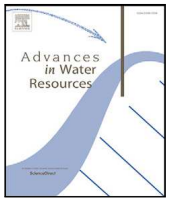
Advances in Water Resources 193 (2024) 104818

Contents lists available at ScienceDirect

Advances in Water Resources

journal homepage: www.elsevier.com/locate/advwatres



Parallelization of particle-mass-transfer algorithms on shared-memory, multi-core CPUs

David A. Benson [a,*], Ivan Pribec [b], Nicholas B. Engdahl [c], Stephen Pankavich [d], Lucas Schauer [d]

[a] Hydrologic Science and Engineering, Colorado School of Mines, Golden, CO 80401, USA
[b] Leibniz Supercomputing Center, Boltzmannstraße 1, D-85748 Garching bei München, Germany
[c] Department of Civil and Environmental Engineering, Washington State University, Pullman, WA, United States of America
[d] Department of Applied Mathematics and Statistics, Colorado School of Mines, Golden, CO, 80401, USA

ARTICLE INFO

Keywords:
Particle methods
Mass-transfer
SPH
Parallelization

ABSTRACT

Simulating the transfer of mass between particles is not straightforwardly parallelized because it involves the calculation of the influence of many particles on each other. Engdahl et al. (2019) intuited that the number of matrix operations used for mass transfer grows quadratically with the number of particles, so that dividing the domain geometrically into sub-domains will give speed and memory advantages, even on a single processing thread. Those authors also showed the speed scalability of several one-dimensional examples on multiple cores. Here, we extend those results for more general cases, both in terms of spatial dimensions and algorithmic implementation. We show that there is an optimal subdivision scheme for naive, full-matrix calculations on a multi-processor, or multi-threading shared-memory machine. A similar sparse-matrix implementation that also uses row-and-column-sum normalization often greatly reduces the memory requirements. We also introduce a completely new mass transfer algorithm that uses a non-geometric domain decomposition and only matrix row-sum normalization. This allows the mass-transfer "matrix" to be constructed and solved one row at a time in parallel, so it is faster and vastly more memory efficient than previous methods, but requires more care for suitable accuracy.

1. Introduction

Traditional (non-interacting) particle-tracking methods are often used for advection-dominated groundwater systems because of their numerical stability and because they simulate advection without incurring spurious numerical diffusion that impacts dispersion, dilution, and mixing (Labolle et al., 1996; Pérez-Illanes and Fernàndez-Garcia, 2024). This is especially important when simulating chemically-reactive transport because the typically nonlinear reactions are highly sensitive to concentration fluctuations and/or errors (Tartakovsky et al., 2012; Paster et al., 2014). In order to simulate chemical reactions, the particles must either exchange mass, or have their masses mapped to concentrations through spatial averaging (Tompson and Dougherty, 1992; Benson and Bolster, 2016; Sole-Mari and Fernàndez-Garcia, 2018; Sole-Mari et al., 2020; Perez et al., 2019a). The mass-transfer particle tracking (MTPT) techniques move mass between all particles according to their proximity, and any geochemical reactions may be completed on the particles. The MTPT algorithm is uniquely useful for simulating transport and chemical reaction because it does not necessarily solve an upscaled, volume-averaged advection–dispersion-reaction equation. First, MTPT is able to realistically simulate mixing at a rate slower than spreading because the two processes are simulated separately (Benson et al., 2019). As a result, a single diffusion coefficient is not forced to represent the two processes in one equation. Second, the MTPT methods simulate a stochastically-perturbed equation and may faithfully preserve the sub-grid fluctuations of velocity, dispersion, and/or chemical concentrations (Schmidt et al., 2017; Ding et al., 2017; Benson et al., 2017, 2019; Sole-Mari et al., 2017; Sole-Mari and Fernàndez-Garcia, 2018). In the case of highly nonlinear chemical reactions, these perturbations may dominate the overall behavior of the systems.

The capability of MTPT to simulate systems undergoing chemical reaction in a purely Lagrangian manner was unlocked by allowing each particle to carry an unlimited number of chemical species (Bolster et al., 2016; Benson and Bolster, 2016). The particles (which may be fluid or solid, Schmidt et al. (2019, 2020)) undergo reactions occurring at the particle level. This means that mass must move *between* numerical particles in a physically realistic way. This mass-transfer may, therefore, be formulated as a true, local, mixing process (Benson et al., 2020; Tran et al., 2021), which is straightforward to implement when a good

* Corresponding author.
E-mail address: dbenson@mines.edu (D.A. Benson).

https://doi.org/10.1016/j.advwatres.2024.104818
Received 16 March 2024; Received in revised form 8 September 2024; Accepted 9 September 2024
Available online 11 September 2024

approximation of the probability distribution (i.e., the Green's function) of local diffusion or transverse dispersion is known (Benson et al., 2019; Perez et al., 2019b).

A physically-based model for the amount of mass transfer between any two particles originally was given by the probability of particle "interaction" based on their separation vector (Benson and Meerschaert, 2008). For example, a locally isotropic and Fickian diffusion process dictates that the mass transfer between any two particles labeled i and j is proportional to the density function of their co-location (Benson and Meerschaert, 2008; Schmidt et al., 2018) and given by

$$p_{i,j} = \frac{1}{(2\pi(D_i + D_j)\Delta t)^{d/2}} \exp\left[\frac{-|\mathbf{x}_i - \mathbf{x}_j|^2}{2(D_i + D_j)\Delta t}\right], \quad (1)$$

where $p_{i,j}$ is the probability that the two particles will interact over a time-step of length Δt, D_i and D_j are each particle's local diffusion coefficient, d is the number of dimensions, and $\mathbf{x}_i$ and $\mathbf{x}_j$ are the particle position vectors (Benson and Meerschaert, 2008; Schmidt et al., 2018). This is easily generalized when the local dispersion is anisotropic and each D is a $d \times d$ matrix (Appendix). These probability density values dictate the proportion of the mass of each species that is transferred between particles or the amount that stays "at home" when $i = j$. We denote this specific approach as Mass-Transfer Particle-Tracking or MTPT. Clearly the mass transfer decreases rapidly with particle separation distance (which can be exploited algorithmically), but theoretically, each particle can interact with all others, even if the amounts are minuscule. This latter feature becomes a complication and a bottleneck when compared to traditional PT methods, in which each particle is completely independent of all others.

The trade-off for these new capabilities of MTPT is increased computational complexity. A naive application of Eq. (1) among a total of N_P particles in a domain would build an $N_P \times N_P$ matrix of all particle interaction probabilities (Appendix). For more than a few tens of thousands of particles, the full interaction matrix approach would exceed the memory capacity of most desktop computers. This approach would also be inefficient because many of the particle interactions are negligible due to large separation distances. As a result, the full-matrix approach required improvement to make it computationally feasible. One of the traditional ways of reducing the size of large matrices is called domain decomposition, which can be visualized as a geometric process in which the physical domain is subdivided into a set of adjacent tiles (Fig. 1). Computations can be performed on these subsets of the domain, as long as potential interactions between adjacent tiles are properly treated. These geometric-based decompositions include very simple "checkerboard" tilings (Schauer et al., 2023) or more advanced tree-based subdivisions, including quad-tree (2-d), oct-tree (3-d), and kd-tree (Bentley, 1975; Kennel, 2004), wherein each tile is created to possess approximately equal numbers of particles. In this paper we also develop a novel non-geometric, or list-based, decomposition that simplifies computations significantly. The common thread is that any decomposition seeks to reduce the overall computational load and has the added benefit that parallel processing threads may handle the subdivisions simultaneously.

The foundations for parallelizing MTPT were explored by Engdahl et al. (2019) who found that simply subdividing the spatial domain resulted in a reduction of the total number of calculations, or floating-point operations (flops) for a full-matrix formulation. Even when implemented serially on a single-processor machine, the algorithm showed remarkably linear speedup with the number of subdivisions. The basis of the speedup derives from the fact that a forward full-matrix calculation requires a number of flops proportional to n^2, where n is the number of rows and/or columns. Using Eq. (1) on all N_P particles in a simulation requires roughly kN_P^2 flops, where k is an arbitrary constant related to the difficulty of certain operations (e.g., calculating particle separations versus actual mass transfer). In a perfect world, subdividing the particles into S roughly equal load-balanced sub-domains means that each sub-domain requires $k(N_P/S)^2$ flops for a total of $S \times k(N_P/S)^2 = kN_P^2/S$ flops. However, the approach still built a "full" interaction matrix for all particles within each subdomain, leaving significant room for improvement.

Despite the notable speedup, the performance of the simple domain decomposition algorithm can be rapidly outpaced by that of tree-based search algorithms. These searches can efficiently find particles within a defined radius, so that more distant particles can be ignored and the matrices become much more sparse, which serves the dual purpose of reducing the number of flops and significantly decreasing memory requirements. Depending on the tree construction and data clustering, each search among N_P particles in a single tree may scale somewhere between $\mathcal{O}(\ln N_P)$ and $\mathcal{O}(N_P^2)$, with $\mathcal{O}(N_P)$ being typical (Finkel and Bentley, 1974). The gains afforded by search trees means domain decomposition might not be necessary in many cases, but the key advantage of the decomposition is that it allows the set of particles to be distributed among a collection of processors when each has its own memory (Schauer et al., 2023). However, the question of speedup on a shared memory system is not as clear. For example, subdividing the domain reduces computations within each subdivision, but if these computations are performed in parallel, the demands on shared memory increase substantially. In short, performance and efficiency under a shared memory architecture are under-explored facets of the MTPT acceleration schemes, and the question becomes: How fast can the MTPT methods be made without sacrificing accuracy?

In this paper, we examine the effects of problem physics, user decisions, and algorithm design on the efficiency of the MTPT method. Here, we focus exclusively on shared-memory multi-core and multi-threading CPUs because of the ease of implementation using OpenMP directives. This allows us to concentrate on speed and memory requirements of the algorithms themselves. We investigate geometric domain decomposition using the full-matrix scheme of Engdahl et al. (2019) and a sparse-matrix version thereof. Even though it is known to be a non-optimized scheme, the full matrix version provides a standard benchmark against which several improvements may be judged. To this end we also develop and test a non-geometric, list-based decomposition that should allow for optimal parallelization.

Before embarking on the analyses, we emphasize that the term "core" refers to a separate processing unit on a multiprocessor chip. Each core makes calculations independent of each other. Any core will have some idle time when making calculations and moving data to and from memory, so compilers may take advantage and run other sets of instructions on a core during idle time in a process called multi-threading. The term "thread" refers to a set of calculations performed on a core, and (for example) the parallel compiler commands in OpenMP allow two threads to be run simultaneously per core. A single thread on a core runs at full speed, but a second thread, looking for idle time, will run somewhat slower than the "first". Most users looking for parallel performance will use all available threads when they get a chance, so we will show performance for all thread numbers, up to twice the number of physical cores, so scaling behaviors will change when more threads are requested than physical cores. Our estimates of speedup below are based on each core running at full speed.

2. Parallelization memory requirements and speed

Before exploring the various means of parallelization, it is instructive to examine the basic operation of the mass transfer algorithm. A vector of the masses on all particles at any time t is advanced over some finite timestep via a forward matrix multiplication $\mathbf{P} \times \boldsymbol{m}(t) = \boldsymbol{m}(t + \Delta t)$ (see, e.g., Schmidt et al. (2018), Engdahl et al. (2019)). The matrix $\mathbf{P}$ encodes the degree to which each particle pair will exchange mass, based on their separation distance and direction. To conserve mass, the matrix is doubly-stochastic, being square, symmetric, with all row- and column-sums equal to unity (Appendix). Either during the creation of the initial condition, or during disordered transport, the particle masses in the vector $\boldsymbol{m}(t)$ may become irregularly positioned in the

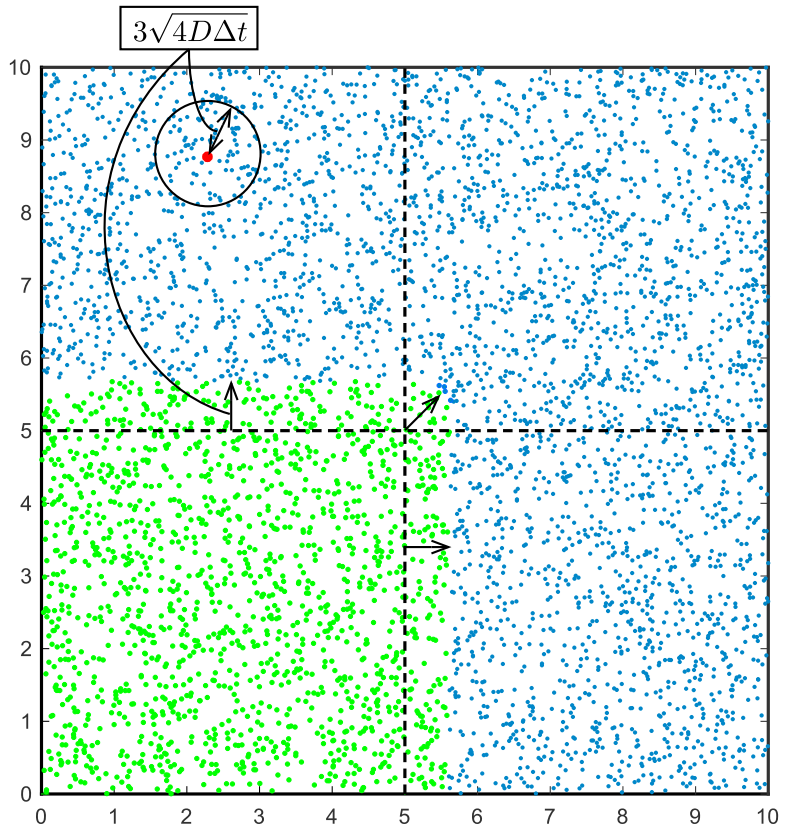


Fig. 1. Schematic of domain decomposition. Full domain shown here is $[0,10]\times[0,10]$. Four geometric subdivisions are denoted by the dashed lines. Computations in the first subdomain $[0,5]\times[0,5]$ involve particles highlighted in green, which includes so-called "ghost particles" located within a distance of $\approx 3\sqrt{4D\Delta t}$ of the subdomain. For list-based decomposition, only those particles within that radius of any particle (i.e., the red particle) must be considered.

domain. As a result, the matrix **P** in the operation of mass transfer will be non-structured (Fig. 4). Furthermore, many of the entries in **P** will be negligible in magnitude and may be omitted, as in Fig. 4, leading to a sparse matrix. When the matrix is first constructed by applying Eq. (1) for each particle pair i, j, it is symmetric, inasmuch as the Gaussian probability density associated with any two particles is the same, irrespective of the order. However, because of the particles' discrete nature of approximating the continuous Gaussian, the rows will not exactly sum to unity (as required for mass conservation — see Appendix). One method (Herrera et al., 2009; Schmidt et al., 2018) normalizes each value in the matrix by the average of its row-sum and column-sum. For the red-highlighted entry in Fig. 4, this is represented by the red highlighting of row 20 and column 71. Because of matrix symmetry, the symmetric "partner" probability (the blue entry at the end of the arrow) has a column-sum equal to the row-sum of the partner and vice-versa (denoted by the blue row and column). Therefore, normalization using an arithmetic average of the row- and column-sums maintains matrix symmetry. However, because each value in a row gets a unique renormalization factor, the row sums (hence column sums) are not guaranteed to sum to unity. The renormalization can be repeated (this is called the Sinkhorn–Knopp algortihm (Knight, 2008; Young, 2024)) until a suitable closure is reached.

A new matrix normalization algorithm (Appendix), eliminates the column sums by estimating the particle spacing *a priori*. Then the row-sum alone is used to estimate probabilities and any discrepancy between the row-sum from unity is adjusted at the main diagonal term alone. This algorithm maintains whole matrix symmetry, and also provides a new form of parallelism, because without the column-sums, a single row may be created, and the mass transfer calculated, independently of all others. The rows may be calculated in the order they are listed in memory, on different cores or threads, without any regard for the actual particles' spatial locations. We call this algorithm "list-based decomposition", because any subset of rows corresponding to a list of particles may be sent to a processor. This method relies on normalization using the row-sums alone so that any row of the matrix is independent of all others. Because this method is completely new, we provide an analysis of the accuracy immediately.

2.1. List-based algorithm accuracy

When $\beta = 1$, the solution progresses via convolution of the probability matrix with the current masses (Appendix). Therefore, the probabilities in matrix **P** are discrete approximations of the Green's function of the local mixing — without loss of generality a Gaussian is used here. In the new algorithm, the "peak" value is adjusted, based on the sum (discrete integral) of the probability values in a row. Of course, for multiple time steps, the continued convolution of this near-Gaussian with itself will converge to a Gaussian, but for a smaller number of timesteps, and a smaller number of particles, how error-prone is the method?

First, we simulate two similar scenarios as for the speed tests that follow: a 2-d domain $\Omega = [0,10]\times[0,10]$, $D = 10^{-3}$, total time = 100, with $\Delta t = 10$ and 1. A single particle at the center of the domain is given a mass of Ω/N_P, where N_P is the total number of particles. A range of particle numbers were used, giving a range of particle spacings $\Delta x = (\Omega/N_P)^{1/2}$. Here we use uniform, non-random spacing and no random walks to isolate the performance of the algorithm in its pure form. The performance is judged by the RMSE between particle masses domain-wide and the analytical 2-d Gaussian Green's function.

Clearly, as the number of particles is increased (and the inter-particle spacing, Δx is decreased), the discrete integral of the Gaussian approaches unity and the adjustment at the center particle (i.e., $\mathbf{P}_{i,i}$) tends to zero. The transition from poor fits to essentially perfect fits

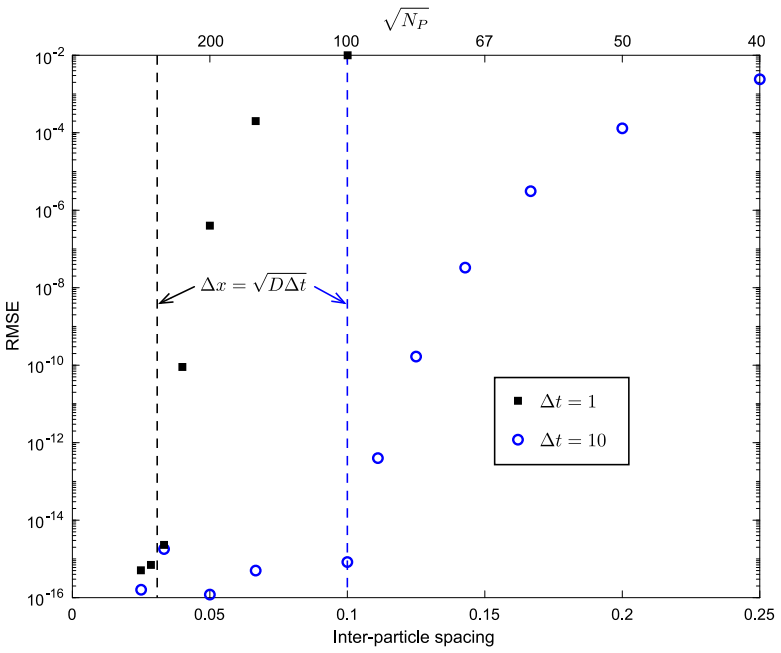


Fig. 2. RMSE difference between simulations by new algorithm and Gaussian analytic solution, as a function of uniform, stationary particle spacing (i.e., number of particles N_P), defined by $(\Omega/N_P)^{1/d}$. Two sets of simulations were run, with $D = 10^{-3}$ and $\Delta t = 1$ and 10 for total times of 100. All simulations are run in a 2-d domain of size $[0, 10] \times [0, 10]$.

occurs fairly rapidly, at around $\Delta x \approx \sqrt{D\Delta t}$ (Fig. 2). For smaller spacings, the error drops to the level of machine precision ($\sim 10^{-16}$).

Constructing a discrete version of the Green's function is best done with regularly-spaced (and stationary) particles, as in the last example. But when particles move due to differential advection and/or random walks, the particle spacings, hence number of particles in each matrix row, may change drastically. There is no explicit analytic solution for this problem, so we compare the new algorithm to the row- and column-normalized method that has been tested extensively elsewhere (Sole-Mari et al., 2019a). We generated a divergence-free, two-dimensional velocity field $\mathbf{v}(\mathbf{x})$ and placed 20,000 initially mass-free particles randomly in a circle with radius = 10 surrounding a single particle with mass representing a Dirac delta function (Fig. 3a). The velocity field was generated using a finite-difference solution of the equation $\nabla \cdot (K\nabla h) = 0$, with Dirichlet conditions for hydraulic head $h = 1$ and $h = 0$ on the left and right and $\partial h/\partial y = 0$ on the top and bottom boundaries (Fig. 3a), and a grid spacing of $\Delta x = \Delta y = 1$. The underlying random correlated hydraulic conductivity (K) field is log-Normal with an isotropic correlation length of $ln(K)$ of 10 length units, $E(ln(K)) = 0$ and $\sigma(ln(K)) = 0.5$. Once h was solved, the velocity is given at each grid-cell interface by $\mathbf{v}(\mathbf{x}) = K\nabla h$. Velocity vectors at each particle were then calculated by linear interpolation of interface velocities, preserving the divergence-free velocity (Labolle et al., 1996). The number of particles was chosen so that the inter-particle spacing $\Delta x = (\pi 10^2/20000)^{1/2} = 0.125$, giving each finite-difference cell on the order of 64 particles for good resolution of the sub-grid velocity. The same random seed was used for initial particle positions for both mass transfer algorithms, so the particles would be advected to the same positions at any time. We chose an isotropic mass-transfer dispersion coefficient $D = D_{mol} + \alpha_T|\mathbf{v}|$, with $D_{mol} = 10^{-8}$, $\alpha_T = 5 \times 10^{-3}$, and the mean velocity was on the order of 0.05.

It is important to note that the optimal time steps for advection and mass transfer are very different. From Fig. 2 we see that the algorithm requires the condition $\Delta x = (V/N_p)^{1/d} < \sqrt{D\Delta t}$, or $\Delta t > (V/N_p)^{2/d}/D$. Using the largest D in the field, this gave a value of $\Delta t > 35$. On the other hand, the particles were advected using simple forward Euler methods and limited to traversing 1/2 of each underlying velocity grid with block of size 1 on each side, giving $\Delta t < 5$. As a result, during the operator splitting, a number of advective steps were performed before each mass transfer step.

The plume is split and strongly elongated in the mean travel direction (Fig. 3a). The plume, shown in red, is also elongated and slightly multi-modal (Fig. 3b). Even with very different normalization algorithms, the two methods agree quite well, even at extremely low concentrations (Fig. 3b,c).

2.2. Memory-usage comparison

It is clear that the row-sum normalization will be faster than the row-and-column-sum normalization for the same problem, simply because the column-sum calculations are eliminated. Also, the calculations that determine which particles are in a geometric subdomain are eliminated. But another major concern is the memory requirement. Referring to Fig. 1, we can estimate bounds on how much more computer memory the domain decomposition with row- and column-normalization will consume.

2.2.1. Full-matrix, domain decomposition

For a full-matrix computation, the number of particles within a subdomain, plus the neighboring "ghost particles" that may influence particles inside that subdomain (Fig. 1), define the number of rows in one subdomain's matrix (full or sparse) (Fig. 4). For a domain with fairly uniform particle density and cubic subdivisions, the total number of rows in a subdomain is somewhere between N_P/S and $N_P/S + (N_P/S)(2 \times d \times 3\sqrt{4D\Delta t}(\Omega/S)^{(d-1)/d})$, where N_P is the total number of particles in the entire domain and S is the number of (evenly distributed) subdivisions (Schauer et al., 2023). The second term counts ghost particles in all directions in d-dimensions. Subdomains along the full domain edge have fewer ghost particles, and the distribution of particles in the domain can also create irregularities, so this is an approximate upper bound. If the number of ghosts is considered small compared to the subdomain size, then a fair approximation for the number of rows is just N_P/S, and if C cores are used simultaneously (each given a separate subdomain so that $C \leq S$), then the number of matrix entries required for the parallel, full-matrix domain-decomposition method (denoted M_{FM}), has a lower bound of

$$M_{FM} > C\left(\frac{N_P}{S}\right)^2. \tag{2}$$

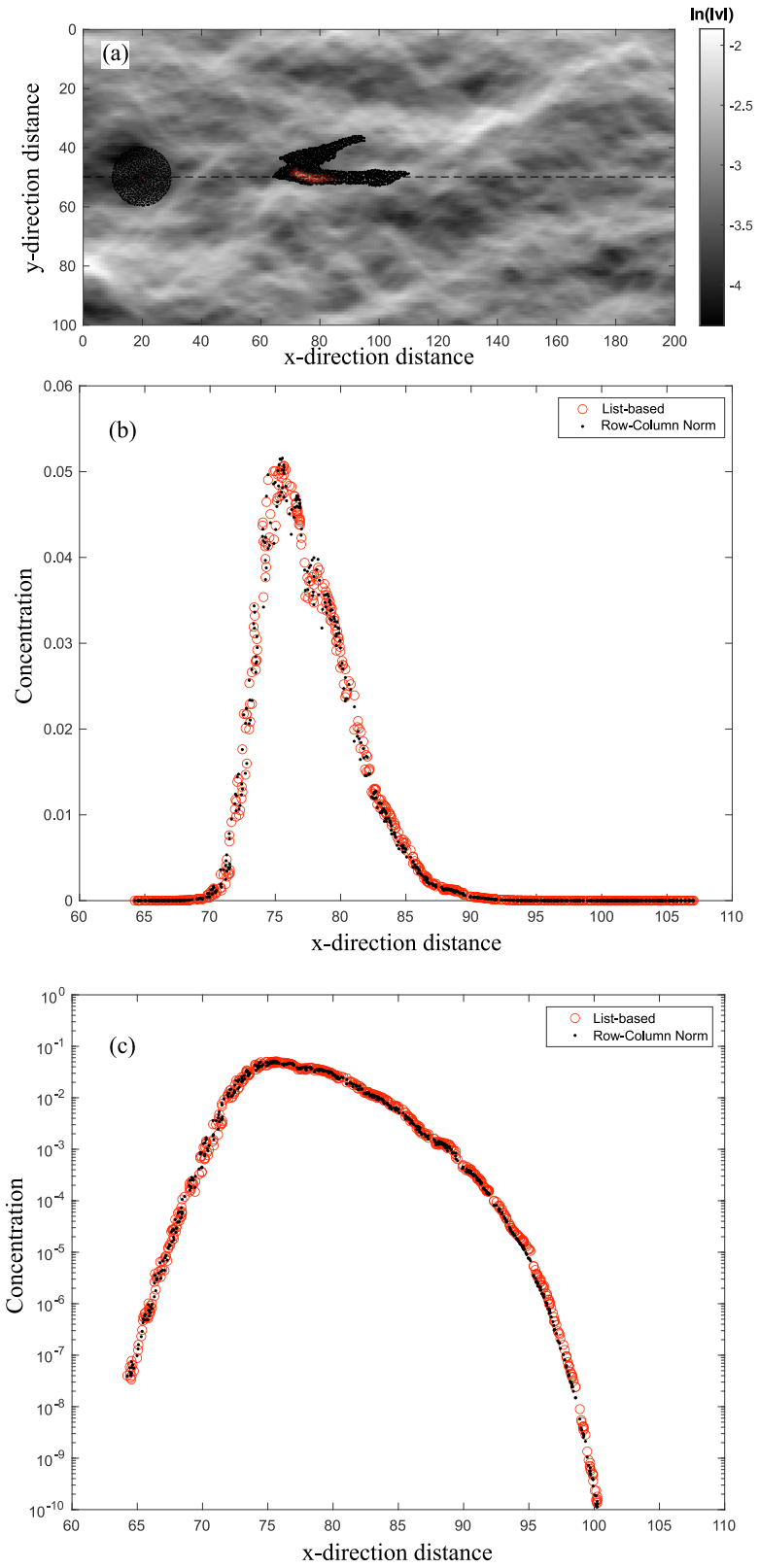


Fig. 3. (a) Plume of 20,000 particles initially placed surrounding a single particle (red) with mass at $[x, y] = [20, 50]$ (b) Linear and (c) Semi-log plots of concentrations along the line $y = 50$ from the row-normalized, list-based algorithm (circles) and the row-column-normalized (dots) algorithm.

2.2.2. *Sparse-matrix, domain decomposition*

A sparse matrix row stores only the entries for particles within the region of influence (defined by the user), of the ith particle, as some multiple of the local diffusion distance. This is the average particle density times the volume of that region of influence $\frac{N_P}{\Omega}\pi\kappa(\sqrt{36D\Delta t})^d$. The constant κ is a function of the dimensionality of the system: $\kappa = [2/\pi, 1, 4/3]$ for $d = [1, 2, 3]$. Therefore, a sparse matrix has on the order of $\frac{N_P^2}{S\Omega}\pi\kappa(36D\Delta t)^{d/2}$ entries as a lower bound. Given a certain number of

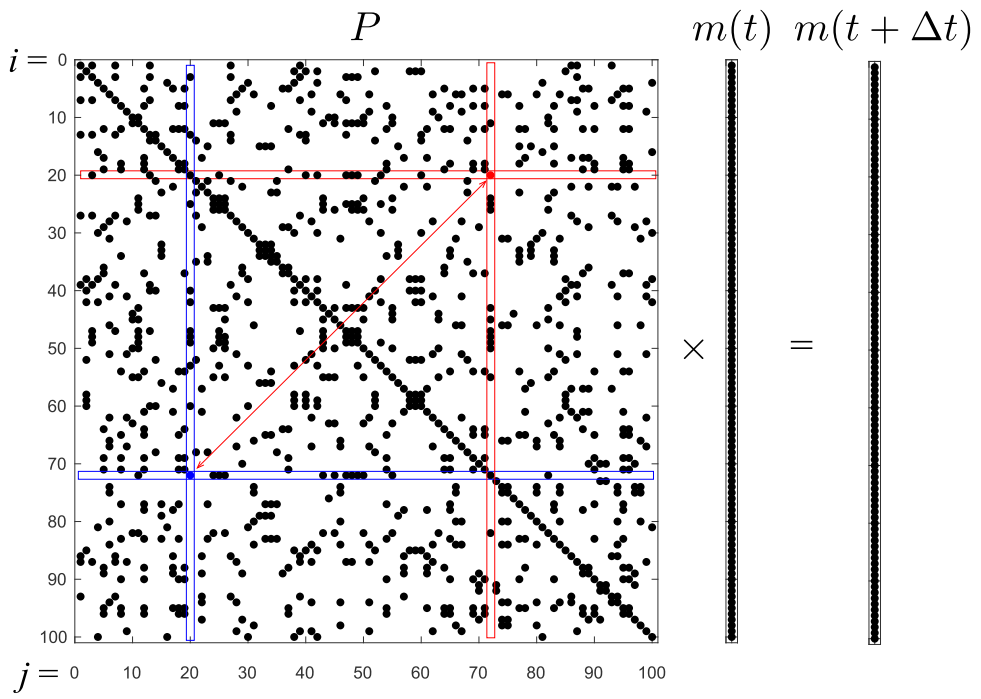


Fig. 4. Snapshot of non-zero entries in the mass-transfer matrix operation for a single timestep. For clarity, only the first 100 particles are shown out of a 10,000-particle simulation.

cores, C, on which to make the calculation in parallel, the total number of entries held in memory in the separate **P** matrices is on the order of

$$M_{SM} \approx C \frac{N_P^2}{S\Omega} \pi\kappa (36 D\Delta t)^{d/2}. \tag{3}$$

The ratio of total full- to sparse-matrix entries for any given simulation (M_{FM}/M_{SM}) is therefore on the order of $\frac{\Omega/S}{\pi\kappa(36D\Delta t)^{d/2}}$. The numerator is the volume of real particles in a subdomain and the denominator is a measure of the ghost particle volume. When the latter is as large as the former, then the sparse matrices are full again, as every particle in the subdomain has an influence on all others. In other words, the sparse matrices only "help" when the user-controlled ratio $\frac{\Omega/S}{\pi\kappa(36D\Delta t)^{d/2}}$ is large.

2.2.3. List-based decomposition

Particle attributes are typically held in some sort of array, and as the particles move around the domain under the influence of heterogeneous velocity and random walks representing macrodispersion, there is no guarantee that neighbors in the array are also neighbors in the spatial domain. This leads to an unstructured matrix and the need to consider all rows when employing previous row- and column-normalization. The large matrices then necessitate the creation of geometric subdomains. In the methods previously discussed, a normalization of the probability values in a matrix row must also be transferred to all of the other rows that represent nearby particles. In other words, a row i contains information about nearby particles with values of j that may be anything between 1 and N_P. Any changes to these values (the $\mathbf{P}_{i,j}$s) must then be reflected in each of the many j rows to maintain a symmetric **P** (Appendix).

A possible solution to this bottleneck in performance is to modify the previous algorithms so that only row *or* column normalization is required. We propose a new list-based decomposition that adjusts only the diagonal value in each row to ensure each row-sum is equal to unity (Appendix). Doing so, all values of $\mathbf{P}_{i,j}$ remain unchanged for $i \neq j$ during the renormalization process and symmetry is always maintained. Any single row (any single particle's new mass) may now be calculated independently of all others, and the calculations can proceed down the "list" of particles, irrespective of their spatial positions. Arbitrary portions of the list can be sent to different processors in chunk sizes that are best suited for the problem at hand and the computational architecture. The caveat to this is that updated masses $\boldsymbol{m}(t + \Delta t)$ must be written to a temporary array until the list is completed, but this is simply a vector of length N_P for each compound, rather than a square matrix.

In practice, the j particles surrounding particle i are generally never exactly spaced $\Delta\mathbf{x}$ away from each other because of random or non-uniform particle motion. The non-uniform spacing gives a non-unit total probability, and the difference is absorbed by the i particle. This also means that widely spaced particles (i.e. fewer than average particles) will more likely integrate to less than unity, and the probability of mass staying "home" at the i node is increased. Overall this reflects less mixing when local particle densities are lower than average. This can be the result of divergence-free fluid deformation (Engdahl et al., 2014), which may increase or decrease mixing depending on the relationship of concentration gradients to particle position changes.

In the list-based decomposition, a single particle (i.e. matrix row) may be computed independently of all other rows. Each matrix row has the number of entries in the surrounding volume already shown to be $\frac{N_P}{\Omega}\pi\kappa(36D\Delta t)^{d/2}$. Using C cores simultaneous means a list-based memory consumption (M_L) of

$$M_L \approx C \frac{N_P}{\Omega} \pi\kappa (36 D\Delta t)^{d/2}. \tag{4}$$

Therefore, the ratio of the sparse row-column normalization to the list-based row-normalization (M_{SM}/M_L) is simply N_P/S. Under normal conditions, the number of particles far outnumbers the number of subdivisions — this ratio is typically in the hundreds to millions range.

To give an idea of the scaling, imagine a typical 2-d simulation with 100,000 particles, $D = 10^{-3}$, $\Delta t = 10$, and $S = 100$ subdivisions run on $C = 10$ cores. Then the three methods require storing, for the MT matrices, $> 10^7$, 1.9×10^7, and 1.9×10^4 entries, respectively. In this case, the full-matrix and the sparse-matrix methods are comparable because the subdomains are small enough (relative to the ghost-particle distance) that the sparse matrices are essentially full, and the list-based algorithm requires 1000 times less memory.

2.3. Computation time and parallel speedup

Intuitively, one would like to use as many threads as are within reach for any given simulation. But this can be a problem with diminishing or even decreasing returns due to the physics of the problem and/or the structure of the processors and memory. For the sake of analysis, we will assume here that the problem domain Ω is relatively uniformly filled with N_P particles. We may define the domain as rectilinear with volume $\Omega_x \times \Omega_y \times \Omega_z$. For the geometric domain decomposition, the domain is subdivided by integers S_x, S_y, S_z, so that the total number of sub-domains is $S = S_x S_y S_z$. Also for simplicity of overall analysis we assume that D is a constant. Here we are only looking at performance as a function of domain partitioning, so a simple "checkerboard" decomposition is very close to optimally load-balanced. The particle influence probability density Eq. (1) implies that particles within about three standard deviations of a sub-domain should be included as ghost particles (e.g., Fig. 1).

For each of the following examples that have accompanying simulations, Fortran codes were compiled with the GNU gfortran (gcc) compiler version 13.1 using -O3 optimization and OpenMP directives. The fortran codes may be found in the Github repository https://github.com/dbenson5225/parallel_particle_mass_transfer. Unless otherwise noted, the simulations use a 2-d domain of size $[0, 10]\times[0, 10]$, with $N_P = 100,000$ particles, $D = 10^{-3}$, $\Delta t = 10$, and total time = 100.

2.3.1. Tree-based particle searches

One of the keys to either geometric or list-based decomposition is using an efficient algorithm for nearest-neighbor searches, because a naive brute-force search through N_P particles requires $\mathcal{O}(N_P^2)$ calculations. Tree-based algorithms, including quad-tree, kd-tree, and recursive-coordinate-bisection, subdivide the domain (in various ways) into smaller and smaller chunks, keeping track of which chunks are nearest to each other (Finkel and Bentley, 1974; Bentley, 1975; Berger and Bokhari, 1987). When a request is made to find particles near one another, the algorithm only needs to look at particles in nearby chunks, eliminating extra calculations. In previous work (Paster et al., 2014; Benson et al., 2017), we have used a publicly-available Fortran code KDTREE2 (Kennel, 2004), but the code is not easily made safe in a shared-memory parallel environment. Instead, we coded a Fortran version of a Python quad-tree code (https://scipython.com/blog/quadtrees-2-implementation-in-python/) and provide a brief analysis of its performance here.

A quad-tree mesh is based on subdividing a rectangular domain with successive identical quadrants (Finkel and Bentley, 1974). When the number of particles in any quadrant exceeds some user-defined threshold value, that quadrant is subsequently subdivided in equal quadrants. This happens recursively until the final quadrants all have less than the threshold "bucket" number of particles. A large bucket enables faster tree generation because of fewer subdivisions. It is thought that a large bucket also makes for slower searches — especially when looking for a fixed number of nearest neighbors (a kNN search). However, we require all particles within a certain radius, so it is unclear how big the bucket size should be. To illustrate, a bucket size of four particles was initially chosen (Fig. 5a). In keeping with simulations to come later in this paper, the domain is set to $[0, 10] \times [0, 10]$ with total number of particles ranging from 100,000 to 5,000,000. After the trees were generated (typically taking less than a few seconds), a loop was set up to vary a search radius from 0.25 to 5 (half the domain size. The total search time versus the number of particles found shows a clear scaling trend: the search times for a given radius are approximately linear with the total number of particles N_P. However, for large particle numbers there is an obvious performance degradation for the smaller searches. This led us to increase the bucket size systematically over a range of search radii for a single large $N_P = 5 \times 10^6$ (Fig. 5b). In this case, a bucket size of approximately 500 (10^{-4} times the total number of particles) minimizes the search times. We also compared the performance of the (Kennel, 2004) KDTREE2 code (circles, Fig. 5b). That kd-tree code is less susceptible to performance degradation due to bucket size, and is also about 2 times faster than the quad-tree code when both are optimized. As mentioned before, we choose the quad-tree because of ease of parallelization. Interestingly, the kd-tree also performs best with a bucket size of 500 with 5×10^6 total particles.

One might think that the creation of the tree is time-consuming compared to the search for particles. However, the previous results were for finding a number of particles near a single particle in the center of the domain. In the MTPT case, each particle must find its neighbors, which for the worst case is an $N_p \times N_p$ problem. Here we gauge the relative time requirements for searching for nearest neighbors for every particle in a square domain, and compare that to the tree creation time. We choose two search radii to assess the effect on search times and bucket sizes: $r = 0.5$ and $r = 5$ in a $[0, 10] \times [0, 10]$ domain with either $N_p = 10^5$ or 10^6 particles. Simulations were run on a Macbook air with M2 chip and ample memory. For all values of N_p and r, a bucket size of approximately $N_p/1000$ performs best, and searching most of the domain (i.e., $r = 5$) for all 10^6 particles is costly in terms of CPU time (Fig. 6). Also, the time to create the trees is vastly shorter than the searches themselves.

2.3.2. Geometric decomposition with full matrices

One advantage of using a full matrix (i.e., using Eq. (1) for all particles in every subdomain) is the algorithmic simplicity. There is no need to determine which subsets of particles should be included in the MT matrix before making costly calculations such as distance and total probability. For sub-domains away from the domain edges, the ghost-particle zones will be on all sides of the sub-domain, so the number of rows in the mass-transfer matrix calculation is $\frac{N_P}{S_x S_y S_z} = \frac{N_P}{S}$. Each row has a number of entries (counting ghost particles in both directions) on the order of

$$N = N_P \left[\frac{(\Omega_x/S_x + 6\sqrt{4D\Delta t})(\Omega_y/S_y + 6\sqrt{4D\Delta t})(\Omega_z/S_z + 6\sqrt{4D\Delta t})}{\Omega_x \Omega_y \Omega_z} \right]. \tag{5}$$

If the number of flops for a full matrix is proportional to the product of rows and columns, then the flops for a sub-domain is

$$\text{flops}_S = k \frac{N_P^2}{S\Omega} \left[(\Omega_x/S_x + 6\sqrt{4D\Delta t})(\Omega_y/S_y + 6\sqrt{4D\Delta t})(\Omega_z/S_z + 6\sqrt{4D\Delta t}) \right]. \tag{6}$$

So as $D\Delta t \to 0$, the approximation used by Engdahl et al. (2019), $N_S = N_P/S$, is valid and the total number of flops for the S sub-domains is $Sk(N_P/S)^2 = kN_P^2/S$. In this case, the ratio of flops for the subdivided domain to the undivided one is approximately $1/S$. This naive calculation would encourage a user to use as many sub-domains as possible, even with one processing thread, because the total number of flops decreases with the number of sub-domains S. Furthermore, another naive assumption, that each sub-domain could be sent simultaneously to a different core (where there are a number C of available processing cores), decreases the relative computation time T_R like $T_R = 1/CS = 1/C^2$. In a perfect world, the speedup by parallelization of the mass transfer algorithm scales like C^2 — dramatically faster than the "optimal" linear speedup of most algorithms with number of processors. However, one simple factor limits the speedup of the algorithm. When the $12\sqrt{D\Delta t}$ ghost-particle term becomes large relative to the sub-domain size (as must happen when S gets very large), the constant penalty in (6) becomes large.

It is worth examining Eq. (6) for some simple examples to see overall behavior. First, let the domain be cubic with cubic sub-domains, so $\Omega_x = \Omega_y = \Omega_z$ and $S_x = S_y = S_z$. Furthermore, assume that the contribution of over-counted ghosts from the sub-domains on the edges is reasonably small compared to the interior. Then in d-dimensions (and

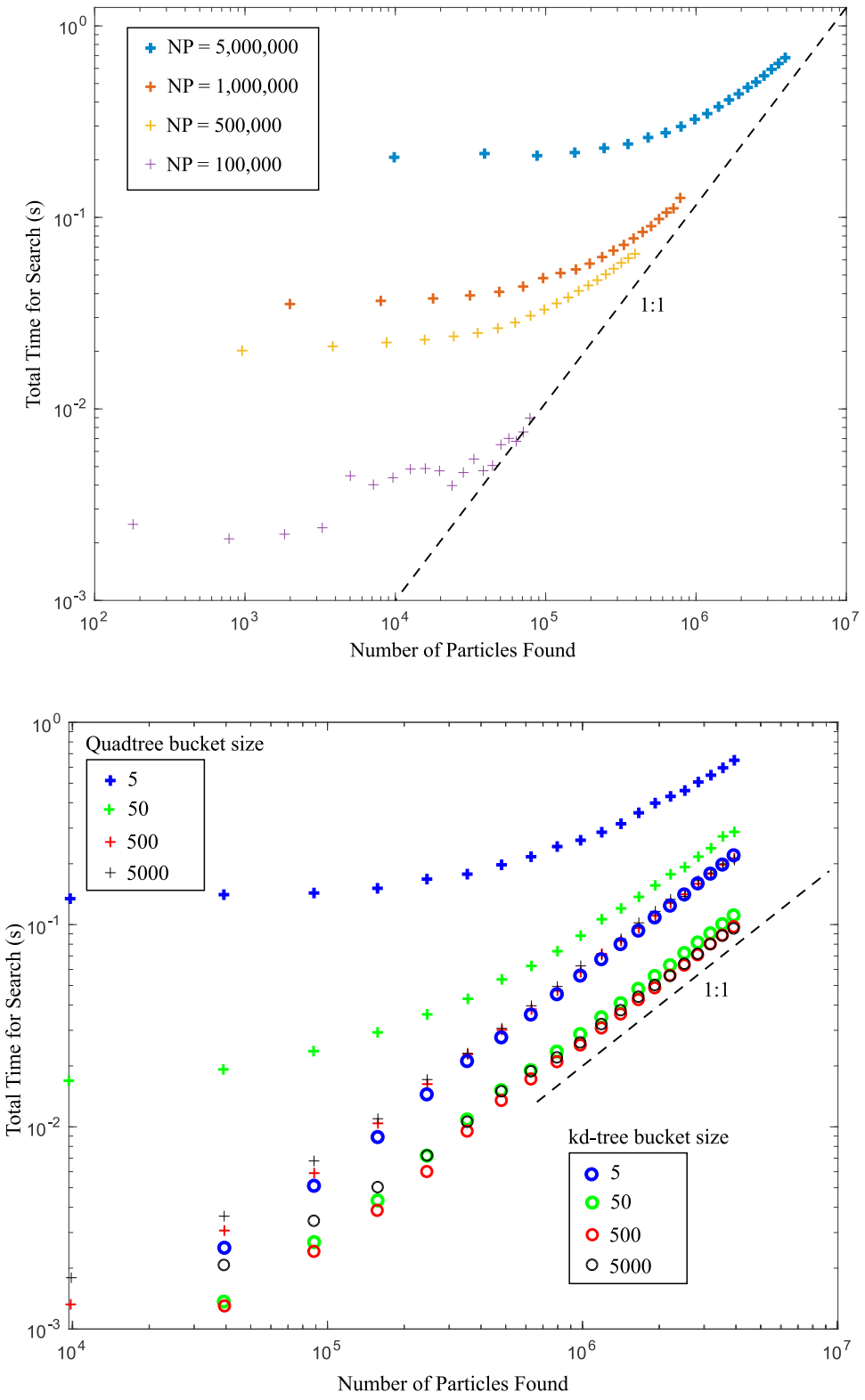


Fig. 5. Top: Computation time for quad-tree searches of a range of radii with the tree bucket size set to four particles. Bottom: Total search times for the $N_P = 5 \times 10^6$ runs with a range of bucket sizes. Plus signs (+) are for quad-tree; circles for kd-tree.

neglecting a few smaller terms), for each sub-domain the number of entries in **P** is the square of the particle density times the volume of the cube:

$$\text{flops}_S \approx k \left[\frac{N_P}{\Omega} \left(\frac{\Omega_x}{S_x} + 6\sqrt{4D\Delta t} \right)^d \right]^2 = k N_P^2 \left(\frac{1}{S_x} + \frac{6\sqrt{4D\Delta t}}{\Omega_x} \right)^{2d} . \quad (7)$$

Now multiplying by the total number of subdomains $S = S_x^d$, and assuming that there is enough memory to store and calculate subdomains on C cores simultaneously (with $C \leq S$), we may calculate the approximate amount of clock time spent in calculations relative to an

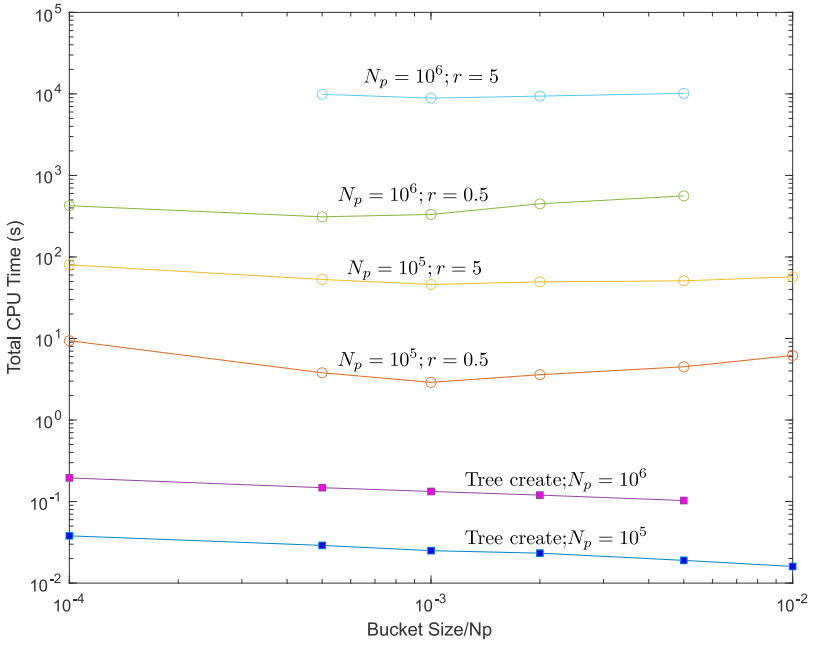


Fig. 6. Total CPU times to search for neighbors within radius r of every one of N_p particles (circles) and quad-tree creation (filled squares). Various bucket sizes are used, and a bucket size approximately $10^{-3} \times N_p$ gives the lowest search times.

undivided domain on a single core (kN_P^2) via

$$T_R \approx \frac{\frac{kN_P^2 S}{C}\left(\frac{1}{S_x} + \frac{6\sqrt{4D\Delta t}}{\Omega_x}\right)^{2d}}{kN_P^2} \tag{8}$$

$$= \frac{1}{C}\left[\frac{1}{S_x^{1/2}} + \frac{S_x^{1/2} 6\sqrt{4D\Delta t}}{\Omega_x}\right]^{2d}. \tag{9}$$

This simplified equation shows several features that are common to all geometric domain decomposition. First, for a small number of subdomains, the first number of the brackets dominates, and substantial speed gains are had with increasing subdivisions. At some point, however, the first term is smaller than the second and further subdivisions only provide more speed when there are additional cores to reduce the factor $1/C$ in front of the brackets. Second, there is both a blessing and a curse of dimensionality: while more dimensions require many, many more particles for a given density, the leading term in the brackets S_x^{-d} means a much greater increase in speed when the domain is subdivided. This is true even when computed in serial on one thread on one core, i.e., when $C = 1$ (Fig. 7, top). The speedup is substantially amplified if each sub-domain's computation can be made simultaneously (with no memory transfer cost); then $C = S_x^d$ and the theoretical relative speedup is greater than linear when the first term in (8) dominates (Fig. 7, bottom).

Note also that for this example shown in Fig. 1 ($D = 10^{-3}$, $\Omega_x = 10$, $\Delta t = 10$), in which the typical diffusion distance per time-step $\sqrt{2D\Delta t}$ is about 1.4% (1/70th) of the domain length of $\Omega_x = 10$, the optimal total number of sub-domains for serial execution, using Eq. (8) and $C = 1$, is about 8 in each dimension (i.e., 64 total in 2-d, 512 in 3-d). Actual simulations in 1-, 2-, and 3-d (filled symbols in the top plot of Fig. 7) show good agreement in 1-d, and fewer optimal subdivisions in 2- and 3-d. We conjecture that the main culprit for the mismatch is the assumption that each subdivision resembles an interior region with no edges: this assumption is worse as the domain surface area to volume ratio increases with dimension. However, using the simple Eq. (8) will get the user close enough that some minor experimentation will easily find the optimal vale of S_x. On the other hand, if sub-domains can be computed simultaneously, there is no absolute maximum number of sub-domains indicated, and the theoretical speedups are substantial in 3-d (Fig. 7 bottom). In 1-d, on the other hand, the speedup levels off after a few tens of sub-domains, even for perfectly parallel computations. The reason for this difference is that the ratio of subdomain volume to ghost-particle volume decreases much more rapidly in 1-d as the domain is subdivided. For these full-matrix computations in 2-d and 3-d, a fairly large number of subdomains is required for the optimal conditions, which realistically can only be achieved on distributed-memory, multi-processor machines, and then the transfer of data becomes a major consideration on relative speedup (e.g., Schauer et al., 2023).

2.3.3. Geometric decomposition with sparse matrices

In addition to saving memory relative to the full matrices, the sparse matrices also reduce the number of calculations and therefore speed up overall simulations. A question remains regarding the relative speedup. The number of entries in the mass-transfer matrix for a single subdomain is the number of rows $N_P/(S_x S_y S_z) = N_P/S$, multiplied by the number of entries for each row $\frac{N_P}{\Omega}\pi\kappa(D\Delta t)^{d/2}$. The number of flops in a subdomain is therefore

$$\text{flops}_S \approx k\frac{N_P^2}{S\Omega}\left[\pi\kappa(36D\Delta t)^{d/2}\right]. \tag{10}$$

To estimate the parallel computation time, we merely multiply by the number of subdomains S and divide by the number of cores C, with the constraint that $C \le S$. Then the computation time follows

$$T \approx k\frac{N_P^2}{C\Omega}\left[\pi\kappa(36D\Delta t)^{d/2}\right]. \tag{11}$$

The term in the brackets may be considered a constant, so that the relative computation time for the entire domain among C cores is independent of S and scales like

$$T_R = \frac{T(C=1)}{T(C)} = \frac{1}{C}. \tag{12}$$

Importantly, this estimate of speed suggests that there is little to no penalty for using more subdivisions. Previously, the total memory requirement was found to be proportional to N_P^2/S ((3)), so that even simulations with a large number of particles should be able to run if S is made large enough.

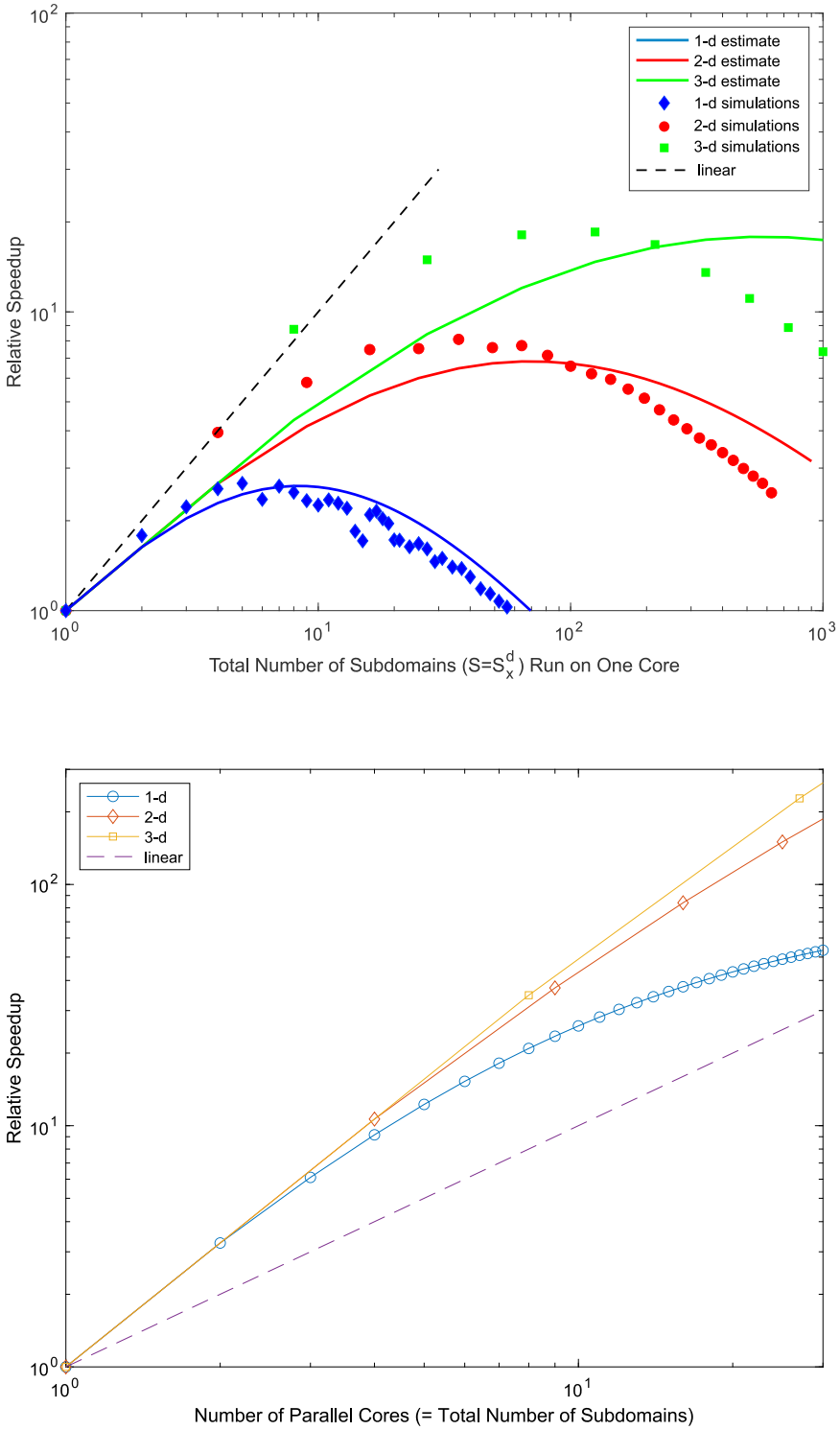


Fig. 7. Theoretical full-matrix speedups: top) Approximate speedup on one core, relative to an undivided domain, versus total number of subdomains S for 1-, 2- and 3-d systems with size $\Omega_x = 10$, $D = 10^{-3}$, and $\Delta t = 10$. Also plotted (solid circles) are speeds from a series of 2-d simulations with the same parameters and using $N_P = 10{,}000$ particles. bottom) Log–log plot of approximate relative speedup for the same system assuming a separate processing core for each sub-domain.

We verified this somewhat counter-intuitive result — that the overall speed is independent of S, by specifying one thread and varying $S_x = 1, 2, \ldots, 10$, i.e., S varies from 1 to 100 in a square domain $\Omega = [0, 10] \times [0, 10]$, with $D = 10^{-3}, \Delta t = 10$, over 10 timesteps using $N_P = 100{,}000$ particles. The clock times actually increase, although not drastically, as S increases (Fig. 8a). This slowdown can be attributed to moving each subdomain's data in and out of the CPU's cache memory. More subdomains mean more duplicate ghost particle information being swapped in and out.

We then picked the simulation with $S = 16$ to verify that speedup is fairly linear with the number of computational threads (Eq. (12)) by varying the thread numbers from one to eight while holding all else constant (Fig. 8b). The simulations were run on a MacBook Air with an M2 processor containing 4 performance and 4 efficiency cores. The

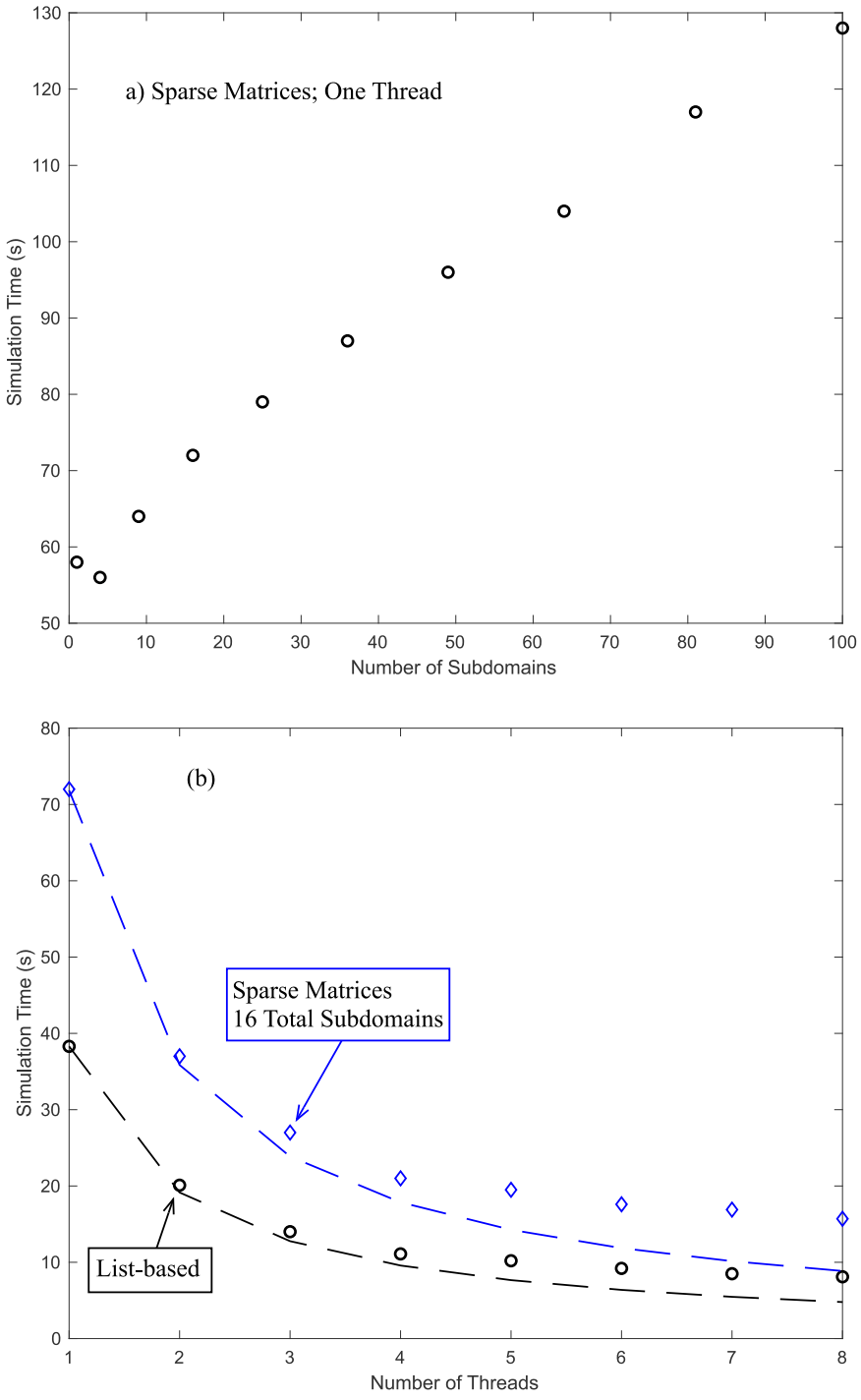


Fig. 8. (a) Sparse-matrix, row-and-column normalization algorithm clock times versus total number of subdomains in 2-d. (b) Clock times versus threads for a fixed 16-subdomain sparse-matrix simulation (blue diamonds) and for a list-based decomposition (black circles). Dashed lines signify speedup that is linear with number of threads. All simulations run on Mac M2 processor with 4 performance and 4 efficiency cores.

speedup relative to using a single core $T_R(C=1)/T_R(C)$ is close to the theoretical linear model up to 4 threads, with a dropoff in speed from 5 to 8.

2.3.4. List-based (non-geometric) decomposition

The number of rows in the matrix to be computed remains N_P, and the number of entries in each row is the particle density times the volume of the d-dimensional sphere around a typical point, $\frac{N_P}{\Omega}\pi\kappa(36D\Delta t)^{d/2}$, so the total flops is

$$\text{flops} \approx k\frac{N_P^2}{\Omega}\left[\pi\kappa(36D\Delta t)^{d/2}\right]. \tag{13}$$

with the caveat that the computation constant k will be smaller because, all things equal, the list-based method only calculates the

row-normalization. For C cores in parallel, the computation time follows

$$T \approx k\frac{N_P^2}{C\Omega}\left[\pi\kappa(36D\Delta t)^{d/2}\right]. \tag{14}$$

which is identical to the time for sparse matrices Eq. (11) except for the smaller value of k. This also shows that the algorithm should have linear speedup with the number of cores using Eq. (12).

We ran the new, list-based method for identical simulations, first using the same parameters as the previous simulations for the matrix-based solutions, namely a domain $\Omega = [0, 10]\times[0, 10]$, $D = 10^{-3}$, $\Delta t = 10$, and $N_P = 100{,}000$ particles that are uniformly randomly distributed in space. The simulations were parallelized using OpenMP on the loop that traversed the i index of the matrix shown in Fig. 4. Special care must be taken to make sure that overly large arrays are not packed into the CPU cache, so the value of $\boldsymbol{m}_i(t+\Delta t)$ was passed to/from subroutines as a private scalar, not as a member of a larger public array (e.g., Chabbi et al., 2018).

Scaling appears reasonably linear with the number of threads up to the 4 performance cores on a MacBook Air (Fig. 8b). These simulations verify that scaling of the sparse-matrix and list-based methods are similar, with the list-based algorithm performing about 2× faster across all parallelization levels.

The total times of less than 10 s may mean that a significant amount of work was just overhead like input/output, so we re-ran the simulations using 500,000 particles (5 times the original simulations) on five different machines: (1) a two-Xeon, 20-core (40-thread) Ubuntu desktop; (2) a single Xeon, 18-core (36-thread) Mac-Pro running MacOS; (3) an 8-core (16-thread) Xeon iMac running MacOS, (4) a Macbook Air with M2 processor with 4 performance and 4 efficiency cores, and (5) a MacStudio with an M2 Ultra chipset of 16 performance and 8 efficiency cores.

The expected linear speedup is seen across a range of different processors, up to the number of cores available (Fig. 9a). As expected, when running multiple threads (up to a maximum of two) per core, the speedup drops somewhat. Furthermore, the newer M2 chips contain performance and efficiency cores (denoted by (performance + efficiency) in the figure legends), and the linear speedup only applies to the number of performance cores. Adding more cores or multiple threads per core does not slow down a simulation (as it might with distributed work that must synchronize), but speedup is less than linear with the number of threads. In terms of raw speed, the Macbook Air with the M2 chip executed the serial runs fastest, but was eventually overtaken when the number of performance cores (four) was surpassed (Fig. 9b).

3. Conclusions

Extending the mass-transfer particle-tracking technique to 2-d and 3-d is not a trivial matter when the computational details are considered. More dimensions require many more particles to resolve the concentration field and this can easily overwhelm a typical machine's RAM without efficient computational schemes. The literature revealed that the extent of these burdens was, at best, qualitatively understood, so we investigated various numerical techniques here in a more quantitative manner. The main driver of speed and memory use is the number of entries in a mass-transfer matrix **P**, and reducing this will lead to increased speed. Previous methods (Engdahl et al., 2019) used full matrices to track all particle interactions, so we began with those and expanded our consideration to sparse arrays, despite the naive nature of using full versions of sparse matrices in terms of memory requirements. Both methods conserve mass by normalizing array elements using an average of row- and column-sums, but differ only in the number of elements stored and used. We also developed and tested a novel list-based decomposition that only uses the row-sum normalization. The full matrices require on the order of $\frac{\Omega/S}{\pi\kappa(36D\Delta t)^{d/2}}$ times more memory than sparse matrices, which require another $\frac{N_P}{S}$ times more memory than the list-based decomposition. Each of these factors are user-controlled by specifying the number of particles and subdivisions, but for most problems the memory savings for the list-based method will be orders-of-magnitude.

For practical reasons, we only checked the parallel speedup of the sparse-matrix, row-and-column normalization scheme and the list-based (row-only normalization) decomposition scheme. Both showed linear speedup with increased numbers of (performance) cores, with a drop-off of speedup when either threads were run on slower cores or when multiple threads were run on some cores. Overall, given the same number of single threads per core, the new list-based decomposition runs about 2× faster than the geometric, sparse-matrix decomposition. The latter of these also showed speed degradation as the number of spatial subdomains is increased, which was required on the Macbook Air because of memory constraints when moving from 100,000 to 500,000 particles in 16 Gb of RAM.

We further performed a limited investigation of the accuracy of the list-based decomposition and showed that evenly-spaced, stationary particles yield machine-precision levels of accuracy when the constraint on inter-particle spacing $\Delta x < \sqrt{D\Delta t}$ is maintained. As such, the assessment of the proposed algorithm is that it is faster, consumes far less memory, and is just as accurate as the other schemes. Within a deforming plume in a spatially heterogeneous velocity field, the newer algorithm gave very similar results to prior methods that have been extensively benchmarked. However, a more detailed error analysis could be performed using highly heterogeneous variable-velocity fields, to assess whether the new algorithm is similarly robust when local particle spacings may change drastically over time.

CRediT authorship contribution statement

David A. Benson: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Ivan Pribec:** Software, Methodology. **Nicholas B. Engdahl:** Writing – original draft, Methodology, Conceptualization. **Stephen Pankavich:** Funding acquisition, Formal analysis. **Lucas Schauer:** Writing – review & editing, Software.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: David Benson reports financial support was provided by National Science Foundation. Stephen Pankavich reports financial support was provided by National Science Foundation. If there are other authors they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Link to all data provided in manuscript.

Acknowledgments

Funding was provided by the U.S. National Science Foundation under awardsCBET-2129531, EAR-2049687, EAR-2049688, DMS-1911145, and DMS-2107938. Fortran and Matlab codes used to produce all results may be found in the Github repository https://github.com/dbenson5225/parallel_particle_mass_transfer.

Appendix

Following (Benson and Meerschaert, 2008; Benson and Bolster, 2016; Schmidt et al., 2018; Sole-Mari et al., 2019b), the physically-

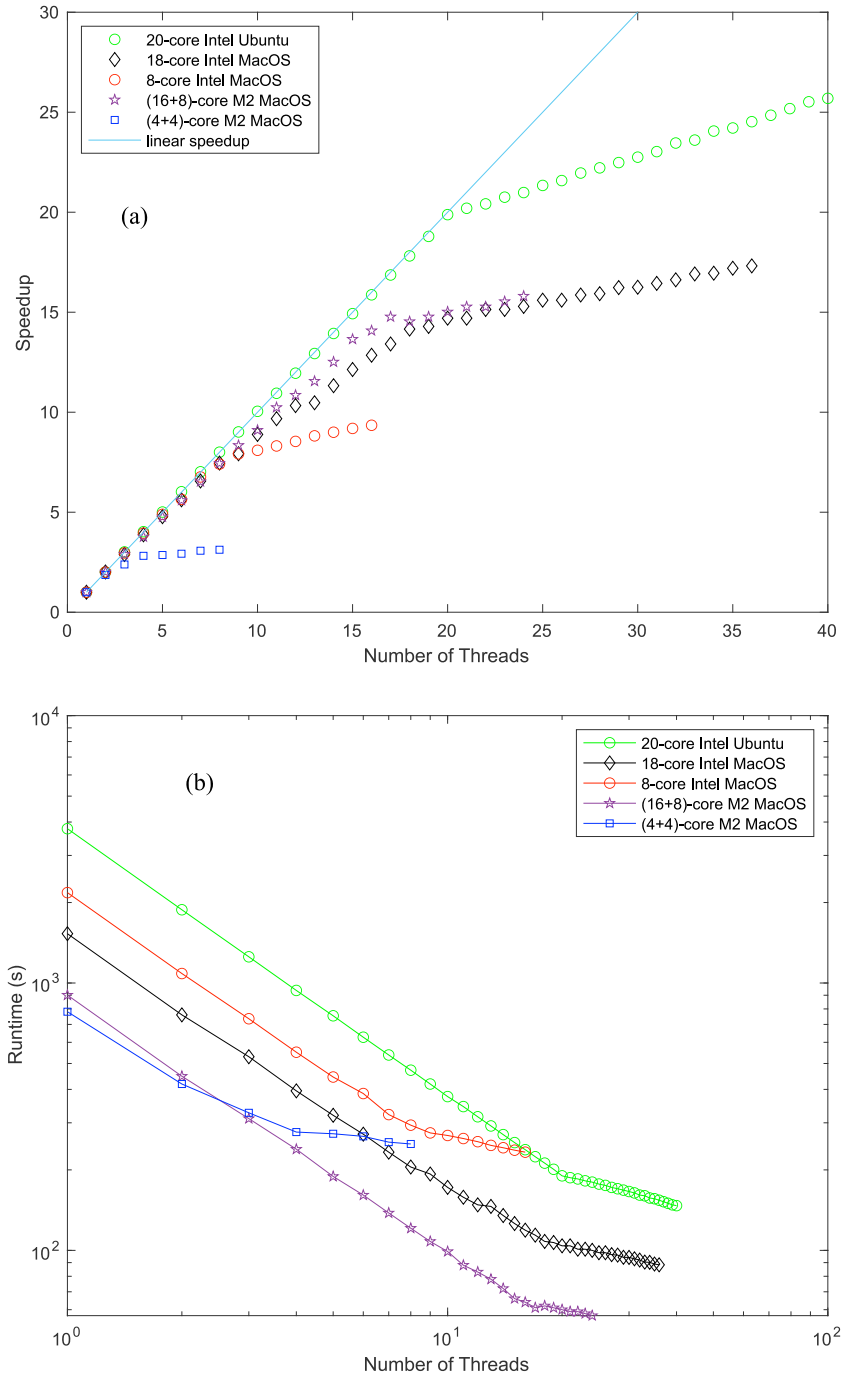


Fig. 9. (a) Parallel scaling of the list-based method on five different machines. (b) Wall-clock speeds for the same runs as in (a). All simulations use 500,000 particles over 10 timesteps. The abrupt changes in slope generally represent exceeding the performance cores on M-series processors, or when the number of threads exceeded the number of cores on Intel architectures.

based probability density function of mass transfer between particles i and j for locally Fickian dispersion in d-dimensions is the convolution of the location densities of the two particles. This represents the PDF of the particles' co-location:

$$\mathbf{p}_{i,j} = \frac{1}{(4\pi\Delta t)^{d/2}\det(\mathbf{D}_i + \mathbf{D}_j)^{1/2}} \exp\left[-\frac{1}{4\Delta t}(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{D}_i + \mathbf{D}_j)^{-1}(\mathbf{x}_i - \mathbf{x}_j)\right], \tag{15}$$

where det() is the determinant, and $\mathbf{D}_i^{-1}$ is the inverse of the dispersion matrix $\mathbf{D}_i$. To simplify notation and without loss of generality we will consider isotropic and constant-magnitude mixing dispersion so that $\mathbf{D}_i = D\mathbb{I}$. Then, Eq. (15) reduces to

$$\mathbf{p}_{i,j} = \frac{1}{(8\pi D\Delta t)^{d/2}} \exp\left[-\frac{1}{8D\Delta t}(|\mathbf{x}_i - \mathbf{x}_j|)^2\right]. \tag{16}$$

A key to using this PDF is that the total probability of particle co-location dictates the mass transfer, so that the matrix describing mass transfer has elements that approximate

$$\mathbf{P}_{i,j} \approx \Delta\mathbf{x}\mathbf{p}_{i,j} = \frac{\Delta\mathbf{x}}{(8\pi D\Delta t)^{d/2}} \exp\left[-\frac{1}{8D\Delta t}(|\mathbf{x}_i - \mathbf{x}_j|)^2\right], \tag{17}$$

where $\Delta\mathbf{x}$ is a particle support volume designed so that the d-dimensional sum of Eq. (17) for any i over the index j is unity, i.e.,

the values of $\mathbf{P}$ represent probabilities. In the original formulation (Benson and Bolster, 2016) for a finite number of particles, the mass transfer for the ith particle can be written to first order as

$$\boldsymbol{m}_i(t+\Delta t) = \boldsymbol{m}_i(t) + \sum_{j=1}^{N} \frac{1}{2}\mathbf{P}_{i,j}(\boldsymbol{m}_j(t) - \boldsymbol{m}_i(t)). \tag{18}$$

However, because Eq. (16) is a density function, Sole-Mari et al. (2019b) recognized that Eq. (18) has the form of smoothed-particle-hydrodynamics, and the PDF can be generalized to have a variable bandwidth dictated by parameter β, yielding

$$\mathbf{p}_{i,j} = \frac{1}{(4\pi\beta^{-1}D\Delta t)^{d/2}} \exp\left[-\frac{1}{4\beta^{-1}D\Delta t}(|\mathbf{x}_i - \mathbf{x}_j|)^2\right], \tag{19}$$

with a corresponding mass transfer equation of

$$\boldsymbol{m}_i(t+\Delta t) = \boldsymbol{m}_i(t) + \sum_{j=1}^{N} \beta\mathbf{P}_{i,j}(\boldsymbol{m}_j(t) - \boldsymbol{m}_i(t)), \tag{20}$$

where the original MT algorithm (Benson and Bolster, 2016) uses $\beta = 1/2$. Sole-Mari et al. (2019b) found that a value of $\beta = 1$ demonstrates greater accuracy for fixed particle positions. Later in this appendix we show why this is the case. First, though, we consider two important aspects of constructing the probability (MT weighting) matrix $\mathbf{P}$: (i) mass conservation and (ii) the normalization for a total probability of unity in Eq. (17).

For expression (20) to conserve mass, it is necessary that $\mathbf{P}_{i,j}$ is a symmetric matrix, i.e. $\mathbf{P}_{i,j} = \mathbf{P}_{j,i}$. To show this, denote the total mass at time $t \geq 0$ by

$$\mathcal{M}(t) = \sum_{i=1}^{N} \boldsymbol{m}_i(t). \tag{21}$$

We wish to impose the condition

$$\mathcal{M}(t) = \mathcal{M}(0) \tag{22}$$

for all $t \geq 0$. Summing Eq. (20) over all particles $i = 1, \ldots, N$ gives

$$\mathcal{M}(t+\Delta t) = \mathcal{M}(t) + \beta \sum_{i,j=1}^{N} \mathbf{P}_{i,j}\left(\boldsymbol{m}_j(t) - \boldsymbol{m}_i(t)\right). \tag{23}$$

Hence, the property of mass conservation between consecutive time steps is equivalent to the condition

$$\sum_{i,j=1}^{N} \mathbf{P}_{i,j}\left(\boldsymbol{m}_j(t) - \boldsymbol{m}_i(t)\right) = 0 \tag{24}$$

or

$$\sum_{i,j=1}^{N} \left(\mathbf{P}_{i,j}\boldsymbol{m}_j(t) - \mathbf{P}_{i,j}\boldsymbol{m}_i(t)\right) = 0. \tag{25}$$

By re-indexing the second sum and switching j and i, this can be rewritten as

$$\sum_{i,j=1}^{N} \left(\mathbf{P}_{i,j} - \mathbf{P}_{j,i}\right)\boldsymbol{m}_j(t) = 0. \tag{26}$$

This condition is satisfied by having each $\mathbf{P}_{i,j} = \mathbf{P}_{j,i}$, i.e. if $\mathbf{P} = \mathbf{P}^T$.

A second constraint on the $\mathbf{P}$ matrix can be visualized by imposing a Dirac-delta function initial condition. In this case, the vector $\mathbf{m}(t=0)$ contains only one non-zero value in the ith entry. Assume this value is unity. When this is operated on by the matrix $\mathbf{P}$ (see Fig. 4), then the masses in the vector $\boldsymbol{m}(\Delta t)$ is the sum of the probabilities in the column $j = i$. These probabilities must sum to unity to conserve mass. Because the delta function could be placed in any row, then every column must sum to unity. And because the matrix is symmetric, that means both row- and column-sums must equal unity. Schmidt et al. (2017) and Schauer et al. (2023) construct $\mathbf{P}$ from $\mathbf{p}$ by computing the average of the row sums and column sums of $\mathbf{p}$ and normalizing every value by these averages, i.e.,

$$\mathbf{P}_{RC} = \frac{\mathbf{p}}{\frac{1}{2}(\sum_{i=1}^{N}\mathbf{p}_{i,j} + \sum_{j=1}^{N}\mathbf{p}_{i,j})}, \tag{27}$$

where the subscript RC denotes Row-Column-sum normalization. This operation maintains the symmetry of $\mathbf{P}$ because, for any entry $\mathbf{p}_{i,j}$, its row-sum is equal to the column-sum for entry $\mathbf{p}_{j,i}$ (see Fig. 4). This has the added desirable effect of eliminating an estimation of the particle support volume at different locations in the domain: that calculation is included in the normalization. However, because each value in a row is normalized by a different value (i.e., each entry has a different column and therefore different column-sum), this form of normalization does not guarantee that any row-sum is equal to unity. Therefore, either (1) the process is repeated until closure is reached (this is called the Sinkhorn–Knopp algorithm (Knight, 2008; Young, 2024)) or (2) a final adjustment takes any deviations of the row-sums from unity and adds them to the main diagonal (Schmidt et al., 2018).

A new and novel approach to satisfy the requirements on $\mathbf{P}$ (symmetric with row and column sums equal to unity) is to use the average particle spacing for $\Delta\mathbf{x}$, then calculate the row-sums and take any deviations from unity and add them to the main diagonal term:

$$\mathbf{P} = \Delta\mathbf{x}\mathbf{p} - \mathrm{diag}\left(\sum_{j=1}^{N} \Delta\mathbf{x}\mathbf{p}_{i,j}\right). \tag{28}$$

This method only requires a calculation of the row-sum, so that all values in a row may be calculated independently of all others, and the forward matrix mass-transfer calculation is also performed without regard for other rows. This method leads to the list-based decomposition, because any groups of rows may be sent to any processor, without having to take into account the spatial proximity of the particles corresponding to the rows.

A closer look at Eq. (20) gives some insight into the accuracy given by different choices of the bandwidth parameter β. Recall that the variable bandwidth probability density $\mathbf{p}$ in Eq. (19) is used to construct $\mathbf{P}$. With this, the mass-transfer algorithm (Eq. (20)) can be expressed as

$$\boldsymbol{m}_i(t+\Delta t) = (1-\beta)\boldsymbol{m}_i(t) + \beta\sum_{j=1}^{N} \mathbf{P}_{i,j}\boldsymbol{m}_j(t), \tag{29}$$

i.e., the updated mass is a weighted average of the *previous* time-step's mass and the convolution of the diffusion Green's function for $1/\beta$ time-steps (see Eq. (19)). For example, the choice of $\beta = 1/2$ is an equal weighing of the prior mass and the Green's function for the future time $2\Delta t$ (see Eq. (19)). If the mass at every point changed linearly over time, then any choice of β between zero and one would suffice. But masses do not change linearly in time, and clearly as $\beta \to 1$, the formulas converge to the "correct" answer: that the progress of the diffusion-type equation is a convolution of the current condition with the Green's function discretized over a single time-step. See Sole-Mari et al. (2019b) for a numerical demonstration. With the realization that $\beta = 1$ is the most accurate algorithm, Eq. (29) is a simple convolution for $\beta = 1$, namely

$$\mathbf{m}(t+\Delta t) = \mathbf{P}\mathbf{m}(t), \tag{30}$$

where $\boldsymbol{m}$ is a vector of particle masses, and $\mathbf{P}$ is the matrix of transfer probabilities representing the Green's function of local mixing. This is the form of the MT algorithm we use in the text.

References

Benson, D.A., Aquino, T., Bolster, D., Engdahl, N., Henri, C.V., Fernàndez-Garcia, D., 2017. A comparison of Eulerian and Lagrangian transport and non-linear reaction algorithms. Adv. Water Resour. 99, 15–37. http://dx.doi.org/10.1016/j.advwatres.2016.11.003.

Benson, D.A., Bolster, D., 2016. Arbitrarily complex chemical reactions on particles. Water Resour. Res. 52 (11), 9190–9200. http://dx.doi.org/10.1002/2016WR019368.

Benson, D.A., Meerschaert, M.M., 2008. Simulation of chemical reaction via particle tracking: Diffusion-limited versus thermodynamic rate-limited regimes. Water Resour. Res. 44, W12201. http://dx.doi.org/10.1029/2008WR007111.

Benson, D.A., Pankavich, S., Bolster, D., 2019. On the separate treatment of mixing and spreading by the reactive-particle-tracking algorithm: An example of accurate upscaling of reactive poiseuille flow. Adv. Water Resour. 123, 40–53. http://dx.doi.org/10.1016/j.advwatres.2018.11.001, URL: http://www.sciencedirect.com/science/article/pii/S0309170818304354.

Benson, D.A., Pankavich, S., Schmidt, M.J., Sole-Mari, G., 2020. Entropy: (1) the former trouble with particle-tracking simulation, and (2) a measure of computational information penalty. Adv. Water Resour. 137, 103509. http://dx.doi.org/10.1016/j.advwatres.2020.103509, URL: https://www.sciencedirect.com/science/article/pii/S0309170819303458.

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 509–517.

Berger, M.J., Bokhari, S.H., 1987. A partitioning strategy for nonuniform problems on multiprocessors. IEEE Trans. Comput. C-36 (5), 570–580.

Bolster, D., Paster, A., Benson, D.A., 2016. A particle number conserving Lagrangian method for mixing-driven reactive transport. Water Resour. Res. 52 (2), 1518–1527. http://dx.doi.org/10.1002/2015WR018310.

Chabbi, M., Wen, S., Liu, X., 2018. Featherlight on-the-fly false-sharing detection. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. PPoPP '18, Association for Computing Machinery, New York, NY, USA, pp. 152–167. http://dx.doi.org/10.1145/3178487.3178499.

Ding, D., Benson, D.A., Fernández-Garcia, D., Henri, C.V., Hyndman, D.W., Phanikumar, M.S., Bolster, D., 2017. Elimination of the reaction rate "scale effect": Application of the Lagrangian reactive particle-tracking method to simulate mixing-limited, field-scale biodegradation at the schoolcraft (MI, USA) site. Water Resour. Res. http://dx.doi.org/10.1002/2017WR021103.

Engdahl, N.B., Benson, D.A., Bolster, D., 2014. Predicting the enhancement of mixing-driven reactions in nonuniform flows using measures of flow topology. Phys. Rev. E 90, 051001. http://dx.doi.org/10.1103/PhysRevE.90.051001, URL: http://link.aps.org/doi/10.1103/PhysRevE.90.051001.

Engdahl, N.B., Schmidt, M.J., Benson, D.A., 2019. Accelerating and parallelizing Lagrangian simulations of mixing-limited reactive transport. Water Resour. Res. 55 (4), 3556–3566. http://dx.doi.org/10.1029/2018WR024361, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR024361.

Finkel, R.A., Bentley, J.L., 1974. Quad trees a data structure for retrieval on composite keys. Acta Inform. 4 (1), 1–9. http://dx.doi.org/10.1007/BF00288933.

Herrera, P.A., Massabó, M., Beckie, R.D., 2009. A meshless method to simulate solute transport in heterogeneous porous media. Adv. Water Resour. 32 (3), 413–429. http://dx.doi.org/10.1016/j.advwatres.2008.12.005, URL: http://www.sciencedirect.com/science/article/pii/S0309170808002273.

Kennel, M.B., 2004. KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. arXiv Physics, URL: https://arxiv.org/abs/physics/0408067v2.

Knight, P.A., 2008. The Sinkhorn–Knopp algorithm: Convergence and applications. SIAM J. Matrix Anal. Appl. 30 (1), 261–275. http://dx.doi.org/10.1137/060659624.

Labolle, E.M., Fogg, G.E., Tompson, A.F.B., 1996. Random-walk simulation of transport in heterogeneous porous media: Local mass-conservation problem and implementation methods. Water Resour. Res. 32 (3), 583–593.

Paster, A., Bolster, D., Benson, D.A., 2014. Connecting the dots: Semi-analytical and random walk numerical solutions of the diffusion–reaction equation with stochastic initial conditions. J. Comput. Phys. 263, 91–112. http://dx.doi.org/10.1016/j.jcp.2014.01.020, URL: http://www.sciencedirect.com/science/article/pii/S0021999114000473.

Perez, L.J., Hidalgo, J.J., Dentz, M., 2019a. Reactive random walk particle tracking and its equivalence with the advection-diffusion-reaction equation. Water Resour. Res. 55 (1), 847–855. http://dx.doi.org/10.1029/2018WR023560, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2018WR023560, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR023560.

Perez, L.J., Hidalgo, J.J., Dentz, M., 2019b. Upscaling of mixing-limited bimolecular chemical reactions in poiseuille flow. Water Resour. Res. 55 (1), 249–269. http://dx.doi.org/10.1029/2018WR022730, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR022730.

Pérez-Illanes, R., Fernàndez-Garcia, D., 2024. MODPATH-RW: A random walk particle tracking code for solute transport in heterogeneous aquifers. Groundwater 62 (4), 617–634. http://dx.doi.org/10.1111/gwat.13390, arXiv:https://ngwa.onlinelibrary.wiley.com/doi/pdf/10.1111/gwat.13390, URL: https://ngwa.onlinelibrary.wiley.com/doi/abs/10.1111/gwat.13390.

Schauer, L., Schmidt, M.J., Engdahl, N.B., Pankavich, S.D., Benson, D.A., Bolster, D., 2023. Parallelized domain decomposition for multi-dimensional Lagrangian random walk mass-transfer particle tracking schemes. Geosci. Model Dev. 16 (3), 833–849. http://dx.doi.org/10.5194/gmd-16-833-2023, URL: https://gmd.copernicus.org/articles/16/833/2023/.

Schmidt, M.J., Pankavich, S., Benson, D.A., 2017. A kernel-based Lagrangian method for imperfectly-mixed chemical reactions. J. Comput. Phys. 336, 288–307. http://dx.doi.org/10.1016/j.jcp.2017.02.012.

Schmidt, M.J., Pankavich, S.D., Benson, D.A., 2018. On the accuracy of simulating mixing by random-walk particle-based mass-transfer algorithms. Adv. Water Resour. http://dx.doi.org/10.1016/j.advwatres.2018.05.003, URL: https://www.sciencedirect.com/science/article/pii/S0309170818301830.

Schmidt, M.J., Pankavich, S.D., Navarre-Sitchler, A., Benson, D.A., 2019. A Lagrangian method for reactive transport with solid/aqueous chemical phase interaction. J. Comput. Phys. X 2, http://dx.doi.org/10.1016/j.jcpx.2019.100021, URL: https://www.sciencedirect.com/science/article/pii/S259005521930037X.

Schmidt, M.J., Pankavich, S.D., Navarre-Sitchler, A., Engdahl, N.B., Bolster, D., Benson, D.A., 2020. Reactive particle-tracking solutions to a benchmark problem on heavy metal cycling in lake sediments. J. Contam. Hydrol. 234, http://dx.doi.org/10.1016/j.jconhyd.2020.103642, URL: https://www.sciencedirect.com/science/article/pii/S0169772219304279?via%3Dihub.

Sole-Mari, G., Fernàndez-Garcia, D., 2018. Lagrangian modeling of reactive transport in heterogeneous porous media with an automatic locally adaptive particle support volume. Water Resour. Res. 54 (10), 8309–8331. http://dx.doi.org/10.1029/2018WR023033, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2018WR023033, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR023033.

Sole-Mari, G., Fernández-Garcia, D., Rodríguez-Escales, P., Sanchez-Vila, X., 2017. A KDE-based random walk method for modeling reactive transport with complex kinetics in porous media. Water Resour. Res. 53 (11), 9019–9039. http://dx.doi.org/10.1002/2017WR021064, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2017WR021064, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017WR021064.

Sole-Mari, G., Fernàndez-Garcia, D., Sanchez-Vila, X., Bolster, D., 2020. Lagrangian modeling of mixing-limited reactive transport in porous media: Multirate interaction by exchange with the mean. Water Resour. Res. 56 (8), http://dx.doi.org/10.1029/2019WR026993, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019WR026993, URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019WR026993, e2019WR026993 10.1029/2019WR026993.

Sole-Mari, G., Schmidt, M.J., Pankavich, S.D., Benson, D.A., 2019a. Numerical equivalence between SPH and probabilistic mass transfer methods for Lagrangian simulation of dispersion. Adv. Water Resour. 126, 108–115. http://dx.doi.org/10.1016/j.advwatres.2019.02.009, URL: http://www.sciencedirect.com/science/article/pii/S0309170818310820.

Sole-Mari, G., Schmidt, M.J., Pankavich, S.D., Benson, D.A., 2019b. Numerical equivalence between SPH and probabilistic mass transfer methods for Lagrangian simulation of dispersion. Adv. Water Resour. 126, 108–115. http://dx.doi.org/10.1016/j.advwatres.2019.02.009, URL: http://www.sciencedirect.com/science/article/pii/S0309170818310820.

Tartakovsky, A.M., de Anna, P., Le Borgne, T., Balter, A., Bolster, D., 2012. Effect of spatial concentration fluctuations on non-linear reactions in diffusion-reaction systems. Water Resour. Res. 48, W02526.

Tompson, A., Dougherty, D., 1992. Particle-grid methods for eacting flows in porous-media with application to Fisher equation. Appl. Math. Model. 16 (7), 374–383.

Tran, N.T., Benson, D.A., Schmidt, M.J., Pankavich, S.D., 2021. A computational information criterion for particle-tracking with sparse or noisy data. Adv. Water Resour. 151, http://dx.doi.org/10.1016/j.advwatres.2021.103893, URL: https://www.sciencedirect.com/science/article/pii/S0309170821000488.

Young, D., 2024. Sinkhorn-Knopp Algorithm for Matrix Normalisation. (https://www.mathworks.com/matlabcentral/fileexchange/52930-sinkhorn-knopp-algorithm-for-matrix-normalisation). Retrieved August 12, 2024, MATLAB Central File Exchange.