INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/854-15

College on Computational Physics

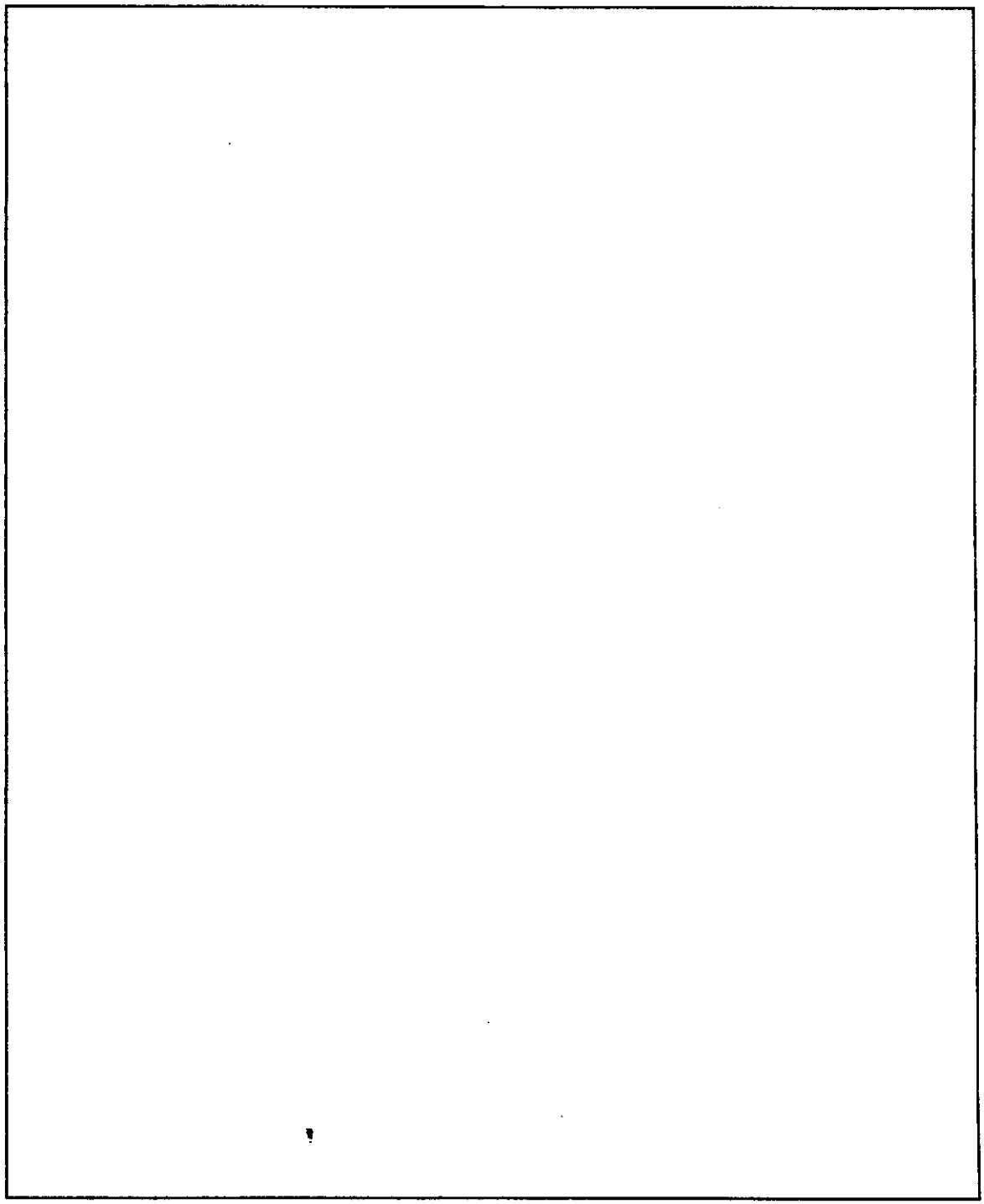
15 May - 9 June 1995

High-resolution lattice-Boltzmann computing on the IBM SP1 scalable parallel computer

F. Massaioli

Università "la Sapienza"
Rome, Italy

COMPUTERS IN PHYSICS



AMERICAN INSTITUTE OF PHYSICS

OFFPRINT SERIES

This article originally appeared in
COMPUTERS IN PHYSICS

High-resolution lattice–Boltzmann computing on the IBM SP1 scalable parallel computer

Giovanni Punzo
IBM ECSEC, V.le Oceano Pacifico 171, I-00144 Roma, Italy

Federico Massaioli
CASPUR, c/o Università di Roma 'La Sapienza', P.le Aldo Moro 5 I-00185 Roma, Italy

Sauro Succi
IBM ECSEC, V.le Oceano Pacifico 171, I-00144 Roma, Italy

(Received 14 April 1994; 29 June 1994)

We discuss the implementation of the Lattice–Boltzmann method, a relatively new technique for computational fluid dynamics, on the IBM SP1 Scalable Parallel processor. Sustained rates of almost 4 Gflops on high-resolution simulations with up to nearly 800 million of degrees of freedom are demonstrated.

INTRODUCTION

In the recent years, distributed multiprocessor architectures obtained by pooling together powerful RISC processors linked via optical switching networks have gained an increasing popularity within the arena of parallel computing.

The success of this architecture is due to the fortunate conspiration of two technological factors: the impressive rise of raw computer power afforded by RISC technology, and the wide acceptance of open-system software Unix-based environments. A typical forerunner of this kind of architectures is the cluster of workstations, linked via standard local area networks, such as Ethernet, Token-Ring, FDDI, and other interconnecting media.[1]

More recently, the cluster concept has evolved into a truly integrated, scalable parallel (SP) architecture, the IBM 9076-SP1 machine (9076-SP1 is a trademark of International Business Machines Corporation), consisting of a rack of RISC processors connected via an optical switching network.[2]

The key point in the SP approach is the fact that each processor is "per-se" a self-standing computer (workstation) with independent I/O and processing capabilities. This means that the programmer can develop and test his/her parallel applications on a single workstation completely "off-line," i.e., by emulating parallel execution via the standard Unix multiprocessing functions with no need to access the physical nodes of the parallel machine. The application can subsequently be taken to the fully configured parallel machine in a fairly smooth fashion once ready for production runs. This offers a significant advantage in terms of man-effort savings for both scientific and industrial end-users.

In this article we present a series of performance data which prove that by use of the Lattice–Boltzmann method, a recent technique for computational fluid dynamics (CFD) especially suited to parallel computers, very high-resolution CFD problems involving over a half a billion (0.5G) degrees of freedom can be executed on the IBM 9076 Scalable Parallel system at sustained rates of almost 4 Gflop.

The purpose of this article is not to break any world record or surpass previous high-resolution simulations with Boolean automata,[3] finite differences,[4] or spectral methods,[5,6] but simply to show that the SP strategy is quite cost-effective and very convenient to adopt over a wide range of resolutions.

I. DESCRIPTION OF THE LBE METHOD

The Lattice–Boltzmann equation (LBE) is a direct method to solve the Navier–Stokes equations on a digital computer. LBE is rooted in Boolean lattice gas techniques, a kind of "minimal" molecular dynamics based on the observation that the large-scale dynamics of fluid flow is largely independent of the details of the underlying microdynamics. This suggests that, to numerically integrate the differential equations describing the motion of a fluid, it may be convenient to use a population of microvariables ("particles") whose microdynamics can be freely adjusted to match the Navier–Stokes equations on a macroscopic scale.[7]

The LBE method takes this approach one step-forward towards the macroscopic world, from the molecular to the kinetic level, by replacing the Boolean microdynamical variables with their corresponding floating point expectation values. This move, while maintaining the locality in space and time of the evolution rules, which are key to the amenability to parallel computing, offers three main advantages: a greater suitability to present-day computing architectures (increasingly faster on the floating point side); a wider degree of latitude in choosing the details of the evolution rule; a reduction of the separation in scale between the microworld and the macroworld (i.e., the averaging operation on a suitable region of the microdynamical lattice needed in Boolean simulations to remove statistical noise is no longer necessary).

Some brief highlights of the method will be given here, for the sake of completeness. A more detailed description can be found in Refs. 8–12.

The method is based on the definition of a discrete lattice of points. On each site of the lattice, b populations

$\{N_i\}$ are defined, corresponding to b vectors $\{\mathbf{c}_i\}$, joining neighboring points on the lattice. Populations evolve in (discrete) time according to a propagation and collision rule:

$$N_i(\mathbf{x}+\mathbf{c}_i,t+1)=N_i(\mathbf{x},t)+\sum_j A_{i,j}[N_j(\mathbf{x},t)-N_j^{eq}(\mathbf{x},t)], \tag{1}$$

where $A_{i,j}$ is the collision matrix, and N_i^{eq} are the values of the populations at local equilibrium.

By a suitable choice of the lattice, i.e., the $\mathbf{c}_i$, $A_{i,j}$, and N_i^{eq}, it can be shown that the following linear combinations,

$$\rho(\mathbf{x},t)=\sum_i N_i(\mathbf{x},t)$$

$$\mathbf{J}(\mathbf{x},t)=\sum_i \mathbf{c}_i N_i(\mathbf{x},t)$$

$$=\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t)$$

behave according to the Navier–Stokes equations for an incompressible fluid.

The most common boundary conditions (e.g., free-slip, no-slip, periodic) can be easily implemented by suitably modifying the propagation phase. This is obtained by wrapping the physical lattice with a one site thick border or layer, taking part to the propagation step and ignored in the collision step. The population values of those additional sites are aptly set to obtain second-order accurate boundary conditions on virtual points lying midway between the border sites and the external lattice sites.

In a similar way, grossly irregular geometries (i.e., porous media) can be studied, by replicating the trick on "solid" sites in the lattice bulk. These situations are more readily accounted for by LBE than by traditional computational techniques, because only nearest-neighbor communication is required even for describing diffusion processes usually described by second-order difference operators.

The primary advantage of the LBE approach relates to its amenability to parallel computing. This is readily appreciated by noting that the collision phase, the one accounting for most of the CPU time of the application, is completely local in configuration space.

II. SHORT SURVEY OF THE SP1 SYSTEM

The SP1 is a parallel computer designed to make the best use of IBM's RISC technology combined with a high-speed switch for interprocessor communication.[13] Special features of this machine are

(1) large memory per node (64-256MB),
(2) high-performance nodes (125Mflops peak),
(3) high-performance switch (0.5 μs hardware latency, 40 MB/s peak bidirectional bandwidth),
(4) high I/O bandwidth off-nodes (7MB/s), and
(5) full Unix functionality on each node.

The base model starts with eight processors nodes, with a modular growth path up to 64 nodes, corresponding to a peak system performance from 1 to 8Gflops and an aggregated capacity of 0.512–16GB. Two transport layers are available for use with the high-performance switch. The first is IP (Internet Protocol), providing enhanced performance to code written using Unix sockets. The second is an IBM's proprietary transport layer that is used by MPL (message passing library), the IBM's proprietary library, and IBM AIX PVMe, the IBM version of the public domain PVM library, developed at the IBM European Center for Scientific and Engineering Computing, Rome, Italy.[14]

(6) The SP1 operates under the IBM AIX Parallel Operating Environment (POE), which provides support for parallel application development and execution.[15,16]
(7) Parallel programming on SP1 is based on the message-passing paradigm which can be implemented either via MPL, PVMe, or via other popular communication libraries such as PVM, Express, or Linda.

III. SP1 IMPLEMENTATION OF THE LBE METHOD

The actual LBE-MPL implementation is based upon a two-dimensional domain-decomposition method. This choice could upset the scalability of the application for massively parallel configurations (thousands of processors or more), but represents a good compromise between latency and bandwidth considerations within the range of processors examined in this study (1–128).

The global domain is a parallelepiped containing $N_x \times N_y \times N_z$ sites along the three coordinate directions.

The P processors, labeled from 0 to $P-1$, are logically mapped onto the global lattice as a rectangular mesh $P_y \times P_z$, each processor owning a parallelepipedal slice $n_x \times n_y \times n_z$, with $n_x = N_x$, $n_y = N_y/P_y$, and $n_z = N_z/P_z$.

As an example, a 3×4 processor configuration is mapped as follows:

	9	10	11
↑	6	7	8
z	3	4	5
	0	1	2

$y \rightarrow$

Diagram 1. Processor numbering in the y–z plane for a 3×4 processor configuration

Within this domain-decomposition strategy, the collision phase is embarrassingly parallel in that no communication is required between neighboring processors. The propagation phase, however, requires nearest-neighbor communication along the 18 directions defined on the discrete lattice (6 nearest-neighbors plus 12 next-to-nearest neighbors). As a result, despite its conceptual transparence,

this application sets a real challenge on the interconnecting network for a high-rate data traffic between different processors has to be sustained.

In the streaming phase each processor must communicate with its neighbors, whose number (3–8) depends on the processor location within the mesh. Communications occur across the four faces parallel to the x axis of the subdomains; whenever one or two of these faces lie on a boundary of the global domain, they are treated accordingly. Our starting point was a thoroughly modular and carefully optimized FORTRAN-VS code, running on scalar, vector or shared-memory parallel machines with very little modifications. Keeping the SP1 version coherently aligned with this policy was a must for this project.

We achieved this goal by simply extending the trick of the additional wrapping layer, described above for the boundary conditions. In line with the SP strategy, every processor runs a straightforward LBE serial code, with the exception that the routine responsible of setting the populations of the additional layer so as to satisfy the boundary conditions, now, wherever the border is internal to the global domain, takes the right values from the neighboring processors. The net result is that the informations from the neighboring nodes are simply used as boundary conditions for the local subdomain.

Besides the changes in the boundary conditions routine, some minor modifications had to be made in the I/O procedures and in the routines in charge of tracing the time behavior of selected global quantities. All accounted for, the extra work amounted to less than 10% of the original code, and was accomplished (debugging included) in approximately a 2 week time using a software emulator running on a single workstation. The final code ran smoothly without problems on the real SP1 machine.

While organizing the interprocessor communications, we had three objectives in mind. First, "parallelize" the communication phase to the highest possible extent, i.e., let as many processors as possible to communicate concurrently.

Second, mask the communication latency, i.e., the time needed to establish a communication between two processors, which may severely degrade the efficiency of short and/or frequently interrupted transmissions.

Third, get rid of the detrimental impact of the system buffering activity needed to deal with hanging messages (send operations awaiting for the corresponding receives) behind the scene.

Owing to the fact that the SP1 switch allows any disjoint pair of processors to communicate concurrently, the solution is to organize the communication pattern in terms of a set of mutually disjoint sender/receiver pairs ("communication dimers").

To avoid contentions for communications with a given processor, all pairs should remain tied-up for the same time lapse, i.e., they should transmit the same amount of data and eventually synchronize only at completion of the transmission activity.

To achieve this goal, we organized the communications steps so that the data movement occurs first in the y, then in the z direction. This ensures that each "bound" pair exchanges the same amount of data. Mutual disjointness can be achieved by arranging the communication according to a "two-color" checkerboard pattern whereby odd nodes first talk to even nodes and vice versa.

For example, in the 3×4 partition above, the pairs in the communications along the z axis can be selected according to their parity: first, each even-numbered node "talks" with the upper neighbor node, then each odd-numbered node talks to the upper neighbor, and so on. This "parlor game" technique can be quite effective in leveling off load unbalances.

Care was also taken to avoid short messages which would hit the machine latency. For example, diagonal communications (say 0⇔4) are obtained implicitly as a combined effect of horizontal (3←4 and 0→1) and vertical (0←3 and 1→4) communications.

At this stage, it is worth mentioning that the implementation of the above strategy was greatly facilitated by use of the visualization tool (VT), i.e., the component of POE which permits to monitor and visualize application as well as system activity. With VT the programmer can visualize message passing events between processors, type, duration and connectivity of communication events, CPU utilization of processor nodes, load balance, and many other indicators of the parallel job activity. A qualitative example of the group of displays, or "views" provided by VT is shown in Fig. 1.

IV. PERFORMANCE MODEL

In order to gain a quick grasp on the parameters controlling the parallel efficiency, we have developed a simple analytical performance model. This model is deliberately discarding real-life effects, such as load-unbalance and size-dependent efficiency of memory access. While the former assumption proved basically correct upon experimental verification, the latter does only partially, as it will be discussed in the sequel.

The amount of data to be exchanged by a given processor is given by

$$M = 2\times 4\times 6(2n_x n_y + 2n_x n_z)(\mathrm{B}), \tag{2}$$

where 2 stems from bidirectional flow (send/receive), 4 is the number of B/variable, 6 is the number of exchanged variables per grid point, and the factor within brackets is the number of surface grid points.

The corresponding amount of calculations ("grain") is given by:

$$G = 957\ \ n_x n_y n_z(\mathrm{Flops}), \tag{3}$$

where we have neglected the processing time required by the routine move (18 $n_x n_y n_z$ memory references).

The communication time associated with the message-passing is given by

$$T_{\mathrm{com}} = M/V_{\mathrm{com}} + 24T_0 + 2N_s T_0 \ln P, \tag{4}$$

where V_{com} is an effective peer-to-peer communication rate and T_0 is the machine latency (time to set-up the communication) and N_s is the number of synchronization points per time step (two in our case). The factor 24 comes from the fact that each discrete speed in the lattice gives rise to a separate message.

Since the computing time is given by G/V_{cal}, the parallel efficiency, defined as

$$\eta = \frac{1}{1 + T_{\mathrm{com}}/T_{\mathrm{cal}}}$$

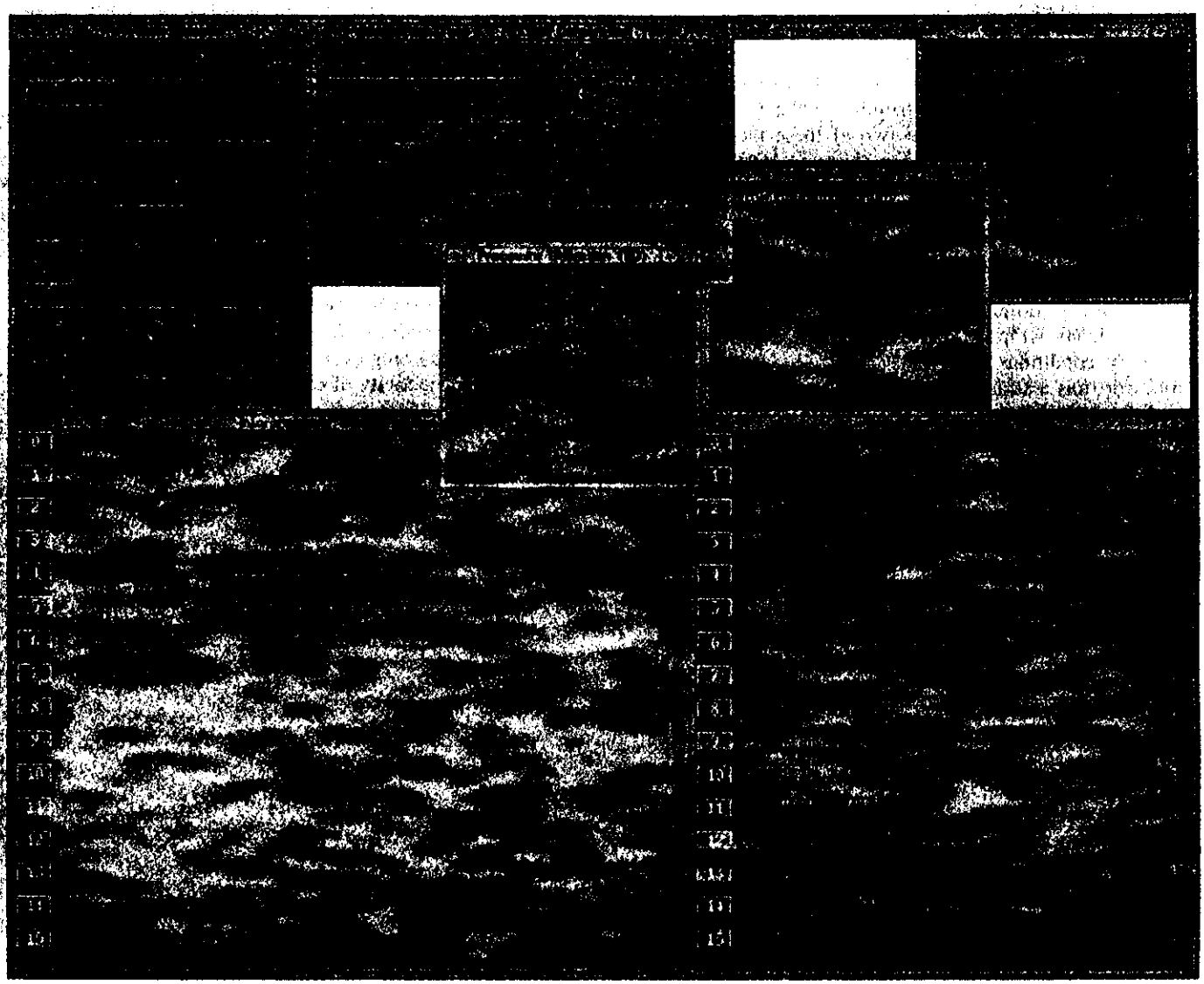


Figure 1. A typical group of displays ("views") of the visualization tool.

reads as follows:

$$\eta(P_y,P_z) = \frac{1}{1+(V_{cal}/V_{com})[a(n_y^{-1}+n_z^{-1})+\lambda_0 n^{-1}(b+c\ \ln P)]}, \quad (5)$$

where $n=n_x n_y n_z$, $\lambda_0=V_{com}T_0$ is the minimal message length below which latency is felt, and $a=96/957$, $b=24/957$, $c=4/957$.

Latency and synchronization overheads are no concern for the efficiency as long as the number of processors is low enough to fulfill the following inequality

$$\lambda_0\left(\frac{1}{4}+\frac{\ln P}{24}\right)<n_x(n_y+n_z). \quad (6)$$

Typical values for this application are V_{cal}/V_{com} ~6(Flop/B) as deduced from experimental measurements, T_0~100 μs, λ_0~0.5 KB. With these values, it is readily checked that latency and synchronization issues are largely immaterial for the results discussed in this article.

As a result, the analytical model indicates that the application should scale up to a number of processors of the order of the square size of the global computational domain, or, equivalently like the total number of grid points to the power 2/3. This is precisely what one expects from a two-dimensional partitioning of a three-dimensional domain.

V. PERFORMANCE DATA

Several grid sizes were run, ranging from a 32×48×64 to 256×512×256 grid sizes. The raw performance in Gflops for the following representative grids ("S," "M," "B," "G1," "G2," "G3" stand for small, medium, big, gigal-2-3, respectively)

(S) 32×48×64 (9MB),

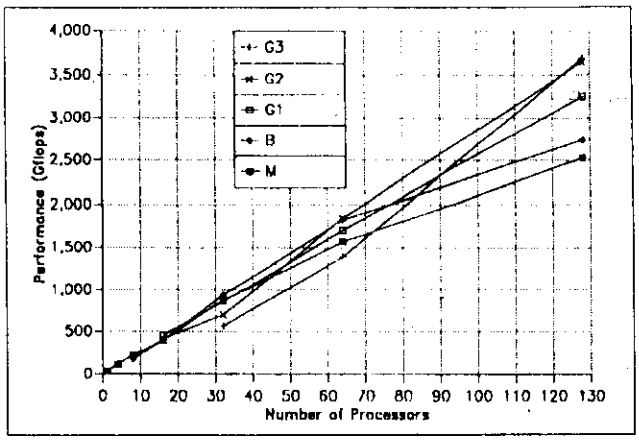


Figure 2. Raw performance (Gflops) of the cases (S) to (G3) as a function of the number of processors.

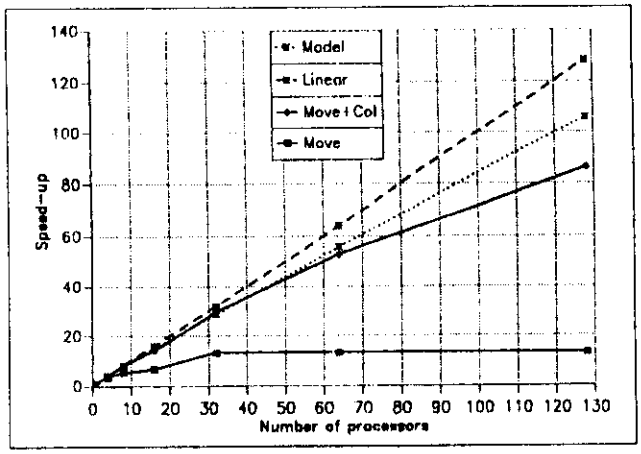


Figure 3. Speedup as a function of the number of processors for the case (M) (64×96×128). The dotted line corresponds to the prediction of the analytical model with T_0=100 μs and V_{com}=5MB/s.

(M) 64×96×128 (72MB),
(B) 128×192×256 (576MB),
(G1) 256×192×256 (1.152GB),
(G2) 256×256×256 (1.536GB),
(G3) 256×512×256 (3.072GB)

are presented in Fig. 2.

Run (M) can be accommodated in-core within the central memory of a single node (128MB) thus allowing our study to encompass the full range of available processors, from 1 to 128, without being obscured by paging issues.

For the same reason, runs (B), G1-2, G3 have not been performed for less than 8, 16, and 32 processors, respectively.

From Fig. 2, we see that there is no single best performer across the whole range of processors, but who does best depends on the number of processors. In principle, we identify three regions

(L) up to 32 processors,
(M) from 32 to 64 processors,
(H) from 64 to 128 processors.

In region (L) (low parallelism) the smaller cases perform better; this is likely to result from better memory access due to the smaller size of the problem. As the number of processors is raised [region (M), for moderate parallelism] the intermediate cases (B) and (G1) take over while the largest jobs are still outperformed by the smaller ones. As the number of processors is further increased, the largest run tends to win the race. In any event, the pleasing remark is that if the number of processors is sufficiently high [region (H)], the largest jobs prove capable of extracting basically the same percentage of peak performance of the single processor (about 25%). This leads to more than 3 Gflops performance for all but the smallest jobs and, more importantly, shows that performance gain is not about to cease even moving to the very high region (thousands of processors) of parallelism. This bodes well for future larger installations with a few hundred of nodes.

In order to gain a deeper insight into the code performance we analyze the loss of efficiency of case (M) in the limit of a high number of processors. To this purpose, in Fig. 3 we present the speedup $S=\eta P$ as a function of the number of processors. The labels move refers to the move routine, while move+col refers to the sum of the two. The move time includes both free-streaming in the interior of the domains and data exchange across the interdomain boundaries.

This figure clearly highlights that the collision phase scales linearly while move saturates around a factor 13 for more than approximately 30 processors. As a matter of fact, while most results are relatively well predicted by the analytical model with $T_0=100$ μs and $V_{com}=5$MB/s, the result 86/128 is definitely not.

A careful analysis revealed that this is due some constant overhead in the message-passing stage which saturates at about 40 ms/step irrespective of the number of processors.

This is probably due to processor contentions generated by a nonoptimal message-passing scheduling strategy. As of today, message passing is organized in two passes: first even processors talk to the odd-numbered ones and vice versa. With the processor numbering indicated in Diagram 1, "dimerization" (simultaneous communication of disjoint pairs of processors) is achieved only along the y direction, while transmissions along the z direction may generate contentions because neighbors along z have the same parity if P_x is an even number. Work is in progress to implement a fully red-black message-passing pattern, which should minimize processor contention under heavily fragmented traffic conditions thus curing this flaw.

Besides the possibility to cut turnaround time for CPU-bounded problems, parallel computing is expected to offer significant advantages also in terms of job-upsizing, i.e., the capability of tackling problems whose size would be unaccessible to serial computers. Actually, this is really what parallel computing should be used for. Roughly speaking, cutting turnaround time points to higher productivity and is consequently a primary issue for the industrial community whereas job-upsizing is mostly hunted for within the academic community.

This points to the issue of scalability. Scalability refers to the ability of a given platform to sustain continued returns of performance in proportion to the number of processors employed. Mathematically, this means that the efficiency stays almost constant over a wide range of processors (scaling region). Obviously, scalability cannot hold indefinitely (apart from completely noncommunicative applications), in the sense that one is usually presented with

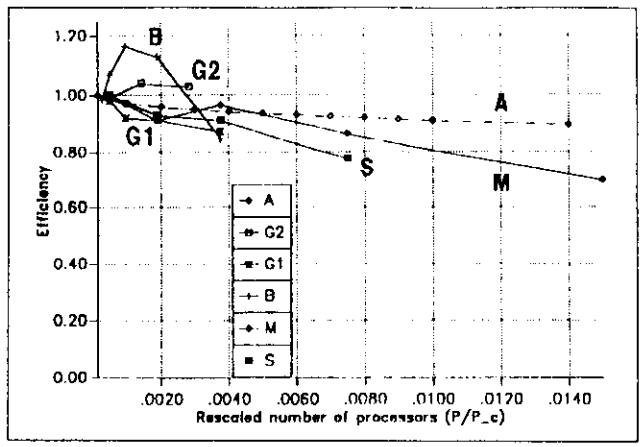


Figure 4. Efficiency as a function of the scaled number of processors $x=P/P_c$ for the set of cases (S) to (G3). The curve A represents the universal function $\eta(x)=(1+x^{1/2})^{-1}$.

an upper threshold for the number of processors beyond which the efficiency drops rapidly down. The existence of such a natural scale, say P_c, implies that the efficiency can be recast in a self-similar form as

$$\eta(P,N)=\Psi[P/P_c(N)], \tag{7}$$

where Ψ is a universal function, while the details of the platform and the application are lumped into the specific expression of P_c as a function of N. In the case of our analytical model, making abstraction of latency and sync overheads, and assuming $P_x=P_y=P^{1/2}$, one derives

$$\Psi(x)=\frac{1}{1+x^{1/2}};$$

$$P_c\sim N_yN_z/(2\times 6/10)^2=N_yN_z/1.44. \tag{8}$$

This formula is tested against experimental data, as shown in Fig. 4.

From this figure we see that scalability is well fulfilled by runs (S), (M), (G1), while (B) and (G2) display "bumps" in the region of low parallelism. This anomaly can be traced again to the size-dependence of the efficiency of memory access, a typical feature of hierarchical-memory machines.

In fact, the efficiency is defined as

$$\eta(P,P_0)=\frac{\text{Speed}(P)}{\text{Speed}(P_0)}\frac{P_0}{P}, \tag{9}$$

where P_0 is the minimum number of processors for which the job has been run. Clearly, if P_0 is so small as to generate paging, which occurs at approximately more than 90 MB per processor, superlinear speedup (a finite-size anomaly not encompassed by scaling models) results as soon as P crosses the threshold beyond, which paging is no longer needed. For the set of jobs examined here the page-free conditions are fulfilled for any number of processors only by runs (S) and (M). In fact, these runs exhibit a good match with scaling predictions on a wide range of values of P/P_c.

VI. PHYSICAL RESULTS

The work described in this article is part of an international cooperation aimed at the understanding, both theoretical and experimental, of the convective phenomena taking place in a Rayleigh–Bénard cell, namely a fluid-filled tank heated from below.

While speculation in this field dates back to the 60s, renewed interest has been fueled by an experiment by now famous in the field.[17] The situation is very appealing for computational physicists, because the computational effort needed to study the problem numerically is (maybe just a little) more than the computational power afforded by today's computers. This puts a high premium on carefully optimized computational experiments on advanced architectures.

Our group has worked for more than 2 years on this subject, and produced new results concerning the probability distribution functions of the temperature fluctuations in the cell, the scaling of correlations functions, the anomalies arising from intermittence, as obtained from two-dimensional simulation and confirmed by experimental results.[18] An interesting by-product of this work was the discovery of a generalized form of scaling exhibited by fluid turbulence, which was named extended self-similarity (ESS).[19]

The main content of ESS is that structure functions of velocity field fulfill an algebraic scaling law of the form

$$S_p\sim S_q^{\alpha(p)/\alpha(q)}, \tag{10}$$

where the velocity structure functions are defined as

$$S_p(r)\equiv\langle|\delta v(r)|^p\rangle. \tag{11}$$

Here $\delta v(r)$ is a typical velocity fluctuation at the scale r and brackets denote statistical ensemble averaging. The coefficients $\alpha(p)$ are known as scaling exponents and measure the degree of intermittency, i.e., short-scale spottings, of the turbulent flow.

To appreciate the innovative content of ESS, it is useful to remind that scaling, as implied by the celebrated Kolmogorov theory (K41), is a property commonly attributed exclusively to fully developed turbulent flows, i.e., flows whose dynamics are dominated by inertial over dissipative effects. This dominance is expressed via the Reynolds number, defined as

$$\text{Re}=\frac{UL}{\nu}, \tag{12}$$

where U is a typical velocity scale of the flow, L is a typical macroscopic scale, and ν is the molecular viscosity of the fluid. Fully developed turbulent (FDT) flow is characterized by the condition

$$\text{Re}\gg 1. \tag{13}$$

According to Kolmogorov theory, FDT flows are self-similar, i.e., they exhibit a scaling dependence of the velocity structure of functions $S_p(r)$ on the space separation r:

$$S_p(r)\equiv\langle|\delta v(r)|^p\rangle=r^{\alpha(p)}. \tag{14}$$

Clearly, Eq. (14) implies Eq. (10), while the reverse is not true, whence its definition of ESS.

The Eq. (14) represents a true challenge to the computational physicist. To understand why, let us remind that the dynamical evolution of a fluid flow at a given Reynolds number Re entails about $\text{Re}^{9/4}$ active degrees of freedom.[20]

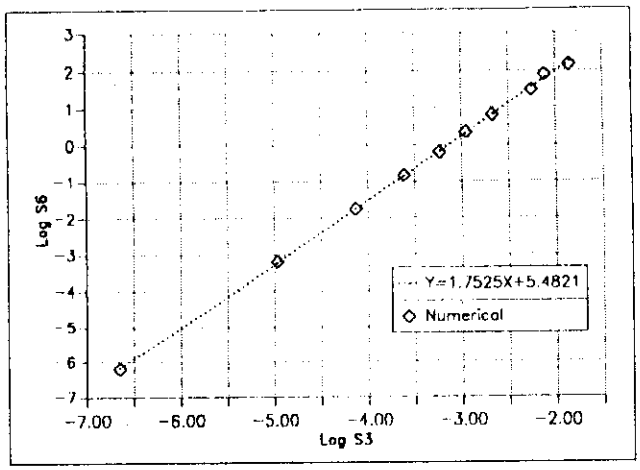


Figure 5. Log–log plot of the sixth-order structure function S_6 vs S_3. The straight line is a best fit corresponding to a slope of 1.75.

Assuming that each degree of freedom can be represented by a single grid point, this means that the amount of memory required to simulate a flow at Reynolds Re is given by $M=\mathrm{Re}^{9/4}$ and the computational work $W=\mathrm{Re}^3$. Once mapped to present-day computer architectures, this translates into an upper limit of about $N=512$,[6] which means Re~1000. The problem is that, at such low values, a numerical detection of the scaling regime is hardly feasible because the scaling region is too small to ensure an accurate evaluation of the scaling exponent.

Now, the innovative content of ESS rests mainly with the two following points:

(1) Equation (10) holds even at moderate Reynolds numbers $\mathrm{Re}\sim O(100)$.
(2) The scaling region, over which Eq. (10) holds, is much wider than the scaling region associated with Eq. (14).

A typical signature of ESS is displayed in Fig. 5 (from Ref. 18) in which S_6 is log-plotted as a function of S_3. From this figure, a strikingly wide linear scaling regime is apparent which would have never shown up had we plotted S_3 and S_6 separately as a function of r.

Whether ESS bears a profound physical meaning possibly associated with some hitherto overlooked "hidden" symmetry of the fluid equations, is still an open question.

VII. CONCLUSION

We believe that the work presented in this article represents a successful case study on how scalable parallel and distributed computing can contribute to the advancement of physical science.

The most immediate result of this article is that use of Lattice–Boltzmann schemes on the IBM Scalable Parallel platform offers Gflops performance today.

Most important, scalability arguments indicate that significant further increase of performance can be achieved by moving to a few-hundred processors. These considerations, combined with the recently announced availability of enhanced SP platforms based on IBM POWER-2 technology (faster CPU, more memory per node, higher bandwidths, and lower latencies), justify the expectation of tens of Gflops for Lattice–Boltzmann flow simulations by 1994.

ACKNOWLEDGMENTS

P. Sguazzero is kindly acknowledged for invaluable suggestions and useful discussions. F.M. is grateful to IBM ECSEC where the development work as well as some early simulations described in this article were performed. Remote availability of the 128-node SP1 platform at the Center for Scalable Computing Solutions of IBM T. J. Watson Research Center, which was key to scalability assessments, is kindly acknowledged. Finally, the authors wish to acknowledge a nonanonymous referee for many valuable comments.

REFERENCES

1. J. Dongarra, Comput. Phys. **7**, 166 (1993).
2. IBM 9076 Scalable POWER Parallel Systems, General Information, GH26-7219 (1992).
3. G. A. Kohring, Int. J. Mod. Phys. C **2**, 755 (1991).
4. E. Oran and J. Boris, Comput. Phys. **7**, 523 (1993).
5. E. Jackson, E. Jackson, and S. Orszag, J. Sci. Comput. **6**, 1 (1991).
6. S. Chen and X. Shan, Comput. Phys. **6** (1992).
7. U. Frisch, D. d'Hummières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, Complex Systems **1**, 649 (1987).
8. F. J. Higuera and S. Succi, Europhys. Lett. **8**, 517 (1989).
9. F. J. Higuera, S. Succi, and R. Benzi, Europhys. Lett. **9**, 345 (1989).
10. R. Benzi, S. Succi, and M. Vergassola, Phys. Rep. **222**, 3 (1992).
11. B. Boghosian, Comput. Phys. **4** (1989).
12. H. Chen, Comput. Phys. **7** (1993).
13. P. A. Franaszek, C. J. Georgiou, and A. N. Tantawi, IBM J. Res. Dev. **35** (1991).
14. IBM AIX PVMe, User's Guide and Reference, SH23-0019 (1994).
15. IBM AIX Parallel Environment, Operation and Use, SH26-7230 (1993).
16. IBM AIX Parallel Environment, Programming Reference, SH26-7228 (1993).
17. B. Castaing, G. Gunaratne, F. Heslot, L. Kadanoff, A. Libchaber, S. Thomae, X. Wu, S. Zaleski, and G. Zanetti, J. Fluid Mech. **204**, 1 (1989).
18. R. Benzi, R. Tripiccione, F. Massaioli, S. Succi, and S. Ciliberto, Europhys. Lett. (in press).
19. R. Benzi, S. Ciliberto, R. Tripiccione, F. Baudet, F. Massaioli, and S. Succi, Phys. Rev. E **1**, R29 (1993).
20. E. M. Karniadakis and S. Orszag, Phys. Today **46** (3) (1993).

Dimensional perturbation theory on the Connection Machine

Timothy C. Germann and Dudley R. Herschbach
Department of Chemistry, Harvard University, Cambridge, Massachusetts 02138

Bruce M. Boghosian
Center for Computational Studies, Boston University, Massachusetts 02215

(Received 13 December 1993; accepted 16 February 1994)

A recently developed linear algebraic method for the computation of perturbation expansion coefficients to large order is applied to the problem of a hydrogenic atom in a magnetic field. We take as the zeroth order approximation the $D \to \infty$ limit, where D is the number of spatial dimensions. In this pseudoclassical limit, the wave function is localized at the minimum of an effective potential surface. A perturbation expansion, corresponding to harmonic oscillations about this minimum and higher order anharmonic correction terms, is then developed in inverse powers of $(D-1)$ about this limit, to 30th order. To demonstrate the implicit parallelism of this method, which is crucial if it is to be successfully applied to problems with many degrees of freedom, we describe and analyze a particular implementation on massively parallel Connection Machine systems (CM-2 and CM-5). After presenting performance results, we conclude with a discussion of the prospects for extending this method to larger systems.

INTRODUCTION

The spatial dimension has long been treated as a variable parameter in analyzing critical phenomena and in other areas of physics.[1] However, only in the past ten years has this concept been extensively applied to atomic and molecular systems, particularly to develop dimensional scaling methods for electronic structure.[2] The motivation for this unconventional approach is that the Schrödinger equation reduces to easily solvable forms in the limits $D \to 1$ and/or $D \to \infty$. When both limiting solutions are available, interpolation in $1/D$ may be used to approximate the physically meaningful $D=3$ result; this has yielded excellent results for correlation energies of two-electron atoms[3] and for H_2 Hartree–Fock energies.[4] Alternatively, if the $D \to \infty$ solutions are available for both the problem of interest and a simpler model problem (e.g., Hartree–Fock) for which $D=3$ results are easier to calculate, the latter may be used to "renormalize" some parameter (e.g., nuclear charge). Then the $D \to \infty$ solution with the renormalized parameter may give a good approximation to the $D=3$ solution with the actual parameter value.[5]

Our work deals with another widely applicable dimensional scaling method, a perturbation expansion in inverse powers of D or a related function, about the solution for the $D \to \infty$ limit. That limit is pseudoclassical and readily evaluated, as it reduces to the simple problem of minimizing an effective potential function.[2] For large but finite D, the first-order correction accounts for harmonic oscillations about this minimum, and higher-order terms provide anharmonic corrections. Dimensional perturbation theory has been applied quite successfully to the ground[6] and some excited states[7] of two-electron atoms and to the hydrogen molecule–ion[8] using a "moment method" to solve the set of perturbation equations. However, this method is not easily extended to larger systems. It also requires a different program for each eigenstate, and does not directly provide an expansion for the wave function.

A recently developed linear algebraic method has overcome these shortcomings.[9] This is conceptually quite simple, so it can easily be applied to systems with any number of degrees of freedom. It permits calculation of ground and excited energy levels using a single program and the wave function expansion coefficients are directly obtained in the course of computing the perturbation expansion for the energy. This method has thus far been applied to central force problems, including quasibound states for which the complex eigenenergy represents both the location and width of the resonance.[10]

The linear algebraic version of dimensional perturbation theory is also well suited to parallel computation. Here we demonstrate this for a prototype problem with two degrees of freedom, the hydrogen atom in a magnetic field. This system has received much attention; it exhibits chaotic behavior and poses difficulties that have challenged many theoretical techniques. Most theoretical approaches treat either the magnetic field or the Coulomb potential as a perturbation and therefore work best near either the low- or high-field limit, respectively. However, the leading terms of a perturbation expansion in inverse powers of D include major portions of the nonseparable interactions in all field strengths. The efficacy of methods equivalent to the $1/D$ expansion has been demonstrated for the hydrogen atom in a magnetic field[11] and for kindred problems with an electric field or crossed electric and magnetic fields,[12] although not in formulations suited to parallel computation. The present paper is devoted solely to implementing of the linear algebraic method on the Connection Machine, and to evaluating the performance of the computational algorithm as well as prospects for treating systems with more degrees of freedom. Numerical results for the ground and several excited states over a wide range of field strengths will be presented in a separate paper.[13]