INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 24, 939–951 (1997)

IMPLEMENTATION OF VORTEX FILAMENT METHODS ON PARALLEL MACHINES WITH DISTRIBUTED ADAPTIVE DATA STRUCTURE

Y. L. SHIEH,[1] J. K. LEE,[1] J. H. TSAI[2] AND C. A. LIN[2]*

[1] *Department of Computer Science, National Tsing Hua University, Hsinchu 30043, Taiwan*
[2] *Department of Power Mechanical Engineering, National Tsing Hua University, Hsinchu 30043, Taiwan*

SUMMARY

This paper addressed the implementation of vortex filament methods on parallel machines with distributed memory to simulate a three-dimensionally evolving jet. Vortical structure developments due to Kelvin–Helmholtz instability of the axially perturbed jet are also examined. The implementation is conducted in a single-programme multiple-data (SPMD) environment and the parallelism is focused on issues of data distribution, efficient support of parallel I/O and overlapping of communications with computations. In addition, since the number of segment markers in a filament is dynamically growing according to the requirement of numerical accuracy, a novel packet-oriented data structure is proposed not only to partition filament segment markers among distributed processors but also to support dynamical load balancing at run time. This work is the first to apply packet-oriented structures to implement a parallel vortex filament method. Experimental results indicate performance improvement from 1·5 to 2·6 times over static schemes on nCUBE2, DEC Alpha and IBM SP2 by incorporating the proposed scheme with packet-oriented structures. © 1997 by John Wiley & Sons, Ltd. Int. j. numer. methods fluids 24: 939–951, 1997.

(No. of Figures: 18. No. of Tables: 4. No. of Refs: 10.)

KEY WORDS: vortex filament method; parallel machines

1. INTRODUCTION

Jet flow involves various flow motions, Kelvin–Helmholtz instability, roll-up, pairing and ultimately breakdown.[1] As the jet flow injects into the circumferential flow, the viscous effect induces a shear layer to balance the velocity differences between the two flows. If the shear layer is thin enough, it can be regarded as a vortex sheet. Subjected to some wavy perturbations in the streamwise direction, Kelvin–Helmholtz instability is sequentially excited. Kelvin–Helmholtz instability causes the concentration of vorticity; thus the vortex sheet begins to roll up form vortex rings. The concentrated vortices induce velocity fields to drive neighbouring ring-like structures together, which is called pairing. If these rings are subjected to critical perturbations, they will be stretched until the ring structures are broken by viscous effects.

The vortex method[2,3] is widely used to simulate vortex-induced flow problems and is adopted here to simulate the evolution of a three-dimensionally periodic jet under axial perturbations. The rationale

* Correspondence to C. A. Lin, Department of Power Engineering, National Tsing Hua University, Hsinchu 30043, Taiwan.

Contract grant sponsor: National Science Council of Taiwan; Contract grant number: NSC-85-2212-E-007-057

CCC 0271–2091/97/100939–13$17.50

Received November 1995
Revised August 1996

behind the usage of the vortex filament method is the fact that the method is Lagrange-based and mainly concerns the concentrated vorticity in a flow field, so that the vortical structures can be easily traced during the evolution. If a Euler-based method is used, there may be too many grid points located in the regions where the vorticity is nearly zero. Although the method can keep track of the vortical structures, it becomes inefficient when those structures are destroyed. However, the vortex filament method is grid-free and hence does not introduce the types of diffusive errors which arise from the grid mesh in finite difference schemes.

Despite its appealing feature of being able to track the movements of an individual vortex, the vortex method is a computationally intensive scheme. Basically, each point vortex velocity is induced, through the Biot–Savart law, by all the surrounding vortices. The global interactions of all point vortices, a typical N-body problem with an operation cost proportional to $N(N-1)$, make it a good candidate for computations on parallel computers.[4–6]

When a vortex filament is severely stretched by Kelvin–Helmholtz instability excitation, to the extent that the segment is unable to resolve the curvature of the filament, the segment is bisected into two segments.[7] Therefore the number of segments, hence segment markets, in a filament is adaptive and is dynamically growing according to the requirement of numerical accuracy in order to simulate the physical environment. This adaptive data structure imposes load imbalance in simulating the flow on parallel machines if a static partition method is employed.

The present parallelization of the vortex filament method is designed in the native SPMD C environment and emphasis is on data distribution schemes, efficient support of parallel I/O and overlapping of communications with computations. For the static data structure a 'block' partitioning scheme is adopted. However, for the adaptive data structure, owing to the excessive stretching of vortex filaments, a novel packet-oriented technique of filament segment data distribution within computations is proposed to achieve good performance for this problem.

2. VORTEX FILAMENT METHOD

2.1. Computational formulation

The flow motion of an incompressible, inviscid fluid can be described by

$$\frac{\partial \Omega}{\partial t} + (V \cdot \nabla)\Omega = (\Omega \cdot \nabla)V, \tag{1}$$

where Ω and V represent vorticity and velocity respectively.

The vortex filament method approximates the vorticity field by numerous vortex filaments with an assumed vorticity distribution around the filament centrelines. The velocity, based on the Biot–Savart law, is calculated by filaments of finite core radius, as suggested by Leonard,[2] as

$$V \approx \sum_{i=1}^{N} \sum_{j=1}^{M_i} \Gamma_i K_\sigma(X - X_{i,j}) \delta l_{i,j}, \tag{2}$$

with

$$K_\sigma(X) = -\frac{1}{4\pi |X|^3} f\left(\frac{X}{\sigma}\right) \begin{pmatrix} 0 & x_3 & -x_2 \\ -x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{pmatrix}, \qquad f(r) = \frac{r^3}{r^2 + \alpha^{3/2}},$$

where N is the number of vortex filaments, M is the number of segment markers per filament, X is the location of a segment marker, Γ_i is the circulation, δl is the length of a segment, σ is the filament radius and α is a numerical parameter depending on the vorticity distribution of the vortex core. In

this paper the vorticity distribution is assumed to be constant and Gaussian and α is selected to be 0·413.[8]

Spatial curve integration is performed by cutting each filament into many line segments which are short enough to express the curvature of the filament. The length of each segment is represented by $\delta l_{i,j} = |X_{i,j+1} - X_{i,j}|$, where $X_{i,j}$ indicates the adjoint segment marker between segments.

Trajectories of each adjoint segment marker can be obtained by integrating

$$\frac{\mathrm{d}X_{i,j}}{\mathrm{d}t} = V(X_{i,j}, t) \tag{3}$$

with time. In the present methodology a second-order time scheme is used to move the adjoint segment marker X_i:

$$X^*_{i,j} = X_{i,j}(t) + V(X_{i,j}, t)\Delta t, \tag{4}$$

$$X_{i,j}(t + \Delta t) = X_{i,j}(t) + \frac{V(X_{i,j}, t) + V^*(X^*_{i,j}, t)}{2}\Delta t. \tag{5}$$

Consequently, the stretching effect in the vorticity equation is then implied in

$$\delta l_{i,j}(t + \Delta t) = X_{i,j+1}(t + \Delta t) - X_{i,j}(t + \Delta t). \tag{6}$$

When a vortex filament is severely stretched to the extent that the segment is unable to resolve the curvature of the filament, the segment will be bisected into two segments. Here we follow the algorithm by Kino and Ghoniem,[7] where the segment is bisected equally when it is longer than the longest segment at the initial time.

2.2. Numerical procedure

The data structures are composed of a group of filaments distributed throughout three-dimensional space, with each filament consisting of a collection of segment markers. All the data (location, velocity, etc.) in the filaments are stored in one-dimensional arrays, with pointers designed to locate the beginning and end of each filament.

The solution procedure of the vortex filament method can be summarized as follows.

1. Set up initial conditions $X_{i,j}$ and Γ_i.
2. Calculate initial maximum segment length $\delta l_{\mathrm{max,initial}}$.
3. Calculate velocity of segment marker, $V(X_{i,j}, t)$.
4. Calculate intermediate segment marker location $X^*_{i,j}$.
5. Calculate velocity of segment marker, $V^*(X^*_{i,j}, t)$.
6. Calculate segment marker location $X_{i,j}(t + \Delta t)$.
7. Calculate segment length $\delta l_{i,j}(t + \Delta t)$.
8. Check $\delta l_{i,j}(t + \Delta t) > \delta l_{\mathrm{max,initial}}$. False: go to step 3. True: insert a new segment marker between $X_{i,j}$ and $X_{i,j+1}$; go to step 3.

3. PARALLEL ALGORITHM

In the present parallel implementation the single-programme multiple-data (SPMD) environment is adopted. Therefore the key to multiple-CPU computations is the design of the data structure distribution. For parallel computations the straightforward method is to partition and distribute the group of filaments equally among processors, i.e. the static 'block' partitioning scheme. The data distribution scheme is illustrated in Figure 1, which shows a system of six filaments distributed

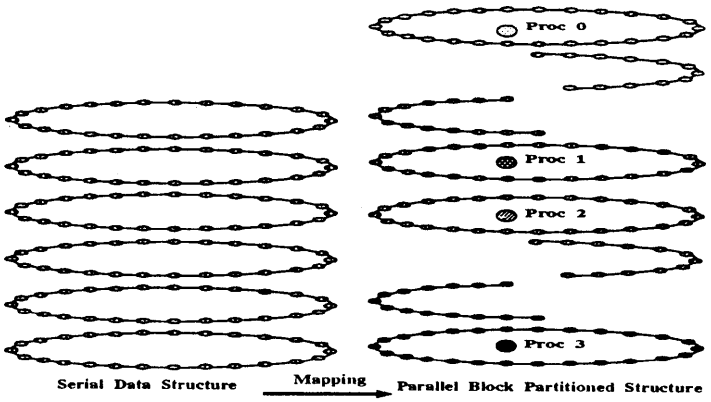

Figure 1. Global view of static 'block' partitioning scheme

among four processors, each having 1·5 filaments. The serial version is also shown on the left of the figure, indicating a non-partitioned data structure.

There are two problems with using the 'block' partitioning scheme to support the data structure in the problem. Firstly the number of segment markers in the data structure may increase dynamically and therefore data need to be inserted into the array. The 'block' array implementation requires all the data following the insertion point to be moved back. Secondly, and most importantly, the irregular growth of the segments results in load imbalance, so that the performance of the parallel programmes deteriorates.

3.1. Adaptive data structure with load-balancing scheme

Figure 2 shows an instance of the data structure at a given time. It represents an irregular aggregate structure with non-rectangular index sets. The number of rows of the structure corresponds to the number of filaments in the physical domain. The length of each row is related to the number of segment markers in a filament. Since the number of segment markers in a filament may increase

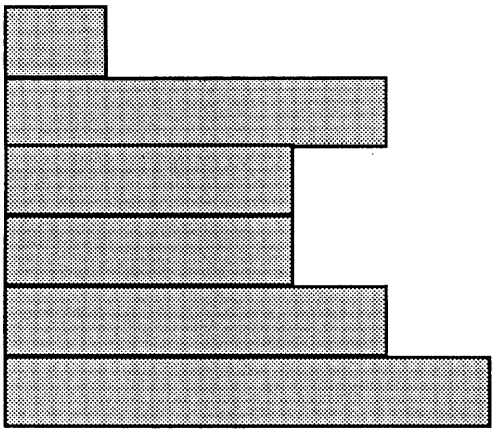

Figure 2. Irregular aggregate structure with non-rectangular index sets

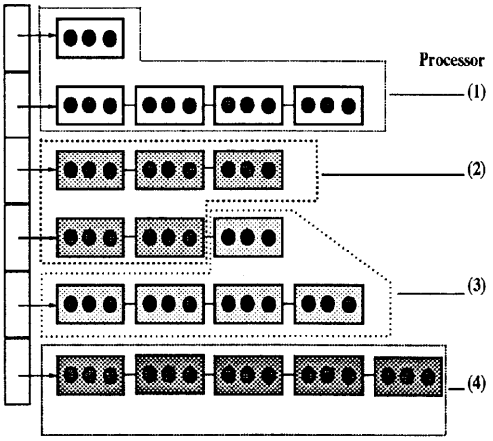

Figure 3. Global view of packet-oriented partitioning scheme

Table I. Packet-related functions

GetPack(ArrayX,Index)	Return the address of the packet containing the element of the index
IsPackEnd(ArrayX,Index)	Given an index and decide if is is the end element of a packet
IsPackStart(ArrayX,Index)	Given an index and decide if it is the starting element of a packet
SendPack(ArrayX,Index)	Given an idex and send the packet to the processor who needs it
RecvPack(ArrayX,Index)	Given an index and receive the packet from the processor
AddPack(ArrayX,PackY)	Given a packet and add it to the ring
DelPack(ArrayX,PackY)	Remove a packet from a ring

during run time, the static 'block' partitioning scheme can cause load imbalance in parallel computations.

The load imbalance problem is solved by decomposing the data structure into a packet-oriented parallel data structure as shown in Figure 3. The structure is designed to support dynamically growing structures with non-rectangular index sets. In the scheme the filament is divided into a set of packets, which are then distributed among processors, and the balance of the computation is controlled by the number of packets in each processor. The data (segment markers) may grow according to the requirement of numerical accuracy and, instead of being inserted into an array, the newly created segment markers are actually put into the packet. The number of segment markers in a packet increases and in the present construction the packet is automatically split if the number of data in the packet is above a preset bound. Packets are moved around different processors to maintain the balance of computational loads among processors. Figure 3 shows that in the global view each processor possesses a portion of the filaments.

The structure is very much like a distributed array,[9,10] except that it allows the size of the distributed array to be increased and works comparably with a set of load-balancing algorithms. Arrays are distributed among processors by decomposing into packets. When the imbalance begins, the work load is rebalanced by dividing packets equally among processors at run time. To support the abstract data structure, a set of operations for packets is constructed, as listed in Table I. AddPack and DelPack control the caching of the remote data. SendPack and RecvPack transfer the packets between different processors. IsPackStart and IsPackEnd are used to control the beginning and end of each packet.

4. RESULTS AND DISCUSSION

4.1. Behaviour of three-dimensionally evolving jet

The primary vortical structure of the circular inviscid jet is observed to be dominated by inviscidity. Spatial periodicity further simplifies the calculations of the jet development. The infinite period was approximated with three upstream and downstream periods. The axial wavelength was discretized into 39 filaments, each of which initially contained 90 segment markers. The radius of the jet was taken to be $R = 5$ and a constant filament radius $\sigma = 0{\cdot}1R$ was adopted. The initial axial perturbation was simulated by a sinusoidal variation in circulation: the undisturbed circulation is $\Gamma_0 = 0{\cdot}1$; $\Gamma = \Gamma_0[1 + \epsilon \sin(2\pi z/\lambda)]$, where λ is the axial wavelength, chosen as 2π, and $\epsilon = 0{\cdot}05$ is the perturbation strength. The simulation time step was chosen to be 0·05 s.

Because of Kelvin–Hemlholtz instability, a wave distribition of vortex filaments was observed in side view about 8 s after simulations started (Figure 4). This concentration of filaments indicates the formation of a concentrated vorticity. It can also be observed that filaments behind the concentrated zone have smaller radii than those ahead of it. This situation, known as leap-frogging, is observed when two vortex rings proceed in the same direction with the front ring expanding and the rear one

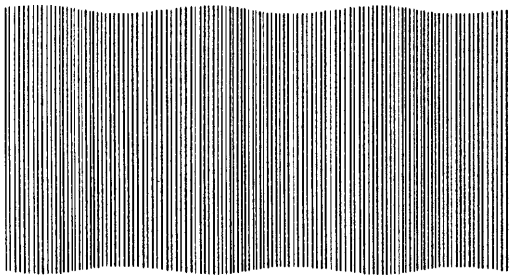

Figure 4. Vortex filament distribution (three periods) at 8 s

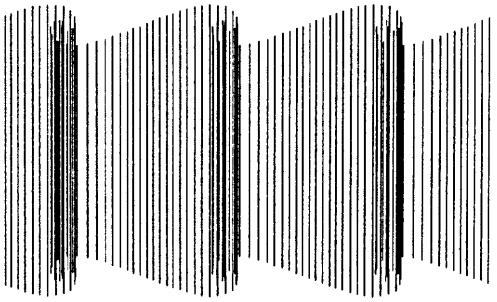

Figure 5. Vortex filament distribution (three periods) at 16 s

contracting. This phenomenon is caused by the different directions of induced velocities, as the rear part of the vortex ring experiences an induced velocity towards the centre while in the region ahead of the ring the induced velocity is away from the centre. From the experience of a single vortex ring the proceeding speed decelerates when the ring starts to expand and therefore the filaments behind the concentrated filaments proceed faster than the preceding ones. Consequently, one observes that some filaments originally neighbouring the concentrated filaments merge into and even precede the concentrated ones.

As time proceeded, those concentrated filaments were perceived to revolve around each other (Figure 5), a phenomenon similar to the vortex roll-up observed in mixing layers or shear layers. Looking in the direction along the axis of the filament centre, all the filaments were found to be circular, which was a consequence of no azimuthal perturbations being imposed on the filaments. As a result, no vorticity was directed in the streamwise direction but only in the azimuthal direction. From the stretching term in the vorticity equation one can infer that the vorticity field was stretched azimuthally, though there is a velocity strain in the streamwise direction.

4.2. Parallelism performance results with static data structures

In the first experiment the set of filaments was statically partitioned among different processors. This scheme works well with small circulation and in the initial stages when the number of segment markers in each processor remains constant and is evenly distributed.

There are two more issues to be addressed for achieving good performance other than the data distribution issue. Firstly, asynchronous message passing is needed to overlap communications with computations. Communication and computation can be blocked into k stages and software pipes messages one stage ahead of the computations. The idea is illustrated in Figure 6, where k is the number of stages of communications, c_i is the communication time spent in the ith stage and τ_i is the computation time spent in the ith stage. The speed-up of overlapping is two if the communication and computation times are the same, while there will be no speed-up if the amount of computation is far greater than the communication time or if the communication time is far greater than the computation time. In actual experiments the speed-up factor becomes more significant as the processor number grows and the problem size becomes fine grain, so that the communication time gradually increases to about the same as the computation time. The overlap speed-up factor then decreases, since the granularity of the parallelism grows even smaller than the communication granularity as the processor number grows even bigger.

In some applications the scientific data can be numerous and frequently have to be written out to be visualized, e.g. the intermediate results for a transient problem; therefore efficient support for parallel I/O operation is important. Figure 7 illustrates the time spent using both sequential and parallel I/O. Time for the sequential version is measured using the accumulated time for all the processors

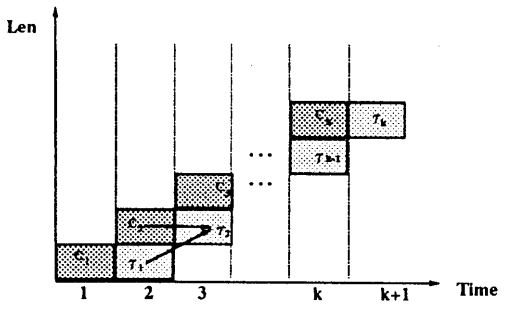

Figure 6. Communications overlapping steps with message pipelining

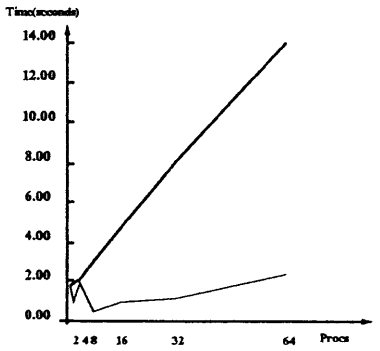

Figure 7. Time spent for sequential (————) and parallel (————) I/O

performing I/O, i.e. sequentially. The parallel version is timed in the following way. Since the segment markers are distributed among processors, the sizes of segment markers in the processors are needed to serve as the file point offset when each processor needs to write data to a file. Then the parallel I/O scheme supported by nCUBE2 is used to co-ordinate the concurrent I/O operations. Although nCUBE2 supports parallel I/O logically, there are not as many physical I/O nodes as processor nodes. Therefore the parallel I/O performance increases only slightly as the processor number grows. It significantly outperforms sequential I/O, however.

The final performance with the static distribution scheme, which is computed on nCUBE2, is listed in Table II. The parallel performance speeds up reasonably well and the best performance is with 32 filaments running on 128 processors, the speed-up being around 87·5 when compared with the one-processor run. The ratios of communication time to computation time with 10 filaments using eight and 16 processors are 15% and 35% respectively. Figures 8 and 9 illustrate the performance profiles from nCUBE2.

4.3. Load-balancing effects with packet-oriented structures

In order to explore the load-balancing strategy based on the packet-oriented data structure, the initial undisturbed circulation Γ_0 was set to be 5·0, in strong contrast with the weak strength of 0·1 in the previous case. This excessive strength provides a faster growth of the vortex motion. The axial wavelength was discretized into 10 filaments, each of which initially contained 40 segment markers.

Owing to Kelvin–Helmholtz instability excitation, the vortex filament is severely stretched and, in order to preserve accuracy, the lengthened segment is consecutively bisected into two segments. This

Table II. Performance results on nCUBE2 with static data structure

Problem size	Measurement	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	$p=64$	$p=128$
10 rings	Time	861	428	215	108	58	35·8	28·7	34
	Speed-up	1	2·0	3·99	7·9	14·8	24·0	30·0	25·3
20 rings	Time	3504	1748	872	438	223	118	71	57
	Speed-up	1	2·0	4·0	7·9	15·7	29·6	49·3	61·5
32 rings	Time	—	—	2188	1095	552	281	154	100
	Speed-up	1	2·0	4·0	7·9	15·8	31·4	56·8	87·5
39 rings	Time	—	—	3341	1675	847	432	239	157
	Speed-up	1	2·0	4·0	7·9	15·8	30·9	55·9	85·1

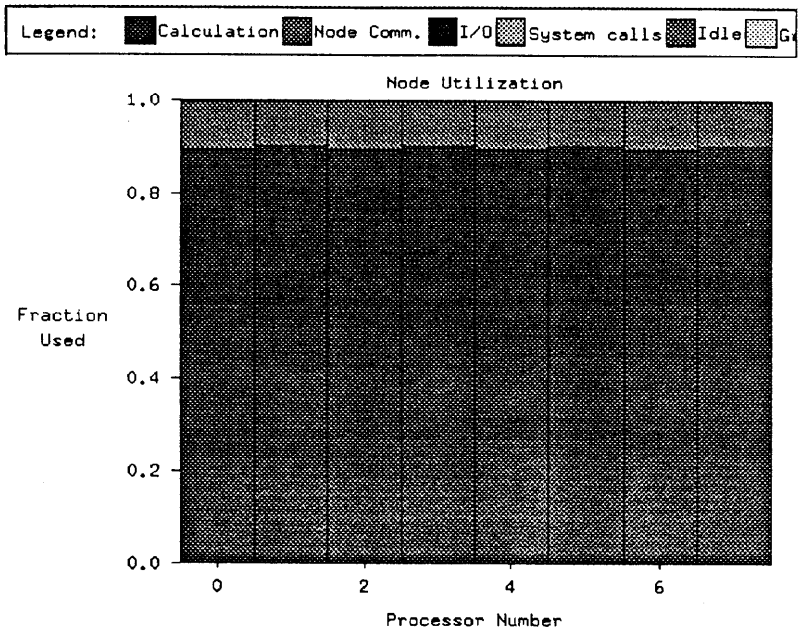

Figure 8. Profile with static partitioning scheme on eight-node nCUBE2

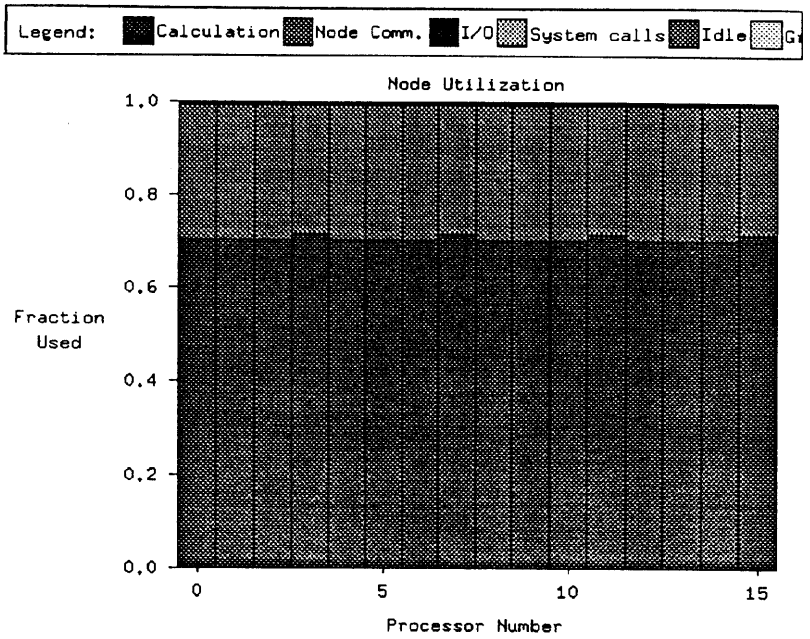

Figure 9. Profile with static partitioning scheme on 16-node nCUBE2

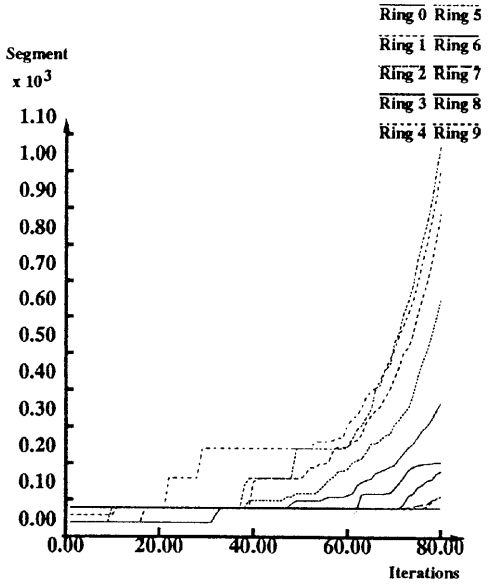

Figure 10. Segment marker distribution on different filaments

phenomenon can be clearly seen from Figure 10, which shows the number of segment markers on each filament (ring), indicating the irregular segment marker growth pattern.

Since the computational loading is proportional to the number of distributed segment markers on individual processors, it is essential to maintain an equal number of segment markers on each processor. Figures 11 and 13 indicate the extent of load imbalance on different processors for the eight- and 16-processor runs with static data partition. The figures show that the numbers of segment

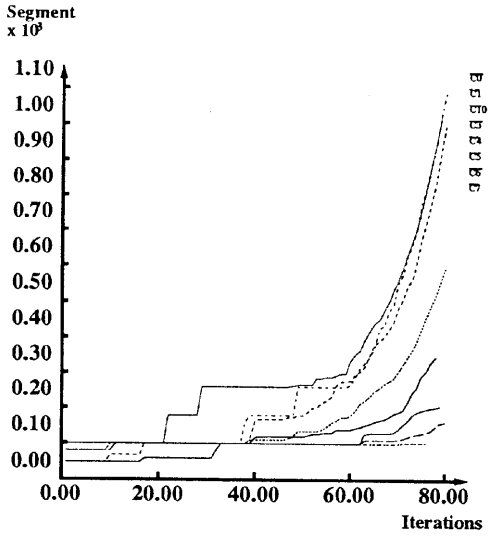

Figure 11. Number of segment markers in eight-node machine with static partitioning scheme

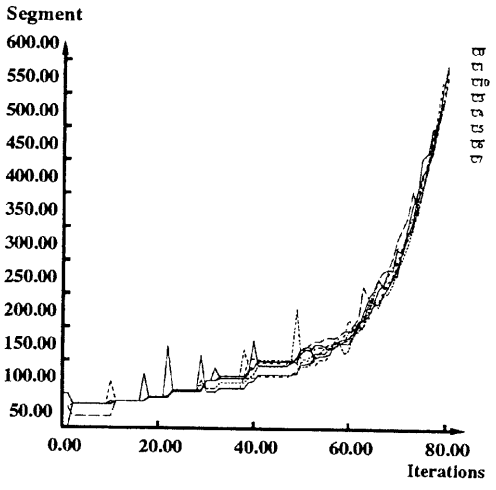

Figure 12. Number of segment markers in eight-node machine with run time load-balancing scheme

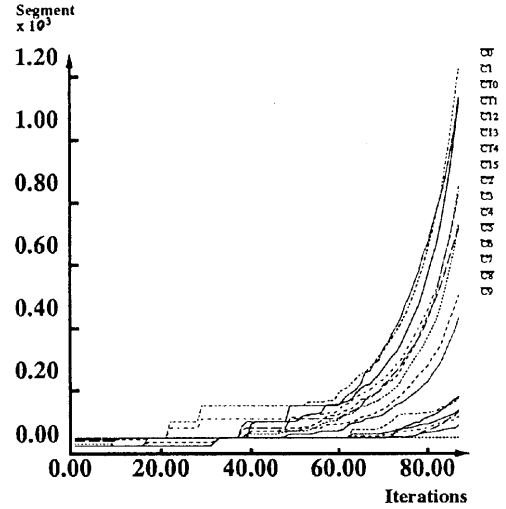

Figure 13. Number of segment-markers in 16-node machine with static partitioning scheme

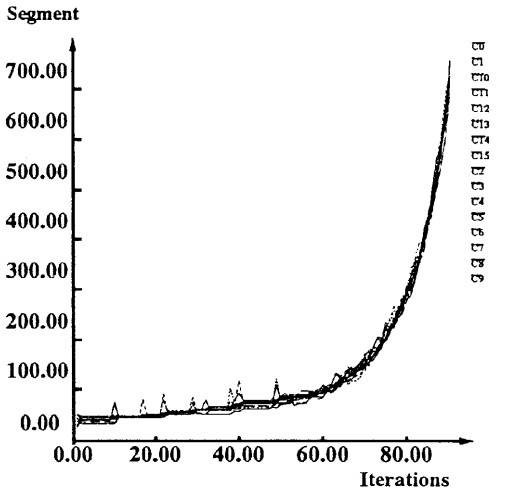

Figure 14. Number of segment markers in 16-node machine with run time load-balancing scheme

markers on each processor vary significantly from each other at a given iteration. In contrast, the runs with run time load balancing by incorporating the packet-oriented scheme, shown in Figures 12 and 14, demonstrate the superiority of this method in achieving load balance.

Attention is now directed to the CPU time performance on different machines. Table III shows the normalized accumulated CPU time on DEC ALPHA workstation cluster, IBM SP2 and nCUBE2 up to 50 iterations. The superiority of the load balancing can be further affirmed by the results shown in Table IV, giving the ratio of the execution times of the unbalanced (T_u) and balanced (T_b) runs. It should be pointed out that DEC Alpha workstation clusters and IBM SP2 are not dedicated machines, so the advantage of run time load balancing on these machines, though favourable, is obscured by the non-dedicated environment.

The maximum performance improvement of 2·6 times over the static scheme was achieved using nCUBE2, which is a dedicated environment. The difference in speed-up can be attributed to how evenly the filament segment markers were distributed among the processors. This might be due to the fact that the newly generated segment markers, owing to the stretching of the vortex filament, cannot be divided evenly among the processors in the form of packets with a finite number of segment markers in them.

In order to examine the scalability of the present scheme, focus is directed on the CPU time history of the nCUBE2 runs, which is a dedicated machine, shown in Figures 15 and 16. The advantages of load balancing can be demonstrated via the CPU time history, which again indicates the superior performance of the load-balancing scheme over the static partitioning scheme. As was indicated by the evenly distributed segment markers (Figures 12 and 14), good scalability of the scheme is achieved.

Finally, examples of performance profiles are shown in Figures 17 and 18, indicating the percentage of computation time spent with each processor. A profile of the unbalanced case is presented in Figure 17, which shows that the computational times spent in different processors vary significantly from each other. In contrast, with the dynamic packet-oriented data structure support a much more balanced performance profile, shown in Figure 18, is obtained on each processor.

Table III. Normalized accumulated execution time without and with load-balancing scheme at 50 iterations

Number of processors	DEC 3000/500		SP2 9076		nCUBE2	
	Unbalanced	Balanced	Unbalanced	Balanced	Unbalanced	Balanced
4	0·131	0·085	0·172	0·095	1·220	0·734
8	0·138	0·089	0·121	0·093	1	0·377
16			0·094	0·039	0·484	0·210

Table IV. Accumulated execution time ratio witout and with load-balancing scheme, I_u/I_b, at 50 iterations

Number of processors	DEC 3000/500	SP2 9076	nCUBE2
4	1·534	1 792	1·664
8	1·547	1·311	2·647
16		2·379	2·307

5. CONCLUSIONS

A circular, inviscid, spatially periodic jet flow subjected to axial perturbations was simulated by the vortex filament method on parallel machines with distributed memory. This parallel experiment was conducted using the SPMD (single-programme multiple-data) programming model on nCUBE2, DEC Alpha workstation clusters and IBM SP2.

For a three-dimensionally evolving jet it was observed that the concentrated vortex rings due to Kelvin–Helmholtz instability revolve around each other, a phenomenon similar to the vortex roll-up

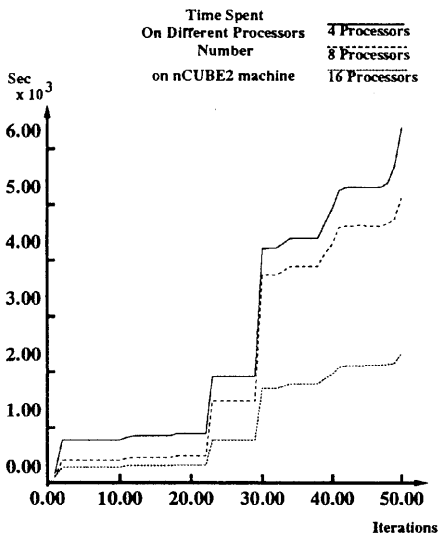

Figure 15. CPU time on nCUBE2 with static data partitioning

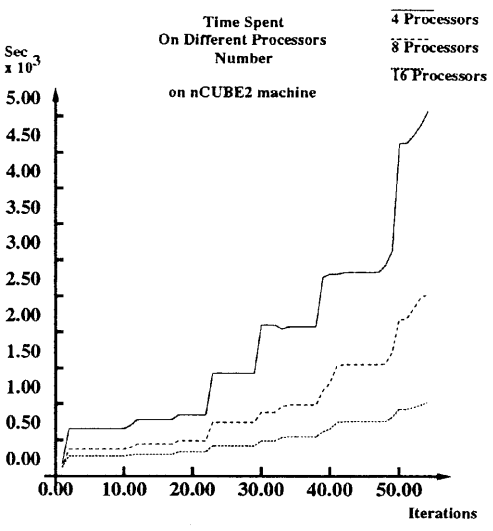

Figure 16. CPU time on nCUBE2 with run time load balancing

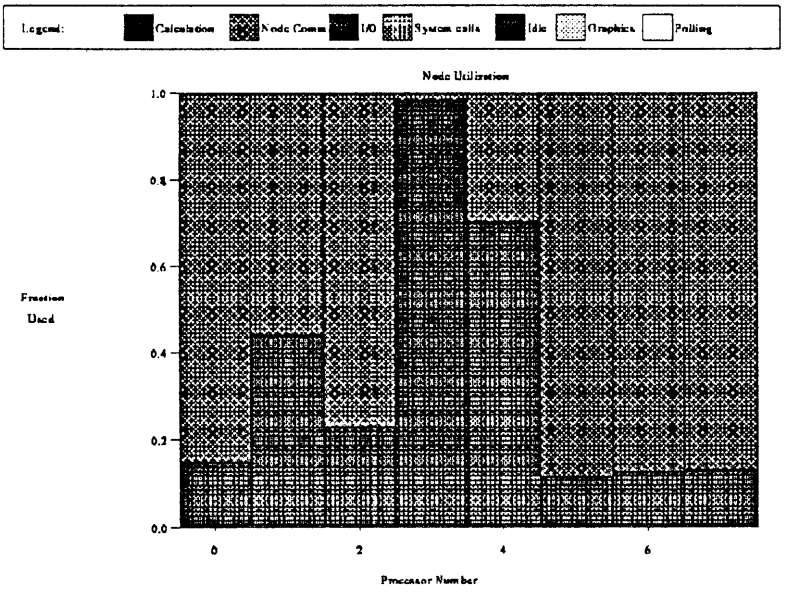

Figure 17. Profile of performance on eight-node nCUBE2 with static data partitioning

observed in mixing layers or shear layers. The absence of axial direction vorticity development was attributed to the lack of azimuthal perturbations imposed on the field.

The implementation of vortex filament methods with adaptive data structure on parallel machines is accomplished through a packet-oriented data structure to support the dynamically growing data structure, i.e. the increase in filament segments due to vortex stretching, with non-rectangular index

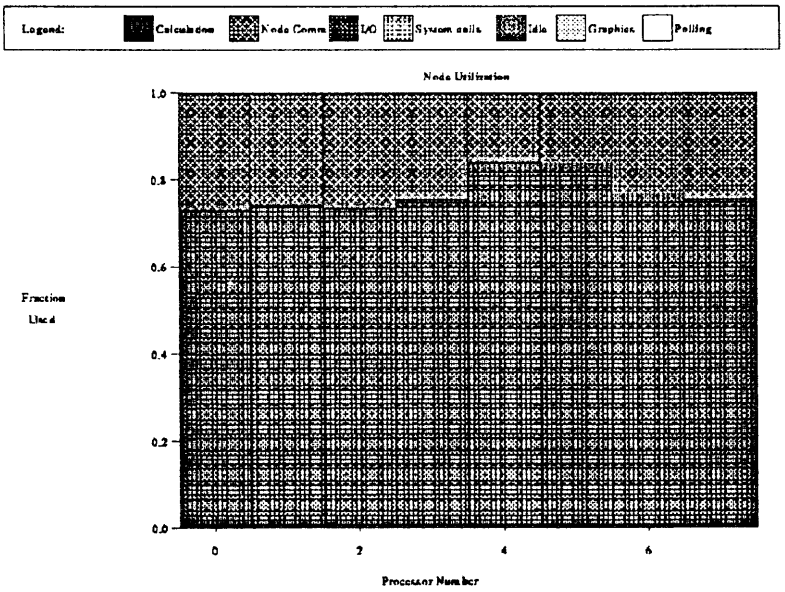

Figure 18. Profile of performance on eight-node nCUBE2 with run time load balancing

sets. The number of elements of a packet grows with the data and in the present construction the packet was automatically split if the number of elements in the packet was above a present bound. Packets were moved around different processors to maintain the balance of computational loads among processors. Computational results indicate that the load-balancing scheme performs much better than the static data-partitioning scheme. It is concluded that the run time load-balancing scheme is preferred in computations where the computational load is changing dynamically.

ACKNOWLEDGEMENTS

The work documented herein was supported by the National Science Council of Taiwan under grant NSC-85-2212-E-007-057 which the authors gratefully acknowledge. Gratitude is also expressed to the National Centre for High Performance Computing, Taiwan for providing access to its 8-node Dec Alpha workstation clusters and 32-node IBM SP2, and to San Diego Supercomputing Centre for providing access to its 128-node nCUBE2.

REFERENCES

1. G. K. Batchelor and A. E. Gill, 'Analysis of the stability of axisymmetric jets', *J. Fluid Mech.*, **14**, 529 (1962).
2. A. Leonard, 'Computing three-dimensional incompressible flows with vortex elements', *Ann. Rev. Fluid Mech.*, **17**, 523 (1985).
3. J. E. Martin and E. Meiburg, 'Numerical investigation of three-dimensionally evolving jets subject to axisymmetric and azimuthal perturbations', *J. Fluid Mech.*, **230**, 271 (1991).
4. J. A. Sethian, J.-P. Brunet, A. Greenberg and J. P. Mesirov, 'Computing turbulent flow in complex geometries on a massively parallel computer', *Proc. Supercomputing '91*, Albuquerque, NM, ACM, November 1991, pp. 230–241.
5. D. J. Doorly and M. Hilka, '3D point vortex methods for parallel flow computations,' *Proc. Sixth SIAM Conf. on Parallel Processing for Scientific Computing, SIAM, Philadelphia, PA, 1993, pp. 35–39.*
6. Y. L. Shieh, J. K. Lee, J. H. Tsai and C. A. Lin, 'Computations of three-dimensionally evolving jets with vortex methods on parallel machine with distributed memory', *Parallel CFD: New Algorithms and Applications*, Ed. A. Ecer *et al*. Elsevier Science B.V., 1995, pp. 119–126.
7. O. M. Kino and A. F. Ghoniem, 'Numerical study of a three-dimensional vortex method', *J. Comput. Phys.*, **86**, 75 (1990).
8. W. T. Ashurst and E. Meiburg, 'Three-dimensional shear layers via vortex dynamics', *J. Fluid MEch.*, **189**, 87 (1988).
9. J. K. Lee and D. Gannon, 'Object-oriented parallel programming: experiments and results', *Proc. Supercomputing '91*, Albuquerque, NM, ACM, November 1991, pp. 273–282.
10. S. X. Yang, J. K. Lee, S. P. Narayana and D. Gannon, 'Programming an astrophysics application in an object-oriented parallel language', *Proc. Scalable High Performance Computing Conf.*, Williamsburg, Virgina, June, IEEE Computer Society, 1992, pp. 236–239.

