A .NET grid computing system applied to Lattice-Boltzmann

Xiaohan Lin*, John R. Williams

Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts, MA 02139, USA

Abstract

The Lattice-Boltzmann method is an appealing numerical simulation technique for Computational Fluid Dynamics due to its easy parallelism. Grid Computing has been proposed as a framework for distributing computations over a network of loosely coupled computers and this paper presents a Grid Computing solution in the context of Lattice-Boltzmann simulation. Both the Grid middleware and the numerical simulation are developed using Microsoft's Common Language Runtime (CLR) and recent standards such as XML, SOAP messaging and Web Services. The goal is to examine the capability of this middleware layer for managing computing cycles over a network of machines and solving engineering problems efficiently with a good programming model.

Keywords: Lattice-Boltzman method; Grid Computing; Web Services; Common Language Runtime

1. Introduction

This Grid Computing middleware, described by Lin et al. [1], targets the problem of distributing numerical simulations across both computer clusters and more loosely coupled machines. This style of computing is often referred to as Grid Computing [2,3]. In Grid Computing, the assumptions that are usually made in cluster computing are *not* valid, including:

- dedicated machines running a single operating system with all resources available;
- all machines reside on the same local network and security is not a concern;
- reliable message passing exists.

In Grid Computing we envisage computers that 'belong' to many different owners in different administrative domains that are willing to provide compute cycles but perhaps with restrictions on resources, such as disk access.

In general a user distributes code to be executed across a network of machines and coordinates the messaging between the machines so that some computational goal is achieved. The coordination of the machines can be achieved by using a master machine. Here we assume that the Master is a 'special' trusted machine that has coordination and other responsibilities not shared by the Workers (Fig. 1).

Details of the underlying Grid environment appear in a previous paper by Lin et al. [1]. This paper describes the Lattice-Boltzmann simulation using some features of the Grid system.

2. The Lattice-Boltzmann method

Scientists and engineers usually describe a fluid flow by introducing a representative control-volume element on which macroscopic mass and momentum are conserved. This leads to a 'macroscopic' mathematical model, governed by the Navier–Stokes equation. More recently, 'bottom-up' particle methods, such as Lattice-Boltzmann, have been formulated based on a microscopic model derived from statistical particle mechanics

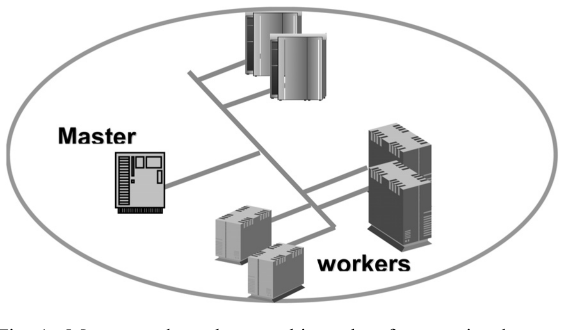

Fig. 1. Master and worker machines that form a simple computational grid.

* Corresponding author. E-mail: linxh@mit.edu

have been developed in fluid mechanics [4]. The motion of the fluid particles is described by particle velocity distribution functions valid at each element (lattice-grid point). The method calculates physical variables such as velocity and pressure by tracking the probability distribution of fluid particles moving in different directions.

The Lattice-Boltzmann equation is given by [5,6,7]:

$$f_i(\overline{x} + \overline{e}_i\Delta t, t + \Delta t) = f_i(\overline{x}, t) - \frac{\Delta t}{\tau}(f_i(\overline{x}, t) - f_i^{eq}(\overline{x}, t)) \tag{1}$$

where f_i is the concentration of particles that travels with velocity $\bar{e}_i$. With the discrete velocity $\bar{e}_i$ the particle distributions travel to the next lattice node in one time step Δt. The relaxation parameter τ determines the kinematical viscosity υ of the simulated fluid, according to

$$\upsilon = \frac{(2\tau - 1)}{6} dx^2/dt \tag{2}$$

The discrete velocity vectors in 2D have the following value and direction:

$$e_i = \begin{cases} dx/dt, i = 0, 1, 3, 5, 7 \\ \sqrt{2}dx/dt, i = 2, 4, 6, 8 \end{cases} \tag{3}$$

where dx/dt is the ration between lattice size and time step.

The equilibrium distribution function f_i^{eq} is calculated as

$$f_i^{eq} = w_i\rho(1 + 3\frac{\overline{e_i u}}{c^2} + \frac{9}{2}(\frac{\overline{e_i u}}{c^2})^2 - \frac{3}{2}(\frac{\overline{uu}}{c^2}) \tag{4}$$

where $w_0 = 4/9$, $w_1 = w_3 = w_5 = w_7 = 1/9$, and $w_2 = w_4 = w_6 = w_8 = 1/36$. The macroscopic density ρ and velocity vector $\bar{u}$ are governed by distribution function

$$\sum_{i=0}^{8} f_i = \rho \tag{5}$$

$$\sum_{i=1}^{8} f_i\overline{e}_i = \rho\overline{u}$$

Lattice-Boltzmann simulates a slightly compressible fluid; consequently, the fluid pressure p is given by

$$p = c_s^2\rho \tag{6}$$

where the speed of sound is given by

$$c_s = \frac{c}{\sqrt{3}} \tag{7}$$

In the lid-driven cavity flow [8] simulation, the boundary meets the no-slip condition. The Reynolds number is 1000 and top lid velocity is 0.2 m/s.

3. Domain decomposition

The computational domain is partitioned into sub-domains to be processed by different worker machines. There are two methods of decomposition over the 2D fields of the simulation, 1D or 2D. In the 1D partition, the whole domain is split in one direction, either vertical or horizontal slices, whereas in 2D partition the domain is divided into two directions, resulting in several rectangular sub-domains. Previous work by Satofuka et al. [9] showed that 2D decomposition is not superior to 1D decomposition for lid-driven cavity problem in a 2D domain. In this simulator only 1D vertical decomposition was studied but this is sufficient to illustrate the important points.

4. Inter-worker communication

Each sub-domain needs to communicate with its neighbors to communicate information about the particles coming in and to send outgoing particle information to corresponding neighbors. In this particular problem and domain decomposition scheme, we need two columns of 'ghost' lattice elements in each sub-domain to store the data for adjacent particles computed by neighbors at the previous time step. The Grid Computing system provides a convenient programming model to manage this communication.

Conventionally, using Message Passing Interface (MPI, [10]) programming, the data are packed into an array of primitive data types before being sent to another computing node. MPI is not easy to program and requires a significant time investment to become proficient. In the Grid system, we have developed higher level abstractions to aid the less skilled programmer master the messaging process. Instead of meticulously writing code for the outbound data, using the Grid system, programmers can send virtually any object directly:

```
...
// Identify myself.
if (this.id == 1)

// Send object ''obj'' to ''2''.
this.SendObj(2, obj);
...
```

The simulation program maintains a 'particle' class. At the end of each time step when 'ghost' particles need to be exchanged, each worker simply sends out the two columns of particle objects to its two neighbors. Each particle may contain many types of data but this is handled transparently.

Receiving an object is also very different in our system, compared with MPI. A data type, 'InBox', is provided as a container for receiving objects. The InBox acts as a message buffer with queuing characteristics that can easily be set by changing the InBox's instance fields. For example, InBox.order specifies how arrived objects line up:

```
...
//First in first out.
inBox1.order=InBox.FIFO;
//Only preserve the last object.
inBox2.order=InBox.LAST;
...
//Oldest object.
obj1=inBox1.pop();
//Last object.
Obj2=inBox2.pop();
...
```

We can achieve blocking receiving, as in MPI, by modifying InBox.mode:

```
...
//Blocking receiving.
inBox1.mode =InBox.BLOCKING;
//Non-blocking
inBox2.mode=InBox.NONBLOCKING;
...
//Blocked here if no object.
obj1=inBox1.pop();

//Always return immediately,
//null if no object.
Obj2=inBox2.pop();
...
```

Event hooks are built into the framework to allow process-upon-receipt using the event handling mechanism in C#. In this case the inbox.Ereceiving delegate will be fired upon message receipt. In the example below we bind our own function 'receiving' to the event handler list so that it will be fired:

```
{
 ...
 //Add event handler.
 inBox1.Ereceiving +=
   System.EventHandler(receiving);
 ...
}
// user defined function
private void receiving(...,...)
{
 //handle the event.
 ...
}
```

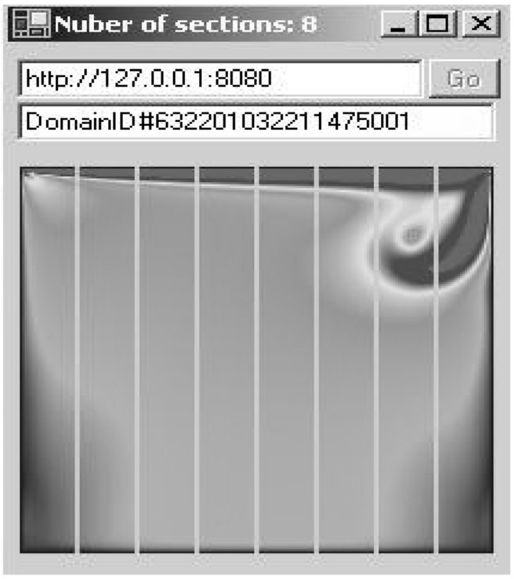

Fig. 2. A real-time GUI client program for lid-driven cavity flow simulator.

In the simulation, we use blocking and first-in-first-out inbox property to receive 'ghost' particles.

5. Expose the result in real-time

The results are stored in a 'Proxy' machine in a form called 'Net Application Domain Globals' or NAD globals [1]. NAD globals are different from the inter worker communication data since they are accessible from the Internet via Web Services and SOAP messaging. The Grid system gives users the power to access data from virtually everywhere on the Web in a machine-independent manner. In the simulation here, a real-time Graphical User Interface (GUI) program was written to monitor the velocity distribution in an intuitive way. In Fig. 2, velocity is profiled from low to high with color from blue to purple. The lattice size is 400×400, divided into eight sub-domains. The gray lines denote the sub-domain boundaries. The lid is at the top boundary.

6. Discussion

One concern in programming using high-level abstractions is that the computational speed of execution is degraded. In particular the efficiency of .NET and the CLR have rarely been tested in high performance computing. The CLR code is compiled to Intermediate Language code (IL) similar to Java byte code. This means that we can distribute IL assemblies to various

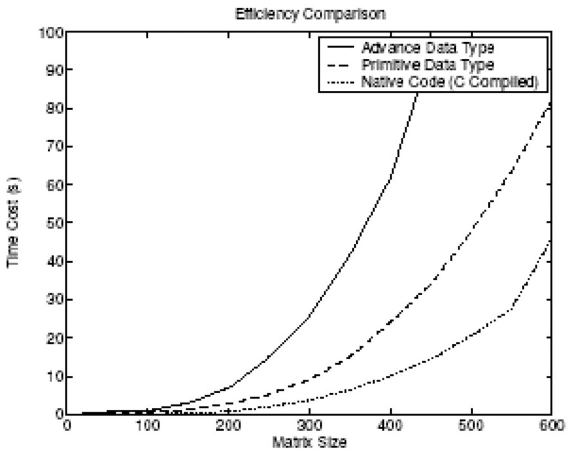

Fig. 3. Comparison of native C, C# with primitive types, and C# with complex types for matrix multiplication.

computer architectures because it is then compiled locally into executable code. However, there is a time penalty for this compilation. To address the efficiency of C# code in numerical computations, a scheme similar to the one Matlab uses was introduced. Manipulation of large volume data was moved to routines developed in C code.

This scheme was tested in the simple scenario of computing the matrix product of two square matrices. The function was wrapped into the 'Matrix' data type. This data type is seen on the third level of GridLib. The usage of this function is shown below:

```
...
Matrix m1, m2, m3;
...
m3=Matrix.product(m1, m2);
...
```

An experiment was developed to measure the efficiency of different code (Fig. 3).

Three types of code were studied, namely: natively compiled code, pure CLR code using primitive types (arrays of double precision values) and CLR code with advanced data types (linked-list of objects). The time cost for different size of matrix and different programming methods are plotted in Fig. 3[1]. Not surprisingly, the native code-based matrix manipulation is the most efficient one. The difference between CLR code using primitive data types and native code is approximately doubled in CLR code, but still within the same order of magnitude. Thus, CLR code can be regarded as 'moderately efficient'.

The inter-worker communication is through a SOAP channel, which is considered slower than tightly packed data directly through network connection. This has been tested and is not a significant issue in solving lid-driven cavity problems by the Lattice-Boltzmann method. The simulator shows almost linear speedup until the sub-domain is too thin (50:1 for high/width ration).

Note

[1] Time measured is only the time for product calculation.

7. Conclusions

The Grid Computing system has been successfully used in simulating cavity flow by the Lattice-Boltzmann method. The quick development and runtime efficiency of the simulator showed that the Grid Computing environment is promising for some engineering problems.

References

[1] Lin X, Williams JR. A parallel computing framework for computer cluster. In: The 16th IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, 9–11 Nov, T. Gonzalez, editor 2004.

[2] Foster I, Kesselman C, Nick J, Tuecke S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Alliance, http://www.globus.org/research/papers/ogsa.pdf

[3] Foster I, Kesselman C, Nick J, Tuecke S. The Anatomy of the Grid-Enabling Scalable Virtual Organizations. Int J of High Performance Computing Applications 2001;15:200–222.

[4] Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. Annual Rev Fluid Mech 1998;30:329–364.

[5] Chen H, Chen S, Matthaeus WH. Recovery of the Navier–Stokes equation using a lattice-gas Boltzmann method. Phys Rev A 1992;45:5539–5542.

[6] Qian YH, D'Humieres D, Lallemand P. Lattice BGK models for Navier–Stokes equations. Europhys Lett 1992;17:479–484.

[7] Bhatnagar PL, Gross EP, Krook M. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. Phys Rev 1954;94:511–525.

[8] Hou S, Zou Q, Chen S, Doolen G, Cogley AC. Simulation of cavity flow by the lattice Boltzmann method. J Comput Phys 1995;118:329–347.

[9] Satofuka N, Nishioka T. Parallelization of lattice Boltzmann method for incompressible flow computations. Comput Mech 1999;23:164–171.

[10] The MPI Forum. MPI: a message passing interface. In: Proc of Supercomputing '93, Portland, Oregon, 1993, pp. 878–883.