Proceedings of the 10th Brazilian Congress of Thermal Sciences and Engineering -- ENCIT 2004
Braz. Soc. of Mechanical Sciences and Engineering -- ABCM, Rio de Janeiro, Brazil, Nov. 29 -- Dec. 03, 2004

Paper CIT04-0241

THREE-DIMENSIONAL HEAT TRANSFER SIMULATION AS A BENCHMARK FOR AN IN-HOUSE BEOWULF-CLASS CLUSTER

Marinho, Wellington P.
Universidade Federal de Uberlândia
Faculdade de Engenharia Mecânica
wellmarinho@yahoo.com.br

Campregher, Rubens
Universidade Federal de Uberlândia
Faculdade de Engenharia Mecânica
campregher@mecanica.ufu.br

Silveira Neto, Aristeu
Universidade Federal de Uberlândia
Faculdade de Engenharia Mecânica
aristeus@mecanica.ufu.br

Abstract. *The aim of this work is to present the resources involved in the implementation of an in-house Beowulf class cluster to solve the heat transfer problem in a solid. It is well-known that the simulation of huge and complex numerical problems needs very large computational resources at very high costs, especially for universities and research institutions in developing countries. So, the in-house cluster becomes a viable and reliable solution.* ***This paper discusses also the parallel methodology as domain decomposition technique, parallel results analysis and discussion.***

Keywords*. Beowulf cluster, parallel program, heat transfer, speedup, parallel efficiency.*

1. Introduction

One of the greatest problems involving complex physical problems simulations is the computational resources available today and considering that, in most of the time, latest generation hardware are not commonly at hand. So, an in-house Beowulf cluster turns out to be very suitable, mainly due to its comparatively low cost when comparing to the supercomputers and due to its scalability, that is, the possibility of increasing the cluster resources by adding more processors. The cheaper assembly is ensured by employing on-the-shelf hardware and open source software, which can be downloaded from the internet at no cost.

The phenomena simulated in this work are related to the pure heat transfer conduction through a solid, where the computational domain was split into several processors. The knowledge background gained here would help to develop a parallel convection-diffusion equation solver. A cluster of PCs follows a parallel architecture, that is, the data processed inside each machine be exchanged through the network nodes. This enables huge complex tasks been split in smaller ones and distributed among the several processors to be simultaneously executed.

2. Beowulf clusters

The architecture models based on clusters have their more popular version the so called Beowulf class cluster. As a definition, one can say that the Beowulf is a multiprocessor architecture employed in parallel computation, composed by a group of machines distributed in a master node and clients nodes connected by a network (Ethernet or another topology), running a operational system that supports whether a Parallel Virtual Machines (PVM) structures and/or Massage Passing Interface (MPI). The server controls the entire cluster, distributes the data and the tasks to client's nodes as well as it manages the output to the external link, e.g., an internet access. Its possible, also, to have more than one server in a Beowulf cluster, which could be dedicated to more specific duties as monitoring the local network, communicating to the external link or even as a console. Each Beowulf cluster node can be as simpler as possible. It is recommended to employ machines having the less number of peripherals, i.e., audio or video interfaces, external disc units, etc. Every client node is configured from the master node, executing only the tasks that it has demanded.

The *Beowulf* name was borrowed from one of the most ancient Nordic epic poems, which describe the journey of a great and brave hero to kill the monster of Grendel. The first Beowulf cluster was developed in 1994 by Thomas Sterling and Don Becker (Pitanga, 2002), at the Centre of Excellence in Space Data and Scientific Information (CESDIS) of the NASA's Goddard Space Centre in Greenbelt, MA. It was implemented in a joined project of the Earth and Space Sciences (ESS) and the High Performance Computing and Communication (HPCC), where the researchers were located.

This first Beowulf cluster was formed by a 16 processors connected via 10MBps Ethernet link. By that time, due to the net low transfer rate and switch high costs, the researchers needed to develop a specific driver to handle the dual Ethernet boards, creating the channel bounded Ethernet, which each node could access directly other nodes. The operational system used was the Linux that, due to the GNU public license, already supported the MPI and PVM libraries. The hardware specification were common – each node was formed by a Intel™ 486DX4 100MHz processor, 16Mb/60ns RAM, 540MB IDE hard drive and a 10Mbps dual network card, every one of them easily acquired in ordinary market at low cost. All of these characteristics have contributed to the Beowulf class parallel architecture granted success and popularity and, very soon, had been implemented in several universities and research centre. In 1996, at Supercomputing fair, was demonstrated that an affordable US$ 50.000,00 Beowulf cluster could reach 1Gigaflop/s. Nowadays, there clusters of more than 1000 processors and reaching performance peaks of more than 1 Teraflop/s (10^{12} floating points operations per second).

2.1 Beowulf clusters features

A Beowulf cluster is not a software package, nor a special hardware, or even a special operational system kernel (Gottlieb, 2001). This configuration is more a way of connect the Linux (or another OS) workstations, to simulate the behavior and the performance of a parallel supercomputer. Although there are lots of software packages, OS kernel versions, and PVM / MPI libraries that make cluster architecture easier to set up or even run faster, is possible to assemble a Beowulf cluster using any of the common Linux distributions. Roughly speaking, any combination of two or more machines connected by a network, running Linux, sharing at least one common NFS directory, having remote access capability, and parallelization libraries, could be considered a Beowulf cluster.

In the parallel computing traditional taxonomy, a Beowulf cluster can be considered as a half-way between the massively parallel processing (MPP) machines – such as Cray or Hypercubes – and the Network of Workstations (NOW), with the characteristics and the benefits of both worlds. The MPPs are typical larger and more expensive machines, but having its internal data transfer connections faster than the clusters. However, its programming is more complex, requiring careful load balancing, granularity (ratio between the data being computed and the data being exchanged) studies, and communication overhead to achieve its performance optimum. For NOWs programming, very tolerant algorithms concerning to the load balancing and communicating latency are needed. By the other hand, any MPP or NOW algorithm that doesn't require fine granularity can be ported to a Beowulf cluster without difficulties.

There are some few but very important characteristics that separate the Beowulf class cluster from a cluster of workstations (NOW). Firstly, the cluster nodes are dedicated exclusively to the cluster internal processing tasks. This makes the load balancing easier, in a way that each node performance doesn't change by external factors; they are isolated from them, so the slave nodes run tasks determined only by the master node. Secondly, in the cluster of workstations, the algorithm has to compete against several others task and processes not related to the problem itself, which diminishes its efficiency and may affect the network security. Every process in a cluster node is controlled by an ID number generated and maintained by the master node, avoiding running unidentified tasks. Finally, in a NOW the Operational System tries to be as friendly as possible to the user, which drains computer memory. By the other hand, the user doesn't need to have directly access to the master node, saving important machine resources.

2.2 Beowulf clusters components

2.2.1 Hardware

A Beowulf cluster is assembled from ordinary machines, easily acquired in the local commerce, constituting one of its stronger features. So, any group of machines connected by a local network is enough to build a Beowulf cluster. There are two different possibilities to make a connection among the machines: directly connected or using a switch. The main advantage of employing a switch is to avoid dual port network cards. However the cost of this device, although continuously decreasing, is still high.

In a very simple classification, one can divide de class of clusters in two groups: the *class 1* and the *class 2 clusters*. The *class 1* clusters are built with general devices from renowned companies. Their main advantages are easier maintenance and acquisition of drivers, the components have compatibility among them, and new hardware is easily found. The *class 2* clusters are assembled using specific devices or OEM versions from several manufacturers. For their turn, this kind of clusters have allows a more freely choose of hardware, especially in case to increase performance. By the other hand, may be a little difficult to find drivers and/or spare parts.

A Beowulf cluster can be assembled with homogenous or heterogeneous machines. One says that a cluster is homogenous when all of its nodes, sharing the same network, have the same memory and processing resources. A heterogeneous cluster has different machines connected among them or different kind of network linking the machines. The aftermath is that in a homogenous cluster, the load balancing is an easier task than in a heterogeneous one; where deeper analyses must be done for take into account the nodes differences.

This work was run in a homogenous cluster of five machines, where the master node configuration is composed by a Intel® 865PERL mother-board, a Pentium4® 2.8GHz processor, 1024 MB DDR RAM, 80 GB IDE hard drive and a Radeon® 9200 128MB DDR AGP8x video card. As a homogenous cluster, the slave nodes have the same configuration, except by the video card, which is a Geforce® 64MB AGP4x – the postprocessing tasks are done by the

master node. All the machines are connected by a Gigabit network via 3COM® 8port switch. The cluster connection sketch can be seen on Fig. (1).

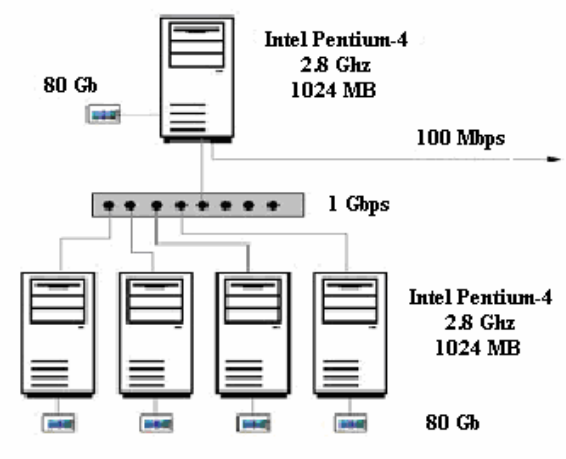

Figure 1. Cluster architecture

2.2.1 Software

The cluster operational system running is a Red Hat® Linux, chosen by its easy package management and upgrades availability (Sonzogni *et al.*, 2002). The network management has been analyzed by the bWatch® tool. The posprocessing tools were the Paraview® and Opendx® software.

In order to change the data among the nodes, the MPI (Message Passing Interface) has been chosen as a library. The MPI is a communication pattern developed by about 60 people in 40 organizations, most of them inside the United States. Many computer vendors are involved with the MPI project by research done in University laboratories as well as in industrial and governmental laboratories. The “philosophy” behind the MPI project is to promote frequent meetings to consider the inclusion of many new features that have been tested and implemented by the users around the world.

3. The three-dimensional parallel heat transfer problem.

A three dimensional transient heat conduction problem has been considered for illustrating the parallelization strategy and to analyze the performance. The energy transport equation, from where a temperature field T(x,y,z,t) can be evaluated, is:

$$\frac{\partial(\rho T)}{\partial t} = \frac{\partial}{\partial x}\left(\frac{k}{c_p}\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{k}{c_p}\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(\frac{k}{c_p}\frac{\partial T}{\partial z}\right) + S \tag{1}$$

where the ρ stands for solid density , κ is the thermal conductivity, c_p is the specific heat and S is the source term.

The continuous domain was discretised by the finite difference method (Fortuna, 2000). The numerical domain can be seen in the Fig. (2).

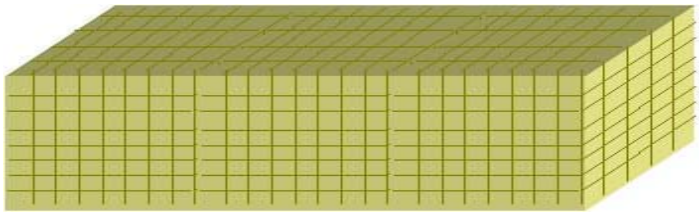

Figure 2. Numerical domain

The system of algebraic equations Eq. (2) , derived from the numerical discretization was solved by a parallel SOR, as described below (the superscript *k* represents the current iteration):

$$\phi_{i,j,k}^{k+1} = \frac{\omega\left(A_L\phi_{i+1,j,k}^{k} + A_W\phi_{i-1,j,k}^{k+1} + A_N\phi_{i,j+1,k}^{k} + A_S\phi_{i,j-1,k}^{k+1} + A_T\phi_{i,j,k+1}^{k} + A_B\phi_{i,j,k-1}^{k+1} + S_{i,j,k}\right)}{A_P} + (1-\omega)\phi_{i,j,k}^{k} \tag{2}$$

The boundary conditions for the 3D domain consists in Dirichlet type at laterals walls and Neumann type at the upper and lower walls, as can be seen in the Fig. (3). All the initial conditions were 0°C for interior domain and y axis faces. The remaining faces were initially set at 150°C. Along the simulation the x and z axis faces temperatures were kept at 150°C and the y axis faces were considered insulated. The results for a domain with dimensions Lx, Ly, Lz having 240 X 30 X 90 points, respectively, are depicted in the Fig. (7).

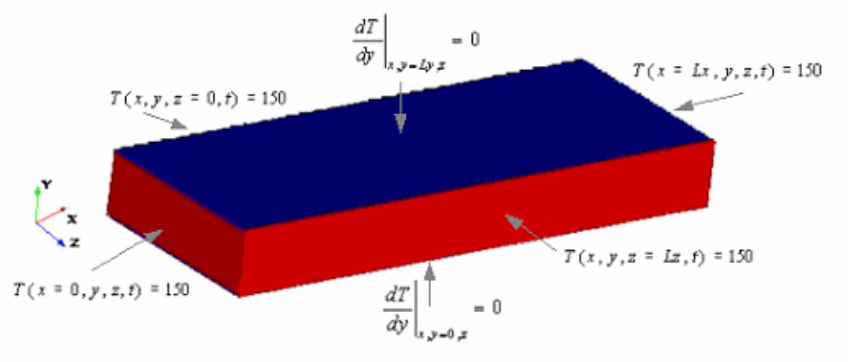

Figure 3. Numerical domain boundary conditions.

An important attention has to be focused on the domain decomposition, especially for the load balancing where domains of equal size are desirable. The decomposition produces a certain number of tasks, each of them having their amount of data and processes. The domain decomposition can be done in different ways, depending on the data structure (Pacheco, 1996). The Fig. (4) shows three different ways of decomposition it computational domain 3D.

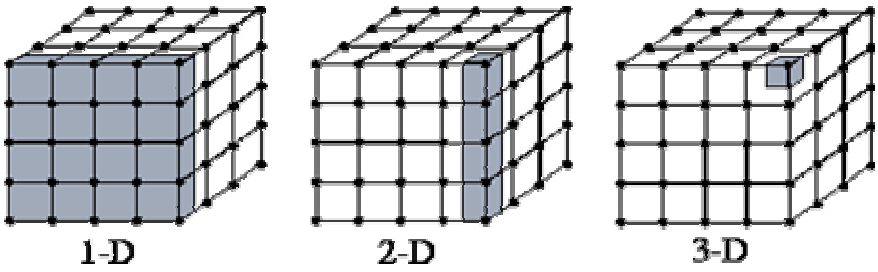

Figure (4). Different kind of domain decomposition for a three-dimensional domain, the data associated to a task is shaded.

Decomposition in one, two and three dimensions are possible for a three-dimensional mesh. In the present work the heat conduction problem is decomposed in one dimension. This method is easier to implement for communication processes. An example of domain decomposition is depicted in the Fig. (5).

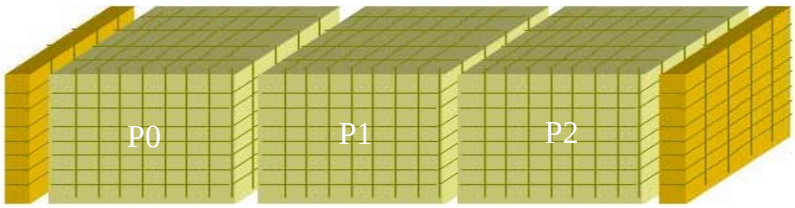

Figure 5. One dimensional decomposition of a three-dimensional domain.

The processes in the domains, although parallelized, are not independently among them for most of the numerical problems. The computation performed in each subdomain requires data that belong to the adjacent one. So, the data must then be exchanged between processors to update the subdomain boundary conditions.

The Fig. (6) depicts the message exchange scheme where only one plane of cells is overlapped, the minimum required in a second-order approximation. It is very important to note that the physical boundary conditions along the x axis (Dirichlet type) are applied only at the first and the last processors, i.e., at the x = 0 face in the P0 subdomain and at the x = Lx face in the PN subdomain. The boundary conditions required to solve the problem in the remaining subdomains x faces are provided by the data exchange describer above. Along the other axis, the physical boundary conditions are exactly the same for all subdomains and no data exchange is performed (due to the fact that is an 1-D domain decomposition).

The synchronization procedure is one of the most difficult tasks when implementing parallel algorithms. Transferring information in a synchronized way requires a strict combination between sending and receiving messages. In fact, a well-parallelized algorithm must furnish results exactly as a serial algorithm would yet within, of course, the solver precision.

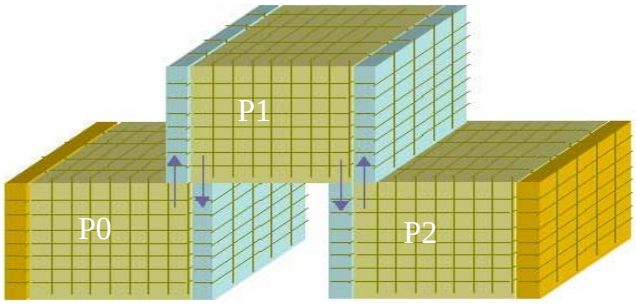

Figure 6. Overlapping and message exchange between subdomains.

4. Results e discussions

Firstly, the parallel results have been compared against the serial ones under the same initial and boundary conditions. The Fig (7) represents the temperature field results in y = Ly/2 plane, obtained for simulations from one (N = 1) processor to five (N = 5) processors. Moreover, below each temperature field, a temperature profile extracted from 40 points along the x axis is displayed, making possible to quantify the results.

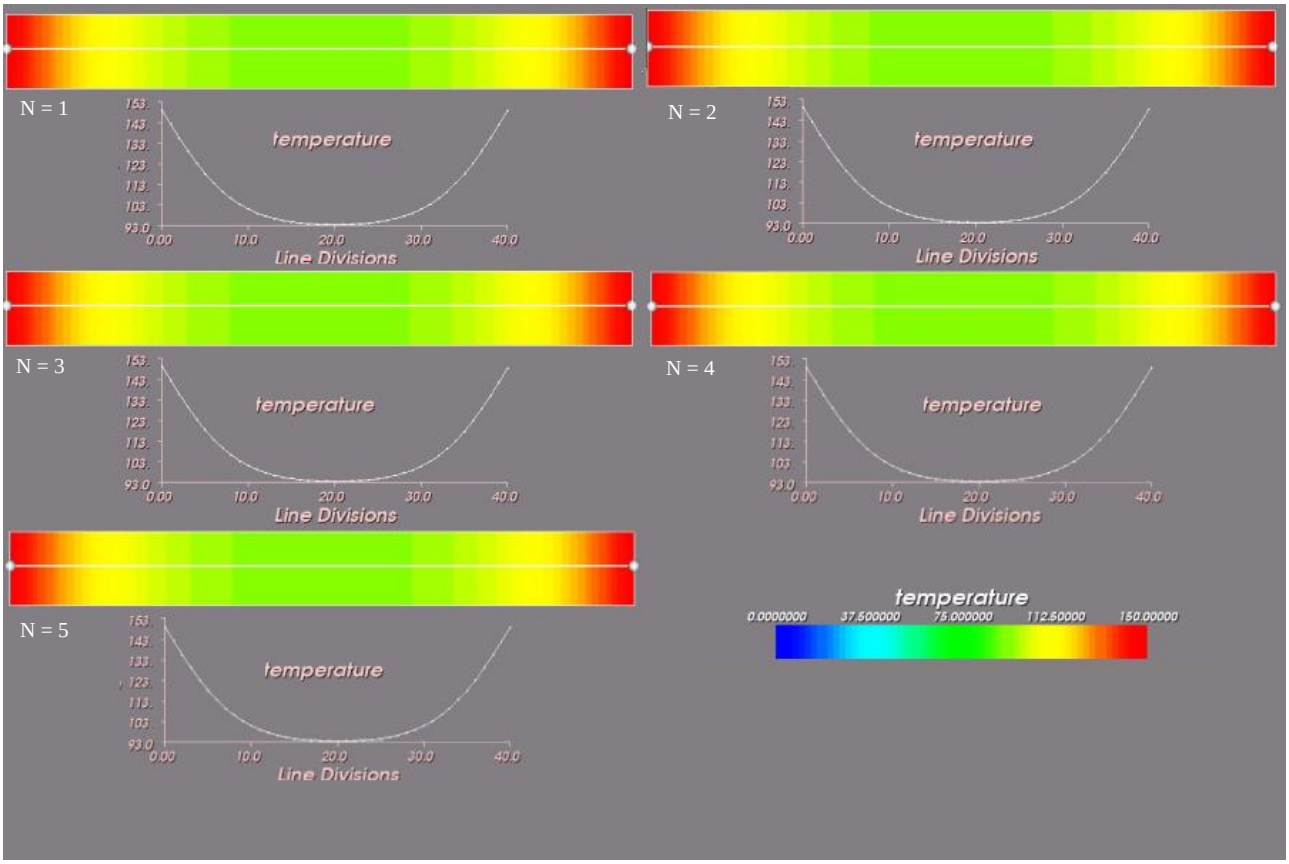

Figure 7. Temperature field in the xz plane for five different numbers (N) of processors.

It is very interesting to note that, in all simulations, the temperature field is the same up to the solver tolerance, i.e., 1.0×10^{-6}. This characteristic denotes the parallel algorithm consistency when comparing to the serial algorithm. The Fig. (8) represents the time evolution of the temperature field for the mesh size of 240 x 30 x 90, employing five processors. There are five different time positions, from t_0 = 2s to t_4 = 10s, with time steps of 2 seconds.

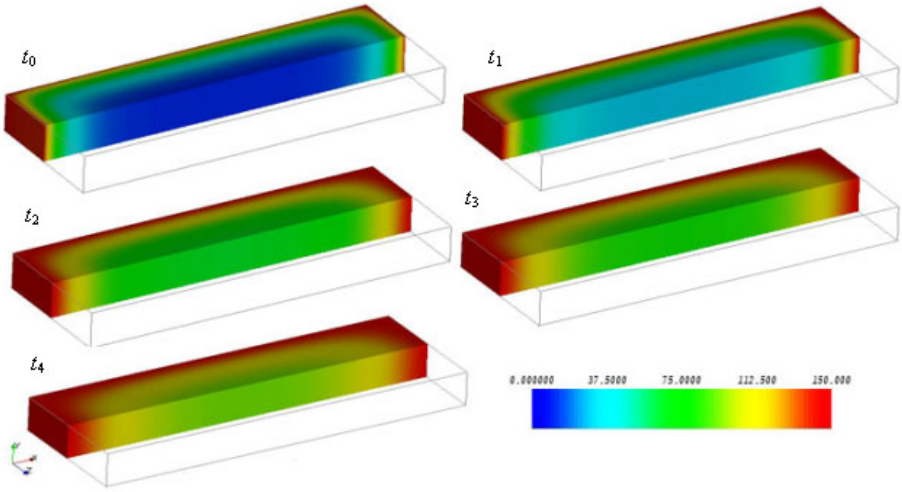

Figure 8. Temperature field time evolution, from t = 2 to t = 10 seconds.

In order to evaluate parallel algorithm performance, two concepts are commonly used (Baker &Smith, 1996), i.e., speedup and the efficiency. The first one is defined as:

$$S(N) = T(1)/T(N) \tag{3}$$

where T(1) is the timing achieved with only one processor, T(N) is the time required to solve the same problem in a parallel architecture having a number N of processors. The efficiency is defined by:

$$E(N) = S(N)/N \tag{4}$$

also represented in a percentage (%) basis. In other words, efficiency can be understood as the fraction of time that processors spend doing useful work.

The Tab. (1) represents the speedup and the efficiency for the results obtained running from a single processor to the maximum number of processors, i.e., using five machines. In the Fig. (9), the results for the speedups are plotted and compared against the ideal speedup, represented by the blue curve. The Fig. (10) depicts the efficiency obtained for the same mesh sizes as the Fig. (9). An optimal efficiency would be at E = 1.0 (or 100%).

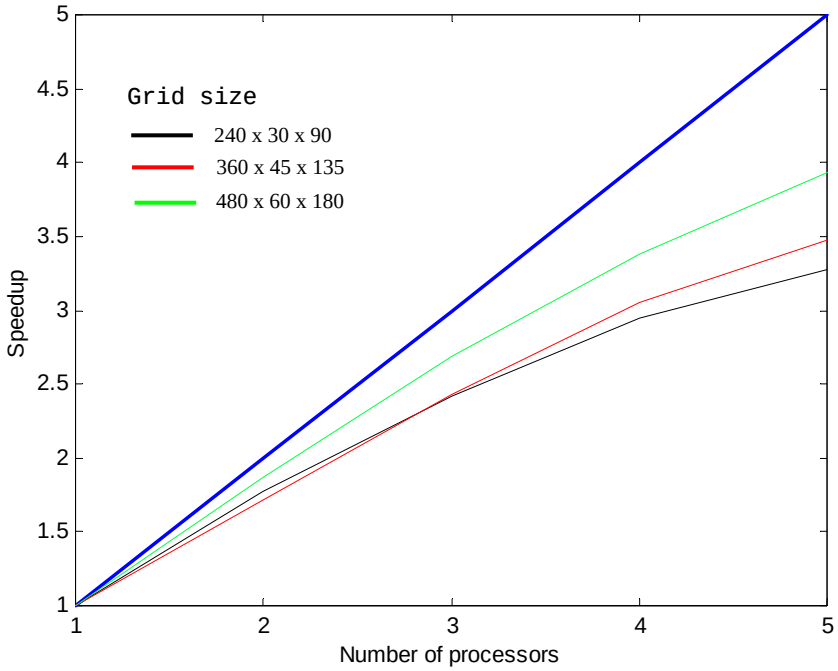

Figure 9. Speedup results for three different mesh sizes. The blue curve represents ideal speedup.

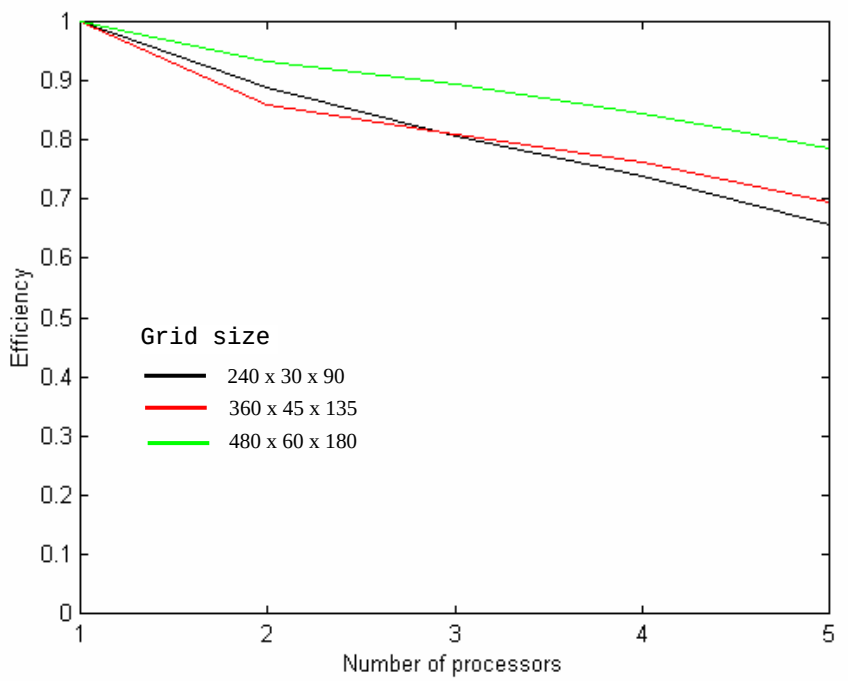

Figure 10. Efficiency results for three different mesh sizes.

Table 1 – Results for speedup and efficiency from single to five processors

Mesh size	N. of processors	Speedup	Efficiency (%)
240x30x90	1	1,00	100,0
	2	1,78	88,9
	3	2,42	80,5
	4	2,95	73,8
	5	3.28	65,6
360 x 45 x 135	1	1,00	100,0
	2	1,72	85,9
	3	2,43	80,9
	4	3,05	76,2
	5	3,48	69,6
480 x 60 x 180	1	1,00	100,0
	2	1,87	93,4
	3	2,69	89,5
	4	3,38	84,4
	5	3,93	78,6

It can be seen that with five processors, the speedup is increasing for all mesh sizes. This feature is very important to evaluate the algorithm speedup optimum, i.e., what is the processor number in which the speedup is highest. The results have shown that this number hasn't been reached yet, meaning the more processors could be added with no scalability reduction. Another observation has to be done for the mesh size. A higher speedup is reached as the mesh

becomes finer. One of the explanations for this behavior is that the time spent on the linear system solution is responsible for the most part of the total simulation time, when comparing to the time spent on data transferring between processors. For coarser meshes, the time spent on data exchange tends to overcome the solution time itself. In the Fig. (10) the idea of "useful work" that can be derived from the efficiency definition is well noted. There, one can see that the relation between computing time and communication time is more favorable for the finer mesh. Roughly speaking, for the 460 x 60 x 180 grid that has run in five processors, e.g., approximately 80% of the simulation time where spent on computations and the remaining time on communication tasks. A similar analysis can be done for the other mesh size results. Although these results are very interesting, it has to be note that they are specific to this hardware (network and machine) and software (algorithm, data exchange library, operational system, etc.) configuration.

5. Conclusion

The Beowulf cluster is a successful project. Its idealizer's option for ordinary hardware and open source software make them very easy to use and maintain. The increasing number of Beowulf clusters doesn't deny that so, it can be said that this kind of cluster is more than an experiment: one was obtained a very practical architecture that is still getting better. Moreover, the recent low cost processors, with characteristics only found on supercomputers, make this kind of clusters viable when applications doesn't depend on very high network speed.

The speedup curves behaves as found in the literature and the temperature field results keep unchanged with the number of processors, at least within the solver tolerance. When the application granularity is increased, the speedup curve becomes closer to the ideal one, as can be seen in the Fig. (9).

Finally, it can be said that, for a low cost equipment, was possible to simulate problems as large as 5 million cells. Just few years ago, this performance was impracticable even on very expensive supercomputers.

6. References

Baker, L. and Smith, B.J., 1996, "Parallel Programing", Computing McGraw-Hill.
Fortuna, A. O., 2000, "Técnicas Computacionais Para Dinâmica Dos Fluidos: Conceitos Básicos e Aplicações"
Gotlieb, S., 2001, "Cost-Effective Clustering", Computer Physics Communications, Vol.142, pp. 43-48.
Pitanga, M., 2002, "Construindo supercomputadores em Linux", Ed. Brasport.
Pacheco, P.,1996, "Parallel Programming with MPI", Morgan Kaufmann.
Sonzoni,V.E., Yommi, A.M., Nigro,N.M., Storti, M.A., 2002, "A parallel finite element program on a Beowulf cluster", Advances in Engineering Software, Vol. 33, pp. 427-433

7. Copyright Notice