

Massively Parallel Implementation of a 3D Vortex-Boundary Element Method

Adrin Gharakhani and Ahmed F. Ghoniem
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Abstract

Vortex-boundary element simulation of three-dimensional wall-bounded flows is performed on a massively parallel architecture using the data parallel paradigm. With proper optimization, implementation on the Thinking Machines CM5 is shown to yield over an order of magnitude speed-up over the Cray C90, using 128 processors. This makes the direct evaluation of the flow field represented by up to 100 000 vortex elements feasible. In this paper, the CPU time and memory requirements for the direct evaluation of the vortical field using three parallelization algorithms are compared. In addition, performance results for the evaluation of the vortical and potential velocities, and their gradients, are presented as a function of the number of vortex elements and the number of processing nodes. Finally, to demonstrate the capability of the developed method, preliminary results from the case of flow around a stationary, idealized trailer truck near the ground level at $Re = 500$ are presented.

1 Introduction

A vortex-boundary element method has been developed for the grid-free simulation of time dependent, incompressible, three-dimensional wall-bounded flow. The numerical scheme is based on a combination of the Lagrangian vortex method to simulate the convection and stretch of the vortical field, the random walk method to describe the diffusion process, and the boundary element method to impose the normal flux boundary condition on the boundary surface. The no-slip boundary condition is satisfied using an extended vortex tile generation mechanism [2, 3].

At present the velocity and its gradients of N vortex elements are obtained by the direct formulation and comprise: (1) an $O(N^2)$ evaluation of the vortical component using the Biot-Savart representation of the kinematics of the vorticity field, and (2) an $O(NK)$ evaluation of the potential component due to the interaction of the vortex elements with K boundary elements discretizing the boundary surface. Both components are expensive to compute and may require hundreds of hours on a super vector computer to simulate the simplest of unsteady 3D flow phenomena. For example, one predictor-corrector timestep using only 20 000 vortex elements and 2 000 boundary elements requires 10 minutes on the Cray C90. The third factor which may contribute to the CPU cost significantly is the $O(n^3)$ LU decomposition of the fully populated matrix resulting from the boundary element solution of the Laplace equation—used to impose the normal flux boundary condition over the n collocation points on the boundary surface. This constitutes a negligible percentage of the cost for problems with fixed geometries, since it is performed once at the beginning of the computation; only the back substitution process is repeated in the subsequent timesteps. However, for problems with time varying geometries, such as an IC engine, the LU

decomposition is repeated at every timestep and, depending on the boundary element resolution, can become significant.

An effective approach to reduce the CPU cost for the velocity evaluations is the application of the multipole expansion concept, where the computational complexity is reduced greatly from a quadratic to a near linear order in N [1, 4, 5]. Similarly, the cost as well as the storage requirements for the boundary element matrix assembly and inversion can be reduced with the aid of the multipole expansion and Krylov-subspace iterative algorithms [6]. However, an efficient implementation of these algorithms for a complex 3D configuration, with higher order boundary elements (such as the ones used in our vortex-boundary element method) requires considerable development and programming effort. An alternative approach which requires less effort is to exploit the inherently parallel properties of the velocity evaluations, and to reduce the solution time by some nominal factor using P processors in parallel [7]. In this case, although the computational complexity remains quadratic, the reduction in the proportionality constant facilitates the use of a large number of vortex elements comfortably. This “quick” improvement endows us with the computational power to explore the accuracy and the versatility of the 3D vortex-boundary element method using “realistic” example problems.

In this paper, the data parallel implementation of a 3D vortex-boundary element method [2, 3] on the Thinking Machines CM5 is presented. In particular, emphasis is given to the direct evaluation of the vortical and potential velocity fields and their gradients. Three parallelization algorithms are applied to the vortical field evaluations and the best method is selected as a model for converting the potential field calculations. The CM5 performance is measured in terms of its CPU speed-up over the C90. Performance results are presented as a function of increasing number of vortex elements, as well as processing nodes. The robustness of the fully parallelized code is demonstrated by simulating the flow around a simplified stationary truck near the ground level at $Re = 500$, based on the truck width and the free stream velocity. Preliminary results are given without detailed analyses.

2 Data parallel and the Thinking Machines CM5

The vortex-boundary element method can most naturally be coded using the data parallel programming style, which offers some significant advantages over other paradigms. To begin with, the order of events in data parallel programming is sequential and essentially the same as in the serial version. Similarly, the flow control in the data parallel approach is nearly identical to that of the serial, with the important consequence that problems with synchronization and deadlock are automatically alleviated. Furthermore, extension from the serial to the parallel programming is straightforward and requires little addition of syntax.

At the time of code development, the Thinking Machines CM-FORTRAN offered the most advanced High Performance FORTRAN compiler—the language for data parallel programming—with an extended math library. Thus we selected the Thinking Machines CM5 at the National Center for Supercomputing Applications, which has a total of 512 nodes (or processors) to perform these calculations. Each node is a RISC-based Sparc Station with 32 Mbytes of memory, and a peak performance of 160 Mflops and 160 Mops. The nodes are upgraded by vector units, which allow pipelining and vectorization and enhance the floating point performance by at least a factor of three. Communication between the nodes is handled efficiently by separating the data network from the control network. The bandwidth is rated at 160 Mbytes/second.

We should mention here that, despite the simplicity of the data parallel paradigm and the existence of advanced routines (and hardware) to minimize the communication overhead, it is quite

easy to design an inefficient or slow program using CM-FORTRAN. For example, arrays with differing geometries or maximum lengths reside in logically different locations and an operation involving such arrays can generate tremendous communication. The process of matrix assembly is one example of operations that involve arrays with different geometries; i.e., 1D and 2D arrays. The evaluation of the potential velocity at the vortex element centers is an example of operations that involve multiple array sizes; i.e., arrays with maximum lengths equal to the number of boundary elements, the node numbers, and the number of vortex elements. Tile generation, evolution, and conversion to vortex elements are all processes which include many complications. On a vector architecture, tile routines constitute a few percent of the entire computational cost; however, with an improper assignment of the arrays the cost can rise to 10–20 percent on the CM5! In the next section we will demonstrate how simple algorithmic changes can lead to significantly different performance results.

3 Results

3.1 Parallel computation of the vortical velocity field

We begin by experimenting with the simple case of evaluating the velocity induced by the vortex elements on each other. The velocity at \mathbf{x}_i —induced by a collection of N_V vortex elements, each centered at \mathbf{x}_j with element volume ΔV_j and vorticity vector ω_j —is given by:

$$\mathbf{u}_\omega(\mathbf{x}_i, t) = \sum_{j=1}^{N_V} K_\sigma(\mathbf{x}_i - \mathbf{x}_j) \wedge \tilde{\Gamma}_j(t), \quad i = 1, \dots, N_V$$

where $\tilde{\Gamma}_j(t) = \omega_j(t)\Delta V_j$, $K_\sigma(\mathbf{x}) = -\frac{\mathbf{x}}{4\pi|\mathbf{x}|^3}f\left(\frac{|\mathbf{x}|}{\sigma}\right)$, $K_\sigma(0) = 0$, and $f(r) = \tanh(r^3)$. The velocity gradients are evaluated in a similar manner and will not be presented here. (See [2, 3] for details.) Note that all arrays in these computations have identical geometries and maximum lengths. Therefore, being logically local to each other, all operations are performed with minimum communication cost. Clearly, the parallelization of problems involving only vortical flows in free space is simple.

Two algorithms are immediately obvious from the equation above. In the first approach, the inner loop ($j = 1 : N_V$) can be viewed in parallel. Here, the influences $K_\sigma(\mathbf{x}_i - \mathbf{x}_j) \wedge \tilde{\Gamma}_j(t)$ of all ($j = 1 : N_V$) elements on an individual element i are evaluated simultaneously. The computation for each i is completed by performing a parallel sum over all j 's and storing the result in \mathbf{u}_i . This procedure is then repeated for all i 's, consecutively, to obtain $\mathbf{u}_i (i = 1 : N_V)$. The second approach parallelizes the outer loop ($i = 1 : N_V$). In this case, the influence $K_\sigma(\mathbf{x}_i - \mathbf{x}_j) \wedge \tilde{\Gamma}_j(t)$ of an individual element j on all ($i = 1 : N_V$) elements is evaluated simultaneously, and its contribution added to $\mathbf{u}_i (i = 1 : N_V)$ as j is looped N_V times. A third approach was also experimented with, in which case the inner loop method was modified such that the influences of ($j = 1 : N_V$) elements are evaluated simultaneously on blocks of N_B elements at a time, instead of an individual element i . This procedure is then repeated $(N_V - 1)/N_B + 1$ times.

Figure 1a depicts the elapsed times versus the number of vortex elements, for evaluating the vortical velocity and its gradients, using the C90 and the three parallelization schemes discussed above. Figure 1b shows the memory required for the three methods as a function of the number of elements. The parallel runs were made using 32 processing nodes. The CM5 is at least a factor of two slower than the C90 when the inner loop approach is applied. Note that the efficiency of this approach (not shown) is relatively high; however, while efficiency is an important factor to consider, we are primarily interested in obtaining a shorter elapsed time than we can already achieve using the

C90. For this reason the first method is rejected. The third method, the blocked inner loop scheme with $N_B = 64$, speeds up the first approach considerably to an elapsed time which is comparable to that obtained by the C90. This implies that a favorable speed-up over the C90 is actually possible if larger node numbers than 32 are used. However, as indicated in Fig. 1b, this improvement in the CPU time is accompanied by an increase in memory which grows with the number of vortex elements. As a result, for a larger number of vortex elements, where parallelization is needed the most, this approach becomes impractical and is thus rejected. The outer loop method outperforms the other two, both in terms of speed-up and memory requirements. For example, using 20 000 vortex elements, a factor of two speed-up is observed compared to the C90. This is an interesting example of how simple algorithmic changes may yield dramatically different performances on the CM5. Note that the only significant difference between the first and the second methods is the handling of the summation at the end of each loop. (Interestingly, the parallel sum operation that is used in the slowest method is recommended by the manufacturer as a fast operation, because of the software and hardware support for it.) In Fig. 1b, the memory curves for the inner and outer loops remain horizontal, because all the arrays are assigned a fixed maximum size that is larger than the number of vortex elements reported in the figure. The extra memory requirement in the first method, compared to the second, is due to the parallel summation. We have also experimented with a block orrery approach, which is expected to minimize interprocessor communication [7]. However, using up to 40 000 vortex elements, we found the block orrery approach to be slower than the outer loop method. We have no explanation for this.

Having selected the outer loop approach, we now examine in Fig. 2 the CM5 speed-up over the C90 as a function of the number of vortex elements, using different machine partitions. The computations include the velocity and its gradients. All curves display low speed-up values at smaller number of vortex elements, which deteriorate sharply with an increase in the number of node partitions—indicating that interprocessor communication is relatively high compared to the actual computations. However, the curves asymptote fairly quickly to fixed speed-up values at larger number of elements. The asymptotes are reached earlier for smaller number of processors. At 40 000 vortex elements a factor of 2.4 speed-up is observed using the 32 node partition, while a factor of 7.2 is achieved using the 128 node partition. The curve for the latter has not leveled off yet and a higher speed-up is expected at larger number of vortex elements. In fact, the speed-up at 100 000 vortex elements is 9.4. The vortical field using 100 000 vortex elements on the 256 node partition is obtained in 190 seconds—a speed-up of 15.8 over the C90.

The CM5 does not offer a utility to evaluate the rate of the floating point operations. However, knowing that the code runs at 400 Mflops on the C90, and using the CM5 timing data for the larger number of elements (to minimize communication effects,) we estimate the parallel code to operate at the rate of 30–35 Mflops per processor.

3.2 Parallel computation of the potential velocity field

The j -th component of the potential velocity at \mathbf{x}_j due to its interaction with K boundary elements; each assigned a linear variation of the normal flux $q_k(\mathbf{x}_o)$ and tangential velocities $\Phi_{k,\tau}(\mathbf{x}_o)$ and $\Phi_{k,\rho}(\mathbf{x}_o)$ is:

$$u_{P_j}(\mathbf{x}_i) = \sum_{k=1}^K \int_{\partial D_k} -G_{,p}(\mathbf{x}_o, \mathbf{x}_i) \{ \varepsilon_{jpl} [\rho_l^k \Phi_{k,\tau}(\mathbf{x}_o) - \tau_l^k \Phi_{k,\rho}(\mathbf{x}_o)] - \delta_{jp} q_k(\mathbf{x}_o) \} dS_k(\mathbf{x}_o)$$

$$i = 1, \dots, N_V, \quad (j, l, p) = 1, 2, 3$$

where $G(\mathbf{x}_o, \mathbf{x}_i) = \frac{1}{4\pi|\mathbf{x}_o - \mathbf{x}_i|}$; j , l and p indices indicate direction with respect to the global coordinate system and follow the Einstein rule; $(\cdot)_{,r}$ represents differentiation with respect to \mathbf{x}_o in the r -th direction, δ_{jp} is the Kronecker delta, and ε_{jpl} is the antisymmetric permutation tensor. $dS_k(\mathbf{x}_o)$ is the surface area of the k -th element on the domain boundary, ∂D_k , and $(\boldsymbol{\pi}, \boldsymbol{\rho}, \mathbf{n})$ is the orthonormal coordinate system local to it. (See [2, 3] for the velocity gradient formulation.)

We parallelize the outer loop ($i = 1 : N_V$) based on the conclusions made from the parallelization of the vortical velocity evaluations. In this case, the influence of an individual boundary element k is evaluated on all ($i = 1 : N_V$) vortex elements simultaneously, and its contribution is added to $\mathbf{u}_i (i = 1 : N_V)$ as k is looped K times. An additional advantage of the outer loop parallelization is that K is nominally smaller than N_V ; therefore, the total number of serial loops is reduced. Note that the maximum lengths of the arrays in the potential velocity formulation are N_V , K or n (the number of collocation points.) Therefore, variables that must be operated on in parallel reside in logically different locations. Subsequently, unlike the simpler vortical flow computations, there is communication overhead inherent in the potential velocity computations.

Figure 3 depicts the CM5 speed-up over the C90 versus the number of vortex elements, using three node partitions. The computations include the potential velocity and its gradient. Note that the parallelization is obtained with respect to the vortex elements and that the number of boundary elements does not enter in the parametric study. The trend of the curves in Fig. 3 is very similar to that observed in the previous case; however, it is interesting to note that the speed-up obtained for the potential solution is higher than that for the vortical solution.

Finally, Fig. 4 depicts an analysis of the LU decomposition of a typical matrix obtained from the boundary element solution of the Neumann problem. n represents the number of collocation points; thus, the matrix size is n^2 . The C90 solution was obtained using the Cray library routine, whereas the CM5 solution was obtained using its own IMSL routine—the CMSSL library routine. The parallelization efficiency is quite poor within the tested range. Furthermore, speed-up is observed for n larger than 3 000.

3.3 Flow over a simplified truck

In this section, we present preliminary results from the simulation of flow over a simplified trailer truck near the ground level, where the latter is allowed to satisfy the no-slip condition. The Reynolds number based on the free stream velocity and the truck width, w , is 500. The objective of this example is to demonstrate the efficacy of the parallel vortex-boundary element method in simulating 3D flow with large number of vortex elements. No attempt has yet been made to analyze the complex vortical structures in the flow; a task which is left to future publication.

In this problem, the velocity was normalized by the uniform inlet velocity U_{in} and the lengths were non-dimensionalized by the truck width. The computational domain consisted of a confining cube with dimensions 16:16:16. The origin of the coordinate system was positioned at the lower left corner of the inlet surface of the confining cube, such that x and y were set parallel to the sides of the inlet, and the z coordinate pointed in the streamwise direction toward the exit surface. In addition to the cube, the domain included (1) a bluff body representing the truck with dimensions 1:1:0.5 in the x - y - z directions, respectively, and centered at (8, 0.7, 3.45); and (2) a bluff body representing the trailer with dimensions 1:1.3:4 in the x - y - z directions, respectively, and centered at (8, 0.85, 5.8). $16 \times 16 \times 16$, $10 \times 10 \times 6$ and $10 \times 14 \times 40$ uniformly distributed linear boundary elements were used to discretize the confining cube, the truck and the trailer, respectively. The no-slip boundary condition on the truck and the trailer was satisfied using $8 \times 8 \times 4$ and $8 \times 11 \times 32$ uniformly distributed tiles. For the confining cube, the free-slip boundary condition was imposed everywhere, with the exception of the region spanning ($6.5 < x < 9.5$, $y = 0$, $0 < z < 16$),

where 6×16 tiles were used to satisfy the no-slip condition. The trailer truck was assumed to be stationary. Furthermore, the following parameters were set: $\Delta t = 0.05$, $\text{BLRP} = 3$, and $\text{BLTC} = 2$ (See [2, 3] for the definition of the variables.) Time integration was performed by a second order predictor-corrector scheme. A total of 4 176 linear boundary elements were assigned and the number of vortex elements reached a plateau at 45 000. The elapsed time was approximately 3.5 minutes per computational timestep on a 256 node partition. The same run is estimated to take at least 35 minutes per timestep on the Cray C90. Note that 45 000 (vortex elements) is not large enough to realize the full parallelization potential of the 256 node partition, and the speed-up is expected to improve as the number of elements increases.

Figure 5 depicts the side view of the trajectories of the vortex elements in the plane of symmetry of the truck (on the left), as well as the top view of a plane just above the roof of the truck (on the right), at various timeslices. The planar cuts are centered in a slice of volume with thickness 0.15. Notice that, in general, the vortex elements generated on the top of the truck do not convect over to the trailer top. Instead, the flow seems to be partially relaxed into the gap between the truck and the trailer, but mostly forked out to the sides of the trailer. The latter quickly develops a necklace like vortex around the trailer (at the top, upstream section of the body) and slightly tilted towards the ground. The subsequent interaction of this vortex structure with the newly generated vortices on the trailer sides is complicated and requires a detailed analysis. Also, the development of an unsteady wake behind the trailer is noticeable. The eddy size is in the order of the trailer height. The wake height far downstream of the trailer seems to be dropping, instead of expanding upward. This is due to the spreading of the wake in the transverse direction.

4 Conclusion

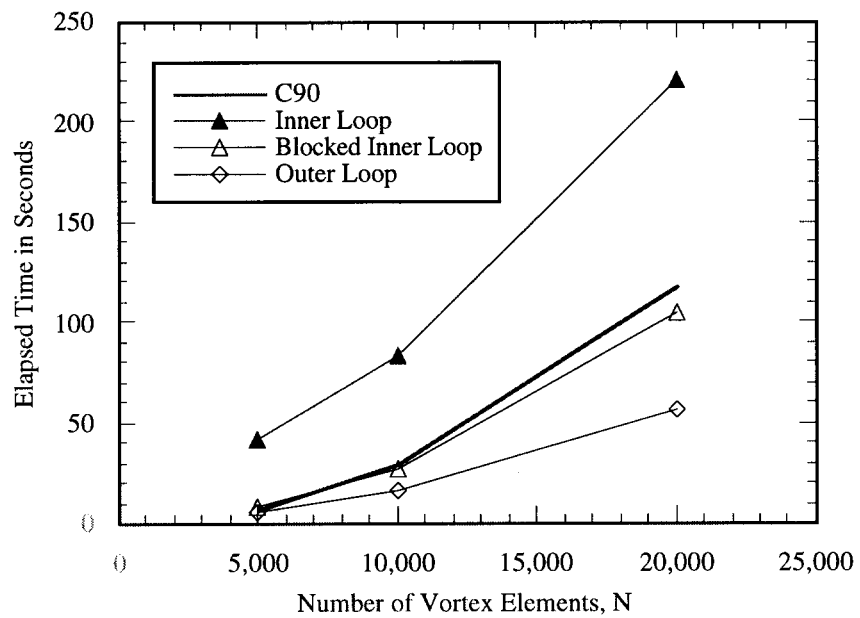
A parallel vortex-boundary element method has been developed for the simulation of three-dimensional wall-bounded flow on the Thinking Machines CM5. Various parallelization paradigms have been tested for the direct evaluation of the vortical and potential velocities (and their gradients,) and the fastest approach implemented in the code. Both vortical and potential flow evaluations displayed good parallelization efficiency and acceptable speed-up over the Cray C90. Consequently, vortex-boundary element simulation of three-dimensional flow involving up to 100 000 vortex elements is now feasible. The functionality of the code has been ascertained by the test problem of flow over an idealized trailer truck near the ground level at $Re = 500$. A total of 45 000 vortex elements and 4 176 linear boundary elements were used in this example. The average CPU time per computational timestep was 3.5 minutes on the 256 node partition, compared to the estimated 35 minutes required for the C90.

Acknowledgments

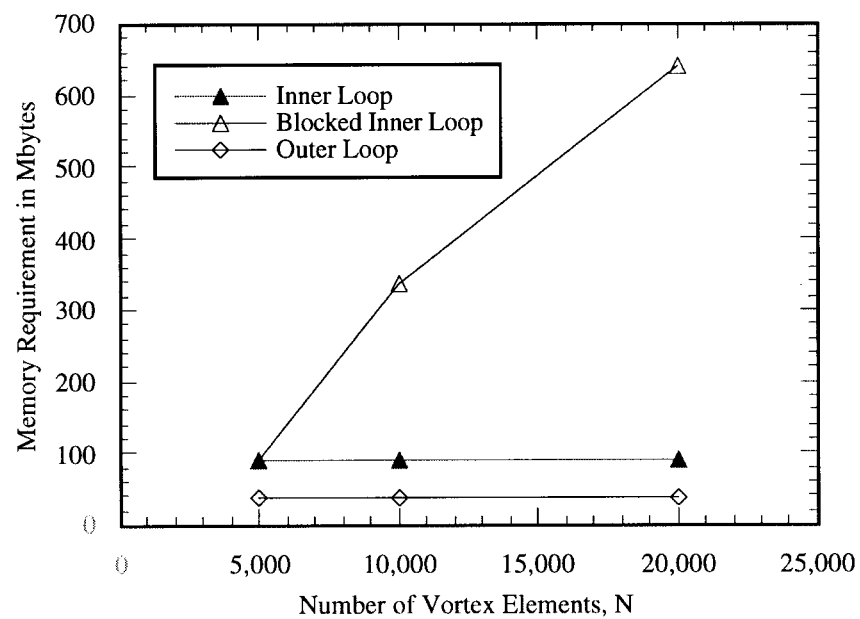
This project was funded by FORD Motor Company and Gas Research Institute. The numerical experiments were performed on the Thinking Machines CM5 at the National Center for Supercomputing Applications, NCSA. We wish to thank Prof. Melanie Loots of NCSA for her keen interest in this project and administrative support throughout the development process. We also thank Prof. Andrei Malevsky who served as a consultant for questions regarding the CM5, and contributed directly towards the conversion of the code.

References

- [1] Buttkke, T.F., “,” in *Vortex Dynamics and Vortex Methods, Lectures in Applied Mathematics*, **28**, edited by C. R. Anderson and C. Greengard, American Mathematical Society, 1990.
- [2] Gharakhani, A. and Ghoniem, A.F., “Three-Dimensional Vortex Simulation of Time Dependent Incompressible Internal Viscous Flows,” submitted to *J. Comp. Phys.*
- [3] Gharakhani, A. and Ghoniem, A.F., “Simulation of Three-Dimensional Internal Flows by the Random Vortex and Boundary Element Methods,” in *Vortex Flows and Related Numerical Methods*, Sec. Int’l. Workshop, Montreal, Canada, 1995.
- [4] Greengard, L. and Rokhlin, V., “A Fast Algorithm for Particle Simulations,” *J. Comp. Phys.*, **73**, pp. 325, 1987.
- [5] Koumoutsakos, P.D., Direct Numerical Simulations of Unsteady Separated Flows Using Vortex Methods, Ph.D. Thesis, Dept. of Mech. Eng’n., Caltech, Pasadena, CA, 1993.
- [6] Nabors, K., Kormsmeier, F.T., Leighton, F.T. and White, J., “Preconditioned, Adaptive, Multipole-Accelerated Iterative Methods for Three-Dimensional First-Kind Integral Equations of Potential Theory,” *SIAM J. Sci. Comp.*, **15 (3)**, pp. 713, 1994.
- [7] Sethian, J.A., “Two-Dimensional, Viscous, Incompressible Flow in Complex Geometries on a Massively Parallel Processor,” Thinking Machines Corp. Internal Report, CS90-6, 1990.



(a)



(b)

Figure 1: Performance comparison between three parallelization paradigms for evaluating the vortical velocity and its gradients, using a 32 node partition. (a) Elapsed time vs. the number of vortex elements. (b) Required memory vs. the number of vortex elements.

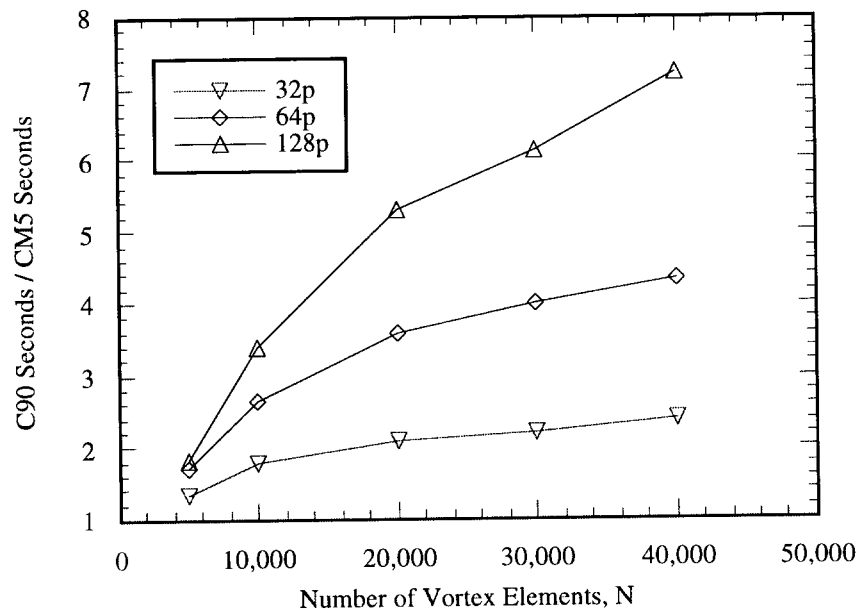


Figure 2: Parametric study of the performance of the outer loop algorithm in evaluating the vortical components as a function of the number of partitions.

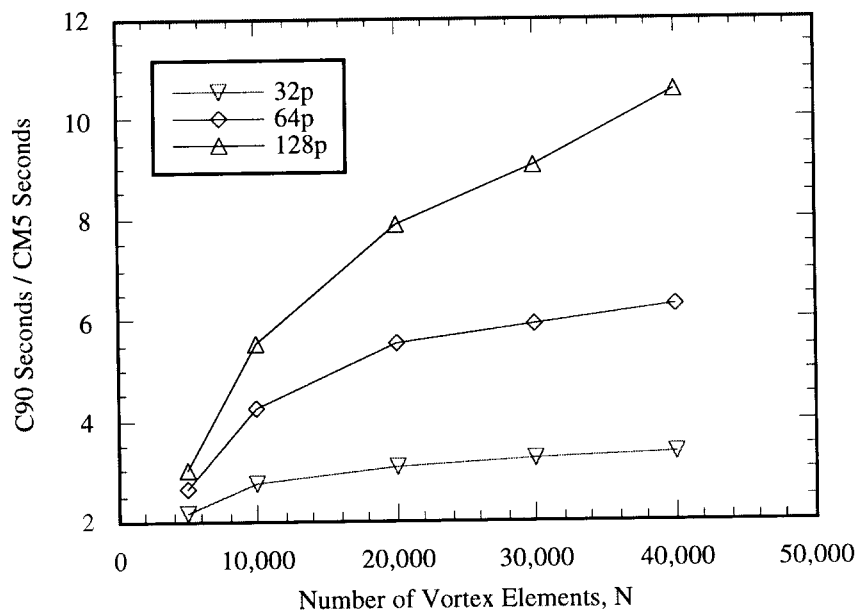


Figure 3: Parametric study of the performance of the outer loop algorithm in evaluating the potential components as a function of the number of partitions.

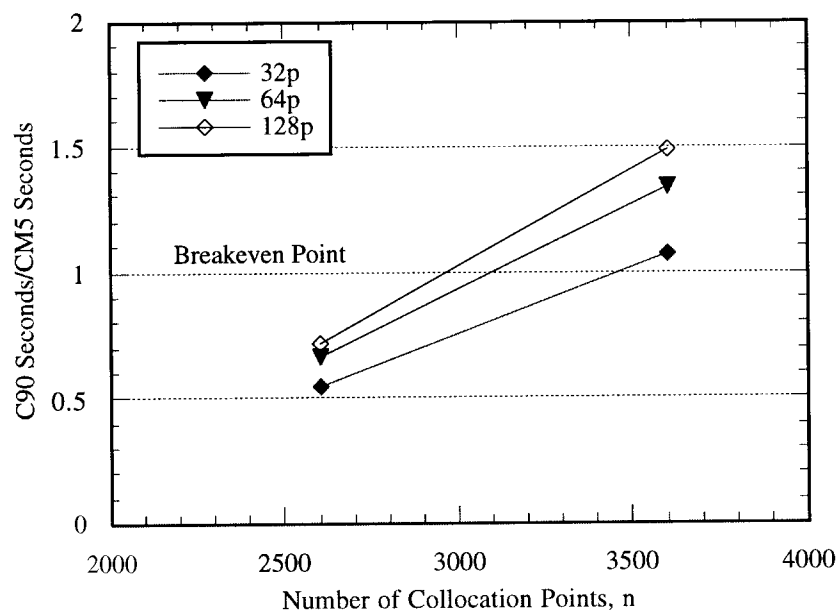


Figure 4: Parametric study for the LU decomposition of the fully populated matrix that results from the boundary element solution of the Neumann problem. The CM5 CMSSL library routine and the Cray standard library routine were used for this purposes.

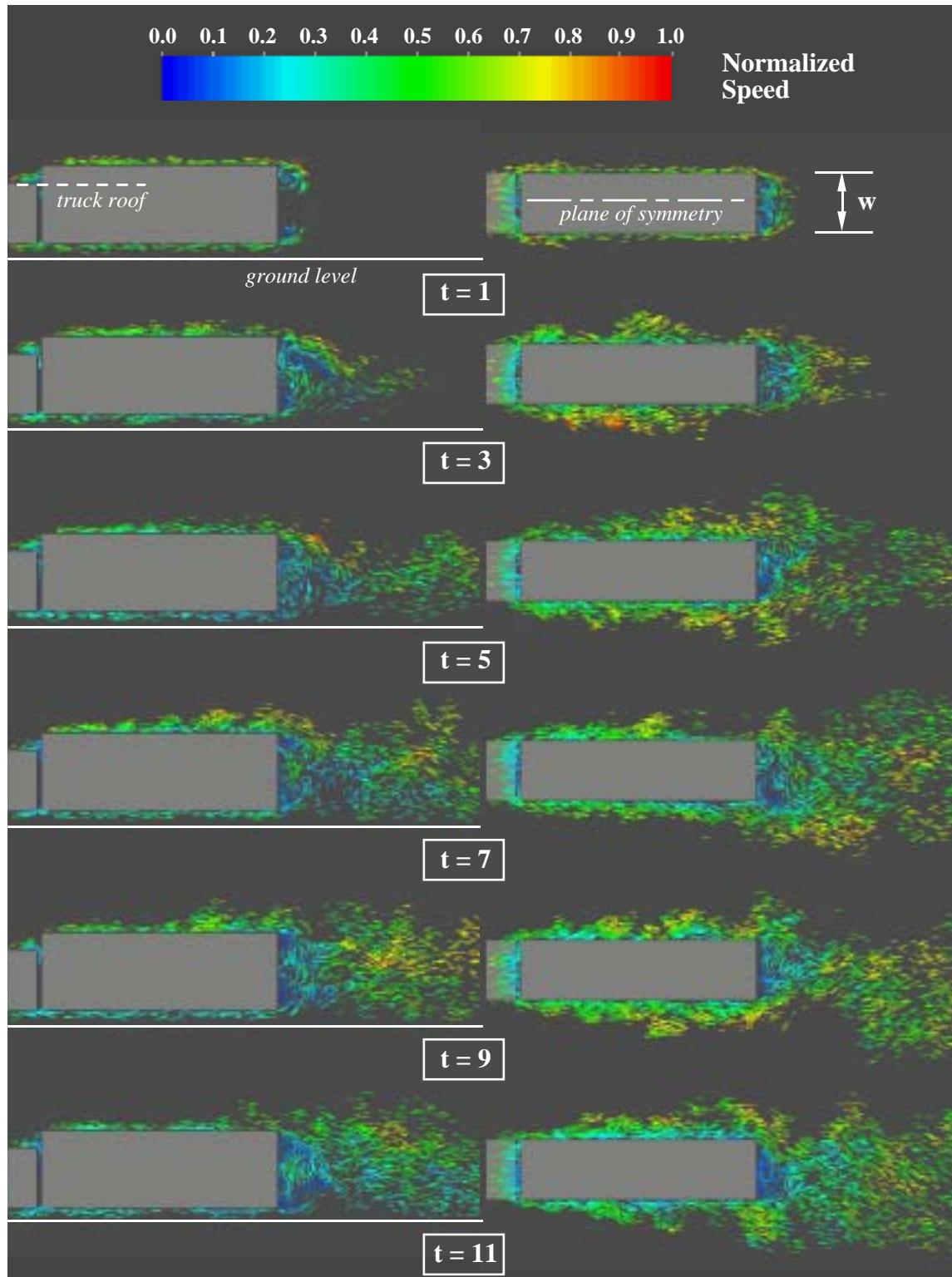


Figure 5: Vortex element trajectories within a volume of thickness $0.15w$, slicing through the plane of symmetry of the truck (left) and the truck roof (right). Velocity vectors are fixed in length. Normalization in each time frame is based on the instantaneous maximum speed in the field.