Solving hyperbolic partial differential equations using CUDA

Víctor Osores, Diego Maldonado

Departamento de Matemática, Física y Estadística, Universidad Católica del Maule

E-mail: `vosores@ucm.cl, dmaldonado@ucm.cl`

Abstract. Hyperbolic partial differential equations play a crucial role in modeling phenomena such as fluid dynamics, traffic flow, and wave interactions. Due to the complexity of these equations, numerical methods are essential for obtaining solutions, particularly with the advent of parallel computing. Graphics Processing Units (GPUs), combined with NVIDIA's CUDA programming model, have enabled significant advancements in scientific computing.

This work focuses on solving hyperbolic PDEs using CUDA to exploit the massive parallel processing power of GPUs. Specifically, we address Burgers' equation, the batch monodisperse sedimentation equation, and the polydisperse sedimentation model. Different numerical approaches are implemented, considering the architecture of NVIDIA GPUs and the parallel computing model. The results are compared to identify the most efficient methods, supported by numerical simulations that highlight the accuracy and computational performance of the solutions.

1. Introduction

Partial differential equations (PDEs) are fundamental for describing a wide range of phenomena in geophysics [1], biology, chemistry [2] and in the mining industry [3]. Hyperbolic PDEs, in particular, model complex processes such as mudflows, avalanches, and sedimentation, with diverse applications. However, their nonlinear nature and the complexity of the modeled phenomena make analytical solutions infeasible in most cases.

Technological advancements, especially in parallel computing [4], have revolutionized numerical methods for PDEs. GPUs, with their massive parallel processing capabilities, enable the efficient execution of thousands of threads simultaneously. Using programming environments like CUDA by NVIDIA [5], GPUs allow for significant reductions in computation times for high-dimensional dynamic systems.

This work extends classical numerical methods for solving hyperbolic PDEs using CUDA. It focuses on parallelizing methods for three problems: Burgers' equation, the monodisperse sedimentation model, and the polydisperse sedimentation model. These conservation law systems address scenarios involving particles of varying properties, offering realistic and scalable numerical solutions. Optimization strategies include efficient memory use, thread management, and minimization of high-latency global memory usage. Results are evaluated in terms of computational efficiency, numerical accuracy, and scalability. Simulations demonstrate the superior performance of GPU implementations over traditional CPU-based methods for complex problems.

2. Preliminaries

2.1. Parallel computing and CUDA

Parallel computing has its roots in the 1960s, when the first multiprocessor systems were developed. For these machines, parallel algorithms were developed, whose core lies in dividing a problem into independent or interdependent subproblems and solving them simultaneously across multiple processors.

CUDA was introduced by NVIDIA in 2007 as a platform and programming model for computing on its Graphics Processing Units (GPUs) [5]. Before CUDA, programming GPUs required the use of graphics-specific APIs, which limited their adoption in general-purpose applications. CUDA architecture organizes the hardware into *blocks* and *threads*. Each thread executes an instance of the program, while blocks group threads into a scalable hierarchy. Key elements include shared memory, which allows efficient communication between threads within the same block; global memory, accessible by all threads but with higher latency; and registers, which are private to each thread and used to store thread-specific variables, allowing fast and efficient access.

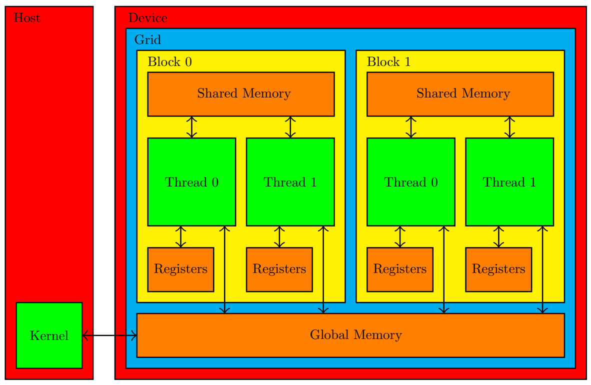

Figure 1. CUDA architecture.

The CUDA model distinguishes between the *Host* and the *Device*. The Host refers to the CPU that coordinates execution, while the Device is the GPU where parallel computations are performed. A function written to run on the Device is called a *Kernel*. Kernels are launched from the Host and executed by multiple threads in parallel within the Device.

2.2. Mathematical models

We begin this section by considering the inviscid Burgers' equation in the quasilinear form [6]

$$u_t + uu_x = 0, \tag{1}$$

This conservation law with a nonlinear flux $f(u) = \frac{1}{2}u^2$ describes a wide variety of physical phenomena related to the propagation of nonlinear waves. Here, u can represent, the velocity of a flow, the height of a wave, or the density or concentration associated with a given phenomenon. Although the equation is conceptually simple, it poses significant numerical challenges in accurately capturing the discontinuities. In many cases, excessive domain refinement is required to achieve a high-quality solution, which significantly increases the computational cost.

Let us now consider the following Riemann problem for the inviscid Burgers' equation (1)

$$u_t + uu_x = 0, \quad u(x,0) = \begin{cases} u_l, & \text{if } x < 0, \\ u_r, & \text{if } x > 0, \end{cases} \tag{2}$$

where u_l and u_r represent the left and right states, respectively, defining an initial discontinuity. It is well known that if $u_l < u_r$, the characteristic curves diverge, resulting in a rarefaction wave. On the other hand, if $u_l > u_r$, the characteristic curves intersect, generating a shock wave (discontinuity). The speed of the shock, s, is determined by the Rankine-Hugoniot condition, which in this case simplifies to $s = \frac{1}{2}(u_r + u_l)$. The exact solution of (2) can be find in [6].

The following system of conservation laws (see [7]), describes the polydisperse sedimentation process of N different solid species dispersed in the viscous fluid in one spatial dimension

$$\frac{\partial \phi_i}{\partial t} + \frac{\partial \phi_i v_i^{MLB}(\Phi)}{\partial z} = 0, \quad i = 1, \dots, N, \tag{3}$$

complemented with an appropriate initial condition and boundary conditions. Here, t represents time, z is the spatial variable (height), and $(\phi_1, \dots, \phi_N) \in \boldsymbol{\mathcal{D}}$, where

$$\boldsymbol{\mathcal{D}} = \left\{ \Phi = (\phi_1, \dots, \phi_N) \in \mathbb{R}^N : \phi_1 \geq 0, \dots, \phi_N \geq 0; \phi_1 + \dots + \phi_N \leq \phi_{\max} \right\},$$

where ϕ_i are the volumetric concentrations of the solid species at position z and time t and $\phi_{\max}$ is the maximum concentration allowed in the mixture. The hindered settling velocity proposed by Masliyah [8] and Lockett and Bassoon [9], $v_i^{\mathrm{MLB}}(\Phi)$ is defined as

$$v_i^{\mathrm{MLB}}(\Phi) = \mu V(\phi) \left[\delta_i \left(\bar{\rho}_i - \bar{\rho}^{\mathrm{T}} \Phi \right) - \sum_{l=1}^{N} \delta_l \phi_l \left(\bar{\rho}_l - \bar{\rho}^{\mathrm{T}} \Phi \right) \right], \quad i = 1, \dots, N,$$

where the vectors $\bar{\rho}_i := \rho_i - \rho_0, \quad i = 1, \dots, N$, $\bar{\rho} := (\bar{\rho}_1, \dots, \bar{\rho}_N)^{\mathrm{T}}$ and the parameter $\mu := -\frac{g d_1^2}{18 \mu_{\mathrm{f}}}$, where g is the gravitational acceleration, μ_{f} is the viscosity of the fluid, and $\delta_i := \frac{d_i^2}{d_1^2}$, with d_i being the diameter of species i. The hindered settling factor $V(\phi)$ is defined as

$$V(\phi) = \begin{cases} (1-\phi)^{\lambda - 2}, & \text{if } \Phi \in \boldsymbol{\mathcal{D}}, \quad \lambda > 2, \\ 0, & \text{otherwise.} \end{cases}$$

2.3. Numerical schemes

In this section, we present numerical methods to solve the previously described models. Considering $\Omega = [0, L] \times [0, T]$, the spatial discretization is carried out using $\mathcal{N}$ finite volumes of size $\Delta x = L/\mathcal{N}$. Each volume $V_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is delimited by its interfaces $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$, defined by $x_{i+\frac{1}{2}} = (i+1)\Delta x$ and the volume centers are located at $x_i = \left(i + \frac{1}{2}\right) \Delta x$, for $i = 0, 1, \dots, \mathcal{N} - 1$. For the temporal discretization, we introduce an integer $\mathcal{T}$, and Δt_n satisfies $t_{n+1} = t_n + \Delta t_n$ for $n = 0, 1, \dots, \mathcal{T}$ ($t_0 = 0$) and $\Delta t_0 + \dots + \Delta t_{\mathcal{T}-1} = T$. The numerical solution is approximated in each finite volume by conservative finite volume scheme

$$u_i^{n+1} = u_i^n - \frac{\Delta t_n}{\Delta x} \left(\hat{f}^n_{i+\frac{1}{2}} - \hat{f}^n_{i-\frac{1}{2}} \right) \qquad \text{where,} \qquad u_i^n \approx \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u(x, t^n)\, dx,$$

u_i^n represents the average value of the solution $u(x, t^n)$ in V_i at time t^n, $\hat{f}^n_{i+\frac{1}{2}}$ is the numerical flux that approximates $f(u(x_{i+\frac{1}{2}}, t^n))$ at the interface $x_{i+\frac{1}{2}}$.

For model (2), we will use the Lax-Friedrichs method, whose numerical flux is given by

$$\hat{f}_{i+\frac{1}{2}} = \frac{1}{2} \left(f(u_i) + f(u_{i+1}) \right) - \frac{\Delta x}{2 \Delta t_n} \left(u_{i+1} - u_i \right),$$

where the time step $\Delta t_n := \mathrm{CFL} \cdot \Delta x / \max_i |f'(u_i^n)|$ is computed according to the CFL condition. Analogously, from (3) considering $N = 1$ we obtain the monodisperse model, which is

$$\frac{\partial \phi}{\partial t} + \frac{\partial f(\phi)}{\partial z} = 0 \text{ with, } f(\phi) = -\frac{g d^2 \bar{\varrho}_{\mathrm{s}}}{18 \mu_{\mathrm{f}}} V(\phi) \phi (1-\phi)^2. \tag{4}$$

To solve (4), we will consider the Godunov method with Δt_n follows the same definition before,

$$\phi_i^{n+1} = \phi_i^n - \frac{\Delta t_n}{\Delta z}\left(\hat{f^{\mathrm{G}}}^n_{i+\frac{1}{2}} - \hat{f^{\mathrm{G}}}^n_{i-\frac{1}{2}}\right), \text{ with } \hat{f^{\mathrm{G}}}(u,v) = \begin{cases} \min_{u\leqslant\phi\leqslant v} f(\phi), & \text{if } u \leqslant v, \\ \max_{u\geqslant\phi\geqslant v} f(\phi), & \text{if } u > v. \end{cases}$$

Finally, for the polydisperse model, we follow the numerical method proposed in [10],

$$\begin{aligned} \phi_{i,j}^{n+1} &= \phi_{i,j}^n - \frac{\Delta t_n}{\Delta z}\left(\hat{f}^n_{i,j+\frac{1}{2}} - \hat{f}^n_{i,j-\frac{1}{2}}\right), \text{with numerical flux} \\ \hat{f}^n_{i,j+\frac{1}{2}} &= \frac{1}{2}\left(\phi_{i,j+1}v_i(\Phi_{j+1}) + \phi_{i,j}v_i(\Phi_j)\right) - \frac{|v_i(\Phi_{j+1})|}{2}\left(\phi_{i,j+1} - \phi_{i,j}\right) \\ &- \frac{\phi_{i,j}}{2}\left|v_i(\Phi_j) - v_i(\Phi_{j+1})\right| \operatorname{sgn}\left(\phi_{i,j+1} - \phi_{i,j}\right), \quad i = 1,\ldots,N. \end{aligned}$$

Here, the time step is defined $\Delta t_n = 0.5 \cdot \Delta x / \max_{j=0,1,\ldots,\mathcal{N}-1} \max_{i=1,\ldots,N} \left|v_i(\Phi_j^n)\right|$. Boundary conditions are imposed at $x_{-\frac{1}{2}} = 0$ and $x_{\mathcal{N}-\frac{1}{2}} = L$.

The core of this numerical methods can be summarized by the following parallel algorithms

Algorithm 1.

```
Input: PDE parameters (C.T., constants, etc),
       Numerical parameters ($\Delta x$, CFL, etc)
Output: Solution of the PDE at a given time.
1 while $T_{Current} < T$ do
2     for $i = 1, ..., \mathcal{N}-1$ do in parallel
3         $\phi_i^{t+1} = \phi_i^t + \frac{\Delta t}{\Delta x}\left(F(\phi_i^t, \phi_{i+1}^t) - F(\phi_{i-1}^t, \phi_i^t)\right)$;
4     $\phi_0^{t+1}, \phi_{\mathcal{N}}^{t+1}$ = Boundary condition;
5     $T_{\text{Current}} = T_{\text{Current}} + \Delta t$;
6 return $\phi$;
```

Algorithm 2.

```
Input: PDE parameters (C.T., constants, etc),
       Numerical parameters ($\Delta x$, CFL, etc)
Output: Solution of the PDE at a given time.
1 while $T_{Current} < T$ do
2     for $e = 1, ..., N$ do in parallel
3         for $i = 1, ..., \mathcal{N}-1$ do in parallel
4             $\phi_i^{t+1} =$
                  $\phi_i^t + \frac{\Delta t}{\Delta x}\left(F(\phi_i^t, \phi_{i+1}^t) - F(\phi_{i-1}^t, \phi_i^t)\right)$;
5     $\phi_0^{t+1}, \phi_{\mathcal{N}}^{t+1}$ = Boundary condition;
6     $T_{\text{Current}} = T_{\text{Current}} + \Delta t$;
7 return $\phi$;
```

Figure 2. Algorithm 1: Pseudocode for Burgers and monodisperse equations. Algorithm 2: Pseudocode for polydisperse equation.

3. Implementation

In this section, we will show the implementation of the numerical method for the monodispersed sedimentation, as it is slightly more complex than Burgers' equation and provides insights into how the polydispersed case works. The three Kernels written in CUDA and a PyCUDA example can be found in `https://github.com/DAMM16/hyper_parallel/`.

- **Host:** We will control the Host using PyCUDA. The GPU configuration will be omitted. The code implementing Algorithm 1 in PyCUDA is as follows

```python
t = np.int32(0)
T_actual = 0
result = np.empty_like(phi)
while T_actual < T:
    compute_phi_new(phi_gpu, dt_dx, phimin, phimax, v0, nrz, n, t, block=block_size, grid=grid_size)
    pycuda.autoinit.context.synchronize() # Synchronize threads
    T_actual = T_actual + dt              # Time step
    t = np.int32(t+1)
drv.memcpy_dtoh(result, phi_gpu)          # Copy data back to the host
```

In this code, `compute_phi_new` is the Kernel that computes in parallel the solution value at the next time step based on the current solution using all the necessary parameters to compute the numerical flux with the Godunov method.

- **Kernel CUDA:** We implement `compute_phi_new` in CUDA. We divide the code into three functions: `fbk`, flux function; `Godunov`, which computes a flux approximation; and `compute_phi_new`, which computes lines 4 and 5 of Algorithm 1.

```
__device__ double fbk(double phi, double phimax, double v0, double nrz) {
    double fbk_val = 0.0;
    if ((phi >= 0.0) && (phi <= phimax)) {
        fbk_val = phi * v0 * pow(1.0 - phi/phimax, nrz);}
    return fbk_val;}
__device__ double Godunov(double Cj, double Ck, double phimax, double v0, double nrz) {
    double gd = 0.0; double hat_phi = phimax / (1.0 + nrz);
    if (Cj <= Ck) {gd = fmin(fbk(Cj, phimax, v0, nrz), fbk(Ck, phimax, v0, nrz));}
    else if ((hat_phi - Cj) * (hat_phi - Ck) < 0.0) {gd = fbk(hat_phi, phimax, v0,nrz);}
    else {gd = fmax(fbk(Cj, phimax, v0, nrz), fbk(Ck, phimax, v0, nrz));}
    return gd;}
__global__ void compute_phi_new(double* phi,double dt_dx, double v0, double nrz, int n, int t) {
    int i = blockIdx.x * blockDim.x + threadIdx.x + 0;
    int even = n*(t%2); int odd = n*((t+1)%2);
    if (1<i && i<n-2) {
        phi[i+odd] = phi[i + even] - dt_dx * (Godunov(phi[i + even], phi[i + 1 + even], phimax, v0, nrz)
        - Godunov(phi[i-1 + even], phi[i + even], phimax, v0, nrz));}
    // Boundary conditions
    if (i == 0) {phi[i+odd]=phimin;}
    if (i == 1) {phi[i+odd]=phi[i + even] - dt_dx * (Godunov(phi[i + even], phi[i + 1 + even], phimax, v0, nrz));}
    if (i == n-2) {phi[i+odd]=phi[i + even] - dt_dx * ( - Godunov(phi[i-1 + even], phi[i + even], phimax, v0, nrz));}
    if (i == n-1) {phi[i+odd]=phimax;}}
```

`compute_phi_new` updates the new values of ϕ not using the usual given by Strategy 1,

```
for t in range(T):
    phi_new=F(phi_old)
    phi_old=phi_new
```

Strategy 1.

```
for t in range(T):
    even = n*(t%2); odd  = n*((t+1)%2)
    phi[0+odd:n+odd]=F(phi[0+even:n+even])
```

Strategy 2.

instead, a single array `phi` of length $2n$ is used, where the first n values store the solution at even times, and the following n values store the solution at odd times.

CUDA stores `phi` in global memory, which has high latency. Therefore, the Strategy 1 is unfavorable, as it computes in one vector and then copies it to another. The Strategy 2 calculates and stores the result to another vector in a single step without moving data.

3.1. Computational complexity

Lemma 1. *Let u be the solution obtained by a numerical method with $\mathcal{N}$ cells and $\mathcal{T}$ time steps. If the smallest value of $\mathcal{T}$ satisfying the CFL condition is chosen, then $\mathcal{T} = \mathcal{O}(\mathcal{N})$.* [1]

Proof. From CFL condition, we have that $\Delta t_n := \mathrm{CFL} \cdot \Delta x / \max_i |f'(u_i^n)|$ then $\mathcal{T} = \mathcal{O}(\mathcal{N})$. $\square$

Theorem 1. *Algorithms 1 solve the Burgers' equation and the monodisperse and Algorithms 2 solve polydisperse sedimentation equations using times and processors specified in Tables 1-2.*

Equation	Time	Processors
Burgers	$\mathcal{O}(\mathcal{N}^2)$	1
Monodisperse	$\mathcal{O}(\mathcal{N}^2)$	1
Polydisperse	$\mathcal{O}(N \cdot \mathcal{N}^2)$	1

Table 1. Sequential computation.

Equation	Time	Processors
Burgers	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
Monodisperse	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
Polydisperse	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(N \cdot \mathcal{N})$

Table 2. Parallel computation.

Proof. In both algorithms, the functions F and boundary conditions uses constant time. Thus, the times and processors used depend on the `while`, `for`, and `for ... in parallel` loops.
- **Sequential Case (Table 1)**: Since it is a sequential machine, the `for ... do in parallel` loops become traditional `for` loops. For Algorithm 1, `lines` 2-3 use $\mathcal{O}(\mathcal{N})$ operations because the computation of F is $\mathcal{O}(1)$. `Line` 5 takes $\mathcal{O}(1)$ because at the domain's boundary the solution is known, then it is used $\mathcal{O}(1)$ time. Summarizing, `lines` 2-5 use $\mathcal{O}(\mathcal{N}) + \mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(\mathcal{N})$.

[1] $\mathcal{O}(n)$ is the well-known big O notation in computational complexity.

`Lines` 1-5 repeat `lines` 2-5 $\mathcal{O}(\mathcal{T})$ times. Thus, their time complexity is $\mathcal{O}(\mathcal{T} \cdot \mathcal{N})$. However, by Lemma 1, $\mathcal{T} = \mathcal{O}(\mathcal{N})$. Therefore, the complexity of Algorithm 1 is $\mathcal{O}(\mathcal{T} \cdot \mathcal{N}) = \mathcal{O}(\mathcal{N}^2)$. Then Burgers' equations and monodisperse sedimentation require that time.

The analysis of Algorithm 2 is analogous. However, a `for` loop between `lines` 2 and 4 repeats N times, resulting in a time complexity of $\mathcal{O}(N \cdot \mathcal{N}^2)$.

- **Parallel Case (Table 2)**: This case is analogous to the sequential one shown earlier, but the `for ... in parallel` loops execute in time $\mathcal{O}(1)$, yet require $\mathcal{O}(\mathcal{N})$ and $\mathcal{O}(\mathcal{N})$ processors, respectively. This reduces the time to $\mathcal{O}(\mathcal{N})$ while using $\mathcal{O}(N\mathcal{N})$ processors. $\square$

4. Simulations and experimental time

In this section, we present the results of parallel implementations, showing the expected behavior and confirming the consistency of the observed outcomes. Additionally, we perform timing experiments to experimentally validate the analytical bounds for execution times.

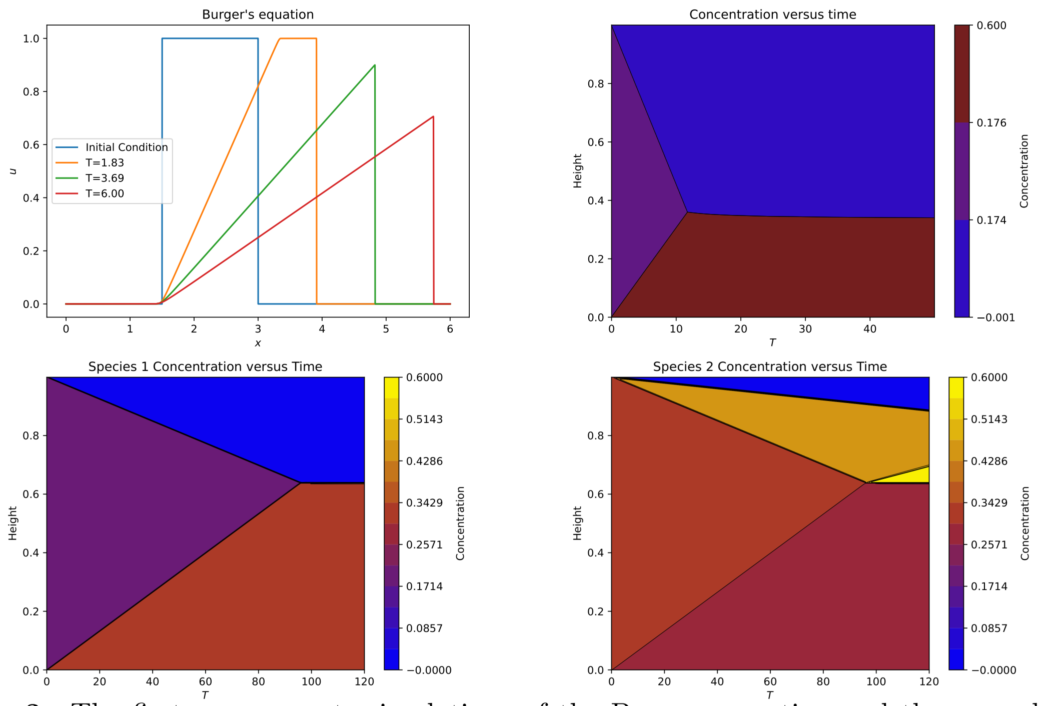

Figure 3. The first row presents simulations of the Burgers equation and the monodisperse sedimentation model. In the second row, there is a simulation of bidisperse sedimentation.

Figure 3 illustrates the simulation of the Burgers equation, where a rarefaction wave and a shock wave are observed, as expected. In the monodisperse model, the particles settle at the bottom within 15 seconds, resulting in a clear water column above 0.4 m. For bidisperse sedimentation ($\phi_1(z, 0) = 0.2$, $\phi_2(z, 0) = 0.3$, $d_1 = 4.96 \times 10^{-4}$, $d_2 = 2.25 \times 10^{-4}$, $\rho_1 = \rho_2$), the larger particles settle first, followed by the smaller ones, showing the stratification process. By the end of the simulation, the uppermost 0.2 m of the column is free of particles.

Below we show experimental results for computation times for the Burgers' equation, the monodisperse sedimentation model, and the polydisperse sedimentation model. In all cases, the same simulation time interval, domain, and CFL condition were considered for both CPU and GPU computations. All experiments were conducted using a CPU Intel(R) Core(TM) i7-9750H and a GPU GeForce GTX 1660 Ti Mobile. The slope of the log-log graph gives the exponent of the growth order of time as a function of input size.

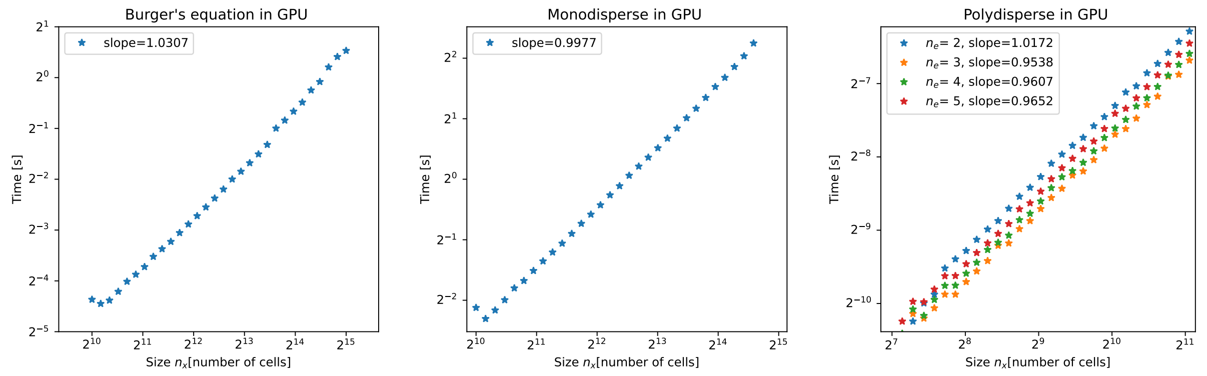

Figure 4. Loglog plot for parallel computation. For the polydisperse simulation we consider cases with 2-5 species.

The results obtained are consistent with the analytically derived computation times in Tables 1 and 2 and are summarized in Table 3.

Equation:	Burgers	Mono	Poly 2	Poly 3	Poly 4	Poly 5
Slope GPU:	1.0307	0.9977	1.0171	0.9538	0.9606	0.9651

Table 3. Slopes for the different equations for parallel case.

5. Conclusions

We have implemented a parallel version using CUDA of well-known numerical methods to calculate solutions to the Burgers' equation and the mono- and polydisperse sedimentation model. From the CFL condition, it is derived that the computation time depends only on the spatial discretization. We have provided analytical bounds for these implementations in their sequential and parallel versions (Tables 1 and 2), showing that the computation times decrease from quadratic to linear complexity. This represents a significant improvement, particularly for fine spatial discretizations. Finally, we present computational experiments showing that the obtained solutions align with expectations (Figure 3) and that the analytical time bounds are achieved in the parallel case (Table 3). This analysis identifies suitable approaches to efficiently and rapidly obtain numerical solutions, which is important in decision-making contexts. Fast simulations enable the anticipation of potential catastrophes and the design of effective policies to mitigate risks, especially when modeling physical problems applied to real contexts.

Acknowledgments VO is partially funded by UCM-IN-24200 internal grant.

References

[1] Bürger R, Fernández-Nieto E D and Osores V 2020 *Journal of Scientific Computing* **85** ISSN 1573-7691
[2] Careaga J and Osores V 2024 *Applied Mathematical Modelling* **134** 570–590 ISSN 0307-904X
[3] Betancourt F, Bürger R, Diehl S and Farås S 2014 *Minerals Engineering* **62** 91–101 ISSN 0892-6875
[4] JáJá J 1992 *An introduction to parallel algorithms* (USA: Addison Wesley Longman Publishing Co., Inc.) ISBN 0201548569
[5] NVIDIA, Vingelmann P and Fitzek F 2020 *CUDA, release: 10.2.89*
[6] LeVeque R J 1992 *Numerical Methods for Conservation Laws* (Birkhäuser Basel) ISBN 9783034886291
[7] Tory E M, Karlsen K H, Bürger R and Berres S 2003 *SIAM Journal on Applied Mathematics* **64** 41–80 ISSN 1095-712X
[8] Masliyah J H 1979 *Chemical Engineering Science* **34** 1166–1168 ISSN 0009-2509
[9] Lockett M and Bassoon K 1979 *Powder Technology* **24** 1–7 ISSN 0032-5910
[10] Bürger R, García A, Karlsen K H and Towers J D 2007 *Journal of Engineering Mathematics* **60** 387–425 ISSN 1573-2703