Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2009, Article ID 497013, 21 pages
doi:10.1155/2009/497013

Research Article

Simulation of Supersonic Flow in an Ejector Diffuser Using the JPVM

Carlos Couder-Castañeda

Mexican Institute of Petroleum, Department of Applied Mathematics and Computation, Eje Central Lázaro Cárdenas 152, P.O. Box 07730, Mexico City, Mexico

Correspondence should be addressed to Carlos Couder-Castañeda, ccouder@hotmail.com

Received 7 July 2009; Revised 22 September 2009; Accepted 17 December 2009

Recommended by James Buchanan

The ejectors are used commonly to extract gases in the petroleum industry where it is not possible to use an electric bomb due the explosion risk because the gases are flammable. The steam ejector is important in creating and holding a vacuum system. The goal of this job is to develop an object oriented parallel numerical code to investigate the unsteady behavior of the supersonic flow in the ejector diffuser to have an efficient computational tool that allows modeling different diffuser designs. The first step is the construction of a proper transformation of the solution space to generate a computational regular space to apply an explicit scheme. The second step, consists in developing the numerical code with an-object-oriented parallel methodology. Finally, the results obtained about the flux are satisfactory compared with the physical sensors, and the parallel paradigm used not only reduces the computational time but also shows a better maintainability, reusability, and extensibility accuracy of the code.

Copyright © 2009 Carlos Couder-Castañeda. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Steam jet ejectors offer a simple, reliable, low-cost way to produce vacuum. They are especially effective in the chemical industry where an on-site supply of the high-pressure motive gas is available. The ejector operation consists of a high-pressure motive gas that enters the steam chest at low velocity and expands through the converging-diverging nozzle. These results show a decrease in pressure and an increase in velocity. Meanwhile, the fluid enters at the suction inlet. The motive fluid, which is now at high velocity, enters and combines with the suction fluid [1, 2].

Ejector operations are classified as critical or noncritical flow mode; in the critical mode; the incoming flow is supersonic; other related experimental work and also work with CFD are in [3, 4]. This paper investigates the behavior of the supersonic flow for value

of Mach number of 2.0 in the diffuser. The form of the ejector diffuser is not rectangular; therefore, it is necessary to transform the physical plane to a computational plane where the grid is rectangular.

The implementation was done using JPVM to run the program in a collection of computing system interconnected by one or more networks as a single logical computational resource to reduce the computational time. Figure 1 shows the ejector parts commonly used in the petroleum industry. Ejectors have no moving parts and operate by the action of high-pressure incoming stream like air and other vapors at a lower pressure into the moving stream and thereby removing them from the process system at intermediate pressure.

2. The Governing Equation

We consider the two-dimensional compressible nonlinear Navier-Stokes equation written in the conservation form as

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0, \tag{2.1}$$

where U, F, and G are column vectors defined as,

$$U = \begin{cases} \rho, \\ \rho u, \\ \rho v, \\ \rho e, \end{cases}$$

$$F = \begin{cases} \rho u, \\ \rho uv + p - \tau_{xx}, \\ \rho uv - \tau_{xy}, \\ \rho u\left(e + \dfrac{u^2 + v^2}{2}\right) + pu - u\tau_{xx} - v\tau_{xy} + q_x, \end{cases} \tag{2.2}$$

$$G = \begin{cases} \rho v, \\ \rho uv - \tau_{xy}, \\ \rho v^2 + p - \tau_{yy}, \\ \rho v\left(e + \dfrac{u^2 + v^2}{2}\right) + pv - u\tau_{xy} - v\tau_{yy} + q_y. \end{cases}$$

For a calorically perfect gas, it is possibly, to eliminate the energy e in favor of u, v, p, and ρ as follows: $e = u^2 + v^2/2 + p/(\gamma - 1)\rho$. For clarity in calculations, the elements of the vector U could be denoted as

$$\begin{aligned} U_1 &= \rho, \\ U_2 &= \rho u, \\ U_3 &= \rho v, \\ U_4 &= \frac{1}{\gamma - 1} p + \frac{u^2 + v^2}{2} \rho, \end{aligned} \tag{2.3}$$

the elements of the column vector F are denoted by

$$\begin{aligned} F_1 &= \rho u, \\ F_2 &= \rho u^2 + p - \tau_{xx}, \\ F_3 &= \rho u v - \tau_{xy}, \\ F_4 &= \frac{\gamma}{\gamma - 1} p u + \rho u \frac{u^2 + v^2}{2} - u\tau_{xx} - v\tau_{xy} + q_x, \end{aligned} \tag{2.4}$$

also, the elements of the column vector G are denoted by

$$\begin{aligned} G_1 &= \rho v, \\ G_2 &= \rho u v, \\ G_3 &= \rho v^2 + p, \\ G_4 &= \frac{\gamma}{\gamma - 1} p v + \rho v \frac{u^2 + v^2}{2} - u\tau_{xy} - v\tau_{yy} + q_y, \end{aligned} \tag{2.5}$$

the viscous stress terms are written in terms of velocity gradients as

$$\begin{aligned} \tau_{xy} &= \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right), \\ \tau_{xx} &= \lambda(\nabla \cdot \mathbf{V}) + 2\mu\left(\frac{\partial u}{\partial x}\right), \\ \tau_{yy} &= \lambda(\nabla \cdot \mathbf{V}) + 2\mu\left(\frac{\partial v}{\partial y}\right). \end{aligned} \tag{2.6}$$

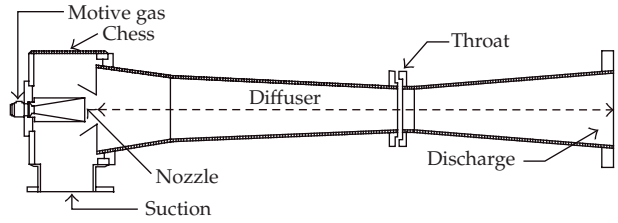

Figure 1: Ejector parts.

Likewise, the components of the heat flux vector (from Fourier's law of heat) are defined as

$$q_x = -k\frac{\partial T}{\partial x},$$
$$q_y = -k\frac{\partial T}{\partial y}. \tag{2.7}$$

The variations of the properties as dynamic viscosity and thermal conductivity are considered temperature-dependent and they are computed by Sutherland's law as

$$\mu(T) = \mu_0\left(\frac{T}{T_0}\right)^{3/2}\frac{T_0 + S}{T + S},$$
$$k(T) = \frac{\mu(T)\gamma R}{(\gamma - 1)\mathrm{Pr}}. \tag{2.8}$$

3. The Numerical Scheme

An explicit predictor-corrector scheme for (2.1) is formulated as follows.

Predictor:

$$U^*_{i,j} = U^t_{i,j} - \frac{\Delta t}{\Delta x}\left(F^t_{i+1,j} - F^t_{i,j}\right) - \frac{\Delta t}{\Delta y}\left(G^t_{i+1,j} - G^t_{i,j}\right) + S_{i,j}(U_{i,j}), \tag{3.1}$$

where the artificial viscosity S is given by

$$S^t_{i,j} = \frac{C_x\left|p^t_{i+1,j} - 2p^t_{i,j} + p^t_{i-1,j}\right|}{p^\tau_{i+1,j} + 2p^\tau_{i,j} + p^\tau_{i-1,j}}\left(U^t_{i+1,j} - 2U^t_{i,j} + U^t_{i-1,j}\right)$$
$$+ \frac{C_y\left|p^t_{i+1,j} - 2p^t_{i,j} + p^t_{i-1,j}\right|}{p^t_{i+1,j} + 2p^t_{i,j} + p^t_{i-1,j}}\left(U^t_{i,j+1} - 2U^t_{i,j} + U^t_{i,j-1}\right), \tag{3.2}$$

where the C_x and C_y are two parameters; typical values of C_x and C_y range from 0.01 to 0.3, for this application $C_x = C_y = 0.1$.

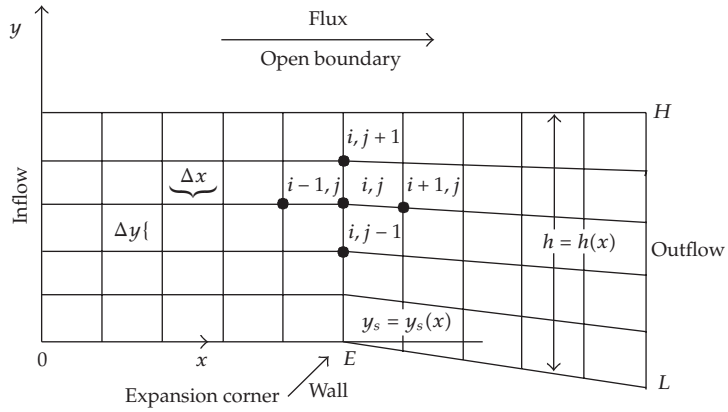

Figure 2: Physical plane.

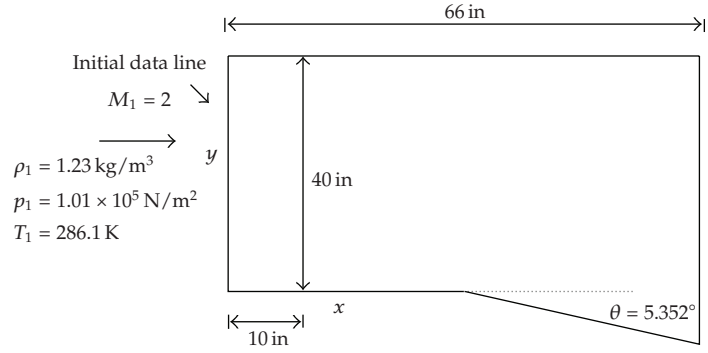

Figure 3: Expansion corner initial parameters.

Corrector:

$$U_{i,j}^{k+1} = \frac{1}{2}\left\{U_{i,j}^{t} + U_{i,j}^{*} - \frac{\Delta t}{\Delta x}\left(F_{i+1,j}^{*} - F_{i,j}^{*}\right) - \frac{\Delta t}{\Delta y}\left(G_{i+1,j}^{*} - G_{i,j}^{*}\right) + S_{i,j}\left(U_{i,j}^{*}\right)\right\}. \tag{3.3}$$

4. Validating the Numerical Code

Before managing the ejector's more complex geometry, it is possible to test the code with a simpler case with analytical solution in order to compare against the result of the numerical solution. The case is an expansion corner as sketched in Figure 2; for this problem an exact analytical solution exists in order to obtain a reasonable feeling for the accuracy of the numerical simulations result for the ejector shape; in this case in (2.1) the vector $G = 0$ and the viscous stress terms are suppressed [5].

It is necessary to establish some details of the particular problem to be solved. The physical plane drawn in Figure 3 is considered. The flow at the upstream boundary is at Mach 2.0 with a pressure, density, and temperature equal to $1 \times 10^5\,\mathrm{N/m^2}$, $1.23\,\mathrm{Kg/m^3}$ and 286 K, respectively. The supersonic flow is expanded at an angle of 5.352°; the reason for this

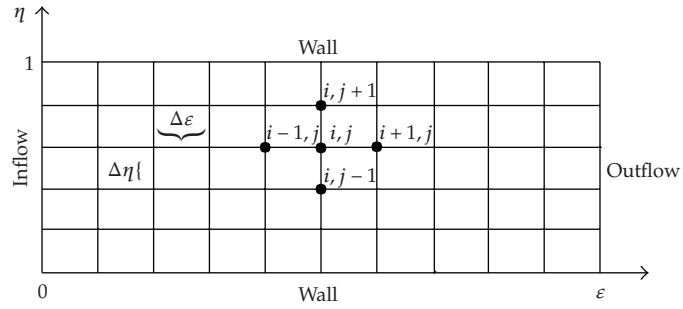

Figure 4: Computational plane.

choice is to have analytical solution. As shown in Figure 3, the calculations will be made in the domain from $x = 0$ to $x = 66$ in and from the wall from $y = 0$ to $y = 40$ in.

The location of the expansion corner is at 10 in, for this geometry; the variation of $h = h(x)$ is given by

$$h(x) = \begin{cases} 40, & \text{for } x \leq 10 \text{ in}, \\ 40 + (x - 10)\tan\theta, & \text{for } x \geq 10 \text{ in}. \end{cases} \tag{4.1}$$

We can construct a proper transformation as follows (Figure 4): let h denote the local height from the lower to the upper boundary in the physical plane, clearly $h = h(x)$. Denote the y location of the solid surface (the lower boundary in the physical plane) by y_s, where $y_s = y_s(x)$.

We define the transformation as

$$\begin{aligned} \xi &= x, \\ \eta &= \frac{y - y_s(x)}{h(x)}. \end{aligned} \tag{4.2}$$

With this transformation, in the computational plane ξ varies from 0 to L and η varies from 0.0 to 1.0; $\eta = 0.0$ corresponds to the surface in the physical plane, and $\eta = 1.0$ corresponds to the upper boundary. The lines of constant ξ and η form a rectangular grid in the computational plane (Figure 4). The lines of constant ξ and η are also sketched in the physical plane; they form a rectangular grid upstream of the corner and a networks of divergent lines downstream of the corner.

The partial differential equations for the flow are numerically solved in the rectangular space and therefore must be appropriately transformed for use in the computational plane.

That is, the governing equation must be transformed into terms dealing with ξ and η. The derivative transformation is given by the following equations:

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial \xi}\left(\frac{\partial \xi}{\partial x}\right) + \frac{\partial}{\partial \eta}\left(\frac{\partial \eta}{\partial x}\right),$$
$$\frac{\partial}{\partial y} = \frac{\partial}{\partial \xi}\left(\frac{\partial \xi}{\partial y}\right) + \frac{\partial}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right). \tag{4.3}$$

The metrics $(\partial\xi/\partial x)$, $(\partial\eta/\partial x)$, $(\partial\xi/\partial y)$, and $(\partial\eta/\partial y)$, in (4.3), are obtained from the transformation given by (4.2), that is, $\partial\xi/\partial x = 1$ and $\partial\xi/\partial y = 0$:

$$\begin{aligned}\frac{\partial \eta}{\partial x} &= \frac{\partial\left[y - y_s(x)/h(x)\right]}{\partial x} \\ &= \frac{\partial}{\partial x}\left[\frac{y}{h(x)}\right] - \frac{\partial}{\partial x}\left[\frac{y_s(x)}{h(x)}\right] \\ &= -\frac{y h'(x)}{[h(x)]^2} - \left[\frac{y_s'(x)h(x) - y_s(x)h'(x)}{[h(x)]^2}\right] \\ &= \frac{h'(x)\left[y_s(x) - y\right]}{[h(x)]^2} - \frac{y_s'(x)h(x)}{[h(x)]^2};\end{aligned} \tag{4.4}$$

if $y = h(x)\eta + y_s(x)$, then

$$\begin{aligned}\frac{\partial \eta}{\partial x} &= h'(x)\frac{\left[y_s(x) - \left[h(x)\eta + y_s(x)\right]\right]}{[h(x)]^2} - \frac{y_s'(x)}{h(x)} \\ &= \frac{h'(x)\left[-h(x)\eta\right]}{[h(x)]^2} - \frac{y_s'(x)}{h(x)} \\ &= -\frac{\eta}{h(x)}h'(x) - \frac{1}{h(x)}y_s'(x);\end{aligned} \tag{4.5}$$

if $x < 10$, then $y_s'(x) = 0$ and $h'(x)$; if $x \geq 10 = y_s'(x) - \tan\theta$ and $h'(x) = \tan\theta$, so the metrics could be finally expressed as $\partial\xi/\partial x = 1, \partial\xi/\partial y = 0$:

$$\frac{\partial \eta}{\partial x} = \begin{cases} 0, & \text{if } x < 10\,\text{in}, \\ (1-\eta)\dfrac{\tan\theta}{h(x)}, & \text{if } x > 10\,\text{in}, \\ \dfrac{\partial \eta}{\partial y} = \dfrac{1}{h(x)}. \end{cases} \tag{4.6}$$

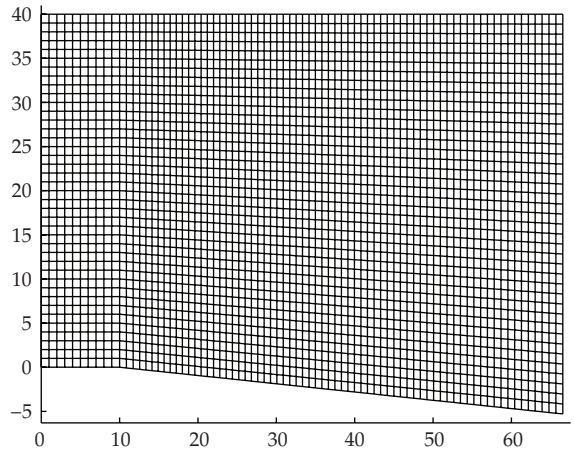

Figure 5: Mesh generated for the expansion corner.

The mesh for the expansion corner and for the ejector is automatically generated to avoid stability numerical problems using the CFL criterion where the values of $\Delta\xi_i$ are given by

$$\Delta\xi_i = C\frac{\Delta y}{\left|\tan\theta \pm \mu\right|_{\max}}, \tag{4.7}$$

where $\theta = a\sin(1/M_{i,j})$ and $\mu = a\tan(v_{i,j}/u_{i,j})$.

The mesh obtained is shown in Figure 5. The size of the ejector grid is 79×41 discrete points.

In the results of the Mach number, pressure and density for the expansion corner problems are shown in the results. The analytical solution of the Mach Number for the expansion corner could be obtained with the Prandtl-Meyer function (4.8):

$$f(M) = \sqrt{\frac{\gamma+1}{\gamma-1}}\tan^{-1}\sqrt{\frac{\gamma-1}{\gamma+1}(M^2-1)} - \tan^{-1}\sqrt{M^2-1}. \tag{4.8}$$

The analytical solution of the expansion corner is shown in Figure 6.

The leading edge of the expansion fan makes an angle μ_1 with respect to the upstream flow direction, and the trailing edge of the wave makes an angle μ_2 with respect to the downstream flow direction. The angles μ_1 and μ_2 are mach angles, defined as $\mu_1 = \sin^{-1}(1/M_1)$ and $\mu_2 = \sin^{-1}(1/M_2)$.

The numerical solution of the expansion corner is depicted in Figure 7.

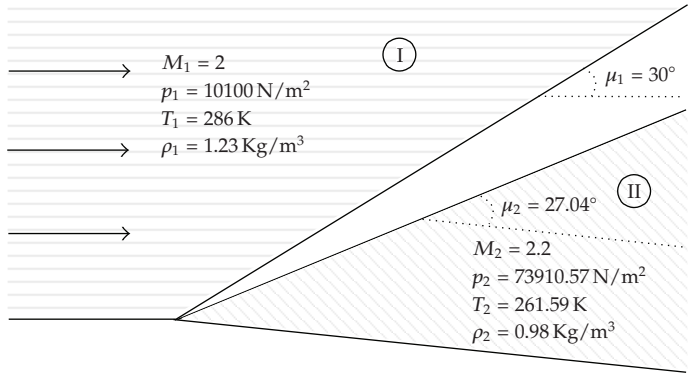

Figure 6: Analytical solution of the expansion corner.

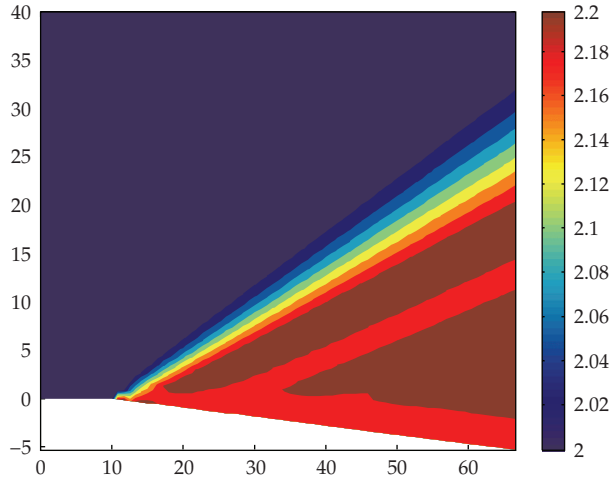

Figure 7: Numerical solution of the expansion corner.

The numerical solution matches the analytical with an error of less than 1% percent at all points and the expansion Mach fan is well formed in the numerical solution.

5. The Transformation of the Physical Ejector Diffuser

The appropriate transformation to generate a boundary-fitted coordinate system for the ejector diffuser is defined as follows:

$$\xi = x, \tag{5.1}$$

$$\eta = \frac{y - y_s(x)}{y_z(x) - y_s(x)}, \tag{5.2}$$

where $y_s(x)$ and $y_z(x)$ are defined as

$$y_s(x) = \begin{cases} 0.0, & \text{for } x \le 10, \\ \dfrac{8}{167.875}(x-10), & \text{for } 10 < x \le 177.875, \\ 8.0, & \text{for } 177.875 < x \le 299.125, \\ -\dfrac{8}{89}(x-299.125)+8, & \text{for } x > 299.125, \end{cases}$$
$$y_z(x) = \begin{cases} 42.0, & \text{for } x \le 10, \\ -\dfrac{8}{167.875}(x-10)+42, & \text{for } 10 < x \le 177.875, \\ 34.0, & \text{for } 177.875 < x \le 299.125, \\ \dfrac{8}{89}(x-299.125)+34, & \text{for } x > 299.125. \end{cases} \tag{5.3}$$

The metrics in (4.3) given by (5.1) and (5.2) are calculated as follows: $\partial\xi/\partial x = 1$, $\partial\xi/\partial y = 0$, $\partial\eta/\partial y = 1/(y_z(x) - y_s(x))$, by (5.2), $y = \eta[y_z(x) - y_s(x)] + y_s(x)$, so $\partial\eta/\partial x = (\eta[y_s'(x) - y_z'(x)] - y_s'(x))/(y_z(x) - y_s(x))$. The metric $\partial\eta/\partial x$ can be obtained differentiating $y_s(x)$ and $y_z(x)$:

$$y_s'(x) = \begin{cases} 0, & \text{for } x \le 10, \\ \dfrac{8}{167.875}, & \text{for } 10 < x \le 177.875, \\ 0, & \text{for } 177.875 < x \le 299.125, \\ -\dfrac{8}{89}, & \text{for } x > 299.125, \end{cases}$$
$$y_z'(x) = \begin{cases} 0, & \text{for } x \le 10, \\ -\dfrac{8}{167.875}, & \text{for } 10 < x \le 177.875, \\ 0, & \text{for } 177.875 < x \le 299.125, \\ \dfrac{8}{89}, & \text{for } x > 299.125. \end{cases} \tag{5.4}$$

Therefore

$$\frac{\partial\eta}{\partial x} = \begin{cases} 0.0, & \text{for } x \le 10, \\ -\dfrac{9.53\times10^{-2}\eta - 4.76\times10^{-2}}{-42.95 + 9.53\times10^{-2}x}, & \text{for } 10 < x \le 177.875, \\ 0.0, & \text{for } 177.87 < x \le 299.125, \\ -\dfrac{-0.17\eta + 8.98\times10^{-2}}{27.77 - 0.17x}, & \text{for } x > 299.125. \end{cases} \tag{5.5}$$

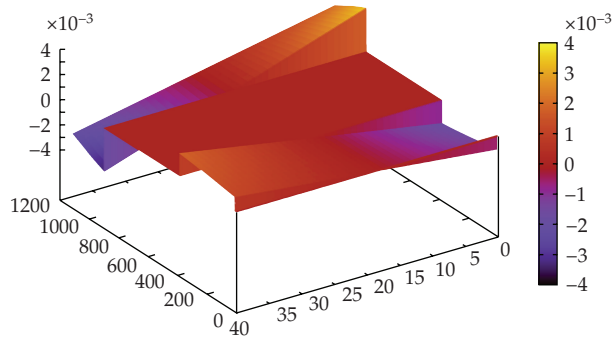

Figure 8: Distribution of $\partial\eta/\partial x$.

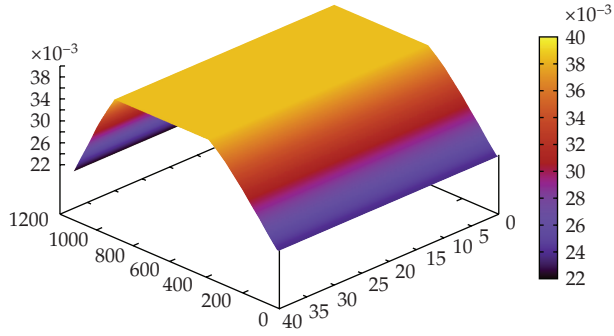

Figure 9: Distribution of $\partial\eta/\partial y$.

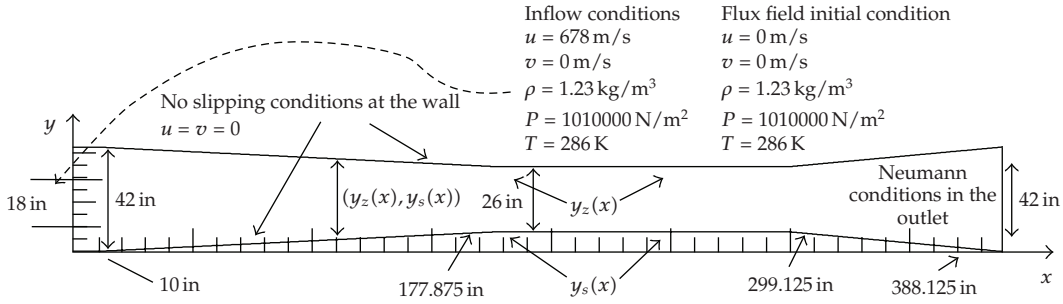

Figure 10: Ejector plane with the initial parameters.

In Figures 8 and 9 is depicted the distribution of the metrics $\partial\eta/\partial x$ and $\partial\eta/\partial y$, respectively. The size of the ejector grid is 1101 × 41 discrete points and it was generated using the CFL marching criterium.

The boundary condition in the outlet is Neumann and in the walls is applied the nonslipping condition. The flux field is initialized with a pressure, density, and temperature equal to $1 \times 10^5\,\mathrm{N/m^2}$, $1.23\,\mathrm{Kg/m^3}$, and 286 K, respectively; the velocity field is initialized with $u = v = 0.0\,\mathrm{m/s}$. In the upstream boundary a velocity of Mach 2.0 ($u = 678\,\mathrm{m/s}$ and $v = 0.0\,\mathrm{m/s}$) is injected. The simulation is performed for 5,000,000 time steps, with a Δt calculated using the CFL criterium (see Figure 10).

5.1. Viscous Stress Terms

Of course is necessary to use the metrics transformation for the viscous stress terms. Once applied the metrics the terms are expanded as follows:

$$\begin{aligned}
\tau_{xy} &= \mu\left(\frac{\partial u}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right) + \frac{\partial v}{\partial \xi} + \frac{\partial v}{\partial \xi}\left(\frac{\partial \eta}{\partial x}\right)\right),\\
\tau_{xx} &= \lambda\left(\frac{\partial u}{\partial \xi} + \frac{\partial u}{\partial \eta}\left(\frac{\partial \eta}{\partial x}\right) + \frac{\partial v}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right)\right) + 2\mu\left(\frac{\partial u}{\partial \xi} + \frac{\partial u}{\partial \eta}\left(\frac{\partial \eta}{\partial x}\right)\right),\\
\tau_{yy} &= \lambda\left(\frac{\partial u}{\partial \xi} + \frac{\partial u}{\partial \eta}\left(\frac{\partial \eta}{\partial x}\right) + \frac{\partial v}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right)\right) + 2\mu\left(\frac{\partial v}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right)\right).
\end{aligned} \tag{5.6}$$

For the heat flux vector the transformation is

$$\begin{aligned}
q_x &= -k\left(\frac{\partial T}{\partial \xi} + \frac{\partial T}{\partial \xi}\left(\frac{\partial \eta}{\partial x}\right)\right),\\
q_y &= -k\left(\frac{\partial T}{\partial \eta} + \left(\frac{\partial \eta}{\partial y}\right)\right).
\end{aligned} \tag{5.7}$$

for this application the values of $T_0 = 273K, S = 110.5, \mu_0 = 1.68 \times 10^{-5}, \gamma = 1.4$, and $\mathrm{Pr} = 0.71$ are used.

6. Parallelizing the Numerical Scheme

The JPVM (Java Parallel Virtual Machine) library is a software system for explicit message passing-based distributed memory MIMD parallel programming in Java. The library supports an interface similar to the C and FORTRAN interfaces provided by the Parallel Virtual Machine (PVM) system, but with syntax and semantics enhancements afforded by Java and better matched to Java programming styles. The similarity between JPVM and the widely used PVM system supports a quick learning curve for experienced PVM programmers, thus making the JPVM system an accessible, low-investment target for migrating parallel applications to the Java platform. At the same time, JPVM offers novel features not found in standard PVM such as thread safety, multiple communication end-points per task, and default-case direct message routing. JPVM is implemented entirely in Java and is thus highly portable among platforms supporting some version of the Java Virtual Machine. This feature opens up the possibility of utilizing resources commonly excluded from network parallel computing systems such as Mac-, Windows- and Linux-based systems [6, 7].

The method used is an explicit finite-difference technique which is second-order accurate in both space and time with artificial viscosity.

In the predictor step the governing equation is written in terms of forward differences (predictor), later on is written in backward differences (corrector). Following the scheme we can divide the computation into tasks that perform over a part of the solution space only in the axis η; so if we have m discrete points that divide the space over η, then the computation could be performed into k tasks, where the number of points in which each task operates

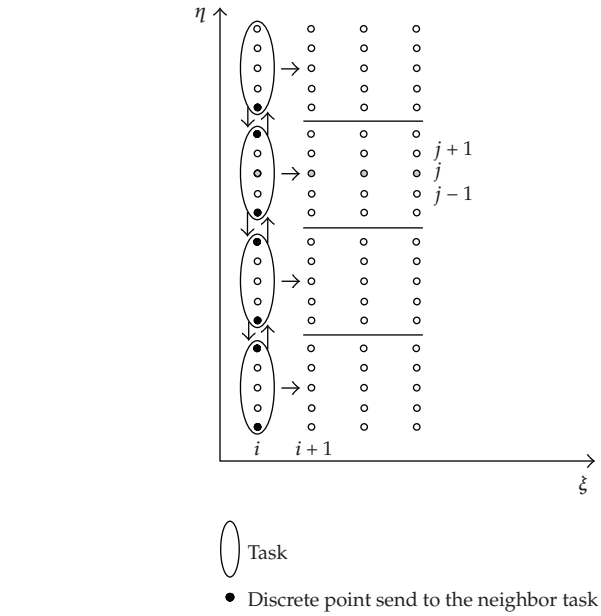

Figure 11: Fine grained.

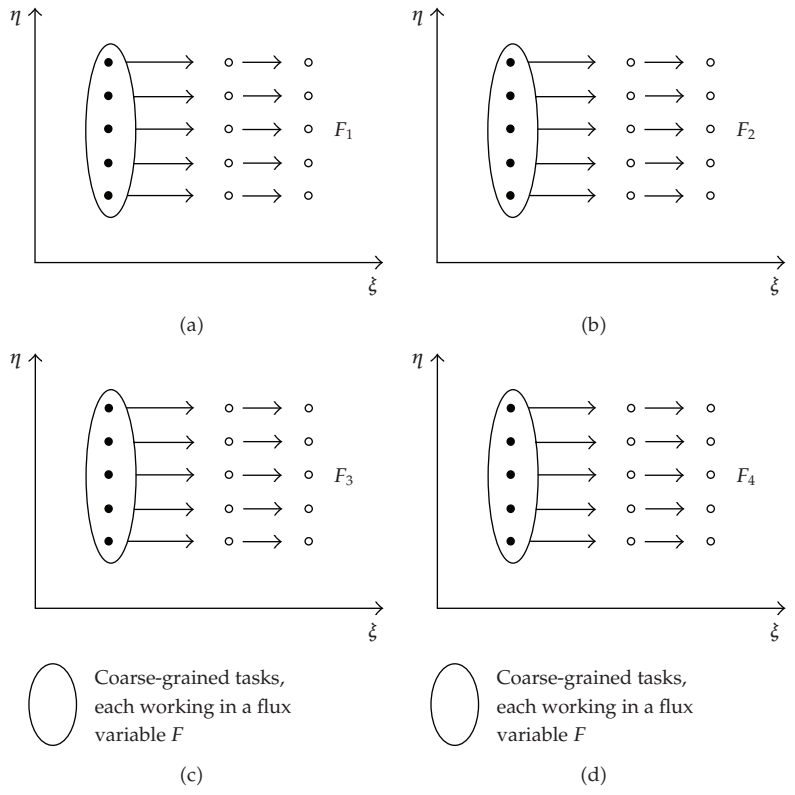

Figure 12: Coarse grained.

is m/k; if the division is not exact, then the number of remaining points will be distributed among the tasks. For example, if we have 40 points on η, and we want to divide it into 6 tasks, then there will be 4 tasks that take charge of working on 7 points and two tasks on 6 points. If the calculation is wanted to divide among 4 tasks, then each task will work on 10 points.

If $m = 40$ and $k = 4$, then the iterators j_1, j_2, j_3, j_4 of each task will work on the following points: $j_1 = 1, \ldots, 10$, $j_2 = 11, \ldots, 20$, $j_3 = 21, \ldots, 30$, $j_4 = 31, \ldots, 40$.

Figure 11 shows the set-up of four tasks working over the same solution space, the arrows that arise of the tasks indicate the messages, and the arrows ahead of each task indicate the direction of the calculation.

The main issue in a traditional parallel algorithm for finite differences problem, is the intensive message passing, obtaining a poor performance in distributed memory systems, where the latency time is higher. Suppose that we want to advance on 300 discrete points over the axis ξ and we have 4 tasks; so in every iteration it is necessary to send 6 messages; therefore it will be $300 \times 6 = 1800$ messages that holds just one value, and this is just for the case of the variable F_1. Additionally, also we have to create 3 tasks for the other flux variables F_2, F_3, F_4, that give a total of $1800 \times 4 = 7200$ messages.

To reduce the quantity of messages it is necessary to increase the tasks granularity, assigning a tasks for flux variable. Figure 12 depicted four tasks working in their own flux variable.

Creating four tasks by every term of the flux F_1, F_2, F_3, and F_4, the task q $(q = 1, 2, 3, 4)$ carries out the necessary calculations to obtain the new value of F_q at $i + 1$. The procedure is listed as follows.

(1) Calculate the forward differences of the predictor step.

(2) Calculate the artificial viscosity of the predictor step.

(3) Calculate the predicted value of F_q.

(4) Calculate the rearward differences of the corrector step, using the predicted values of F_q.

(5) Calculate the artificial viscosity for corrector step.

(6) Calculate the average derivative.

(7) Calculate the value of the flux variable F_q at $i + 1$.

Additionally to the four tasks (slave tasks), we need the master task to control the execution of the slave tasks and achieve the following calculations.

(1) Calculate $\Delta\xi$ at i to obtain the predicted values of the flux variables F_q.

(2) Apply the boundary conditions.

(3) Adjust the flux variables F at boundary.

The former procedure indicates that the work of every task has to perform over the points collection (i, j) that conform the bidimensional mesh. Every task q $(q = 1, 2, 3, 4)$ will calculate the new values of the flux variables F_q at localizations $\Delta\xi_1, \Delta\xi_1 + \Delta\xi_2, \ldots, \sum_{i=1}^{n} \Delta\xi_i$.

We follow a similar reasoning if we want to increase the granularity of tasks.

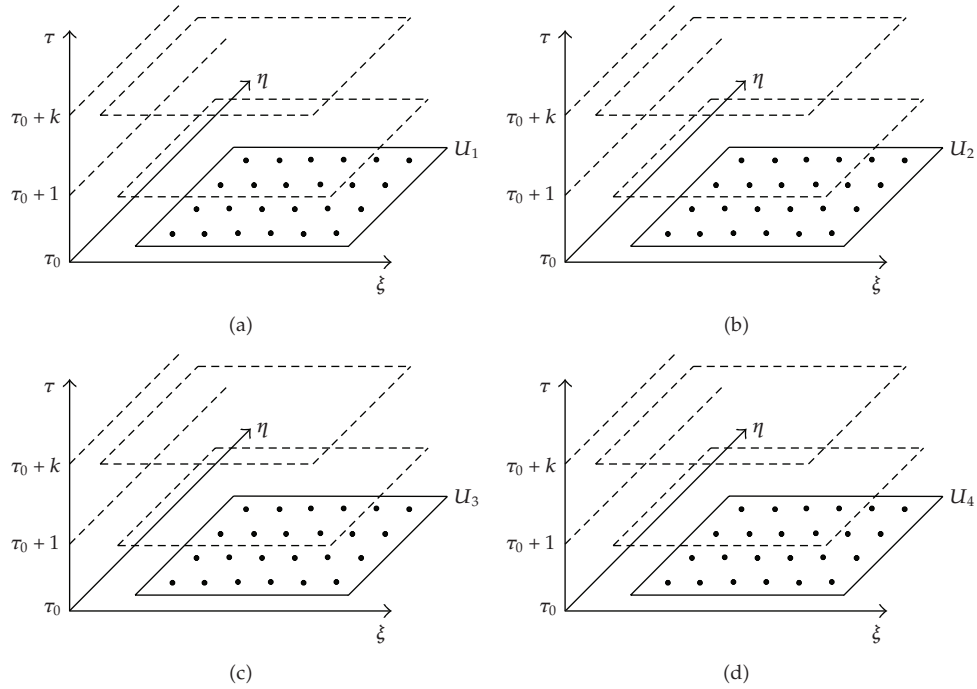

Figure 13: Coarse-grained 3D.

Figure 13 depicted the setup of four tasks working over its own flow field, in this case every task $q(q = 1, 2, 3, 4)$ carried out the calculations to find the new values at each grid point (i, j) of the variables U_q in the time $\tau_0 + 1$, $\tau_0 + 2$, ..., $\tau_0 + k$.

The advantage of this paradigm for the parallelism is that we can made quickly hybridizations between a coarse and fine granularity, and the parallel tasks are created recursively easily.

7. Simulation Results

In this section we show the results of the simulations about the expansion corner and the ejector.

7.1. Expansion Corner Results

The expansion corner results for the Mach number, density, and temperature are shown in Figures 14 and 15 in steady state.

7.2. Ejector Results

Figures 16, 18, 20, and 22 show the contour graphs of the density and the Figures 17, 19, 21, and 23 of the Mach number, after 0.2 s, 0.9 s, 1.6 s, and 5.0 s of real simulation when the flux is stable.

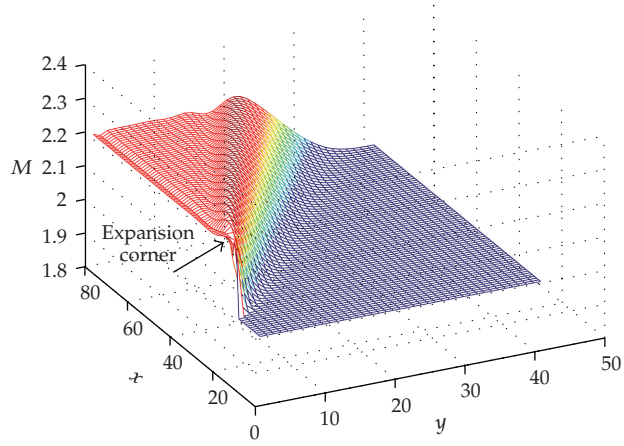

Figure 14: Mach number in the expansion corner.

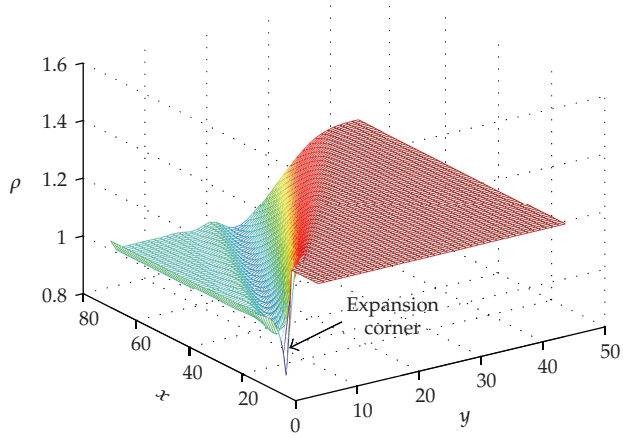

Figure 15: Density in the expansion corner.

In Figure 24, is depicted the profile of the Mach number in the different sections of the diffuser; when the compression, transfer; and expansion occur, three numerical visors are taken at 5 inches, 21 inches, and 37 inches in the vertical, and compared with the experimental processed results when the flux is stable [1].

8. Performance Results

To gain a better perspective of the performance options, the JPVM was compiled using a native compiler gcj (gnu java compiler) to avoid the overload of the virtual machine for Linux and Windows to generate a native version and carry out comparatives with the byte-code version. The creation and communication is under the same machine; a slow Pentium IV processor was used because is easier to measure the time. Table 1 shows the creation time of 1, 2, 4, 8, and 16 tasks.

In Table 1 we can see that the time of creation is practically the same in both versions.

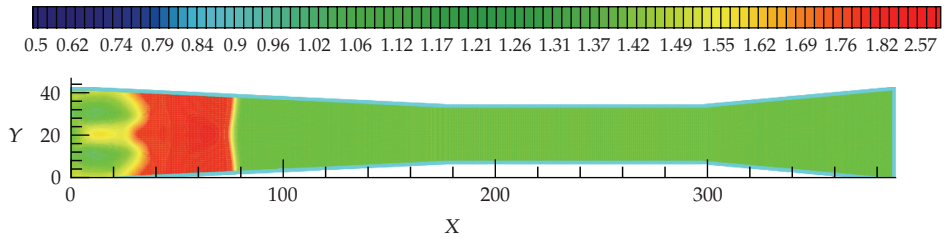

Figure 16: Density through the diffuser section at 0.2 s

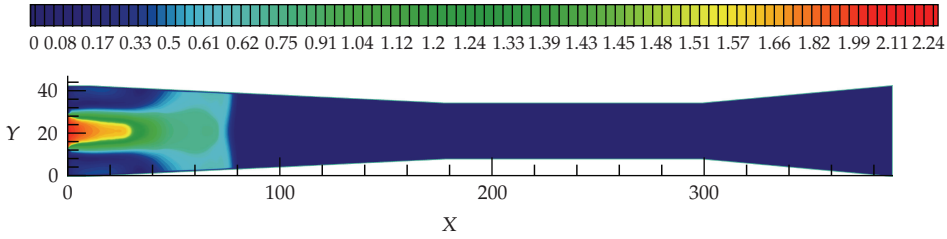

Figure 17: Mach number through the diffuser section at 0.2 s.

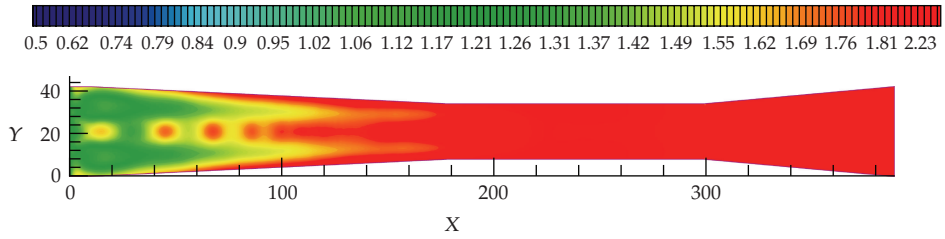

Figure 18: Density through the diffuser section at 0.9 s.

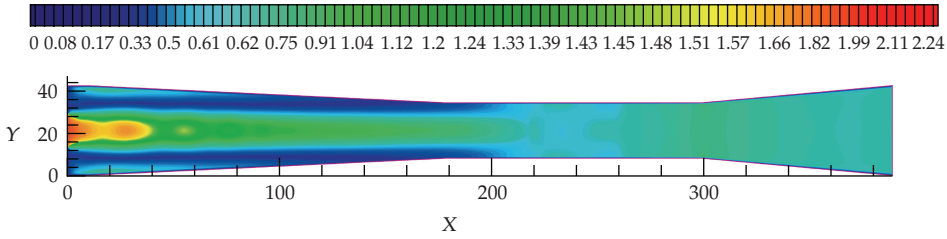

Figure 19: Mach number through the diffuser section at 0.9 s.

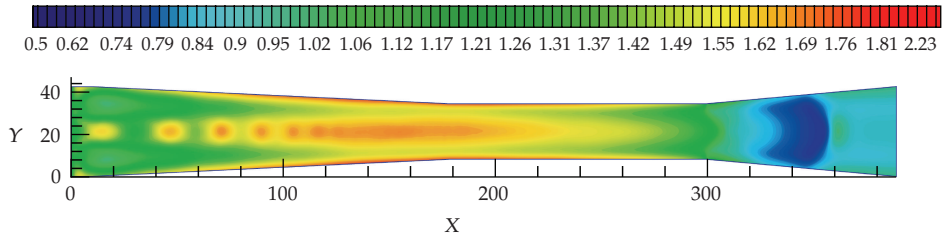

Figure 20: Density through the diffuser section at 1.65 s.

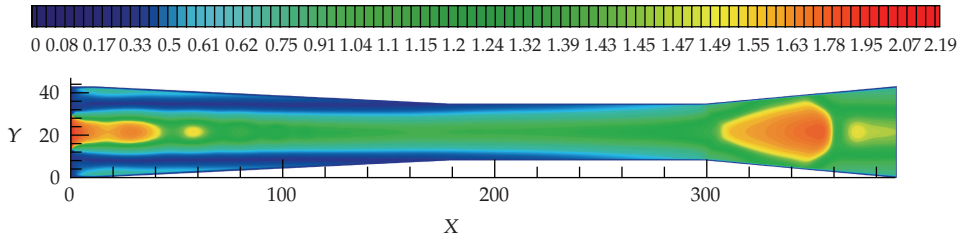

Figure 21: Mach number through the diffuser section at 1.65 s.

Table 1: Required memory.

	Used memory Physical/Virtual/Threads	Creation time of tasks
Native `jpvmDaemon .exe`	3060K/1352K/3	creation of 1 task: 2234.0 msecs creation of 2 tasks: 4105.0 msecs. creation of 4 tasks: 7972.0 msecs. creation of 8 tasks: 16183.0 msecs. creation of 16 tasks: 33998.0 msecs.
Using the JVM `java jpvmDaemon`	6020K/8920K/9	creation of 1 task: 2233.0 msecs. creation of 2 tasks: 3986.0 msecs. creation of 4 tasks: 7992.0 msecs. creation of 8 tasks: 16363.0 msecs. creation of 16 tasks: 34079.0 msecs.

Table 2: Communication time.

	`Communication time` 4 bytes	`Communication time` 1024 bytes	`Communication time` 10240 bytes	`Communication time` 102400 bytes	`Communication time` 1048576 bytes
Native `jpvmDaemon .exe`	pack: 0.625 msecs. comm: 36.3125 msecs. unpk: 0.0 msecs.	pack: 0.0390625 msecs. comm: 25.39453125 msecs. unpk: 0.625 msecs.	pack: 0.00244140625 msecs. comm: 32.274658203125 msecs. unpk: 0.0390625 msecs.	pack: 8.750152587890625 msecs. comm: 154.7671661376953 msecs. unpk: 1.25244140625 msecs.	pack: 41.859384536743164 msecs. comm: 1255.672947883606 msecs. unpk: 21.453277587890625 msecs.
Using the JVM `java jpvmDaemon`	pack: 0.0 msecs. comm: 33.1875 msecs. unpk: 0.625 msecs.	pack: 0.0 msecs. comm: 25.88671875 msecs. unpk: 0.0390625 msecs.	pack: 0.0 msecs. comm: 30.992919921875 msecs. unpk: 0.00244140625 msecs.	pack: 6.875 msecs. comm: 155.9370574951172 msecs. unpk: 3.750152587890625 msecs.	pack: 44.1796875 msecs. comm: 1281.7460660934448 msecs. unpk: 21.484384536743164 msecs.

Table 3: Performance Metrics using JPVM over the cluster.

Cores	Time	Speed-up	Max speed-up	Efficiency	Cost
1	4752 minutes	1	1	100%	1
2	2540 minutes	1.87	1.9	98%	1.02
3	1724 minutes	2.76	2.8	98%	1.01
4	1300 minutes	3.65	3.7	99%	1.01
5	1050 minutes	4.52	4.6	98%	1.02
6	950 minutes	5.4	5.5	98%	1.02
7	760 minutes	6.25	6.4	97%	1.02
8	670 minutes	7.1	7.3	97%	1.03
9	600 minutes	7.92	8.2	97%	1.04
10	540 minutes	8.8	9.1	97%	1.03
11	490 minutes	9.7	10	97%	1.03
12	450 minutes	10.56	10.9	97%	1.03
13	415 minutes	11.45	11.8	97%	1.03
14	385 minutes	12.34	12.7	97%	1.03
15	360 minutes	13.2	13.6	97%	1.03
16	350 minutes	13.58	14.5	94%	1.07

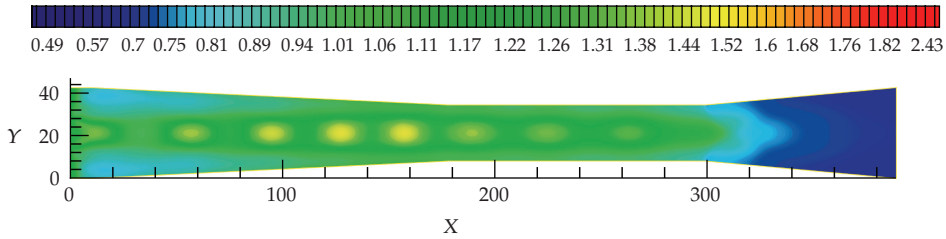

Figure 22: Density through the diffuser section at 5.0 s.

In Table 2 we can observe that there is not significative difference between the native version and the byte-code version in the communication time of the tasks. Nevertheless the advantage of the native version is the saving memory storage. About the execution time, gcj can produce programs that are faster than the byte code version around 30%.

The cluster configuration used to execute the simulation is

(i) 14 nodes, every node with two-processor Xeon DP 2.8 Ghz, 4 GB RAM, in total 28 cores.

For this application is supposed 10% of the serial part, the part that could not be parallelized (when the primitive variables are decoded); so the max *speed-up* for 16 tasks is 14.5; Table 3 shows the performance metrics for 16 tasks.

9. Conclusions

A numerical parallel code has been developed to simulate the supersonic flow in the ejector diffuser in a parallel-distributed system with an object-oriented methodology. The model was validated with cases where there is an exact solution. The design of a parallel program allows to reduce the execution time; nevertheless the design and build of a parallel program is

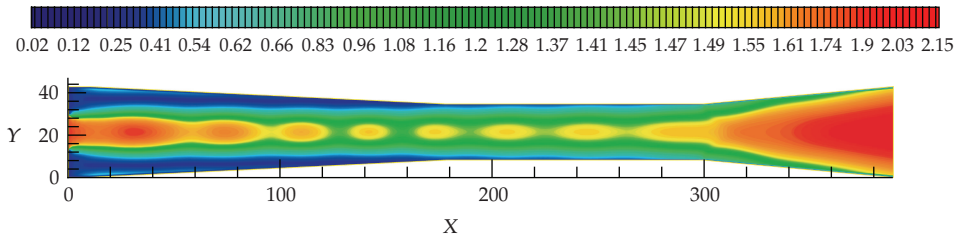

Figure 23: Mach number through the diffuser section at 5.0 s.

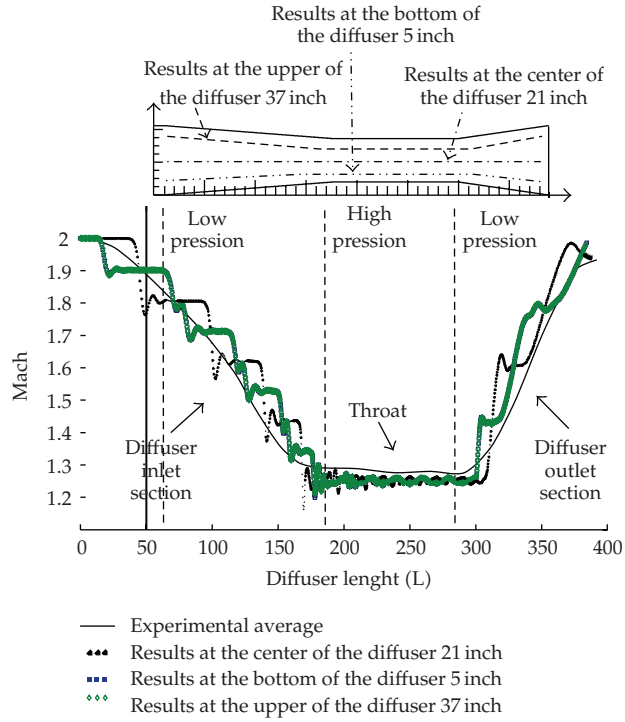

Figure 24: Mach profile of fluid through the diffuser section when the flux is stable.

complex due to the no deterministic execution and in a scientific computing ambient is more difficult build and debug because it is necessary to manage several data, and so it is necessary to use a good methodology to minimize the risk of bugs in the program construction.

We have discussed implementations of object-oriented design using Java in computational fluid dynamics simulations. This also provides the benefits of better maintainability, reusability, and extensibility of the code. For examples, is possible to create recursively new parallel tasks to control easily the granularity; for this reason Java is a serious language suitable for demanding applications in science and engineering and the JPVM is a good MPI-based tool for parallel cluster computing.

Finally, this program is a good tool to investigate the behavior of the flux in the ejector diffuser and shows how to transform the solution space in an easy way.

Nomenclature

u: Component velocity in the x direction (m/s)
v: Component velocity in the y direction (m/s)
T: Temperature (K)
P: Pressure (N/m^3)
ρ: Density (Kg/m^3)
e: Energy
M: Mach number
c_p: Specific heat capacity at constant pressure ((J/(kg K)))
γ: Ratio of heat capacities
Pr: Prandtl number
q_i: Heat flux along x_i direction (W/m^2)
R: Ideal gas constant ((J/(kg K)))
S: Sutherland's constant (K)
k: Thermal diffusivity (W/(m K))
v: Kinematic viscosity (m^2/s).

Acknowledgment

This research was supported by the Mexican Petroleum Institute by Grant 204005.

References

[1] H. E. Hage, "Steam ejector fundamentals: an alternative to vacuum pumps," *Chemical Processing*, vol. 61, no. 7, pp. 70–71, 1998.
[2] G. R. Martin, "Understand real-world problems of vacuum ejector performance," *Hydrocarbon Processing*, vol. 76, no. 11, pp. 63–75, 1997.
[3] K. Pianthong, W. Seehanam, M. Behnia, T. Sriveerakul, and S. Aphornratana, "Investigation and improvement of ejector refrigeration system using computational fluid dynamics technique," *Energy Conversion and Management*, vol. 48, no. 9, pp. 2556–2564, 2007.
[4] A. Hemidi, F. Henry, S. Leclaire, J.-M. Seynhaeve, and Y. Bartosiewicz, "CFD analysis of a supersonic air ejector—part I: experimental validation of single-phase and two-phase operation," *Applied Thermal Engineering*, vol. 29, no. 8-9, pp. 1523–1531, 2009.
[5] J. D. Anderson Jr., *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, New York, NY, USA, 1995.
[6] A. J. Ferrari, "JPVM: network parallel computing in Java," Tech. Rep. CS-97-29, Department of Computer Science, University of Virginia, Charlottesville, Va, USA, 1997.
[7] C. Estrada and H. César, "A virtual distributed JAVA machine for heterogeneous plataforms," Tech. Rep., CIC-IPN, Mexico City, Mexico, 1999.