HIGH PERFORMANCE COMPUTING FOR NUMERICAL FLUID FLOW SIMULATION

Cristiane Cozin, criscozin@yahoo.com.br
Rigoberto Morales, rmorales@utfpr.edu.br
PPGEM – Federal University of Technology-Paraná - Av. Sete de Setembro 3165, CEP. 80230-901, Curitiba-PR-Brasil.

Ricardo Lüders, luders@utfpr.edu.br
CPGEI – Federal University of Technology-Paraná - Av. Sete de Setembro 3165, CEP. 80230-901, Curitiba-PR-Brasil.

Abstract. *In recent years, computer cluster has emerged as a real alternative to solution of problems which require high performance computing. Consequently, the development of new applications has been driven. Among them, flow simulation represents a real computational burden specially for large systems. This work presents a study of using parallel computing for numerical fluid flow simulation in pipelines. A mathematical flow model is numerically solved. In general, this procedure leads to a tridiagonal system of equations suitable to be solved by a parallel algorithm. In this work, this is accomplisehd by a parallel odd-oven reduction method found in the literature which is implemented on Fortran programming language. A computational platform composed by four desktop computers with a total of twelve processors was used. Many measures of CPU times for different tridiagonal system sizes and number of processors were obtained, highlighting the communication time between processors as an imporant aspect to be considered when evaluating the performance of parallel applications.*

Keywords*: computational flow simulation, tridiagonal system equations, high performance computing*

1. INTRODUCTION

In computational mechanics, physical phenomena are mostly modeled by differential equations that do not have an analytical solution. In this case, a discretization is necessary to numerically solve them and results are compared with analytical solutions when it exists or with experimental data.

Depending on the application, numerical simulation requires a high computational effort. Although supercomputers can be considered in this case, the associated cost is generally prohibitive. However, High Performance Computing (HPC) has widely been used not only to designate supercomputers, but to characterize an intensive use of computational resources to solve problems that could take a prohibitive time to be solved in a single CPU. This term includes not only computers, but also networking technology, algorithms and the environment necessary to build such a system which can be represented by clusters of personal computers (Carvalho Jr., 2007).

Processing architecture in parallel computing is for the resolution of problems calling large computational effort and to submit a resolution to the appropriate structure in parallel. This structure provides the subdivision of the problem on tasks that could be implemented in parallel. Unfortunately, not all problems are parallel, and its resolution inherently sequential.

Parallel algorithms has been implemented for the solution of the conservative equations (Dou e Phan-Thien, 1997; Byrde et al., 1999; Passoni, 2001) using domain decomposition. The authors examined communication influence between the processors for each form of domain decomposition. The conclusion was obtained by the authors that the efficiency of the algorithm is heavily dependent on refining and partitioning the mesh, and the number of processors.

In this paper, we developed a parallel program for the solution of the conservative equations with the finite differences method. The program is the optimized solution tridiagonal system of linear equations from discretization of the equations of conservation. To execute the program was used a cluster of computers administered by Windows® Compute Cluster Server® 2003. We compute some examples to test the parallel algorithm. The communications effects on the performance of the algorithm are discussed.

This paper is organized as follows. Section 2 presents some mathematical background for modeling a laminar incompressible fluid flow. Emphasis is given to the algebraic system structure obtained after discretization. Section 3 discuss the parallel algorithm used to solve a tridiagonal linear system which results are presented in Section 4. Finally, Section 5 presents some concluding remarks.

2. MATHEMATICAL AND NUMERICAL MODELING

In this section, a general modeling for a laminar incompressible fluid flow is presented. Such model requires a numerical method to be solved since analytical solutions exist only for particular cases. Numerical methods are based on discretization of equations which leads to algebraic systems. In general, a first-order discretization generates a tridiagonal linear system of equations to be solved. In this work, we are interested on the solution of such systems by parallel algorithms. This procedure can be summarized as below.

The motion of a laminar incompressible fluid can be described by Eq. (1) for mass conservation and linear momentum.

$$\frac{\partial(\rho\phi)}{\partial t}+\frac{\partial(\rho u_j\phi)}{\partial x_j}=\frac{\partial}{\partial x_j}\left(\Gamma\frac{\partial\phi}{\partial x_j}\right)+S_\phi \tag{1}$$

with dependent variables vector ϕ, Γ denotes diffusivity of ϕ and source term S_ϕ.

Discretization of Eq. (1) over a control volume is done by using the hybrid difference scheme. The final discrete equation can be expressed as

$$a_P\phi_P=\sum\left(a_{NB}\phi_{NB}\right)+b \tag{2}$$

where

$$a_P=\sum\left(a_{NB}\right)+a_P^0-S_P\Delta V \tag{3}$$

$$b=S_C\Delta V+a_P^0\phi_P^0 \tag{4}$$

The subscript *P* in Eq. (3) and (4) denotes values for the current point. The sum reflects the idea that defines the structure of the matrix, depending on the order of discretization. If the discretization is the first order get a tridiagonal matrix, the second order get a pentadiagonal matrix and so on. The superscript *0* denotes values for past time step and *b* includes all terms explicitly calculated (including source terms). In addition, the source term is arbitrarily split into a constant and a linear terms in ϕ. Further information can be found in Patankar (1980).

Particularly, we consider the system of equations given by Eq. (1) as tridiagonal systeml. It can be solved by TDMA (Tridiagonal Matrix Algorithm - Versteeg and Malalasekera, 1995). This kind of linear system arises on many other problems when partial differential equations are numerically solved by a first-order discretization.

3. TRIDIAGONAL SYSTEMS SOLUTION

Several methods are found in the literature for solving tridiagonal linear system of equations. Among them, the Odd-Even Reduction method is chosen here due to its simple implementation (both for sequential and parallel versions) and previous success works (Stone, 1973) and (Saukas and Song, 1997). Stone (1973) has proposed a methodology using a doubling recursive technique suitable to implement on vector computers with SIMD architecture (Single Instruction Multiple Data). Saukas and Song (1997) has proposed a more simple method using substitution of variables which is suitable to implement on clusters as presented below.

3.1. Sequential Odd-Even Reduction

Consider a linear system $Ax=b$ where [*A*] is a tridiagonal matrix of order *N*. It means that all its elements in *A* are null, except those of diagonal and immediately above and below it. The numerical example given by Eq. (5) is used to show how the sequential Odd-Even Reduction algorithm works.

$$\begin{pmatrix}2&-1&0&0&0&0\\-1&2&-1&0&0&0\\0&-1&2&-1&0&0\\0&0&-1&2&-1&0\\0&0&0&-1&2&-1\\0&0&0&0&-1&2\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{pmatrix}=\begin{pmatrix}1\\0\\0\\0\\0\\1\end{pmatrix} \tag{5}$$

By replacing x_1, x_3 and x_5 as function of x_1, x_1 the reduced system of Eq. (6) is obtained.

$$\begin{pmatrix} 2 & -1 & 0 \\ 0 & 3 & -2 \\ 0 & -1 & 3 \end{pmatrix}\begin{pmatrix} x_2 \\ x_4 \\ x_6 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \tag{6}$$

Similarly, by replacing x_2 and x_6 as function of x_1 results $x_4 = 1$. Backing to Eq. (6) $x_2 = 1$ and $x_6 = 1$ are obtained and finally $x_1 = 1$, $x_3 = 1$ and $x_5 = 1$ from Eq. (5). In general, for each step of the Odd-Even Reduction algorithm the original system is reduced by alternating a substitution of odd and even variables. This algorithm presents good numerical properties for a diagonally dominant matrix (Saukas and Song, 1997), i.e. for all i such that $\left|A_{ii}\right| > \sum_{j \neq i} \left|A_{ij}\right|$.

Table 1 – Numerical Example for Parallel Odd-Even Reduction

Step	**Main Processor**	
1	The original system is split into two systems for processors 1 and 2	
Step	**Processor 1**	**Processor 2**
2	$\begin{cases} 2x_1 - x_2 = 1 \\ -x_1 + 2x_2 - x_3 = 0 \\ -x_2 + 2x_3 - x_4 = 0 \end{cases}$	$\begin{cases} -x_3 + 2x_4 - x_5 = 0 \\ -x_4 + 2x_5 - x_6 = 0 \\ -x_5 + 2x_6 = 1 \end{cases}$
	Variable x_2 is isolated: $x_2 = \frac{1}{2}\left(x_1 + x_3\right)$	Variable x_5 is isolated: $x_5 = \frac{1}{2}\left(x_4 + x_6\right)$
	By replacing x_2 into the system gives: $\begin{cases} 3x_1 - x_3 = 2 \\ -x_1 + 3x_3 - 2x_4 = 0 \end{cases}$	By replacing x_5 into the system gives: $\begin{cases} -2x_3 + 3x_4 - x_6 = 0 \\ -x_4 + 3x_6 = 2 \end{cases}$
	The above two-dimensional system is send back to the main processor.	The above two-dimensional system is send back to the main processor.
Step	**Main Processor**	
3	It receives the reduced systems generated by processors 1 and 2 and it composes a new system: $\begin{cases} 3x_1 - x_3 = 2 \\ -x_1 + 3x_3 - 2x_4 = 0 \\ -2x_3 + 3x_4 - x_6 = 0 \\ -x_4 + 3x_6 = 2 \end{cases}$	
	The above system can then be solved for x_1, x_3, x_4 and x_6 which are send back to processors 1 and 2, respectively.	
	Main Processor sends to Processor 1 the results for x_1 and x_3 and, Processor 2 the results for x_4 and x_6.	
Step	**Processor 1**	**Processor 2**
4	Receive solution: x_1=1 and x_3 = 1	Receive solution: x_4=1 e x_6 = 1
	Replacing x_1=1 and x_3 = 1 in $x_2 = \frac{1}{2}\left(x_1 + x_3\right)$, obtained the result x_2 = 1.	Replacing x_4=1 and x_6 = 1 in $x_5 = \frac{1}{2}\left(x_4 + x_6\right)$, obtained the result x_5 = 1.
	The result is send back to the main processor.	The result is send back to the main processor.
Step	**Main Processor**	
5	Composes the complete system solution.	

3.2. Parallel Odd-Even Reduction

By considering the same system given by Eq. (5), this section presents a parallel implementation of the Odd-Even Reduction Method using a cluster of computers with three processors. This is summarized in Tab. 1, where five steps are assigned to execute on processors 1, 2 and on the main processor.

According to Tab.1, step 1 is necessary to split by two the original system. This corresponds to divide matrix A into the first three lines for processor 1 and the last ones to processor 2. Each processor reduces in step 2 its corresponding system by replacing variables until only two variables remain. Then, each one sends back the reduced system to the main processor. The main processor receives in step 3 the system from each processor and solve it. Note that the main processor always solves a 2p-linear system of equations (where p is the number of processors). Finally, processors 1 and 2 complete in step 4 their solutions by substituting values received from the main processor which composes the complete system solution in step 5.

4. RESULTS

According to Section 1, HPC applications are used to provide a computational time reduction compared to a single processor. However, the results obtained show that significant differences are observed indicating that communication time between processors cannot be underestimated.

The results are obtained by implementing in Fortran the Odd-Even parallel reduction in a cluster given by Tab. 2. The numerical tridiagonal linear system is similar to the one given by Eq. (5), but varying its dimension n. Again, the system solution is $x_i=1$ for all $i=1,\ldots,n$.

Table 2 - Configuration of desktops that make up the cluster

Computer	**Settings**
NODE 1 (main)	Intel Core 2. 2.4 Gb, 1 Gb de RAM
NODE 2	Intel Core 2. 2.4 Gb, 1 Gb de RAM
NODE 3	Intel Xenon Quad Core, 1.6 Gb, 4 Gb de RAM
NODE 4	Intel Core 2 Quad, 2.4 Gb, 2 Gb de RAM

Execution time measures were obtained through CPU_TIME function available on Intel$^{®}$ Visual Fortran. This function was inserted on suitable points of Fortran program to allow independent measures of calculation and communication times. Moreover, time measures corresponds to mean values obtained for ten replications of each case.

Figure 2 shows total CPU time for different system dimensions and number of processors. Note that CPU time increases with the number of processors. Moreover, a large increase is observed for big values of n.

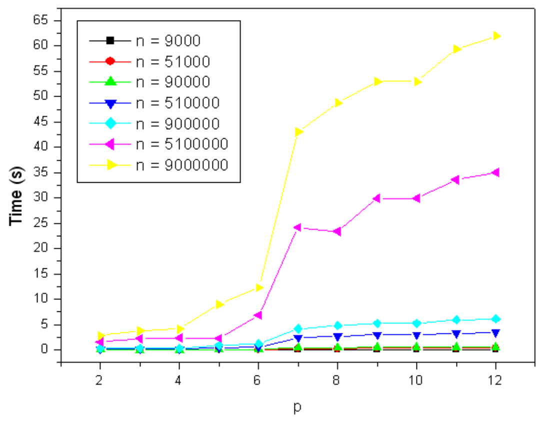

Figure 2 – Total CPU time for different system dimensions and number of processors.

This is contradictory to the HPC goal of CPU time reduction. However by dividing time measures between calculation and communication times, other information can be obtained as shown in Fig. 3 e Fig. 4.

Figure 3 shows the corresponding calculation time, i. e. CPU time due to computation of the odd-even algorithm. In contrast, Fig. 4 shows the corresponding communication time between processors, i. e. time spent for waiting data exchange to be finished. Note that Fig. 3 presents an expected behavior for a parallel implementation since CPU time decreases as the number of processors increases (for all values of n). This behavior is also reported by Saukas and Song (1997) using a vector computer which has tight coupled processors.

The timing of communication is a factor of extreme importance which should be given special attention. "Time of communication" refers to all computational processes that are not routines of calculation, thus, the time of communication also includes other factors, the time for the allocation of memory. In these tests used a hub for transmitting data at speeds of 100 MBits. The timing of communication is highly influenced by the topology of network used.

Figure 4 presents a graph is the time for communication with the number of processors used for various values of n. In this case, it appears that the time of communication increases as increasing the number of processors.

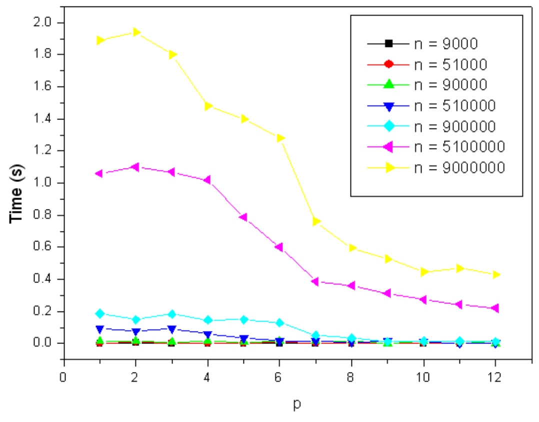

Figure 3 - Calculation time

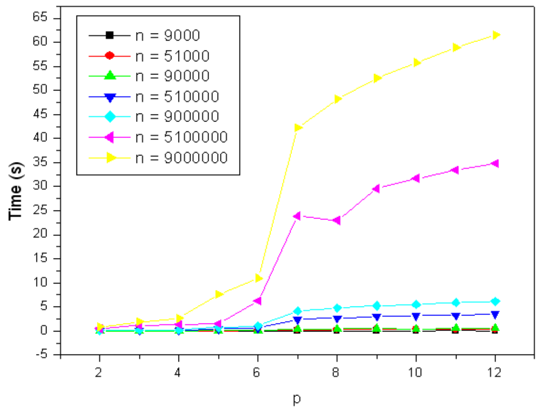

Figure 4 - Communication time between processors

It can be seen from Fig. 3 and Fig. 4 that the communication time has the same order of calculation time for p<6 and n<900000, but for big values of n the communication time is about 30 times of the calculation time. This means that almost all time is spent for data communication between processors and no useful computation is made. This clearly shows that applications having a fast distributed calculation time could be depreciated in computer architectures with low coupled processors like clusters.

5. CONCLUSIONS

Fluid flow behavior can be simulated by numerically solving a mathematical model of partial differential equations. In general, such procedure requires a high computational effort and consequently large execution times. High performance computing (HPC) is therefore an attractive alternative. However, a carefully analysis should be made to evaluate suitable applications. This paper presents a study of HPC application for fluid flow simulation, where the numerical solution of a tridiagonal system of equations is considered. A computer cluster was used to compute a parallel tridiagonal system solution implemented on Fortran.

Results show for this application that total CPU time increases with the number of processors. This means that a superficial analysis could conclude that it is better to compute the application in a single processor. However, a more detailed analysis shows that calculation time and communication time between processors should be properly evaluated. Although calculation time decreases with the number of processors as expected, communication time significantly increases for large values of system dimension. Communication time between processors has thus a significant impact on the overall performance when using clusters. This behavior is quite different from observed for tight coupled architectures such as vector computers. Therefore, more promising computer cluster applications have to present a minor data exchange effort compared to the calculation effort. Other applications can be suitable for multicore computers having no distributed memory. This will be subject of future research.

6. ACKNOWLEDGEMENTS

The authors acknowledge the financial support from the National Agency for Petroleum, Natural Gas and Biofuels (ANP) through its Human Resources Program in UTFPR (PRH-10), CAPES and from TE/CENPES/PETROBRAS.

7. REFERENCES

Byrde, O., Couzy, W., Deville, M. O., Sawley, M. L, 1999, “High-performance parallel computing for incompressible flow simulations, Computational Mechanics”, Vol 23, pp. 98-107, Springer-Verlag.

Carvalho Junior, F. H., 2007, “Computação de Alto Desempenho em Plataforma Windows”, The Brazilian Symposium on Programming Languages (SBLP).

Dou, H.-S., Phan-Thien, N, 1997, “A domain decomposition implementation of the SIMPLE method with PVM”. Computational Mechanics, Vol 20, pp. 347-358, Springer-Verlag.

Patankar, S. V., 1980, “Numerical Heat Transfer and Fluid Flow”. McGraw-Hill, New York.

Saukas, E. L. G., Song, S. W., 1997, “A Parallel Algorithm for Solving Tridiagonal Linear Systems on Coarse Grained Multicomputers”. São Paulo: USP, IX Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho.

Stone, H.S., 1973, “An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations”. Journal of the Association, Vol 20, N° 1, pp 27-38.

Versteeg, H. K.; Malalasekera, W., 1995, “An introduction to computational fluid dynamics: the finite volume method”. Essex, Engl.: Longman Scientific & Technical, 257 p.

8. RESPONSIBILITY NOTICE