Parallel Performance of the Interpolation Supplemented Lattice Boltzmann Method

C. Shyam Sunder, *Indian Institute of Technology, Madras*
Baskar Ganapathysubramanian, *Indian Institute of Technology, Madras*
V. Babu, *Indian Institute of Technology, Madras*
David Strenski, *Cray, Inc.*

Parallel Performance of the Interpolation Supplemented Lattice Boltzmann Method

C. Shyam Sunder — 3rd Year Student

G. Baskar — Final Year student

V. Babu — Associate Professor [*]

Department of Mechanical Engineering,

Indian Institute of Technology,

Madras, India 600 036.

and

David Strenski

Applications Analyst

Cray Inc., Seattle,

Washington, USA.

[*]Corresponding author - vbabu@iitm.ac.in

Abstract

The interpolation supplemented lattice Boltzmann (ISLB) method was proposed by He *et. al.* (*J. Comput. Phys.*, **129**, 357 (1996)) as a modification of the traditional lattice Boltzmann (LB) method for simulating incompressible flows. The traditional LB method has been shown to be a highly parallel method (*Computers in Physics*, **8**, No. 6,705 (1994)). In this paper, the parallel performance of the ISLB method, which has different computational requirements than the traditional LB method, is investigated. The performance of the ISLB method on Cray X1, Cray T3E-900 and SGI Origin 3000 is presented. The noteworthy feature of the present implementation of the ISLB method is that it was able to achieve a sustained speed of 353 Gflops on a 60 processor Cray X1.

Keywords: Lattice Boltzmann Parallelization SHMEM

1 Introduction

The lattice Boltzmann method is a very promising method for simulating incompressible fluid flows [1]. The method is appealing due to its simplicity - simplicity in the mesh used (cartesian mesh) and simplicity in the operations performed (collision, advection and boundary update). However, the simplicity of the uniform cartesian mesh is also a liability for two reasons: one, complex, curved geometries cannot be handled without losing boundary integrity and two, uniform mesh everywhere results in a large number of mesh points, which in turn, increases the computational cost tremendously. The first issue has been addressed by Mei *et al.*, [2, 3]. The second issue has been addressed by He *et al.,* [4], wherein they proposed the interpolation supplemented lattice Boltzmann (ISLB) method. This allows non-uniform as well as curvilinear body-fitted meshes to be used in the lattice Boltzmann calculations. An alternative approach for using non-uniform meshes based on the concept of hierarchical grid refinement was proposed by Filippova and Hanel[5, 6]. In the present work we have chosen to use the former approach as it requires only a relatively straightforward modification of the traditional lattice-BGK procedure for simulating incompressible flows and is thus quite easy to implement.

2 A Brief Review of the Lattice Boltzmann Method

In this work, the 2D LB method on a square lattice is considered. Some of the basic features of the LB method alone are presented here. The details are available elsewhere [4]. Each node of the lattice is populated by three kinds of particles - a rest particle that resides in the node, particles that move in the coordinate directions and particles that move in the diagonal directions. The total number of particles in each node in this model (called the d2q9i) model is 9. The speed of the particles are such that they move from one node to another during each time step. These speeds can be written as

$$\mathbf{e}_i = \begin{cases} 0, & i=0 \\ c(\cos((i-1)\pi), \sin((i-1)\pi)), & i=1,2,3,4 \\ \sqrt{2}c(\cos((i-5)\pi/2+\pi/4), \sin((i-5)\pi/2+\pi/4)), & i=5,6,7,8 \end{cases} \tag{1}$$

Here $c = \delta_x/\delta_t$, where δ_x and δ_t are the lattice spacing and the time step respectively. In the traditional LB method the particles at each node undergo collision followed by advection. In terms of distribution functions, this can be written as [4]

$$p_i(\mathbf{x} \ + \ \mathbf{e}_i\delta_t, t+\delta_t) - p_i(\mathbf{x}, t) = -\frac{1}{\tau}\left[p_i(\mathbf{x}, t) \ - \ p_i^{(eq)}(\mathbf{x}, t)\right], \tag{2}$$

where τ is the dimensionless collisional relaxation time and is related to the kinematic viscosity ν of the lattice fluid as [4]

$$\nu = \frac{(2\tau-1)}{6}\frac{\delta_x^2}{\delta_t} \tag{3}$$

Equation (2) above describes the evolution of the LB automaton. The equilibrium functions that appear in the right hand side of this equation can be evaluated [4] as follows:

$$p_i^{(eq)} = w_i\left[p \ + \ \rho_0\left((\mathbf{e}_i\cdot\mathbf{u}) \ + \ \frac{3}{2}\frac{(\mathbf{e}_i\cdot\mathbf{u})^2}{2} \ - \ \frac{1}{2}\mathbf{u}^2\right)\right], \tag{4}$$

where, p is the pressure and $\mathbf{u}$ is the velocity vector of the fluid and are given as

$$p = \sum_{i=0}^{8} p_i \tag{5}$$

$$\rho_0 \mathbf{u} = \frac{1}{c_s^2} \sum_{i=1}^{8} \mathbf{e}_i p_i \tag{6}$$

where c_s is the speed of sound and is equal to $c/\sqrt{3}$ for the model under consideration. The quantity ρ_0 can be thought of as a reference density and has been taken to be equal to 1.

In case of the ISLB method, the lattice Boltzmann automaton is assumed to reside on a *lattice* with spacing exactly equal to the finest mesh spacing overlaid on the *computational grid*. During each time step, the particles on this *lattice* undergo collision followed by advection as dictated by the lattice-BGK model[4]. The particle distribution function at each node of the *computational grid* can then be determined by using second order accurate Lagrangian interpolation [4] (this is not required in the finest mesh region as the mesh spacing here equals the lattice spacing). The interpolation is done by using values upwind (based on the direction of the particle velocity not the fluid velocity) of the node. The particle distributions on the boundary nodes of the *computational grid* are modified next according to the imposed boundary conditions. This way, although the automaton resides on a fine mesh, the particle distribution functions are calculated and stored only on the nodes of the *computational grid* which can be tailored to the problem being solved *i.e.,* fine mesh in regions of strong gradients and coarse mesh everywhere else.

3 Parallel Implementation of the ISLB Method

3.1 Computational Issues

The computations of the ISLB method can be summarized as follows:

1. Start with an initially guessed fluid velocity field
2. Calculate the equilibrium particle distribution based on the velocity and pressure field based on equation (4)
3. Perform the collision operation at each node (right hand side of equation (2))
4. Communicate the post collision distribution data from appropriate nodes to neighboring processors that need the data.
5. Calculate particle distribution function at each node from the post collision distribution using second order accurate interpolation
6. Update the particle distributions on physical boundaries based on the prescribed boundary conditions
7. Calculate the fluid velocity and pressure at each node using equations (5,6)

8. Repeat steps (2) through (7) until the flow field reaches a steady state or until enough cycles of the unsteady flow field have been obtained.

From a computational perspective, steps (2), (3), (5) and (7) are the most important as they account for nearly all the computations. In the actual implementation, steps (7), (2) and (3) can be combined which results in considerable reuse of data already in the cache thereby improving the computational speed. Mapping the arrays onto the L1 and L2 cache properly with pads between them and stripmining the loops also results in further improvement of speed. All the computational kernels have been written in `Fortran77` and everything else in `C`. In the present case, the total number of floating points operations per node per time step comes out to be 120 multiplications and 101 additions/subtractions. This allows the computational speed to be calculated very accurately, once the execution time is measured. In calculating the computational speed, both additions/ subtractions and multiplications were treated equally. It should be noted that step (5) above is not applicable in the traditional LB method, but in the ISLB method it accounts for about 45% of the total computations. This increase in computations is more than offset by the decrease in the number of nodes in the computational grid for the ISLB method.

3.2 Communication Issues

The amount of data to be communicated to the neighboring processors in step (4) above is more in the ISLB method than the traditional LB method. This is because the second order accurate interpolation procedure (step 5) at each node requires data from that node plus the nearest as well the next nearest neighboring node in the upwind direction. This means that, for particles with velocity in the directions $i = 1, 2, 3, 4$ in equation (1) above, data from three nodes is required, while for particles with velocity in the directions $i = 5, 6, 7, 8$ data from a total of 9 nodes is required. In the current implementation, communication between processors has been done using `SHMEM` routines, which offer high data transfer rates between processors on the Cray and the SGI machines. Porting to MPI is currently in progress. Also, `SHMEM_PUT` routines have been used throughout, as they offer better performance than the `SHMEM_GET` routines.

3.3 Load Balancing Issues

In the present work, all the parallel computations have been done by dividing the computational domain into equal parts and then assigning each part to a processor. The size of each part is determined based on considerations of surface-to-volume ratio [7]. The computations involved in steps (2),(3) and (7) require only local data and so these steps can be parallelized trivially and they also do not pose any difficulties related to load balancing. Step (4), namely communication of data has to be completed before the computations in steps (5) and (6) can be done (it should be kept in mind that step (7) has been merged with step (2)). This poses some load balancing problems as the processors that have been assigned the interior of the computational domain communicate more data than the others. Step (6), namely, implementation of the boundary conditions also presents difficulties in load balancing as the processors assigned the interior of the domain do less computations (if any) in this step. However, good load balance can be achieved without too much additional effort as will be shown in the next section.

4 Results and Discussion

4.1 Cray X1

Parallel performance of the code on a 16 node Cray X1 machine is shown in Figs. 1 and 2. The Cray X1 is a scalable vector MPP architecture with a peak flop rate of 12.8 Gflops (64-bits) per MSP processor or 24 Gflops for 32 bit calculations. The latter number is relevant in the present case as all the calculations have been done using 32 bit floating point numbers. Each node of the machine contains 4 MSP processors. It is also possible to run parallel applications in an SSP mode where each MSP processor behaves as though it is a collection of 4 tightly coupled SSP processors. The Cray X1 is a shared memory architecture within each node whereas the memory is logically shared but distributed across the nodes. It is also possible to specify the number of processors to be used per node for parallel applications. For example, an 8 processor run can be done by using 8 nodes with 1 MSP processor per node or by using 4 nodes with 2 MSP processor per node or by using 2 nodes with 4 MSP processors (the maximum possible) per node. The same concept can be used for execution in SSP mode also, except that, now, upto 16 SSP processors are available per node.

The performance curves in Fig.1 show the results for parallel runs with MSP processors. In these figures, speedup values are presented for the slowest execution times. Also, the solid line in this figure (as well as the subsequent figures) indicates linear speedup. The general trend in Fig. 1 is that the load balance is quite good when the number of processors is less than 50 and it worsens somewhat afterwards (the difference in CPU time between the processor finishing first and last in the worst case was about 3% of the average CPU time). The maximum speed achieved in MSP mode was 10 Gflops per processor and the minimum was 5 Gflops. In general, the performance deteriorated somewhat as the number of processors per node was increased. This can be attributed to the fact that the bandwidth available for each processor in a particular node decreases as more processors in that node are used. The maximum speed achieved was **353 Gflops** for a run with 60 MSP processors.

Performance curves for parallel calculations with SSP processors are shown in Fig. 2. The speedup is very close to being linear for all the cases for number of processors upto 30 or so. Beyond that, as the number of SSPs per node is increased, the speedup falls off considerably, as the available bandwidth per node has to be shared among more SSPs. This trend is similar to what is seen with increasing number of MSP processors per node. However, the load balance, even for the run with 260 processors is quite good as can be seen from this figure. This clearly shows that as the problem size per processor is reduced, the increased communication cost more than offsets the performance gains due to better positioning in the cache. The maximum speed achieved was **293 Gflops** for a run with 260 SSP processors. The overall performance of the MSP processors is better because the parallelization is at the streaming level. In the case of SSP processors, more parallelization is at the process level and the overheads associated with this are more than those of the MSP processors.

4.2 Cray T3E

Parallel performance of the code on the distributed memory Cray T3E-900 architecture is discussed next. This architecture utilizes 450 MHz DEC EV5 processors and on the machines that were used for the timing studies, each processor has 256 MB of memory, 8 KB of primary cache and 96 KB of secondary cache. In this work, performance studies were carried out on two T3E machines, which are identical except for the number of processors. All the calculations on the T3E were done with 64 bit numbers as it is not possible to do 32 bit calculations.

The computational speed per processor for a 2 processor run came out to be 144 Mflops (the size of the problem in each processor was too big for single processor runs). The speed increased to 165±3 Mflops per processor for runs using larger number of processors, which indicates that the problem fits better in the cache as the number of processors is increased. This is reflected in the super-linear speed-up seen in Fig. 3 right from the beginning. For the run with 260 processors, the difference in CPU time between the processor finishing first and last was about 0.1% of the average CPU time, indicating extremely good load balance. The speed-up is 14% super-linear for the run with 260 processors. The noteworthy trend from Fig. 3 for the T3E architecture is that the degree of super-linearity continues to increase for number of processors upto 260. This is in contrast to the usual trend on distributed memory machines where the speed-up tapers off beyond a certain number of processors as applications become communication bound. The maximum sustained speed achieved for this architecture was 43 Gflops for a run with 260 processors. It should be kept in mind that these speeds are for 64 bit calculations, so when comparing with the speeds from the other architectures such as the Cray X1 above or the SGI Origin (see below), these should be multiplied by a factor of two.

4.3 SGI Origin 3000

Parallel performance of the code on a shared memory SGI Origin 3000 machine running 500 MHz, R14000 processors is presented next. This machine has 64 processors, main memory of 32 GB and primary and secondary caches of size 32 KB and 8 MB respectively. Figure 4 shows the performance of the current implementation of the ISLB method on this machine. The speed-up is slightly super-linear for number of processors upto 10. Beyond this, as the problem size in each processor becomes small enough to fit in the secondary cache, the speed-up is highly super-linear. The computational speed per processor also increases to about 650 Mflops from 430 Mflops. For number of processors equal to 24, the speed-up is 75% super-linear, but for 63 processors, it is only 50% super-linear, indicating that the application is becoming communication bound. The maximum speed achieved on this architecture was 40 Gflops for a run with 63 processors.

Finally, the overall speeds achieved on the three architectures are shown in Fig. 5. Here, the computational speed (calculated based on the CPU time of the processor finishing last) is plotted against the number of processors using a *log-log* scale. The values plotted in this figure for the case of the Cray T3E machine are actual speeds (for 64 bit calculations) multiplied by two. For a given number of processors, say, for example 60, the Cray X1 in MSP mode gives speeds that are about 3.5 times better than the Cray X1 in SSP mode, about 18 times better than the Cray T3E and about 9 times better than the SGI Origin 3000

5 Conclusions

The parallel performance of the Interpolation Supplemented Lattice Boltzmann method on different architectures has been investigated. Although the communication and computational requirements per node per time step of the ISLB method are higher than the traditional LB method, the parallel performance is, nevertheless, quite good, particularly on cache based machines. Some further improvement in the single processor performance of the implementation appear to be possible and these are currently being explored. Performance of the method when using MPI is of considerable interest as the data transfer rates between processors is lesser in the MPI environment than with `SHMEM` routines. This is very likely to introduce some load imbalance and thus affect the parallel performance.

Acknowledgements

The authors would like to thank the Computer Center, IIT Madras for granting dedicated time on their SGI Origin 3000. The authors would also like to thank Cray Inc, for allowing timing studies to be conducted on the Cray X1 and the Cray T3E at their Chippewa Falls facility in Wisconsin. The authors would further like to thank the Arctic Region Supercomputer Center at Fairbanks, Alaska for granting permission to perform timing studies on their Cray T3E.

References

[1] S. Chen and G.D. Doolen, Lattice Boltzmann Method for Fluid Flows, *Annual Review of Fluid Mechanics*, **30**, 329 (1998).

[2] R. Mei, L.-S Luo and W. Shyy, An Accurate Curved Boundary Treatment in the Lattice Boltzmann Method, *Journal of Computational Physics*, **155**, 307 (1999).

[3] R. Mei, W. Shyy, D. Yu and L.-S Luo, Lattice Boltzmann Method for 3D Flows with Curved Boundary, *Journal of Computational Physics*, **161**, 680 (2000).

[4] X. He, L.-S. Luo and M. Dembo, Some progress in lattice Boltzmann method. Part 1. Nonuniform mesh grids, *J. Comput. Phys.*, **129**, 357 (1996).

[5] O. Filippova and D. Hanel, Grid refinement for lattice-BGK models, *J. Comput. Phys.*, **147**, 219 (1998).

[6] O. Filippova and D. Hanel, Acceleration of lattice-BGK schemes with grid refinement, *J. Comput. Phys.*, **165**, 407 (2000).

[7] M. J. Quinn, *Parallel Computing, Theory and Practice*, 2nd Edition, McGraw-Hill Inc, 1994, pp. 241-243.

List of Figures

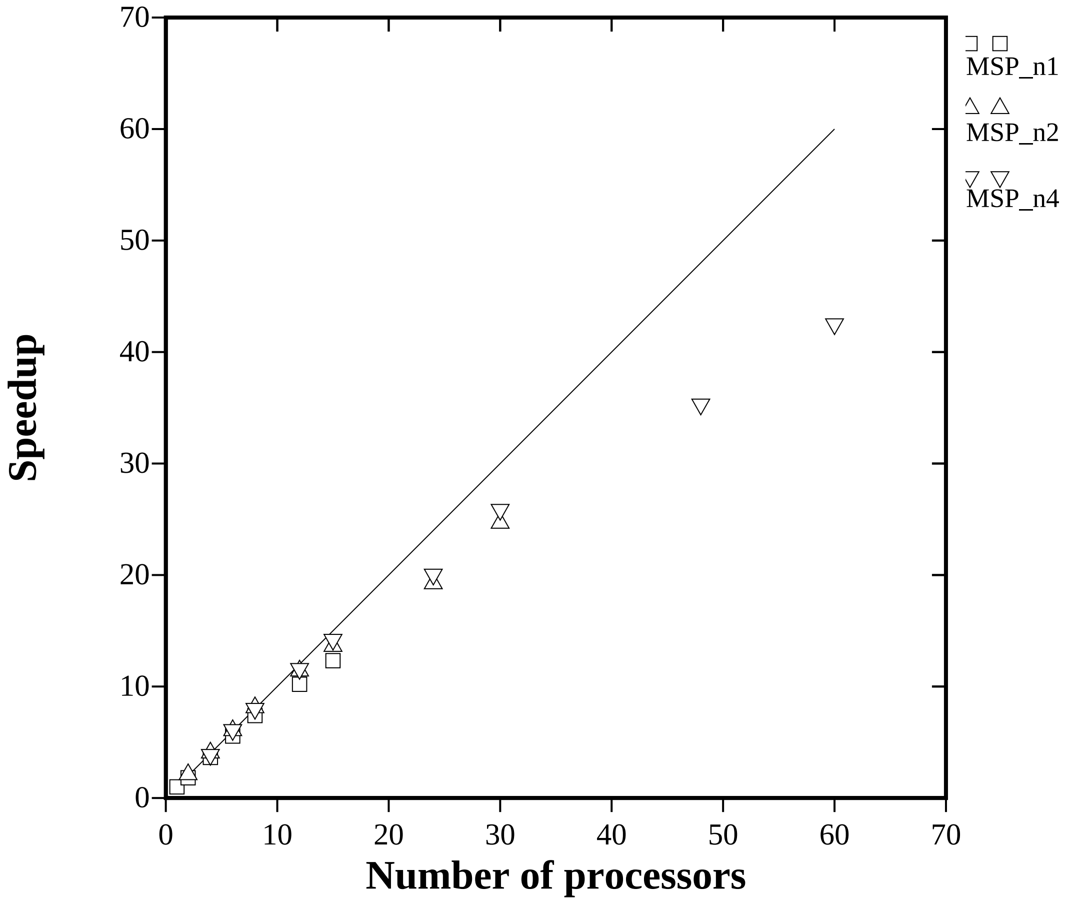

Figure 1: Performance curve on Cray X1 for 1, 2 and 4 MSPs per node

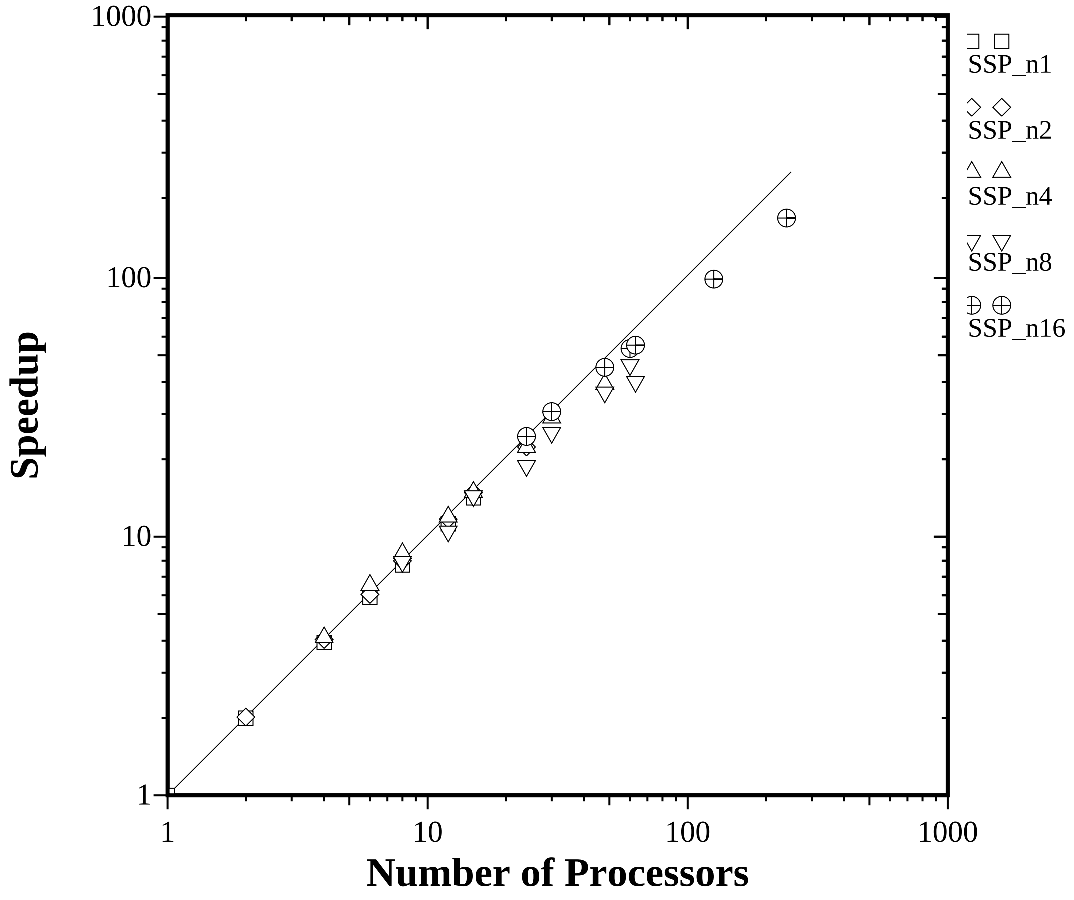

Figure 2: Performance curve on Cray X1 for 1, 2, 4, 8 and 16 SSPs per node

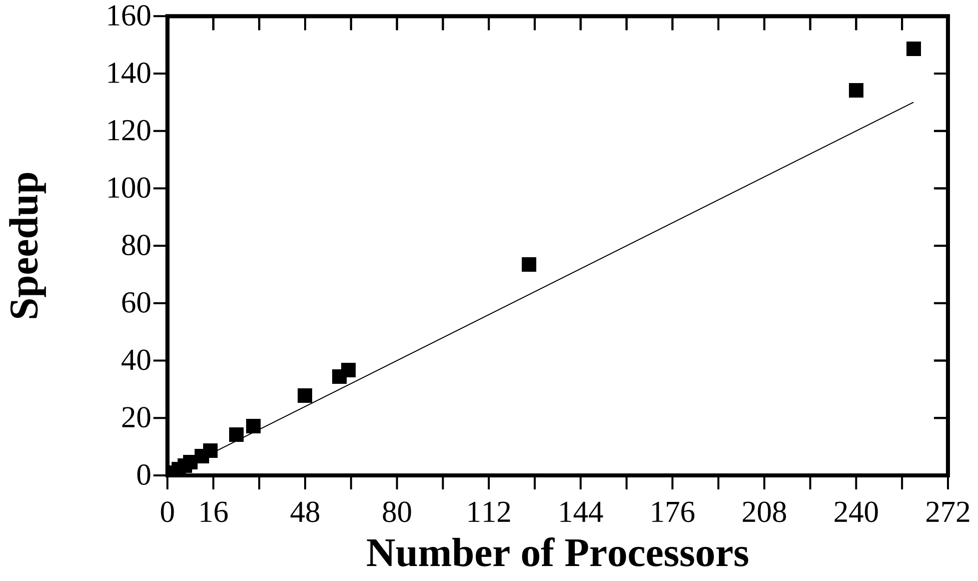

Figure 3: Performance curve on Cray T3E

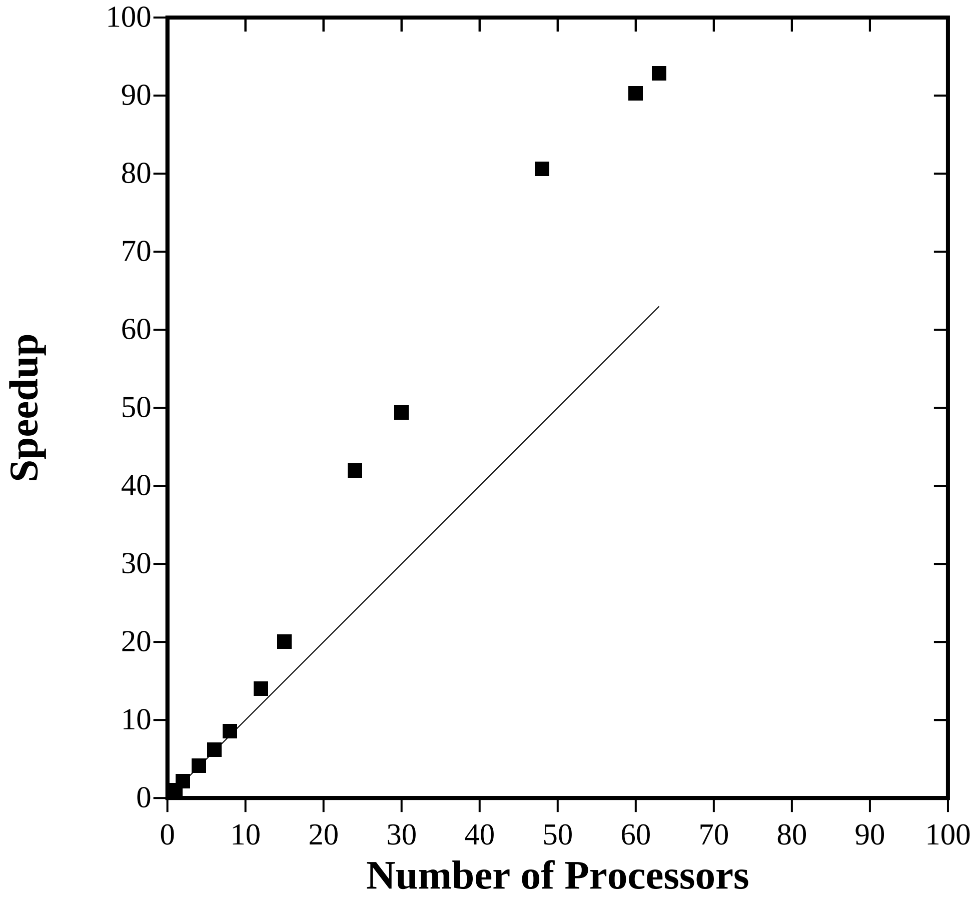

Figure 4: Performance curve on Origin 3000

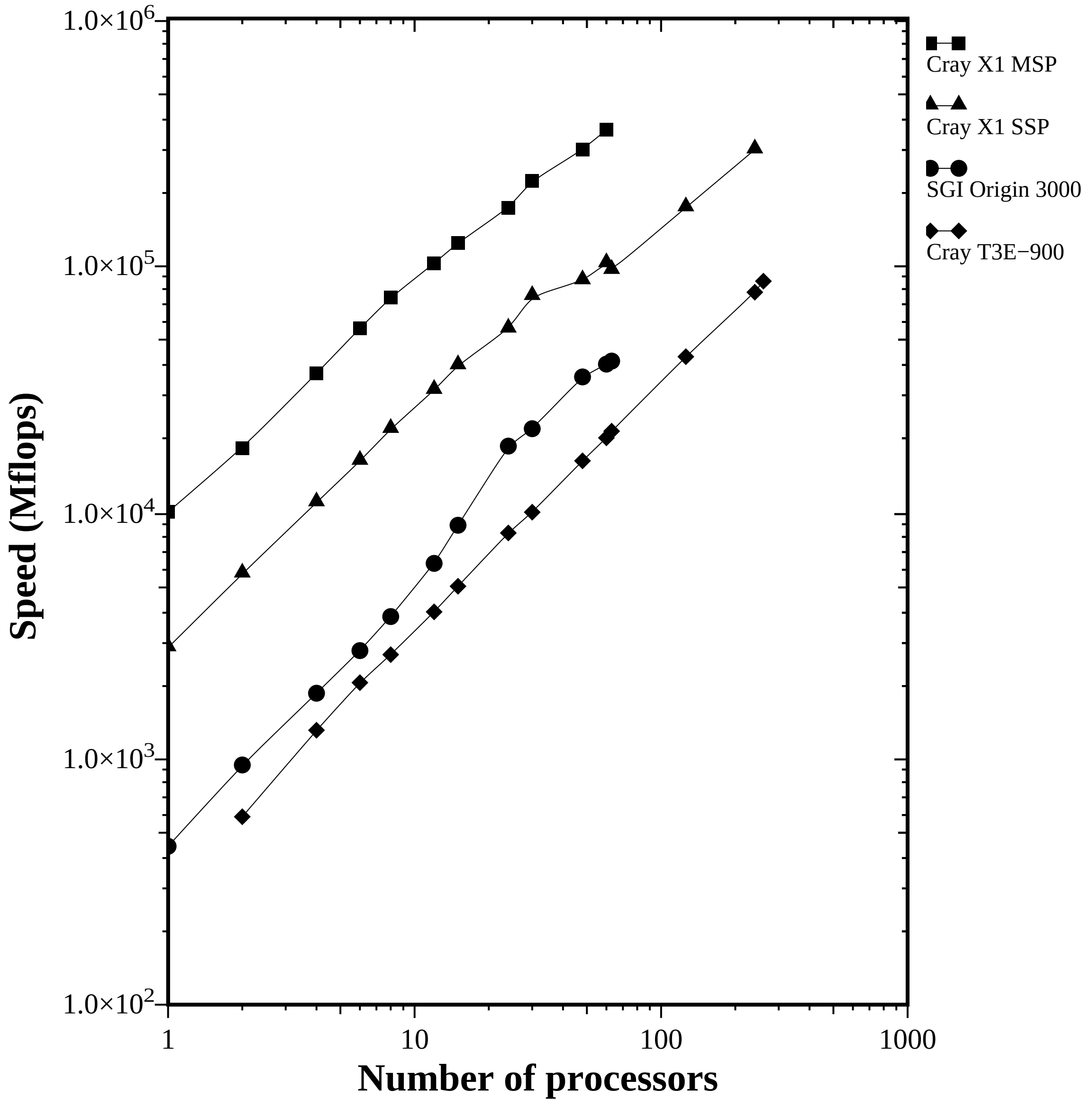

Figure 5: Computational Speeds on Cray X1, Cray T3E and Origin 3000