ANNALS OF FACULTY ENGINEERING HUNEDOARA
– INTERNATIONAL JOURNAL OF ENGINEERING
Tome IX (Year 2011). Extra Fascicule. (ISSN 1584 – 2673)

[1.] Diana Alina BISTRIAN, [2.] Mihaela OSACI, [3.] Marcel TOPOR

NUMERICAL INVESTIGATION OF SWIRLING FLOWS STABILITY USING MATLAB DISTRIBUTED COMPUTING SERVER ON A WINDOWS OPERATING SYSTEM ENVIRONMENT

[1-3.] "POLITEHNICA" UNIVERSITY OF TIMISOARA, FACULTY OF ENGINEERING OF HUNEDOARA, DEPARTMENT OF ELECTRICAL ENGINEERING AND INDUSTRIAL INFORMATICS, ROMANIA

ABSTRACT: This paper presents an evaluation of the parallel processing for meshless algorithm for solving Navier-Stokes problems in fluid dynamics. The fluid dynamics existing solvers take use of finite element methods which are cumbersome and very exhaustive in computational resources. Meshless algorithms uses a different approaches using high level eigenvalue solvers. In order to evaluate the parallel processing performance, a cluster using Matlab development environment was build. Using the Matlab parallel and distributed toolbox it was possible to perform a profiling of the algorithm and to observe the eventual bottlenecks and flaws of the algorithm. Using this approach we obtained a very effective tool for processing the numerical stability investigation of the swirling flow in the Francis hydropower turbine.
KEYWORDS: Cluster technology, parallel computational algorithms, fluid dynamics spatial stability, swirling flow, meshless method

❖ CONSIDERATIONS ABOUT PARALLEL COMPUTING

Parallel programming and the design of efficient parallel programs have been well established in high-performance, scientific computing for many years. The simulation of scientific problems is an important area in natural and engineering sciences of growing importance. More precise simulations or the simulations of larger problems need greater and greater computing power and memory space. In the last decades, high-performance research included new developments in parallel hardware and software technologies [1-4] and a steady progress in parallel high-performance computing can be observed. Popular examples are simulations of weather forecast based on complex mathematical models involving partial differential equations or crash simulations from car industry based on finite element methods [5].

Other examples include drug design and computer graphics applications for film and advertising industry [6]. Depending on the specific application, computer simulation is the main method to obtain the desired result or it is used to replace or enhance physical experiments. A typical example for the first application area is weather forecast where the future development in the atmosphere has to be predicted, which can only be obtained by simulations [7]. In the second application area, computer simulations are used to obtain results that are more precise than results from practical experiments or that can be performed with less financial effort. An example is the use of simulations to determine the air resistance of vehicles [8]. Compared to a classical wind tunnel experiment, a computer simulation can give more precise results because the relative movement of the vehicle in relation to the ground can be included in the simulation. This is not possible in the wind tunnel, since the vehicle cannot be moved. Crash tests of vehicles are an obvious example where computer simulations can be performed with less financial effort.

Computer simulations often require a large computational effort. A low performance of the computer system used can restrict the simulations and the accuracy of the results obtained significantly. In particular, using a high-performance system allows larger simulations which lead to better results. Therefore, parallel computers have often been used to perform computer simulations. Today, cluster systems built up from server nodes are widely available and are now often used for parallel simulations. To use parallel computers or cluster systems, the computations to be performed must be partitioned into several parts which are assigned to the parallel resources for execution. These computation parts should be independent of each other, and the algorithm performed must provide enough independent computations to be suitable for a parallel execution. This is normally the case for scientific simulations. To obtain a parallel program, the algorithm must be formulated in a suitable programming language. Parallel execution is often controlled by specific runtime libraries or compiler directives which are added to a standard programming language like C, Fortran or Java [9-13].

MATLAB [14] is currently the dominant language of technical computing with approximately one million users worldwide, many of whom can benefit from the increased power offered by widely available multicore processors and multinode computing clusters. MATLAB is also an ideal environment for learning about parallel computing, allowing the user to focus on parallel algorithms instead of the details of the implementation.

Higham & Higham [15] and Moler [16] offer an introduction to MATLAB in their surveys. In [17] Jeremy Kepner introduces the theory, algorithmic notation, and an „under the hood" view of distributed array programming. He discusses metrics for evaluating performance and coding of a parallel program. A selected survey of parallel application analysis techniques are presented in this book, with a particular emphasis on how the examples used in the book relate to many wider application domains. In [18], Rauber and Runger take up the new development in processor architecture by giving a detailed description of important parallel programming techniques that are necessary for developing efficient programs for multicore processors as well as for parallel cluster systems or supercomputers. Both shared and distributed address space architectures are covered.

Parallel programming is an important aspect of high-performance scientific computing but it used to be a niche within the entire field of hardware and software products. However, more recently parallel programming has left this niche and will become the mainstream of software development techniques due to a radical change in hardware technology. Major chip manufacturers have started to produce processors with several power efficient computing units on one chip, which have an independent control and can access the same memory concurrently. Normally, the term core is used for single computing units and the term multicore [19, 20] is used for the entire processor having several cores. Thus, using multicore processors makes each desktop computer a small parallel system. The technological development toward multicore processors was forced by physical reasons, since the clock speed of chips with more and more transistors cannot be increased at the previous rate without overheating. Multicore architectures in the form of single multicore processors, shared memory systems of several multicore processors, or clusters of multicore processors with a hierarchical interconnection network will have a large impact on software development. In 2009, dual-core and quad-core processors are standard for normal desktop computers, and chip manufacturers have already announced the introduction of oct-core processors for 2010. It can be predicted from Moore's law [21] that the number of cores per processor chip will double every 18–24 months. According to a report of Intel, in 2015 a typical processor chip will likely consist of dozens up to hundreds of cores where a part of the cores will be dedicated to specific purposes like network management, encryption and decryption, or graphics [22].

As computational fluid dynamics (CFD) matures so rise the expectations of what it can or should deliver. Practitioners and designers are no longer content with qualitative statements on trends, but judge the utility and value of CFD by its ability to provide quantitatively accurate predictions for property fields and engineering parameters derived therefrom. In the extreme, theoretical fluid dynamicists expect fully-resolved simulations of turbulence to provide fundamental information on turbulence mechanics of greater accuracy and detail than can be derived from the most sophisticated experimental techniques.

In computational fluid dynamics, several software packages solving either the Euler or the Navier-Stokes equations around complex geometries have been developed and are currently used by aircraft or engine manufacturers. The use of unstructured meshes is one of the most efficient ways to solve complex problems, because they allow high flexibility in specifying a geometry. The possibility of using the modern parallel machines even for unstructured mesh based codes implies the necessity of dealing with two problems: the first one lies on the strategies and algorithms which can be used for the partitioning of the grid and the mapping of the subdomains among the processors; the second one is related to the structure of the code, which has to be designed is such a way that the features of the modern parallel machines are fully exploited.

The first efficient implementation of a 2D unstructured flow solver on MIMD parallel computers was presented by Venkatakrishnan et al [23] in 1992: they demonstrated that a good supercomputer performance can be reached and that a careful implementation of the message passing routines is a critical point, even for an explicit code. In 1995, Lanteri [24] developed a parallel version of an industrial code based on a mixed finite element/finite volume method. The parallelization strategy combines mesh partitioning and message passing programming model: basically, the same old serial code is going to be executed within every subdomain. Modifications occurred in the main loop in order to take into account the assembly phases of the subdomain results.

The literature on these topics can be considered exhaustive for two-dimensional applications and parallel machines of the old generation. The same cannot be said for three-dimensional complex flows and for the new machines in terms of optimization of the performances and assessment of algorithms and techniques (i.e. [25-28]).

❖ CLUSTER CONFIGURATION

In this paper we have build and tested a small scale cluster based on a Matlab Parallel Processing Toolbox. Using the internal cluster manager from Matlab we were able to evaluate the algorithm behaviour using a distributed process. In this situations we have perform the profiling and we have noticed a speed increase of the algorithm compared to a single computer run. The cluster was conceived using homogenous hardware: Dell Optiplex 755, Intel(R) Core(TM)2 Duo CPU, 2.66GHz 1.97 GHz, 1.95 GB of RAM.

MATLAB Distributed Computing Server (MDCS) is a toolbox that lets users solve computationally and data-intensive problems by executing MATLAB and Simulink based applications on a computer cluster.

For numerical investigation of swirling flows stability, a Windows operating system cluster was configured in Computer Aided Mathematics and Numerical Analysis Laboratory of the Engineering Faculty of Hunedoara (Figure 1).

Figure 1. Computer Aided Mathematics and Numerical Analysis Laboratory at the Engineering Faculty of Hunedoara.

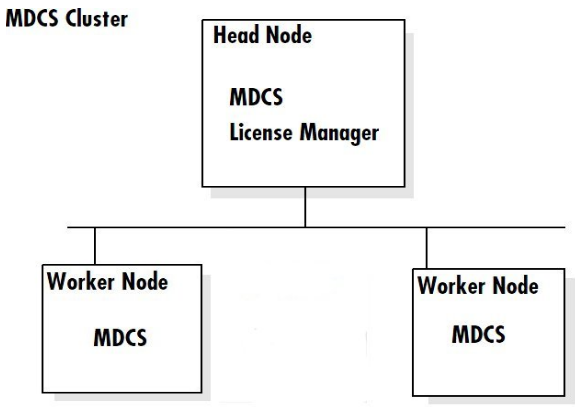
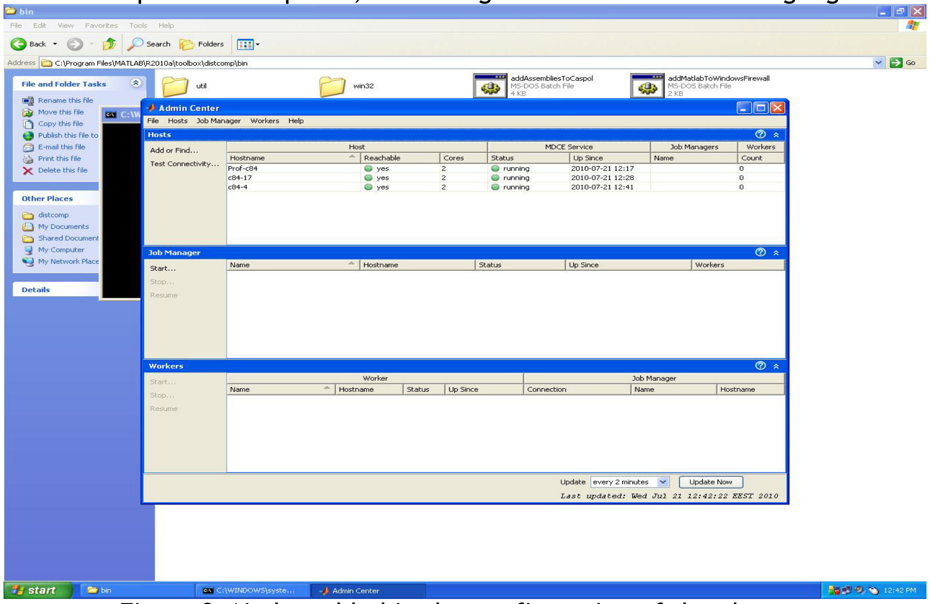


Figure 2. MDCS cluster with local access configuration.

The first step to set up the cluster configuration was to install the MDCS on a node called the head node. The license manager was also installed on the head node. After performing this installation, the MDCS was installed on the other cluster nodes, called worker nodes. Figure 2 shows the installations that was performed on our MDCS cluster nodes.

The MDCS service must be running on all machines being used for job managers or workers. This service manages the job manager and worker processes. One of the major tasks of the MDCS service is to recover job manager and worker sessions after a system crash, so that jobs and tasks are not lost as a result of such accidents. To run the MDCS, the license manager must be running on the head node. We made sure the license manager was running by performing a Status Enquiry. The next step in configuring the cluster was to start the job manager and workers. From the Admin Center, the setup and monitoring utility for the Matlab Distributed Computing Server, we select the list of the hosts in the text box. When Admin Center completed the update, the listing looked like the following Figure 3.



Hosts

Host			MDCE Service		Job Managers	Workers
Hostname	Reachable	Cores	Status	Up Since	Name	Count
Prof-c84	yes	2	running	2010-07-21 12:17		0
c84-17	yes	2	running	2010-07-21 12:28		0
c84-4	yes	2	running	2010-07-21 12:41		0

Job Manager

Name	Hostname	Status	Up Since	Workers

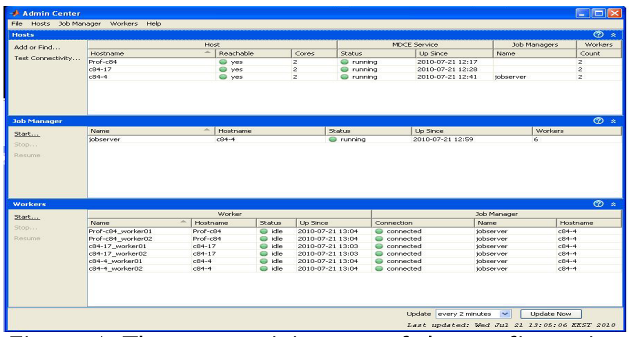
Workers

Worker				Job Manager		
Name	Hostname	Status	Up Since	Connection	Name	Hostname

Last updated: Wed Jul 21 12:42:22 EEST 2010

Figure 3. Nodes added in the configuration of the cluster.

At this point, the connectivity between nodes was tested. This assures that the cluster can perform the necessary communications for running MDCS processes. After the cluster configuration was completed, job managers have been started using the Job Manager module. In the Start Workers dialog box, we specify the number of workers to start on each host, a value that cannot exceed the total number of licenses we have. A number of two workers were set for each host. The Connectivity Testing dialog box shows the results of the last test, reported in Figure 4.



Admin Center

File Hosts Job Manager Workers Help

Hosts

Add or Find...
Test Connectivity...

Host			MDCE Service		Job Managers	Workers
Hostname	Reachable	Cores	Status	Up Since	Name	Count
Prof-c84	● yes	2	● running	2010-07-21 12:17		2
c84-17	● yes	2	● running	2010-07-21 12:28		2
c84-4	● yes	2	● running	2010-07-21 12:41	jobserver	2

Job Manager

Start...
Stop...
Resume

Name	Hostname	Status	Up Since	Workers
jobserver	c84-4	● running	2010-07-21 12:59	6

Workers

Start...
Stop...
Resume

Worker				Job Manager		
Name	Hostname	Status	Up Since	Connection	Name	Hostname
Prof-c84_worker01	Prof-c84	● idle	2010-07-21 13:04	● connected	jobserver	c84-4
Prof-c84_worker02	Prof-c84	● idle	2010-07-21 13:04	● connected	jobserver	c84-4
c84-17_worker01	c84-17	● idle	2010-07-21 13:03	● connected	jobserver	c84-4
c84-17_worker02	c84-17	● idle	2010-07-21 13:03	● connected	jobserver	c84-4
c84-4_worker01	c84-4	● idle	2010-07-21 13:04	● connected	jobserver	c84-4
c84-4_worker02	c84-4	● idle	2010-07-21 13:04	● connected	jobserver	c84-4

Update every 2 minutes Update Now

Last updated: Wed Jul 21 13:05:06 EEST 2010

Figure 4. The connectivity test of the configuration of the cluster passed.

❖ PARALLEL AND DISTRIBUTED INVESTIGATION OF THE VORTEX ROPE MODEL

The helical vortex breakdown (also known as precessing vortex rope) is a self-induced instability of a swirling flow, encountered in the draft tube cone of hydraulic Francis turbines operated far from the best efficiency. Figure 5 shows the axial and the swirl velocity profiles of the vortex rope model used in our stability analysis.

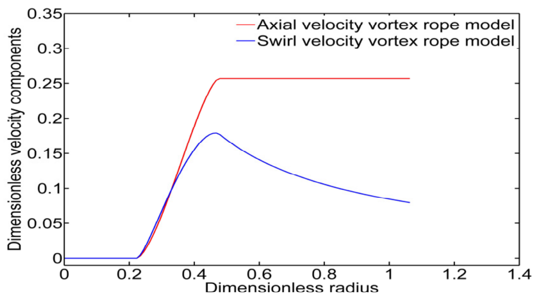


Figure 5. Axial and circumferential velocity profiles

Let us define the eigenvalue problem governing the hydrodynamic stability in operator formulation

$$\left(k\,L^{[k]}+\omega L^{[\omega]}+L\right)\mathbf{u}=0,\quad \mathbf{u}=\begin{pmatrix}F & G & H & P\end{pmatrix}^{T} \qquad (1)$$

where

$$L^{[k]}=\begin{pmatrix} r & 0 & 0 & 0\\ 0 & U & 0 & 0\\ 0 & 0 & rU & 0\\ U & 0 & 0 & 1\end{pmatrix},\quad L^{[\omega]}=\begin{pmatrix} 0 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\\ 0 & 0 & -r & 0\\ -1 & 0 & 0 & 0\end{pmatrix},\quad L=\begin{pmatrix} 0 & 1+d_r & -m & 0\\ 0 & mW/r & 2W/r & -d_r\\ 0 & W+rdW/dr & mW & m\\ mW/r & dU/dr & 0 & 0\end{pmatrix} \qquad (2)$$

The boundary relations are translated into equations that complete the computational model and can be transposed as

$$\sum_{1}^{N}(-1)^{k+1}g_k=\sum_{1}^{N}(-1)^{k+1}h_k=0\,, \qquad (3)$$

$$f_2\frac{2}{r_{\max}}+\sum_{\substack{3\\ k\ odd}}^{N}f_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ even}}^{2}(-2)\right]+\sum_{\substack{4\\ k\ even}}^{N}f_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ odd}}^{2}2+1\right]=0\,, \qquad (4)$$

$$p_2\frac{2}{r_{\max}}+\sum_{\substack{3\\ k\ odd}}^{N}p_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ even}}^{2}(-2)\right]+\sum_{\substack{4\\ k\ even}}^{N}p_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ odd}}^{2}2+1\right]=0\,, \qquad (5)$$

$$\frac{2W_{r\max}}{r_{\max}}\sum_{1}^{N}h_k-p_2\frac{2}{r_{\max}}-\sum_{\substack{3\\ k\ odd}}^{N}p_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ even}}^{2}2\right]-\sum_{\substack{4\\ k\ even}}^{N}p_k\frac{2(k-1)}{r_{\max}}\left[\sum_{\substack{r=k-1\\ k\ odd}}^{2}2+1\right]=0\,, \qquad (6)$$

$$\sum_{1}^{N}g_k=0\,,\ \left(kU_{r\max}-\omega\right)\sum_{1}^{N}h_k=0\,,\ k\left(U_{r\max}\sum_{1}^{N}f_k+\sum_{1}^{N}p_k\right)-\omega\sum_{1}^{N}f_k=0\,. \qquad (7)$$

The numerical investigation employed both the positive and the negative modes. Let us define by the critical frequency that temporal frequency corresponding to a maximum growth rate for a given mode number. The numerical results are summarized in Table 1.

Table 1. The critical frequency and the maximum growth rates obtained for the investigated modes

Mode m	−3	−2	−1	0	1	2	3
Critical Frequency	0.1	0.3	0.3	0.3	0.3	0.3	0.35
Maximum Growth Rate	14.129	11.947	6.932	0.043	0.544	0.535	0.395

Figure 6 presents the three dimensional map of the growth rate as function of the frequency and mode number and the density map of the growth rate. The negative modes present an amplitude increase depending on the mode number. However, these amplitude growth rates increase linear up to a frequency of 0.25 both for the positive and negative modes.

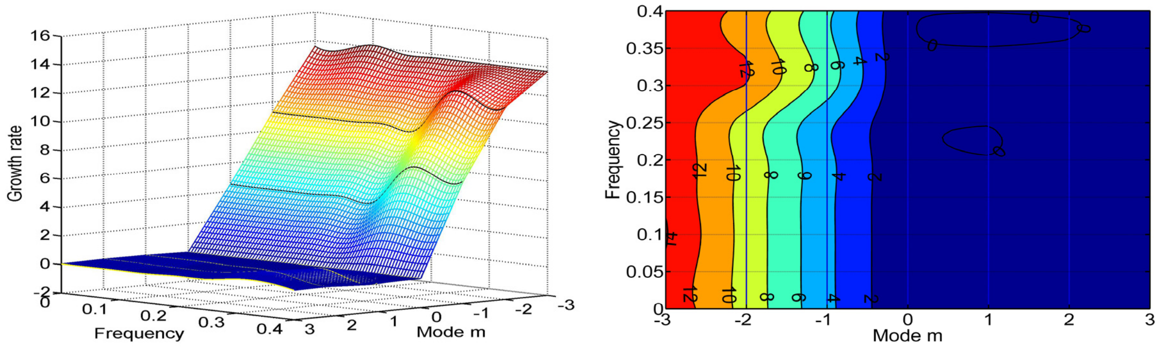


Figure 6. Plot of the growth rate as function of frequency and mode (left) and density map (right).

The convergence behavior of the collocation algorithm for $m=-1$ case is reported in Table 2 on few cluster configurations with six, four and two labs, respectively. It is noticeable that the time increase is relevant when the numerical experiments run on four labs configuration instead of 2 labs configuration. The time increase becomes irrelevant when the number of cluster nodes is increased at six labs, thus the conclusion that the cluster has no need to be extended for numerical improvements of our stability analysis.

Table 2. Elapsed time (in seconds) of the numerical computation for mode $m=-1$, on three cluster configurations.

N	Critical frequency	Elapsed time on 2 labs	Elapsed time on 4 labs	Elapsed time on 6 labs
5	0.25	1.463658	0.196285	0.211601
6	0.1	1.569234	0.225536	0.193199
8	0.1	1.584621	0.210344	0.222434
10	0.25	1.611874	0.328014	0.341408
12	0.3	1.632951	0.345925	0.360558
16	0.3	1.751369	0.431654	0.431196
17	0.3	1.836951	0.460168	0.451093
19	0.3	1.854693	0.519268	0.524420
29	0.3	2.234852	1.005637	1.015383
32	0.3	2.672955	1.479596	1.448778
37	0.3	2.895647	1.682333	1.726657
40	0.3	3.269854	2.064365	2.087408
46	0.3	4.125965	2.942432	2.948638
61	0.3	8.264985	6.453132	6.318485

❖ CONCLUSIONS

The cluster technology represents the state of the art in computational technology. This extends the limit imposed by the single computer analysis, reducing in a large manner the amount of time required for solving complex mathematical models. Using a rather usual hardware (desktop PCs) it is possible to obtain a significant acceleration of the computation process. However, the acceleration depends on the algorithm structure and it is necessary to perform a profiling of the solving process (that means to analyze carefully the behavior of the computational process- the number of steps required, the mathematical operations involved, the size of matrix variables) in order to avoid the eventual bottlenecks in the algorithm.

❖ REFERENCES

[1.] Culler, D.E., Singh, J.P., Gupta, A., Parallel Computer Architecture: A Hardware Software Approach, Morgan Kaufmann, San Francisco, 1999.
[2.] Conway, M.E., A Multiprocessor System Design, In Proceedings of the AFIPS 1963 Fall Joint Computer Conference, volume 24, pages 139–146, Spartan Books, New York, 963.
[3.] Chin, A., Complexity Models for All-Purpose Parallel Computation, In Lectures on Parallel Computation, chapter 14. Cambridge University Press, Cambridge, 1993.
[4.] Bertsekas, D.P., Tsitsiklis, J.N., Parallel and Distributed Computation, Athena Scientific, Nashua, 1997.
[5.] Braess, D., Finite Elements, 3rd edition, Cambridge University Press, Cambridge, 2007.
[6.] Krikelis, A., Parallel Multimedia Computing, Parallel Computational Fluid Dynamics Recent Developments and Advances Using Parallel Computers, Elsevier Science B.V., 1998.
[7.] Gulzow, V., Diehl, T., Foelkel, F., About the Parallelization of Climate Models, Parallel Computing: Fundamentals, Applications and New Directions, Elsevier Science B.V., 1998.
[8.] Reid, J., Supalov, A., Thole, C.A., PARASOL Interface to New Parallel Solvers for Industrial Applications, Parallel Computing: Fundamentals, Applications and New Directions, Elsevier Science B.V., 1998.
[9.] Allen, R., Kennedy, K., Optimizing Compilers for Modern Architectures, Morgan Kaufmann, San Francisco, 2002.
[10.] Bishop, P., Warren, N., JavaSpaces in Practice, Addison Wesley, Reading, 2002.
[11.] Bodin, F., Beckmann, P., Gannon, D.B., Narayana, S., Yang, S., Distributed C++: Basic Ideas for an Object Parallel Language. In Proceedings of the Supercomputing'91 Conference, pages 273–282, 1991.
[12.] Dongarra, J., Performance of various Computers using Standard Linear Equations Software in Fortran Environment. Technical Report CS-89–85, Computer Science Department, University of Tennessee, Knoxville, 1990.

[13.] Kennedy, K., Koelbel, C., Zima, H., The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In HOPL III: Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages, pages 7–1–7–22, ACM, New York, 2007.
[14.] www.mathworks.com
[15.] Higham, D.J., Higham, N.J., MATLAB Guide, Second Edition, SIAM, Philadelphia, 2005.
[16.] Moler, C., Numerical Computing with MATLAB, SIAM, Philadelphia, 2004.
[17.] Kepner, J., Parallel MATLAB for Multicore and Multinode Computers, SIAM, Philadelphia, ISBN 978-0-898716-73-3, 2009.
[18.] Rauber, T., Runger, G., Parallel Programming for Multicore and Cluster Systems, Springer-Verlag Berlin Heidelberg, ISBN 978-3-642-04817-3, 2010.
[19.] Jaja, J., An Introduction to Parallel Algorithms, Addison-Wesley, New York, 1992.
[20.] Lenoski, D.E., Weber, W., Scalable Shared-Memory Multiprocessing, Morgan Kaufmann, San Francisco, 1995.
[21.] Koch, G., Discovering Multi-core: Extending the Benefits of Moore's Law, Intel White Paper, Technology@Intel Magazine, 2005.
[22.] Kuck, D., Platform 2015 Software-Enabling Innovation in Parallelism for the Next Decade, Intel White Paper, Technology@Intel Magazine, 2005.
[23.] Venkatakrishnan, V., Simon, H.D., Barth, T., A MIMD implementation of a parallel Euler solver for unstructured grids, The J. of Supercomputing 6, pp. 117-137, 1992.
[24.] Lanteri, S., Parallel solutions of Three-Dimensional compressible flows', INRIA, Rapport de recherche n. 2594, June 1995.
[25.] Diurno, W.G., Higher Order Solution of the Compressible Viscous Flows Arising in Aerothermodynamics Using a Finite Element Method, Proc, AIDAA Congress, Roma (Italy), 11-15 September 1995.
[26.] Barth, T.J., Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier Stokes Equations, VKI - AGARD R 787, 1992.
[27.] Peraire, J., Peiro, J., Morgan, K., Multigrid Solution of the 3-D Compressible Euler Equations on Unstructured Tetrahedral grids, Int. J. Num. Meth. in Engin. 36, 1993.
[28.] Bucchignani, E., Diurno, W.G., Parallel computation of inviscid 3D flows with unstructured domain partitioning: performances on SGI-Power Challenge Supercomputer, Parallel Computing: Fundamentals, Applications and New Directions Elsevier Science B.V., 1998.