3-21-2012

Combustion Simulations Using Graphic Processing Units

Mingjie Wang
Mingjie Wang, mjw.gucas@gmail.com

Combustion Simulations

Using Graphic Processing Units

Mingjie Wang

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

May 2012

APPROVAL PAGE

Master of Science Thesis

Combustion Simulations

Using Graphic Processing Units

Presented by

Mingjie Wang, Mechanical Engineering.

Major Advisor _____
Tianfeng Lu

Associate Advisor _____
Chih-Jen (Jackie) Sung

Associate Advisor _____
Mandhapati P. Raju

University of Connecticut
2012

Acknowledgements

I would like to express my appreciation to my advisors: Dr. Tianfeng Lu, Dr. Chih-Jen (Jackie) Sung, and Dr. Mandhapati P. Raju. Thank you for giving me the opportunity to participate in the combustion research projects.

My sincere gratitude goes as well to my colleagues Zhaoyu Luo, Mehrnaz Rouhi-Youssefi and Ruiqin Shan. Thank you for spending time discussing questions with me. It was a pleasure working with you.

Very special thanks go to my wife, also my best friend and partner, Mengyao Zhang, for your unconditional support and love.

Contents

Abstract

Graphic processing units (GPUs) are powerful graphics engines featuring high levels of parallelism and extreme memory bandwidth, which constitute a powerful computing platform to solve complex problems involving chemically reacting flows. In the present study, computer programs for combustion simulations with detailed chemical kinetic mechanisms were compiled in the Compute Unified Device Architecture (CUDA) language for NVIDIA GPU architecture. Ignition processes were simulated under constant pressure and constant volume conditions using an explicit 4^{th} order Runge-Kutta algorithm for time integration. Sufficiently small time steps were identified with time scale analysis to ensure the integration stability. The program was validated with the results from simulations with CPUs using detailed mechanisms of various fuels including H_2, and CH_4. It was found that the GPU-accelerated simulations can be approximately 10-20 times faster than those on CPUs for solving identical problems. Furthermore, the newly implemented GPU solver for detailed chemical kinetics was employed for quasi 2-D simulations.

1. Introduction

Detailed chemical kinetic mechanisms have been developed for a variety of fuels during the last few decades. However, the availability does not mean that these mechanisms are ready to be adopted to numerical simulation. In fact, these detailed mechanisms are getting more and more complex. To demonstrate this, the statistic plot of the major chemical kinetic mechanisms developed in the last two decades for a variety of hydrocarbon fuels is given in Figure 1.1 [1]. From this figure, for instance, GRI1.2 mechanism [2] developed before 2000 for methane oxidation consists of 32 species and 177 elementary chemical reactions, while a detailed mechanism for methyl decanoate in 2008 [3] contains 2877 species and 8555 elementary reactions. Considering that 2-D or 3-D simulations of turbulent flames may require 10^6-10^9 spatial grid points, it is difficult to apply the detailed mechanisms of practical fuels in combustion simulations. Furthermore, due to limited access to supercomputers in solving many engineering problems, GPU with high level of parallelism can be a solution to make it possible to accommodate large detailed chemical kinetic mechanisms in combustion simulations.

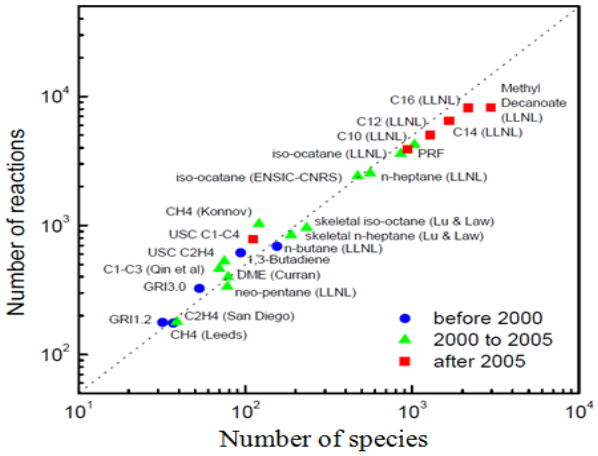


Figure 1.1 Sizes of selected detailed chemical kinetic mechanisms of typical hydrocarbon fuels (taken from [1])

High performance computing with GPUs has been studied in multiple disciplines, such as molecular dynamics [5-7], computational biology [8], linear algebra [9-10], weather forecasting [11] and artificial intelligence [12]. In the field of computational fluid dynamics, Foster et. al. (1996) [13] used an explicit integration scheme to solve the Navier-Stokes equations to simulate the turbulence of smoke using computer graphics. Stam (1999) [14] introduced an unconditionally stable Semi-Lagrangian method and solved the issue associated with the small time steps in Foster's method. With the increasing programmability of GPUs, Harris et al. (2002) [15] solved the coupled map lattice problem and simulated cloud dynamics involving partial differential equations. Goodnight et al. (2003) [16] implemented a multi-grid method on GPU, and applied it to the modeling of heat transfer and fluid mechanics. Kruger et al. (2003) [17] computed the basic linear algebra problems, and further computed the 2-D wave equations and the Navier-Stokes equations on GPU. Bloz et al. (2003) [18] utilized the multi-grid method to solve fluid dynamic problems by rearranging the sparse matrix into textures. Hagen (2006) [19] developed an Euler equation solver by using both GPUs and CPUs for simulating two types of flows, namely the shock-bubble interaction and Rayleigh-Taylor instability. Compared to an equivalent CPU code, a performance increase by 10-20 times was reported by Hagen. A code to simulate complex flow in 3-D geometries was developed for GPU by Brandvink and Pullan (2008) [20], who also performed 2-D and 3-D compressible flow simulations on GPU and achieved speedup factors of 29 for 2-D cases and 16 for 3-D cases. Currently, GPUs can be used for flow simulations on highly complex geometries (2008) [21].

However, although the utilization of GPUs as general-purpose computing devices has resulted in many breakthroughs in computational sciences [22], complex reacting flow visualization and simulations with GPUs are only studied in a limited number of cases. Stam [23], proposed a thermodynamic simulation method for flames on computer graphics in 1995. Wang et al. [24] presented a survey on flame simulations in computer animation with a detailed introduction to different methods employed in the GPU field before 2005. In 2006, Storli et al. [25] developed an approach for simulating and visualizing 3-D flames in real time on GPUs. Turbulence and flicking of flames were presented in the paper by using an underlying fluid simulation, modeling the mass and heat transfer aspects of the combustion processes. The operations were implemented completely on the GPU to ensure high frame rates without compromising the visual quality. In 2010, a high-fidelity turbulent reacting flow solver (S3D) was accelerated on GPUs by Spafford, Merdith, Vetter, Chan, Grout, Sankaran in Oak Ridge National Laboratory and Sandia National Laboratory [26]. The calculation of the rates of chemical reactions was identified as a major performance bottleneck by using CPU based profiling tool. A speedup factor of 17 was reported for double precision reaction rates calculation on GPUs compared with that on a single CPU. Later in 2011, Shi et al [27] developed a GPU-enhanced algorithm for reaction rate evaluations. A speedup factor of 20 was achieved when using mechanisms with more than 2000 species, compared with that on a single CPU.

In the present study, a new GPU-based explicit chemical solver was developed to take full advantage of GPUs in accelerating combustion simulations with detailed chemical kinetics. The implementation of a parallel explicit auto-ignition solver on GPUs will be presented in

Chapter 2. The solver will be validated in Chapter 3, and its application in 2-D combustion simulations with transport processes ignored will be demonstrated in Chapter 4.

2. An explicit solver for chemical kinetics integration on GPUs

2.1. Governing equations

A closed system is studied in the present work for GPU simulations. The equation for mass conservation in a closed system is:

$$\frac{dm}{dt} = 0, m = \sum_{i=1}^{K} m_i \tag{1}$$

where m_i is the mass of the *i*-th species and *K* is the total number of species in the mixture. Eq. (1) can also be written in the from of species mass fractions:

$$\rho \frac{dY_i}{dt} = \dot{\omega}_i W_i \tag{2}$$

where $Y_i = m_i/m$ is the mass fraction of the *i*-th species, t is time, ρ is the density of the mixture, $\dot{\omega}_i$ is the production rate of *i*-th species, and W_i is the molecular weight of the *i*-th species.

The energy equation for a closed adiabatic system can be described as:

$$de + pdv = 0 \tag{3}$$

where e is the specific internal energy, v is specific volume, and *p* is the pressure. If the volume of the system is fixed, $pdv = 0$, so that $de = 0$. Differentiating the internal energy of the mixture, we have

$$de = \sum_{i=1}^{K} d(e_i Y_i) = \sum_{i=1}^{K} Y_i de_i + \sum_{i=1}^{K} e_i dY_i \tag{4}$$

where e_i is the specific internal energy of the *i*-th species. Substituting $de_i = c_{v,i} dT$ into (4):

$$\sum_{i=1}^{K} Y_i c_{v,i} dT + \sum_{i=1}^{K} e_i dY_i = 0 \tag{5}$$

$c_{v,i}$ is specific heat capacity for constant volume of the *i*-th species. By defining a mean heat capacity under constant volume $\overline{c_v} = \sum_{i=1}^{K} Y_i c_{v,i}$, Eq. (5) becomes

$$\overline{c_v} \frac{dT}{dt} = -\sum_{i=1}^{K} e_i \frac{dY_i}{dt} \tag{6}$$

Substitution equation (2) to (6) resulted in:

$$\rho\overline{c_v}\frac{dT}{dt} = -\sum_{i=1}^{K} e_i\,\dot{\omega}_i W_i \tag{7}$$

For adiabatic systems under constant pressure, the energy equation is:

$$dh = d(e + pv) = de + pdv + vdp=0 \tag{8}$$

$$\frac{dh}{dt} = d(\sum_{i=1}^{K} h_i Y_i)/dt = \sum_{i=1}^{K} Y_i\frac{dh_i}{dt} + \sum_{i=1}^{K} h_i\frac{dY_i}{dt}=0 \tag{9}$$

where h_i is the specific enthalpy of the *i*-th species. Substituting $dh_i = c_{p,i}dT$ into (9):

$$\sum_{i=1}^{K} Y_i c_{p,i}\frac{dT}{dt} + \sum_{i=1}^{K} h_i\frac{dY_i}{dt} = 0 \tag{10}$$

where $c_{p,i}$ is heat capacity under constant pressure of the *i*-th species. As before, the energy equation for a constant-pressure system becomes

$$\rho\overline{c_p}\frac{dT}{dt} = -\sum_{i=1}^{K} h_i\,\dot{\omega}_i W_i \tag{11}$$

where $\overline{c_p} = \sum_{i=1}^{K} Y_i c_{p,i}$.

2.2. Explicit integration on GPUs

The time-dependent ordinary differential equations (ODEs) in Eqs. (2) (7) (11) can be solved by explicit or implicit solvers. As an example of implicit solvers, the differential algebraic sensitivity analysis code (DASAC^2) solves ODEs by the implicit backward Euler method [28] on CPUs. An attempt was made by Shi et al [27] on using GPUs to enhance the performance of implicit solver recently. It was observed that the evaluations of the reaction rates and the matrix operations in ODEs integration induce significant computational time. Hence, the rate of each reaction and the *LU* factorization of the Jacobian were calculated on the GPUs to improve the performance of the implicit solver. It is noted that this approach speed ups the simulation only for mechanisms with more than about 500 species. For smaller mechanisms,

CPU computation is more efficient than GPU enhanced computation. . In the present work, a 4th-order explicit Runge-Kutta (ERK) solver is implemented on the GPUs. The explicit solver does not involve evaluation and factorization of the Jacobian matrix, thereby renders it amenable to be implemented on GPUs. The 4^{th}-order explicit Runge-Kutta (ERK) method is outlined.

In general, initial value problems can be expressed as

$$\boldsymbol{y}' = f(t, \boldsymbol{y}), \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \tag{12}$$

Then, the ERK method for this problem is given by the following equations [29]

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{i=1}^{s} b_i \boldsymbol{k_i} \tag{13}$$

where h and b_i are constant coefficients, $k_i$can be shown as:

$$k_1 = \boldsymbol{f}(t_n, \boldsymbol{y}_n),$$

$$\boldsymbol{k}_2 = \boldsymbol{f}(t_n + c_2 h, \boldsymbol{y}_n + a_{21} h \boldsymbol{k}_1),$$

$$\boldsymbol{k}_3 = \boldsymbol{f}(t_n + c_3 h, \boldsymbol{y}_n + a_{31} h \boldsymbol{k}_1 + a_{32} h \boldsymbol{k}_2), \ldots$$

$$\boldsymbol{k}_s = \boldsymbol{f}\big(t_n + c_s h, \boldsymbol{y}_n + a_{s1} h \boldsymbol{k}_1 + a_{s2} h \boldsymbol{k}_2 + \cdots + a_{s,s-1} h \boldsymbol{k}_{s-1}\big).$$

The coefficients (c_s, $a_{s,s-1}$, b_s) above are shown in the following table known as the Butcher tableau [29]:

$$\begin{array}{ccccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \end{array}$$

$$b_1 \quad b_2 \quad \cdots \quad b_{s-1} \quad b_s$$

Specifically, the coefficients of the 4th order ERK method are [29]

0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1932/2197	−7200/2197	7296/2197			
1	439/216	−8	3680/513	-845/4104		
1/2	−8/27	2	−3544/2565	1859/4104	−11/40	
	16/135	0	6656/12825	28561/56430	−9/50	2/55

2.3. Time scale analysis for determination of integration time steps

The explicit integration may be unstable when integration steps are too large because of the chemical stiffness [30], which is caused by large differences between the timescales of different species [31]. As such, sufficiently small integration steps needed to be identified for integrating the detailed chemical kinetics. Note that the integration time step is determined prior to the GPU simulations using timescale analysis as described in the following.

For a typical reacting flow, the ODEs in the Lagrangian coordinate can be expressed as:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{g}(\boldsymbol{y}) \tag{14}$$

where $\boldsymbol{y}$ is the vector of dependent variables, including, for example, species concentrations and temperature. $\boldsymbol{g}(\boldsymbol{y})$ includes the chemical source term as well as all the non-chemical

source terms. For the homogeneous auto-ignition process, $\boldsymbol{g}(\boldsymbol{y})$ only contains the chemical source term. As a similar procedure to that in Computational Singular Perturbation (CSP) [32-33], by using chain rule, eq(14) can be expressed in the form involving the Jacobian:

$$\frac{d\boldsymbol{g}}{dt} = \boldsymbol{J} \cdot \boldsymbol{g}(\boldsymbol{y}) \, , \boldsymbol{J} = \frac{d\boldsymbol{g}}{d\boldsymbol{y}} \tag{15}$$

Solving the eigenvalues and eigenvectors of the Jacobian, $\boldsymbol{J}$ can be diagonalized as:

$$d\boldsymbol{f}/dt = \boldsymbol{\Lambda} \cdot \boldsymbol{f} \, , \boldsymbol{f} = \boldsymbol{B} \cdot \boldsymbol{g} \tag{16}$$

where $\boldsymbol{B}$ is the eigenvectors. Assuming that **B** and the eigenvalues of Jacobian $\lambda_1, \lambda_2, \cdots, \lambda_k$ are time independent. The chemical term $\boldsymbol{g}$ can then be represented as

$$\begin{aligned} \frac{df_1}{dt} &= \lambda_1 f_1 \\ \frac{df_2}{dt} &= \lambda_2 f_2 \\ &\vdots \\ \frac{df_k}{dt} &= \lambda_k f_k \end{aligned} \tag{17}$$

The solution of Eq. (17) is:

$$\begin{aligned} f_1 &= c_1 e^{\lambda_1 t} \\ f_2 &= c_2 e^{\lambda_2 t} \\ &\vdots \\ f_k &= c_k e^{\lambda_k t} \end{aligned} \tag{18}$$

where $\boldsymbol{c}$ is a constant vector determined by the initial condition. If λ_{max} is eigenvalue with the largest absolute value of the real part, the minimum time scale of chemical reactions can be estimated as

$$\tau_{min} = O(1/|\lambda_{max}|) \tag{19}$$

Thus the integration time step can be roughly determined for the explicit integration.

Figure 2.1 shows the minimum time scales of H_2, CH_4 and dimethyl ether (DME) mechanisms [3, 34-35] for constant pressure auto-ignition. It is seen that cases with higher initial pressure or temperature tend to have smaller time scales.

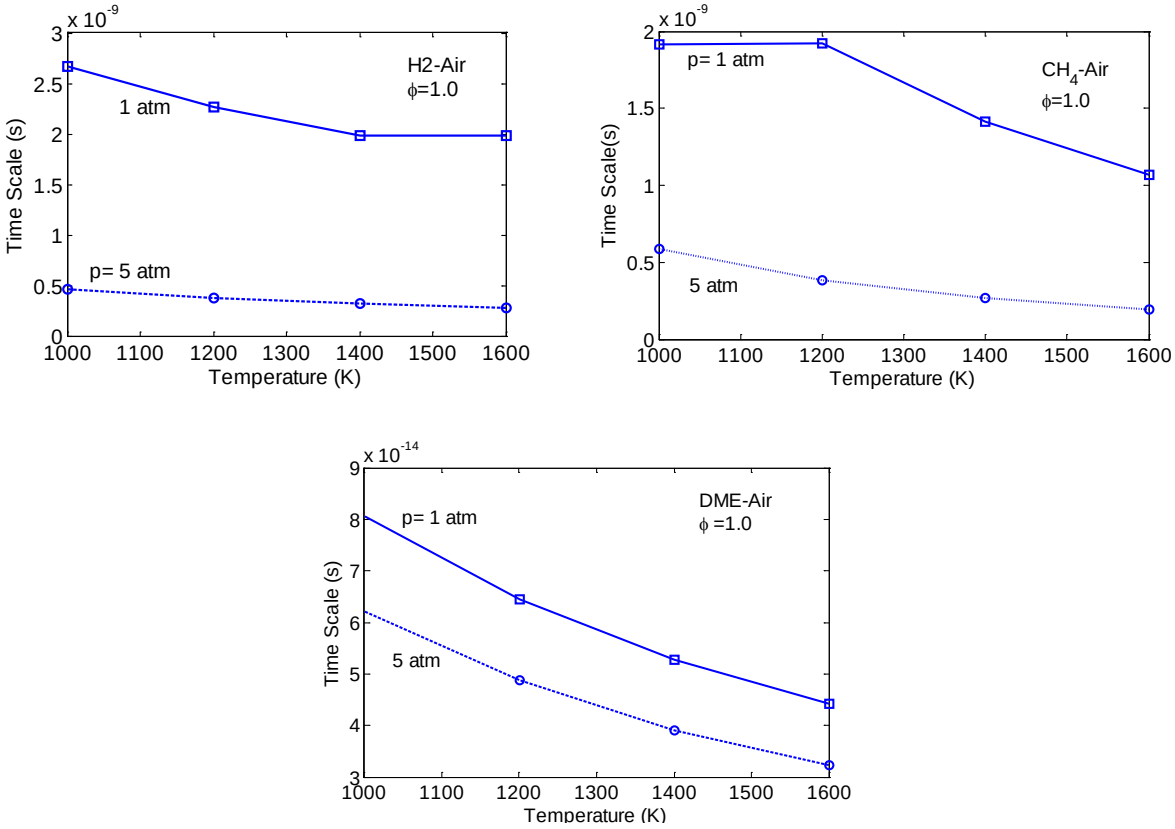


Figure 2.1 The shortest time scales in detailed mechanisms for H_2, CH_4 and DME, respectively, for auto-ignition under constant pressure.

Figure 2.2 shows the shortest time scales of H_2, CH_4 and DME for constant-volume auto-ignition. It was observed again that the time scales decrease with increasing pressure.

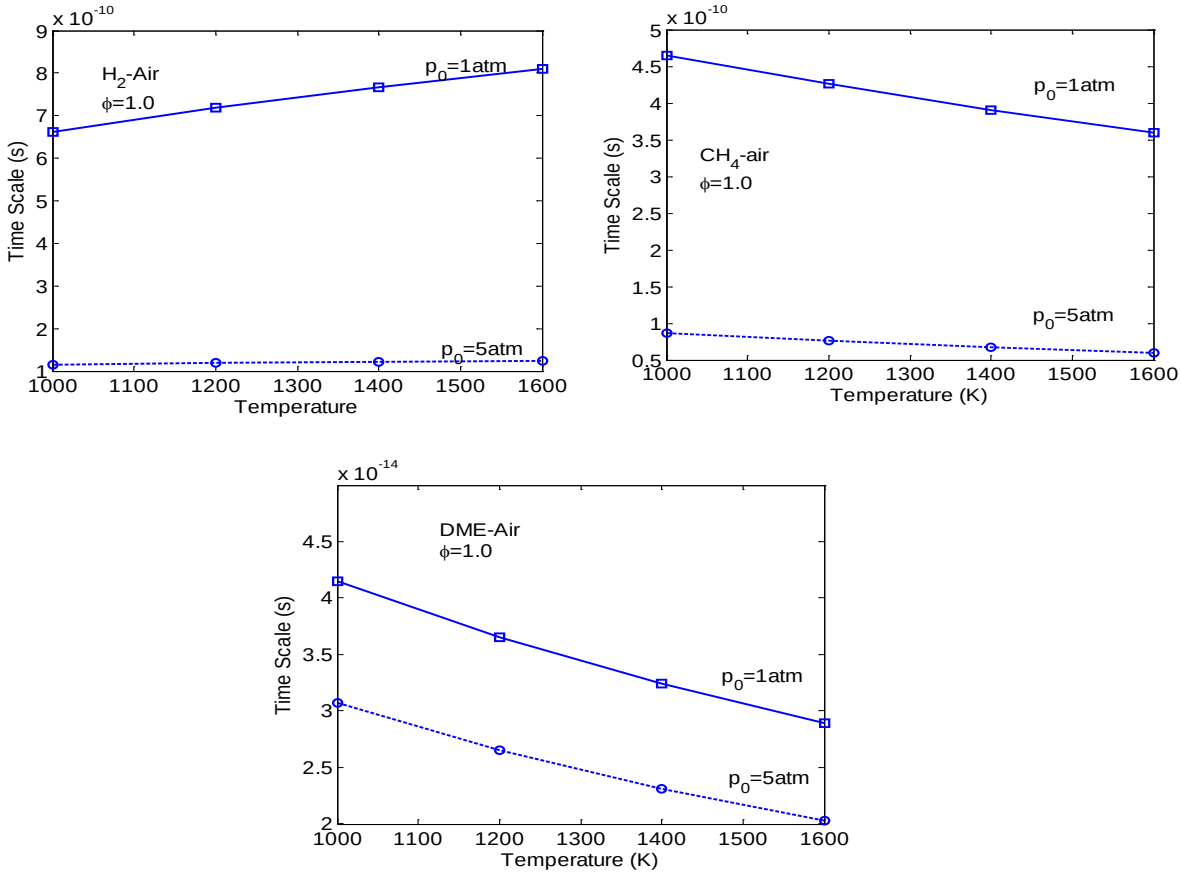


Figure 2.2 The shortest time scales in detailed mechanisms for H_2, CH_4 and DME, respectively, for auto-ignition in constant volume.

The information in Figs. 2.1 and 2.2 will then be utilized in the following to determine appropriate integration time steps for explicit integration of detailed chemical kinetics. For example, when simulating the CH_4-air auto-ignition with initial temperature of 1000K and a constant pressure of 1 atmosphere, a time step of 1.0e-9s can be employed to achieve stable explicit integration using the ERK scheme.

2.4. Implementation of ERK with CUDA on GPUs

In the current study a Tesla S1070 GPU, introduced by NVIDIA in 2008, was used for GPU simulations. Based on the product manual [36], an S1070 GPU unit consists of four hierarchical arrays of T10 processors with 4GB of associated global off-chip high speed DRAM. Each array of processors consists of 30 streaming multiprocessors (SMs). As shown in Fig. 2.3 [36], each SM contains eight cores called streaming processors (SPs), for a total of 240 cores in one T10 processor. Threads are grouped into batches of 32, called warps, which execute in lockstep SIMD fashion by running across the 8 cores of a SM at the same time. As such, a single instruction can be run across 32 threads in 4 clock cycles on one SM. All the SMs can access the global off-chip DRAM to load and store the same instructions. Data is transferred between the CPU and the GPU by crossing the PCIE bus, which typically has a lower bandwidth than either the CPU-to host memory or GPU-to-device memory interfaces, as shown in Figure 2.3.

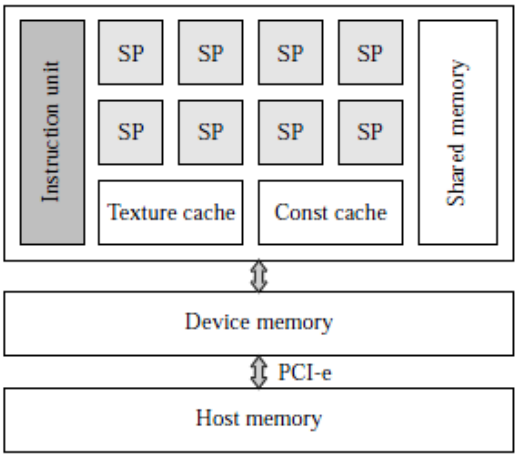


Figure 2.3 Schematic view of a GPU streaming multiprocessor with 8 scalar processor cores (Image from [36]).

The first version of GPU programming language CUDA was released by NVIDIA in 2006. After several updates, CUDA has become a general-purpose parallel computing architecture

that is compatible with several standard programming languages, as listed in Figure 2.4 [37], with new parallel programming model and instruction set, it leverages the parallel compute engine in GPUs to solve many complex computational problems in a more efficient way than that with CPUs.

GPU Computing Applications			
C	C++	OpenGL	FORTRAN
NVIDIA GPU with the CUDA Parallel Computing Architecture			

Figure 2.4 CUDA application programming interfaces (Taken from [37])

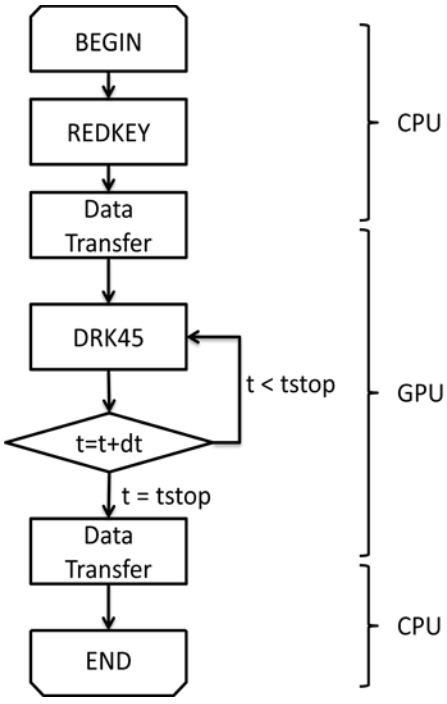


Figure 2.5 Flow chart for GPU-based auto-ignition with explicit integration.

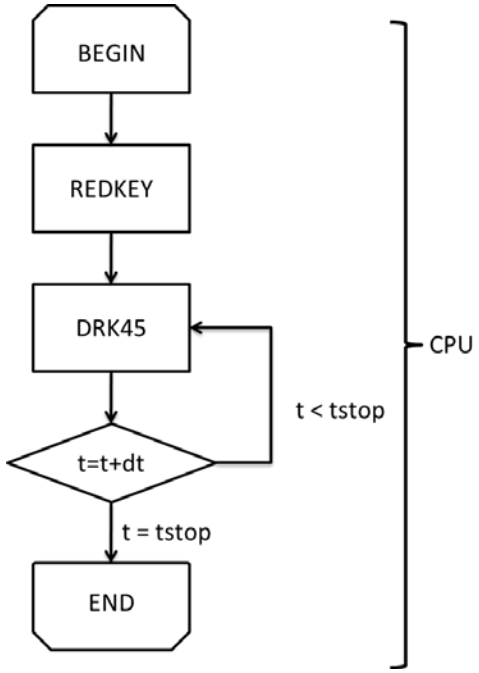


Figure 2.6 Flow chart for CPU-based auto-ignition with explicit integration.

Figures 2.5 and 2.6 show the flow charts for auto-ignition simulations using the 4^{th} order ERK integration method on GPUs and CPUs, respectively. For the conventional computing

shown in Fig. 2.6, CPU is the only device involved. In Fig. 2.5, compared with CPU computing, GPU can be employed to speed up the integration once the initialization of the problem is completed on CPU. For example, in the auto-ignition simulation, the temperature, pressure, species mole fractions, integration time step and overall integration time are initialized on the CPU's memory. These values are then passed to GPU's memory via the PCI bus. The species composition, temperature and pressure are then calculated at each time step on the GPU until the integration is complete. Finally, the results are sent back to CPU's memory. During this process, most of the calculation operations are performed on GPU.

In the present work, the CUDA kernels for chemical reaction rates evaluation on GPU was automatically generated by a Matlab script, considering that a typical rate in detailed chemistry can be expressed in the Arrhenius form [38]:

$$\dot{\omega}_i = k_{fi} \prod_{j=1}^{n} (M_j)^{v_j^{'}} - k_{ri} \prod_{j=1}^{n} (M_j)^{v_j^{''}}, \tag{20}$$

$$k_{fi} = AT^n \exp(-E/RT) \tag{21}$$

$$k_{ri} = \frac{k_{fi}}{K_{ci}} \tag{22}$$

where $\dot{\omega}_i$is net reaction rate, $M_j$is mole concentration of the *j*th chemical species, $v_j^{'}$ is the stoichiometric coefficient of the *j*-th reactants, k_{fi} and k_{ri} are the forward and backward temperature-dependent rates, respectively, A is the kinetic pre-exponential factor that takes into account the collision frequency and the steric factor other than the species concentrations, E is activation energy, R is gas constant, T is temperature, K_{ci} is the equilibrium constant that

is a function of temperature. Furthermore, reaction involving third bodies and fall-off phenomena can be handled in a similar way.

3. Validation of the GPU solvers in homogeneous systems

The parallel GPU solvers are first validated with SENKIN [39], an application in the CHEMKIN software package, for constant pressure and constant volume auto-ignition, respectively. SENKIN uses DASAC^2 [28] to solve the ODEs in Eqs. (2), (7), and (11). To further compare the efficiency of the GPU solvers, the ERK method was implemented for both CPU and GPU, and the comparison in the following is based on wall-clock time.

3.1. Constant-pressure auto-ignition for H_2 and CH_4

First, the GPU solver is validated by simulating constant pressure auto-ignition with a detailed hydrogen mechanism [34].

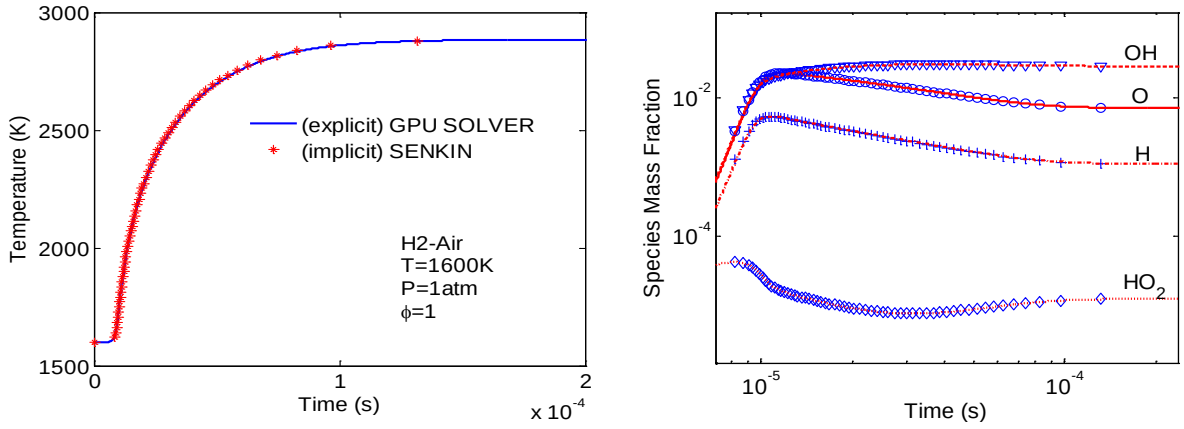


Figure 3.1 Temperature and mass fraction of major species as functions of time in a constant–pressure auto-ignition for stoichiometric H_2-air at atmospheric pressure and initial temperature of 1600K, calculated with SENKIN (symbols) and the explicit GPU solver (lines), respectively.

Figure 3.1 shows temperature and mass fractions for OH, O, H and H_2O in constant-pressure auto-ignition of stoichiometric H_2-air at atmospheric pressure with initial temperature of

1600K, calculated using SENKIN and the GPU solver, respectively. A time step of 1.0e-9s was assumed for the explicit GPU solver. For comparison, SENKIN employs adaptive time steps. It is seen that the GPU solver gives identical result as that from SENKIN. The GPU solver is then validated for a CH4 mechanisms [3]. Figure 3.2 shows the results obtained with the GPU solver and SENKIN, respectively. The integration time step for the explicit GPU solver is 5.0e-11s. The results from the GPU solver are again identical to that from SENKIN.

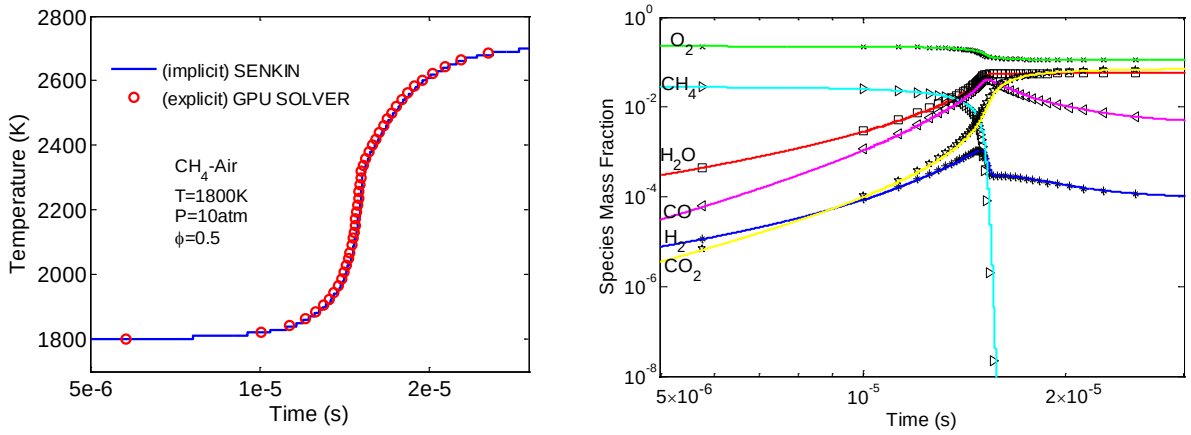


Figure 3.2 Temperature and mass fractions of major species as functions of time for constant-pressure CH_4-air auto-ignition, calculated with SENKIN (symbols) and the explicit GPU solver (lines), respectively.

3.2. Constant-volume auto-ignition for H_2 and CH_4

The GPU solver is then validated for constant-volume auto-ignition. Figure 3.3 shows the temperature and species mass fraction for constant-volume auto-ignition of stoichiometric H_2-air with initial temperature of 1600K at 5 atm. A time step of 1.0e-10s was chosen for the explicit GPU solver. Figure 3.4 compares the GPU solver with SENKIN for stoichiometric CH_4-air at 10 atm with initial temperature of 1800K. The time step is 5.0e-11s for the explicit

integration. Identical results were obtained with the GPU compared with those from SENKIN.

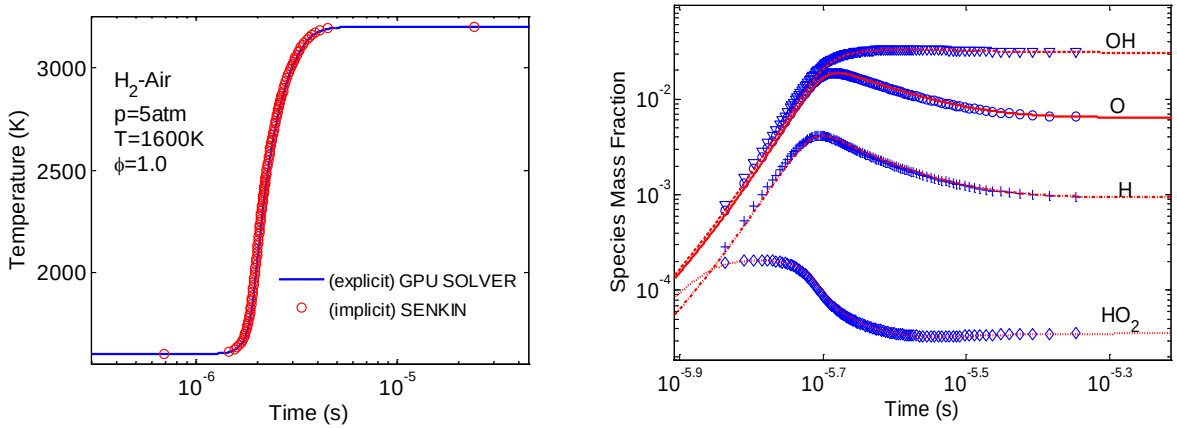


Figure 3.3 Temperature and mass fractions of selected species in a constant–volume auto-ignition for stoichiometric H_2-air at 5atm and initial temperature of 1600K, calculated with SENKIN (symbols) and the explicit GPU solver (lines), respectively.

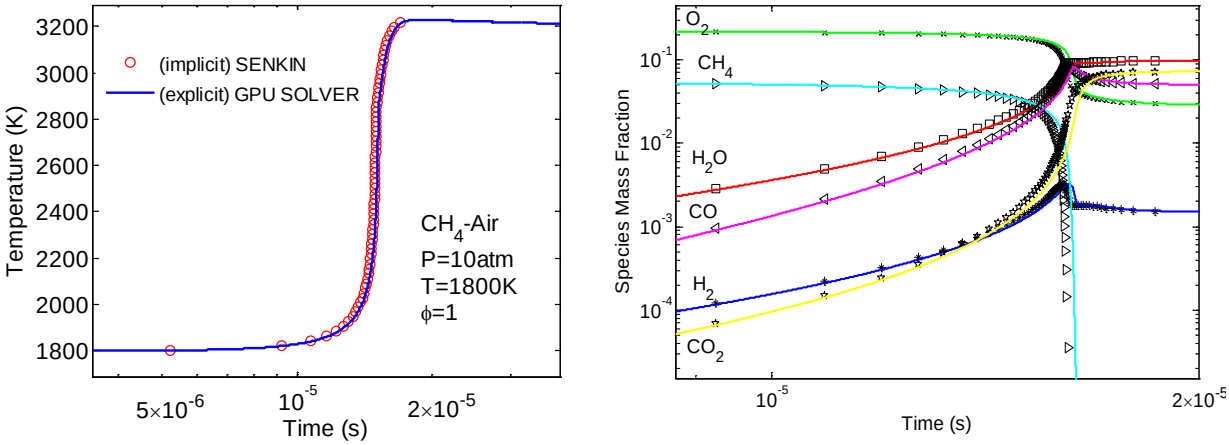


Fig. 3.4 Temperature and mass fractions of selected species concentrations in a constant–volume auto-ignition for stoichiometric CH_4-air at 10 atm with initial temperature of 1800K, calculated with SENKIN (symbols) and the explicit GPU solver (lines), respectively.

3.3. Efficiency of GPU-based chemical kinetics solver

Computational efficiency is of primary concern in combustion simulations, and parallel computing on many core processors has been widely adopted to accelerate numerical simulation. In an ideal case, linear speedup can be achieved using parallel computing. That is, compared with a single-core processor, a program running on n cores simultaneously can reduce the wall-clock time by a factor of n. In practical simulations, speedup achieved using parallel computing is typically not linear, limited by various factors, such as hardware configuration, parallelization scheme, inter-process communication overhead. GPU computing is subjected to additional bottlenecks, such that its efficiency may be severely compromised if a program was not appropriately implemented. In the following, the efficiency of the present GPU solver for will be measured through numerical experiments using a Tesla T10 processor and an Intel Core Duo E8500 processor. The specifications of the two hardware platforms are listed in Table 3.1. The efficiency comparison is based on auto-ignition of hydrogen [34]. The 4th order ERK algorithm was implemented for both GPUs (shown in Figure 2.1) and CPU (shown Figure2.2).

Table 3.1 Hardware specifications

	Tesla T10 processor	**Intel Core(TM)2 Duo E8500**
Number of cores	30*8	2
Number of threads	240	2
Clock rate	1.44GHz	3.16GHz

3.3.1. Effect of number of cases on efficiency

In order to study how the overall work load affects the performance of parallel computing, a set of constant-volume auto-ignition cases with the number of tasks ranging from 16 to 1.0e+6 were dispatched to the CPU and GPU solvers, respectively. The numerical integration time is 5.0e-6s with a fixed time step of 5.0e-10s. These auto-ignition cases have different

initial conditions. The initial temperature for the simulations ranges from 1000K to 1600K and the pressure is 1atm. In general a linear speedup could be achieved by running more threads concurrently. However, due to the limitation of the clock rates of both the GPU and the CPU and the inter-process communication, the linear speedup by GPU typically cannot be achieved in practical simulations. Figure 3.5 plots the wall-clock time for GPU and CPU-based simulations, respectively.

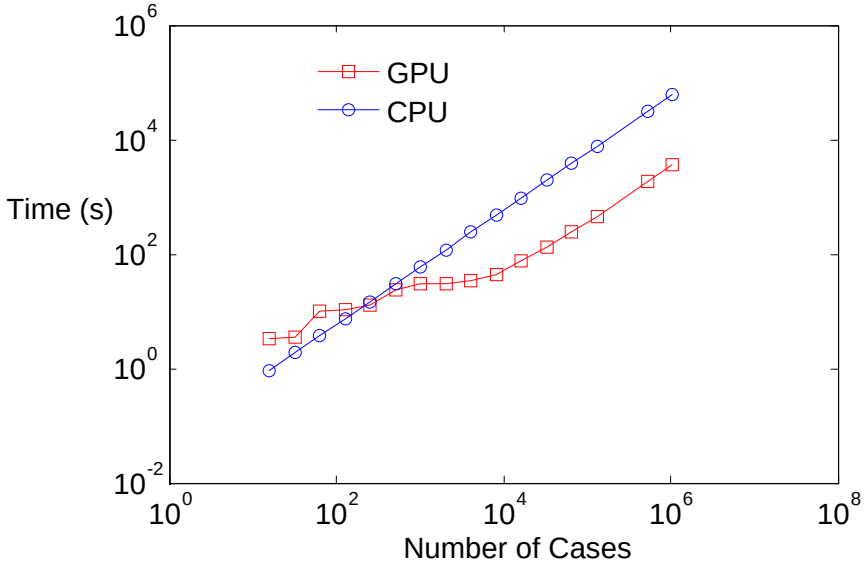


Figure 3.5 Wall-clock time for the GPU and CPU based explicit solvers, respectively, as a function of the number of auto-ignition cases.

The data from Figure 3.5 indicates that the wall-clock time of GPU-based cases are higher than those on CPU when the number of cases is below 200. This reduction in performance is largely caused by overheads as data transfer through the PCI bus. For example, if some data is being transferred from the GPU's memory to the main memory through the PCI bus, the GPU will send a request to the PCI bus arbiter on the motherboard and wait for responses. After receiving permission from the PCI bus arbiter, the GPU can start sending the data.

Because each T10 processor can run 240 threads simultaneously, while only one process is running on the Intel E8500 processor, the computational time by the GPU solver eventually becomes lower than that on CPU as the number of tasks increases. When the number of tasks is large, the CUDA solver can be 16 to 17 times faster than the CPU solver.

3.3.2. Effect of the number of threads in a single block on efficiency

The basic executing unit of GPU is called block, which consists of a large amount of threads. When the GPU is given one or more blocks [37] to execute, the blocks will be distributed to stream processors with available processing capacity automatically, as illustrated in Figure 3.6 [37]. However, the number of threads in each block cannot be calculated automatically by GPU, due to the memory usage of each thread. Instead, it needs to be specified by the user. Therefore, the number of threads in a single block is evaluated by the data size processed by each thread. The physical maximum number of threads in a single block is 512.

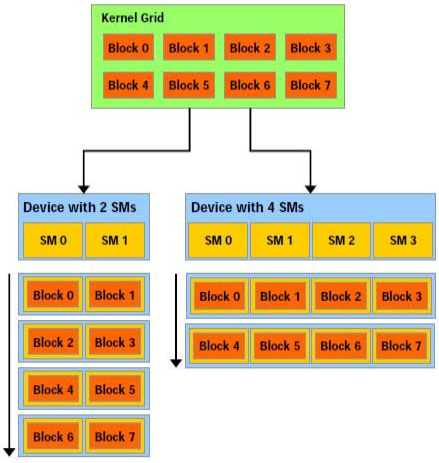


Fig. 3.6 Multiprocessor automatic scalability (Image from [37])

Figure 3.7 shows the wall-clock time as a function of block size for 4096 cases. The size of block increases from 2 threads to the maximum of 512 threads, and the results indicate that the GPU solver has low efficiency when the block size is small, say close to 2, and achieves high efficiency when there are approximately 32 active threads, which is a characteristic of the single instruction multiple data (SIMD) architecture.

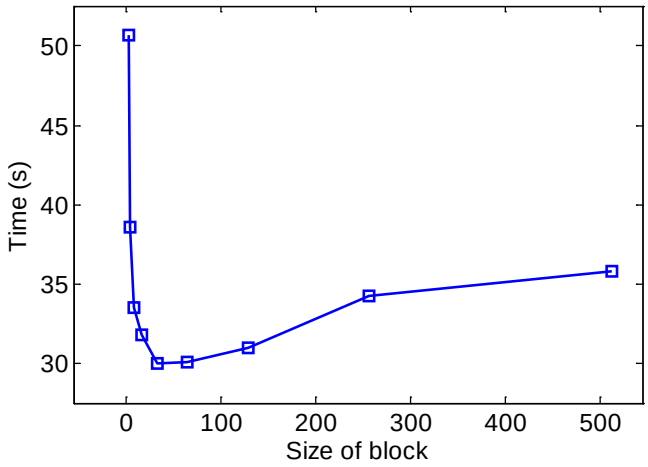


Figure 3.7 Wall-clock time for explicit integration as a function of block size.

4. Quasi-2D simulations using the GPU solver

In this chapter, the CUDA solver is applied to simulate auto-ignition processes in a 2-D domain. This domain is comprised of a number of independent combustion zones. Transport between different zones is ignored for simplicity. The domain consists of 32 by 32 zones. The zones are initialized with non-uniform temperatures as shown in Fig. 4.1. The x and y coordinates are normalized by the number of zones in each direction. A detailed mechanism for DME oxidation [36] is used with the GPU solver.

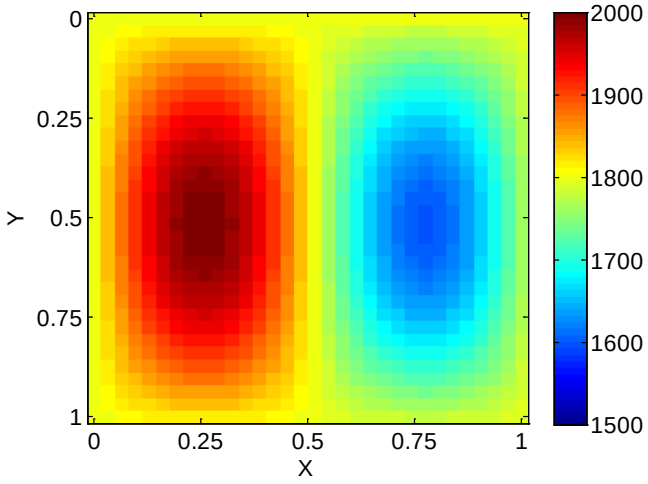
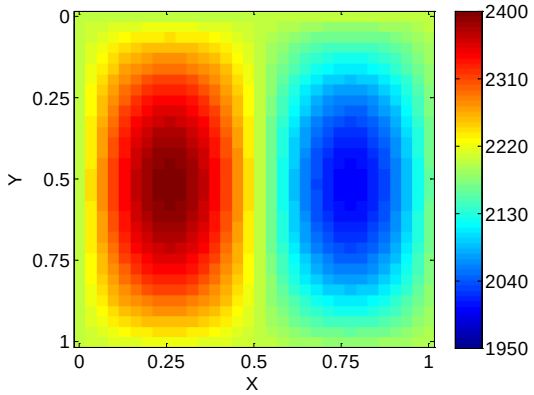
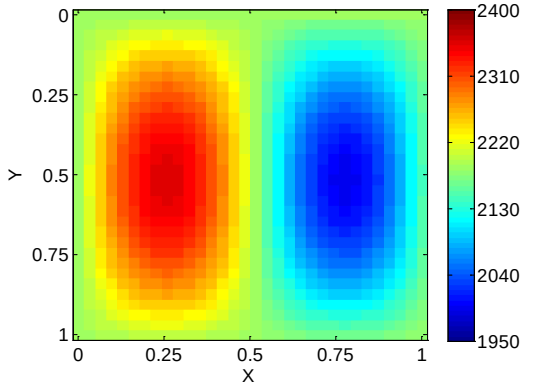
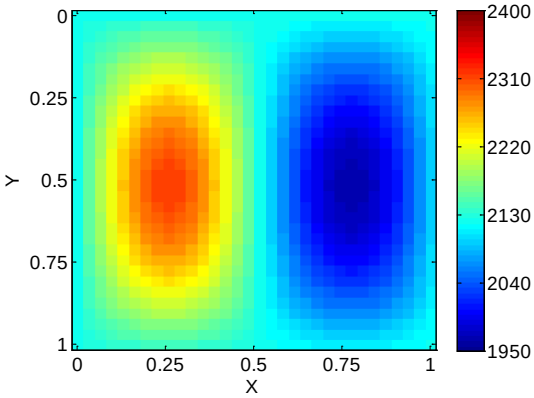
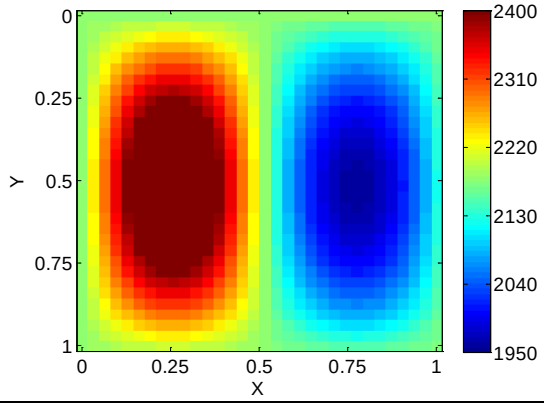


Figure 4.1 Distribution of initial temperature in the 2-D domain (unit: K)

DME is an environmental-friendly fuel that features reduced emission of CO, NO_x, formaldehyde, particulates and non-methane hydrocarbons [35]. It has been extensively used as a fuel or fuel additive for internal combustion engines, or as an ignition enhancer for using methanol in diesels [41-42]. The chemical kinetic mechanism of DME employed in the current study consists of 55 intermediate species and 290 element reactions [35].

Figure 4.2 shows the temperature iso-contours at different time obtained by the GPU solver. Transport between zones is not considered. The initial temperature spans from 2000K to 2400K. When time reaches 1.0e-8s, the highest initial temperature starts to decrease. It drops to approximately 2300K at t=1.0e-7s. At t=1.0e-6s, the temperature difference between the hot area and the cold area reaches 700-800K. Hot ignition then follows.

<table>
<tr><td>t=1.0e-11s</td><td>t= 1.0e-8s</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>t=1.0e-7s</td><td>t=2.0e-7s</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>t=5.0e-7s</td><td>t=1.0e-6s</td></tr>
</table>

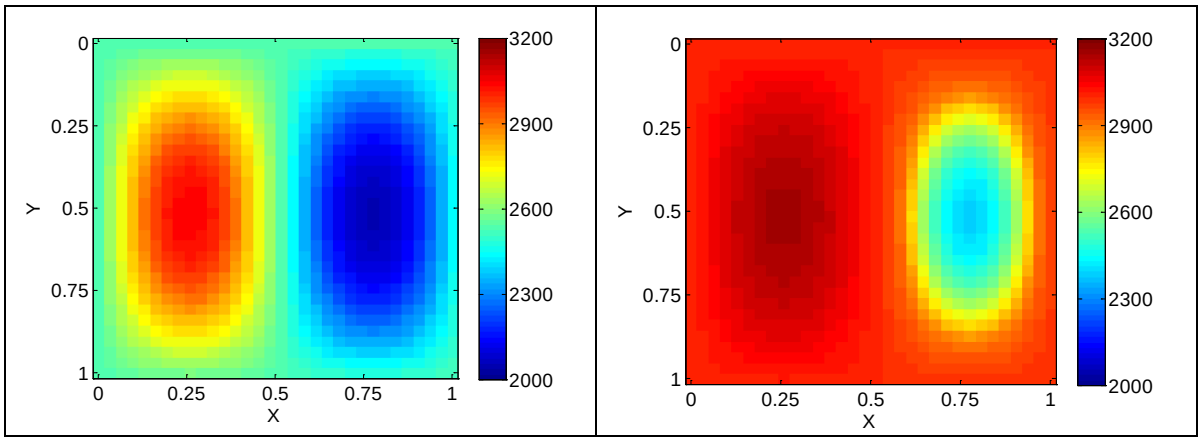
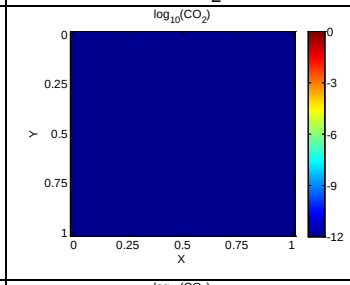
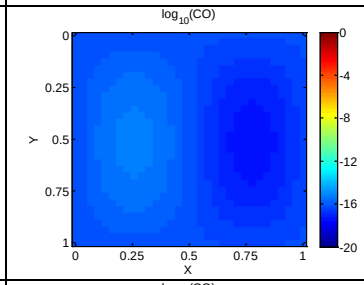
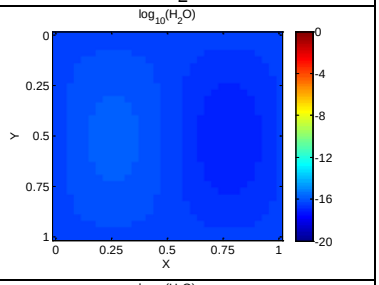
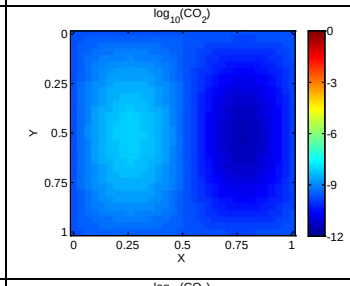
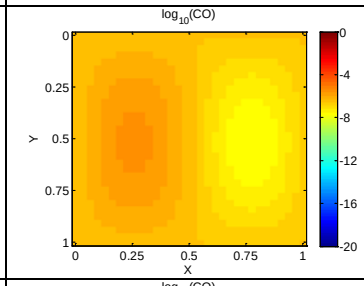
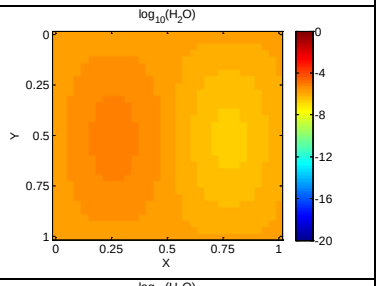
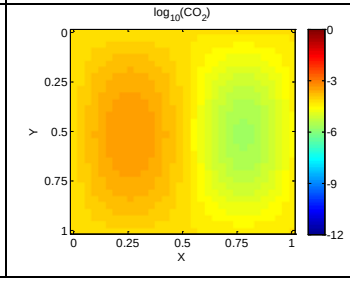
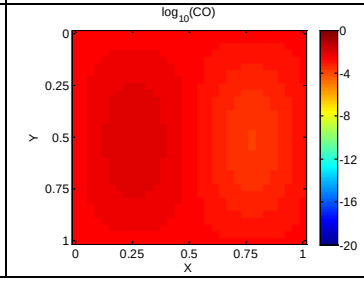
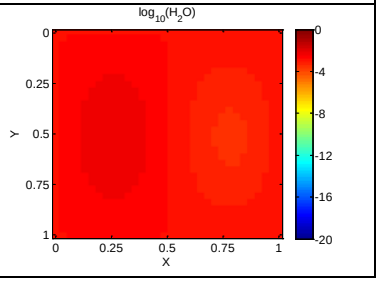


Figure 4.2 Temperature iso-contours of DME auto-ignition in the 2-D domain.

Figure 4.3 shows the iso-contours for the concentration of CO, CO_2, and H_2O at different time. It is seen that the CO and H_2O are already formed at 1.0e-8s, while the formation of CO_2 occurs at a later time.

Time	CO_2	CO	H_2O
t=1.0e-11s			
t=1.0e-8s			
t=1.0e-7s			

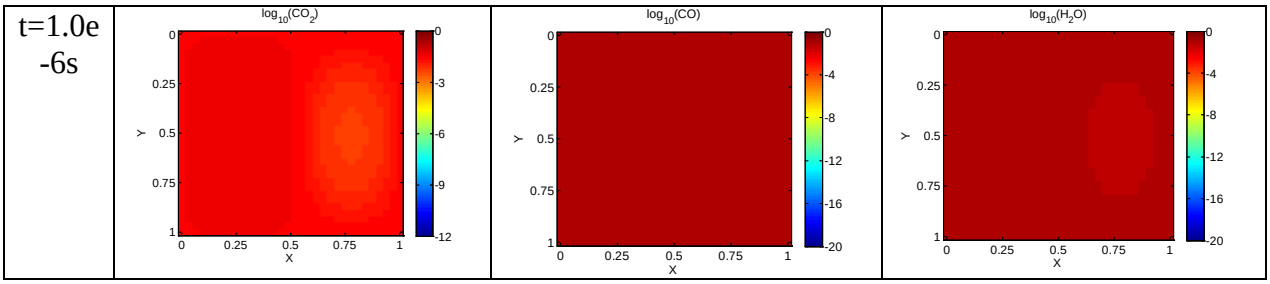


Figure 4.3 Mass fractions of CO, CO_2, and H_2O for DME auto-ignition in the 2-D domain.

The wall-clock time for a single zone per integration time step is shown in Fig. 4.4. It is seen that the averaged time cost for the GPU solver decreases with the number of independent zones, while, the averaged cost for the CPU solver remains mostly constant as the number of zones changes. i.e., the averaged time cost for the GPU solver is significantly lower when the number of zones is large. With more than 10,000 zones, the GPU solver can be approximately 20 times faster than the CPU solver for the quasi 2-D simulations for DME auto-ignition.

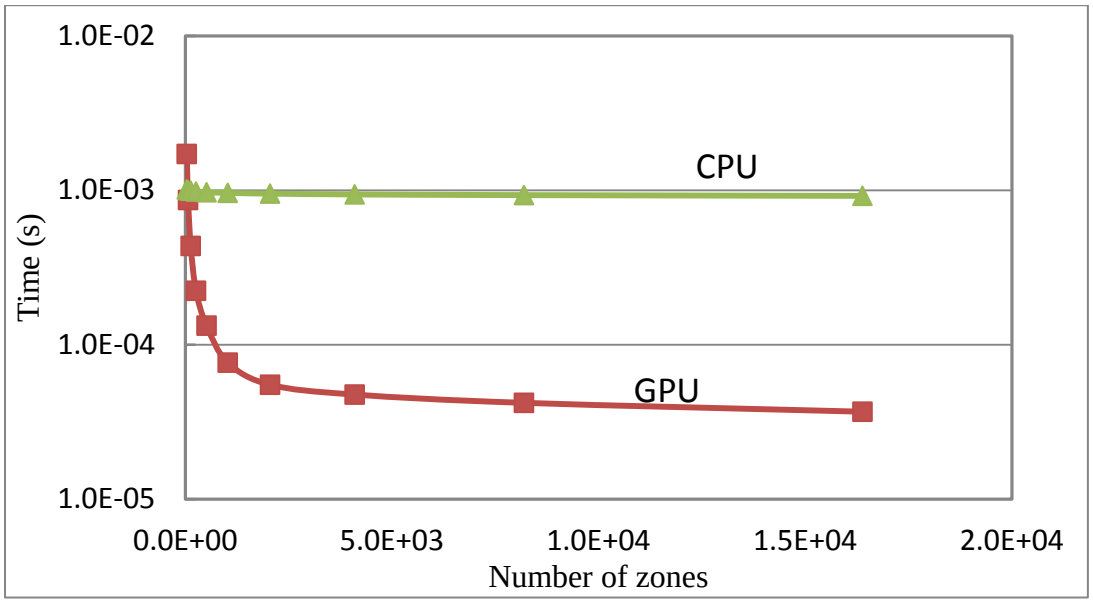


Figure 4.4 Wall clock time per zone per integration for DME auto-ignition in the 2-D domain.

5. Conclusions

In the current work, GPU was employed to numerically simulate chemically reacting flows with detailed chemistry using a 4^{th}-order Runge-Kutta method for explicit time integration. The solver was implemented with CUDA for GPUs. Its accuracy and efficiency were compared with CPU computing with implicit and explicit solvers respectively. The results show that GPU computing is accurate and can be more efficient than CPU computing for combustion simulations. Specifically, the CUDA solver tested with a single NVidia T10 GPU processor can be 16 to 17 times faster than that on a dual-core CPU (E8500). This increase in performance is largely attributed to the many-core architecture of the GPU processors and the high bandwidth of the GPU memory. The GPU performance was further found to be strongly dependent on the bandwidth between the processors and the device, such as the PCI bus.

The CUDA-based explicit integration solver was further applied to a quasi 2-D domain for DME auto-ignition. Each zone in the flow field is independent and ignition occurs at constant-volume condition. The results show that by increasing the number of zones, the averaged time cost decreases for GPU, while remains mostly constant for CPU, indicating that GPU can be used to significantly speedup combustion simulations with detailed chemistry involving a large number of zones.

Nevertheless, limitations of GPU computing was also discussed in the present study. For example, the SIMD architecture makes it more difficult to implement implicit integration algorithms in comparison with CPU. Furthermore, there are only limited number of libraries

that can be employed for CUDA programming. The bottlenecks in GPU computing also need to be carefully avoided in using GPU for combustion simulations.

Reference

[1] Herbinet O., Pitz W. J., and Westbrook C. K., "Detailed Chemical Kinetic Oxidation Mechanism for a Biodiesel Surrogate," Combust. Flame 154 507-528, 2008.

[2] Lu T. F., Law C. K., Yoo C. S., and Chen J. H., "Dynamic Stiffness Removal for Direct Numerical Simulations," Combust. Flame, 156, pp. 1542-1551, 2009.

[3] http://www.me.berkeley.edu/gri_mech

[4] Lu T. F., Yoo C. S., Chen J. H., and Law C. K., 2008, "Analysis of a Turbulent Lifted Hydrogen/Air Jet Flame from Direct Numerical Simulation with Computational Singular Perturbation," Reno, Nevada, US, No. AIAA-2008-1013.

[5] Anderson J., Lorenz C. and Travesset A., “General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units,” Journal of Computational Physics, Vol. 227, No. 10, pp. 5342-5359, 2008.

[6] Liu W., Schmidt B., Voss G. and Muller-Wittig W., “Molecular Dynamics Simulations on Commodity GPUs with CUDA,” Lecture Notes in Computer Science, High Performance Computing – HiPC 2007, Vol. 4873, Springer, New York, pp.185-196, 2007.

[7] Ufimtsev I. and Martinez T., “Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-electron Integral Evaluation,” Journal of Chemical Theory and Computation, Vol. 4, No. 2, pp. 222-231, 2008.

[8] Schatz M. C. and Trapnell C., Delcher A. L. and Varshney A., “High-throughput Sequence Alignment using Graphics Processing Units,” BMC Bioinformatics, BioMed Central, 8:474, 2007.

[9] Barrachina S., Castillo M., Igual F. D., Mayo R. and Quintana-Orti E. S., “Solving Dense Linear Systems on Graphics Processors,” Technical Report ICC 02-02-2008, Universidad Jaume I, Depto. de Ingenieria y Ciencia de Computadores, February 2008.

[10]Castillo M., Chan E., Igual F. D., Mayo R., Quintana-Orti E.S., Geijn R. and Van Zee F.G. “Making Programming Synonymous with Programming for Linear Algebra Libraries”, Technical Report, University of Texas at Austin, Department of Computer Science, Vol. 31, April 17, pp. 8-20, 2008.

[11] Michalakes J. and Vachharajani M., “GPU Acceleration of Numerical Weather Prediction,” Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, IEEE Computer Society, Washington, DC, 2008.

[12] Bleiweiss A., “GPU Accelerated Pathfinding,” Proceedings of the 23rd ACM SIGGRAPH/Eurographics Symposium on, Graphics Hardware, Eurographics Association, Aire-la-Ville, Switzerland, pp. 65-74, 2008.

[13] Foster N. and Metaxas D, "Realistic Animation of Liquids," Graphical Models and Image Processing, pp.471-483, Volume 58, Issue 5. September 1996.

[14] Stam J., "Stable Fluids," In Proceedings of SIGGRAPH, pages121-128, July 1999

[15] Harris M. J., Coombe G., Scheuermann T. and Lastra A., "Physically-Based Visual Simulation on Graphics Hardware," In Proceedings of Graphics Hardware, pages109-118, September 2002.

[16] Goodnight N., Woolley G., Luebke D.and Humphreys G., "A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware," In Proceeding of Graphics Hardware, pages102-111, July 2003.

[17] Krüger J. and Westermann R., "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," ACM Transactions on Graphics (Proceedings of SIGGRAPH), pages908-916, July 2003.

[18] Bolz J., Farmer I., Grinspun E. and Schröoder P., "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," ACM Transactions on Graphics (Proceedings of SIGGRAPH), pages917-924, July 2003.

[19] Hagen T. R., Lie K., and Natvig J. R., "Solving the Euler Equations on Graphics Processing Units," Computational Science-ICCS, 3994, pp. 220–227, 2006.

[20] Brandvik T., and Pullan G., "Acceleration of a 3d Euler Solver Using Commodity Graphics Hardware," Reno, Nevada, US, 2008.

[21] Elsen E., Legresley P., and Darve E., "Large Calculation of the Flow over a Hypersonic Vehicle Using a Gpu," J. Computational Physics, 227, pp. 10148-10161, 2008.

[22] Thompson C. J., Oskin M., "Using Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis," Microarchitecture, (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium , Istanbul, Turkey, pp. 306-317, 2002.

[23] Stam J, Eugene F., "Depicting fire and other gaseous phenomena using diffusion processes," Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, Los Angeles, US: ACM Press, 129-36, 1995.

[24] Wang J., Gu Y., "Flame Simulation Method of Review,". Journal of Image and Graphics, 12(11), pp. 1961-1970, 2007.

[25] Rødal S., Storli G., Gundersen O. E., "Physically based simulation and visualization of fire in real-time using the GPU," In Theory and Practice of Computer Graphics, Eurographics UK Chapter proceedings. Lever L., McDerby M. (Eds.), (Middelsbrough, UK, 2006), Eurographics Association, pp. 13-20, 2006.

[26] Spafford K., Meredith J., Vetter J.S., Chen J., Grout R. and Sankaran R., "Accelerating S3D: A GPGPU Case Study," Seventh International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar 2009) Delft, The Netherlands

[27] Shi Y., Wong H., Oluwole O. O., "Redesigning Combustion Modeling Algorithms for the Graphics Processing Unit (Gpu): Chemical Kinetic Rate Evaluation and Ordinary Differential Equation Integration," Combust. Flame, 158(5), pp. 836-847,2011.

[28] Reaction Design, "A Program for Predicting Homogeneous Gas-Phase Chemical Kinetics in a Closed System with Sensitivity Analysis," 2000.

[29] Morris, T., and Harry, P., Ordinary Differential Equations, 1985.

[30] Hindmarsh A. C., "LSODE and LSODI, Two New Initial Value Ordinary Differnetial Equation Solvers," ACM Sigum Newsletter, 15(4), 1980.

[31] Lu T. F., Wang M., Yoo C. S., Chen J. H. and Law C. K., "A Mode Tracking Method for Flow Classification," Reno, Ann Arbor, MI, 2009.

[32] Lam S. H., "Using Csp to Understand Complex Chemical Kinetics," Combust. Sci. Technol., 89, pp. 375-404, 1993.

[33] Goussis, D. A., and Lam, S. H., "A Study of Homogeneous Methanol Oxidation Kinetics Using CSP," Proc. Combust. Inst., 22, pp. 113–120, 1992.

[34] Li, J., Zhao, Z., Kazakov, A., and Dryer, F. L., 2004, "An updated comprehensive kinetic model of hydrogen combustion," Int. J. Chem. Kinet., 36, pp. 566-575.

[35] Zhao Z., Chaos M., Kazakov A., and Dryer F. L., "Thermal Decomposition Reaction and a Comprehensive Kinetic Model of Dimethyl Ether," Int. J. Chem. Kinet., 40, pp. 1-18, 2008.

[36] Nvidia, 2008, Tesla S1070 Gpu Computing System.

[37] Nvidia, 2009, Nvidia Cuda Programming Guide 2.3, NVIDIA.

[38]. Laidler K. J., 1997, Chemical Kinetics, Benjamin-Cummings.

[39] Reaction Design, 2000, "A Software Package for the Analysis of Gas-Phase Chemical and Plasma Kinetics " Technical Report.

[40] Li J., Zhao Z., Kazakov A., Chaos M., Dryer F. L. and Jr. J. J. S., "A Comprehensive Kinetic Mechanism for Co, Ch2o, and Ch3oh Combustion," Int. J. Chem. Kinet., 39(3), pp. 109-136, 2007.

[41] Green C. J., Cockshutt N. A. and King L., 1990, "Dimethyl Ether as a Methanol Ignition Improver. Substitution Requirements and Exhaust Emissions Impact," SAE Technical Paper 902155, 1990.

[42] Fleisch, T. H., and Meurer, P. C., "DME- the Diesel Fuel for the 21st Century?," AVL Confernce on Engine and Enviroment, Graz, Austria, pp. 3-11, 1995.