4-2002

Performance Analysis of a Parallel Implementation of the Lattice Boltzmann Method for Computational Fluid Dynamics

David M. Cherba
Grand Valley State University

Performance Analysis of a Parallel Implementation of the Lattice Boltzmann Method for Computational Fluid Dynamics

By
David M. Cherba

A Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University

April, 2002

TABLE OF CONTENTS

ACKNOWLEGEMENTS

I would like to express my appreciation to Professor Dr. Greg Wolffe my thesis advisor. His initiative and enthusiasm in promoting student research and use of the Beowulf cluster systems at Grand Valley State University provides an excellent educational opportunity. Professor Wolffe has been extremely supportive with guidance not only for this research study but also with encouragement for continued graduate education. In addition, I would like to thank Professor Trefftz for agreeing to be on the thesis committee and for including me in the trip to Michigan State University computer science open house. That trip was pivotal in deciding to continue my educational pursuits and to Professor Tao who provided insight for the statistical modeling in addition to serving on the thesis committee.

None of this would have been possible without the support and encouragement of my wife Wendy who has made numerous sacrifices that allowed me the time to work on my masters' degree and this thesis. Her understanding and tolerance has been instrumental in my being able to complete this work.

Special thanks goes to John Oleszkiewicz who helped me get started on the Beowulf clusters and whose original code gave me an excellent starting point to construct the working parallel program.

Yes Wendy, that pile of junked old computers I brought home from work is now part of a working Beowulf cluster and contrary to some popular opinion my computer does have grammar and spelling check turned on but it can only do so much.

Faculty support

Dr. Greg Wolffe- advisor Assistant Professor GVSU
Dr Christian Trefftz- committee member Assistant Professor GVSU
Dr Yonglei Tao- committee member Associate Professor GVSU

Research Group

Dr Dewei Qi Assistant Professor WMU Department of Paper and Printing Science and Engineering

Programming and Cluster operations support from John Oleszkiewicz Student GVSU

LIST OF EQUATIONS

LIST OF EQUATIONS

LIST OF FIGURES

List OF TABLES

LIST OF GRAPHS

ABSTRACT

We propose to use benchmark data combined with detailed (n) analysis to predict the performance of a parallel Lattice-Boltzmann Method (LBM) for 2D fluid dynamics simulation with solid particles on various configurations of cluster computers. The LBM has super step synchronism, phase concurrent components and non- critical size of division properties. Our results demonstrate accurate predictions for LBM simulation performance. The CPU benchmark indicates that increased variable data precision does not degrade execution time significantly on Pentium class processors. We show that improved communication and calculation strategies for solid particles yielded better speedup and scalability. A theoretical analysis demonstrates that worst case speedup occurs when all the solid particles saturate a single workspace.

1.0 INTRODUCTION

The Lattice-Boltzmann Method (LBM) is receiving a lot of attention as a method for modeling a broad range of physical phenomenon. The popularity of the method is due primarily to the cellular automata nature of the algorithm and its computational efficiency. Even though the LBM coupled with cellular automata did not start to gain notice until 1986 it has already been used to model heat flow, fluid flows, particle suspensions, and a large variety of aerodynamics problems. The connection between the Boltzmann equation and cellular automata was first developed in 1983 [WOLFRAM83]. This connection combined statistical methods and cellular automata calculation techniques. With an established and efficient fluid flow model it is possible to expand into more complex problems.

The problem domain we are interested in is that of a moving fluid with suspended particles. This particular problem domain has a very broad range of interest. The ability to transport, control, and distribute suspended particles is a key component in many manufacturing processes. A few of these processes are paper production, film production, coating, and surface treatment. Effective simulation of these processes will allow for a better understanding of the science. The ability to visualize the motion and study the behavior of the particles will allow engineers to create better processes. One of the current limitations that affect this problem is poor efficiency when scaling LBM simulations with increased quantities of particles. This study will focus on the issues related to performance of the LBM simulations and the impact of particles on that performance.

Traditional approaches for creating a simulation in this domain have included macroscopic and microscopic interactions. Macroscopic approaches have isolated the time scale and interactions between the suspended particles and the fluids. An example of this approach is to find the fluid velocity profile as a function of time for a given flow problem and use velocity information to determine the forces that the flow would exert on a particle in the fluid. The forces are then used to calculate the motion of the particle. The fact that the particle would also exert forces on the fluid has been entirely ignored. As the interactions between the fluid and particles are tied closer together the impact on the computational resources required are significant. In some cases the computational cost rises as the cube of the number of particles [LADD94]. In addition, nearly all of these methods are based on solving a series of Laplacian equations

applied to a finite difference grid that has a computationally expensive convergence process at each time step.

The microscopic approaches have focused on the Brownian motions of the fluid/gas. Collisions between objects produce forces that affect both the suspended particles and fluid/gas. By its very nature, the number of interactions between all objects on a microscopic level is at least on the order of N^2 and often on the order of N^3 for a full diffusion matrix [LADD94]. Where the value of N describes the size of the simulation space or the number of objects in the simulation. As simulations increase in size the solution method is computationally expensive.

The traditional method for macro simulations of fluid suspended particle type is the Navier-Stokes. This method uses finite difference equations derived from the imposition of a grid arrangement. For complex geometry the grid has to be small or non-uniform to support timely convergence at each time step. Supporting suspended particles adds more computational expense for each time step and introduces problems for non-uniform grid configuration.

In contrast, the Lattice-Boltzmann Method has a mesoscopic nature where the lattice spacing is not critical to convergence or accuracy. The term lattice refers to an array of points that are distributed with uniform spacing in row and columns thought the simulation space. Each of these points is modeled with fluid flow values. As long as the spacing between lattice points is smaller than the particles by some modest factor and the velocity is small in relation to the lattice spacing the simulation generates good results. Even more significant is the fact that each time step for the simulation can be found with a single relaxation equation. The relaxation equation is a completely cellular automaton relying only on local values for the next state. Each state of the model uses a number of discrete flows at each lattice point. The advantage of discrete flows is extremely important when a flow impinges on a particle. Each of the discrete flows that cross a particle boundary exerts a force on the particle and the particle will cause an adjustment to the flow. This combines the macro effect of the fluid on the particle at numerous locations and at the same time accounts for the effect of the particle on the fluid flow. Closely tying the fluid and particle interactions together leads to an accurate simulation.

As the simulation size increases both in volume and in the number of particles, researchers are taking advantage of parallel computing techniques. The LBM is ideal for application to parallel computing

because the cellular autonomy allows for straightforward division of the problem among processors. The real problem becomes the processing of the particles as they cross processor boundaries and the impact they have on the fundamental LBM process. Parallel implementations do not always provide the expected speedup.

The prediction of performance requires careful analysis based on implementation details, communication patterns and performance evaluation of both computation and communication. It is possible that for some applications a small change in a communication parameter can cause a 5-fold increase in solution time [CULLER97]. Prediction of performance for any program can be difficult without knowledge of resource limitations. Even prediction of performance on a single computer for a given problem size can be difficult if certain limitations are exceeded. For example, once a program and data memory exceed the available RAM a significant decrease in performance is observed due to swapping data to and from disk. The relationship between problem size and execution time can have many major and minor discontinuities.

There are two very distinct problems to overcome when predicting the performance of the Lattice-Boltzmann Method. The first problem is to take the mathematical theory of the LBM and transform it into an implementation. Choosing to focus on 2D models allows for smaller resources and eliminates the more complex impact of moment of inertia. At the same time a 2D analysis is complex enough to demonstrate the performance prediction problems for a parallel solution. Key to the implementation is reducing communications associated with particles. Some current implementation methods pass through the entire lattice structure multiple times in order to complete the particle boundary interactions. Particle interaction is expected to have a major effect on the overall performance of the model. Therefore, we developed an efficient implementation that restricts communications to those lattices flows that cross particle boundaries [WOLFE02]. A detailed presentation of the Lattice-Boltzmann Method as it applies to the implementation is presented in section 3.

The second major problem is identification of benchmarks that lead to meaningful performance predictions. Information about communications, CPU computational power and program execution times are needed. There are numerous benchmark programs available for both communications and CPU computational power. Some of the benchmarks strive to characterize memory speeds, disk access, and

general IO communications. These benchmarks are aimed not at providing performance predictions, but at measuring relative performance over a large group of applications. A decision was made to design two simple benchmark programs that produce information on CPU power and communications performance. The processor computational power benchmark focuses on array calculations in a manner similar to the core algorithm of the LBM. The communications performance benchmark focuses on communication patterns for the expected parallel implementation. Final performance prediction also needs to combine a single processor implementation execution time characteristic for the program with the benchmark data. The plan for predicting performance of the Lattice-Boltzmann Method requires the following steps.

Table 1 Outline of Plan

1. Understand the Lattice-Boltzmann Method in enough detail to design a working program.
2. Create a working LBM program with particle interactions capable of parallel computation.
3. Collect benchmark data on the CPU power for each computer considered.
4. Collect benchmark data on the MPI communications for target configurations.
5. Develop a model that determines execution speed for non-parallel parts of the program.
6. Combine time coefficients and the benchmark data to predict the performance of the program in parallel configurations.
7. Compare predictions to actual performance for several configurations.

The ability to accurately predict performance requires three characterizations to be effective: the CPU computational powers, the communications speed parameters and the characterization of the basic program operation. Relative performance indicators provided by the benchmarks provide a basis to make predictions of performance on a variety of computers. These benchmarks combined with the significant focus on implementation of particles in the model forms an effective means for analysis and prediction of performance.

2.0 LATTICE-BOLTZMANN METHOD

At the heart of the Lattice-Boltzmann Method are two very important concepts. First is that the fluid flows of the model are discrete quantities with fixed velocities in quantized directions and second is that given a known rate of change, the final equilibrium state can be calculated. Both of these concepts have some very desirable effects on the overall computation. The model of fluid in the lattice is based on the way that fluid is distributed to all the adjacent lattices at each time step. Imagine a cup of water and turn it over on a flat surface. The water is going to move away in all directions from the point at which the cup is turned over. The rate of flow is going to be almost entirely due to how much water is in the cup. In the case of an LBM simulation we have several thousand cups arranged on a lattice grid that are turned over at the same time. Even as water is moving out from a cup water from adjacent cups is moving in.

2.1 Background information

The origin of the Lattice-Boltzmann method is an integro-differential equation shown in Equation 1. This equation is a function of f(X,t,v) where f is the number of particles with velocity v at location X at step t of time. From the example above it is possible to see how the density of fluid at each lattice point plays a key role in determining the distribution of velocities and quantities.

$$\partial_t f + v \cdot \nabla f = \left(\frac{df}{dt}\right)_{collision}$$

Equation 1 Bolzmann

[Ladd94] show how hydrodynamic fields for mass density, momentum density and momentum flux can be described in terms of f(X,t,v) and m molecular mass. This set of equations for mass, momentum and flux is shown in Equation 2.

$$p(\mathbf{X},t) = \int mf(\mathbf{X},v,t)dv$$
$$j(\mathbf{X},t) = \int (mv)f(\mathbf{X},v,t)dv$$
$$\prod(\mathbf{X},t) = \int (mvv)f(\mathbf{X},v,t)dv$$

Equation 2 Hydrodynamic integral form

The term collision phase or scattering comes from the Boltzmann equation. The collision term can be rewritten in terms of current state and equilibrium state as shown in Equation 3. The term collision is based on the concepts of Brownian motion of particles colliding. From this equation the rate of change is based on the difference between the final state and the current state. For the purposes of the Lattice-Boltzmann Method knowing this rate of change allows for a single step calculation to the equilibrium state. The term relaxation refers to the single step for finding the equilibrium value.

$$\left(\frac{df}{dt}\right)_{collision} = -\frac{f - f^{eq}}{\tau}$$

Equation 3 Relaxation constant

For a 2D lattice, the flows can be discretized in axial and diagonal directions with an additional flow representing fluid that does not stream to adjacent lattice points. Figure 1 shows the representation of the flows at a single lattice point. The figure has nine flows and each flow can thought of as moving in the direction specified in Table 2. These vectors are used to identify boundary crossing relative to lattice point positions and for calculation of the values associated with flows, forces, and torque.

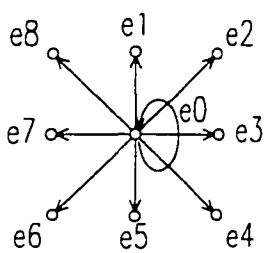


Figure 1 Discrete flow

Table 2 Flow vector values

$\mathbf{e}_0=\{0,0\}$
$\mathbf{e}_1=\{0,1\}$
$\mathbf{e}_2=\{1,1\}$
$\mathbf{e}_3=\{1,0\}$
$\mathbf{e}_4=\{1,-1\}$
$\mathbf{e}_5=\{0,-1\}$
$\mathbf{e}_6=\{-1,-1\}$
$\mathbf{e}_7=\{-1,0\}$
$\mathbf{e}_8=\{-1,1\}$

Another important point in the discrete model is that all the fluid velocities are set so that each flow moves from lattice to lattice in one unit time step. This is the stream step where the flows actually move from lattice point to lattice point. Each of the direction vectors is used repeatedly in the algorithm to calculate mean velocity and force interactions with particles. For the conventions used in this implementation straight up is index 1. Index 0 is the flow that will remain at this lattice point during the streaming step and the remaining flows are labeled in a clockwise direction.

2.2 EDF (equilibrium distribution function)

The function for equilibrium distribution of flows can be calculated by using current local flow values only. The term EDF refers to this equilibrium distribution function. The first step in finding the solutions at each lattice point for the next step is to calculate the total density at this point. The i suffix is for the discrete flow in each direction including the 0 suffix for stationary fluid.

$$p = \sum_{i=0}^{8} f_i(\mathbf{X},t)$$

Equation 4 Total density at lattice point

The next step is to find the mean velocity of fluid at this lattice point using Equation 5. Each of the flow values is multiplied by the appropriate direction vector and the resulting normalized vector is the mean value.

$$\mathbf{u} = \frac{1}{p}\sum_{i=1}^{8} f_i(X,t) \bullet \mathbf{e}_i$$

Equation 5 Mean velocity at lattice

Now that the density and mean velocity at the lattice point is known the equilibrium values can be calculated in each of the nine directions. The axial, diagonal and stationary flows have normalized coefficients that take into account the velocity in each direction.

$$f_{i=0}^{eq}(X,t) = p\left(\frac{4}{9} - \frac{2}{3}(\mathbf{u} \bullet \mathbf{u})\right)$$

Equation 6 Stationary fluid

$$f_i^{eq}(X,t) = p\left(\frac{1}{9} + \frac{1}{3}(\mathbf{e}_i \bullet \mathbf{u}) + \frac{1}{9}(\mathbf{e}_i \bullet \mathbf{u})^2 - \frac{1}{6}(\mathbf{u} \bullet \mathbf{u})\right)$$

i = 1,3,5,7

Equation 7 axial directions

$$f_i^{eq}(X,t) = p\left(\frac{1}{36} + \frac{1}{12}(\mathbf{e}_i \bullet \mathbf{u}) + \frac{1}{8}(\mathbf{e}_i \bullet \mathbf{u})^2 - \frac{1}{24}(\mathbf{u} \bullet \mathbf{u})\right)$$

i = 2,4,6,8

Equation 8 Diagonal directions

Using the equilibrium flow it is possible to calculate the new flow for each direction at the lattice points using Equation 9. This equation is shown in the general form with a relaxation constant of . The equation has values for the current flow in the specified direction and the equilibrium value just calculated.

$$f_i(X,t+) = f_i(X,t) - \left(\frac{f_i(X,t) - f_i^{eq}(X,t)}{\tau}\right)$$

Equation 9 New time step flow

A relaxation constant of τ=1 yields the new time step value as the equilibrium value calculated. The choice for relaxation constant is based on the viscosity of the fluid in the model. For the purposes of this study the value τ=1 is appropaite for the fluids of interest. It is important to note that the time step is shown at t+ not t+1. Equation 10 shows the simplified value for the new flow based on the calculated equilibrium value. The final values for the flow are calculated after the particle boundary interactions.

$$f_i(X,t+) = f_i^{eq}(X,t)$$

Equation 10 New step value

2.3 Boundary interactions

Up to this point in time no interactions with particles or moving walls has occurred. A major advantage of the Lattice-Boltzmann Method is the ease in which particles and fluids interact. This interaction must be calculated before the fluid flow stream to the adjacent lattices points because the particle motion will alter the flows.

The magnitudes of these flow adjustments are based on the velocity at the boundary point where the fluid intersects the particle boundary. There are two important simplifications: the crossing points are assumed to be half way between two lattice points along the vector $\mathbf{e}_i$ and fluid can flow into the solid. The boundary velocity at the flow crossing point has two components. The first is the linear velocity in the x and y-axis. The second component is due to the rotation of the particle and can be calculated given the rate of rotation and the length of the radius to the crossing point. Let **R** be the center of mass for the particle with $\mathbf{X}_b$ the vector from the center **R** to the crossing point $\mathbf{X} + 1/2\ \mathbf{e}_i$. The rate of rotation is Ω and the linear velocity is the vector **V**. The resulting vector velocity V_b is used to calculate the adjustment to flows and the forces. Figure 2 shows the vector diagram for this arrangement. The point X is the exterior point of the flow crossing pair. Only exterior flows create forces on the particle.

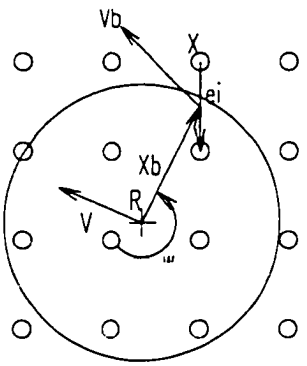


Figure 2 Velocity at boundary crossing

Calculation of the radius vector is made relative to the center of the particle **R** as shown in Figure 2 and calculated in Equation 11. While the actual implementation keeps track of the interior lattice point the flow value associated with the exterior lattice point is the basis for force, torque and flow adjustments.

$$\mathbf{X}_b = \mathbf{X} + \frac{1}{2}\mathbf{e}_i - \mathbf{R}$$

Equation 11 Radius vector

To find the instantaneous velocity at the flow crossing point the rotation of the particle and the radius of rotation add a component of velocity to the point. Equation 12 shows the combination of the rotational and linear velocity components. Taking the cross product of the radius and rotation then adding it to the linear velocity will produce the velocity at the fluid crossing point.

$$\mathbf{V}_b = \mathbf{V} + \boldsymbol{\Omega} \times \mathbf{X}_b$$

Equation 12 Velocity at fluid particle boundary

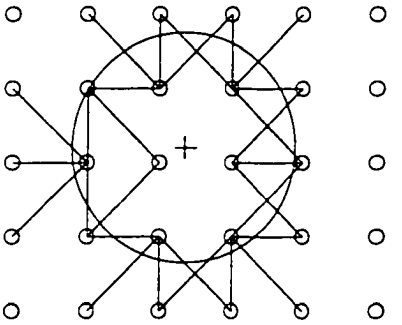


Figure 3 Boundary crossings

Figure 3 shows the entire set of boundary flow crossings for this small circular particle. In this case there are 28 crossings to calculate. Even for this simple particle determining all the crossings is a sophisticated process. The forces and flow adjustments are based on the energy transfer from the fluid to the particle. This energy will be based on the velocity of the flow and the magnitude. For the adjustment and force calculations a set of normalized values that compensate for the diagonal and axial direction differences are shown in Equation 13.

β_i has the value of

$\frac{1}{3}\rho$ for i = 1,3,5,7

$\frac{1}{24}\rho$ for i = 2,4,6,8

where ρ is the total density at the lattice

Equation 13 Axial and diagonal factors

Before the flows can be adjusted it is necessary to calculate the force exerted on the particle at the boundary crossing. The force is determined entirely by the amount of fluid and the instantaneous velocity at the point of crossing. From the Figure 3 it can be seen that each particle can have many crossing points. In Equation 14 the j index is the identification for each crossing point. The force has two effects on the particle. First the force produces an effect on the linear motion of the particle. Second the force acts to spin the particle around its center of mass given as torque causing the particle to rotate.

$$\mathbf{F}_j = 2\mathbf{e}_i\left(f_i(X,t+) - \beta_i(\mathbf{V}_b \bullet \mathbf{e}_i)\right)$$

Equation 14 Force at crossing point j

$$\mathbf{T}_j = \mathbf{X}_b \times \mathbf{F}_j$$

Equation 15 Torque contribution crossing j

Equation 14 and Equation 15 show the calculation for the force and the torque at each boundary flow crossing *j*. Once the force and torque contributions have been calculated it is possible to finish adjusting the flows for each lattice point pair associated with the boundary crossing. Equation 16 and Equation 17 show the final adjustment of the flow values.

$$f_{i'}(X + \mathbf{e}_i, t+) = f_i(X, t+) - 2\beta_i(\mathbf{V}_b \bullet \mathbf{e}_i)$$

Equation 16 Exterior lattice flow adjustment.

$$f_i(X, t+) = f_{i'}(X + \mathbf{e}_i, t+) + 2\beta_i(\mathbf{V}_b \bullet \mathbf{e}_i)$$

Equation 17 Interior lattice flow adjustment

There are several important points concerning this adjustment. It adjusts the flow on the interior node in the opposite direction of the flow impinging on the boundary. This has the effect during the streaming step of placing the adjusted flow as reflected off the boundary. The index *i'* indicates the mirror direction. To complete the flow adjustment the exterior flow must be adjusted by the same amount. The direction of reference is from the exterior lattice point flow vector and so the *i* index must be the mirror image for the interior node.

2.4 Particle collisions

Before the total forces and torque on a particle can be calculated, collision detection must occur. Collisions in this sense are at a macro level, between two particles or a particle and a wall. The collision detection method could examine all pairs of particles and walls for intersections, but that is computationally expensive as the number of particles grows. However, during the boundary crossing calculation it is possible to determine if there is an object close to the particle that needs to be checked for collision. In this case the intersection between objects can be checked on an as-needed basis.

There are two types of collisions in the model. The first is a collision between a particle and wall. This collision results in a reflection of the velocity that is normal to the wall. The walls are assumed to have infinite mass and therefore only the particle is affected. The second type of collision is between two particles and requires determining the vector between particle centers passing though the point of contact. Once this vector has been found then the collision calculation can proceed by finding the component of velocity for each particle projected on the vector connecting the centers. Full momentum transfer must be considered in this case. Even though the particles are uniform in size and mass for the study, the theory and implementation is described for non-uniform particles.

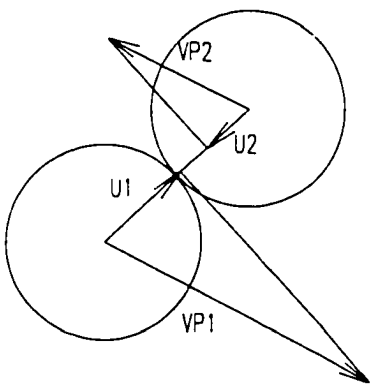


Figure 4 Collision Velocities

The first step in calculation of the collision effect is to determine the component of the velocity vector for each particle aligned with the vector connecting the centers of the particles. In Figure 4 VP1 is the velocity vector of particle 1 and U1 is that component of VP1 along the vector connecting the centers. The same is true for particle 2. Figure 4 shows the projection of the particle velocities on to the unit vectors in the direction between the centers. It is these two velocities that are used in the calculation. Starting with these velocities and the mass the new velocities can be calculated as shown in Equations 18 and 19.

$$v_1 = \frac{(m_1 * u_1) - (m_2 * u_1) + 2(m_2 * u_2)}{m_2 + m_1}$$

Equation 18 New velocity particle 1

$$v_2 = \frac{(m_2 * u_2) - (m_1 * u_2) + 2(m_1 * u_1)}{m_2 + m_1}$$

Equation 19 New velocity particle 2

The total shift in velocity for particle 1 is the difference between the starting velocity and the final velocity along the vector connecting the centers. The same is true for particle 2. To make the adjustment in velocity for each particle it is possible to calculate an impulse force for a time step that will produce the proper change in velocity. Equation 20 shows the change in velocity per time step as a function of acceleration *a*.

$$\Delta V = v_1 - u_1 = \Delta t * a$$

Equation 20 Change in velocity

Unitizing the common relationship between force, mass and acceleration it is possible to determine the amount of force to apply in one time step to produce the desired change in velocity as shown in Equation 21.

$$\Delta V = \Delta t * \frac{F}{m}$$

Equation 21 Velocity change in terms of force and mass

As shown in Equation 22 and Equation 23 the force must act in the direction of the vector between centers. Adding the unit vector along the centers $\mathbf{X}_c$ and setting t=1 yields the force impulse vector that will provide the exact changes to the velocity for the particle as the result of the collision.

$$\mathbf{F}_{collision\_1} = m * (v_1 - u_1)\mathbf{X}_c$$

Equation 22 Impulse force vector particle 1

$$\mathbf{F}_{collision\_2} = m * (v_2 - u_2)\mathbf{X}_c$$

Equation 23 Impulse force vector particle 2

Using an impulse force has the added benefit of allowing multiple collisions with different particles. When a simulation is configured for parallel execution the ability to summarize the net effect of

multiple collisions calculated in separate processors is required. The use of impulse force provides the means to summarize the collision results and combine them with the fluid forces. For particle collisions with walls, the velocity in the x direction is used to calculate the impulse force required to bounce off the wall.

2.5 Total Forces

Once the individual forces and torque for each flow crossing have been calculated, the effect on the particle can be determined. This process has two distinct parts. The change in position is calculated and the change in rotational angle is calculated. These calculations are very similar and although the change is position is shown first the order is not important to the method. Table 3 summarizes the steps for the update in position calculation. The total forces on the particle have contributions other than the boundary crossings

Table 3 Summary of position update process

1. Sum all the forces on the particle
2. Find the acceleration vector due to the forces
3. Use the old velocity together with acceleration to find new position
4. Update the velocity of the particle

$$\mathbf{F} = \sum_{j=1}^{j=n} \mathbf{F}_j + \sum \mathbf{F}_{collision} + \sum \mathbf{F}_{in} - \sum \mathbf{F}_{out} + \mathbf{F}_{gravity}$$

Equation 24 Total force on particle

. Note that gravity and interior lattice points can also have some effect on the total forces. The forces include the boundary crossing contribution, impulse forces from the collisions, forces of gravity and forces that recover energy from the lattice points that enter and leave the interior of the particle [QI99]. Equation 24 shows all these terms. These last three terms are not implemented in the current model but are shown for completeness. The force **in** and **out** is based on the mean velocity of the lattice points that transition from the perimeter to the interior. The gravity term represents the force exerted on the particle by the earth. The relative density of the particles is greater than 1 so that particles without other forces eventually sink to the bottom of the experimental space. Each time step is assumed to have value t of 1 and this simplifies the calculations.

$$\mathbf{a} = \frac{\mathbf{F}}{m}$$

Equation 25 Acceleration vector

The total force for each particle is known from summation of contributions. Equation 25 calculates the acceleration that will take place this time step. Acceleration is used to find the new velocity at the end of the time step. This new velocity and the old velocity are averaged over the time step to determine the movement of the particles. Equation 26 shows the new position in terms of the old position and the average of the old and new velocity.

$$\mathbf{R}(t+1) = \mathbf{R}(t) + \Delta t \frac{1}{2}(\mathbf{V}(t+1) + \mathbf{V}(t))$$

Equation 26 New Position as the average of old and new velocity

$$\mathbf{R}(t+1) = \mathbf{R}(t) + \Delta t \mathbf{V}(t) + (\Delta t)^2 \frac{1}{2}\mathbf{a}$$

Equation 27 New position in terms of old velocity and acceleration

Substitution of the acceleration into Equation 26 provides the new position as a function of the old velocity, old position and the new acceleration. The last step in the procedure is to update the velocity from the old velocity and the new acceleration as shown in Equation 28.

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \Delta t \mathbf{a}$$

Equation 28 Update new velocity

Once the linear forces on the particle have been used to update position and velocity the effect of rotation can be calculated. The process is very similar in nature to the one used in the calculation of position. The new angular acceleration is used to determine new angle of rotation, then the angular velocity is updated. As with the position, the average of old and new velocities are used. The contribution to the torque of lattice points that enter and exit the interior of the particle are not implemented in the model but are shown for completeness in Equation 29. Also, no effects on the rotation of the particle from collisions have been included in the model. The moment of inertia for a particle is represented by I, for the 2D case it is a simple scalar in the z direction. Also in the 2D case the torque value is a simple scalar in the z direction as well.

$$\mathbf{T} = \sum_{j=1}^{j=n} \mathbf{T}_j + \sum \mathbf{T}_{in} - \sum \mathbf{T}_{out}$$

Equation 29 Total torque

$$\varpi = \frac{\mathbf{T}}{\mathbf{I}}$$

Equation 30 Angular acceleration

Equation 30 shows the relationship between moment of inertia and torque. This relationship is defined by the angular acceleration . Following a process similar to the position calculation the new angle is shown in terms of average rotational velocity in Equation 31. Then the new angle is determined in terms of old angular velocity, old angle and new angular acceleration in Equation 32. Finally the angular velocity is updated in Equation 33.

$$A(t+1) = A(t) + \Delta t \frac{1}{2}(\mathbf{\Omega}(t+1) + \mathbf{\Omega}(t))$$

Equation 31 New angle in terms of old and new angular velocity

$$A(t+1) = A(t) + \Delta t \mathbf{\Omega}(t) + \frac{1}{2}(\Delta t)\varpi$$

Equation 32 New angle in terms of old velocity and new angular acceleration

$$\mathbf{\Omega}(t+1) = \mathbf{\Omega}(t) + \Delta t \varpi$$

Equation 33 Updated angular velocity

$$\mathbf{I} = \frac{1}{2} m r^2$$

Equation 34 Disk Moment

In the study circular particles are used that are modeled as disks. Equation 34 shows the calculation for the moment of inertia for a disk. Normally moment of inertia is a vector quantity with components in all three axes based on the shape of the object, but for the 2D case of a disk type particle the value reduces to a scalar in a single axis. In the cross product of force and radius the torque value calculation will only have a term in the z-axis. This simplifies the calculation. In the axis of rotation the particles use the moment of inertia shown in Equation 34.

2.6 Streaming step

The final element to the Lattice-Boltzmann Method is the stream step. This step takes all the newly calculated flow values and moves them in the direction of the flow to the adjacent lattice point. It is very simple in nature. Each new flow at each lattice point is moved to the old flow at the adjacent lattice point in the direction of flow. Equation 35 describes the process. For all lattice points in the simulation and for all flows in the simulation move the value from the old to the new in the direction of flow.

$$\forall \mathbf{X}$$
$$\forall i = 0...9$$
$$f_i(X, t+1) = f_i(X + \mathbf{e}_i, t+1)$$

Equation 35 Stream step

2.7 Lattice-Boltzmann Method summary

The LBM process starts with the equilibrium calculation for the new flows in each direction of adjacent lattice points. New flows are calculated entirely from local information at the lattice point. Once the flows have been calculated the boundary interactions are calculated. Any collisions can also be detected and the appropriate impulse forces calculated. The flow crossing the boundary of the particle is adjusted to allow for the interaction between the particle and fluid. Once that is completed the stream step can take place and all flows migrate one unit step in the proper direction. The entire process is computationally efficient with only local variables used in the calculations or at most the flow from one adjacent lattice point. It is this adjacent flow that forces the parallel implementation to have a halo of duplicate lattice points. This reduces the requirement for time critical individual communications.

3.0 IMPLEMENTATION

There are two key parts to the implementation. First is the concepts that have been developed to improve the implementation and the constraints that the Lattice-Boltzmann Method places on those concepts. The other part is the structure and design of the program. Both of these parts have a major influence on the outcome of the study. Implementation concepts that are significant will be discussed followed by a summary of the program structure with the application of the key concepts.

3.1 Implementation concepts

The implementation strategy has a major impact on performance, and starting with the Lattice-Boltzmann theory and finding an effective way to implement the method was challenging. The details of the theory have subtle constraints that for an effective implementation must be considered. This is especially true of particles that span multiple processors. Another key issue is the difficulty in scaling the problem on parallel computers with increasing number of particles. Minimizing the impact of particle motion and computation was a key goal for this implementation.

In some of the original implementations studied the entire lattice array is repeatedly examined in order to accomplish all the necessary steps. Reducing the number of passes through the lattice is important for two reasons. These passes represent a significant computational cost and also require additional storage. Further, the communications between processors impacts the execution speed of parallel programs. The Lattice-Boltzmann Method is not a communication critical type of program, but minimizing the communication costs still improves performance.

In this study the motion of solid particles is the key point of interest. If however, the key point of interest includes the fluid flow data the implementation strategy would shift to improving other areas such as storage of data, since disk IO becomes the bottleneck.[RESCHKE96] [DESPLAT99] Even a small space of 100 by 100 produces 90,000 data values to record every cycle. This amount of data presents a problem for most disk IO systems. For the case where particle motion for 1000 particles is the focus, only 3000 data values would need to be recorded.

The entire program is written in SPMD (Single Program Multiple Data) style. Each processor in the cluster runs the same program but acts on a different set of data. The clusters used for this study are not homogenous. Each cluster has a master processor that is different from the computational processors. In

some cases the main node is higher performance and in others it is slightly slower. The slowest processor in each cluster determines performance. The clusters use standard Ethernet cards, switches and PC based computers. The Lam version of MPI is used on all clusters to provide communications.

The style of communications is primarily point-to-point making use of non-blocking messages wherever possible. The non-blocking messages avoid a potential communication deadlock for certain configurations of processors. [SNIR98] Because the communication takes place in bursts to adjacent processors and communication must be complete before processing can continue a Wait_All MPI command is used to determine when communication is complete. [PACHECO97] This allows processors that have completed communications to proceed in contrast to a Barrier call that blocks all processors until they have reached the barrier point..

A processor's location in the overall lattice space is determined by it's rank in the MPI structure. Lowest rank contains the 0,0 point and moves left to right across the columns and then up the rows. The program is constructed without a restrictive arrangement. That is, arbitrary arrangements of rows and columns can be processed.

3.1.1 Stream Step

A primary issue in our implementation is improvement of the streaming step. From the CPU power benchmarks it is evident that array index calculations are as computationally expensive as floating point calculations. In this implementation a setup step pre-calculates the destination addresses of all the flow streams and stores them in the lattice data structure. This improves performance of the stream step by eliminating the complex series of index calculations that is found in many other implementations.

Not described in the theory is the problem of what to do with the stream flows at the edge of a processor space. In the parallel implementation those streams must be communicated to adjacent processors. In fact, even in a single processor space the stream flows wrap around the lattice making each processor space similar to the surface of sphere. This has the key effect of conservation of mass in the model. For those sides with walls the effect of the stream is small and for the ends without walls the wrapping effect maintains the fluid momentum. In a parallel implementation the lattice points from adjacent space must share at least some flow values. When particles are present near the edge of the lattice

space a halo of duplicate lattice points is required [DESPLAT99]. Around each lattice space in a processor there needs to be a ring of lattice points from all the adjacent lattice spaces. The need for this ring is demonstrated by reduction in time critical communications. This is true for particle interactions and streaming. The ring of lattice points resembles a halo.

During the evaluation of boundary flow crossings it is necessary to adjust flows from adjacent lattice points. If the points reside in adjacent processors the Lattice-Boltzmann Method requires timely communications for the flows affected. In a parallel implementation, if the lattice points on the perimeter are duplicated during the streaming process the need for communication is eliminated. Duplication of the lattice points achieves this goal. This eliminates the need to communicate adjusted flows for boundary crossings and allows the added benefit of having complete flow data for the stream step data prior to communication. Not only is there a duplicate set of lattice points on those perimeters with adjacent processors but communication of all 9 flows per lattice point is required instead of the minimum 3 flows. Because the cost per communication is large with respect to the cost per unit length this trade off of duplicate lattice points and additional flow data is desirable. The incoming and outgoing flow data each has a complete array that is packed and unpacked manually. This negatively impacts efficiency by duplicating lattice calculations and data manipulations. The advantage is that communications and processing can take place at the same time. For this implementation no special design to take advantage of the concurrent possibilities was implemented. However, provisions for this are included in the program structure.

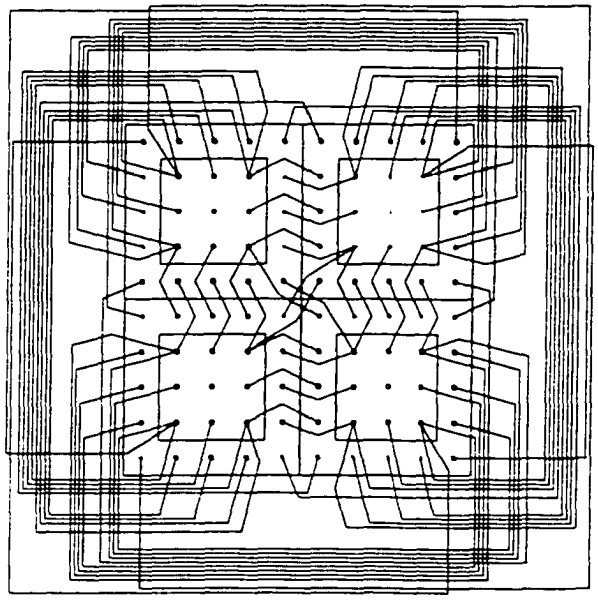


Figure 5 Communication pattern of 2 X 2 processors

Figure 5 shows a highly simplified arrangement of lattice points. Each space is 3 by 3 lattice points. The figure shows the halo around each processor space that is made up of duplicate lattice points from the adjacent spaces. The lines represent the flow of data between the processor spaces. Even for this very simple representation the mass of communications and the arrangement of data is significant.

3.1.2 Particle implementation

There are several key concepts and data structures that are incorporated into the implementation of the solid particles. A list of all the flows crossing particle boundaries is maintained for each particle. Each processor is responsible for creating the list of crossings for particles currently resident in its space. The space with the center of mass of a particle is responsible for summarizing the forces and calculating its next position. This implies only two communications between adjacent processors. One communication provides the force and torque contributions to the processor with the particle's center of mass. The second communication provides an updated position from the processor with the center of mass to each processor that part of the particle resides in.

In this implementation having a list of particle flow boundary crossings is advantageous because the entire lattice space does not have to be examined checking for the crossings. This list keeps track of the interior lattice points that have flow boundary crossings and the direction of the crossing. During

generation of the list the individual crossings are checked instead of examining the entire lattice. This eliminates a pass though the lattice and focuses the checking for boundary crossings.

This method of keeping track of the particles implies that two communications are needed between adjacent spaces to totally account for the particle motion. Each lattice space creates a packet containing all force contributions for boundary crossings in its space caused by particles whose center is in the adjacent space. After the "owning" processor space has received force packets from all it's adjacent spaces it adds its own force contributions and calculates the new particle position. Using this new position it determines which adjacent spaces need particle position information and prepares a packet for each adjacent space. The adjacent spaces receive these packets and proceed to update their list of positions and boundary crossings. If an adjacent space had a portion of a particle in the last time step but has no portion this time, a final position update is needed to clear the old position. This update will indicate that the particle no longer resides in that space and will prevent erroneous force contributions.

Because there is a duplicate row of lattice points around the perimeter of the processor lattice space there needs to be an additional check before adding the contribution of force from a particle boundary crossing. The duplicate lattice is used to provide the means for correct flow adjustment prior to streaming. Only those flow crossings where the exterior lattice point is in the current space, not the duplicate lattice space, will be added to the force contributions. The shaded lines in Figure 6 identify which of the crossing will be used in the total calculation of forces and torque on this example particle.

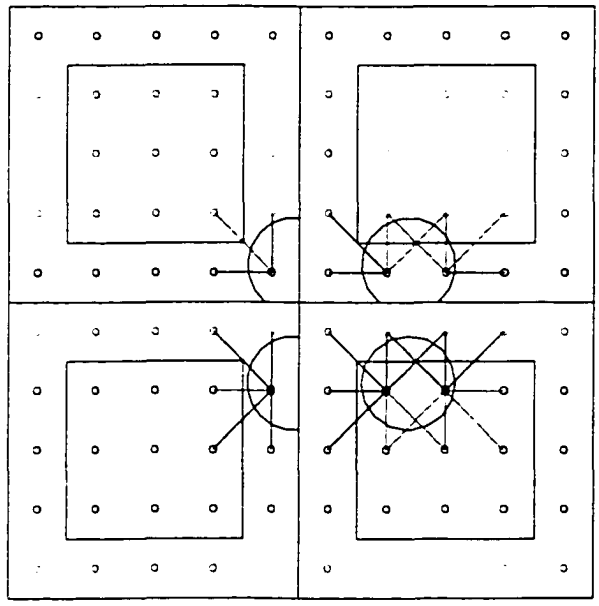


Figure 6 Particle spanning multiple spaces

3.1.3 Particle collisions

In a similar fashion, only collisions that take place in a given processor space are calculated and added to the force contributions. Because a collision is only calculated once between any two particles in a time step but may be checked repeatedly when two particles are in close proximity, a space maintains a collision history structure. This structure holds information concerning collision calculation for a given two particles in the space. When an intersection check is requested, a check for a prior calculation is performed. If a collision calculation exists then no further processing is performed.

The introduction of an impulse force provides a convenient way to communicate the results of a collision between two particles. Isolating the collision detection within a space reduces the need for additional communications, but each processor space may have packets with little or no data. The processor time cost for small packets is significant and reduces the efficiency, but compared to the All_Gather process and broadcast used to share particle data in other implementations this represents a significant improvement. A final constraint for collision detection is needed. The fact that two objects intersect is not sufficient to determine that a collision has occurred. It is necessary to determine if the two objects are actually converging. Failure to do this results in an oscillation.

3.1.4 Sequence of execution

The Lattice-Boltzmann Method has a set of requirements for the sequence of execution. The EDF process must be completed for all lattice points and then the stream step can occur. After streaming, sharing the halo of duplicate lattice points is needed. For the particle process, the order is making the list of boundary crossings, calculating each boundary point and then sharing the force contribution. Once the force contributions are available the new position is calculated and then shared with those adjacent processor spaces that need the new position. While both sequences are simple they do require some critical points of coordination. Figure 7 shows the two cycles side by side with the critical points identified. The boundary calculations cannot proceed until the EDF process is complete and the streaming can not occur until after the particle boundary calculations have adjusted all the flows. The order in which the various steps of the Lattice-Boltzmann Method and the particle process are completed can be mixed as long as the individual process requirements are satisfied along with the critical points.

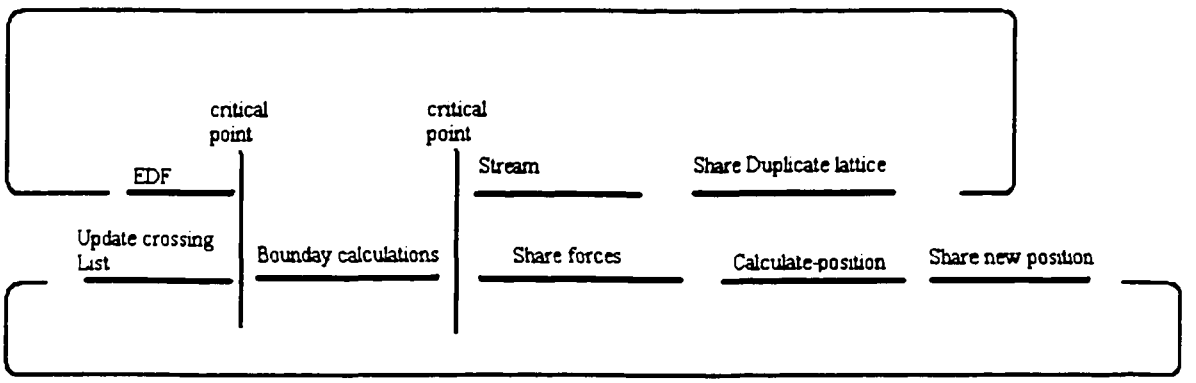


Figure 7 Critical points

3.1.5 Arrangement of processor spaces

The parallel program is constructed in such a way that the lattice space can be arranged in either vertical or horizontal slices and in X by Y block arrangements. Most of the test data is shown for multiple vertical slices by a single horizontal column arrangement. The individual processor determines its location base on its MPI rank. Using this procedure the rank of each processor adjacent to the current processor can

be determined. The rank is the main method for directing communications. Communications patterns are configured to allow eight adjacent communications as required when configured in X by Y blocks. Additional buffers are constructed to allow the greatest flexibility in the arrangement of the main loop in order to explore concurrent communications and processing. The eight directions include the four axial adjacent processors and four diagonally adjacent processors. No special method to minimize the diagonal communications has been implemented.

The risk of deadlock communications can occur for some communication patterns based on the processor arrangement and command selection. Many patterns of communications can become deadlock or order sensitive without special care [SNIR98]. No restriction on arrangement of processor spaces was imposed in this implementation. Extensive use of non-blocking MPI commands avoids potential deadlock configurations. Checks are made for needless communications that may be indicated by attempts to communicate with itself.

3.2 Program Structure

The main loop of the program is fairly short and contains only key tasks that need to be completed during each pass of the Lattice-Boltzmann Method. The current structure of the communications has not been configured to take advantage of any concurrent opportunities at this time. The basic constraint on the order of operations is enforced by a sequence of procedure calls, since each call must complete prior to the next call. Communication procedures are coded in such a way as to allow for multiple phases of communication. The first call sends all the data and the second call to the procedure is configured to receive data and place it into the proper locations.

3.2.1 Recording Data

The record data procedure is primarily concerned with recording the positions of solid particles. The particles tend to move slowly so the recording system only records position once every N passes. All particles must be recorded and this places an extra burden on the main processor in the cluster to initiate IO communications with the disk drive. In addition, passes with data recording have significant increases in messages sent to the master computer in the cluster. The rate of increase depends on the frequency with which the data is recorded. If data is recorded on the average of once per 10 passes the average impact will be 1/10 the worst case time for the extra communications and storage overhead on that pass.

3.2.2 EDF Process

The EDF process is the calculation of the next state for each flow at each lattice point. This represents the major time execution in the Lattice-Boltzmann Method for typical populations of solid particles. This does not include the stream step of the process, only the calculation of the next flow values from current flow values. This step must be completed before boundary calculations can be completed. This process makes use of duplicate flows. One set of flows is time step t and the other is the eventual result of the new time step t+1. This duplicate set of flows has several advantages. The stream step does not have to concern itself with buffering a temporary value to prevent loss of data. Another advantage is that multiple adjustments can be made to the new flows before the stream step.

3.2.3 Paired list creation

The use of an updated pair list is new for implementing the Lattice-Boltzmann Method. In other implementations it is common to mark each lattice point as belonging to a particle, then traverse the entire lattice to identify flow crossings. In our implementation the lattice points are still marked for detection of likely particle collisions during evaluation of each flow crossing. However, only those crossings on the list are processed. Examining all the lattice points contained in a particle's bounding box for boundary flow crossings is used to create the list. Only those lattice points that are interior to the particle are checked for crossings. The list identifying the flow boundary crossings is distributed to the processors that have a portion of the particle. If the particle spans more than one processor then each processor is responsible for creating the portion of the list involving lattice points in its space.

3.2.4 Boundary Crossing

The boundary flow crossing procedure performs the following activities: calculate the force and torque contribution, checks for collisions between particles and walls, check for collisions between particles, adjust the flow values for the boundary crossing flows. This simple list of activities has several constraints. Because of the duplicate lattice points care must be taken not to replicate force and torque contributions. Also multiple processor spaces may detect particle collisions. To insure that this replication

does not take place additional checks are required. Only those boundary flow crossings where the external lattice point is in the current space provides force and torque contributions. Only collisions that take place within the limits of the non-duplicated lattice points are calculated and only objects that are moving towards each other are considered to collide.

For all boundary flow crossings the flow adjustments are made to facilitate streaming without waiting for communications. This also allows for various arrangements of processor spaces without additional programming steps.

3.2.5 Communications

After the paired lists have been calculated it is possible to stream the flows and send the results of the particle forces to processors that need them to calculate the motion of the particle. Only those forces associated with particle whose center of mass are in adjacent processor spaces will be shared. The size of this communication packet is generally small because the probability of a particle spanning two spaces is low for non-saturated particle distributions. The "communicate particle forces procedure" sorts though all particles that have some portion in its space and builds communication packets for all processors in adjacent spaces. Then the stream procedure moves the flows from one lattice point to another. Because the perimeter lattice points are duplicates the flows can be streamed without waiting for the communications of duplicate lattice points.

After the stream step is complete the duplicate lattice points must be shared with adjacent processors. The program retrieves the 9 flows from each point duplicated and packs them in a contiguous array to send. This procedure also receives the duplicate lattice points from the adjacent processors, unpacks them and transfers the flow values to the actual lattice points.

One final communication is required after the particle position has been updated. This will be described in section 3.2.6.

3.2.6 Particle position updates

The processor that owns the center of mass of a particle is responsible for accumulating the forces acting on it and calculates it's new position. "Update particle position" does all the calculations for creating a new position every pass. The calculation averages old and new velocities. Once the new position has

been calculated for all the particles with center of mass in a space a communication is required to transmit the new position to all adjacent processor spaces.

The procedure "communicate particle position" sorts through all particles in the current space and sends the position data to any adjacent spaces that require it. The procedure builds a packet of information for each of the adjacent spaces and once all the particles have been checked sends the packets. These position update packets are expected to be small in size based on the probability of a particle spanning two adjacent spaces.

3.2.7 Synchronizing

The final part of the main loop is the bulk synchronization barrier that requires all processors to reach this point in the program before continuing. The nature of the communication pattern during program execution tends to keep the processors in step throughout the main loop. The main purpose of this synchronization is data collection, which needs to avoid recorded data skew. This is necessary because no means to record pass number is included with the data. The data file is expected to be in order of passes.

Table 4 Main loop code

```
record_data();
edf();
update_pair_list();
calc_pair_list();
comm_part_forces(0);
stream();
comm_perim(0);
update_particle_pos();
comm_part_motion(0);
MPI_Barrier(MPI_COMM_WORLD);
```

4.0 RESULT

The result of this study is broken into three major areas. Collection of benchmark data is the first major area. The benchmark data collection is divided by CPU, communications, and format of communications and single processor program execution. From the benchmark data, a series of performance predicts are made that apply to multiple Beowulf clusters. The measured performance of the multiple clusters is shown in section 4.3

4.1 Benchmark information

The use of benchmark programs is nearly always problematic. In the most common form the purpose of benchmarks is to allow people to judge the relative merits of two items. The problem is that a benchmark cannot always compare the exact way the items will be used. For example, buying a hammer based on the weight alone does not take into account the length of the handle or the shape of the head. Performance benchmarks include a wide variety of operations and attempt to weight them based on their relevance to common problems. Benchmarks for high performance computing typically run series of application problems whose nature is well understood. The timed performance is used to make a judgement regarding communications and processor computational capability. Another style of benchmarks is simplified so that they mimic the expected operations of a given problem. To show how misleading they can be [HENNESSY90] uses an example of a simple benchmark. In the example, a Hitachi S810/20 and a Cray X-MP are compared. The small benchmark program used is identical in structure to the one proposed for our study. The results showed that the Cray was two times faster for the small benchmark but was almost two times slower running a typical application program.

The search for a suitable benchmark to use for performance prediction of the Lattice-Boltzmann Method focuses on three areas. The first requirement is to measure in some basic way the CPU ability to process the method using a series of floating point calculations on an array of structures. The goal of this benchmark is to provide an estimate of the execution speed for a given portion of the program on a specific CPU. This is accomplished by collecting information on the CPU to be used in the experiment and calculating a simple ratio for each unit based on the slowest processor.

The second focus is the way communication impacts the overall process. Communications performance is characterized by using several general parameters. The logPG format for parameters is an attempt to provide a way to predict the impact of communication parameters on the performance of parallel programs [CULLER97].

The last requirement is a way to measure parts of program execution in a single processor implementation. The importance of knowing how long portions of the program take to execute is needed to predict effects of size and scale. All three areas are important to the accurate prediction of performance.

4.1.1 CPU Benchmark

The benchmark used to measure the ability of the CPU to process the Lattice-Boltzmann Method is based on the example used in [HENNESSY90]. The example was used to compare two very different high performance computers. Because our research is restricted entirely to cluster computers based on a common PC architecture the difference is expected to be much less significant. For comparison purposes the SPECFP95 benchmark is also used. The most recent SPEC benchmark was not used because there is no published data available for the older CPU used in two of the clusters studied.

There are many factors that affect the usefulness of a benchmark such as operating system, compilers, and memory. Therefore, the CPU benchmark is intended only to yield an overall ratio in order to calibrate known execution times on one type of CPU to another. The application program is designed in such a way as to avoid major performance degrading factors as memory exhaustion, disk IO and cache memory impacts. In the case of cache memory the board architecture and chip set impact is included in the measurement of overall CPU performance. Another issue is the multi-tasking nature of the Linux operating system and its impact on the benchmark. To insure that an appropriate impact is measured the benchmark test was run for a sufficient duration to allow for the inclusion of asynchronous system tasks.

A serious concern in the design of the Lattice-Boltzmann Method program is the impact of data precision on performance and on the accuracy of results. Specifically, what is the effect of lower precision variables on execution speed? Based on these concerns, a benchmark was designed to measure the performance of a series of calculations on array elements, similar to the basic EDF phase in the Lattice-Boltzmann Method. The operations are noted in short hand form where I is a normal integer, L is a long, F

is a float and D is a double precision variable. The benchmark uses two cases for each precision level. One involves a multiplication and assignment and the other involves two multiplication, one addition and one assignment.

Preliminary studies with the benchmark program indicated that array index calculations were a significant part of the operation. To this effect one CPU showed more than a 2 to 1 difference in MFLOPS between the two test cases for the same precision variable. A 50% deviation in benchmark performance makes it difficult to predict any type of performance. Comparing some of the published data on CPU performance to the results of the benchmark it was easy to see how actual performance of a program could be twice as bad as the published data might suggest. By examination of the calculations used in the benchmark it was possible to produce a consistent value of performance for each CPU via an operations count that includes each index calculation, each assignment, and each arithmetic operation used, instead of counting arithmetic operations only. This performance measure is termed Mega operations per second (MOPS) and Equation 36 demonstrates how the operations are counted.

$$F(j) = F(j+1) * F(j+2) + F(j+3) * F(j+4)$$
$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \text{ total ops count}$$

Equation 36 Operation counts

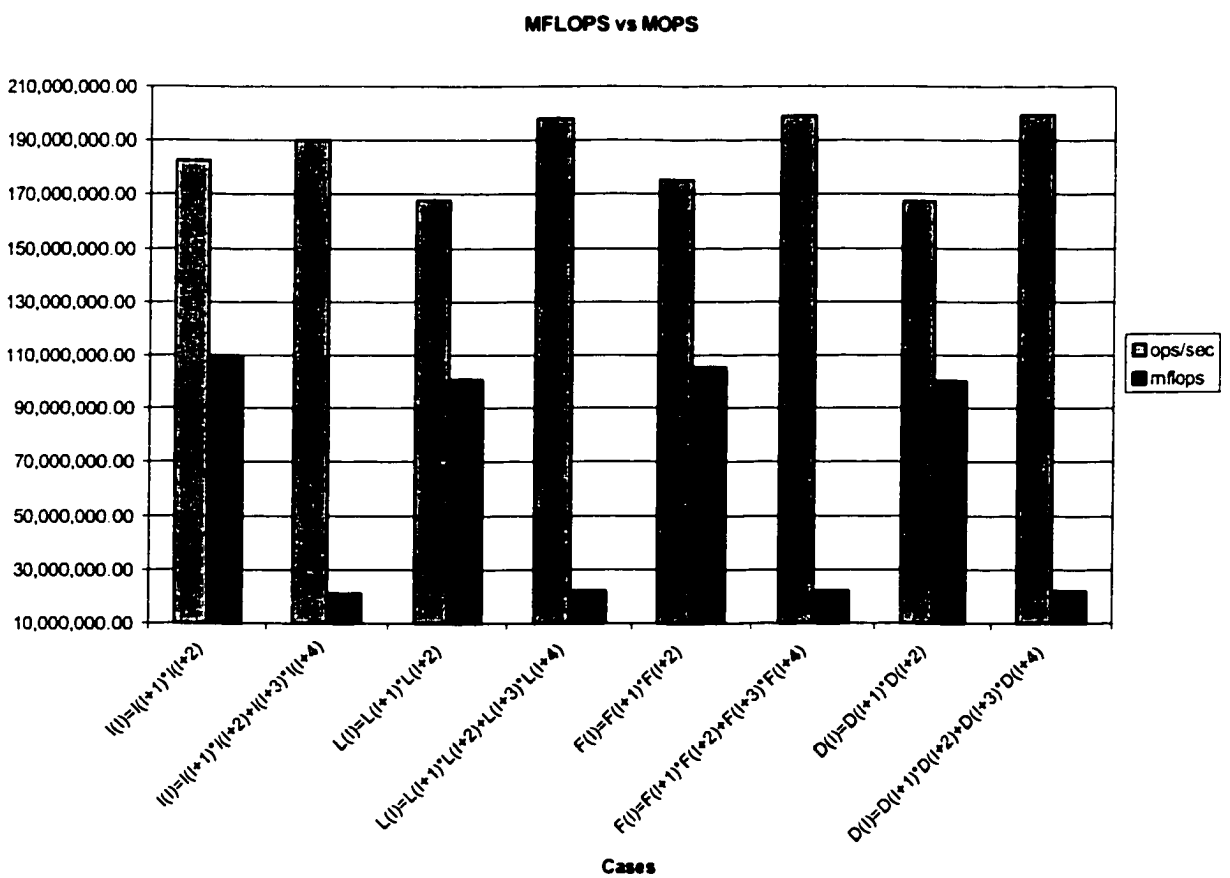
The original MIPS measure of instructions per second has been shown ineffective when comparing different computer architectures. The attempt to find a better measure for comparing CPU led to the introduction of MFLOPS as a metric. However, even for the simple benchmark this does not provide a consistent tool for predicting performance.

Using the total operations count and factoring in the timed execution for each CPU that is part of a cluster configuration provides a metric that characterizes each CPU. In Equation 36 the *F* stands for floating point variables and j is the array index. A 5000 element array is created and used for repetitive calculations. Table 5 list the eight cases executed as part of the simple benchmark test.

Table 5 Case definitions

case 0	I(j)=I(j+1)*I(j+2)
case 1	I(j)=I(j+1)*I(j+2) +I(j+3)*I(j+4)
case 2	L(j)=L(j+1)*L(j+2)
case 3	L(j)=L(j+1)*L(j+2) +L(j+3)*L(j+4)
case 4	F(j)=F(j+1)*F(j+2)
case 5	F(j)=F(j+1)*F(j+2) +F(j+3)*F(j+4)
case 6	D(j)=D(j+1)*D(j+2)
case 7	D(j)=D(j+1)*D(j+2) +D(j+3)*D(j+4)

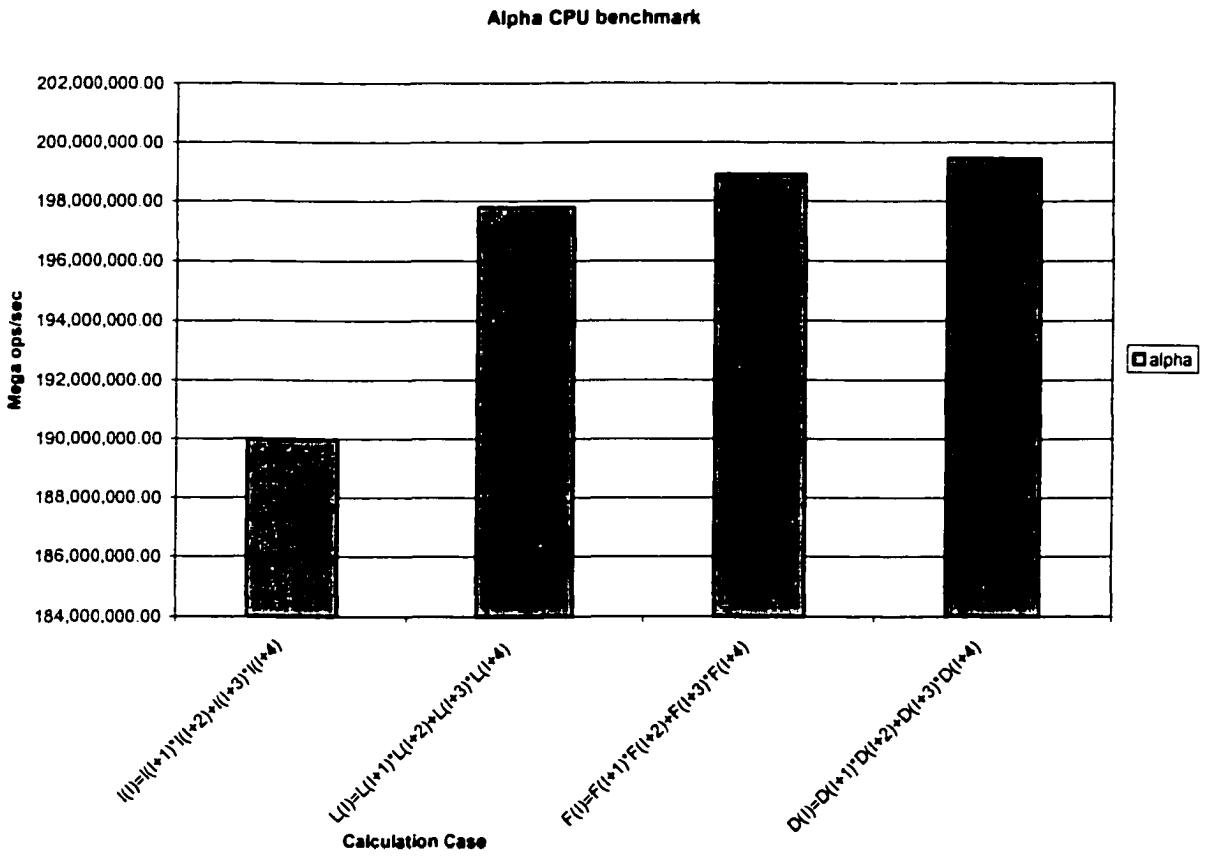
Additional cases were run on stand-alone computers to explore some of the unexpected results. Graph I shows the comparison on the Alpha processor for MFLOPS and MOPS. From Graph 1 it appears that the speed of floating point calculations seems to vary between test cases. The inclusion of the array index calculations and assignments produce a much more consistent measure of CPU performance.



Graph 1 Comparison of MFLOPS verse MOPS

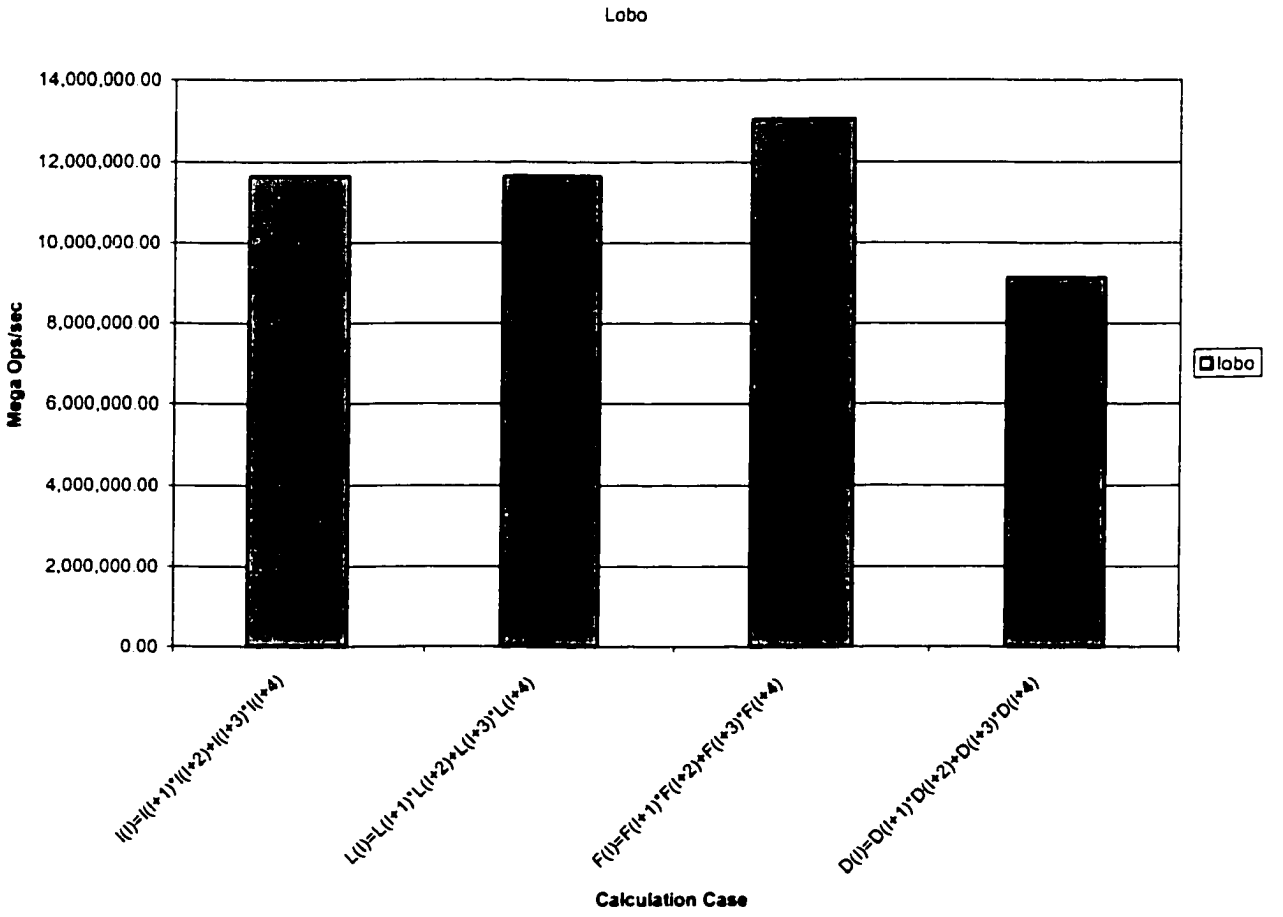
It is very clear that the use of integer variables to gain speed of execution is not a valid strategy. In fact for some of the test cases the raw data shows that the floating-point calculations are actually faster than the long variable calculations. With the current segregated pipeline architecture it is probable that the integer index calculations are using one part of the processor and the floating-point calculations another. Exploiting possible concurrency of operations actually leads to faster execution speed for a more complicated calculation.

From the data for the Alpha machine the deviation in MFLOPS is 70% and for MOPS it is only 7%. Based on this information the average MOPS is used to calculate the ratios for predicting the speed of execution of each of the CPU's in the study Performance prediction using MOPS ratios overestimated performance by a factor of 4..



Graph 2 MOPS Alpha AMD K7 Processor

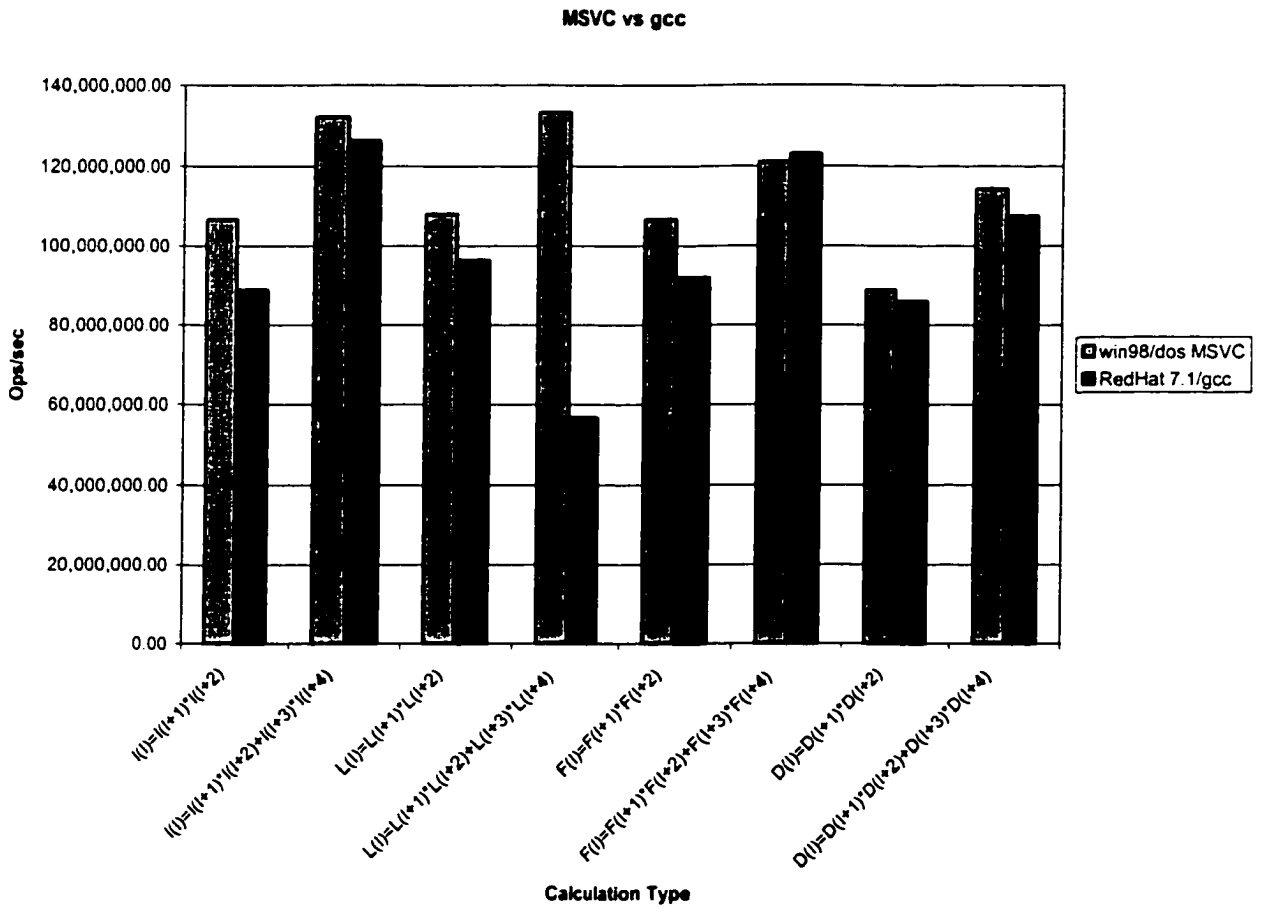
One very puzzling result is that on the Alpha AMD CPU the floating point calculation was slower than the double precision calculation. Graph 2 shows the difference in performance for the four types of data variables on the Alpha AMD CPU used in two of the Beowulf cluster configurations. The difference in performance is small, with a standard deviation of less than 4%. This is in contrast to the older CPU in both the Lobo machines and the DMC cluster, which show significant performance degradation with double precision variables.



Graph 3 MOPS Lobo/Cyrix processor

Graph 3 plots data for the older Cyrix chip and indicates that the performance of floating point calculations is faster than long or integer calculations. In order to rule out compiler effects a test program

was run on a dual boot machine, comparing the gcc and Microsoft compilers. The Microsoft compiler doesn't behave entirely as expected. Long data types have a better performance than integers and are not significantly different from floats or doubles. Graph 4 shows the results of the simple benchmark on the dual boot computer comparing gcc vs. the Microsoft compiler. This graph does indicate that the gcc compiler has a problem with long variables when case 3 calculation is done. For almost all of the cases the Microsoft compiler produced a faster program than the gcc compiler with the exception of the multiple floating point case.



Graph 4 Comparison of Microsoft and gcc compilers

All of these graphs and data only emphasize how difficult it is to produce a meaningful metric predicting the performance of programs across computers. The summary chart of execution adjustment ratios is shown in Table 6. The last three columns are Spec95 benchmark data for floating point

calculations, integer calculations and the predicted performance ratio for the floating-point benchmark. There is an approximate fourfold difference between the Spec95 benchmark and the simple benchmark. The slowest CPU is chosen as unity and values for the other machines are calculated relative to it.

Table 6 Comparison of CPU benchmarks

Machine	MOPS/sec	MOPS Factor	FP95	INT95	FP factor
Wilk Pentium III	278,160,394.80	28.35977	29.9	38.4	7.12
Alpha AMD K7	184,954,383.19	18.85697	27.4	36.9	6.5
Lobo Cyrix	10,296,034.31	1.049729	3.4*	2.4*	.7*
dmc_m Pentium	13,877,661.68	1.414893	4.82	3.4	1.15
dmc_c1 Pentium	9,808,275.25	1	4.2	3.08	1

* Estimated benchmarks for the Cyrix processor based on single processor performance for application program.

4.1.2 Communication benchmarks

There is no question that communications for parallel implementations is a key factor in performance. However, some application programs vary significantly in sensitivity to the various characterization parameters used in the model.[CULLER97] Therefore, communications benchmarks may be the most critical piece of information needed to predict the performance of the LBM program. Based on reasoning in the current literature the LogPG model has been shown to accurately predict execution time for a wide variety of application programs. In some experiments a small increase in the latency caused a five fold slow down. Table 7 lists the definition for each of the components in the model.

Table 7 Terms for communication characteristics.

L: the latency is the delay in communication of a short message from one processor to another.

o: overhead is defined as the amount of time that the processor is directly involved in preparing and sending a message and is unavailable for any other operation.

g: gap is defined as the minimum time between consecutive messages

G: time per byte for large transfers

P: number of processors

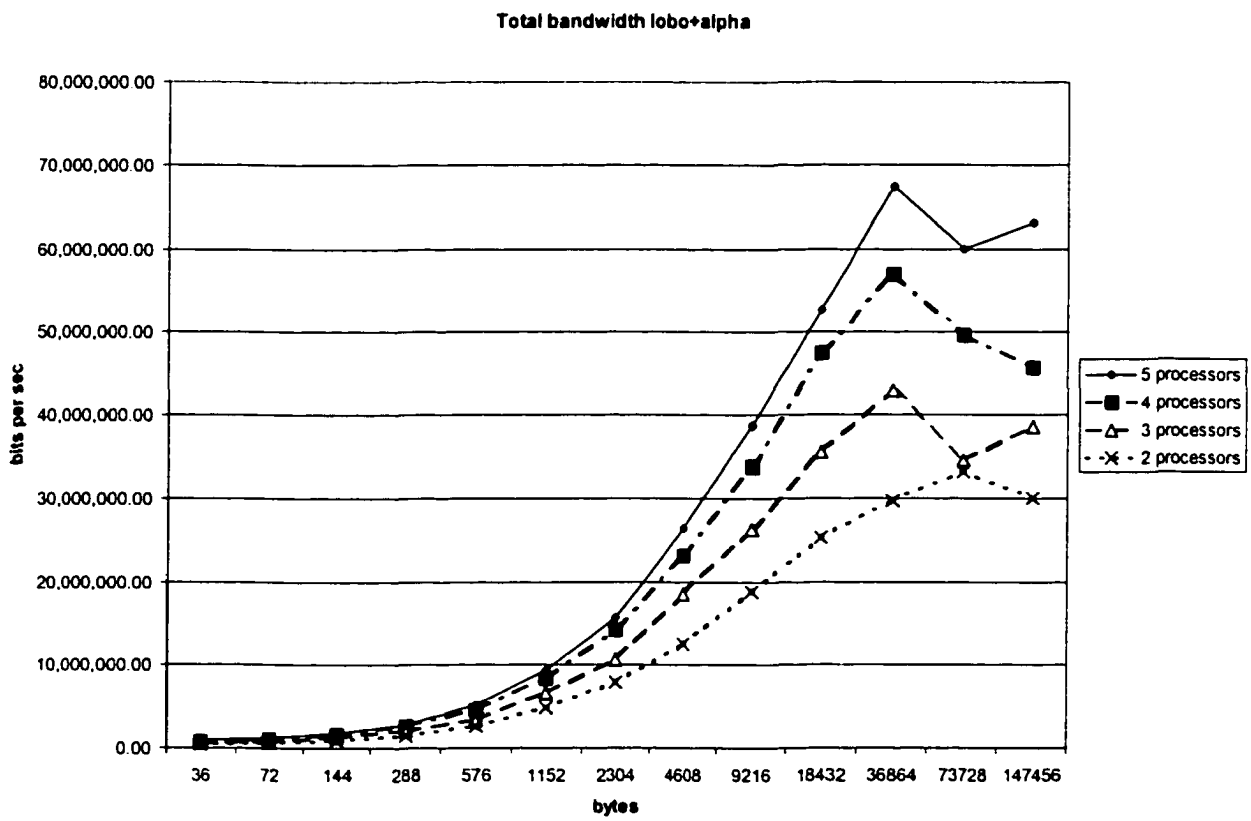
It is difficult to determine these parameters for the clusters used in the study. Based on that fact and the low expected sensitivity to communication latency a simpler set of characteristics is used. No individual calculations are contingent on sharing of results. The LBM implementation studied is considered a Bulk Synchronous Parallel (BSP) application. The BSP model, while less detailed, models the LBM parallel implementation with good results. For communications in a BSP model, two communication parameters must be provided, time per message and the time per byte. These parameters lump together the performance of the communication medium and the CPU's impact on communications.

Even though the CPU's computational power has an impact on the communications benchmark the needed performance parameters can be extracted. Both time per message and time per byte have components that are a function of processor and media speed. The media speed is a function of network loading and hardware characteristics. [RANWAWAKE97] [GROPP99] For example, changing the 10 Mb hub on the DMC cluster to a 100Mbit switching device improved performance by 30%.

The communications benchmark commonly used consists of two processors communicating in a rapid-fire succession of messages. This is sometimes referred to as the Ping-Pong style of measurement. [DONALDSON99] By specifying a single type of message the communication characteristics can be extracted from the time information. The biggest issue with this type of test is that it does not show the effects of communications patterns and the benefit of switches on the overall system. But it does allow for a relative measure of the impact of different types of messages. In a real parallel environment the multiple processors in the Lattice-Boltzmann Method communicate in many directions within a short duration of time. Starting with information from the standard benchmarks and following the style used for the CPU benchmarks, a small benchmark was created to model the communications pattern of the proposed Lattice-Boltzmann Method implementation. This benchmark focuses on the use of non-blocking send and receive style of messages. The benchmark also needs to model communications with adjacent processors and synchronization at key points. Because the model is BSP the slowest processor in the parallel system dictates overall performance and hence it is possible to relate CPU benchmark to some of the communication characteristics. It is also noted that actual system communication rates may exceed individual network card rates because of the concurrent communication ability provided by switches.

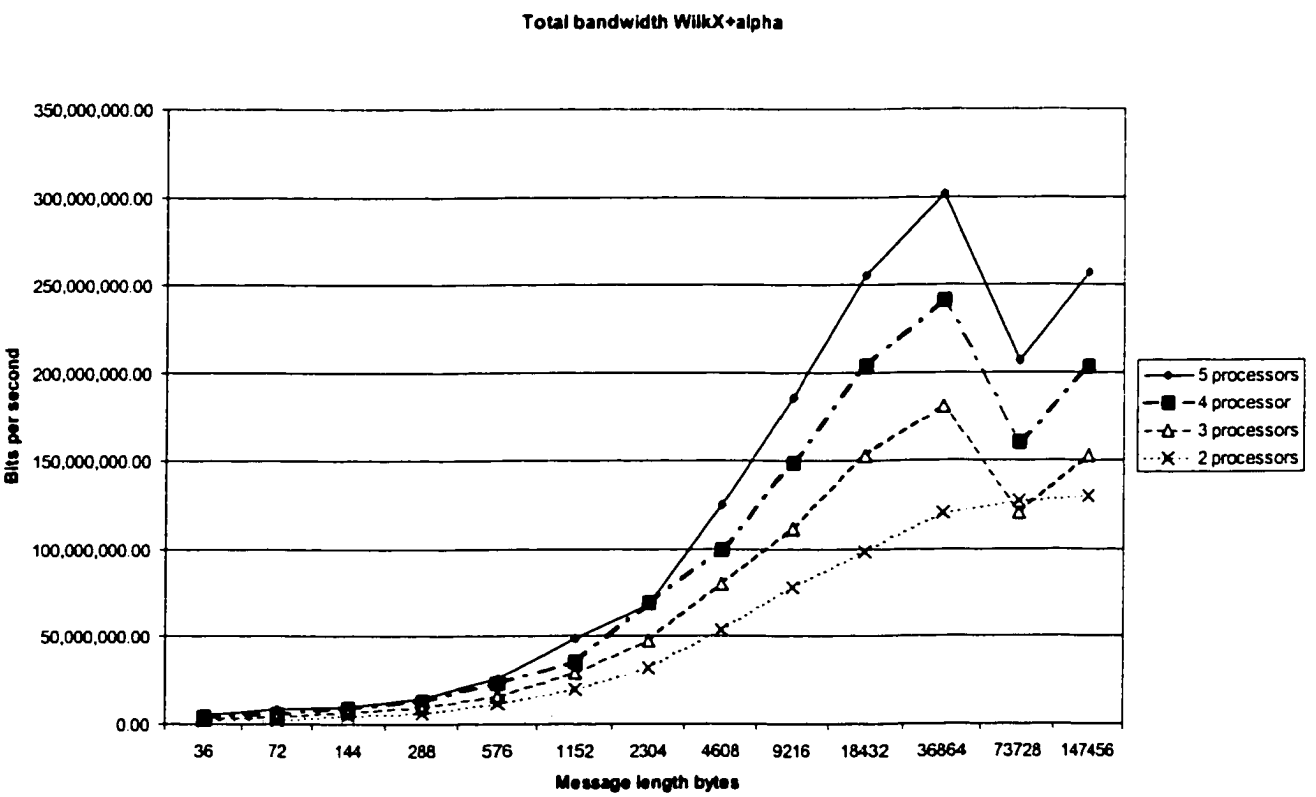
4.1.3 Format of the Communications Benchmark

All of the cluster configurations use MPI as the primary communications interface to achieve the parallel computing solution to the Lattice-Boltzmann Method. Three sets of systems are included in the performance evaluations. The first is the GVSU system consisting of Lobo processors and the Alpha main node. The second is the GVSU system consisting of the Wilk processors with Alpha again as the main node. The final set is the DMC personal cluster assembled from scrap computers. Based on the analysis of communication requirements for LBM a mixture of very short messages and long messages are used in the benchmark to meet the profile of the parallel program. The messages sharing the lattice perimeter data are fairly long compared to the short messages containing information about particle forces and motion. Many of the particle messages may even be empty because the probability of a particle spanning two spaces is relatively small.



Graph 5 Communications benchmark Lobo & Alpha

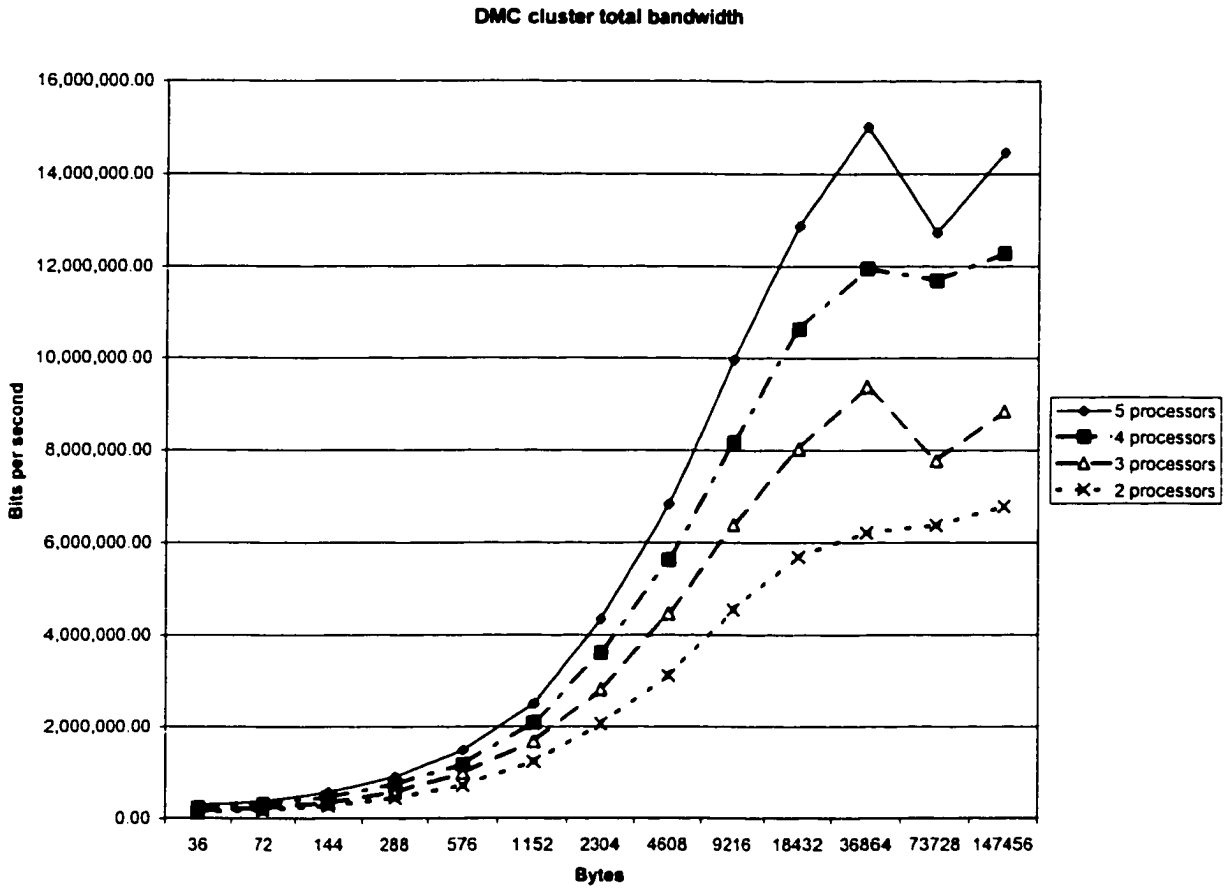
The data format of the benchmark test consists of 2 long messages and 4 short messages together with a synchronizing message. The length of the long messages varies. Each message, regardless of length, imposes a measurable time burden on the processor. The benchmark is designed to test an arrangement of lattice spaces in which the problem is partitioned into slices. The smallest test uses two processors and the largest tested on these clusters is 5 processors. Graphs 5, 6, and 7 show the results for each cluster. As expected, the addition of each processor produces a marked increase in the total bandwidth of the system. The impact of message length on bandwidth is significant for very long messages. Message length is in uncommon increments because the model has 9 floating-point values to transmit for every lattice point. Therefore, adding columns or rows to the system will cause the number of bytes to increase by this atypical amount.



Graph 6 Communication bandwidth Wilk & Alpha

The Lobo-Alpha combination has a significantly lower bandwidth as shown in the Graph 5 than the Wilk-Alpha combination in Graph 6. The Lobo processors have 100Mbit network cards and use the

same switch but processor performance affects overall communications. The DMC cluster combination has an even more dramatic degradation in performance, but this is due to the use of 10 Mbit network cards.



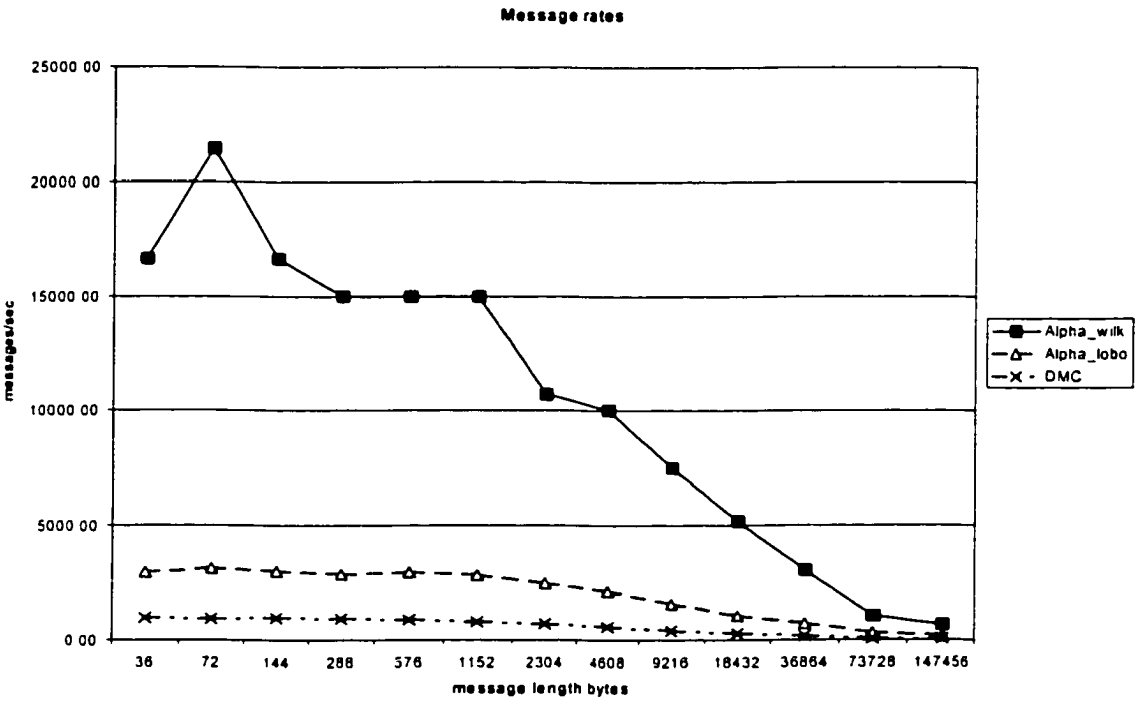
Graph 7 DMC cluster bandwidth

It is possible to extract key characteristics from the MPI performance benchmarks, notably, the time for messages and the transmission speed per byte on each system configuration. This corresponds to latency based on message per second and bulk transfer bandwidth. Table 8 shows the extracted characteristics and the idealized data for the network medium. The characteristics show a significant dependency on processor performance for communication performance.

Table 8 Communication parameters

	Time per message usec	Time per byte Usec	Bandwidth Mbit/sec	Media time per byte usec	Media Mbit/sec
Wilk & Alpha	294.94	0.132	60.30	.08	100.00
Lobo & Alpha	1678.75	0.592	13.49	.08	100.00
DMC	5389.85	2.664	3.00	.80	10.00

The fact that the Wilk CPU's are faster helps reduce congestion by dispersion of communication messages in time. As usual the slowest CPU determines the overall speed of the cluster for applications requiring synchronization. One of the difficulties in extracting communication characteristics is that network delays, operating system delays and processor burden are all mixed together. What is known is the maximum communication rate for the media used; in the Wilk & Alpha cluster and in the Lobo & Alpha cluster the media is 100 Mbit, in the DMC cluster it is only 10 Mbit. In all cases the bandwidth is less than the theoretical limit of media speed, even for long messages, indicating blocking on the network or that the overhead associated with the processor is the actual bottleneck. Graph 8 shows the message rate as a function of message length in bytes.



Graph 8 Message rate as a function of length

As seen in the Graph 8 all three systems show an initial increase in message rate as the message length increases, then the expected decrease in rates for longer messages. For the Alph-Wilk combination the initial increase is very pronounced. One hypothesis for this increase is based on the timing of task switching as it applies to the communication handlers and application program. In several other studies of communications on clusters constructed from common PCs, similar results were obtained.

4.1.4 Single processor benchmark

The basis for performance prediction of a parallel implementation of the Lattice-Boltzmann Method is the single processor version of the program. The implementation is carefully designed to permit individual portions of the program to be disabled. Using this method each part of the program can be characterized in units appropriate to the size metrics of the model, for example, the size of the space in lattice points per row and column. Another metric is the number of solid particles and their size. These metrics are a key element in characterizing the performance of the program when running on different cluster configurations.

It is possible to measure the metrics for each part of the program. The following Table 9 shows the main metrics based on the slowest processor in the DMC cluster. The units are based on the parameters used to predict the performance on various processors and communication configurations.

Table 9 Single processor benchmark data

Activity		units
EDF	60635.46	Lattice points/sec
Make crossing list	41129.03	square units/sec
Calculate crossing	111290.3	crossing/sec
Stream flows	222222.2	Lattice points/sec
Update pos	2419.5	particle/sec

Two methods are used to extract the characterization from the test cases. The selective disabling of main routines permits measurement of the each part of the program and the quantity of units for each test. The second method uses a partial increase of each quantity. This is used to confirm the values calculated from data in the selective disable test.

4.2 Performance Prediction

The base data required for predicting performance of an application on a particular system is CPU computational power, communication latencies, and program characterization. The original benchmark for CPU computational power is based on array calculations for the various types of data variables. In a simple test of the benchmark two areas of concern were identified. Performance predictions of a stand-alone program based on the simple CPU benchmark yield execution times that are off by a factor of 2 or more. Therefore, the simple CPU benchmark was not used in the performance prediction. Instead the CPU power benchmark based on the SpecFP95 published data for the various CPU was used. The data extracted from the communications benchmark is used along with the single program characterization parameters.

Predictions are constructed in two steps. Step one requires all size parameters such as rows, columns, particle count, particle size and processor arrangement to estimate the CPU total time for the slowest processor in the study. This time is adjusted based on the CPU power benchmark so that time per pass can be calculated. The second step requires an estimate of the CPU time used to handle communications as based on the communication benchmark data. The two times are combined producing the final prediction of performance. In Table 10 the various predictions are summarized based on different configurations of lattice columns. The space was divided into slices with the processing partitioned among

5 processing units. The slowest of the 5 processors is used to predict performance. The prediction is in seconds for 1000 iterations of the program's main loop. All times and predictions ignore startup procedure cost and focus on the computational aspects of the program including communications. The structure of the main loop in the program does not take advantage of any possible concurrent operations. This will gives a better estimate of worst case performance than a program structured for concurrent processing and communications. The ratio of CPU computational power used to predict performance on the Wilk & Alpha cluster is based on the SpecFP95 benchmark. The normalized CPU factor for the slowest DMC machine in the cluster verses the Alpha machine in the Wilk & Alpha cluster is 6.5. This factor is used to adjust the computational times for the prediction model.

Table 10 Predicted run times for 1000 passes

Total space	100 by 500	200 by 500	300 by 500	400 by 500	500 by 500
DMC	303.0	536.3	769.5	1002.0	1139.2
Wilk & Alpha	40.9	74.8	108.6	142.5	176.7

It is also possible to predict the single processor solution time and the multiple processor Beowulf cluster solution times. These two times can be used to predict the speed up expected by using multiple processors. Ideally the speed up for 5 processors is 5. However, because of communication burden and extra calculations due to duplicate lattice rows and columns the speed up is less than linear. In many applications the speedup achieved is much less than the number of processors. For example, in some cases for 16 processors a speed up of only 4 is achieved. In general, better speed up can be achieved by reducing the ratio of communication time to calculation time. Since slower processors have a higher ratio of computation time to communication time, they actually have better speed up factors. [BARTON89]

In the parallel LBM application, minimizing communication leads to a better speedup, as does increasing the computational workload. For example, calculations on very large spaces will reduce the impact of communications as long as communication is based on the perimeter of the space. Table 11 shows the predicted speed up for the two clusters used in the study. The predicted speedup for 5 processors as a function of lattice space size shows expected improvements in speedup for larger workload per processor.

Table 11 Predicted speedup 5 processors

Total space	100 X 500	200 X 500	300 X 500	400 X 500	500 X 500
DMC	3.14	3.78	4.08	4.25	4.36
Wilk & Alpha	3.35	3.94	4.21	4.36	4.45

For configurations with 2 rows by 2 columns of processors the predicted speedup on a 500 X 500 lattice is 2.97 vs. 3.63 for a 4 rows by 1 column arrangement. The use of slices is clearly predicted to be a better data partitioning strategy. This allocation method also reduces the probability of particles spanning multiple spaces.

The impact of particle communication is predicted to be small with this implementation. The average communication packet for particles is expected to be empty, given a sparse density of particles. The probability of a particle spanning multiple processors is relatively small based on the area of the particle compared to the area of the workspace. The worst case allocation of all particles in the slowest processor is used in the prediction. Two examples of the calculations used in the prediction are shown in the following Tables 12 and 13. These tables show the predictions for 500 X 500 total lattice points divided among 5 processors in slices.

Table 12 Wilk & Alpha 500 X 500 with 5 processors

Process time			Quantity	Processor Time/pass	Unit
EDF per lattice	60635	Lattice points/sec	51000	0.1294	Sec
Streaming per lattice	222222	Lattice points/sec	51000	0.0353	Sec
Update xcrossing list per particle	41129	Area/sec	173.4	0.0006	Sec
Calc xcrossing force and flow	111290	Crossing/sec	702	0.0009	Sec
Update particle pos	2419	Particles/sec	3	0.0000	Sec
Communicate lattice	0.00033404	Sec/lattice	2	0.0007	Sec
Communicate particle forces	0.00023916	Sec/particle	2	0.0005	Sec
Communicate particle position	0.00024639	Sec/particle	2	0.0005	Sec
			Total	0.1679	Sec

Table 13 DMC cluster 500 X 500 with 5 processors

Process time			Quantity	Processor Time/pass	Unit
EDF per lattice	60635	Lattice points/sec	51000	0.8411	Sec
Streaming per lattice	222222.2	Lattice points/sec	51000	0.2295	Sec
Update xcrossing list per particle	41129	Area/sec	867	0.0211	Sec
Calc xcrossing force and flow	111290	Crossing/sec	1110	0.0099	Sec
Update particle pos	2419	Particles/sec	3	0.0012	Sec
Communicate lattice	0.0073042	Sec/lattice	2	0.0146	Sec
Communicate particle forces	0.00540496	Sec/particle	2	0.0108	Sec
Communicate particle position	0.00553796	Sec/particle	2	0.0111	Sec
			Total	1.1393	Sec

4.3 Measured performance

The actual performance tests were conducted on two Beowulf clusters; the Wilk & Alpha system and the DMC system. Five processors were used and the number of columns was varied to measure the effect of increasing the ratio of computation to communication. Additional tests were conducted on the Alpha & Lobo cluster but only the number of processors was varied. Table 14 shows the effect on solution time for 5 processors when the space ranges from 100 by 500 lattice points to 500 by 500 lattice points.

Table 14 Space size vs. Execution time

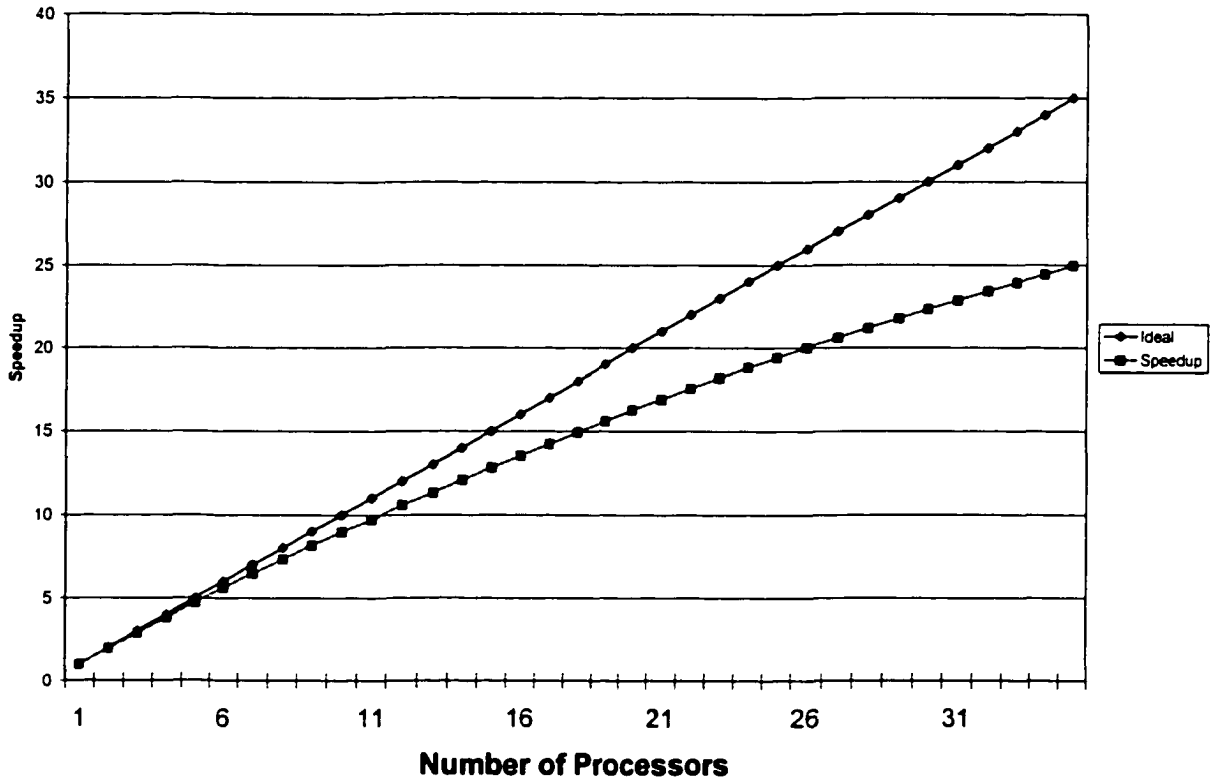
Total space	100 X 500	200 X 500	300 X 500	400 X 500	500 X 500
DMC	282 sec	530 sec	773 sec	1019 sec	1261 sec
Wilk & Alpha	38 sec	70 sec	100 sec	134 sec	166 sec

The execution time for the Alpha & Lobo combination was unexpected based on the CPU benchmark and the communications benchmark. Table 15 shows the test conducted with a fixed space size per processor. The execution times for the Lobo & Alpha systems was expected to be faster than the DMC cluster. However, the DMC cluster was noticeably faster even with 10 Mbit per second network cards. The lobo machines have a Cyrix processor and this may account for the difference in performance. A single processor version of the program was executed on the Lobo CPU and verified a significantly slower CPU computational speed. For the current model to accurately predict the performance of the Cyrix processor it needs an estimated CPU performance factor of 0.7.

Another series of test were conducted on each system to identify any significant network loading. In these test each processor had 100 by 100 lattice points. As the number of processors was increased from 2 to 5 no significant increase in solution times is observed as shown in Table 15. This is due to the network switch used in all cluster configurations, which allows for a significant amount of concurrent network traffic.

Table 15 Number of Processors vs. Execution Time

Number processor	2	3	4	5
DMC	278 sec	276 sec	280 sec	282 sec
Alpha & Lobo	367 sec	368 sec	368 sec	369 sec
Wilk & Alpha	37 sec	38 sec	38 sec	38 sec



Graph 9 Speedup Wilks & Alpha

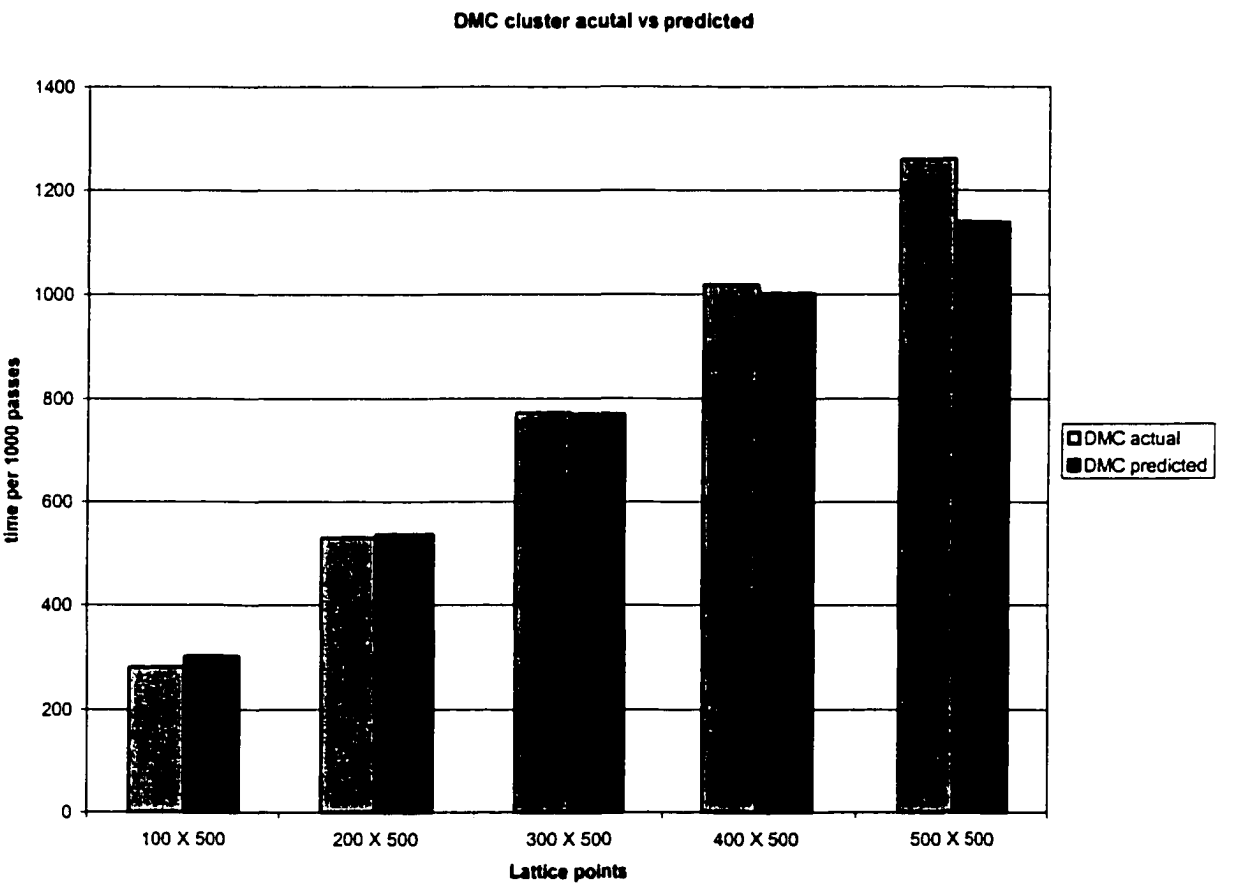
Graph 9 shows speedup vs. number of processors. The speedup graph shows good efficiency as the number of processors is increased. The limiting factor is workload verses communication time. The Lattice-Boltzmann Method is robust with division of workload as supported by Graph 9. The experimental clusters confirmed the speedup for small number of processors.

5.0 ANALYSIS

Analysis will be divided into two areas. First, the comparison of predicted and measured performance will be examined. This will show the accuracy of prediction model and validation of the performance trends. The second part of the analysis will be a theoretical analysis of process space saturation by particles and the worst-case speedup prediction expected. This analysis supports some of the observed problems when LBM simulations that have increased quantities of particles that sediment in one processor.

5.1 Comparison of Prediction and Results

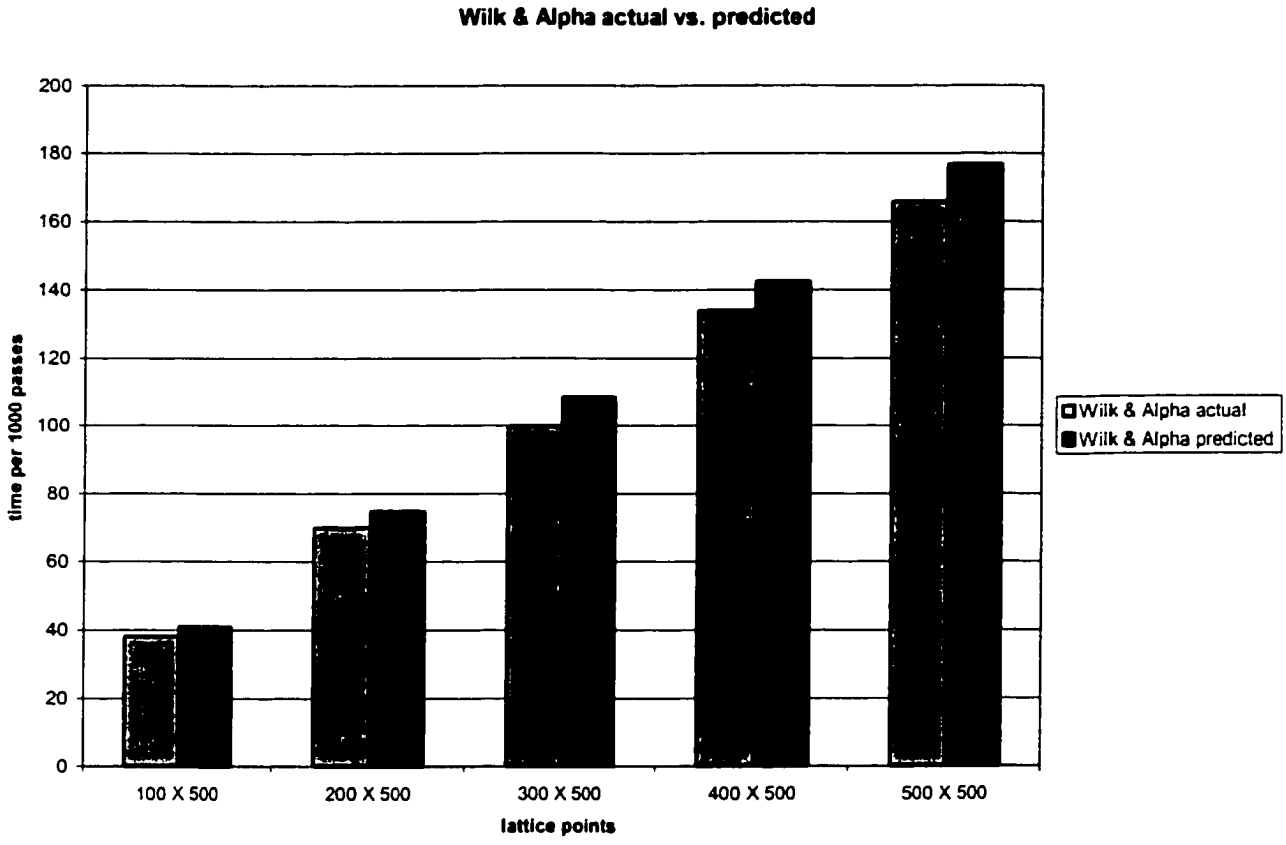
The first significant observation is that the simple CPU benchmark cannot be used to predict the general operation of the Lattice-Boltzmann Method. Because of the inaccuracies of the simple benchmark, published benchmark for Spec95fp is used to provide a better indicator of CPU performance.



Graph 10 DMC cluster actual vs. predicted

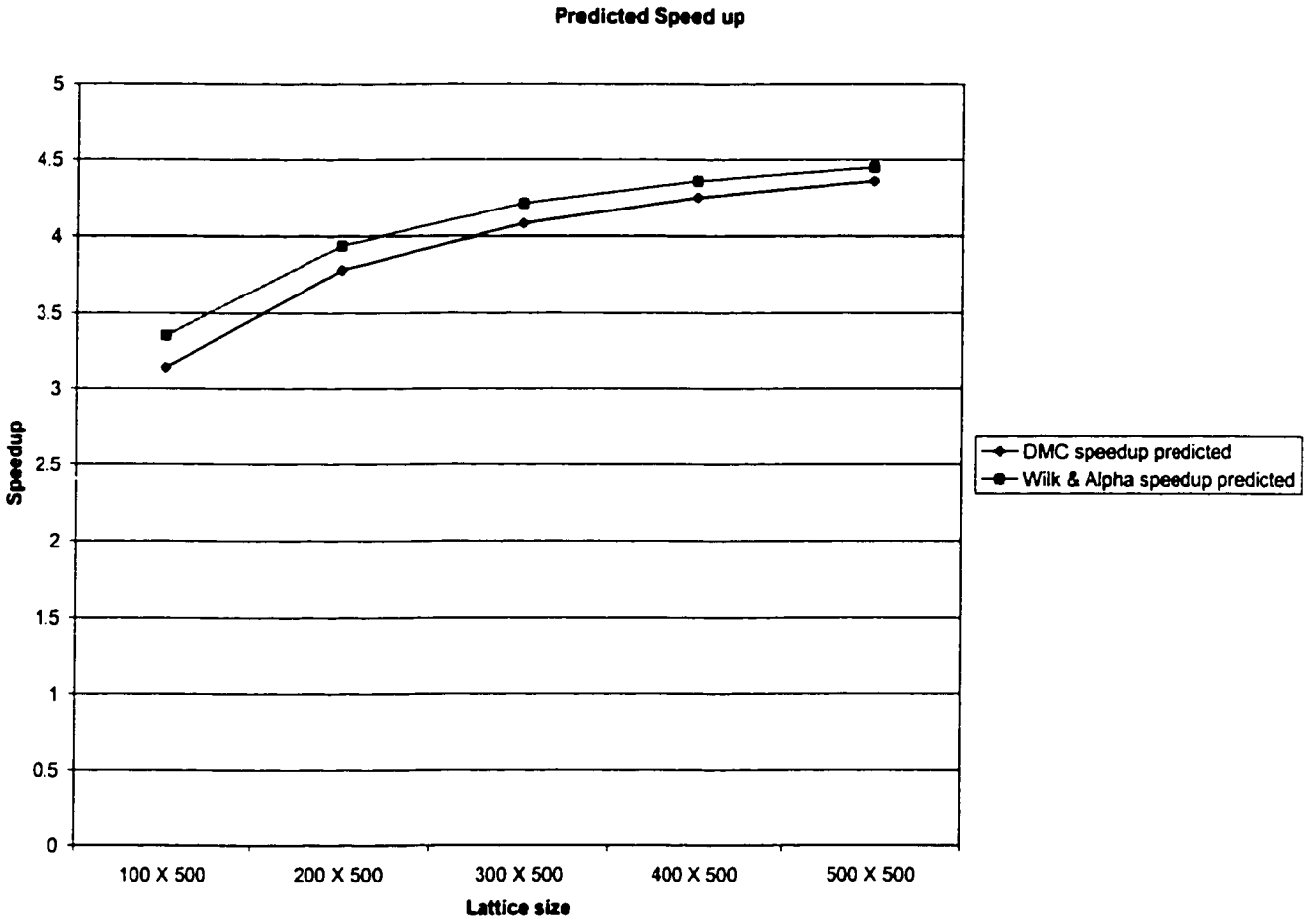
In general, cluster performance on the DMC is predicted very closely for various sizes. Graph 10 shows the comparison of actual performance to actual measurements. There is a trend evident that for small lattice spaces the predicted time is slightly longer and for large spaces the predicted time is slightly smaller. This implies that either the per byte overhead for communications is greater than measured or that the single processor characterization understated the metric for each lattice point. Graph 11 shows that the Wilk & Alpha cluster predictions are very close to the actual performance but consistently longer than the actual times. This implies that the characterization of the time per lattice point is a little low for this processor or possibly the CPU factor is slightly off. The fact that the predictions are within 10 % on all cases is very encouraging. The average error for all cases predicted vs. actual time is less than 6 %.

The actual speedup measurements for test cases is also within 5 % of the predictions and are close to the number of processors used in the test cases. Actual speed up for the Wilk & Alpha cluster was 4.71 verses 4.45 for the prediction. It is interesting to note that the speedup attained for the DMC cluster and the Wilk & Alpha cluster is about the same. This is due to the corresponding relationship between processor power and communication bandwidth. The ratio of communication time to processing time is an underlying factor in the calculation of speedup. [HWANG98] For the DMC cluster the ratio is 31 to 1 and for the Wilk & Alpha cluster the ratio is 85 to 1. This higher ratio of processing time to communication time supports the slightly better speed up.



Graph 11 Wilk & Alpha cluster actual vs. prediction

From the measured results and the benchmark data collected, it can be seen that processor computational power plays a significant role in the communication burden. The effect of media speed did not affect the execution speed except when near saturation. Many other applications are far more sensitive to the media delays than this application.



Graph 12 Predicted speedup

5.2 Theoretical analysis

The goal of the theoretical analysis is to set the expected limits on the solution time of the Lattice Boltzmann Method. From this analysis several characteristics of program operation should be clear. The operation can be modeled by several key quantities. In this case the dimensions of the test space, number of particles, size of the particles and the number of processors are used. For each type of activity a coefficient with the appropriate units is assigned to calculate the time. Figure 8 shows the dimensions for a typical layout of a test space. Dimension w is the width in lattice points, l the total length in lattice points and N is the number of slices distributed among processors. The two vertical lines on the right and left are the moving wall boundaries.

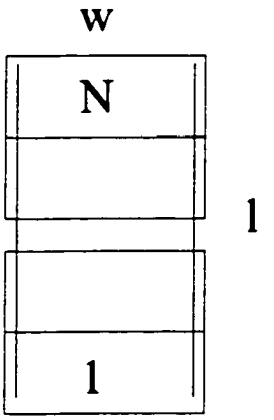


Figure 8 Slice layout

Equation 37 describes the solution time as a function of the dimension plus the number of particles in the test space. The coefficient A is the time factor per lattice point for streaming and the EDF step. Particles have three terms: B is the time to make the list of boundary crossings that is a function of the radius of the particle squared, C is the time per solving boundary crossings and D is the time to update position information after each pass. The coefficient E is the time to evaluate the flows that cross the wall boundary in the model. The value of E is less than C because the evaluation does not require collision checking because of the order of execution. This order is determined by the wall and particle list used in the set up file. Requiring that the walls be listed first will ensures the order of execution during simulation.

$$T_1 = A(l * w) + k(BR^2 + CR + D) + E(l)$$

l = length of space in lattice points

w = width of space in lattice points

R = radius of particle

k = number of particles

Equation 37 Time of single processor execution

All of the coefficients B, R, C, and D are fixed and can be combined into a single coefficient B'. The coefficient is only a function of the size of particle. Note that we assume uniform particle size for ease of analysis. For purposes of analysis the walls are assumed to be vertically oriented on the right and the left extreme edges of the experiment space.

$$B' = BR^2 + CR + D)$$
$$T_1 = A(l * w) + kB' + E(l)$$

Equation 38 Reduce equation execution time

The single processor solution time is given in Equation 38. Using this as the basis another equation for time of execution of the longest slice of a parallel solution is derived. The difficulty with this derivation lies in the distribution of number of particles in the problem. Because one of the experimental problems is scaling, special attention is given to worst case scenarios for the solid particles. The Figure 9 is a graphical representation of the probability that a particle can be in more than one space. The probability that a randomly placed particle has a center in the border area is the ratio of the border area divided by the total area. The boarder area is defined as one radius of the particle from the edge. The boundaries on the left and the right are removed for this case with slices only reducing the probability further. The bottom edge of the figure shows the worst case placement that maximizes the number of particle in more than one space. The actual particle count expected to be in more than one space is between these two limits.

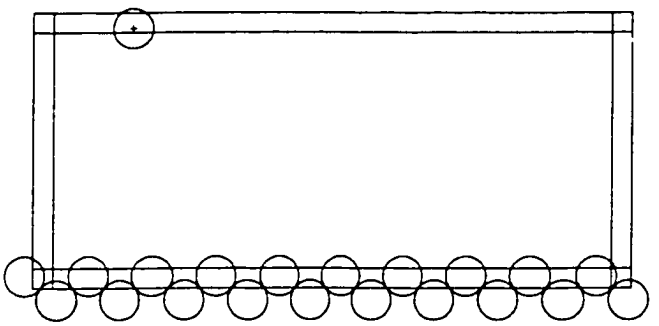


Figure 9 Probability of particle placement

The communication model for this analysis is straightforward. Each message is modeled as having a fixed amount of overhead time, processor burden is based on the total bytes transferred. Variations in network loading are not considered in this analysis. The current implementation uses 2 messages to transmit the lattice data along the edges of the slices, the particle force and position data is then transmitted along each edge. In this case a total of 6 messages are sent. Equation 39 describes the solution time with the division of work by N and the time of communications added.

$$T_l = A\left(w\frac{l}{N} + 2w\right) + \frac{kB'}{\gamma} + E\frac{l}{N} + 6G + H(2wb_l + \alpha b_p)$$

N = number of processors

b_l = bytes per lattice point

b_p = bytes per particle forces update and position update

γ = distribution of particles (ranges from 1 to N)

α = number of particles in more than one space

Equation 39 Longest time of execution for slice

The number of particles that span two spaces is difficult to determine precisely. The average number can be calculated using the probability a particle is near the edge times the number of particles in that space. Care must be taken to ensure the number does not exceed the maximum number that will fit along two edges. At the same time if the number of particles are distributed in some uniform manner then the number in two spaces cannot exceed the number in any given space. Equation 40 lists the three main possibilities for the distribution of particles among processor spaces.

$$\alpha = \frac{2Rw}{wl} * \frac{k}{\gamma} \text{ average probability of particles in more than one space}$$

or

$$\alpha = \frac{w}{R} \text{ worst case number spanning two spaces}$$

or

$$\alpha = \frac{k}{\gamma} \text{ total particles in space}$$

Equation 40 Particle probability

Regardless of how the particles are distributed in any space the worst case number of particles in more than one space is given by Equation 41. Equation 42 shows this term used for communication length associated with force and position updates.

$$\alpha = \frac{w}{R}$$

Equation 41 Max number of particles in two spaces

$$T_l = A\left(w\frac{l}{N} + 2w\right) + \frac{kB'}{\gamma} + E\frac{l}{N} + 6G + H(2wb_l + \frac{w}{R}b_p)$$

Equation 42 Maximum execution time per slice

The maximum number of particles in one slice can be calculated by approximating the particles as a square with side 2R. Equation 43 describes the worst case particle distribution in terms of slice area and radius of the particles. This worst case will occur if all the particles in the simulation are in one slice.

$$\frac{k_{\max}}{\gamma} = \frac{l * w}{4NR^2}$$

Equation 43 Maximum number of particles in a space

$$S = \frac{T_1}{T_l}$$

$$S = \frac{Awl + \frac{lw}{4NR^2}B' + El}{Aw\left(\frac{l}{N} + 2\right) + \frac{lw}{4NR^2}B' + E\frac{l}{N} + 6G + Hw\left(2b_l + \frac{b_p}{R}\right)}$$

Equation 44 Speedup for worst case particles distribution

The speed up is shown as the ratio of the execution time for a single processor divided by the longest time for a single slice. Equation 44 describes the speed up with the worst case distribution of particles. There are several observations that can be made concerning the speedup for the parallel Lattice Boltzmann Method. If the ratio of length to the number of slices is much larger than 2 then the basic speed up will not degrade as duplicate lattice points are added to the slices. As the ratio gets smaller it will have two effects. More particles will be in two spaces at the same time, and the implementation has a problem if the factor l/N is smaller than 2R. If the total number of particles is larger than will fit in a single slice then the worst case still shows a significant speedup. Particles are computationally expensive compared to lattice points. Both the communication terms and the particle count limit the available speedup. In the case where all the particles are in one space the speedup is limited by the relative magnitudes of the coefficients. If there are so many particles that they cannot all fit in one space then the speedup is based on the distribution of particles among the slices.

$$Aw\left(\frac{l}{N}+2\right)+\frac{lw}{4NR^2}B'+E\frac{l}{N} >> 6G+Hw\left(2b_l+\frac{b_p}{R}\right)$$

Equation 45 Releative size of terms

The empirical evidence supports this for the clusters studied. In the experiments conducted the number of particles is small compared to the worst case. This ratio gets larger for a space with a saturated number of particles. Equation 45 shows the relationship between communications and work. The ratio from experimental data for a small number of particles was on the order of 70 to 1 for the Alpha &Wilks combination. Based on this relationship the analysis of speedup can be simplified as shown in Equation 46.

$$S=\frac{Awl+\frac{lw}{4NR^2}B^{\cdot}+El}{Aw\left(\frac{l}{N}+2\right)+\frac{lw}{4NR^2}B'+E\frac{l}{N}}$$

Equation 46 Simplified expression of speedup

The relationship between the wall coefficient and the number of particles can be calculated. For the case where the total perimeter of all particles in a space is much greater than the length of the wall in a space the term with E becomes much smaller than the $B^{\cdot}$ term and can be ignored. If the number of particles is small but the number of lattice points is moderate then the time to solve is dominated by the lattice point term.

$$S=\frac{Awl+\frac{lw}{4NR^2}B^{\cdot}}{Aw\left(\frac{l}{N}+2\right)+\frac{lw}{4NR^2}B'}$$

Equation 47 Further simplified speedup

The relation ship of A to $B^{\cdot}$ from the experimental data is 1 to 374 for a particle radius R=8.5. Each particle requires the same amount of time to solve as 374 lattice points. Substitution of A and $B^{\cdot}$ with the ratio and factoring the *wl* yields a speedup as shown in Equation 48.

$$S = \frac{1 + \frac{1.3}{N}}{\frac{2.3}{N}} = \frac{N + 1.3}{2.3}$$

Equation 48 Final speedup for saturated particle case

If the distribution of particles is such that they saturate a single lattice space then the speed up approaches 1 as the number of processors increases. If the number of particles remains constant as the number of processors is increased then the model forces the particles to be spread among several processors and the speedup is much better depending on the division of particles among the processors.

For the case where the number of particles is distributed in a random or uniform fashion, the speedup can be found with the following Equation 49. Speedup is much closer to linear. At this point the cost of communications becomes more significant than in the worst case and needs to be considered. This equation resembles the standard measure of speedup combining communication cost and processor workload as shown in Equation 50.

$$S = \frac{Awl + kB'}{Aw\left(\frac{l}{N} + 2\right) + \frac{k}{N}B' + 6G + Hw\left(2b_l + \frac{b_p}{R}\right)}$$

Equation 49 Speedup with even distribution of particles

$$S = \frac{work}{\frac{work}{N} + T_{communication}}$$

Equation 50 Standard for speedup

6.0 Conclusion

The overall goal of this study was to develop a performance prediction model for parallel implementation of the Lattice Boltzmann Method on cluster computer systems. As part of this process a new implementation of the 2D LBM method was constructed as the base for analysis. This implementation introduced improved streaming of flows, list of boundary crossings, and new solid particle communications strategies. Using the new implementation as the base benchmark, data was collected to provide the means for performance prediction. A detailed spreadsheet was developed that combined the dimensional data with benchmark data to predict the time of execution. As this study demonstrates it is possible to predict performance of the LBM Method with reasonable accuracy on a range of Beowulf clusters. Using the limited benchmark data combined with the program characterization it was demonstrated that speedup and execution time meet performance predictions for the Beowulf clusters used in the study.

The analysis demonstrates the limitations on speedup as a function of solid particle distributions. Even though the Lattice-Boltzmann Method normally achieves good speedup it is possible that solid particles could be distributed among parallel processors in such a way that poor speedup occurs. This processor lattice space particle saturation can be overcome by changes in the arrangement of the processor spaces that force the particles to more evenly distribute among the processors.

The introduction of the lattice pair method for computation of particle forces and the impulse force method for collisions enable a dramatic improvement in particle communication times. Reducing the particle communication to adjacent processors represents a significant improvement over the use of the MPI_gather types of commands used in prior implementations.

6.1 Benchmarks

The simple CPU computational benchmark intended as the basis for prediction of execution speeds for the main body of the LBM program was identified as unreliable early in the study. But data collected with the simple benchmark did quantify the impact of the different precision data types on performance. It was determined that the impact of array index calculations is very significant; implementations that reduce the number of index calculations can be expected to have improved execution times. It was also determined that the use of lower precision variables on modern CPU in the Pentium and

AMD K series does not significantly improve execution speed. The data clearly indicates that the use of floating point variables yielded the best performance while providing the desired accuracy. The inability of the simple benchmark to predict overall program execution time is due to its single focus. A single loop with a repetitive calculation does not represent the typical mix of program instructions that normally includes a variety of branching constructs. Based on this limitation the well-recognized Spec95fp benchmark is used to provide the necessary accuracy used in the performance prediction.

The simple communications benchmark was designed to identify any significant problems with the communication patterns used in the Lattice-Boltzmann Method as applied to parallel programs executed on Beowulf clusters. The simple communications benchmark provides the necessary performance data to accurately predict performance when combined with the CPU computational power benchmark. Communication benchmark data showed that for standard network cards the amount of processor overhead for messaging was the major bottleneck in the system. The effect of using a switch was evident in that system bandwidth exceeds the media's maximum bandwidth. Switches in effect provide concurrent communications and increase the threshold of network loading degradation.

6.2 Summary

This study implemented the 2D parallel Lattice-Boltzmann Method with efficient, new strategies that support good speedup performance on Beowulf clusters constructed from standard processor and network components. The use of common CPU benchmarks, simple communications benchmarks and program characterization is sufficient to accurately predict program performance.

Bibliography

[BARTON89]
M.L. Barton and G.R. Withers, "Computing performance as a function of the speed, quantity and cost of the processors", ACM 1998 089701-341-8-/89/001/0759

[CULLER97]
David E. Culler, Richard P. Marin, Amin M. Vahdat, and Thomas E. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture", ISCA 1997 Denver, CO, USA ACM 0-89791-901 7/97/0006

[CULLER98]
David E. Culler, "High Performance Clusters part 1: Performance" PODC/SPAA June (1998)

[CUNHA2001]
Rudnei Dias Da Cunha and Alvaro Luiz de Bortoli,"A parallel Navier-Stokes solver for the rotating flow problem", Concurrency and computation: Practice and Experience, 2001; 13:163-180

[DESPLAT99]
J. C. Desplat and M.E. Cates, "Ludwig- a genral purpose Lattice-Boltzmann code on the Cray T3E" Fifth European SGI/Gray MPP Workshop, Bologna Italy, (1999) Sept.

[DIMITROV99]
Rossen Dimitrov and Anthony Skjellum, "Impact of Latency on Applications' Performance" (1999)

[DONALDSON99]
Stephen R. Donaldson, Jonathan M. D. Hill , and David B. Skillicorn, "Predictable communication on unpredictable networks: implementing BSP over TCP/IP and UDP/IP", (1999) Concurrency: Practice and Experience, vol. 11(11), pp. 687-700

[FAHRINGER2001]
T. Fahringer, P. Blaha, A. Hossinger, J. Luitz, E. Mehofer, H. Moritsch and B. Scholz, "Development and performance analysis of real-world applications for distributed and parallel architectures", Concurrency and Computation: Practice and Experience, (2001) 13:84-868

[GROPP]
William Gropp and Ewing Lusk, "Reproducible Measurements of MPI performance Characteristics"

[HWANG98]
Kai Hwang, Scalable Parallel Computing Technology, Architecture, Programming, New York: WGB/McGraw-Hill (1998)

[LADD94]
Anthony J.C. Ladd, " Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation" J. Fluid Mech (1994), vol. 271, pp. 285-309
Part 2. Numerical results" J. Fluid Mech (1994), vol. 271, pp. 311-339

[MERIAM71]
J.L. Meriam, Dynamics. New York: John Wiley & Sons, Inc. 1971

[MEI99]
Renwei Mei, Li-Shi Luo, and Wei Shyy, "An Accurate Curved Boundary Treatment in the lattice Boltzmann Method", Journal of Computational Physics (1999) 155, 307-330

[PACHECO97]
P.S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1997, ISBN 1-55860-339-5.

[HENNESSY90]
David A. Patterson and John L. Hennessy, Computer Architecture A Quantitative Approach
San Francisco, California: Morgan Kaufmann Publishers, Inc. (1990)

[QI99]
Dewei Qi, "Lattice-Boltzmann simulations of particles in non-zero-Reynolds-number flows"
J. Fluid Mech (1999), vol. 385 pp. 41-62

[RANWAWAKE97]
Udaya A. Ranawake, "Performance Evaluation of Piecewise Parabolic Method (PPM) on the Hive"
http://webserv.gsfc.nasa.gov/neumann/ppm_hive/ppm_hinve.html (1997) Oct

[RESCHKE96]
Chance Reschke, Thomas Sterling Donald J. Becker, Michael R. Berry "Achieving a Balanced Low-Cost Architecture for Mass Storage Management through Multiple Fast Ethernet Channels on the Beowulf Parallel Workstation" www.beowulf.org/papaers/IPPS96/ipps96.html (1996)

[SKORDOS95]
Panayotis A. Skordos, "Parallel Simulation of Subsonic Fluid Dynamics on a Cluster of Workstation", Proceeding of Fourth IEEE International Symposium on High Performance Distributed Computing, Washington D.C., USA (1995)

[SNIR98]
Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra, ``MPI - The Complete Reference, Volume 1, The MPI-1 Core, Second Edition",
MIT Press, September 1998, ISDN 0-262-69215-5.

[SPECFP95]
www.dl.ac.uk/CFS/docs/doc2000/beowulf.htm
www.hydrosoft.com/spec95.htm

[SUMIMOTO99]
Shinji Sumimoto, Hiroshi Tezuka, Atsushi Hori, Hiroshi Harada, Toshiyuki Takahashi, and Yutaka Ishikawa, "The Design and Evaluation of High Performance Communications using a Gigabit Ethernet", ICS 1999 Rhodes Greece ACM 1999 1-58113-164-x/99/06

[WOLFE02]
Gregory Wolffe, John Oleskiewicz, David Cherba, and Dewei Qi "Parallelizing Solid Particles in Lattice-Boltzmann Fluid Dynamics", Int. Conference on Parallel and Distributeded Processing, Techniques, and Applications, Las Vegas, 2002

[WOLFRAM83]
S. Wolfram, "Statistical mechanics of cellular automata", Rev Mod. Phs., 55, 601-644 (1983)