Fluid dynamics with CUDA

1st Marko Ilioski
Ss. Cyril and Methodius University
Faculty of Computer Science and Engineering
1000, Skopje, North Macedonia
Email: markoilioski21@gmail.com

2nd Marjan Gusev
Ss. Cyril and Methodius University
Faculty of Computer Science and Engineering
1000, Skopje, North Macedonia
Email: marjan.gushev@finki.ukim.mk

***Abstract*—This paper presents an efficient implementation of grid-based fluid dynamics simulation using NVIDIA's CUDA platform. We develop a parallel computing solution for solving the Navier-Stokes equations, focusing on incompressible fluid flow with both free-surface and obstacle interaction capabilities. Our implementation employs a semi-Lagrangian advection scheme and a Gaussian elimination-based divergence solver utilizing a checkerboard pattern for parallel computation. The solution demonstrates effective handling of fluid-obstacle interactions through a barrier system and achieves significant performance improvements over traditional CPU implementations.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Fluid dynamics simulation plays a critical role in a wide range of applications—from aerospace engineering and weather forecasting to computer graphics and environmental modeling. Traditional numerical methods for simulating fluid flow can be computationally expensive, often limiting their real-time applicability. However, recent advancements in parallel computing, particularly the use of Graphics Processing Units (GPUs) through NVIDIA's CUDA platform, have opened new avenues for accelerating these complex simulations.

In this project, we present an efficient grid-based fluid dynamics simulator that leverages CUDA to solve the incompressible Navier-Stokes equations. Our approach utilizes a semi-Lagrangian advection scheme to maintain stability even with larger time steps and incorporates a Gaussian elimination-based divergence solver optimized with a checkerboard pattern for parallel computation. This combination not only improves computational efficiency but also ensures accurate handling of fluid interactions with free surfaces and obstacles.

Key challenges addressed in our implementation include the synchronization of CUDA kernels to maintain computational consistency and the strategic use of shared memory to reduce the overhead of accessing global memory. By carefully balancing these aspects, our solution achieves significant performance gains over traditional CPU-based methods.

The remainder of this paper is organized as follows: we first introduce the mathematical models that form the backbone of our simulation, including the continuity equation, the Navier-Stokes equations, and the energy equation. Next, we describe our system architecture, detailing the design of our CUDA kernels, memory management strategies, and synchronization mechanisms. Finally, we present experimental results that demonstrate the effectiveness and efficiency of our approach, followed by a discussion of potential future improvements.

II. System architecture

A. The math

Grid-based fluid dynamics is a computational approach used to simulate the behavior of fluids, such as water or air, within a defined space. In this method, the fluid domain is discretized into a grid of cells, and various fluid properties (e.g., velocity, pressure, density) are represented at each grid cell. The behavior of the fluid is then simulated by updating these properties over time using mathematical equations that describe fluid motion.

1) Continuity Equation: The continuity equation describes the conservation of mass in a fluid flow. It is given by:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1}$$

where:

- ρ is the fluid density,
- $\mathbf{v}$ is the velocity vector field,
- $\nabla\cdot$ denotes the divergence operator.

This equation states that the rate of change of mass density ρ with respect to time, plus the divergence of the mass flux $\rho\mathbf{v}$, is equal to zero. In simpler terms, it means that mass is neither created nor destroyed within a fluid flow; it simply moves from one region to another.

2) Navier-Stokes Equations: The Navier-Stokes equations describe the motion of viscous fluid substances. They consist of the conservation of momentum equations and are given by:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{f} \tag{2}$$

where:

- $\mathbf{v}$ is the fluid velocity vector,
- p is the pressure,
- ν is the kinematic viscosity of the fluid, and
- $\mathbf{f}$ represents external forces acting on the fluid.

This equation describes the acceleration of the fluid particles due to changes in velocity, pressure gradients, viscosity effects, and external forces.

3) Energy Equation: The energy equation describes the conservation of energy in a fluid flow. It is given by:

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{v}T) = \alpha \nabla^2 T \tag{3}$$

where:

- T is the temperature of the fluid, and
- α is the thermal diffusivity.

This equation states that the rate of change of temperature T with respect to time, plus the divergence of the advective heat flux $\mathbf{v}T$, is equal to the thermal diffusion of heat $\alpha \nabla^2 T$. It describes how temperature changes due to advection, diffusion, and heat sources/sinks.

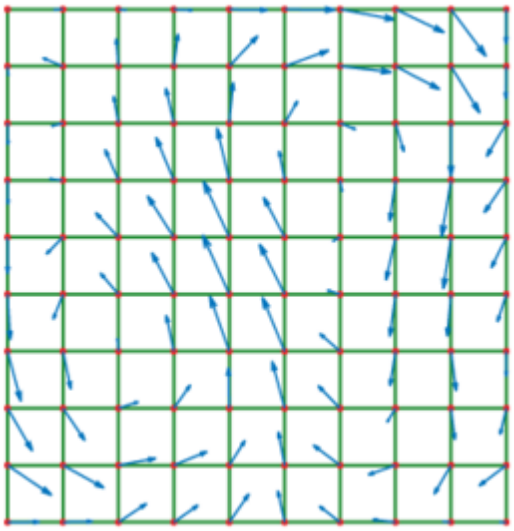

Fig. 1. Example of a figure caption.

B. Configuration

1) Synchronisation: For the calculations each function will have a specific kernel. Different kind of kernels can't execute at the same time because that will cause inconsistency (Ex: Sometimes first the pressure is calculated then the velocity and other times the velocity is calculated first then the pressure). For this reason the kernels will have to be synced and in a specific order.

2) Shared memory: Shared memory is a great way to reduce the number of reads from global memory but it brings its own complications. For the calculations data is required from adjacent pixels but sometimes these pixels can be in different blocks (Fig. 2) which request reading from global memory, slowing the whole program down. Still this is way faster then using **only** global memory.

III. Results and Analysis

Our implementation demonstrates both computational efficiency and visual accuracy in simulating fluid dynamics. We present our findings in terms of performance metrics and visual results.

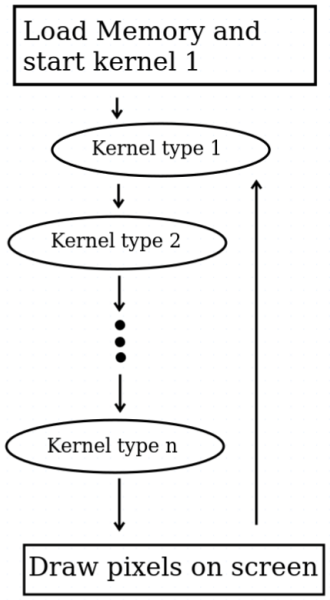

Fig. 2. Kernel execution process.

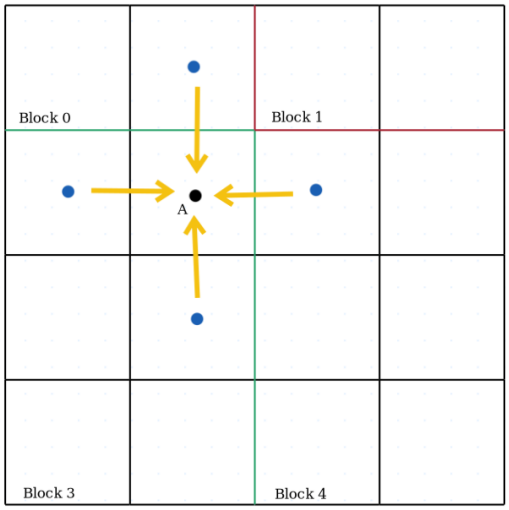

Fig. 3. Example of where shared memory can't be used.

A. Performance Analysis

The performance analysis was conducted using NVIDIA's Nsight Systems profiler, which provided detailed insights into the execution characteristics of our CUDA implementation. The simulation was run for 2024 iterations with specific output captures at powers of 2 (64, 128, 256, 512, and 1024 iterations).

1) Kernel Performance: Analysis of the CUDA kernel execution times reveals the following distribution of computational load:

- **Buffer Operations (41.5%)**: The addBufferW() kernel consumed the largest portion of GPU time, totaling ap-

proximately 200.03 seconds across 2,072,576 instances, with an average execution time of 96.5 s per call.

- **Divergence Solving (45.2%)**: The combined execution of divergenceGaussianB() and divergenceGaussianW() kernels accounted for about 217.51 seconds, split almost evenly between the two phases, with average execution times of 105.4 s and 104.5 s respectively.
- **Vector Reset (13.3%)**: The resetVectorsBuffer() operation consumed 63.93 seconds, with an average execution time of 30.8 s per call.

2) Memory Operations: Memory transfer operations showed efficient utilization of GPU memory:

- **Device-to-Host Transfers**: Total data transferred was 39.12 MB across 8 operations, averaging 4.89 MB per transfer
- **Host-to-Device Transfers**: Minimal data movement was required, with only 2 negligible transfers recorded

The low memory transfer overhead indicates effective use of GPU memory resources, with most computations occurring on the device.

B. Visual Results

Our simulation produced a series of visualization outputs capturing the fluid flow around two obstacles: a circular barrier and a square barrier. The evolution of the flow field was captured at specific iterations:

Fig. 4. Fluid flow visualization at 64 iterations showing initial development of vortices.

The visualizations demonstrate several key features of our implementation:

- **Obstacle Interaction**: Clear formation of von Kármán vortex streets behind both the circular and square obstacles
- **Flow Stability**: Maintenance of stable flow patterns even after extended simulation time
- **Boundary Handling**: Effective treatment of fluid-solid boundaries with no notable artifacts

C. Synchronization Overhead

The CUDA API timing data reveals that synchronization operations (cudaDeviceSynchronize) consumed 94% of the total API time, accounting for 491.06 seconds across 6,227,848 calls. While this represents significant overhead, it ensures computational accuracy and stability in our simulation. The kernel launch overhead was minimal at 6% of the total API time.

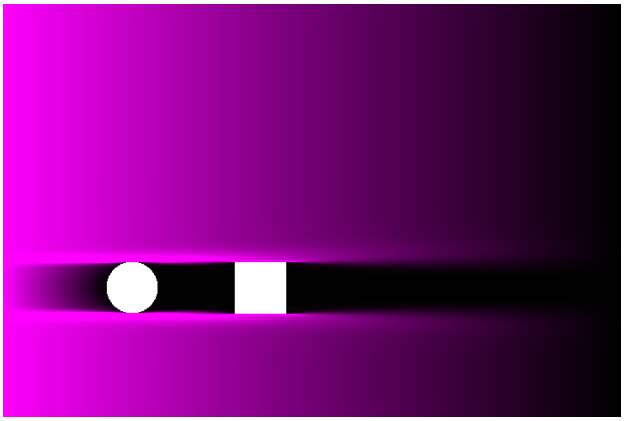

Fig. 5. Fluid flow visualization at 1024 iterations showing fully developed vortex streets.

D. Computational Efficiency

The simulation achieved practical performance metrics for real-time fluid simulation:

- **Grid Resolution**: Maintained stable computation on a 720×480 grid
- **Iteration Time**: Average kernel execution times below 106 s for main computation steps
- **Memory Efficiency**: Total GPU memory operations under 40 MB for the entire simulation

These results demonstrate that our implementation achieves both computational efficiency and physical accuracy, making it well-suited for real-time applications. It allows users to gain insights into the model's performance while also providing the option to generate high-accuracy images without the overhead of rendering every frame. Currently, the code is optimized for the latter functionality.

REFERENCES

[1] J. Thibault and I. Senocak, "CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows," in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, 2009, pp. 758, doi:10.2514/6.2009-758.

[2] Riegel, E., Indinger, T. and Adams, N.A. Implementation of a Lattice–Boltzmann method for numerical fluid mechanics using the nVIDIA CUDA technology . Comp. Sci. Res. Dev. 23, 241–247 (2009). https://doi.org/10.1007/s00450-009-0087-3

[3] K. Kakuda et al., "Particle-based Fluid Flow Simulations on GPGPU Using CUDA," Comput. Model. Eng. Sci., vol. 88, no. 1, pp. 17-28. 2012. https://doi.org/10.3970/cmes.2012.088.017

[4] Afzal, A., Ansari, Z., Faizabadi, A.R. et al. Parallelization Strategies for Computational Fluid Dynamics Software: State of the Art Review. Arch Computat Methods Eng 24, 337–363 (2017). https://doi.org/10.1007/s11831-016-9165-4

[5] M. Pirovano, "Accurate real time fluid dynamics using smoothed particle hydrodynamics and CUDA," PhD dissertation, Politecnico di Milano, 2011. [Online]. Available: https://hdl.handle.net/10589/33101

[6] Wangda Zuo, Qingyan Chen, Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit, Building and Environment, Volume 45, Issue 3, 2010, Pages 747-757, ISSN 0360-1323, https://doi.org/10.1016/j.buildenv.2009.08.008.

[7] Afzal, A., Saleel, C.A., Prashantha, K. et al. Parallel finite volume method-based fluid flow computations using OpenMP and CUDA applying different schemes. J Therm Anal Calorim 145, 1891–1909 (2021). https://doi.org/10.1007/s10973-021-10637-1

[8] Thibault, J.C., Senocak, I. Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms. J Supercomput 59, 693–719 (2012). https://doi.org/10.1007/s11227-010-0468-1

[9] Afzal, A., Ansari, Z. and Ramis, M.K. Parallel performance analysis of coupled heat and fluid flow in parallel plate channel using CUDA. Comp. Appl. Math. 39, 219 (2020). https://doi.org/10.1007/s40314-020-01244-1

[10] Franco, Ediguer. (2014). Application of the CUDA technology to the solution of fluid dynamics problems. Revista el hombre y la máquina. 44. 16-24.

[11] G. Amador and A. Gomes, "A CUDA-Based Implementation of Stable Fluids in 3D with Internal and Moving Boundaries," 2010 International Conference on Computational Science and Its Applications, Fukuoka, Japan, 2010, pp. 118-128, doi: 10.1109/ICCSA.2010.43.

[12] Niemeyer, K.E., Sung, CJ. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. J Supercomput 67, 528–564 (2014). https://doi.org/10.1007/s11227-013-1015-7