Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 207 (2005) 493–528
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical *N*-body simulations

Youssef M. Marzouk [1], Ahmed F. Ghoniem *

Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Room 3-342, Cambridge, MA 02139-4307, USA

Received 6 August 2004; received in revised form 24 January 2005; accepted 26 January 2005
Available online 13 April 2005

Abstract

A number of complex physical problems can be approached through *N*-body simulation, from fluid flow at high Reynolds number to gravitational astrophysics and molecular dynamics. In all these applications, direct summation is prohibitively expensive for large *N* and thus hierarchical methods are employed for fast summation. This work introduces new algorithms, based on *k*-means clustering, for partitioning parallel hierarchical *N*-body interactions. We demonstrate that the number of particle–cluster interactions and the order at which they are performed are directly affected by partition geometry. Weighted *k*-means partitions minimize the sum of clusters' second moments and create well-localized domains, and thus reduce the computational cost of *N*-body approximations by enabling the use of lower-order approximations and fewer cells.

We also introduce compatible techniques for dynamic load balancing, including adaptive scaling of cluster volumes and adaptive redistribution of cluster centroids. We demonstrate the performance of these algorithms by constructing a parallel treecode for vortex particle simulations, based on the serial variable-order Cartesian code developed by Lindsay and Krasny [Journal of Computational Physics 172 (2) (2001) 879–907]. The method is applied to vortex simulations of a transverse jet. Results show outstanding parallel efficiencies even at high concurrencies, with velocity evaluation errors maintained at or below their serial values; on a realistic distribution of 1.2 million vortex particles, we observe a parallel efficiency of 98% on 1024 processors. Excellent load balance is achieved even in the face of several obstacles, such as an irregular, time-evolving particle distribution containing a range of length scales and the continual introduction of new vortex particles throughout the domain. Moreover, results suggest that *k*-means yields a more efficient partition of the domain than a global oct-tree.

* Corresponding author. Tel.: +1 617 253 2295; fax: +1 617 253 5981.
E-mail addresses: ymarzou@sandia.gov (Y.M. Marzouk), ghoniem@mit.edu (A.F. Ghoniem).
[1] Present address: Sandia National Laboratories, Livermore, CA 94551, USA.

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.01.021

Keywords: *k*-means clustering; Treecode; *N*-body problems; Hierarchical methods; Parallel processing; Load balancing; Particle methods; Vortex methods; Three-dimensional flow; Transverse jet

1. Introduction

A number of complex physical problems can be approached through *N*-body simulation. High-Reynolds number flows computed with vortex methods [2] are one example. Other important applications range from gravitational astrophysics and cosmology [3] to smoothed particle hydrodynamics, molecular dynamics, non-Newtonian flows [4], and electrodynamics [5].

In all these applications, a dense system of pairwise particle interactions leads to a computational cost of $O(N^2)$, which is prohibitive for large N. Fast summation algorithms that reduce this cost to $O(N \log N)$ or $O(N)$ are necessary to achieve high resolution and realistic scale. Typically, these methods must contend with irregular particle distributions of non-uniform density; in dynamic *N*-body problems, the algorithms also face a particle distribution that evolves in time. Large, realistic physical problems require efficient implementation of these algorithms on massively parallel distributed memory computer architectures.

The present work employs hierarchical methods for fast summation. Hierarchical methods construct approximations for the influence of a cluster of particles and, where possible, use these approximations to replace pairwise particle interactions with a smaller number of particle–cell or cell–cell interactions. Based on the latter, these methods may be classified into treecodes (particle–cell interactions) [6,7] and fast multipole methods (cell–cell interactions) [8]. The focus here is on treecodes; a more comprehensive background on hierarchical methods is provided in Section 2.2.

The "quality" of spatial partitioning is central to the performance of a hierarchical method. The spatial partition determines cell moments and cell proximities (including neighbor relationships), and thus controls the number and order of particle–cell interactions necessary to achieve a given level of accuracy. For an efficient parallel implementation, one also must devise a spatial partition that is compatible with distributing hierarchical interactions over many processors.

In this paper, we introduce new algorithms, based on *k*-means clustering, for partitioning parallel hierarchical *N*-body interactions. The advantages of cluster partitions stem from their *geometric* properties. *K*-means partitions optimize cluster moments and other quantities that control the error bounds of a treecode, and thus reduce the computational cost of *N*-body approximations. The clustering procedure is inherently adaptive – an important feature for non-uniform distributions of particle positions and weights – and itself may be parallelized efficiently. All these features are preserved as the number of processors is scaled. Alternative algorithms for spatial partitioning of parallel treecodes – namely orthogonal recursive bisection (ORB) [9,3] or the hashed-oct-tree (HOT) algorithm [10] – do not yield similar geometric properties.

We demonstrate the parallel performance of clustering by constructing a parallel treecode for vortex particle simulations, based on the serial variable-order treecode developed by Lindsay and Krasny [1]. For simplicity, we do not focus on distributed data and the communications algorithms required to fetch non-local cell data efficiently. On a modern computer, locations, weights, and cell moments for up to 10^7 particles will fit on one processor's memory, so this problem becomes less important. We also note that the spherical domain geometries favored by clustering minimize surface area to volume ratios often associated with communications overhead, and thus may be advantageous to any distributed data implementations we develop in future work.

We also present new heuristics for dynamically load balancing cluster partitions. These techniques include dynamic scaling of cluster metrics and adaptive redistribution of cluster centroids. Load balance is always an issue in *N*-body problems with non-uniform particle distributions, but a unique impediment

to load balance in the present context is the continual introduction of *new* vortex elements. As detailed in Section 4, new element introduction is crucial to vortex simulation of turbulent flow: resolving the stretching of vortical structures and the resulting breakdown of the flow into small scales requires a continual remeshing of vortex filaments. We demonstrate the performance of load-balanced clustering on three-dimensional vortex simulations of a transverse jet [11,12].

2. Background

We begin by reviewing the fundamentals of vortex particle methods for fluid dynamics, illustrating how the formulation gives rise to a classical N-body problem. We then discuss hierarchical solvers that have been developed for efficient solution of such N-body problems in serial code.

2.1. Vortex methods and N-body problems

Vortex methods are a computational approach to systems governed by the Navier–Stokes or Euler equations, employing a particle discretization of the vorticity field and transporting vorticity along particle trajectories [2,13–16]. Originally conceived of for high Reynolds number flows [17] and for flows dominated by vortex dynamics [18], these methods have received significant attention over the past 30 years, maturing into tools for direct simulation, supported by several convergence results and a rigorous error analysis [19–21,2]. New particle methods for solving the diffusion equation, coupled with viscous splitting, have extended the applicability of vortex methods to flows of finite Reynolds number [22–25], enabling direct simulation of the Navier–Stokes equations, including boundary layer phenomena. We also mention a range of techniques for dealing with complex boundaries [26], as well as extensions to stratified flows [27], aeroacoustics [28], and reacting flows [29].

In all these contexts, vortex methods are attractive for their ability to simulate convection without numerical diffusion [17,2]. Also, inherent in the grid-free nature of the method is a dynamic clustering of computational points only where they are needed, i.e., over the small support of the vorticity field. The essence of these methods is the discretization of the vorticity field onto Lagrangian computational elements, or particles. In three dimensions, these particles have vector-valued weights $\boldsymbol{\alpha}_i(t) \equiv (\boldsymbol{\omega}\,\mathrm{d}V)_i(t)$ and trajectories $\boldsymbol{\chi}_i(t)$.

$$\boldsymbol{\omega}(\mathbf{x},t) \approx \sum_{i}^{N} \boldsymbol{\alpha}_i(t) f_\delta(\mathbf{x} - \boldsymbol{\chi}_i(t)). \tag{1}$$

The vorticity associated with each element is desingularized with a radially symmetric core function $f_\delta(\mathbf{r})$ of radius δ, where $f_\delta(\mathbf{r}) \equiv \delta^{-3} f\left(\frac{|\mathbf{r}|}{\delta}\right)$. The function f must be smooth and rapidly decreasing, satisfying the same moment properties as the Dirac measure up to order $m > 1$ [2].

The following background focuses on inviscid flows, for it is the *convective* step in vortex methods that embodies the computational challenges addressed in this paper. Equations of motion for inviscid, incompressible flow may be written in vorticity transport form, where $\boldsymbol{\omega} = \nabla \times \mathbf{u}$:

$$\frac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{D}t} = \boldsymbol{\omega} \cdot \nabla \mathbf{u}, \tag{2}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{3}$$

In this Lagrangian description, the right-hand side of (2) accounts for stretching and tilting of the vorticity as it is convected by the flow.

Using the Helmholtz decomposition of the velocity field, we write

$$\mathbf{u} = \mathbf{u}_\omega + \mathbf{u}_p, \tag{4}$$

where $\mathbf{u}_\omega$ is the curl of a vector potential ($\mathbf{u}_\omega = \nabla \times \boldsymbol{\psi}$) and $\mathbf{u}_p$ is the velocity of a potential flow ($\mathbf{u}_p = \nabla\phi$). Given a distribution of vorticity $\boldsymbol{\omega}$, the vortical velocity $\mathbf{u}_\omega$ may be recovered from the Biot–Savart law

$$\mathbf{u}_\omega(\mathbf{x}) = -\frac{1}{4\pi}\int_D \frac{(\mathbf{x}-\mathbf{x}')\times\boldsymbol{\omega}(\mathbf{x}')}{|\mathbf{x}-\mathbf{x}'|^3}\mathrm{d}\mathbf{x}' = \mathbf{K} * \boldsymbol{\omega}. \tag{5}$$

Here, $\mathbf{K}$ denotes the matrix-valued Biot–Savart kernel.

The above equations are closed by choosing a divergence-free potential velocity field to satisfy a prescribed normal velocity $\mathbf{n}\cdot\mathbf{u}$ on the boundary of the given domain D:

$$\begin{aligned} &\nabla^2\phi = 0, \\ &\mathbf{n}\cdot\nabla\phi = \mathbf{n}\cdot\mathbf{u} - \mathbf{n}\cdot\mathbf{u}_\omega \quad \text{on } \partial D. \end{aligned} \tag{6}$$

Together, Eqs. (2)–(6) completely specify the motion of an incompressible, inviscid fluid [30].

Given a regularized particle discretization of the vorticity field as in (1), the Biot–Savart law (5) may be rewritten as follows:

$$\mathbf{u}_\omega(\mathbf{x}) = \sum_i^N \mathbf{K}_\delta(\mathbf{x}, \boldsymbol{\chi}_i)\times\boldsymbol{\alpha}_i, \tag{7}$$

where the regularized kernel $\mathbf{K}_\delta$ results from convolution with the core function, $\mathbf{K}_\delta = \mathbf{K} * f_\delta$.

Vortex methods solve the inviscid equations of motion via numerical integration for the particle trajectories $\boldsymbol{\chi}_i(t)$ and weights $\boldsymbol{\alpha}_i(t)$. Computing particle trajectories $\boldsymbol{\chi}_i(t)$ requires evaluation of the velocity at each particle at every timestep. In three-dimensional vortex methods, one must also evaluate velocity gradients in order to compute the vortex stretching term, whether through a finite-difference operator along the local vorticity vectors or through differentiation of the Biot–Savart kernel, thus incurring additional computational cost. As each vortex element induces a velocity on every other vortex element, this is an N-body problem; direct evaluation of (7) at every element yields a computational cost of $O(N^2)$. For large numbers of particles, this clearly can be prohibitively expensive.

The $O(N^2)$ bottleneck is not unique to vortex methods; indeed, it is a feature inherent to N-body problems in a variety of contexts, whether the result of summation or quadrature (as in (7) or (5)) is a velocity, a force, or a potential. Gravitational N-body simulations are an essential tool in astrophysics, where they are used to study galaxy dynamics and cosmological structure formation [31,9]. Here, as in vortex methods, large N is essential to resolve fine features and the necessary large scales [3]. N-body problems are also encountered in smoothed particle hydrodynamics [32,33] and plasma physics. Coulomb potentials and other, more complicated short-range potentials give rise to N-body problems in molecular dynamics [34,35], with increasingly important biological applications [36,37]. Overcoming the $O(N^2)$ bottleneck is thus essential to progress across a variety of scientific fronts.

2.2. Hierarchical methods

Hierarchical methods for N-body problems construct approximations for the influence of a cluster of particles and, where possible, use these approximations to replace particle–particle interactions with a smaller number of particle–cluster or cluster–cluster interactions. The construction of these approximations is typically organized by a recursive tree structure. Treecodes, introduced for gravitational problems by Appel [6] and Barnes–Hut [7], organize a group of N particles into a hierarchy of nested cells, e.g., an oct-tree in three dimensions. At subsequent levels of the tree, each "parent" cell is divided into smaller "child" cells representing finer spatial scales.

Treecodes have found wide application in particle methods. The original Barnes–Hut (BH) algorithm employs an oct-tree with a monopole moment calculated at each cell. Tree construction proceeds until leaf nodes each contain only a single particle. The tree is traversed once for every particle using a divide-and-conquer strategy of particle–cell interactions; if the monopole approximation at a given cell cannot provide the force on the target particle to a sufficient level of accuracy, the contribution of the cell is replaced by the contribution of its child cells. The total computational cost scales as $O(N \log N)$. Variations on this treecode algorithm have been numerous; broadly speaking, these differ in terms of physics – i.e., the kernel describing the influence of each particle [10,35,36,5] – and in the *type* and *order* of series approximation used to describe the influence of a cluster [38,39,1]. Other improvements encompass adaptive features of the tree construction [40] and more sophisticated error estimates [31,41].

Fast multipole methods (FMM), introduced by Greengard and Rokhlin [8,42], employ additional analytical machinery to translate the centers of multipole expansions and to convert far-field multipole expansions into local expansions, reducing the total operation count to $O(N)$. Like many BH-type codes, these codes also use higher-order approximations, typically a multipole expansion involving spherical harmonics in three dimensions [43,44], although additional schemes and different bases have been proposed [39,36,45,46].

2.2.1. Lindsay–Krasny treecode

Lindsay and Krasny have introduced a BH-type treecode with many adaptive features well-suited to vortex particle methods [1]. This serial code provides a convenient platform on which to develop and test the clustering and load-balancing algorithms described in this paper, so we will review its essential features. The Lindsay–Krasny (LK) code organizes particles in an oct-tree. Adaptive features of the tree include non-uniform rectangular cells that shrink to fit their contents at every level of the tree and a leaf size parameter N_0 below which a cell is not divided. The velocity induced by each particle is given by the Rosenhead–Moore kernel, a regularized form of the Biot–Savart kernel; in vortex methods, this regularization is also known as the low-order algebraic smoothing [47].

$$\mathbf{K}_\delta(\mathbf{x},\mathbf{x}') = -\frac{1}{4\pi}\frac{\mathbf{x}-\mathbf{x}'}{(|\mathbf{x}-\mathbf{x}'|^2+\delta^2)^{3/2}}. \tag{8}$$

Because this kernel is not harmonic, it cannot be expanded in a classical multipole series; instead the treecode employs a Taylor expansion in Cartesian coordinates to approximate the influence of each cell at a target point $\mathbf{x}$:

$$\mathbf{u}(\mathbf{x}) \approx \sum_{i=1}^{N_c}\sum_{\mathbf{k}}^{\|\mathbf{k}\|<p}\frac{1}{\mathbf{k}!}\mathbf{D}_{\mathbf{y}}^{\mathbf{k}}\mathbf{K}_\delta(\mathbf{x},\mathbf{y}_c)(\mathbf{y}_i-\mathbf{y}_c)^{\mathbf{k}}\times\boldsymbol{\alpha}_i \tag{9}$$

$$\approx \sum_{\mathbf{k}}^{\|\mathbf{k}\|<p}\mathbf{a}_{\mathbf{k}}(\mathbf{x},\mathbf{y}_c)\times\mathbf{m}_{\mathbf{k}}(c), \tag{10}$$

where $\mathbf{y}_c$ is the coordinate of the cell centroid, N_c is the number of particles in the cell, $\mathbf{y}_i$ are their coordinates, $\mathbf{k} = (k_1, k_2, k_3)$ is an integer multi-index with all $k_i \geqslant 0$,

$$\mathbf{a}_{\mathbf{k}}(\mathbf{x},\mathbf{y}_c) = \frac{1}{\mathbf{k}!}\mathbf{D}_{\mathbf{y}}^{\mathbf{k}}\mathbf{K}_\delta(\mathbf{x},\mathbf{y}_c) \tag{11}$$

is the $\mathbf{k}$th Taylor coefficient of the Rosenhead–Moore kernel at $\mathbf{y} = \mathbf{y}_c$ and

$$\mathbf{m}_{\mathbf{k}}(c) = \sum_{i=1}^{N_c}(\mathbf{y}_i-\mathbf{y}_c)^{\mathbf{k}}\boldsymbol{\alpha}_i \tag{12}$$

is the $\mathbf{k}$th moment of cell c about its center. Taylor expansions are computed to arbitrary order p up to a user-specified maximum order of approximation $p_{\max}$; a typical choice is $p_{\max} = 8$. A recurrence relation allows the Taylor coefficients $\mathbf{a}_{\mathbf{k}}$ to be computed cheaply (each successive coefficient in O(1) operations)

for each particle–cluster interaction. Cell moments, on the other hand, are computed as needed for each cell then stored for use in subsequent interactions. Unlike other vortex particle treecodes, the LK treecode incorporates the regularization of the kernel directly into the expansion [48,49].

The velocity at each target particle is evaluated using a "divide-and-conquer" strategy governed by a user-specified accuracy parameter ϵ and an error estimate derived from the approximation error for the vector potential:

$$\epsilon \geqslant \frac{M_p(c)}{4\pi R^{p+1}}, \tag{13}$$

where

$$M_p(c) = \sum_{i}^{N_c} |\mathbf{y}_i - \mathbf{y}_c|^p |\boldsymbol{\alpha}_i| \tag{14}$$

is the pth absolute moment of cell c and $R = (|\mathbf{x} - \mathbf{y}_c|^2 + \delta^2)^{1/2}$ is the regularized distance between the target particle and the cell center. Velocity evaluation for each target particle begins at the root cell and proceeds recursively. For each cell c encountered, the code computes the minimum order of approximation p that satisfies inequality (13). If this $p < p_{\max}$, the particle–cell interaction is evaluated for the cell c; otherwise the velocity evaluation descends the hierarchy and sums the velocities induced by the children of cell c. This procedure is modified by a run-time choice between Taylor expansion and direct evaluation at each cell, which may become active at lower levels of the hierarchy; if the estimated time for direct summation is smaller than the estimated time for Taylor expansion, the former method is used to compute the influence of the cell.

Some of our ongoing work [50] develops recurrence relations for other regularizations of the Biot–Savart kernel, including the higher-order algebraic smoothing proposed in [47].

3. Computational approach

Efficient parallelization of an algorithm depends on avoiding the duplication of work among processors, ensuring equal workload at each processor, and minimizing additional costs, such as the time for inter-processor communication and domain decomposition. In the case of treecodes for particle methods, a good spatial decomposition is essential to all of the goals just mentioned.

3.1. Partition geometry in parallel N-body problems

We begin by considering *why* partition geometry is important to hierarchical N-body solvers – that is, the mechanisms by which partition geometry affects computational cost. We likewise seek to describe, qualitatively, what constitutes a "good" partition. In later sections, we will discuss the algorithms used to compute our domain decomposition (Section 3.2) as well as the quantitative features of good spatial partitions (Section 3.3).

In the following discussion, we make a distinction between "source" particles and "target" particles. Targets are the points at which the velocity or force is computed; sources are the particles, or quadrature points, inducing the velocity or force. In most situations – e.g., a vortex element code or a gravitational N-body simulation – the source sets and target sets are exactly the same, and each particle simply takes turns in either role. For simplicity, in the following discussion we let the set of source particles be identical to the set of target particles.[2]

[2] In a vortex filament code, the two sets may differ slightly depending on the quadrature rule used along the filament coordinate; the targets (filament nodes) may be staggered with respect to the sources (element centers). In this case, however, the displacement between members of the source and target sets is less than half a core size and relatively negligible in discussing cluster geometry.

A given domain decomposition method admits many schemes for parallelizing the treecode calculations. Consider first the possibility of using a global tree. This process may be driven by "parallelizing over targets." By this we mean that the domain decomposition scheme yields a certain partition of particles and that each processor is responsible for computing the influence of the entire domain on the particles assigned to it by the partition. A global tree is thus constructed on each processor, but only in structure: Cells subdividing the root cell are included only if they will be requested during tree traversal for velocity evaluation. Because velocity is evaluated only for the assigned particles, the resulting set of tree cells with filled-in moments (to varying order) is in fact the *locally essential tree* [9,51]. Each locally essential tree *is* a global tree, describing the influence of the entire domain on the assigned target particles.

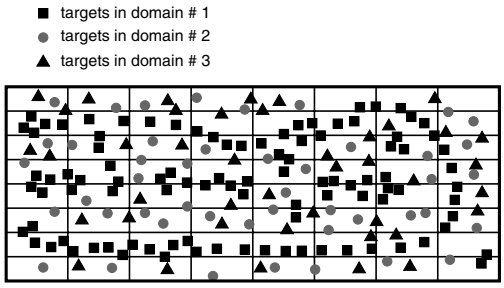
The topology of the locally essential tree and the methods by which it is constructed depend strongly on the domain decomposition scheme and on the overall parallel implementation. Section 3.5 discusses these parallel implementation issues in detail.

Regardless of the global tree's topology, the fundamental issues of geometry in domain decomposition are the same. Contrast cases of good and bad partition over targets, illustrated in Fig. 1 for three processors. In the worst case, case (a), the partitions are interleaved; that is, the convex hull of particles assigned to one processor overlaps with the hull of points on another processor. Even if this is not the case – consider, for instance, long and narrow non-overlapping domains as in Fig. 1(b) – the locally essential trees on different processors can still overlap strongly. This means that computations evaluating the influence of cells deeper in the hierarchy and at higher order are duplicated across processor domains, and parallel efficiency will be poor. To minimize the overlap of locally essential trees, one should seek better-separated domains with minimal surface area to volume ratios, thus favoring more spherical, non-overlapping partitions as in Fig. 1(c). Note that in this discussion, the concept of "overlap" characterizes not the spatial extent of trees but the duplication of locally essential cells at each level of the tree hierarchy.

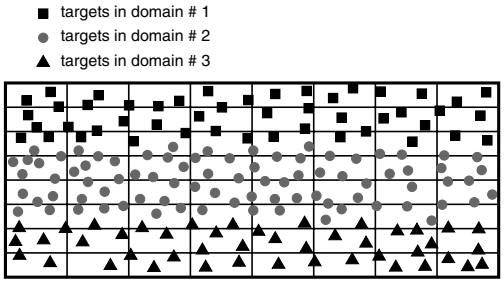
An alternative approach to parallelization avoids constructing a global tree (i.e., a locally essential tree), and instead can be viewed as "parallelizing over sources." Here, the domain decomposition scheme is applied to the particles and a local, possibly adaptive, oct-tree is constructed in each processor's domain, as shown schematically in Fig. 2. The target particles are left unorganized, and each processor computes the influence of its source tree on the entire set of target particles. Global reduction operations then sum the contribution of each processor to the velocity at every target.

With this approach, the geometric considerations governing good domain decomposition are analogous to those described before. Cases of good and bad source partition are shown schematically in Fig. 2 for three processors. Again, the worst case partition is one in which source points belonging to the three processors are interleaved. Though these sets of points are interleaved in space, they belong to distinct local trees, and thus their influence must be approximated with up to three times as many particle–cluster interactions as necessary (k times in the case of k interleaved partitions). But the situation persists even in the case of non-overlapping partitions. Consider the partition shown in Fig. 2(b). The source domains are long and narrow, and they constrain the shape of each local oct-tree accordingly. One can enumerate Nk pairs of source domains interacting with target particles; *few* of these pairs specify targets that are well-separated from sources. The closer a target particle lies to a source domain, the more expensive the interaction; evaluation of the velocity induced on the target will descend to cells deeper in the local source hierarchy and/or employ higher-order expansions. The relative lack of well-separated domain-target pairs is equivalent to noting that the surface-area to volume ratio of each source domain is large, compared to the partition in Fig. 2(c). Here, the domains are more compact, and the domain boundaries are more nearly spherical. As a result, velocities at the targets may be computed with fewer particle–cluster interactions.

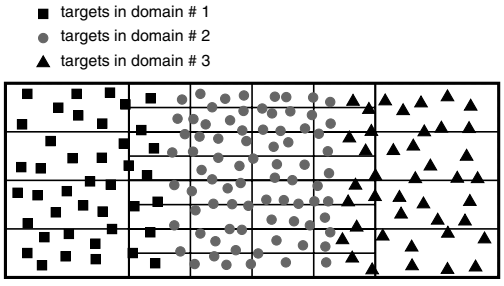
The qualitative discussion above emphasizes the critical role of *partition geometry* in N-body problems, whether the partition is used to separate target particles or to fix the root cells of local source trees. For identical sets of particle distributions and weights, the partition geometry directly determines the number



(a) Interleaved.



(b) Non-interleaved.



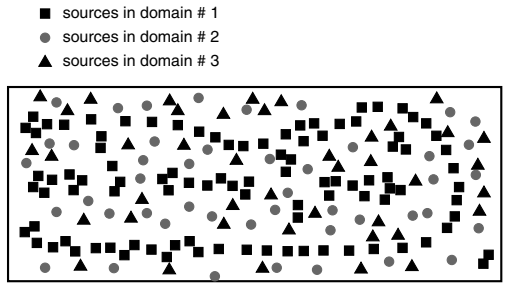
(c) Well-localized.

Fig. 1. The geometry of partitions over target particles, illustrated with three processors. Solid lines outline the locally essential tree for domain #2.

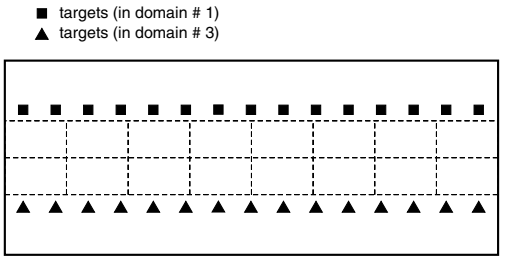
and order of particle–cluster interactions necessary to evaluate the velocity at each particle, summed over all domains.

3.2. K-means clustering

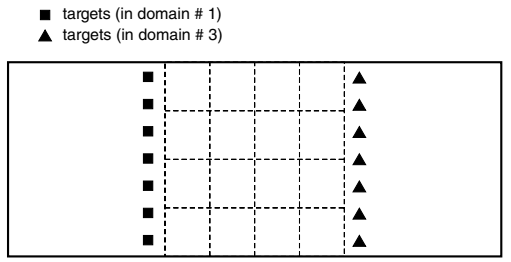
We propose a new approach for parallel domain decomposition of vortex particles, based on *k-means clustering* of the particle coordinates. Clustering procedures are essential tools for multivariate statistical analysis, data mining, and unsupervised machine learning [52,53]; *k*-means clustering [54] is a classical algorithm in these contexts. In the new context of domain decomposition, however, we develop a variant of the *k*-means algorithm yielding a partition with many desirable properties.



(a) Interleaved.



(b) Non-interleaved.



(c) Well-localized.

Fig. 2. The geometry of partitions over source particles, illustrated with three processors. Dashed lines represent the local source tree for domain #2. Note that the quad-tree in (b) employs an adaptive bisection to control the aspect ratio of its cells, but still demonstrates the larger number of particle–cluster interactions that accompany poor domain geometry.

K-means takes a set of N observations $\{\mathbf{x}_i\}$ in d-dimensional space as input and partitions the set into k clusters with centroids $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$, where k is prescribed. The partition is chosen to minimize the cost function

$$J = \sum_{i=1}^{N} \min_{k'} \left(|\mathbf{x}_i - \mathbf{y}_{k'}|^2 w_i \right) = \sum_{j=1}^{k} \sum_{i}^{N_j} |\mathbf{x}_i - \mathbf{y}_j|^2 w_i, \tag{15}$$

where w_i is a scalar weight associated with each observation. In other words, each observation is assigned to the nearest centroid, and the centroid positions are chosen to minimize the weighted within-cluster sum of squared Euclidean distances. In our implementation, for reasons that will be made clear below, we weigh each particle's contribution by its vorticity magnitude; in other words, we set $w_i = |\boldsymbol{\alpha}_i|$. *K*-means results in a *flat* or non-hierarchical clustering, in contrast to other clustering algorithms that construct hierarchical partitions, either from the bottom up (agglomerative) or the top down (divisive) [53].

The k-means algorithm can be viewed as an iterative optimization procedure for the cost function defined in (15), beginning with a choice of centroids $\{\mathbf{y}_{i=1,\ldots,k}\}$ and iteratively updating them to reduce J. K-means will find a local minimum of J, and thus the solution may depend on the initial choice of centroids; the problem of finding a global minimum is in fact NP-complete. We implement a "batch" version of the k-means algorithm, in which each particle is assigned to its closest centroid before the centroids are updated, at each iteration. This is in contrast to the "online" approach, in which the centroid locations are updated as each particle is individually classified [55]. In either case, the resulting classification boundaries are the Voronoi tesselation of the cluster centroids, and thus they bound convex subsets of $\mathbb{R}^d$.

An outline of the algorithm, using vorticity magnitudes $|\boldsymbol{\alpha}_i|$ as particle weights, is as follows:

Algorithm. *K*-means clustering

initialize N, k, $\mathbf{y}_1, \ldots, \mathbf{y}_k$
do $l = 1$ to $l_{\max}$
  assign each particle $\mathbf{x}_i$ to cluster $k_i^* = \underset{k'}{\operatorname{argmin}} (|\mathbf{x}_i - \mathbf{y}_{k'}|^2)$
  put each $\mathbf{y}_{k'} = \frac{\sum_i^{N_{k'}} |\boldsymbol{\alpha}_i| \mathbf{x}_i}{\sum_i^{N_{k'}} |\boldsymbol{\alpha}_i|}$, where $N_{k'}$ is the number of particles assigned to cluster k'
  recompute $J^{(l)}$
until $J^{(l-1)} - J^{(l)}$ small
return centroids $\mathbf{y}_1, \ldots, \mathbf{y}_k$ and memberships $\{k_i^*\}_{i=1,\ldots,N}$

The computational complexity of this algorithm is $O(Nk\,dT)$, where T is the number of iterations and d is the dimension of the space containing the observations, i.e., $\mathbf{x}_i \in \mathbb{R}^d$. It is straightforward to implement k-means clustering in parallel, however, and we do so using the parallel implementation proposed by Dhillon and Modha [56]. Since we typically seek a number of clusters k equal to the number of processors, an ideal parallelization reduces the computational complexity to $O(N\,dT)$. This scaling is what we find in practice, as will be shown in Section 4. Recent work has demonstrated new algorithms for fundamentally accelerating k-means clustering, using kd-trees to reduce the number of nearest-neighbor queries [57]. We do not pursue these approaches here, but note that they may become useful in ensuring that the time for parallel domain decomposition ($O(N)$ with parallel k-means) remains a small fraction of the time required for velocity evaluation ($O(N/k \log N)$ in the ideal case) for very large k.

It is worthwhile to note that the partition of vortex elements resulting from the clustering procedure may bear no relation to the data structure elsewhere used to represent the vortex particles in memory. This is particularly relevant to vortex filament methods, in which an ordering, or connectivity, between neighboring elements must be preserved. In this case, it may be necessary to maintain separate data structures or attribute lists, one appropriately representing filament connectivity, and another encoding the flat k-means partition, which considers the vortex elements as a set of completely independent particles.

3.3. Towards optimal geometry of clusters

We use k-means clustering to construct a partition of the source particles, "parallelizing over sources" as described in Section 3.1. A local adaptive oct-tree is then constructed from each processor's assigned particles and the velocities induced by each processor's source tree are summed at each target. The root cell of each source tree is thus a k-means cluster, as illustrated in Fig. 3. Because these clusters minimize the cost function J in (15), their boundaries define tightly localized, convex sets. Based on the preceding discussion, this partition geometry should favor smaller numbers and lower orders of particle–cluster interactions, for greater parallel efficiency.

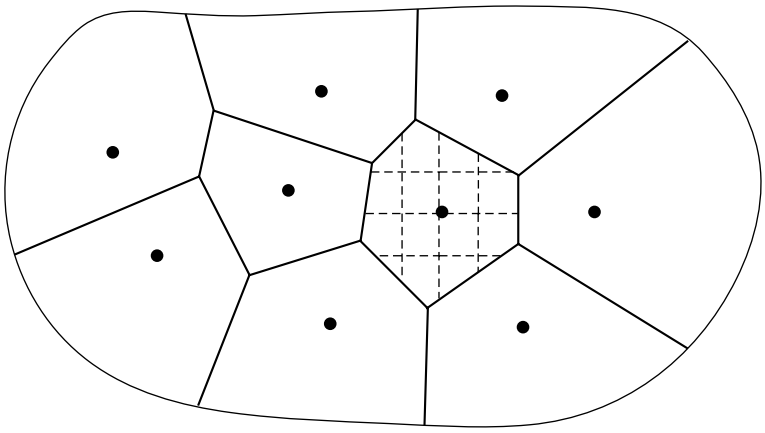


Fig. 3. Schematic of k-means partition with a local source tree. Solid lines are k-means cluster boundaries, defining the Voronoi tesselation of the cluster centroids (solid circles). Dashed lines represent the local (quad-)tree for a single domain.

Considerations of optimal source geometry can be made yet more precise, however. The numerator in the error criterion of the LK treecode, (13), is the pth absolute moment of cell c, $M_p(c)$. Note the correspondence between this sum and the cost function in (15). Our vorticity-weighted k-means algorithm finds a partition yielding a local minimum of $\sum_k^K M_2(c_{0,k})$ where $c_{0,k}$ are the resulting K root cells. While this flat partition cannot minimize individual absolute moments $M_2(c_{0,k})$, it will tend to make each one small. Furthermore, while the minimization of $\sum M_p$ strictly occurs for $p = 2$, $\sum M_p$ will generally remain small for other values of p, with possible exception of pathological cases.

Error estimates containing absolute moments of the form $M_p(c)$ are not limited to the LK treecode. In fact, they are a general feature of multipole expansions [31,10]. For multipole expansions of the singular Biot–Savart kernel employing spherical harmonics, Winckelmans et al. [48,10] report the following error bound:

$$e_p(\mathbf{x}) \leqslant \frac{1}{(d-b)^2}\left[(p+2)\frac{B_{p+1}}{d^{p+1}} - (p+1)\frac{B_{p+2}}{d^{p+2}}\right], \tag{16}$$

where $e_p(\mathbf{x})$ is the L_2 error on $\mathbf{u}_\omega$ at the evaluation point $\mathbf{x}$, $d = |\mathbf{x} - \mathbf{y}_c|$, b is the radius of the smallest sphere centered at x_c and containing all the particles in the cell, and

$$B_p(c) = \int_c \|\mathbf{x}' - \mathbf{y}_c\|^p \|\boldsymbol{\alpha}'\| \mathrm{d}\mathbf{x} \approx M_p(c). \tag{17}$$

In this case, the error depends not only on the cell moment but on the effective cell radius b, another quantity that will generally be small in a k-means partition.

In all these cases, error bounds or error criteria directly control the computational cost of the treecode by affecting the order of expansion (in a variable-order code) and the choice of cells used to sum the velocity at each target. When the inequalities in (13) and (16) cannot be met, evaluation of the velocity on a target particle must be performed at higher order or descend to the children of the cell; in the latter case, a single particle–cluster interaction may be replaced with up to eight interactions. (In the limit, tree descent typically devolves into direct summation; criterion for this depend on the structure of the particular treecode.) Reducing the cell moment $M(c)$ and cell radius b allows an error criterion to be met for *smaller* cell-to-target distances R or d, avoiding the need to increase the order of expansion p or descend further into the hierarchy.

In the present implementation, k-means clustering determines the configuration of each root cell. The geometries of child cells in the local hierarchy are strongly influenced by the root, however. In other words,

it is reasonable to expect a good root cell geometry to maintain small cell moments and radii several levels into the hierarchy. We explore the impact of the depth of local trees on performance in Section 4.

For clarity, we have thus far focused our analysis on computational time, not on the communication time associated with retrieving non-local cell or particle data. In fact, our current implementation keeps copies of all the particle positions and weights on each processor; on a modern computer, this allows for problem sizes of up to $N = 10^7$ and thus does not constrain the present vortex simulations. But we concentrate our discussion as stated above for more fundamental reasons. First, we view the relationship of partition geometry to computational time as more fundamental, inherent to the accuracy and error bounds of multipole expansions – e.g., the distance from a target to a cell center and the magnitude of the multipole moments of the cell. Communication time is typically more implementation-dependent – it can depend on the hardware interconnect or on latency-hiding features of the message passing architecture – and in N-body problems is usually much smaller. Secondly, computational time and communication time are not really separable issues. Minimizing the number and order of particle–cluster interactions has the simultaneous benefit of minimizing time spent on interprocessor communication, sending and receiving the cell or particle data. Also, k-means clustering yields nearly spherical domains, which in turn minimize the surface area to volume ratios often associated with communication overhead.

As described in Section 2.2, the serial LK treecode uses a user-specified accuracy parameter ϵ to control the velocity error at each target point, induced by all the sources. In order to maintain the same global error specification in the present implementation, the global accuracy parameter ϵ must be distributed to each processor's local tree. We take the following approach, patterned after the fractional distribution of ϵ from parent cells to child cells inside the LK oct-tree [1]:

$$\epsilon(c) = \frac{M_0(c)}{M_0}\epsilon, \tag{18}$$

where $M_0(c)$ is the 0th absolute moment of cluster c and $\epsilon(c)$ is the accuracy parameter assigned to the oct-tree in cluster c. In other words, the global accuracy parameter is distributed to each k-means cluster c in proportion to its total vorticity magnitude.

A few other approaches to domain decomposition of treecodes will be reviewed in Section 3.5.

3.4. Dynamic load balancing

While k-means partition yields domain geometries that favor reduced computational cost, this partition comes with no guarantee of load balance. Load balance is a difficult issue in N-body problems. Irregular particle distributions, often with a wide range of particle densities, are typical and thus equipartition (ensuring the same number of particles in each domain) does not ensure load balance. Contrast, for instance, the cost of computing the velocity at a target that is well-separated from other particles to the cost of computing the velocity at a target in a densely populated region; clearly, more particle–cluster interactions must be used to compute the latter [10]. Indeed, it is difficult to define or estimate a spatial distribution of "per-particle-cost" a priori, as this quantity depends both on the particle distribution and on how the distribution is partitioned.[3]

[3] The concept of "per-particle-cost" is perhaps most meaningful when considering each particle's role as a target, for then the time or number of interactions required for velocity evaluation at each particle is directly measurable, though even this measurement may be disrupted by one-time costs like the fill-in of multipole moments or the retrieval of faraway data to build locally essential trees. When considering partition over sources, the "per-particle-cost" becomes somewhat more ill-defined as the influence of each particle is replaced by multipole expansions of source cells. Allowing variable orders of expansion could make this per-particle cost even less consistent.

In dynamic N-body problems, particle locations and weights change in time, and thus it is advantageous to repartition the domain and re-balance computational loads as the particle distribution evolves. Vortex particle methods render the load balancing process more challenging because of local mesh refinement; at each timestep, new particles are introduced throughout the domain in order to maintain resolution and core overlap [14,13,58]. The spatial distribution of the newly inserted particles is itself highly irregular and difficult to predict.

To address these difficulties, we develop several heuristics for the dynamic load balancing of k-means clusters. The first of these introduces scaling factors s_k into the weighted k-means cost function along with a rule to adapt their values in time:

$$J = \sum_{i=1}^{N} \min_{k'} \left(s_{k'} |\mathbf{x}_i - \mathbf{y}_{k'}|^2 |\boldsymbol{\alpha}_i| \right). \tag{19}$$

The factors s_k scale the squared Euclidean distance between each cluster centroid and the surrounding particles, and thus modify the assignment at each k-means iteration. Each particle is assigned to the centroid from which its *scaled* distance is smallest. The space around each centroid is effectively "zoomed" in or out by the scaling factor. The resulting classification boundaries are no longer planes equidistant from the nearest two centroids, as in a Voronoi tesselation; instead, they are curved shells shifted closer to the centroid with larger s_k. Fig. 4 illustrates the geometry of these scaled k-means clusters. Cluster boundaries are now the *multiplicatively weighted* Voronoi tesselation of the centroids [59].

To dynamically load balance the cluster populations, we update the scaling factors s_k at the start of each timestep. In general, the adaptation rule for scaling factors should express dependence on the previous timestep's scalings s_k^n as well as the times t_k^n spent evaluating the influence of each cluster's particles on the whole domain. Here, the superscript n denotes the preceding time layer and $n+1$ is the current time.

$$s_k^{n+1} = f\left(s_k^n, t_k^n; k = 1, \ldots, K\right). \tag{20}$$

In the present implementation, we choose a simple case of this adaptation rule, multiplicatively updating each s_k based on each cluster's deviation from the mean source evaluation time $\bar{t}^n = \sum_k t_k^n / k$.

$$s_k^{n+1} = s_k^n \left(1 + \alpha \tanh\left(\beta \frac{t_k^n - \bar{t}^n}{\bar{t}^n} \right) \right). \tag{21}$$

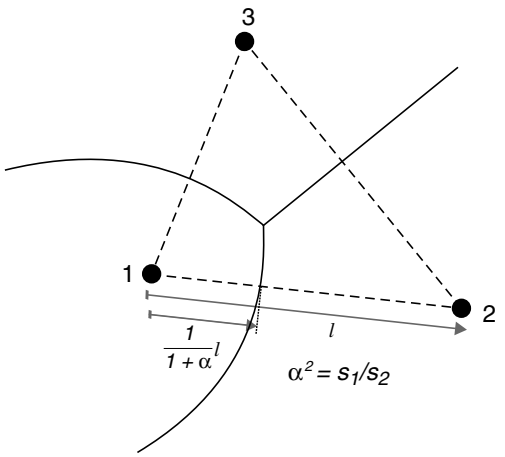


Fig. 4. Two-dimensional schematic of scaled k-means cluster boundaries, with three clusters. The numbered solid circles are cluster centroids. Clusters have scaling factors s_i; here, $s_1 > s_2 = s_3$.

In this sense, we are using the *relative time* required to evaluate the velocity due to all the sources in clusters at the previous timestep as an estimate of the relative time required by a similarly composed cluster at a nearby position in space. Cluster boundaries and centroids will change from timestep to timestep, as will the actual memberships – due to movement of particles, evolution of particle weights, and new particle introduction – but these changes should be incremental.

After each iteration, s_k will increase for a "high-cost" cluster and vice-versa. Expensive clusters will thus lose particles to their neighbors, while clusters with below-average t_k will incrementally become more attractive. The parameters α and β can be tuned; we find that $\alpha = 0.1$ and $\beta = 2$ give good performance. We also impose safety bounds to avoid unreasonable values of the scaling factors; after each application of (21) we require that $0.25 < s_k < 4.0$.

A new k-means partition is computed at each timestep, immediately after updating the cluster scalings. At the first timestep, all the s_k are set to unity and initial guesses for the centroids $\mathbf{y}_k$ are randomly chosen from among the particle locations. At subsequent timesteps, converged centroid locations from the preceding step are used as initial guesses for the current k-means partition. As a result, later applications of k-means converge much more quickly than the first. In practice, three or four k-means iterations are sufficient to achieve convergence for k-means processes that are initialized with the preceding step's centroids.

A second heuristic for load balancing the k-means domain decomposition involves modifying the centroid initializations themselves. As discussed in Section 3.2, the k-means algorithm yields only a local optimum, and thus the final partition may be quite dependent on the initial $\mathbf{y}_k$. We take advantage of this dependence by *splitting* the centroid of the highest-cost cluster at the end of each timestep. We select the particles that were assigned to the cluster with maximum t_k^n and partition them with a local application of k-means, putting $k = 2$. The two resulting converged centroid locations are used as initial guesses for the full k-means iterations that partition the entire domain. Since the total number of centroids must remain constant (and equal to K), the centroid of the lowest-cost cluster is removed from the initialization list. Aided by these successive splittings, the centroid distribution will adapt itself to the particle distribution over time.

A final heuristic enables the load balancing scheme to recover from situations in which the random initial choice of centroids may be poor. If the load imbalance, defined as $(\max_k t_k)/\bar{t}$, exceeds a chosen safety threshold for two timesteps in a row, the centroids are "reseeded" – in other words, new centroid initializations are randomly chosen from among the current particle locations and the scaling factors s_k are all set to unity. For most of the runs reported herein, we set the threshold imbalance to 1.5 and observe that this level of imbalance is rarely encountered. (For more details, see Section 4.6 below.)

3.5. Other frameworks for treecode parallelization

The preceding sections introduced k-means clustering as a new tool for the partition of hierarchical N-body methods. Other frameworks for parallelizing treecodes have been developed in the literature, however, and it is worthwhile to contrast their approaches to domain decomposition, load balancing, and interprocessor communication with the present k-means-based implementation.

3.5.1. ORB

Beginning with a parallel BH-type code developed by Warren and Salmon for astrophysical N-body simulations [9], there have emerged a family of parallel treecodes employing ORB for domain decomposition [3,51,60]. ORB recursively partitions the computational domain into rectangular cells. At each level of the hierarchy, the domain is bisected along its longest coordinate dimension. The result of this partition is a binary tree with each leaf node corresponding to the domain of a single processor; for a tree with p levels, there are 2^{p-1} processors.

Load balance in ORB partition is controlled by positioning each bisecting plane so that equal amounts of computational work lie on either side. Computational work is estimated on a per-particle basis, usually by counting the number of interactions necessary to evaluate a particle's velocity at the previous timestep.[4] Schemes have been devised for incrementally updating these bisector positions to maintain load balance [51]; other codes recompute the ORB partition at each timestep [9,3].

Following ORB partition, velocity evaluation in all these codes proceeds by parallelization over targets, as described in the first half of Section 3.1. That is, each processor is responsible for evaluating the influence of the whole domain on its own particles. Each processor constructs a *local* BH tree in its own domain. The root nodes of these trees, corresponding to the leaf nodes of the ORB tree, are all shared among processors. Then, each processor builds a unique locally essential tree, i.e., imports the cells of non-local BH trees required to evaluate the velocity on local particles. Rather than transmitting this data as needed during tree traversals for velocity evaluation, these codes use simple multipole acceptability criteria (MAC) [7,40,9] to construct locally essential trees a priori. This process is typically organized by sender-driven communication; the owner of an ORB domain determines which of its cells may be essential to other ORB leaf nodes and sends appropriate multipole data. This is only possible for simple MACs, like the original cell-opening criterion of Barnes and Hut or variations thereof [7]. More complex and accurate error bounds like those in (16) and (13) preclude the a priori construction of locally essential trees, particularly for variable-order treecodes [10].

Consistent with our description in Section 3.1, the locally essential tree is effectively a "pruned" global tree, incorporating the influence of the entire domain on the local particles. In [9] and [3], this tree is hybrid in structure – a binary tree on top (due to ORB) partition, and an oct-tree below the ORB leaf nodes. Bhatt et al. [51] go further by explicitly constructing a global oct-tree, resolving levels between the local oct-trees; this process is rendered more difficult by adaptive features in the BH trees, like variable-size leaf nodes.

3.5.2. HOT

In contrast to these ORB-based codes, the hashed oct-tree (HOT) code of Warren and Salmon directly performs domain decomposition on the bodies of a global, distributed oct-tree [61,49,10]. Particle coordinates are mapped to 64-bit keys; the mapping is designed so that keys can identify not just particles (i.e., leaf nodes of the tree) but higher nodes of the tree. A hashing function maps keys to cell data, e.g., multipole moments and cell centers. In contrast to the pointer-based tree traversals employed above, hashing is designed to allow easier access to non-local cell data.

Domain decomposition in the HOT code proceeds by sorting the body key ordinates. Sorting these keys amounts to constructing a space-filling curve passing through all the particles with Morton ordering [62]. The curve is then partitioned into K segments, one for each processor, using estimates of per-particle cost to ensure that segments represent equal work. Branch nodes – the smallest oct-tree cell containing all the particles on a particular processor's segment of the curve – are then shared among processors to build the upper levels of the global oct-tree. Because it is the oct-tree itself that is partitioned, this stage of the algorithm avoids many of the complications of the ORB scheme [63].

Morton ordering preserves reasonable spatial locality, but is not ideal in this regard; the sorted list still contains spatial discontinuities that may be spanned by a single processor domain. These discontinuities can lead to inefficiencies in velocity evaluation [61], particularly in light of the discussion in Section 3.1. The HOT scheme also requires that leaf nodes contain single particles, unlike adaptive treecodes that allow

[4] Per-particle cost estimates are possible because velocity evaluation in these codes is parallelized over targets, as will be described below; and typically, the order of multipole expansion is fixed, so per-particle costs will remain more consistent. The "granule" of parallel partition (a target particle) can be directly associated with a cost, since velocity evaluation involves target particles interacting with source clusters. New particle introduction, however, will disrupt these estimates; this issue has not been addressed in the cited codes.

variable leaf-node size [51]. Also, cells of the HOT are uniform rectangular prisms, and cannot shrink to fit their data, an adaptive feature that was found to improve the efficiency of the present LK treecode [1].

As in the ORB codes, velocity evaluation proceeds by parallelizing over targets. To allow the use of data-dependent error criteria like that in (16) the HOT code does not construct a locally essential tree a priori. Instead, each processor requests cell data from other processors as needed while evaluating the velocity on its own particles. A complicated system of communication lists is used to hide the latency of requesting far-away data.

A simpler counterpart to the HOT code for shared-memory architectures is the *costzones* approach developed by Singh et al. [60]. Costzones also partitions the tree directly, using estimates of per-particle cost to achieve load balance. A local ordering for the children of every cell ensures that partitions of the tree are also physically contiguous.

3.5.3. Graph partitioning

A different, more theoretical approach is taken by Teng [64]; his work analyzes the *communication graphs* of hierarchical N-body algorithms, including the BH scheme, and proposes algorithms for their load-balanced partition. The communication graph, defined on particles and oct-tree cells, represents the interactions between these objects during the execution of the N-body algorithm; the edge weights reflect communication requirements of each interaction. A good partitioning algorithm, in this analysis, yields an edge-partition of the communication graph into two disjoint graphs of equal (vertex-weighted) computational cost, while keeping the "cost" – the total weight of all edges removed – small. While this and other studies of graph partition algorithms [65,66], including recursive bisection [67], provide useful theoretical results on the partition of oct-trees and other data structures, they do not address the problem of what the tree objects themselves should look like. Teng's algorithm only considers partitioning the cells of an existing, generic oct-tree. Yet cell geometry can have a profound effect on computation and communication costs, as discussed above and as will be demonstrated in the next section. Guided by these considerations, k-means clustering yields an entirely new class of geometric objects, forming an adaptive spatial partition of N-body interaction.

In summary, we note the following:

- More accurate, more complex error estimates preclude the a priori construction of locally essential trees, as do certain adaptive features, like a run-time choice between direct interaction and multipole expansion.
- Variable-order expansions and adaptive features of the tree (like cells that shrink to fit, or a run-time choice between direct interaction and multipole expansion) make per-particle cost more difficult to define, even when parallelizing over targets. Moreover, the present implementation parallelizes over sources.
- ORB and sorted hash keys do not create optimal domain geometries. ORB domains can be long and narrow (see Fig. 3 in [3], for instance); domains defined by Morton ordering may even span spatial discontinuities [61]. The HOT construction further limits tree adaptivity.
- None of the studies reviewed here show the performance of load balancing while new particles are continually being introduced, e.g., through filament remeshing; we will do so below.

4. Results

In the following, we examine the performance of cluster partition of N-body interactions by a variety of measures – speed and parallel efficiency, error control, load balance, and particle–cluster interaction counts.

In particular, we apply the k-means clustering and load balancing algorithms developed in Section 3 to the parallel hierarchical evaluation of vortical velocities in a vortex simulation of a transverse jet at high Reynolds number.

4.1. Transverse jet simulations

The dynamics and mixing properties of the transverse jet – a jet issuing normally into a uniform crossflow – are important to a variety of engineering applications. Transverse jets may function as sources of fuel in industrial furnaces, or as diluent jets for blade cooling or exhaust gas cooling in industrial or airborne gas turbines. The transverse jet is a canonical example of a flow dominated by large-scale "coherent structures." Experimental observations by Fric and Roshko [68] identify four such structures in the transverse jet: jet shear layer vortices; "wake vortices" arising from interaction between the jet and the channel wall boundary layer; horseshoe vortices that wrap around the jet exit; and a counter-rotating vortex pair that forms as the jet bends into the crossflow, persisting far downstream. The evolution of these structures is inherently three-dimensional and characterized by topological changes in the vorticity field. Vortex methods are attractive in this context for their explicit link to the formation and dynamics of vortical structures in the flow.

Details of our vorticity formulation and a thorough analysis of the flow physics revealed by vortex simulation are presented elsewhere [11,69,12,70]. Here, we merely summarize aspects of the simulation that are relevant to the parallel N-body problem. Vorticity entering the flow at each timestep is discretized with vortex particles that lie on partial filaments [11]. Filaments result from our physically motivated expression of vorticity flux boundary conditions, but also provide a convenient mechanism for local remeshing in response to flow strain. Filament geometries are described by cubic splines supported by a finite set of nodes. Nodes are advected by the local velocity field using a second-order predictor/corrector method with timestep control. Advecting the nodes accounts for deformation of the material lines and thus for stretching and tilting of the vorticity and the corresponding modification of element weights $\boldsymbol{\alpha}_i = \boldsymbol{\omega}\, \mathrm{d}V_i$, since vortex lines and material lines coincide. When the length $|\boldsymbol{\delta}\chi_i|$ of a given element exceeds 0.9δ, where δ is the regularization radius in (8), a new node is added at the midpoint of the element, thus splitting the element into two connected elements and enforcing the core overlap condition along the filament [58].

In addition to local insertion of vortex elements/nodes, we implement hairpin removal algorithms to remove small-scale folds along vortex filaments [71]; this process regularizes the formation of small scales and thus reduces the rate at which elements proliferate. We also merge neighboring elements along filaments whenever the linear extent of an element becomes too small. The result of all these operations on filament geometry is an incremental remeshing which modifies the vortex particle distribution – a distribution that is also being modified by advection and by the evolution of particle weights $\boldsymbol{\alpha}_i(t)$. On balance, local element insertion, hairpin removal, and small element merging result in a net positive introduction of elements. Thus, not only do elements enter the flow at the jet nozzle, but they are created throughout the domain. This is typical of three-dimensional vortex methods [14,72], and corresponds to the turbulent cascade towards smaller length scales via stretching and folding of vortex lines.

4.2. Single-step performance and scaling

We first examine timings for a single evaluation of vortical velocities. A "single evaluation" involves calculation of the full N-body problem, computing the velocity induced by every vortex element on every other vortex element. Of course, a higher-order time integration scheme (e.g., a Runge–Kutta scheme) may require multiple such evaluations in a single timestep. The timing data reported below are obtained on the IBM SP RS/6000 at the National Energy Research Scientific Computing Center (NERSC), which is composed of 375 MHz POWER 3 processors arranged in 16-processor nodes.

4.2.1. Test cases

Fig. 5 shows a representative particle distribution from transverse jet simulation, containing $N = 157\,297$ particles. Each vortex particle is represented by a sphere with radius proportional to the norm of the particle's vector weight $\|\boldsymbol{\alpha}_i\|_2$. The crossflow is directed in the positive x direction; the jet centerline is aligned with the y axis; and the z axis is in the spanwise direction. Flow variables are made dimensionless by d, the nozzle diameter, and U_∞, the crossflow velocity. The jet orifice is thus a disc of diameter one centered at the origin of the x–z plane; the remainder of the x–z plane is a solid wall through which we enforce a no-flow boundary condition. The ratio of the jet velocity to the crossflow velocity, denoted by r, is 7.

A few comments on the particle distribution are in order. Vortex particles are introduced at the edge of the jet nozzle every Δt_{noz} time units; the number of particles introduced at each such instant is n_θ. Here, we take $\Delta t_{\text{noz}} = 0.01$, $n_\theta = 64$, and the particle core radius $\delta = 0.1$. As particles enter the flow, they initially compose a cylindrical shear layer which rolls up 1–2 diameters above the jet. Roll-up of the shear layer is manifested by grouping of the particles into vortex rings; but as these rings form, they stretch and deform out-of-plane in a process that is intimately linked to the formation of a counter-rotating vortex pair aligned with the jet trajectory. Counter-rotating vorticity further disrupts the particle distribution as vortex filaments wind and stretch and as opposite-signed vorticities approach each other. The jet then bends further into the crossflow and the particle distribution – in terms of both locations and weights – becomes enormously complicated as smaller scales are generated and the particles fill the space. Again, a complete analysis of these vortical transformations is given in [12].

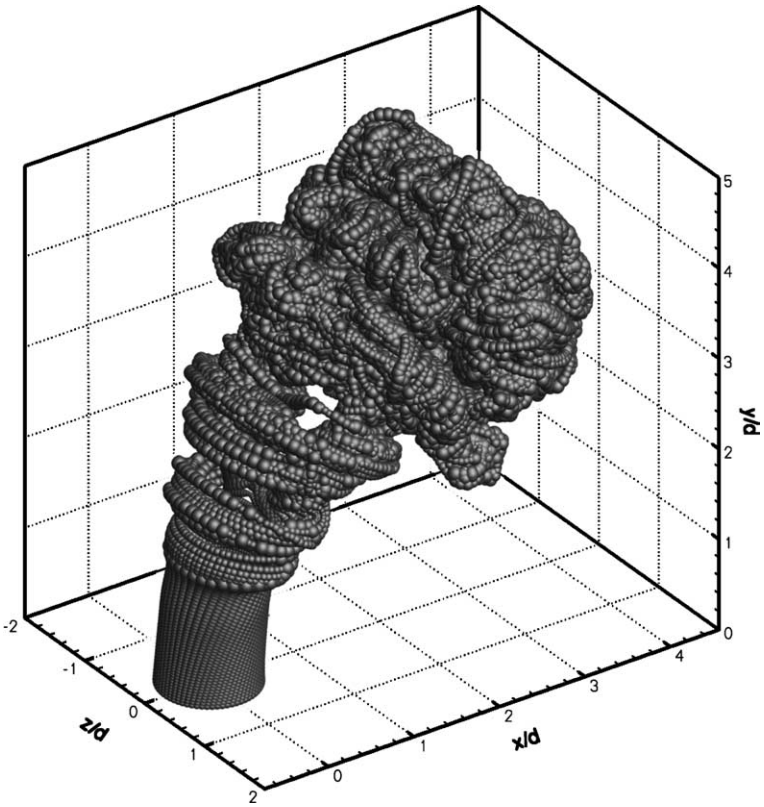


Fig. 5. Vortex elements in the transverse jet at $t = 2.0$; $N = 157\,297$. Particle sizes are proportional to $\|(\boldsymbol{\omega}\,\mathrm{d}V)_i\|_2$.

We consider two different vortex element distributions, both drawn from simulations of an evolving transverse jet – one with smaller N ($N = 261\,481$) and one with larger N ($N = 1\,164\,184$). The larger case is reached approximately 0.4 convective time units after the smaller one, and thus represents not only more particles but a particle distribution that has evolved slightly further downstream and developed more small scales.

For each of these two cases, we compare three partitioning schemes. The first, labeled "block distribution," is simply a block partition of vortex element arrays. Within the arrays, elements are arranged in order of (1) *where* they appear along a filament and (2) *when* the filament was introduced into the flow. Since elements on successive filaments will share somewhat similar trajectories, this distribution preserves some data locality, as shown in Fig. 6(a). Nonetheless, some interleaving is present. Performance of this "naive" partition is not expected to be good, but it is a convenient and straightforward partition to compute for purposes of comparison. Each domain contains essentially the same number of particles, N/k.

The second partitioning scheme is k-means clustering as presented in Section 3.2, with no attempt at correcting the load balance. Thus, all the scaling factors s_k are set to 1.0, and the iterations for cluster centroids are allowed to begin from a random initial seed of particle locations. The third partitioning scheme employs k-means clustering but adds the load balance heuristics developed in Section 3.4. The precise procedure for obtaining this partition is as follows: First, k-means clustering is performed with unity scalings and random initial seed. Then, the velocity is evaluated and cluster timings t_k^n are obtained. The highest-cost cluster is split and the scalings s_k are updated with one application of (21), then the partition is re-computed. In other words, the difference between the second and third partitioning schemes is one iteration of the load balance heuristics.

Two exceptions to this procedure are the $k = 1024$ "scaled clusters" cases in Figs. 7–10, identified by the filled-in square symbol in each figure. Timings for these cases were obtained from an actual, dynamic vortex element simulation, and hence are the result of many load-balancing iterations applied to an evolving particle distribution. These cases serve to illustrate the realistic performance of load-balanced clustering in a dynamic simulation.

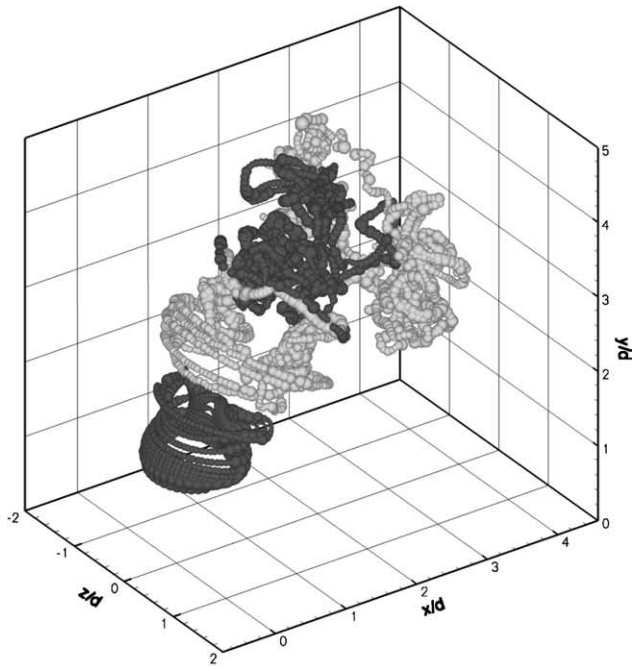
4.2.2. Timing, speedup, and parallel efficiency

For each particle distribution, with each of the three partitioning schemes, we scale the number of processors from 1 to 1024, choosing $k \in \{1, 16, 64, 128, 512, 1024\}$. Fig. 7(a) and (b) shows the total time of velocity evaluation in each of these cases. We set the treecode accuracy parameter $\epsilon = 10^{-2}$ and the leaf size parameter $N_0 = 512$. Velocity evaluation times reflect both the time necessary to evaluate the velocity at each vortex particle in parallel and the overhead of interprocessor communication (i.e., for global reduction operations after every processor has finished evaluating the velocity induced by sources in its domain). This can be broken down, using the notation of the previous section, as $T = \max_k t_k + t_{\text{comm}}$. From these figures, it is clear that the block distribution results in slower velocity evaluations than either of the cluster distributions. Load-balanced clustering results in faster evaluations than plain k-means clustering, particularly for $k > 128$. Regardless of the partition scheme, adding more processors leads to faster evaluations (not a surprising result).

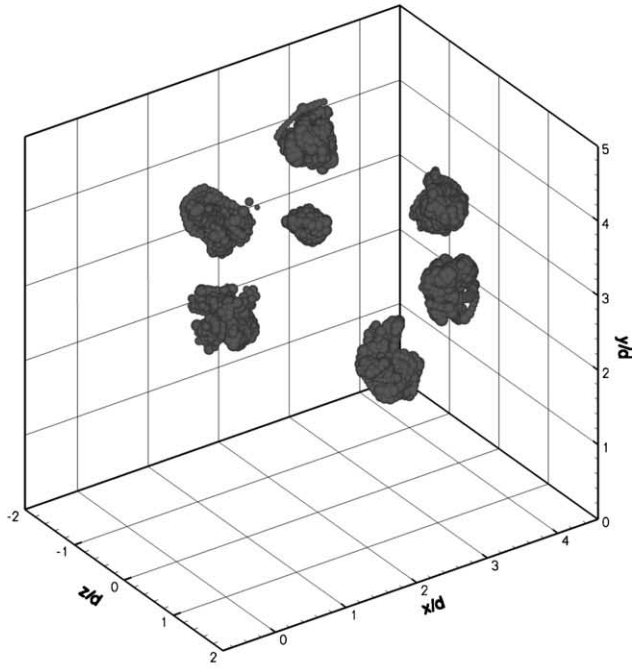
It is instructive to cast the timing data in terms of speedup S, where

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}. \tag{22}$$

These data are shown in Fig. 8. Ideal speedup is equal to k. Clearly, the block distribution performs poorly compared to the clustered distributions, yielding a speedup of less than 200 when using 1024 processors. Scaled k-means outperforms plain k-means, especially for $k > 128$. An additional gain in speedup is seen in the scaled $k = 1024$ cases, denoted with solid squares in Fig. 8(a) and (b); recall that these partitions result from successive load-balance iterations, whereas scaled cases with $k \neq 1024$ are only one load

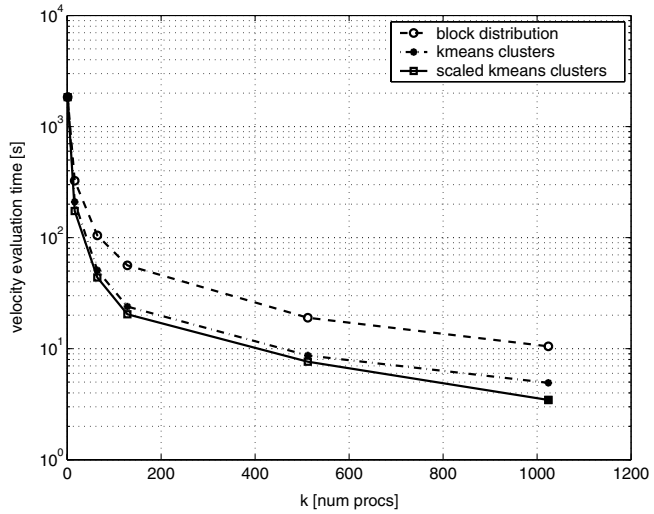


(a) Block partition, 4 domains.

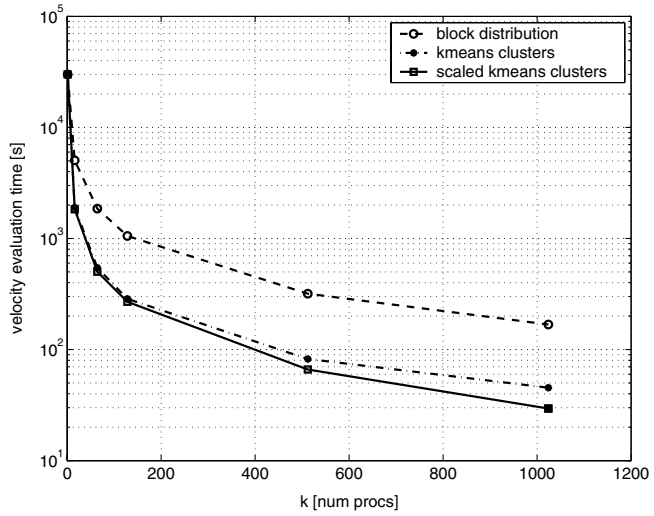


(b) Cluster partition, 7 domains.

Fig. 6. Representative domains resulting from block and cluster partitioning of the vortex elements in Fig. 5; $t = 2.0$, $N = 157\,297$, $k = 128$.



(a) $N = 261481$.



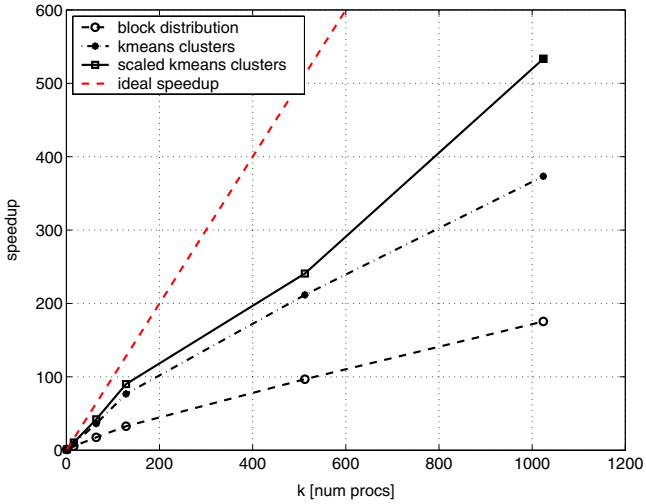
(b) $N = 1164184$.

Fig. 7. Velocity evaluation time for the parallel treecode versus number of processors, testing three different domain decomposition schemes.

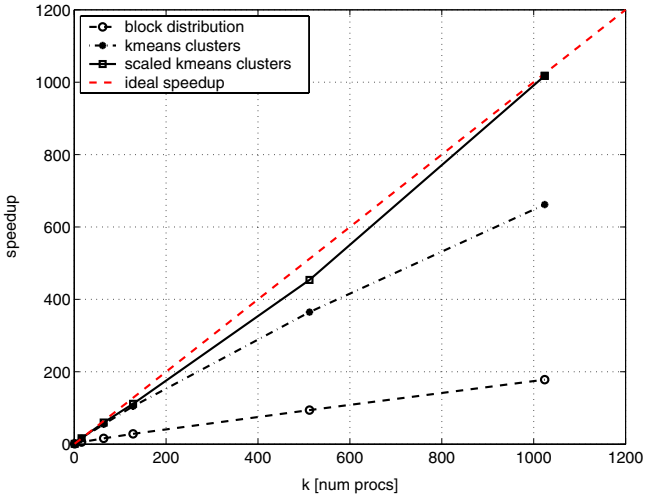
balance iteration away from their unscaled counterparts. It is remarkable that the speedup of scaled k-means clusters in the larger N case is quite close to the ideal speedup. This comparison is better distilled by plotting parallel efficiency P, defined as follows

$$P = \frac{T_{\text{serial}}}{k * T_{\text{parallel}}} \tag{23}$$

and shown in Fig. 9. Parallel efficiency of block distribution falls off rapidly at relatively small k and approaches a value below 20% in both the large N and small N cases. With the cluster partitions, the parallel efficiency observed in the large N case is significantly better than in the small N case. This may be due to the



(a) $N = 261481$.



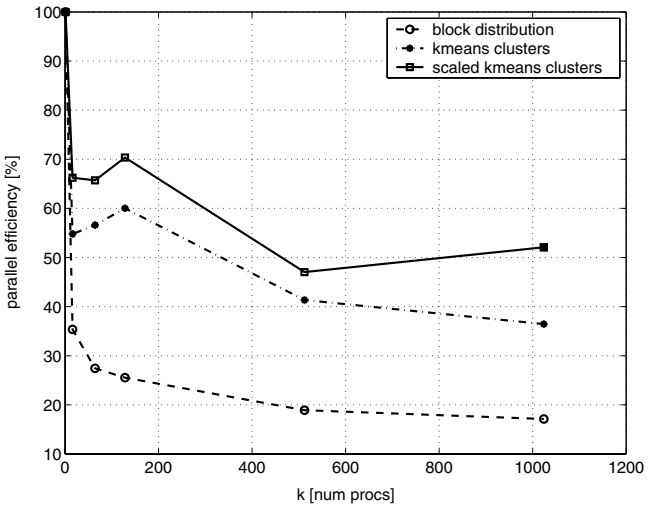
(b) $N = 1164184$.

Fig. 8. Speedup for the parallel treecode versus number of processors, testing three different domain decomposition schemes.

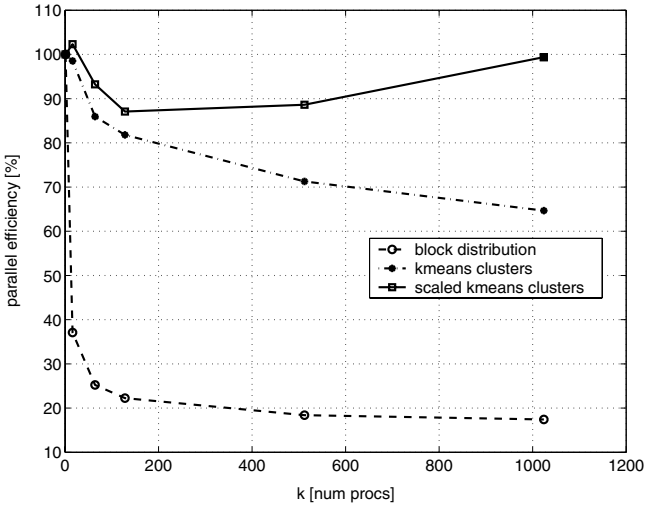
proportionally smaller communication cost of the former, or may point to some subtle interaction between the granularity N/k and the maximum leaf node size N_0. In any case, problem sizes of $N > 10^6$ are much more representative and computationally demanding targets for parallel hierarchical methods, particularly for large $k \approx 10^3$. In this case, we observe parallel efficiencies consistently above 85% for scaled k-means clusters. The most realistically load-balanced case, $k = 1024$, shows a remarkable parallel efficiency of 98%.

4.2.3. Load imbalance

The load imbalances underlying the single-step timings just presented are shown in Fig. 10. We define load imbalance I as follows:



(a) $N = 261481$.

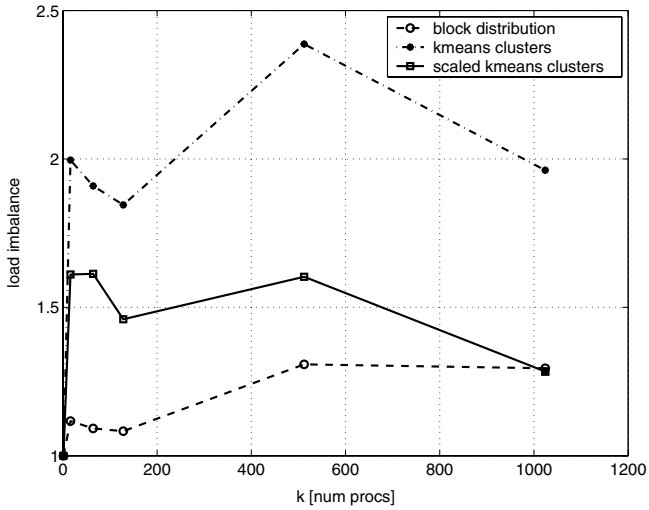


(b) $N = 1164184$.

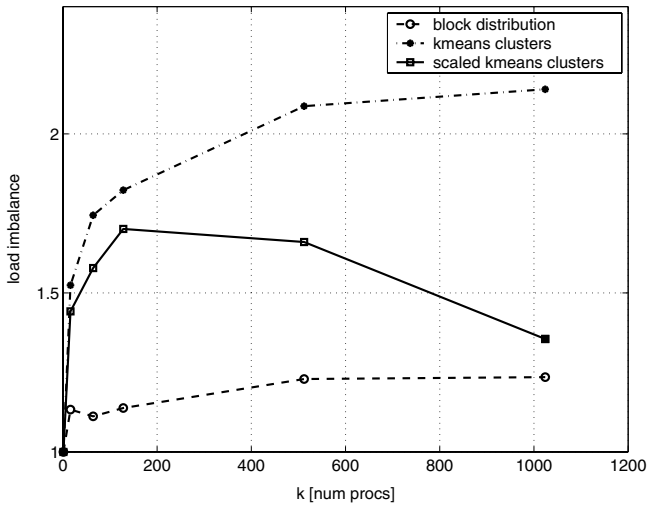
Fig. 9. Parallel efficiency versus number of processors, testing three different domain decomposition schemes.

$$I = \frac{\max_k t_k}{\bar{t}}. \tag{24}$$

Several features are worth noting. First, block distribution shows the best overall load balance, with an imbalance below 1.3 in all cases of k and N. Although every domain in a given block distribution has essentially the same number of particles, per-particle cost is not uniform, as discussed in Section 3.4, and thus some imbalance will be present. Nonetheless, this imbalance is relatively small, and illustrates the fact that good load balance is no guarantee of parallel efficiency. *Domain geometry* has a key role in determining parallel efficiency; and thus all the cluster partitions, though they may exhibit larger load imbalances, show vastly better parallel performance than the block partition.



(a) $N = 261481$.



(b) $N = 1164184$.

Fig. 10. Load imbalance for each test case in Figs. 7–9, testing three different domain decomposition schemes.

Load imbalance with plain k-means clustering, shown by the dash-dotted line in Fig. 10(a) and (b), tends to increase with the number of processors, although jaggedness in this curve reflects the fact that, without any attempts at controlling the relative cluster populations, load balance in the clustered case depends on the random initial seed. After all, it is the highest-cost cluster that determines the load imbalance. One application of the load-balancing heuristics reduces the imbalances to those observed on the solid line, for $k \neq 1024$. Successive load-balancing iterations, even though they are performed on a dynamically evolving particle distribution, reduce the imbalance even further – to approximately 1.3 in both $k = 1024$ cases. Again, we emphasize that this value is typical of the imbalance observed in full simulations. We will comment further on the performance of successive load-balance iterations in Section 4.6.

Examination of Figs. 9(b) and 10(b) together motivates an additional observation. Consider the scaled k-means partitions for large N: while these cases have load imbalances of 1.3–1.7, they have parallel efficiencies above 85%. In particular, consider the $k = 1024$ case, with its load imbalance of 1.355 and parallel efficiency of 98%. If the load balance were further improved, the parallel efficiency would clearly be higher than 100%. Suppose, for instance, that the load imbalance in this case could somehow be reduced to 1.0, and suppose further that perfecting the load balance would not shift the mean cluster time $\bar{t}$ or change the communication overhead T_{comm}. Then, using a simple breakdown of computational costs,

$$P = \frac{T_{\text{serial}}}{k * T_{\text{parallel}}} \approx \frac{T_{\text{serial}}}{k(T_{\text{comm}} + I * \bar{t})}, \tag{25}$$

we would find a parallel efficiency of 130%. Without communication overhead, we would observe a speedup of 1536 – a "parallel efficiency" of 150%. While this situation seems entirely hypothetical, it illustrates that our actual parallel partition performs better than the load balance would lead one to expect. Why is this the case? There may be some gains in speed due to better use of cache in the parallel computation. But a factor that cannot be overlooked is the difference in the geometry of the hierarchical partition between the serial and parallel cases. The serial case has no k-means clusters; instead it has a single adaptive oct-tree covering the entire domain. Clusters partition the domain differently, and it may be that the resulting hybrid partition – with k-means clusters serving as root cells of small oct-trees – is more efficient than even the original oct-tree.

4.3. Particle–cluster interaction counts

While the timings reported in the preceding section are the ultimate practical measure of performance, additional insight into the effect of geometry on computational cost may be gained by counting particle–cluster interactions.

The possibility raised at the close of the previous section – that k-means clustering may provide a better partition of the domain – has implications beyond parallel decomposition. It is difficult to explore this possibility with measures of computational time alone, however, as factors like communication overhead, along with memory and cache access patterns, will color the timing data in a hardware-dependent fashion.

Fig. 11 shows the number of source particles evaluated at each order of expansion p or with direct summation, for a single computation of vortical velocities. We use the $N = 1\,164\,124$ particle distribution presented earlier and consider three different partitions: (1) block distribution with 1024 domains, (2) k-means clustering with $k = 1024$ clusters, and (3) a single adaptive oct-tree. The last partition is simply the serial ($k = 1$) case in Section 4.2, employing all the adaptive oct-tree features discussed in [1].

Particle–cluster interactions are counted as follows. Each time the velocity induced by a source cell of n_c particles is evaluated at some order p, n_c is added to a counter corresponding to that p. Of course, every target particle interacts with N source particles through expansions at different levels of the tree, and thus the sum of the ordinates on each curve in Fig. 11 is equal to N^2. The curves differ, however, in their distribution. The block partition yields a very large number of direct interactions, accounting for its poor parallel efficiency. The global oct-tree shows that the largest number of particles participate in order $p = 7$ interactions, and that very few cells are expanded at $p \leqslant 5$. The k-means partition, by contrast, shows lower orders of expansion. Some particles are expanded at order $p = 4$, and the distribution peaks at $p = 6$. Lower-order expansions are less expensive – both on a per-cell basis, because the number of terms at each order is $O(p^3)$ – and on a per-particle basis, because the initial calculation of higher-order moments may be avoided. K-means thus yields a partition on which hierarchical velocity evaluations may be performed more efficiently, at lower computational expense.

Fig. 11 very clearly shows the impact of geometry on computational cost – in particular, how geometry directly controls the number and order of particle–cluster interactions necessary to evaluate velocity within

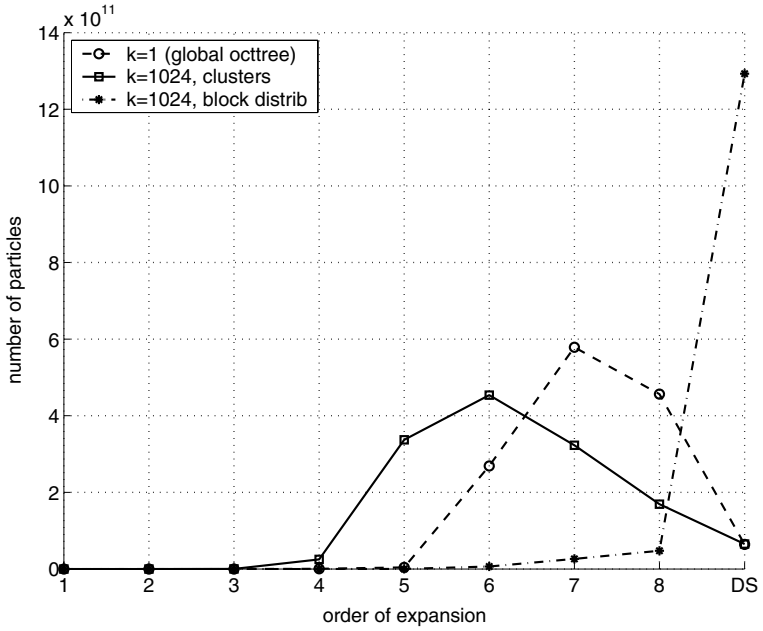


Fig. 11. Number of particles evaluated at each order of expansion p or with direct summation, using different global partitions; $N = 1\,164\,184$ case.

a given accuracy. This confirms the mechanisms discussed in Section 3.1 and carries implications for the geometric constructions on which hierarchical methods are built, transcending issues of serial or parallel implementation. Future work that extends these ideas will be discussed in Section 5.1.

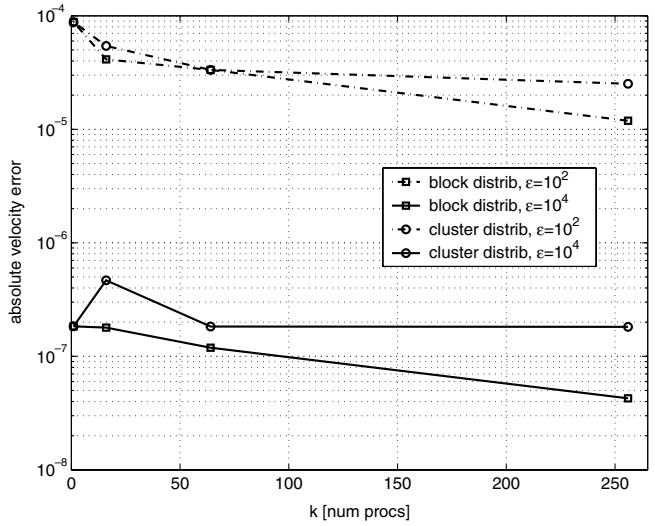
4.4. Error control

While noting substantial gains in parallel performance, it is important to verify that the accuracy of velocity evaluation is well-controlled with the present algorithm. We compute the exact velocity at each target particle $\mathbf{u}_i^{\text{dir}}$ using direct summation and compare these values to those obtained with the parallel treecode, using both block and cluster partitions. This comparison is performed for two cases of N: $N = 261\,481$ (the small-N case used in the previous two sections) and $N = 102\,505$. Because of the high cost of direct summation, performing this comparison for much larger N would have been computationally prohibitive. The treecode velocities were obtained for two different values of the accuracy parameter, $\epsilon = 10^{-2}$, 10^{-4}. Cluster partition is performed without splitting of high-cost clusters or scaling, as load balance is not expected to have much effect on velocity error.

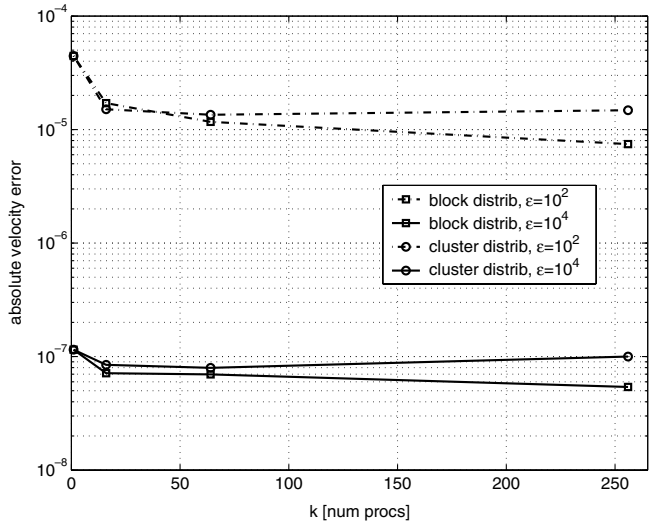
Fig. 12 shows the absolute error in velocity as a function of k, for $k \in \{1, 16, 64, 256\}$. This error is defined as the maximum, over all the target particles, of the velocity error magnitude:

$$e_{\text{abs}} = \max_i \left\| \mathbf{u}_i - \mathbf{u}_i^{\text{dir}} \right\|_2. \tag{26}$$

We find that this error is very well-controlled as the velocity evaluation is further parallelized – it remains at or below the serial ($k = 1$) error for all k, with only one exception ($k = 16$ and $N = 102\,505$). Block distribution seems to produce smaller errors than cluster distribution as k increases. With either partition, the reduction in error with higher k is more pronounced for the $\epsilon = 10^{-2}$ case than the $\epsilon = 10^{-4}$ case. Similar trends are observed with the relative magnitude of the velocity error, defined as



(a) $N = 102505$.

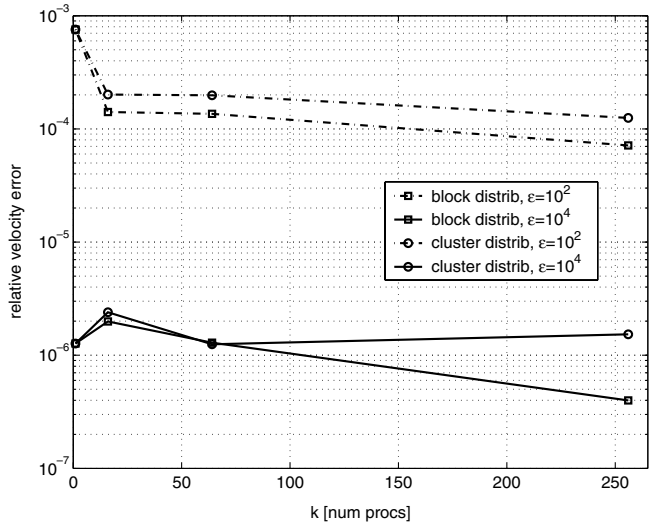


(b) $N = 261481$.

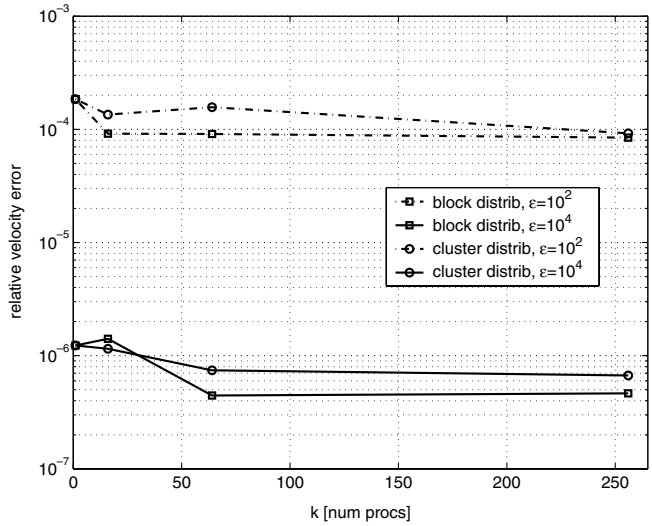
Fig. 12. Velocity magnitude error versus number of processors, for $\epsilon = 10^{-2}$, 10^{-4}, testing block and cluster partitions.

$$e_{\text{rel}} = \max_i \left(\frac{\left\| \mathbf{u}_i - \mathbf{u}_i^{\text{dir}} \right\|_2}{\left\| \mathbf{u}_i^{\text{dir}} \right\|_2} \right) \tag{27}$$

and shown in Fig. 13. The fact that errors are smaller in the parallel cases than in the serial case suggests that we are over-constraining the error. By distributing fractions of the global accuracy parameter ϵ to all clusters, as in Eq. (18), we are in a sense replacing one constraint with k independent constraints. As a result, the bound on the total error is too conservative. A more sophisticated distribution of the error parameter to clusters – one that keeps e_{abs} or e_{rel} from declining with higher k – could conceivably further reduce the time of parallel velocity evaluation, for additional gains in computational efficiency.



(a) $N = 102505$.



(b) $N = 261481$.

Fig. 13. Relative velocity magnitude error versus number of processors, for $\epsilon = 10^{-2}$, 10^{-4}, testing block and cluster partitions.

4.5. K-means performance and scaling

The calculation of a k-means partition carries its own computational cost, and it is desirable, for an effective parallel domain decomposition, that this cost (1) remain small relative to the actual cost of evaluating the N-body interaction and (2) scale well with problem size. Fig. 14 shows the time per iteration of the k-means algorithm as a function of the number particles N for different values of k. In this context k is both the number of clusters and the number of processors performing the clustering. As a result, the overall computational complexity of a single k-means step, O(Nkd), is divided by k, and we should see O(N) scaling. This is borne out in the figure, as the lines corresponding to different k lie on top of each other and are

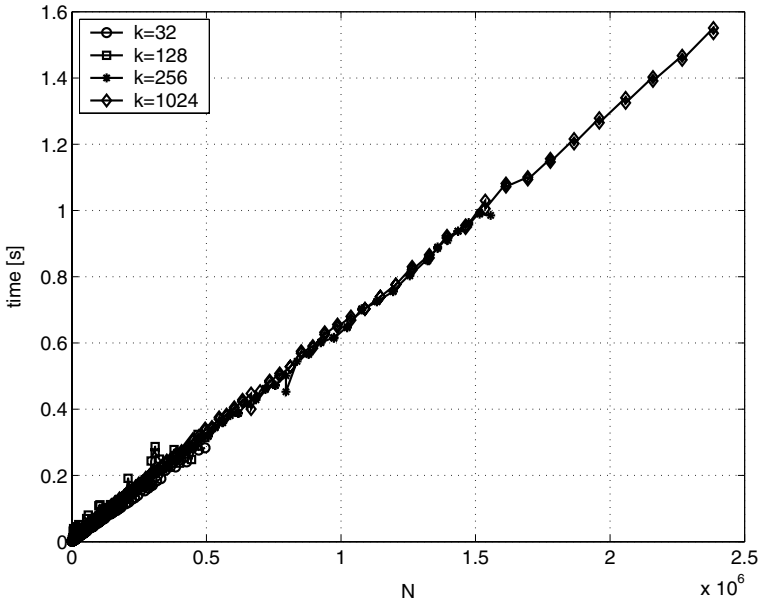


Fig. 14. Time per iteration of parallel k-means clustering.

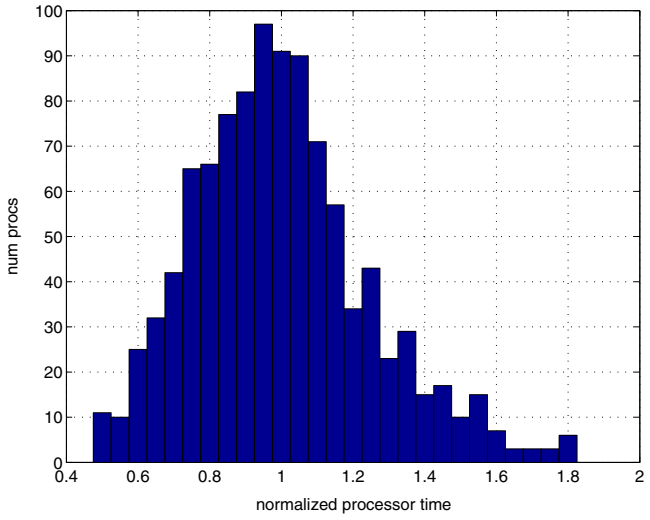
relatively indistinguishable. Thus, the parallel efficiency of our parallel k-means implementation is very near 100%. The absolute time per iteration is quite small compared to the velocity evaluation times in Fig. 7 for similar N, and, for a given k, the O(N) scaling of k-means is asymptotically smaller than the O($N \log N$) scaling expected of a BH-type treecode. Converging to a k-means partition may require several iterations; in fact we set $l_{\max} = 7$ when starting from a random seed of centroids and $l_{\max} = 4$ otherwise. But the total cost of these iterations is still small compared to the cost of velocity evaluation, especially when one considers that higher-order time integration schemes like Runge–Kutta will perform several velocity evaluations with a single k-means partition.

4.6. Dynamic load balance

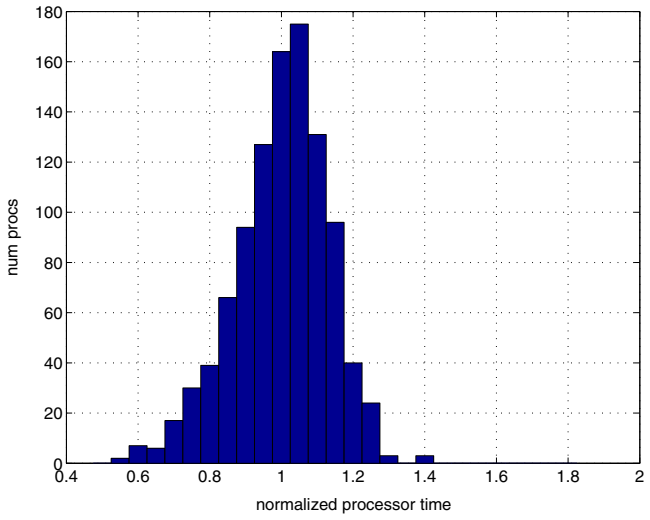
The performance of the load-balancing algorithms developed in Section 3.4 is best demonstrated in a dynamic N-body simulation; here, we use the transverse jet simulation, complete with evolving vortex particle locations and weights and new particle introduction throughout the domain.

First, we examine the distribution of processor times t_k for a single velocity evaluation step. We choose the $N = 1\,164\,184$ case used earlier, extracting this case from two simulations. One simulation employs the load-balancing algorithms – i.e., iterated scaling and split/merge of cluster centroids – and the other does not, instead reseeding the centroids at each step. The processor times t_k do not include any communication overhead; they consist of the time required by each processor to evaluate the influence of its source particles on the whole domain. These times are normalized by $\bar{t}$ and used to populate the histogram in Fig. 15. Clearly, the distribution of processor times is much narrower in the load-balanced case. The maximum processor time determines the load imbalance, which is approximately 1.4.

We can extend this analysis to successive velocity evaluations and thus find the averaged normalized load distribution on an evolving field of particles. Fig. 16 shows normalized processor times for 36 successive velocity evaluations, with N growing from 10^6 to 2.5×10^6. The load-balanced simulation again shows a



(a) Clustering, no scaling or split/merge.

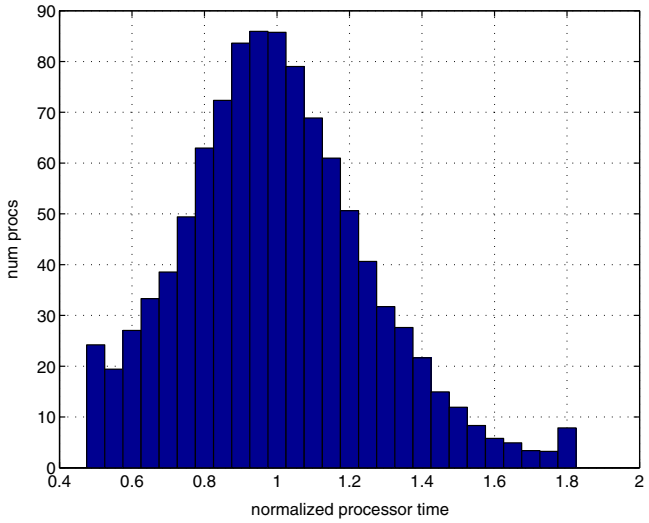


(b) Clustering, with iterated scaling and split/merge.

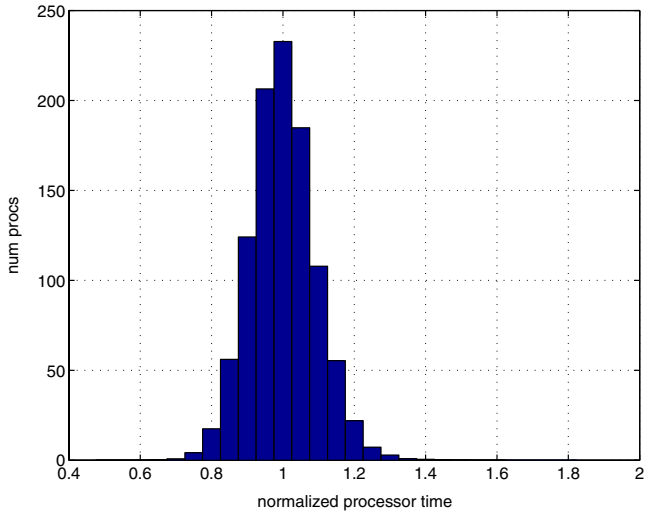
Fig. 15. Distribution of normalized processor times for a test case with $N = 1\,164\,184$, $k = 1024$, drawn from a simulation of an evolving transverse jet.

significantly narrower load distribution than the simulation performed with plain k-means clustering. The load distribution in the latter case has a long tail above the mean processor time; iterated scaling and split/merge in Fig. 16(b) seem to control the extent of this tail quite effectively.

Fig. 17 shows the load imbalance at each step of the simulation. Since we are using a second-order Runge–Kutta method for time integration, there are two values of the imbalance at every value of N, corresponding to two different velocity evaluations. The load imbalance with plain k-means ranges up to 2.5 and shows significant variation from step to step. In contrast, the load-balanced clustering maintains an



(a) Clustering, no scaling or split/merge.



(b) Clustering, with iterated scaling and split/merge.

Fig. 16. Normalized processor times in a simulation of an evolving transverse jet, averaged over 36 successive velocity evaluations, for N growing from 10^6 to 2.5×10^6.

imbalance well below 1.5 for much of the simulation, and has a better bounded step-to-step variation as well. The contrast in values of the load imbalance is particularly appreciable at large values of N.

Of course, the ultimate measure of performance is velocity evaluation time, to which load balance is a contributing factor. Fig. 18 plots the total velocity evaluation time – including communication overhead – at every step of a simulation of the evolving transverse jet. The dashed line shows velocity evaluation times using plain k-means clustering; as suggested by the plots of load balance, this line is quite jagged, and its deviation from the load-balanced case becomes significant at large N. Introducing the load-balance heuristics results in a smoother profile of evaluation time versus N; at larger values of N, the computational

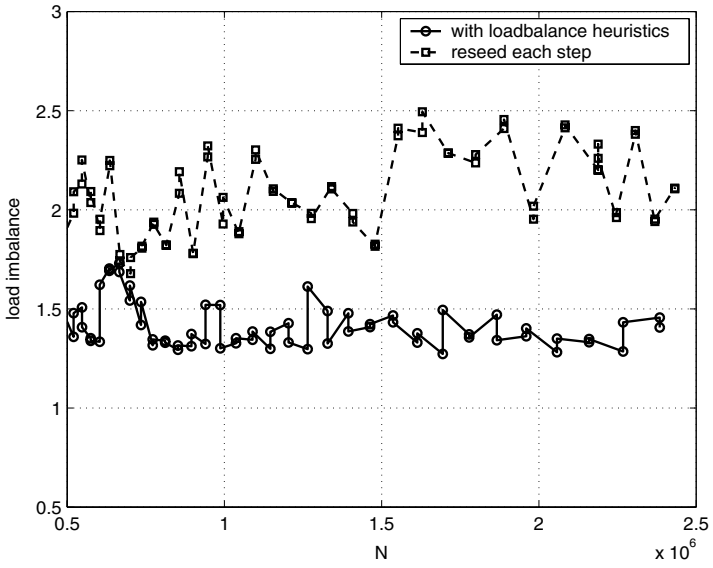


Fig. 17. Load imbalance versus N for an evolving transverse jet, showing the effect of load-balancing heuristics.

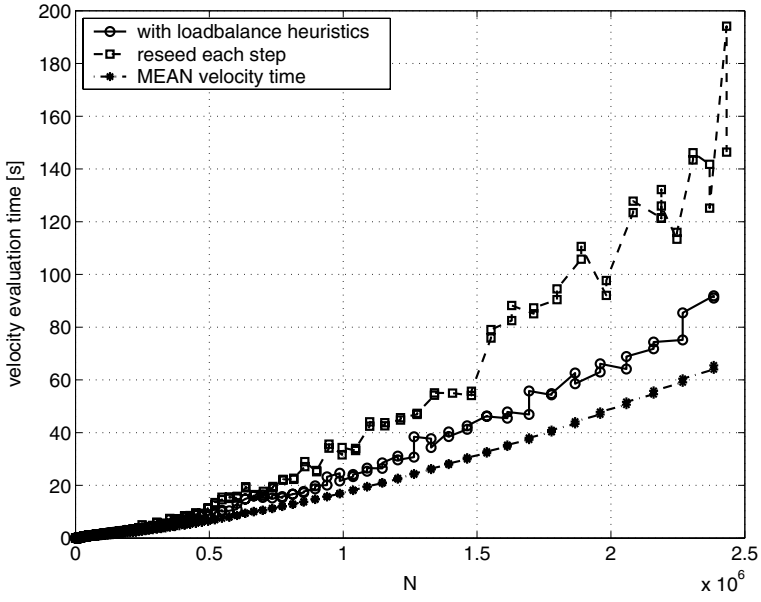


Fig. 18. Velocity evaluation time versus N for an evolving transverse jet. Mean velocity time (shown with asterisks) represents the case of best possible load balance.

savings, relative to the unbalanced case, can be as much as 50 s per evaluation. The hypothetical ideal performance is represented by the dash-dotted line, which shows the mean velocity evaluation time $\bar{t}$ plus communications overhead. This is the velocity evaluation time that could be achieved with perfect load

balance. While there is some gap between this line and the actual load-balanced simulation at large N, the present simulations perform remarkably well relative to this ideal.

Fig. 18 also shows a relatively favorable increase in velocity evaluation time versus N. The trend is best observed with either the load-balanced evaluation time or the mean evaluation time. While we cannot make strict conclusions about scaling, since the particle distribution is evolving as N increases, the mean velocity evaluation time shows a growth that appears somewhere between $O(N)$ and the $O(N \log N)$ that would be expected for a Barnes–Hut-type treecode.

5. Conclusions

The high-resolution vortex particle simulations described in this paper present a number of computational challenges. Chief among these – in terms of computational cost – is the N-body problem of evaluating vortical velocities at every particle. Direct summation is prohibitively expensive for large N and thus we employ a hierarchical method, specifically an adaptive treecode based on Taylor expansions of the Rosenhead–Moore kernel (8) in Cartesian coordinates [1]. This method must contend with an irregular, time-evolving particle distribution of non-uniform density and with the continual introduction of new vortex particles throughout the domain. Moreover, the size of our problem requires that we implement the hierarchical method in parallel on a distributed-memory machine.

This paper introduces new algorithms, based on weighted k-means clustering, for partitioning and dynamic load balancing of N-body interactions. Good spatial partitioning is central to the performance of a hierarchical method. We demonstrate that the number of particle–cluster interactions and the order at which they are performed is directly affected by partition geometry, and that the relationship between partition geometry and computational cost is expressed in the error bounds of various cluster approximations. Weighted k-means creates well-localized convex domains and minimizes a sum of cluster moments, reducing the cost of computing the N-body problem.

We also introduce heuristics for dynamic load balancing that are compatible with k-means; these include iterative scaling of cluster sizes and adaptive redistribution of cluster centroids.

Application of load-balanced k-means partition to the parallel hierarchical evaluation of vortical velocities results in outstanding parallel efficiencies; velocity evaluation errors, on the other hand, are maintained at or below their serial values. On a realistic distribution of 1.2 million vortex particles, obtained from the transverse jet, we observe a parallel efficiency of approximately 98% on 1024 processors; naive approaches to domain decomposition show parallel efficiencies below 20% on the same problem. In simulations of the evolving transverse jet, we find that load imbalance is typically maintained below 1.5. Additionally, we find that (1) load balance provides no guarantee of parallel efficiency, and (2) with k-means partition, the parallel efficiency of the hierarchical method is better than the load imbalance would suggest.

The utility of these clustering algorithms extends beyond vortex particle methods to N-body problems in a variety of fields.

5.1. Future work

The performance of parallel cluster partitioning and the accompanying dynamic load balancing techniques, while very good, suggests a number of extensions and improvements.

First, one may explore alternative means of obtaining load-balanced cluster partitions. Load balancing in this problem may be cast as a constrained optimization problem. Optimal geometry – i.e., minimization of the cost function J in (15) with all the scalings s_k set to unity – must be subject to a constraint ensuring equal computational cost for each cluster. One way to approach this constraint is to add a penalty term of the form $\gamma(N_k - \bar{N})^2$ or $\gamma(t_k - \bar{t})^2$ to the k-means cost function. How best to solve this optimization problem

– in other words, how this new term would modify (or invalidate) the k-means algorithm – will require some consideration.

Second, a logical step forward for our parallel treecode is to extend the implementation to distributed data – that is, to no longer store copies of all the particle locations and weights on all processors. Doing this will add additional communication steps to the current parallel framework, but eliminate any realistic constraints that memory may place on problem size. We may be able to take advantage of the k-means partition in designing the necessary algorithms. Mapping a point in space to the domain that contains it in $\log k$ time could easily be accomplished with hierarchical clustering (see below); we also may be able to use the Voronoi tesselation of cluster centroids to do the same.

Finally, while the primary objective of this paper has been to develop and demonstrate the advantages of flat k-means clustering for parallel domain decomposition and dynamic load balancing, the performance of these algorithms suggests a broader applicability to N-body problems. When k-means clusters are used for domain decomposition, the parallel efficiency of the N-body calculation is better than the load imbalance alone would suggest (i.e., $P > 1/I$) because k-means creates a different overall partition geometry. As shown in Fig. 11, more source particles have their influence computed at lower order in the parallel case (a hybrid k-means + local oct-tree partition) than in the serial case (an adaptive global oct-tree partition). This result is not surprising, given the geometric optimality of k-means clusters. Results thus suggest that *hierarchical* k-means clustering could replace traditional oct-tree partitioning schemes to optimize the computational efficiency of serial N-body algorithms, extending geometric optimality to every level of the hierarchy. The opportunities for adaptivity in this context are enormous. The number of child clusters within each parent cell is not constrained in any way, and could be locally optimized at each node of the tree. In the parallel context, hierarchical clustering may also offer a simpler means of load-balancing by *localizing* competition among centroids for particles. The improved interaction counts in Fig. 11 may just scratch the surface of potential gains.

Acknowledgments

This work is supported by the US Department of Energy, Office of Science, MICS. Computational support is provided by the National Energy Research Supercomputing Center (NERSC). We thank K. Lindsay and R. Krasny for making the source of their serial treecode available. Y.M. Marzouk gratefully acknowledges the support of a graduate fellowship from the Fannie and John Hertz Foundation. Y.M. Marzouk also acknowledges helpful discussions with H. Shu.

References

[1] K. Lindsay, R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, Journal of Computational Physics 172 (2) (2001) 879–907.
[2] G.-H. Cottet, P.D. Koumoutsakos, Vortex Methods: Theory and Practice, Cambridge University Press, Cambridge, MA, 2000.
[3] J. Dubinski, A parallel tree code, New Astronomy 1 (2) (1997) 133–147.
[4] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, Journal of Computational Physics 196 (2) (2004) 591–626.
[5] L. Greengard, Fast algorithms for classical physics, Science 265 (5174) (1994) 909–914.
[6] A.W. Appel, An efficient program for many-body simulation, SIAM Journal on Scientific and Statistical Computing 6 (1985) 85–103.
[7] J. Barnes, P. Hut, A hierarchical O($N \log N$) force-calculation algorithm, Nature 324 (6096) (1986) 446–449.
[8] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, Journal of Computational Physics 73 (2) (1987) 325–348.
[9] M.S. Warren, J.K. Salmon, Astrophysical N-body simulations using hierarchical tree data structures, in: Proceedings of Supercomputing'92, IEEE, 1992, pp. 570–576.

[10] M.S. Warren, J.K. Salmon, A portable parallel particle program, Computer Physics Communications 87 (1–2) (1995) 266–290.
[11] Y.M. Marzouk, A.F. Ghoniem, Vorticity formulation for an actuated jet in crossflow, in: 42nd Aerospace Sciences Meeting and Exhibit, No. AIAA-2004-0096, AIAA, 2004.
[12] Y.M. Marzouk, Vorticity structure and evolution in a transverse jet with new algorithms for scalable particle simulation, Ph.D. thesis, Massachusetts Institute of Technology (June 2004).
[13] A. Leonard, Computing three-dimensional incompressible flows with vortex elements, Annual Review of Fluid Mechanics 17 (1985) 523–559.
[14] O.M. Knio, A.F. Ghoniem, Numerical study of a three-dimensional vortex method, Journal of Computational Physics 86 (1990) 75–106.
[15] A. Majda, A.L. Bertozzi, Vorticity and Incompressible Flow, Cambridge University Press, Cambridge; New York, 2001.
[16] E.G. Puckett, Vortex methods: an introduction and survey of selected research topics, in: M.D. Gunzburger, R.A. Nicolaides (Eds.), Incompressible Computational Fluid Dynamics: Trends and Advances, Cambridge University Press, Cambridge; New York, 1993, pp. 335–407.
[17] A.J. Chorin, Numerical study of slightly viscous flow, Journal of Fluid Mechanics 57 (1973) 785–796.
[18] L. Rosenhead, The formation of vortices from a surface of discontinuity, Proceedings of the Royal Society A 134 (1931) 170–192.
[19] O. Hald, V.M. Delprete, Convergence of vortex methods for Euler's equations, Mathematics of Computation 32 (143) (1978) 791–809.
[20] J.T. Beale, A. Majda, Vortex methods: 1. Convergence in three dimensions, Mathematics of Computation 39 (159) (1982) 1–27.
[21] C. Anderson, C. Greengard, On vortex methods, SIAM Journal on Numerical Analysis 22 (1985) 413–440.
[22] A.F. Ghoniem, F.S. Sherman, Grid-free simulation of diffusion using random-walk methods, Journal of Computational Physics 61 (1) (1985) 1–37.
[23] P. Degond, S. Masgallic, The weighted particle method for convection-diffusion equations: 1. The case of an isotropic viscosity, Mathematics of Computation 53 (188) (1989) 485–507.
[24] S. Shankar, L. van Dommelen, A new diffusion procedure for vortex methods, Journal of Computational Physics 127 (1) (1996) 88–109.
[25] S. Mas-Gallic, The diffusion velocity method: a deterministic way of moving the nodes for solving diffusion equations, Transport Theory and Statistical Physics 31 (4–6) (2002) 595–605.
[26] P. Ploumhans, G.S. Winckelmans, Vortex methods for high-resolution simulations of viscous flow past bluff bodies of general geometry, Journal of Computational Physics 165 (2) (2000) 354–406.
[27] M.C. Soteriou, A.F. Ghoniem, Effects of the free-stream density ratio on free and forced spatially developing shear layers, Physics of Fluids 7 (8) (1995) 2036–2051.
[28] J.D. Eldredge, T. Colonius, A. Leonard, A vortex particle method for two-dimensional compressible flow, Journal of Computational Physics 179 (2) (2002) 371–399.
[29] I. Lakkis, A.F. Ghoniem, Axisymmetric vortex method for low-Mach number, diffusion-controlled combustion, Journal of Computational Physics 184 (2) (2003) 435–475.
[30] A.J. Chorin, J.E. Marsden, A Mathematical Introduction to Fluid Mechanics, Springer-Verlag, Berlin, 1993.
[31] J.K. Salmon, M.S. Warren, Skeletons from the tree-code closet, Journal of Computational Physics 111 (1) (1994) 136–155.
[32] R.A. Gingold, J.J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Monthly Notices of the Royal Astronomical Society 181 (2) (1977) 375–389.
[33] L. Hernquist, N. Katz, TreeSPH – a unification of SPH with the hierarchical tree method, Astrophysical Journal Supplement Series 70 (2) (1989) 419–446.
[34] Z.H. Duan, R. Krasny, An Ewald summation based multipole method, Journal of Chemical Physics 113 (9) (2000) 3492–3495.
[35] Z.H. Duan, R. Krasny, An adaptive treecode for computing nonbonded potential energy in classical molecular systems, Journal of Computational Chemistry 22 (2) (2001) 184–195.
[36] A.H. Boschitsch, M.O. Fenley, W.K. Olson, A fast adaptive multipole algorithm for calculating screened Coulomb (Yukawa) interactions, Journal of Computational Physics 151 (1) (1999) 212–241.
[37] T. Schlick, R.D. Skeel, A.T. Brunger, L.V. Kale, J.A. Board, J. Hermans, K. Schulten, Algorithmic challenges in computational molecular biophysics, Journal of Computational Physics 151 (1) (1999) 9–48.
[38] L. van Dommelen, E.A. Rundensteiner, Fast, adaptive summation of point forces in the two-dimensional poisson equation, Journal of Computational Physics 83 (1) (1989) 126–147.
[39] C.R. Anderson, An implementation of the fast multipole method without multipoles, SIAM Journal on Scientific and Statistical Computing 13 (4) (1992) 923–947.
[40] J.E. Barnes, A modified tree code: Don't laugh, it runs, Journal of Computational Physics 87 (1) (1990) 161–170.
[41] A. Grama, V. Sarin, A. Sameh, Improving error bounds for multipole-based treecodes, SIAM Journal on Scientific Computing 21 (5) (2000) 1790–1803.
[42] L. Greengard, V. Rokhlin, The rapid evaluation of potential fields in three dimensions, Lecture Notes in Mathematics 1360 (1988) 121–141.

[43] J.H. Strickland, R.S. Baty, A pragmatic overview of fast multipole methods, in: J. Renegar, M. Shub, S. Smale (Eds.), The Mathematics of Numerical Analysis: 1995 AMS-SIAM Summer Seminar in Applied Mathematics, American Mathematical Society, Providence, RI, 1996, pp. 807–830.
[44] H.G. Petersen, D. Soelvason, J.W. Perram, E.R. Smith, The very fast multipole method, Journal of Chemical Physics 101 (10) (1994) 8870–8876.
[45] L. Greengard, V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numerica 6 (1997) 229–269.
[46] H. Cheng, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, Journal of Computational Physics 155 (2) (1999) 468–498.
[47] G.S. Winckelmans, A. Leonard, Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows, Journal of Computational Physics 109 (2) (1993) 247–273.
[48] G.S. Winckelmans, J.K. Salmon, M.S. Warren, A. Leonard, B. Jodoin, Application of fast parallel and sequential tree codes to computing three-dimensional flows with the vortex element and boundary element methods, ESAIM Proceedings: Vortex Flows and Related Numerical Methods II 1 (1996) 225–240.
[49] J.K. Salmon, M.S. Warren, G.S. Winckelmans, Fast parallel tree codes for gravitational and fluid dynamical *N*-body problems, International Journal of Supercomputer Applications and High Performance Computing 8 (2) (1994) 129–142.
[50] D.H. Wee, Y.M. Marzouk, A.F. Ghoniem, Fast, parallel, hierarchical N-body solvers for arbitrary kernel, Reacting Gas Dynamics Laboratory report, MIT, http://centaur.mit.edu/rgd, 2004.
[51] S. Bhatt, P. Liu, V. Fernandez, N. Zabusky, Tree codes for vortex dynamics: application of a programming framework, in: Workshop on Solving Irregular Problems on Parallel Machines, International Parallel Processing Symposium, Santa Barbara, CA, 1995.
[52] M.R. Anderberg, Cluster Analysis for Applications, Academic Press, New York, 1973.
[53] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, Wiley, New York, 2001.
[54] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: L.M.L. Cam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, Berkeley, 1967, pp. 281–297.
[55] L. Bottou, Y. Bengio, Convergence properties of the k-means algorithmAdvances in Neural Information Processing Systems, vol. 7, MIT Press, Denver, 1995.
[56] I.S. Dhillon, D.S. Modha, A data-clustering algorithm on distributed memory multiprocessors, in: M.J. Zaki, C.-T. Ho (Eds.), Large-scale Parallel Data Mining, Lecture Notes in Artificial Intelligence, vol. 1759, Springer-Verlag, Berlin, 2000, pp. 245–260.
[57] D. Pelleg, A. Moore, Accelerating exact k-means algorithms with geometric reasoning, in: KDD-99, in: Proceedings of the Fifth ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, 1999, pp. 277–281.
[58] Y.M. Marzouk, A.F. Ghoniem, Mechanism of streamwise vorticity formation in a transverse jet, in: 40th Aerospace Sciences Meeting and Exhibit, No. AIAA-2002-1063, AIAA, 2002.
[59] F. Aurenhammer, Voronoi diagrams – a survey of a fundamental geometric data structure, ACM Computing Surveys 23 (3) (1991) 345–405.
[60] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy, Load balancing and data locality in adaptive hierarchical *N*-body methods – Barnes–Hut, fast multipole, and radiosity, Journal of Parallel and Distributed Computing 27 (2) (1995) 118–141.
[61] M.S. Warren, J.K. Salmon, A parallel hashed oct-tree *N*-body algorithm, in: Proceedings of Supercomputing'93, IEEE, 1993, pp. 12–21.
[62] H. Samet, The design and analysis of spatial data structuresAddison-Wesley Series in Computer Science, Addison-Wesley, Reading, MA, 1990.
[63] A. Grama, V. Kumar, A. Sameh, Scalable parallel formulations of the Barnes–Hut method for *N*-body simulations, Parallel Computing 24 (5–6) (1998) 797–822.
[64] S.-H. Teng, Provably good partitioning and load balancing algorithms for parallel adaptive *N*-body simulation, SIAM Journal on Scientific Computing 19 (2) (1998) 635–656.
[65] M. Parashar, J.C. Browne, On partitioning dynamic adaptive grid hierarchies, in: Proceedings of the 29th Annual Hawaii International Conference on System Sciences, 1996, pp. 604–613.
[66] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.
[67] H.D. Simon, S.H. Teng, How good is recursive bisection? SIAM Journal on Scientific Computing 18 (5) (1997) 1436–1445.
[68] T.F. Fric, A. Roshko, Vortical structure in the wake of a transverse jet, Journal of Fluid Mechanics 279 (1994) 1–47.
[69] Y.M. Marzouk, A.F. Ghoniem, Vorticity structure and evolution in transverse jets, Journal of Fluid Mechanics (2005), submitted for publication.
[70] Y.M. Marzouk, A.F. Ghoniem, D. Wee, Simulations of high Reynolds number transverse jets and analysis of the underlying vortical structures, in: 43rd Aerospace Sciences Meeting and Exhibit, No. AIAA-2005-0308, AIAA, 2005.
[71] A.J. Chorin, Hairpin removal in vortex interactions II, Journal of Computational Physics 107 (1993) 1–9.
[72] W.T. Ashurst, E. Meiburg, Three-dimensional shear layers via vortex dynamics, Journal of Fluid Mechanics 189 (1988) 87–116.