Proceedings of the Combustion Institute 40 (2024) 105512

Contents lists available at ScienceDirect

Proceedings of the Combustion Institute

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

An integrated framework for accelerating reactive flow simulation using GPU and machine learning models

Runze Mao [a], Min Zhang [a,b], Yingrui Wang [c], Han Li [a,b], Jiayang Xu [b], Xinyu Dong [a], Yan Zhang [d,e], Zhi X. Chen [a,b,*]

[a] State Key Laboratory of Turbulence and Complex Systems, College of Engineering, Peking University, Beijing, 100871, China
[b] AI for Science Institute (AISI), Beijing, 100080, China
[c] Shanghai SenseTime Intelligent Technology Co., LTD, Shanghai, 200233, China
[d] CAEP Software Center for High Performance Numerical Simulation, Beijing 100088, China
[e] Institute of Applied Physics and Computational Mathematics, Beijing 100088, China

ARTICLE INFO

Keywords:
Compressible reacting flow
GPU acceleration
HPC
Machine learning
Chemical kinetics
LES
Quasi-DNS

ABSTRACT

Recent progress in machine learning (ML) and high-performance computing (HPC) have brought potentially game-changing opportunities in accelerating reactive flow simulations. In this study, we introduce an open-source computational fluid dynamics (CFD) framework that integrates the strengths of ML and graphics processing unit (GPU) to demonstrate their combined capability. Within this framework, all computational operations are solely executed on GPU, including ML-accelerated chemistry integration, fully-implicit solving of fluid transport PDEs, and computation of thermal and transport properties, thereby eliminating the CPU–GPU memory copy overhead. Optimisations both within the kernel functions and during the kernel launch process are conducted to enhance computational performance. Strategies such as static data reorganisation and dynamic data allocation are adopted to reduce the GPU memory footprint. The computational performance is evaluated in two turbulent flame benchmarks using quasi-DNS and LES modelling, respectively. Remarkably, while maintaining a similar level of accuracy to the conventional CPU/implicit ODE-based solver, the GPU/ML-accelerated approach shows an overall speedup of over two orders of magnitude for both cases. This result highlights that high-fidelity turbulent combustion simulation with finite-rate chemistry that requires normally hundreds of CPUs can now be performed on portable devices such as laptops with a medium-end GPU.

1. Introduction

The imperative to decarbonise combustion systems has escalated due to the pressing need to mitigate climate change issues. To further advance reacting flow systems in the energy transition era, the development of high-fidelity numerical tools has become a central topic [1]. However, modelling methods that involve the direct interplay of turbulent flow and detailed chemistry still pose formidable costs. The majority of the computational expense is attributed to stiff chemistry and multi-species transport [2], which require game-changing acceleration techniques, at both software and hardware levels.

Recent developments in machine learning (ML) and high-performance computing (HPC) have brought inspirations for the broad scientific computing community [3]. ML have emerged as a new, efficient, powerful and easy-to-use modelling paradigm. On the hardware side, the graphics processing units (GPUs) have become a driving force for the development of computing infrastructure [4]. For combustion research, prior efforts attempting to accommodate these new techniques mainly focused on accelerating the most time-consuming chemical reaction rate integration. Leveraging ML algorithms, many groups have attempted neural networks to predict chemical kinetics [5–7]. From a hardware perspective, it is also natural to accelerate the ordinary differential equations (ODEs) using GPUs, which offer enhanced performance for these communication-free computations [8–11].

To facilitate the integration of ML and HPC in reactive flow simulation, we recently developed an open-source platform DeepFlame [12,13] , based on a code coupling of OpenFOAM, Cantera, and Torch libraries. Using DeepFlame, we demonstrated that when the chemical source term is computed through the inference of deep neural networks (DNNs, developed in [6]) on GPU, as compared to the conventional iterative approach of solving stiff chemical ODEs, a significant speedup of two orders of magnitude was achieved, even just for a small mechanism with 9 species and 12 reactions.

* Corresponding author at: State Key Laboratory of Turbulence and Complex Systems, College of Engineering, Peking University, Beijing, 100871, China.
E-mail address: chenzhi@pku.edu.cn (Z.X. Chen).

https://doi.org/10.1016/j.proci.2024.105512
Received 4 December 2023; Accepted 20 June 2024
Available online 31 July 2024

However, currently not all computational procedures in reactive CFD can be reliably accelerated by ML-driven models. The flow and species transport is governed by a system of partial differential equations (PDEs). Novel data-driven methods such as physics informed neural network (PINN) [14] seems promising but yet to mature for complex geometry and reactive flows. In the pursuit of speeding up the PDE computations, HPC hardware accelerators, particularly GPUs, have received significant attention within the combustion community [15–18]. This can be mainly attributed to two key factors: (i) GPU is proven to be advantageous for CFD due to its inherent data parallelism in calculations, i.e. the per-cell parallelism in finite volume method (FVM) [2]; and (ii) GPU features a greater number of computing units with simplified control mechanisms within each chip, affording powerful computing capacity and high throughput.

Le et al. [15] pioneered the development of one of the first reactive-flow GPU solvers. Their work demonstrated a maximum speedup factor 40x compared to a single CPU thread. Levesque et al. [16] successfully ported S3D to GPUs, utilising OpenACC directives in conjunction with MPI and OpenMP to achieve an architecture-agnostic, multi-level parallelism. Recently, Uranakara et al. [17] achieved a maximum speedup factor 7x by integrating the GPU-based chemical solver UM-ChemGPU [10] into the GPU-compatible DNS solver KARFS. Henry et al. [19] developed PeleC, an explicit GPU-based CFD platform for combustion, exhibiting an impressive parallel scalability extending to the entire Summit supercomputer.

Despite these notable advances, several challenges persist. First, existing studies have focused on explicit methods for flow transport terms. It is well-known that implicit FVM is of practical importance for low Mach-numbers flows since the computational efficiency is not constrained by the Courant number limit. However, implicit methods require additional matrix discretisation and solving linear equations, both are non-trivial for GPU implementation. Second, an integrated framework to accommodate both ML models and GPU-like accelerator hardware is still absent and their combined performance is yet be demonstrated.

In this work, we present a GPU porting and performance study of a fully implicit solver called *dfLowMachFoam* in the DeepFlame open-source framework [13]. The implementation, based on CUDA kernel functions, closely mirrors OpenFOAM, including the discretisation of conservation equations and the assembly of sparse linear matrices for implicit solving. The GPU-accelerated sparse linear solver, AmgX by NVIDIA [20], is employed to solve the implicit linear system. In addition, various optimisations are implemented to enhance computational performance and reduce GPU memory footprint. Chemistry integration is performed via inferencing pretrained DNN models. Two case studies are presented: (i) quasi-DNS of a turbulent H_2-air diffusion flame and, (ii) large eddy simulation (LES) of a lab-scale turbulent stratified CH_4-air premixed flame. Significant acceleration over two orders of magnitude is achieved for both cases using GPU with DNN and detailed discussion is provided.

The remainder of this paper is organised as follows. In Section 2, we present the theoretical models governing fluid dynamics and chemistry in this study. Section 3 introduces the implementation of the GPU-based solver and outlines the primary optimisations undertaken. Two validation cases, along with a presentation of computational performance, are detailed in Section 4. Finally, the conclusions are summarised in Section 5.

2. Modelling methodology

The solver introduced here uses a fully implicit pressure-based solving procedure for low-speed reactive flows with the pressure-implicit split-operator (PISO) algorithm. As per OpenFOAM's standard numeric, the solver employs a cell-centred finite volume scheme with second-order accuracy for discretising the conservation equations and up to fourth-order-accurate for the face flux interpolation [21]. Detailed chemical kinetics and molecular transport models are implemented via the Cantera interface. When the mesh resolves the smallest flow and chemical scales, the modelling fidelity at the quasi-DNS level can be achieved. For the LES case presented later, subgrid scale (SGS) closure models are required. The Smagorinsky model is used for the SGS turbulence, and the partially-stirred reactor (PaSR) model [22] is chosen to account for the SGS turbulence-chemistry interaction.

The chemical reaction rate integration can be performed using two options: CVODE [23] on CPU or DNN on GPU. Detailed validation for the accuracy and generalisation ability of the DNN method is available in [13]. In this study, we employed individual neural networks to predict the evolution of each species, excluding inert gases. The network features three hidden layers comprising 1600, 800, and 400 perceptrons, respectively. The input layer parameters include temperature, pressure and mass fractions, while the output is the rate of change for a given species. Uniform across all networks, the Gaussian Error Linear Unit (GELU) is chosen as the activation function, and the Adam algorithm is used for hyper-parameter optimisation. Additionally, the precision of the weights and activation functions of the DNN model is set to FP16 (*half* precision) to decrease the memory footprint and enhance performance.

3. Implementation and optimisations

GPUs provide much higher compute intensity than CPUs, and are widely used in many scientific computing fields. CUDA is a general-purpose parallel computing platform and programming model developed by NVIDIA, which aims to solve complex computational problems on NVIDIA GPUs more flexibly and efficiently. Porting a program from CPU to GPU is a complex system engineering task, involving data migration and management, and appropriate parallelisation of each computation step. Optimisations on GPU are also complex processes involving multiple aspects, such as hardware architecture, programming model and computation features. Based on these fundamental knowledge, the implementations and optimisations conducted in this study are discussed in subsequent sections.

3.1. GPU implementation

In this work, the primary objective of GPU porting is to migrate a significant portion of computational operations to the GPU while minimising data transfer between the GPU and CPU. To realise this objective, the internal computational workflow and code structure of the GPU-based solver are illustrated in Fig. 1. Operations highlighted in orange denote computations conducted on the GPU, while those in blue signify CPU-based processes. The functions within the shadowed box aligned with the specified computation process represent the corresponding sections in the code.

As illustrated in Fig. 1, it is important to note that the CPU is exclusively engaged in initialising data at the initial time step and writing simulation results at the conclusion. Once the initial data is transferred to the GPU, all the computation required for time advancement are solely performed there. Basically, the operations on GPU can be categorised into three typical types: (i) Solving the PDEs: Five distinct PDEs are encompassed in the solver for resolving conservation equations related to mass, momentum, species, energy, and pressure. We employ specialised classes to tackle each PDE. (ii) Solving the chemical source. As introduced in Section 2, DNN method is adopted here for chemistry integration. The related operations are encapsulated in class *DNNInference* and implemented based on libTorch. (iii) Calculating the thermophysical and transport property. These operations involve Newton's method and high-order temperature polynomials, conducted by the class *Thermo* with CUDA functions.

Among these operations, the implementation of implicit solving for PDEs stands out for its high complexity. In contrast to the explicit

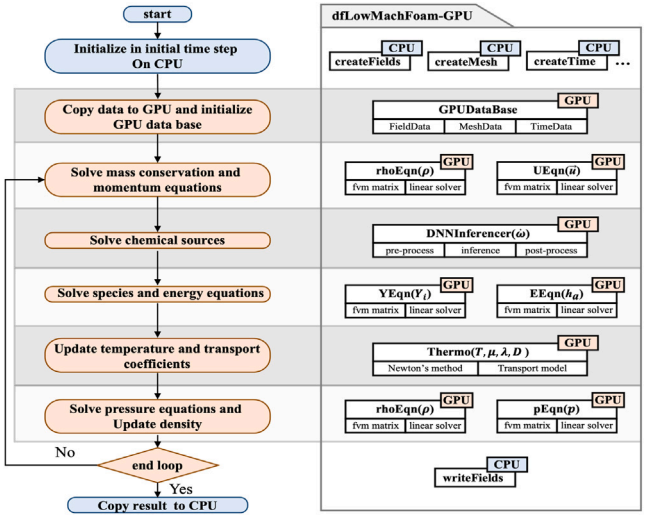

Fig. 1. Computational workflow and code structure of GPU-based solver *dfLowMachFoam*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Algorithm 1 implicit discretisation of Laplacian term

1: <**GPU kernel begin**>{internal field}
2: faceid←threadIdx.x+blockIdx.x×blockDim.x
3: $\gamma_f \leftarrow$ interpolation(w, γ_c)
4: $upper,\ lower \leftarrow \gamma_f \delta_f S_f$
5: **atomicAdd**$(diag[ownCellIndex[\text{faceid}]], -upper[\text{faceid}])$
6: **atomicAdd**$(diag[neighborCellIndex[\text{faceid}]], -lower[\text{faceid}])$
7: <**GPU kernel end**>
8: **for** int bouid=1 to N_{bou} **do**
9: <**GPU kernel begin**>{boundary field}
10: $internalCoeffs, boundaryCoeffs \leftarrow$ updateBouCoeffs$(bouType, \gamma_{bou}, S_{bou})$
11: <**GPU kernel end**>
12: **end for**

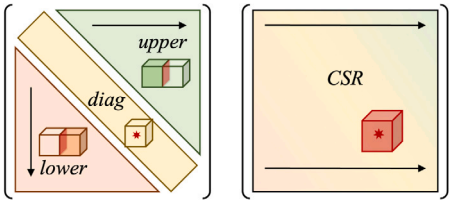

Fig. 2. Schematic of the *ldu* and *CSR* formats for sparse matrices. The red colour highlights the mesh structure associated with the matrix component, with colour intensity indicating the relationship between faces and cells. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

method, implicit FVM introduces an additional sparse matrix, representing a linear system obtained through discretisation. In OpenFOAM, the sparse matrix is stored in *ldu* format, where the lower, diagonal, and upper parts are managed individually, as depicted schematically on the left in Fig. 2. The *ldu* format is introduced to ensure good data locality in matrix assembly, taking into account the inherent relationship between the matrix and mesh structure (see Fig. 2). So it is also employed in our work for matrix assembly.

Providing a more in-depth look into the matrix assembly implementation in our work, the algorithm for discretising the term $\nabla \cdot (\gamma \nabla \psi)$ is outlined in Algo. 1. Here, γ represents the coefficient of the Laplacian term, and ψ is the unknown field. The subscripts $_f$, $_c$, and $_{bou}$ denote the face, cell, and boundary field, respectively. As seen in line 4 of Algo. 1, the upper and lower entries are directly obtained through the multiplication of face values. However, computing the diagonal part necessitates the accumulation of face terms. In the GPU implementation, this poses the challenge of potential simultaneous writes to the same memory location, known as a *race condition*. To address this issue, we employ *atomicAdd* in lines 5 to 6 to calculate the diagonal entries, a method proven to exhibit better GPU performance in 3D FVM compared to the general *graph colouring method*.

Handling operations related to boundary fields is detailed in line 8 to 12 of Algo. 1. Notably, we iterate through the boundary patches and implement distinct kernel functions based on the patch types. This approach accommodates the diverse requirements for various boundary conditions. The multiple kernel launches introduced by the patch loop can be effectively mitigated by leveraging CUDA *graph*, as will be introduced in Section 3.2.

While the *ldu* format proves suitable for matrix assembly, its performance becomes an issue in *sparse matrix–vector products* (SPMVs), the primary operation in solving linear system. This is mainly due to the non-sequential memory access patterns of the off-diagonal matrix parts. Therefore, in our implementation, we convert the matrix from *ldu* format to *compressed sparse row (CSR)* format (where values are accessed per line, see Fig. 2), and subsequently utilise the AmgX library to solve the linear system.

3.2. Optimisation

In this section, we report a series of optimisations to enhance computational performance and minimise the memory footprint on the GPU. Computational efficiency receives a boost through improvements in the kernel function, kernel launch process, and the parallelism of multiple GPUs. Simultaneously, GPU memory usage is curtailed through static data reorganisation and dynamic data allocation approach.

The initial focus is on the optimisations applied to kernel functions. Essentially, algorithms can be classified into memory-bound and compute-bound categories, with their performance mainly dictated by memory access and elementary computational steps, respectively. In this study, discretisation typically display a memory-bound characteristic due to their simple calculations and small stencil sizes. Conversely, operations related to calculating thermophysical and transport properties exhibit a higher arithmetic intensity, categorising them as compute-bound.

For the memory-bound kernels, the paramount optimisation strategy is coalesced data, which serves to augment data locality and reduce data retrieval overhead. In the CPU code, fields undergo operations via cell/face loops, with fields containing multiple components (such as vector fields, tensor fields, and species fields) stored along the component dimension for contiguous data access. However, this data structure disrupts data locality on the GPU. Consequently, we have coalesced the same components of fields to minimise memory access transactions, leading to a significant improvement in bandwidth utilisation efficiency. For the compute-bound kernels, there are more effective methodologies available for enhancing the computational performance. Firstly, strategic optimisations have been implemented, featuring computation consolidation and storage substitution. These endeavours alleviate the impact of long-latency computational instructions, particularly the functions such as $sqrt()$ and $pow()$. Moreover, for the intricate computational kernels, we have adopted constant memory for storing constant coefficients and shared memory for mass and mole fractions, thereby reducing the dependence on global memory access. Additionally, we have partitioned these heavy kernels into 2 or 3 segments, effectively minimising the register usage of each kernel and achieving higher occupancy.

In addition to optimising the kernels, we have taken steps to minimise the overhead associated with kernel launches by leveraging CUDA graphs. Conventionally, each execution of a GPU kernel requires

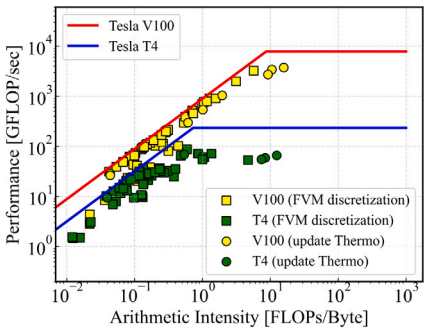

Fig. 3. Roofline model for V100 and T4 GPUs and mesh size of 128^3.

a submission operation on the CPU, thereby incurring supplementary overhead. Given the persistent nature of the discretisation procedure across each time step, the pertinent kernels can be encapsulated within a CUDA graph. This enables a singular CPU submission to initiate a suite of kernels on the GPU for each time step. This approach markedly augments performance in the discretisation procedure. Furthermore, we have opted for the NVIDIA Collective Communication Library (NCCL) over MPI for achieving multi-processor parallelism. Although MPI features better portability, NCCL supports peer-to-peer communication between GPUs, facilitating direct data exchange between GPUs via PCIe or NVLink. This eliminates the need for CPU–GPU data transfers and further streamlines the communication process.

Finally, we introduce the GPU memory management strategy here. Initially, in comparison to the CPU implementation, we reorganise the static data by heightening the granularity of operations, thereby eliminating the storage of intermediate fields in memory. For instance, as delineated in line 3 of Algo. 1, the interpolation of cell fields is no longer an independent procedure but has been seamlessly integrated into the Laplacian operation. This strategic consolidation of operations effectively mitigates the burden on GPU memory. Moreover, data exclusively utilised in each operation – such as the field of ∇U for the *UEqn* class – is allocated and freed during each time step. Such the dynamic data allocation is implemented by the CUDA stream-ordered allocator, which manages memory as stream-ordered operations, and thus circumventing resource-intensive device-wide synchronisation. This approach substantially reduces memory overhead with minimal impact on performance.

3.3. Performance

First, the performance of the optimised kernels in simulating the reactive TGV case with mesh size of 128^3 is demonstrated using the roofline model, as shown in Fig. 3. It is evident that all kernels achieve commendable efficiency, closely approaching the hardware limit (depicted by the solid line) on both NVIDIA Tesla V100 and T4 platforms. Specifically, the kernels associated with implicit FVM discretisation are marked by squares. These kernels primarily exhibit a sparse nature and are memory-bound, positioning them on the left half of the roofline model curve. Conversely, the operations related to the thermophysical update procedure, marked by circles, involve high-order polynomials and are computation-intensive, typically appearing on the right half of the curve.

The overall optimised simulation performance is analysed through reactive TGV simulations with varying grid resolutions. These tests were conducted on a single CPU core and one GPU card. The CPU used in this study is an AMD ZEN1, features a peak performance of 320 GFLOPS and a bandwidth of 147 GB/s. The GPU employed is a Tesla V100. As shown in Table 1, the simulation time achieves a maximum speedup of approximately 310 at the largest mesh size. Additionally, a detailed performance breakdown of this 310x acceleration is illustrated

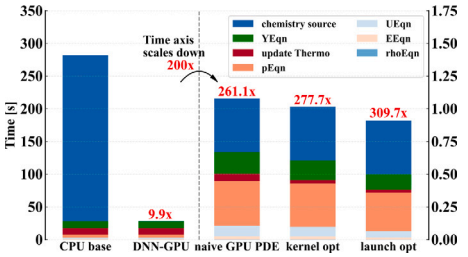

Fig. 4. Step-by-step performance improvement of reactive hydrogen TGV case with varying grid size on one GPU card.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

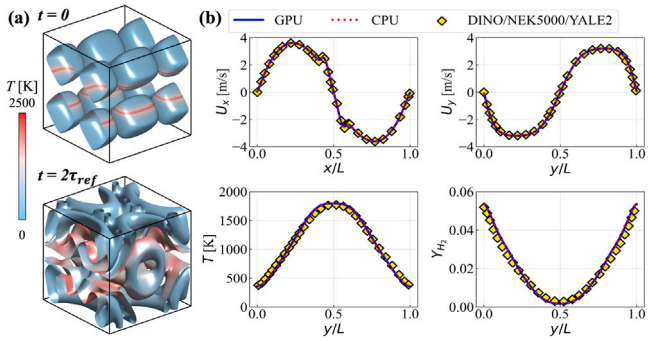

Fig. 5. (a) The temperature and Q-criterion field at $t = 0$ and $t = 2\tau_{ref}$. (b) Compares of U_x, U_y, T, and Y_{H_2} at $t = 2\tau_{ref}$.

in Fig. 4. It is evident that the chemical DNN, GPU porting, and the optimisation of kernel functions and launches all significantly contribute to the overall speedup. Further analysis of performance in simulating this TGV case is discussed in Section 4.1.

4. Validation and computational performance

4.1. Quasi-DNS of 3D reactive Taylor–Green vortex

As a recently established benchmark [24], the 3D TGV interacting with a H_2 diffusion flame has been widely adopted for code verification and validation [21]. The cubic computational domain with an edge length of $2\pi L$ ($L = 1$ mm, $Re \approx 250$) is uniformly discretised with 256^3 cells. The benchmark specified chemical mechanism has 9 species and 12 reversible reactions [25]. Fig. 5a depicts the TGV evolution of Q-criterion coloured by temperature from the initial field to $t = 2\tau_{ref}$ (the flow reference time is $\tau_{ref} = L/u_0$). The initial maximum velocity $u_0 = 4$ m/s and further details regarding the initial setup can be found in [21]. We conducted two simulations of reactive TGV to provide a direct comparison as shown in Fig. 5b, where GPU indicates full simulation on GPU including the flow and DNN chemistry. In contrast, the CPU simulation follows a standard OpenFOAM procedure with the chemistry solved using CVODE provided by Cantera. As seen, an excellent agreement is obtained between the GPU and CPU simulations, also very close to the reference results from the high-order DNS codes [21]. This suggests that our GPU implementation is consistent with the original CPU code and the DNN model coupled with the CFD solver is of good accuracy for complex reactive flow simulation.

The GPU speed-up and memory footprint in the 3D reactive TGV case are analysed and discussed. The CPU simulation is conducted on one CPU chip with 32 processing cores, equivalent to 32 MPI ranks. The GPU results are simulated using 4 NVIDIA V100 GPUs. Note that this configuration is typical for one computing node today (1CPU+4GPU or 2CPU+8GPU). Fig. 6a presents the computational cost of each operation outlined in Fig. 1. Three cases are compared: (i)

Table 1
Comparison of computational performance for the H_2-air mechanism across various mesh size. Table lists the sum time per step, chemistry time per step, and fluid time per step for a single AMD CPU core and one V100 GPU card.

Mesh size	CPU sum [s]	GPU sum [s]	Overall speedup	CPU chemistry [s]	GPU chemistry [s]	Chemical ODE speedup	CPU fluid [s]	GPU fluid [s]	Fluid speedup
32^3	5.16	0.091	56.7	4.7	0.007	671.42	0.46	0.084	5.45
64^3	37.93	0.183	207.26	34.24	0.053	646.03	3.69	0.13	28.38
128^3	281.84	0.91	309.7	253.86	0.41	619.1	27.938	0.5	55.876

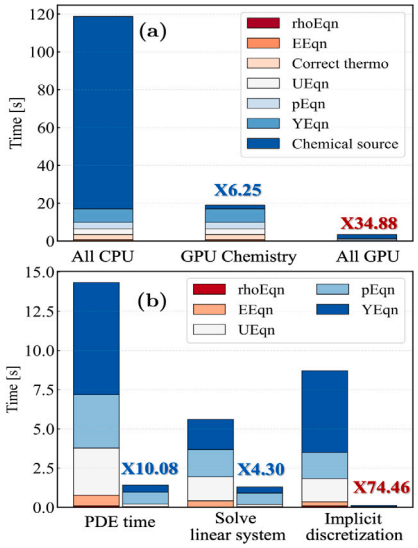

Fig. 6. Computational cost in simulating 3D reactive TGV in one time step. (a) Overview of the entire simulation process. (b) Breakdown of computational costs specifically for PDE solving components.

all operations including CVODE chemistry on CPU, (ii) flow on CPU with DNN chemistry on GPU, and (iii) all on GPU. For the all CPU case, it is as expected that the chemical source integration is the most time-consuming operation. With the excellent speed-up provided by DNN chemistry (demonstrated in Table 1), the overall simulation is accelerated by 6.25 times as compared to the all CPU case. After porting all the remaining operations (PDE and thermo/transport) to the GPU, the overall speedup further increases to nearly 35 times. This implies that for a quasi-DNS case with tens of millions of cells, 4 GPU cards within a small workstation at a much lower power cost can provide an equivalent performance to a supercomputer with thousands of CPUs.

Specifically, Fig. 6b details the computational cost for the PDE procedures, which is a key contribution of this work. Bars on the left and right denote CPU and GPU times, respectively. It can be seen that time is evenly distributed in implicit discretisation and solving the linear system when using the CPU. For the GPU solver, however, the operations of discretisation are implemented by CUDA kernels, ultimately achieving a speedup of 75 over the CPU. By contrast, the linear system solving function provided by the AmgX library shows relatively weaker acceleration, especially when solving the pressure Poisson equation. This can mainly be attributed to the fact that the AmgX library cannot access mesh information, limiting it to using the algebraic multigrid method (AMG), whereas the CPU solver adopts the more efficient geometry multigrid method (GAMG). Further optimisation on this implementation is required for better acceleration performance.

The memory footprint is of particular importance for GPU codes since the memory space is quite limited compared to the RAM size of CPU. For the 4 GPUs used for the TGV case, it is approximately 11 GB on each card and the detailed distribution is listed in Table 2. It is evident that the CFD data, including the mesh and field information, accounts for the majority of the memory, taking up about 44.3% of the total used. Notably, the memory footprint for storing CFD data has been significantly reduced by 2.27 times with the CUDA stream-ordered allocator. The AmgX handle and CUDA handle together consume 34.86% and 17.1% of GPU memory, respectively. These two components primarily consist of compiled kernel functions, CUDA context, and other CUDA resources. Therefore, these portions exhibit minimal growth with the size of the simulation case. The DNN inference procedure utilises minimal memory after optimising both the inference batch size and numerical precision. In summary, a reasonable GPU memory management strategy ensures that only about 45% of the total memory is allocated to the CFD data, and thus only this part of the GPU memory increases with the size of the system simulated, either via a larger mesh or a more complex chemical mechanism. Also, note that the memory consumption attributed to the AmgX handle will only exhibit a slight increase, while the memory allocations for the CUDA handle and the DNN inference remain consistent.

Table 2
The memory usage distribution for each GPU.

Memory usage	CFD data	AmgX handle	CUDA handle	DNN inference
Footprint [MB]	4978	3917	1919	418
Percentage	44.33%	34.86%	17.1%	3.72%

4.2. LES of cambridge stratified burner

The Cambridge burner is a well-known TNF Workshop benchmark for turbulent stratified premixed combustion, previously simulated by many research groups [26]. Thus, this configuration is chosen to further demonstrate the practical applicability of the proposed GPU/ML framework. The setup features a central bluff-body surrounded by two co-annular premixed CH_4-air mixture streams, with co-flowing air under ambient conditions. The moderately stratified SWB5 case is simulated. The respective velocities of the inner, outer, and air streams are 18.7 m/s, 8.3 m/s, and 0.4 m/s. The stratification is achieved by maintaining an equivalence ratio of 1.0 for the inner and 0.5 for the outer annular flows.

Fig. 7 presents the typical instantaneous temperature and CH_4 mass fraction fields obtained from the simulation. A reduced methane mechanism [27] with 20 species and 85 reactions is used. To encompass the typical range for common LES studies [26], we tested two non-uniform grids comprising 2.5 million and 10 million cells. The validation of the simulation results, particularly for the 2.5M-cell LES case, is detailed in Fig. 8. In this figure, mean values are depicted in black and RMS values in red. The notation CPU-DI refers to simulations performed using the OpenFOAM solver with the direct chemical integrator CVODE, implemented by Cantera. Experimental data referenced are sourced from [28]. The analysis in Fig. 8 examines the radial distribution of temperature and the concentrations of two major and minor species-methane (CH_4) and the methyl group (CH_3), respectively. It is evident that the results from CPU-DI and GPU-DNN simulations are nearly identical, and both exhibit good agreement with experimental data. Notably, there is a minor shift in the peak position within the RMS results, which is mainly due to the settings of turbulence and combustion model. Further refinements to precisely match the experimental results are beyond the scope of this study. In addition, the validation of this

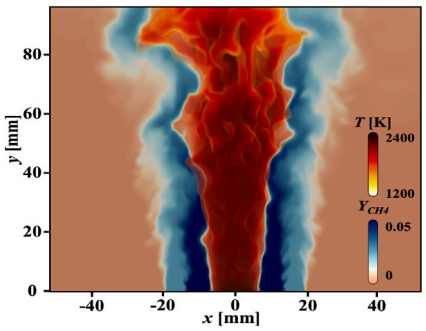

Fig. 7. Contour plot depicting the mass fraction of methane in the burner mid-section, superimposed with the rendering volume for temperature.

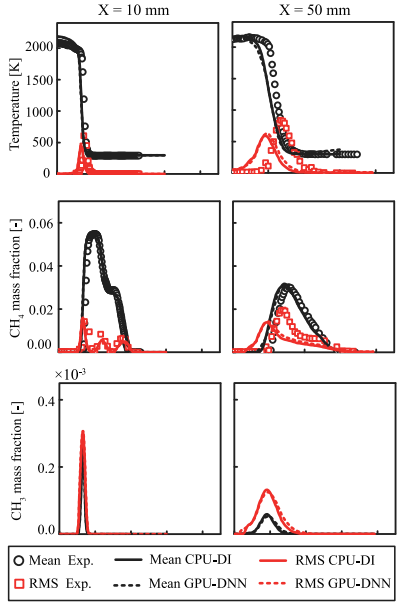

Fig. 8. Radial profiles of mean and RMS for temperature, CH_4 and CH_3 mass fraction at two different axial locations X = 10 mm, 50 mm in the experiment, CPU-DI, and GPU-DNN cases. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

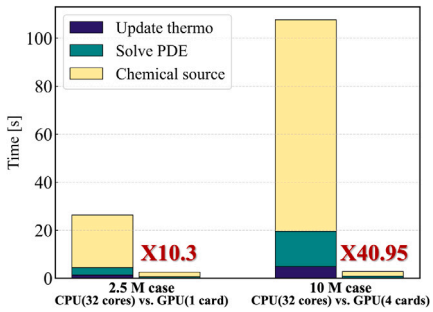

Fig. 9. One step computational cost in the LES case.

GPU-ML framework under more complex conditions such as swirling flows has been thoroughly addressed in [29].

To demonstrate the computational performance, Fig. 9 shows the time-to-solution comparison for the two cases. For the 2.5M case, which can be run quite easily using a single GPU, the speedup is 10 times compared to 32 CPU cores. For the 10M case, the simulation is accelerated by a factor of nearly 41 using 4 GPU cards. It is noteworthy that the performance here is improved from that of TGV (see Fig. 6) and this can be attributed to: (i) the larger mechanism of CH_4 makes the speedup provided by the DNN chemistry more pronounced; (ii) the reduced flow complexity (less vortical fields compared to TGV) relaxes the relatively weaker acceleration by AmgX's linear solving, especially for the pressure Poisson equation. However, a further implication here is that with the proposed GPU/ML tool, high-fidelity LES of turbulent combustion can be adequately performed on portable devices such as laptops with a consumer-level GPU at the speed of several-hundred core clusters.

5. Conclusions

In this work, we introduce an open-source numerical framework leveraging GPU and ML techniques to accelerate reactive flow simulations. To avoid the CPU–GPU memory copy overhead, the entire computational process is executed on GPU, including ML-accelerated chemistry integration, fully-implicit solving of PDEs, and computation of thermal and transport properties. Specifically, the FVM implicit discretisation of conservation equations and explicit computations are implemented through CUDA kernel functions. The multi-processor parallelism is achieved through NCCL, thereby enabling direct communication between GPUs. The linear system is solved using the NVIDIA-produced AmgX library, and ML-related operations are performed with the libTorch library. Various optimisations have been performed to enhance computational performance and reduce GPU memory footprint.

The capabilities of the framework are evaluated for handling two different turbulent flame test cases using quasi-DNS and LES modelling. While maintaining similar level of accuracy, for both cases the GPU/ML accelerated solver achieves an overall speed-up of over two orders of magnitude. These results underscore the immense potential of machine learning and GPU technologies in empowering reactive flow simulations. Looking ahead, plans are in place to adapt this framework for use across various GPU-like accelerator platforms, further broadening its applicability. In addition, the open-source nature of the proposed framework is expected to facilitate further exploration and collaboration on GPU code development, and bring together diverse machine learning models to become indeed useful for the combustion modelling community.

Novelty and Significance Statement

This work is one of the first attempts to evaluate the combined capability of the emerging machine learning and GPU technologies in accelerating reactive flow simulations. We propose an open-source framework with excellent support to ML and HPC infrastructure, which marks a pioneering contribution within the combustion community. The framework seamlessly executes on GPUs throughout the entire computational workflow, eliminating the common challenging issue associated with the CPU-GPU data transfer. To achieve this, a series of code migration efforts were undertaken, featuring key contributions such as the novel GPU-porting of an implicit finite volume method (FVM) PDE solver and the application of NCCL, instead of MPI, for multi-processor parallelism. With the accuracy validated against the state-of-the-art CPU codes, this framework achieves a significant speedup of over two orders of magnitude in two benchmark turbulent flame cases. The ML modelling approach, GPU porting techniques, as well as the open-source numerical codes will potentially make a significant impact in the combustion modelling community.

CRediT authorship contribution statement

Runze Mao: Code implementation, Performed research, Analysed data, Writing – original draft. **Min Zhang:** DNN development, Analysed data. **Yingrui Wang:** Code implementation, Writing – review & editing. **Han Li:** DNN development. **Jiayang Xu:** Analysed data. **Xinyu Dong:** Analysed data. **Yan Zhang:** Writing – review & editing. **Zhi X. Chen:** Supervision, Writing – review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 92270203, 52276096, and 523B2062), GHfund C, China (202302032372) and CCF-Baidu Open Fund, China.

References

[1] T. Poinsot, D. Veynante, Theoretical and Numerical Combustion, RT Edwards, Inc., 2005.
[2] K.E. Niemeyer, C.-J. Sung, Recent progress and challenges in exploiting graphics processors in computational fluid dynamics, J. Supercomput. 67 (2013) 528–564.
[3] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, et al., Scientific discovery in the age of artificial intelligence, Nature 620 (7972) (2023) 47–60.
[4] Top 500 supercomputers, https://www.top500.org/.
[5] K. Wan, C. Barnaud, L. Vervisch, P. Domingo, Chemistry reduction using machine learning trained from non-premixed micro-mixing modeling: Application to DNS of a syngas turbulent oxy-flame with side-wall effects, Combust. Flame 220 (2020) 119–129.
[6] T. Zhang, Y. Yi, Y. Xu, Z.X. Chen, Y. Zhang, E. Weinan, Z.-Q.J. Xu, A multi-scale sampling method for accurate and robust deep neural network to predict combustion chemical kinetics, Combust. Flame 245 (2022) 112319.
[7] T. Ding, T. Readshaw, S. Rigopoulos, W. Jones, Machine learning tabulation of thermochemistry in turbulent combustion: An approach based on hybrid flamelet/random data and multiple multilayer perceptrons, Combust. Flame 231 (2021) 111493.
[8] K. Spafford, J. Meredith, J. Vetter, J. Chen, R. Grout, R. Sankaran, Accelerating S3D: a GPGPU case study, in: Euro-Par 2009–Parallel Processing Workshops, Springer, 2010, pp. 122–131.
[9] Y. Shi, W.H. Green, H.-W. Wong, O.O. Oluwole, Redesigning combustion modeling algorithms for the graphics processing unit (GPU): Chemical kinetic rate evaluation and ordinary differential equation integration, Combust. Flame 158 (5) (2011) 836–847.
[10] S. Barwey, V. Raman, A neural network-inspired matrix formulation of chemical kinetics for acceleration on GPUs, Energies 14 (9) (2021) 2710.
[11] K.E. Niemeyer, N.J. Curtis, C.-J. Sung, pyJac: Analytical Jacobian generator for chemical kinetics, Comput. Phys. Comm. 215 (2017) 188–203.
[12] deepflame-dev, https://github.com/deepmodeling/deepflame-dev.
[13] R. Mao, M. Lin, Y. Zhang, T. Zhang, Z.-Q.J. Xu, Z.X. Chen, DeepFlame: A deep learning empowered open-source platform for reacting flow simulations, Comput. Phys. Comm. 291 (2023) 108842.
[14] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[15] H.P. Le, J.-L. Cambier, L.K. Cole, GPU-based flow simulation with detailed chemical kinetics, Comput. Phys. Comm. 184 (3) (2013) 596–606.
[16] J.M. Levesque, R. Sankaran, R. Grout, Hybridizing S3D into an exascale application using OpenACC: an approach for moving to multi-petaflops and beyond, in: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2012, pp. 1–11.
[17] H.A. Uranakara, S. Barwey, F.E.H. Pérez, V. Vijayarangan, V. Raman, H.G. Im, Accelerating turbulent reacting flow simulations on many-core/gpus using matrix-based kinetics, Proc. Combust. Inst. 39 (2023) 5127–5136.
[18] R. Bielawski, S. Barwey, S. Prakash, V. Raman, Highly-scalable GPU-accelerated compressible reacting flow solver for modeling high-speed flows, Comput. & Fluids (2023) 105972.
[19] M.T. Henry de Frahan, J.S. Rood, M.S. Day, H. Sitaraman, S. Yellapantula, B.A. Perry, R.W. Grout, A. Almgren, W. Zhang, J.B. Bell, et al., PeleC: An adaptive mesh refinement solver for compressible reacting flows, Int. J. High Perform. Comput. Appl. 37 (2) (2023) 115–131.
[20] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, R. Strzodka, AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods, SIAM J. Sci. Comput. 37 (2015) S602–S626.
[21] T. Zirwes, M. Sontheimer, F. Zhang, A. Abdelsamie, F.E.H. Pérez, O.T. Stein, H.G. Im, A. Kronenburg, H. Bockhorn, Assessment of numerical accuracy and parallel performance of OpenFOAM and its reacting flow extension EBIdnsFoam, Flow Turbul. Combust. 111 (2023) 567–602.
[22] M. Evans, C. Petre, P.R. Medwell, A. Parente, Generalisation of the eddy-dissipation concept for jet flames with low turbulence and low Damköhler number, Proc. Combust. Inst. 37 (4) (2019) 4497–4505.
[23] S.D. Cohen, A.C. Hindmarsh, CVODE User Guide, Tech. rep., Citeseer, 1994.
[24] A. Abdelsamie, G. Lartigue, C.E. Frouzakis, D. Thévenin, The Taylor–Green vortex as a benchmark for high-fidelity combustion simulations using low-Mach solvers, Comput. & Fluids 223 (2021).
[25] P. Boivin, C. Jiménez, A.L. Sánchez, F.A. Williams, An explicit reduced mechanism for H2–air combustion, Proc. Combust. Inst. 33 (1) (2011) 517–523.
[26] TNF workshop – International workshop on measurement and computation of turbulent nonpremixed flames. https://tnfworkshop.org/.
[27] F.M. Kazakov A, Reduced reaction sets based on GRI-Mech 1.2, http://www.me.berkeley.edu/drm/.
[28] M.S. Sweeney, S. Hochgreb, M.J. Dunn, R.S. Barlow, The structure of turbulent stratified and premixed methane/air flames I: Non-swirling flows, Combust. Flame 159 (9) (2012) 2896–2911.
[29] M. Zhang, R. Mao, H. Li, Z. An, Z.X. Chen, Graphics processing unit/artificial neural network-accelerated large-eddy simulation of swirling premixed flames, Phys. Fluids 36 (5) (2024).