Computer Physics Communications 239 (2019) 265–271

Contents lists available at ScienceDirect

Computer Physics Communications

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

PAR^2: Parallel Random Walk Particle Tracking Method for solute transport in porous media ☆

Calogero B. Rizzo [a,*], Aiichiro Nakano [b], Felipe P.J. de Barros [a]

[a] Sonny Astani Department of Civil and Environmental Engineering, University of Southern California, Los Angeles, USA
[b] Department of Computer Science, University of Southern California, Los Angeles, USA

ARTICLE INFO

Article history:
Received 8 June 2018
Received in revised form 9 November 2018
Accepted 16 January 2019
Available online 29 January 2019

Keywords:
Groundwater
Flows through porous media
Random Walk Particle Tracking
CUDA
Solute transport

ABSTRACT

Computational modeling of solute migration in groundwater systems is a fundamental component in water resources management and risk analysis. Therefore, it is imperative to have fast and reliable computational tools to simulate solute transport in groundwater systems. In this work we present PAR^2, a GPU-accelerated solute transport simulator based on the Random Walk Particle Tracking (RWPT) technique, a Lagrangian method particularly suited for parallelization. PAR^2 is able to run on any computing platform equipped with an NVIDIA GPU, such as common desktops and High-Performance Computing (HPC) nodes. The program is developed in C++/CUDA. In our illustration, groundwater flow is simulated on a structured grid using MODFLOW, which can be linked to PAR^2 using the LMT package. Simulation parameters can be defined through a convenient YAML configuration file. Additionally, we propose an analytical treatment of the dispersion tensor that allows the RWPT to be effectively implemented using GPU parallelization. The speedup gained with the parallelization drastically reduces the total simulation time, allowing the application of computationally expensive algorithms (e.g., Monte-Carlo simulation) on large-scale stochastic hydro-systems.

Program summary

Program Title: PAR^2.
Program Files doi: http://dx.doi.org/10.17632/4pkhgx8wcb.1
Licensing provisions: GPLv3
Programming language: C++/CUDA
Nature of problem: GPU-accelerated simulation of solute transport in saturated porous media.
Solution method: Implementation of the particle tracking method, a fully Lagrangian approach to solve the advection–dispersion equations.

1. Introduction

Groundwater flows are plagued by spatio-temporal fluctuations at multiple scales. Subsurface properties such as the hydraulic conductivity field and the porosity vary many orders of magnitude which leads to variability in the flow field [1]. These velocity fluctuations have a strong impact on the overall dispersive behavior of a solute plume. Together with local scale dispersive mass transfer mechanisms, the spreading dynamics induced by the variability of the groundwater flow field steepens concentration gradients which in turn augments solute mixing [2]. The interplay between flow variability and mass transfer mechanisms is crucial to understand contaminant exposure as well as bio-chemical activities, both critical to risk analysis and aquifer remediation. Furthermore, due to high costs of data acquisition, groundwater models are subject to uncertainty. Estimating the uncertainty due to data scarcity in solute transport predications can be achieved by adopting probabilistic tools in combination with geostatistical techniques [3]. Within the stochastic framework, several flow and transport simulations on equiprobable realizations, i.e. Monte Carlo simulations, of the subsurface environment are carried out (e.g., [4]). As an outcome of the above mentioned factors, capturing the spatio-temporal dynamics of a solute plume and the uncertainty associated with model predictions requires intensive computational effort [5,6].

Many softwares and numerical methods have been developed in order to simulate solute transport in groundwater systems. Numerical methods are usually classified as Eulerian, Lagrangian, or mixed Eulerian–Lagrangian, based on the type of discretization employed [7]. Within the groundwater community, the most comprehensive software for solute transport is MT3DMS [8,9].

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: calogerr@usc.edu (C.B. Rizzo).

https://doi.org/10.1016/j.cpc.2019.01.013
0010-4655/

MT3DMS provides several numerical methods to solve solute transport (reactive and non-reactive). For example, it is possible to use the standard finite-difference method, mixed Eulerian–Lagrangian approach, and the total-variation-diminishing (TVD) method. At the same time, a fully Lagrangian solver using the Random Walk Particle Tracking (RWPT) method is provided by the RW3D code developed by Fernàndez-Garcia et al. [10,11] (for recent developments, see [12]) and RWHet developed by LaBolle [13]. Comparison between Eulerian and Lagrangian approaches can be found in Salamon et al. [14] and Boso et al. [15]. Eulerian methods can introduce a non-physical numerical diffusion to the system as a result of the discretization scheme. Therefore, the numerical grid must be refined where high concentration gradients are expected. This process can be cumbersome in cases where the dimension of the solute plume is small compared to the cell size (e.g., instantaneous point or line injection), when the transport is highly advective (i.e., high Péclet number) or when the system is highly heterogeneous. On the contrary, Lagrangian approaches do not suffer from numerical diffusion. For this reason they are more convenient when advection is the key transport mechanism or when the subsurface environment displays strong heterogeneity in its hydraulic properties. For such reasons, the RWPT method is widely used to model solute transport in heterogeneous groundwater systems [16].

One of the key drawbacks of particle tracking techniques is that it may require a very high number of particles to accurately represent the solute plume and avoid artificial concentration fluctuations [12,14,15,17–19]. This may lead to a prohibitive computational time. However, the high number of particles problem can be remarkably attenuated by parallelizing the RWPT method using modern parallel computing techniques. In fact, each particle trajectory can be computed independently and, for such reasons, the RWPT can be efficiently parallelized.

To verify the benefits due to the parallelization, we developed PAR^2, an open-source software implementing a parallelized version of the RWPT method. Using the General-Purpose Computing on Graphics Processing Units (GPGPU) methodology, we develop an efficient GPU parallelized particle tracking code able to run on NVIDIA GPUs. The code is written using C++/CUDA programming language and it can be downloaded from the CPC library entry or Git repository [20]. The results show the possibility to simulate solute transport on a common desktop setup using a large number of particles in a relatively short time. Moreover, given the increasing number of High Performance Computing (HPC) clusters equipped with NVIDIA GPUs, it is possible to include the software in a massively parallel Monte-Carlo framework, where each GPU simulates one realization. In addition, we propose an analytical methodology to compute the anisotropic displacement matrix needed for the GPU parallelization.

The paper is structured as follows. The theoretical background regarding the physics of groundwater flows and RWPT methods are provided in Section 2. Section 3 describes the approach used for the GPGPU parallelization along with the implementation details. An illustrative example is shown in Section 4 as well as a comparison with another existing software. Finally, concluding remarks are provided in Section 5.

2. Background

2.1. Physics of flow and transport

In this work, we consider a steady-state flow through a saturated 3D porous formation. The Cartesian coordinate system is denoted by $\mathbf{x} = (x, y, z)$. The hydraulic conductivity field is spatially heterogeneous and denoted by $\mathbf{K} \equiv \mathbf{K}(\mathbf{x})$. The flow field is governed by the continuity equation:

$$\nabla \cdot \left[\frac{\mathbf{K}(\mathbf{x})}{\Theta(\mathbf{x})} \nabla \phi(\mathbf{x}) \right] = 0, \tag{1}$$

where ϕ is the hydraulic head and Θ is the effective porosity.

With the solution for ϕ, the spatially variable velocity field $\mathbf{u}$ is obtained through the use of Darcy's law:

$$\mathbf{u}(\mathbf{x}) = -\frac{\mathbf{K}(\mathbf{x})}{\Theta(\mathbf{x})} \nabla \phi(\mathbf{x}), \tag{2}$$

Given the elliptic nature of the flow equations, Eulerian numerical methods, such as Finite Difference (FD) or Finite Volume (FV), can be effectively used to compute the spatially heterogeneous velocity field $\mathbf{u}$. MODFLOW [21] provides several numerical methods for the solution of the flow equations. An alternative solution is provided by PFLOTRAN [22], a parallel solver to simulate subsurface flows.

Next, we consider transport of an inert solute to be governed by advection and local-scale dispersion mechanisms. The concentration of the solute is represented by c and satisfies the Advection–Dispersion equation [23]:

$$\frac{\partial \left(\Theta(\mathbf{x}) c(\mathbf{x}, t)\right)}{\partial t} + \mathbf{u}(\mathbf{x}) \cdot \nabla \left(\Theta(\mathbf{x}) c(\mathbf{x}, t)\right) = \nabla \cdot \left[\Theta(\mathbf{x}) \mathbf{D}(\mathbf{x}) \nabla c(\mathbf{x}, t)\right], \tag{3}$$

with given boundary and initial conditions. The local dispersion tensor is denoted by $\mathbf{D}$ and models both the sub-Darcy scale heterogeneity and molecular diffusion. A possible choice for the dispersion tensor is (see Chapter 7 of Bear and Cheng [7]):

$$\mathbf{D}(\mathbf{x}) = \left(\alpha_T |\mathbf{u}(\mathbf{x})| + D_m\right) \mathbf{I} + \frac{\alpha_L - \alpha_T}{|\mathbf{u}(\mathbf{x})|} \mathbf{u}(\mathbf{x}) \mathbf{u}(\mathbf{x})^T, \tag{4}$$

where α_L and α_T are the longitudinal and transverse dispersivities, D_m is the molecular diffusivity and $\mathbf{I}$ is the identity matrix. Other possible choices of the dispersion tensor can be found in the literature [24,25]. Note that the implementation method we propose can be adapted to different local-scale dispersion models.

2.2. Random walk particle tracking (RWPT)

The RWPT method is a Lagrangian framework that aims to find a solution of (3). The solute plume is represented by N_p particles, each one carrying a fraction of the solute mass $M_p = M_0/N_p$, where M_0 is the mass of solute injected. The particles are moved by three physical mechanisms: advection, diffusion and local-scale dispersion. Advection is simulated by moving particles according to the velocity field obtained through the solution of Eqs. (1) and (2). In order to simulate the diffusion and local-scale dispersion, a random displacement is added to the particle. Molecular diffusion does not depend on the current velocity direction, thus the displacement is radially symmetric (i.e., isotropic). On the other hand, local-scale dispersion is anisotropic and depends on the direction of the particle velocity.

The trajectory of a particle i can be computed using the Itô–Taylor integration scheme [26]:

$$\mathbf{X}_i(t + \Delta t) = \mathbf{X}_i(t) + \mathbf{A}(\mathbf{X}_i(t))\Delta t + \mathbf{B}(\mathbf{X}_i(t)) \cdot \boldsymbol{\xi}(t)\sqrt{\Delta t}, \tag{5}$$

where $\boldsymbol{\xi}$ is a normal-Gaussian white noise vector, Δt is the time step, and the drift vector $\mathbf{A}$ and displacement matrix $\mathbf{B}$ are defined as follows:

$$\mathbf{A}(\mathbf{x}) = \mathbf{u}(\mathbf{x}) + \nabla \cdot \mathbf{D}(\mathbf{x}) + \frac{1}{\Theta(\mathbf{x})} \mathbf{D}(\mathbf{x}) \cdot \nabla \Theta(\mathbf{x}), \tag{6}$$

$$2\mathbf{D}(\mathbf{x}) = \mathbf{B}(\mathbf{x}) \cdot \mathbf{B}(\mathbf{x})^T. \tag{7}$$

Note that the trajectory of each particle is independent from the others. Moreover, the new position of the particle can be found only using the current position, therefore not requiring the knowledge of the full particle history. Since the governing Eq. (3) can be transformed into an equivalent of the Fokker–Planck equation [26], it can be proved that the particles density converges to the solution of (3) when increasing the number of particles and reducing the time step, therefore:

$$c(\mathbf{x}, t) = \frac{1}{\Theta(\mathbf{x})} \lim_{\delta \to 0} \frac{1}{\left|V_{\mathbf{x},\delta}\right|} \sum_{i=1}^{N_p} M_p \mathbb{1}_{V_{\mathbf{x},\delta}} (\mathbf{X}_i(t)) , \tag{8}$$

where $V_{\mathbf{x},\delta}$ is a control volume centered in $\mathbf{x}$ with characteristic dimension δ (e.g., a sphere with center $\mathbf{x}$ and radius δ), $\left|V_{\mathbf{x},\delta}\right|$ is the volume of $V_{\mathbf{x},\delta}$, and $\mathbb{1}_{V_{\mathbf{x},\delta}}(\mathbf{x})$ is the indicator function (i.e, 1 if $\mathbf{x} \in V_{\mathbf{x},\delta}$, 0 otherwise). Eq. (8) can be used to effectively compute an approximation of the concentration $c(\mathbf{x}, t)$ by fixing δ with an appropriate small number. Moreover, the average concentration over an arbitrary control volume can be computed in a similar fashion. Note that the smaller is the control volume, the more particles are needed for the convergence. Several studies focused on improving concentration estimation in situations where the number of particles is low. These techniques allow to effectively reduce the number of particles needed for the convergence of the concentration field. For example, Fernàndez-Garcia and Sanchez-Vila [27] proposed a Kernel Density Estimator (KDE) method to reconstruct the concentration (we refer to [28] for a practical application of the KDE method).

A comprehensive review about the RWPT method can be found in [29]. Comparison with other numerical methods, e.g., Eulerian, mixed Eulerian–Lagrangian and TVD methods, indicates that Particle Tracking is particularly suited for advection-dominated problems. Since the particles position is not linked to the underlying grid, it is possible to use a coarser grid for the resolution of the flow field. Moreover, the choice of the numerical grid is not affected by the initial contaminant distribution.

The main disadvantage of the Particle Tracking method is the high number of particles required for the convergence. However, since each particle trajectory derived in (5) is independent, this technique can be efficiently parallelized.

3. GPU parallelization

With the particle tracking method, the solute is represented by a set of particles each one carrying a small portion of the total mass. Each particle follows Eq. (5). Since the trajectory of each particle is independent, the particle positions can be updated in parallel. The random walk particle tracking method is particularly suited for GPU parallelization. The hardware architecture of a GPU is different from a CPU. A CPU is composed of few powerful cores, able to run multiple processes independently. On the contrary, a GPU is composed of thousands of small cores, therefore suited to take a huge array of data (e.g., particle positions) and apply the same simple transformation on each element in parallel (e.g., update position following Eq. (5)).

The implementation of the particle tracking method follows these steps. First, the velocity field is imported and copied to the GPU and the particle positions are initialized on the GPU. Then, for each particle the current velocity is evaluated using the appropriate interpolation method. Since the entire velocity field is inside the GPU, there is no transfer of data between the GPU and the host. Once the velocity is computed, we can use Eq. (5) to update the particle positions. Again, this step is executed in parallel on the CUDA device. Finally, we can proceed to the next step. Given the particle positions, it is possible to compute global quantities of interest, e.g., the center of mass of the particles or the number of particles inside a given control volume. These quantities can be efficiently computed inside the GPU, avoiding again the need to transfer all the particle positions between GPU and host.

PAR^2 is a C++/CUDA Random Walk Particle Tracking simulator implementing the strategy discussed above. In order to take advantage of Object Oriented programming features, we employ the Thrust library [30]. The Thrust template library provides a STL interface of the CUDA language with some common algorithm already implemented and optimized. Within the next sections we analyze the main features and the design principles used in PAR^2.

3.1. Data structures

Three arrays in the device global memory are used to store the position of the particles at the current time step. Each array is storing the coordinate of all the particles in a given direction (i.e., x, y and z coordinates). The dimension of these arrays is equal to the number of particles N_p. To exploit the benefits of the Thrust library, the algorithm is rewritten in terms of two basic operations: transform and reduce. For example, the particles position arrays are transformed following Eq. (5) and a single CUDA kernel is called for each time step. This ensures a coalesced access to the coordinate arrays improving the overall computational speed-up.

Next, the velocity field must be stored on the device. The velocity field is obtained by numerically solving the flow Eq. (1) on a Cartesian grid with dimension $n_x \times n_y \times n_z$. The final discrete velocity is defined by the normal velocity in each face of the grid. Thus, three arrays in the device global memory are used to store the x, y and z velocities at the interface with dimensions $(n_x + 1) \times n_y \times n_z$, $n_x \times (n_y + 1) \times n_z$, and $n_x \times n_y \times (n_z + 1)$ respectively. This data will be used for the linear interpolation of the velocity on each particle position. At the same time, for the trilinear interpolation of the velocity (and its derivative), the velocity is evaluated at the corners of each cell. These velocities are precomputed and loaded into the device global memory in three arrays of size $(n_x + 1) \times (n_y + 1) \times (n_z + 1)$.

Finally, we point out that the memory required for the simulation scales linearly with the number of particles N_p and linearly with the number of cells $n_x \times n_y \times n_z$.

3.2. Pseudo-random number generator

One of the most important component of the particle tracking method is the random displacement. This allows to simulate diffusion and dispersion. A reliable pseudo-random number generator must be used in order to compute the vector $\boldsymbol{\xi}$ in (5). Massively parallel generation of independent pseudo-random numbers can be cumbersome and it has been subject of several studies (see [31] for more information on the topic). An inappropriate choice of a parallel pseudo-random number generator can lead to time and/or space cross-correlations between particles, compromising the final results. For this task, the well-established CURAND library is used, allowing the generation of the pseudo-random numbers on the GPU device in parallel.

3.3. Computing the drift vector and the displacement matrix

The drift vector defined in (6) is composed of the advection component $\mathbf{u}$, a correction term due to spatially variable dispersion $\nabla \cdot \mathbf{D}$, and a correction term due to spatially variable porosity $\frac{1}{\Theta}\mathbf{D} \cdot \nabla\Theta$. The correction term due to spatially variable porosity is

not yet implemented and will be added to a future release of PAR^2. Following the approach described in Salamon et al. [14], we use a hybrid interpolation scheme, using linear interpolation (i.e., using velocities at the interfaces) to compute the advection component and trilinear interpolation (i.e., using velocities at the corners) for the correction term due to spatially variable dispersion.

Then, the displacement matrix defined in (7) is evaluated. Including a non-constant dispersion requires to compute the eigenvalues and eigenfunctions of $\mathbf{D}$ for each particle and for each time step. Using classical iterative methods to compute eigenvalues of many small matrices in CUDA threads would dramatically impact the GPU efficiency. Thus, the eigenvalues and eigenvectors of $\mathbf{D}$ are computed analytically.

The velocity at the particle position is evaluated using a trilinear interpolation scheme. Thus, given the Darcy-scale velocity $\mathbf{u} = [u_1, u_2, u_3]^T$ at the particle position, the displacement matrix $\mathbf{B}$ is computed according to (7). First, we note that the dispersion matrix (4) can be rewritten as:

$$\mathbf{D} = a\mathbf{I} + b\mathbf{W}, \tag{9}$$

where $a = \alpha_T|\mathbf{u}| + D_m$, $b = (\alpha_L - \alpha_T)/|\mathbf{u}|$ and $\mathbf{W} = \mathbf{u}\mathbf{u}^T$. Let $\mathbf{v}_i$ be an eigenvector of $\mathbf{W}$ and λ_i the corresponding eigenvalue, then:

$$\mathbf{D}\mathbf{v}_i = a\mathbf{v}_i + b\mathbf{W}\mathbf{v}_i = (a + b\lambda_i)\mathbf{v}_i = \gamma_i\mathbf{v}_i. \tag{10}$$

The result above implies that $\mathbf{v}_i$ is also an eigenvector of $\mathbf{D}$ with eigenvalue $\gamma_i = a + b\lambda_i$. Since $2\mathbf{D} = \mathbf{B}\cdot\mathbf{B}^T$ (see Eq. (7)), given all the eigenvalues and the corresponding orthogonal eigenvectors of $\mathbf{W}$, the matrix $\mathbf{B}$ can be computed as:

$$\mathbf{B} = \sum_{i=1}^{3} \sqrt{2\gamma_i}\frac{\mathbf{v}_i\mathbf{v}_i^T}{\mathbf{v}_i^T\mathbf{v}_i}. \tag{11}$$

More details about the derivation of Eq. (11) can be found in Appendix. If $\mathbf{u} \neq 0$, one of the eigenvalues of $\mathbf{W}$ is the square of the modulus of the velocity $|\mathbf{u}|^2$ with corresponding eigenvector $\mathbf{u}$ (i.e. $\mathbf{W}\mathbf{u} = (\mathbf{u}\mathbf{u}^T)\mathbf{u} = |\mathbf{u}|^2\mathbf{u}$). Moreover, if $u_1 \neq 0$, we can analytically compute the other two pairs such that:

$$\lambda_1 = |\mathbf{u}|^2 \text{ and } \mathbf{v}_1 = \mathbf{u}, \tag{12}$$

$$\lambda_2 = 0 \text{ and } \mathbf{v}_2 = [-u_2, u_1, 0]^T, \tag{13}$$

$$\lambda_3 = 0 \text{ and } \mathbf{v}_3 = [-u_3u_1, -u_3u_2, u_1^2 + u_2^2]^T. \tag{14}$$

Note that $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ form an orthogonal base. This decomposition is valid if all the orthogonal eigenvectors in (12)–(14) are non-zero. A sufficient condition for this to happen is $u_1 \neq 0$. In the case where $u_1 = 0$, a small number ε is added to u_1 such that the numerical error introduced is masked by the molecular diffusion (i.e., $\varepsilon \ll D_m/\alpha_L$). The displacement matrix can be computed analytically for each particle in parallel in the GPU using Eqs. (12)–(14) together with (11).

3.4. Input/output

In order to use PAR^2, a configuration file must be prepared for each simulation. We use the YAML markup language for this file, providing an easy and convenient interface to the code. Therefore, knowledge of C++/CUDA programming language is not required to use PAR^2. Moreover, given the popularity of the YAML format, most of the programming languages already have a library able to read and write a YAML file, thus PAR^2 can be easily collocated in a broader simulation framework. An example of configuration file is:

```
# Grid parameters
grid:
    dimension: [<int>, <int>, <int>]
    cell size: [<float>, <float>, <float>]

# Physical parameters
physics:
    porosity: <float>
    molecular diffusion: <float>
    longitudinal dispersivity: <float>
    transverse dispersivity: <float>
    velocity:
        type: modflow
        file: </path/to/modflow-output.ftl>

# Particle tracking simulation parameters
simulation:
    particles:
        N: <int>
        start:
            p1: [<float>, <float>, <float>]
            p2: [<float>, <float>, <float>]

    dt: <float>
    steps: <int>

# Output parameters
output:
    csv:
        file: </path/to/output.csv>
        skip: <int>
        # List of values to compute
        items:
            - label: <output1>
              type: <tag>
              # options (different for each tag)
            - label: <output2>
              type: <tag>
              # options
            ...

    snapshot:
        file: </path/to/snapshot-*.csv>
        skip: <int>
```

The parameters are divided into four blocks:

1. *grid*: dimensions and parameters of the simulation grid.
2. *physics*: physical parameters of the site.
3. *simulation*: parameters required by the particle tracking method.
4. *output*: define quantities to output during the simulation.

Note that the particle starting position in the simulation block is defined by two points $\mathbf{p}_1$ and $\mathbf{p}_2$. These two points define a volume (i.e., all $\mathbf{x}$ such that $p_{1,i} < x_i < p_{2,i}$ for $i = 1, 2, 3$) in which the particles are randomly and uniformly placed. Finally, the link to MODFLOW is made using the LMT package. Adding this package to the MODFLOW simulation will generate the Flow-Transport Link (FTL) file containing the volumetric flow rates at cell interfaces. The path to this file must be defined in the physics block of the YAML parameter file.

4. Verification and speed-up analysis

In this section we perform a simulation using data from a realistic aquifer. We consider the fluvio-aolian deposits located in Southeast Brazil close to the town of Descalvado in the state of São Paulo. Geological and laboratory analysis of the site have been provided by Bayer et al. [32,33]. The geological formation is characterized

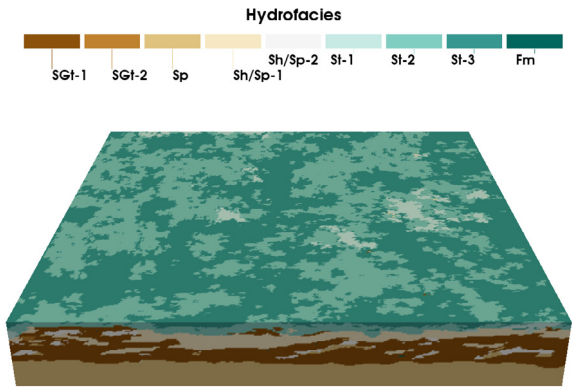

Fig. 1. MPS simulation of the fluvio-aolian deposits located in Descalvado, Brazil [32]. The size of the field is 42 m × 42 m × 5.8 m. Each color represents a hydroface (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

Table 1
Lithofacies and hydrofacies distribution of the Descalvado site.

Lithofacies	Description	K [m/s]	%
SGt	Trough-cross-bedded sand and gravel	3.0×10^{-4}	29.12
		9.4×10^{-5}	1.72
Sp	Planar-cross-bedded aeolian sand	1.6×10^{-4}	34.97
Sh/Sp	Horizontally laminated to planar cross-stratified sand	1.4×10^{-3}	8.84
		7.8×10^{-5}	1.33
		6.0×10^{-5}	5.31
St	Trough-cross-bedded sand	2.5×10^{-5}	9.52
		6.2×10^{-6}	9.16
Fm	Massive clay intraclasts	7.8×10^{-8}	0.03

by 5 lithofacies and 9 hydrofacies (see Table 1). Using multiple-point statistics (MPS) simulations, Bayer et al. [32,33] randomly generated different realizations of the hydraulic conductivity spatial distribution. For this work, we use one of these realizations, depicted in Fig. 1. The aquifer's dimension is $L_x \times L_y \times L_z$, where $L_x = L_y = 42$ m and $L_z = 5.8$ m. The grid resolution is $420 \times 420 \times 58$ cells (i.e., the cell size is 0.1 m × 0.1 m × 0.1 m), with a total of 10, 231, 200 cells. For simplicity, the effective porosity is assumed to be constant and equal to the average porosity of the entire field ($\Theta = 0.285$). The flow field is computed using MODFLOW [21]. The hydraulic head on the boundaries orthogonal to the x-axis is set such that the difference $\Delta\hat{\phi}$ satisfies $\Delta\hat{\phi}/L_x = 0.01$ (i.e. Dirichlet boundary conditions). A no-flow condition is used for the other boundaries (i.e., Neumann type boundary conditions).

The flow field generated by MODFLOW is used for the simulation of the solute transport. For the transport simulation we use the GPU-accelerated PAR^2. To verify the correctness and the performance of the software, we compare the results with the RW3D [10–12], a robust software implementing the RWPT method without parallelization. For the purpose of illustration, the dispersivities are assumed to be $\alpha_L = 0.01m$ and $\alpha_T = 0.001$ m. The molecular diffusion coefficient is $D_m = 10^{-9} m^2/s$. We consider a line source located between the points $\mathbf{s}_1 = (5, 10, 2.9)$ and $\mathbf{s}_2 = (5, 32, 2.9)$. The particles are uniformly distributed along the line source. The total simulation time is 30 days with a time step $\Delta t = 1h = 3600$ s for a total of 720 steps. Two snapshots of the particle positions are displayed in Fig. 2.

All the simulations are carried out using a desktop computer equipped with a CPU Intel Core i7-6700 CPU @ 3.40 GHz x 8 (for RW3D) and a GPU NVIDIA GeForce GTX 745/PCIe/SSE2 (for PAR^2) equipped with 384 CUDA Cores and 4GB of internal RAM memory. First, we set the number of particles $N_p = 20{,}000$. Two snapshots at time $t = 7.5$ days and $t = 30$ days of the particle positions are shown in Fig. 2. The particles distribution is in agreement in both methods. In Fig. 3 we compare the normalized cumulative Break-Through Curve (BTC) at two control planes (CP) orthogonal to the x-axis located at $5m$ (CP-1) and $25m$ (CP-2) from the injection line-source. The results reported in Fig. 3 show an excellent agreement between RW3D and PAR^2.

Finally, we report the computational run time for the whole simulation using different number of particles in both RW3D and PAR^2. As shown in Fig. 4, PAR^2 is able to complete the entire simulation within few seconds. When utilizing 100,000 particles, the simulation carried out with PAR^2 is roughly 98 times faster than the corresponding non-parallelized simulation. PAR^2 can be compiled using single-precision or double-precision float numbers. As shown in Fig. 4, using single-precision usually leads to faster simulations and we have not seen any remarkable difference on the results due to the float-precision choice. Note that the speed-up difference due to the float precision depends on the GPU device used. It is clear that the proposed GPU parallelization provides a manner to mitigate the high number of particles constrain that typically characterizes the particle tracking methods. For example, using 1 million particles for the same simulation would require a total computation time of 43.8 seconds (using our desktop computer and single-precision float numbers). Simulation time can be further reduced by using GPU cards dedicated to scientific computing, usually more powerful than desktop GPUs. To understand the amount of memory used in the device, we run a simulation with 200 million particles. In this case, the simulation was completed in less than 2 hours. It is interesting to note that PAR^2 used only 2.6GB of the GPU internal memory, showing the possibility to perform relatively large simulations despite GPU memory limitations.

Being able to perform simulations with millions of particles allows to compute the solute concentration over very small control volume (e.g., an observation well). Moreover, it is possible to obtain a better convergence for the high-order spatial-moments of the plume (e.g., skewness and kurtosis) or the time-derivative of the spatial-moments (e.g., effective dispersion).

5. Summary and future work

We developed PAR^2, an open source, simple-to-use, GPU-accelerated Random Walk Particle Tracking C++/CUDA code to simulate solute transport in groundwater flows. The RWPT method is particularly suited for parallelization, since the trajectory of each particle is independent from the others. All the simulation is carried out inside the GPU device, avoiding massive transfer of data between the host and the device. To achieve this goal, we provide an alternative method to compute the displacement matrix associated with the local-scale dispersion tensor. Furthermore, simulation parameters are defined in a YAML configuration file, where it is also included a convenient interface with MODFLOW, a commonly used groundwater flow simulator. The usage of PAR^2 does not require any particular knowledge about C++/CUDA programming techniques.

We provide a benchmark simulation, where solute transport in a realistic aquifer located in Brazil is simulated. In addition, we provide a comparison between PAR^2 and another available particle tracking code in the literature. Our results indicate a significant reduction of computational time, being able to complete the full benchmark simulation in less than a minute. Due to its efficiency, the software can be adopted within a Monte Carlo framework on desktops equipped with NVIDIA GPU cards or GPU-accelerated HPC clusters, by accelerating the transport simulation in each realization. RWPT simulations using millions of particles are possible

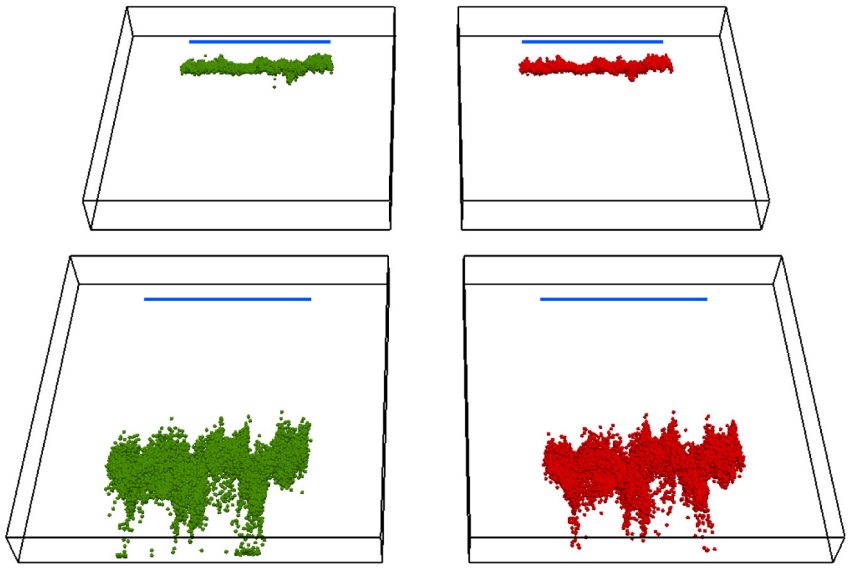

Fig. 2. Raw particles position using RW3D (red particles) and PAR^2 (green particles) after 7.5 days (top) and 30 days (bottom). The number of particles is $N_p = 20{,}000$ in both cases. Blue lines indicate the initial position of the particles (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

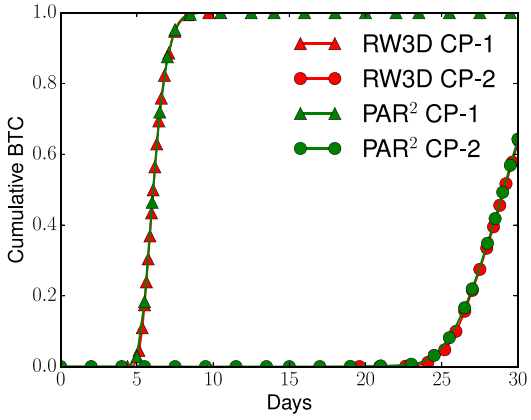

Fig. 3. Normalized cumulative Break-Through Curve (BTC) computed at two control planes (CP) orthogonal to the x-axis located at 5 m (CP-1) and 25 m (CP-2) from the injection line-source using RW3D and PAR^2.

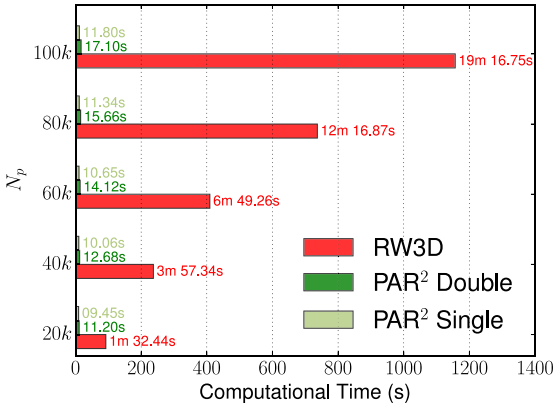

Fig. 4. Computational time required to run the full benchmark simulation using RW3D and PAR^2 with different number of particles N_p. Computational times for PAR^2 are reported using both single-precision and double-precision float numbers.

thanks to the GPU parallelization, providing a way to ameliorate the computational cost of large numbers of particles that might limit the range of applications. As shown in the benchmark test, we were able to complete the simulation in a grid with 10 million cells using 200 million particles. These numbers can be easily increased using modern and more capable GPUs.

PAR^2 alleviates the computational costs associated with stochastic simulations of hydrogeological systems. However, the particle tracking software can also be employed in other physical systems that are governed by highly advective (or convective) processes. As for possible future work, the software needs to be expanded to account for more complex flow scenarios such as partially saturated porous media flows, non-constant porosity, transient conditions, unstructured grid (e.g., see [34]), and discrete fracture networks (e.g., see [35]). Probably the most challenging problem, from an implementation point of view, is the inclusion of complex biogeochemical reactions (see [15]). Many efficient approaches to include reactions in RWPT have been proposed recently (e.g., see [12,36–38]). However, in this case a highly scalable approach may be more indicated than the most efficient one.

Acknowledgments

The authors acknowledge Daniel Fernàndez-Garcia and Christopher V. Henri for providing RW3D, the CPU Random Walk Particle Tracking code, Peter Bayer, Alessandro Comunian, Dominik Höyng and Gregoire Mariethoz for providing the open dataset used for the benchmark, and Jeremy Bennett for the suggestion of such dataset. This work was funded by the National Science Foundation under Grant No. 1654009.

Appendix

In this section, we prove that choosing $\mathbf{B}$ as in Eq. (11) satisfies the relation $2\mathbf{D} = \mathbf{B} \cdot \mathbf{B}^T$ (Eq. (7)). If γ_i and $\mathbf{v}_i$ are the eigenvalues and orthogonal eigenvectors of $\mathbf{D}$, then:

$$\mathbf{B} \cdot \mathbf{B}^T = \left(\sum_{i=1}^{3} \sqrt{2\gamma_i} \frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i} \right) \cdot \left(\sum_{i=1}^{3} \sqrt{2\gamma_i} \frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i} \right)^T$$
$$= \left(\sum_{i=1}^{3} \sqrt{2\gamma_i} \frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i} \right) \cdot \left(\sum_{i=1}^{3} \sqrt{2\gamma_i} \frac{(\mathbf{v}_i \mathbf{v}_i^T)^T}{\mathbf{v}_i^T \mathbf{v}_i} \right).$$

Since $\mathbf{v}_i\mathbf{v}_i^T$ is symmetric, then:

$$= \left(\sum_{i=1}^{3}\sqrt{2\gamma_i}\frac{\mathbf{v}_i\mathbf{v}_i^T}{\mathbf{v}_i^T\mathbf{v}_i}\right)\cdot\left(\sum_{i=1}^{3}\sqrt{2\gamma_i}\frac{\mathbf{v}_i\mathbf{v}_i^T}{\mathbf{v}_i^T\mathbf{v}_i}\right)$$

$$= \sum_{i=1}^{3}\sum_{j=1}^{3}\sqrt{2\gamma_i}\sqrt{2\gamma_j}\frac{\mathbf{v}_i\mathbf{v}_i^T\mathbf{v}_j\mathbf{v}_j^T}{\mathbf{v}_i^T\mathbf{v}_i\mathbf{v}_j^T\mathbf{v}_j}.$$

Since $\mathbf{v}_i$ are orthogonal, $\mathbf{v}_i^T\mathbf{v}_j = \delta_{ij}\mathbf{v}_i^T\mathbf{v}_i$, therefore:

$$= \sum_{i=1}^{3} 2\gamma_i\frac{\mathbf{v}_i\mathbf{v}_i^T}{\mathbf{v}_i^T\mathbf{v}_i}.$$

This final form is equal to $2\mathbf{D}$, in fact for every eigenvector $\mathbf{v}_j$ (i.e., $\mathbf{D}\mathbf{v}_j = \gamma_j\mathbf{v}_j$):

$$\mathbf{B}\cdot\mathbf{B}^T\mathbf{v}_j = \sum_{i=1}^{3} 2\gamma_i\frac{\mathbf{v}_i\mathbf{v}_i^T}{\mathbf{v}_i^T\mathbf{v}_i}\mathbf{v}_j = 2\gamma_j\mathbf{v}_j = 2\mathbf{D}\mathbf{v}_j.$$

References

[1] Y. Rubin, Applied Stochastic Hydrogeology, Oxford University Press, 2003.
[2] M. Dentz, F.P.J. de Barros, J. Fluid Mech. 777 (2015) 178–195.
[3] D. Allard, J.-p. chilès, p. delfiner, Geostatistics: Modeling Spatial Uncertainty, 2013.
[4] F. Ballio, A. Guadagnini, Water Resour. Res. 40 (4).
[5] M. Moslehi, R. Rajagopal, F.P.J. de Barros, Adv. Water Resour. 83 (2015) 299–309.
[6] M. Moslehi, F.P.J. de Barros, F. Ebrahimi, M. Sahimi, Adv. Water Resour. 96 (2016) 180–189.
[7] J. Bear, A.H.-D. Cheng, Modeling Groundwater Flow and Contaminant Transport, vol. 23, Springer Science & Business Media, 2010.
[8] C. Zheng, MT3D, a modular three-dimensional transport model.
[9] C. Zheng, P.P. Wang, Tech. rep., Alabama Univ University, 1999.
[10] D. Fernàndez-Garcia, T.H. Illangasekare, H. Rajaram, Water Resour. Res. 41 (3).
[11] D. Fernàndez-Garcia, T.H. Illangasekare, H. Rajaram, Adv. Water Resour. 28 (7) (2005) 745–759.
[12] C.V. Henri, D. Fernàndez-Garcia, Water Resour. Res. 50 (9) (2014) 7206–7230.
[13] E. LaBolle, User's Manual and Program Documentation, Univ. of Calif. Davis.
[14] P. Salamon, D. Fernàndez-Garcia, J.J. Gómez-Hernández, J. Contam. Hydrol. 87 (3–4) (2006) 277–305.
[15] F. Boso, A. Bellin, M. Dumbser, Adv. Water Resour. 52 (2013) 178–189.
[16] C. Zheng, G.D. Bennett, Applied Contaminant Transport Modeling, vol. 2, Wiley-Interscience New York, 2002.
[17] A.E. Hassan, M.M. Mohamed, J. Hydrol. 275 (3–4) (2003) 242–260.
[18] D. Fernàndez-Garcia, X. Sanchez-Vila, J. Contam. Hydrol. 120 (2011) 99–114.
[19] D. Ding, D.A. Benson, A. Paster, D. Bolster, Adv. Water Resour. 53 (2013) 56–65.
[20] C.B.R. Rizzo, Par^2, 2018, https://github.com/GerryR/par2.
[21] A.W. Harbaugh, MODFLOW-2005, The US Geological Survey Modular Ground-Water Model: The Ground-Water Flow Process, US Department of the Interior, US Geological Survey Reston, 2005.
[22] G.E. Hammond, P.C. Lichtner, R. Mills, Water Resour. Res. 50 (1) (2014) 208–228.
[23] E.M. LaBolle, G.E. Fogg, A.F. Tompson, Water Resour. Res. 32 (3) (1996) 583–593.
[24] R. Burnett, E. Frind, Water Resour. Res. 23 (4) (1987) 683–694.
[25] P.C. Lichtner, S. Kelkar, B. Robinson, Water Resour. Res. 38 (8).
[26] H. Risken, Fokker–Planck Equation, Springer, 1996.
[27] D. Fernàndez-Garcia, X. Sanchez-Vila, J. Contaminant Hydrol. 120 (2011) 99–114.
[28] D. Pedretti, D. Fernàndez-Garcia, D. Bolster, X. Sanchez-Vila, Water Resour. Res. 49 (7) (2013) 4157–4173.
[29] P. Salamon, D. Fernàndez-Garcia, J.J. Gómez-Hernández, J. Contaminant Hydrol. 87 (3–4) (2006) 277–305.
[30] N. Bell, J. Hoberock, GPU Computing Gems Jade Edition, Elsevier, 2011, pp. 359–371.
[31] C.L. Phillips, J.A. Anderson, S.C. Glotzer, J. Comput. Phys. 230 (19) (2011) 7191–7201.
[32] P. Bayer, A. Comunian, D. Höyng, G. Mariethoz, Sci. Data 2 (2015) 150033.
[33] P. Bayer, A. Comunian, D. Höyng, G. Mariethoz, Physicochemical properties and 3d geostatistical simulations of the herten and the descalvado aquifer analogs.
[34] S. Painter, C. Gable, S. Kelkar, Comput. Geosci. 16 (4) (2012) 1125–1134.
[35] N. Makedonska, S.L. Painter, Q.M. Bui, C.W. Gable, S. Karra, Comput. Geosci. 19 (5) (2015) 1123–1137.
[36] M. Rahbaralam, D. Fernàndez-Garcia, X. Sanchez-Vila, J. Comput. Phys. 303 (2015) 95–104.
[37] D.A. Benson, D. Bolster, Water Resour. Res. 52 (11) (2016) 9190–9200.
[38] G. Sole-Mari, D. Fernàndez-Garcia, P. Rodríguez-Escales, X. Sanchez-Vila, Water Resour. Res. 53 (11) (2017) 9019–9039.