A PARALLEL COMPUTING FRAMEWORK AND A MODULAR COLLABORATIVE CFD WORKBENCH IN JAVA

ABSTRACT

The aim of this paper is to give means for writing parallel programs and to transform sequential/shared memory programs into distributed programs, in an object-oriented environment and also to develop a parallel CFD workbench utilizing the framework. In this approach, the programmer controls the distribution of programs through control and data distribution. The authors have defined and implemented a parallel framework, including the expression of object distributions, and the transformations required to run a parallel program in a distributed environment. The authors provide programmers with a unified way to express parallelism and distribution by the use of collections storing active and passive objects. The distribution of classes/packages leads to the distribution of their elements and therefore to the generation of distributed programs. The authors have developed a full prototype to write parallel programs and to transform those programs into distributed programs with a host of about 12 functions. This prototype has been implemented with the Java language, and does not require any language extensions or modifications to the standard Java environment. The parallel program is utilized by developing a CFD workbench equipped with high end FEM unstructured mesh generation and flow solving tools with an easy-to-use GUI implemented entirely on the parallel framework.

KEY WORDS

Java, parallel computing, unstructured mesh, CFD, computational efficiency

1. Introduction

The chief aim here is to provide a few tracks in the use and development of an environment or more specifically a programming framework for the development of CFD engineering software with parallel approaches. Many authors have shown the strength of the approach in different fields of mechanics, including parallel and/or CFD computations: e.g. a study of a transient model of fluid mechanics fully coupled to an electrochemistry model in [1] , some object-oriented techniques dedicated to CFD in[2], a finite element model for modeling heat and mass transfer using the Diffpack library in [3], an arbitrary Lagrangian-Eulerian stabilized formulation for hydrodynamics with shock capturing techniques in [4] , a use of the PVM library to parallelize explicit computations in structural dynamics in [5] , a general framework for managing parallel simulations based on domain decomposition methods in [6], etc. Following a similar path, the authors have developed approaches to realize the design of finite element formulations and corresponding numerical codes by the way of symbolic concepts ([7, 8]). In [9], the problem of the utilization of Java for numerical computation in the industrial real life problems is raised up, and no definitive response is brought probably because of lack of experiments in the domain. One aim of the present work is to give an example of large scale coding in java significantly more complex than sequential programming. The idea of this work is to develop a pure JAVA framework for finite elements or finite volume parallel computations. In this paper, the authors would like to describe some aspects of developing an application in Java for domain decomposition in CFD with examples and proves of data convergence and comparative speedups taking into account another problem of some computational complexity all along using the authors' parallel framework. To begin, the authors show some pure performance comparison tests between C/C++ and JAVA on a classical matrix/vector product and data convergence with a program written for calculating lift and drag over a NACA -0012 aerofoil (using Lifting-Line theory). At last, the authors show a tentative development for an overlapping domain decomposition method for the Navier-Stokes problem implemented entirely on the framework to illustrate the capabilities of the framework. The library named JPE includes an easy and intuitive programming model based on distributed threads; object-based, message-passing APIs; and distributable data collection.

2. Computational Problems and the Approach

Roughly speaking, we distinguish the Java programming language from the Java Virtual Machine (JVM). The JVM is an interpreter that executes the program compiled to Java byte codes. The main consequence is that a program compiled on a system can be run on all systems. This very attractive aspect could hide a major drawback especially in CFD computation: the efficiency. Most computations

in mechanics involve a large number of scalar products (elemental contributions computation, Crout reduction in direct linear systems solvers, matrix/vector products in iterative linear system solvers). Here, the same code has been tested. (Java has a C++ syntax, only memory allocation) for computation of matrix/vector products, with a direct addressing and with an indirect addressing, i.e. code respectively corresponding to v[j] = A[i][j] * x[j] and v[j] = A[i][j] * x[table[j]], where table[] enables us to address the elements in the array x[] which is often needed for multithreaded applications. It is worth noting that the code in C/C++ and JAVA are exactly the same. Various number of matrix/vector products are done, for various sizes of matrices. Results are shown on Figure 1. Products vs. (tc / tj*100) as a horizontal bar diagram.

Results are similar on different platforms (Windows XP x Tru 64 Dec-Unix on a 4 processors EV6 – Version 1.3.0 and 1.4.2 version for JAVA virtual machine and J2SDK1.4.2) and shows roughly speaking that Java is from 72% to 85% within the C compiled code with maximal optimization options for direct memory access, and from 65% to 82% with indirect addressing. It should be noted that with reference to Amdahl's law of speedup in parallel systems, the best results are obtained for large sized matrices. The drawn conclusion is that good performances rates can be achieved for computational tools in Java using threads. This efficiency is acceptable to develop tools for the fast design of numerical algorithms for large applications on shared/distributed memory systems.

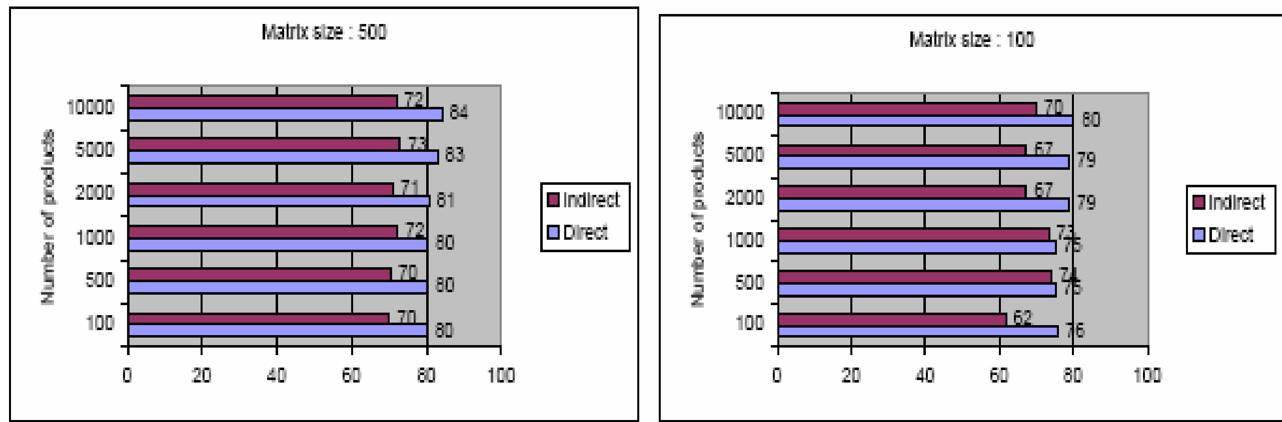

Figure 1: Comparison between C/C++ and Java code for matrix/vector multiplications using threads on single processor systems.

3. Parallel Algorithm and Approach

3.1 The Parallel Framework

Looking at MPI which has been accepted as the standard for parallel computing on distributed platforms in C, the library comes with similar functions with almost similar syntax as well as functions. The use of non-blocking communication alleviates the need for buffering since a sending process may progress after it has posted a send. Therefore the constraints of safe programming can be relaxed. However some amount of storage is consumed by a pending communication. At a minimum the communication subsystem needs to copy the parameters of a posted send or receive before the call returns. If this storage is exhausted then a call that posts a new communication will fail since post send or post receive calls are not allowed to block. A high quality implementation will consume only a fixed amount of storage per posted non-blocking communication thus supporting a large number of pending communications. The failure of a parallel program that exceeds the bounds on the number of pending non-blocking communications like the failure of a sequential program that exceeds the bound on stack size should be seen as a pathological case due either to a pathological program or a pathological implementation. Table1 lists a host of the available library functions.

Table 1: Method Specifications

int JPE_Init(int num_procs,String mother_machine)	The first and foremost of the functions that has to be called to initialize the framework. Return value is 1 if successful else returns error code(0 to 7 except 1)	Is highly dependent on the machine identifier.
int JPE_getID(void)	This method returns the local id of the machine i.e. the integer id of the current processor.	Often used in identifying processors using ids and not machine id.
int JPE_Send(datatype[] data,int size,int hid)	This method can be used to send data to another processor with id hid. (Overloaded)	The hid parameter must be correct to ensure data transfer. Available as both blocking and non-blocking.
nt JPE_Recv(datatype[] data,int size,int hid)	This method can be used to receive data from another processor with id hid.	The hid parameter must be correct to ensure data transfer. Available as both blocking and non-blocking.
int JPE_Bcast(datatype[] data,int size	This method can be used to send data to all another processors in the connection. Returns 1 if ok else 0-7 except 1 in case of errors.	Comes in two formats –blocking and non-blocking. Available as both blocking and non-blocking.
int JPE_iAllReduce(int data,int operatio	This method can be used to accumulate the results obtained as a result of certain computations in each processor by the operation parameter and saved in each processor.	Similar implementations exist for short, unsigned short, unsigned int ,long ,unsigned long ,float ,unsigned float ,double, unsigned double as well as unsigned long double as well as for classes with applicable fields.
int JPE_iReduce(int data,int operation,int hid)	This method can be used to accumulate the results obtained as a result of certain computations in each processor by the operation parameter and saved in the target processor given by the parameter hid->"host id " to receive final value.	Similar implementations exist for short, unsigned short, unsigned int ,long ,unsigned long ,float ,unsigned float ,double, unsigned double as well as unsigned long double as well as for classes with applicable fields. Available as both blocking and non-blocking.
int JPE_Finalise()	Returns 1 if ok else returns -1.	Mandatory method. Must be called at the end of every program to finalise socket connection, memory allocation,mem_buf allocation flags etc.

4. Results

4.1 Data Convergence

Here to verify the convergence of the local and global solutions and to yield a satisfactory result that satisfies both advantages of time and space complexity, the authors have considered the computation of drag and lift (along with pressure) distribution on a NACA-0012 aerofoil at a given angle of attack, free-stream conditions etc. making use of the thin aerofoil theory. Figure 1 shows the plot of lift coefficient along a NACA-0012 aerofoil for various numbers of processors against the serial code. The plot indicates the excellent converging behavior of the parallel code. As said before the library takes care of the data transfer so that round –off errors as well as data encryption errors are avoided ensuring exact data transfer. Again, system transparency enables the similar code to be executed with similar proficiency on any platform irrespective of the o.s implementations.

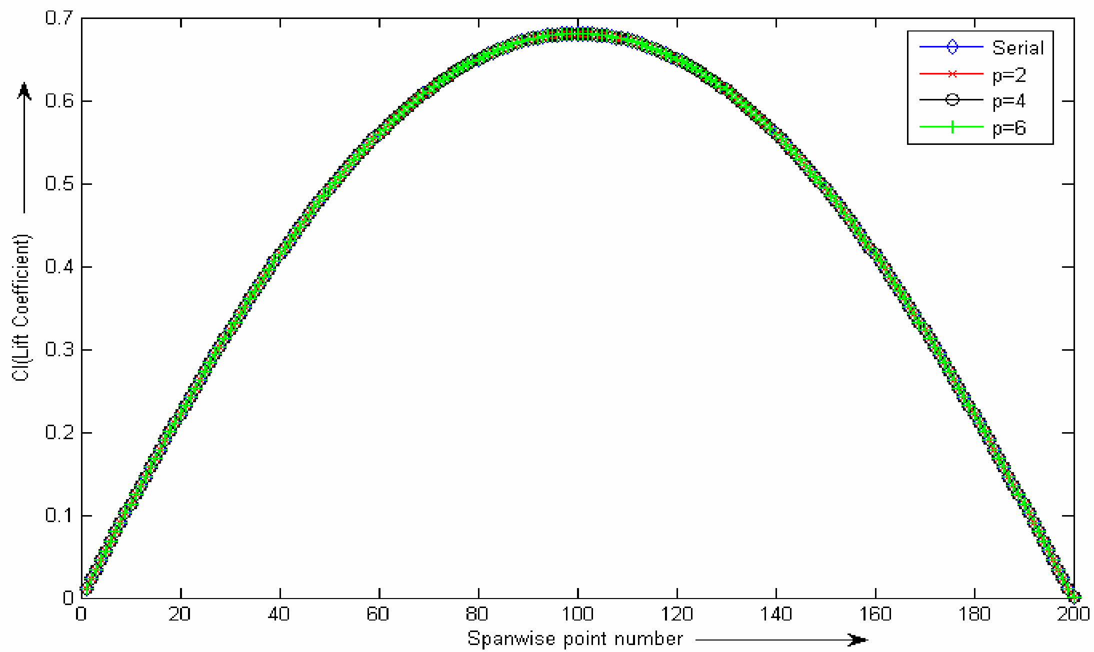

Figure 1: Comparison between data generated between serial and processors (p) =2, 4 and 6 parallel algorithm.

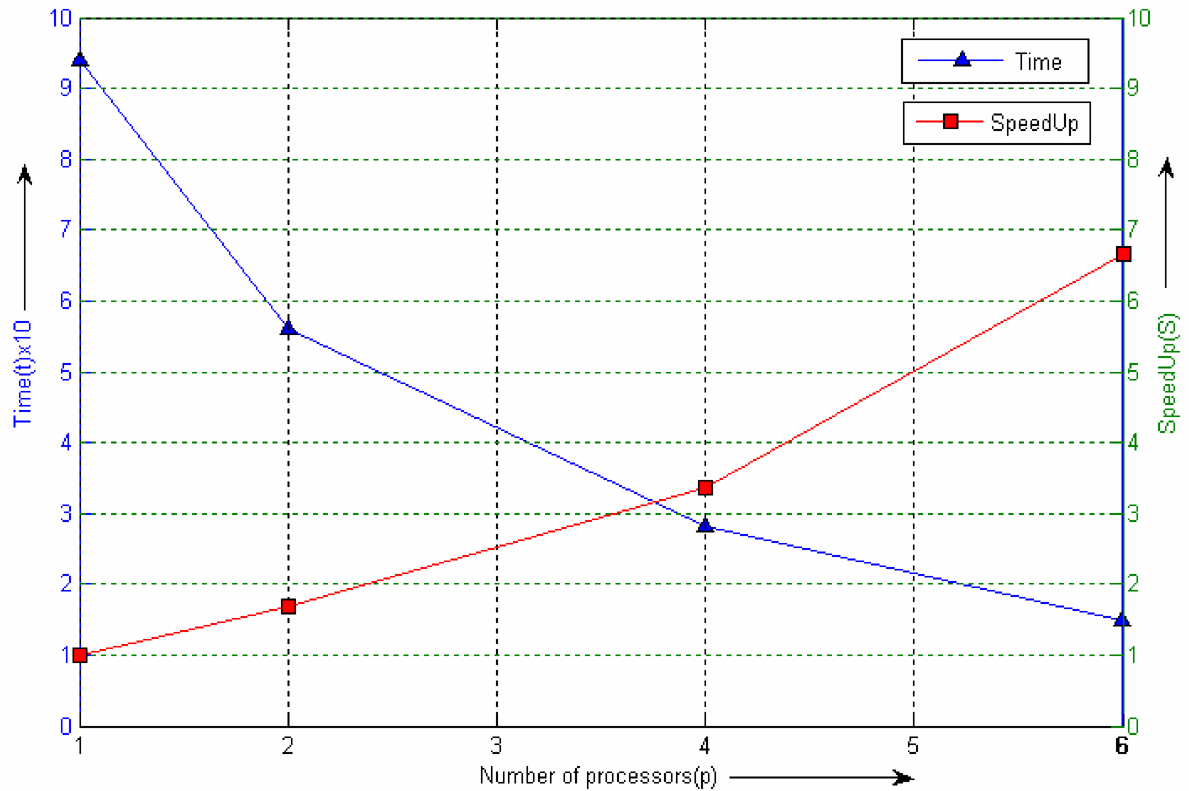

Figure 2: Computational efficiency (Speedup) achieved in case of time dependent solutions over increasing number of processors

4.2 Speed Up

In Java, parallel programming is embedded into the language. The key point of this kind of programming is the class JDC present in the package JPE. The question is then to check the performance of this class in the context of a CFD code. The test done here is to parallelize an unstructured mesh generation algorithm: the code being tested on Linux systems-Intel-80386 Pentium processor s.Figure 2 show the speedup achieved over number of processors for a mesh size of 160,000 triangular elements. The time for execution varies to a very little extent from platform to platform due to different implementations of system calls.

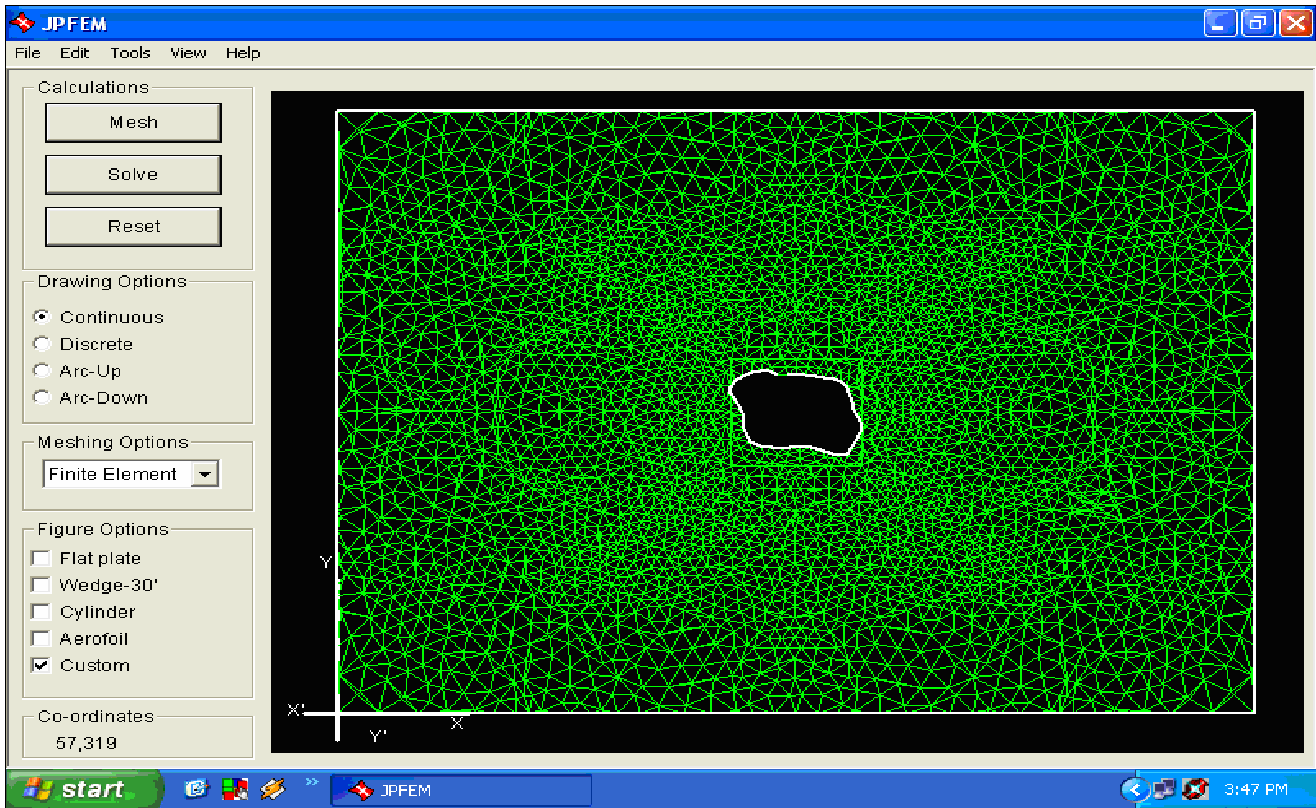

Figure 3: Screenshot showing unstructured mesh generated around an arbitrary body. (2-D)

4.3 The CFD Workbench

The workbench was written in Java and the GUI was implemented using the swing utility. The look and feel is set to platform default look by the Java code piece: '*UIManager.SetLookandFeel(default)*'. The workbench provides users with a canvas to draw arbitrary geometries as well as select certain standard features. The mesh button displays a dialog which prompts the user to select mesh fineness. Solve button displays a dialog prompting the equation to be solved and tolerance factors to be considers. The top-level menus include options to display pressure plots, streamlines and as well as vibration plots along time. The mesh generation is achieved by domain-decomposing the entire flow domain into sub-domains and distributing the computational load among the participating processors. The method presented in this paper is geometry-based, in that the geometrical data is us

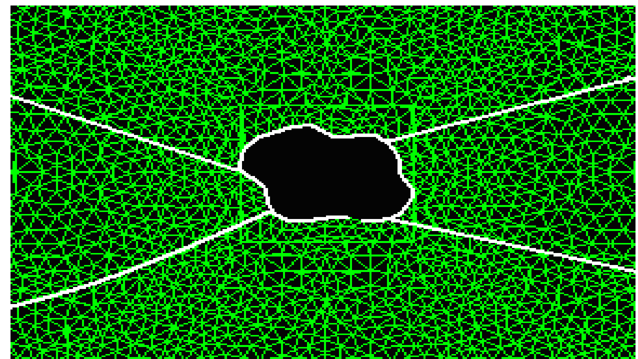

Figure 4: Mesh generated by domain-decomposition around an arbitrary body. (2-D).Boundary lines indicate load-balancing across 4 processors by geometry distribution and inter-zonal boundaries

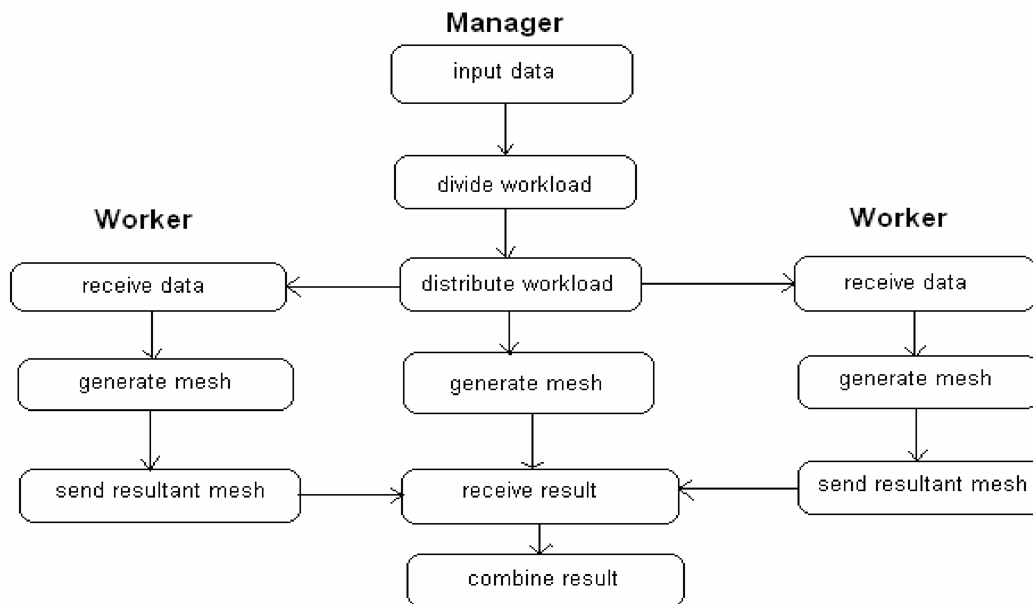

Figure 5: Manager–Worker model to distribute computational load.

5 Conclusions

The central point of this project is the development of a parallel framework for developing FEM components ,FEM discretizations , adaptivity and multi-grid solvers and their realization in a CAD software package as shown, which directly includes tools for parallelism and hardware-adapted high-performance in low level kernel routines; completely platform independent. It is the special goal in this project to realize and to optimize the algorithmic concepts used internally in the environment for specific computers (Sun Solaris, Linux/Unix) and to adapt the mathematical components to complex configurations. In this paper we have presented an expressive parallel programming model implemented by a framework in the Java language. We have not made any extension to the language. The synchronization model is very simple and will be extended in order to enlarge the application domain of our programming model.

References

[1] A. P. Peskin and G. R. Hardin, An object-oriented approach to general purpose fluid dynamics software, Computers & Chemical Engineering, Vol. 20, 1996, pp. 1043-1058.

[2] O. Munthe and H. P. Langtangen, Finite elements and object-oriented implementation techniques in computational fluid dynamics, Computer Methods Applied Mechanics and Engineering, Vol. 190, 2000, pp. 865-888.

[3] S.-H. Sun and T. R. Marrero, An object-oriented programming approach for heat and mass transfer related analyses, Computers & Chemical Engineering, Vol. 22, 1998, pp. 1381-1385.

[4] D. S. Kershaw, M. K. Prasad, M. J. Shaw and J. L. Milovich, 3D element Unstructured mesh ALE hydrodynamics with the upwind discontinuous Finite element method, Computer Methods in Applied Mechanics and Engineering, Vol. 158, 1998, pp. 81-116.

[5] P. Krysl and T. Belytschko, Object-oriented parallelization of explicit structural dynamics with PVM, Computers & Structures, Vol. 66,1998, pp. 259-273.

[6] A. S. Charao, Multiprogrammation parallèle générique des méthodes de decomposition de domaine, PhD thesis report, Institut National Polytechnique deGrenoble, 2001.

[7] D. Eyheramendy, An object-oriented hybrid Symbolic/Numerical Approach for the Development of Finite Element Codes, Finite Element Analysis and Design, vol. 36 (2000) pp. 315-334.

[8] D. Eyheramendy and Th. Zimmermann, Object-oriented Finite elements: IV.Application of symbolic derivations and automatic programming to nonlinear formulations, Computer Methods in Applied Mechanics and Engineering, vol. 190 n° 22-23 (2001) pp. 2729-2751.

[9] M. Ginsberg, J. Hauser, J. E. Moreira, R. Morgan, J. C. Parsons and T. J. Wielenga, Panel session: future directions and challenges for Javaimplementations of numeric-intensive industrial applications, Advances in Engineering Software, Vol. 31,2000, pp. 743-751.