GPU-based flow simulation with detailed chemical kinetics

Hai P. Le a,*, Jean-Luc Cambier b, Lord K. Cole a

a Mechanical and Aerospace Engineering Department, University of California, Los Angeles, CA 90095-1597, United States
b USAir Force Research Laboratory, Edwards AFB, CA 93524, United States

A B S T R A C T

The current paper reports on the implementation of a numerical solver on the Graphic Processing Units (GPUs) to model reactive gas mixtures with detailed chemical kinetics. The solver incorporates high-order finite volume methods for solving the fluid dynamical equations coupled with stiff source terms. The chemical kinetics are solved implicitly via an operator-splitting method. We explored different approaches in implementing a fast kinetics solver on the GPU. The detail of the implementation is discussed in the paper. The solver is tested with two high-order shock capturing schemes: MP5 [Suresh and Huynh, 1997] [9] and ADERWENO [Titarev and Toro, 2005] [10]. Considering only the fluid dynamics calculation, the speed-up factors obtained are 30 for the MP5 scheme and 55 for ADERWENO scheme. For the fully-coupled solver, the performance gain depended on the size of the reaction mechanism. Two different examples of chemistry were explored. The first mechanism consisted of 9 species and 38 reactions, resulting in a speed-up factor up to 35. The second, larger mechanism, consisted of 36 species and 308 reactions, resulting in a speed-up factor of up to 40.

1. Introduction

Detail study of complex physical phenomena associated with high-speed fluid flow requires a deep understanding of the fundamental physics which involves many highly non-linear processes evolving in different spatial and temporal scales. Inevitably, the challenge of solving these problems is due to the coupling mechanism between these processes and what impact they have on the flow solution. For example, in combustion study, one has to pay close attention to the coupling between the fluid transport and chemical kinetics in order to characterize the combustion process. Computational Fluid Dynamics (CFD) techniques can be used to obtain a detailed flow solution which can be applied in practical applications. Although the mathematical formulation of the physics can be addressed in great detail, numerical simulation of high-speed fluid flow in a non-equilibrium environment is often limited by the computational power demanded for solving the governing equations. Modern CFD codes are designed to take advantage of high performance computing (HPC) platform to reduce run time. Unfortunately, the tradition HPC resources are very limited due to their cost and maintenance requirement. These limitations have accentuated a need for a compact and low-cost HPC solution where numerical solvers can be effectively implemented.

During the last eight years, the Graphic Processing Unit (GPU) has been introduced as a promising alternative to high-cost HPC platforms. Within this period, the GPU has evolved into a highly capable and low-cost computing solution for scientific research. Fig. 1 illustrates the superiority of GPU over the traditional Central Processing Unit (CPU) in terms of floating point calculation. This is due to the fact that the GPU is designed for the highly parallel process of graphic rendering. Starting in 2008, the GPU began to support double precision calculation, which is necessary for scientific computing. The newest generation of NVIDIA GPUs called “Fermi” has been designed to enhance the performance on double precision calculation over the previous generations of NVIDIA GPUs. CUDA [1], which is currently the most popular programming environment for general purpose GPU computing, has undergone several development phases and reached a certain level of maturity, which is essential for the design of numerical solvers. Other alternatives such as OpenCL, DirectCompute, Stream, etc. have also begun to mature which encourages new developments in scientific computing using GPU. Several attempts had been made in writing scientific codes on the GPU either by directly using CUDA or via some wrapper which calls the CUDA kernel functions, and promising results were obtained both in terms of performance and flexibility. Previous implementations of CFD codes on the GPU focused purely on solving the fluid dynamics using either finite volume [2,3] or finite element methods [4]. Recently, there have been a number of GPU-based implementations of multiphysics simulation. Of particular note are the extension to magnetohydrodynamics simulation by Wong et al. [5] and the automated preprocessor tool to model finite rate chemical kinetics by Linford et al. [6,7]. In all cases, the reported speed-ups show...
promising performance results and clearly demonstrate that GPU is suitable for massively parallel scientific computing.

In this paper, we describe the detail code implementation of a numerical solver coupling the fluid dynamics with detailed chemical kinetics. Unlike previous attempts at adapting the GPU to numerical solver, we have placed our emphasis on the kinetics solver rather than the fluid dynamics. This is due to the fact that for the simulation of high-speed fluid flow, the computation is dominated by solving the kinetics. While the current implementation is only for chemical kinetics, it is easy to extend it to a more general kinetics (collision-radiative kinetics for plasma) since all the elementary processes for a plasma (excitation/de-excitation, ionization/recombination, etc.) can be represented by a chemical reaction with the rate computed a priori and tabulated as a function of temperature.

The rest of the paper is organized as follows. The governing equation and numerical formulation for both the fluid dynamics and chemical kinetics are described in Sections 2 and 3. Section 4 gives some background on GPU computing. The code implementation is detailed in Section 5, highlighting several optimization techniques for maximizing the performance of the solver. The results of several benchmark test cases both for non-reactive and reactive flow fields are presented in Section 6 as well as the performance results of the solver. Section 7 gives the conclusions and points out future works.

2. Governing equations

The flow is modeled as a mixture of gas species while neglecting viscous effects. The chemical reactions taken place between the gas components are to be modeled in great detail. The set of the Euler equations for a reactive gas mixture can be written as:

\[ \frac{\partial \rho}{\partial t} + \nabla \cdot \vec{F} = \dot{\Omega} \]

where \( \rho \) and \( \vec{F} \) are the vectors of conservative variables and inviscid fluxes, respectively. We assumed that there is no species diffusion and the gas is thermally equilibrium (i.e., all species have the same velocity and all the internal energy modes are at equilibrium). The right hand side (RHS) of Eq. (1) denotes the vector of source terms \( \dot{\Omega} \), which are composed here of exchange terms due to chemical reactions. We solve the system (1) in a finite-volume formulation, by applying Gauss’s law to the divergence of the fluxes:

\[ \frac{\partial Q}{\partial t} + \frac{1}{V} \oint \mathbf{F}_n dS = \dot{\Omega} \]

where \( Q, F_n \) and \( \dot{\Omega} \) now denote volume-averaged quantities, which can be written as:

\[ \begin{align*}
Q &= \begin{pmatrix} \rho_s \\ \rho u_s \\ \rho v_s \\ \rho w_s \\ E \end{pmatrix}, \\
F_n &= \begin{pmatrix} \rho \dot{u}_s \\ \rho \dot{v}_s \\ \rho \dot{w}_s \\ \rho \dot{E} \\ M \dot{\vec{u}} \end{pmatrix}, \\
\dot{\Omega} &= \begin{pmatrix} \dot{\omega}_s \\ \dot{\omega}_v \\ \dot{\omega}_w \\ \dot{\omega}_E \end{pmatrix}
\end{align*} \]

where \( \dot{u}_s \) is the velocity vector normal to the interface and \((n_x, n_y, n_z)\) is the corresponding unit vector. The total energy is the sum of the internal energies from each species and the total kinetic energy:

\[ E = \sum_i \rho_i e_i + \frac{1}{2} \rho \vec{u}^2. \]

Since the species formation energies \( e_i \) are not included in that definition, we must account for their change in the source term \( \dot{\omega}_E \). The convective terms and the source terms are solved independently of each other by making use of an operator-splitting technique.

\[ \frac{\partial Q}{\partial t} = \left( \frac{\partial Q}{\partial t} \right)_{conv} + \left( \frac{\partial Q}{\partial t} \right)_{chem} = \frac{1}{V} \oint F_n dS + \dot{\Omega}. \]

The equation of state (EOS) is that of an ideal gas, i.e. Dalton’s law of partial pressures:

\[ P = NRT = \sum_i \rho_i (R/M_i)T \]

where \( R \) is the Boltzmann constant (in J/mol-K) and \( M_i \) the species molar mass. The pressure can also be determined from the conserved variables, \( \{ \rho, \vec{m}, E \} \), where \( \vec{m} = \rho \vec{u} = \sum \rho_i \vec{u}_i \), by:

\[ P = (\gamma - 1) \left( E - \frac{\vec{m}^2}{2} \right). \]

This formulation allows us to compute the pressure derivatives with respect to the conservative variables, needed for the flux Jacobian. Comparing (6) and (7), we find the expression for the effective ratio of specific heats \( \gamma \):

\[ \gamma = 1 + \frac{\sum \rho_i/M_i}{\rho \vec{e}_i} \quad \text{with} \quad \rho \vec{e}_i = \sum \rho_i e_i. \]

Using

\[ \left( \frac{\partial \gamma}{\partial \rho_s} \right)_{\dot{\omega}, m} = \frac{RT}{\rho \vec{e}_i} \left( \frac{1}{M_i} - \frac{e_i}{M \vec{e}_i} \right) \]

with \( \vec{M} = \rho/N \), we find (using the notation \( P_{\dot{\omega}} = \partial P/\partial \dot{\omega} \)):

\[ P_{\dot{\omega}} = (\gamma - 1) \frac{\vec{u}^2}{2} + \left( \frac{RT}{M_i} - \frac{\dot{\omega}_E}{M \vec{e}_i} \right). \]
Fig. 2. Schematic of computational stencil for MP5 scheme with left and right states of an interface.

Note that \( \sum_i \rho_i \rho_{i_j} = (\gamma - 1) u_i^2 / 2 \). The other derivatives are:

\[
P_{m_0} = - (\gamma - 1) u_{\alpha} \quad \alpha = x, y, z
\]  \(\text{(11)}\)

and

\[
P_e = \gamma - 1.
\]  \(\text{(12)}\)

The speed of sound is defined as

\[
c^2 = \sum_i \hat{c}_i \rho_i + \left( h - u^2 \right) P_e = \gamma \frac{P}{\rho}
\]  \(\text{(13)}\)

where \( \hat{c}_i = \rho_i / \rho \) is the species mass fraction and

\[
h = \frac{H}{\rho} = \frac{E + P}{\rho}
\]  \(\text{(14)}\)

is the specific enthalpy.

3. Numerical formulation

3.1. Fluid dynamics

A dimensional splitting technique [8] is utilized for solving the convective part of the governing equations. In order to achieve high-order both in space and time, we employed a fifth-order Monotonicity-Preserving scheme (MPS) [9] for the reconstruction, and a third-order Runge–Kutta (RK3) for time integration. For the MP5 scheme, the reconstructed value of the left and right states of interface \( j + \frac{1}{2} \) is given as (see Fig. 2):

\[
u_i^{j+1/2} = \frac{1}{60} \left( 2u_{i,j-2} - 13u_{i,j-1} + 47u_{i,j} + 27u_{i,j+1} - 3u_{i,j+2} \right)
\]  \(\text{(15a)}\)

\[
u_i^{j+1/2} = \frac{1}{60} \left( 2u_{i,j+1} - 13u_{i,j+2} + 47u_{i,j} + 27u_{j+1} - 3u_{j+2} \right).
\]  \(\text{(15b)}\)

The reconstructed values are then limited to avoid instability.

\[
u_i^{j+1/2} = \min \left( \nu_i^{j+1/2}, u_i, u_{\text{lim}} \right)
\]  \(\text{(16)}\)

where

\[
u_{\text{lim}} = u_i + \min \left( u_{i+1} - u_i, \alpha \left( u_j - u_{j-1} \right) \right)
\]  \(\text{(17)}\)

with \( \alpha = 2 \).

In addition to the MP5 scheme, we also considered the Arbitrary Derivative Riemann Solver using the Weighted Essentially Non-Oscillatory reconstruction procedure, the so-called ADERWENO scheme [10]. At each interface of 1 one-dimensional stencil, we seek the solution of the generalized Riemann problem (GRP)

\[\partial_t Q + \partial_x F(Q) = 0\]  \(\text{(18)}\)

with the following initial conditions

\[
Q^{(i)}(x, 0) = \begin{cases} 
q_i^{(i)}(x) & \text{if } x < 0 \\
u_i^{(i)}(x) & \text{if } x > 0.
\end{cases}
\]  \(\text{(19)}\)

The solution of Eq. (18) can be expanded using the Taylor Series expansion in time.

\[
Q(x_{j+1/2}, t + h) = Q(x_{j+1/2}, t) + \sum_{k=1}^{\frac{h^k}{k!}} \frac{\partial^k}{\partial t^k} Q(x_{j+1/2}, t)
\]  \(\text{(20)}\)

where all the temporal derivatives can be determined using the Cauchy–Kowalewski procedures [10]. The solution obtained in this form is high-order both in space and time, so no additional time stepping (i.e., multi-stage RK) is required. This provides certain advantages over the MP5 scheme since the overheads since the RK additional stages for high temporal order are no longer required. However, one disadvantage of the ADERWENO scheme is that the scheme is not guaranteed to be total variation diminishing (TVD) which might be an issue in the region of strong compression or expansion waves. It is worthwhile to mention that there are other formulations of ADER schemes available in the literature [11] which are proven to be TVD. Although such schemes have not been implemented, the core implementation of the solver described in this work can be used as the building blocks for other formulations of ADER schemes.

The interface fluxes are solved by employing the HLLE Riemann solver [12], which is given as

\[
f_{\text{HLLE}}^{j+1/2} = \frac{b^+F_k - b^-F_k}{b^+ - b^-} + \frac{b^+b^-}{b^+ - b^-} \Delta Q_{j+1/2}
\]  \(\text{(21)}\)

where

\[
b^+ = \max(0, \hat{u}_n + \hat{c}, \hat{u}_{n+1} + c_k)
\]  \(\text{(22)}\)

\[
b^- = \min(0, \hat{u}_n - \hat{c}, \hat{u}_{n-1} - c_k)
\]  \(\text{(23)}\)

3.2. Chemical kinetics

An elementary chemical reaction takes the form

\[
\sum_s v_i^s[X_i] \leftrightarrow \sum_s v_i^v[X_i]
\]  \(\text{(24)}\)

where \( v_i^s \) and \( v_i^v \) are the molar stoichiometric coefficients of the reactants and products of each reaction. The forward rate can be expressed in modified Arrhenius form as:

\[
K_{fr} = A_{fr} T^{b_{fr}} \exp \left( - \frac{E_{fr}}{RT} \right).
\]  \(\text{(25)}\)

The backward reaction rate is calculated from the equilibrium constant, which is given as

\[
K_e = \frac{K_{fr}}{K_{or}} = \left( \frac{P_e}{RT} \right)^{\sum_s v_i^v} \exp \left( - \frac{\Delta G^o}{RT} \right).
\]  \(\text{(26)}\)

The species net production rate due to all reactions can then be determined from

\[
\dot{\omega}_s = \sum_i M_i v_i^v \left[ K_{fr} \prod_s [X_i]^{v_i^v} - K_{or} \prod_s [X_i]^{v_i^v} \right]
\]  \(\text{(27)}\)

where \( v_i r = v_i^v - v_i^s \). By conservation of mass, sum of all the species production rates should be equal to zero which yields the following expression.

\[
\sum_s \dot{\omega}_s = 0.
\]  \(\text{(28)}\)

In order to solve for the change in the species concentration, one needs to know all the changes in the thermodynamics for each reaction as well as their rates (production/ destruction). The backward rate for each reaction is calculated based on detail balancing. In practice, acceptable result can also be obtained using curve-fitting technique with the temperature as an input without the expense of calculating the equilibrium constant. In this work, we have computed the backward rates from the equilibrium constant.
Fig. 3. Data parallelism in GPU [1].

The chemical kinetics, expressed in the form of an ODE, is solved using a point implicit solver to ensure stability. The formulation can be obtained by using a Taylor series expansion in time of the RHS

\[
\frac{dQ}{dt} = \dot{\Omega}^{n+1}
\]

(29)

\[
\frac{dQ}{dt} = \dot{\Omega}^n + \Delta t \frac{\partial \dot{\Omega}}{\partial Q}.
\]

(30)

By applying chain rule to the time derivatives on the RHS, one could obtain

\[
\left(1 - \Delta t \frac{\partial \dot{\Omega}}{\partial Q}\right) \frac{dQ}{dt} = \dot{\Omega}.
\]

(31)

As a linear system of equations, Eq. (31) can be solved using a variety of numerical methods. In the current work, a direct Gaussian elimination procedure is carried out in order to solve for the linear system of the chemical kinetics. It must be pointed out that the computational cost of the Gaussian elimination procedure scales as \(N_s^3\) where \(N_s\) in this case is the number of species. For large/detailed kinetics, solving the system at every cell is clearly a computationally intensive task.

4. GPU computing

The GPU processes data in a Single-Instruction-Multiple-Thread (SIMT) manner. The instruction for executing on the GPU is called a kernel which is invoked from the host (CPU). The CUDA programming model consists of grid and thread block. A grid consists of multiple thread blocks and each thread block contains a number of threads. When a kernel is called, the scheduler unit on the device will automatically assign a group of thread blocks to the number of available streaming multi-processors (SM or GPU core) on the device. Once the SM has completed the calculation, it will be assigned another block. Since there is no communication between the thread blocks, the execution order is automatically optimized so GPU with more cores will perform the calculation faster, which is shown in Fig. 3.

The data parallelism is also inherent at the thread level. An instruction given to a thread block is handled by a SM which contains a number of streaming processors (SPs). All the threads within each block will be organized into groups of 32 threads called warps which are executed in a SIMT manner. The difference in the data parallelism between grid and thread block is that there is a synchronization mechanism for all the threads in a same block but not for all the blocks in the grid. It is therefore important to ensure that there is no data dependency between thread blocks.

5. Implementation

The overall implementation of the code can be divided into two parts: the fluid dynamics and kinetics. The fluid dynamics module is responsible for the advection calculation. The kinetics module, on the other hand, calculates the species consumption/production due to chemical reactions and ensures detail balance is satisfied. The overall flow chart of the program is shown in Fig. 4 (see also Fig. 5 for the flow chart of the fluid dynamics module). After all the flow variables have been initialized and transferred to the GPU, the entire calculation is performed on the device. There is no memory transfer between each iteration unless there is a need to output the flow solution. This effectively reduces the memory transfer time between iterations, which can be significant for large-scale problems. However, for calculation utilizing more than one GPU, it is unavoidable to transfer memory back to the host for boundary exchange (ghost cells).

5.1. Computational fluid dynamics

The parallelization is done by directly mapping the computational domain to a CUDA grid. The face values can be mapped the same way with a larger grid since the number of faces in each direction is always 1 greater than the number of cells in that direction. For a rectilinear grid, each CUDA thread can be associated with one cell/face inside the computational domain.
A three-dimensional data array is stored in a memory space. For example, if the stencil size is such that all the elements of the stencil are located in contiguous memory space. The computational domain can be up to three-dimensional, one can split the stencil in different ways. However, it is desired to split the stencil so that all the elements of a stencil are located in contiguous memory space. For example, if i is the fastest varying index of a three-dimensional data array A(i, j, k), the stencil is created by splitting the domain along the i direction. The directional indices (i, j, k) of a three-dimensional domain with lengths IDIM, JDIM and KDIM can be calculated from the thread index, as shown in the following CUDA code snippets:

```c
int tid = blockIdx.x*blockDim.x+threadIdx.x;
int k = tid % IDIM;
int ij = tid / IDIM;
int j = ij / JDIM;
int i = ij % IDIM;
```

Since each stencil can be fitted into a block of threads, each component of the stencil is associated with a thread. Since all the threads within a block are accessing consecutive memory address, the access pattern is coalesced resulting in high memory bandwidth.

The calculation inside the CFD kernels requires a certain amount of memory which cannot be fitted in shared memory. This is because the size of the conservative variable vector is directly proportional to the number of species which can be quite large for a reacting mixture. In addition, high-order reconstruction requires an interpolating stencil whose length is three or five computational cells depending on the order of the scheme. Hence, the shared memory was not found useful in this case. In the CFD calculation, the size of the thread block can have an impact on the performance of the kernel. It is usually recommended by the CUDA programming guide [1] to maximize the block occupancy to make up for the memory latency. Since the occupancy factor is proportional to the block size, a large block size would result in high occupancy. However, Volkov [13] has shown that in the case where there are multiple independent instructions in the kernel, it is more advantageous to make the block size smaller and utilize more registers to cover for the memory latency. Since each entry of the eigensystem can be constructed independently of the others, the kernel performs faster in the case of smaller block size. This is referred as Instruction-level Parallelism (ILP). This is the approach utilized in the CFD kernels to achieve high performance. One example is the construction of the eigensystem in the fluid solver. More information on how to optimize the performance of a kernel using ILP can be found in Volkov [13].

5.2. Chemical kinetics

The parallelization of the kinetics solver can greatly benefit from GPU acceleration. The problem can be described as a simple linear algebra problem \( A \cdot x = b \) where \( A \) is the Jacobian matrix mentioned in Eq. (31). The solution of the system contains the change in molar concentration of all the species due to chemical reactions. It must be noted that the kinetics calculation is complex and the computation time depends on a variety of parameters such as domain size, number of species, number of reactions, etc. In the optimization study of the kinetics solver, we have neglected the effect of the number of reactions which is present in the rate calculation (i.e., construction of the Jacobian), so the focus can be placed on the matrix inversion algorithm where the system size \((N_s)\) can be varied. In general, the construction of the Jacobian is also a time consuming process if one considers a large number of reactions, but case dependent optimization techniques can be applied, making generalization inefficient. Since the solver is designed to solve a general set of kinetics, the detail in the construction of the Jacobian is not discussed here, although in this work, we have also attempted to optimize the rate calculation. The efficiency of the rate calculation will be shown implicitly later in a comparison of two fully-coupled solutions using two different chemistry mechanisms.

The linear system \( A \cdot x = b \) is solved using a Gaussian elimination algorithm, which is a sequential method. Since the kinetics in each computational cell are independent of other cells, one can parallelize the system on thread-per-cell basis. The Gaussian elimination process requires a considerable amount of memory access for read and write instructions to use and modify the values of the Jacobian. We investigated different approaches to maximize the performance of the kinetics solver. The first approach was to store the entire system for all cells on global memory. This is referred as the global memory approach. Although global memory is the slowest type of memory available on the device, coalesced memory access can result in high memory bandwidth close to the theoretical limit. The advantage of this approach lies in its simplicity. This is similar to the approach taken by Linford et al. [6, 7] for their implementation of the chemical kinetics kernel on the GPU. Since the global memory is the largest on the device, the restriction on the number of species can be relaxed. This is referred as the global memory (coalesced) approach throughout the text. A known issue of the Gaussian elimination is the significant accumulation of round-off error for large linear system, so double-precision is rapidly a necessity.

![Fig. 4. The overall work flow of the program. The fluid dynamics calculation and chemical kinetics are done by several GPU kernels.](image-url)
Fig. 5. Flow chart of the fluid dynamics calculation. The loop shown here represents the number of dimensions considered in the problem. For a three-dimensional problem, one needs to sweep in all i, j, and k directions.

It is always recommended [1] to utilize shared memory whenever possible to reduce global memory traffic. In this case, solving the chemical system requires inverting a $N_s$-by-$N_s$ matrix and the system needs to be solved at every computational cell. Storing the whole Jacobian and the RHS vector on shared memory is ideal in this particular case. Fig. 6 shows the memory requirement for storing the Jacobian and RHS on the typical shared memory (i.e., a Tesla C2050/2070 has 48 KB of shared memory per CUDA block). If we associated an entire thread block to the chemical system in a cell, the number of species is limited to 75. Storing more than 1 system per block decreases this limit further, which is the case for 1 thread per cell (32 threads block size). The approach of storing the entire system of one computational cell per block is referred as the shared memory (full storage pattern) approach.

In order to overcome the shared memory limit, we considered storing only two rows of the Jacobian in shared memory since the sequence of the elimination is done row-by-row. In this approach, referred as the shared memory (reduced storage pattern) approach, one needs to store values for the current row and the pivot row for each row elimination. Fig. 7 shows a comparison of the species limit for two approaches; the species limit for the reduced storage pattern is much higher than the full storage pattern. The draw-back of this approach, however, is that there are multiple memory transfers between global and shared memory, since we are required to copy back the values of each row after being eliminated. The parallelization is only effective when the calculation time dominates the global-shared memory transfer time (i.e., $N_s$ is large). It will be shown later that this algorithm is only fast for linear systems with a large number of species. It must be noted that the shared memory approach stated in the remaining of paper will refer to the reduced storage pattern approach.

6. Results

6.1. Solver results

The first objective is to verify that the solver is correctly implemented using the CUDA kernels. For this purpose, we can compare the results with a pure-CPU version, but also compute a set of standard test cases. The first of those is a Mach 3 wind tunnel problem (a.k.a. the forward step problem) using the MPS scheme, whose solution is shown in Fig. 8. This problem had been utilized by Woodward and Colella [14] to test a variety of numerical schemes. The whole domain is initialized with Mach-3 flow and reflective boundary conditions are enforced on the step and the upper part of the domain. The left and the right boundary conditions are set as in-flow and out-flow, respectively. Special attention is usually
required at the corner of the step since this is a singular point of the flow which can create numerical instabilities. Woodward and Colella treated this by assuming the flow near the corner is nearly steady. However, this artificial fix was not used in this simulation since we want to test the robustness of the solver in the case of strong shocks and how it handles the singularity in wall curvature, responsible for very strong expansion.

The second test involves a similar problem of a diffraction of a shock wave ($M = 2.4$) down a step [15]. The strong rarefaction at the corner of the step can cause a problem of negative density when performing the reconstruction. The problem is modeled here using 27,000 cells, and the numerical simulation is shown in pair with the experimental images in Fig. 9. The solver was able to reproduce the correct flow features with excellent accuracy.

We also modeled the Rayleigh–Taylor instability problem [16] (see Fig. 10). The problem is described as the acceleration of a heavy fluid into a light fluid driven by gravity. In this test case, the specific heat ratio is set to be a constant ($\gamma = 1.4$). For a rectangular domain of $(0.25 \times 1)$, the initial conditions are given as follows:

\[
\begin{align*}
\rho &= 2, \quad u = 0, \quad v = -0.025 \cos(8\pi x), \\
P &= 2y + 1 \quad \text{for } 0 \leq y \leq \frac{1}{2} \\
\rho &= 1, \quad u = 0, \quad v = -0.025c \cos(8\pi x), \\
P &= y + \frac{3}{2} \quad \text{for } \frac{1}{2} \leq y \leq 1
\end{align*}
\]  

(32)

(33)

where $c$ is the speed of sound. The top and bottom boundaries are set as reflecting and the left and right boundaries are periodic. As the flow progresses, the shear layer starts to develop and the Kelvin–Helmholtz instabilities become more evident. A momentum and energy source is added to account for the gravitational effects. This source is relatively simple and contributes very little to the overall computational time. The performance of the fluid dynamics calculation is discussed in the next section of this report.

Fig. 10. Rayleigh–Taylor instability computed with the MP5 scheme with 640,000 cells.

We now turn the attention to the modeling of a reactive flow field. We simulated a spark-ignited detonation wave both in one- and two-dimension to demonstrate the capability of the solver. At a well-resolved scale, the detonation wave can be described as a strong shock wave supported by the heat release from a high-temperature flame behind an induction zone. Interesting features have been observed both in the 1-D and 2-D simulations, characterized by the coupling of the fluid dynamics and chemical kinetics. The study of flame-shock coupling is an on-going research topic [17] and certainly can be aided with GPU computing when the evolution of the detonation wave needs to be resolved at a very fine spatial scale.

The evolution of the pressure and temperature of a wall-spark ignited detonation is shown in Fig. 11. The chemical kinetics is modeled using the reduced $\text{H}_2$-air mechanism which consists of 9 species gas mixture with 38 reactions. The mechanism used for the simulation is taken from the shock tube study by Jachimowski [18]. The computational domain is rectangular with a length of 20 cm and a height of 2 cm. The grid spacing in both directions is 50 µm. The detonation cells, between the shock and the multiple triple points in transverse motion, are clearly seen. Fig. 12 illustrates the numerical soot film produced by recording the maximum density reached at each computation cell over the entire simulation time which is used to measure the cell structure. This well-known cellular structure has been observed both in experiments and numerical simulations. Various techniques in reproducing these images are discussed by Sharpe and Radulescu [19]. We will show in the next section how the superior performance of the GPU can enhance our ability in modeling reactive flows.

6.2. Performance results

Fig. 13 shows the performance of the solver for the simulation considering only the fluid dynamics aspect using the MP5 and the
ADERWENO schemes. All the comparisons reported in this paper are made between a Tesla C2070 GPU and an Intel Xeon X5650 CPU (single thread), both of which are using double precision calculation. Since the ADERWENO scheme only requires single-stage time integration, it is faster than the MP5 scheme. For the ADERWENO scheme, we can obtain almost 60 times speed-up for a large grid which is about twice faster than the MP5 scheme. The speed-ups obtained in both cases are very promising.

The performance of the kinetics solver depends strongly on the memory access efficiency. Since the Gaussian elimination algorithm requires issuing a large amount of memory instructions (both Read and Write) to modify all the entries of the Jacobian, it is important to achieve high memory bandwidth while maintaining sufficient independent arithmetic operations to hide memory latency. The efficiency of the memory access scheme is very crucial in the case when the whole Jacobian is stored inside global memory (DRAM) which has much higher latency than shared memory. Fig. 14 illustrates the memory access efficiency of the GPU kernel performing the Gaussian elimination procedure. The figure shows that coalesced memory access results in much higher memory bandwidth comparing to the non-coalesced pattern for the same number of operations. The memory bandwidth obtained with this operation is approximately 80% of the theoretical peak limit of the device (144 GB/s for a Tesla C2050). Similar tests were performed for a species count ranging from 5 to 200. The results shown in Fig. 14 indicate that the global memory access pattern in the first approach (global memory) with coalesced memory access is very efficient. This access pattern is used consistently in the global memory approach stated in the remainder of the paper.

In the second approach, referred as the shared memory approach, we utilized shared memory to compensate for DRAM latency issues exhibited in the first approach. However, due to the memory intensive nature of the detailed chemical kinetics problem and the limitation of shared memory storage, there is a substantial amount of DRAM access required which cannot be avoided. The memory bottleneck introduces addition memory latency which can affect the performance of the kernel. Fig. 15 shows the performance of the kinetics solver only (i.e., without convective transport). Although the construction of the chemical Jacobian and source terms can also be a time intensive process, in the present study, we wish to focus on the more computational intensive process of solving the linear system and its scaling. Hence, the performance is measured by solving a number of linear systems $A \cdot x = b$ with different system sizes ($N_s$) and grid sizes ($N_{cell}$).

We described two different implementation approaches in our earlier discussion. The first approach is to store everything on global memory and try to achieve high memory bandwidth by coalesced memory reads. The second approach is to transfer memory to shared memory for each row elimination. Fig. 15 shows that the global memory approach outperforms the shared memory version in all cases. Since the shared memory approach requires additional memory transfers between each row elimination, it is only effective when $N_s$ is large. It is shown in Fig. 15 that the algorithm is only effective when $N_s > 100$. In contrast, the performance of the global memory version depends strongly on the size of the grid. Since the parallelization is achieved across all the computational cells, solving a large number of systems makes it much more efficient. The global memory version seems to perform well in all cases. The speed-up obtained for the global memory is at least 30.
Fig. 13. Performance of the fluid dynamics simulation only (i.e., no chemical reactions). Comparison is made between two different schemes: MP5 and ADERWENO (both 5th order in space and 3rd order in time).

Fig. 14. Memory bandwidth of the kinetics solver measured on a Tesla C2050 for both coalesced and non-coalesced memory access. The memory bandwidth for both cases are compared with the theoretical bandwidth of a Tesla C2050 GPU to demonstrate the efficiency of the memory access scheme.

Fig. 15. Comparison of the speed-up factor obtained from the kinetics solver using both global memory (square line) and shared memory (circle line) approaches. Different colors indicate different grid sizes.

Fig. 16. Kinetics calculation of a 128 × 128 grid with 100 species. The calculation is done by decomposing the computational domain into numbers of segments (i.e., number of kernel calls).

The speed-up obtained in the shared memory approach is approximately 20 for a species count ranging from 100 to 300.

Due to memory constraint, it became necessary for us to vary the grid size as the number of species increases. Thus, we were able to consider a 512 × 512 grid of cells for up to 25 species, then a 128 × 128 grid was possible up to 200 species, etc. The intense memory usage comes from the need to store a $N_s \times N_s$ Jacobian for each computational cell (e.g., this variable alone adds up to more than 5 GB of global memory for 200 species on the 128 × 128 grid). Some of the GPU's memory load can be alleviated by storing only the Jacobian for a segment of the domain; for example, a calculation of a 512 × 512 grid can be decomposed into 4 kernels, each of which performs calculation on a 512 × 128 grid. For this strategy to be effective, the decomposed domain must be sufficiently large, i.e., of the other of the number of streaming processors in the GPU, and the overhead for each kernel call is small. The important of the domain size is highlighted in Fig. 16 where the calculation is performed for a grid with a fixed size of 128 × 128 with 100 species and varying number of segments. In this test case, the overhead due to the decomposition is negligible up to 4 segments. When the domain is further divided into 8 segments, the size of each segment gets smaller (2048 cells) and this strategy is no longer effective.

Fig. 17 shows the performance of the flow solver coupling with the chemical kinetics. The result shown in the plot is for a reduced H$_2$-air mechanism consisting of 9 species and 38 reactions (both forward and backward). MP5 scheme is utilized in all cases. Although ADERWENO scheme has shown to be faster than MP5 in the case of convective transport, we do not expect to see significant difference in the performance because the computation time is dominated by chemical kinetics rather than convection. The overall performance is greatly dependent on the performance of the kinetics solver. It is clear that the global memory approach
outperforms the shared memory approach. This is consistent with the results obtained earlier for the fluid dynamics and the kinetics separately. As the grid size increases, the speed-up factor obtained for the shared memory approach of the kinetics solver does not change rapidly. In contrast, the global memory approach results in a 40 times speed-up for a large grid.

We extend the simulation to model a larger mechanism of CH₄-air detonation with 36 species and 308 reactions. The performance is compared with the previous result of H₂-air detonation. Fig. 18 illustrates how the CH₄-air detonation simulation achieves a greater speed-up than the H₂-air detonation mainly because of better performance in the construction of the chemical kinetics Jacobian. This indicates that the performance of the rate calculation scales with the number of reactions; thus, we can expect better performance for a large number of reactions which makes it very attractive for CFD calculation of hypersonic flow and plasma where additional processes such as transitions between excited states of atoms and molecules need to be modeled.

7. Conclusion and future works

In the current paper, we described the implementation of a numerical solver for simulating chemically reacting flow on the GPU. The fluid dynamics is modeled using high-order shock-capturing schemes, and the chemical kinetics is solved using an implicit solver. Results of both the fluid dynamics and chemical kinetics are shown. Considering only the fluid dynamics, we obtained a speed-up of 30 and 55 times compared to the CPU version for the MP5 and ADRWENO scheme, respectively. For the chemical kinetics, we presented two different approaches on implementing the Gaussian elimination algorithm on the GPU. The best performance obtained by solving the kinetics problem ranges from 30 to 40 depending on the size of the reaction mechanism. When the fluid dynamics is coupled with the kinetics, we obtained a speed-up factor of 40 times for a 9-species gas mixture with 38 reactions. The solver is also tested with a larger mechanism (36 species, 308 reactions) and the performance obtained is faster than the small mechanism.

The current work can be extended in different ways. First, since the framework is performing well in shared memory architecture, it is possible to also extend it to distributed memory architecture utilizing Message Passing Interface (MPI). The extension permits using multi-GPU which is attractive for performing large-scale simulations. On the other hand, although the current simulation is done for chemically reacting flow, it is desired to extend it to simulate ionized gas (i.e., plasma) which requires modeling additional physical processes to characterize different excitation levels of the charged species (Collisional-Radiative kinetics). In addition, the governing equations also need to be extended to characterize the thermal non-equilibrium environment of the plasma. Given that the physics has been well established [20–22], the extension is certainly trivial.

Acknowledgment

The authors would like to thank Prof. Ann Karagozian of UCLA for countless support on performing simulation on the Hoffman2 GPU cluster.

References