Annales UMCS Informatica AI 2 (2004) 301-308

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Small clusters – MPI usage for solving MHD equations with the FLASH code

Małgorzata Selwa*, Krzysztof Murawski

Institute of Physics, Maria Curie-Skłodowska University,
Radziszewskiego 10, 20-031 Lublin, Poland

Abstract

Despite fast development of computer technology some numerical problems are still too complex to be solved using personal computers. In such cases the use of supercomputers can be a good choice. However, running jobs at computer centers involves necessity of queuing. Additionally, problems with fast transmission may arise if output files are large enough. An alternative solution is to build multiprocessor clusters of computers on which Message Passing Interface is implemented. In this paper application of MPI is presented for the case of the FLASH code which solves initial-value problems for magnetohydrodynamic equations.

1. Introduction

Construction of supercomputers is based on cooperation – the arrays of fast processors that work together to solve complex tasks in the field of, e.g. modeling of climate, controlled fusion, nanotechnology, medicine and biology, advanced engineering or astronomical processes. These supercomputers are usually too expensive for most research centers which have at their disposal small amount of budget. For the past few years scientists used to link relatively cheap personal computers and adapt their codes to be run on that kind of platforms.

The idea of joining computers was put forward for the first time by the U.S. Air Force to avoid Soviet nuclear attack in the 1950s and 1960s [1]. The first PC cluster was made in 1994 at the NASA Goddard Space Flight Center and was called Beowulf [1] to commemorate the of medieval hero who defeated a giant monster called Grendel. Since then this name has been adopted to characterize all clusters constructed from PCs.

The use of clusters is possible because of parallel computing strategy. A parallel processing system can divide complexity of a problem into smaller and

* Corresponding author: *e-mail address*: mselwa@kft.umcs.lublin.pl

simpler tasks which are assigned to nodes of a system that solve them simultaneously. The efficiency of that kind of work depends on the nature and complexity of a problem. An important consideration is frequency with which the nodes must communicate and exchange data to coordinate the work of the cluster. Problems which have to be divided into enormous number of simple tasks requiring communication will not be well suited for parallel processing.

Another problem which can occur while using clusters is inefficient usage of their speed. It occurs that the efficiency of the clusters can be reduced to a few percent for some kind of applications. A good solution of this problem is to build in processors which are constructed on the basis of hybrid technology multithreaded (HTMT) [2].

In this paper the application of a cluster will be presented to simulate wave processes in astrophysical plasma. This simulation is performed with the use of the FLASH code which is developed at the Accelerated Strategic Computing Initiative (ASCI) FLASH Center [3]. The SOLAR cluster that consists of two SMP (symmetric multiprocessing) nodes with two multithreaded processors Intel Xeon was tested with MPICH-1.2.5 version.

2. Numerical model

In order to check the properties of the cluster we solve the ideal MHD equations:

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \mathbf{V} \right) = 0, \\
&\rho \frac{\partial \mathbf{V}}{\partial t} + \rho \left(\mathbf{V} \cdot \nabla \right) \mathbf{V} = -\nabla p + \frac{1}{\mu} \left(\nabla \times \mathbf{B} \right) \times \mathbf{B}, \\
&\frac{\partial p}{\partial t} + \nabla \cdot \left(p \mathbf{V} \right) = -p \left(\gamma - 1 \right) \nabla \cdot \mathbf{V}, \\
&\frac{\partial \mathbf{B}}{\partial t} = \nabla \times \left(\mathbf{V} \times \mathbf{B} \right), \nabla \cdot \mathbf{B} = 0.
\end{aligned} \tag{1}$$

Here ρ denotes the mass density, $\boldsymbol{V}$ is the plasma velocity vector, p is the gas pressure, $\boldsymbol{B}$ denotes the magnetic field, $\gamma = 5/3$ is the adiabatic index and μ is magnetic permeability.

In our studies we consider two coronal loop models in cartesian geometry. Since our aim is modelling of a straight coronal loop, the complexity of adequate 1d or 2d models is similar to chosen models.

2.1. One-dimensional model

We consider first the problem of one-dimensional wave propagation in a coronal loop which is anchored in much denser photosphere. The equilibrium mass density profile is

$$\rho_0(x) = \frac{\rho_L}{2} \cdot \left[c_{tr} \cdot \tanh\left[s_t \cdot \left(x - x_{tr}\right) \cdot \left(x - L + x_{tr}\right) \right] + 3 \right], \tag{2}$$

where $\rho_L = 10^{-15}$ g/cm^3 denotes the mass density of the loop, $c_{tr} = 10^4$ is the mass density contrast between the loop and the photosphere, $s_t = 0.5 \cdot 10^{-17}$ corresponds to the steepness of the density profile, $L = 50 \cdot 10^8$ cm is the loop length and $x_{tr} = 0.2842 \cdot 10^8$ cm is the width of the transition region [4]. At the equilibrium the gas pressure p_0 and magnetic field B_0 are constants. We perturb this equilibrium by launching initially (at $t = 0$) pulses in the mass density and pressure:

$$\rho(x, t=0) = A_d \cdot \exp\left[-\left(\frac{x - x_0}{w} \right)^2 \right], \tag{3}$$

$$p(x, t=0) = A_p \cdot \exp\left[-\left(\frac{x - x_0}{w} \right)^2 \right]. \tag{4}$$

Here $w = 1.25 \cdot 10^8$ is the width of the pulse, $x_0 = 10^9$ denotes its initial position, $A_d = 0.125$ and $A_p = 0.25$ are relative amplitudes of the pulses which excite a packet of slow magnetosonic waves.

2.2. Two-dimensional model

We discuss a two-dimensional coronal loop model with the mass density given by:

$$\rho_0(x) = 1 + \frac{2.89}{\cosh^4(x)}. \tag{5}$$

All quantities are expressed in the units of the ambient plasma. The equilibrium magnetic field had only *y* component:

$$B_y(x) = 1 + \frac{1.97}{\cosh^4(x)}. \tag{6}$$

Waves are excited impulsively by launching the pulse:

$$V_x(x, y, t=0) = \frac{0.1}{\cosh^2\left(\sqrt{(x+10)^2 + y^2}\right)}. \tag{7}$$

3. Numerical tests

3.1. One-dimensional computations

We consider the one-dimensional model which is described in Sect. 2.1. Our aim is to show that for such a simple case there is no need to run in a parallel mode. This is a consequence of the fact that communication between different nodes makes the computational time longer (Fig. 1, Tab. 1). The running option with eight processes on the cluster containing four processors is checked because of the use of the multithreaded processors. All numerical computations were carried up to time $t = 50$ s. The FLASH code was run with the grid of 500 blocks.

The standard device for which MPI is compiled is *ch_p4* which works at workstations or Beowulf clusters and supports SMP nodes [5]. After running *tstmachines* which check the available nodes and choose a remote shell for the sake of the level of security the cluster is properly configured. In the case of the systems with symmetric multiprocessors SMPs we can use the compile option –*comm=shared* which gives us the chance to control the number of processes that communicate on each node. By default the system reserves 4 MB of shared memory but the quantity of reserved memory in which messages are transferred can be increased. Long messages are transferred into pieces so that 4 MB does not pose any problem.

If we possess a multiprocessor computer which is not a part of a cluster we can compile MPI with the device option *ch_shmem* that is appropriate for a single shared memory system which uses shared memory to pass messages between different processes.

From the performed test we can conclude that in the case of both 2 processors and in 4 processors the most favourable option of compilation for the FLASH code is *ch_shmem* device. We must notice that running an application on four processors in that case means running four processes on two processors because that device is not adopted to share tasks by a network. The less favourable option for the SMP cluster is using *ch_p4* device without sharing memory (Fig. 2, Table 2). That fact results from the difficulties in communication between processors.

Table 1. The execution time as a function of a number of processors used by MPI: 0 means running without the usage of MPI on 1 processor

Number of processors	0	1	2	4	8
Execution time	312.647s	333.363s	335.939s	339.920s	347.570s

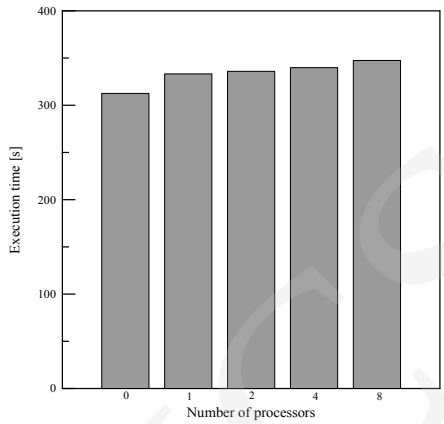

Fig. 1. The execution time as a function of a number of processors used by MPI: 0 means running without the usage of MPI on 1 processor

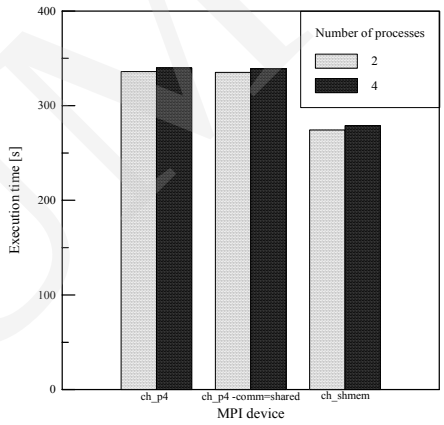

Fig. 2. The execution time as a function of different compilation options for MPI in the case of 2 or 4 running processes

Table 2. The execution time as a function of different compilation options for MPI in the case of 2 or 4 running processes

Number of processes	2		
Device compilation option for MPI	*ch_p4*	*ch_p4 –comm=shared*	*ch-shmem*
Execution time	335.939s	335.264s	274.375s
Number of processes	4		
Device compilation option for MPI	*ch_p4*	*ch_p4 –comm=shared*	*ch-shmem*
Execution time	339.920s	339.132s	278.909s

3.2. Two-dimensional computations

The aim of the first test we perform for the two-dimensional plasma that is described in Sect. 2.2. is to evaluate the execution time for a different number of processes (Fig. 3, Tab. 3). A comparison of execution times of this model and simple models (Fig. 1, Tab. 1) leads to a conclusion that only for complex problems the usage of MPI and parallel computing is justified. All computations were done with the grid of 9000 blocks. We stop these computations at $t = 10\,\mathrm{s}$.

The next test was similar to the test shown in Tab. 2 and Fig. 2 that checked the efficiency of different compilation devices for MPI. As we can see in Fig. 4 and Tab. 4 while running codes on different amount of processors the tendency observed on Fig. 2 (Tab. 2) is not retained. However, for two processors the tendency changes and our statement is no longer valid. Increasing the execution time for the *ch_shmem* device for large number of processes is caused by almost complete memory fulfillment. Slowing down the speed of computations brings about the necessity of the use of swap partition.

Depending on the aim of research the cluster can be configured in two different ways. If we intend to run parametric studies for complex cases like the problem which is modeled by Eqs. (6)-(8) the more efficient option could be configuration of a few unlinked SMP computers working with the maximum of efficiency (the number of processors = the number of processes). However, if the aim of the computations is a model development evolution, e.g. by making it more difficult more favourable option as regards the efficiency is configuring the typical Beowulf cluster and making computations on more than one node for one problem. The maximum of efficiency would be obtained for the case in which the number of processors equals the number of processes.

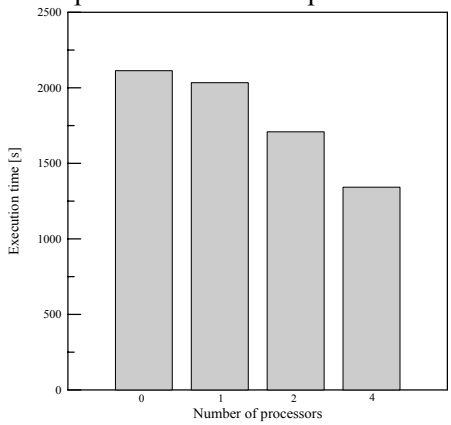

Fig.3. The execution time as a function of a number of processors used by MPI (0 means running without the usage of MPI on 1 processor)

Table 3. The execution time as a function of a number of processors used by MPI: 0 means running without the usage of MPI on 1 processor

Number of processors	0	1	2	4
Execution time	35m 14.099s	33m 54.335s	28m 29.153s	22m 23.061s

Table 4. The execution time as a function of different compilation options for MPI in the case of 1, 2 or 4 running processes

Number of processes	1		
Device compilation option for MPI	*ch_p4*	*ch_p4 –comm=shared*	*ch-shmem*
Execution time	33m 54.335s	34m 15.180s	34m 4.218s
Number of processes	2		
Device compilation option for MPI	*ch_p4*	*ch-shmem*	
Execution time	28m 29.153s	19m 38.073s	
Number of processes	4		
Device compilation option for MPI	*ch_p4*	*ch-shmem*	
Execution time	22m 23.061s	34m 53.505s	

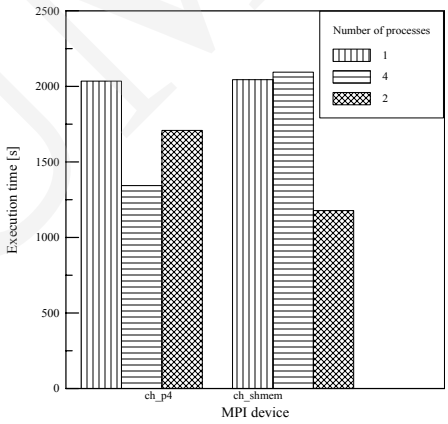

Fig. 4. The execution time as a function of different compilation options for MPI in the case of 1, 2 or 4 running processes

4. Summary

The concept of using multiprocessor computers or linking them into a Beowulf cluster can speed up computations significantly. However, before running numerical codes we should choose the most optimized compilation option which leads to maximum efficiency for the problem.

For simple cases the increase of efficiency with using MPI is low or does not exist. The advantages of MPI interface can be appreciated for the computations of complex problems which require more memory and processor time. In that case the owners of a few SMP computers should consider the basic dillema if it

is better to configure them as a cluster and run multiprocessor applications or use them in parametric studies as separate shared memory systems. The only verification of this conclusion can be done by testing various compilation options of MPI for the planned problems. Such tests we performed with the FLASH code adopted for coronal loop problems.

This work was financially supported by the grant from the State Committee for Scientific Research, Republic of Poland, KBN grant no. 2 PO3D 016 25. The software used in this work was in part developed by the DOE-supported ASCI/Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago.

References

[1] Hargrove W.W., Hoffman F.M., Sterling T., *The Do-It-Yourself Supercomputer*, Scientific American, August (2001).
[2] Sterling T., *How to Build a Hypercomputer*, Scientific American, July (2001).
[3] http://flash.uchicago.edu/flashcode/
[4] Ofman L., *Chromospheric leakage of Alfvén waves in coronal loops*, Astrophysical Journal, 568;L135-L138 (2002).
[5] http://www-unix.mcs.anl.gov/mpi/