer Configurations*, Proceedings of the International Conference on Parallel Computing in Electrical Engineering - PARELEC 2002, IEEE Computer Society, Los Alamos, California, pp 205-210
5. CARRIERO, N., GELERNTER, D. *How to Write Parallel Programs: A Guide to the Perplexed*, ACM Computing Surveys, Vol.21, N0.3, 1989
6. FOSTER, I. *Designing and Building Parallel Programs*, Addison Wesley Publishing Company, 1995
7. HAUSER, R., MANNER, R., *Implementation of Standard Genetic Algorithm on MIMD Machines* in Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner (Eds.) Parallel Problem Solving from Nature - PPSN III, Springer Verlag Berlin Heidelberg 1994
8. LIN, S.C., PUNCH, W.F., GOODMAN, E.D. *Coarse Grain Parallel Genetic Algorithms: Categorization and New Approach* Accepted for publication, Parallel&Distributed Processing, Dallas TX, Oct 1994
9. TOP 500 Supercomputer Sites, http://www.top500.org/
10. BAL, H.E., ALLIS, L.V., *Parallel Retrograde Analysis on a Distributed System*, Vrije Universiteit, Amsterdam, 1995
11. CRISTEA, V. *Parallel and Distributed Software Engineering*, in Proceedings of the International Symposium "The Role of Academic Education and Research in the Development of Information Society", Bucharest May 18-19, 2000, pp. 57-80
12. MATEI, C *Parallel algorithms for image enhancement and segmentation*, Research report, CoLaborator UPB, 2002 (Coordinator CRISTEA, V.)
13. ROSCA, R. *Parallel algorithms for the Conjugate Gradient method*, Research report, CoLaborator UPB, 2002 (Coordinator CRISTEA, V.)
14. NENCIU, D. *Channel Builder*, Research report, CoLaborator UPB, 2003 (Coordinator CRISTEA, V.)
15. GRIGORE, M. uPortal *Channels for High Performance Applications*, Research report, CoLaborator UPB, 2003 (Coordinator CRISTEA, V.)
16. CRISTEA, V., GODZA, G. *Developing Quality MPI Programs*, Proceedings of the 13th International Conference on Control Systems and Computer Science, CSCS13, Bucharest, May 31 – June 2, 2001, pp 296-303
17. CRISTEA, V. *A Collaborative Environment for High Performance Computing,* in Advanced Environments, Tools, and Applications for Cluster Computing, NATO Advanced Research Workshop, IWCC 2001, Mangalia, Romania, September 2001 Revised Papers, Springer-Verlag 2002, pp 47-60
18. CULLER, D., et al. *LogP: Towards a Realistic Model of Parallel Computation*, In Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993
19. SMITH, L.A. *Mixed Mode MPI / OpenMP Programming*, Edinburgh Parallel Computing Centre, Edinburgh, EH9 3JZ