Modeling liquid crystal structures on an SMP workstation cluster*

Paul A. Farrell† Hong Ong‡ and Arden Ruttan§
Department of Mathematics and Computer Science,
Kent State University, Kent, OH 44242.
† farrell@mcs.kent.edu ‡ hong@mcs.kent.edu § ruttan@mcs.kent.edu.

Abstract

We examine the scalability and performance of a legacy liquid crystal code on a PC (Beowulf) cluster consisting of 16 dual-processor Pentium III/450s. This code was originally designed for use on a Unix workstation cluster of less than 8 machines. In particular, we examine the effectiveness of using potentially more efficient techniques such as non-blocking communication calls and whether the use of SMP nodes enhances or detracts from the overall performance.

1 Introduction

The recent availability of comparatively inexpensive Beowulf clusters has made it possible for computational scientists to run large scale computational codes at their local institutions rather than at a remote supercomputer center. The availability of such resources also facilitate the running of codes designed for multi-processor Unix workstations, servers and small clusters of networked workstations (NOWs). Beowulf clusters have the further advantage compared with NOWs that they tend to be dedicated resources, as opposed to NOWs which often were also used as general purpose access machines. In this paper, we wish to examine how well a legacy liquid crystal code performs on a medium size Beowulf Cluster, and to what extent performance improvements result from more sophisticated handling of the communications. In particular, we examine scalability of the code for numbers of processors up to 32. An additional question considered is whether or not it is cost-effective to employ 2 processor SMP nodes in such a Beowulf cluster. Popular "folk-lore" and some anecdotal evidence suggest that bus or network interface card (NIC) contention may lead to considerable inefficiencies and thus lack of scalability. This is of some importance in purchasing decisions for Beowulf clusters, since 2 processor SMP nodes are significantly less expensive that two separate machines, and thus the price performance of SMP based clusters should be better, if the performance is comparable. We examine this question in the context of a liquid crystal calculation which involves fairly standard matrix-vector and vector-vector operations.

*The authors were supported in part by NSF grants CDA-9617541, ASC-9720221 and DMR-8920147 ALCOM and by the OBR Investment Fund OCARnet.

2 Liquid Crystal Physics

Liquid crystals are so called because they exhibit some of the properties of both the liquid and crystalline states. In fact they are substances which, over certain ranges of temperatures, can exist in one or more *mesophases* somewhere between the rigid lattices of crystalline solids, which exhibit both orientational and positional order, and the isotropic liquid phase, which exhibits neither. Liquid crystals resemble liquids in that their molecules are free to flow and thus can assume the shape of a containment vessel. On the other hand they exhibit orientational and possibly some positional order. This is due to the intermolecular forces which are stronger than those in liquids and which cause the molecules to have, on average, a preferred direction. Liquid crystals may exist in a number of mesophases, such as the *nematic*, *smectic*, *cholesteric* phases (see [**?**]). Here we confine ourselves to nematic liquid crystals, which exhibit orientational but no positional order.

In many liquid crystal devices, the liquid crystal molecules at the edge of the device are mechanically given a specific orientation which induces the molecules in the interior of the device to mimic that orientation to the extent possible. For a given device geometry and a given molecular edge orientation, it may not be possible for the liquid crystal molecules to duplicate that edge orientation throughout the interior of the device. For instance consider a thin slab containing liquid crystal material in which the molecules at the edge of the device are oriented radially. This is illustrated in Figure **??**. Since the radial orientation cannot be maintained throughout the interior, pockets of liquid crystal with no fixed orientation form. These pockets, called defects, are illustrated by the empty spaces in Figure **??**, and, depending on the temperature and magnetic or electric field strength, will form in different places. It is of significant importance in liquid crystal research to develop computer models which can accurately determine the orientation and the defects of liquid crystals in a confined medium, especially in the presence of changing temperature and field intensities.

The paper is organized as follows: In section **??**, we give a brief description of the mathematics of the liquid crystal model. In section **??**, we describe how the code is implemented on a SMP workstation cluster. In section **??**, we describe the hardware and software of the cluster we are using, and how the performance of the cluster is affected by the TCP and MPI implementation. Finally, in section **??**, we discuss the performance of the application

on the cluster.

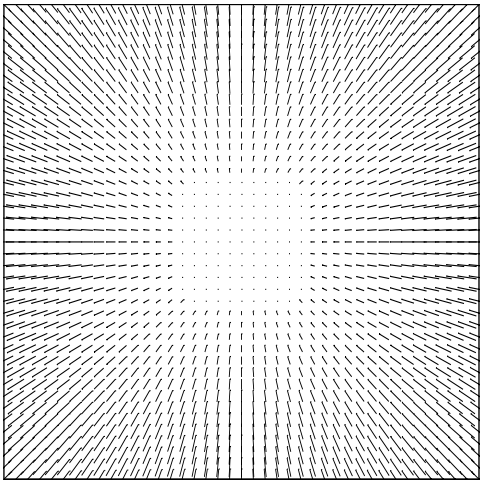

Figure 1: Liquid Crystal Orientation for Scaled Temperature $\mathcal{T} = 0.3013$

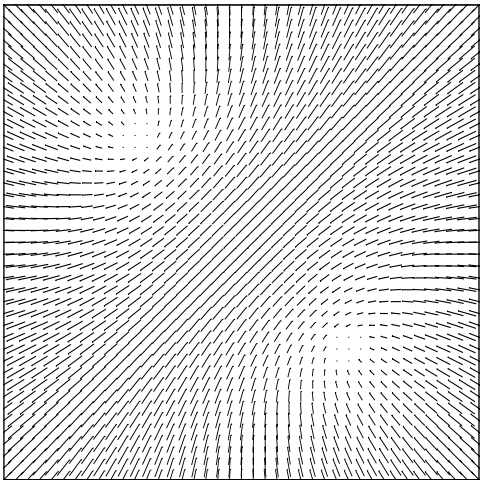

Figure 2: Liquid Crystal Orientation for Scaled Temperature $\mathcal{T} = 0.3174$

3 The Liquid Crystal Model

The problem under consideration is to determine the minimum energy configuration of liquid crystals in a slab

$$\begin{aligned}\Omega &= \{(x_1, x_2, x_3) : 0 \le x_1 \le a, 0 \le x_2 \le b, \\ &\qquad 0 \le x_3 \le c\}\end{aligned} \tag{1}$$

with surface $\partial\Omega$. Using the Landau-de Gennes formulation, the free energy can be expressed in terms of a tensor order parameter field Q; see Priestly et al. [**?**]. The free energy is given by

$$\begin{aligned}F(Q,\mathcal{T}) &= F_{\rm vol}(Q,\mathcal{T}) + F_{\rm surf}(Q) \\ &= \int_\Omega f_{\rm vol}(Q,\mathcal{T})dV + \int_{\partial\Omega} f_{\rm surf}(Q)dS,\end{aligned} \tag{2}$$

where $Q = Q(p)$, $p \in \Omega$, is a 3×3 symmetric traceless tensor, which is represented by

$$\begin{aligned}Q(p) &= q_1(p)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \\ &+q_2(p)\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + q_3(p)\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \\ &+q_4(p)\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} + q_5(p)\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}\end{aligned} \tag{3}$$

and the q_i are real-valued functions on Ω. The q_i are to be determined so that the free energy (**??**) is minimal. The representation

$$\begin{aligned}f_{\rm vol}(Q,\mathcal{T}) &= \tfrac{1}{2}\mathcal{L}_1 Q_{\alpha\beta,\gamma}Q_{\alpha\beta,\gamma} \\ &+\tfrac{1}{2}\mathcal{L}_2 Q_{\alpha\beta,\beta}Q_{\alpha\gamma,\gamma} + \tfrac{1}{2}\mathcal{L}_3 Q_{\alpha\beta,\gamma}Q_{\alpha\gamma,\beta} \\ &+\tfrac{1}{2}\mathcal{A}\,\mathrm{trace}(Q^2) - \tfrac{1}{3}\mathcal{B}\,\mathrm{trace}(Q^3) \\ &+\tfrac{1}{4}\mathcal{C}\,\mathrm{trace}(Q^2)^2\end{aligned} \tag{4}$$

uses the convention that summation over repeated indices is implied and indices separated by commas represent partial derivatives. For example,

$$Q_{\alpha\beta,\gamma}Q_{\alpha\gamma,\beta} = \sum_{\alpha=1}^{3}\sum_{\beta=1}^{3}\sum_{\gamma=1}^{3}\frac{\partial Q[\alpha,\beta]}{\partial x_\gamma}\cdot\frac{\partial Q[\alpha,\gamma]}{\partial x_\beta}.$$

The bulk parameter $\mathcal{A}$ is assumed to be of the form $\mathcal{A} = \mathcal{A}_0(\mathcal{T} - \mathcal{T}_0)$, where $\mathcal{A}_0$ and $\mathcal{T}_0$ are constants and $\mathcal{T}$ is the normalized temperature of the liquid crystals. In this paper we take $\mathcal{T}_0 = 0$ and $\mathcal{A}_0 = 2$, which gives $\mathcal{T} = \frac{1}{2}\mathcal{A}$. The quantities $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ are elastic constants, and $\mathcal{B}$ and $\mathcal{C}$ are bulk constants. In addition, we assume

$$f_{\rm surf}(Q) = \mathcal{W}\,\mathrm{trace}((Q - Q_0)^2), \tag{5}$$

where $\mathcal{W}$ is a constant and the tensor Q_0 is determined by the boundary conditions for the functions q_i. The weak anchoring model is obtained when small to moderate values of $\mathcal{W}$ are used, while large values of $\mathcal{W}$ give the strong anchoring model. Details can be found in [**?**, **?**, **?**, **?**].

The minimum energy equilibrium configuration of the liquid crystals is determined by solving the Euler-Lagrange equations associated with (**??**). These equations yield a boundary value problem for a system of nonlinear partial differential equations with unknown q_i.

In this paper, we will assume that the liquid crystal material is contained in a small cube. By changing the length scale, the domain of the problem can be changed from a small cube to a standard $1 \times 1 \times 1$ cube. Discretizing the Euler-Lagrange equations on the unit cube using finite differences produces a large three dimensional grid of points, typically with at least 100 points along each edge, that is about 100^3 points total; see Figure **??**.

For each point $p_{i,j,k} = (x_1^i, x_2^j, x_3^k)$ in this grid, there are 5 variables representing the values of

$$q_1(p_{i,j,k}), \ldots, q_5(p_{i,j,k})$$

in the expansion of the tensor at that point, and 5 discretized Euler Lagrange equations. This produces a system on the order of 5×10^6 nonlinear equations in 5×10^6 unknowns. We represent this by

$$G(Q, \mathcal{T}) = 0. \quad (6)$$

We solve this system by Newton's method. Each iteration of Newton's method requires the solution of a linear system of equations with the matrix of partial derivatives $G_Q(Q, \mathcal{T})$ obtained from the discretized Euler-Lagrange equations. Since the matrix $G_Q(Q, \mathcal{T})$ is extremely large, on the order of $10^6 \times 10^6$ and sparse, each row containing no more than 5×27 entries, iterative methods must be used to solve the linear system in which it appears. Because $G_Q(Q, \mathcal{T})$ can be indefinite or singular for some values of Q and $\mathcal{T}$, we use the Krylov subspace method MINRES [**?**] as our linear systems solver.

Due to the complexity of the liquid crystal equations, the C code for the Newton's method is generated using a Maple program. It calculates the Euler-Lagrange equations symbolically and discretizes them using centered difference approximations for the derivatives. The resulting equations for a given point depend on the data at that point and the data at the closest 26 points, that is the points in the $3 \times 3 \times 3$ sub-cube centered at that point. To conserve memory we use a matrix-free formulation, that is the entries of $G_Q(Q, \mathcal{T})$ are not stored, but are calculated as needed.

4 The implementation of the parallel method

The use of Newton's method together with MINRES produces a program that spends nearly all its time doing matrix-vector multiplications and vector vector operations (the evaluation of $G(Q, \mathcal{T})$ in Newton's method costs about as much as one matrix-vector multiply), and because of this the efficiency of the program depends primarily on the implementation of these operations.

However, the program we are using is legacy code which was designed to run on a small (< 8) workstation cluster and which uses a data distribution scheme which is efficient for a cluster with only a few nodes, but is less so on our current cluster. Our data is arranged so that each processor communicates with at most two other processors. To see how this is done, consider that the cubic grid of points produced by the discretization is sliced by planes parallel to the XY axis so that there is one slice per processor and so that each slice contains approximately the same number of grid points. We associate the data in each slice with a processor as in Figure **??**.

For a point in the interior of a slice, that is a point all of whose neighbors in the grid are contained in the same processor, all of the data required to form the discretized Euler Lagrange equations at that point is stored in the associated processor. Also, at an interior, all the data required to form the associated entry in the product of the matrix $G_Q(Q, \mathcal{T})$ with a vector X arising in the iterative method will reside in the same processor. So for interior points no communications is required to implement Newton's method.

For a point on the edge of a slice, not all the data required for Newton's method is located in that processor, so edge data must be exchanged by processors associated with contiguous slices. It is convenient to add an additional plane of variables at each interior edge of a slice to receive the data being exchanged. Then after a communication step to fetch the required data, the required calculations can be performed using standard indexing. This is illustrated in Figure **??**. The communication and synchronization of the processes are handled by MPI. Accordingly, each processor performs $O(N^3)$ calculations, where N is the number of points in each direction, and exchanges at most $2N^2$ data points per MINRES iteration.

An asymptotically more efficient data distribution algorithm would result if each processor, where possible, were given all the data of a sub-cube of the larger cube. This is possible when using 8, 27, or 64 processors. The efficiency of this type of data distribution, as compared to our present scheme, increases with the number of processors and benefits of the more efficient data distribution will ultimately make it worthwhile to replace our legacy code.

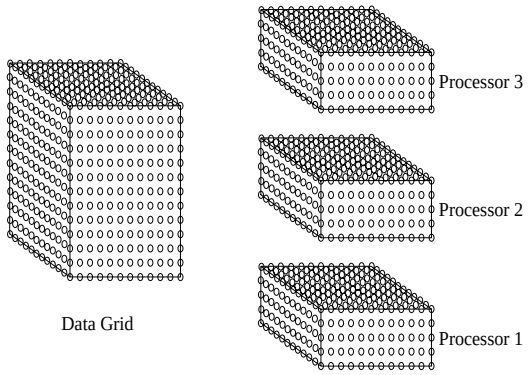

Figure 3: Processor Assignment

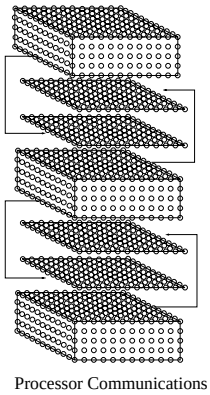

Figure 4: Communication

5 Experimental Framework

The local experimental environment used in this paper consists of 16 dual processor Pentium III/450s running Linux connected by switched 100 Mbps Ethernet. The machines were custom built from parts for cost reasons and to maintain commodity components. Each has a SuperMicro motherboard based on the Intel 440BX chip set, and 256 MB of PC 100 memory. The 100 Mbps Ethernet cards are based on the DEC Tulip chipset. The Red Hat 6.1 distribution of Linux was used with the 2.2.10 SMP kernel. The particular application that we are testing is parallelized using

MPI. We have chosen to use the LAM implementation of MPI as opposed to the more widely used MPICH distribution, since it is more tuned for use in a cluster environment, and previous experience has indicated that it achieves higher throughput.

Distributed applications using MPI (either LAM or MPICH) as communication transport on a cluster of computers (such as a Networks of Workstations (NOWs) or PC/Beowulf cluster) impose heavy demands on communication networks. In practice, MPI communications are often implemented over socket communication using TCP/IP. Therefore, it is important, among other factors, to have high TCP performance in order for distributed applications to communicate efficiently and thus achieve high speedup. The TCP protocol was originally designed to run reliably over various network media regardless of transmission rate, delay, corruption, duplication, or reordering of segments. In order to take advantage of high speed networks, systems must be configured to support and utilize extensions to the basic TCP/IP protocol suite. We have taken care to tune the TCP implementation for high performance high bandwidth communication, in particular by using the TCP/IP Extensions for High-Performance (*RFC 1323* [**?**]), which defines a set of TCP extensions to improve performance over large $bandwidth \times delay$ paths and to provide reliable operation over high-speed paths. RFC 1323 defines new TCP options for scaled windows and timestamps, which are designed to provide compatible interaction with TCP's that do not implement the extensions. The host systems must support large enough socket buffers for reading and writing data to the network. We set the system level maximum socket buffer size to the maximum allowable in Linux, which is 128KB in Linux 2.1.x and later, and is set by setting the default socket size to 64KB.

It should be noted that the version of the Linux kernel has considerable influence on the attainable TCP performance and hence on the MPI performance. Early implementation of the TCP/IP drivers exhibited noticeable performance dropoffs at certain message sizes. In later kernels most of these have been eliminated. In the case of Gigabit Ethernet, these performance variations with kernel version are particularly noticeable. For example, in [**?**], it is shown that the peak performance was approximately 226 Mbps for kernel 2.0.35, whereas it had improved to approximately 322 Mbps in the development kernels 2.1.x and the more recent production kernel 2.2.1. However Linux kernel versions 2.2.0 through 2.2.9 still had some TCP/IP performance problems. In particular, 2.2.5 has many severe drops in the performance curves, caused by Linux kernel delayed TCP acknowledgments. The LAM implementors warn that LAM may exhibit poor performance due to these Linux TCP/IP kernel bugs, particularly in client-to-client (C2C) communications, which bypass the LAM daemon. Since client-to-client mode is normally recommended for high performance LAM communication, it is important to use a kernel version, such as those greater than 2.2.10, which rectify this problem.

The version of LAM used in these tests is LAM 6.3.1. As we mentioned above the LAM implementation of MPI is optimized for use in a clustered environment, particularly clusters consisting of SMP nodes. Among the optimizations is the fact that it sets the socket size to the system default size, that is 64KB rather than to a smaller size. As indicated in [**?**], in the case of ATM networks, this can increase the throughput by of the order of 18%. All versions of LAM, since LAM 6.2b, provide three client-to-client transport layers which implement the request progression interface (RPI). The three client-to-client transports are TCP, USYSV and SYSV. In these tests, we employed the USYSV transport layer, which is a multi-protocol transport layer. Processes on the same node communicate via SYSV shared memory and processes on different nodes communicate via TCP sockets. USYSV uses spin-locks for shared memory synchronization as well as a SYSV semaphore or pthread mutex for synchronizing access to a per node global shared memory pool. The USYSV transport is intended to give the best performance on SMPs. One of the purposes of this paper is to test whether the USYSV transport layer overcomes the problems which have traditionally arisen in MPI based programs, which use TCP to perform interprocessor communication on SMP nodes.

6 Application Performance

In this section, we consider the resulting performance for the liquid crystal application on meshes of $4 \times 4 \times 4$, $8 \times 8 \times 8$, $16 \times 16 \times 16$, $32 \times 32 \times 32$, and $64 \times 64 \times 64$ points. Note that, with the slicing algorithm described, since we are communicating $N \times N$ sets of 5 tensor values each of 8 bytes, we will be communicating $40 * N^2$ bytes in each case. This is 640, 2560, 10240, 40960 and 163840 bytes for the $4 \times 4 \times 4$, $8 \times 8 \times 8$, $16 \times 16 \times 16$, $32 \times 32 \times 32$, and $64 \times 64 \times 64$ cases respectively. It was not possible to produce full scalability results for problems larger than $64 \times 64 \times 64$ due to the 256 MB memory limit on the PCs. A $128 \times 128 \times 128$ problem would require in excess of 800 MB of memory, and so could only be run on 4 or more nodes.

Figure **??** shows the variation in total execution time for the Newton iterations in the liquid crystal simulation, the total number of MINRES iterations, and the time per MINRES iteration with the size of the problem. It should be noted that the total time increases due both to the increase in the number of MINRES iterations, which is due to the increase in the difficulty in convergence of the Newton process, and due to the increasing time per MINRES iteration, which is due to the larger matrices being manipulated. Since the increase in the number of MINRES iterations is purely a matter of numerical conditioning, and cannot be addressed by parallelization, we factor this effect out and consider only the time per MINRES iteration in subsequent discussions.

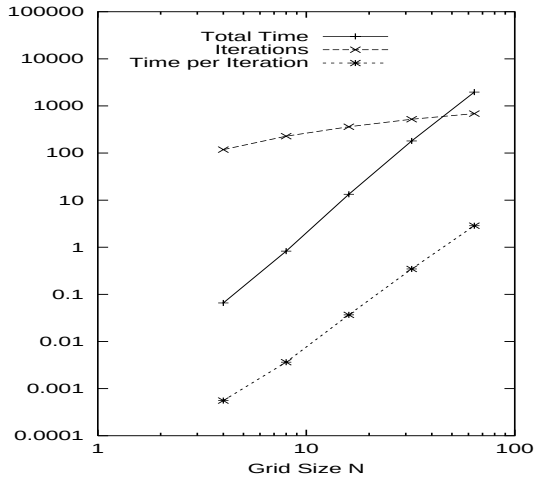

Figure 5: Total Time, Number of Minres Iterations and Time per Iteration

The most computationally intensive part of the algorithm consists of matrix-vector multiplies. In the case of small problems, such as $4 \times 4 \times 4$ ones, these account for approximately 56% of the total time, and in larger ones, such as $32 \times 32 \times 32$ and $64 \times 64 \times 64$,

for between 60% and 65%. Since each multiply involves fetching the data for two adjacent slices through the network, that is $80*N^2$ bytes of data is sent and $80 * N^2$ bytes of data is received by most processors, it would seem an obvious optimization to overlap communication and computation. This can be accomplished in MPI by using Non-Blocking calls rather then blocking `sendrec` calls. This permits initiating the transfer, performing the interior computations, and only then blocking or waiting if the data has not arrived. As indicated in Figure **??**, this provides surprisingly little improvement.

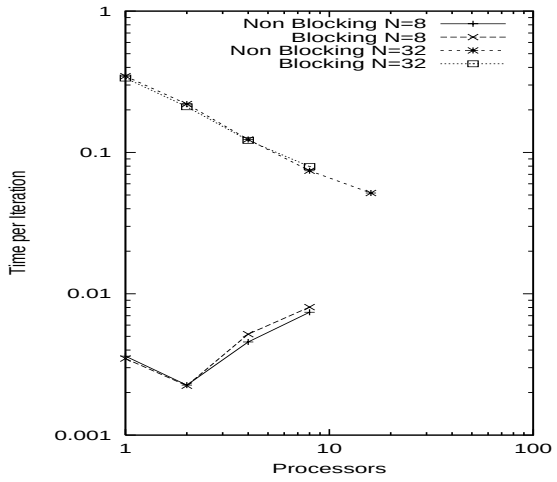

Figure 6: Comparison of Blocking and Non Blocking MPI Calls

Another issue of interest is the comparative efficiency of SMP nodes for practical computations. Early reference and public domain versions of MPI tended to be designed for generic networks, including wide area networks. As such, unlike proprietary implementations, such as that of Cray, they were not optimized for particular platforms or architectures. The LAM implementation was aimed at networks of workstations, particularly in a local area network environment, and after release 6.2b at clusters consisting of SMP nodes. In the next series of Figures **??**–**??**, we examine the effects of SMP nodes on the performance. These plot the time per MINRES iteration against the numbers of processors for various sizes of problem and variations on SMP use. In each figure, Uniprocessor indicates the performance graph for the case where only one processor per machine is used. We remark that the machines as still configured using the SMP version of the Linux kernel. Tests have shown that the results using a uniprocessor version of the same kernel do not differ substantially. By default, LAM will bind processes (copies of the program) to MPI nodes in a round robin manner, that is, if the user has booted LAM daemons on 16 dual processor machines using `lamboot` and then specifies

```
mpirun -c2c -c 32 prog
```

LAM will assign the first process to LAM node `n0`, the second to LAM node `n1`, etc., the sixteenth to LAM node `n15`, the seventeenth again to LAM node `n0`, etc. and finally the thirty-second to node `n31`. In the case of our computation, this is far from optimal, since it means that each processor performs two off machine communications per matrix multiply or a total of four per machine. This case is referred to as SMP Default in the figures. If sequential processes had been assigned to the same LAM node and thus to the two processors on the same machine, only two off machine communications per matrix multiply would be necessary. When we explicitly force this to occur, we refer to the resulting computations as SMP Explicit in the figures. Figure **??** and **??** give the results for small problems and larger problems respectively using Non-Blocking communications. Figures **??** and **??** give the corresponding results for small problems and larger problems using standard blocking `sendrec` communications. To interpret these

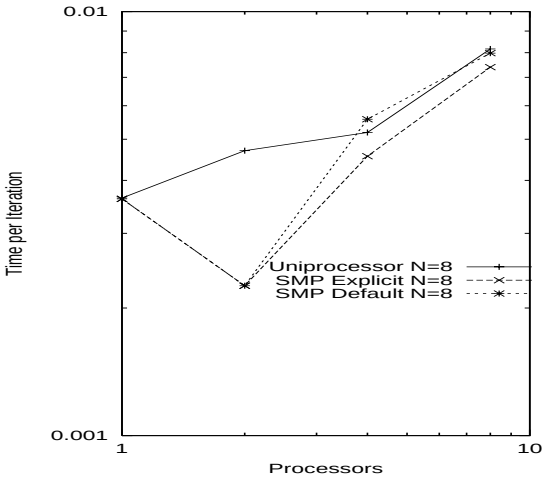

Figure 7: Comparison of SMP modes for Non Blocking MPI Calls on Small Problems

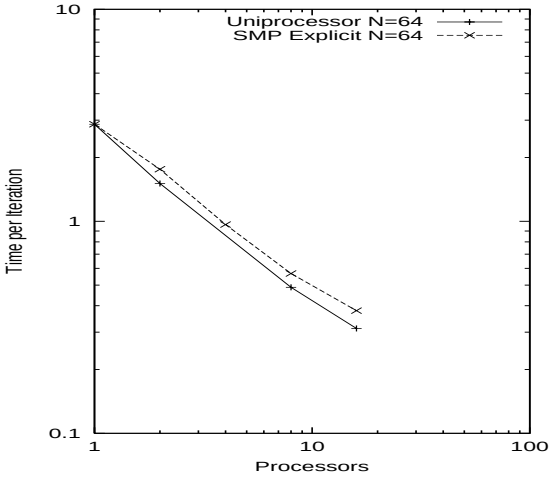

Figure 8: Comparison of SMP modes for Non Blocking MPI Calls on Large Problems

figures, it is necessary to understand that by virtue of the structure of the algorithm, each processor will ultimately have to wait for all others to complete a phase of the computation. This is enforced by the various collective communication operations involved such as the calculation of dot products and norms, and also explicitly by barriers which ensure that no processor proceeds to use old data in

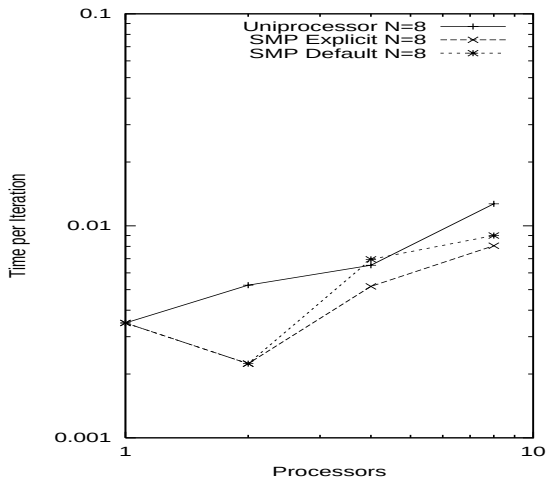

Figure 9: Comparison of SMP modes for Blocking MPI Calls on Small Problems

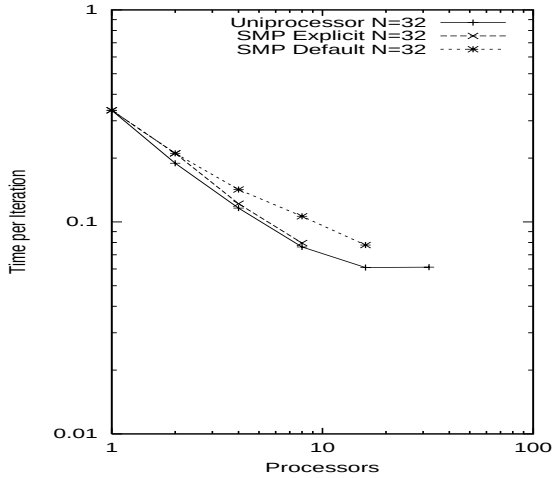

Figure 10: Comparison of SMP modes for Blocking MPI Calls on Large Problems

the next iteration. Thus one would not normally expect the SMP nodes to complete an iteration faster than uniprocessor nodes. At best in the SMP Explicit case, the SMP nodes have to perform two off-machine data exchanges similar to a uniprocessor node, and, in addition, have to perform an on-machine communication via shared memory, which adds to the bus contention.

Figures **??** and **??** are for the case of $N = 8$ that is a grid with $8 \times 8 \times 8$ points. This gives 512 points or 2560 independent variables. Naturally, we do not advocate trying to parallelize such a small problem. It is presented here only for its interest in the context of the relative performance of uniprocessor and SMP nodes. Since the amount of data and computation in this case is small, network latency will have a significant effect on the performance. Assuming that only one SMP machines is booted as a LAM node, in the case of two processors, the SMP Explicit and Default are the same. Thus the time per iteration will be the same. Further all the communication is on the one machine using shared memory, which is faster and, in particular has lower latency, than network communication. This explains the superior performance of the SMP case. The case of four processors is also a special case, in that for the SMP Explicit case, each machines has only to send and receive one slice of data through the network, whereas in the uniprocessor and SMP Default case, some of the machines must send and receive two slices. Thus the network communication overhead is doubled. This explains why the SMP Explicit case has slightly superior performance. The other differences apparent on the graph are within the margin of error.

Figures **??** and **??** indicate that for realistic sized problems there is no significant difference between the performance of uniprocessor and SMP Explicit methods. The minor difference may be attributable to increased bus contention due to the shared memory exchanges for large messages. On the other hand there is a more significant difference between uniprocessor and SMP Explicit methods on the one hand and SMP Default methods on the other hand, which is attributable to the fact that most processors are performing twice the network communication in the case of the SMP Default method.

Finally as an indication of the overall performance achievable for realistic sizes of problem, Figure **??** gives the speedups for uniprocessor blocking and non-blocking methods on problems with $N = 32$ and 64 respectively. It should be noted that, with the

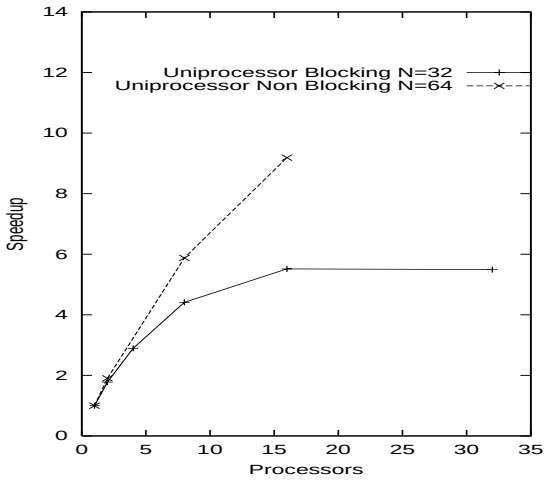

Figure 11: Speedups for Blocking and Non Blocking MPI Calls

slicing method of domain decomposition used in our algorithm, for a problem with $N = 32$ on 32 processors, there is only one layer of points per slice. This must be communicated to the slices above and below, and thus communication is the dominant effect. A similar circumstance holds for the case where the number of processors is $N/2$. On the other hand, acceptable performance is achieved when the number of processors is a reasonable proportion of the number of slices, which is likely to be the case in practice.

7 Conclusions

We have shown that a legacy liquid crystal code can be ported successfully and with reasonable performance to a Beowulf SMP

cluster, and that re-engineering it by using non-blocking MPI commands does not significantly improve the performance. Finally, we have shown that using LAM version 6.3.1, with the USYSV transport layer, which includes support for shared memory communication on the SMP node, the performance of SMP nodes is not significantly different from that of uniprocessor nodes.

References

[1] J. Baglama, D. Calvetti, L. Reichel, and A. Ruttan, "Computation of a few small eigenvalues of a large matrix with application to liquid crystal modeling", *J. Comp. Phys.*, 146 (1998), pp. 203–226.

[2] D. Calvetti, L. Reichel, and Q. Zhang, "Conjugate gradient algorithms for inconsistent linear system", in *Proceedings of the Cornelius Lanczos International Centenary Conference*, eds. J. D. Brown, M. T. Chu, D. C. Ellison, and R. J. Plemmons, SIAM, Philadelphia, 1994, pp.267-272.

[3] P. J. Collings, *Liquid Crystals, Nature's Delicate Phase of Matter*, Princeton Science Library, Princeton University Press, New Jersey, 1990.

[4] T. A. Davis and E. G. Gartland, Jr., "Finite element analysis of the Landau-de Gennes minimization problem for liquid crystals", *SIAM J. Numer. Anal.*, 35 (1998), pp.336-362.

[5] P. A. Farrell, Hong Ong, "Communication Performance over a Gigabit Ethernet Network", 19th IEEE International Performance, Computing, and Communications Conference – IPCCC 2000, pp. 181–189, 2000.

[6] P. A. Farrell, Hong Ong, A. Ruttan, "Modeling liquid crystal structures using MPI on a workstation cluster", Proceedings of MWPP' 99, to appear.

[7] P. A. Farrell, A. Ruttan and R. R. Zeller, "Finite difference minimization of the Landau-de Gennes free energy for liquid crystals in rectangular regions", *Comput. Appl. Math.*, I, C. Brezinski and U. Kulish, eds., Elsevier, Amsterdam, 1992, pp. 137–146.

[8] Jacobson, Braden, & Borman, *TCP Extensions for High Performance*, RFC 1323, May 1992.

[9] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgement Options*, RFC 2018 October 1996.

[10] V. Paxson, M Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, B. Volz , *Known TCP Implementation Problems*, RFC 2525, March 1999.

[11] E. B. Priestly, P. J. Wojyowicz and P. Sheng, eds., *Introduction to Liquid Crystals*, Plenum Press, New York, 1975.

[12] E. G. Virga, *Variational Theories for Liquid Crystals*, Chapman and Hall, London, 1994.